С. В. БЕЛУГИНА

РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

Учебное пособие

ЛАНЬ®

САНКТ-ПЕТЕРБУРГ
МОСКВА
КРАСНОДАР
2020

УДК 004
ББК 32.81я723

Б 43 **Белугина С. В.** Разработка программных модулей программного обеспечения для компьютерных систем. Прикладное программирование : учебное пособие / С. В. Белугина. — Санкт-Петербург : Лань, 2020. — 312 с.: ил. — (Учебники для вузов. Специальная литература). — Текст : непосредственный.

ISBN 978-5-8114-4496-0

Курс лекций междисциплинарного курса МДК 01.02 «Прикладное программирование» профессионального модуля ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» соответствует содержанию ФГОС по специальности «Программирование в компьютерных системах».

Предлагаемый краткий курс лекций содержит учебный материал, включающий знакомство с интернет-технологиями, языками создания интернет-приложений и веб-документов, инструментальными средствами создания приложений компьютерных сетей, универсальным языком для работы с приложениями Office.

Учебное пособие поможет систематизировать знания, полученные на лекциях и практических занятиях, выполнить внеаудиторную самостоятельную работу, подготовиться к текущему и промежуточному контролю.

Издание адресовано студентам среднеспециальных образовательных учреждений, а также всем интересующимся данной тематикой.

УДК 004
ББК 32.81я723

Обложка
П. И. ПОЛЯКОВА

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Курс лекций междисциплинарного курса МДК 01.02 «Прикладное программирование» профессионального модуля ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» соответствует содержанию ФГОС специальности 09.02.03 «Программирование в компьютерных системах».

Предлагаемый краткий курс лекций по МДК 01.02 «Прикладное программирование» содержит учебный материал, включающий знакомство с интернет-технологиями, языками создания интернет-приложений и веб-документов, инструментальными средствами создания приложений компьютерных сетей, универсальным языком для работы с приложениями Office.

Тематический план междисциплинарного курса включает 4 темы:

1. Технологии разработки веб-приложений.
2. Основные принципы технологии структурного и объектно-ориентированного программирования.
3. Веб-программирование.
4. Методы и средства разработки технической документации.

В МДК 01.02 на теоретические занятия отводятся 342 часа, на самостоятельное изучение отдельных тем МДК 01.02 студентами — 170 часов учебного времени. Назначение самостоятельной работы — способствовать совершенствованию практических умений по разработке и тестированию программных продуктов, систематизации и обобщению теоретических знаний и получению первоначального опыта творческой деятельности программиста.

По окончании изучения профессионального модуля у студентов должны сформироваться следующие общие и профессиональные компетенции:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.

ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.

ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), за результат выполнения заданий.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

ПК 1.1. Выполнять разработку спецификаций отдельных компонент.

ПК 1.2. Осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять оптимизацию программного кода модуля.

ПК 1.6. Разрабатывать компоненты проектной и технической документации с использованием графических языков спецификаций.

В результате освоения профессионального модуля обучающийся должен уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- создавать программу по разработанному алгоритму как отдельный модуль;
- выполнять отладку и тестирование программы на уровне модуля;
- оформлять документацию на программные средства;
- использовать инструментальные средства для автоматизации оформления документации.

В результате освоения профессионального модуля обучающийся должен знать:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;
- основные принципы отладки и тестирования программных продуктов;
- методы и средства разработки технической документации.

Лекции охватывают содержание междисциплинарного курса МДК 01.02 «Прикладное программирование». Могут использоваться студентами на уроках, при подготовке к практическим и лабораторным работам, в самостоятельной работе.

Тема 1. Технологии разработки веб-приложений
Лекция 1. Введение в веб-программирование

План:

1. Основы технологии «клиент-сервер».
2. Процесс-сервер, процесс-клиент. Схема взаимодействия клиента и сервера.
3. Серверы приложений: типы, назначение, функции.
4. Серверы Интернета.
5. Веб-сервер, его функции и предъявляемые к нему требования.

1. Основы технологии «клиент-сервер»

При взаимодействии по сети производится множество операций, обеспечивающих передачу данных с одной рабочей станции на другую. При этом передаваемые данные проходят множество этапов обработки.

Сначала данные разбиваются на отдельные блоки, из которых формируются кадры данных, снабженные метками, и идентификаторы для того, чтобы потом данные можно было снова «собрать» воедино такими, какими они были до разбиения. Полученные кадры кодируются и передаются по сети в виде электрических или световых сигналов. На стороне принимающей рабочей станции данные проходят обработку в обратном порядке: принимаются, декодируются, собираются из блоков, принимая первоначальный вид.

Для упорядочивания и обеспечения возможности стандартизировать все выполняемые при взаимодействии по сети процедуры используют архитектурный подход, согласно которому сетевые системы описываются с помощью сетевых моделей. Сетевая модель устанавливает соглашения о том, как передавать и принимать данные для всех этапов взаимодействия по сети, начиная от передачи битов, до определения того, как информация должна быть интерпретирована.

С развитием сетевых технологий возникла необходимость в создании стандартов взаимосвязи в сетях открытых систем. В начале 1980-х гг. Международной организацией по стандартизации (ISO) на основе сетевой архитектуры SNA (System Network Architecture) компании IBM была разработана модель взаимодействия открытых систем (Open System Interconnection — OSI). Модель OSI, очень часто называемая эталонной моделью взаимодействия открытых систем, разбивает все процессы взаимодействия и передачи данных по сети на семь уровней.

Формализованные правила, определяющие порядок и формат сообщений, которыми обмениваются сетевые компоненты, представляющие один уровень, но находящиеся в разных узлах сети, называются *протоколами.*

Для взаимодействия между собой протоколы смежных уровней, находящиеся в одном узле сети, используют интерфейсы — четко определенные правила и стандартизованные форматы сообщений.

Задачи и функции по уровням модели OSI

Физический уровень (Physical Layer) определяет физические, механические и электрические характеристики линий связи, к которым относятся:

- тип кабелей и разъемов;
- разводка контактов в разъемах;
- схемы бинарного кодирования сигналов.

На этом уровне осуществляется прием и передача данных по линиям связи. Данные, поступающие с канального уровня, кодируются в электрические или световые сигналы, после этого передаются. При приеме полученные с линии связи данные декодируются и передаются на дальнейшую обработку на канальный уровень. Со стороны компьютера функции физического уровня в локальных сетях обычно выполняются сетевым адаптером, а в глобальных — модемом.

Канальный уровень (Data Link Layer) определяет формирование пакетов соответствующего используемой сети вида. Канальный уровень обычно разделяют на два подуровня:

- управление логической связью (Logical Link Control — LLC);
- доступ к среде передачи данных (Media Access Control — MAC).

Верхний подуровень управления логической связью осуществляет установку и поддержку виртуального канала связи, а также обеспечивает взаимодействие с сетевым уровнем.

Нижний подуровень доступа к среде передачи данных обеспечивает непосредственный доступ к каналу связи и связан с аппаратурой сети. Помимо контроля над состоянием сети, повторной передачи кадров при их утере, приема кадров и проверки правильности передачи, в функции этого уровня входит взаимодействие с физическим уровнем.

Сетевой уровень (Network Layer) обеспечивает адресацию пересылаемых пакетов. Также здесь решаются задачи управления потоками данных в сети и маршрутизации, т. е. выбора маршрута передачи данных по узлам сети.

Транспортный уровень (Transport Layer) является своего рода связующим звеном между более высокими уровнями, сильно зависящими от приложений, и нижними уровнями, более привязанными к линиям связи. На транспортном уровне происходит разбиение передаваемой информации на пакеты и сборка принимаемых данных из пакетов. Здесь же обеспечивается контроль над передачей данных.

На транспортном уровне реализованы возможности обнаружения и исправления ошибок передачи данных, вызванных искажениями, потерями или дублированием пакетов.

Кроме того, на этом уровне происходит согласование сетевых уровней различных несовместимых сетей.

Сеансовый уровень (Session Layer) обеспечивает координацию связи между двумя рабочими станциями сети. Уровень организует сеанс обмена данными, управляет приемом и передачей пакетов, обеспечивает завершение сеанса. Кроме того, осуществляется контроль над степенью завершения длинных передач, что позволяет избежать повторной пересылки данных при разрывах связи и возобновить передачу с прерванного места.

Уровень представления (Presentation Layer) имеет дело с внешним представлением данных и обеспечивает преемственность передаваемой информации

с одной системы для другой системы. С его помощью преодолеваются различия, например, между всевозможными кодировками символов или синтаксиса.

Средства уровня представления позволяют выполнять различные виды преобразования данных: шифрование, дешифрование, сжатие.

Прикладной уровень (Application Layer) реализует взаимодействие прикладных программ пользователя с процессами модели OSI, обеспечивая им набор определенных сетевых услуг, таких как передача файлов, обмен почтовыми сообщениями, управление сетью и т. д.

Клиент и сервер какого-либо ресурса могут находиться как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью.

Основной принцип технологии «клиент – сервер» заключается в разделении функций приложения на три группы:

- ввод и отображение данных (взаимодействие с пользователем);
- прикладные функции, характерные для данной предметной области;
- функции управления ресурсами (файловой системой, базой данных и т.д.)

Поэтому в любом приложении выделяются следующие компоненты:

- компонент представления данных;
- прикладной компонент;
- компонент управления ресурсом.

Связь между компонентами осуществляется по определенным правилам, которые называют «протокол взаимодействия».

Централизованная и клиент-серверная модели

По мере развития представлений о распределенных вычислительных процессах и процессах обработки данных сложилась концепция архитектуры «клиент – сервер» — обобщённое представление о взаимодействии двух компонент информационной технологии (технического и/или программного обеспечения) в вычислительных системах и сетях, среди которых логически или физически могут быть выделены:

- активная сторона (источник запросов, клиент);
- пассивная сторона (сервер, обслуживание запросов, источник ответов).

Взаимодействие «клиент – сервер» в сети осуществляется в соответствии с определённым протоколом.

Серверные приложения (брокеры, роботы) устанавливаются между разнопротокольными компонентами и осуществляют трансформацию протоколов Сети архитектуры «клиент – сервер» позволяя обеспечить:

- доступ к базам данных таких приложений, как электронные таблицы, бухгалтерские программы;
- коммуникационные приложения, системы управления документами;
- управление сетью;
- централизованное хранение файлов.

Сегодня большинство сетей использует модель «клиент – сервер». Сеть архитектуры «клиент – сервер» — это сетевая среда, в которой компьютер-клиент инициирует запрос компьютеру-серверу, который этот запрос выполняет.

«Клиент – сервер» — архитектура или организация построения, в которой производится разделение вычислительной нагрузки между включёнными в нее компьютерами, выполняющими функции клиентов, и одной мощной центральной ЭВМ — сервером.

Клиентом будет обычная программа, расположенная на любой ЭВМ, включенной в сеть, а также сама ЭВМ, которая по мере необходимости запрашивает данные с сервера.

Достоинствами клиент-серверной архитектуры являются большой объем памяти и ее пригодность для решения разнородных задач, возможность подключения большого количества рабочих станций.

Основные понятия клиент-серверной архитектуры.

Клиент — сторона (ЭВМ, программа или пользователь), запрашивающая и использующая информацию и/или ресурсы у сервера в среде «клиент – сервер».

Пользователь (клиент) генерирует запрос с помощью интерфейсного приложения, которое выполняет следующие функции:

- обеспечивает интерфейс пользователя;
- формирует запросы;
- отображает данные, полученные с сервера.

Тонкий клиент — терминал сети без жестких дисков и флоппи-дисководов, вычислительная мощность которого и объем памяти определяются задачами пользователя. Все программы и приложения, хранящиеся на сервере, становятся доступными для пользователя при включении его устройства и выполнении процедуры регистрации на сервере. Тонким клиентом называют также ПК с минимизированной мощностью процессора, оперативной и внешней памяти, позволяющий пользователю осуществлять ввод и отображение данных за счет выполнения вычислений и хранения данных на более мощном ПК или сервере, с которыми он может осуществлять связь, при помощи каналов средней пропускной способности. К тонкому клиенту могут подключаться внешние устройства ввода/вывода данных (сканеры, мониторы, принтеры и проекторы).

Брокеры объектных запросов (ORB, Object Request Brokers) — форма промежуточного программного обеспечения для разработки систем «клиент – сервер».

Двухзвенная модель — архитектура построения сети, предусматривающая один сервер и несколько клиентов, является наиболее простой и распространённой. Недостаток — ограниченное число клиентских рабочих мест.

Трехзвенная модель — архитектура построения системы «клиент – сервер», в которой предусмотрено промежуточное звено (дополнительный компьютер), расположенное между сервером и клиентом двухзвенной модели. Промежуточное звено работает как монитор обработки транзакций или брокер объектных запросов. Трехзвенные модели обеспечивают работу существенно большего числа клиентов, чем двухзвенные модели.

N-звенная модель — архитектура построения сети, предусматривающая наличие нескольких серверов приложений, число которых определяется необходимым уровнем нагрузки сети. При многозвенной модели построения системы количество возможных клиентских мест значительно больше, чем при использовании двухзвенной модели.

2. Процесс-сервер, процесс-клиент. Схема взаимодействия клиента и сервера

Каждый из составляющих архитектуру «клиент-сервер» элементов играет свою роль: сервер владеет и распоряжается информационными ресурсами системы, клиент имеет возможность воспользоваться ими.

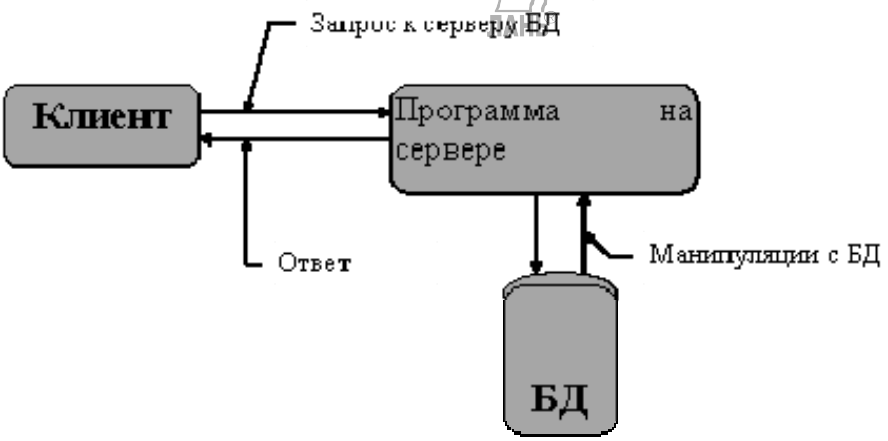


Рисунок 1 — Архитектура «клиент-сервер»

Сервер базы данных представляет собой мультипользовательскую версию СУБД, параллельно обрабатывающую запросы, поступившие со всех рабочих станций. В его задачу входит реализация логики обработки транзакций с применением необходимой техники синхронизации — поддержка протоколов блокирования ресурсов, обеспечение, предотвращение и/или устранение тупиковых ситуаций.

В ответ на пользовательский запрос рабочая станция получит не «сырье» для последующей обработки, а готовые результаты.

Программное обеспечение рабочей станции при такой архитектуре играет роль только внешнего интерфейса централизованной системы управления данными. Это позволяет существенно уменьшить сетевой трафик, сократить время на ожидание блокированных ресурсов данных в мультипользовательском режиме, разгрузить рабочие станции и при достаточно мощной центральной машине использовать для них более дешевое оборудование.

Для современных СУБД архитектура «клиент – сервер» стала фактически стандартом. Если предполагается, что проектируемая информация будет иметь архитектуру «клиент – сервер», то это означает, что прикладные программы, реализованные в ее рамках, будут иметь распределенный характер, т. е. часть функций приложений будет реализована в программе-клиенте, другая — в программе-сервере.

Формы организации архитектуры «клиент – сервер» отличаются тем, какие задачи решаются клиентом и сервером.

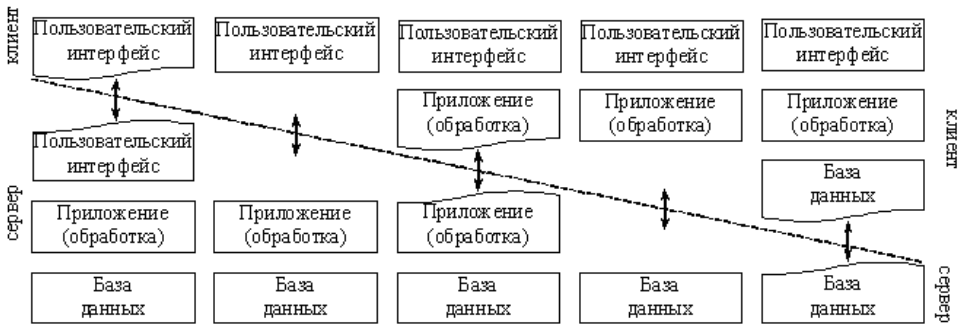


Рисунок 2 — Задачи клиента и сервера

Взаимодействие клиента и сервера в Интернете осуществляется с помощью запросов, посылаемых клиентом серверу, и ответов сервера на запрос клиента.

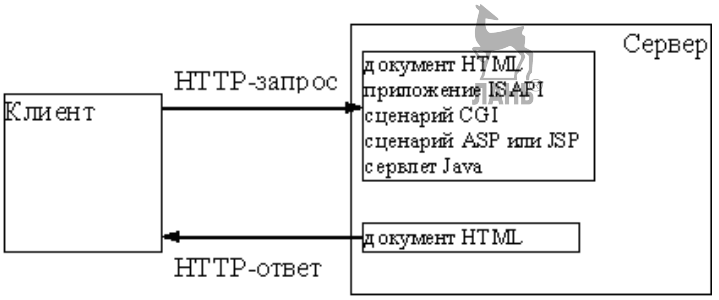


Рисунок 3 — Взаимодействие клиента и сервера

Суть распределенных систем — связь между процессами, реализующими не только взаимодействие компьютеров, но и частей (уровней) приложений. Взаимодействие частей приложений реализуется с помощью протоколов, описывающих состав и формат данных, пересылаемых соответствующими частями клиентских и серверных приложений друг другу для решения поставленной задачи.

В Интернете разбиение приложений на части осуществляется на базе стека протоколов TCP/IP.

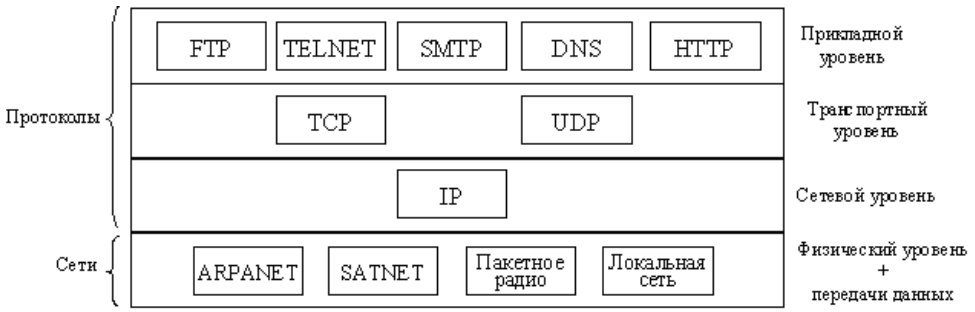


Рисунок 4 — Стек протоколов TCP/IP

Первоначально в Интернете существовала только возможность пересылки файлов (протокол FTP), почтовых сообщений (e-mail) и подключение компьютера как удаленного терминала сервера (TELNET).

Технология World Wide Web (WWW) и производство доступных для пользователей средств коммуникации (модемы) изменили Интернет и сделали его основным средством получения и опубликования информации.

Основу WWW составляет гипертекстовый документ, создаваемый с помощью языка разметки гипертекстовых документов (HTML), и протокол передачи гипертекстовых документов (HTTP).

Суть гипертекстового документа заключается в том, что весь документ разбит на части, которые могут храниться не обязательно на одном компьютере в сети, а в соответствующие части документа встроены гиперссылки на его другие части.

В настоящее время используется язык HTML версии 4.01, поддерживаемый организацией WWW-консорциум. Его основу составляют HTTP-сообщения, подразделяемые на:

- запрос (request) клиента к серверу;
- ответ (response) сервера клиенту.

Для сервера был разработан специальный интерфейс Common Gateway Interface (CGI) (стандарт), доступный приложениям, выполняемым на машине сервера (используются языки C, Perl).

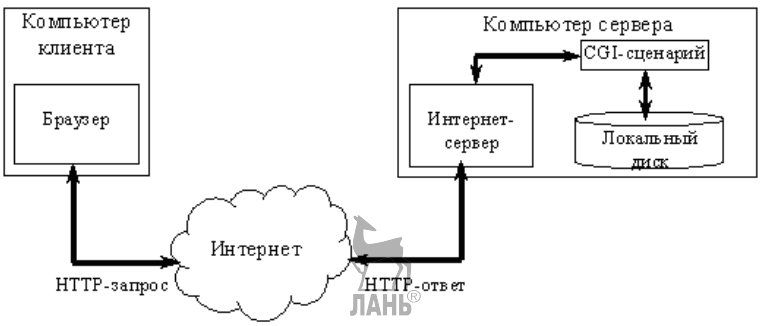


Рисунок 5 — Интерфейс CGI

Каждое обращение клиента к CGI-сценарию выполняется в собственном процессе, создаваемом операционной системой. Получение передаваемой и отсылаемой информации реализовано через стандартные потоки ввода/вывода.

Компания Microsoft для своего сервера IIS реализовала кроме CGI-интерфейса дополнительный интерфейс ISAPI — Internet Server Application Programming Interface. Этот интерфейс позволяет создавать серверные приложения, выполняющиеся в том же процессе, в котором выполняется и программа самого сервера, а также он позволяет создавать сценарии, обладающие дополнительными возможностями взаимодействия сервера и сценария.

Одним из недостатков CGI-технологии является то, что страница HTML формируется полностью с нуля при каждом запросе к CGI-сценарию. Компания Microsoft, разработав технологию IDC (Internet Database Connector), уже в ней

реализовала идею *шаблона* — документа HTML, в определенные места которого вставлялись данные по запросу пользователя из базы данных. Эта технология была реализована через ISAPI.

Дальнейшее развитие серверных технологий привело к появлению технологии ASP — Active Server Pages, в которой совмещены наилучшие стороны всех рассмотренных технологий. Страница ASP — это заготовка документа HTML, в который кроме кода HTML встроен выполняемый во время обработки до отсылки клиенту код, написанный на одном из двух, поддерживаемых в ASP языков сценариев — JavaScript (JScript) или VBScript.

Модели технологии «клиент – сервер»

Основной принцип технологии «клиент – сервер» заключается в разделении функций стандартного интерактивного приложения на четыре группы:

- функции ввода и отображения данных;
- прикладные функции, характерные для предметной области;
- фундаментальные функции хранения и управления ресурсами (базами данных);
- служебные функции.

Исходя из этого деления, любое приложение может состоять из следующих компонентов:

- компонент представления (функции 1-й группы);
- прикладной компонент (функции 2-й группы);
- компонент доступа к информационным ресурсам (функции 3-й группы и протокол их взаимодействия).

Различия определяются четырьмя факторами:

- какие виды программного обеспечения в логических компонентах;
- какие механизмы программного обеспечения используются для реализации функций трех групп;
- как логические компоненты распределяются компьютерами в сети;
- какие механизмы используются для связи компонент между собой.

Рассмотрим четыре подхода, реализованные в моделях технологии «клиент – сервер»:

1) FS-модель;
2) RDA-модель;
3) DBS-модель;
4) AS-модель.

Приложения типа «клиент – сервер», предназначенные для организации доступа пользователей к базам данных, делятся на три уровня.

Уровень пользовательского интерфейса. Уровень пользовательского интерфейса обычно реализуется на клиентах. Этот уровень содержит программы, посредством которых пользователь может взаимодействовать с приложением.

Уровень обработки. Многие приложения модели «клиент – сервер» построены как бы из трех различных частей: части, которая занимается взаимодействием с пользователем, части, которая отвечает за работу с базой данных или файловой системой, и средней части, реализующей основную функцио-

нальность приложения. Эта средняя часть логически располагается на уровне обработки. В рамках модели «клиент – сервер» часть, которая отвечает за выборку информации, обычно находится на уровне обработки.

Уровень данных. Уровень данных в модели «клиент – сервер» содержит программы, которые предоставляют данные обрабатывающим их приложениям. Специфическим свойством этого уровня является требование ***сохранности*** **(*persistence*)**. Это означает, что, когда приложение не работает, данные должны сохраняться в определенном месте в расчете на дальнейшее использование. В простейшем варианте уровень данных реализуется файловой системой, но чаще для его реализации задействуется полномасштабная база данных. В модели «клиент – сервер» уровень данных обычно находится на стороне сервера.

3. Серверы приложений: типы, назначение, функции

Сетевая услуга или сетевой сервис — это процесс обслуживания объектов сети, обычно связанный с распределенной обработкой данных и информационным обменом. Объектами сети могут быть пользователи, программы, операционные системы, функциональные блоки, вычислительные процессы и т. д.

Примерами сетевых услуг являются следующие распространенные виды сервисов:

- хранение данных;
- поиск информации;
- почтовые услуги (например, электронная почта);
- передача сообщений и блоков данных между узлами сети;
- организация сеансов взаимодействия между прикладными процессами.

Сетевой сервис определяет интерфейс между потребителем и поставщиком сетевых услуг.

Потребителями сетевых услуг могут являться пользователи, прикладные программы, другие объекты сети.

Поставщиком сетевых услуг является сетевая служба — некая сетевая компонента, совокупность средств, которые позволяют реализовать услугу либо набор услуг.

К таким средствам относятся:

- средства обеспечения общего доступа и пользования локальных ресурсов и услуг — серверная часть программного обеспечения, реализующего сетевую службу;
- средства получения доступа и обеспечения использования удаленных ресурсов и услуг — клиентская часть программного обеспечения, реализующего сетевую службу.

Взаимодействие между клиентами и сервером осуществляется посредством телекоммуникационных средств сетевой службы и сети передачи данных, выполняющих формирование сообщений запросов и ответов, разбиение этих сообщений при необходимости на отдельные блоки данных, обеспечение адресации, маршрутизации, надежной доставки этих сообщений и т. д. в соответствии с правилами, которые определяются используемыми коммуникационными протоколами.

Обычно сетевая служба располагается на прикладном уровне модели OSI, т. е. выполняет его функции, иногда может занимать и уровень представления. При этом сетевая служба, используя средства нижележащих уровней, не зависит от типа используемой коммуникационной сети.

По степени интеграции сетевой службы в операционную систему различают следующие виды программной реализации сетевой службы:

- высокая степень интеграции — сетевая служба является частью операционной системы;
- средняя степень интеграции — сетевая служба представляет собой надстройку над операционной системой;
- низкая степень интеграции — сетевая служба является самостоятельным программным продуктом.

Задача сетевого программного обеспечения состоит в приеме запроса от приложения на одной машине, передаче его на другую машину, выполнения запроса на удаленной машине и возврате результата на первую машину. Таким образом, сетевое ПО может быть выполнено как в виде отдельных модулей, так и в виде компонентов самой ОС. Фактически эта последовательность и была повторена в ходе развития сетевого ПО: в настоящее время фактически все ОС включают штатные компоненты сетевого ПО. Сетевое ПО является достаточно сложным программным обеспечением, к тому же надежность его функционирования зависит от корректности работы физической компоненты сети (сетевые карты, линии связи), поэтому важной частью сетевого ПО являются подсистема анализа показателей и поиска неисправностей в сетях.

Типы серверов приложений:

- файловый сервер (File Server — FS);
- доступ к удаленным данным (Remote Data Access — RDA);
- сервер баз данных (Data Base Server — DBS);
- сервер приложений (Application Server — AS);
- в сервер WWW.

Файловый сервер (FS). Этот подход является базовым для локальных сетей ПК. Один из компьютеров сети назначается файловым сервером и предоставляет другим компьютерам услуги по обработке файлов. Файловый сервер работает под управлением сетевой операционной системы и играет роль компонента доступа к информационным ресурсам.

Доступ к удаленным данным (RDA) существенно отличается от FS методом доступа к информационным ресурсам. В данной технологии программы компонента представления и прикладного компонента совмещены и выполняются на компьютере-клиенте. Доступ к информационным ресурсам обеспечивается операторами специального языка (например, языка запросов SQL, если речь идет о базах данных) или вызовами функций специальной библиотеки (если имеется специальный интерфейс прикладного программирования — API).

Запросы к информационным ресурсам направляются по сети удаленному компьютеру, который обрабатывает и выполняет их, возвращая клиенту блоки данных.

Достоинства RDA:

- унификация интерфейса «клиент – сервер» в виде языка запросов;
- широкий выбор средств разработки приложений.

Недостатки

- существенная загрузка сети при взаимодействии клиента и сервера посредством запросов;
- невозможность администрирования приложений в RDA, так как в одной программе совмещаются различные по своей природе функции (представления и прикладные).

Сервер баз данных (DBS) — технология реализована в некоторых реляционных (табличных) СУБД (Informix, Ingres, Sybase, Oracle). Ее основу составляет механизм хранимых процедур — средство программирования SQL-сервера. Процедуры хранятся в словаре баз данных, разделяются между несколькими клиентами и выполняются на том же компьютере, где функционирует SQL-сервер. В сервере баз данных компонент представления выполняется на компьютере-клиенте, в то время как прикладной компонент оформлен как набор хранимых процедур и функционирует на компьютере-сервере БД. Там же выполняется компонент доступа к данным, т. е. ядро СУБД.

Сервер приложений (AS) представляет собой процесс, выполняемый на компьютере-клиенте, отвечающий за интерфейс с пользователем (т. е. реализует функции первой группы). Прикладной компонент реализован как группа процессов, выполняющих прикладные функции, и называется сервером приложения.

Сервер WWW — программа, которая принимает запросы от WWW-клиентов и отвечает на них. В качестве ответа может быть возвращен HTML-документ, хранящийся в базе данных сервера, графический образ, аудиозапись, фильм или ответ внешней программы. Сервер обменивается данными не только с клиентами, но и с CGI-скриптами. В настоящее время серверы WWW существуют для всех типов компьютерных платформ и операционных систем.

Серверы для UNIX-систем:

- HTTPD (NCSA) — наиболее распространен в сети; большое количество клиентов настроены для работы с этим типом сервера;
- Apachie — некоммерческое развитие сервера NCSA с учетом спецификаций защиты данных от несанкционированного доступа;
- WN-сервер — реализует механизм графического стека ссылок в себе самом, а не через внешний скрипт, что повышает защищенность данных. Кроме того, данный сервер позволяет воспользоваться механизмом обновления информации протокола HTTP 1.0 для организации видеоклипов. В настоящее время наиболее завершенным выглядит WN.

4. Серверы Интернета

Сервер «Интернет – компьютер», подключенный к сети, или выполняющаяся на нем программа, предоставляющие клиентам доступ к общим ресурсам и управляющие этими ресурсами.

Каждый компьютер, подключенный к сети Интернет, имеет два равноценных уникальных адреса: цифровой IP-адрес и символический доменный адрес.

IP-адрес — уникальный адрес компьютера в сети Интернет, имеющий длину 4 байта. Обычно первый и второй байты определяют адрес сети, третий байт определяет адрес подсети, а четвертый — адрес компьютера в подсети. IP-адрес записывают в виде четырех чисел со значениями от 0 до 255, разделенных точками.

Пространство доменных имен имеет иерархическую структуру. На самом верхнем уровне иерархии располагается корневой домен, который обычно обозначается точкой («.»). Следующий уровень иерархии составляют домены верхнего, или первого, уровня. Каждый домен верхнего уровня включает в себя домены второго уровня и т. д. Теоретически домен любого уровня может содержать в себе как отдельные узлы, представленные своими именами, так и домены более низкого уровня (субдомены). Домены первого уровня делятся на три группы:

- домены общего назначения;
- национальные домены;
- обратный домен.

Наиболее важными типами серверов являются:

- веб-серверы;
- серверы электронной почты (e-mail);
- серверы FTP, предназначенные для обмена файлами;
- серверы общения в реальном времени (чаты, IRC);
- серверы, обеспечивающие работу интернет-телефонии (IP-телефония);
- системы трансляции радио и видео через Интернет;
- телеконференции, или группы новостей (Usenet), обеспечивающие возможность коллективного обмена сообщениями;
- сервис Telnet, предназначенный для управления удалёнными компьютерами в терминальном режиме;
- сервис DNS, или система доменных имен.

Программы-серверы

Сервер WWW — программа, которая принимает запросы от WWW-клиентов и отвечает на них. В качестве ответа может быть возвращен HTML-документ, хранящийся в базе данных сервера, графический образ, аудиозапись, фильм или ответ внешней программы. Сервер обменивается данными не только с клиентами, но и с CGI-скриптами.

Клиент, которым обычно является веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URL-адресами.

Ресурсы — это HTML-страницы, изображения, файлы, медиапотоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Серверы электронной почты

Почтовый сервер, сервер электронной почты, мейл-сервер — в системе пересылки электронной почты так обычно называют агент пересылки сообщений.

Это компьютерная программа, которая передаёт сообщения от одного компьютера к другому. Обычно почтовый сервер работает «за кулисами», а пользователи имеют дело с другой программой — клиентом электронной почты (англ. *mail user agent, MUA*).

Сервер, который принимает сообщения от отправителя и пересылает их другим серверам, принято называть сервером исходящей почты. А сервер, обеспечивающий хранение поступившей почты и предоставление ее получателю, — сервером входящей почты. Сервис электронной почты допускает, что функции серверов входящей и исходящей почты для конкретного пользователя могут выполнять как два различных почтовых сервера, так и один и тот же, имеющий в своем составе соответствующие программные модули.

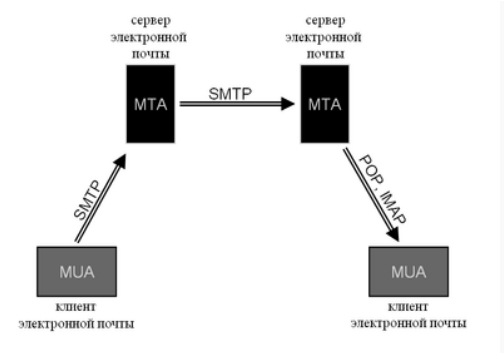


Рисунок 6 — Работа почтового сервера

Каждый абонент с точки зрения почтовой службы представлен почтовым ящиком.

Почтовый ящик (mailbox) — это информационное пространство, выделенное на некотором узле сети для хранения почтовых сообщений и обладающее уникальным именем в рамках узла.

В структуре сервиса электронной почты предусмотрены следующие компоненты:

1) информационный ресурс;
2) почтовый сервер;
3) почтовый клиент;
4) протоколы взаимодействия почтовых клиентов с серверами. В настоящее время основными протоколами являются:

- SMTP (Simple Mail Transfer Protocol), использующийся сервисом электронной почты для передачи сообщений от отправителя к получателю;
- POP3 (Post Office Protocol версии 3) и IMAP4 (Internet Message Access Protocol версии 4), обеспечивающие выборку входящих сообщений из почтового ящика.

5. Веб-сервер, его функции и предъявляемые к нему требования

Системы управления веб-контентом — программное обеспечение, устанавливаемое на веб-сервере. Их основной задачей является контроль контента, поступающего на сайт, для обеспечения достоверности и

поступающего на сайт, для обеспечения достоверности и своевременности информации, размещенной на сайте.

В зависимости от уровня сложности системы управления контентом можно разделить на три группы:

первая группа — это статическая веб-страница, которую, как правило, делает ИТ-специалист, а затем при необходимости он же и вносит туда изменения;

ко второй группе следует отнести сайты, на которых администратор может самостоятельно изменять содержание, но не его структуру и дизайн;

к третьей группе относятся системы, позволяющие администратору вносить изменения в структуру сайта, добавлять и удалять разделы в рамках дизайна и навигации.

Цель системы управления контентом — обновление содержания сайта

Системы управления сайтом (контентом) имеют слабые места — это работа сайта *под нагрузкой*. Ресурс мощности всегда ограничен, при пиковой нагрузке система может отказать. Поэтому следует помнить о таком важном требовании к системам управления контентом, как наличие возможности *кэширования информации*. Смысл кэширования в следующем: для первого пользователя, пришедшего на ресурс, страница действительно собирается, а вот для всех последующих она просто высылается в виде статик-контента, поскольку она уже была сохранена в кэш-памяти и в любой момент готова к отсылке. Данный способ снижает нагрузку на сервер в десятки раз. Существует и более сложное кэширование — так называемое горячее кэширование, когда после запроса администратора все страницы сайта заранее собираются и в таком виде хранятся в ожидании запроса.

Еще один недостаток существующих систем управления контентом — это их *привязка к платформе*. Не всегда провайдерские базы данных совпадают, к тому же и каждая платформа имеет свою специфику.

Типовая структура систем управления веб-контентом имеет трёхзвенную архитектуру технологии «клиент – сервер» (клиентом; сервером приложений; хранилищем данных).

Основные параметры систем управления веб-контентом можно сгруппировать в три категории.

1. Разработка контента. На этом этапе происходит создание, редактирование и утверждение контента, а роль системы заключается в автоматизации этих процессов. Задача поддержки совместной работы авторов, редакторов, программистов и менеджеров полностью перекладывается на систему. Эта задача осуществляется благодаря разделению контента и дизайна. Все компоненты сайта, включая шаблоны и наполнение, хранятся в определенных местах хранилища данных. Система же автоматически обращается в нужные места хранилища, позволяя множеству пользователей, даже не являющихся техническими специалистами, работать над подготовкой контента к публикации, включая проверку его достоверности.

2. Управление сайтом. На этом уровне происходит разработка самого сайта, предварительный просмотр и публикация подготовленного контента.

Здесь разрабатывается внешний вид, подготавливаются шаблоны, распределяются роли пользователей и производится классификация необходимой бизнес-информации (например, товары, цены). Важными компонентами этого уровня являются службы, поддерживающие своевременность поступления необходимого контента.

3. Доставка контента. Когда сайт полностью подготовлен к публикации, необходимы средства для динамического формирования веб-страниц в зависимости от вида конкретных пользователей. В этой связи одним из важных компонентов данного этапа является персонализация или распределение профилей, чтобы каждый пользователь получал только ту информацию, которая соответствует его роли.

Программное обеспечение WWW

Программное обеспечение World Wide Web можно разделить на группы по направлениям использования. Принята следующая классификация программного обеспечения World Wide Web:

- программы-клиенты (в том числе мультипротокольные программы-браузеры);
- программы просмотра документов в форматах, отличных от стандартных форматов Web;
- программы-серверы протокола обмена гипертекстовой информацией (веб-серверы);
- программы подготовки публикаций;
- поисковые машины;
- программы анализа статистики посещений.

Рассмотрим каждый из этих типов программ более подробно.

1. Программы-клиенты

Программы-клиенты — стандартны для World Wide Web. Наиболее распространенными некоммерческими программами этого типа являются *Mosaic* (графический интерфейс) и *Lynx* для алфавитно-цифрового режима доступа.

Mosaic — графический интерфейс доступа в WWW, интерпретирует язык гипертекстовой разметки HTML и позволяет обмениваться данными по протоколу HTTP.

Arena позволяет интерпретировать версии языка, которые в дополнение к возможностям, существующим в Mosaic, также реализуют математические формулы, обтекание графики текстом, прозрачные графические образы и ряд других изобразительных средств.

Lynx — полноэкранный интерфейс доступа к WWW с алфавитно-цифровых устройств типа терминала VT-100. Интерфейс поддерживает все возможности языка HTML 2.0 за исключением графики. Используя Lynx, можно не только просматривать базы данных WWW, но и обмениваться данными с CGI-скриптами, размещенными на удаленных серверах.

Line Mode Browser — самый простой интерфейс WWW. Он используется на любых устройствах отображения информации, в том числе и на терминалах типа TTY (телетайп).

2. Мультипротокольные программы-браузеры

В настоящее время на роль стандартов в этом классе программного обеспечения претендуют две программы: Netscape Communicator и Microsoft Internet Explorer. По своим возможностям и внешнему оформлению они довольно похожи. Основная задача этих программ — интерпретация разметки на языке HTML, интерпретация встроенных в HTML программ на одном из командных языков Web: JavaScript или VBScript, интерпретация Java-апплетов, разбор спецификации ресурсов сети (обработка URL), взаимодействие с серверами по протоколам прикладного уровня стека протоколов TCP/IP (примеры: Netscape Navigator (NN), Opera).

3. Программы просмотра файлов специальных форматов

Обычно вся информация, которая передается в рамках Web, просматривается программой-браузером. Однако некоторые информационные ресурсы требуют запуска дополнительного программного обеспечения. Наиболее часто используемыми являются:

- Acrobat для просмотра PDF-файлов;
- Ghost в комплекте с GhostView для просмотра файлов формата Postscript;
- программы просмотра аудио-, видеоинформации в формате MPEG;
- программы просмотра графических файлов;
- и ряд других.

4. Программы-серверы

Сервер WWW — программа, которая принимает запросы от WWW-клиентов и отвечает на них. В качестве ответа может быть возвращен HTML-документ, хранящийся в базе данных сервера, графический образ, аудиозапись, фильм или ответ внешней программы. Сервер обменивается данными не только с клиентами, но и с CGI-скриптами.

5. Программы подготовки публикаций

Все редакторы можно разделить на коммерческие и свободно распространяемые, а также на обычные и WYSIWYG. Коммерческие редакторы продаются и стоят около 100$ за одну установку. Как правило, можно скопировать редактор по Интернету и пользоваться им в течение 30 дней бесплатно, а затем уже решать вопрос о его приобретении (так называемые trial-версии).

Свободно распространяемые программы — среди этих программ следует выделить HTML Writer, HTMLed 1.2e, Web Wizard 16-bit и HotMetal Free.

6. Программы анализа статистики посещений

Программа анализа статистики посещений выдает статистику в виде ASCII-файлов, в которых содержится и графическое представление в виде, принятом для просмотра на алфавитно-цифровых дисплеях, подготавливает сводный отчет, в который входит общее количество посещений сервера за анализируемый период; общее число ошибок за исследуемый период; число перенаправлений; среднее число запросов в день; число обслуженных хостов; количество страниц, с которых осуществлялся доступ; число некорректных записей

в файле посещений; общее число байтов, переданных клиентам, и среднее число байтов, переданных за сутки.

7. Информационно-поисковые системы Интернета

Некоторые тенденции развития сетевых технологий (технологии Intranet)

Сервисы, предоставляемые Intranet, образуют вместе целостную сетевую инфраструктуру, в которой различают два вида сервис пользовательские и сетевые. Среди пользовательских сервисов выделяются четыре основных типа:

- создание и публикация документов;
- координация работ и взаимодействие пользователей информационной системы — системы электронной почты и средства коллективной работы (GroupWare);
- навигация (быстрый поиск и доступ к информации);
- доступ к приложениям.

К сетевым сервисам относятся: справочники — управление информацией о людях и ресурсах (единая справочная служба); репликация — прозрачное распространение данных по сети; безопасность; управление.

Информационно-поисковые системы Интернета могут быть разделены по функционально-структурному принципу на следующие классы:

- полностью распределенные системы, где реализуются принципы распределенных вычислений и распределенного хранения данных;
- частично распределенные — распределенные данные и локализованная обработка;
- локальные системы — локализованные данные и их обработка.

К первому типу относятся системы, использующие принципы WAIS. Здесь процесс поиска реализуется на совокупности распределенных по сети серверов, которые опрашивают друг друга при обработке запроса, причем исходные и промежуточные данные поиска имеют также распределенный характер.

К сетевым сервисам относятся: справочники — управление информацией о людях и ресурсах (единая справочная служба); репликация — прозрачное распространение данных по сети; безопасность; управление.

Ко второму типу относятся системы, использующие данные, находящиеся на веб-серверах, в качестве распределенных первичных ИР; вторичные и индексные данные сосредоточены на поисковом сервере, осуществляющем обслуживание пользователей.

Это такие системы, как AltraVista, Lycos, OpenText и пр.

Системы третьего типа — локальные, представляют собой функциональный аналог ранее рассмотренных BBS и онлайновых хостов, обеспечивая доступ удаленных пользователей к ресурсам, сосредоточенным на поисковом сервере; основным отличием от ранее рассмотренных средств является использование веб-технологий (HTTP, CGI, веб-серверов).

8. Информационно-поисковые системы World Wide Web

Прежде чем описать проблемы построения информационно-поисковых систем Web и пути их решения, рассмотрим типовую структуру такой системы.

В различных публикациях, посвященных конкретным системам, приводятся схемы, которые отличаются друг от друга только применением конкретных программных решений, но не принципом организации различных компонентов системы. Поэтому рассмотрим эту схему, где:

User Client — программа просмотра конкретного информационного ресурса. В настоящее время наиболее популярны мультипротокольные программы типа Netscape Navigator, MS Internet Explorer. Такая программа обеспечивает просмотр документов World Wide Web, Gopher, WAIS, FTP-архивов, почтовых списков рассылки и групп новостей Usenet. В свою очередь все эти информационные ресурсы являются объектом поиска ИПС;

User Interface — интерфейс пользователя — в данном случае это способ общения пользователя с поисковым аппаратом системы, т. е. с системой формирования запросов и просмотра результатов поиска. Просмотр результатов поиска и информационных ресурсов сети — это совершенно разные вещи, на которых мы остановимся чуть позже;

Search Engine — поисковая машина — служит для трансляции запроса пользователя, который подготавливается на информационно-поисковом языке (ИПЯ), в формальный запрос системы, поиска ссылок на информационные ресурсы сети и выдачи результатов и этого поиска пользователю;

Index — индекс — служит для поиска адреса информационного ресурса. Структура индекса организована таким образом, чтобы поиск происходил максимально быстро и чтобы при этом можно было определить ценность каждого из найденных информационных ресурсов сети;

Queries — запросы пользователя — сохраняются в его личной базе данных. На отладку каждого запроса уходит достаточно много времени, и поэтому чрезвычайно важно хранить запросы, поиск по которым привел к удовлетворительному результату;

Spider (Index Robot) — робот-индексировщик — служит для сканирования Internet и поддержки базы данных индекса в актуальном состоянии. Эта программа является основным источником информации о состоянии информационных ресурсов Сети;

Web Sites — это те информационные ресурсы, доступ к которым обеспечивает ИПС. Рассмотрим теперь назначение и принцип построения каждой из этих компонент более подробно, чтобы определить, в чем отличие данной системы от традиционной информационно-поисковой системы локального типа.

Конрольные вопросы

1. Что называется сервером в архитектуре «клиент – сервер»?
2. Что называется клиентом в архитектуре «клиент– сервер»?
3. Какие задачи решает сервер базы данных?
4. Какие задачи решает клиент?
5. Что такое сетевой сервис? Приведите примеры.
6. Какие виды имеет программная реализация сетевой службы?
7. Как классифицируются серверы приложений?
8. Какие функции выполняет сервер WWW?
9. Какие функции выполняет сервер приложений?

10. Что такое сервер Интернета?
11. Как классифицируют серверы Интернета?
12. Какие компоненты входят в структуру сервиса электронной почты?
13. Какое назначение имеет сервис TELNET?
14. Какое назначение имеет сервис FTP?
15. На какие группы делятся системы управления контентом?
16. Перечислите программное обеспечение WWW.

Лекция 2. Основы веб-программирования

План:

1. Основы веб-программирования: основные понятия и термины.
2. Веб-дизайн и веб-программирование.
3. Протоколы прикладного уровня: HTTP, FTP, POP, IMAP, SMTP Telnet. Их назначение и применение.
4. Взаимодействие с сервером HTTP. Компоненты запроса клиента и ответа сервера.
5. Веб-сервис, его функциональные блоки и конструктивные решения.

1. Основы веб-программирования: основные понятия и термины

Веб-программирование — это частный случай программирования *клиент-серверного приложения*.

Клиент-серверное приложение — это вид распределенной информационной системы.

Веб-программирование — это частный случай программирования распределенной информационной системы.

Распределенной системой в контексте разработки программных продуктов называют систему независимых приложений, которые зачастую выполняются на различных вычислительных машинах, обмениваются информацией по сетевым протоколам, но решают одну общую задачу или группу задач одного направления.

Самыми известными клиент-серверными информационными системами являются системы управления базами данных (СУБД), такие как Oracle и Microsoft SQL Server. Серверная их часть обслуживает файлы самой базы данных и обрабатывает SQL-запросы от десятков, сотен и тысяч клиентов.

Веб-программирование имеет дело с сервером, клиентом и сетевым протоколом, по которому сервер и клиент общаются. В данном случае сетевой протокол — это HTTP, веб-клиент — это интернет-браузер, а веб-сервер — приложение, которое умеет обрабатывать HTTP-запросы.

Наиболее популярными веб-серверами являются сервер Apache, который работает под управлением серверной операционной системы UNIX, и Internet Information Server (IIS), функционирующий под управлением серверной версии Microsoft Windows.

HTTP (HyperText Transfer Protocol) — это прикладной сетевой протокол на базе TCP/IP, который и предназначен для передачи гипертекста.

Документы для системы WWW подготавливаются с использованием языка HTML, который весьма прост в освоении, к тому же существуют специальные

HTML-редакторы и средства конвертирования в HTML-формат документов, подготовленных в среде популярных текстовых процессоров, например MS Word.

Технология WWW

WWW работает по принципу «клиент – сервер»: существует множество серверов, возвращающих по запросу клиента гипермедийный документ — документ, состоящий из частей с разнообразным представлением информации (текст, звук, графика, трехмерные объекты и т. д.), в котором каждый элемент может являться ссылкой на другой документ или его часть. Каждый информационный ресурс в глобальной сети Интернет имеет определенный адрес, и любой документ WWW способен ссылаться на другие документы. Программные средства WWW являются универсальными для различных служб Интернета, а сама информационная система WWW играет интегрирующую роль.

Технология WWW состоит из четырех компонентов:

- язык гипертекстовой разметки документов HTML (HyperText Markup Language);
- универсальный способ адресации ресурсов в сети URL (Universal Resource Locator);
- протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol);
- универсальный интерфейс шлюзов CGI (Common Gateway Interface).

Идея HTML — пример чрезвычайно удачного решения проблемы построения гипертекстовой системы при помощи специального средства управления отображением.

Вторым краеугольным камнем WWW стала универсальная форма адресации информационных ресурсов. ***Универсальная идентификация ресурса*** (Universal Resource Identification, URI) представляет собой довольно стройную систему, учитывающую опыт адресации и идентификации электронной почты, Gopher, WAIS, telnet, ftp и т. п. URL используется в гипертекстовых ссылках и обеспечивает доступ к распределенным ресурсам Сети. В формате URL можно адресовать как гипертекстовые документы формата HTML, так и ресурсы электронной почты, telnet, ftp, Gopher, WAIS.

Третьим компонентом World Wide Web является ***протокол обмена данными — HyperText Transfer Protocol (HTTP)***. Данный протокол предназначен для обмена гипертекстовыми документами и учитывает специфику такого обмена. Так, в процессе взаимодействия клиент может получить новый адрес ресурса в Сети (relocation), запросить встроенную графику, принять и передать параметры и т. п.

Последняя составляющая технологии WWW — ***Common Gateway Interface — общий шлюзовой интерфейс.*** CGI был специально разработан для расширения возможностей WWW подключением всевозможного внешнего программного обеспечения. Такой подход логично продолжал принцип общедоступности и простоты разработки и наращивания возможностей WWW. Предложенный и описанный в CGI способ подключения не требует дополнительных библиотек. Сервер взаимодействует с программами через стандартные потоки вво-

да/вывода, что упрощает программирование до предела. Программа, написанная в соответствии с общим шлюзовым интерфейсом CGI, называется сценарием CGI. CGI-сценарии могут быть написаны на любом языке программирования (C, C++, Pascal, Fortran и т. п.) или командном языке sll, cshell.

WWW — система прямого доступа, требующая полноценного подключения к Интернету и, более того, часто требующая быстрых линий связи для просмотра документов, содержащих много графики или другой нетекстовой информации.

Java-технологии

Одним из перспективных и многообещающих направлений развития Интернета и сетевых технологий в целом является Java.

Java — интерпретируемый язык с синтаксисом C++, специально рассчитанный на работу в открытой сетевой среде. Текст программы на Java может компилироваться в бинарный псевдокод и передаваться по сети для исполнения на виртуальной машине в удаленном интерпретаторе. При этом доступ к ресурсам рабочей машины для Java-программы может быть ограничен с целью обеспечения безопасности.

Маленькие программы на языке Java называются апплетами.

Java позволит решить самые глубокие проблемы WWW: отсутствие интерактивности, ограниченный контроль вида документа, ограниченный набор форматов встроенной графики и других объектов мультимедиа.

Сегодня Java применяется для передачи через Интернет апплетов — маленьких программ, обычно реализующих простые вещи для украшения www-страниц. Однако возможности и перспективы проекта Java уходят далеко за горизонты WWW.

Инструментальные средства создания сайтов

Для создания и просмотра веб-страниц требуется два программных продукта — текстовый редактор и браузер.

В настоящее время существуют программы для подготовки веб-страниц:

1. визуальные редакторы HTML, то есть редакторы типа WYSIWIG (What You See Is What You Get — что видишь, то и получаешь), например Microsoft FrontPage, Macromedia Dreamweaver и др. При работе в этих программах пользователь имеет дело с графическими образами элементов HTML, а не с кодом документа;

2. конверторы текстовых документов, подготовленных в текстовом процессоре Word, в формат HTML-файлов;

3. редакторы HTML-текстов, позволяющие автоматизировать набор и редактирование кода (например, HomeSite, HotDog, Ken Nesbitt, Web Editor и многие другие). Эти программы дают возможность пользователю быстро и легко вставлять в документ элементы HTML, проверять синтаксис команд, выполнять запуск и отладку страницы, не покидая окна редактора. Работа в таких редакторах напоминает работу в интерактивной среде программирования типа Delphi или Visual Basic;

4. программы, объединяющие в себе черты текстовых и визуальных HTML-редакторов. К таким программам относится Hot-Metal разработки компании SoftQuad Software.

2. Веб-дизайн и веб-программирование

Веб-дизайн. Основная функция веб-сайта — это передача информации. Лучшему восприятию сведений способствует графическое оформление и удобная компоновка материала. Хороший дизайн веб-сайта — это не просто удачное сочетание цветов и грамотно подобранные картинки. Это способ наглядного представления информации. Это удобная навигация, приятные и простые в работе меню, доступность популярных разделов и быстрый поиск остальной информации.

Перед тем как приступить к разработке дизайна сайта, заказчик рассказывает о назначении проекта и аудитории, для которой он предназначен. Обсуждается список сервисов, рисуется схема сайта.

Важный этап подготовки к работе — выделение наиболее посещаемых разделов и продумывание наиболее удобных, коротких путей перемещения по сайту. Если пути перемещения пользователей логичны, а информационные блоки расположены грамотно, работа с таким сайтом — удовольствие для пользователей. Неправильная навигация сайта затрудняет поиск информации, наиболее ценные сведения могут ускользнуть от внимания посетителей.

Создание качественного дизайна веб-сайта возможно только при тесном сотрудничестве и полном взаимопонимании заказчика и исполнителя. Терпеливая и слаженная работа представителей обеих сторон позволяет разработать удачную концепцию проекта и грамотно воплотить задумку в жизнь учетом всех пожеланий и требований клиента.

Качественный сайт имеет удобную навигацию, упорядоченные информационные блоки, все выложенные на сайте материалы должны иметь тематическую направленность, должны адресоваться определенному кругу пользователей. Посетитель ищет на страницах сайта информацию — тексты и рисунки. Свежая и полная информация по той или иной теме — гарантия популярности сайта. Графическое оформление сайта также играет немаловажную роль, как бы дополняет содержание страниц. Многие специалисты считают более успешными и перспективными так называемые динамические сайты.

Основные принципы веб-дизайна.

Первый принцип — удобная навигация. Таким образом, посетитель вашего сайта, имея определенный навык работы в Интернете, не будет теряться и выискивать нужные ссылки. Необходимо добавить, что доступ к интересуемой информации у посетителя должен быть по возможности облегчен.

Второй принцип — легкий «вес» веб-сайта. Чем быстрее он откроется вниманию посетителя, тем скорее он выполнит свою функцию.

Третий принцип веб-дизайна — анимация сайта. Задумайтесь, насколько она важна именно для вашего сайта. Возможно, в ней нет необходимости. Гармония фона и шрифта на страничке важна, но в меру. Имеется в виду заблуждение о том, что ярко-красные буквы на розовом фоне (или наоборот) состав-

ляют отличную цветовую гамму. Может быть, в живописи и так, но не в нашем случае. Попробуйте прочитать такой текст сами и убедитесь, что это очень нелегко.

Шрифты, используемые при оформлении веб-сайта, желательно выдержать в одном стиле, чтобы внимание посетителей не рассеивалось. Ведь вы можете только предполагать, что наиболее интересно на вашем сайте тому или иному клиенту.

Веб-дизайн — понятие многоплановое и разностороннее. От того, насколько профессионально оформлен веб-сайт, в немалой степени зависит успех вашего дела. Придерживаясь вышеизложенных принципов, вы сможете более конкретно определить свои пожелания при заказе оформления веб-сайта веб-дизайнеру или студии.

Веб-программирование. Прикладное программирование для Web начиналось с обработки запросов пользователя и динамической генерации страниц на стороне сервера. Эта же тенденция получила развитие и в языках программирования в виде вставок в HTML-документы. Затем появились языки программирования элементов HTML-документов на стороне клиента.

Программирование стороны клиента означает, что веб-браузер используется для выполнения какой-нибудь работы, которую он может выполнить, а результат для пользователя — это скоростная и более интерактивная работа на веб-сайте.

Языки веб-программирования: HTML, PHP, Perl, JavaScript, JScript, VBScript, и еще много разных замечательных универсальных и узкоспециальных языков описания сценариев.

HTML — не язык программирования в том смысле, как C++ или Visual Basic; он больше напоминает средства форматирования документов с использованием управляющих последовательностей.

В своих первых попытках повысить интерактивность HTML веб-страниц разработчики обратились к *сценариям* (scripting), добавляя функциональность путем комбинирования языка программирования с HTML. В результате зачастую получается странный гибрид кода и тегов, что вынуждает разработчиков вернуться к текстовым редакторам.

Скрипт — это «программа», которая создается в виде обычного текстового файла специального формата (сценария) и передается для обработки на заранее установленный в системе клиента интерпретатор.

По мере совершенствования технологии программ просмотра зависимость от платформы усилилась — она была привнесена компонентами ActiveX, технологией, основанной на **COM** — модели многокомпонентных объектов Microsoft (**Component Object Model**).

Компоненты ActiveX варьируются от причудливых элементов управления, таких как движки (spinners), до невизуальных компонентов, обеспечивающих доступ к базам данных или электронной почте. Подобные компоненты делают страницы в Internet Explorer более функциональными и привлекательными.

Документы ActiveX представляют собой программные объекты, которые могут загружаться и работать внутри ActiveX-контейнера, такого как Internet Explorer. Документы ActiveX позволяют разработчикам на Visual Basic немедленно применить свой опыт работы на Visual Basic для создания приложений для Интернета.

Dynamic (динамический) HTML позволяет посредством сценариев программно изменять теги.

Технология серверных сценариев, примером которой является **Active Server Page (ASP)**, позволяет создавать великолепные, не зависящие от платформ веб-страницы, которые можно просматривать любой программой просмотра.

В своей основе ASP — это сценарий, исполняемый на сервере IIS. Этот код динамически выполняется при запросе страницы, а получившийся HTML-текст отправляется программе просмотра.

К серверным языкам сценариев относятся также широко известные языки Perl и PHP. В отличие от ASP, они совместимы практически с любым веб-сервером, включая и IIS (PWS), которые по умолчанию поддерживают только ASP (IIS — сразу после установки, PWS — после установки свободно распространяемого модуля ASP.EXE).

3. Протоколы прикладного уровня: HTTP, FTP, POP, IMAP, SMTP Telnet. Их назначение и применение

Протокол — совокупность правил, регламентирующих процедуру коммуникации. Протоколы, используемые для обмена данными в локальных сетях, делятся по своей функциональности на три типа:

- прикладные;
- транспортные;
- сетевые.

Прикладные протоколы выполняют функции трех верхних уровней модели OSI — прикладного, уровня представления и сеансового. Они обеспечивают взаимодействие приложений и обмен данными между ними. К наиболее популярным прикладным протоколам относятся:

- FTAM (File Transfer Access and Management) — протокол OSI доступа к файлам;
- X.400 — протокол OSI для международного обмена электронной почтой;
- X.500 — протокол OSI служб файлов и каталогов на нескольких системах;
- SMTP (Simple Mail Transfer Protocol) — протокол Интернета для обмена электронной почтой;
- FTP (File Transfer Protocol) — протокол Интернета для передачи файлов;
- SNMP (Simple Network Management Protocol) — протокол для мониторинга сети, контроля за работой сетевых компонентов и управления ими;
- Telnet — протокол Интернета для регистрации на удаленных серверах и обработки данных на них;

- SMB (Server Message Blocks) — протокол взаимодействия рабочей станции и сервера фирмы Microsoft;
- NCP (NetWare Core Protocol) — протокол передачи данных между сервером NetWare и рабочей станцией фирмы Novell;
- Apple Talk и Apple Share — набор сетевых протоколов фирмы Apple;
- AFP (AppleTalk Filling Protocol) — протокол удаленного доступа к файлам фирмы Apple;
- DAP (Data Access Protocol) — протокол доступа к файлам сетей DECnet.

Транспортные протоколы реализуют функции транспортного и сеансового уровня модели OSI. Они инициализируют и поддерживают сеансы связи между узлами сети и обеспечивают требуемый пользователем уровень надежности передачи данных. Наиболее популярны из них следующие:

- TCP (Transmission Control Protocol) — протокол Интернета для гарантированной доставки данных, разбитых на последовательность фрагментов;
- SPX (Sequential Packet Exchange) — протокол стека IPX/SPX для передачи данных, разбитых на последовательность фрагментов, фирмы Novell;
- NetBIOS (Network Basic Input/Output System) — протокол устанавливает и контролирует сеансы связи между компьютерами;
- ATP (AppleTalk Transaction Protocol), NBP (Name Binding Protocol) — протоколы сеансов связи и транспортировки данных фирмы Apple.

Сетевые протоколы выполняют функции трех нижних уровней модели OSI — сетевого, канального и физического. Эти протоколы управляют адресацией, маршрутизацией, проверкой ошибок и повторной передачей кадров, обеспечивая услуги связи, и определяют правила осуществления связи в отдельных средах передачи данных, например Ethernet или Token Ring. К наиболее популярным сетевым протоколам относятся:

- IP (Internet Protocol) — протокол Интернета для передачи пакетов;
- IPX (Internetwork Packet Exchange) — протокол для передачи и маршрутизации пакетов фирмы Novell;
- NetBEUI — транспортный протокол, обеспечивающий услуги транспортировки данных для сеансов и приложений NetBIOS фирмы Microsoft;
- DDP (Datagram Delivery Protocol) — AppleTalk-протокол транспортировки данных фирмы Apple.

Кроме особенностей, обусловленных выполняемыми функциями, различия и особенности протоколов характеризуются их ориентацией на работу в различных операционных системах и с различными аппаратными платформами.

Принципы работы протоколов разных уровней

Протокол TCP/IP фактически представляет собой два различных протокола, работающих совместно. В локальной сети TCP/IP-пакеты упаковываются в «обертку» пакетов Ethernet, сами же TCP/IP-пакеты являются «оберткой» для HTTP. *Межсетевой протокол IP* (Internet Protocol — IP) был создан для использования в сложных сетях, объединенных из разнородных подсетей на основе коммутации пересылаемых пакетов. *Протокол управления передачей TCP* (Transmission Control Protocol — TCP) представляет собой высоконадежный

протокол, обеспечивающий взаимодействие между хостами, а точнее, между выполняемыми на них прикладными процессами в коммуникационных компьютерных сетях, а также в объединенных системах таких сетей. Протокол управления передачей является транспортным средством прикладных процессов, обеспечивающим для них интерфейс со средствами передачи данных межсетевого протокола.

Протокол SLIP (Serial Line Internet Protocol) позволяет изолированным компьютерам связываться с TCP/IP через телефонную сеть. Этот протокол определяет метод разбиения датаграмм на фреймы при передаче их по последовательному каналу и указывает конец одной и начало другой датаграммы. Хотя протокол SLIP вполне подходит для установления связи с дисковым набором, но недостатки в адресации, идентификации типа и сжатии данных делают его негибким, медленным и трудным в конфигурации.

Межузловой протокол PPP (Point-to-Point Protocol) был разработан для устранения недостатков SLIP; для PPP разработано несколько расширений, таких как опция предоставления имен серверов (DNS, Domain Names Service — служба доменных имен), обеспечение безопасной идентификации пользователя и объединение многочисленных соединений в одно логическое соединение с повышенной полосой пропускания.

Среди прочих существующих или находящихся в стадии разработки протоколов разработки фирмы Microsoft Corp. и других компаний можно назвать:

IPX/SPX (Internet Packet eXchange / Sequest Packet eXchange) — набор транспортных протоколов, используемых программным обеспечением NetWare фирмы Novell Corp.;

DECnet — используемый фирмой Digital Equipment Corp. транспортный протокол, предназначенный для связи систем Windows' NT с сетями DECnet;

AppleTalk — разработанный фирмой Apple Corp., Inc протокол для взаимодействия WINDOWS'NT с компьютерами Apple Macintosh;

XNS (Xerox Network Systems) — транспортный протокол, разработанный фирмой Xerox Corp. и использовавшийся в первых сетях Ethernet.

В **Интернете** часто применяются следующие протоколы:

TIME — наиболее простой протокол, с помощью которого извлекаются данные времени из соответствующего сервера;

NFS (Network File System) — сетевая файловая система, представляет собой скорее не протокол, а сетевую службу, позволяющую объединить в одно целое файловые системы нескольких удаленных машин. При этом пользовательские приложения, запущенные на рабочей станции, получают возможность работать с файлами и каталогами сетевой файловой системы так, как будто эта система является для данной рабочей станции локальной. Для передачи файлов, а также информации и содержимого каталогов и логических дисков NFS так же, как и TFTP, использует протокол пользовательских дейтаграмм.

Простой протокол управления сетью SNMP. SNMP (Simple Network Management Protocol) — простой протокол управления сетью, позволяющий производить мониторинг сетевой активности, а также осуществлять управление

рабочими станциями и коммутационным оборудованием сети. Мониторинг сети и управление осуществляются с помощью станции управления сетью, задача которой — контроль работы сетевых устройств, поддерживающих SNMP-агентов сети. Протокол дает возможность передавать между станцией управления и агентами сети информацию, представляющую собой как простую статистику функционировании того или иного сетевого агента, так и адресованные этим же агентам управляющие команды.

Протокол SMTP (Simple Mail Transfer Protocol) известен с 1980 года и был рассчитан на обмен почтой между «большими» ЭВМ (mainframe — мейнфрейм), которые имеют постоянное соединение (но не на имеющие случайное, непостоянное соединение ПЭВМ). Протокол SMTP протокол передачи почты, использующийся для передачи текстовых файлов и почтовых сообщений с использованием средств надежной доставки транспортного протокола TCP между сетевыми системами. SMTP-сообщение состоит из заголовка и непосредственно тела почтового сообщения. Заголовок содержит сведения обо всех промежуточных узлах, через которые данное сообщение прошло, эти сведения включают в себя сетевой адрес промежуточного узла, а также временную метку, поставленную на каждом из этих узлов. При передаче сообщения отправитель выступает в роли клиента почтовой службы, который инициирует соединение с получателем, являющимся сервером, и посылает запросы на обслуживание.

Работа протоколов стека IPX/SPX. Стек протоколов IPX/SPX имеет ряд особенностей, обусловливаемых тем, что входящие в него протоколы разрабатывались для работы в операционной системе Novell NetWare. Данная система изначально ориентировалась на функционирование в небольших локальных сетях, рабочие станции которых не имели особо мощных ресурсов. Стек IPX/SPX имеет структуру, схожую со стеком TCP/IP, соответственно и принцип его работы напоминает работу TCP/IP. NNTP (Network News Transfer Protocol) — относительно сложный протокол, служит для передачи новостей между серверами новостей и от сервера к клиенту.

Протокол POP (Post Office Protocol), созданный в 1984 году, снял ограничения протокола SMTP путем добавления двух новых функций — восстановления всех сообщений (в случае аварии) и удаления их с сервера (при успехе передачи). Текущая версия, POP3, добавляет несколько новых характеристик, сохраняя многое из структуры первоначальной версии. Однако и SMTP, и POP поддерживают только поток текста ASCII и не стандартизируют обмен данными современных форматов.

Формат MIME (Multipurpose Internet Mail Extension) появился в 1992 году и снял ограничения SMTP и POP в области передачи двоичных файлов (графика, мультимедиа и др.); промежуточные (взятые из ОС UNIX) преобразования «формат ASCII двоичный формат» UUEncode и UUDecode в былое время широко применялись в FidoNet.

Протокол IPX (Internetwork Packet eXchange — межсетевой обмен пакетами), разработанный компанией Novell, в некоторой степени является аналогом межсетевого протокола стека TCP/IP. Сетевой адрес IPX состоит из трех элементов и имеет следующую структуру:

- номер сети;
- номер узла;
- сокет.

Протокол SPX (Sequence Packet eXchange) является транспортным протоколом стека IPX/SPX. Этот протокол использует метод взаимодействия с установкой логического соединения и имеет средства обеспечения гарантированной доставки передаваемых данных.

Протокол объявления услуг SAP (Service Advertising Protocol) используется сервером сетевой операционной системы Novell NetWare для оповещения рабочих станций о предоставляемых им сетевых услугах, их адресах.

HTTP (HiperText Transfer Protocol) — протокол передачи гипертекста в Интернете. Достоинства http:

Создание виртуальных хостов. Протокол HTTP позволяет одному ВЕБ-серверу иметь несколько доменных имен. В настоящее время эта ситуация распространена широко (например, когда поставщик услуг Интернета поддерживает несколько доменов).

Консорциум W3C работает над протоколом HTTP-NG (Next Generation), который, как предполагается, заменит HTTP. К HTTP-NG предъявляются следующие требования: *простота* — протокол HTTP-NG должен быть прост для реализации и обслуживания; *расширяемость* — на случай ситуации, не предусмотренной в процессе разработки; *масштабируемость* — вне зависимости от того, используется ли HTTP-NG в маленькой локальной сети или в сети Интернет.

Эффективность — ожидается, что протокол HTTP-NG будет намного эффективнее HTTP. Последний плохо работает в сетях с большим временем задержки. Причина в том, что HTTP — протокол одиночных запросов и ответов. Кроме того, он перегружен информацией. Протокол HTTP-NG призван устранить эти и другие недостатки.

Протоколы уровня приложений

Основные протоколы прикладного уровня, обеспечивающие доступ к информационным ресурсам Интернет, а также соответствующее программное обеспечение (программы-клиенты и программы-серверы). К указанным средствам относятся:

- протокол эмуляции терминала Telnet;
- протоколы электронной почты SMTP, UUCP;
- протоколы распределенных файловых систем — NNTP, Gopher, FTP;
- протокол пересылки гипертекста — HTTP.

Каждый из перечисленных протоколов предполагает наличие некоторой совокупности команд (командный язык), которыми обмениваются программы-клиенты и программы-серверы данного протокола. Естественно, целью такого взаимодействия является обмен пользовательскими данными.

Протокол эмуляции удаленного терминала Telnet

Протокол эмуляции удаленного терминала Telnet — одна из самых старых информационных технологий Интернета. Протокол Telnet впервые описан в RFC-854 (май, 1983 г.).

Протокол telnet (telecommunications network) — протокол сети передачи данных телекоммуникационной сети, обеспечивающий передачу данных между процессами или между процессом и терминалом и обычно использующийся для эмуляции терминала удаленной станции.

Терминалом называют устройство, состоящее из средств ввода-вывода, обычно клавиатуры и монитора, и используемое для взаимодействия с хост-компьютерами, на которых выполняются прикладные программы.

Для передачи своих данных в качестве транспорта telnet использует протокол управления передачей, в сегменты которого инкапсулирует свои сообщения. Telnet достаточно простой и весьма распространенный протокол, присутствует практически в каждой реализации стека TCP/IP.

Сетевой виртуальный терминал (Network Virtual Terminal — NVT) — это стандартное описание наиболее широко используемых возможностей реальных физических терминальных устройств, что позволяет описать и преобразовать в стандартную форму способы отображения и ввода информации. Терминальная программа (user) и процесс (server), работающий с ней, преобразовывают характеристики физических устройств в спецификацию NVT, что позволяет, с одной стороны, унифицировать характеристики физических устройств, а с другой — обеспечить принцип совместимости устройств с разными возможностями.

4. Взаимодействие с сервером HTTP. Компоненты запроса клиента и ответа сервера

HTTP — протокол прикладного уровня для передачи гипертекста.

Центральным объектом в HTTP является *ресурс*, на который указывает *URI* в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы.

Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Данный протокол предназначен для передачи символьной информации. Данные в символьном виде занимают больше памяти, сообщения создают дополнительную нагрузку на каналы связи, однако подобный формат имеет много преимуществ. Сообщения, передаваемые по Сети, удобочитаемы, и, проанализировав полученные данные, системный администратор может легко найти ошибку и устранить ее.

В отличие от многих других протоколов, HTTP является протоколом без памяти. Это означает, что протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера. Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами.

Все программное обеспечение для работы с протоколом HTTP разделяется на три основные категории:

- *серверы* — поставщики услуг хранения и обработки информации (обработка запросов);
- *клиенты* — конечные потребители услуг сервера (отправка запросов);
- *прокси-серверы* для поддержки работы транспортных служб.

«Классическая» схема HTTP-сеанса выглядит так.

1. Установление TCP-соединения.
2. Запрос клиента.
3. Ответ сервера.
4. Разрыв TCP-соединения.

Таким образом, клиент посылает серверу запрос, получает от него ответ, после чего взаимодействие прекращается. Обычно запрос клиента представляет собой требование передать HTML-документ или какой-нибудь другой ресурс, а ответ сервера содержит код этого ресурса.

В состав HTTP-запроса, передаваемого *клиентом серверу*, входят следующие компоненты:

- строка состояния (строка-статус или строка запроса);
- поля заголовка;
- пустая строка;
- тело запроса.

Строка состояния имеет следующий формат:

метод_запроса URL_ресурса версия_протокола_HTTP

Рассмотрим компоненты строки состояния.

Метод, указанный в строке состояния, определяет способ воздействия на ресурс, URL которого задан в той же строке. Метод может принимать значения GET, POST, HEAD, PUT, DELETE и т. д. Несмотря на обилие методов, для веб-программиста по-настоящему важны лишь два из них: GET и POST.

GET. Метод GET предназначается для получения ресурса с указанным URL. Получив запрос GET, сервер должен прочитать указанный ресурс и включить код ресурса в состав ответа клиенту. Ресурс, URL которого передается в составе запроса, может представлять собой HTML-страницу, файл с изображением или исполняемый код программы, который должен быть запущен на сервере. Метод GET подходит для передачи небольших фрагментов данных на сервер.

POST. Основное назначение метода POST — передача данных на сервер. Однако подобно методу GET метод POST может применяться по-разному и нередко используется для получения информации с сервера. Метод POST также может использоваться для запуска процесса.

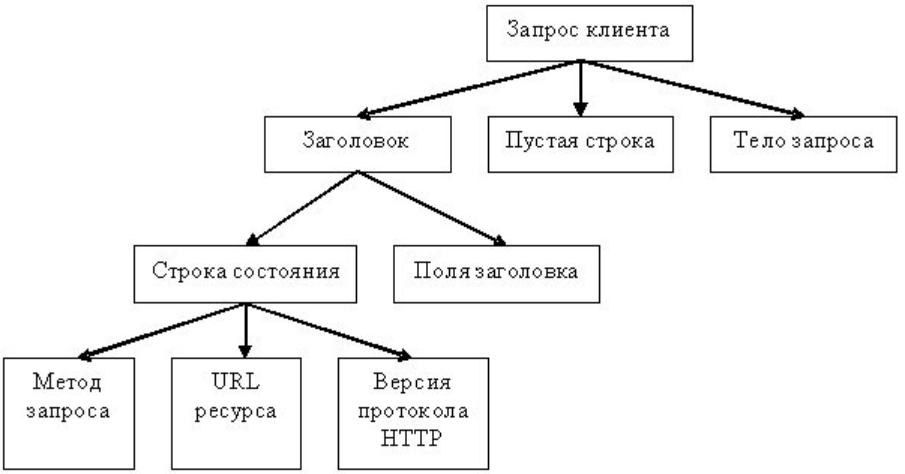


Рисунок 7 — Структура запроса клиента

Версия протокола http, как правило, задается в следующем формате:

HTTP/версия.модификация

Поля заголовка, следующие за строкой состояния, позволяют уточнять запрос, т. е. передавать серверу дополнительную информацию. Поле заголовка имеет следующий формат:

Имя_поля: Значение

Подобно запросу клиента, ответ сервера также состоит из четырех компонентов:

- строка состояния;
- поля заголовка;
- пустая строка;
- тело ответа.

Ответ сервера клиенту начинается со строки состояния, которая имеет следующий формат:

Версия_протокола Код_ответа Пояснительное_сообщение

Версия_протокола задается в том же формате, что и в запросе клиента, и имеет тот же смысл.

Код_ответа — это трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером.

Пояснительное_сообщение дублирует код ответа в символьном виде. Это строка символов, которая не обрабатывается клиентом. Она предназначена для системного администратора или оператора, занимающегося обслуживанием системы, и является расшифровкой кода ответа.

Для однозначной идентификации ресурсов в сети Web используются уникальные идентификаторы URL.

Единообразный идентификатор ресурса URI (Uniform Resource Identifier) представляет собой короткую последовательность символов, идентифицирую-

щую абстрактный или физический ресурс. URI не указывает на то, как получить ресурс, а только идентифицирует его. Это дает возможность описывать с помощью RDF (Resource Description Framework) ресурсы, которые не могут быть получены через Интернет (имена, названия и т.п.). Самые известные примеры URI — это URL и URN.

URL (Uniform Resource Locator) — это URI, который, помимо идентификации ресурса, предоставляет еще и информацию о местонахождении этого ресурса.

URN (Uniform Resource Name) — это URI, который идентифицирует ресурс в определенном пространстве имен, но, в отличие от URL, URN не указывает на местонахождение этого ресурса.

URL имеет следующую структуру:

<схема>://<логин>:<пароль>@<хост>:<порт>/<URL=путь>

где:

схема — схема обращения к ресурсу (обычно сетевой протокол);

логин — имя пользователя, используемое для доступа к ресурсу;

пароль — пароль, ассоциированный с указанным именем пользователя;

хост — полностью прописанное доменное имя хоста в системе DNS или IP-адрес хоста;

порт — порт хоста для подключения;

URL-путь — уточняющая информация о месте нахождения ресурса.

Поскольку протокол HTTP предназначен для передачи символьных данных в открытом (незашифрованном) виде, то лица, имеющие доступ к каналу передачи данных между клиентом и сервером, могут без труда просматривать весь трафик и использовать его для совершения несанкционированных действий. В связи с этим предложен ряд расширений базового протокола, направленных на повышение защищенности интернет-трафика от несанкционированного доступа.

Самым простейшим является расширение HTTPS, при котором данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивая защиту этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443. Чтобы подготовить веб-сервер для обработки HTTPS соединений, администратор должен получить и установить в систему сертификат для этого веб-сервера.

SSL (Secure Sockets Layer) — криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет. При его использовании создается защищенное соединение между клиентом и сервером. SSL изначально разработан компанией Netscape Communications. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший название TLS. Этот протокол использует шифрование с открытым ключом для подтверждения подлинности передатчика и получателя. Поддерживает надежность передачи данных за счет использования корректирующих кодов и безопасных хеш-функций. На нижнем уровне многоуровневого транспортного протокола (например, TCP) он является протоколом записи и используется для инкапсуляции различных протоколов (POP3, IMAP, SMTP или HTTP). Для каждого ин-

капсулированного протокола он обеспечивает условия, при которых сервер и клиент могут подтверждать друг другу свою подлинность, выполнять алгоритмы шифрования и производить обмен криптографическими ключами, прежде чем протокол прикладной программы начнет передавать и получать данные.

Для доступа к веб-страницам, защищенным протоколом SSL, в URL вместо схемы http, как правило, подставляется схема https, указывающая на то, что будет использоваться SSL-соединение. Стандартный TCP-порт для соединения по протоколу https — 443. Для работы SSL требуется, чтобы на сервере имелся SSL-сертификат.

В сети Web поддерживаются 3 типа аутентификации при клиент-серверных взаимодействиях:

1) Basic — базовая аутентификация, при которой имя пользователя и пароль передаются в заголовках http-пакетов. Пароль при этом не шифруется и присутствует в чистом виде в кодировке base64. Для данного типа аутентификации использование SSL является обязательным;

2) Digest — дайджест-аутентификация, при которой пароль пользователя передается в хешированном виде. По уровню конфиденциальности паролей этот тип мало чем отличается от предыдущего, так как атакующему все равно, действительно ли это настоящий пароль или только хеш от него: перехватив удостоверение, он все равно получает доступ к конечной точке. Для данного типа аутентификации использование SSL является обязательным;

3) Integrated — интегрированная аутентификация, при которой клиент и сервер обмениваются сообщениями для выяснения подлинности друг друга с помощью протоколов NTLM или Kerberos. Этот тип аутентификации защищен от перехвата удостоверений пользователей, поэтому для него не требуется протокол SSL. Только при использовании данного типа аутентификации можно работать по схеме http, во всех остальных случаях необходимо использовать схему https.

Cookie. Поскольку HTTP-сервер не помнит предыстории запросов клиентов, то каждый запрос обрабатывается независимо от других, и у сервера нет возможности определить, исходят ли запросы от одного клиента или разных клиентов.

Если сервер будет проверять TCP-соединения и запоминать IP-адреса компьютеров-клиентов, он все равно не сможет различить запросы от двух браузеров, выполняющихся на одной машине. И даже если допустить, что на компьютере работает лишь одна клиент-программа, то никто не может утверждать, что в промежутке между двумя запросами она не была завершена, а затем запущена снова уже другим пользователем.

Тем не менее, если вы когда-нибудь пользовались почтовым ящиком на mail.ru или на другом сервере, предоставляющем почтовые услуги пользователям Web, вспомните, как вел себя клиент после того, как вы создали для себя почтовый ящик на сервере. Когда вы в следующий раз обратились с того же компьютера к mail.ru, вы, вероятно, заметили, что после загрузки веб-страницы ваше регистрационное имя уже отображалось в соответствующем поле ввода.

Такие сведения позволяет получить дополнительное средство под названием cookie. Механизм cookie позволяет серверу хранить информацию на компьютере клиента и извлекать ее оттуда.

Инициатором записи cookie выступает сервер. Если в ответе сервера присутствует поле заголовка Set-cookie, клиент воспринимает это как команду на запись cookie. В дальнейшем, если клиент обращается к серверу, от которого он ранее принял поле заголовка Set-cookie, помимо прочей информации он передает серверу данные cookie. Для передачи указанной информации серверу используется поле заголовка Cookie.

Поле Set-cookie имеет следующий формат:

Set-cookie: имя = значение; expires = дата; path = путь;
домен = имя_домена, secure

Для передачи данных cookie серверу используется поле заголовка Cookie. Формат этого поля:

Cookie: имя=значение; имя=значение; ...

С помощью поля Cookie передается одна или несколько пар имя = значение. Каждая из этих пар принадлежит записи cookie, для которой URL запрашиваемого ресурса соответствуют имени домена и пути, указанным ранее в поле Set-cookie.

Веб-приложение — приложение, в котором клиентом выступает браузер, а сервером — веб-сервер.

Ко всем программам, которые передаются с сервера на клиент-машины и запускаются на выполнение, предъявляется одно общее требование: эти *программы должны быть лишены возможности обращаться к ресурсам компьютера, на котором они выполняются*. Такое требование вполне обосновано. Ведь передача по сети и запуск Java-апплетов и JavaScript-сценариев происходят автоматически *без участия пользователя*, поэтому работа этих программ должна быть *абсолютно безопасной для компьютера*. Другими словами, языки, предназначенные для создания программ, выполняющихся на клиент-машине, должны быть абсолютно непригодны для написания вирусов и подобных программ.

Код программы, работающей на сервере, не передается клиенту. При получении от клиента специального запроса, предполагающего выполнение такой программы, сервер запускает ее и передает параметры, входящие в состав запроса. Средства для генерации подобного запроса обычно входят в состав HTML-документа. Результаты своей работы программа оформляет в виде HTML-документа и передает их веб-серверу, а последний, в свою очередь, дополняет полученные данные HTTP-заголовком и передает их клиенту.

Rich Internet Application — еще один подход, который заключается в использовании программных модулей, например Adobe Flash или Java-апплетов, для полной или частичной реализации пользовательского интерфейса, поскольку большинство браузеров поддерживает эти технологии (как правило, с помощью *плагинов*).

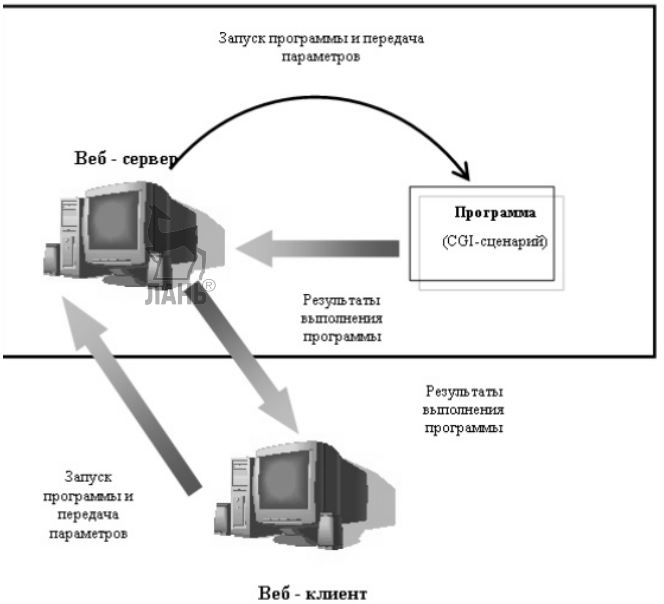


Рисунок 8 — Взаимодействие клиента с программой, выполняющейся на сервере Насыщенное интернет-приложение

Возникновение данного подхода обусловлено тем, что в рамках веб-приложений с «тонким» клиентом взаимодействие пользователя с приложением реализуется в существенной степени через сервер, что требует отправки данных на сервер, получения ответа от сервера и перезагрузки страницы на стороне клиента.

При использовании Java-апплетов в состав HTML-документа включается специальный дескриптор, описывающий расположение файла, содержащего код *апплета*, на сервере. После того как клиент получает HTML-код документа, включающего *апплет*, он генерирует дополнительный запрос серверу. После того как сервер пересылает клиенту код *апплета*, сам *апплет* запускается на выполнение.

5. Веб-сервис, его функциональные блоки и конструктивные решения

Для расширения возможностей клиент-серверного взаимодействия в рамках протокола HTTP помимо создания на клиентской стороне расширений стандартных возможностей, предоставляемых языками разметки и браузерами, можно также разрабатывать на стороне веб-сервера *приложения, плагины* и *сценарии*, расширяющие возможности самого веб-сервера.

Плагин (plug-in) — независимо компилируемый программный модуль, динамически подключаемый к основной программе, предназначенный для расширения или использования ее возможностей. Обычно выполняются в виде разделяемых библиотек.

Сценарий (скрипт, script) — программа, которая автоматизирует некоторую задачу, которую пользователь выполняет вручную, используя интерфейсы программы.

Веб-сервис — программная система, имеющая идентификатор URI и общедоступные интерфейсы которой определены на языке XML. Описание этой программной системы может быть найдено другими приложениями, которые могут взаимодействовать с ней в соответствии с этим описанием посредством сообщений, основанных на XML и передаваемых с помощью интернет-протоколов.

Веб-сервис является единицей модульности при использовании *сервис-ориентированной архитектуры* приложения.

Сервис-ориентированная архитектура

Сервис-ориентированная архитектура (*SOA*, *service-oriented architecture*) — модульный подход к разработке программного обеспечения, основанный на использовании сервисов со стандартизированными интерфейсами.

В основе *SOA* лежат принципы многократного использования *функциональных элементов* ИТ, унификации типовых операционных процессов. Компоненты программы могут быть распределены по разным узлам сети и предлагаются как независимые и слабосвязанные, заменяемые сервисы-приложения.

Интерфейс компонентов SOA-программы осуществляет инкапсуляцию деталей реализации конкретного компонента (ОС, языка программирования и т. п).

SOA хорошо зарекомендовала себя при построении крупных корпоративных программных систем. Целый ряд разработчиков и *интеграторов* предлагают инструменты и решения на основе SOA (например, платформы Microsoft .NET, IBM WebSphere, SAP NetWeaver, Diasoft и др.).

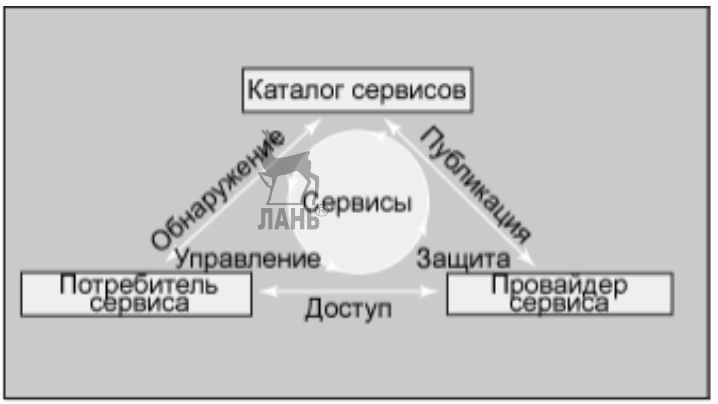


Рисунок 9 — Базовая архитектура SOA

Облачные вычисления (от *англ.* cloud computing) — технология обработки данных, основанная на том, что компьютерные ресурсы и вычислительные мощности предоставляются пользователю как интернет-сервис. При этом пользователь может имеет доступ к собственным данным, но не имеет возможности управлять (и не должен это делать) инфраструктурой, операционной системой и другим программным обеспечением, которые фактически и обеспечивают его работу.

Термин «облако» употребляется в качестве метафоры, которая подразумевает сокрытие от конечного потребителя всех технических деталей процессов, поддерживающих его работу.

Парадигма (система идей и понятий) облачной обработки данных предполагает, что вся необходимая информация постоянно хранится на удаленных серверах в сети Интернет и лишь временно кэшируется на клиентской стороне. Это могут быть персональные компьютеры, смартфоны, ноутбуки и т. д.

При этом пользователю такой системы предоставляются услуги, которые можно разделить на следующие виды:

- *IaaS* (Infrastructure-as-a-Service) — инфраструктура как сервис;
- *PaaS* (Platform-as-a-Service) — платформа как сервис;
- *SaaS* (Software-as-a-Service) — программное обеспечение как сервис.

Международный стандарт ISO 9241-11 определяет *usability* как «степень, с которой продукт может быть использован определенными пользователями при определенном контексте использования для достижения определенных целей с должной эффективностью, продуктивностью и удовлетворенностью».

При разработке пользовательских интерфейсов словом *юзабилити* обозначают общую концепцию их удобства при использовании программного обеспечения, логичность и простоту в расположении элементов управления.

Usability можно рассматривать как качественную характеристику того, насколько легким является использование интерфейса. Сам термин *usability* подразумевает методы, направленные на улучшение легкости-в-использовании, в процессе проектирования интерфейса.

XML и SOA. Архитектура SOA основывается на открытых стандартах и поддерживает платформенно-независимую бизнес-интеграцию, но она нуждается в общей технологии представления данных, на которой будет базироваться ее инфраструктура. Эта инфраструктура должна поддерживаться всеми участвующими сторонами. Чтобы служить основой для взаимопонимания, в центре этой инфраструктуры находится технология XML. Тому есть целый ряд причин:

- XML является фундаментом практически всех стандартов веб-сервисов, в том числе XML Schema, SOAP, WSDL (Web Services Description Language) и UDDI (Universal Description, Discovery, and Integration). Эти стандарты опираются на основополагающую концепцию основанных на XML представлений — поддерживаемый во всем мире формат обмена информацией между провайдерами сервисов и инициаторами запросов в SOA.
- использование XML решает проблему работы с различными форматами данных в различных приложениях, работающих на разных платформах;
- преимущество XML заключается в простоте представления, являющегося по своей природе текстовым, гибким и расширяемым.

Примеры стандартов, основанных на XML и используемых в SOA.

SOAP. Этот простой основанный на XML протокол позволяет приложениям обмениваться информацией по транспортным протоколам, таким как HTTP. Благодаря этим преимуществам SOAP является рекомендованным и самым широко используемым коммуникационным протоколом для веб-сервисов. А так как веб-сервисы являются краеугольным камнем архитектуры SOA, этот прото-

кол является также основным коммуникационным протоколом для основанных на SOA решениях.

WSDL. Это документ, написанный на XML и описывающий веб-сервис. Он определяет месторасположение сервиса и отображаемые им операции (или методы), позволяющие обращаться к этому сервису. WSDL-файл описывает четыре главные вещи:

1) сервисы, доступные через интерфейс веб-сервиса, такие как список имен методов и сообщений-атрибутов;

2) тип данных сообщений;

3) информация о связывании для транспортного протокола, такого как HTTP и JMS;

4) адрес сервиса, используемый для его вызова.

Реестры сервисов. Реестр сервисов представляет собой каталог сервисов, доступных в системе SOA. Он содержит физическое месторасположение сервисов, версии и их срок действия, а также документацию по сервисам. Реестр сервисов является одним из основных строительных блоков архитектуры SOA.

- Реестр сервисов реализует SOA слабое связывание. Храня месторасположения оконечных точек сервисов, он устраняет тесное связывание, приводящее к жесткой привязке потребителя к провайдеру.
- Реестр сервисов позволяет системным аналитикам исследовать корпоративный портфель бизнес-сервисов. Исходя из этого, они могут определить, какие сервисы доступны для автоматизации процессов с целью удовлетворения актуальных бизнес-потребностей, а какие нет. Это, в свою очередь, позволяет узнать, что нужно реализовать и добавить в портфель, формируя каталог доступных сервисов.
- Реестр сервисов может выполнять функцию управления сервисами, обязывая подписывающиеся сервисы быть согласованными. Это помогает гарантировать целостность руководства (governance) сервисами и стратегиями.

Контрольные вопросы

1. Что такое веб-программирование?
2. Что такое клиент-серверное приложение?
3. На чем основана технология WWW?
4. Какие категории ПО предназначены для работы с протоколом HTTP?
5. Что такое насыщенное интернет-приложение?
6. Что такое cookie?
7. Что такое облачные вычисления?
8. Что такое *usability*?
9. Какие аспекты имет хороший *usability* веб-сайт?
10. Для чего предназначен язык XML?

Лекция 3. Языки и средства гипертекстовой разметки

План:

1. Принципы гипертекстовой разметки. Структура гипертекстовых документов.

2. Понятие о стандартном обобщённом языке разметки SGML. Консорциум W3C.

3. Версии языка гипертекстовой разметки HTML. HTML-редакторы.

4. Универсальные редакторы веб-страниц.

5. Организация веб-страниц.

1. Принципы гипертекстовой разметки. Структура гипертекстовых документов.

Основой Всемирной паутины World Wide Web является язык гипертекстовой разметки HTML. Создатель этого языка Тим Бернерс-Ли, разрабатывая его для нужд одного крупного научного центра, даже не предполагал, что его детище вырастет до общемирового масштаба и будет определять развитие компьютерных технологий на многие десятилетия. Приступая к изучению HTML, нужно знать, что это не вполне обычный язык. Он не относится к языкам программирования высокого уровня, таким как C++, Java или Visual Basic. HTML, — это, прежде всего, язык разметки, и код, написанный на нем, исполняется на компьютере клиента в приложении веб-браузера. С этим связана его относительная простота и легкость освоения.

В случае веб-страниц пользователь имеет дело не с бумажными, а с электронными документами, полученными через Интернет. Принцип отображения документа средствами форматирования родительского приложения здесь неприемлем. Слишком много приложений или всевозможных конверторов пользователю пришлось бы иметь на своём компьютере, чтобы эффективно работать с множеством возможных форматов документов. Благодаря языку разметки HTML клиент Web может на экране своего компьютера просмотреть документ в том виде, в каком его задумал разработчик: с определенными размерами шрифта и разбивкой на абзацы, с определенным расположением рисунков, гиперссылок и пр. Текстовый документ, составленный на HTML, имеет размер в байтах в несколько раз меньший, чем размер аналогичного документа, подготовленного в текстовом процессоре (например, Word).

В языке HTML имеется множество тегов, среди которых — теги создания заголовка документа, задания параметров шрифта, вычерчивания линий, вставки гиперссылок, вставки графических элементов и т.д. В итоге веб-страница, кроме текста и ссылок, может содержать графику, звуки, видео, то есть иметь такой вид, который вы и видите на экране компьютера.

Гипертекст — это способ организации текста, графики и других данных, при котором элементы данных связаны между собой. Связанными могут быть как элементы одного документа, так и элементы различных документов. Гипертекстовая структура лежит в основе World Wide Web.

Веб-сайт — это набор веб-страниц, связанных между собой гиперссылками.

Веб-страницы создаются с помощью специального языка HTML. Сегодня кроме HTML применяются и другие языки разметки: WML, XML.

WML — язык разметки документов для использования в сотовых телефонах и других мобильных устройствах по стандарту WAP.

XML — универсальный язык разметки, предназначенный для описания структурированных данных, обмена информацией между программами и создания специализированных языков разметки (XML-словарей).

Принципы создания веб-страниц

Веб-страница, загружаемая в браузер, представляет собой HTML-файл — текстовый ASCII-файл с расширением .htm, .html. Файлы .htm создаются для обработки в операционной системе DOS (где принято расширения обозначать тремя буквами). Для работы в Windows 9x/2000 допустимо сохранять HTML-файлы с расширениями .htm и .html, причем более предпочтительным является .html. Если работа с файлами веб-документов будет выполняться в ОС Unix, можно использовать расширение .html.

В настоящее время для создания интерактивных сайтов применяются различные современные технологии: PHP, ASP, Perl, JSP, CSS, базы данных DB2, MsSQL, Oracle, Access и т.д.

2. Понятие о стандартном обобщённом языке разметки SGML. Консорциум W3C

Язык разметки SGML. История языков разметки уходит в 1960-е годы, когда сотрудники компании IBM взялись за решение задач переноса документов между различными платформами и операционными системами.

Результатом их усилий стал язык GML (General Markup Language — общий язык разметки), который предназначался для использования на ЭВМ семейства IBM. Язык GML в дальнейшем был расширен, а в 1980-х годах прошёл стандартизацию в ISO (Международная организация стандартизации).

Этот мощный и универсальный язык разметки, названный SGML (Standard General Markup Language), использовался военным ведомством США при оформлении технической документации. Однако SGML широкого распространения не получил ввиду своей сложности и дороговизны реализации.

Разработанный в 1989 году Бернерсом-Ли язык разметки HTML был основан на языке SGML. Средства разметки абзацев, заголовков, списков и прочие элементы, имеющиеся в HTML, были предусмотрены и в SGML. Заслуга изобретателя HTML состоит в том, что он ввёл в язык разметки то, чего не было SGML, — это гипертекстовые ссылки.

SGML, Standard Generalized Markup Language — стандартный обобщённый язык разметки; произносится [эс-джи-эм-эл]) — метаязык, на котором можно определять язык разметки для документов. SGML — наследник разработанного в 1969 году в IBM языка GML (Generalized Markup Language), который не стоит путать с Geography Markup Language, разрабатываемым Open GIS Consortium. Изначально SGML был разработан для совместного использования машиночитаемых документов в больших правительственных и аэрокосмических проектах. Он широко использовался в печатной и издательской сфере, но его сложность затруднила его широкое распространение для повседневного использования.

Консорциум W3C. W3C разрабатывает для Интернета единые принципы и стандарты (называемые «рекомендациями»), которые затем внедряются производителями программ и оборудования. Таким образом, достигается совмести-

мость между программными продуктами и аппаратурой различных компаний, что делает Всемирную сеть более совершенной, универсальной и удобной.

Миссия W3C: «Полностью раскрыть потенциал Всемирной паутины путём создания протоколов и принципов, гарантирующих долгосрочное развитие Сети».

Более конкретная цель W3C — помочь компьютерным программам достичь способности ко взаимодействию в Сети. Применение единых стандартов в Сети — это ключевой шаг для достижения такого взаимодействия.

Две другие важнейшие задачи консорциума — обеспечить полную «интернационализацию Сети» и сделать Сеть доступной для людей с ограниченными возможностями. Для решения первой задачи консорциум активно сотрудничает с организацией «Юникод» (англ. *Unicode*) и рядом других рабочих групп, занимающихся международным сотрудничеством в Интернете и языковыми технологиями. Для решения второй задачи консорциум не только сотрудничает с организациями соответствующего профиля, но и разработал свои собственные рекомендации, которые сейчас активно набирают популярность.

Рекомендации консорциума Всемирной паутины открыты, то есть не защищены патентами и могут внедряться любым человеком без всяких финансовых отчислений консорциуму. В отличие от других организаций, занимающихся разработкой стандартов для Интернета, консорциум Всемирной паутины не имеет программ сертификации (на соответствие рекомендациям консорциума) и не планирует их вводить, поэтому рекомендации W3C получили гораздо большее распространение, нежели стандарты любых других организаций. В то же время из-за отсутствия сертификации многие производители следуют рекомендациям лишь частично. Рекомендации консорциума построены таким образом, что частичное внедрение не нарушает общих стандартов. Некоторые популярные рекомендации имеют несколько степеней внедрения — кому как удобнее. Степени внедрения — это новое слово в сетевых стандартах, которое принесло консорциуму Всемирной паутины и его рекомендациям заслуженную популярность.

Рекомендации W3C зачастую хорошо проработаны и детализированы. С другой стороны, большинство рекомендаций доступны для любых категорий пользователей — от экспертов-программистов до начинающих веб-мастеров.

Любой стандарт W3C проходит 5 стадий согласования:

1. черновик спецификации;
2. рабочий проект;
3. последний созыв;
4. возможная рекомендация;
5. предлагаемая рекомендация,

и только после этого официально становится рекомендацией W3C.

3. Версии языка гипертекстовой разметки HTML. HTML-редакторы

Стандарт HTML 2.0 был утвержден в ноябре 1944 г. организацией IETF (Internet Engineering Task Force). В нем были расширены возможности предыдущей версии языка, и он получил широкое распространение как у профессионалов, так и любителей. В 1992 году Д. Раггетт из компании Hewlett-Packard

Labs составил более расширенную версию HTML, которая получила обозначение HTML+. Начиная с 1992 года, многие разработчики браузеров начали расширять HTML за счёт добавления собственных элементов. Это привело к тому, что документы, разработанные с учётом возможностей конкретного браузера, не отображались другими браузерами.

Поэтому группа под руководством Дэна Коннолли в 1994 году проанализировала наиболее широко используемые элементы HTML и составила новую версию HTML, которая получила обозначение HTML 2. Эта версия была утверждена Международной комиссией по стандартам в Интернете (IETF).

HTML 3.0 — проект версии языка был опубликован в марте 1995 г. В нем были произведены радикальные изменения предыдущих версий, включены дополнительные возможности, включая таблицы, математические выражения и т.д. Это стало причиной того, что он не стал официальной спецификацией и был заменён спецификацией HTML 3.2.

В январе 1997 года версия HTML 3.2 рекомендована в качестве стандарта. Данная версия получила популярность из-за совместимости с HTML 2.0.

Версия HTML 4.0 была опубликована в декабре 1997 г., в ней поддерживаются листы стилей, сценарии, фреймы, внедрённые объекты, улучшены средства работы с таблицами и формами.

Версия HTML 4.01 принята в конце 1999 г. Основной целью HTML 5 (с 2013 г.) является улучшить язык, поддерживающий работу с новейшими мультимедийными приложениями, при этом сохраняется лёгкость чтения кода для человека и ясность исполнения для компьютеров и приспособлений (веб-браузеры, синтаксические анализаторы и т. д.).

DHTML (Dynamic HTML) — «Динамический HTML»: развитие языка HTML для создания движущихся («динамических») эффектов на веб-страницах.

Редакторы HTML-файлов. Для разработки сайта используются различные средства:

- конструкторы сайтов (дизайнеры) WebCoder 1.6.0.0;
- профессиональные приложения: Macromedia HomeSite Plus v5.1 for Windows XP, Macromedia Dreamweaver, Microsoft FrontPage и т. д.

Для создания сайтов целесообразно использовать редакторы на русском языке Macromedia Dreamweaver 8.0.1 или FrontPage 2003.

4. Универсальные редакторы веб-страниц

Редакторы для верстки веб-страниц бывают двух типов: **визуальные** и **текстовые**. Визуальные редакторы не требуют от вас знаний html, css и прочих технологий для разметки страниц. В визуальном редакторе вы располагаете различные элементы вашего сайта, как будто на листе бумаги, а редактор пишет за вас код самостоятельно. Именно поэтому визуальные редакторы еще называют WYSIWYG-редакторами. Аббревиатура WYSIWYG расшифровывается как What You See Is What You Get — что видишь, то и получаешь. Однако следует заметить, что ни один визуальный редактор не совершенен и все они так или иначе ограничены в своих возможностях, поэтому от профессиональных

кодеров требуется умение писать код руками, именно поэтому профессиональным кодерам нужны текстовые редакторы. В этих редакторах вы пишете код своими руками. В текстовых редакторах, как правило, бывают разные функции, облегчающие кодеру написание кода, такие как подсветка кода (так легче видеть, где в коде вставлены стили или скрипты, а где просто текст), различные горячие кнопки и клавиши, которые вставляют уже готовые конструкции (куски кода, спецсимволы) в код, и т. д.

Какого же типа вам следует заводить редактор? Если вы изучаете html, css или другие технологии для разметки страниц, если вы хотите уметь создавать качественные страницы и быть профессиональным кодером, то, безусловно, вам нужен текстовой редактор. Если же у вас нет времени на изучение html, css и прочих технологий, если перед вами не стоит очень сложных задач в выполнении страницы, то смело заводите себе визуальный редактор и пользуйтесь им, он очень экономит время и силы. А лучше всего иметь у себя на компьютере и визуальный, и текстовой редакторы для разных нужд.

Визуальные редакторы

Macromedia Dreamweaver MX. Профессиональный инструмент для создания веб-сайтов и приложений. Разработчики утверждают, что Macromedia Dreamweaver MX предназначена для проектирования, разработки и администрирования профессиональных веб-сайтов и приложений. Кроме того, Dreamweaver легко интегрируется с другими программами от Macromedia, например, такими как Flash.

Dreamweaver гораздо больше чем просто визуальный редактор, это достаточно мощный и сложный инструмент, а всякий сложный инструмент требует, чтобы на его освоение было потрачено какое-то время, прежде чем пользователь сможет работать в нем.

Adobe GoLive и LiveMotion. Возможно, Adobe GoLive понравится тем, кто любит программы от Adobe и много с ними работал: знакомая среда, достаточно легко разобраться, что к чему. Кроме того, еще один плюс для любителей Adobe — все программы от Adobe прекрасно взаимодействуют друг с другом и дополняют друг друга, GoLive не исключение, он прекрасно «дружит» с собратьями.

Однако GoLive не более чем визуальный редактор для верстки веб-страниц, больше чем поддержки таких технологий, как HTML, DHTML, CSS, XML и нескольких готовых Javascript`ов, не стоит ждать от этой программы. Однако следует отметить, что встроенный редактор кода (текстовый) в этой программе очень хороший. То есть возможно, что GoLive является оптимальным сочетанием визуального и текстового редактора (два в одном).

Хочу также добавить, что, вероятно, стоит использовать с GoLive такую программу, как Adobe LiveMotion (в предыдущих версиях она называлось Image Styler). Хотя, по идее, LiveMotion является графическим редактором, в котором вы можете создавать незатейливую, но качественную графику для своих веб-сайтов, эта программа также умеет верстать веб-странички на основе того, что вы нарисовали (не лучшим образом, но довольно прилично, плюсом является то, что LiveMotion сам режет готовый макет на много картинок, которые вы потом можете использовать при верстке сайта в GoLive).

Microsoft FrontPage. Если верить разработчикам, то программа FrontPage позволяет создавать веб-узлы, обладающие широкими возможностями, а также предоставляет средства управления ими. На деле, FrontPage дружит с HTML, CSS, DHTML, Javascript. Дает достаточно широкие возможности по управлению изображениями и флеш-ролликами. Кроме того, FrontPage дружит с такими технологиями, как ASP, XML, VBScript, XSL. Также разработчики утверждают, что в последней версии FrontPage теперь борется за чистоту кода, что ж, остается надеяться, что это так, так как раньше данный пункт был не на высоте (раньше FrontPage вставлял в код очень много ненужных комментариев и другой лишней информации, так что правильностью и чистотой кода странички, сделанные в этом редакторе, похвастаться не могли).

Hotdog. Программа имеет простой и понятный пользователю интерфейс. Кроме того, что Hotdog «дружит» с пользователем, эта программа «дружит» также с Flash, SQL, PHP, ASP, умеет работать с GIF-изображениями (оптимизация, анимация), включает в себя HTML-компрессор, может создавать файлы справки (CHM).

Текстовые редакторы

Homesite. Этот редактор, пожалуй, самый популярный и мощный среди текстовых, и не зря. Кроме того, что в нем достаточно легко работать не только с HTML-кодом (есть все, от списка всевозможных атрибутов ко всем тегам вплоть до проверки кода (правильность проверяется с точки зрения W3C.org)), но также есть поддержка XHTML, CSS-редактор и т. д.

HTML Pad. Эта программа тоже пользуется большой любовью пользователей. Помимо всего стандартного HTMLPad поддерживает JavaScript, VBScript, SSI, ASP и Perl, умеет создавать макросы (наподобие Word и Excel), включает в себя кучу различных справочных материалов по CSS и Html и многое другое.

Notepad, он же Блокнот. В этой программе нет никаких функций, которые облегчат вашу жизнь: ни подсветки кода, ни вставки готовых конструкций кода, ничего, но зато эта программа есть в Стандартных на компьютере у каждого пользователя. С нее вы можете начать свои первые шаги в написании кода, а затем уже сменить на более понравившийся редактор.

Универсальные веб-редакторы

Namo Webeditor. Namo Webeditor — визуальный редактор для создания веб-сайтов, включает более 80 встроенных Java Script скриптов. С Namo Webeditor вы сможете создать и разработать ваш сайт практически без знания HTML-программирования, используя Namo как обычный текстовый процессор. Все, что вам нужно, — так это набрать текст и поместить вашу графику и все. Таблицы и кадры создаются очень легко, вы сможете изменять их размеры обычным щелчком мыши. Продвинутые пользователи могут использовать в своей работе и встроенный редактор HTML-кода. Редактор поддерживает цветную разметку тегов. Namo позволяет добавлять на ваши страницы Java-скрипты, VB-скрипты и Java-апплеты. Поддерживает DHTML- и CSS- стили. Namo WebCanvas — программа для создания и редактирования графических элементов для веб-дизайна. Программа позволяет в кратчайшие сроки с минимальным

уровнем знаний создать оптимизированные изображения для веб-сайта, позволяя вам экономить время при разработке проекта. Продуманный интерфейс программы и ее уникальные функциональные возможности позволяют называть Namo WebCanvas одним из лучших графических редакторов. Namo Capture — программа для захвата изображений. Namo Image Slicer — программа для разделения громоздкой страницы на части. Namo GIF Animator — программа для создания анимированных изображений (баннеров).

RSS Wizard — это программа для конвертирования HTML в RSS, которая может генерировать RSS из любой веб-страницы, не требуя предварительного редактирования ее кода. Программа имеет встроенный планировщик, XML-редактор и способна работать в полностью автоматическом режиме.

Serif WebPlus предлагает все необходимое для создания и размещения выдающихся интерактивных веб-сайтов — никакой опыт не требуется! С WebPlus любой может создать элегантный вебсайт в считанные минуты благодаря его уникальному подходу к созданию проекта Сети и немедленного доступа к интерфейсу. Теперь нет абсолютно никакой потребности изучать любые сложные языки программирования — WebPlus сделает всю работу за вас. Если вы когда-либо задавались вопросом, как ведущие веб-сайты добавляют динамические особенности типа blogs, опросов и галерей фотографии, или как они создают корзины посещения магазина, ответ обычно простой — они нанимают талантливых и опытных проектировщиков.

Web Page Maker. Удобный инструмент для создания веб-страниц, который позволяет вам создавать их на профессиональном уровне всего за несколько минут и не требует при этом знания HTML.

Денвер. Базовый комплект пакета Денвер, предназначенный для веб-программистов и дизайнеров, состоящий из дистрибутивов, используемый программистами и дизайнерами для отладки сайтов на домашней (локальной) Windows-машине без необходимости выхода в Интернет. Базовый пакет Денвер включает: Apache, SSI, mod_rewrite, mod_php; PHP с поддержкой GD и MySQL; MySQL с поддержкой транзакций (mysqld-max); phpMyAdmin (система управления MySQL через веб-интерфейс); ядро Perl без стандартных библиотек (они поставляются отдельно); эмулятор sendmail (отладочная «заглушка», складывающая приходящие письма в /tmp) и т. д.

5. Организация веб-страниц

Сайт является набором веб-страниц, объединенных общей тематикой и связанных между собой гиперссылками, единой системой навигации.

В зависимости от технологии создания можно выделить следующие типы сайтов.

1. Статические сайты, содержащие статические HTML- или XHTML- страницы. Статические веб-страницы — это статические файлы (набор текста, таблиц, рисунков и т. д.), которые создается с помощью языка разметки HTML (имеют расширение .html или .htm) и хранятся в готовом виде в файловой системе сервера.

2. Динамические сайты, в которых веб-страницы генерируются или формируются (создаются динамически) в процессе исполнения запроса пользователя. Динамические сайты бывают двух типов. В первом типе сайтов, веб-страницы генерируются или формируются из данных хранящихся на сервере в базе данных. Во втором типе сайтов веб-страницы генерируются на стороне клиентского приложения (в браузере).

3. Flash-сайты — это интерактивные приложения, разработанные в среде Macromedia Flash. Основным инструментом разработки flash-программ является векторная графика (интерактивная векторная анимация для Web). Flash придает сайтам динамичность и интерактивность.

4. Комбинированные сайты, в которых используются вышеизложенные технологии создания сайтов. Сайты по взаимодействию пользователя с ресурсами веб-страницы можно разделить на пассивные и активные или интерактивные.

- Пассивные сайты — это сайты с пассивными веб-страницами. В пассивных сайтах пользователь имеет возможность только просматривать информацию на веб-страницах.
- Интерактивные сайты — это сайты с активными веб-страницами. При работе с интерактивными веб-страницами пользователь имеет возможность обмениваться данными с сервером, участвовать в интерактивном диалоге.

Статические сайты с пассивными веб-страницами

Технология создания веб-страницы статических сайтов: язык HTML (Hyper Text Markup Language), который является языком разметки гипертекста, и каскадные таблицы стилей CSS (Cascading Style Sheets). CSS используется для оформления и форматирования различных элементов веб-страниц, в результате чего значительно снижаются размеры веб-страниц. Создание веб-страниц статических сайтов — это трудоемкий процесс. Статические сайты с пассивными веб-страницами создаются вручную, с помощью какого-либо редактора HTML в файловой системе компьютера, потом загружаются на сайт. Создание новых веб-страниц или редактирование существующих страниц пользователь выполняет на ПК в редакторе, а затем вновь загружает на веб-сайт.

В основном статические сайты с пассивными веб-страницами применяются для создания небольших и средних сайтов с постоянной структурой и внешним видом страниц, которые можно размещать на любых хостингах, в том числе на бесплатных, которые не поддерживают работу скриптов. Обучение школьников и студентов основам создания сайтов целесообразно начинать с создания статических сайтов с пассивными страницами, т. е. с изучения языка разметки HTML и каскадных таблиц стилей CSS. Для создания сайта используют различные средства: редакторы текста типа Блокнот, визуальные редакторы типа Microsoft FrontPage, Macromedia Dreamweaver и множество других редакторов, а также конструкторы сайтов (дизайнеры). Конструкторы веб-сайтов размещаются на некоторых сайтах в сети Интернет.

Статические сайты с интерактивными веб-страницами

Для придания статическим веб-страницам интерактивности и динамичности в веб-страницу можно вставлять скрипты на языках сценариев JavaScript и VBScript, исполняемых на стороне клиента. Скрипты на JavaScript и VBScript

могут исполняться либо при наличии каких-либо действий пользователя, либо автоматически во время загрузки веб-страницы. Кроме того, в HTML-документ можно вставлять элементы DHTML (динамический HTML). DHTML — это способ создания интерактивного веб-сайта. Динамический HTML построен на языке программирования JavaScript, каскадных таблицах стилей CSS и DOM (объектной модели документа). В документ HTML можно вставлять флеш-фрагменты или флеш-ролики (swf-файлы). В документ HTML можно вставлять флеш-формы, аналогичные HTML-формам. Флеш обеспечивает интерактивность за счет интерактивной векторной анимации для Web. Для создания флеш используется язык сценариев ActionScript.

Для обмена данными между пользователем и сервером в веб-страницу можно вставить веб-приложение, называемое HTML-формой (form). Форма — это часть веб-страницы, в которую пользователь может вводить свою информацию и отправлять ее на сервер, где размещена веб-страница, щелчком на кнопке. Запросы обрабатываются на сервере, который генерирует соответствующую выходную информацию. Запросы в форме могут выполняться методами GET или POST.

В связи с тем, что скрипты, исполняемые на стороне клиента, увеличивают объем веб-страниц, их количество и размер на странице должно быть ограниченным. Создание статических сайтов с интерактивными веб-страницами целесообразно выполнять в редакторе Macromedia Dreamweaver 8.

Динамические сайты, веб-страницы которых генерируются на стороне серверного приложения

Динамические сайты, веб-страницы которых генерируются или формируются из данных, хранящихся на сервере в базе данных.

В настоящее время для создания динамических сайтов применяются различные веб-приложения. Для разработки веб-приложений применяются различные технологии, обеспечивающие создание динамических веб-страниц. Динамические сайты способны реагировать на введенную пользователем информацию, т. е. могут быть интерактивными, поэтому динамические сайты, как правило, являются интерактивными, но не всегда. Для разработки веб-приложений используют два подхода:

- на основе компилируемых модулей;
- на основе интерпретируемых сценариев.

Компилируемые модули

Компилируемые модули — это модули типа CGI, которые транслируются в исполняемые файлы и выполняются веб-сервером. Первыми веб-приложениями для создания динамических сайтов были отдельные модули CGI (сценарии, созданные в основном на языке Perl), которые выполнялись на сервере. CGI-сценарии являются обыкновенными программами. Результатом выполнения модуля является страница в формате HTML.

Common Gateway Interface (CGI) — это стандартный интерфейс обмена данных, который определяет способ взаимодействия клиентского приложения и веб-сервера. CGI обеспечивает запуск скрипта на сервере и взаимодействие с ним. В дальнейшем для реализации этого подхода стали применять интерфейсы (серверные расширения) ISAPI и NSAPI.

Подход на основе интерпретируемых сценариев

В этом случае для создания сайта применяются серверные скрипты, так называемые языки сценариев. Код сценариев, как и HTML-код, является интерпретируемым кодом, поэтому HTML и сценарии можно комбинировать. Наиболее распространенные языки серверных скриптов: Perl, ASP, JSP, PHP, Cold Fusion, Python.

Сценарии взаимодействуют с объектами на сервере и генерируют выходную информацию в формате HTML. Тип серверного скрипта определяется по расширению имени файла (.php, .asp, .aspx, .jsp, .cfm). Если веб-сервер получает запрос на страницу такого типа, то он интерпретирует все содержащиеся в ней сценарии, в результате чего генерируется веб-страница в формате HTML, которая передается обратно браузеру.

Наиболее популярными технологиями (средой разработки) создания динамических веб-страниц являются: CGI, PHP, ASP, ASP.NET, JSP, Cold Fusion, AJAX, Python, CSS, базы данных DB2, MsSQL, Oracle, Access и т. д.

В зависимости от решаемых задач для создания сайта выбирают тот или иной язык серверных скриптов. Для создания малых и средних интерактивных сайтов целесообразно применить язык сценариев PHP. Конкурентами PHP являются технологии ASP, JSP, Cold Fusion, Perl. Достоинством языка PHP является то, что он бесплатный, имеет открытые исходные коды и работает почти на всех платформах.

Для создания (разработки) и сопровождения динамических сайтов используют CMS (Content Management System) — систему управления сайтом, которую называют движком сайта. В настоящее время популярными системами управления являются **Drupal**, **Joomla** и **WordPress**. На основе этих CMS можно создавать функциональные и легкоуправляемые PHP-сайты.

Движки для Drupal**,** Joomla и WordPress являются бесплатными. Средства разработки сайтов обеспечивают разделение содержательной части (контента) от дизайна (шаблона веб-страницы), что позволяет изменять содержание веб-страниц, не затрагивая их дизайна, и изменять шаблон сайта, не затрагивая содержания его страниц.

Для поддержки учебного процесса традиционного обучения студентов целесообразно создавать динамические интерактивные сайты, например с помощью движка Joomla. Этот движок имеет множество модулей: форумы, гостевые книги, почтовые рассылки, контакты, опросы, формы регистрации, формы поиска, систему обмена сообщениями между пользователями сайта и другие компоненты, которые превращают сайт из средства информации в средство коммуникации. В этом случае сайт будет местом активного обмена информацией между пользователями Интернета (студентами и преподавателями). На такой сайт пользователи могут самостоятельно добавлять электронные учебные материалы, статьи, фотографии, видео, бесплатно скачивать образовательные ресурсы, т. е. пользователи имеют возможность обмениваться данными с сервером. Кроме того, студенты через опросы могут оценивать работу преподавателей, высказывать свое мнение по различным вопросам, общаться между собой, т. е. участвовать в интерактивном диалоге.

Динамические сайты, веб-страницы которых генерируются на стороне клиентского приложения

Динамические сайты, веб-страницы которых генерируются на стороне клиентского приложения. Для создания таких сайтов используют языки сценариев JavaScript и VBScript, а также Java-апплеты и технологию ActiveX. В Интернете можно скачать движок для создания сайта на JavaScript для бесплатных хостингов.

Flash-сайты. Технология флеш предназначена для создания векторных графических приложений. С помощью флеш можно создать полноценную страницу-ролик для Web, но при этом информация разбивается на крупные файлы, для загрузки которых требуется много времени. В настоящее время более целесообразным является применение флеш в качестве элементов дизайна в HTML-документах (например, для создания логотипов, флеш-меню, информеров и других анимированных графических элементов), в качестве анимированных флеш-баннеров и входных флеш-заставок.

Флеш целесообразно использовать там, где мало текста, но где требуются звуковые или анимационные эффекты, т. е. там, где флеш обеспечивает создание векторных анимационных файлов с небольшим временем загрузки. Основные недостатки этой технологии создания полноценных флеш-сайтов: большой вес веб-страниц и высокая стоимость разработки сайтов. Кроме того, сайты, созданные полностью на основе флеш, плохо индексируют поисковые системы. Флеш-технологии в основном применяются для создания престижных сайтов. Для создания флеш-анимаций применяют технологию Adobe Flash, которая обеспечивает возможность работать с языками ActionScript и ActionScript 2.0.

Контрольные вопросы

1. Что такое гипертекст?
2. Что такое управляемые сайты?
3. Что такое веб-хостинг?
4. Какие этапы включает процесс создания сайта?
5. Перечислите основные части документа SGML.
6. Что такое консорциум W3C?
7. Какие средства используются для разработки сайта?
8. Как классифицируют сайты в зависимости от технологии создания?

Лекция 4. Язык гипертекстовой разметки HTML

План:

1. Общие сведения о языке HTML.
2. Структура HTML-документа.
3. Обзор команд языка HTML.
4. Форматирование HTML-документа и текста.
5. Таблицы.
6. Графика.
7. Списки.

8. Ссылки.
9. Формы.
10.Фреймы.

1. Общие сведения о языке HTML

HyperText Markup Language (HTML) является стандартным языком, предназначенным для создания гипертекстовых документов в среде Web. HTML-документы могут просматриваться различными типами веб-браузеров. Когда документ создан с использованием HTML, веб-браузер может интерпретировать HTML для выделения различных элементов документа и первичной их обработки. Использование HTML позволяет форматировать документы для их представления с использованием шрифтов, линий и других графических элементов на любой системе, их просматривающей.

HTML — это язык тегов (заключённых в угловые скобки кратких предписаний, определяющих параметры отображения информации). Созданная с помощью HTML страница отображается на различных экранах (VGA, SVGA и др.) не абсолютно одинаково, однако с соблюдением некоторых общих правил отображения.

HTML — очень простой язык для разметки страниц, использующий исключительно текстовые команды (при этом для выполнения некоторых команд, например загрузки файлов изображений, мультимедиа и др., используется двоичный формат данных).

Разметка документа — это описание различных фрагментов документа и их взаимного расположения. Выполняется разметка с помощью символов ASCII, а точнее арабских цифр, символов латинского алфавита и некоторых знаков препинания. Из этих символов набираются команды языка HTML — теги, или, иначе говоря, дескрипторы

Достоинства HTML.

1. Основное преимущество HTML заключается в том, что ваш документ может быть просмотрен на веб-браузерах различных типов и на различных платформах.

2. Информация в коде HTML занимает значительно меньший объем, чем та же информация, разработанная в других приложениях Windows (например, размер DOC-файла — 19 Кб, а в формате HTML — 3 Кб).

3. Для подготовки информации, в том числе и составления HTML-кода, желательно применять простые текстовые редакторы типа Блокнот (Notepad), входящие в набор стандартных программ Windows.

Основные правила синтаксиса HTML

- каждый тег должен начинаться с открывающейся угловой скобки, а заканчиваться закрывающейся угловой скобкой;
- браузеры игнорируют нестандартные теги и атрибуты, поэтому имена тегов и их атрибутов должны соответствовать перечню допустимых в HTML имен;
- при записи имен тегов можно пользоваться как верхним, так и нижним регистрами, но нельзя ставить пробелы;

- значения атрибутов можно записывать в любом регистре (за исключением пользовательских значений атрибутов, для которых соблюдается соответствие имен с точностью до регистра);
- если значение атрибута содержит пробелы, оно обязательно должно быть заключено в кавычки;
- в содержимом документа браузеры игнорируют несколько пробелов, следующих подряд, и сжимают их до одного пробела;
- элемент, включающий в себя начальный тег другого элемента, должен содержать и конечный тег этого элемента (за исключением одиночных тегов);
- допускается включать в HTML-документы комментарии.

2. Структура HTML-документа

Разметка HTML-документов выполняется с помощью тегов, которые записываются с соблюдением определенных правил. Теговая модель предполагает разбиение документа на отдельные фрагменты, которые заключаются в теги или начинаются тегом.

Тег (tag — указатель, метка) — это код, идентифицирующий определенный элемент документа HTML, например, абзац, заголовок, ссылку, таблицу и т. д.

Теги заключаются в угловые скобки < >.

Теги HTML являются подмножеством тегов языка SGML. Если какие-либо из тегов непонятны браузеру, они при анализе документа просто игнорируются. HTML-теги могут быть условно разделены на две категории:

- теги, определяющие, как будет отображаться веб-браузером тело документа в целом;
- теги, описывающие общие свойства документа, такие как заголовок или автор документа.

Приведем простой пример HTML-документа:

```
<HTML>
<HEAD><TITLE>Пример веб-документа</TITLE> </HEAD>
 <BODY> <H1> Моя первая веб-страница</H1></B0DY>
</HTML>
```

Парные и одиночные теги, контейнеры

Все теги начинаются с открывающейся угловой скобки <, за которой следует текст, определяющий содержание тега, например TITLE или BODY. Оканчивается тег закрывающейся угловой скобкой >. Содержанием тега может быть просто имя тега либо имя и набор атрибутов тега. Большинство тегов являются парными, то есть для каждого начального тега <Имя> есть конечный тег </Имя>, в котором к имени тега добавляется косая черта "/" (слеш), например:

```
<HTML>...</HTML>
<HEAD>…</HEAD>
```

Здесь многоточие означает, что между начальным и конечным тегами может находиться текст и/или другие теги. Парные теги предназначены для описания содержимого документа: заголовка, абзаца, элементов таблицы и т.д.

Кроме парных, возможны одиночные теги, т. е. теги, в которых имеется только открывающий тег. В соответствии с инструкциями одиночных тегов браузер выполняет определенные действия. Например, согласно тегу
 выполняется разрыв текстовой строки, а в соответствии с тегом <P> формируется новый абзац. При наличии тегов <HR>, <IMG> или <EMBED> производится вставка горизонтальной линии, изображения или звукового файла соответственно.

Имена всех тегов стандартизированы и удобны для запоминания. Например, имя тега <BODY> (в переводе «тело») обозначает основную часть документа, а тег <I> (сокращение от italic — курсив) задает начертание курсивом.

Имена в парных тегах должны быть одинаковыми. Отклонение от этого правила, например <FONT>. </FNT>, будет считаться ошибкой. Однако в некоторых случаях можно опускать закрывающие теги

Пара тегов, состоящая из начального и конечного тегов, называется контейнером. Контейнеры обозначаются по имени начального тега и записываются в угловых скобках. Два выражения «пара тегов <TITLE></TITLE>» и «контейнер <TITLE>» обозначают одно и то же.

Элементы HTML

Документ HTML включает в себя элементы, которые представляют абзацы, заголовки, гиперссылки, списки, таблицы, рисунки и пр. Вообще весь документ можно рассматривать состоящим из определенных элементов.

Элемент — это пара тегов и символьные данные (текст или код), заключенные между ними. То есть элемент состоит из трех компонент: начального тега, содержимого и конечного тега. В некоторых элементах конечный тег может быть опущен (в случае одиночных тегов).

Элемент называется обычно по имени тега (без угловых скобок). Некоторые элементы могут не иметь содержимого, например элемент разрыва строки BR. В таких элементах отсутствует конечный тег

Описывая синтаксис элементов HTML, необходимо указать, что они не чувствительны к регистру символов, то есть браузер одинаково воспринимает теги <TITLE>, <Title> или <title>. Однако для единообразия обозначений элементов мы будем пользоваться прописными буквами. Следует различать термины «элемент» и «тег». Элемент обязательно включает в себя хотя бы один тег и, возможно, содержимое, в то время как у тега содержимое отсутствует.

Классификация элементов

Все элементы, предусмотренные в HTML, можно условно разбить на несколько категорий:

структурные — это элементы, которые обязательны для документа, соответствующего стандарту HTML (например, элементы HTML, HEAD, BODY и TITLE);

блоковые — элементы, которые предназначены для форматирования целых текстовых блоков (например, элементы BLOCKQUOTE, DIV, HI, H2, H3, H4,

H5, H6, P, PRE); часто блоковые элементы отделяются переводом строки от остального содержания документа;

текстовые — элементы, которые задают разметку текста (EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ABBR, ACRONYM), разметку шрифта (I, B, U, TT, BIG, SMALL, SUB, SUP);

специальные — элементы пустой строки (BR, HR, NOBR), якорный элемент A, внедренные элементы (EMBED, IMG, BGSOUND, OBJECT, MAP), элементы формы (INPUT, SELECT, TEXTAREA), элементы таблицы (TABLE) и др.

Вложенные элементы

В документе HTML обязательно присутствуют вложенные элементы, то есть элементы, включенные в состав других элементов. Последовательность, в которой допустимо размещать HTML-теги, то есть правила вложения элементов HTML, зависит от их категории. При составлении кода страницы соблюдайте следующие правила.

Структурные элементы могут включать в себя элементы других категорий. Например, элемент <HTML> является внешним по отношению ко всем остальным элементам. Текстовые элементы также могут быть вложенными, но они не могут включать блоковые элементы. Например, недопустимо, чтобы элементы верхних индексов SUP содержали элементы абзаца P или списков UL.

При составлении HTML-кода нужно следить за правильным написанием имен вложенных парных тегов. Для закрытия тегов лучше придерживаться последовательности: от последнего к первому. Понятно, что подобных проблем не возникает с одиночными тегами.

Элементы должны вкладывать друг в друга таким образом, чтобы каждый внутренний элемент располагался внутри одного и того же внешнего элемента, например:

<OL><UL>Пример правильного вложения элементов</UL></OL>

Пример неправильной последовательности в записи тегов:

<OL><UL> Пример некорректного вложения элементов </OL></UL>

Пустые элементы

Кроме перечисленных выше разновидностей элементов, в HTML предусмотрены пустые элементы. Такие элементы не включают в себя какие-либо текстовые фрагменты и другие элементы. Пустой элемент состоит только из одиночного тега, например тег <HR> является в то же время элементом горизонтальной линии HR.

Атрибуты тегов

Часто теги, помимо имени, содержат дополнительные элементы, которые называются атрибутами. *Атрибуты* — это компоненты тега, содержащие указания о том, как браузер должен воспринять и обработать тег.

В теге может быть несколько атрибутов. Тогда атрибуты отделяются друг от друга пробелами, символами табуляции или возврата к началу строки. Очередность записи атрибутов в теге значения не имеет. Атрибуты записываются только в начальных тегах и отсутствуют в конечных тегах. Значение атрибута указывается после имени атрибута через знак равенства (символ =). Допускается при-

менение пробела до и после знака =, например, записи COLOR="blue", COLOR= "blue" и COLOR ="blue" будут восприняты браузером одинаково.

Все значения атрибутов по умолчанию должны заключаться в двойные (") или одинарные (') кавычки. Имена атрибутов могут набираться как строчными, так и прописными буквами, — браузер будет интерпретировать их одинаковым образом.

Однако для атрибутов элементов HTML мы будем пользоваться прописными буквами, например <INPUT TYPE="button" NAME="rest" VALUE="Сведения">

Если имя атрибута не чувствительно к регистру, то при записи значений атрибутов регистр символов все же нужно учитывать, особенно это касается пользовательских значений, например, имен файлов в качестве значений атрибутов. Для значений, предусмотренных стандартом HTML, учет регистра не обязателен, например значение "button" можно записать как "BUTTON", "Button" и даже buTTon или BUTTON. Имена атрибутов мы будем записывать строчными буквами, например:

<INPUT type="button" name="rest" value="сведения">

Имена элементов и атрибутов должны записываться в соответствии со спецификацией HTML. Если браузер в коде документа встретит тег, отличающийся по написанию от принятого в спецификации, он его просто проигнорирует.

Комментарии

Комментарии — это произвольные фрагменты текста, которые не исполняются браузером. С помощью комментариев вы можете пояснять назначение различных фрагментов кода. Это бывает полезно, когда вы возвращаетесь к анализу страницы по прошествии какого-то времени. Комментарии облегчают также труд другим разработчикам, которые вдруг станут разбирать ваш код.

Комментарии располагаются между группами символов <!-- и -->, которые обозначают фрагмент неисполняемого кода. Стандартом HTML предусмотрено, что все браузеры игнорируют любой текст, оформленный таким образом.

3. Обзор команд языка HTML

Границы документа (элемент HTML)

Для обозначения границ HTML-документа используется парный тег <HTML>. Начальный тег <HTML>, не имеющий атрибутов, располагается в самом начале HTML-файла, а конечный тег </HTML> является последним тегом кода и обозначает окончание всего документа. В состав контейнера HTML входят два структурных элемента: HEAD (элемент заголовка) и BODY (основная часть или тело документа). Таким образом, документ HTML представляется в виде:

```
<HTML>
<!-- Здесь располагаются элементы заголовочной и основной частей документа -->
</HTML>
```

Заголовочная часть документа (элемент HEAD)

Заголовочная часть HTML-документа определяется элементом HEAD. Тег <HEAD> не имеет атрибутов. Элемент HEAD размещается сразу после тега <HTML> и предшествует основной части веб-страницы:

<HTML>
<HEAD>
<! -- Здесь располагается заголовочная часть документа -->
</HEAD> </HTML>

Элемент HEAD (как и элемент HTML) не отображается при просмотре веб-страницы, он предоставляет браузеру общую информацию о HTML-файле. Между тегами <HEAD> и </HEAD> могут размещаться элементы BASE, LINK, META, SCRIPT, STYLE и TITLE.

Элемент META позволяет автору документа определять информацию, не имеющую отношения к HTML. Эта информация используется браузером для действий, которые не предусмотрены текущей версией HTML спецификации.

<META HTTP-EQUIV=refresh" CONTENT="60" URL="www.fdline.org/homepage.html">

Браузеры Netscape Navigator и Internet Explorer поймут эту запись как инструкцию ожидать 60 секунд, а затем загрузить новый документ. Такая инструкция часто используется при изменении местоположения документов. Небольшой документ с приведенной строкой может быть оставлен на старом месте расположения документа для автоматической ссылки на его новое место расположения. Следующая командная строка <META HTTP-EQUIV=refresh" CONTENT="60"> инструктирует браузер перезагружать страницу каждые 60 секунд.

Таблица 1 – Атрибуты элемент META

Атрибут	Назначение
HTTP-EQUIV	Определяет свойство для элемента
NAME	Обеспечивает дополнительное описание элемента. Если этот атрибут опущен, он считается эквивалентным атрибуту HTTP-EQUIV
URL	Определяет адрес документа для свойства
CONTENT	Определяет возвращаемое значение для свойства

Название документа (элемент TITLE)

В заголовочную часть документа входит обязательный элемент, представленный контейнером <TITLE>. Все, что находится между парой тегов <TITLE> и </TITLE>, интерпретируется браузером как название веб-страницы (не нужно путать с именем HTML-файла). Содержимое контейнера <TITLE> отображается в заголовке окна браузера.

Если в документе имеются гиперссылки, то название документа, на который указывает ссылка, будет появляться во всплывающей подсказке при наведении на ссылку указателя мыши.

Элемент TITLE по отношению к элементу HEAD является дочерним, то есть вкладывается в контейнер <HEAD>:

<HTML><HEAD>
<TITLE> название веб-страницы</TITLE>
…</HEAD>… </HTML>

Название веб-страницы — это важный элемент, который имеет значение при предоставлении информации для поисковых систем и может привлечь

внимание потенциальных посетителей. Нужно выбирать краткие содержательные названия. Оптимальный размер названия — от 2 до 6 слов.

Тело документа (элемент BODY). Элемент является следующим компонентом веб-страницы. Парные теги <BODY> и </BODY> указывают на начало и конец тела документа. Вся содержательная часть документа располагается в элементе BODY. Содержимое этого элемента отобразится на экране согласно заданной вами HTML-разметке.

Начальный и конечный теги элемента BODY являются необязательными в структуре HTML-документа. Однако контейнер <BODY> необходим, когда требуется задать свойства всей страницы. Присутствие в HTML-документе элемента BODY является формальным признаком того, что данный документ имеет обычную структуру. Если в документе используются фреймы, то вместо BODY применяется элемент FRAMESET.

Тег <BODY> размещается непосредственно после элемента HEAD, а конечный тег </BODY> является предпоследним тегом документа:

```
<HTML> <HEAD> <TITLE> Название веб-страницы</TITLE>
  </HEAD>
<BODY>
 <!-- Содержательная часть документа -->
…
</BODY>
</HTML>
```

Таблица 2 — Атрибуты BODY

Атрибуты BODY	Назначение
ALINK	Определяет цвет активной ссылки
BACKGRQUND	Указывает на URL-адрес изображения, которое используется в качестве фонового
BGCOLOR	Определяет цвет фона документа
BGPRQPERTIES	Если установлено значение FIXED, фоновое изображение не прокручивается
LEFTMARGIN	Устанавливает границу левого поля в пикселях
LINK	Определяет цвет еще не просмотренной ссылки
TEXT	Определяет цвет текста
TOPMARGIN	Устанавливает границу верхнего поля в пикселях
VLINK	Определяет цвет уже просмотренной ссылки

Элемент ADDRESS. Элемент ADDRESS является одним из важнейших элементов HTML. Он служит для идентификации автора документа и (по вашему желанию) для указания адреса автора. Сюда же обычно помещаются сведения об авторских правах. Этот элемент располагается либо в начале, либо в самом конце документа. Элемент ADDRESS состоит из текста, помещённого между тегами <ADDRESS и </ADDRESS>. Текст, заключённый между этими тегами, обычно показывается в окне браузера курсивом.

```
 <HTML>
    <HEAD>
        <TITLE> A Basic Document Template </TITLE>
```

```
<HEAD>
<BODY BACKGROUND="greybg.jpg" BGCOLOR="GRAY",
TEXT="BLACK",
 LINK="BLUE", ALINK="GREEN", VLINK="RED">
Put the body text in here.
<ADDRESS>Created by Vitaliy Pavlenko<BR>
Created on 31 December 1998</ADDRESS>
</BODY>
</HTML>
```

4. Форматирование HTML-документа и текста

Формирование абзацев и строк. Текст — это основной вид информации, представленный на большинстве веб-страниц. Действительно, пользователи приходят на ваш сайт прежде всего за информацией, для которой наиболее привычное представление — текст. Текстовые блоки состоят из отдельных строк, объединённых в абзацы. Абзацы начинаются с новой строки, а группы абзацев отделяются друг от друга заголовками.

Разбиение текста на абзацы (элементы P). При оформлении обычных текстовых документов законченные мысли представляются в виде абзацев. Этого правила придерживаются и в процессе создания документов для Web. Для создания абзаца в языке HTML предусмотрено несколько возможностей. Простейшая из них — это использование тегов <P> и </P>, между которыми помещается текст абзаца.

Разрывы строк (теги <BR>, <NOBR>). Если в тексте нужно выполнить перевод строки (разрыв строки), используйте элемент <BR>. Тег
 — это один из немногих одиночных тегов, относящийся по классификации HTML к элементам пробелов. Он представляет собой пустой элемент, заставляющий браузер перенести текст на новую строку независимо от состояния текущей строки. Теги
 вам могут понадобиться при публикации текстов, содержащих отдельные строки, например почтовых адресов или стихов. Размер пустого поля между строками, которое будет отображено благодаря тегу <BR>, зависит от установок браузера.

Разрывы строк, заданные тегами <BR>, будут появляться при любой ширине окна браузера. Если вам нужно, наоборот, сделать так, чтобы браузер не образовывал новой строки ни при каком размере окна, воспользуйтесь элементом <NOBR>, который блокирует разрыв строки.

Выравнивание абзацев (атрибут align). Абзацы, которые задаются тегами <P> и <BR>, по умолчанию выравниваются по левому краю. С помощью атрибута align можно задать выравнивание по правому краю страницы (значение right) и по центру (значение center). Значение align="left" определяет выравнивание по левому краю.

Разделы (элементы DIV). Для структуризации текста в HTML-документах, помимо абзацев, используются разделы. Они задаются тегами <DIV> и </DIV>. Элементы DIV относятся к группе блоковых элементов, характерной чертой которых является наличие перед ними пустой строки.

Разделы DIV удобны тем, что они позволяют выровнять любой фрагмент текста, задать его размеры и расположить в нужном месте страницы. К разделам

можно применять любые атрибуты стиля и обращаться к ним как к объектам листов стилей. Большое значение разделы имеют при разработке динамической модели документа.

Начальный тег <DIV> может дополняться атрибутами: align, class, datafld, datasrc, dir, href, id, lang, language, title, style и др. Атрибут align применяется к разделам, так же как к абзацам, например строка <DIV align= "center">. Единственная </DIV> задаёт центрирование содержимого контейнера <DIV>.

Заголовки (элементы Hx). Заголовки позволяют разделить веб-страницу на логически законченные блоки и помогают посетителю сориентироваться в содержимом документа. Именно по заголовкам посетитель получает представление о содержании основного текста. Поэтому заголовки должны нести максимальную смысловую нагрузку.

Заголовки задаются парными тегами <Hx> (x=1, 2...6), причем конечные теги </Hx> являются обязательными. В HTML определено шесть уровней заголовков. Все уровни пронумерованы от H1 до H6. Самый крупный заголовок — первого уровня (H1), а шестого уровня (H6), наоборот, самый мелкий.

Все заголовки обычно отображаются браузерами более крупным полужирным шрифтом. Заголовки как блоковые элементы отделяются от остального текста пустыми строками.

Пример страницы, содержащей заголовки всех шести уровней:

```
<HTHL>
<HEAD>
<TITLE> Примеры заголовков</TITLE>
</HEAD>
<B0DY>
<H1> Заголовок, представленный элементом H1</H1>
<H2> Заголовок, представленный элементом H2</H2>
<H3> Заголовок, представленный элементом H3</H3>
<H4> Заголовок, представленный элементом H4</H4>
<H5> Заголовок, представленный элементом H5</H5>
<H6> Заголовок, представленный элементом H6</H6>
</BODY>
</HTML>
```

Разнообразить оформление веб-страниц можно с помощью их выравнивания. Для этого применяется атрибут align в составе любого из тегов <Hx>. Возможные значения этого атрибута: left, right и center. По умолчанию заголовки размещаются по левому краю страницы.

Центрирование (элемент CENTER). Часто при создании веб-страниц применяют выравнивание текстовых блоков по центру страницы. Центрирование возможно как с помощью атрибута align=" center":

<DIV align="center">Текст по центру страницы </DIV>,

так и с применением парного тега <CENTER>, например:

<CENTER>Размещение текста по центру</CENTER>

Содержимое контейнера <CENTER> при отображении на экране размещается по центру страницы. Если горизонтальная полоса прокрутки отсутствует,

то элемент CENTER будет расположен посредине окна браузера, независимо от размеров самого окна.

Горизонтальные линии в документе (элемент HR). Различные части большого документа можно визуально отделить одну от другой с помощью горизонтальных линий. Для создания таких линий в HTML предусмотрен специальный элемент HR (сокращение от Horizontal Ruler). Пустой элемент HR отображается браузером в виде стандартной линии. По умолчанию стандартная линия занимает всю ширину окна браузера, а ее толщина составляет 2 пикселя. Верхняя часть такой линии несколько темнее, чем ее нижняя часть. Поэтому в этом случае говорят о стандартной линии с тенью.

Для создания горизонтальных линий, отличающихся от стандартной, применяются атрибуты тега <HR>. Этими атрибутами изменяются свойства стандартной линии. Например, можно убрать тень с помощью атрибута NOSHADE: <HR NOSHADE>.

Рассмотрим подробнее каждое из свойств линии. Выравнивание линии назначается атрибутом align. Этот атрибут, обычно применяемый в паре с атрибутом задания ширины линии, принимает значения: center (для центрирования линии), left (выравнивание линии по левому краю страницы) или right (выравнивание линии по правому краю страницы). По умолчанию горизонтальные линии отображаются по центру страницы. Ширина линии формируется атрибутом width, который указывает значение ширины в пикселях или процентах. Например, тег <HR widht=25> определяет горизонтальную линию шириной 25 пикселей. Если значение атрибута width задастся в процентах (ставится символ %), то ширина линии вычисляется относительно ширины окна. Например, запись <HR width="25%"> приводит к отображению горизонтальной линии шириной в одну четвертую окна браузера. Задание длины линии в относительных единицах (в процентах) более предпочтительно, поскольку в этом случае линии будут одинаково отображаться на пользовательских мониторах разного размера.

- Толщина линии задается атрибутом SIZE. Значение этого атрибута назначается в пикселях в диапазоне от 1 до 175. Присвоение больших значений атрибуту SIZE не приводит к увеличению фактической толщины линии в обоих браузерах (Internet Explorer и Netscape). Если значение SIZE не указано, по умолчанию отображается линия толщиной 2 пикселя.
- Цвет линии. Для задания цвета линии в наборе допустимых атрибутов тега горизонтальной линии имеется атрибут color. Например, красная линия задается тегом <HR color="red">.

Элемент FONT и его атрибуты. Свойства шрифта проще всего определять с помощью элемента FONT, для которого предусмотрены следующие атрибуты:

- size — размер шрифта;
- color — цвет шрифта, определяемый именным или кодовым значением.
- face — гарнитура шрифта или список допустимых шрифтов.

Допустим, вы хотите, чтобы фрагмент текста был показан шрифтом Arial, имеющим оливковый цвет и размер 16 пунктов. Для этого введите элемент: <FONT face="Arial" size=16pt color="olive"> Выделенный текст</FONT>

Относительные размеры шрифтов. Размер шрифта в HTML-документах часто выражают не в абсолютных единицах (разделах или пикселях), а в относительных единицах, записываемых целыми числами — от 1 до 7. В этих единицах основному шрифту, используемому по умолчанию, присваивается размер 3. Самый мелкий шрифт имеет размер 1, а самый крупный — размер 7. Размер шрифта 3 выражает некоторый условный размер, который может различаться для разных браузеров. Чтобы задать точный размер шрифта, нужно обратиться к абсолютным единицам — пикселям или разделам.

Текст, выделенный тегом <FONT size=-l>, будет отображён шрифтом, размер которого равен 2 (отсчёт ведётся от размера 3, принятого по умолчанию: 3-1=2). А текстовый фрагмент, предваряемый тегом <F0NT size=+2>, будет выводиться на экран шрифтом размера 5 (3+2=5).

При определении размера шрифта может оказаться, что вычисляемое значение равно нулю или отрицательно. Тогда шрифту назначается размер 1 (самый мелкий шрифт в шкале 1–7). Если же вычисляемый размер оказывается большим 7, то размер шрифта все равно считается равным 7 (самый крупный шрифт).

Пример:

```
<HTML>
   <HEAD> <TITLE> Размеры шрифта</TITLE> </HEAD>
 <BODY>
      <FONT size=7>Шрифт размером 7</FONT><BR>
      <FONT size=б>Шрифт размером 6</F0NT><BR>
      <FONT size=+2>Шрифт размером 5</FONT><BR>
      <FONT size=+1>Шрифт размером 4</FONT><BR>
      <FONT size=3> Шрифт размером 3
       (основной шрифт по умолчанию) </F0NT> <BR>
      <FONT size=-l> Шрифт размером 2</FONT> <BR>
      <FONT size=-2> Шрифт размером 1</FONT> <BR>
</BODY> </HTML>
```

Выбор гарнитуры шрифта (атрибут face). Чтобы определять гарнитуру шрифта, фирма Microsoft ввела элемент FONT атрибут face.

Например, элемент **<FONT face="Helvetica">.**

Текст, отображаемый шрифтом Helvetica</FONT>, задаст отображение содержимого шрифтом Helvetica (Гельветика). В принципе, существует множество допустимых гарнитур шрифтов, однако отображаться будут лишь те шрифты, которые установлены на компьютере пользователя. Если же разработчик страниц сомневается в наличии какого-либо шрифта у пользователей, он указывает в атрибуте face несколько шрифтов.

<FONT face="Helvetica, Arial"> Текст, отображаемый шрифтом Helvetica.

Если этот шрифт отсутствует, получим вывод шрифтом Arial</FONT>.

В атрибуте face в качестве значений допустимо указывать несколько гарнитур шрифтов, разделяя их запятой. Для отображения текста браузер применит одну из перечисленных гарнитур, имя которой первой совпадет с имеющимся в его распоряжении набором гарнитур.

Пример страницы, содержащей текст, отображаемый различными гарнитурами шрифтов:

```
HTML><HEAD><TITLE>Гарнитуры шрифтов</TITLE></HEAD>
<BODY>
<FONT size=5> Примеры гарнитур шрифтов:</FONT>  <BR><BR>
<FONT face="Arial">Arial Cyr</FONT><BR>
<FONT face="Arial Black">Arial Black</FONT><BR>
<FONT face=" Comic Sans MS">Comic Sans MS</FONT><BR>
<FONT face="Courier New">Courier New</FONT><BR>
</BODY> </HTML>
```

Управление цветом шрифта и фона страницы. Цвет текста веб-страницы и ее фона задаются атрибутами тегов <BODY>, <FONT>. В HTML цвета определяются цифрами в шестнадцатеричном коде. Цветовая система базируется на трёх основных цветах — красном, зелёном и синем — и обозначается **RGB**. Для каждого цвета задаётся шестнадцатеричное значение в пределах от 00 до FF, что соответствует диапазону 0–255 в десятичном исчислении. Затем эти значения объединяются в одно число, перед которым ставится символ #. Например, число #800080 обозначает фиолетовый цвет.

*Таблица 3 — Цветовая система **RGB***

Цвет	Код
Black (черный)	#000000
Maroon (темно-бордовый)	#800000
Green (зеленый)	#008000
Olive (оливковый)	#808000
Navy (темно-синий)	#000080
Purple (фиолетовый)	#800080
Teal (чирок)	#008080
Gray (серый)	#808080
Silver (серебряный)	#C0C0C0
Red (красный)	#FF0000
Lime (известь)	#00FF00
Yellow (желтый)	#FFFF00
Blue (синий)	#0000FF
Fuchsia (фуксия)	#FF00FF
Aqua (аква)	#00FFFF
White (белый)	#FFFFFF

Цвет фона всего документа определяется атрибутом bgcolor тега <BODY>. Например, тег <BODY bgcolor="#ffffff">.

Для определения цвета шрифта в HTML-документе вы можете воспользоваться одним из трех способов:

- назначение цвета текстовых символов всего документа осуществляется атрибутом text тега <BODY>; например синий цвет текста устанавливается тегом <BODY text="blue">;

- если в HTML-документе определен основной шрифт с помощью элемента BASEFONT, то можно дополнить этот элемент атрибутом color, например <BASEFONT size=5 color="red">;
- для текстовых фрагментов цвет шрифта задается с помощью атрибута color в тегах <FONT>, например отображение текста красным шрифтом можно задать одной из следующих инструкций:

<FONT color="red">Текст </FONT> или

<FONT color="#ff0000">Текст </FONT>

Элементы стиля шрифтов. Элементы стиля шрифтов дают прямое указание браузеру для оформления текстового фрагмента в определенном стиле. При этом фрагмент отображается одинаково всеми браузерами, например как при выделении текста курсивом или полужирным шрифтом. Перечислим возможные элементы стиля шрифтов.

- B, I, U — элементы начертания шрифта (полужирное, курсивное, подчеркнутое).
- TT — моноширинный шрифт. Этот элемент является аналогом уже рассматривавшихся элементов KBD, CODE, SAMP;
- BIG — увеличение шрифта. Данный элемент задает увеличение размера шрифта на 10–20% относительно размера основного шрифта. Тег <BIG> не определяет точный размер увеличенного шрифта — выбор увеличения остается за браузером;
- SMALL — уменьшение шрифта. С помощью данного элемента можно уменьшить шрифт на 10–20%. Как и в случае тега <BIG>, степень изменения размера шрифта определяется браузером;
- STRIKE, S — зачеркнутый текст. Эти два вида элементов определяют зачеркнутый текст (тег <S> является сокращенным вариантом записи <STRIKE>).

Верхние и нижние индексы (элементы SUP и SUB). Для отображения индексов в HTML предусмотрены следующие элементы:

- верхние индексы. Надстрочный текст, показатели степени, а также верхние индексы формируются парными тегами . К примеру, выражение S= Pi*R2 записывается в виде: S=Pi*R²;
- нижние индексы. Формирование подстрочного текста или нижних индексов выполняется парными тегами . Например, химическая формула воды (H_2O) может быть задана строкой вида: H₂O.

5. Таблицы

В издательском деле таблицы являются одним из основных методов дизайна. Они представляют данные в виде удобных для восприятия колонок и строк и значительно упрощают анализ информации. С их помощью отделяют одну часть публикации от другой, улучшая структуризацию документа. Таблицы должны стать одним из главных элементов дизайна и ваших веб-страниц.

Элемент TABLE представляет собой тег-контейнер, в котором размещается содержимое таблицы. Этот контейнер напоминает рассмотренный в предыдущей главе контейнер списка.

Заголовки столбцов и строк выводятся полужирным шрифтом и располагаются по центру своей ячейки. Данные имеют обычный шрифт и выравниваются по левой стороне ячейки. При помощи атрибутов ALIGN и VALIGN можно по-разному размещать данные относительно границ ячейки. Эти атрибуты используются совместно с элементами <CAPTION>, <TR>, <TH> и <TD> в самых различных комбинациях. Ниже приведены значения атрибутов для перечисленных элементов.

Таблица 4 — Основные теги таблиц

Тег	Назначение
<TABLE></TABLE>	Контейнер таблицы
<TR></TR>	Контейнер строки таблицы. Допускается отсутствие закрывающего тега
<TD></TD>	Контейнер ячейки таблицы. В ячейку можно включить другую таблицу. Закрывающий тег может быть опущен
<TH></TH>	Контейнер заголовка, располагающегося обычно в первой строке либо в первом столбце таблицы. Закрывающий тег также необязателен.
Атрибут BORDER тега <TABLE>	Для определения рамки таблицы. Ширина рамки устанавливается в пикселях, например BORDER=1. (Значение, равное нулю, означает отсутствие рамки.)
Атрибут ALIGN тега <TABLE>	Для выравнивания таблицы в окне браузера. Имеет значения LEFT, CFNTER и RIGHT (по умолчанию LEFF)

Таблица 5 — Атрибуты ALIGN и VALIGN

Атрибут	Назначение
<CAPTION>	Атрибут ALIGN может иметь значения TOP и BOTTOM (по умолчанию TOP); размещает заголовок таблицы сверху или снизу
<TR>	Атрибут ALIGN может принимать значения LEFT, CENTER и RIGHT (по умолчанию LEFT для данных и CENTER для заголовков). Атрибут VALIGN может иметь значения TOP, BOTTOM, MIDDLE и BASELINE (по умолчанию MIDDLE). BASELINE применяется ко всем элементам строки и выравнивает их по базовой линии.
<TH>	Атрибут ALIGN может принимать значения LEFT, CENTER и RIGHT (по умолчанию CENTER). Атрибут VALIGN может иметь значения TOP, BOTTOM и MIDDLE (по умолчанию MIDDLE)
<TD>	Атрибут ALIGN может принимать значения LEF'I, CENTER и RIGHT (по умолчанию LEFT). Атрибут VALIGN может иметь значения TOP, BOTTOM и MIDDLE (по умолчанию MIDDLE)

Создание более сложных таблиц

Таблицы с рамками и без рамок. Как было отмечено выше, за наличие и вид рамки таблицы отвечает атрибут BORDER тега <TABLE>. При отсутствии атрибута BORDER рамка не прорисовывается. Однако, как показывает жизненный опыт, данные в таблице воспринимаются легче, если каждая ячейка имеет рамку. В HTML таблицы применяются не только для представления табулированных данных. Они дают возможность гибкого группирования и форматирования информации.

Объединение ячеек. Смежные ячейки таблицы могут объединяться с целью размещения большего количества данных. Например, в таблице из пяти строк и пяти столбцов все ячейки первой строки можно объединить и поместить в этой строке красивый заголовок таблицы. Возможно также объединение нескольких строк или создание пустой прямоугольной области.

Для соединения двух смежных ячеек в одном столбце нужно использовать атрибут ROWSPan тега <TH> или <TD>, например <TD ROWSPan=2>.

Для объединения двух смежных ячеек в одной строке нужно использовать атрибут COLSPan тех же тегов, например <TD COLSPan=2>

Таблица 6 — Новые атрибуты таблиц

Атрибут	Назначение
WIDTH	Определяет ширину всей таблицы в пикселях либо в процентах от ширины окна браузера. Может также использоваться для отдельной ячейки
HEIGHT	Определяет высоту всей таблицы в пикселях либо в процентах от высоты окна браузера. Может также использоваться для отдельной ячейки
CELLPADDING и CELLSPACING	Добавляют свободное пространство между данными внутри ячейки и ее границами и соответственно между ячейками внутри всей таблицы. Если рамка отсутствует, то результат их действия одинаков

Использование цветов. В HTML не предусмотрены специальные средства раскрашивания таблиц. Однако Microsoft Internet Explorer имеет расширения, позволяющие изменять цвет ячеек и рамок. Вы можете изменить цвет фона ячейки при помощи атрибута BGCOLOR перед размещением в ней текста или изображения, а также использовать атрибут BORDERCOLOR для изменения цвета рамки ячейки. Теги <TABLE>, <TD>, <TH> и <TR> допускают использование в них указанных атрибутов.

Практические примеры создания таблиц. По определению таблицы используются для представления табличных данных. Таблицы могут быть также полезны при работе с формами HTML, так как предоставляют возможность создавать очень хорошо организованные формы для ввода информации. Таблицы можно применять и для других целей, упоминавшихся ранее. Благодаря разнообразным возможностям группировать текстовую и графическую инфор-

мацию таблицы помогут вашим HTML-документам обрести более впечатляющий облик.

6. Графика

Вставка изображения в документ. Помещение изображения на страницу не вызовет у вас никаких затруднений. Для этого следует воспользоваться тегом <IMG> совместно с атрибутом SRC, поместив их в надлежащее место вашего HTML-документа и вставив вместо filename URL-адрес изображения: <IMG SRC="filename">

По умолчанию браузер выводит изображение немедленно после текста или другого объекта, описанного предыдущими инструкциями. По умолчанию, когда изображение вставляется в строку текста, строка выравнивается по низу изображения. Изменить эту установку можно при помощи атрибута ALIGN тега <IMG>. По умолчанию программа просмотра выводит изображение в текущей строке. Текст не «обтекает» его. Однако при помощи атрибута ALIGN тега <IMG> изображение можно сделать «плавающим», т. е. заставить текст расположиться вокруг изображения. В таблице 7 приведены значения атрибута ALIGN, позиционирующие обтекаемое текстом изображение относительно краев окна браузера.

Таблица 7 — Свойства атрибута ALIGN

Значение	Описание
TOP	Выравнивает текст по верху изображения
MIDDLE	Выравнивает текст по середине изображения
BOTTOM	Выравнивает текст по низу изображения
LEFT	Обтекаемое текстом изображение прижато к левой стороне окна браузера
RIGHT	Обтекаемое текстом изображение прижато к правой стороне окна браузера

При помощи тега <IMG> программе просмотра можно сообщить размеры изображения, которое затем размещено на странице. Для указания размеров изображения (в пикселях) служат атрибуты HEIGHT и WIDTH тега <IMG>. Если указанные размеры не совпадают с размерами изображения, программа просмотра изменяет масштаб изображения.

Альтернативное описание изображения. Для этого воспользуйтесь атрибутом ALT тега <IMG>:

<IMG SRC="filename" ALT="Description">

Если браузер читателя не выводит изображение, на его месте будет помещено альтернативное описание. Если изображение выводится, это описание будет находиться на месте иллюстрации до начала ее загрузки. Еще лучше использовать эту возможность совместно с указанием размеров изображения при помощи рассмотренных выше атрибутов. *Если указатель мыши задержать на иллюстрации на одну-две секунды, этот же текст будет выведен в специальном всплывающем окошке.*

Помещение изображения в рамку. Эта простая операция предполагает применение атрибута BORDER тега <IMG>. По умолчанию программа просмотра использует рамку, которая указана в соответствующей ссылке.

```
<HTML>
<HEAD> <TITLE>Using the BORDER attribute</TITLE></HEAD>
<BODY>
<A HREF=""><IMG SRC="book.gif" BORDER=0></A><BR>
<A HREF=""><IMG SRC="book.gif" BORDER=5></A><BR>
<A HREF=""><IMG SRC="book.gif" BORDER=10></A></BODY>
</HTML>
```

Фоновая графика. Чтобы украсить страницу, можно заполнить фон картинкой из графического файла. Фоновое изображение — это графический файл, содержащий картинку (желательно небольшого размера), которая многократно выводится на экран, заполняя все окно. Картинка может представлять собой небольшой прямоугольник или длинную узкую полоску (например, залитую градиентом).

Фоновая графика задается в теге <BODY> в начале документа HTML подобно тому, как задается цвет фона. При этом используется атрибут BACKGROUND, значением которого является имя графического файла. Например: <BODY BACKGROUND="fon.gif">.

7. Списки

В своем HTML-документе вы можете представить информацию различными способами. Одним из наиболее эффективных форматов являются списки. HTML имеет специальные элементы-контейнеры для представления данных в виде списков. Основными типами списков являются нумерованные и маркированные списки, меню, перечни каталогов и определений. Для получения дополнительных эффектов различные типы списков могут вкладываться друг в друга.

Последние спецификации HTML (4.0 и 4.01) предоставляют возможность создания следующих списков:

- маркированный (неупорядоченный) список;
- нумерованный (упорядоченный) список;
- список определений.

Структура маркированных и нумерованных списков

Списки строятся по принципу вложения элементов, вначале определяется родительский элемент, задающий тип списка, после чего следуют дочерние элементы заголовка и отдельных строк списка. Родительский элемент, внутри которого размещается список, задается следующими парными тегами:

<UL> . . . </UL> — маркированный список. Сокращение UL происходит от Unordered List — неупорядоченный список;

<OL> . . . </OL> — нумерованный список. Сокращение OL означает Ordered List — упорядоченный список;

<DL> . . .</DL> — список определений (сокращение от Definition List).

Заголовок списка является необязательным элементом, он формируется с помощью парного тега<LH>:

<LH>Заголовок списка</LH>

Содержимое списка состоит из отдельных строк. Каждая строка задается одиночным тегом <LI> (сокращение от List Item — элемент списка), например <LI> Содержимое строки списка.

Таким образом, количество элементов LI равно количеству строк в списке. В результате маркированный список оформляется следующим образом:

```
<UL>
   <LH><!—Заголовок списка--></LH>
   <LI> <!—Первая строка списка-->
   …
   <LI> <!—Последняя строка списка-->
</UL>
```

Маркированный список (элемент UL)

Рассмотрим подробнее создание маркированного списка, например списка рубрик некого сайта:

```
<HTML>
 <HEAD> <TITLE> Маркированный список</TITLE> </HEAD>
 <BODY>
   <H2> Список рубрик:</H2>
    <UL>
         <LI>Новости
        <LI>Финансы
        <LI>Спорт
        <LI>Погода <LI>Горячая линия
    </UL>
</BODY>
</HTML>
```

По умолчанию маркеры отображаются в виде маленьких затемненных кружков. Но можно воспользоваться предусмотренным атрибутом type.

Например, элемент списка *<LI type=square> Новости* будет иметь маркер в виде заполненного квадратика, а элемент *<LI type=circle> Новости* будет отображаться с маркером — незаполненным кружком.

Нумерованный список (элемент OL)

Формирование нумерованного списка сводится к применению парного тега <OL></OL> для обрамления элементов списка.

```
 <HTML>
<HEAD> <TITLE> Нумерованный список</TITLE> </HEAD>
 <BODY>
 <OL>
     <H2> Состав интегрированного пакета Microsoft Office 2000:</H2>
     <LI>Word;
      <LI>Excel;
      <LI>Access;
     <LI>PowerPoint;
      <LI>Outlook;
     <LI>FrontPage;
```

```
    <LI>Publisher;
    <LI>Project;
    <LI>PhotoDraw;
    <LI>Team Manager.
</OL>
 </BODY>
</HTML>
```

В HTML допускается несколько стилей нумерации, которые задаются атрибутом type. В зависимости от значения атрибута type списку присваиваются следующие стили нумерации:

type = l — нумерация арабскими цифрами (1, 2, 3, ...). Такая нумерация применяется по умолчанию;

type=A — нумерация прописными буквами английского алфавита (A, B, C, ...);

type=a — нумерация строчными буквами английского алфавита (a, b, c,...);

type=I — нумерация с помощью римских цифр (I, II, III, ...);

type=i — нумерация строчными римскими цифрами (i, ii, iv,...).

Задание номера первой строки. Этот номер определяется атрибутом start в элементе OL. Например, все строки списка будут последовательно пронумерованы, начиная с буквы E, если вы в качестве начального установите тег <OL type=A start=5>;

Списки определений

Рассмотренные выше маркированные и нумерованные списки состоят из элементов одного типа: UL либо OL. Более сложную структуру, представленную двумя типами элементов DT и DD, имеют списки определений. Эти списки представляют собой списки терминов вместе с их определениями, то есть они напоминают обычный толковый словарь. Каждому термину соответствует абзац определения, расположенный с отступом ниже термина. Задаются списки определений с помощью родительского элемента DL (сокращение от слов Definition List — список определений). Каждая позиция списка формируется двумя одиночными тегами, вложенными в контейнер <DL></DL>

<DT> — тег, задающий определяемый термин (DT — сокращение от Definition Term);

<DD> — тег, за которым следует собственно определение термина.

Базовый шаблон списка определений выглядит следующим образом:

```
<DL>
<DT>Term
 <DD>Definition of term
 <DL>
```

Пример документа, содержащего список определений:

```
<HTML>
<HEAD> <TITLE>Список определений</TITLE> </HEAD>
<BODY>
<CENTER>
   <H1>Словарь школьника</H1>
   <H2>B</H2>
```

```
  </CENTER>
  <DL>
 <DT>Вакуум
 <DD>ЭТО состояние разреженного газа, находящегося при давлении ниже атмосферного,
 <DT> Векторная физическая величина
 <DD>ЭТО физическая величина, которая имеет не только численное значение, но и направление
 <DT>Вес тела
  <DD>ЭТО сила, с которой тело, вследствие его притяжения к Земле, действует на опору или подвес.
  </DL>
</BODY> </HTML>
```

8. Ссылки

Работая в WWW, вы понятия не имеете, где находится та или иная нужная вам страница. Поэтому ссылки здесь являются единственной возможностью перейти от одного документа к другому.

Гипертекстовый документ — это документ, содержащий ссылки на другие документы, позволяющие при помощи нажатия кнопки мыши быстро перемещаться от одного документа к другому.

Гипермедийный документ основан на гипертекстовом документе, но в дополнение к тексту содержит разнообразную графику, видео и аудиообъекты. В таких документах в качестве ссылок часто используются изображения. Существует очень много типов мультимедийных объектов, которые могут быть размещены на веб-странице.

Известно, что наиболее простой путь путешествия по Сети — это переходы по гиперссылкам. Этим объясняется популярность использования гиперссылок на веб-страницах. С помощью гиперссылок или, проще говоря, ссылок пользователь может переходить к различным частям просматриваемой страницы, обращаться к другим страницам или к другим сайтам.

Ссылка представляет собой логическую связь одного фрагмента (элемента) документа с другим фрагментом в том же самом или в другом документе. Если щелкнуть мышью по ссылке, то выполняется переход по указанной связи.

Ссылка характеризуется двумя точками, называемыми закладками (anchor — якорь, закладка). Различают начальную закладку (source anchor) и конечную закладку (destination anchor).

Начальная закладка устанавливается на конкретном HTML-элементе и определяет точку, из которой задается ссылка. Начальные закладки можно размещать в тексте, в таблицах, в списках, изображениях или фреймах. Обычно начальная закладка выделяется относительно других элементов документа (цветом, подчеркиванием и т. д.).

Конечная закладка определяет точку назначения ссылки. В отличие от начальной закладки, эта закладка может относиться не только к конкретному

HTML-элементу (тексту, изображению, аудио- или видеоклипу), но и к программе или документу в целом.

В HTML различают внутренние и внешние гиперссылки. Внутренние ссылки осуществляют переход в пределах одного и того же документа. Их применение целесообразно в больших документах для перемещения по разделам. Например, в начале документа может быть размещено содержание (оглавление), состоящее из ссылок на заголовки разделов документа. Внешние ссылки обеспечивают переходы к другим документам, расположенным на других веб-серверах. Внешние ссылки иногда называют удаленными гиперссылками.

Принципы создания ссылок. Ссылка состоит из двух частей. Первая из них — это то, что вы видите на веб-странице; она называется указатель (anchor). Вторая часть, дающая инструкцию браузеру, называется адресной частью ссылки (URL-адресом, URL reference). Когда вы щелкаете мышью по указателю ссылки, браузер загружает документ, адрес которого определяется URL-адресом.

Для создания ссылок (как внешних, так и внутренних) применяется элемент <A>, называемый *элементом привязки* или, другими словами, *якорным элементом* (буквальный перевод термина *anchor element*). Название элемента <A> происходит от первой буквы слова *anchor* — якорь.

Элемент привязки <A> должен выполнять простую задачу: отображать содержимое ссылки на экране и указывать браузеру, к какому ресурсу необходимо перейти при щелчке по ссылке. Для этого внутри тега <A> ставится обязательный атрибут href, с помощью которого определяется целевой ресурс (конечная закладка), а внутри контейнера <A> размещается содержимое ссылки. Содержимым элемента А могут быть любые текстовые символы или графические элементы. Место расположения ссылки в документе (начальная закладка) определяется непосредственно местом вставки элемента <A> в HTML-код.

Простейшая ссылка создается по следующей схеме:

<A href="адрес_перехода"> текст ссылки </A>

Содержимое ссылки в данном случае — это выделенный «текст ссылки», щелчок по которому инициирует переход. Вообще, если код между тегами <A> и </A> представляет собой текст, элемент <A> называется тестовым элементом привязки. Элементы ссылок лучше всего размещать, не прерывая текста документа дополнительными пробелами или разрывами строк. Текст ссылки должен быть кратким и содержательным, чтобы пользователь представлял, какую информацию он получит, перейдя по данной ссылке. При наличии в документе нескольких ссылок нужно учитывать, что вложение ссылок недопустимо.

Протоколы доступа

Перечислим наиболее часто используемые протоколы:

- http — это протокол, на котором основан Web. Такой префикс является указанием браузеру для поиска документа по всему WWW. Например, записав код: **A href="http://www.kodak.com/us/en/health»> Текст ссылки</A>**, вы зададите ссылку на веб-страницу медицинского отделения компании Kodak;
- mailto — префикс вызова протокола электронной почты SMPT (Simple Mail Transport Protocol — простой протокол передачи почты). Например, ссылка вида **<A href="mailto://medinews@kodak.com»> Текст ссылки</A>** позво-

лит посетителю вашей страницы перейти к открытию на экране его компьютера окна для ввода исходящей почты, а в диалоге «Куда» будет указан адрес MediNews@Kodak.com медицинского журнала компании Kodak;

- ftp — протокол обмена данными с ftp-серверами (ftp — сокращение от File Transfer Protocol — протокол передачи файлов). Если, например, создать ссылку вида **<A href="ftp://dartmouse.edu/pub/something.exe»> текст ссылки</A>,** то можно организовать подключение к ftp-серверу dartmouse.edu с тем, чтобы получить доступ к файлу something.exe из папки pub.

Абсолютные и относительные URL. Обычно в относительном URL опускается название протокола и имя сервера. То есть документы с относительным URL размещены на том же сервере, что и текущий документ. Например, если в браузере загружена страница http://www.uniservice.com/transl/eng-rus, то относительный адрес fr-greek указывает на то, что этому адресу отвечает абсолютный URL: http://www.uniservice.com/transl/fr-greek.

Создание ссылок на другие веб-узлы. Такие ссылки задаются якорным элементом вида:

<A href="URL"> Текст ссылки</A>

В качестве URL может быть использован как абсолютный, так и относительный адрес.

Внутренние ссылки. Этот вид ссылок применяется для переходов из одной части документа в другую его часть, что полезно в случае больших документов со многими разделами, которые не отображаются целиком в окне браузера. Применение внутренних ссылок дает возможность пользователю легко перемещаться по странице.

Внутренняя ссылка определяется с помощью элемента привязки <A> с атрибутом href. При щелчке по внутренней ссылке вы попадаете в определенное место документа. Это место обозначается меткой. Причем метка создается с помощью якоря <A>, однако в нем вместо атрибута href используется атрибут name. Напомним, что точка размещения ссылки называется начальной закладкой, а точка размещения метки — конечной закладкой. Рассмотрим подробнее создание закладок в случае внутренних ссылок.

Поскольку переход задается в пределах одного и того же документа, внутренние ссылки записываются без URL-адреса. Вместо URL задается имя метки, к которой выполняется переход:

<A href = " #имя метки "> текст ссылки</A>

Символ # означает, что ссылка указывает на метку, а не на внешний файл. Для создания конечной закладки ссылки применяется тег <A> с атрибутом name:

<A name #"имя метки"> <! - - текст элемента метки - -> </A>

Допустим, документ имеет несколько разделов, из названий которых составлен список подобно оглавлению). Оформим элементы этого списка в виде внутренних ссылок:

<A href="#глав1"> Глава 1. Безенчук и нимфы </A>

<A href="#глав2"> Глава 2. Кончина мадам Петуховой </A>

При создании веб-страниц часто применяются ссылки из одного HTML-документа на другой документ, размещенный на том же сайте и в той же папке,

что и исходный документ. Для таких ссылок в атрибуте href элемента привязки не требуется задавать абсолютный адрес URL, можно воспользоваться относительным URL. В качестве конечной закладки указывается имя целевого файла, например:

<A href ="bonaqua.html "> Попробуйте свежесть воды BonAqua</A>.

Чтобы переход происходил к определенному фрагменту документа, выполните следующее:

- в атрибуте href тега начальной закладки определите целевой файл и имя закладки (через разделительный символ #), например: <A href="bonaqua.html#price">Условие поставки BonAqua</A>;
- в атрибуте name конечной закладки укажите имя закладки. В данном примере — это price:

<A name="price"> <!- - Содержание условий поставки - -> </A>.

Этот элемент А должен быть установлен в начале фрагмента, на который указывает ссылка.

Оформление ссылок. По умолчанию браузер устанавливает определенное цветовое оформление ссылок. Разработчик по своему усмотрению может изменить эти цвета. Поскольку ссылки оформляются единым образом для всего документа, атрибуты цвета ссылок вводятся в элемент тела документа BODY. Различают следующие атрибуты цвета:

- link — устанавливает цвет неактивизированных ссылок, то есть ссылок, которые еще не посещались;
- vlink — задает цвет просмотренных ссылок;
- alink — определяет цвет активной ссылки, то есть ссылки, на которую наведен указатель мыши.

Представление ссылки рисунком. Если вместо текста ссылки используется какой-либо рисунок, щелчок по которому приводит к переходу, то говорят о графическом элементе привязки. Для отображения рисунка в контейнер <A> вставляется элемент IMG, например:

<A href="applehistory.html"><IMG scr=apple.gif></A>,

где IMG — одиночный тег вставки изображения, а атрибут scr определяет файл-источник изображения.

Всплывающая подсказка. Если вы хотите, чтобы пользователь получил информацию о содержании ссылки, не загружая ее, создайте подпись к ссылке. Текст подписи будет появляться в виде всплывающей подсказки при наведении указателя мыши на ссылку. Задается подпись с помощью атрибута title якорного тега. Добавим к графической ссылке небольшую всплывающую справку:

<A href="applehistory.html" title=”История компьютеров Apple”>
<IMG scr=apple.gif></A>

9. Формы

Формы — это простейший интерфейс для получения отзывов от пользователей подачи заявок и оформления заказов и т. д. Каждому пользователю знакомы многочисленные элементы управления (кнопки, списки, текстовые поля,

флажки, переключатели и пр.), встречающиеся в различных диалогах и окнах современных приложений.

Объединение логически связанных элементов управления в документе HTML называется формой. Формы являются основным средством HTML, предназначенным для ввода и обработки информации.

Как происходит работа с формой? Посетитель сайта вводит в поля формы какие-либо значения и щелкает на определенной кнопке. После этого введенная им информация передается на сервер, где обрабатывается соответствующей программой — обычно CGI-сценарием. Данные, введенные посетителем, претерпевают изменения, и в зависимости от алгоритма сценария вы можете получить с сервера преобразованную веб-страницу, можете отправить письмо по e-mail, ваш браузер может отобразить результат поиска по ключевым словам, и т. д.

CGI — это аббревиатура от Common Gateway Interface (общий шлюзовой интерфейс), которая обозначает протокол для взаимодействия внешних программ с веб-сервером. Программы, соответствующие этому протоколу, называются CGI-программами или CGI-сценариями.

Задание формы (элемент FORM). В HTML для создания формы применяется контейнер <FORM>, который может размещаться в любом месте основной части документа. Запись кода формы с учетом допустимых атрибутов имеет вид:

<FORM name=«имя_формы» action=«URL» method=«Метод» enctype=«Тип кодировки»>

…

</FORM>

Имя формы (атрибут name). Элементом FORM с помощью атрибута name можно дать имена. В принципе присваивать имена формам не обязательно. Однако имя необходимо, чтобы формой можно было манипулировать из сценария.

Представление формы на сервер (атрибут action). В атрибуте **action** содержится адрес URL, по которому будет представляться форма (при нажатии на кнопку Submit). Значением атрибута может быть просто e-mail или указание на сервер, который занимается обработкой форм с помощью CGI-программы.

Веб-мастеры помещают программы для обработки форм в специальную папку — cgi-bin (сокращение от слов CGI и binary — двоичный код). Вид тега <FORM> с атрибутом action, может быть следующим:

<FORM action=”http://www.earthweb.com/cgi-bin/s97r”>….</FORM>

Action в данном случае означает, что браузер установит связь с сервером www.earthweb.com и данные, записанные в форму, передаст для обработки в приложение s97r, размещенное в папке cgi-bin.

Способ представления данных формы зависит от протокола, указанного в action, а также от значений других атрибутов (method и enctype). Например, action=mailto:serg@ip.com.ru браузер автоматически отправит результаты, введенные в форму по указанному адресу.

Отметим, что атрибут action, как правило, присутствует в тегах <FORM>. Если атрибута нет, то в качестве значения action подставляется URL самого документа.

Передача данных (атрибут method). Существуют два метода передачи информации: get или post.

Метод get — более простой, он применяется по умолчанию и осуществляет передачу данных в один этап. Метод get представляет оптимальную схему передачи в случае, когда для обработки формы не требуется какого-либо дополнительного внешнего процесса.

Метод post. Этот метод используется, когда нужно передать большие объемы информации. Передача происходит как минимум в два этапа.

Кодировка (атрибут enctype). Прежде чем передать данные из формы на веб-сервер, их необходимо закодировать. Можно воспользоваться типом кодировки по умолчанию application/x-www-form-urlencoded. Тогда атрибут enctype в составе тега FORM можете вообще не указывать. Другой тип кодировки — text-plain (более подходит, когда используется URL mailto в атрибуте action) и multipart /form-data (подходит при отправке файлов).

Создание элементов управления

На веб-страницу элементы управления вводятся тегом <INPUT>.

Атрибуты тега <INPUT>

- type — определяет тип элемента, предназначенного для ввода данных (по умолчанию описывает простое текстовое поле);
- align — задает расположение элемента по вертикали. Возможные значения как в теге <IMG>;
- checked — отвечает установке флажка или переключателя. Установленный флажок или переключатель возвращает пару значений name/value при представлении формы;
- name — определяет имя элемента, которое используется при передаче формы;
- size — задает размер текстового поля в символах;
- maxlength — указывает максимальное число символов, которые могут быть введены в текстовое поле;
- src — используется совместно с атрибутом type=IMAGE и задает URL нужного изображения;
- value — определяет значение (текстовую строку или число) для элемента, задаваемого атрибутом type;
- tabindex — задание этого атрибута позволяет установить порядок перемещения фокуса по элементам формы при нажатии клавиши Tab;
- и др.

Перечислим возможные элементы управления:

- текстовые поля (text). Они называются иногда полями ввода или полями редактирования;
- поля ввода пароля (password);
- скрытые поля (hidden);
- многострочное текстовое поле (textarea);
- поле выбора файлов (file);
- флажки (checkbox);

- переключатели (radio);
- кнопки — элементы управления, которые используются для представления формы (кнопка submit), сброса данных формы (кнопка reset), создания эффектов для кнопки (кнопка button);
- списки (select). Существуют два типа списков: раскрывающиеся списки и списки с прокрутками;
- надписи (label). Для создания надписей, связанных с элементами управления (флажками, переключателями и др.);
- группы элементов (FIELDSET). Эти группы создаются для четкого разделения страниц и форм на области связанного содержания.

Диалоговые текстовые поля. Поле для ввода одной строки текстовой информации задается значением text атрибута type:

<INPUT type="text" name="reply" size=15 maxlength=30>

Атрибут name идентифицирует элемент управления и набирается латинскими буквами.

Значения атрибута size и maxlength будут переданы на сервер под именем reply:

- size — определяет количество видимых (отображаемых) символов, то есть ширину текстового поля. В браузере Internet Explorer по умолчанию принимается size=2;
- maxlength — задает максимально допустимое количество символов, которое пользователь может внести в текстовое поле. Значение maxlength может превышать size (по умолчанию maxlength не ограничено).

Поле ввода пароля. В этом поле при вводе каждого символа на его месте отображается звездочка «*», скрывающая на экране содержимое поля. Поле ввода пароля создается тегом

<INPUT type="password">.

В этом теге можно указывать те же атрибуты (size, maxlength, value).

Скрытые поля. Это поля, которые не видны на странице и чаще всего вставляются в HTML-файл, когда необходимо передать серверу данные, не откорректированные посетителем.

<INPUT type="hidden">

Поле выбора файлов задается поле выбора файла с помощью значения атрибута type="file". На форме вместе с полем ввода имеется кнопка Обзор (Browse).

Флажки. Флажки дают возможность с помощью одного щелчка мыши выбрать тот или иной параметр в вашей форме. Создание флажков (атрибут checkbox)

<INPUT type="checkbox">

Для каждого флажка нужно ввести свой тег <INPUT> и задать его атрибуты:

- name — имя флажка или группы флажков;
- value — значение установленного флажка, считываемое при представлении формы.

По умолчанию начальное положение флажка считается не установленным. При представлении формы на сервер для установленных флажков считываются определенные значения согласно атрибутам value. Значением установленного флажка является текстовая строка, заданная в этом атрибуте. Не установленные флажки вклада в значения формы не дают.

Всплывающие подсказки (атрибут title). Многие пользователи привыкли к тому, что при наведении указателя мыши на элементы управления должна появляться всплывающая подсказка. Для создания всплывающей подсказки в тег <INPUT>, который задает элемент управления, необходимо добавить атрибут title.

<INPUT type="checkbox" id="ll" name="menu" value="shark" ti-
tle="Щелкните здесь, если вы любитель чего-нибудь этакого”>

Переключатели (атрибут radio). Переключатели оформлены аналогично элементам CHECKBOX, но в виде кружков, и отличаются от флажков тем, что в группе переключателей можно установить только один переключатель.

Для создания переключателя необходимо ввести тег

<INPUT type="radio">

Каждый переключатель вставляется в форму по отдельности, но все вместе они работают как одна группа. Для каждого переключателя необходимо задать атрибуты:

- name — имя переключателя или группы (переключатели с одинаковыми именами образуют группу);
- value — значение установленного переключателя, считываемое при представлении формы.

В группе только один из переключателей может быть первоначально установлен. Чтобы переключатель был установлен сразу после загрузки формы, в его тег нужно ввести атрибут checked. Если атрибут checked не присвоен ни одному из переключателей группы, браузер при загрузке установит по умолчанию первый переключатель.

Кнопки. С кнопками связывается ряд событий, из которых чаще всего используется щелчок левой кнопкой мыши.

Создание кнопки (атрибут button).

Чтобы разместить на форме (или веб-странице) прямоугольную кнопку, необходимо задать следующий атрибут: <INPUT type="button">

Для формирования кнопки с надписью достаточно дополнить тег атрибутом value, например:

<INPUT type="button" value="Пуск" name="start">

Кнопки submit и reset. В HTML предусмотрены два типа кнопок, которые создаются без использования значения button. Это кнопки специального назначения: Подача запроса (Submit) и Сброс (Reset).

Кнопка Submit предназначена для запуска процедуры передачи формы на сервер и формируется, например, тегом следующего вида:

<INPUT type="submit" value="Передача" name="trans">

Тег кнопки reset может иметь вид:

<INPUT type="reset" value="Повторный ввод">

Кнопки с изображениями (атрибут type="image"). Для создания кнопки с изображением нужно ввести в тег <INPUT> атрибут type=" image" и задать URL файла рисунка (атрибут src). Тег такой кнопки может иметь вид:

<INPUT type="image" src="rastr\hand12.gif" align="top" name="result">

Многострочные поля (элемент TEXTAREA). В текстовых полях, рассматривавшихся ранее, текст представлялся одной строкой. Снять это ограничение позволяет элемент, называемый многострочным полем. Для создания которого служат теги <TEXTAREA> и </TEXTAREA>, задать атрибуты cols, rows и ввести собственно текст.

Атрибуты cols и rows определяют соответственно ширину многострочного (максимальное количество символов в строке) и высоту поля (число отображаемых строк). Если весь текст не помещается в прямоугольную область поля, то появится вертикальная полоса прокрутки.

На машине клиента в форму, имеющую определенное имя, пользователь вводит информацию, а именно: указывает значения всех элементов ввода на форме. Заполненная форма может быть отправлена на сервер для обработки содержащихся в ней данных, но может быть обработана и на компьютере клиента. Перед отправкой формы браузер выполняет кодирование пересылаемой информации, чтобы обеспечить надежность и безопасность передачи. На сервере полученная информация просматривается и обрабатывается с помощью CGI-сценария. Результатом обработки может быть ответ, обычно это новая страница в формате HTML (например, благодарность за присланную информацию или просьба заполнить отсутствующие поля). Ответ пересылается с сервера, поддерживающего CGI, на компьютер пользователя.

10. Фреймы

Фреймы (frames — кадры или рамки**) —** это несколько видоизмененные окна. Отличаются они от обычных окон тем, что размещается внутри них. У фрейма не может быть ни панели инструментов, ни меню, как в обычном окне. В качестве поля статуса фрейм использует поле статуса окна, в котором он размещен. Фреймы разбивают окно на несколько независимых областей, причём в каждую из областей выводится отдельный документ (текст, изображения, таблицы и пр.).

Атрибуты элементов FRAMESET и FRAME:

cols — определяет количество вертикальных фреймов (фреймов-столбцов) и их размеры. В качестве значений используется список размеров каждого фрейма, разделенных запятыми, например <FRAMESET cols="20%,55%,*">;

rows — назначает количество и размеры горизонтальных фреймов. Значением этого атрибута является список размеров (аналогично значению атрибута cols). В последовательности перечисляемых чисел или символов сначала указывается высота верхнего фрейма, завершается список размером нижнего фрейма. Например, атрибут rows="20%, *, 30%";

- border — задает толщину рамок;
- bordercolor — задает цвет рамок;
- scrolling — отображает полосы прокрутки. Если содержимое фрейма не помещается в отведенную ему область окна браузера, то по умолчанию вдоль

границы фрейма появляется полоса прокрутки. Атрибут scrolling может принимать значения: yes (полосы прокрутки отображаются), no (не отображаются) и auto (это значение устанавливается по умолчанию и соответствует появлению полос прокрутки лишь в случае необходимости).

Теги <FRAMESET> обрамляют текст, описывающий компоновку фреймов. Здесь размещается информация о числе фреймов, их размерах и ориентации (горизонтальной или вертикальной). У тега <FRAMESET> только два возможных атрибута: ROWS, задающий число строк, и COLS, задающий число столбцов. Между тегами <FRAMESET> не требуется указывать тег <BODY>, но его можно поместить между тегами <NOFRAMES> в конце фреймовой структуры.

Между тегами <FRAMESET> не должно быть никаких тегов или атрибутов, которые обычно используются между тегами <BODY>. Единственными тегами, которые могут находиться между тегами <FRAMESET> и </FRAMESET>, являются теги <FRAME>, <FRAMESET> и <NOFRAMES>.

Атрибут ROWS тега <FRAMESET> задает число и размер строк на странице. Количество тегов <FRAME> должно соответствовать указанному числу строк. Справа от знака "=" можно определить размер каждой строки в пикселях, процентах от высоты экрана или в относительных величинах (обычно это указание занять оставшуюся часть места). Следует пользоваться кавычками и запятыми, а также оставлять пробелы между значениями атрибутов. Например, следующая запись формирует экран, состоящий из трех строк: высота верхней — 20 пикселей, средней — 80 пикселей, нижней — 20 пикселей:

<FRAMESET ROWS="20, 80, 20">

Следующий тег — <FRAMESET> — создает экран, на котором верхняя строка занимает 10% высоты экрана, средняя — 60%, а нижняя — оставшиеся 30%:

<FRAMESET ROWS="10%, 60%, 30%">

Можно задать относительные значения в комбинации с фиксированными, выраженными в процентах или пикселях. Например, следующий тег создает экран, на котором верхняя строка имеет высоту 20 пикселей, средняя — 80 пикселей, а нижняя занимает все оставшееся место:

<FRAMESET ROWS="20, 80, *">

Столбцы задаются так же, как строки. Для них применимы те же атрибуты.

Тег <FRAME> определяет внешний вид и поведение фрейма. Этот тег не имеет закрывающего тега, поскольку в нем ничего не содержится. Тег <FRAME> имеет атрибуты: NAME, MARGINWIDTH, MARGINHEIGHT, SCROLLING, NORESIZE и SRC.

Если вы хотите, чтобы при щелчке мышью на ссылке соответствующая страница отображалась в определенном фрейме, необходимо указать этот фрейм, чтобы страница «знала», что куда загружать. Фрейм, в котором отображаются страницы, называется целевым (target). Фреймы, которые не являются целевыми, именовать не обязательно. Например, можно записать такую строку:

<FRAME SRC="my.html" NAME="main">

Атрибут MARGINWIDTH действует аналогично атрибуту таблиц CELLPADDING. Он задает горизонтальный отступ между содержимым кадра и его границами. Наименьшее значение этого атрибута равно 1. Нельзя указать 0.

Можно не присваивать ничего — по умолчанию атрибут равен 6. Атрибут MARGINHEIGHT действует так же, как и MARGINWIDTH. Он задает поля в верхней и нижней частях фрейма.

Атрибут SCROLLING дает возможность пользоваться прокруткой во фрейме. Возможные варианты: SCROLLING=yes, SCROLLING=no, SCROLLING=auto. SCROLLING=yes означает, что во фрейме всегда будут полосы прокрутки, даже если это не нужно. Если задать SCROLLING=no, полос прокрутки не будет, даже когда это необходимо. Если документ слишком большой, а вы задали режим без прокрутки, документ просто будет обрезан. Атрибут SCROLLING=auto предоставляет браузеру самому решать, требуются полосы прокрутки или нет. Если атрибут SCROLLING отсутствует, результат будет таким же, как при использовании SCROLLING=auto.

Атрибут SRC применяется в теге FRAME при разработке фреймовой структуры для того, чтобы определить, какая страница появится в том или ином кадре.

Целевые фреймы TARGET задаются в ссылках левого фрейма. Вот зачем всем кадрам во фреймовой структуре были присвоены имена. Правый фрейм называется main, так что нужно в каждой ссылке добавить атрибут TARGET="main", в результате чего соответствующая страница появится во фрейме main. Обратите внимание: каждая ссылка содержит атрибут TARGET="main", который по щелчку мыши отображает страницу во фрейме main.

Контрольные вопросы

1. Что такое тег или дескриптор?
2. Какой минимальный набор тегов должен присутствовать в HTML-документе?
3. Что такое свойство или атрибут тега?
4. Какие основные теги имеют таблицы?
5. Как задается основная цветовая схема веб-страницы?
6. Как вставить изображение в документ?
7. Какие типы списков поддерживает язык HTML?
8. Какие теги служат для организации списков?
9. Что такое абсолютная ссылка?
10. Что такое относительная ссылка?
11. Как создаётся гиперссылка на какое-либо место в одном и том же документе?
12. Для чего предназначен атрибут alt в теге для вставки изображений?
13. Для чего предназначен контейнер <FORM> и </FORM>?
14. Что такое элементы управления формы?
15. Перечислите все типы элементов интерфейса (элементов управления), которые вы можете определять с помощью тега <INPUT>.
16. Для ввода каких данных служит контейнер <SELECT> и </SELECT>?
17. Что такое фрейм?
18. Как организуется фреймовая структура?

Лекция 5. Каскадные таблицы стилей CSS

План:

1. Каскадные таблицы стилей.
2. Способы определения стилей.
3. Элементы стилей.
4. Синтаксис стилей.
5. Свойства элементов, управляемых с помощью CSS.
6. Способы динамического управления страницей.
7. Команды Dynamic HTML.

1. Каскадные таблицы стилей

Каскадные таблицы стилей CSS — это довольно позднее нововведение. Если сам HTML появился в 1989 году, то таблицы стилей — только в 1997-м. Таблицы стилей не считаются частью HTML, а «гуляют сами по себе», как кошка Киплинга. Связано это с тем, что WWW была создана учеными как средство для обмена текстовыми документами, а HTML был языком, с помощью которого создавали эти документы. Для ученых главным было содержимое документа, а не его оформление. Но время шло, и в Интернет пришел обыватель, тотчас потребовавший от веб-дизайнеров «сделать ему красиво». А веб-дизайнеры, в свою очередь, потребовали от разработчиков стандарта HTML средств, облегчающих им работу. Так и возникли каскадные таблицы стилей. В настоящее время приняты спецификации CSS1 и CSS2, идет работа над CSS3. Однако современные веб-обозреватели, да и то самые последние их версии, полностью поддерживают только CSS1 и частично CSS2.

Каскадные таблицы стилей или просто *таблицы стилей* (*CSS* — Cascading Style Sheets) — это набор правил, описывающих форматирование разных фрагментов HTML-кода и хранящихся отдельно от него.

Одно такое правило, отображающее форматирование какого-то одного фрагмента или однотипной группы фрагментов кода, называется *стилем.*

Таблицы стилей описываются на особом языке CSS и хранятся в особых файлах с расширением css, хотя могут быть внедрены в саму веб-страницу. Таблицы стилей, как и многое другое, лучше представить на примере.

Пример HTML-кода, созданного без использования таблиц стилей.

```
<P><FONT COLOR="#FF0000" size="-1"><I>Это цитата.</I></FONT></P>
```

Перепишем наш пример с использованием стилей. Сначала напишем саму таблицу стилей:

```
.cit { font-style: italic; color: #FF0000 }
```

Эта таблица стилей содержит определение всего одного стиля — cit. Такой стиль, имеющий уникальное имя, называется *стилевым классом.*

Этот стиль содержит определение параметров для двух *атрибутов стиля.* Первый атрибут — font-style — задает «стиль» текста; в нашем случае значение italic делает текст курсивом, словно бы его поместили внутрь тега <i>. Второй атрибут — color — задает цвет текста. Как видите, два атрибута стиля cit заменили оба тега: и <FONT>, и <i>.

Собственно, красивое форматирование текста таблицы стилей, которое вы видите здесь, совсем не обязательно. Ваша таблица стилей может выглядеть и следующим образом:

.cit {font-style:italic;color: #FF0000}

Сохраним нашу таблицу стилей в файле styles.css. Перепишем фрагмент HTML-кода так, чтобы он использовал эту таблицу стилей.

<P CLASS="cit"> ЭТО цитата. </P>

Как видите, достаточно просто добавить в тег <p> атрибут CLASS и в качестве значения присвоить ему имя определенного нами стилевого класса, в нашем случае cit. Теперь этот фрагмент текста будет форматироваться курсивным красным шрифтом уменьшенного размера.

Но, кроме того, нам еще нужно добавить в секцию HTML-заголовка (тег <HEAD>) нашей страницы ссылку на таблицу стиля, чтобы веб-обозреватель смог ее найти. Эта ссылка будет иметь следующий вид:

<LINK REL="stylesheet" HREF="styles.ess" TYPE="text/css">

Таблицу стилей, сохраненную в отдельном файле, можно использовать во многих веб-страницах. Более того, она может находиться вообще на другом сайте. Кроме того, веб-страница может ссылаться одновременно на несколько таблиц стилей. Например, так:

<LINK REL="stylesheet" HREF="styles1.css">

<LINK REL="stylesheet" HREF="styles2.css">

В этом случае она будет использовать стили, определенные в обеих этих таблицах.

Как уменьшить шрифт всех цитат? Для этого достаточно добавить в определение стилевого класса еще один атрибут и присвоить ему соответствующее значение. В этом случае таблица стилей будет иметь такой вид:

.cit {color: #FF0000;

font-style: italic; font-size: smaller }

Здесь мы поместили в определение стиля помещен новый атрибут font-size, задающий размер шрифта. И присвоили ему значение smaller, задающее шрифт на одну ступень меньший, чем у родительского элемента.

А еще с помощью таблицы стилей можно изменить внешний вид любого тега HTML. Для этого нужно просто переопределить его в таблице стилей следующим образом:

H1 {color: #FF0000; font-size: smaller }

После этого все фрагменты текста, заключенные внутрь тега <i> (курсив), будут отображаться уменьшенным шрифтом красного цвета. Такой стиль называется *стилем переопределения тега.*

А если вы создадите такой стиль:

H1 I { color: #FF0000; font-size: smaller },

то уменьшенным шрифтом красного цвета будет отображаться только текст, заключенный внутрь тега <i>, который, в свою очередь, находится внутри тега <H1>.

<H1><I>Курсивный</I> заголовок</H1>

А следующий текст:

<I> Обычный курсив</I>

<H2> Экспериментируем с курсивом</H2>

будет отображаться как обычно. А такой стиль:

I.cit {color: #FF0000; font-size: smaller}

будет применяться только к тексту, помещенному внутрь тега с атрибутом CLASS, установленном в cit (своеобразный гибрид стилевого класса и переопределения тега; этот стиль так и называется — *гибридный*), т. е. к такому тексту: Маленький зеленый курсивчик

Тег также поддерживает атрибут CLASS. Есть еще один способ применения стиля к элементу страницы: воспользоваться атрибутом ID, которые также поддерживается почти всеми тегами.

<P ID="cit"> это цитата.</P>

Тогда определение стиля в таблице должно выглядеть следующим образом:

#cit { font-size: smaller; font-style: italic }

Такой стиль называется *стилем-селектором.*

Конечно, в одной таблице стилей вы можете определить одновременно несколько стилей:

.cit { font-size: smaller;

font-style: italic } I { color: #00FF00 } HI { color: #FF0000; font-size: larger}

Используя стили, можно задать для текста даже такие параметры, которые не поддерживает HTML. Например, можно создать рамку вокруг фрагмента текста или сделать так, что при наведении курсора мыши на текст он (курсор) будет менять форму.

С помощью таблиц стилей можно форматировать не только текст. Любому элементу страницы — изображению, таблице, горизонтальной линии — может быть присвоен стилевой класс.

Например, вы можете переопределить поведение тега <BODY> таким образом:

BODY (background-color: #000000)

Здесь мы задали черный цвет фона страницы с помощью нового атрибута background-color.

Рассмотрим пример. Предположим, у нас есть внешняя таблица стилей.

P (font-size: 9pt } HI { font-size: 24pt;

text-align: center } .copyright { font-size: 8pt;

font-style: italic;

text-align: right }

Здесь мы переопределили внешний вид текста, отформатированного тегами <p> и <HI>, и задали новый стиль copyright. Атрибут text-align задает выравнивание текста параграфа; значение center задает выравнивание по центру, а right — по правому краю.

Сохраним эту таблицу стилей в файле 10.css и создадим небольшую веб-страницу.

<HTML>

<HEAD>

```
<TITLE> Стили</TITLE>
<LINK REL="stylesheet" HREF="10.css">;
<STYLE>
H1 {font-size: 16pt; color: #00FF00 } I { font-size: larger } </STYLE>
</HEAD> <BODY>
<H1> Заголовок</H1>
<P>Параграф
<P>Параграф
<HR>
<P CLASS="copyright">
Авторские <SPAN STYLE=" font-style: normal!"> права</SPAN>.
</BODY> </HTML>
```

Во внешней таблице стилей для тега <HI> задаются шрифт размером 24 пункта и выравнивание текста параграфа по центру. Однако во внутренней таблице стилей для того же тега задается размер шрифта 16 пунктов и зеленый цвет текста. Какому определению стиля верить?

Веб-обозреватель «верит» обоим. Точнее, делает следующее. К определению стиля, сделанному во внешней таблице, добавляет определение, сделанное во внутренней таблице. А если определение затрагивает один и тот же атрибут (в нашем случае — размер шрифта), берется определение, сделанное во внутренней таблице.

Каскадность — огромное преимущество. Благодаря ему мы можем сократить размеры своих таблиц стилей до минимума. Нам достаточно будет просто переопределить необходимые атрибуты во внутренней таблице стилей или во внутреннем стиле, чтобы изменить веб-страницу до неузнаваемости. Веб-обозреватель однозначно разрешает все конфликты стилей, пользуясь *правилом каскадности и приоритета*.

Каскадные таблицы стилей, уровень 1, представляют собой простую технологию определения и присоединения стилей к документам HTML. Стиль, говоря житейским языком, — это все то, что определяет внешний вид документа HTML при его отображении в окне браузера: шрифты и цвета заголовков разных уровней, шрифт и разрядка основного текста, задаваемого в теге абзаца <p>, и т. д. Стиль задается по определенным правилам, а таблица стилей — набор правил отображения, применяемых в документе, к которому присоединена соответствующая таблица стилей.

Таблица стилей — это шаблон, который управляет форматированием тегов HTML в веб-документе. Почему в название таблиц стилей включено определение «каскадные»? Дело в том, что рекомендации Консорциума W3 позволяют использовать несколько таблиц стилей для управления форматированием одного документа HTML, а браузер по определенным правилам выстраивает приоритетность применения этих таблиц.

Любое правило каскадных таблиц стилей состоит из двух частей: селектора и определения. Селектором может быть любой тег HTML, для которого определение задает, каким образом необходимо его форматировать. Само определение, в свою очередь, также состоит из двух частей: свойства и его значения,

разделенных знаком двоеточия (:). Назначение свойства очевидно из его названия. В приведенном правиле селектором является элемент HI, а определение, записанное в фигурных скобках, задает значения двух свойств заголовка первого уровня: цвет шрифта (свойство color) определен как синий (значение blue) и размер шрифта (свойство font-size) определен в 16 пунктов (значение 16pt). В одном правиле можно задать несколько определений, разделенных символом точка с запятой (;).

Встраивание таблиц стилей в документ. Чтобы таблица стилей могла воздействовать на внешнее представление документа, браузер должен знать о ее существовании. Для этого ее необходимо связать с HTML-документом.

Существуют четыре способа связывания документа и таблицы стилей.

1. Связывание — позволяет использовать одну таблицу стилей для форматирования многих страниц HTML. Связывание позволяет хранить таблицу стилей в отдельном файле и присоединять ее к документам с помощью тега <LINK>, задаваемого в разделе <HEAD>:<LINK REL="stylesheet" TYPE="text/css" HREF="mystyles.css">.

Связываемый файл содержит набор правил каскадных таблицей стилей, определяющих форматирование документа, и должен иметь расширение CSS. Связывание позволяет разработчику применить одинаковый набор правил форматирования к группе HTML-документов, что приводит к единообразному отображению различных документов и придает некоторую системность серверу разработчика.

2. Внедрение — позволяет задавать все правила таблицы стилей непосредственно в самом документе. При внедрении таблицы стилей в документ правила, ее составляющие, задаются в стилевом блоке, ограниченном тегами <STYLE TYPE="text/css"> и </STYLE>, который должен размешаться в разделе <HEAD> документа:

```
<HEAD><STYLE TYPE="text/css">
<! -- B {text-transform: uppercase;}
P { background-color: lightgray;text-align:center; }-->
 </STYLE></HEAD>
```

3. Импортирование — позволяет встраивать в документ таблицу стилей, расположенную на сервере. В теге <STYLE> можно импортировать внешнюю таблицу стилей с помощью свойства @import таблицы стилей: @import: url(mystyles.css); его следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значением свойства @import является URL-адрес файла таблицы стилей.

4. Встраивание в теги документа — позволяет изменять форматирование конкретных элементов страницы.

Каждый тег HTML имеет параметр STYLE, в котором можно задать значения его свойств в соответствии с синтаксисом каскадных таблиц стилей. Например, в следующем примере задается форматирование заголовка первого уровня, определяющее его отображение шрифтом красного цвета:

```
<H1 style="color: red”> Заголовок отображается шрифтом красного цвета</H1>
```

2. Способы определения стилей

Всего стандарт CSS определяет три способа задания стиля для элемента страницы. Перечислим их и дадим краткое описание каждому способу.

1. *Внешняя* (или *привязанная*) таблица стилей. Стили сохраняются в отдельном файле с расширением css и привязываются к веб-странице с помощью особого тега <LINK>. Вы уже познакомились с внешними таблицами стиля, так что не будем сейчас подробно на них останавливаться.

2. *Внутренняя* (или *внедренная*) таблица стилей. Таблица стилей в этом случае имеет тот же самый формат, что и внешняя, но располагается в секции заголовка той же веб-страницы и помещается внутрь специального тега <STYLE>.

3. *Внутренние* (также *встроенные* или *внедренные*) стили. Определение стиля помещается прямо в нужный тег, для чего используется специальный атрибут STYLE.

Внутренняя таблица стилей по своему устройству аналогична внешней. Разница между ними в том, что внутренняя таблица стилей помещается прямо в заголовке HTML-файла и соответственно может быть использована только в нем. Давайте превратим внешнюю таблицу стилей, созданную выше, во внутреннюю.

```
<STYLE>
.cit { font-size: smaller;
font-style: italic } I {color: #00FFOO } HI I { color: #FF0000;
font-size: larger } </STYLE>
```

Как видите, ничего радикально не изменилось. Единственное, добавился парный тег <STYLE>, определяющий таблицу стилей. Вся эта конструкция помещается в секции заголовка веб-страницы, т. е. внутри тега <HEAD>. Само собой, внутренняя таблица стилей может быть только одна на страницу.

Обращаться к внутренней таблице стилей можно так же, как и к внешней:

```
<P CLASS="cit">Это цитата.</P>
```

Внутренние стили вообще помещаются внутри тегов, определяющих тот или иной элемент страницы. Делается это с помощью атрибута STYLE, поддерживаемого, как и CLASS, практически всеми тегами.

```
<P STYLE="font-size: smaller; font-style: italic">это цитата.</P>
```

Таким образом, пользуясь внутренними стилями, вы можете переопределить внешний вид любого элемента страницы, вообще не создавая таблицы стилей, ни внешней, ни внутренней. Внутренние стили незаменимы для создания уникальных элементов, не встречающихся больше нигде на странице.

В каких случаях следует применять тот или иной способ задания стилей? На этот случай есть простое правило: выясните, где и как часто будет использован тот или иной стиль.

Если стиль необходим во многих веб-страницах, вынесите его во внешнюю таблицу стилей. Например, если вы создали стиль основного текста страницы, который должен быть использован на всех страницах сайта, определите его в *глобальную* таблицу стилей, одну на весь сайт.

Правила каскадных таблиц стилей, в которых в качестве селектора используются теги HTML, влияют на отображение всех элементов заданного типа в

документе. Следующее правило отображает без подчеркивания все ссылки (тег <A>) в документе:

```
<STYLE TYPE="text/css"> <! -- A {text-decoration: none;}--> </STYLE>
```

Селектор CLASS. Класс позволяет задать разные правила форматирования для одного элемента определенного типа или всех элементов документа. Имя класса указывается в селекторе правила после имени тега и отделяется от него точкой. Можно определить несколько правил форматирования для одного элемента и с помощью параметра CLASS соответствующего тега применять разные правила форматирования в документе. Например, можно определить два класса отображения заголовка первого уровня:

```
<STYLE TYPE="text/css">
    <! -- H1.red {color: red;} H1.blueBgrd
    { color: red; background-color: blue}-- >
</STYLE>
```

В тексте документа ссылка на соответствующий класс задается в параметре CLASS:

```
<H1 CLASS="red"> Красный шрифт</H1>
<H1 CLASS="blueBgrd"> Красный шрифт на синем фоне</H1>
```

Имя класса в параметре CLASS задается без лидирующей точки. Оно может быть заключено в двойные или одинарные кавычки или задаваться вообще без кавычек, например CLASS=red.

Если класс должен применяться ко всем элементам документа, то в селекторе задается имя класса с лидирующей точкой без указания конкретного элемента:

```
<STYLE TYPE="text/css">
<!--.red { color: red; } .blueBgrd { color: red; background-color: blue} -->
</STYLE>
```

Теперь два класса red и blueBgrd можно применять к любым элементам документа:

```
<P ClASS=red>первый абзац</P>
<P ClASS=blueBgrd>Второй абзац</P>
```

Первый абзац отобразится красным шрифтом, а второй — красным шрифтом на синем фоне.

Селектор ID. Параметр ID, как и параметр CLASS, не влияет на отображение браузером элемента HTML, но он задает уникальное имя элемента, которое используется для ссылок на него в сценариях и таблицах стилей. Параметр ID можно применять к любому элементу документа.

Правила таблиц стилей регламентируют использование уникального идентификационного имени элемента в качестве селектора:

```
<STYLE TYPE="text/css"> <!-- #par24 { letter-spacing: lem; }
 H1#form3 {color: red; background-color: blue}-->
</STYLE>
  <BODY>
  <P ID=par24> Разреженные слова в абзаце</P>
  <H1 ID=form2> Черный шрифт</H1>
```

В этом примере абзац идентифицирован именем pag24 в параметре ID, поэтому к нему применимо правило с селектором #pag24. Второе правило в таблице стилей должно применяться к заголовку первого уровня с идентификатором form3. Такого элемента в нашем фрагменте нет, и поэтому заголовок form2 отображается с применением правила по умолчанию.

Контекстные селекторы. При разработке страниц HTML часто приходится одни элементы вкладывать в другие, например выделять слова тегом <EM> в каком-нибудь абзаце, задаваемом тегом <p>. В этом случае говорят, что элемент p порождает элемент EM и является его предком, а сам элемент EM является потомком элемента p. Некоторые свойства предка наследуются потомком, например цвет шрифта (свойство color). Чтобы вложенные элементы отображались со своими значениями свойств, можно определить для них правила форматирования:

<P> {color: blue} <EM> (color: yellow}

Однако это приведет к тому, что все выделяемые в документе элементы будут отображаться шрифтом желтого цвета. А если необходимо, чтобы выделяемые только в абзаце элементы отображались желтым цветом, а в других частях документа каким-то другим цветом? Здесь помогут *контекстные селекторы*. Поставленную задачу решит следующее правило: P EM {color: yellow)

Контекстный селектор состоит из нескольких простых, разделенных пробелами. Интерпретатор браузера просматривает в стеке все открытые элементы, находит элементы EM, порожденные элементом p, и применяет к ним указанное правило форматирования.

Псевдоклассы. Обычно правила форматирования присоединяются к элементу страницы, имеющему определенное положение в структуре документа. Концепция псевдоклассов и псевдоэлементов расширяет адресацию правил форматирования, позволяя внешней информации влиять на процесс форматирования. *Псевдоклассы и псевдоэлементы* можно использовать в селекторах правил форматирования, однако реально в исходном тексте документа HTML они не существуют. Вернее, они как бы вставляются браузером при определенных условиях в документ, и на них можно ссылаться в таблицах стилей. Их называют классами и элементами, так как это удобный способ описания их поведения. Говоря точнее, их поведение определяется фиктивной последовательностью тегов. Псевдоэлементы применяются для адресации некоторых частей элементов, тогда как псевдоклассы позволяют таблицам стилей для элементов разных типов применять разные процедуры форматирования.

Применение таблиц стилей. К одному документу можно присоединить несколько таблиц стилей, которые одновременно будут влиять на представление документа в окне браузера. Этот принцип является основополагающим принципом применения каскадных таблиц стилей, и можно указать на две причины, по которым он рекомендован группой разработчиков Консорциума W3.

1. Реализация модульности — разработчики таблиц стилей могут комбинировать таблицы стилей, каждая из которых отвечает за определенный этап форматирования документа. Например, в одной таблице можно определить все

правила форматирования шрифтов, во второй — правила позиционирования элементов и т. д.

2. Соблюдение равновесия между авторской разработкой и пристрастиями читателя — и авторы страниц, и читатели могут влиять на представление страницы через таблицы стилей. Пользователь может определить собственные правила отображения элементов документа в своих личных таблицах стилей, а браузер (на основании механизма приоритетности) будет использовать те или иные правила из таблиц пользователя или автора страницы.

3. Элементы стилей

На самом простом уровне стиль — это не что иное, как правило, указывающее браузеру, как выводить содержимое какого-то определенного HTML-тега. У каждого тега есть ряд ассоциированных с ним стилевых свойств, значения которых определяют, как этот тег воспроизводится браузером.

Правило приписывает определенное значение одному или нескольким свойствам тега. Например, большинство тегов имеют свойство color, значение которого определяет цвет, который современные GUI-браузеры должны использовать при отображении содержимого тега.

Встроенные стили. Атрибут style (внедренная таблица стилей)

Встроенный стиль — это простейший способ применения стиля к тегу. Просто включите в тег атрибут style со списком свойств и их значений. Браузер использует их при выводе содержимого в соответствии с требованиями тега.

Элемент style (в переводе с англ. — «стиль») — внедренная таблица стилей. Предназначен для определения CSS стилей документа HTML.

```
<h1 style="color: blue; font-style: italic"> Синий цвет! </h1>
```

Встроенные стили трудно поддерживать, поскольку они придают дополнительный смысл определениям тегов, затрудняя их восприятие.

Таблицы стилей на уровне документа. Расположенные внутри тега <head> и заключенные в собственный тег <style> таблицы стилей на «уровне документа» действуют на все вхождения тегов в документ, за исключением тех, что содержат обладающие более высоким приоритетом встроенные определения с атрибутом style.

Все, что находится между тегами <style> и </style>, рассматривается как часть стилевых правил, которые должны быть применены к документу. На самом деле содержимое тега <style> не относится к HTML и не подчиняется обычным правилам размеченного содержимого. Тег <style>, таким образом, позволяет вставить в документ чужеродную информацию, которую браузер использует для форматирования тегов.

К примеру, браузеры, понимающие таблицы стилей, отобразят синим курсивом содержимое всех тегов заголовков в HTML-документе, имеющем в своем заголовке следующее определение таблиц на уровне документа:

```
<head> <title>Все синее</title>
<style type="text/css">
<!--/* делаем все заголовки 1-го уровня синими и курсивом */
h1 {color: blue; font-style: italic}-->
```

```
</style>
</head>
<h1> Синий цвет! </h1>
...
<h1> Синий цвет! </h1>
```

Атрибуты

Атрибут id. Атрибут id позволяет присвоить элементу уникальную метку (id — сокращенно от *англ.* identifier — идентификатор).

Атрибут type. Атрибут type — определяет **MIME** тип содержимого элемента. Все каскадные таблицы стилей имеют тип text/css.

Атрибут media. Атрибут media обозначает тип устройства просмотра.

Значением атрибута по умолчанию является screen (дисплей компьютера). Другие определенные в стандарте значения — это tty (телетайпы, терминалы или портативные устройства с ограниченными возможностями дисплея), tv (устройства типа телевизора с низким разрешением и плохой цветопередачей), projection (проекционные аппараты), handheld (портативные компьютеры и сотовые телефоны), print (принтеры, печатающие чернилами на бумаге), braille (тактильные устройства), embossed (устройства для вывода текста в коде Брайля, для слепых), aural (аудиоустройства или, например, синтезаторы речи) и all (многие другие типы устройств).

Если вы хотите вместо all явно перечислить несколько типов устройств, вы можете использовать в качестве значения атрибута media заключенный в кавычки список, элементы которого разделяются запятыми.

Например, <style type="text/css" media="screen, print"> сообщает браузеру, что документ содержит таблицу стилей и предназначен для вывода на принтер или на дисплей компьютера.

Как обеспечить определение разных стилей для разных устройств, не создавая нескольких копий документа?

Стандарт CSS2 позволяет определять стили, специфичные для того или иного устройства вывода, через свои специальные директивы @import и @media.

Атрибуты dir, lang и title

dir — определяет, в каком направлении браузер должен выводить текст в элементе, которому он присвоен.

lang и xml:lang — естественный язык. Определяют язык, который главным образом употребляется в документе.

title — предназначен для произвольных кратких сообщений и описаний в теге самого элемента.

Внешние таблицы стилей. Кроме того, можно помещать определения стилей в отдельный документ (текстовый файл, MIME-типом которого является text/css) и импортировать такие «внешние» таблицы стилей в документы. Чтобы придать своим документам унифицированный вид, пользуйтесь единой таблицей стилей для других документов в этом собрании и даже для других собраний документов. Поскольку внешняя таблица является отдельным файлом и загру-

жается браузером из Сети, ее можно хранить где угодно, использовать многократно и даже применять чужие таблицы стилей.

Например, допустим, что мы создали файл gen_styles.css, включающий следующее стилевое правило:

h1 {color: blue; font-style: italic}

Можно предложить браузеру прочитать файл gen_styles.css для каждого документа из нашей подборки, а он в ответ покрасит содержимое каждого тега <h1> в синий цвет и выведет текст курсивом. Конечно, это произойдет только при условии, что машина пользователя способна совершать такие фокусы со стилем, на ней установлен работающий с таблицами стилей браузер и назначения стиля не отменены на уровне документа или встроенным определением.

Загрузить в документ внешнюю таблицу стилей можно двумя способами: при помощи тега <link> или директивы @import.

Присоединение внешних таблиц стилей при помощи тега <link>

Один способ загрузить внешнюю таблицу стилей — использовать тег <link> внутри тега <head> в вашем документе:

```
<head> <title>Присоединение стиля с помощью link</title>
<link rel=stylesheet type="text/css" href=http://www.kumquats.com/styles/gen
styles.css title="Синева">
</head>
<h1> Синий цвет!</h1>
...
<h1> Синий цвет! </h1>
```

В примере мы сообщаем браузеру, что документ, поименованный в атрибуте href, имеет тип stylesheet, как это указано в атрибуте type. Ссылка на внешнюю таблицу стилей в теге <link> требует указания атрибутов href и type. Далее сообщаем браузеру, что отношение файла к нашему документу определяется значением stylesheet (таблица стилей). Мы также добавили атрибут title, определяющий название таблицы стилей, создав возможность для ссылок на нее в будущем.

Импортированные внешние таблицы стилей

Второй способ загрузки внешней таблицы стилей импортирует файлы, применяя специальную директиву (так называемое правило at) в теге <style>:

```
<head><title>Импортированный стиль</title>
<style type="text/css">
<!—@import url(http://www.kumquats.com/styles/gen_styles.css);
@import "http://www.kumquats.com/styles/spec_styles.css";body
 {background: url(backgrounds/marble.gif)}-->
</style>
</head>
```

Директива @import ожидает параметр в виде URL, указывающего путь в сети к внешней таблице стилей. Как показано в этом примере, URL может быть заключенной в двойные кавычки строкой, заканчивающейся точкой с запятой, или следовать за ключевым словом url, также оканчиваясь точкой с запятой, но

помещаясь при этом в скобки. URL может быть абсолютным или относительным, отсчитываемым от базового URL документа.

Директива @import может появиться в определениях стиля на уровне документа или даже в другой таблице стилей, что позволяет создавать вложенные таблицы стилей.

4. Синтаксис стилей

Синтаксис стиля CSS имеет следующую структуру:

Селектор {свойство:значение [; свойство:значение [...]]}

Например: p{color:green; font-size:14pt}

Если нужно определить стиль не для всех элементов на странице (в данном примере не для всех абзацев), можно определить несколько вариантов стиля:

p.red{color:red}
p.blue{color:blue}

Если нам нужно определить один стиль для различных элементов, можно описать класс следующим образом:

.red{color:red}

Основные правила создания стиля.

- Определение стиля включает селектор и список атрибутов стиля с их значениями.
- Селектор используется для привязки стиля к элементу веб-страницы, на который он должен распространять свое действие. Фактически селектор однозначно идентифицирует данный стиль.
- За селектором через пробел указывают список атрибутов стиля и их значений, заключенный в фигурные скобки.
- Атрибут стиля (не путать с атрибутом тега!) представляет собой один из параметров элемента веб-страницы: цвет шрифта, выравнивание текста, величину отступа, толщину рамки и др. Значение атрибута стиля указывают после него через символ «:» (двоеточие). В некоторых случаях значение атрибута стиля заключают в кавычки.
- Пары «атрибут стиля»: «значение» отделяют друг от друга символом «;» (точка с запятой).
- Между последней парой «атрибут стиля»: «значение» и закрывающей фигурной скобкой символ «;» не ставят, иначе некоторые веб-обозреватели могут неправильно обработать определение стиля.
- Определения различных стилей разделяют пробелами или переводами строк.
- Внутри селекторов и имен стилей не должны присутствовать пробелы и переводы строки. Что касается пробелов и переводов строк, поставленных в других местах определения стиля, то веб-обозреватель их игнорирует. Поэтому мы можем форматировать CSS-код для удобства его чтения, как проделывали это с HTML-кодом.

Например, в следующем случае фон будет черным, а не белым или серым, даже если вы укажете белый и черный цвета в стилевом правиле: body {background: white, black}.

Множественные селекторы. Ко всем перечисленным через запятую в селекторном списке элементам применяются значения свойств, определенных в стилевом правиле.

К примеру:

h1, h2, h3, h4 {text-align: center}делает в точности то же, что и:

h1 {text-align: center}

h2 {text-align: center}

h3 {text-align: center}

h4 {text-align: center}

Контекстные селекторы. Обычно поддерживающие стили браузеры применяют определенные на уровне документа или встроенные стили к тегам везде, где они обнаруживаются в тексте документа, безотносительно к контексту. Однако стандарт CSS2 определяет способ применения стиля только к тегам, находящимся в определенном контексте, например содержащимся внутри другого тега. Чтобы создать контекстный селектор, перечислите теги в том порядке, в котором они должны вкладываться в документ, начиная с самого внешнего. Когда браузер встретит такой порядок вложения, стилевое правило будет применено к последнему тегу списка. Вот, например, как можно использовать контекстные стили для определения классической последовательности нумерации в упорядоченных списках: заглавные латинские буквы для самого внешнего уровня, римские цифры в верхнем регистре для следующего, затем арабские цифры и строчные латинские буквы для самого внутреннего уровня:

ol li {list-style: upper-roman}

ol ol li {list-style: upper-alpha}

ol ol ol li {list-style: decimal}

ol ol ol ol li {list-style: lower-alpha}

Следуя приведенной таблице стилей, поддерживающий стили браузер, встречая тег <li>, вложенный в один тег <ol>, применяет значение upper-roman для свойства list-style тега <li>. Когда тот же браузер видит тег <li>, вложенный в два тега <ol>, он использует значение upper-alpha свойства list-style. Вложите тег <li> в три и четыре тега <ol>, и вы увидите соответственно decimal- и lower-alpha-стили нумерации.

В селекторный список могут входить как селекторы, являющиеся именами тегов, так и контекстные селекторы, при этом все они должны отделяться друг от друга запятыми. Все элементы селекторного списка подчиняются общему для них стилевому правилу.

Пример: h1 em, p strong, address {color: red} означает, что вы увидите красный цвет всюду, где тег содержится в теге <h1> или тег (strong) появляется внутри тега <p>, а также в содержимом тега. Для того чтобы было применено контекстное правило, последовательность вложений не обязана точно совпадать с контекстным селектором.

К примеру, если тег (strong) вложен в тег <ul>, входящий, в свою очередь, в тег <p>, он все еще подчиняется контекстному правилу p strong, определенному выше. Если некоторое вложение тегов подходит для применения нескольких стилевых правил, употребляется более специфичное. В частности, если вы определили два контекстных селектора:

p strong {color: red}

ul strong {color: blue}

и использовали в документе последовательность <p>
(strong), будет применено второе, более специфическое, правило, и содержимое тега (strong) будет синим.

Универсальные, дочерние и смежные селекторы

Стандарт CSS2 определяет дополнительные конструкции селекторов, кроме уже известных, составляемых при помощи запятых и пробелов. Следующие примеры допустимых селекторов иллюстрируют это утверждение:

* {color: purple; font: ZapfDingBats}

ol > li {font-size: 200%; font-style: italic}

h1 + h2 {margin-top: +4mm}

В первом примере звездочка, универсальный селектор, указывает, что стилевое правило должно быть применено ко всем элементам документа, так что любой текст в нем должен быть представлен символами набора ZapfDingBats. Во втором примере устанавливаются отношения типа дочерний/родительский, в данном случае между элементами упорядоченного списка. Третий пример иллюстрирует тип смежного селектора, который относится к тегу, непосредственно следующему за другим тегом в документе. В этом случае всякий раз, когда за заголовком уровня 1 непосредственно стоит заголовок уровня 2, между ними увеличивается вертикальный интервал.

Селекторы атрибутов. Существует возможность присоединить стиль только к тем элементам HTML/XHTML, которые обладают специфическими атрибутами. Для этого следует перечислить нужные атрибуты в квадратных скобках рядом с именем элемента до определения стиля:

div[align] { font-style: italic }

div[align=left] {font-style: italic }

div[title~="bibliography"] { font-size: smaller }

div[lang|="en"] {color: green }

Первый пример — самый простой. Он выделяет курсивом текстовое содержимое только тех тегов , которые имеют атрибут align, независимо от значения этого атрибута. Второй пример чуть сложнее. Он затрагивает только те теги , у которых атрибут align имеет значение left.

Третья строчка относится к тегам , чей атрибут title содержит слово bibliography, отделенное одним или несколькими пробелами.

Частичное совпадение снова не считается. Если бы вы указали div[title~="a"], стиль относился бы только к тем тегам , у которых атрибут title содержит букву «a», отделенную пробелами или стоящую в начале или конце названия.

Последний пример определяет теги , у которых атрибуту lang присвоен список слов, разделенных дефисами и начинающихся на «en». Под это условие подходят такие атрибуты, как lang=en, lang=en-us и lang=en-uk.

Вы можете сочетать универсальный селектор с селекторами атрибутов, чтобы вашему определению соответствовал любой элемент с указанным атрибутом. Например,

*[class=comment] { display: none }

Этот селектор спрячет все элементы документа, у которых атрибут class имеет значение comment.

Псевдоэлементы. В документах встречаются отношения между элементами, которые невозможно или неудобно выражать при помощи тегов.

CSS2 вводит четыре псевдоэлемента, позволяющие определять особые отношения и стили для их отображения: first-line (первая строка), first-letter (первая буква), before (перед чем-либо) и after (после чего-либо). Объявляйте эти отношения, добавляя к стандартному элементу разметки отделенный двоеточием суффикс. Например, p:first-line {font-size: 200%; font-style: italic}
означает, что браузер должен отображать первые строки абзацев курсивом и шрифтом крупнее остального текста в два раза.

Подобным образом: p:first-letter {font-size: 200%; float: left} предлагает браузеру сделать первую букву абзаца в два раза крупнее остального текста, сдвинуть ее влево и разрешить первым двум строкам абзаца обтекать большую начальную букву. Псевдоэлементы :before и :after позволяют указывать в документе места, куда вставляется автоматически генерируемое содержимое, в частности особые заголовки и т. д.

5. Свойства элементов, управляемых с помощью CSS

Различают числовые и символьные значения свойств.

Числовые значения применяются для задания размеров, например ширины и высоты блока, размера шрифта, толщины рамки, межстрочных интервалов и т. д. Значение выражается десятичным числом, за которым обычно следует размерность. Размерность записывается после числа без дополнительного пробела, например 8pt, 1cm, 2.5in, 130%. В случае отрицательного значения перед числом ставится знак «минус» (например -15px).

Числовые значения могут выражаться в абсолютных или относительных единицах:

- *абсолютные значения* задают точный размер элемента и приводятся в стандартных единицах измерения длины, например в дюймах, сантиметрах или миллиметрах;
- *относительные значения* определяют размер элемента относительно другого элемента. Например, ширина изображения может выражаться в процентах относительно ширины браузера. Межсимвольный интервал часто задается в единицах em ширины символа основного шрифта (буквы «m»).

Таблица 8 — Единицы измерения значений свойств

Условное обозначение	Наименование единицы измерения	Пример
Абсолютные единицы		
in	дюйм	width:.25in
cm	сантиметр	height:1.5cm
mm	миллиметр	margin-left:12mm
pt	пункт (lpt = l/72in)	font-size:16pt
pc	пик (lpc = 12pt)	line-height: 1.2pc
Относительные единицы		
px	пиксел	left:200px
em	ширина буквы «m»	letter-spacing:0.3em
ex	высота буквы «x»	font-size:2ex
%	процент	width:150%

Символьные значения состоят из букв латинского алфавита или из комбинаций букв, цифр и специальных символов. Форма записи определяется стандартом CSS. Символьные значения присваиваются многим стилевым свойствам, например, свойствам шрифтов font-family (гарнитура шрифта) и font-style (стиль вывода шрифта), свойство текста text-align (выравнивание). Другим примером использования символьных значений являются свойства цвета color и background-color, значения которых задаются в символьном формате RGB-модели.

Свойства, вводимые листами стилей, условно можно разбить на группы (например, группа текста или группа свойств шрифтов).

Свойства шрифтов:

- **font-family** — задает гарнитуру шрифта, которая будет использована для вывода текста. Значением этого свойства может быть название конкретного шрифта (например, Arial) либо название семейства шрифтов. В спецификации CSS предусмотрены следующие семейства: serif (например, Times New Roman, Bodoni), sans-serif (например, Arial, Futura, Helvetica), cursive (например, Caflisch Script, Zapf-Chancery), fantasy (например, Alpha Geometrique, Western), monospace (например, Courier New). При определении свойства font-family можно указывать список шрифтов, разделенных запятыми, аналогично тому, как это делается при задании атрибута face;
- **font-size** — определяет размер шрифта. Значение размера может быть задано в абсолютных единицах (например, 10pt, 8mm, 2in), в процентах от основного шрифта (например, 130%), а также с помощью размерных выражений: xx-small (сверхмелкий), x-small (очень мелкий), small (мелкий), medium (средний), large (крупный), x-large (очень крупный), xx-large (сверхкрупный), larger (крупнее), smaller (мельче);
- **font-style** — задает стиль вывода символов. Возможны следующие значения этого свойства: normal (обычный), italic (курсив) и oblique (наклонный);

- **font-weight** — назначает вес или степень «жирности» шрифта. Для этого используются целые числа из диапазона от 100 до 900 с шагом 100 единиц. Однако чаще применяют ключевые слова: **bold** (жирный), **bolder** (более жирный) и lighter (более тонкий);
- **font-variant** — указывают вариант начертания текущего шрифта. Для этого свойства браузеры поддерживают только два значения: small-caps (отображение малыми прописными буквами) и normal (не влияет на отображение).

Примеры задания свойств шрифта:

P (font-family: helvetica, arial)
STRONG (font-size: 150%)
DIV.mono (font-family: monospace;
font-size: larger;
font-variant: small-caps}

Свойства текста:

- letter-spacing — устанавливает расстояние между буквами (межсимвольный интервал). По умолчанию этому свойству присваивается значение normal. Расстояние можно задать в любых абсолютных единицах (см. табл. 8);
- word-spacing — устанавливает расстояние между словами. Аналогично letter-spacing значением этого свойства по умолчанию считается normal. Иные значения могут быть заданы в абсолютных единицах (например, 10px, 2mm);
- text-indent — задает отступ первой строки абзаца (красную строку). Это свойство применяется к блокам, и его значение выражается в абсолютных единицах (например, 4mm, 1cm, 20pt и т. д.) либо в процентах от ширины блока (например, 10%). По умолчанию значение свойства text-indent равно нулю. Если свойству присвоить отрицательное значение, то вместо абзацного отступа получим выступ первой строки;
- text-align — задает горизонтальное выравнивание для текста, размещенного внутри элемента (например, P или DIV). Это свойство принимает значения: center (по центру), left (влево), right (вправо) и justify (по ширине). По умолчанию текст выравнивается по левому краю;
- vertical-align — устанавливает расположение текста и рисунков по вертикали относительно базовой линии. Свойство может принимать значения: baseline (выравнивание по базовой линии, принимается по умолчанию, sub (выравнивание по линии нижнего индекса), super (выравнивание по линии верхнего индекса);
- line-height — определяет межстрочный интервал. Значение этого свойства можно задавать в абсолютных единицах (например, 16pt, 3mm), в процентах (130%), а также количеством строк.

Пример:

<P style="font-family:Helvetica Cyr; font-size:12pt; font-style:italic">
Абзац, который воспроизведен курсивным шрифтом Helvetica, имеющим размер 12pt.
<P style="font-family:Helvetica Cyr; font-size:larger; font-variant: small-caps">
Абзац, который воспроизведен курсивным шрифтом Helvetica, имеющим размер larger и начертание small-caps.
<P style="font-weight : bold; text-indent: 12mm">

Данный абзац имеет отступ 12 мм, текст отображается полужирным шрифтом

<SPAN style="letter-spacing:3pt">Times New Roman </SPAN>,

причем название шрифта набрано с разрядкой – межсимвольным расстоянием 3pt.

<P style="font-size:large; line-height:9pt">

Этот абзац демонстрирует стилевые возможности по отображению строк. Размер шрифта large, межстрочный интервал 9pt.

</P>

Свойства цвета и фона

Перечислим свойства цвета и фона.

- color — устанавливает цвет шрифта. Пример: <p style=”color : red”>
- background-color — цвет шрифта.

  <BODY style="background-color : olive">

  <P style="color : yellow">
- background-image — определяет вставку фонового изображения.

Значением этого свойства является URL рисунка, например следующее CSS-правило:

.pog {background-image: url(myfig.jpg);

- background-repeat — назначает повтор фонового рисунка. Это свойство применяется, если размер рисунка меньше видимой области элемента. Для повторения рисунка выбирают значение по горизонтали — repeat-x, по вертикали — repeat-y, отключение повторения — *no-repeat.*

Свойства списков

Листы стилей позволяют управлять отображением списков: задавать гарнитуру, размер и цвет шрифта, выбирать вид маркеров (кружок, квадратик, рисунок и т. д.).

Рассмотрим свойства, предназначенные для форматирования маркеров списком: list-style-type, list-style-image, list-style-position.

Сокращенная форма записи этих трех свойств имеет вид list-style.

- **list-style-type** — задает маркеры для упорядоченных (нумерованных) и неупорядоченных (маркированных) списков. Набор допустимых значений свойств: list-style-type включает 22 значения, многие из которых пока еще не поддерживаются распространенными браузерами. Поэтому остановимся только на часто используемых значениях:

square — маркер в виде квадратика □;

circle — кружок ○;

disk — затемненный кружок ● (значение по умолчанию);

decimal — десятичные числа, начиная с 1;

lower-roman — строчные римские цифры, например, i, ii, iii;

upper-roman — прописные римские цифры, например, I, II, III;

lower-latin или lower-alpha — строчные латинские буквы, например, a, b, c;

upper-latin или upper-alpha — прописные латинские буквы, например, A, B, C;

none — маркер не отображается;

- **list-style-image** — задает маркер в виде картинки (изображения). Например, правило

  UL{ list-style-image :url ("mymarker.gif")} устанавливает для маркеров изображение, которое хранится в файле mymarker.gif;
- **list-style-position** — устанавливает позицию маркера в строке списка.

Значения outside-маркер отображается за пределами области строки списка, по умолчанию; inside — маркер находится внутри области списка.

Например, <LI style="list-style-type:circle"> задает отдельную строку, маркированную кружком.

Пример использования стилевых свойств при оформлении маркированного списка:

```
<HTML>
    <HEAD><TITLE>Стили списка</TITLE >
      <STYLE type="text/css">
        UL {list-style-position:inside; list-style-image:url("Newz.gif”);
            color: red; font-size: 20px; font-family: Arial Black}
      </STYLE>
    </HEAD>
    <BODY>
   <H2>Пример оформления списка с помощью стилей</H2><HR><BR>
      <UL>
        <LI>Архив </LI>
        <LI>Рассылка</LI>
        <LI>Подписаться</LI>
      </UL>
    </BODY></HTML>
```

6. Способы динамического управления страницей

Под динамическим изменением документа понимается изменение его отображения в окне браузера как результат перемещения курсора мыши над элементами страницы, которые, изменяя свой внешний вид, подсказывают пользователю, что можно увидеть скрытую часть документа и щелчком мыши инициировать выполнение некоторого действия. Например, при расположении курсора мыши над каким-либо текстом последний меняет цвет или размер и появляется всплывающая подсказка, что при щелчке кнопкой мыши можно увидеть элементы раскрывающегося списка.

При разработке динамических HTML-страниц следует всегда продумывать удобные и ясные подсказки пользователю, чтобы он смог полностью насладиться встроенными автором эффектами.

Фильтры и переходы. Создавать HTML-страницы с мультимедийными эффектами так же просто, как и применять каскадные таблицы стилей. Разнообразные визуальные динамические эффекты: постепенное «проявление» изображения или текста, изменение контрастности графического изображения, «свечение» букв текста и т. п. — все это и многое другое можно увидеть при отображении страницы в Internet Explorer 4.0 и более поздних версий. Реализу-

ются мультимедийные эффекты применением фильтров к элементам страницы и организацией переходов из одного визуального состояния к другому.

Фильтр — это некоторый алгоритм, преобразующий визуальное отображение элемента в окне браузера. Он может быть статическим или динамическим. Статический фильтр преобразует элемент, и после этого он отображается. Динамический фильтр воздействует во времени на визуальное отображение элемента, меняя его непосредственно на HTML-странице, что приводит к эффекту анимации. Динамический фильтр еще называют переходом из одного состояния отображения в другое.

Таблица 9 — Элементы, к которым применяются фильтры

Элемент	**Фильтр применяется**
BODY	Всегда
BUTTON	Всегда
DIV	Если заданы ширина (свойство width), высота (свойство height) или элемент абсолютно позиционирован
IMG	Всегда
INPUT	Всегда
MARQUEE	Всегда
SPAN	Если заданы ширина (свойство width), высота (свойство height) или элемент абсолютно позиционирован
TABLE	Всегда
TD	Всегда
TEXTAREA	Всегда
TH	Всегда

Фильтры можно применять и к элементам управления ActiveX, встраиваемым в HTML-страницу.

Фильтры не применяются к следующим элементам HTML-страницы: апплеты Java, IFRAME, SELECT, OPTION, P, EM, STRONG и ко всем заголовкам H1, H2 и т. д.

Применить фильтр к элементу просто: достаточно задать значение его свойства filter, следуя правилам задания свойств каскадных таблиц стилей filter: значение. Каждый фильтр, как отмечалось выше, реализует определенный алгоритм преобразования видимого отображения элемента, поэтому значение свойства filter задается в форме функции:

filter: имя_фильтра([параметры]);

параметры фильтра, если они присутствуют, задаются с использованием синтаксиса именованных параметров функции:

имя_параметра=значение_параметра

Некоторым фильтрам требуется несколько параметров, задаваемых через запятую, а некоторым фильтрам параметры вообще не нужны, но круглые скобки должны присутствовать обязательно.

К элементу можно применить несколько фильтров одновременно. В этом случае они задаются в виде списка с пробелом в качестве разделителя:

<IMG ID=img1 SRC="пример1.gif" STYLE="filter: blur(strength=50) fliph()>

В данном примере к графическому изображению применяются два фильтра: первый (blur) размазывает изображение на глубину в 50 пикселей, а второй (fliph) просто зеркально его отображает в горизонтальном направлении.

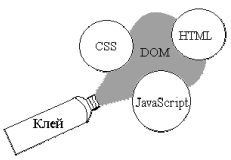
Интересные динамические эффекты достигаются использованием фильтров совместно со сценариями. В процессе выполнения сценария можно устанавливать или изменять параметры применяемых к объектам фильтров, можно назначать новые фильтры, создавать визуальные эффекты через определенные интервалы времени и делать многое другое.

Доступ к фильтрам и их параметрам в программируемых сценариях осуществляется, как обычно, с помощью объектной модели, предоставляемой браузером. В этой модели любой фильтр представляется в виде обычного объекта со своими свойствами и методами. Большинство свойств и методов соответствуют параметрам фильтра, определяемым при задании фильтра в параметре STYLE. Некоторые свойства и методы доступны только из программируемого сценария.

В наборе style объекта, в котором хранятся значения всех свойств каскадных таблиц стилей соответствующего объекту элемента HTML, хранится и значение свойства filter объекта, являющееся строкой, содержащей список применяемых к элементу фильтров.

Динамический HTML (Dynamic HTML или DHTML) — это термин, применяемый для обозначения HTML-страниц с динамически изменяемым содержимым.

Реализация DHTML покоится на трех «китах»: непосредственно HTML, каскадных таблицах стилей (Cascade Style Sheets — CSS) и языке сценариев (JavaScript или VBScript). Эти три компонента DHTML связаны между собой объектной моделью документа (Document Object Model — DOM), являющейся интерфейсом прикладного программирования (API). DOM связывает воедино три перечисленных компонента, придавая простому документу HTML новое качество — возможность динамического изменения своего содержимого без перезагрузки страницы.



Объектная модель документа (Document Object Model — DOM) — это то, без чего невозможен динамический HTML.

Объектная модель документа связывает в единое целое HTML, язык сценариев и каскадные таблицы стилей, предоставляя разработчикам веб-документов инструмент с совершенно новыми качествами — динамический HTML.

Динамический HTML в Internet Explorer 4.01 реализован на базе объектной модели DHTML, разработанной фирмой Microsoft и вошедшей в качестве подмножества в объектную модель Консорциума W3. Версия браузера Internet Explorer 5.0 и последующие полностью реализует объектную модель документа, совместимую с объектной моделью DHTML.

Объектная модель DHTML предоставляет разработчикам веб-документов прямой программируемый доступ ко всем элементам документа, а совместно с событийной моделью подобный подход позволяет браузеру обрабатывать ввод пользователя, выполнять встроенные сценарии и динамически менять содержимое документа, не перезагружая его.

Каждый элемент HTML-документа является в этой модели программируемым. Это означает, что к каждому элементу можно привязать сценарий, выполняемый в зависимости от того, какое действие над элементом произвел пользователь: поместил ли курсор мыши на элемент, перемещает курсор мыши в границах элемента, нажал или отпустил кнопку мыши и т. д. Результатом выполнения указанных действий является генерация определенного события, которое может быть привязано к конкретному сценарию, выполняемому в случае возникновения события. Для каждого элемента, задаваемого определенным тегом HTML, можно определить свой стиль отображения в окне браузера. Объектная модель документа делает все элементы страницы программируемыми объектами. С ее помощью через языки сценариев можно получить доступ и управлять всем, что есть в документе. Каждый элемент HTML доступен как индивидуальный объект, а это означает, что можно изменять значение любого параметра любого тега HTML-страницы, и, как следствие, документ действительно становится динамическим. Любое действие пользователя (щелчок кнопкой мыши, перемещение мыши в окне браузера или нажатие клавиши клавиатуры) объектной моделью документа трактуется как событие, которое может быть перехвачено и обработано процедурой сценария.

7. Команды Dynamic HTML

Dynamic HTML — это не какой-то новый язык, отличный от стандартного HTML, а набор определенных команд и способ их использования, позволяющий динамически управлять веб-страницей.

Внешний вид страницы, написанной на стандартном HTML, после загрузки страницы изменен быть не может. Для того чтобы сделать страницу на чистом HTML после ее полной загрузки хотя бы немного по-другому выглядящей, требуется ее полная перезагрузка. То есть страница не может быть интерактивной: не может изменяться, реагируя на действия посетителя. Для возможности изменять внешний вид веб-страницы без ее перезагрузки в ответ на определенные действия пользователя и был придуман Dynamic HTML. Чтобы возможности Dynamic HTML могли быть использованы, веб-страница должна просматриваться в браузере, способном обрабатывать команды Dynamic HTML.

Принцип Dynamic HTML. Каждому элементу страницы может быть присвоено имя — идентификатор ***id***. По этому идентификатору к элементу можно обращаться с помощью скриптов или команд Dynamic HTML, изменяя свойства этого элемента. Изменения сразу же вступят в силу, и внешний вид страницы станет другим. Перезагрузка страницы не потребуется. С помощью команд Dynamic HTML можно обращаться не только к поименованным с помощью идентификаторов объектам, но и вообще к любым объектам на странице. Можно, например, сделать так, что при определенном действии пользователя изменится

стиль (цвет, шрифт, размер) всех заголовков на странице или всех гиперссылок. Скажем, была страница оформлена в зеленых тонах, а теперь стала оформлена в синих. Причем для этого не нужны громоздкие долго загружающиеся конструкции — достаточно небольшого скрипта, использующего возможности Dynamic HTML. В Dynamic HTML существует возможность изменять не только свойства элементов страницы, но и ее содержание. К примеру, проводить замену одного текста на странице другим. Можно менять не только текст на странице, но и элементы ее оформления, например фон, рисунки или заглавие страницы (отображающееся в заголовке окна браузера).

Команды Dynamic HTML. Команды Dynamic HTML построены так же, как и команды любого современного обьектно-ориентированного языка программирования: *вначале пишется имя обьекта, над которым выполняется действие или свойство которого нужно узнать, а затем через точку — его подобъекты или свойства*. Они могут быть использованы как в отдельно взятом виде, так и в составе скрипта на VBScript или JavaScript. В первом случае они выполняются при происхождении какого-либо события, в описании которого и помещаются команды. Во втором случае выполнение команд происходит при выполнении скрипта.

Для изменения какого-либо элемента веб-страницы нужно присвоить этому элементу идентификатор (включить в его тег параметр ***"id="оригинальное имя""***) и затем с помощью команды изменить какое-либо свойство этого элемента.

Все элементы веб-страницы, как именованные, так и неименованные, связаны для браузера, поддерживающего Dynamic HTML, в разветвленную иерархическую структуру. Скрипты, использующие возможности Dynamic HTML, могут обращаться к различным элементам в этой структуре, учитывая его расположение в ней.

Команды Dynamic HTML можно включать как в скрипты, так и в текст веб-страницы. В последнем случае они должны быть включены в обработчики событий элементов страницы. С каждым элементом веб-страницы могут происходить различные события: на него могут навести курсор мыши, его могут кликнуть, его могут выделить и т. д. При каждом совершении того или иного события может быть выполнен скрипт или какая-либо команда Dynamic HTML.

VBScripts и JavaScripts, таблицы стилей, Dynamic HTML — все это не что иное, как «языки общения» сайта с посетителем, призванные сделать веб-страницу удобной и красивой. В то же время в основе любой веб-страницы, даже самой сложной, лежит стандартный язык HTML.

Ключевыми понятиями в DHTML являются слои и события.

Слой — это некий прямоугольный элемент, содержащий в себе любую разметку HTML. Слоем может быть как простая строка текста, так и сложная форма, сверстанная в таблице. С помощью JavaScript можно манипулировать таким слоем — изменять его размеры, видимость, перемещать его, а также изменять его <высоту>. В общем случае слой — это часть HTML-файла, выделенная тегом DIV, которому присвоен некий идентификатор (ID).

```
<div id="navmenu">
...
</div>
```

Но слой не станет слоем, если его не описать с помощью стилевых таблиц (CSS).

```
<style type="text/css">
#navmenu {POSITION: absolute; TOP:0; LEFT:0;
 Z-INDEX: 100; VISIBILITY: hidden; WIDTH: 250px;  HEIGHT: 400px;}
...
</style>
```

С помощью CSS описываются следующие параметры слоя.

POSITION — определяет точку отсчета координат. Если он равен absolute, то координаты отсчитываются относительно верхнего левого угла документа. Если он равен relative, то координаты отсчитываются от верхнего левого угла слоя, включающего данный слой.

TOP — определяет координату Y верхнего левого угла слоя.

LEFT — определяет координату X верхнего левого угла слоя.

Z-INDEX — описывает <уровень> слоя. Дело в том, что слои могут частично или полностью перекрываться. В этом случае Z-INDEX определяет, какой слой будет <выше> другого. Все слои, которые в принципе могут перекрываться, должны иметь различный Z-INDEX.

VISIBILITY — описывает начальную видимость слоя. Слой может быть изначально виден (visible) или спрятан (hidden).

WIDTH — задает ширину слоя. При этом надо помнить, что слой не будет растягиваться при увеличении в размерах его содержимого, например при увеличении пользователем размера шрифта. Поэтому все, что не поместится, будет обрезано по краю слоя. В принципе можно не указывать ширину слоя, но тогда в IE проявится один неприятный побочный эффект — ширина слоя будет принята равной ширине окна или фрейма. В результате, перекрыв собой другие элементы страницы, он не позволит выделить текст, нажать на ссылки или совершить другие мышиные действия над элементами в <нижележащих> слоях.

HEIGHT — задает высоту слоя, которая также должна быть достаточной, чтобы вместить все содержимое слоя.

Манипулирование слоем. В браузере IE доступ к свойству, определяющему видимость слоя, реализован так:

```
document.all["layername"].style.visibility= "visible";
```

После этого можно манипулировать свойствами слоев, пользуясь функцией **eval**.

```
eval(layerRef+'["layername"]'+styleSwitch+ '.visibility="visible"');
```

Изменение видимости слоя. Самое простое, что можно сделать со слоем, — это его спрятать или, наоборот, показать. Осуществляется это изменением свойства **visibility**. Если значение этого свойства устанавливается в visible, то слой будет отображен (конечно, если он находится в пределах окна), а если значение установлено в hidden, то слой будет спрятан. Для удобства напишем две функции — скрывания и показывания слоя.

```
function hideLayer(layerName){
 eval(layerRef+['"'+layerName+'"']'+styleSwitch+ '.visibility="hidden"');
}
function showLayer(layerName){
 eval(layerRef+['"'+layerName+'"']'+styleSwitch+ '.visibility="visible"');
}
```

Контрольные вопросы

1. Что такое CSS?
2. Что такое каскадность?
3. Как записывается любое правило каскадных таблиц?
4. Как описываются и сохраняются таблицы стилей?
5. Что такое стилевой класс?
6. Что такое селектор?
7. Что такое стиль-селектор?
8. В каких случаях применяют связывание документа и таблицы стилей? Какими способами?
9. Как задается стиль для элемента страницы?
10. Какое назначение имеет селектор CLASS?
11. В каких случаях применяют селектор ID?
12. В каких случаях применяют контекстные селекторы?
13. Укажите две причины использования CSS по рекомендациям Консорциума W3?
14. Что такое фильтр?
15. Как проявляются визуальные динамические эффекты?
16. Как реализуется DHTML?
17. Что такое DOM?
18. Понятие слоя в HTML?

Лекция 6. Расширяемый язык гипертекстовой разметки XML

План:

1. Характеристика, и возможности расширяемого языка разметки XML.
2. Язык описания схемы данных XML (DTD).
3. Правила создания XML-документа.
4. Способ формального описания структуры XML-документа.

1. Характеристика и возможности расширяемого языка разметки XML

XML (eXtensible Markup Language) — это упрощенный диалект языка SGML, предназначенный для описания иерархических структур данных в World Wide Web.

XML — это попытка решить перечисленные проблемы путем создания простого языка разметки, описывающего произвольные структурированные данные. Точнее говоря, это метаязык, на котором пишутся специализированные языки, описывающие данные определенной структуры. Такие языки называются XML-словарями. В отличие от HTML, XML не содержит никаких указаний

на то, как описанные в XML-документе данные должны отображаться. Способ отображения данных для различных устройств задается языком описания стилей XSL, который играет для XML примерно ту же роль, что CSS дл HTML. Другое принципиальное его отличие от HTML состоит в том, что XML может содержать любые теги, которые сочтут нужным использовать создатели XML-словаря.

Приведем список лишь нескольких специализированных языков на базе XML, которые сегодня находятся в разных стадиях разработки рабочими группами W3C:

MathML — язык математических формул;

SMIL — язык интеграции и синхронизации мультимедийных средств;

SVG — язык двумерной векторной графики;

RDF — язык метаописаний ресурсов;

XHTML — переформулировка HTML в терминах XML.

Основные отличия XML от HTML

1. XML был создан для хранения данных.
2. XML — это не замена HTML.
3. XML и HTML преследуют различные цели:

- XML создан для описания данных, концентрируя свое внимание на сущности этих данных;
- HTML был создан для показа данных, концентрируя внимание на том, как данные будут отображаться.
- HTML заботится об отображении информации для человека, тогда как XML заботится о сущности информации.

Процесс обработки XML-документа состоит в следующем. Его текст анализируется специальной программой, которая называется XML-процессором. XML-процессор ничего не знает о семантике данных в документе; он только производит синтаксический разбор (parsing) текста документа и проверяет его правильность с точки зрения правил XML. Если документ правильно оформлен (well-formed), то результаты разбора текста передаются XML-процессором прикладной программе, которая выполняет их содержательную обработку; если же документ оформлен неверно, т. е. содержит синтаксические ошибки, то XML-процессор должен сообщить о них пользователю.

Возникает вопрос: а какой смысл в использовании «пустого языка», лишенного собственного содержания? Дело в том, что, несмотря на внешнюю простоту, XML обладает достаточно изощренными механизмами контроля правильности данных, позволяет производить проверку иерархических отношений внутри документа и, самое главное, устанавливает единый стандарт для документов, хранящих данные, какова бы ни была природа этих данных.

2. Язык описания схемы данных XML (DTD)

Каждый XML имеет логическую и физическую структуру. Физически документ состоит из элементов, называемых сущностями. Любая сущность может ссылаться на другие сущности, обеспечивая их включение в данный документ. Документ начинается с «корня» или сущности документа. С логической точки

зрения документ строится из деклараций, элементов, комментариев, ссылок на символ и инструкций обработки. Все они размечаются в документе явным образом.

XML создан для описания данных, концентрируя свое внимание на **сущности** этих данных. HTML был создан для **показа** данных, концентрируя внимание на том, как данные будут отображаться.

XML не предназначен для осуществления каких-либо действий. Может быть это нелегко понять, но XML создан не для того, чтобы что-то делать. Он создан для хранения информации. Аня оставила Андрею записку. Посмотрим, как можно записать её в XML-виде:

```
<записка>
<кому>Андрей</кому>
<от_кого>Аня</от_кого>
<заголовок>Внимание!</заголовок>
<содержание>Не забудь перед уходом выключить утюг! </содержание>
</записка>
```

Как мы видим, в записке есть *заголовок* и *содержание* (смысловая часть) этой записки. В ней также есть информация о том, *кому* и *от кого* поступила эта записка. Но тем не менее этот XML-документ *не выполняет* никаких действий. Это всего лишь информация, заключенная в XML-теги. Например, для такого формата записки кто-нибудь может написать программу, которая будет составлять и посылать подобные записки.

XML — это свободный и расширяемый язык. XML-теги не определены заранее. Вы должны сами придумать свои теги. С помощь XML вы можете создать структуру документа, используя *свои собственные* теги. Теги примера, которые используются (например, <кому> и <от_кого>), нигде в XML стандарте не описаны. Эти теги придумал автор этого примера.

XML — это дополнение к языку HTML. Очень важно осознать, что XML — это не замена языку HTML. В будущем на плечи XML ляжет задача *описания данных*, тогда как HTML будет использоваться для представления этих данных. XML — это платформо-, программно- и аппаратно-независимое средство для передачи информации.

XML-шутка. ***Вопрос: когда мне нужно использовать XML? Ответ: когда надо вставить в резюме модное словечко.***

XML-валидация. XML-документ с правильным синтаксисом — это *правильный* документ. XML-документ, соответствующей определенному ОТД (определения типа документа, DTD, document type definition), — это *валидный* XML-документ.

«*Правильные» XML-документы*. «Правильные» ("well formed") XML документы составлены согласно XML-синтаксису. «Правильный» XML-документ — это документ, составленный согласно правилам языка XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<note>
    <to>Сергей</to>
    <from>Наталья</from>
```

```
    <heading>Напоминание</heading>
    <body>Не забудь про наши планы на эти выходные!</body>
</note>
```

Валидные XML документы. XML-документ, который также соответствует какому-либо **ОТД,** называется **валидным**. Валидный ("valid") XML-документ — это «правильный» документ, который также соответствует определенным правилам некоторого ОТД.

ОТД определяет элементы, которые можно использовать в XML-документе.

Цель ОТД — обозначить **правила** взаимного расположения элементов и конструкций в вашем типе XML-документа. ОТД определяет структуру XML документа и набор используемых элементов (т. е. их названий). Более подробно об ОТД и валидации вы можете прочитать в руководстве ОТД.

XML-схема — это **альтернатива** использованию ОТД, основанная на синтаксисе языка XML. Консорциум W3 поддерживает также и другой способ валидации, названный **XML-схемой**. Более подробно об XML-схеме вы можете прочитать в соответствующем руководстве.

Ошибки в XML-документе должны приводить к **прекращению** обработки XML-документа. Спецификация языка XML указывает на то, что программы должны **остановить обработку** XML-документа, если найдут в нем **синтаксические ошибки.** Такое правило вводится для того, чтобы *упростить* написание программ, читающих XML, а также для того, чтобы сделать XML-документы *совместимыми*. В HTML вы могли себе позволить написать страничку с кучей ошибок (забыть закрыть некоторые теги). Основная причина *громоздкости и несовместимости* работы веб-браузеров как раз в том, что каждый из браузеров имеет свое мнение на то, как надо вести себя при встрече ошибок. В случае XML такого быть не должно.

3. Правила создания XML-документа

Базовые правила XML

1. Форматированный документ соответствует минимальному набору правил, обеспечивающих возможность обработки документа браузером или другой программой.

2. Документ должен иметь только один элемент верхнего уровня (элемент Документ или корневой элемент). Все другие элементы должны быть вложены в элемент верхнего уровня.

3. Элементы должны быть вложены упорядоченным образом. То есть если элемент начинается внутри другого элемента, то он должен и заканчиваться внутри этого элемента.

4. Каждый элемент должен иметь начальный и конечный теги.

5. Имя типа элемента в начальном теге должно в точности соответствовать имени в соответствующем конечном теге.

6. Имена типов элементов чувствительны к регистру, в котором они набраны.

Поскольку описание XML-документа представляет собой простой текст, вы можете создать его, используя ваш любимый текстовый редактор, например

редактор Notepad, входящий в состав Microsoft Windows. Еще лучше воспользоваться редактором, в котором предусмотрена возможность анализа исходных кодов, например текстовым редактором Microsoft Visual Studio, рассчитанным на работу с Microsoft Visual C++, Microsoft Visual InterDev, Microsoft Visual J++ и другими приложениями Visual Studio.

Вы можете открыть XML-документ непосредственно через Internet Explorer точно так же, как вы бы открыли HTML веб-страницу. Если XML-документ не содержит связи с таблицей стилей, Internet Explorer помечает различные составные части документа разным цветом, чтобы облегчить их распознавание, а также представляет элемент Документ в виде иерархического дерева с возможностью свертывания и развертывания структуры и просмотра с меньшей или большей степенью детализации. Если же XML-документ имеет связь с таблицей стиля, Internet Explorer 5 и выше отобразит только символьные данные из элементов документа, отформатировав их в соответствии с правилами, установленными в таблице стиля. Вы можете использовать либо таблицу каскадных стилей (CSS-таблицу, аналогичную той, которая используется для HTML-страниц), либо XSL-таблицу стилей (Extensible Stylesheet Language), которая является более мощным инструментом и строится в соответствии с синтаксисом, принятым для XML. Такие таблицы могут использоваться исключительно для XML-документов.

Синтаксические правила языка XML однозначны и очень просты. Эти правила легко выучить и легко использовать. Именно по этой причине упрощается разработка программ, способных читать и обрабатывать XML.

Пример XML-документа. В XML используется самоопределяющийся и простой синтаксис.

```
<?xml version="1.0" encoding="Windows-1251"?>
<note>
<to>Сергей</to>
<from>Наталья</from>
<heading>Напоминание</heading>
<body>Не забудь про наши планы на эти выходные! </body>
</note>
```

Все элементы XML должны иметь закрывающий тег

Составляя XML-документ, вы не можете опускать закрывающие теги. В HTML многие элементы могут не иметь закрывающего тега. В XML все элементы должны иметь закрывающий тег:

```
<p>Это параграф</p>
<p>А это другой параграф</p>
```

Объявление — это не элемент XML-документа, а некий особый элемент, который не должен иметь закрывающего тега. В отличие от HTML, теги в XML чувствительны к регистру. В XML-документе тег <Letter> — это не то же самое, что тег <letter>. Открывающий и закрывающий теги должны быть написаны одинаково с учетом регистра.

```
<Message> Это неправильно! </message>
<message> А это правильно </message>
```

XML-элементы должны быть строго вложенными

Нестрого вложенные элементы можно позволить себе в HTML (но и это будет ошибкой):

 <i Этот текст печатается жирным курсивом

В XML-документе все элементы обязательно должны быть строго вложенными:

Этот текст печатается жирным курсивом

Все XML-документы должны иметь корневой элемент

Все XML-документы должны содержать единственную пару тегов, определяющую корневой элемент. Все остальные элементы должны быть потомками этого корневого элемента. Любой элемент может иметь свой дочерний элемент. Дочерние элементы должны быть строго вложены в родительский элемент:

```
<root>
   <child>
   <subchild>……</subchild>
   </child>
</root>
```

В этом примере: root — корень, child — потомок, subchild — потомок потомка.

Значения атрибутов должны быть в кавычках

В XML считается ошибкой, если вы напишете значение параметра, не заключив его в кавычки. XML-элементы могут иметь атрибуты, которые идут парой «название атрибута / значение атрибута» совсем как в HTML. В XML значения атрибутов должны записываться в кавычках.

```
<?xml version="1.0" encoding="Windows-1251"?>
<note date="12/11/22002">
<to>Сергей</to>
<from>Наталья</from>
</note>
```

В XML пробелы сохраняются

В XML все подряд идущие пробелы не сокращаются до одного, а воспринимаются как есть.

В HTML была другая ситуация, так как HTML подряд идущие пробелы, табуляции, переводы строки превращал в один-единственный пробел. В XML CR/LF преобразуется в LF.

В XML новая строка всегда обозначается как LF. В Windows приложениях новая строка обычно записывается как пара специальных символов: возврат каретки (CR — carriage return) и перевод строки (LF — line feed). Эта пара спецсимволов подобна действиям при печати на печатной машинке, в случае, когда вы переходите на следующую строку. В Unix приложениях новая строка обычно обозначается только символом LF.

Комментарии в XML. Синтаксис для записи комментариев в XML такой же, как и в HTML, <!-- Это комментарий --> *Именование элементов* XML элементы должны именоваться в соответствии со следующими правилами:

1. Имена могут состоять из букв, цифр и других символов.

2. Имена не могут начинаться с цифры или знака препинания.

3. Имена не должны начинаться с последовательности xml (или XML, или Xml и т. д.),

4. Имена не могут содержать пробелы.

Когда вы придумываете имена своим XML-элементам, придерживайтесь следующих простых правил:

- для имен нет ограничений, и у вас есть полная свобода, но важно, чтобы ваши имена элементов было легко понять. Имена с символом подчеркивания в качестве разделителя — это хороший стиль. Например: <first_name>, <last_name>;
- избегайте использования символов «-» и «.» в именах. Например, если элементу вы дали название «first-name», вполне вероятна ошибка, что часть программ попытается от first вычесть name. В случае если же вы назвали элемент «first.name», некоторые программы могут воспринять это имя как «параметр name элемента first».

XML-документы часто привязаны к какой-либо существующей базе данных, в которой элементы базы соответствуют элементам XML-документа. Хорошим тоном считается давать имена XML-элементам в соответствии с именами, существующими в базе данных.

В спецификации XML разрешается использовать кириллицу для имен элементов, но лучше избегать этого, поскольку различное ПО может неправильно обрабатывать документы, имена в которых написаны не английским алфавитом.

В именах не следует использовать символ «:», поскольку он зарезервирован для пространств имен, о которых можно узнать в более подробных учебниках.

4. Способ формального описания структуры XML-документа

Элементы и атрибуты

XML-документ состоит из деклараций, элементов, комментариев, специальных символов и директив.

XML — это теговый язык разметки документов. Иными словами, любой документ на языке XML представляет собой набор элементов, причем начало и конец каждого элемента обозначаются специальными пометками, называемыми тегами.

Элемент состоит из трех частей: начального тега, содержимого и конечного тега. Пример XML-элемента:

<author>Сергей Довлатов</author>

Имена элементов зависят от регистра, т. е. <author>, <Author> и <AUTHOR> — это имена различных элементов. Наличие закрывающего тега всегда обязательно. Если тег является пустым, т. е. не имеет содержимого и закрывающего тега, то он имеет специальную форму: <элемент/>.

Любой элемент может иметь атрибуты, содержащие дополнительную информацию об элементе. Атрибуты всегда включаются в начальный тег элемента и имеют вид:

имя_атрибута="значение_атрибута"

Атрибут обязан иметь значение, которое всегда должно быть заключено в одинарные или двойные кавычки. Имена атрибутов также зависят от регистра. Пример элемента, имеющего атрибут:

<author country="USA"> Сергей Довлатов </author>

Элементы должны либо следовать друг за другом, либо быть вложены один в другой.

Пролог и директивы

Любой XML-документ состоит из пролога и корневого элемента, например:

```
<?xml version="1.0"?>
    <books>
        <book isbn="0345374827">
         <title> Марш одиноких</title>
         <author> Довлатов, Сергей</author>
 <present/>
 </book>
</books>
```

Директива (processing instruction) — это выражение, заключенное в специальные теги "<?" и "?>", которое содержит указания программе, обрабатывающей XML-документ.

Стандарт XML резервирует только одну директиву <?xml version="1.0"?>, указывающую на версию языка XML, которой соответствует данный документ (второй версии XML пока нет). В действительности эта директива несколько богаче и в самом общем виде выглядит так: <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?> Здесь атрибут encoding задает кодировку символов документа. По умолчанию считается, что XML-документы должны создаваться в формате UTF-8 или UTF-16.

Разделы и их декларации

Разделы XML-документа. Физически XML-документ может состоять из нескольких разделов (entities). При этом корневой элемент документа также является разделом, который называется разделом документа, хотя он никак специально не оформлен. Все разделы имеют содержимое; все они, кроме раздела документа и внешней DTD, имеют имя.

С точки зрения синтаксического разбора документа разделы подразделяются на анализируемые и неанализируемые. Неанализируемый раздел (unparsed entity) — это ресурс, содержимое которого XML-процессор воспринимает как внешние данные без их синтаксического анализа (например, текст, не являющийся XML-документом). Неанализируемые разделы всегда имеют нотацию, указывающую на их формат. Анализируемые разделы (parsed entities) предназначены для текстовой подстановки: всякий раз, когда XML-процессор встречает в документе имя такого раздела, он заменяет его на содержимое этого раздела.

Декларации разделов подразделяются на внутренние и внешние.

Декларация внутреннего раздела имеет вид <!ENTITY имя значение>

Она включает в себя содержимое объекта (параметр значение) и используется для подстановки этого значения вместо имени раздела. Из этого примера видно, что ссылка на раздел (entity reference) выглядит точно так же, как ссылка на специальный символ, т. е. имеет вид &имя;. На самом деле специальные символы — это точно такие же ссылки, но соответствующие разделы заданы неявно во внутренней декларации языка XML. Подобные текстовые подстановки удобны для задания сокращений, позволяющих уменьшить объем документа, и для введения обозначений для часто изменяемых полей документа.

Существуют два варианта деклараций внешнего раздела:

<!ENTITY имя SYSTEM URI [NDATA нотация]?>

<!ENTITY имя PUBLIC строка? URI [NDATA нотация]?>

Первый вариант называется системным разделом, второй — публичным разделом. Они оба связывают имя раздела с внешним ресурсом, заданным своим URI, который должен иметь кодированную форму и не содержать закладок. URI внешнего ресурса называется системным идентификатором раздела. Использование внешнего ресурса зависит от нескольких факторов.

Если декларация содержит параметр NDATA, задающий нотацию раздела, то раздел является неанализируемым. Если параметр NDATA не задан, то раздел анализируемый и соответствующий ресурс должен быть XML-документом. Это означает, что вместо ссылки на раздел в текст документа будет включаться текст соответствующего ресурса.

Публичный раздел может содержать строку, задающую публичный идентификатор раздела. XML-процессор может использовать этот идентификатор для генерации альтернативного URI данного раздела. Если ему это не удалось, то он должен использовать системный идентификатор для загрузки содержимого раздела.

Внешний анализируемый раздел должен начинаться с директивы <?xml …?>, которая может не содержать номера версии, но обязана содержать кодировку символов. Эта директива не входит в состав подставляемого текста.

Декларация типа документа

Декларация типа XML-документа (document type declaration) содержит определение типа документа (document type definition, DTD) или указывает на него. DTD — это специальная грамматика, описывающая синтаксис определенного класса документов. Декларация типа документа, как и декларация раздела, может быть внутренней или внешней. Внутренняя декларация имеет вид: <!DOCTYPE имя [тело]>, а внешняя — те же два варианта, что и внешние разделы:

<!DOCTYPE имя SYSTEM URI [тело]>

<!DOCTYPE имя PUBLIC строка? URI [тело]>

Таким образом, отличие декларации типа документа от декларации раздела состоит только в том, что:

• она начинается с ключевого слова !DOCTYPE, а не !ENTITY; она может иметь тело, заключенное в квадратные скобки.

Имя такой декларации должно совпадать с именем корневого элемента, который она описывает, а тело должно соответствовать правилам построения DTD. Примеры внешних деклараций:

<!DOCTYPE spec SYSTEM "xml/1998/06/xmlspec-v20.dtd">

<!DOCTYPE spec PUBLIC "-//W3C//DTD Specification V2.0//EN" "/XML/1998/06/xmlspec-v20.dtd">

Отметим, что внешняя декларация типа документа может содержать и ссылку на DTD, которая называется внешним подмножеством DTD, и тело, которое описывает дополнения к внешней DTD (оно называется внутренним подмножеством DTD).

Анатомия XML-документа. XML-документ состоит из двух основных частей: пролога и элемента Документ (его также называют корневым элементом).

Пролог

В данном примере документа пролог состоит из двух строк:

<?xml version="1.0"?>

<!-- File Name: Inventory.xml -->

Первая строка представляет собой объявление XML, указывающее на то, что это XML-документ, и содержащее номер версии. Объявление XML не является обязательным, хотя спецификация требует его включения. Если вы включаете XML-объявление, оно должно находиться в начале документа.

Вторая строка пролога состоит из пробела. С целью улучшения внешнего вида документа вы можете вставлять любое количество пустых строк между элементами пролога. При обработке они будут игнорироваться.

Третья строка пролога представляет собой комментарий.

Пролог может также содержать следующие необязательные компоненты: объявление типа документа, определяющее тип и структуру документа.

Второй основной частью XML-документа является единый элемент Документ, или корневой элемент, который, в свою очередь, содержит дополнительные элементы.

В XML-документе элементы определяют его логическую структуру и несут в себе информацию, содержащуюся в документе (в нашем примере это информация о книгах, такая как название, автор, цена). Типовой элемент состоит из начального тега, содержимого элемента и конечного тега. Содержимым элемента могут быть символьные данные, другие (вложенные) элементы либо сочетание данных и вложенных элементов.

Контрольные вопросы

1. Что такое XML?
2. Каково назначение языка XML?
3. Как различаются языки HTML и XML?
4. Как организована структура XML-документа?
5. Какие особенности имеет язык XML?
6. Где применяется XML?

Тема 2. Основные принципы технологии структурного и объектно-ориентированного программирования

Лекция 7. Основные принципы технологии структурного и объектно-ориентированного программирования

План:

1. Структурное программирование.
2. Объектно-ориентированное программирование.

1. Структурное программирование

Структурное кодирование (программирование) — это метод кодирования (программирования), предусматривающий создание понятных, простых и удобочитаемых программных модулей и программных комплексов на требуемом языке программирования.

Структурнное программирование основано на следующих идеях:

- задача разбивается на большое число мелких подзадач, каждая из которых решается своей процедурой или функцией (декомпозиция). При этом проектирование программы идет по принципу сверху вниз: сначала определяются необходимые для решения программы модули, их входы и выходы, а затем уже эти модули разрабатываются. Такой подход вместе с локальными именами переменных позволяет разрабатывать проект силами большого числа программистов;
- любой алгоритм можно реализовать, используя лишь три управляющие конструкции: последовательное выполнение, ветвление и цикл; обратите внимание на обозначения в соответствии с действующим стандартом. Данное обстоятельство позволяет при наличии соответствующих операторов исключить из языка команду перехода GOTO.

Примеры языков программирования структурного стиля: Паскаль, Алгол.

В процессе разработки программы ее модульная структура может поразному формироваться и использоваться для определения порядка программирования и отладки модулей, указанных в этой структуре. Поэтому можно говорить о разных методах разработки структуры программы.

Обычно обсуждаются два метода: метод восходящей разработки и метод нисходящей разработки.

1) <u>Метод восходящей разработки</u>

Метод восходящей разработки заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня (листья дерева модульной структуры программы), в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование.

2) <u>Метод нисходящей разработки</u>

Метод нисходящей разработки заключается в следующем. Как и в предыдущем методе, сначала строится модульная структура программы в виде дерева.

Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается. После того как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке. Каждый имитатор модуля представляется весьма простым программным фрагментом, сигнализирующим в основном о самом факте обращения к имитируемому модулю с необходимой для правильной работы программы обработкой значений его входных параметров (иногда с их распечаткой) и с выдачей, если это необходимо, заранее запасенного подходящего результата. После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Таким образом, большой объем «отладочного» программирования заменяется программированием достаточно простых имитаторов используемых в программе модулей. Кроме того, имитаторы удобно использовать для подыгрывания процессу подбора тестов путем задания нужных результатов, выдаваемых имитаторами.

Конструктивный подход к разработке программы представляет собой модификацию нисходящей разработки, при которой модульная древовидная структура программы формируется в процессе программирования модуля. Сначала программируется головной модуль исходя из спецификации программы в целом, причем спецификация программы является одновременно и спецификацией ее головного модуля, так как последний полностью берет на себя ответственность за выполнение функций программы. В процессе программирования головного модуля, в случае, если эта программа достаточно большая, выделяются подзадачи (внутренние функции), в терминах которых программируется головной модуль. Это означает, что для каждой выделяемой подзадачи (функции) создается спецификация реализующего ее фрагмента программы, который в дальнейшем может быть представлен некоторым поддеревом модулей. Важно заметить, что здесь также ответственность за выполнение выделенной функции берет головной (может быть, и единственный) модуль этого поддерева, так что спецификация выделенной функции является одновременно и спецификацией головного модуля этого поддерева. В головном модуле программы для обращения к выделенной функции строится обращение к головному модулю указанного поддерева в соответствии с созданной его спецификацией.

2. Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Примеры языков программирования: C++, Object Pascal (Delphi).

ООП представляет собой отображение объектов реального мира, их свойств (атрибутов) и связей между ними при помощи специальных структур данных.

Структурное программирование подразумевает наличие ряда встроенных структур данных: целых, вещественных и строковых переменных, массивов, записей, при помощи которых и производится отображение свойств объектов реального мира.

При объектно-ориентированном подходе для объекта создается своя структура данных (класс), содержащая как свойства объекта (поля), так и процедуры для управления объектом (методы).

Базовые понятия ООП:

Объект — совокупность свойств (параметров) определенных сущностей и методов их обработки (программных средств). Объект содержит инструкции (программный код), определяющие действия, которые может выполнять объект, и обрабатываемые данные.

Свойство — характеристика объекта, его параметр. Все объекты наделены определенными свойствами, которые в совокупности выделяют объект множества других объектов.

Объект обладает качественной определенностью, что позволяет выделить его из множества других объектов и обусловливает независимость создания и обработки от других объектов.

Метод — программа действий над объектом или его свойствами. Метод рассматривается как программный код, связанный с определенным объектом; осуществляет преобразование свойств, изменяет поведение объекта. Объект может обладать набором заранее определенных встроенных методов обработки, либо созданных пользователем либо заимствованных в стандартных библиотеках, которые выполняются при наступлении *заранее определенных событий*, например однократное нажатие левой кнопки мыши, вход в поле ввода, выход из поля ввода, нажатие определенной клавиши и т. п. По мере развития систем обработки данных создаются стандартные библиотеки методов, в состав которых включаются типизированные методы обработки объектов определенного класса (аналог — стандартные подпрограммы обработки данных при структурном подходе), которые можно заимствовать для различных объектов.

Событие — изменение состояния объекта. Внешние события генерируются пользователем (например, клавиатурный ввод или вне кнопки мыши, выбор пункта меню, затек макроса); внутренние события генерируются системой.

Объекты могут объединяться в *классы* (группы или наборы). *Класс* — совокупность объектов, характеризующихся общностью применяемых методов обработки или свойств. Один объект может выступать объединением вложенных в него по иерархии других объектов.

Принципы ООП:

Абстрагирование — это выделение существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов и таким образом четко определяют его концептуальные границы для дальнейшего рассмотрения и анализа. Абстрагирование концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности его поведения от деталей их реализации. Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования.

Инкапсуляция — это процесс отделения друг от друга отдельных элементов объекта, определяющих его устройство и поведение. Инкапсуляция служит для того, чтобы изолировать интерфейс объекта, отражающий его внешнее поведение, от внутренней реализации объекта. Объектный подход предполагает, что собственные ресурсы, которыми могут манипулировать только методы самого класса, скрыты от внешней среды.

Абстрагирование и инкапсуляция являются взаимодополняющими операциями: абстрагирование фокусирует внимание на внешних особенностях объекта, а инкапсуляция не позволяет объектам-пользователям видеть внутреннее устройство объекта.

Модульность — это свойство системы, обусловленное возможностью ее декомпозиции на ряд внутренне связанных, но слабо объединенных между собой модулей. Инкапсуляция и модульность создают барьеры между абстракциями.

Иерархия — это ранжированная, или упорядоченная, система абстракций, расположение их по уровням. Основными видами иерархических структур применительно к сложным системам являются структура классов (иерархия по номенклатуре) и структура объектов (иерархия по составу). Примерами иерархии классов являются простое и множественное наследование (один класс использует структурную или функциональную часть соответственно одного или нескольких других классов), а примером иерархии объектов — агрегация.

Типизация — это ограничение, накладываемое на класс объектов и препятствующее взаимозаменяемости различных классов (или сильно сужающее ее возможность). Типизация позволяет защититься от использования объектов одного класса вместо другого или, по крайней мере, управлять таким использованием.

Параллелизм — это свойство объектов находиться в активном или пассивном состоянии и различать активные и пассивные объекты между собой.

Устойчивость — это свойство объекта существовать во времени (независимо от процесса, породившего данный объект) и/или в пространстве (при перемещении объекта из адресного пространства, в котором он был создан).

Важным качеством объектно-ориентированного подхода является согласованность моделей деятельности организации и моделей проектируемой системы, начиная со стадии формирования требований и до стадии реализации.

Программный продукт, созданный с помощью инструментальных средств ориентированного программирования, содержит объекты с их характерными свойствами, для которых разработан графический интерфейс пользователя.

Контрольные вопросы

1. Что такое структурное программирование?
2. Перечислите типы алгоритмов.
3. Какие формы служат для представления алгоритма?
4. Дайте определения базовым понятиям ООП.
5. Что такое объектно-ориентированное программирование?
6. Перечислите принципы ООП.

Лекция 8. Синтаксис языка VBA

План:

1. Обзор редактора Visual Basic.
2. Константы. Типы переменных.
3. Объектные переменные.
4. Использование массивов. Динамические массивы.
5. Области действия переменных, констант, процедур и функций.

1. Обзор редактора Visual Basic

VBA (Visual Basic for Applications) — это диалект языка Visual Basic, расширяющий возможности Visual Basic и предназначенный для работы с приложениями Microsoft Office и другими приложениями от Microsoft и третьих фирм. В настоящее время VBA встроен во все главные приложения MS Office — Word, Excel, Access, PowerPoint, Outlook, FrontPage, InfoPath; в другие приложения Microsoft, например Visio и Microsoft Project; в более чем 100 приложений третьих фирм, например CorelDraw и CorelWordPerfect Office 2000, AutoCAD и т. п.

VBA — самый удобный универсальный язык для работы с приложениями Office, у него есть множество преимуществ:

- создавать полноценные приложения на Visual Basic;
- использовать все возможности языка VBScript для создания скриптов администрирования Windows, для создания веб-страниц и многое другое.

VBA изначально был ориентирован на пользователей, а не на профессиональных программистов, поэтому создавать программы на нем можно очень быстро и легко. Кроме того, в Office встроены мощные средства, облегчающие работу пользователя: подсказки по объектам и по синтаксису, макрорекордер и т. п.

Программирование в Office — это прежде всего уменьшение количества повторяющихся действий (и ручной работы, которая для этого требуется).

Всего в редакторе Visual Basic предусмотрены 9 окон.

1. Project Explorer — окно проводника проекта. По умолчанию оно открыто и находится в левой части окна редактора Visual Basic. В нем можно просмотреть компоненты проекта и выполнить с ними множество операций.

2. UserForm — окно формы. Появляется тогда, когда вы редактируете пользовательскую форму из окна дизайнера формы.

3. Toolbox — панель инструментов управления. Из него можно добавить элементы управления в форму или в документ.

4. Properties — одно из самых важных окон. Через него можно просмотреть свойства элемента управления или компонента проекта и их изменить.

5. Code — окно программного кода. В этом окне выполняется основная работа по написанию кода макроса. При открытии программного модуля открывается автоматически.

6. Object Browser — обозреватель объектов. Необходим для получения информации о классах, доступных программе.

7. Watch — окно контролируемых выражений. Используется во время отладки для отслеживания значений выбранных переменных программы и выражений.

8. Locals — окно локальных переменных. Нужно для отслеживания во время отладки значений переменных текущей процедуры.

9. Immediate — возможность опять-таки при отладке выполнить отдельные строки программного кода и немедленно получить результат.

Окно проводника проектов (Project Explorer) при первой активизации редактора Visual Basic обычно открыто. Если оно случайно было закрыто, то вызвать его можно:

- нажав на клавиши <Ctrl>+<R>;
- нажав на кнопку Project Explorer на панели Standard;
- воспользовавшись меню View -> Project Explorer.

В окне Project Explorer представлено дерево компонентов вашего приложения VBA.

Стандартные модули — это просто блоки с текстовым представлением команд VBA. В нем может быть только два раздела:

- раздел объявлений уровня модуля (объявление переменных и констант уровня модуля);
- раздел методов модуля (расположение процедур и функций).

Модули классов позволяют создавать свои собственные классы — чертежи, по которым можно создавать свои собственные объекты. Обычно используются только в очень сложных приложениях.

Пользовательские формы являются одновременно хранилищем элементов управления и программного кода, который относится к ним, самой формой и происходящими с ними событиями.

Еще одна полезная возможность Project Explorer — возможность настроить свойства проекта. Для этого нужно щелкнуть правой кнопкой мыши по узлу Project (VBAProject в Excel) и в контекстном меню выбрать Project Properties (меню Tools -> Project Properties). В этом окне можно:

- изменить имя проекта. Эта возможность потребуется, если у вас есть ссылки на проект с одинаковыми именами;
- ввести описание проекта, информацию о файле справки и параметры, которые будут использоваться компилятором;
- защитить проект, введя пароль. Не зная этот пароль, проект нельзя будет просмотреть или отредактировать.

Если вам нужно создать свой макрос вручную, а макросов в данном документе еще нет, то нужно будет щелкнуть правой кнопкой мыши по узлу проекта и в контекстном меню дать команду Insert -> Module. В проекте будет создан новый модуль и сразу открыт в окне редактора кода.

Если вам нужно создать графическую форму с элементами управления (кнопками, текстовыми полями, ниспадающими списками и т. п.), то нужно щелкнуть правой кнопкой мыши по узлу проекта и в контекстном меню вы-

брать Insert -> UserForm. Новая форма будет создана и открыта в окне дизайнера форм.

Запуск редактора кода VBA, приемы редактирования текста

В редакторе кода выполняется основная часть работы по программированию, поэтому знать приемы работы с ним нужно очень хорошо. Открыть окно редактора кода можно множеством способов:

- выбрать View -> Code;
- нажать на кнопку <F7>;
- дважды щелкнуть по объекту модуля в Project Explorer.

Редактор программного кода — это обычный текстовый редактор, предназначенный для специализированной задачи — создания кода программы.

Список объектов и список событий в окне редактора кода VBA, раздел Declarations

В верхней части окна редактора кода находятся два списка. Список слева — это список объектов. В нем вы можете выбрать объект, к которому будет относиться ваш код. Если вы открыли программный код модуля, то здесь будет только пункт (General). Другое дело, если открыта форма — в этом списке вы сможете выбрать саму форму или любой ее элемент управления и записать для него код.

Список справа — это список процедур/событий. В нем есть раздел (Declarations) — объявления уровня всего модуля и список всех процедур (макросов) для стандартного модуля или событий, если создается код для формы. При выборе нужного события будет автоматически создана нужная процедура, обрабатывающая это событие.

В редактор кода встроено множество средств, которые облегчают жизнь разработчику. Самое полезное средство — это *получение списка свойств и методов*. В большинстве VBA-программ используются свойства и методы различных объектов.

2. Константы. Типы переменных.
Синтаксис VBA (Visual Basic for Applications)

Некоторые основные синтаксические принципы этого языка:

- VBA нечувствителен к регистру;
- чтобы закомментировать код до конца строки, используется одинарная кавычка (') или команда REM;
- символьные значения должны заключаться в двойные кавычки;
- максимальная длина любого имени в VBA (переменные, константы, процедуры) — 255 символов;
- начало нового оператора — перевод на новую строку;
- ограничений на максимальную длину строки нет (хотя в редакторе умещается только 308 символов). Несколько операторов в одной строке разделяются двоеточиями.

Типы данных

Типы данных относятся к самым фундаментальным понятиям любого языка программирования. В языке VBA предусмотрены следующие типы данных:

- числовые;
- строковые (символьные);
- дата и время (Data — от 01.01.100 до 31.12.9999);
- логический (Boolean — для хранения значений True и False);
- объектный (хранит ссылку на любой объект в памяти);
- Variant — специальный тип данных, который может хранить любые другие типы данных.

Таблица 10 — Числовые типы данных

Тип данных	Описание	Диапазон допустимых значений
Byte	Достаточно малое целое число	От 0 до 255
Integer	Не слишком большое целое число	От –32768 до 32767
Long	Большое целое число	От –2147483648 до 2147483647
Single	Значение одинарной точности с плавающей запятой	От –3,402823E38 до –1,401298E-45 для отрицательных значений и от 1,401298E-45 до 3,402823E38 для положительных значений
Double	Значение двойной точности с плавающей запятой	От –1,79769313486231E308 до 1,79769313486232E308
Currency	Большое число, для которого выделено 19 позиций, включая 4 позиции после запятой	От –922337203685477,5808 до 922337203685477,5807

Различные типы данных введены для рационального использования памяти ЭВМ.

Переменные — контейнеры для хранения изменяемых данных. Без них не обходится практически ни одна программа. Для простоты переменную можно сравнить с номерком в гардеробе — вы сдаете в «гардероб» какие-то данные, в ответ вам выдается номерок. Когда вам опять потребовались эти данные, вы «предъявляете номерок» и получаете их. Перед использованием переменной ее необходимо объявить, т. е. указать имя переменной и тип данных, который хранится (будет храниться в ней). Объявление каждой переменной делает программу надежной, ускоряет ее работу, уменьшает количество ошибок. В программе переменные объявляются в разделе переменных по следующему формату:

DIM имя_переменной AS тип [, имя_переменной AS тип],

Примеры:

DIM V **AS** SINGLE, POS **AS** INTEGER

DIM TOK **AS** SINGLE

Dim — это область видимости переменной. В VBA предусмотрены 4 ключевых слова для определения области видимости переменных:

Dim — используется в большинстве случаев. Если переменная объявлена как Dim в области объявлений модуля, она будет доступна во всем модуле, если в процедуре — только на время работы этой процедуры;

Private — при объявлении переменных в VBA значит то же, что и Dim;

Public — такая переменная будет доступна всем процедурам во всех модулях данного проекта, если вы объявили ее в области объявлений модуля. Если вы объявили ее внутри процедуры, она будет вести себя как Dim/Private;

Static — такие переменные можно использовать только внутри процедуры. Эти переменные видны только внутри процедуры, в которой они объявлены, зато они сохраняют свое значение между разными вызовами этой процедуры. Обычно используются для накопления каких-либо значений.

Можно вообще не применять раздел описания переменных, в этом случае всем переменным по умолчанию присваивается тип Variant. (Это плохой стиль программирования.)

Для запрета использования переменных без их объявлений рекомендуется в разделе объявлений модуля указать команду **OPTION EXPLICIT.** В этом случае при попытке использования предварительно не объявленной переменной редактор VBA будет сообщать об ошибке.

Правила выбора имен в VBA едины для многих элементов (переменные, константы, функции и процедуры и т. п.). Имя:

- должно начинаться с буквы;
- не должно содержать пробелов и символов пунктуации (исключение — символ подчеркивания);
- максимальная длина — 255 символов;
- должно быть уникальным в текущей области видимости;
- зарезервированные слова использовать нельзя.

Константы. *Константы* — еще один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения VBA-программы.

Константы очень удобны при работе с группами именованных элементов (дни недели, месяцы, цвета, клавиши, типы окон и т. п.). Они позволяют использовать в коде программы легко читаемые обозначения вместо труднозапоминаемых числовых кодов.

В VBA встроено множество служебных констант: календарных, для работы с файлами, цветами, формами, типами дисков и т. п. Просмотреть их можно через справочную систему VBA: Microsoft Visual Basic Documentation -> Visual Basic Reference -> Constants.

В программе константы объявляются в разделе констант. Записывается ключевое слово **CONST** и за ним имена (идентификаторы) и значения констант программы. Например, CONST g = 9.81, pi = 3.1415926, Rmin = 0.1.

3. Объектные переменные

Переменные и свойства объектов в VBA

Описанные в процедурах VBA действия выполняются над переменными или объектами. Переменные VBA аналогичны переменным других языков программирования.

Объекты, с которыми работают процедуры и функции VBA, представляют собой средство программного управления приложениями Office и созданными с их помощью документами. Как и в других объектно-ориентированных языках программирования, у объектов VBA имеются *свойства,* которые могут принимать различные значения. Эти значения можно использовать в инструкциях VBA в качестве переменных.

Иерархия общих объектов и отдельных приложений описывается объектными моделями.

Объектная модель описывает вложенность классов объектов, свойства, имеющиеся у объектов каждого из классов, а также *методы* — операции, с помощью которых можно менять значения этих свойств.

Использованию переменной обычно предшествует ее объявление — инструкция Dim, в которой указывается как имя переменной, так и тип данных, для хранения которых она предназначена. Хотя такая инструкция и не обязательна, и не должны содержать пробелов, знаков препинания и перечисленных выше специальных символов (за исключением последнего знака). Они не могут также совпадать с ключевыми словами VBA и именами стандартных объектов.

Область действия переменных VBA определяется *местом их описания*. Если переменная описана внутри процедуры или функции, ее область действия ограничивается этой процедурой или функцией. Любая попытка использования имени этой переменной вне процедуры или функции, содержащей ее описание, приведет либо к ошибке, либо к созданию новой переменной с тем же именем, но другой областью действия. Если переменная описана на уровне модуля (соответствующая ей инструкция Dim помещена непосредственно в модуль перед описанием процедур и функций), тогда ею можно будет пользоваться из любой процедуры или функции, описанной в этом модуле. Если переменной придется пользоваться из процедур и функций, описанных в различных модулях проекта, в этом случае ее следует описать в одном из его модулей, воспользовавшись вместо инструкции Dim совпадающей с ней по синтаксису инструкцией Public.

Имена стандартных объектов и их свойств определены в пределах всего проекта после подключения соответствующей *библиотеки объектов.* Библиотеки объектов (файлы с расширением .olb) содержат в себе описания классов объектов. Две стандартные библиотеки объектов содержат в себе объектную модель самого VBA и объектную модель компонентов, являющихся общими для всех приложений пакета Office. Кроме объектных библиотек, содержащих модели компонентов каждого из приложений, имеются библиотеки объектов, реализующих связь с Web и другие новые возможности пакета, а также библиотека OLE Automation, организующая автоматизацию совместной работы приложений пакета.

Основные объекты MS Excel

При создании приложения в VBA в основном происходит работа с объектами. Можно использовать объекты, предоставляемые VBA: элементы управления, формы и объекты доступа к данным. Можно также управлять объектами других приложений из приложения VBA. Можно даже создавать свои собственные объекты и определять для них дополнительные свойства и методы.

Объект — это комбинация кода и данных, которую можно рассматривать как одно целое. Объект может быть частью приложения, как элемент управления или форма. Целое приложение также может быть объектом.

Таблица 11 — Основные объекты MS Excel

Объект	Описание
Кнопка управления	Элементы управления на форме, например, кнопки управления и рамки, являются объектами
Форма	Каждая форма в проекте VBA является отдельным объектом
База данных	Базы данных являются объектами и содержат другие объекты, например поля и индексы
Диаграмма	Диаграмма в Microsoft Excel является объектом

Происхождение объектов

Каждый объект в VBA определен классом (class). Класс используется для создания объектов и определяет их характеристики. Приведем два примера взаимоотношений между классами и объектами в VBA:

- элементы управления Панели элементов управления в VBA представляют классы. Объект, известный как элемент управления, не существует, пока он не нарисован на форме. Когда создается элемент управления, создается копия, или экземпляр (instance) класса элемента управления;
- форма, используемая во время разработки, является классом. Во время выполнения VBA создает экземпляр класса формы.

Окно **Properties (Свойства)** отображает класс и свойство **Name (Имя)** объектов в разработанном приложении VBA.

Все объекты создаются как идентичные копии своих классов. Свойства индивидуальных объектов можно изменять.

Что можно делать с объектами?

Объект предоставляет в распоряжение разработчика готовый исполняемый код. Например, вместо того чтобы программировать собственные диалоги File Open (Открыть файл) и File Save (Сохранить файл), можно использовать элемент управления общим диалогом (объект, предоставляемый VBA). Можно написать собственный код управления планированием и ресурсами, но вместо этого проще использовать объекты Calendar (Календарь), Resources (Ресурсы) и Task (Задача), предоставляемые Microsoft Project (Проект).

VBA предоставляет инструменты, которые позволяют комбинировать объекты из различных источников. Можно строить решения, комбинируя самые мощные возможности VBA и приложений, поддерживающих Automation (Автоматизация) (ранее известная как OLE Automation).

Автоматизация (automation) — это свойство составной модели объекта (Component Object Model, COM) промышленного стандарта, используемого приложениями для раскрытия своих объектов инструментам разработки и другим приложениям.

Можно строить приложения, объединяя внутренние элементы управления VBA и объекты, предоставляемые другими приложениями.

Основы работы с объектами

Объекты VBA поддерживают свойства, методы и события. В VBA данные объекта (установки или атрибуты) называются свойствами, тогда как процедуры, которые оперируют с объектом, называются его методами.

Событие — это действие, распознаваемое объектом, например щелчок кнопкой мыши или нажатие клавиши клавиатуры, и программист может написать код, реагирующий на это событие.

Можно изменять характеристики объекта, меняя его свойства. Рассмотрим радио: одно из свойств радио — громкость (volume).

Radio.Volume = 5

Методы — это такая же часть объектов, как и свойства. В целом методы — это действия, которые можно выполнить, тогда как свойства — это атрибуты, которые устанавливаются или восстанавливаются. Например, чтобы позвонить по телефону, надо набрать номер (dial). Можно было бы сказать, что телефоны обладают методом Dial, и использовать этот синтаксис для набора номера 555111: Phone.Dial 5551111

События инициируются, когда изменяются некоторые свойства объекта. Например, радио может иметь событие VolumeChange (Изменение громкости). Телефон может иметь событие Ring (Звонок).

Управление объектами с помощью свойств

Индивидуальные свойства меняются. Некоторые можно установить во время разработки. Для этого лучше использовать окно *Properties (Свойств),* что позволяет вообще не писать никакого кода. Другие свойства недоступны во время разработки, следовательно, необходимо программировать установку таких свойств во время выполнения.

Свойства, которые можно установить или значения которых можно получить только во время выполнения, называются изменяемыми. Свойства, значения которых можно только прочитать во время выполнения, называются неизменяемыми.

Установка значений свойств

Значение свойства устанавливается, только если необходимо изменить внешний вид или поведение объекта. Например, свойство Text элемента управления TextBox изменяют, если необходимо изменить содержимое поля.

Для установки значения свойства применяется следующий синтаксис:

object.property = expression

Например,

Textl.Top = 200 ' Значение свойства Top равно 200 твипам.

Textl.Visible = True ' Отображает текстовое поле.

Textl.Text = "hello" ' Отображает 'hello' в текстовом поле.

Иерархия объектов

Иерархия объектов (object hierarchy) определяет, как объекты связаны друг с другом и как к ним можно обратиться. В большинстве случаев программисту нет необходимости заботиться об иерархии объектов VBA, однако:

- при обращении к объектам других приложений следует знать иерархию объектов этого приложения;
- работая с объектами доступа к данным, следует знать иерархию Data Access Objects (Объектов доступа к данным)

Создание объектов. Способы создания объекта:

- можно создавать ссылки на объект с помощью переменных;
- можно создавать собственные объекты с самого начала с помощью модулей классов;
- Можно создавать собственные наборы с помощью объекта collection (Набор).

4. Использование массивов. Динамические массивы

Массив — это коллекция переменных, которые имеют общие имя и базовый тип. Все элементы данных, сохраняемых в массиве, должны иметь один и тот же тип. Информация, сохраненная в массиве, может быть доступна в любом порядке.

Массив позволяет сохранять и манипулировать многими элементами данных посредством единственной переменной. Обработку массивов значительно упрощает использование циклов.

Одномерные массивы. Одномерный массив — это самый простой вариант массива, использующий обыкновенный список данных. Например, Вася, Петя, Коля, Миша, Ваня, Слава, Игорь, Юра, Саша, Вова. Это строковый массив, состоящий из 10 элементов. Дадим ему название My_Array.

Нумерация элементов в массиве начинается с 0. Такая система нумерации довольно распространена в программировании и называется нумерацией с нулевой базой.

Для доступа к данным, хранящимся в определенном элементе массива, следует указывать имя массива с последующим числом, называемым индексом элемента. Индекс всегда заключается в круглые скобки. Например, My_Array(3) — этому элементу нашего массива соответствует "Миша".

Поскольку нумерация с нулевой базой не очень удобна (так как мы привыкли считать с 1, а не с 0), то в VBA имеется директива компилятора, позволяющая исправить это «неудобство»: Option Base.

Директива компилятора имеет два варианта написания:

Option Base 0 — индексы массивов начинаются с 0 (установка по умолчанию). Option Base 1 — индексы массивов начинаются с 1.

Данная директива компилятора помещается в область объявлений модуля перед объявлениями любых переменных, констант или процедур. Нельзя помещать Option Base внутри процедуры. Можно иметь только один оператор Option Base в модуле, который влияет на все массивы, объявляемые в модуле.

Многомерные массивы. Одномерные массивы хорошо подходят для представления простых списков данных. Однако часто бывает необходимо представить таблицы данных в программах с организацией данных в формате строк и столбцов, подобно ячейкам в рабочих листах Excel. Для этого необходимо использовать многомерные массивы. Так адрес каждой ячейки листа состоит из двух чисел, одно из которых (номер строки) является первым индексом, а второе (номер столбца) — вторым индексом массива. Такой массив называется двумерным массивом. Добавив еще номер листа, получим трехмерный массив. VBA позволяет создавать массивы, имеющие до 60 измерений.

Статические и динамические массивы. Массивы, не меняющие число своих элементов, называются статическими массивами. Примером такого массива может служить вышеприведенный массив My_Array, содержащий 10 элементов.

Однако бывают ситуации, когда изначально неизвестно количество элементов в массиве, или же в процессе работы это количество может изменяться. Такие массивы называются динамическими массивами.

Динамический массив может увеличиваться или сжиматься, чтобы вмещать точно необходимое число элементов без напрасного расходования памяти.

Объявление массивов. Объявление массива с использованием оператора Dim имеет следующий синтаксис:

Dim VarName([Subscripts]) [As Type]

VarName — любое имя массива, использующее допустимый идентификатор имени;

Subscripts — измерение массива.

Если размерность массива больше единицы, то Subscripts разделяются запятыми.

Оператор Subscripts имеет следующий синтаксис:

[lower To] upper [,[lower To] upper]

lower — определяет нижний диапазон допустимых индексов для массива (необязательный аргумент);

upper — определяет верхний предел для индексов массива (обязательный аргумент).

Примеры правильного объявления массивов:

Dim Array_Str (1 To 10) As String — одномерный статический строковый массив, включающий 10 элементов;

Dim Array_Var() — динамический массив; Dim Array_Mult (0 To 5, 0 To 7) As Integer — двумерный статический массив целых чисел, включающий 6*8=48 элементов.

При объявлении массивов следует помнить, что включение оператора Subscripts в объявлении массива создает статический массив с фиксированным числом элементов, пропуск оператора Subscripts в объявлении массива создает динамический массив, а установка директивы компилятора Option Base влияет на общее число элементов в массиве.

Использование массивов. Для доступа к элементу массива необходимо указывать имя массива, за которым следует значение индекса, заключенное в круглые скобки.

В VBA имеется возможность при помощи оператора ReDim переопределять размерность массива, а во время объявления не указывать его размерность.

Синтаксис ReDim:

ReDim [Preserve] varname(subscripts) [As Type] [, varname(subscripts) [As Type]]

Varname — имя существующего массива; subscripts — размерность существующего массива;

Type — любой тип VBA.

Необходимо использовать отдельный оператор As Type для каждого массива, который определяется; Preserve — необязательный аргумент. Его использование приводит к тому, что данные, уже имеющиеся в массиве, сохраняются после изменения его размерности.

Примеры правильного использования оператора ReDim:

Dim Array_Month() As String — одномерный строковый динамический массив.

ReDim Array_Month(29) — устанавливает размерность динамического массива, равную 29 элементам.

ReDim Array_Month(1 To 30) — изменяет размер массива до 30 элемента.

ReDim Preserve Array_Month(1 To 31) — изменяет размер массива до 31 элемента, сохраняя содержимое.

Dim Array_DBL() As Single — объявляет динамический массив.

ReDim Array_DBL(2, 9) — делает массив двумерным.

ReDim Array_DBL(3, 7) — изменяет размер двумерного массива.

ReDim Preserve Array_DBL(1 To 3, 1 To 5) — изменяет последний размер массива, сохраняя содержимое.

Можно изменять только последнее измерение многомерного массива, когда используется ключевое слово *Preserve.*

Функции LBound, UBound возвращают нижнее и верхнее граничные значения индексов статического или динамического массива.

Синтаксис: **LBound (array_Name [, dimension]) UBound (array_Name [, dimension])**

array_Name — имя массива dimension — целое число (необязательный аргумент).

Определяет измерение массива, для которого надо получить верхний или нижний предел. При отсутствии dimension возвращается предел для первого измерения массива.

Очистка и удаление массивов при помощи Erase. Оператор Erase позволяет выполнять очистку для статических массивов и удаление — для динамических. Когда элементы массива заполнены, данные в массиве остаются до тех пор, пока пользователь не присвоит новые значения элементам массива или пока VBA не освободится от массива. Зачастую бывает, что в дальнейших вычислениях динамический массив ни при каких обстоятельствах использоваться не будет, поэтому нецелесообразно «держать» его в памяти компьютера, так как

это может сказаться на скорости работы программы. Или же может понадобиться очистить все значения в статическом массиве, устанавливая числовые значения на 0, а строковые — на пустые строки. Это можно осуществить при помощи вложенных циклов. Но можно сделать гораздо проще:

Erase My_Array.

Данный оператор обнуляет (если массив статический) или очищает (если массив динамический) массив My_Array.

Таблица 12 — Поведение оператора Erase

Тип статического массива	Действие оператора Erase
Любой числовой тип	Устанавливает элементы массива на 0
Любой строковый тип	Устанавливает элементы массива на строку нулевой длины, а для строк фиксированной длины — как все символы пробела
Тип Variant	Устанавливает элементы массива на Empty
Тип Object	Устанавливает элементы массива на Nothing
Любой пользовательский тип	Устанавливает каждую переменную в пользовательском типе индивидуально: численные — на 0; строковые — на строки нулевой длины; Variant — Empty; Object — Nothing

Оператор Erase удаляет из памяти динамические массивы, освобождая область памяти, ранее используемую этим массивом. При удалении динамического массива с помощью оператора Erase необходимо повторно создать массив с помощью оператора ReDim перед тем, как можно будет использовать этот определенный динамический массив снова. Поведение оператора Erase для статических массивов зависит от конкретного типа элементов массива.

5. Области действия переменных, констант, процедур и функций

Главное предназначение VBA — обработка данных. Некоторые данные сохраняются в объектах, например диапазонах рабочих листов. Другие данные хранятся в созданных вами переменных.

Переменная представляет собой именованное место хранения данных в памяти компьютера.

Переменные могут содержать данные разных *типов* — от простых логических, или булевых, значений (True или False) до больших значений с двойной точностью. Значение присваивается переменной с помощью оператора равенства.

Определение типов данных

- Тип данных указывает, в каком виде данные хранятся в памяти: как целые значения, действительные числа, текст и т. п.
- VBA может автоматически типизировать данные, что приводит к медленному выполнению операций и неэффективному использованию памяти.
- При явном объявлении типа данных всех используемых переменных VBA может выполнять дополнительную проверку ошибок на этапе компиляции.

• При явном объявлении типа данных программа работает быстрее и занимает меньше места в оперативной памяти.

• Чтобы обеспечить обязательное объявление всех используемых переменных, необходимо включить строку **Option.Explicit** в качестве первой инструкции в модуле VBA.

• Чтобы в тексте программы распознать тип данных переменной или константы, можно использовать стандартную приставку (префикс) в нижнем регистре в названии переменной в соответствии с приведенной таблицей.

Таблица 13 — Префиксы

Префикс	Тип данных
b	Boolean
i	Integer
l	Long
s	Single
d	Double
c	Currency
dt	Date / Time
str	String
obj	Object
v	Variant
u	Пользовательский

Объявление переменных

Если для переменной, используемой в процедуре VBA, не объявлен тип данных, то по умолчанию будет задан тип данных Variant. Данные, которые хранятся в Variant, изменяют свой тип в зависимости от того, какие операции над ними выполняются.

Функция определения типа данных

Для определения типа данных переменной используется функция VBA TypeName. *Пример:*

```
MyVar = "123"
MsgBox TypeName(MyVar)
MyVar = MyVar / 2
MsgBox TypeName(MyVar)
MyVar = "Ответ: " & MyVar
```

Область действия переменных. *Область действия* переменной определяет, в каких модулях и процедурах она может использоваться. Существуют следующие типы областей действия переменных (табл. 14).

Таблица 14 — Типы областей действия переменных

Область действия	Способ объявления переменной
Отдельная процедура	В процедуру включается оператор Dim или static
Отдельный модуль	Перед первой процедурой в модуле вводится оператор Dim или private
Все модули	Перед первой процедурой в модуле вводится оператор Public

Локальные переменные

Локальная переменная — это переменная, объявленная в процедуре.

Локальные переменные могут использоваться только в процедуре, в которой они объявлены.

После выполнения процедуры переменная становится невостребованной, поэтому Excel освобождает соответствующую область памяти.

Если требуется сохранить значение переменной, объявите ее как static.

Чтобы объявить локальную переменную, вставьте оператор Dim между операторами Sub и End Sub. Dim — сокращение от *Dimension (Размерность)*. В старых версиях BASIC этот оператор использовался исключительно для объявления размерности массива.

Другой способ указания типа данных для переменной: язык VBA позволяет присоединить символ к названию, чтобы указать ее тип данных. Пример, можно объявить переменную MyVar как целое число, добавив к ее названию символ %: Dim MyVar %.

Символы объявления типов данных представлены для большинства типов данных VBA (отсутствующие в таблице типы данных не имеют собственного символа объявления типа).

Локальные переменные позволяют экономно использовать память, так как VBA освобождает память, которую они используют, после окончания выполнения процедуры.

Таблица 15 — Типы данных и символы объявления типа

Тип данных	Символ объявления типа
Integer	%
Long	&
Single	!
Double	#
Currency	@
String	S

Переменные уровня модуля. Иногда необходимо, чтобы переменная была доступна во всех процедурах модуля. В таком случае объявите переменную перед первой процедурой модуля (за пределами процедур или функций).

```
Dim CurrentValue As Integer
Sub MySub ()
  ' Текст процедуры
End Sub
Sub YourSub ()
  ' Текст процедуры
End Sub
```

Значение переменной уровня модуля не изменяется при окончании выполнения процедуры.

Переменные Public. Чтобы сделать переменную доступной во всех процедурах всех модулей VBA в проекте, необходимо объявить переменную на уровне модуля с помощью ключевого слова Public перед первой процедурой модуля, например, так: Public CurrentRate as Long.

Код объявления переменных Public должен вводиться в стандартном модуле VBA, а не в коде модуля листа или формы.

Переменные Static. Переменные Static — особый случай. Они объявляются на уровне процедуры и сохраняют свое значение после окончания процедуры.

```
Sub MySub()
Static Counter As Integer
   ' Текст процедуры
End Sub
```

Работа с константами

Константа — именованное значение или строка, которая не меняется при выполнении программы. Константы объявляются с помощью оператора Const.

Примеры:

```
Const NumQuarters as Integer = 4
Const Rate = .0725, Period = 12
Const ModName as String = "Budget Macros"
Public Const AppName as String = "Budget Application"
```

Во втором примере тип данных не объявлен. Следовательно, указанные две константы имеют тип Variant.

Таблица 16 — Область действия констант

Область действия	Способ объявления константы
Отдельная процедура	В процедуре или функции
Отдельный модуль	Перед первой процедурой в модуле
Все модули	Перед первой процедурой в модуле с ключевым словом Public

Управление строками

В VBA представлено два типа строк.

Строки фиксированной длины объявляются с определенным количеством символов. Максимальная длина строки составляет 65535 символов.

Строки переменной длины теоретически могут вмещать до 2 млрд символов.

Работа с датами. Переменная, определенная как Date, занимает 8 байт памяти и может содержать даты в диапазоне от 1 января 100 года до 31 декабря 9999 года. В VBA дата и время определяются как значения, заключенные между знаками #.

Переменная объекта — это переменная, представляющая целый объект, например диапазон или рабочий лист. Переменные объектов имеют особое значение по двум причинам:

- значительно упрощают программу;
- ускоряют выполнение программы.

Переменные объектов, как и обычные переменные, объявляются с помощью оператора Dim или Public.

Контрольные вопросы

1. Для чего предназначен язык VBA?
2. Какие преимущества имеет язык VBA?
3. Как открыть редактор Visual Basic?
4. Сколько окон имеет редактор Visual Basic?
5. Что такое стандартные модули?
6. Что такое переменная?
7. Что такое константа?
8. Как объявляются переменные в программе?
9. Как объявляются константы в программе?
10. Как запретить использование переменных без их объявлений?
11. Какие типы предусмотрены в VBA?
12. Что такое массив? Как объявляется массив?
13. Какие существуют типы областей действия переменных?
14. Что такое локальная переменная? Как она задается?
15. Как представлены строки в VBA?

Лекция 9. Операторы языка VBA

План:

1. Структура программы.
2. Процедуры. Расположение процедур в модуле.
3. Передача данных при вызове подпрограмм. Функции.
4. Условный оператор.
5. Операторы цикла.

1. Структура программы

Программы на VBA хранятся в проектах. Проект содержит модули различных типов, а модули включают различные процедуры.

Процедура — это наименьшая единица программного кода, на которую можно ссылаться по имени и которая может выполняться независимо. В VBA основные типы процедур — это Sub и Function. Любая процедура содержит один или более операторов, помещенных между двумя специальными операторами: объявлением процедуры в начале и оператором завершения процедуры в конце.

Модуль — это именованная единица, состоящая из одной или нескольких процедур, а также объявлений, относящихся ко всем процедурам в модуле.

В VBA имеются два типа модулей: стандартный модуль, который содержит программный код, предназначенный непосредственно для выполнения;

модуль класса, в котором определяются пользовательские объекты с их свойствами и методами.

Программа на языке VBA записывается в стандартном модуле и оформляется в виде процедуры пользователя. Формат процедуры (программы) в общем случае следующий:

SUB имя_процедуры (список аргументов)

последовательность инструкций (операторов)

END SUB

Инструкции представляют собой последовательность логических строк. Каждая строка состоит из одного или нескольких операторов, или процедур. Несколько операторов в одной строке друг от друга отделяются двоеточием

Ключевые слова должны записываться в точном соответствии с их синтаксисом. Рекомендуется использовать комментарии, поясняющие ход выполнения программы. Комментарии вводятся в программу с помощью оператора REM. Формат оператора REM ***комментарий*** или ‘***комментарий*.**

Обычно в начале программы указывается ее название, затем идут разделы описания констант и переменных, затем текст собственно программы. При составлении программ желательно использовать форматирование (структурирование) программы, т. е. выделять блоки или фрагменты, применять отступы и т. д.

Основные операторы (инструкции) языка

Оператор — это наименьшая способная выполняться единица кода VBA. Операторы VBA: арифметические, логические, сравнения, присвоения. Оператор может объявлять или определять переменную, устанавливать параметр компилятора VBA или выполнять какое-либо действие в программе.

Арифметических операторов в VBA всего 7. Четыре стандартных: сложение (+), вычитание (-), умножение (*), деление (/). и еще три: возведение в степень (^); целочисленное деление (\); деление по модулю (Mod).

Операторов сравнения в VBA всего 8: равенство (=); больше, чем, и меньше, чем (> и <); больше или равно и меньше или равно (>= и <=); не равно (<>); сравнение объектов (Is). Определяет, ссылаются объектные переменные на тот же объект или на разные, например If(obj1 is obj2); подобие (Like). Сравнивает строковый объект с шаблоном и определяет, подходит ли шаблон.

Логические операторы:

AND — логическое И, должны быть истинными оба условия;

OR — логическое ИЛИ, должно быть истинным хотя бы одно из условий;

NOT — логическое отрицание, возвращает TRUE, если условие ложно;

XOR — логическое исключение. В выражении E1 XOR E2 возвращает TRUE, если только E1 = TRUE или только E2 = TRUE, иначе — FALSE;

EQV — эквивалентность двух выражений, возвращает TRUE, если они имеют одинаковое значение;

IMP — импликация, возвращает FALSE, если E1 = TRUE и E2 = FALSE, иначе — TRUE.

Операторы конкатенации: + или &.

Пример: MsgBox "Сообщение пользователю" & vUserName

Оператор присваивания служит для вычисления значения выражения и присваивания этого значения переменной.

Формат оператора:

Имя_переменной = выражение

Примеры: A=2.1: SUMMA=X+COS(X) ^2

При присвоении переменным строковых значений их необходимо заключать в кавычки:

T = «Параметр 1», а значения даты\времени заключать в символы # («решетка»): D = #11/29/2006#

Операторы (процедуры) ввода данных

Для работы практически любой программы необходимы исходные данные различных типов. Передать их в программу можно несколькими способами.

1. Использование *оператора присваивания*.

Примеры: A = 3: TOK = 480: I = 1. Этот способ используется, если исходные данные не изменяются при нескольких исполнениях программы.

2. При работе с электронными таблицами применяют считывание данных из ячеек листа рабочей книги Excel. Для этого используется инструкция **Cells(i, j)**, которая в данном случае выступает как функция ввода данных. Формат использования:

Имя_переменной = Cells(i, j),

где **i, j** — соответственно номер строки и номер столбца, на пересечении которых находится ячейка, т. е. адрес ячейки. Пример: A = Cells(1, 2).

После выполнения этой строки переменной A присвоится значение, которое хранится в ячейке B1 электронной таблицы.

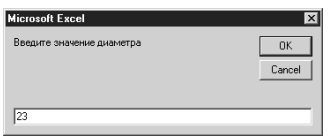
3. Ввод данных непосредственно в ходе выполнения программы, т. е. в диалоговом режиме, выполняется с помощью диалогового окна ввода информации, реализуемого функцией **InputBox.**

Формат использования (в простейшем случае):

Имя_переменной = InputBox(“Сообщение”).

В ходе работы программы при выполнении указанной функции на экране монитора появляется диалоговое окно, содержащее текст, указанный в «Сообщении», а также поле ввода. После ввода информации и нажатия на кнопку OK переменной присваивается значение типа **String** (строковый тип данных), содержащее текст, введенный в поле ввода.

Пример: d = InputBox (“Введите значение диаметра”). На экране появится диалоговое окно.

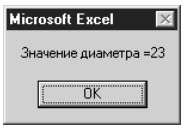


В примере переменной d будет присвоено значение строки “23”, а не числа 23.

Для преобразования строкового типа данных в числовой тип используется функция **Val(Строка)**, которая возвращает число, содержащееся в строке, как числовое значение соответствующего типа.

При записи в коде программы d = Val(InputBox (“Введите значение диаметра”)) и вводе в поле ввода цифр 23 переменной d присвоится число 23.

Операторы (процедуры) вывода данных



Вывод информации на экран монитора в VBA осуществляется двумя способами.

1. С помощью процедуры вывода **MsgBox (Сообщение).**

Эта процедура выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия пользователем кнопки (выполнение программы приостанавливается). После нажатия кнопки выполнение программы продолжается.

Пример: MsgBox (“Значение диаметра =” & d)

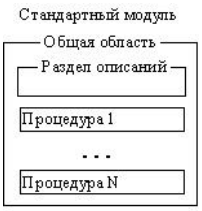
Сообщение может содержать комментарий, заключенный в кавычки, а также имя переменной, значение которой необходимо вывести в окне. Символ «&» означает слияние в одну строку всех символов, записанных в скобках. В результате на экране появится следующее диалоговое окно.

2. Вывод данных на лист рабочей книги Excel с использованием инструкции **Cells(i, j)**. В этом случае, в отличие от ранее рассмотренного, она выступает как процедура вывода:

Cells(i,j)= «результат»,

где «результат» — значение переменной, которое помещается в ячейку с адресом, определяемым номером строки **i** и номером столбца **j**.

2. Процедуры. Расположение процедур в модуле



Основой программ на VBA являются процедуры и функции, состоящие из инструкций, которые выполняют необходимые действия и вычисления.

Модуль представляет собой текстовый ASCII-файл с программным кодом, содержащим подпрограммы, переменные и константы. Проект может состоять из множества программных модулей. Для их создания необходимо выполнить команду **Вставка** ⟶ **Модуль**.

Основу программ в VBA составляют процедуры и функции.

Процедура Sub — это обособленная совокупность операторов VBA, выполняющая определенные действия. В общем случае процедура принимает некоторые параметры (переменные, которые передаются процедуре в качестве исходных данных), выполняет программу и может возвращать результирующие значения, которые присваиваются параметрам внутри процедуры. Однако чаще используются процедуры без параметров. Например, процедуры, выполняющиеся при возникновении определенных событий. Вложенность процедур в другие процедуры не допускается.

Структура процедуры:

[ДОСТУП] Sub ИМЯ_ПРОЦЕДУРЫ ([СПИСОК_ПАРАМЕТРОВ])

ТЕЛО_ ПРОЦЕДУРЫ

End Sub

Ключевое слово *ДОСТУП* является необязательным и определяет область видимости процедуры. **Public** указывает, что процедура доступна для всех других процедур во всех модулях (глобальная). **Private** указывает, что процедура

доступна для других процедур только того модуля, в котором она описана (локальная).

СПИСОК_ПАРАМЕТРОВ также является необязательным элементом и позволяет передавать процедуре различные исходные данные при вызове, которые называются формальными параметрами. При этом ключевое слово **Dim** не указывается.

ТЕЛО_ПРОЦЕДУРЫ состоит из описательной части и блока операторов, выполняющихся один за другим. Если необходимо прекратить выполнение процедуры в некотором конкретном месте, это можно сделать с помощью оператора **Exit Sub.**

ИМЯ_ПРОЦЕДУРЫ — это любой идентификатор, определенный пользователем.

Функция Function во многом похожа на процедуру, но, в отличие от неё, при вызове всегда возвращает значение. Функция получает параметры, называемые **аргументами**, и выполняет с ними некоторые действия, результат которых возвращается функцией. Структура функции следующая:

[ДОСТУП]* Function *ИМЯ_ФУНКЦИИ*(*СПИСОК_АРГУМЕНТОВ*) As *ТИП
ТЕЛО_ ФУНКЦИИ
ИМЯ_ФУНКЦИИ = ВЫРАЖЕНИЕ
End Function

ТИП определяет тип данных возвращаемого результата. В теле функции обязательно должен присутствовать, по крайней мере, один оператор, присваивающий имени функции значение вычисляемого выражения. Досрочное завершение функции возможно с помощью оператора **Exit Function.** В программе вызов функции осуществляется с помощью оператора присваивания, в правой части которого указывается имя функции с перечнем фактических параметров.

Процедуры и функции, не описанные явно с помощью ключевых слов Public или Private, по умолчанию являются общими.

Для быстрого добавления в модуль подпрограмм удобно воспользоваться командой **Вставка → Процедура.** В появившемся окне нужно выбрать необходимые опции.

Общие принципы организации программ VBA в модуле следующие.

- Обычно текст программы начинается с опций, которые управляют описанием переменных, способом сравнения строк и т. д.
- Затем следует объявление **глобальных** для данного модуля переменных и констант, т. е. таких, которые используются во всех процедурах модуля.
- Далее располагают непосредственно текст функций и процедур, составляющих саму программу.
- Разделителем операторов в одной строке при записи программы является символ «:».
- Для переноса оператора на другую строку используется символ «_» (знак подчеркивания).

Служебные слова Private и Public задают область видимости процедур и функций. Private делает объект доступным только внутри данного модуля. Public делает объект доступным из другого модуля.

Выполнение программного кода процедуры или функции происходит при передаче на нее управления. Передача управления (вызов) может осуществляться различными способами.

В общем случае подпрограмма вызывается из программного кода с помощью специальной инструкции Call, в которой кроме этого ключевого слова указываются имя процедуры и фактические параметры вызова, список которых заключается в круглые скобки.

3. Передача данных при вызове подпрограмм. Функции

Visual Basic — *процедурный язык программирования*. Это означает, что в нем можно создавать блоки программного кода, на которые можно впоследствии ссылаться по их имени. После того как блок кода получит имя, он может быть *вызван* и выполнен. Маленькие программы, «живущие» в больших программах, называются *функциями*, если они возвращают какое-либо значение, и *процедурами*, если они значений не возвращают, и объединенными общим термином *подпрограмма*. Они делают процесс программирования более простым и быстрым, а создаваемый код — более надежным.

Разработки собственных процедур и функций — первый шаг к разработке *инкапсулированного* и повторно используемого кода. Под инкапсуляцией понимают сокрытие реализации свойств и методов объекта за его внешним интерфейсом.

Передача аргументов в процедурах и функциях

Возможность подпрограмм и функций можно расширить с помощью *аргументов*. Аргумент, который также называют *параметрами*, — это переменные, используемые для хранения значений, которые будут переданы в процедуру или функцию. Они записываются в круглых скобках в заголовке соответствующего блока через запятую:

Public Sub SubName(NumOne As DataType, NumTwo As DataType)

Использование аргументов в процедурах и функциях делает код более универсальным. При использовании процедур и функций очень важно, чтобы передаваемые значения соответствовали типам данных аргументов и их порядку. Тем не менее аргумент можно сделать необязательным, если поместить ключевое слово «Optional» перед его описанием в операторе объявления. Все аргументы, описанные после этого ключевого слова, также будут необязательными и все они должны иметь тип «Variant»:

Public Sub SubName(NumOne As DataType, Optional NumTwo)

Для облегчения передачи в подпрограмму аргументов их можно именовать.

Именованный аргумент — это буквальное имя аргумента в подпрограмме.

Чтобы передать в нее значения, нужно использовать имена аргументов и присвоить им значения с помощью символов «:=». Для процедуры

Public Sub SubName(NumOne As Integer, NumTwo As String)

ее вызов с применением именованных аргументов выглядит следующим образом:

SubName(NumTwo:=3, NumOne:=4)

При работе с аргументами можно использовать два необязательных ключевых слова: «ByVal» и «ByRef», записываемых для каждого параметра.

Ключевое слово «ByVal» предусматривает, что в аргумент может быть передано только одно конкретное значение (число, строка, но не массив!), при этом соответствие типов не является жестким и находится в рамках автоматического их преобразования. Для передачи параметров по значению используется ключевое слово ByVal, например:

```
Sub PostAccounts(ByVal intAcctNum as Integer)
    <операторы тела процедуры>
End Sub
```

Фактические параметры, заданные константами, всегда передаются по значению. Ключевое слово «ByRef» означает, что в аргумент может быть передан любой объект Visual Basic (числа, строки, массивы, классы и др.). Соответствие типа в этом случае является обязательным. Другой его особенностью является то, что возможна обратная связь при вызове подпрограммы: при изменении значения аргументов происходит соответствующее изменение переменных, указанных в ее вызове. Иногда возникает необходимость досрочного выхода из процедуры или из функции, не дожидаясь ее окончания. Для этого пользуются операторами Exit: Exit Sub или Exit Function.

В целом смысл применения процедур и функции в программном коде приложения имеет следующие основные преимущества:

- вызов одной и той же подпрограммы удовлетворяет множество запросов из разных частей программы;
- чтобы каким-то образом усовершенствовать вычисления, не придется просматривать весь программный код. Достаточно лишь изменить его фрагмент внутри блока данной подпрограммы;
- внутренние структуры подпрограммы можно сделать полностью локальными, не влияющими на остальную часть программного кода (инкапсуляция).

VBA содержит встроенные или стандартные функции, например, sqr cos или chr. Кроме того, с помощью оператора Function можно писать собственные процедуры Function. Эти функции называются нестандартными или пользовательскими.

Встроенные функции

В VBA имеется большой набор встроенных функций, использование которых существенно упрощает программирование. Эти функции можно разделить на следующие основные категории:

- математические функции;
- функции обработки строк;
- функции проверки, определения и преобразования типов данных;
- функции времени и даты;
- финансовые функции и др.

Функция задается с помощью указателя функции, который записывается в виде имени, и аргументов, заключенных в круглые скобки. Аргументами могут быть константы, переменные, функции, арифметические выражения.

Таблица 17 — Математические функции

Название функции	Запись в VBA
Синус	SIN(X)
Косинус	COS(X)
Тангенс	TAN(X)
Арктангенс	ATN(X)
Показательная	EXP(X)
Логарифм натуральный	LOG(X)
Логарифм десятичный	LOG(X)/ LOG (10)
Корень квадратный	SQR(X)
Абсолютное значение	ABS(X)
Случайное число	RND(n)

Как и процедура Sub, процедура Function является самостоятельной и может принимать параметры, выполнять ряд операторов и изменять значения своих параметров. В отличие от процедуры Sub, имя процедуры Function может возвращать значение в вызывающую процедуру. Существуют три различия между процедурами Sub и Function:

- возвращаемое процедурой Function значение присваивается самому имени <имя процедуры> процедуры. Возвращаемое процедурой Function значение можно использовать в выражениях в программе;
- как и переменные, процедуры Function имеют тип, который определяет тип возвращаемого значения;
- вызов процедуры Function, или просто функции, в основном осуществляется заданием ее имени и параметров в правой части большого оператора или в составе выражения.

Обычно вызов пользовательской функции (нестандартной функции, написанной программистом) аналогичен вызову встроенной (стандартной) функции VBA, например Abs.

Функцию можно вызывать так же, как и процедуру sub.

Следующие операторы вызывают одну и ту же функцию:

Call Year(Now)

Year Now

Значение функции при ее вызове подобным образом игнорируется.

Рекурсия. Под рекурсией понимают способность программы вызывать саму себя. Процедуры выделяют для хранения переменных ограниченный объем памяти в стеке.

Стек — это рабочая область памяти, которая заполняется и освобождается динамически в соответствии с потребностями выполняемой программы. При каждом вызове процедурой самой себя выделяется дополнительный объем этой памяти. Процедура, вызывающая сама себя, называется рекурсивной. Рекур-

сивная процедура, которая бесконечно вызывает саму себя, приводит к ошибке. Например,

```
Function RunOut(Maximum)
    RunOut = RunOut(Maximum)
End Function
```

Эта ошибка менее очевидна, если две процедуры бесконечное число раз вызывают друг друга, или некоторое условие, ограничивающее рекурсию, никогда не выполняется. Рекурсия имеет свои области применения.

Необходимо проверить, что рекурсивная функция не вызывает себя столько раз, что начинает сказываться нехватка памяти. При возникновении ошибки, убедитесь, что процедура не вызывает себя бесконечное число раз. Можно сэкономить память с помощью:

- устранения ненужных переменных;
- использования типов данных, отличных от Variant;
- переосмысления логики процедуры. Часто вместо рекурсии можно воспользоваться вложенными циклами.

4. Условный оператор

В подавляющем большинстве задач не удается представить алгоритм в виде линейной структуры, т. к. задачи обычно содержат различные условия, требующие выбора одного из возможных направлений продолжения программы.

Условный переход реализуются с помощью оператора условия (условного оператора), который бывает двух типов: линейный и блочный.

Линейная форма условного оператора записывается в следующем формате:

IF условие **THEN** Выражение_1 **ELSE** Выражение_2

Линейной форма оператора называется потому, что условный оператор записывается только в одной строке, без переноса его на другую строку.

Ключевое слово **ELSE** и следующее за ним Выражение_2 могут отсутствовать. Это так называемая усеченная форма оператора.

Пример 1. Необходимо вычислить значение функции $y = \frac{1}{a \cdot b}$ при любых значениях a и b с проверкой ОДЗ.

IF a*b=0 THEN MsgBox (“ab=0”) ELSE y=1/ (a*b): MsgBox (“y=”; &y)

Блочная форма условного оператора применяется, когда в случае истинности условия необходимо выполнить несколько программных операторов (блок операторов). Существует несколько модификаций блочного оператора IF. Рассмотрим безальтернативную (усеченную) модификацию.

Формат оператора:

IF условие **THEN**

Программный оператор 1

…

Программный оператор n

END IF

Альтернативный блочный оператор **If** применяется в тех случаях, когда при выполнении условия необходимо выполнить один набор программных операторов, а при невыполнении — другой. Формат такого оператора:

IF условие **THEN**

Блок программных операторов, выполняемых при соблюдении условия

ELSE

Блок программных операторов, выполняемых при несоблюдении условия

END IF

Вариант программы с блочным оператором **If** можно записать в следующем виде:

```
IF a*b=0 THEN
  MsgBox ("ab=0")
ELSE
  y=1/(a*b)
      MsgBox ("y="; & y)
END IF
```

В том случае, если необходимо рассмотреть еще несколько условий в дополнение к исходному, применяется полная форма условного оператора. Формат оператора:

IF условие_1 **THEN**

Блок программных операторов, выполняемых при соблюдении условия_1

ELSEIF условие_2 **THEN**

Блок программных операторов, выполняемых при соблюдении условия_2

…

ELSEIF условие_n **THEN**

Блок программных операторов, выполняемых при соблюдении условия_n

ELSE

Блок программных операторов, выполняемых при несоблюдении условий

END IF

Пример 2. Вычислить

$$y = \begin{cases} \sin x & \text{при} \quad x < 0 \\ 1 & \text{при} \quad x = 0 \\ \cos x & \text{при} \quad x > 0 \end{cases}$$

Фрагмент программы, реализующий ветвление, может быть представлен в виде:

```
IF x<0 THEN
  y = sin(x)
  ELSEIF   x=0  THEN
      y = 1
      ELSE
```

y = cos(x)

END IF

MsgBox (" При x =" & x & " y =" & y)

Оператор передачи управления — эти операторы применяются в программе для реализации безусловных алгоритмических конструкций. Они выполняют переход с одного участка программы на любой другой без какого-либо условия. Оператор передачи управления (оператор безусловного перехода) имеет следующий формат:

GOTO идентификатор, где **GOTO** — ключевое слово, идентификатор — одна из меток программы. Метка может быть номером строки или комбинацией букв и цифр. Метка ставится в начале строки и отделяется от последующих операторов двоеточием, например m1: y=cos(x)

Тогда оператор перехода будет иметь вид: **GOTO** m1.

Следует избегать применения оператора **GOTO**, так как в этом случае программа плохо читается и ухудшается ее понимание.

Оператор выбора Select Case

Если нужно проверить несколько условий, применяется оператор выбора Select Case, который улучшает читаемость программы.

Оператор выбора имеет следующий вид:

```
Select Case переменная
   Case значение1
         Блок операторов1
   Case значение2
         Блок операторов2
   ……————————
   Case Else
         Блок операторов
End Select
```

5. Операторы цикла

Цикл — это алгоритм, в котором предусмотрено неоднократное выполнение одной и той же последовательности действий.

5.1. Счетный цикл

Формат оператора:

FOR x = xНач **TO** xКон **STEP** hx ‘Заголовок цикла

Тело цикла

NEXT x

Здесь x — параметр цикла (или управляющая переменная цикла, или счетчик), т. е. переменная, значение которой при каждом выполнении цикла изменяется по вполне определенному закону;

xНач, xКон — начальное и конечное значения параметра цикла;

hx — шаг изменения параметра цикла. Если шаг равен 1, то ключевое слово STEP и hx могут не указываться.

Пример 3. Составить программу вычисления значения функции $y = \sin x_i$ при $x_i = 0; 0{,}1; 0{,}2; \ldots 1$.

```
Sub PRC1()
DIM x AS SINGLE, y AS SINGLE, i As Integer
        ‘x — параметр цикла (управляющая переменная цикла)
        ‘i — счетчик номера строк таблицы Excel
i = 1 ‘ вывод с первой строки
FOR x=0 TO 1 STEP 0.1
y = SIN(x)
Cells(i, 1) = x: Cells(i, 2) = y: i = i + 1
NEXT x
End Sub
```

Пример 4. Составить программу суммы M натуральных чисел

```
Sub Summa ()
DIM n, M As Integer
M=InputBox(“Введите M”)
FOR n= 1 TO M STEP 1
S:=S+n
NEXT n
MsgBox “Сумма равна ”&S
End Sub
```

5.2. Циклы с неизвестным числом повторений

Оператор цикла WHILE

Формат оператора:

```
WHILE   условие
   Тело цикла
WEND
```

Здесь WHILE — ключевое слово (пока, выполняется), заголовок цикла;

WEND — ключевое слово (идти, возвращаться), указывает на окончание циклической конструкции;

Условие — логическое выражение, определяющее условие выполнения цикла;

Тело цикла — последовательность операторов (или строк программного кода), число которых не ограничено. Цикл WHILE выполняется, пока истинно (выполняется) условие.

Пример 5. Составить программу вычисления значения функции $y = \sin x_i$ при $x_i = 0; 0{,}1; 0{,}2; \ldots 1$.

```
Sub PRC2 ()
CONST xMin = 0, xMax = 1, hx = 0.1
DIM x AS SINGLE, y AS SINGLE
x = xMin
WHILE  x<= xMax
    y = SIN(x)
    MsgBox (" При x =" & x & "  y =" & y)
    x = x + hx      ‘Изменение параметра цикла (управляющей   перемен-
ной)
WEND
End Sub
```

Пример 6. Вычислить наибольшее положительное целое число n, удовлетворяющее условию

$3n^5$–690n <= 7.

Решить поставленную задачу с помощью счетного цикла невозможно, поскольку неизвестно число повторений цикла. В то же время задача решается очень просто с помощью цикла WHILE:

```
Sub PRC3 ()
DIM N As Integer
N=1  ‘ Начальное значение переменной N (параметр цикла)
        WHILE   3*N^5 — 690*N<= 7
                N=N+1  ’ Следующее значение переменной N
        WEND
MsgBox “Наибольшее целое N=” & N-1
End Sub
```

В списке вывода в качестве наибольшего целого указана величина N-1. Почему это так, выяснить самостоятельно.

Цикл WHILE иногда называют циклом с условием. В последних версиях Бейсика (в том числе и VBA) разработаны и более совершенные конструкции циклов с условием. Это циклы DO – LOOP.

5.3. Оператор цикла DO

Основное отличие цикла DO от цикла WHILE в том, что в цикле DO условие выполнения цикла может располагаться или в начале (заголовке) цикла (как в цикле WHILE), или в конце цикла. Поэтому циклы DO разделяются на циклы с *предусловием* и циклы с *постусловием*.

Кроме того, в цикле DO условие записывается с использованием ключевых слов WHILE (пока) или UNTIL (до). Таким образом, возможны четыре формы цикла DO, что существенно расширяет возможности программиста.

```
Формат  1 оператора:
DO WHILE условие                ‘цикл с предусловием
  Тело цикла
LOOP                            ‘(петля, делать петлю)
Формат  2 оператора:
DO
  Тело цикла
LOOP   WHILE условие            ‘цикл с постусловием
```

Цикл DO WHILE выполняется до тех пор, пока истинно (выполняется) условие.

```
Формат  3 оператора:
DO UNTIL условие                ‘цикл с предусловием
  Тело цикла
LOOP
Формат  4 оператора:
DO
  Тело цикла
LOOP   UNTIL условие            ‘цикл с постусловием
```

Цикл DO UNTIL выполняется до тех пор, пока ложно (не выполняется) условие. Отличие цикла с постусловием от цикла с предусловием в том, что тело цикла с постусловием всегда выполнится хотя бы один раз.

5.4. Вложенные циклы

В теле цикла могут быть размещены один или несколько других циклов, называемых вложенными. Цикл, в котором создается другой цикл, называется внешним, а цикл, размещаемый внутри,— внутренним. Число вложений циклов не ограничено. Вложенные циклы могут состоять как из операторов одного типа, так и из операторов любого другого типа в любой последовательности. Параметры (управляющие переменные) во вложенных циклах изменяются неодновременно, порядок их изменения определяется условиями конкретной задачи.

Контрольные вопросы

1. Что такое оператор?
2. Как осуществить ввод данных в программе на VBA?
3. Как осуществить вывод данных в программе на VBA?
4. Какие типы модулей используются в программах VBA?
5. Какие типы процедур используются в программах VBA?
6. В чем различие в создании процедур sub и function?
7. В каких случаях применяется оператор if?
8. В каких случаях применяется оператор select case?
9. В каких случаях предпочтительнее использовать операторы повтора while и do? В чем их различие?
10. В каких случаях предпочтительнее использование для организации циклов оператор повтора for?

Лекция 10. Создание форм рабочего листа

План:

1. Объекты, используемые для создания форм рабочего листа
2. Иерархия объектов. Уровни ссылок.
3. Единичные объекты и объекты из семейств.

1. Объекты, используемые для создания форм рабочего листа

Форма — это главный объект, образующий визуальную основу приложения. По своей сути форма представляет собой окно, в котором можно размещать различные управляющие элементы при создании приложений.

Microsoft Excel может генерировать встроенную форму данных (Форма данных. Диалоговое окно, поочередно отображающее полные записи. В форме данных можно добавлять, изменять, удалять записи и проводить их поиск) для списка. Форма данных отображает все заголовки столбцов списка в одном диалоговом окне, с пустым полем рядом с каждым заголовком, предназначенным для ввода данных в столбец. При этом можно ввести новые данные, найти строки на основе содержимого ячейки, обновить имеющиеся данные и удалить строки из списка.

Форма данных может облегчить ввод данных, например когда имеется широкий список, количество столбцов которого превышает число столбцов, которое может одновременно отображаться на экране.

Если требуется создать сложную или специализированную форму для ввода данных, следует создать лист или шаблон для использования его в качестве формы и затем настроить лист формы в соответствии с необходимыми требованиями.

Шаблон — книга, создаваемая и используемая как начальный вариант всех новых книг. Можно создавать шаблоны книг и листов. Используемый по умолчанию шаблон книги называется Книга.xlt. Используемый по умолчанию шаблон листа называется Лист.xlt.

Например, можно создать форму отчета о расходах, которая будет заполняться в электронном виде или в виде печатных копий. Этот способ используется в случае, если для настройки форм требуется максимальная гибкость. Формы на листе особенно удобны, если требуется получить отдельные печатные копии формы.

Для сохранения данных из форм в списке Microsoft Excel можно разработать приложение для ввода данных с помощью редактора Microsoft Visual Basic.

Создание диалогового окна. В Microsoft Excel можно создать диалоговое окно двумя путями.

1) Создание формы на рабочем листе и использование элементов управления.

Элемент управления — объект графического интерфейса пользователя (такой как поле, флажок, полоса прокрутки или кнопка), позволяющий пользователям управлять приложением для того, чтобы форма выглядела и работала как диалоговое окно. Элементы управления используются для отображения данных или параметров, для выполнения действий, либо для упрощения работы с интерфейсом пользователя. В Microsoft Excel нет специальных инструментов для редактирования диалоговых окон.

2) Создание диалогового окна в Microsoft Visual Basic для приложений (VBA).

Создание электронных форм. Microsoft Excel хранит информацию в ячейках рабочего листа. Однако не всегда удобно вводить данные прямо в ячейки, Вы можете упростить этот процесс, автоматизировав ввод часто встречающихся данных или предлагая выбрать значение из списка. Поля, вычисляемые по достаточно сложному алгоритму, имеет смысл рассчитывать с помощью процедуры, запускаемой нажатием определенной кнопки. Все эти возможности реализуются через элементы управления, совокупность которых образует электронную форму. В Excel существуют две встроенные панели инструментов, которые предназначены для создания электронных форм: Forms (Формы) и Control Toolbox (Элементы управления). Кроме того, вы можете создать пользовательскую форму средствами встроенного редактора Visual Basic.

Формы в приложениях VBA, применение форм. С самыми простыми возможностями организации взаимодействия с пользователем (применение встроенных функций MsgBox() и InputBox()).

Существует несколько типов диалоговых окон, которые необходимы для поддержания в программе интерактивного режима работы конечного пользова-

теля (вывод сообщений пользователю, прием и интерпретация указаний, введенных пользователем, и др.).

Диалоговые окна для обмена сообщениями

Встроенная функция MsgBox() обеспечивает создание диалоговых окон различных типов.

1. Простое окно-сообщение

MsgBox ("строка_сообщения")

Если в сообщении должно присутствовать значение переменной или элемента массива переменной, элемент структуры пользовательского типа данных и т.п., следует преобразовать значения в строковый и обеспечить конкатенацию строк.

2. Окно-сообщение с командными кнопками

Общий формат оператора:

MsgBox("строка_сообщение"[, <кнопки>] [, "заголовок_окна" [, <файл-подсказки>, <контекст>])

где <строка_сообщение> — максимальная длина строки — 1024 символа;

<кнопки> — число, являющееся суммой кодов выбранных типов кнопок и пиктограммы, или имена кнопок;

<заголовок_окна> — строка символов;

<файл-подсказки> — имя файла-подсказки для контекстно-зависимой помощи при работе в окне;

<контекст> — число, которое назначено подсказке дня данного окна.

Коды задания командных кнопок и пиктограмм в функции MsgBox() приведены в таблице 18. Функция MsgBox() возвращает код (число), соответствующий нажатой кнопке.

Таблица 18 — Список видов командных кнопок и их кодов

Код	Константа	Описание
		Коды командных кнопок
0	vbOKOnly	OK
1	vbOKCancel	OK, Отмена
2	vbAbortRetryIgnore	Прекратить, Повторить, Игнорировать
3	vbYesNoCansel	Да, Нет, Отмена
4	vbYesNo	Да, Нет
5	vbRetryCansel	Повторить, Отмена
		Коды активности по умолчанию
0	vbDefaultButton 1	Активная 1
256	vbDefaultButton 2	Активная 2
512	vbDefaultButton 3	Активная 3

Продолжение табл. 18

Код	Константа	Описание
		Коды пиктограмм
16	vbCriticalvbQuestion	Важное сообщение
32	vbExclamation	Предупредительный запрос
48	vbInformation	Предупредительное сообщение
64		Информационное сообщение
		Коды модальности
0	vbApplicationModal	Программное модальное описание (требуется обязательный ответ, работа приложения приостанавливается)
4096	vbSystemModal	Системное модальное описание (требуется обязательный ответ, работа всех приложений приостанавливается)

Объект панели сообщений	Код	Пиктограмма
vbCritical	16	
vbQuestion	32	
vbExclamation	48	
vbInformation	64	

Рисунок 10 — Коды и пиктограммы командных кнопок

Диалоговые окна для ввода данных

Функция InputBox() обеспечивает формирование окна для вывода строки сообщения и ожидания ввода строки символов или нажатия кнопки. Возвращает содержание текстового блока:

InputBox(<строка_сообщение> [,<заголовок_окна>] [,<текст_по_умолчанию>] [,Хпоз][,Yпоз][, <файл-подсказки>, <контекст>])

где <текст_по_умолчанию> — строка символов, выводимая в текстовом блоке (может быть пустой);

Хпоз, Yпоз — позиция левого верхнего угла окна.

Диалоговые окна пользователей

Диалоговые окна пользователей обеспечивают ввод и редактирование данных файлов и таблиц. Для создания диалогового окна выполняется команда ВСТАВКА, Макрос с указанием типа вставляемого листа — Диалог. Экран содержит основу для построения диалогового окна пользователя, а также панель инструментов *Формы* для размещения и настройки (форматирования) элементов окна.

В диалоговом окне могут размещаться элементы произвольного вида, которые создают наборы объектов. Доступ к конкретному элементу — по имени набора и имени или порядковому номеру элемента внутри набора.

Элементы управления — это специализированные объекты, которые можно размещать на формах VBA (и непосредственно в документах), используемые

Элементы управления — это специализированные объекты, которые можно размещать на формах VBA (и непосредственно в документах), используемые для организации взаимодействия с пользователем. В VBA вы можете использовать как стандартные элементы управления (CommandButton, CheckBox, OptionButton), так и нестандартные (любые другие, которые есть на вашем компьютере, например Internet Explorer, Calendar и т. п.). Элементы управления реагируют на события, которые генерирует пользователь (нажатие на кнопку, ввод значения, перемещение ползунка и т. п.).

Элементы, расположенные на панели инструментов Control Toolbox (Элементы управления), называются элементами Active X. Они несколько отличаются от элементов управления, расположенных на панели Forms (Формы). Скорее они ближе к элементам управления Visual Basic, так как при добавлении объекта Active X на рабочий лист создается макрос, который сохраняется вместе с этим элементом, а не только запускается при его выборе. При копировании или перемещении такого объекта автоматически будут скопированы или перемещены все процедуры, связанные с ним.

Добавление элементов управления можно производить и программным способом (при помощи метода Add() коллекции Controls), однако вам при этом придется указывать в коде программы большое количество свойств создаваемого элемента управления, что не очень удобно.

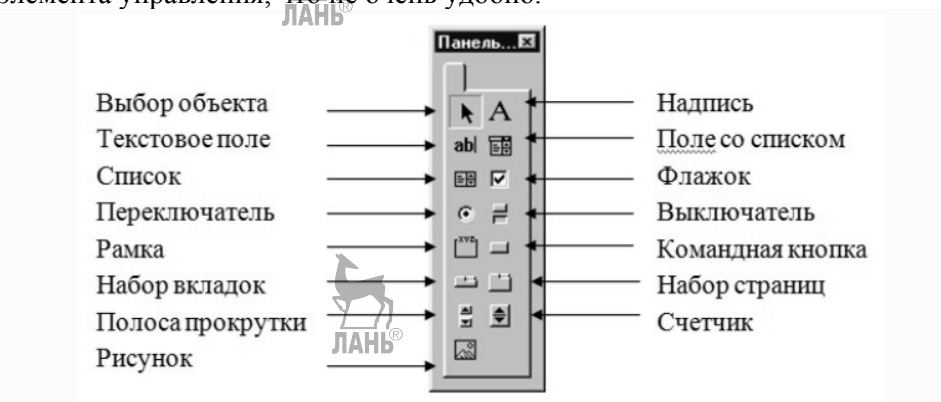


Рисунок 11 — Панель Control Toolbox

Для редактирования элемента Active X необходимо войти в Design Mode (Режим конструктора), выбрав соответствующий инструмент на панели Control Toolbox (Элементы управления).

Чтобы сделать элемент активным, режим конструктора следует отключить. Выбрав кнопку Properties (Свойства) на этой панели инструментов, вы сможете задать свойства элемента Active X, инструмент ***(Исходный текст)*** позволяет просмотреть модуль Visual Basic, содержащий текст макроса, связанного с элементом.

2. Иерархия объектов. Уровни ссылок

Объектная модель Excel. В объектной модели Excel и других приложений Office объекты связаны между собой отношением встраивания. На нулевом

уровне иерархии существует некоторый центральный объект, в который встроены другие объекты, составляющие первый уровень иерархии. В каждый из объектов первого и последующих уровней могут быть встроены объекты следующего уровня. Формально встраивание реализуется с помощью свойств объектов. Свойства могут быть как терминальными, не являющимися объектами, и так называемыми свойствами-участниками, которые возвращают объекты при их вызове.

Иерархия объектов. Объектная библиотека VBA располагает более 100 различных объектов, находящихся на различных уровнях иерархии. Иерархия определяет связь между объектами и показывает пути доступа к ним. На рисунке 12 приведена модель встроенных объектов VBA.

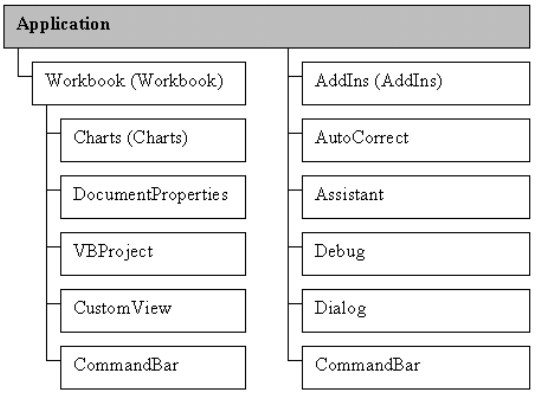


Рисунок 12 — Иерархия встроенных объектов VBA

Полная ссылка на объект состоит из ряда имен, вложенных последовательно друг в друга объектов. Разделителями имен объектов в этом ряду являются точки, ряд начинается с объекта Application и заканчивается именем самого объекта. Например, полная ссылка на ячейку A1 рабочего листа Лист1 рабочей книги с именем Архив имеет вид:

Application.Workbooks("Архив")

.Worksheets("Лист1").Range("A1")

Приводить каждый раз полную ссылку на объект совершенно не обязательно. Обычно достаточно ограничиться только неявной ссылкой на объект.

В неявной ссылке, в отличие от полной, объекты, которые активны в данный момент, как правило, можно опускать. В рассмотренном случае, если ссылка на ячейку A1 дана в программе, выполняемой в среде Excel, то ссылка на объект Application может быть опущена, т. е. достаточно привести относительную ссылку:

Workbooks("Архив").Worksheets("Лист1").Range("A1")

Если рабочая книга Архив является активной, то ссылку можно записать в виде Worksheets("Лист1").Range("A1").

Если и рабочий лист Лист1 активен, то в относительной ссылке вполне достаточно ограничиться упоминанием только диапазона A1: Range("A1").

Объект сам по себе не представляет большого значения. Намного значительнее то, какие действия можно совершать над объектом, и какими свойствами он обладает. *Метод* представляет собой действие, выполняемое над объектом.

Синтаксис применения метода: **Объект.Метод**

Метод можно применять ко всем объектам семейства.

Свойство представляет собой атрибут объекта, определяющий его характеристики, такие как размер, цвет, положение на экране и состояние объекта, например доступность или видимость. Чтобы изменить характеристики объекта, надо просто изменить значения его свойств.

Синтаксис установки значения свойства:

Объект.Свойство = ЗначениеСвойства

Свойство можно изменять сразу у всех объектов семейства. Приведем наиболее часто употребляемые подобные свойства.

Таблица 19 — Свойства объектной модели Excel

Свойства	Значение
ActiveWindow	Возвращает активное окно Excel
ActiveWorkbook	Возвращает активную рабочую книгу активного окна Excel
ActiveSheet	Возвращает активный лист активной рабочей книги
ActiveDialog	Возвращает активное диалоговое окно активного рабочего листа
ActiveChart	Возвращает активную диаграмму активного рабочего листа
ActiveCell	Возвращает активную ячейку активного рабочего листа

Событие представляет собой действие, распознаваемое объектом (например, щелчок мышью или нажатие клавиши), для которого можно запрограммировать отклик. События возникают в результате действий пользователя или программы, или же они могут быть вызваны системой.

Суть программирования на VBA как раз и заключается в этих двух понятиях: *событие* и *отклик* на него. Если пользователь производит какое-то воздействие на систему, скажем нажимает кнопку, тогда в качестве отклика выполняется код созданной пользователем процедуры. Если такой отклик не создан, т. е. не написана соответствующая процедура, то система никак не реагирует на данное событие и оно остается безответным. Таким образом, действия, происходящие в системе, являются событиями, а отклики на них — процедурами. Этот специальный вид процедур, генерирующих отклик на события, называется *процедурами обработки событий.* В целом программирование на VBA состоит в создании кода программ, которые генерируют прямо или косвенно отклики на события.

3. Единичные объекты и объекты из семейств

Объектная модель Microsoft Excel насчитывает множество объектов. Для разработки большинства программ достаточно знать основные из них, такие как Application, Workbook(s), Worksheet(s), Range, Chart.

Объектом VBA считается некоторый элемент, который можно отобразить в окне приложения и, главное, на который можно *воздействовать* некоторым образом, *изменяя его состояние*.

Например, диапазон ячеек рабочего листа можно увидеть в окне, и можно изменить его состояние, введя в ячейки этого диапазона данные, сменив цвет ячеек, используемый шрифт или иные характеристики. Таким образом, диапазон ячеек — это объект. Существуют элементы окна приложения, не являющиеся объектами. Например, кнопки *Свернуть окно* и *Развернуть окно* не являются объектами. Вы можете пользоваться этими кнопками, но не можете изменить их. Напротив, само окно рабочей книги является объектом, поскольку оно может быть свернуто или развернуто с помощью этих кнопок.

Изменить состояние объекта в VBA можно одним из двух способов:

- изменив одно из свойств (Properties) объекта;
- выполнить некоторые действия, применив один из методов (Methods), ассоциированных с этим объектом.

Различают объекты и их семейства.

Все объекты Excel можно разделить на два класса:

- единичные объекты. На единичные объекты ссылаются непосредственно по имени;
- объекты, принадлежащие семействам. На объекты в семействах ссылаются по индексу в семействе.

Например, объект Application является единичным, так как Excel имеет только один объект Application. Объект Worksheet — это объект семейства. Каждая рабочая книга может иметь несколько рабочих листов.

Индексы объектов в семействах в объектной модели Excel всегда начинаются с 1. Если номер объекта в семействе не известен, к нему можно обратиться по имени. Каждый объект Worksheet в семействе Worksheets имеет связанное с ним имя; эти имена выводятся на ярлычках рабочего листа в нижней части окна Excel. Предположим, нужно установить свойство Visible рабочего листа. Это свойство может принимать значения True, False. Если рабочий лист имеет имя «Пример», даже не зная, какой номер у этого конкретного объекта Worksheet в семействе Worksheets, можно обратиться по имени, а не по номеру.

Worksheets(«Пример»). Visible= False

Семейство (объект **Collection**) представляет собой объект, содержащий в себе несколько других объектов одного типа. Все объекты Excel и их семейства имеют родовые иерархические отношения.

Главным в иерархии объектов Excel является объект Application (Приложение), которое представляет само приложение Excel. Этот объект имеет более 120 свойств и 40 методов, которые предназначены для установки параметров приложения Excel. Кроме того, объект Application позволяет вызывать более 400 встроенных функций рабочего листа при помощи конструкции вида: **Application.ФункцияРабочегоЛиста(Аргументы)**

Например,

Application.Pi() ‘Вычисление числа пи.

Application.Sum(Аргументы)

Подчиненными объектами в иерархии объектов являются: объекты семейств WorkBooks (рабочие книги), Worksheets (рабочие листы), Range (Диапазон).

Формат программного кода, задающего установку свойства и использование метода объекта:

Объект. Свойство = ЗначениеСвойства

Объект. Метод [Параметр1 […]]

Если X является свойством-участником объекта Application, то обращение к этому свойству возвращает ссылку на объект X.

Обращение **Application.X.Y.Z**, где X, Y и Z — свойства-участники, позволяет добраться до объекта Z, находящегося на третьем уровне вложенности.

Обычно цепочка именования начинается спецификатором (объектом) Application, но иногда его можно опустить. Некоторые свойства и методы объекта Apiplication относятся к глобальным. Для них спецификатор Application разрешается опускать, непосредственно именуя глобальный элемент. Вот пример нескольких обращений к элементам объекта Application:

Application. ActiveDocument или

ActiveDocument

ActiveSheet

Можно указать и полный путь: *Application. ActiveSheet*

Таблица 20 — Свойства объекта Application

Свойства	**Описание и допустимые значения**
ActiveWorkbook, ActiveSheet, ActiveCell, ActiveChart	Возвращают активный объект: рабочую книгу, лист, ячейку, диаграмму. Свойство ActiveCell содержится в ActiveSheet, а свойства ActiveSheet и ActiveChart — в ActiveWorkbook
ThisWorkbook	Возвращает рабочую книгу, содержащую выполняющийся в данный момент макрос
Calculation	Устанавливает режим вычислений. Возможные значения: xlCalculationAutomatic (автоматический режим), xlCalculationManual (вручную)
Caption	Возвращает текст в строке имени активного окна
DisplayFormulaBar	True (False) — строка формул выводится (не выводится) на экран
DisplayScrolBars	True (False) — полосы прокрутки видны (не видны) в окне Excel
ScreenUpdating	Если равно True, то изображение на экране обновляется во время выполнения программы, если False — то нет
DisplayStatusBar	True (False) — строка состояний видна (не видна) в окне Excel

Таблица 21 — Основные методы объекта Application

Методы	Действия
Calculate	Вызывает принудительные вычисления во всех открытых рабочих книгах
Run	Запускает на выполнение подпрограмму или макрос: Run (ИмяМакроса, Аргументы)
Volatile	Вызывает перевычисление функции пользователя при изменении значений параметров. Оператор Application.Volatile нужно поместить в теле функции
Wait	Временно приостанавливает работу приложения: Wait (Time)
OnTime	Назначает выполнение процедуры на определенное время: OnTime(ВремяЗапуска, ИмяПоцедуры, …)
Quit	Закрывает приложение

Таблица 22 — События объекта Application

Событие	Когда происходит
NewWorkBook	При создании новой рабочей книги
WorkbookActivate	При активизации рабочей книги
WorkbookBeforeClose	Перед закрытием рабочей книги
WorkbookBeforeSave	Перед сохранением рабочей книги
WorkbookDeactivate	Когда рабочая книга теряет фокус
WorkbookNewSheet	При добавлении нового листа в рабочую книгу
WorkbookOpen	При открытии рабочей книги

Объект **Workbook** — это файл рабочей книги. Получить объект Workbook можно, используя свойства Workbooks, ActiveWorkbook или ThisWorkbook объекта Application.

Таблица 23 — Основные свойства объектов семейства Workbooks

Свойства	Описание и допустимые значения
ActiveSheet	Возвращает активный лист книги
ActiveChart	Возвращает активную диаграмму
Count	Возвращает количество объектов семейства
WriteReserved	True (False) — документ закрыт (открыт) для записи
Sheets, Worksheets, Charts	Возвращают семейства всех рабочих листов книги и всех диаграмм соответственно

Таблица 24 — Основные методы объектов семейства Workbooks

Методы	Действия
Activate	Активизирует рабочую книгу (первый лист становится активным).
Add	Создает новую рабочую книгу
Close, Open, Open-Text	Закрытие (открытие) рабочей книги, открытие текстового файла с таблицей данных. Например, рабочая книга закрывается без сохранения: Workbooks("Book1.xls").Close SaveChanges:= False Открытие рабочей книги: Workbooks.Open "Book1.xls"
Save, SaveAs	Сохранение рабочей книги (сохранение в другом файле). Например, запросить у пользователя имя файла и сохранить активную рабочую книгу можно кодом программы: fName= Application.GetSaveAsFilename ActiveWorkbook.SaveAs Filename:=fName

Таблица 25 — События объектов семейства Workbooks

Событие	Когда происходит
BeforeClose	Перед закрытием рабочей книги
BeforeSave	Перед сохранением рабочей книги
Deactivate	Когда рабочая книга теряет фокус
NewSheet	При добавлении нового листа в рабочую книгу
Open	При открытии рабочей книги
SheetActivate	При активизации рабочего листа
SheetDeactivate	Когда рабочий лист теряет фокус

Контрольные вопросы

1. Какими основными понятиями оперирует язык VBA?
2. Что такое объект?
3. Что такое метод?
4. Что такое свойство?
5. Как создаются объекты VBA?
6. Перечислите основные объекты MS Excel.

Лекция 11. Работа с элементами управления

План:

1. Размещение элементов управления на рабочем листе. Общие свойства и события элементов управления.
2. Элементы управления

1. Размещение элементов управления на рабочем листе. Общие свойства и события элементов управления

Элементы управления — это специализированные объекты, которые можно размещать на формах VBA (и непосредственно в документах), используемые для организации взаимодействия с пользователем. В VBA вы можете использо-

вать как стандартные элементы управления (CommandButton, CheckBox, OptionButton), так и нестандартные (любые другие, которые есть на вашем компьютере, например, Internet Explorer, Calendar и т. п.). Элементы управления реагируют на события, которые генерирует пользователь (нажатие на кнопку, ввод значения, перемещение ползунка и т. п.).

Добавление элементов управления на форму чаще всего производится из дизайнера форм при помощи Toolbox. Для этого необходимо выбрать элемент управления в Toolbox и перетащить его на форму или выделить элемент управления в Toolbox и затем на форме выделить ту область экрана, которую будет занимать этот элемент управления.

Добавление элементов управления можно производить и программным способом (при помощи метода Add() коллекции Controls), однако вам при этом придется указывать в коде программы большое количество свойств создаваемого элемента управления, что не очень удобно.

Некоторые элементы управления, имеющиеся на панели инструментов Control Toolbox (Элементы управления), отсутствуют на панели Forms (Формы):

Text Box (Поле) — поле, в которое можно ввести текст.

Toggle Button (Выключатель) — кнопка, которая остается нажатой, если вы ее выбираете. Чтобы выключить эту кнопку, необходимо нажать ее еще раз.

Image (Рисунок) — элемент управления, который позволяет внедрить рисунок в форму.

More Controls (Дополнительные элементы) — список дополнительных элементов Active X.

Для редактирования элемента Active X необходимо войти в ***Design Mode (Режим конструктора)***, выбрав соответствующий инструмент на панели ***Control Toolbox (Элементы управления)***. Чтобы сделать элемент активным, режим конструктора следует отключить. Выбрав кнопку ***Properties (Свойства)*** на этой панели инструментов, вы сможете задать свойства элемента Active X, инструмент ***(Исходный текст)*** позволяет просмотреть модуль Visual Basic, содержащий текст макроса, связанного с элементом.

2. Элементы управления

Элемент управления Label, свойство Caption

Надпись — это просто область формы, в которой выводится какой-то текст (рис. 13).

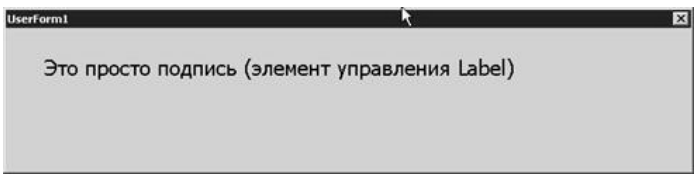


Рисунок 13 — Элемент управления Label на форме

Label используется как строка состояния с объяснением того, что сейчас произошло/происходит/должен сделать пользователь и т. п. Самое главное свойство элемента управления Label — это Caption, тот текст, который будет выводиться на форме. Большая часть остальных свойств относится к форматированию этого текста или настройке внешнего вида этого элемента управления.

Несмотря на то что для этого элемента управления предусмотрен набор событий (Click, Error и т. п.), использовать их не принято: пользователю обычно не приходит в голову, что по надписи нужно щелкать мышью.

Элемент управления TextBox, свойство Value (Text)

Текстовое поле (TextBox) — один из самых часто используемых элементов управления (рис. 14).

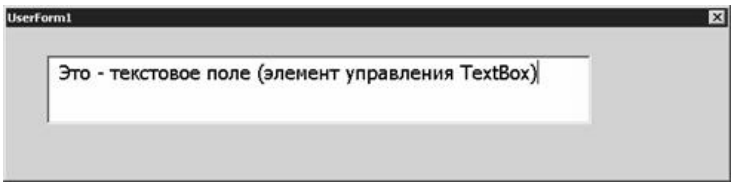


Рисунок 14 — Текстовое поле (элемент управления TextBox) на форме

Текстовое поле используется:

- для приема каких-либо текстовых данных, вводимых пользователем;
- для вывода пользователю текстовых данных с возможностью их редактирования;
- для вывода пользователю текстовых данных с возможностью копирования и печати, но без возможности изменения (классический пример — текст лицензионного соглашения).

Некоторые важные свойства этого элемента управления:

Value (или Text) — то текстовое значение, которое содержится в этом поле. Используется для занесения исходного значения и для приема значения, введенного пользователем, в строковую переменную;

AutoSize — возможность для текстового поля автоматически менять свой размер, чтобы вместить весь текст. Использовать не рекомендуется, так как может нарушиться весь дизайн вашей формы;

ControlSource — ссылка на источник текстовых данных для поля. Может ссылаться, например, на ячейку в Excel, на поле в Recordset и т. п.;

ControlTipText — текст всплывающей подсказки, которая появляется, когда пользователь наводит указатель мыши на элемент управления;

Enabled — если переставить в False, то текст в поле станет серым и с содержимым поля ничего сделать будет нельзя (ни ввести текст, ни выделить, ни удалить). Обычно это свойство используется (для всех элементов управления), чтобы показать пользователю, что этот элемент управления отключен до выполнения каких-либо условий;

Locked — поле будет выглядеть как обычно, пользователь сможет выделять и копировать данные из него, но не изменять их. Обычно используется для показа неизменяемых данных типа лицензионных соглашений, сгенерированных значений и т. п.;

MaxLength — максимальная длина значения, которое можно ввести в поле. Иногда можно использовать свойство AutoTab — при достижении определенного количества символов управление автоматически передается другому элементу управления;

MultiLine — можно ли использовать в текстовом поле несколько строк или необходимо обойтись одной;

PasswordChar — указать, за каким символом будут «прятаться» вводимые пользователем значения.

ScrollBars — будут ли показаны горизонтальная и вертикальная полосы прокрутки (в любом сочетании). Если текст может быть большим, без них не обойтись.

WordWrap — настоятельно рекомендуется включать в тех ситуациях, когда значение MultiLine стоит в True. В этом случае будет производиться автоматический переход на новую строку при достижении границы текстового поля.

Остальные свойства по большей части относятся к оформлению текстового поля и его содержания, а также настройкам редактирования.

Главное событие для текстового поля — это событие Change (то есть изменение содержания поля). Обычно на это событие привязывается проверка вводимых пользователем значений или синхронизация введенного значения с другими элементами управления (например, сделать доступной кнопку, изменить текст надписи и т. п.).

Элемент управления ComboBox, метод AddItem(), свойство Value

Комбинированный список также используется очень часто. Этот элемент управления позволяет пользователю как выбирать «готовые» значения из списка, так и вводить значения самостоятельно.

Рисунок 15 — Комбинированный список (элемент управления ComboBox)

Обычно он помещается в обработчик события Initialize для формы. Применение его может выглядеть так:

```
Private Sub UserForm_Initialize()
ComboBox1.AddItem "Санкт-Петербург"
ComboBox1.AddItem "Ленинградская область"
ComboBox1.AddItem "Москва"
```

```
ComboBox1.AddItem "Московская область"
End Sub
```

Параметр varIndex (необязательный) может использоваться для определения положения элемента в списке, но он не может превышать значения ListCount и поэтому для начальной загрузки ComboBox не подходит.

Самые важные свойства комбинированного списка:

ColumnCount, ColumnWidth, BoundColumn, ColumnHeads, RowSource — свойства, которые применяются при работе со списками из нескольких столбцов. Пользователями нелюбимы и к использованию не рекомендуются (гораздо проще сделать несколько комбинированных списков);

MatchEntry — будут ли при вводе пользователем первых символов значения выбраться подходящие позиции из списка;

MatchRequired — разрешается ли пользователю вводить то значение, которого нет в списке. По умолчанию False, то есть разрешено;

Value (или Text) — позволяет программным способом установить выбранное значение в списке или вернуть выбранное/введенное пользователем значение.

Остальные свойства (AutoSize, Enabled, Locked, ControlText, ControlTipText, MaxLength) — применяются точно так же, как и для TextBox.

Главное событие для комбинированного списка — Change, то же, что и для TextBox.

Элемент управления ListBox на формах VBA, его применение

Этот элемент управления очень похож на комбинированный список, но применяется гораздо реже по двум причинам:

- в нем нельзя открыть список значений по ниспадающей кнопке. Все значения они видны сразу в поле, аналогичном текстовому, и поэтому большое количество позиций в нем уместить трудно;
- пользователь не может вводить свои значения — только выбирать из готовых.

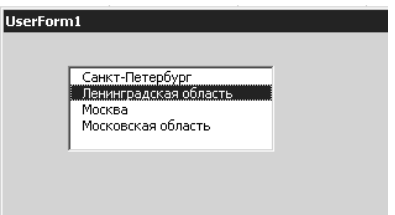


Рисунок 16 — Список (элемент управления ListBox)

Но у этого элемента управления есть и преимущества: в нем пользователь может выбирать не одно значение, как в ComboBox, а несколько.

Обычно ListBox используется:

- как промежуточное средство отображения введенных/выбранных пользователем через ComboBox значений (или любых других списков, например списков выбранных файлов);

• как средство редактирования списка значений, сформированных вышеуказанным образом или полученных из базы данных (для этого можно рядом с ListBox разместить кнопки Удалить или Изменить);

• основные свойства, методы и события у ListBox — те же, что и у ComboBox. Главное отличие — то, что имеется свойство MultiSelect, которое позволяет пользователю выбирать несколько значений. По умолчанию это свойство отключено.

Элементы управления CheckBox (флажок), ToggleButton (кнопка с фиксацией), свойства Caption и Value, событие Change

Флажки и кнопки с фиксацией используются для выбора невзаимоисключающих вариантов (рис. 17).

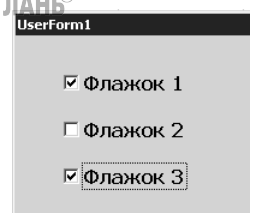


Рисунок 17 — Флажки (элементы управления CheckBox)

У CheckBox три главных свойства:

Caption — надпись справа от флажка, которая объясняет, что выбирается этим флажком;

TripleState — если в False (по умолчанию), то флажок может принимать только два состояния: установлен и нет. Если для TripleState установить значение True, то появляется третье значение: Null, когда установлена серая «галка». Такое значение часто используется, например, при выборе компонентов программы при установке, когда выбраны не все компоненты, а лишь некоторые;

Value — само состояние флажка. Может принимать значения True (флажок установлен), False (снят) и Null — «серый флажок» (когда свойство TripleState установлено в True).

Главное событие — Change.

ToggleButton выглядит как кнопка, которая при нажатии становится «нажатой», а при повторном нажатии отключается. У нее могут быть те же два (или три, в соответствии со свойством TripleState) состояния, что и у CheckBox. Свойства и методы — те же самые. Единственное отличие — в восприятии их пользователем. Обычно ToggleButton воспринимается пользователем как переход в какой-то режим или начало выполнения продолжительного действия.

Элементы управления OptionButton (переключатель) и Frame (рамка), свойства Caption и Value, событие Change

Если CheckBox предназначен для выбора невзаимоисключающих вариантов, то OptionButton (рис. 18) как раз нужен для выбора варианта в ситуации или-или.

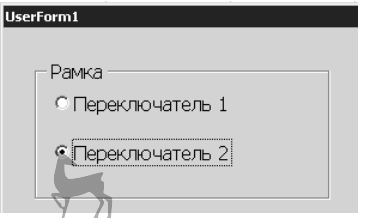


Рисунок 18 — Переключатели (объекты OptionButton) в рамке (объект Frame)

Главных свойств у этого элемента управления два:

Caption — надпись для переключателя;

Value — установлен флажок или нет (только два состояния — True или False).

Главное событие тоже очень привычное — Change.

Выбор должен предоставляться из нескольких вариантов, и при выборе одного из них другой автоматически очищается. Однако в некоторых ситуациях нам необходимо выбрать из нескольких наборов вариантов (например, отчет за месяц/квартал/год, тип отчета, нужный филиал и т.п.). Решение простое — переключатели нужно сгруппировать, для этого необходимо использовать элемент управления Frame.

Frame — это просто рамка, которая выделяет прямоугольную область на форме и позволяет организовать элементы управления. Помещенные внутрь рамки переключатели считаются взаимоисключающими, остальные элементы управления ведут себя точно так же, хотя иногда бывает полезно с точки зрения наглядности свести вместе под одной рамкой, к примеру, набор флажков. При желании рамку можно сделать невидимой, установив для свойства BorderStyle значение 1 и убрав значение свойства Caption.

Элемент управления CommandButton (кнопка), событие Click, свойства Caption, Cancel и Default. Элемент управления CommandButton (кнопка) — самый распространенный элемент управления в формах (рис. 19).

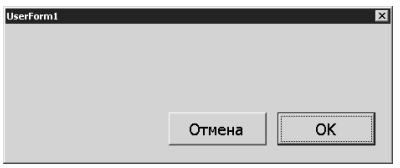


Рисунок 19 — Кнопки (объекты CommandButton) на форме

В большинстве форм обязательно будет по крайней мере две кнопки: Отмена (Cancel) и OK. При нажатии кнопки Отмена форма должна закрыться, при нажатии кнопки OK должно выполниться то действие, ради чего создавалась эта форма.

Главное событие для кнопки — это, конечно, Click. Как правило, к этому событию и привязывается тот программный код, ради которого создавалась кнопка.

Самые важные свойства кнопки:

Cancel — если для него установить значение True, то это значит, что кнопка будет нажиматься при нажатии на клавишу <Esc>. Как правило, на такие кнопки помещаются надписи типа «Отмена», «Выход», «Вернуться в окно приложения». Необходимо будет еще добавить код в обработчик события Click, например такой: Unload Me

Caption — надпись, которая будет на кнопке;

Default — такая кнопка будет считаться нажатой, если пользователь нажал на клавишу <Enter>, а фокус находился в другом месте формы (но не на другой кнопке). Обычно такие кнопки являются главными, по которым выполняется действие, ради которого создавалась форма (печать отчета, занесение информации в базу данных, отправка почты и т. п.);

Picture — если просто надпись вас не устраивает, можно назначить кнопке рисунок;

TakeFocusOnClick — будет ли передаваться управление этой кнопке при нажатии на нее. По умолчанию True.

Элементы управления ScrollBar (полоса прокрутки) и SpinButton (счетчик), их применение, свойства Min, Max, Value. Полосы прокрутки (ScrollBars) чаще всего встречаются в текстовых полях, когда введенный текст полностью на экране не умещается. Однако ничего не мешает вам использовать ScrollBar как отдельный элемент управления (пользователи часто называют его «ползунок») для выбора пользователем какого-то значения из диапазона (рис. 20). Обычно такой элемент управления используется для выбора плавно меняющихся значений — например, уровня громкости, яркости, сжатия, приоритета и т. п.

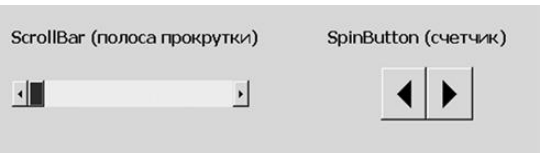


Рисунок 20 — Объекты полосы прокрутки (ScrollBar) и счетчика (SpinButton)

Главное событие для этого элемента управления — Change. Главные свойства выглядят так:

Max и Min — максимальное и минимальные значения, которые можно задать при помощи этого элемента управления. Возможный диапазон — от –32767 до +32767;

LargeChange и SmallChange — какими шагами будет двигаться ползунок при перемещении его пользователем (путем щелчка на полосе ниже ползунка или при нажатии на кнопку направления соответственно);

Orientation — определяет расположение ползунка (вертикальное или горизонтальное). По умолчанию для этого свойства установлено значение 1, что значит, что ориентация определяется автоматически в зависимости от конфигурации отведенного элементу управления пространства на форме (что больше — длина или высота). Однако при помощи этого свойства можно и явно указать вертикальное или горизонтальное расположение ползунка;

ProportionalThumb — определяет размер ползунка: будет ли он пропорционален размеру полосы прокрутки (по умолчанию) или фиксированного размера.

Value — главное свойство этого элемента управления. Определяет положение ползунка и то значение, которое будет возвращать этот элемент управления программе.

В самом простом варианте то, что выбрано при помощи ползунка, просто отображается в текстовой надписи:

```
Private Sub ScrollBar1_Change()
Label1.Caption = ScrollBar1.Value
End Sub
```

В более сложном варианте пользователю можно будет выбирать — использовать ли ползунок или вводить значение в тестовом поле. В этом случае в событии Change для текстового поля необходимо предусмотреть проверку вводимых пользователем значений и обратную связь с ползунком.

Элемент управления SpinButton — эта та же полоса прокрутки, лишенная самой полосы и ползунка. SpinButton используется в тех ситуациях, когда диапазон выбираемых значений совсем небольшой (например, надо выбрать количество копий для печати отчета). Все свойства, которые есть у SpinButton, совпадают со свойствами ScrollBar.

Элементы управления TabStrip и MultiPage, несколько вкладок на форме, свойства MultiRow, TabOrientation, Value. Оба этих элемента управления применяются в одной и той же ситуации — когда элементов управления слишком много, чтобы уместить их на одной странице формы. Эти элементы управления позволяют создавать на форме несколько вкладок (Page, страниц), между которыми сможет переходить пользователь. Принципиальное отличие между этими элементами управления заключается в том, что на вкладках TabStrip всегда располагаются одинаковые элементы управления, а MultiPage — разные.

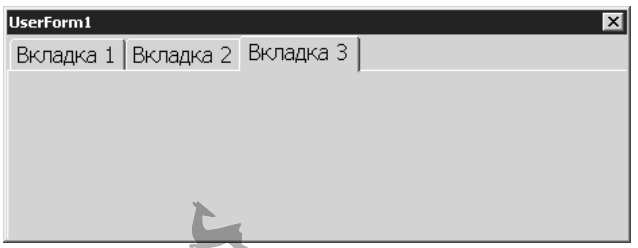


Рисунок 21 — Форма с несколькими вкладками (элементами управления MultiPage)

TabStrip используется реже. Например, возможное его применение — занесение данных по одному шаблону для филиалов или сотрудников (если их не слишком много). Свойства и события у этих элементов управления практически идентичны. Самые важные свойства:

MultiRow — можно ли будет использовать несколько рядов вкладок;

TabOrientation — где будут расположены вкладки (по умолчанию сверху).

Value — номер вкладки, которая открыта в настоящий момент (нумерация начинается с 0).

Главное событие — Change (то есть переход между вкладками). К нему можно привязать, например, проверку уже введенных пользователем значений или выдачу предупреждений.

Элемент управления Image, его применение, свойство Picture. Позволяет отобразить на форме рисунок в одном из распространенных форматов, который будет реагировать на щелчок мышью (украшения формы). Некоторые замечания по использованию Image:

- в качестве альтернативы можно использовать свойство Picture для формы (как фоновый рисунок);
- еще две альтернативы — применение свойства Picture для элементов управления Label или CommandButton. Функциональность получается практически одинаковая;
- при использовании этого элемента управления само изображение копируется внутрь документа и внешний его файл больше не нужен.

Главное событие элемента управления Image — событие Click.

Главные свойства:

Picture — позволяет выбрать само изображение для формы;

PictureAlignment — позволяет выбрать местонахождение изображения в отведенной ему области. По умолчанию по центру;

PictureSizeMode — позволяет выбрать режим растяжения/уменьшения элемента в случае, если он не точно соответствует размеру области;

PictureTiling — размножать ли маленький рисунок, чтобы он покрыл всю отведенную ему область (делать «черепицу»).

Контрольные вопросы:

1. Что такое элемент управления?
2. Для чего предназначен элемент управления Label?
3. Для чего предназначен элемент управления TextBox?
4. Какие свойства и методы имеет элемент управления ListBox?
5. Для чего предназначены элементы управления ScrollBar и SpinButton?
6. Какие свойства и методы имеет элемент управления CommandButton?

Лекция 12. Создание пользовательских диалоговых окон

План:

1. Понятие пользовательской формы. Модальный характер форм в VBA.
2. Отображение и закрытие пользовательской формы с помощью кода.
3. Свойства, методы и события пользовательской формы.

1. Понятие пользовательской формы. Модальный характер форм в VBA

VBA позволяет создавать и использовать в программах нестандартные (настраиваемые) диалоговые окна, добавляя объект UserForm в проект.

Используя разрабатываемые пользователем формы VBA, можно создавать нестандартные диалоговые окна для отображения данных или получения значений от пользователя программы в том виде, который наиболее соответствует потребностям программы. Например, можно создать тест, отобразить диалоговое окно для отображения вопросов с вариантами ответов и предоставить пользователю возможность выбрать один из вариантов ответа, который он считает верным.

Нестандартные диалоговые окна позволяют программе взаимодействовать с её пользователем самым сложным образом и обеспечивают разнообразную форму ввода и вывода данных.

Нестандартное диалоговое окно создаётся в VBA посредством добавления объекта UserForm в проект. Этот объект представляет собой пустое диалоговое окно; оно имеет строку заголовка и кнопку закрытия, но в нём отсутствуют какие-либо другие элементы управления.

Нестандартное диалоговое окно (форма) — создаётся путем добавления элементов управления в объект UserForm. Каждый объект UserForm имеет свойства, методы и события, наследуемые им от класса объектов UserForm.

Каждый объект User Form также содержит модуль класса, в который пользователь добавляет собственные методы и свойства или вписывает процедуры обработки событий для данной формы.

Для того чтобы создать собственный интерфейс, независимый от среды Excel, необходимы экранные формы.

Экранные формы — это окна различного назначения и вида, созданные пользователем для своего приложения. Они содержат элементы управления, позволяющие пользователю обмениваться информацией с приложением.

Объект User Form может содержать элементы управления подобные тем, что находятся в других диалоговых окнах, отображаемых Excel и другими Windows-приложениями.

Элементы управления — это элементы диалогового окна, которые позволяют пользователю взаимодействовать с программой. К этим элементам относят кнопки переключателей, текстовые поля, линейки прокрутки, командные кнопки и т. п.

Каждый элемент управления — объект со специальными свойствами, методами и событиями. Подобно содержащим их формам, средства элементов управления можно определять программным путём или с помощью окна Properties редактора Visual Basic. Значения свойства элемента управления присваиваются или получаются в программе VBA так же, как и для любого другого объекта.

2. Отображение и закрытие пользовательской формы с помощью кода

VBA использует созданный графический дизайн формы — с настройками свойств формы и элементов управления — для получения всей информации, необходимой для отображения диалогового окна: размеров диалогового окна, элементов управления в нём и т. п.

В результате VBA позволяет отобразить форму диалогового окна с помощью единственной инструкции — **метода Show** объекта UserForm. Если в настоящий момент форма не загружена в память, метод Show загружает форму и отображает её. Если форма уже загружена, метод Show просто отображает её. В любом случае этот метод отображает форму, а затем присваивает ей фокус. Форма остается до тех пор, пока не будет выполнен **метод Hide** объекта UserForm, либо пока форма не будет выгружена с помощью инструкции Unload.

Форма остаётся загруженной до тех пор, пока экземпляр формы не выйдет за пределы видимости, т. е. процедура, создавшая этот экземпляр объекта формы, перестанет выполняться либо пока форма не будет выгружена с помощью инструкции **Unload**.

Хотя программа в модуле класса формы будет выполняться в результате события в диалоговом окне, общее выполнение программы приостанавливается до тех пор, пока форма диалогового окна не будет закрыта или скрыта.

Использование VBA с элементами управления в форме

Отображения одного диалогового окна для выполнения задачи обычно недостаточно. Почти всегда требуется определить состояние элементов управления диалогового окна с целью выяснить, какие данные или опции выбрал пользователь. Например, если диалоговое окно используется для получения от пользователя информации о том, по каким столбцам и строкам должно выполняться упорядочение рабочего листа, необходимо иметь возможность выяснить, какие значения пользователь ввел после закрытия диалогового окна и до действительного начала операции упорядочивания.

В других случаях может потребоваться динамическое изменение заголовков кнопок (или других элементов управления) диалогового окна, динамическое обновление надписи или поля, связанного со счетчиком, или динамическое подтверждение введенных в диалоговое окно данных.

Как выглядит применение форм в приложении VBA?

Как правило, форма запускается при открытии пользователем документа. Пользователь выполняет на форме какие-то действия по вводу/выбору информации (например, выбирает значения в ниспадающем списке, устанавливает значения для флажков и переключателей и т. п.), а потом (обычно) нажимает на кнопку на этой форме — и введенная им информация передается в базу данных, отправляется по электронной почте, записывается в файл для распечатки и т. п.

3. Свойства, методы и события пользовательской формы

Создать форму очень просто: для этого достаточно в редакторе Visual Basic щелкнуть правой кнопкой мыши по проекту (то есть документу) в окне Project Explorer и в контекстном меню выбрать Insert -> User Form.

Откроется окно дизайнера форм (Form designer), в котором будет представлено пустое серое окно формы (по умолчанию она будет называться UserForm1) и рядом — Toolbox, панель с набором элементов управления (рис. 22).

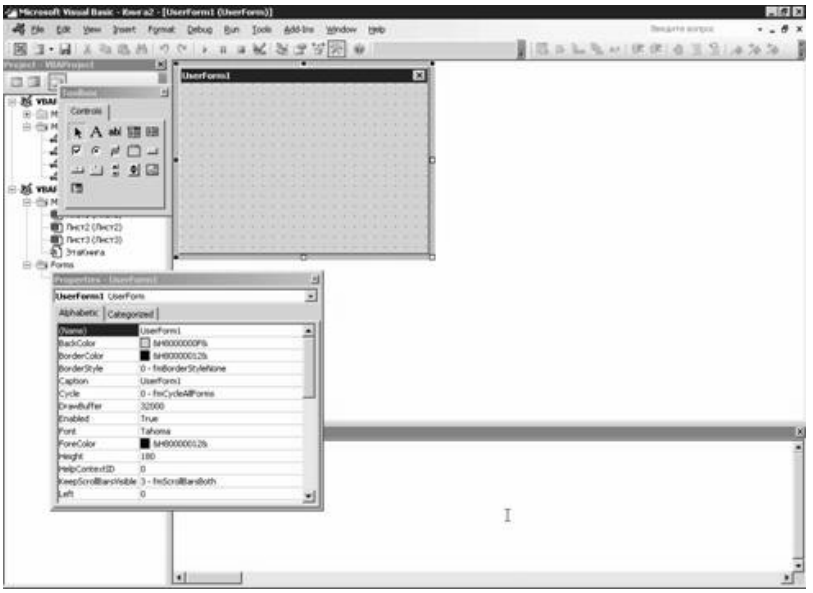


Рисунок 22 — Все готово для работы с формой

Если у вас включен показ окна свойств (он включается клавишей <F4>), то в этом окне будут представлены свойства формы. Переход на код для этой формы (по умолчанию открывается событие Click) — клавиша <F7>, возврат обратно в окно дизайнера форм — <Shift>+<F7>.

Очень удобно, что для форм и элементов управления можно настраивать свойства при помощи графического интерфейса окна свойств — резко уменьшается количество программного кода, которое нужно писать вручную.

Некоторые самые важные свойства форм (кроме ShowModal, все они применимы и для других элементов управления):

- свойство (Name) — определяет имя формы. Пользователь, скорее всего, его никогда не узнает. Имя формы используется только программистом в программном коде для этой формы (и в окнах редактора Visual Basic). После создания формы ее имя, предлагаемое по умолчанию (UserForm), рекомендуется заменить на что-либо более значимое, чтобы было проще ориентироваться в программе (это относится ко всем элементам управления);
- свойство Caption — определяет заголовок формы (по умолчанию совпадает с именем формы). Рекомендуется ввести строку, которая будет напоминать пользователю о назначении формы (например, «Выбор типа отчета»);

• свойство Enabled — если установлено в False, пользователь работать с формой не сможет. Используется для временного отключения формы, например пока пользователь не обеспечит какие-то условия для ее работы;

• свойство ShowModal — если установлено в True (по умолчанию), пользователь не может перейти к другим формам или вернуться в документ, пока не закроет эту форму. В версиях до VBA6 поддерживались только модальные формы.

Большая часть основных свойств относится к внешнему виду, размерам и местонахождению окон.

Самые важные методы форм.

В процессе редактирования формы (из окна редактора Visual Basic) форму можно запускать нажатием клавиши <F5>. После того как форма будет готова, вы должны обеспечить запуск этой формы в документе. Для запуска формы нужно воспользоваться методом Show(): **UserForm1.Show.**

Если форма уже была загружена в память, она просто станет видимой, если еще нет — то будет автоматически загружена (произойдет событие Load).

Саму эту команду, можно вызвать, например:

• из обычного макроса, привязанного к кнопке или клавиатурной комбинации;

• из автозапускаемого макроса;

• из кода для элемента управления, расположенного в самом документе (например, CommandButton) или на другой форме — для перехода между формами;

• поместить ее в обработчик события Open для книги Excel, чтобы форма открывалась автоматически при открытии документа.

После того как пользователь введет/выберет нужные данные на форме и нажмет на требуемую кнопку, форму необходимо убрать. Можно для этой цели воспользоваться двумя способами:

• спрятать форму (использовать метод Hide(), например **UserForm1.Hide)**. Форма будет убрана с экрана, но останется в памяти. Потом при помощи метода Show() можно будет опять ее вызвать в том же состоянии, в каком она была на момент «прятанья», а можно, например пока она спрятана, программно изменять ее и расположенные на ней элементы управления. Окончательно форма удалится из памяти при закрытии документа;

• если форма больше точно не потребуется, можно ее удалить из памяти при помощи команды Unload: **Unload UserForm1**.

Остальные методы относятся либо к обмену данными через буфер обмена (Copy(), Cut(), Paste()), либо к служебным возможностям формы (PrintForm(), Repaint(), Scroll()).

Важнейшая концепция VBA — события.

Событие (event) — это что-то, что происходит с программой и может быть ею распознано. Например, к событиям относятся щелчки мышью, нажатия на клавиши, открытие и закрытие форм, перемещение формы по экрану и т. п. VBA построен таким образом, чтобы создавать на нем программы, управляе-

мые событиями (event-driven). Такие программы противопоставляются устаревшему процедурному программированию.

Самые важные события форм:

- Initialize — происходит при подготовке формы к открытию (появлению перед пользователем). Обычно в обработчик для этого события помещается код, связанный с открытием соединений базы данных, настройкой элементов управления на форме, присвоением им значений по умолчанию и т. п.
- Click (это событие выбирается по умолчанию) и DblClick — реакция на одиночный и двойной щелчок мыши соответственно. Для формы это событие используется не так часто. Обычно обработчик щелчков используется для кнопок (элементов управления CommandButton). По причине простоты мы использовали это событие для демонстрации нашего кода.
- Error — это событие используется при возникновении ошибки в форме, используется как возможность предоставить пользователю исправить сделанную им ошибку. Подробнее — в специальном модуле, посвященном ошибкам и отладке;
- Terminate — событие используется при нормальном завершении работы формы и выгрузке ее из памяти (например, по команде Unload). Обычно используется для разрыва открытых соединений с базой данных, освобождения ресурсов, протоколирования и т.п. Если работа формы завершается аварийно (например, запустившее форму приложение выдало команду End), то это событие не возникает;
- остальные события связаны либо с изменением размера окон, либо с нажатиями клавиш, либо с активизацией (получением фокуса)/деактивизацией (потерей фокуса).

Поскольку форма — это во многом просто контейнер для хранения других элементов управления, главное ее событие — **Initialize**. Все остальные события обычно используются не для формы, а для расположенных на ней элементов управления.

Если все нужные вам элементы управления трудно уместить на одной форме (даже большого размера), в вашем распоряжении два варианта: воспользоваться двумя формами (переходя между ними при помощи методов Show() и Hide(), подвязанных к элементам управления) или воспользоваться несколькими вкладками для формы. Для этой цели в вашем распоряжении имеется специальный элемент управления Multipage.

Контрольные вопросы

1. Области применения пользовательских диалоговых окон.
2. Что такое модальное окно?
3. Перечислите типы модальных окон. Чем они различаются?
4. Как создать нестандартное диалоговое окно?
5. Какие свойства и методы имеет форма?

Лекция 13. Создание пользовательских меню и панелей инструментов

План:

1. Обзор объектной модели CommandBar. Семейство объектов CommandBars.
2. Создание новой панели инструментов в тексте программы.
3. Управление объектами CommandBar.

1. Обзор объектной модели CommandBar. Семейство объектов CommandBars

Очень часто в приложении VBA вам потребуются свои наборы меню и панелей инструментов вместо стандартных, предусмотренных приложением. Работу с меню и панелями инструментов обеспечивает коллекция *CommandBars*, которая находится в объекте Application (об этом важном объекте мы будем говорить в следующих модулях). Коллекция CommandBars содержит, как ясно из названия, набор объектов *CommandBar* (панели инструментов и меню), каждый из них, в свою очередь, — коллекцию *CommandBarControls* (набор элементов, из которых состоит панель/меню), а эта коллекция представляет собой хранилище элементов, из которых и состоит меню. Таких элементов может быть три:

CommandBarButton — кнопка или элемент меню, который используется для выполнения программы или подпрограммы;

CommandBarComboBox — сложный элемент меню/панели управления (поле ввода, раскрывающийся список, поле со списком);

CommandBarPopup — меню или вложенное меню.

Пример создания собственной панели инструментов может выглядеть очень просто:

```
Dim CBar1 As CommandBar
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
CBar1.Enabled = True
CBar1.Visible = True
```

У нас появилась новая панель инструментов **Документы** (рис. 23), которое можно, к примеру, убрать через меню **Настройка** -> **Панели** инструментов, однако пока она совершенно бесполезна: в нем нет ни одной кнопки. Для того чтобы они появились, необходимо добавить новый элемент типа CommandBarControl (одного из трех типов, перечисленных выше) в коллекцию CommandBarControls для этого меню.

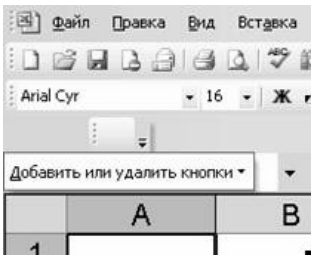


Рисунок 23 — В центре рисунка — пустая панель инструментов ***Документы***

Основные свойства и методы объекта CommandBar:

• *BuiltIn* — это свойство определяет, является ли данная панель/меню встроенной для этого приложения (то есть предусмотренной в нем разработчиками приложения Office). Менять значение этого свойства нельзя. Это свойство очень удобно использовать, чтобы убрать все стандартные меню или, наоборот, убрать все свои меню, оставив только стандартные;

• *Context* — определяет, где именно находится программный код для вашего меню (в Normal.dot, файле документа и т. п.). Можно использовать для проверок в случае потенциальной возможности вызова разных меню с одинаковыми именами;

• *Controls* — через это свойство можно получить коллекцию элементов управления CommandBarControls, которая нам, скорее всего, потребуется;

• *Enabled* — «включение/отключение» панели;

• *Height*, *Left*, *Top* и *Width* — очевидные свойства, относящиеся к расположению панели в окне приложения;

• *Index*, *Name* и *NameLocal* — эти свойства позволяют найти нужную нам панель в коллекции CommandBars. Name — это программное имя объекта, NameLocal — имя, которое будет видно пользователю, Index — это просто номер данной панели;

• *Protection* — позволяет запретить пользователю убирать старые кнопки из этой панели или размещать на ней новые;

• *Type* — наверное, самое важное свойство. Определяет, чем будет данная панель/меню (панелью инструментов, обычным меню или контекстным меню, доступным по щелчку правой кнопкой мыши). Однако это свойство доступно только для чтения. Тип объекта CommandBar определяется при выполнении метода Add() коллекции CommandBars, при этом определяется очень хитро — через второй параметр этого метода. В нашем случае вызов этого метода выглядит как

Set CBar1 = CommandBars.Add("Документы", msoBarTop, True, False)

Первый параметр ("Документы") — это имя объекта (название панели). Второй параметр (msoBarTop) — либо положение пристыкованной панели (msoBarTop, msoBarBottom, msoBarLeft, msoBarRight), либо знак того, что панель непристыкована (msoBarFloating), либо вообще указание на то, что это — контекстное меню, которое до щелчка правой кнопкой мыши не видно (msoBarPopup). Свойство Visible для контекстного меню неприменимо.

После того как создание панели завершено, необходимо разместить в нем элементы. Например, создание панели инструментов может выглядеть так:

Dim But1 As CommandBarControl

Set But1 = CBar1.Controls.Add(msoControlButton)

But1.Caption = "Кнопка 1"

Вроде бы ничего не произошло — перед нами та же пустая панель. Однако, если навести указатель мыши на начало панели, в нем можно увидеть пустую кнопку, которая называется "Кнопка 1" (рис. 24).

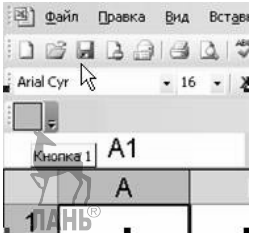


Рисунок 24 — На панели инструментов ***Документы*** *появилась почти невидимая* ***Кнопка 1***

Как видно из кода, мы использовали для создания элемента управления (кнопки в панели инструментов) метод Add() коллекции Controls объекта CommandBar (у нас он называется CBar1). Свойства и методы у этой коллекции стандартные, как у множества других коллекций:

- свойство *Application* возвращает ссылку на объект приложения (Word, Excel и т. п.);
- свойство *Count* позволяет узнать, сколько всего элементов управления помещено в эту коллекцию;
- свойство *Item* позволяет по индексу (номеру) получить ссылку на объект элемента управления CommandBarControl в этой коллекции;
- метод *Add()* (которым мы и воспользовались) позволяет добавить элемент управления в эту коллекцию (удаление элемента управления производится уже при помощи метода Delete() самого элемента управления).

Метод Add() принимает пять необязательных параметров, из которых два первых параметра — очень важны. Первый параметр — Type, он определяет тип передаваемого в коллекцию элемента управления. Таких типов пять: msoControlButton (кнопка — то есть в итоге получится панель инструментов), msoControlEdit (будет создано поле для ввода текста), msoControlDropdown (выпадающий список), msoControlComboBox (комбинированный список), or msoControlPopup (пункт меню), например, чтобы наша кнопка превратилась в начальный пункт выпадающего меню (рис. 25), в нашем коде можно изменить один этот параметр:

```
Dim But1 As CommandBarControl
Set But1 = CBar1.Controls.Add(msoControlPopup)
But1.Caption = "Меню 1"
```

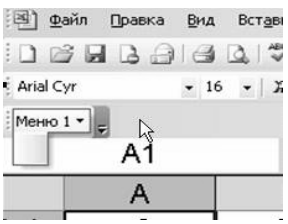


Рисунок 25 — На созданной нами панели инструментов появился пункт меню

Второй важный параметр метода Add() — параметр ID. Этот параметр позволяет привязать элемент управления к уже имеющемуся в системе встроенному элементу управления. Например, чтобы добавить кнопку печати, этот параметр должен быть равен 4, чтобы добавить кнопку предпросмотра документа, этот параметр должен быть равен 5. Если оставить этот параметр пустым или указать для него значение 1, то будет создан пользовательский элемент управления, не привязанный ни к каким встроенным.

Остальные параметры этого метода относятся к созданию идентификатора элемента управления, его положению относительно других элементов управления и должен ли он быть постоянным или временным.

Для объекта CommandBarButton важнейшие свойства, методы и события приводятся ниже:

- *Caption* — надпись на элементе управления. Для кнопки выводится в виде всплывающей подсказки, а для пункта меню — это название пункта;
- *Enabled* — включен или отключен данный элемент управления. Обычно используется для предупреждения ошибок пользователя;
- *FaceId* (только для кнопок) — возможность воспользоваться системной картинкой для кнопки (не назначая ей соответствующую функцию). Например, значение 4 присвоит кнопке изображение принтера. В Word и Excel встроено несколько тысяч иконок, и поэтому вместо создания иконки всегда есть возможность подобрать готовую
- *Id* — идентификатор встроенной функции, назначенной этой кнопке. Если вы назначили этой кнопке пользовательскую процедуру, то значение Id всегда будет равно 1;
- *Index* — номер элемента управления в коллекции Controls. Используется для выполнения всяких служебных операций с элементами управления;
- *Mask* — возможность наложить на рисунок объекта маску для показа только части рисунка. Маска выглядит как отдельный рисунок, прозрачные части которого должны быть белыми, а непрозрачные — черными;
- *OnAction* — самое важное свойство элемента управления. Его значение используется, когда активизируется элемент управления (щелчок по кнопке или пункту меню, завершение ввода текста в текстовом поле, выбор нового значения в списке). Предназначено для указания запускаемой процедуры или внешнего приложения (COM Add In). Пример: *But1.OnAction = "MySub"*;
- *Parameter* — это свойство можно использовать для передачи параметров вызываемой подпроцедуре или просто для хранения своих данных. Работает с типом данных String;
- *Picture* — это свойство позволяет назначить рисунок (иконку) кнопке панели инструментов. Чаще используется не оно, а свойство FaceId (выше). Если есть необходимость, то необходимые иконки можно подобрать из большой коллекции, которая есть в Visual Studio, или воспользоваться одним из свободно доступных генераторов иконок;
- *ShortcutText* — текст с описанием клавиатурного сочетания, назначенного этой кнопке или пункту меню;

- *State* — вид кнопки — обычная, утопленная или обведенная рамкой;
- *Style* — еще одно свойство, влияющее на внешний вид. Позволяет определить, как будет выглядеть кнопка: будет показана только иконка, только надпись или и то, и другое вместе в разных вариантах
- *ToolTip* — определяет текст всплывающей подсказки. По умолчанию «всплывает» значение свойства Caption;
- *Type* — возвращает тип элемента управления (использоваться для изменения типа не может!);
- *Visible* — будет или не будет этот элемент управления видимым;
- метод *Delete()* — позволяет удалить кнопку из коллекции кнопок;
- метод *Execute()* — запускает на выполнение то, что определено при помощи свойства OnAction;
- *Reset()* — вернуться к исходным параметрам кнопки после внесенных изменений.

У этого объекта есть единственное событие — *Click*. Оно также позволяет определить реакцию на нажатие кнопки, но работать с ним сложнее, чем со свойством OnAction.

Для работы с вложенными меню используется точно та же коллекция Controls, которой мы уже пользовались. Единственное отличие — эта коллекция Controls принадлежит не объекту CommandBar, а объекту CommandBarPopup — то есть другому меню!

Контекстные меню — это меню, которые открываются по щелчку правой кнопкой мыши. С ними работа выглядит так:

```
Set CBar1 = CommandBars.Add ("Мое контекстное меню", msoBar-
Popup , , True )
Set MenuItem1 = CBar1.Controls.Add
MenuItem1.FaceId = 3
MenuItem1.Caption = "Элемент меню 1"
Set MenuItem2 = CBar1.Controls.Add
MenuItem2.FaceId = 5
MenuItem2.Caption = "Элемент меню 2"
```

Однако если просто выполним этот код, то никакого контекстного меню не появится: необходимо еще добавить вызов метода ShowPopup(): *CBar1.ShowPopup.*

2. Создание новой панели инструментов в тексте программы

В меню объединяют последовательности, группы, наборы команд (рубрик), одну из которых может выбрать пользователь для совершения очередного действия. Как правило, названия команд в меню достаточно информативны, так что пользователь может легко найти нужную ему команду. Команды для решения близких задач можно объединить в группу, поместив их рядом в одном меню и отделив чертой от команд, решающих другие задачи. В одном меню команды объединяют на основе одного из двух принципов: либо это различные действия над одним объектом, либо однотипные действия над различными объ-

ектами. Первый принцип реализован в меню Файл (File) всех приложений Office, объединяющем различные действия над основным объектом приложения: создание, открытие и закрытие, сохранение, пересылка и печать объекта. Меню Вставка (Insert) построено по другому принципу — действие «вставить» выполняется для различных объектов. Если вариантов исполняемых действий много, их структурируют, используя подменю — списки команд, появляющиеся правее или левее выбранной команды родительского меню. Еще один вид меню — контекстные (shortcut) меню, всплывающие при нажатии правой кнопки мыши в определенном контексте. В них можно объединять действия, допустимые (факультативно) в данном контексте. Доступ пользователя к команде меню можно ускорить, определив для нее «горячие» клавиши, нажатие которых эквивалентно выбору этой команды.

Рассмотрим работу с объектами, относящимися к меню. Эту работу, как правило, можно проводить двумя способами:

- «визуально», используя встроенные команды и диалоговые окна приложений Office;
- «программно», создавая в процедурах VBA объекты указанных классов, задавая и изменяя их свойства соответствующими методами.

Создание собственного головного меню

Согласно существующим формальным и фактическим стандартам проектирования интерфейса, работа прикладной программы должна начинаться с активизации головного меню, которое находится в верхней части окна приложения. Собственное головное меню для прикладной системы можно спроектировать визуально — «руками», вызвав диалоговое окно Настройка (Customize), или используя VBA.

3. Управление объектами CommandBar

Новое меню в Excel, Word или PowerPoint обычно создается средствами VBA, поскольку в этих приложениях диалоговое окно Настройка не содержит кнопки Свойства. Новое меню создается методом Add коллекции CommandBars (Панели команд):

выражение.Add(Name, Position, MenuBar, Temporary),

где выражение — обязательное выражение, возвращающее объект CommandBars, а все параметры в скобках необязательны. Name задает имя нового меню; Position определяет его положение (значения-константы msoBarLeft, msoBarTop, msoBarRight, msoBarButtom определяют положение меню слева, вверху, справа или внизу окна, msoBarFloating задает «плавающее» меню, msoBarPopup указывает, что новое меню будет всплывающим). Значение True параметра MenuBar указывает, что новое меню заменит текущую активную строку меню (по умолчанию False). Значение True параметраTemporary означает, что новое меню будет временным и исчезнет, когда закроется содержащее его приложение (по умолчанию False).

В следующем примере создается новое меню «Головное меню»:

```
Dim CstmBar As CommandBar
Set CstmBar = CommandBars.Add(Name:="Головное меню",
Position:=msoBarTop, MenuBar:=True, Temporary:=False)
```

Добавление выпадающего меню с помощью VBA. Чтобы добавить новое меню программно, нужно использовать метод Add коллекции CommandBarControls, применив его к объекту типа CommandBar, который представляет панель нашего меню. Этот метод позволяет помещать на панель кнопки (CommandBarButton), комбинированные списки (Command-BarComboBox) и выпадающие меню (ComandBarPopup). Его вызов имеет вид:

выражение.Add(Type, Id, Parameter, Before, Temporary),

где выражение должно возвратить объект типа CommandBarsControl, параметры в скобках необязательны. Параметр Type (Тип) задает тип добавляемого объекта. Его значение msoControlPopup указывает, что добавляемый управляющий элемент — выпадающее меню. Для пользовательских меню параметры Id и Parameter можно опустить. Значение аргумента Before — число, указывающее положение нового элемента в последовательности элементов панели (если его нет, элемент помещается в конец). Определить имя и ключ быстрого доступа к созданному меню можно, задав значение свойства Caption.

Добавление подменю. Подменю (или дочернее меню) примыкает к боковой стороне другого меню — родительского — на уровне той команды родительского меню, которая является заголовком подменю. Подменю можно добавлять, как к выпадающим меню, так и к другим подменю и к всплывающим меню. Сначала добавляется пустое подменю, затем в него вставляются команды. Добавить подменю можно двумя способами: с помощью диалогового окна Настройка и через вызов метода Add в VBA.

Чтобы добавить новое подменю программно, к объекту, представляющему родительское выпадающее меню, нужно применить метод Add коллекции CommandBarControls. В качестве параметра Type (Тип) нужно использовать значение msoControlPopup.

В следующем примере в конец выпадающего меню «Ввод документов», расположенного на панели меню «Головное меню», добавляется подменю «о движении товаров».

```
Dim CstmPopUp1 As CommandBarPopup
Set CstmPopUp1 = CstmCtrl.Controls.Add(Type:=msoControlPopup)
CstmPopUp1.Caption = " о движении товаров"
```

Добавление команд с помощью VBA. Добавить новую команду в меню можно, применив метод Add коллекции CommandBarControls к объекту, представляющему изменяемое меню. Чтобы добавить собственную команду, вставьте ее имя в меню, а затем в качестве значения свойства OnAction задайте имя VBA-процедуры, которая должна вызываться при выборе данной команды. В качестве значения аргумента Type (Тип) метода Add укажите msoControlButton, означающее, что вставляемый в меню элемент будет командой. Добавим команду «Накладная» в выпадающее меню «Ввод документов» из панели «Головное меню». Выбор этой команды запускает процедуру Invoice.

```
Set CstmCtrl = CstmPopUp1.Controls.Add(Type:=msoControlButton)
CstmCtrl.Caption = "Накладная"
CstmCtrl.OnAction = "Module1.Invoice"
```

Группировка команд меню

Разделение групп логически связанных команд меню горизонтальными линиями позволяет пользователям более эффективно работать с большими меню, содержащими разнотипные команды. Сами разделяющие линии командами не являются. Установить или убрать их можно в диалоговом окне Настройка. Для этого в этом окне сделайте видимым модифицируемое меню. Затем щелкните правой кнопкой мыши команду, над которой хотите провести линию. В появившемся меню выберите команду Начало группы (Begin Group). Убирается линия аналогично.

Ту же задачу разбиения команд на группы можно решить из VBA, присвоив значение True свойству BeginGroup (Начало группы) объекта, представляющего команду, которая должна открывать очередную группу. Чтобы убрать разделяющую линию, присвойте этому свойству False. Вот как выделить группу команд меню «Ввод документов», начинающуюся со вставленной команды «ввод накладной»:

```
Set InvCommand = CommandBars(«Головное меню»).Controls("Ввод документов") .Controls("ввод накладной")
InvCommand.BeginGroup = True
```

Удаление команды с помощью VBA. Для удаления компонента меню используется метод Delete (Удалить). Этот оператор, например, удаляет выпадающее подменю Favorites из панели меню Web:

```
CommandBars("Web").Controls("Favorites").Delete
```

Собственную (пользовательскую) панель меню "Головное меню" можно целиком удалить оператором:

```
CommandBars("Головное меню").Delete
```

Восстановить можно только встроенный удаленный элемент меню. Если при этом восстановится заголовок выпадающего меню или подменю, вместе с ним восстановится само меню и все его потомки.

Восстановить встроенный компонент меню программно можно методом Reset. Например, следующий оператор восстанавливает выпадающее подменю Favorites из панели меню Web:

```
CommandBars("Web").Controls("Favorites").Reset
```

А вот как восстановить всю панель меню Menu Bar из Word:

```
CommandBars("Menu Bar").Reset
```

Изменение меню во время работы программы. VBA позволяет изменять и настраивать систему меню приложения динамически во время его работы. Предоставляемые для этого возможности велики. Вы можете заменять одну панель меню на другую, удалять команды меню или делать их временно недоступными («серыми»), переименовывать команды.

Вывод собственной панели меню. Чтобы на экране показалась новая панель меню вместо текущей, задайте значение True свойству Visible объекта CommandBar, представляющего панель меню, которая должна появиться. Новая панель меню заменит прежнюю активную панель на экране. Чтобы восстановить прежнюю панель, при завершении работы программы нужно задать свойству Visible значение False. В Excel и PowerPoint для замены основной панели

приложения применяется процедура, обрабатывающая событие, после которого должна появиться новая панель или макрос. В Word можно обеспечить вывод собственной панели меню при запуске приложения, если заранее активизировать ее и сохранить в проекте Normal. Тогда при следующем запуске эта панель появится на экране как основная. Другой вариант — установить свойство Visible в процедуре, обрабатывающей событие Open.

Динамическое изменение видимости команд меню. Если некоторые команды меню выполняют действия, связанные с определенными объектами (документами), имеет смысл делать их видимыми при активизации этих объектов и скрывать, когда соответствующие объекты недоступны. Для этого свойству Visible этих команд нужно задать значение True при активизации объекта и False — при его деактивизации (закрытии, удалении и т. п.). Точно так же можно делать видимой и скрывать и панель меню, предназначенную для работы с определенным объектом.

Проще всего выполнять переустановку свойства Visible в процедурах, обрабатывающих события активизации (открытия, загрузки, появления на экране и т. п.) и деактивизации (закрытия, выгрузки, удаления с экрана и т. п.). Если для данного объекта нет подходящих событий, можно попытаться включить изменение свойства Visible для интересующих нас команд в процедуры, обрабатывающие событие OnAction других команд меню или управляющих кнопок.

Word сохраняет параметры настройки (в том числе и пользовательские меню) в документах и шаблонах. Поэтому эти компоненты меню видны, если соответствующий документ или шаблон доступен в данном контексте, и они исчезают с экрана, когда документ или шаблон недоступен. Excel же сохраняет пользовательские изменения интерфейса в рабочей памяти, поэтому для управления видимостью компонентов меню применяется изменение свойства Visible в процедурах обработки событий.

Управление доступом к командам меню. Иногда удобнее управлять доступом пользователя к командам меню, не скрывая и восстанавливая их на экране, а отключая реакции на их выбор и нажатие. Такие команды видны на экране, но отличаются от активных команд своим тусклым, «серым» видом. Для управления доступом к компонентам меню служит булево свойство Enabled (Включен). Если оно равно True, соответствующий компонент (команда, выпадающее меню или подменю) доступен, False делает компонент недоступным («серым»). Установив для свойства Enabled подменю значение False, можно сделать недоступными все его команды. Так можно управлять доступом к собственным компонентам меню. Для встроенных компонентов меню переустановить свойство Enabled нельзя.

Контрольные вопросы

1. Для чего предназначена коллекция CommandBars?
2. Какие элементы включает семейство объектов CommandBars?
3. Перечислите основные свойства объекта CommandBar.
4. Перечислите основные методы объекта CommandBar.

Лекция 14. Объекты, используемые для анализа данных в Excel

План:

1. Структура объекта PivotTable.
2. Свойства и методы объекта PivotTable.
3. Внутренняя организация интерфейса сводных таблиц.
4. Построение и форматирование диаграмм. Свойства и методы объекта Char

1. Структура объекта PivotTable

Сводные таблицы являются одним из наиболее мощных средств Excel для анализа данных, помещенных в таблицы или списки. Сводные таблицы позволяют группировать данные и производить их анализ. Создавая сводные таблицы, пользователь оперирует именами полей, которые должны помещаться в ее строках и столбцах. Возможно также задание поля страницы, превращающего сводную таблицу в подшивку из нескольких страниц. Иерархически сводная таблица входит в рабочий лист. Все сводные таблицы рабочей книги образуют семейство PivotTabies (сводные таблицы), которое содержит в себе семейство PivotFieids (поля сводной таблицы) всех полей, входящих в сводную таблицу. Объект Pivotitem (элемент сводной таблицы) является конкретным элементом объекта PivotFieid. Все объекты Pivotitem образуют семейство Pivotitems. На рисунке 26 показана иерархия этих семейств.

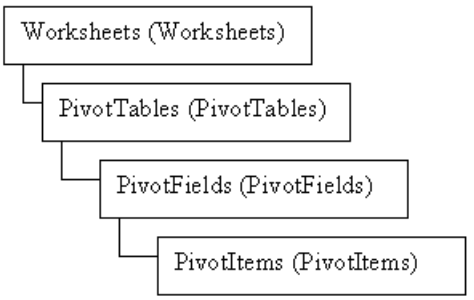


Рисунок 26 — Иерархия семейства объектов PivotTabies, PivotFieids и Pivotitems

2. Свойства и методы объекта PivotTable

Семейство PivotTabies имеет единственный метод item, возвращающий элемент этого семейства, т. е. конкретную сводную таблицу.

Синтаксис: Item(Index), где Index — имя или номер возвращаемого элемента семейства PivotFieids

Из свойств семейства PivotTabies отметим только свойство *Count*, возвращающее число элементов этого семейства.

Метод PivotTableWizard. Программно сводная таблица создается методом PivotTableWizard. Вручную на рабочем листе сводная таблица конструируется с помощью команды **Данные, Сводная таблица** (Data, Pivot Table and Pivot Chart Report).

Синтаксис: Объект.PivotTableWizard (SourceType, SourceData, TableDestination, TableName, RowGrand, ColumnGrand, SaveData, HasAutoFormat, AutoPage, Reserved, BackgroundQuery, OptimizeCache, PageFieldOrder, PageFieldWrapCount, ReadData, Connection).

Таблица 26 — Аргументы метода PivotTableWizard

Объект	**Объект Worksheet (рабочий лист) или PivotTable (сводная таблица)**
SourceType	Тип источника данных. Допустимые значения: xlConsolidation (консолидация нескольких диапазонов рабочих листов Excel); xlDatabase (список или база данных Excel); xlExternal (внешняя база данных); xlPivotTabie (сводная таблица)
SourceData	Определяет вид источника данных в зависимости от значения аргумента SourceType: • диапазон, если значением аргумента является xlDatabase; • массив строк, содержащий строку связи ODBC и SQL-оператор, если xlExternal; • массив диапазонов, если xlConsolidation; • имя существующей сводной таблицы, если xlPivotTable
TableDestination	Диапазон, где будет размещена сводная таблица
TableName	Имя создаваемой сводной таблицы
RowGrand	Допустимые значения: True (отображается суммарный итог по строкам сводной таблицы) и False (итог не отображается)
ColumnGrand	Допустимые значения: True (отображается суммарный итог по столбцам сводной таблицы) и False (итог не отображается)
SaveData	Допустимые значения: True (сохраняются данные вместе со сводной таблицей) и False (сохраняется только сводная таблица)
HasAuto Format	Допустимые значения: True (автоматическое переформатирование сводной таблицы при изменении данных) и False (в противном случае)
AutoPage	Применим только при аргументе sourceType, равным xlConsolidation. Допустимые значения: True (Excel создает поле страницы) и False (пользователь должен создать поле)
Reserved	Не используется
Background Query	True (Excel выполняет запрос в фоновом режиме) и False (в последовательном)
OptimizeCache	Допустимые значения: True (создается сводная таблица в режиме оптимизации) и False (оптимизация выключена, что убыстряет создание сводной таблицы)
PagePieldOrder	Задает ориентацию поля страницы. Допустимые значения: xlDownThenOver и xlOverThenDown
PageFieldWrapCount	Задает номер поля, с которого начинается новая страница. По умолчанию 0, т. е. отменена разбивка на страницы
ReadData	Допустимые значения: True (данные сразу считываются в кэш) и False (данные считываются в кэш по мере необходимости)
Connection	Используется для указания источника данных ODBC, источника данных URL и имени файла, содержащего запрос

С методом pivotTableWizard тесно связан метод PivotTables, применяемый к рабочему листу. Метод PivotTabies возвращает объект PivotTable или семей-

ство сводных таблиц, размещенных на рабочем листе. Этот метод имеет два синтаксиса.

Синтаксис 1: **Объект.PivotTabies**

Возвращает семейство сводных таблиц. Здесь и во втором синтаксисе объект — рабочий лист.

Синтаксис 2: **Объект.PivotTables(Index)**

Возвращает сводную таблицу из семейства сводных таблиц с именем или номером, указанным в аргументе index.

Таблица 27 — Методы объекта PivotTable

Методы	Описание
PivotFields	Возвращает объект, являющийся либо единичным полем (синтаксис 1), либо семейством полей. Синтаксис 1: **PivotFields (Index)** • Index — имя или номер поля сводной таблицы Синтаксис 2: PivotFields
PivotSelect	Выбирает элементы сводной таблицы. Синтаксис: **PivotSelect (Name, Mode)** Аргументы: G Name — строковое выражение, идентифицирующее выбранный элемент • Mode — специфицирует структуры выбранного элемента. Допустимые значения: xlBlanks, xiButton, xlDataAndLabel, xlDataOnly, xlLabelOnly или xlOrigin
RefreshTable	Обновляет данные. Дело в том, что в сводной таблице не происходит автоматический перерасчет при изменении исходных данных. Для перерасчета сводной таблицы вручную надо ее выделить и выбрать команду Данные, Обновить данные (Data, Refresh Data). Программно перерасчет сводной таблицы производится методом RefreshTable
AddFields	Добавляет строки, столбцы и страницы в сводную таблицу. Синтаксис: **AddFields (RowFields, ColumnFields, PageFields, AddToTable)** Аргументы: • RowFields — специфицирует имя или массив имен полей, которые будут играть роль строк сводной таблицы; • ColumnFields — специфицирует имя или массив имен полей, которые будут играть роль столбцов сводной таблицы; • FageFields — специфицирует имя или массив имен полей, которые будут играть роль страниц сводной таблицы; • AddToTabie — допустимые значения: True (добавляет поля в сводную таблицу) и False (заменяет существующие поля)

Таблица 28 — Свойства объекта pivotTabie

Свойства	Описание
ColumnFields, RowFields, Data-Fields и PageFields	Возвращает объект, являющийся либо единичным полем (синтаксис 1), либо семейством полей (синтаксис 2), который является столбцом (строкой, данными или страницей) сводной таблицы. Синтаксис 1: **ColumnFields (Index) RowFields (Index) Data-Fields(Index) PageFields (Index)** • Index — имя или номер поля сводной таблицы Синтаксис 2: **ColumnFields RowFields DataFields PageFields**
VisibleFields и HiddenFields	Возвращает объект, являющийся либо единичным полем (синтаксис 1), либо семейством полей (синтаксис 2), который в данный момент отображается (скрыт) в сводной таблице. Синтаксис 1: **VisibleFields (Index) HiddenFields (Index)** • Index — имя или номер поля сводной таблицы Синтаксис 2: **VisibleFields HiddenFields**

3. Внутренняя организация интерфейса сводных таблиц

Кэш сводной таблицы. При создании или обновлении сводной таблицы, независимо от выбранного типа источника, Excel переносит данные в промежуточное хранилище, так называемый кэш сводной таблицы. Структура организации данных в кэше позволяет существенно оптимизировать агрегацию данных и вычисления в сводной таблице. Хранение данных в собственном кэше позволяет использовать различные источники данных с сохранением схожей функциональности.

Данные в кэше обновляются при нажатии кнопки «Обновить» интерфейса сводной таблицы (кнопка на ленте или в контекстном меню) либо по заданному интервалу времени, если такая установка задана в параметрах. Режим вычислений Excel (автоматический или ручной) при этом никак не влияет на сводную таблицу.

Несколько сводных таблиц (или диаграмм) могут отображать данные одного и того же кэша. Этот вариант работы используется для отображения нескольких отчетных форм одних и тех же данных без использования интерфейса настройки измерений. В этом случае при обновлении одной из таблиц автоматически перестраивается и та, что основана на том же кэше.

Объекты VBA. Доступ к данным программными методами возможен на уровне объектов сводной таблицы — объект PivotTable. Другие объекты сводной таблицы отвечают за расположение и визуальное отображение элементов и данных. К ним относятся коллекции полей: PivotFields, ColumnFields, RowFields, PageFields, DataFields. Варианты значений полей доступны через коллекции объектов PivotItems.

Универсальная возможность обращения к данным непосредственно в кэш (объект **PivotCache**) почему-то не предусмотрена разработчиками Excel. Логика при этом не совсем понятна. Как уже отмечалось, данные кэша хранятся отдельно и их даже можно увидеть в файле формата xlsx, если открыть этот файл как zip-архив. В зависимости от типа источника данных можно попытаться ис-

пользовать свойство **SourceData** (для сводных таблиц на основе диапазона) или **Recordset** (для источников типа «запрос к базе данных»).

Вычисляемые поля и объекты сводной таблицы (**CalculatedFields**, **CalculatedItems**) имеют собственный механизм расчетов и дерево зависимостей формул, не относящееся к формулам рабочего листа Excel. На практике мы рекомендуем по возможности избегать большого количества вычисляемых полей в сводных таблицах, так как это приводит к существенному замедлению расчетов. Для источников данных в виде диапазонов ячеек часто можно просто добавить столбец с обычной формулой в исходные данные, а для запросов к базам данных — добавить вычисления непосредственно в текст SQL-запроса.

Виды источников данных. Глобально можно разделить источники данных на 3 типа:

1. диапазоны ячеек;
2. запросы к базе данных;
3. OLAP-кубы и PowerPivot2010 как один из вариантов реализации OLAP-механизма.

Диапазоны. Стандартный интерфейс Excel не позволяет строить сводный отчет на основе нескольких диапазонов ячеек. Причина такого ограничения не очень понятна. Есть подозрение, что разработчики просто не могут предложить интуитивно понятный интерфейс пользователя для решения данной задачи. Техническая реализация задачи не выглядит слишком сложной — требуется просто заполнить кэш данных.

Запросы к базе данных. Запросы к базе данных могут быть реализованы с использованием различных технических механизмов: Microsoft Query, ADO, ODBC. Независимо от интерфейса доступа к данным объединяющим фактором этого варианта работы является заполнение кэша сводной таблицы непосредственно из внешнего источника. При дальнейшей работе со сводной таблицей запрос может быть выполнен повторно, после чего данные будут заново перенесены в кэш. Этот метод позволяет анализировать данные из внешних источников (учетных систем) в реальном времени. При разрыве связи с источником данных анализ может производиться на последних данных, попавших в кэш.

OLAP-кубы. OLAP-куб предоставляет промежуточный уровень подготовки информации для многомерного анализа в сводных таблицах. Куб хранит информацию о доступных типах полей (измерение или данные), иерархические зависимости полей, агрегированные значения (промежуточные итоги) и другие вычисляемые элементы. Главным преимуществом использования кубов перед прямыми запросами в базу данных является высокая производительность, так как данные перемещаются и агрегируются в промежуточном хранилище. Очевиден и недостаток данного метода — данные OLAP-куба могут содержать неактуальную информацию, что зависит от настроек хранилища.

До версии Office 2007 простой OLAP-куб можно было подготовить при помощи Microsoft Query, но в последних версиях эту возможность по непонятным причинам отключили. Разработчики настоятельно рекомендуют использовать SQL Server Analysis Service для создания и настройки OLAP-кубов. Реко-

мендация полезная, но, во-первых, этот сервис входит в состав только платных версий SQL Server, а во-вторых, требует серьезного изучения как интерфейса, так и языка обработки MDX-запросов.

Обратите внимание на изменения в интерфейсе сводной таблицы при использовании OLAP-куба в качестве источника данных:

- наличие иерархических измерений, нет возможности поменять родительский и дочерний элемент местами;
- недопустимо перемещение измерений в область данных и наоборот;
- промежуточные итоги отображаются для всех элементов, а не по текущему фильтру группы.

PowerPivot. Для Excel 2010 доступна специальная надстройка PowerPivot, которая является, по большому счету, альтернативным механизмом реализации OLAP-кубов. При помощи PowerPivot можно обрабатывать миллионы записей различных информационных файлов и баз данных с огромной производительностью. При этом интерфейс пользователя для конечного анализа данных реализован в Excel 2010.

4. Построение и форматирование диаграмм. Свойства и методы объекта Char

Трудно перечислить все достоинства Excel. Конечно, на первом месте стоит возможность работы с данными, предоставляемая электронной таблицей Excel, его машиной вычислений и мощной библиотекой встроенных функций. Но на второе место по важности, несомненно, претендуют возможности Excel по графическому отображению данных. Для этого используются диаграммы Excel, позволяющие отображать одни и те же данные в самых различных форматах в зависимости от потребностей пользователя. Excel предоставляет самые широкие возможности для варьирования формой представления данных. Диаграммы могут быть плоскими и объемными, двумерными и трехмерными, круговые и цилиндрические, данные можно отображать в виде графиков или гистограмм различного типа. На одной диаграмме может отображаться несколько групп, где каждая группа содержит один или несколько рядов данных (серий), отображаемых в одном формате. Диаграммы выделяются цветом, имеют оси, сопровождаются заголовком, подписями, легендой.

Видов диаграмм в Excel великое множество. По типу диаграммы делятся на стандартные и нестандартные или настраиваемые (Custom). Стандартных типов — 14, но каждый из них имеет до десяти форматов, так что в общей сложности их около 100. Настраиваемые типы, в свою очередь, разделяются на встроенные и определенные пользователем. С их помощью на одной диаграмме можно задать комбинацию нескольких стандартных типов.

Диаграммы предназначены для отображения графиков функций и гистограмм. Если по точкам строится график функции Y=F(X), то, как известно, необходимо задать два множества — аргументов и значений. Множество значений называется в Excel рядом данных, а аргументы называются категориями. Соответственно ось X (аргументов) называется осью категорий, а ось Y — осью значений. При построении графика можно опустить задание аргументов и тогда

по умолчанию за их значения принимается начало натурального ряда чисел — 1, 2, 3 и т. д. Очень часто на одной диаграмме отображается несколько графиков. Если эти графики отражают некоторую тенденцию, например динамику объема продаж во времени, то лучше использовать трехмерную диаграмму, в которой появляется третья ось, чаще всего это ось времени. В Excel она называется осью рядов данных, так как фиксирует изменения ряда данных.

Одно из основных применений Excel — это *анализ данных*. А для анализа данных часто удобно использовать диаграммы с их специальными возможностями, такими как тренды. С диаграммами в Excel существует некоторая терминологическая путаница. То, что на графическом интерфейсе русского Excel называется диаграммой, по-английски называется графиком (Chart), и ему соответствует объект Chart. В объектной модели Excel предусмотрен также и объект Diagram, но он представляет скорее схему отношений. Под диаграммой будем подразумевать график.

Коллекция Charts. Эта коллекция является частью коллекции Sheets, — ее элементами являются объекты класса Chart, представляющие специальный тип листов рабочей книги — листы, содержащие диаграммы. Всякий раз, когда руками или программно создается диаграмма, ее можно либо встроить в рабочий лист, либо расположить на отдельном листе. Если предпочтение отдается второму варианту, то в коллекции Charts появляется новый элемент.

Коллекция Charts содержит тот же набор свойств и методов, что и коллекция WorkSheets. Она содержит типичный набор свойств: Application, Count, Creator, Parent, Item. Кроме этих свойств, имеется менее типичное для коллекций свойство Visible и два свойства VpageBreaks и HpageBreaks, возвращающие одноименные коллекции, элементы которых задают вертикальное и горизонтальное деление листа на страницы, которые могут быть распечатаны. Все методы, которые есть у коллекции WorkSheets, есть и у коллекции Charts. Вот эти методы: Add, Copy, Delete, Move, PrintOut, PrintPreview, Select (отсутствует только один метод FillAcrossSheets, копирующий диапазон ячеек рабочего листа).

Диаграммы в Excel создаются при помощи объекта Chart. Лучше всего вначале этот объект объявить: **Dim oChart As Chart**.

Создание диаграммы при помощи приема — вызова метода Add() коллекции Charts:

Set oChart = ActiveWorkbook.Charts.Add(, ActiveSheet)

В принципе диаграмма уже создана, но поскольку никакие ее свойства не определены, она выглядит просто как пустой лист. Чтобы она обрела содержание, необходимо выполнить еще несколько действий.

Первое (и единственное обязательное действие) — определить источник данных для диаграммы, для чего предназначен метод **SetSourceData().** В качестве источника может выступать только объект Range. Второй параметр (необязательный) определяет, в каком порядке считывать данные — сначала по столбцам или сначала по строкам. Например, в нашем случае это может выглядеть так: **oChart.SetSourceData(Sheets("Лист1").Range("A1:A10")).**

В принципе, если запустить созданный код на выполнение, то диаграмма уже будет создана. Для всех остальных параметров будут приняты значения по умолчанию. Для определения типа диаграммы (по умолчанию «обычная гистограмма») используется свойство ChartType, для которой разработчиками предусмотрено целых 73 значения. Например, чтобы преобразовать диаграмму в обычный график, можно использовать код вида:

oChart.ChartType = *xlLineMarkers*

Еще одна очень распространенная задача — добавить дополнительные ряды на диаграмму. Для этой цели необходимо создать и получить ссылку на объект Series — ряд, а потом для ряда определить свойство Values (ему передается в качестве значения объект Range):

Dim oSeries As Series
Set oSeries = oChart.SeriesCollection.NewSeries
oSeries.Values = Worksheets(1).Range("B1:B10")

Пользователи часто говорят, что им необходимо создавать диаграммы не на отдельном листе, а на том же листе, на котором расположены данные. По умолчанию диаграмма создается в оперативной памяти и помещается на отдельный лист. Если нам необходимо поместить ее на уже существующий лист, то в этом случае ее вначале надо создать на отдельном листе, а затем переместить при помощи метода **Location.** Отдельный лист, созданный для диаграммы, при этом автоматически исчезнет: *oChart.Location xlLocationAsObject, "Лист1".*

Обратите внимание, что метод Location принимает в качестве первого параметра одну из констант (xlLocationAsNewSheet — переместить на специально создаваемый новый лист, xlLocationAsObject — переместить на объект, то есть лист), а в качестве второго — не объект листа, как можно было бы подумать, а обязательно его имя. Если код предполагается использовать и в русской, и в английской версии Excel, то предпочтительнее получить имя листа программным образом.

Большая неприятность, связанная с методом Location, заключается в том, что после перемещения диаграммы внутрь листа объектная ссылка на эту диаграмму теряется и надо находить объект этой диаграммы заново. При попытке повторного обращения к объекту Chart выдается сообщение «Automation Error». Поэтому лучше всего вызов метода Location помещать в самый конец кода, посвященного диаграмме. В противном случае нам придется разыскивать созданную нами диаграмму и заново получать на нее объектную ссылку, например, так:

Dim oSeries As Series
Set oSeries = Worksheets(1).ChartObjects(1).Chart.SeriesCollection.NewSeries
oSeries.Values = Worksheets(1).Range("B1:B10")

Важные свойства объектов Chart:

- свойство ChartArea — это свойство возвращает одноименный объект ChartArea, который представляет собой область, занимаемую диаграммой, и используется для настройки внешнего вида диаграммы (свойства Font, Interior и т. п.). Если необходимо настроить внешний вид не всей диаграммы, а той ее части, которая используется непосредственно для вывода графика, использует-

ся схожее свойство PlotArea. По умолчанию диаграмма размещается прямо по центру листа. Если необходимо ее переместить в точно определенное место листа, используются знакомые нам свойства Top, Height, Left и Width для объекта ChartArea.

- свойство ChartTitle возвращает одноименный объект, при помощи которого можно настроить заголовок диаграммы (с такими свойствами, как Text, Font, Border и т. п.);
- ChartType — важнейшее свойство, про которое мы уже говорили. Определяет тип диаграммы;
- HasDataTable — если установить это свойство в True, то в нижней части диаграммы (по умолчанию) появится таблица с числами, на основе которых была создана диаграмма. Одновременно будет создан программный объект DataTable, при помощи которого можно настроить представление этой таблицы. Схожим образом действуют свойства HasLegend, HasPivotFields и HasTitle;
- Name — это свойство позволяет настроить имя диаграммы (как название вкладки в Excel). По умолчанию диаграммы называются последовательно «Диаграмма1», «Диаграмма2» и т. п.;
- SizeWithWindow — если поставить значение этого свойства в True (по умолчанию False), то размер диаграммы будет подогнан таким образом, чтобы точно соответствовать размеру листа;
- Tab — свойство, о котором мало кто подозревает. Оно позволяет настроить при помощи одноименного объекта внешний вид вкладки в книге Excel для диаграммы (или просто листа). Например, чтобы пометить вкладку зеленым, можно воспользоваться кодом oChart.Tab.Color = RGB(0, 255, 0);
- Visible — возможность спрятать диаграмму без ее удаления.

Остальные свойства в основном относятся к настройке отображения трехмерных диаграмм и к защите диаграммы от изменения пользователем. Floor, Walls и Corners объекты используются при работе с трехмерными диаграммами.

Главные методы объекта Chart:

- метод Add() — создание диаграммы;
- метод Activate() используется очень часто. Он позволяет сделать диаграмму активной (то есть просто перейти на нее);
- метод ApplyCustomType() позволяет поместить создать диаграмму своего собственно пользовательского типа (для этого необходимо вначале создать шаблон для этого типа и поместить его в галерею);
- метод ApplyDataLabels() позволяет поместить на диаграмму метки для размещенных на ней данных. Этот метод принимает множество параметров, которые позволяют настроить отображение данных меток (показывать или не показывать значения и т. п.);
- метод Axes() возвращает объект, представляющий оси диаграммы. Затем этот объект можно использовать для настройки данных осей;
- ChartWizard() — этот метод позволяет быстро переформатировать диаграмму, как будто бы прошли на графическом экране при помощи мастера по-

строения диаграмм и передали ему значения. Позволяет при помощи одной строки кода добиться того, что другими способами потребовало бы минимум несколько строк;

- Copy() — позволяет скопировать диаграмму в другое место книги (например, чтобы создать новую диаграмму, использовав в качестве основы существующую). Для переноса существующей диаграммы в другое место можно воспользоваться методами Location() или Move();
- CopyPicture() — замечательный метод, который позволяет поместить диаграмму в буфер обмена как изображение. Затем это изображение можно вставить, например, в документ Word или в любое другое место. Другой вариант — воспользоваться методом Export(), который позволяет создать рисунок, представляющий диаграмму, в виде файла на диске.
- Delete() — просто удаляет диаграмму;
- Evaluate() — как обычно, этот метод позволяет найти нужную диаграмму в книге по ее имени;
- PrintOut() — возможность отправить диаграмму на печать. Этот метод принимает множество параметров, которые позволяют настроить такой вывод;
- Refresh() — возможность обновить диаграмму, если изменились данные, на основе которых она строилась;
- Select() — возможность выделить диаграмму (равносильно щелчку по ней мышью). Deselect() — снятие выделения (равносильно нажатию на <Esc>);
- SetBackgroundPicture() — возможность «подложить» под диаграмму фоновый рисунок. Конечно, он должен быть не очень ярким;
- SetSourceData() — важнейший метод, который позволяет определить данные, на основе которых строится диаграмма. Про него мы уже говорили.

Для объекта Chart предусмотрены также события «на все случаи жизни» — реакция на щелчки мышью, на выделение/снятие выделения, активизацию, пересчет данных, изменение размера и т.п., однако используются такие события на практике нечасто.

Объекты Excel.CellFormat, Excel.ListObject, Excel.Validation, Excel.Watch

Объект CellFormat позволяет настраивать форматирование ячеек Excel — шрифт, рамки, цветовое оформление и т. п. При этом можно использовать этот объект для копирования оформления с других ячеек, для поиска и замены по оформлению и т. п. Условное форматирование можно настроить при помощи объекта FormatCondition.

ListObject предназначен для работы со списками — наборами взаимосвязанных данных (например, списками сотрудников).

Объект Validation позволяет настроить проверку вводимых пользователем данных.

Объект Watch позволяет настроить контрольное значение для формул (при помощи контрольного значения — меню Сервис -> Макрос -> Показать контрольное значение — можно отслеживать значения тех формул, которые находятся за пределами экрана).

Как получить объект Chart. Объект Chart задает диаграмму, расположенную на листе рабочей книги или на отдельном листе диаграммы.

Диаграмма встроена в рабочий лист. На рабочем листе может находиться несколько встроенных диаграмм. Коллекция ChartObjects задает совокупность объектов-контейнеров, содержащих эти диаграммы. Зная индекс или имя диаграммы, можно получить доступ к нужному элементу этой коллекции, а свойство Chart позволяет получить объект, задающий диаграмму. Так что вызов **ThisWorkbook.Worksheet(3).ChartObjects(1).Chart** вернет объект Chart, задающий первую диаграмму, расположенную на третьем рабочем листе текущей рабочей книги.

Для выделения объекта Chart используем метод Select, но напрямую метод не вызывается, так что сначала активизируем контейнер, а затем — область диаграммы:

```
ThisWorkbook.Worksheets(3).ChartObjects(1).Activate
ActiveChart.ChartArea.Select
```

В этом случае приходится часто использовать оба объекта — ChartObjects и Chart для решения возникающих задач.

Диаграмма расположена на отдельном листе диаграммы. Специальные листы диаграмм рабочей книги составляют коллекцию Charts.

Коллекция представляет собой совокупность специальных листов рабочей книги, элементы этой коллекции являются не столько листами, сколько объектами Chart, задающими диаграммы. Эта ситуация не является типичной для коллекций. Связано это с тем, что на таком листе располагается только один объект Chart.

Активная диаграмма. Добраться до объекта Chart, задающего активную диаграмму, можно с использованием свойства ActiveChart, как показано в только что приведенных примерах. Активизация листа диаграммы приводит и к активизации самой диаграммы. При работе с такими диаграммами можно использовать с тем же успехом и свойство ActiveSheet.

Что же касается встроенных диаграмм, то там необходимо предварительно активизировать соответствующий контейнер.

Несколько диаграмм на одном листе диаграммы. Excel позволяет указать один и тот же лист диаграммы как место расположения нескольких диаграмм. В этом случае новая диаграмма накладывается на область, отведенную первой диаграмме. Руками можно работать с несколькими диаграммами на таком листе, например изменяя их параметры. Но программная работа возможна только с одним объектом, определяющим первую диаграмму. Более того, если, используя MacroRecorder, записать макрос, повторяющий работу руками с несколькими диаграммами одного листа, то этот макрос не будет корректно выполняться при его запуске. Суммируя это, рекомендую в своей работе придерживаться стратегии — одна диаграмма на листе диаграмм. Это обеспечит корректную программную работу с такими листами.

Источники данных и структура объекта Chart

Диаграмма Excel, предоставляя пользователям широкий спектр возможностей по отображению данных, не может быть просто устроена, — она имеет

достаточно сложную внутреннюю структуру. Соответственно такую же сложную структуру имеет и объект Chart. Прежде чем переходить к деталям, давайте рассмотрим общую картину. В диаграмме можно выделить:

1. общую область, занятую диаграммой. Понятно, что можно менять размеры, внешний вид и другие характеристики общей области;
2. область данных, в которой и строится собственно диаграмма;
3. источник данных и данные этого источника — эти объекты играют центральную роль;
4. заголовок диаграммы и другие подписи;
5. оси.;
6. легенду.

Для того чтобы можно было построить диаграмму, необходимо иметь по крайней мере, один ряд данных. Напомню специальные термины, применяемые при построении диаграмм. Ось X называется осью категорий, а значения, откладываемые на этой оси, называются категориями. Значения отображаемых в диаграмме функций и гистограмм составляют ряд данных. Ряд данных, представляющий последовательность числовых значений, является одним из центральных понятий в построении диаграмм. Этому понятию соответствует объект Series. Поскольку при построении диаграммы, как правило, используется несколько рядов данных, то важную роль играет понятие совокупности рядов данных, которой с объектной точки зрения соответствует коллекция — объект SeriesCollection.

Контрольные вопросы

1. Какие свойства и методы имеет объект PivotTable?
2. Как внутренне организован интерфейс сводных таблиц?
3. Опишите алгоритм программного создания диаграмм в Excel.
4. Какие существуют способы расположения диаграмм из коллекции Chart?

Тема 3. Веб-программирование
Лекция 15. Клиентская часть приложения

План:

1. Характеристика типовых задач, решаемых клиентской частью приложений.
2. Функциональные возможности клиентской части.
3. Обзор инструментальных средств (ИС) разработки программ, выполняющихся на стороне клиента.

1. Характеристика типовых задач, решаемых клиентской частью приложений

Прикладное программирование для Web начиналось с обработки запросов пользователя и динамической генерации страниц на стороне сервера. Эта же тенденция получила развитие и в языках программирования вставок в HTML-документы. Затем появились языки программирования элементов HTML-документов на стороне клиента.

В своих первых попытках повысить интерактивность HTML веб-страниц разработчики обратились к сценариям (scripting), добавляя функциональность путем комбинирования языка программирования с HTML. В результате зачастую получается странный гибрид кода и тегов, что вынуждает разработчиков вернуться к текстовым редакторам. Был введен специальный тег <SCRIPT>, который определяет раздел кода на веб-странице.

VBScript представляет собой язык описания сценариев, в основе которого лежит Visual Basic for Applications (VBA) — популярный язык, применяемый, например, в Microsoft Office. VBScript — это не полная версия VBA, а скорее его подмножество, которое сохраняет многие ключевые возможности VBA, но в то же время не реализует те, которые сделали бы его чересчур громоздким и небезопасным. Как и его старший брат, VBA, язык VBScript управляется событиями. Это означает, что написанный вами код выполняется в ответ на событие (event), возникшее в результате взаимодействия пользователя с графическим интерфейсом (graphical user interface, GUI). В нашем случае GUI представляет собой веб-страницу.

Хотя сценарии и представляют собой шаг вперед в развитии интерактивности, у них есть и определенные ограничения. Например, не все программы просмотра распознают и обрабатывают сценарии, а те, которые это делают, используют разные языки. Главным образом это касается Netscape Navigator, который не распознает VBScript, однако работает с JavaScript — языком описания сценариев, первоначально разработанным для Netscape Navigator. По функциональности JavaScript очень похож на VBScript, но по синтаксису эти языки сильно различаются. В отличие от VBScript, JavaScript не поддерживает концепцию процедур обработки событий. Все процедуры в JavaScript — это функции, вызываемые при помощи атрибутов событий, расположенных в HTML-теге.

Плохо не только то, что поддержка сценариев различается в разных программах просмотра, но и то, что сценарии не обеспечивают всей развитой функциональности, которой ожидают от языка программисты. Сценарии предоставляют подмножество тех языковых возможностей, которые обычно используют разработчики: базовые структуры и операторы — циклы и ветвления. По сути, сценарии годятся только для проверки корректности введенных данных перед отсылкой формы на сервер.

Как только к возможностям программ просмотра добавляются сценарии, возрастает сложность клиентской платформы. Очевидно также, что раз отсутствует универсальный язык описания сценариев, то теряются все разрекламированные преимущества платформенной независимости Web. Для многих веб-мастеров и разработчиков постоянная война между программами просмотра за преобладание на рынке создает необходимость поддерживать две версии веб-узла: для Microsoft Internet Explorer и для Netscape Navigator.

Многие мощные веб-сайты сегодня построены по структуре CGI, и вы можете фактически многое с использованием этого. Однако веб-сайты, построенные на CGI программах, могут часто становиться слишком трудными в поддержке, а также проблемой является время ответа. Ответ CGI программы зави-

сит от того, как много данных должно быть послано, так как это загружает и сервер, и Интернет. Первые разработчики Web не предвидели, как часто пропускная способность будет исчерпана для такого рода приложений, разрабатываемых людьми. Например, любой сорт динамической графики часто нельзя применять последовательно, так как должен быть создан GIF-файл и перемещен от сервера клиенту для каждой версии графики. Например, вы нажимаете кнопку подтверждения на странице; данные отправляются назад на сервер; сервер запускает CGI-программу, которая обнаруживает ошибку, формирует HTML-страницу, информирующую вас об ошибке, а затем посылает страницу вам; вы должны вернуться на предыдущую страницу и попробовать вновь. Это не только медленно, это не элегантно.

Решение — это программирование клиентской стороны. Большинство машин, запускающих веб-браузеры, являются достаточно мощными и способны обширной работы, а с оригинальным подходом статического HTLM они простаивают, просто ожидая, когда сервер передаст следующую страницу. Программирование стороны клиента означает, что веб-браузер используется для выполнения какой-нибудь работы, которую он может выполнить, а результат для пользователя — это скоростная и более интерактивная работа на веб-сайте.

2. Функциональные возможности клиентской части

Компоненты ActiveX. По мере совершенствования технологии программ просмотра зависимость от платформы усилилась — она была привнесена компонентами ActiveX — технологией, основанной на COM — модели многокомпонентных объектов Microsoft (Component Object Model). Компоненты ActiveX варьируются от причудливых элементов управления, таких как движки (spinners), до невизуальных компонентов, обеспечивающих доступ к базам данных или электронной почте. Подобные компоненты делают страницы в Internet Explorer более функциональными и привлекательными, но практически бесполезными в среде, не поддерживающей ActiveX, например в Netscape Navigator.

Компонент ActiveX добавляется в веб-страницу при помощи тега <OBJECT>, однозначно определяющего компонент для программы просмотра. Когда Internet Explorer обнаруживает тег <OBJECT>, он обращается к реестру и ищет там GUID, совпадающий со значением атрибута CLASSID. Когда такой GUID найден, из реестра выбирается дополнительная информация, позволяющая отыскать файл, который соответствует данному элементу управления ActiveX.

Если нужный элемент управления ActiveX на клиентской машине отсутствует, Internet Explorer обращается к атрибуту CODEBASE за информацией о том, где находится этот элемент на сервере. Следуя этой информации, файлы данного элемента управления загружаются с сервера, и элемент устанавливается на клиентской машине. Теперь Internet Explorer может свободно работать с ним.

Доступ к компонентам ActiveX посредством тега <OBJECT> не ограничивается элементами управления. Этот тег может активизировать произвольный

компонент ActiveX, в том числе и те компоненты, которые можно написать на языках Visual Basic, C++ и Microsoft FoxPro. В сущности, вы легко можете расширить функциональность, доступную клиенту, написав собственные компоненты ActiveX и загрузив их в программу просмотра. Следует, правда, помнить, что Internet Explorer по умолчанию не загружает и не выполняет компоненты без цифровой подписи разработчика. Окончательное обеспечение компонента данными происходит через тег <PARAM>, имеющий атрибуты NAME и VALUE, при помощи которых задаются начальные значения свойств данного компонента, когда он впервые создается на веб-странице. После того как начальные значения установлены, значения свойств легко изменить во время выполнения из текста сценария.

Документы ActiveX. Visual Basic, начиная с версии 5.0, позволяет, помимо элементов управления ActiveX, создавать документы ActiveX. Документы ActiveX представляют собой программные объекты, которые могут загружаться и работать внутри ActiveX-контейнера, такого как Internet Explorer. Документы ActiveX позволяют разработчикам на Visual Basic немедленно применить свой опыт работы на Visual Basic для создания приложений для Интернета. Что самое существенное, документы ActiveX предоставляют доступ к большей части ключевых возможностей Visual Basic в загружаемом формате.

Plug-ins. Один из наиболее значимых шагов вперед в программировании на стороне клиента — это разработка встраиваемых модулей. Это способ программирования с добавлением браузеру новой функциональности при скачивании части кода, которая встраивает себя в соответствующее место браузера. Он говорит браузеру «с этого момента ты можешь выполнять новые действия». (Вам необходимо загрузить встраиваемый модуль лишь однажды.) Некоторые быстрые и мощные особенности добавляются браузеру через встраиваемые модули, но написание этих модулей — это нетривиальная задача, и вам не нужно это делать никогда, как часть процесса построения обычного сайта. Значение встраиваемых модулей для программирования стороны клиента в том, что он позволяет программисту-эксперту разработать новый язык и добавить этот язык в браузер, не обращаясь к разработчику браузера. Таким образом, встраиваемый модуль обеспечивает «заднюю дверь», которая позволяет создание нового языка программирования стороны клиента (хотя не все языки реализуются как встраиваемые модули).

3. Обзор инструментальных средств (ИС) разработки программ, выполняющихся на стороне клиента

Минимальным набором инструментов веб-программиста является текстовый редактор и браузер, под который оптимизируется сайт. При использовании серверных сценариев требуется и веб-сервер, желательно такой же, как и у заказчика. Избегайте отладки сценариев на сервере клиента — ваши ошибки могут привести к его зависанию, и не всегда у вас есть права и возможность его перезагрузки! Лучше всего установить веб-сервер на рабочей станции или домашнем компьютере и подключить к нему требуемый интерпретатор. Следует, правда, иметь в виду, что функциональные возможности веб-серверов

правда, иметь в виду, что функциональные возможности веб-серверов и интерпретаторов под разными ОС (Windows и UNIX) зачастую различны.

Что касается редактора, то можно использовать как стандартный «Блокнот» или встроенный редактор файлового менеджера FAR (желательно с плагином Colorer для подсветки тегов и операторов), так и какой-либо специализированный WYSIWYG (What You See Is What You Get) HTML-редактор. Однако среди их многообразия практически отсутствует такой, который поддерживает ОДНОВРЕМЕННО ВСЕ серверные языки сценариев, хотя поддержка клиентских языков, как правило, присутствует. Кроме того, зачастую эти редакторы преобразуют русские буквы в их коды, что существенно затрудняет последующее редактирование. Несомненно, самым мощным, хотя и громоздким средством программирования на ASP является Microsoft Visual InterDev, входящий в комплект Microsoft Visual Studio. Будучи интегрирован со справочной системой MSDN, он позволяет быстро получить справку по любому оператору, функции или объекту. Возможен также предварительный просмотр, как в окне редактора, так и в браузере по умолчанию, а также пошаговая отладка.

Что касается технологий программирования, то все современные языки сценариев поддерживают как классическую процедурную, так и объектно-ориентированную, хотя и в различной степени.

Языки сценариев. Встраиваемые модули стали результатом взрывного распространения языков сценария. У языков сценария вы встраиваете исходный код для вашей программы стороны клиента прямо в HTML-страницу, а встраиваемый модуль, который интерпретирует этот язык, автоматически активируется при отображении HTML-страницы. Языки сценариев достаточно легки для понимания и, потому что они являются простым текстом, как часть HTML страницы, они загружаются очень быстро. Минус в том, что ваш код открыт каждому для просмотра (и воровства). Обычно, однако, вы не делаете удивительно сложные вещи с помощью языков сценария, так что это не встречает особых трудностей.

Это говорит о том, что языки сценариев, используемые внутри веб-просмоторщиков, реально предназначены для решения специфических проблем, в первую очередь создание богатого и более интерактивного графического пользователя (GUI). Однако языки сценариев могут решить 80% проблем, возникающих при программировании на стороне клиента. Ваши проблемы могут полностью попадать в эти 80%. Языки сценариев могут предоставить простоту и быстроту разработки, поэтому вам, вероятно, нужно рассмотреть язык сценариев, прежде чем рассматривать более сложные решения, такие как Java или ActiveX.

Наиболее часто обсуждаемые языки сценариев для браузеров — это JavaScript (который не делает ничего, что может Java; его название — это просто способ отобрать часть рынка Java), VBScript (который выглядит как Visual Basic) и Tcl/Tk, который пришел из популярного кросс-платформенного языка GUI-разработки. Есть и другие, нередко более развитые.

JavaScript, вероятно, наиболее часто поддерживается. Он встроен и в Netscape Navigator и в Microsoft Internet Explorer (IE). В дополнение, вероятно, о JavaScript существует больше книг, чем о других языках браузера, а некоторые инструменты автоматически создают страницы, используя JavaScript. Однако, если вы уже владеете Visual Basic или Tcl/Tk, для вас более продуктивным станет использование этих языков сценариев, чем изучение нового.

Java. Если языки сценариев могут решить 80% проблем программирования стороны клиента, что можно сказать об остальных 20% «действительно сложных задач»? Наиболее популярным решением сегодня является Java. Не только потому, что это мощный язык программирования, построенный для безопасности, кросс-платформенности и интернациональности, но Java постоянно расширяется, чтобы обеспечить такие особенности языка и библиотеки, которые элегантно решают проблемы, которые сложны для традиционных языков программирования, такие как многопоточность, доступ к базам данных, сетевое программирование и распределенные вычисления. Java обеспечивает программирование на стороне клиента через апплет.

Апплет — это мини-программа, которая запускается только под управлением веб-браузера. Апплет скачивается автоматически как часть веб-странички (как, например, графика скачивается автоматически). Когда активируется апплет, то выполняется программа. Пользователи получают последнюю версию клиентского программного обеспечения без ошибок и без сложных переинсталяций. Поэтому в том способе, который разработан в Java, программисту необходимо создать только одну программу, а эта программа автоматически работает на всех компьютерах, которые имеют браузеры со встроенным Java-интерпретатором. (Это благополучно включают большинство машин.) Так как Java — полноценный язык программирования, вы можете выполнить столько работы, сколько может клиент как перед, так и после выполнения запроса на сервер. Например, вы не хотите посылать запрос через Интернет, чтобы узнать, что данные или какой-то параметр неверны, а ваш клиентский компьютер быстро выполнит работу по проверке данных, вместо ожидания от сервера проверки и передачи графического изображения к вам обратно. Вы не только получаете преимущество в скорости и отзывчивости, но это снизит сетевой трафик и загрузку сервера, предотвращая от замедления весь Интернет.

Java-апплеты предпочтительнее других программ-сценариев, так как они имеют компилированную форму, так что исходный код не доступен для клиента. С другой стороны, Java-апплет может быть декомпилирован без особых затруднений, но прятанье вашего кода чаще всего не самая важная задача. Два других фактора могут оказаться важнее, как вы увидите далее в этой книге, компилированные Java-апплеты могут включать много модулей и занимать много отправок (обращений) сервера для скачивания. Программы-сценарии просто интерпретируются на веб-странице как часть ее текста (и обычно маленькие и снижают обращения к серверу). Это важно для отзывчивости вашего веб-сайта. Java — это не простой язык для изучения. Если вы программируете на Visual Basic, переход к VBScript будет для вас более быстрым решением и, вероятно, решит большинство типичных проблем клиент – сервер, которые вы

можете с трудом преодолеть, изучая Java. Если вы имеете опыт в языках сценария, вам сначала полезнее будет взглянуть на JavaScript или VBScript, прежде чем переходить на Java, так как они могут легко удовлетворить вашим требованиям и ваша работа будет более продуктивной.

ActiveX. Для некоторых проблем соперником Java является Microsoft ActiveX, хотя он имеет полностью другой подход. ActiveX изначально было только решением для Windows, хотя сейчас это становится кросс-платформенной разработкой независимого консорциума. Действительно, ActiveX говорит: «если ваша программа подключается к окружению, то она может быть перенесена на веб-страницу и работать под управлением браузера, который поддерживает ActiveX». (IE напрямую поддерживает ActiveX, а Netscape использует подключаемые модули.) Таким образом, ActiveX не принуждает вас к специальному языку. Если, например, если вы имеете опыт программирования в Windows на таких языках, как C++, Visual Basic или Delphi от Borland, то можете создать компонент ActiveX почти без изменений вашего знания языка. ActiveX также обеспечивает путь для использования правильного кода на вашей веб-странице.

Программирование с ActiveX — это как программирование Windows — вы можете делать все, что захотите.

Dynamic HTML. В версии Internet Explorer 4.0 Microsoft добавила к клиентской функциональности еще одну особенность — Dynamic (динамический) HTML, который позволяет посредством сценариев программно изменять теги. Это необычайно мощное средство. В Dynamic HTML определяется набор событий, которые можно ассоциировать с тегами HTML. Это расширяет парадигму VBScript управляемости событиями на все элементы веб-страницы — теги HTML, элементы управления ActiveX; даже программа просмотра сама по себе обладает определенными событиями. Dynamic HTML заметно увеличивает мощность веб-клиента и его интерактивность, причем не только за счет динамического стиля манипулирования, но и другими средствами. Так, он умеет располагать элементы на веб-странице. Вы можете, например, изменить изображение, просто изменив атрибуты тега <IMG>. Для изменения содержимого страницы вы можете также добавлять или удалять теги. И наконец, Internet Explorer 4.0 поддерживает привязку данных (data binding) к полям формы. Это означает, что данные из базы данных на сервере могут быть напрямую связаны с полем формы в программе просмотра Web, и тем самым будут мгновенно редактироваться и обновляться. Все это делает Dynamic HTML мощным орудием, достойным вашего внимания.

Прикладное программирование для Web начиналось с SGML, породившего HTML, а когда возможности последнего были исчерпаны, произошел частичный откат к SGML. В результате появился новый язык — XML (eXtensible Markup Language) или, более точно, стек спецификаций языков разметки различного назначения, которые базируются на общих синтаксических правилах. Возникновение XML было обусловлено, в частности, стремлением разработчиков унифицировать форму хранения документов для различных носителей информации. Не последнюю роль сыграла и необходимость развития и унифика-

ции формата хранения метаданных, а также преобразования документов в процессе их отображения и просмотра.

Параллельно с процессом развития формального статического описания содержания документа развивались и способы его изменения. Первоначально они были обозначены в JavaScript, затем язык программирования Java позволил размещать внутри документа и видоизменять информацию любого типа. Появление VBScript и JScript означало, что Microsoft движется в том же направлении. Постепенно технология Java опустилась до уровня средств разработки приложений Web, а сценарное направление оформилось в концепцию DHTML (Dynamic HTML).

И XML, и DHTML, и Java в конечном итоге замкнулись на модель данных Web, множество страниц Web, которые, с точки зрения разработчиков поисковых языков XML, представляют собой сплошной поток разнотипных данных. Один документ (страница) — это подмножество всех документов (страниц) Web. Модель данных Web определяют в виде графа — «лес» из деревьев. Чтобы представить предмет нашего обсуждения в более простом виде, возьмем HTML-документ и «препарируем» его с точки зрения такой графовой модели данных. Весь документ — это один большой элемент разметки HTML. При этом документ является блочным элементом, который не может пересекаться с другими документами, однако может содержать блоки, например, HEAD и BODY. В свою очередь HEAD и BODY тоже могут включать другие блоки. При этом элемент BODY в своих атрибутах способен определить свойства всего отображаемого тела документа, например цвет текста, цвет фона или, скажем, цвет гипертекстовых ссылок. Если двигаться еще дальше внутрь BODY, то с очень большой вероятностью в типичном HTML-документе можно встретить элемент разметки IMG, у которого есть свои свойства. Теперь назовем узлы графа объектами, а программам разрешим изменять свойства этих объектов. Скажем, значение атрибута SRC у объекта, который соответствует элементу разметки IMG. При этом выполнять такие изменения можно, используя метод из набора стандартных методов, общих как для сценариев языков, так и для Java. Все это образует концепцию DOM — Document Object Model. DOM — это интерфейс прикладного программирования в рамках модели данных Web или, другими словами, набор стандартных методов объектов Web. Если нужно вывести текст в тело документа, это можно сделать на любом языке программирования, который поддерживает DOM: document.write(<kuku>).

Здесь задействован стандартный метод write объекта document. Имя метода, значение, которое он возвращает, аргументы метода и их типы — все стандартизовано в DOM.

Таким образом, путь развития веб-технологии пролегает от статической HTML-разметки через скриптовые языки, Java и DHTML к спецификациям XML и DOM.

Модель объектов IE. Динамический HTML. Концепция динамического HTML (DHTML) интерпретируется разработчиками основных браузеров поразному. Для Netscape это тройка: JavaScript, загружаемые шрифты и стили.

При этом предполагается расширение набора элементов разметки за счет элементов разметки стилей и элемента LAYER.

Для Microsoft DHTML — это JScript, расширяющий возможности JavaScript. Он позволяет программировать стили, изменять содержимое документа без его перезагрузки и использовать другую схему обработки событий, которые, в свою очередь, жестко не привязываются к документу.

Стили создают отдельную иерархию объектов, причем очень многое зависит от типа селектора, который описывается стилем. Если это селектор класса элементов разметки, то это одна ветка дерева объектов документа, если это селектор произвольного класса, то это другая ветка дерева объектов документа, если это селектор идентификатор объекта, то это третья ветка объектов документа.

Стоит отметить, что для использования идентификатора у элемента разметки в JScript совсем необязательно создавать связанный с ним селектор идентификатора в описаниях стилей — управлять через него стилями все равно нельзя.

Программирование стилей естественным образом приводит к переформатированию документа, что нарушает принципы неизменности загруженного документа JavaScript. JScript позволяет поименовать любой элемент разметки документа, однако делается это не через атрибут name, а через атрибут id. При этом у поименованного таким образом объекта есть два свойства innerText и innerHTML. Свойства эти можно изменять. Первое из них позволяет изменить текст внутри элемента разметки, а второе — содержание HTML-разметки внутри элемента разметки, что дает возможность создавать новые объекты внутри документа.

Любое событие может быть связано с элементом разметки — это достигается указанием у этого элемента атрибута обработчика данного события. Другими словами, у всех элементов разметки определен весь перечень возможных событий и обработчики этих событий для каждого класса элементов. Программист через атрибут обработчика может переопределить действие, выполняемое по умолчанию.

В JScript применяется «пузырьковый» метод обработки событий. Это означает, что если для какого-то объекта происходит некоторое событие, то сначала вызывается обработчик события, связанный с этим объектом, а потом событие передается обработчику данного события для старшего объекта в иерархии объектов документа. Программист имеет возможность запретить передачу обработки события наверх.

Для поддержки совместимости со статическим HTML и JavaScript большинство обработчиков не выполняют никаких операций, хотя по умолчанию события всегда транслируются на верхний уровень.

JScript можно использовать для программирования Active Server Pages (ASP). Главное здесь — не запутаться между директивами JScript, предназначенными для интерпретации на стороне сервера, и JScript-кодом, который следует исполнять на стороне браузера.

Интерфейс прикладного программирования DOM

Объектные модели данных JavaScript, JScript и Java предшествовали появленю Document Object Model — спецификации интерфейса прикладного программирования для HTML и XML. DOM наследует многие свойства этих моделей и творчески развивает их. DOM определяет логическую структуру документа, способы доступа к его элементам и манипулирования ими. При этом термин «документ» понимается широко — это единица информационного описания, которую можно закодировать на XML. Другими словами, XML-описание рассматривается в качестве документа, а DOM определяет способы манипулирования этим описанием.

Объектная модель документа должна послужить соединительным звеном между программированием и логической структурой документа, формой его представления на носителях. Логическая структура и форма представления описываются в рамках XML. Программирование Web — это языки Java, JavaScript, JScript. Стандарт DOM позволяет формально описать правила манипулирования структурой XML-документов и их содержанием в рамках объектно-ориентированного программирования, которое реализуют перечисленные языки.

Контрольные вопросы

1. Что такое скрипт?
2. Что такое апплет?
3. Какие функциональные возможности имеет клиентская часть?
4. Перечислите инструментальные средства разработки клиентских программ.
5. Что определяет DOM?

Лекция 16. Язык сценариев JavaScript

План:

1. Основные понятия.
2. Базовые конструкции.
3. Объектная модель JavaScript.
4. Окна и динамически создаваемые документы.
5. События и объекты браузера.

1. Основные понятия

Скрипт — это «программа», которая создается в виде обычного текстового файла специального формата (сценария) и передается для обработки на заранее установленный в системе клиента интерпретатор.

JavaScript — это обьектно-ориентированный язык, предназначенный для создания приложений в Интернете, работающих как на стороне клиента, так и на стороне сервера. Поэтому он не является «полноценным» языком программирования, а ориентирован на использование возможностей той среды, в которой сценарии исполняются.

JavaScript — нетипизированный язык и типы данных, хранящиеся в переменных, могут изменяться во время работы программы.

мирования, а ориентирован на использование возможностей той среды, в которой сценарии исполняются.

JavaScript — нетипизированный язык и типы данных, хранящиеся в переменных, могут изменяться во время работы программы.

JavaScript — новый язык для составления скриптов (от *англ.* script — сценарий), разработанный фирмой Netscape.

Язык программирования JavaScript был разработан Бренданом Эйком (Brendan Eich) в Netscape Communications как язык сценариев для обозревателей Netscape Navigator, начиная с версии 2.0.

Название «JavaScript» связано с именем другого мощного языка программирования — Java, который был создан фирмой Sun Microsystems. JavaScript — это не то же, что Java. Первоначально язык сценариев, который знаем как JavaScript, именовался LiveScript. Однако после успешного дебюта языка Java компания Netscape начала сотрудничать с Sun Microsystems. В результате их совместных усилий был создан совершенно новый язык JavaScript, синтаксис и семантика которого непосредственно связаны с Java (язык апплетов).

JavaScript — клиентский язык.

JavaScript является интерпретируемым языком программирования, причем итерпретатором выступает программа браузера. Веб-обозреватель, работающий на компьютере-клиенте, обеспечивает среду, в которой JavaScript имеет доступ к объектам, которые представляют собой окна, меню, диалоги, текстовые области, фреймы, куки и ввод-вывод в веб-страницу.

Синтаксис JavaScript в основном соответствует синтаксису языка Java, но упрощен в сравнении с ним, чтобы сделать язык сценариев легким для изучения.

Язык JavaScript, в отличие от языков Java и C++, не содержит классов объектов в строгом смысле слова. Вместо этого он поддерживает конструкторы, которые создают объекты путем выделения для них памяти и инициализации всех или некоторых их свойств.

Объекты создаются путем вызова конструктора в операции new.

JavaScript поддерживает наследование, основанное на прототипах. С каждым конструктором связан соответствующий прототип, и каждый объект, созданный конструктором, содержит неявную ссылку на этот прототип (называемый прототипом объекта). Прототип, в свою очередь, может содержать ссылку на свой прототип и так далее. Так образуется цепочка прототипов. Ссылка на свойство объекта — это ссылка на первый прототип в цепочке прототипов объекта, который содержит свойство с данным именем. Иными словами, если данный объект имеет свойство с данным именем, то используется ссылка на это свойство; если нет, то исследуется прототип этого объекта и т. д.

В JavaScript текущее состояние и методы реализуются объектами, а структура и поведение наследуются. Все объекты, которые явно содержат свойство, которое содержит их прототип, разделяют это свойство и его значение. В отличие от языков, основанных на классах, свойства могут динамически добавляться к объектам путем присвоения им значений. В частности, конструкторы не обязаны присваивать значения всем или некоторым свойствам создаваемого объекта.

Согласно классическому определению, сценарий (script) — это последовательность команд (а иногда даже программа), которая интерпретируется и обрабатывается другой программой. Это означает, что для написания сценария достаточно текстового редактора, в то время как для создания программы требуется другая программа (по крайней мере, компилятор).

JavaScript — язык третьего поколения, в чем-то родственен языкам C, Pascal или BASIC. Между ними есть сходство, но есть и значительные различия:

- JavaScript — язык свободной формы, то есть тщательное форматирование не обязательно;
- JavaScript — интерпретируемый язык, то есть обрабатывается в компьютере отдельной программой — интерпретатором;
- JavaScript — высокомобильный и не зависящий от аппаратного обеспечения язык, то есть, как и Java, может использоваться на любой платформе;
- JavaScript легко внедряется в другие программы, например в браузеры (что невозможно сделать с C++).

Варианты размещения сценария

- Размещение в теле программы (между тегами <BODY>). В этом случае сценарий исполняется при загрузке страницы в браузере.
- Размещение в заголовке документа (между тегами <HEAD>). При таком размещении сценарий не исполняется сразу при загрузке, а может использоваться как функция другими сценариями.
- Размещение внутри тега HTML (между угловыми скобками <...>). При этом сценарий является обработчиком событий, для его записи не требуются теги <SCRIPT>.
- Размещение в отдельном файле. Язык JavaScript допускает создание собственных файлов с расширением .js. В этих файлах размещаются сценарии, которые вызываются по имени конкретного файла (имя файла записывается между тегами <<SCRIPT>).

2. Базовые конструкции

Константы, переменные, идентификаторы

Константой называется величина, которая в ходе выполнения программы не меняет своего значения.

Переменные, в отличие от констант, могут менять свои значения в процессе выполнения программы. Под переменной в программировании понимается именно область памяти (ячейки), которая хранит определенное значение.

Выражение представляет собой запись, которая может включать константы и переменные, а также знаки операций (арифметических, логических и т. д.).

Имя переменной или константы называется *идентификатором*.

Правила составления идентификаторов.

1. Идентификатор должен начинаться с буквы или с символа подчеркивания (b1, korona, My_Function, _12x).
2. Идентификатор может состоять из любых букв алфавита, цифр (0–9) и символа подчеркивания.

3. Идентификаторы не должны состоять из отдельных слов, разделенных пробелом, дефиса (-), точки и других специальных символов. Например, сочетание param 3 будет рассматриваться как имя переменной param, за которым следует число 3.
4. В языке JavaScript различаются верхний и нижний регистры букв. Например, пользовательские имена ExtVar, Extvar, extVar и extvar будут отвечать различным переменным.

Разделители:

Точка с запятой.

Инструкции кода JavaScript записываются в строки. Каждая новая строка начинается с новой инструкции. Каждая инструкция в одной строке заканчивается разделителем — точкой с запятой.

Точка, запятая.

Точки используются для различного рода ссылок в иерархической структуре объектов. Например, document.forms означает обращение к семейству форм документа. Запятыми отделяются друг от друга аргументы при записи функции. Например, fullData(x, 2, oblig) — функция трёх аргументов.

Скобки. Кроме указанных выше разделителей, в JavaScript используются как разделители трех типа скобок () {} [ ]

Круглые скобки () применяются в вычисляемых выражениях для задания определенного порядка действий, для записи списка параметров функций, а также для обозначения выражений в различных инструкциях (if, while, for,..), например, if (alfa < 0).

Фигурные скобки {}. Фрагмент кода JavaScript, ограниченный двумя фигурными скобками, называется блоком. Блок обычно выражает группировку операторов. Блоки используются, например, в инструкциях условных операторов, операторов цикла и др. В определениях функций скобки {} обозначают начало и конец тела функции:

```
function Degr(x) {x /= 2;
Degr = x*x*x;}
```

Квадратные скобки [ ] используются в теле цикла for-in для обозначения свойства объекта.

Комментарии. Для лучшего восприятия программы в ее текст разработчиком обычно вставляются неисполняемые фрагменты, которые называются комментариями:

- двойной слеш // обозначает комментарий, который начинается от знака слеша и заканчивается концом данной строки. То есть знак // относится к однострочному комментарию.
- текст, заключенный между символами /* и */, интерпретируется как комментарий. Такой текст может занимать более чем одну строку (многострочный комментарий).

Типы данных. В процессе выполнения программы одни переменные могут принимать численные значения, другим переменным присваиваются значения текста, а некоторые переменные могут быть логическими: true (истина) или false (ложь). Над числовыми переменными возможны арифметические опера-

ции, над текстовыми переменными — операция конкатенации (объединения), над логическими переменными — операции логических И, ИЛИ и др.

Числовой тип. В JavaScript используются целые числа (например, 3, -45, 2001) и числа с плавающей запятой (например, 3.1415, -4.5, 0.89991). Возможна запись чисел не только в десятичной системе счисления, но и в восьмеричной, шестнадцатеричной системах. Восьмеричные числа начинаются с цифры 0 или -0 (к примеру, 017 или -0336). Шестнадцатеричные числа начинаются с символов Ox либо -Ox, например: Ox123, Ox34AB, -OxFE12D.

Числа с плавающей запятой могут быть представлены в экспоненциальной форме, которая вводится символами «e» или «E» с последующим значением показателя, например, 1.33e-2 = 0.0133, 4.5E+4 = 45000

*Булевский (логический) тип***.** Данные этого типа могут принимать всего два значения: true и false. Выражение может быть истинным или ложным в зависимости от значений входящих в него аргументов. Это относится к выражениям, которые содержат операции сравнения. Например, выражение 3<5 будет истинным, а 7 = 10 — ложным. Одной из областей применения логического типа являются условные операторы, операторы цикла.

Строковый тип. Строка — это последовательность, составленная из символов кодовой страницы, установленной в компьютере. Когда строку используют в выражениях, то заключают ее в двойные кавычки, например "Аллигатор" или "Ошибка ввода". Такие конструкции называются строковыми литералами. Возможно совместное использование двойных (") и одинарных (') кавычек. Если в строке встречаются одинарные кавычки, то ее лучше заключить в двойные кавычки. Например, 'Это сценарий "Повторная игра" '; " That's good ".

Нулевой тип. Нулевой тип обозначается ключевым словом null. Технически нулевой тип получается, когда переменная в программе не получила определенного значения (числа, строки или логического значения). Однако нужно помнить, что null — это не то же самое, что число 0, хотя при определенных обстоятельствах может быть конвертирован в нуль.

Объявление переменных и отсутствие типизации. Это выполняется с помощью ключевого слова var, например, var x; var result; var x, y, z; Объявление переменной можно совместить с присвоением ей начального значения: var k = 3.

Обратите внимание, что в приведенных примерах объявляется только переменная с помощью слова var, однако не определяется тип данных.

Операции. Выражение состоит из операндов, а также из знаков операций и круглых скобок.

Например, в выражении b+c-10 величины b, c и константа 10 являются операндами, а «+» и «-» — знаками операций. Круглые скобки используются для управления порядком вычислений.

В JavaScript различают следующие типы операций:

- унарные операции, в которых участвует один операнд (например, взятие обратного знака величины: -a);
- бинарные операции, которые выполняются над двумя операндами (например, арифметические операции сложения, умножения и т. д.);

- тернарная операция, которая комбинирует три операнда; такой операцией является условное выражение.

Операции в JavaScript можно разбить на группы соответственно их назначению:

- арифметические операции;
- операции сравнения;
- строковые операции;
- логические операции;
- побитовые операции;
- операции присваивания;
- смешанные операции.

Арифметические операции (операторы сложения, вычитания, умножения и деления).

Таблица 29 — Арифметические операции в JavaScript

Символ оператора	Операция	Пример	Значение
+	Сложение	3+25.1	28.1
-	Вычитание	3-2.2	1.8
*	Умножение	2.1*3	6.3
/	Деление	3/8	0.375
%	Остаток	10/3	1
-	Унарный минус	-3	-3

Операции инкремента и декремента. Операции инкремента и декремента являются унарными операциями (имеют более высокий приоритет) и отвечают соответственно увеличению и уменьшению значения на 1. Эти операции часто применяются к переменным числового типа, которые меняют свои значения на 1, например к счетчикам циклов.

Записываются эти операторы в виде i++, i-- или ++i, --i.

Справедливы следующие правила:

- если операция записана перед операндом: ++i (префиксная форма инкремента), то инкрементирование выполняется до вычисления текущего выражения;
- если операция записана после операнда: i++ (постфиксная форма инкремента), то инкрементирование выполняется после вычисления текущего выражения.

1. Например, в результате операций i =1; k = ++i; i и k примут одинаковые значения, равные 2.

2. Однако инструкции i = 1; k = i++; приведут к значениям i = 2 и k = 1.

3. Пусть исходное значение i = 3, тогда после выполнения k: = i --*3; переменные i и k приобретут значения: i=2, k = 9.

4. Пусть исходное значение i = 3, тогда после выполнения k: = --i *3; i = 2, k = 6.

Операции сравнения. С помощью этих операций выполняется сравнение величин различных типов и возврат значения true или false в зависимости от результата сравнения. Операции сравнения часто используются при организации ветвлений в программах совместно с операторами if и while.

Таблица 30 — Операции сравнения в JavaScript

Символ операции	Сравнение	Пример	Значение
==	Равно	2==3	false
!=	Не равно	2.1!=2	true
<	Меньше	“a”<”b”	true
>	Больше	7>9	false
<=	Меньше или равно	“g”<= “ga”	true
>=	Больше или равно	“6”>=5	true

Строковые операции. К строковым операторам относятся операторы, которые выполняют действия над строками. Среди таких операторов уже рассмотренные операторы сравнения, с помощью которых можно сравнивать значения двух строк. Сравнение выполняется посимвольно по степени отклонения от алфавитного порядка. При этом используется алфавит таблицы ASCII и, согласно этой таблице, прописные буквы располагаются перед строчными буквами (имеют меньшие значения). Еще одной строковой операцией является конкатенация (от *англ.* concatenation — сцепление, соединение). Эта операция объединяет в новую строку два текстовых значения и обозначается, как и арифметическое сложение, символом «+».

Например, выражение "c" + "d" даст результат "cd", записанный без пробелов.

Слово "паровоз" можно получить с помощью операции "пар"+ "о"+"воз"

"1" < 2 // операция возвращает значение true

3 != "3" //результат сравнения — значение false

Логические операции. Значения логического типа допускают операции конъюнкции (логическое И — AND — &&), дизъюнкции (логическое ИЛИ – OR — | |) и отрицания (логическое НЕ — not — !).

Примеры:

1. a && (b++ < 5) или a && (c -- < 0)

поскольку инкрементированное b и декрементированное c могут быть вычислены либо нет, в зависимости от значения a.

2. ! (a && b)

(al && a2 || b1 && Ь2)

Операции присваивания. Операции присваивания, как и отрицание (! и -), инкремент (++) и декремент (--), относятся к унарным операторам. В JavaScript наряду с простым присваиванием = предусмотрено составное присваивание, которое совмещают с присваиванием какую-либо арифметическую операцию. Операция += применима как к числам, так и к строкам, например в результате

выполнения операций: a = "Я люблю "; a+= "кино"; переменная получит значение «Я люблю кино» .

Таблица 31 — Операции присваивания в JavaScript

Символ оператора	Операция	Пример	Значение
=	Присваивание значения	x=2.5	x=2.5
*=	Умножение и присваивание	x*=y	x=x*y
/=	Деление и присваивание	x/=y	x=x/y
%=	Остаток и присваивание	x%=y	x=x%y
+=	Сложение и присваивание	x+=y	x=x+y
-=	Вычитание и присваивание	x-=y	x=x-y

Прочие операторы. Операция условного выражения ?: (в языках C, C++, Java, JavaScript соответствует условному оператору if). Общий вид операции ?: не вполне соответствует ее обозначению (символы ? и : не стоят рядом):

expression1 ? expression2 : expression3

В записи операции участвуют три операнда (expressionl, expression2, ...), то есть является тернарной операцией. Первый операнд принимает булево значение (true или false), а второй и третий операнды могут принадлежать любому типу данных. Если значение expressionl равно true, оператор ?: возвращает значение, вычисляемое выражением expression2. Если же значение первого операнда false, результатом выполнения тернарного оператора будет значение выражения expression3. Например, оператор **x<1 ? 1-x : x - 1** возвращает значение 1-x, если x < 1, и значение x-1 в противном случае.

В языке JavaScript предусмотрены также операции, относящиеся к объектам, массивам, функциям, типам данных и др.

Классификация инструкций

1. Простые инструкции: инструкции присваивания, арифметические операции, объявления переменных и др.

2. Сложные инструкции.

Рассмотрим сложные инструкции JavaScript.

1) Инструкции выбора.

Инструкции выбора (операторы if, else)

Условный оператор if

if (<логическое выражение>) оператор1; [else оператор2;]

Логическое выражение может принимать значения true или false, оператор1 (серия операторов или один оператор) выполняется в случае, когда значение выражения равно true, в противном случае либо пропускается, и выполнение программы продолжается с оператора, следующего за if, либо при наличии оператора else будет выполняться оператор2.

Например:

```
 if (a>b) document.write("A больше B");
else document.write("A НЕ больше B");
 if (day==1) document.write("понедельник");
```

```
 else if (day==2) document.write("вторник");
else if (day==3) document.write("среда");
else if (day==4) document.write("четверг");
 else if (day==5) document.write("пятница");
 else if (day==6) document.write("суббота");
 else if (day==7) document.write("воскресенье");
else document.write("Ошибка: day="+day);
```

Условное выражение

В JavaScript существует операция ?:, называемая условным выражением.

Рассмотрим пример, когда в зависимости от температуры на улице нужно вывести на экран тот или иной комментарий: x=(t>=30) ?alert ("В Москве жара!"):alert("Еще можно жить")

Оператор выбора switch

Условный оператор if обеспечивает ветвление только с двумя вариантами выбора. Для организации множественного ветвления используется более мощный оператор выбора switch. Этот оператор состоит из выражения (селектора) и списка вариантов.

В JavaScript оператор выбора записывается в следующем формате:

```
switch(expression){
case value1: series1;
break;
case value2: series2;
break; ……
case valueN: seriesN;
break; default: seriesD; }
```

Сначала вычисляется значение селектора expression, следующего за словом switch. После этого выполняется серия, константа выбора которой (value) равна значению селектора. Следующий затем оператор break означает выход из инструкции ветвления: выполнение программы возобновляется с первого оператора, следующего за инструкцией switch. Если ни одна из констант не равна текущему значению селектора, то исполняется серия, стоящая после слова default. Часть default: и связанные с нею инструкции в тексте программы можно опустить. Тогда, если среди констант селектора отсутствует нужное значение, выполнение оператора switch ни к чему не приведет.

Рассмотрим пример оператора выбора, с помощью которого на экран выводится сообщение о выборе рубрики. Пусть в качестве селектора выступает переменная theme, принимающая значения 1, 2, 3, 4.

```
switch(theme){
case 1:document.write("Новости")
 break
case 2:
document.write("Финансы")
 break
 case 3 :
 document.write("Спорт")
```

```
break
 case 4 :
document.write("Погода")
 break
 default:document.write("Рубрика не выбрана")}
```

2) Инструкции циклов. Повторяемые группы команд (серии операторов) принято называть циклами. В JavaScript циклы записываются с помощью операторов трёх видов: while, do-while и for. Первый оператор представляет собой цикл с предусловием, второй оператор — цикл с постусловием, а последний оператор — это цикл с параметром, который можно отнести к циклам с предусловием.

Оператор while

while (<логическое выражение>) оператор;

Сначала вычисляется логическое выражение. Если оно истинно, то выполняется оператор. Затем снова вычисляется логическое выражение. Если оно и на этот раз истинно — снова выполняется оператор и снова вычисляется логическое выражение. Так продолжается до тех пор, пока логическое выражение не станет ложным.

Оператор while удобно использовать, когда число повторений цикла заранее не известно. Допустим, нужно просуммировать конечный ряд натуральных чисел: 1, 2, 3, и найти наибольшее значение суммы, не превышающее 1000. На экран нужно вывести количество слагаемых в сумме и значение суммы:

```
summ = 0 ; i = 0 ;
while (summ <= 1000){
i + +; summ+=i}
document.write("Количество слагаемых равно "+(i-1),"<br>");
document.write("Сумма равна "+(summ-i));
```

Оператор do-while

В этом операторе цикла, являющемся оператором с постусловием, логическое условие продолжения цикла следует после тела цикла. Оператор do-while имеет синтаксис:

do {series}
while (expression)

После служебного слова do следует операторный блок series. За while стоит вычисляемое условие expression. Если после выполнения операторов блока значение expression равно true, то блок вычисляется снова. Так продолжается до тех пор, пока expression не окажется равным false. После этого вычисления цикла прекращаются, и происходит переход к следующему оператору программы.

Пример вычисления факториала:

```
factor=l; i=l;
do {factor*=i; i++}
while(i<=4) ;
```

Оператор for

Этот оператор цикла включает в себя переменную, называемую параметром цикла или счетчиком цикла. Последнее название связано с тем, что этот

раметр может принимать значения, равные числу повторений цикла. Оператор for в JavaScript имеет такой же формат, что и одноименный оператор в Java:

Цикл for

В принципе без него можно и обойтись, потому что все, что он делает, можно записать и с помощью простого цикла while, однако иногда цикл for приводит к более простым и изящным программам.

for (выражение1;выражение2;выражение3) оператор;

Выполняется он таким образом:

1. Вычисляется выражение1
2. Вычисляется выражение2
3. Если выражение2 можно рассматривать как истину (либо оно было логическим, либо оно не равно ни нулю, ни пустой строке), то выполняется оператор, иначе цикл завершается и программа продолжает выполнение с точки, следующей за оператором.
4. Вычисляется выражение3
5. Переход к шагу 2

Примеры: Рассмотрим использование инструкции for на примере вычисления значения факториала N!.

```
factor = 1;
  for (i = 1; i <= 10; i++) factor *= i;
```

Например, вычисление факториала и вывод значений на экран можно задать следующей инструкцией:

```
for(i = 1, factor =1; i < 10; i + +, factor *= i);
document.write(factor, "<br>");
```

Запишем оператор для вычисления суммы sum ряда рациональных чисел: 1/2, 1, 3/2,.. n:

```
for (i = 0, sum=0; i<=n; i+=l/2, sum + = i) ;
```

3) Оператор for-in и просмотр свойств браузера.

Оператор for-in удобен при выполнении операций над элементами массивов, а также над свойствами объектов.

Конструкция for-in имеет следующий общий вид: **for (variable in object) {series}**

variable обозначает имя переменной, либо элемент массива, либо свойство объекта. Слово object обозначает имя объекта, а series — блок операторов, образующий тело цикла. Например, если вы хотите обратиться к свойствам объекта navigator (этот объект отвечает браузеру), в заголовке оператора цикла нужно записать for (k in navigator), где индекс k будет пробегать свойства. Например, с оператором for-in выведем на экран все свойства браузера:

```
<script language="JavaScript">
for (k in navigator)
document.write(k+"="+ navigator[k]+"<br>");
</script>
```

4) Прерывание цикла. Выполнение цикла можно прервать break, который размещается в теле цикла. Этот оператор необходим при возникновении каких-либо ошибок или в случае появления некоторого значения переменной

```
 i=0;
 while (true) {
document.write(]I, "<br>"););
 if (i= = 9) break; i++)
```

5) Прерывание итерации цикла. Допустим, вам нужно прервать выполнение не всего оператора цикла, а только выполнение блока операторов при данной итерации цикла. Тогда воспользуйтесь оператором continue, который записывается внутри тела цикла.

Пример: вычисление суммы ряда натуральных чисел от 1 до 99, в котором отсутствуют числа, кратные 5. Найти эту сумму проще всего следующим образом:

```
summ=0;
for (i = 1; i<=99; i++) {
 if (i%5 = = 0) continue;
 summ += i)
document.write("Сумма равна ",summ);
```

Функции

Функция — это определенная заранее последовательность инструкций, которая обозначается идентификатором и которая может возвращать значение, присваиваемое идентификатору функции.

Использованию функции в программе должно предшествовать ее объявление, которое имеет вид:

function name (argl, arg2,..) {series}

где **function** — служебное слово; name — идентификатор функции; arg1, arg2, ..., — список аргументов функции; series — последовательность операторов, которая оставляет тело функции.

Имена функций (идентификаторы) принято в JavaScript записывать строчными буквами. Список аргументов заключается в круглые скобки, и аргументы отделяются друг от друга запятыми. Кстати, список аргументов может быть пустым. Однако круглые скобки () должны обязательно присутствовать, поскольку именно они означают вызов функции.

В большинстве случаев функции представляют собой лишь способ связать вместе нескольких команд. Давайте, к примеру, напишем скрипт, печатающий некий текст.

```
 <html>
 <script language="JavaScript">
 document.write("Это JavaScript!");
 </script>
</html>
```

И такой скрипт напишет следующий текст: Это JavaScript! Или вывод текста **Это JavaScript!** 3 раза

```
<html>
 <script language="JavaScript">
      function myFunction() {
      document.write("Это JavaScript!<br>");
```

```
        }
        myFunction(); myFunction(); myFunction();
</script> </html>
```

Инструкция return. Чтобы функция возвращала в сценарий какое-либо значение, в тело функции вставляется инструкция return. Например, объявление функции для вычисления площади прямоугольника:

```
 function rectangle (a,b) {
s = a*b
 return s }
```

Вызов функции. Для вызова функции нужно обратиться к ней по имени и записать оператор круглых скобок (). В скобках могут быть указаны значения аргументов. Если в качестве аргументов использованы выражения, то они будут вычислены и их значения будут подставлены в функцию. Допустим, вы объявили функцию, которая выводит сообщение с приветствием:

```
function Hellow (xname) { alert ("Привет, " + xname)}
```

Перед выводом сообщения на экран в функцию Hellow (xname) будет подставлено значение переменной xname.

3. Объектная модель JavaScript.

Язык JavaScript основан на понятиях: **объект**, **атрибут** и **функция**.

Объект JavaScript — это неупорядоченный набор свойств, каждое из которых имеет нуль или более атрибутов, которые определяют, как это свойство может использоваться.

Свойства — это контейнеры, которые содержат другие объекты, примитивные значения и методы.

Примитивное значение — это элемент любого из встроенных типов: Undefined, Null, Boolean, Number и String; объект — это элемент еще одного встроенного типа Object; метод — функция, ассоциированная с объектом через свойство.

JavaScript содержит несколько встроенных объектов, таких как Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. JavaScript содержит набор встроенных операций, которые, строго говоря, не обязательно являются функциями или методами, а также набор встроенных операторов, управляющих логикой выполнения программ.

JavaScript очень удобен для создания и отладки веб-страниц.

Объекты — это, например, текущее окно (Window), текущий документ (document), кнопки (button, checkbox, select и др.), дата (date), обработки данных (number, array, math, string и др.).

Атрибут объекта — это входное или выходное данное.

Объектом *в JavaScript (как и в программировании вообще) называется каким-либо образом определенный набор данных и процедур их обработки, которые рассматриваются как единое целое*. Это очень важно, что данные и процедуры их обработки являются одним целым. Формальное определение объекта звучит примерно так: «объект — это обладающий свойствами и методами». Именно таким образом мы прячем всю обработку данных внутри опре-

деления объекта. В обыденной жизни объектами принято называть предметы материального мира, здания, инструменты, книги и т. д. В языках программирования содержание объекта несколько иное: это может быть информационная модель реального объекта, быть модель, существующая только в абстрактном виде.

Свойства *характеризуют состояние, в котором находится объект. Чтобы сослаться на свойство объекта, нужно указать сам объект и через точку "." записать свойство. Имя свойства должно быть идентификатором, не содержащим пробелов.*

Функция, применяемая к объекту JavaScript, называется методом. Методы отвечают действиям, которые можно совершить над данным объектом. Подобно обычным функциям методы возвращают значения.

Объект Document

Объект **document** создан за нас системой незадолго до запуска нашей программы. Для того чтобы обратиться к свойству или методу объекта, нужно написать имя этого свойства (метода) через точку от имени переменной-экземпляра. В таких случаях говорят, что мы обращаемся к методу (свойству) в контексте конкретного экземпляра. В случае метода за именем должен следовать список параметров в скобках. Скобки, как почти везде в этом языке, нельзя опускать.

Встроенные объекты JavaScript

Среди объектов, с которыми имеет дело веб-разработчик при создании, различают встроенные и пользовательские объекты. Сначала рассмотрим встроенные объекты.

1) Объект Date

Конструктор и методы объекта Date позволяют для него определять не только текущие, но и любые дату и время. Мы остановимся на основных операциях объекта Date.

Отображение текущих даты и времени

Объект Date в JavaScript создается оператором new и конструктором Date и выглядит так **datetime = new Date();**

Создать переменную типа Date можно разными способами. Для того чтобы присвоить переменной значение текущей даты и текущего времени (на момент загрузки вашей страницы пользователем), достаточно написать: var a = new Date();

1. Чтобы присвоить переменной значение некоторой конкретной даты, пишем:

var a = new Date(yyyy,mm,dd);

2. Чтобы присвоить переменной значение некоторой конкретной даты и времени суток, пишем:

var a = new Date(yyyy,mm,dd,hh,mm,ss);

3. Чтобы создать новую переменную типа Date и присвоить ей значение уже существующей, пишем:

var a = new Date(otherDate.getTime());

Если аргумент в конструкторе отсутствует, то будет создан объект, представляющий текущую дату и время.

datetime = new Date(); //Создание объекта, представляющего текущие дату и время document.write ("Сегодня: ", datetime . toLocaleString ( ) , "<br>") ;

Таблица 32 — Методы объекта Date

Метод	Операция
setDate()	Задает день месяца
setMonth ( )	Задает месяц (в диапазоне от 0 до 11)
setYear()	Задает год
setTime( )	Задает время в миллисекундах от 1 января 1970 г.
setHours ( ) , setMinutes ( ) , setSeconds ( )	Задает время
getDate ( )	Возвращает день месяца
getMonth ( )	Возвращает месяц
getYear ( )	Возвращает год
getTime ( )	Возвращает время в миллисекундах от 1 января 1970 г.
getHours ( ) , getMinutes ( ) , getSeconds ( )	Возвращает время
toLocaleString ( )	Преобразует значение объекта Date в строку в формате регионального времени
getTimeZoneOffset ( )	Возвращает значение временного сдвига относительно нулевого меридиана

2) Объект Math

Этот объект JavaScript содержит математические функции и константы. Функции представляются методами объекта, а константы — свойствами.

Константы

1. Math.E — основание натурального логарифма, равное 2.7183.
2. LN2 — натуральный логарифм 2, равный 0.6931.
3. Math.LN10 — натуральный логарифм 10, равный 2.3026.
4. Math.LOG2E — логарифм е по основанию 2, равный 1.4427.
5. Math.LOG10E — логарифм е по основанию 10, равный 0.4343.
6. Math.PI — число π = 3.1416.
7. Math.SQRT1_2 — квадратный корень из 1/2, равный 0.7071.
8. Math.SQRT2 — квадратный корень из 2, равный 1.4142.

Примеры:

1. var a = Math.random(); // a получило случайное значение от 0 до 1

Обратите внимание, что после имени функции обязательно пишется список аргументов, даже если он и пустой.

2. Следующий фрагмент программы с вероятность 0.4 печатает слово ДА или НЕТ, а с вероятностью 0.2 печатает НЕ ЗНАЮ.

var a = Math.random(); // a получило случайное значение от 0 до 1

```
if (a < 0.4) document.write("ДА");
else if (a < 0.8) document.write("НЕТ");
else document.write("НЕ ЗНАЮ");
```

3. Тригонометрические функции вычисляются аналогично:

```
document.write("sin(0.5)="+Math.sin(0.5));
document.write("cos(4)="+Math.cos(4));
document.write("tg(0.1)="+Math.tan(0.1));
```

Таблица 33 — Методы объекта Math

Метод	Операция
Math. ceil (x)	Округляет x до ближайшего наибольшего целого числа
Math. floor (x)	Округляет x до ближайшего наименьшего целого числа
Math. round (x)	Округляет x до ближайшего целого
Math. random ( )	Генерация случайных чисел в диапазоне от 0 до 1
Math.abs (x)	Возвращает модуль x
Math.cos(x), Math.sin(x), Math.tan(x)	Возвращают значения тригонометрические функций
Math.acos (x) , Math.asin (x) , Math.atan(x)	Возвращают значения обратных тригонометрических функций
Math.exp (x) , Math . log (x)	Возвращают значения экспоненты и натурального логарифма
Math.sqrt (x)	Возвращает квадратный корень из x
Math.min (x, y)	Возвращает минимальное из двух чисел x, y
Math. max (x, y)	Возвращает максимальное из двух чисел x, y
Math.pow(x, y)	Возвращает значение степени x^y

3) Массивы в JavaScript

Одним из типов объектов являются массивы, которые часто используются для обработки множества значений. Массивы были введены в JavaScript для возможности манипулирования самыми разными объектами веб-страницы: всеми ссылками данной страницы, всеми картинками на данной странице, всеми апплетами, всеми элементами формы и т. п.

Рассмотрим пример введения массива для координат материальной точки (x, y, z). Обозначим совокупность этих координат одним идентификатором r и снабдим этот идентификатор индексом i (= 1, 2, 3). Теперь, если нам надо получить доступ к значениям координат, мы будем пользоваться операциями присваивания типа: x= r [1]; x= r[2]; x = r[3];

Идентификатор r обозначает массив, а индексы в квадратных скобках — элементы массива.

Массив — это тип данных, который состоит из пронумерованных фрагментов данных. Фрагменты данных называются элементами массива, а номера, присваиваемые элементам, — индексами. Например, элементам некого массива examp можно присвоить следующие значения: examp[0]=2.713; examp[1]=4; …., examp[12]=”Урок”; examp[13]=true;

Доступ к элементам массива обеспечивается оператором [ ], аналогично тому, как доступ к функции получаем с помощью оператора (). Заметим, что индекс первого элемента массива не 1, а 0. Например, X[2] является третьим элементом массива. Еще одно отличие массивов в JavaScript от массивов в структурированных языках состоит в том, что количество элементов массива может не быть фиксированным.

Массив создается аналогично тому, как создается объект. При этом можно использовать два способа его создания: с помощью конструктора Object () и с помощью конструктора Array ().

Согласно первому способу, для создания массива нужно сначала определить пустой объект-массив, а затем задать его элементы, например, xar = new Object (); xar[0] = 2; xar[l] = 5; …, xar[7] = 1.5;

Для создания массива чаще используют другой способ: применение конструктора Array. Этот конструктор был введен в версиях Netscape Navigator 3.0 и Internet Explorer 3.0. Например, для создания пустого массива a нужно записать инструкцию: a = new Array();

Возможно задание массива с определением его элементов:

b = new Array(2,41,3.14,"als",22,"fil3");

Конструктор Array автоматически определяет свойство length — количество элементов массива. Так для определенных выше массивов a и b выражения:

a.length = = 0; b.length = = 6

имеют значение true. Поскольку в пустом массиве не определен ни один элемент, значит, length равно нулю.

Операторы цикла при работе с массивами

Многие операции над элементами массива выполняются с помощью операторов цикла.

Ввод данных и суммирование элементов массива, простейший калькулятор

Воспользуемся оператором цикла для нахождения суммы элементов массива и вывода его на экран. Это можно сделать с помощью следующего простого кода (идентификатор массива X):

```
sum = 0;
for (i=0; i<x.length; i++) sum +=x[i];
document.write ("Сумма элементов массива равна: “,sum);
```

Сценарий, имитирующий работу калькулятора при вычислении суммы чисел. С клавиатуры последовательно вводятся числа, а затем при нажатии на клавишу «=» на экран выводится значение их суммы. Ввод данных реализуем с помощью оператора запроса window.prompt. Для автоматизации процедуры ввода применим оператор do-while:

```
X=new Array();
I=0;
Do { Num= window.prompt ("Введите число");
If (num!="=") {num= parseFloat(num);
x[i] = num;
i++}  }
while(num != "=")
```

Методы массивов

Поскольку массивы являются одновременно объектами, для них имеется несколько методов:

1. Array.join () — преобразование всех элементов массива к строковому типу и последующая конкатенация элементов;
2. Array.reverse () — обращение порядка расположения элементов в массиве;
3. Array.sort () — сортировка элементов массива.

Проиллюстрируем действие этих методов на примерах. Допустим, имеется массив, созданный конструктором:

xar = new Array(11,12,13 ,14) ;

В случае преобразования к текстовому типу (метод Array, join) выполнение инструкции jar = xar.join(); приведет к созданию строки "11,12,13,14". В круглых скобках при обращении к методу задается разделитель между элементами массива, например jar =xar.join("+"); будет отвечать преобразованию к строке "11+12+13+14". Полученные таким образом строки можно обратно преобразовать к элементам массива, если воспользоваться методом разбиения строки String.split ().

Обращение порядка элементов в массиве задается инструкцией вида: xar.reverse ();

При этом получаем массив с элементами: xar[0] =14, xar[1] =13, xar[0] =12, xar[0] = 11. Заметим, что метод Array.reverse () не создает нового массива, а только переставляет местами элементы в существующем массиве. Последовательное выполнение инструкций:

xar = new Array(11,12,13,14); xar reverse(); jar = xar.join();

приведёт тому, что значением переменной jar будет строка "14,13,12,11".

Сортировка элементов массива с помощью Array.sort (). Элементы массива размещаются согласно алфавитному порядку составляющих символов:

```
<SCRIPT language=" JavaScript ">
    Xar=new Array("Petr", "Alisa",2,"апрель",3, "Brian");
    Xar.sort(); jar=xar.join(',') ; document .write (jar) ;
</SCRIPT>
```

Таблица 34 — Методы объекта Array

Метод	Описание
concat	Объединяет два массива в один
join	Объединяет все элементы массива в одну строку
pop	Удаляет последний элемент из массива и возвращает этот элемент
push	Добавляет один или более элементов в конец массива и возвращает последний добавленный элемент
reverse	Переворачивает массив так, что нулевой элемент становится последним и т. д.
shift	Удаляет нулевой элемент массива и возвращает его
slice	Выделяет часть массива
splice	Добавляет элементы в массив и удаляет из него
sort	Сортирует элементы массива
toString	Возвращает строковое представление массива
unshift	Добавляет один или несколько элемент(ов) в начало массива

4. Окна и динамически создаваемые документы

Все объекты браузера организованы в иерархическую структуру. Поскольку основные функции браузера реализуются в окне приложения, в котором отображается сам HTML-документ, центральным объектом иерархии является окно браузера. Оно представляется объектом window; все другие объекты HTML рассматриваются как свойства этого объекта. На основе window можно определить объекты, свойства и методы, необходимые для полноценной работы с документами.

Объекту window подчинены объекты следующего уровня, которые можно разделить на две группы:

- объекты браузера (events, location, navigator и screen), предоставляющие доступ к свойствам, методам и событиям, происходящим в окне браузера;
- объекты документа и фреймов (document и frames), позволяющие управлять элементами документов и фреймов, загруженных в браузер.

Вторая группа объектов вместе с содержащимися в ней свойствами, методами и событиями образует так называемую объектную модель документа или сокращенно DOM (Document Object Model). Структура объектов браузера имеет один корень — объект window. Все остальные объекты на стороне клиента существуют как компоненты других объектов, а те, в свою очередь, подчинены текущему объекту window. Объект window содержит ссылки (прямые или через другие ссылки) на все подчиненные объекты.

Операции с окнами (объект window)

При работе браузера всегда существует объект window — это текущее окно. Тем не менее, вы можете создать еще одно или даже несколько окон, которые будут благополучно уживаться в рамках принятой объектной модели. В один и тот же момент времени могут существовать несколько объектов window. В языке JavaScript считается, что ссылка window всегда обозначает текущее окно, то есть окно документа, содержащего исполняемый JavaScript-код. Обращаться к другим объектам window можно с помощью имен, которые задаются при создании окна.

Создание нового окна браузера (метод open). Чтобы создать новое окно, нужно применить метод window.open. Типичные инструкции, вводящие этот метод, имеют следующий вид:

window.open("URL","Namewin","Features");

либо var win=windows.open("URL","Namewin","Features");

Аргументы метода open ()

Метод open() может иметь аргументы: open (“URL”, “Namewin”, “Features”). Перечислим их назначение:

1) аргумент “URL” обозначает URL документа, загружаемого в окне;

2) аргумент “name” вводит название окна. Имя окна Namewin будет присвоено свойству name объекта win;

3) аргумент “features” обозначает список необязательных опций, которые разделена запятой. В этом списке могут быть параметры размеров окна в пикселях (width и height) и возможности изменения размеров границы окна

(resizable), а также параметров наличия панели инструментов (toolbar), меню (menubar), поля ввода адреса (location), строки состояния (status).

Переключение между окнами (методы focus() и blur()). Если на экране открыто несколько окон, вы можете для переключения между ними воспользоваться методом window.focus (). Окно, получившее фокус, будет отличаться от других окон цветной подсветкой строки заголовка. Лишить окно фокуса можно с помощью метода window.blur. Вызов метода focus () можно связать с выполнением соответствующего обработчика событий — onfocus. При этом важно учитывать, что событие onfocus запускается, если фокус получает прежде не активизированное окно. Если же окно активно, то при вызове метода focus () оно уже не будет генерировать событие onfocus.

Присвоение имени окну (свойство name). Проще всего выполняются манипуляции с именем окна. Вообще говоря, все окна, создаваемые в браузере, остаются без имени, пока оно им не присвоено с помощью специальной директивы — присвоения строкового значения свойству name объекта window.

Перемещение окна браузера (методы moveTo, moveBy). Для перемещения текущего окна браузера в объекте window предусмотрены два следующих метода:

1) window.moveTo () — этот метод получает два параметра, отвечающих перемещению левого верхнего угла окна в точку с заданными координатами. Например, метод window.moveTo (100, 200) переместит окно в положение, при котором левый верхний угол окна будет расположен на расстоянии 100 пикселей от левой кромки экрана и 200 пикселей от верхней кромки экрана;

2) window.moveBy () — этот метод осуществляет перемещение окна на заданное расстояние, например window.moveBy (10, 5) переместит окно на 10 пикселей вправо и на 5 пикселей вниз. Возможны как положительные, так и отрицательные значения аргументов. Перемещение будет осуществляться при каждом выполнении метода.

Изменение размеров окна (методы resizeTo и resizeBy). Объект window позволяет изменять размер текущего окна. для этого предназначены два метода window.resizeTo () и window.resizeBy (), которые аналогичны методам перемещения окна.

1. Метод window.resizeTo () приводит к изменению окна до указанных размеров. Например, window.resizeTo (420, 350) изменяет размер окна до 420 пикселей в ширину и 350 пикселей в высоту.

2. Метод window.resizeBy () дает приращения размеров окна на указанную величину. Например, window.resizeBy (20, -15) увеличивает ширину окна на 20 пикселей и уменьшает высоту на 15 пикселей.

Печать веб-страницы (метод Print). Распечатать веб-документ можно с помощью кнопки «Печать» на панели браузера. Однако аналогичное средство имеется и в JavaScript, оно представлено методом window.print(). Например, разместим на веб-странице кнопку «Печать», введя в HTML-код строку.

Закрытие окна (метод close). Окна, которые созданы методом open, могут быть закрыты методом close.

Информация о документе и окне браузера (объект location)

Еще одним дочерним объектом window является объект location. Этот объект определяет точную ссылку (URL) на текущий веб-документ, загруженный в окно браузера.

Свойства объекта location. Свойства объекта location предоставляют информацию о URL документа, а методы объекта позволяют перезагружать документ и заменять текущий документ.

Получение данных URL. Допустим, нас интересует детальная информация о URL какого-либо документа. В этом случае нужно воспользоваться свойствами объекта location

Перезагрузка и замена текущей страницы (методы объекта location). Объект location имеет два метода: reload() и replace(). Метод reload() позволяет перезагрузить страницу, например в случае изменения страницы. То есть reload() в этом случае вызывает то же действие, что и нажатие кнопки «Обновить» в браузере. Однако, если кнопка «Обновить» приводит к перезагрузке лишь в случае, когда страница была изменена, метод reload() позволяет выполнить перезагрузку даже в случае отсутствия изменений. Для этого при вызове метода используется булевский аргумент: reload (force). Если force принимает значение true, перезагрузка производится в безусловном порядке, если же аргумент имеет значение false, страница перезагружается лишь при наличии изменений. С помощью второго метода, replace (url), можно заменить текущую страницу на другую, указанную в аргументе url. Этот метод часто оказывается полезным для выполнения переадресации на стороне клиента.

Управление строкой состояния (свойства status, defaultStatus)

В нижней части окна браузера отображается строка состояния. Содержимое этой строки можно задать с помощью свойств status и defaultStatus. Первое из этих свойств (status) отвечает информации, которая отображается в строке состояния временно, а второе свойство (defaultStatus) — информации, которая отображается по умолчанию, то есть сразу после открытия окна и до тех пор, пока свойство defaultStatus не изменится.

Переходы между веб-страницами (объект history)

Выполнять переходы между веб-страницами, ранее просматривавшимися пользователем, позволяет объект history, который является одним из дочерних объектов окна браузера. history представляет собой строковый массив только для чтения, составленный из URL открывавшихся страниц. Список элементов массива history соответствует содержимому Журнала в Internet Explorer или списку URL в меню GO браузера Netscape.

Свойства и методы, применяемые для переходов. Доступ к информации о ранее отображавшихся страницах обеспечивается следующими свойствами объекта history.

Работа с диалоговыми окнами

В рамках динамического HTML возможно создание нескольких типов окон. Выше было рассмотрено создание окон браузера, которые работают независимо от текущего окна. Такие окна называются немодальными.

Другие типы окон, к которым относятся различные диалоговые окна (диалоги), обычно используются, когда пользователь должен сделать выбор для продолжения работы в приложении. При вызове любого из диалогов работа приложения в текущем окне останавливается и браузер ожидает закрытия диалогового окна. Такие окна называются модальными.

Организация простых диалогов (методы alert, confirm и prompt). Простейшие диалоговые окна запросов и сообщений можно создавать с помощью методов объекта window. С двумя из этих методов (alert и prompt) вы уже знакомы по примерам предыдущей главы, теперь мы рассмотрим их более подробно. Приведем полный перечень методов, отвечающих за вывод простых диалогов в текущее окно:

1. alert () — отображает сообщение в простом диалоговом окне, имеющем одну кнопку OK. Диалоги alert не прерывают выполнение программы, однако alert — это именно тот тип назойливых сообщений, от которых пользователь обычно рад избавиться;
2. confirm (msg) — отображает в диалоговом окне вопрос msg, на который нужно ответить «да-нет» (OK или Отмена). Функция confirm () возвращает значение true, если нажата кнопка OK, либо значение false при нажатии кнопки «Отмена». Таким образом можно организовать ветвление в программе;
3. prompt () — отображает диалоговое окно с текстовым полем, в которое пользователь должен ввести строку. В диалоге имеются кнопка OK (возвращается значение текстовой величины) и кнопка «Отмена» (возвращается значение null).

Создание индивидуального диалогового окна. Еще одним типом диалога, поддерживаемого в DHTML, является индивидуальное диалоговое окно. Такое окно, в отличие от описанных выше диалогов, представляет собой веб-страницу в формате HTML с заданными размерами и своим URL.

Определение окна (метод showModalDialog). Индивидуальное диалоговое окно создается с помощью метода showModalDialog ().

После анализа документа и его разметки браузер переходит к загрузке внедренных элементов. Проследить последовательность загрузки различных элементов и всего документа можно с помощью события onLoad.

5. События и объекты браузера

Очень важное место в программировании веб-страниц занимают события. События генерируются в результате действий пользователя (щелчков мыши, нажатия клавиш и т. д.), а также изменений состояния окна либо документа (загрузка документа изображений, появление ошибки и пр.). Например, если пользователь щелкает кнопкой мыши на определенном элементе, наступает событие Click. Если же указатель мыши пересекает какую-либо ссылку, возникает событие MouseOver. При загрузке документа в окно браузера происходит событие Load. Разрабатывая динамические веб-страницы, вы можете составить сценарий таким образом, что страница будет реагировать на некоторые из событий. Это делается с помощью специальных программ, которые называются обработчи-

ком событий. Программы обработчиков событий представляют собой фрагменты кода и обычно оформляются в виде функций. Обработчик событий, написанный на JavaScript, вводится в сценарий крайне просто — буквально одной строкой.

События и обработчики событий. Допустим, вы хотите, чтобы при щелчке на определенном абзаце документа появлялось сообщение с указанием темы абзаца. Для этого в тег абзаца <P> нужно вставить событие onclick. Вообще говоря, обработчики событий могут размещаться в различных частях документа:

1. в HTML-тегах в виде атрибутов. Обработчики событий в этих случаях дополняют возможности тегов;

2. в сценарии между тегами <SCRIPT> и </SCRIPT>. При этом обработчики событий входят в состав инструкций JavaScript;

3. в теге < SCRIPT>, что предусмотрено в Internet Explorer.

Всплывание событий в DHTML. Каждый раз, когда вы делаете щелчок мышью или перемещаете мышь, меняете размер окна или загружаете страницу, операционная система генерирует сообщения. Эти сообщения в рассматриваемых нами случаях помещаются в очередь сообщений браузера. Из очереди сообщение доставляется соответствующему элементу, например кнопке или гиперссылке, и элемент генерирует событие. В динамическом HTML (Dynamic HTML или сокращенно DHTML) все элементы могут генерировать события, даже теги заголовков. В основу модели событий в DHTML положено так называемое всплывание событий. Событие, сгенерированное одним элементом, передается вверх по иерархической структуре контейнеров, пока не достигнет уровня самого документа. Благодаря эффекту всплывания событие можно обрабатывать на каждом уровне иерархии контейнеров.

Стандартные события DHTML. Каждое событие имеет имя, указывающее на соответствующее действие пользователя. Так, при щелчке кнопкой мыши на каком-либо элементе возникает событие click, при нажатии и отпускании какой-либо клавиши — событие keypress. К именам событий принято добавлять префикс on, например onClick и onKeypress.

Имена событий в коде HTML обычно пишутся с прописной буквы (onClick, onMouseDown и т. д.). Однако в языке JavaScript, который чувствителен к регистру символов, нужно пользоваться строчными буквами. Чтобы события единообразно обозначались как в тегах HTML, так и в сценариях JavaScript, мы будем использовать только строчные буквы (onclick, onmousedown и т. д.).

События мыши. Перечисляемые ниже события мыши поддерживаются большинством браузеров.

1. onmousedown — событие, возникающее при нажатии любой из кнопок мыши.

2. onmouseup — это событие противоположно onmousedown и возникает при отпускании нажатой кнопки мыши.

3. onclick — событие, которое может быть использовано для исполнения функций, когда пользователь щелкает мышью на каком-либо элементе. Это событие возникает после генерации событий onmousedown и onmouseup и др.).

4. ondbclick — событие, происходящее при двойном щелчке мыши на элементе. Двойной щелчок отвечает случаю, когда пользователь два раза нажимает левую кнопку мыши в течение промежутка времени, определенного системой.

5. onmousemove — это событие отвечает произвольному перемещению указателя мыши по документу (событие непрерывно генерируется при перемещении).

6. onmouseover — событие, возникающее при наведении на область данного элемента.

7. onmouseout — событие, противоположное onmouseover. Оно отвечает удалению указателя мыши из области данного элемента.

8. onselectstart — событие, которое возникает каждый раз, когда пользователь начинает выделять некоторую часть текста, являющегося содержимым данного элемента.

9. onselect — событие, которое возникает следом за onselectstart во время выделения текста. Оно отвечает расширению или сужению области выделения.

10. ondragstart — событие, которое генерируется, когда пользователь наводит указатель мыши на элемент (изображение, ссылку и др.), нажимает кнопку мыши и пытается перетащить элемент в другую часть документа.

События клавиатуры

События, связанные с клавиатурой, позволяют определять момент нажатия или отпускания клавиши и какая именно клавиша была нажата.

1. onkeydown — событие, возникающее при нажатии клавиши. Код нажатой клавиши возвращается свойством keyCode объекта event. Если использовать и другие свойства этого объекта (altKey, ctrlKey и shiftKey), можно определять любую комбинацию клавиш, нажатых пользователем.

2. onkeyup — событие, возникающее при отпускании предварительно нажатой клавиши, данное событие возвращает то же значение свойства keyCode объекта event, что и событие onkeydown.

3. onkeypress — событие, которое возникает при нажатии и отпускании любой клавиши ANSI. Код клавиши возвращается свойством keyCode. Событие onkeypress, как и предыдущие два события, чувствительно к регистру клавиши.

4. onhelp — событие, генерируемое, когда пользователь нажимает клавишу F1, запрашивая справку. Действием по умолчанию для этого события является отображение встроенного файла справки. Это действие может быть отменено, если необходимо отобразить, например, индивидуальный файл справки.

События фокуса

1. onfocus — событие, которое возникает при активизации объекта щелчком мыши или с помощью клавиатуры. Это событие поддерживается для объектов button, checkbox, fileupload, radio, reset, select, submit, text, textarea и window.

2. onblur — событие для объекта, который потерял фокус. Кроме того, это событие возникает при активизации другого приложения, окна или фрейма. Событие onblur содержится в тех же объектах, что и событие onfocus.

Фокус могут получать только интерактивные элементы ввода (ссылки, кнопки, текстовые поля, списки и т. д.) и тело документа. Поэтому, если вы щелкнете, например, на заголовке, событие onfocus получит сам документ, а не заголовок.

События загрузки и выгрузки

В объектных моделях браузеров для отслеживания процессов загрузки и выгрузки документа предусмотрены соответствующие события.

1. onload — наступает, когда браузером проанализирован весь документ и загружены все элементы.
2. onreadystatechange — наступает, когда документ или внедренный объект переходит в состояние завершения загрузки.
3. onunload — возникает при выгрузке документа.
4. onbeforeunload — это событие предшествует событию onunload и дает возможность проверить, действительно ли пользователь хочет покинуть документ. Событие сопровождается появлением на экране запроса на подтверждение выхода с текущей страницы.

Перечисленные выше события загрузки и выгрузки применяются к объектам window и frame.

Способы связывания событий

Под *связыванием событий* понимается установление связи между функцией-обработчиком события и объектом, к которому относится функция. Рассмотрим возможные способы связывания, предусмотренные в JavaScript.

Введение событий в качестве атрибутов. Все стандартные события, перечисленные выше, поддерживаются в HTML и могут быть вставлены в качестве атрибутов в элементы документа. Использование событий как атрибутов позволяет устанавливать непосредственную связь между элементом и его поведением.

Обработчики событий как функции. В примерах предыдущего пункта внутри тегов HTML использованы обработчики событий alert(), document.write() (вы уже знаете, что они представляют собой методы объектов). Обработчики оказываются доступными не из JavaScript-сценариев (как полагается элементам JavaScript), а из тегов HTML.

Однако несколько операторов в обработчике события можно заменить одной функцией. Причем использование функции более предпочтительно, поскольку в ней вы можете задать выполнение более сложных операций, нежели в записи атрибута.

Динамическое связывание событий в сценарии. Когда обработчик события присутствует в элементе HTML, событие обрабатывается в безусловном порядке. Если же вам нужно подключать обработчик события в зависимости от выполнения какого-либо условия, то лучше воспользоваться возможностями сценариев.

Свойство объекта динамически связывается с определенными функциями непосредственно во время выполнения сценария. Для любого события (свойства) может быть назначен свой указатель функции. Указатель функции назначается путем присвоения имени функции свойству объекта.

Отметим, что при назначении указателя функции нужно использовать только имя функции. Круглые скобки и аргументы не нужны. Если же аргументы в назначении указателя присутствуют, то будет вычислена функция, а свойству будет присвоено значение, возвращаемое функцией, но не указатель функции. При этом после вычисления, вероятнее всего, последует диагностика ошибки сценария.

Придерживайтесь простого правила: обработчик событий должен обращаться только к тем объектам, которые определены до него. Например, кнопки Submit и Reset размещаются в нижней части формы, что позволяет отправлять форму на сервер после того, как все элементы формы проанализированы и созданы.

Контрольные вопросы

1. Что такое скрипт?
2. Для чего предназначен язык JavaScript? Приведите определение.
3. Какие существуют варианты встраивания кода JavaScript?
4. На чем основана объектная модель языка JavaScript?
5. Какие встроенные объекты используются в JavaScript?
6. Какие существуют типы переменных JavaScript, чем они отличаются друг от друга?
7. Методы и свойства объекта Mach.
8. Что такое массив? Как создается массив?
9. Способы создания массивов.
10. Назначение объектов String и length.
11. Методы объекта Array.
12. Какие существуют типы переменных JavaScript, чем они отличаются друг от друга?
13. Какие основные операторы языка существуют в JavaScript?
14. Что такое объект window?
15. Как происходит управление окнами?
16. Как создать простые диалоговые окна?

Лекция 17. Серверное программное обеспечение

План:

1. Основы разработки сетевых приложений.
2. Приёмы проектирования веб-служб.

1. Основы разработки сетевых приложений

Исходно WWW состояла из HTML, URL и HTTP.

HTML — язык форматирования, используемый для представления содержания в Web.

URL — это адрес, используемый для получения содержимого в формате HTML (или каком-либо ином) с веб-сервера.

HTTP — это язык, который понятен веб-серверу и позволяет клиентам запрашивать у сервера документы. Программа-клиент устанавливает TCP-соединение с сервером (стандартный номер порта — 80) и выдает ему HTTP-запрос. Сервер обрабатывает этот запрос и выдает HTTP-ответ клиенту.

Необходимо знать структуру HTTP-запроса и ответа

Структура HTTP-запроса:

- HTTP-запрос состоит из заголовка запроса и тела запроса, разделенных пустой строкой;
- тело запроса может отсутствовать;
- заголовок запроса состоит из главной (первой) строки запроса и последующих строк, уточняющих запрос в главной строке.
- последующие строки также могут отсутствовать.

Запрос в главной строке состоит из трех частей, разделенных пробелами.

- *Метод* (иначе говоря, команда HTTP): GET — запрос документа. Наиболее часто употребляемый метод. HEAD — запрос заголовка документа. POST — для передачи данных CGI-скриптам. Сами данные следуют в послеующих строках запроса в виде параметров; PUT — разместить документ на сервере. Используется редко. Запрос с этим методом имеет тело, в котором передается сам документ.
- *Ресурс* — это путь к определенному файлу на сервере, который клиент хочет получить (или разместить — для метода PUT). Если ресурс — просто какой-либо файл для считывания, сервер должен по этому запросу выдать его в теле ответа. Если же это путь к какому-либо CGI-скрипту, то сервер запускает скрипт и возвращает результат его выполнения.
- Версия протокола — версия протокола HTTP, с которой работает клиентская программа. Пример: простейший HTTP-запрос может выглядеть следующим образом: GET / HTTP/1.0 — запрашивается корневой файл из корневой директории веб-сервера.
- Строки после главной строки запроса имеют следующий формат: Параметр: значение.
- Таким образом, задаются параметры запроса. Это является необязательным, все строки после главной строки запроса могут отсутствовать; в этом случае сервер принимает их значение по умолчанию или по результатам предыдущего запроса (при работе в режиме Keep-Alive).

Структура HTTP-ответа

- имеет заголовок и тело, разделенные пустой строкой. Заголовок состоит из основной строки и строк параметров. Основная строка запроса состоит из 3 полей, разделенных пробелами:

1) версия протокола — аналогичен соответствующему параметру запроса;

2) код ошибки — кодовое обозначение «успешности» выполнения запроса. Код 200 означает «все нормально» (OK);

3) словесное описание ошибки — «расшифровка» предыдущего кода. Например, для 200 это OK, для 500 — Internal Server Error.

Наиболее употребительные параметры:

1) Connection — аналогичен параметру запроса;

2) Content-Type ("тип содержимого") — обозначение типа содержимого ответа. Некоторые типы содержимого:

text/html — текст в формате HTML (веб-страница);

text/plain — простой текст (аналогичен "блокнотовскому");

image/jpeg — картинка в формате JPEG;

application/octet-stream — поток "октетов" (байт) для записи на диск;

3) Content-Length ("длина содержимого") — длина содержимого ответа в байтах;

4) Last-Modified ("Модифицирован в последний раз") — дата последнего изменения документа.

Спецификация CGI

CGI (Common Gateway Interface — общий шлюзовой интерфейс) — это набор правил, согласно которым программы на сервере могут через веб-сервер посылать данные клиентам. Спецификация CGI ввела изменения в HTML и HTTP, добавив формы и параметры запроса — данные для этой CGI-программы. Распространенные приложения CGI включают в себя:

- динамические сайты — генерируются одной CGI-программой;
- поисковые механизмы, находящие документы с заданными пользователем словами;
- гостевые книги и доски объявлений, в которые пользователи могут добавлять свои сообщения;
- бланки заказов, анкеты;
- извлечение информации из размещенной на сервере базы данных.

Запоминание состояния

- На стороне клиента — cookie, механизм позволяет показывать сне общее количество посещений сайта, а сколько раз обращался к сайту именно он (учет по IP-адресу или по переменной для хранения информации на диске пользователя)
- На стороне сервера — сеансовые переменные. Для передачи их ID применяют:

1) cookie;

2) URL;

3) теги <INPUT> типа HIDDEN в формах.

Меры безопасности

Сам протокол CGI достаточно защищен. CGI-программа получает данные от сервера через стандартное устройство ввода или переменные окружения, и оба эти метода являются безопасными. Плохо написанная CGI-программа может позволить злоумышленнику получить доступ к системе сервера. Одно из решений состоит в синтаксическом анализе поступивших от формы данных с целью поиска злонамеренного содержания. CGI-программы используются в качестве интерфейса:

- к серверам баз данных, таким как MySQL
- к настольным базам данных, таким как Microsoft Access,

а также работают с плоскими текстовыми файлами, являющимися самыми простыми базами данных.

2. Приемы проектирования веб-служб

Что такое веб-служба. Веб-служба (Web Service) — это приложение или блок находящегося на веб-сервере выполняемого кода, функционирование которого основано на применении стандартных форматов XML. Поиск этого кода, его извлечение и получение посредством него требуемого результата выполняется в среде .NET Framework. Вызывается веб-служба .NET так же просто, как и локальная функция.

Веб-служба .NET — это не объект (во всяком случае, не в его традиционном представлении). Веб-метод является, по сути, независимым, «атомарным» и не имеющим постоянного местонахождения. Веб-служба больше подобна библиотеке функций в DLL и ее сложно ограничить рамками объектно-ориентированной абстракции. Это упрощение в значительной мере и обеспечивает преимущества веб-служб. Поскольку веб-службы не ограничены конкретной технологией (безопасности, управления или транспортировки), они могут быть использованы почти в любом разрабатываемом сценарии, что существенным образом отличает их от предыдущих технологий, таких как COM и CORBA.

Для чего нужны веб-службы. Веб-службы предоставляют способ совместного использования программных функций. Их даже можно назвать «COM для Web», хотя в основе работы этих систем лежит совсем другая технология. Веб-служба не является продуктом для конечного пользователя. Она представляет собой основанное на компонентах приложение, позволяя многократно использовать свою функциональность в различных средах и на клиентах разных типов. Пользователем веб-службы всегда является другое приложение.

Для создания серьезного проекта необходимо как минимум два человека: один будет заниматься непосредственно программированием, а второй разрабатывать дизайн будущего проекта. Такое разделение целесообразно практически всегда. Во-первых, универсальные специалисты встречаются, довольно редко: кто-то лучше умеет программировать, а кто-то — рисовать. Кроме того, любой проект необходимо документировать: хотя бы написать простенький Readme с инструкциями по установке проекта на сервер хостера. Для этого вам понадобится еще один человек — технический писатель, который доступным языком объяснит пользователю, как работать с вашим проектом. Получается, что при разработке серьезного проекта происходит разделение на техническую и дизайнерскую части. К технической группе относятся программисты и технические писатели. К группе дизайнеров относится старший дизайнер и его помощники. Вы можете даже отдать разработку дизайна на сторону — в какую-нибудь профессиональную студию веб-дизайна, чтобы сосредоточиться только на проектировании и воплощении своего сайта.

Все сказанное в значительной степени применимо и к случаю, когда проект создается и поддерживается одним человеком. В этом случае сам разработчик должен разделять два процесса: написание кода и создание дизайна. Важно, чтобы эти два процесса были как можно обособлены. Практически во всех современных языках программирования высокого уровня предусмотрена возможность создания модулей. Основная программа подключает модуль, чтобы в процессе дальнейшей работы вызывать реализованные в этом модуле функции. Разработчик выигрывает вдвойне: во-первых, уменьшается размер кода основной программы, и программа выглядит читабельнее, а во-вторых, вы можете написать несколько программ, используя один и тот же модуль.

Создание библиотекаря. Определим требования к библиотекарю:

- библиотекарь должен загружать модуль (библиотеку функций) только один раз;
- библиотекарь должен обеспечивать удобную настройку путей поиска каталогов, в которых хранятся модули;
- загрузка модуля должна осуществляться с помощью какой-нибудь функции. Мы ее назовем Load ();
- библиотекарь должен загружаться автоматически.

Задача программиста упрощается при использовании библиотечного модуля, позволяющий загружать отдельные модули проекта по мере необходимости. Как упростить задачу дизайнеру проекта? Мы отделим код PHP-сценария от его HTML-шаблона, под которым понимается чистая веб-страница до внедрения в нее PHP-кода. Именно разделение PHP-кода и HTML-шаблона позволит спокойно заниматься созданием дизайна веб-страницы в любом HTML-редакторе, а PHP-скрипты подключать потом к уже готовой странице.

Контрольные вопросы

1. Из чего состоит структура HTTP-запроса?
2. Из чего состоит структура HTTP-ответа?
3. Что такое веб-служба?
4. Для чего нужны веб-службы?
5. Какие требования предъявляются к созданию библиотекаря?

Лекция 18. Основы программирования на PHP

План:

1. Синтаксис языка PHP.
2. Элементы и выражения языка ИС.
3. Функции PHP.
4. Работа с формами.
5. Работа с базами данных.

1. Синтаксис PHP

Синтаксис PHP заимствован непосредственно из C. Java и Perl также повлияли на синтаксис данного языка. Есть 4 способа выхода из HTML и перехода в «режим PHP кода»:

1. <? echo("простейший способ, инструкция обработки SGML\n"); ?>

2. <?php echo("при работе с XML документами делайте так\n"); ?>
3. <script language="php"> echo ("некоторые редакторы не любят обрабатывающие инструкции"); </script>
4. <% echo("От PHP 3.0.4 можно факультативно применять ASP-теги"); %>

Инструкции разделяются так же, как в С или Perl, точкой с запятой. Закрывающий тег (?>) тоже подразумевает конец утверждения, поэтому следующие записи эквивалентны: <php echo "Это тест"; ?> или <php echo "Это тест" ?>

Структура PHP-программы

Напишем первую программу (скрипт) на PHP, которая будет здороваться с нами. Текст программы first. php (сценария или скрипта):

```
<? echo "<html> <body><h1>";
echo "Hello, $my_name";
echo "</h1></body></html>";
?>
```

Прежде чем запустить программу, ее нужно правильно «установить» на сервере. Для этого необходимо сохранить сценарий под именем first.php и скопировать его в каталог DocumentRoot вашего сервера. В отличие от CGI-программ, сценарии PHP являются для сервера обыкновенными документами и их не нужно помещать в каталог cgi-bin. Введите в адресную строку браузера: http://localhost/first.php?my_name=Denis

В результате в окне браузера вы увидите Hello, Denis.

Если теперь страницу, отображенную в браузере, просмотреть в виде HTML-кода, то можно увидеть следующее: <html><body><hl>Hello, Denis</hl></body></html>

Как видите, PHP-код выполнился, подставил нужное значение переменной $my_name и выдал готовую HTML-страницу. Пример был дан для запуска PHP-сценария через CGI-интерфейс, передающий переменной $my_name значение Denis. Однако любую PHP-программу можно запустить и так: php file.php. При этом CGI-переменные будут недоступны — ведь вы запустили программу из командной строки. Вводить текст программы лучше в текстовом редакторе, поддерживающем подсветку синтаксиса PHP.

Теперь рассмотрим саму программу. Код PHP заключается в специальные теги <? и ?>. После тега начала сценария (<?) следует первый оператор — echo. Оператор echo осуществляет вывод информации. Его с полной уверенностью можно назвать самым главным оператором — ведь без него был бы невозможен вывод информации в браузер и пользователь не увидел бы результата работы сценария.

Выводимая оператором echo строка заключается в кавычки. Первый оператор выводит теги HTML, BODY и H1. Второй оператор echo «Hello, $my_name»; выводит слово Hello и значение переменной $my_name. Перед именами всех переменных ставится знак доллара. Присваивание переменной какого-либо значения осуществляется оператором присваивания, который обозначается значком «=» (равно). Примеры использования переменных:

$1=0; $d=7 ; $i=5 + $d; // $i = 12

Модифицируем немного наш текст программы first. php, чтобы он выглядел так:

```
<?
    echo "<html><body><hl>";
    echo "Hello, $my_name";
    echo "<br> $var";
    echo "</hl></body></html>";
?>
```

Если мы не передадим программе параметр $var, то есть, как обычно, запустим программу только с параметром $my_name, то в окне браузера мы увидим сообщение о том, что переменная $var не определена. Для того чтобы избавиться от этой ошибки, непосредственно перед оператором echo "
 $var"; добавьте строку: $var=" ";

Любой другой интерпретатор, например Perl, в подобной ситуации сгенерировал бы сообщение 500: «Ошибка сценария». Интерпретатор PHP решает проблему просто и элегантно, избавляя тем самым программиста от просмотра многотомных журналов веб-сервера. Однако PHP-сценарии можно записывать и по-другому.

Сценарий из листинга first. php можно оформить так:

```
<html><body><hl>
<? echo "Hello, $my_name"; ?>
</hl></body></html>
```

Переменные

Переменная — это область оперативной памяти, доступ к которой осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных, в программе на языке PHP перед именем переменной обязательно должен стоять знак доллара. По этому символу интерпретатор выделяет переменные из текста программы. В официальной документации PHP сказано, что имена переменных могут состоять не только из латинских букв, но из любых символов, код которых больше 127. Это означает, что имена переменных могут содержать русские буквы. Однако я настоятельно советую вам не использовать русскоязычные имена — ведь в разных кодировках буквы кириллицы имеют разные коды.

Определим правила задания переменных в PHP:

- имя переменной должно начинаться со знака доллара $;
- имя переменной не должно содержать никаких других символов, кроме символов латинского алфавита, цифр и знака подчеркивания.

Имена переменных в PHP чувствительны к регистру символов, то есть $a и $A — это совершенно разные переменные. Объявлять переменную можно в любом месте программы, но до места первого ее использования.

Типы данных. Переменные в PHP могут содержать данные любого типа. Исключение составляют константы, которые могут содержать только число или строку. Указателей, знакомых вам по языку C (то есть переменных, содержащих не значение, а адрес в памяти, по которому находится это значение), в язы-

ке PHP не существует. При присваивании переменная копируется «один в один», какую бы структуру она ни имела. Предположим, что у нас есть две переменные: массив и строка. Если первой переменной мы присвоим значение второй, то наш массив будет навсегда потерян, а переменная, которая содержала массив, будет содержать строку — это и означает, что копируется не только значение переменной, как в других языках, но и структура. Рассмотрим следующее объявление переменных:

```
$a = 10;
$A = "Hello";
```

Как вы видите, при объявлении переменных мы не указали их тип. Выбор типа осуществляется самим интерпретатором. Однако иногда PHP может ошибиться и сопоставить переменной неправильный тип.

Таблица 35 — Типы переменных в PHP

Тип переменной	Описание
Integer	Целое число (32 бита). Данный тип соответствует типу longint в Pascal
double (или float)	Вещественное число очень большой точности
String	Строка. Это очень важный тип данных, поскольку в большинстве случаев сценарии занимаются обработкой строк
Array	Массив
Object	Язык PHP поддерживает принципы объектно-ориентированного программирования, хотя и самые простые. Программист может создавать свои объекты и использовать их в программе. Доступ к отдельным элементам объекта осуществляется с помощью оператора ->
Bool	Логический тип данных. Переменные этого типа могут принимать только одно из двух значений: true (истина) или false (ложь)

Функции определения и задания типа переменных

Язык PHP предоставляет много средств для определения типа переменных. Вы можете использовать следующие функции для определения типа:

is_int ($x) или is_integer ($x) — возвращает true, если переданная переменная — целое число;

is_double($x) или is_float ($x) — возвращает true, если переданная переменная — вещественное число;

is_string ($x) — возвращает true, если переданная переменная — строка;

is_array ($x) — возвращает true, если переданная переменная — массив;

is_object ($x) — возвращает true, если переменная является объектом;

is_bool($x) — возвращает true, если переменная объявлена, как логическая;

gettype($x) — возвращает строку, описывающую тип переменной: integer, double, string, object, array, bool или unknowntype, если невозможно определить тип. Тип переменной определить невозможно, когда он не встроен в PHP, а добавляется с помощью модулей, расширяющих возможности языка.

Если PHP неправильно определил тип переменной, можно указать его явно. Для этого используется функция settype ($x, $type), где $type — это одна из строк, возвращаемых функцией get type (). Например: settype($x,"double");

Функция возвращает значение false, если невозможно привести тип переменной $x к указанному типу $type.

Логические переменные и их особенности в PHP

Большинство функций для определения типа (все, кроме gettype ()) возвращают логическое значение true или false. В PHP логически эквивалентны любое не равное нулю число, любая непустая строка, значение true: все они являются истиной. Значение false, пустая строка, нуль — это ложь.

Сценарий logical. php:

```
<?
echo false;
echo true;
?>
```

Первый оператор выводит пустую строку, а второй выведет «1».

Константы. Константы содержат постоянные значения. В отличие от переменной, вы не можете изменить значение, присвоенное константе при ее объявлении. Константы удобно использовать для хранения значений, которые не должны изменяться во время работы программы (например, имя разработчика программы).

Для определения константы служит функция define. Эта функция имеет следующий формат:

define ($name, Svalue, $case_sen)

где $name — имя константы, $value — значение константы, а $case_sen — необязательный параметр логического типа, указывающий, следует ли учитывать регистр символов (true) или нет (false).

Примеры: define ("pi", 3.14, true) ; echo pi;

```
define ("pi", 3.14, true) ;
if (defined("pi")==true) echo "Константа объявлена";
```

Таблица 36 — Стандартные константы

Имя	Значение
FILE (два знака подчеркивания до FILE и два после)	Содержит имя программы, которая выполняется в данный момент
LINE	Номер строки, которую сейчас обрабатывает интерпретатор
PHP_OS	Имя и версия операционной системы, под которой запущен PHP
PHP_VERSION	Версия PHP
TRUE, true, FALSE, false	Уже знакомые нам логические константы

Операции с переменными. Мы уже рассмотрели две важнейшие операции с переменными — определение и установку типа переменной. Поэтому нам осталось рассмотреть всего три операции:

1) присвоение значения;

2) проверка существования переменной;

3) уничтожение переменной. Вы можете спросить, а как же арифметические и другие операции? Все остальные операции специфичны для конкретного типа переменной.

Пример:

```
$x = "10"; // строка
$y = "2"; //и это строка
echo $x*$y;
```

Сценарий вместо сообщения об ошибке выведет на экран число 20. Как видите, если строка допускает преобразование типа, то PHP работает со строками как с обыкновенными числами.

Присвоение значений. Оператор присваивания

Оператор присваивания позволяет присвоить переменной некоторое значение.

```
$x= 4 ;
$y =$x;
$x +=10; // эквивалентно $x = $x + 10; $x *=10; // эквивалентно $x = $x*10;
```

Проверка существования переменной

Проверка существования переменной — это очень удобная возможность языка программирования. Благодаря возможности проверки существования переменной вы можете проверить, передан ли сценарию определенный параметр, а только потом начинать с ним работать. Рассмотрим два сценария.

Сценарий 1:

```
<? $x = $my_name;?>
```

Сценарий 2:

```
<? if (isset($my_name)) {$x = $my_name; } ?>
```

В первом случае мы сразу начинаем работать с переменной. Если по каким-либо причинам интерпретатор не передал сценарию параметра m_name, мы получим сообщение о том, что переменная не определена. Во втором сценарии мы с помощью функции isset ($my_name) проверяем существование переменной $my_name и только потом начинаем с ней работать. В результате ваш сценарий не прекратит свою работу из-за банальной ошибки пользователя, если тот забудет ввести свое имя.

Удаление переменных. Чтобы не занимать память, можно удалить ненужные нам переменные. Это делается с помощью функции unset (). Эта функция удаляет указанную в скобках переменную из памяти, и программа продолжает выполняться, как будто бы эта переменная вообще не была инициализирована.

```
<?
 $a = // читаем_большой_файл;обрабатываем_файл;
unset($a); // освобождаем память ?>
```

Эта функция особенно полезна и даже необходима при обработке больших объемов данных, чтобы они не оставались в памяти компьютера и не замедляли его работу.

2. Элементы и выражения языка ИС

Операции в PHP. Выражения являются теми «кирпичиками», из которых состоят PHP-программы. Практически все, что вы пишете в программе, является выражением. Под выражением понимается любая конструкция, у которой есть значение. Самое простое выражение — это константа, стоящая в правой части оператора присваивания: $x = 100; где $x — это переменная, = — оператор присваивания, а 100 — это и есть выражение. Его значение — число 100.

Выражением может служить и переменная, если ей сопоставлено определенное значение. Например, $x = 5; $y =$x; В первом операторе выражением является константа 5, во втором — переменная $x, так как ранее ей было присвоено значение 5. $y = $x тоже является выражением! Ведь не вызывает сомнения то, что его значение равно 5. Это, в свою очередь, позволяет использовать следующие операторы: $y=$x=5; или $y=($x= 5); Таким образом, оператор присваивания можно использовать в середине выражения:

$y =1; $x =100*($y=2)*$y; Очевидно, что переменной $x будет присвоено значение 100*2*2 = 400.

Виды операций

1) *Арифметические операции:*

X + Y — сложение;
X — Y — вычитание;
X * Y — умножение;
X / Y — деление;
X % Y — остаток от деления X на Y.

К арифметическим операциям можно также отнести операции инкремента и декремента: операция инкремента $x++ увеличивает значение на 1; операция декремента $x— уменьшает значение на 1.

2) *Битовые операции* — эти операции предназначены для установки или снятия групп битов целочисленной переменной. Ведь любое число — это просто последовательность битов.

3) *Логические выражения* — это выражения, результатом которых может быть или истина, или ложь.

Примеры логических выражений:

$x = true; // $x = истина
$x = $y < 0; // $x = true, если $y < O
$x = $y == 0; // $x = true, если $y = O

Операторы сравнения:

= = — равно;
!= — не равно;
< — меньше;
> — больше;
>= — больше или равно;
<= — меньше или равно.

В PHP нельзя сравнивать массивы и объекты, разрешается сравнивать только скалярные переменные.

4) Логические операции:

!x — возвращает true, если x — ложь, и наоборот (отрицание, NOT);

x && y — возвращает true, если x и y истинны (умножение, И, AND);

x II y — возвращает true, если истинно хотя бы одно из значений или все они истинны (сложение, ИЛИ, OR).

Строки и строковые выражения

Рассмотрим небольшой пример:

$s = "Hello";

echo "$s"; // строка в кавычках

echo '$s'; // строка в апострофах

Оба оператора echo выведут строки. Первый оператор echo выведет строку «Hello», а второй — «$s». Между строками в кавычках и в апострофах существует большая разница. Если строка заключена в апострофы, то все символы трактуются как есть. Строки в кавычках позволяют, кроме всего прочего, вывести значение переменной, поэтому никогда не используйте строку в апострофах для этой цели.

Предположим, что у нас есть переменная $a = 10. Проанализируем вывод двух операторов echo: echo "$a"; echo '$a';

Первый оператор выведет 10 — значение переменной $a. Второй напечатает то, что заключено между апострофами, — $a.

Строковых операций в PHP всего две:

- $sl . $s2 — конкатенация (слияние) двух строк;
- $s1 [n] — обращение к символу с номером n в строке.

Все остальные действия над строками выполняются стандартными функциями.

Например, присваивание: $S=”hello”;

Оператор эквивалентности (= = =) — сравнивает равенство значения (if ($x===$s) echo “x=s”)

Ссылки. В PHP нет такого понятия, как указатель. Ссылки бывают двух типов: символические и жесткие.

Жесткая ссылка — это просто псевдоним другой переменной. Для создания жесткой ссылки используется оператор &:

```
<?
$x = 77;
$link = &$x; $link = 66;
echo $x;  ?>
```

Программа выведет число 66. Для разрыва связи между переменной и ссылкой используется функция UnSet: UnSet($link);

Саму переменную нельзя удалить до тех пор, пока на нее ссылается хотя бы одна ссылка.

Символическая ссылка — это обыкновенная переменная, содержащая имя другой переменной. Для доступа к значению ссылки используется двойной знак доллара: <? $x = 77; $link = "x"; // символическая ссылка на переменную $x echo $$link; // выводит 77 echo $link; // выводит x $$link = 66; // аналогично $x = 66 echo $x; // выведет 66 ?>

Основные конструкции языка

Любой скрипт PHP состоит из последовательности операторов. Оператор может быть присваиванием, вызовом функции, циклом, условным выражением или пустым выражением.

Условный оператор. Синтаксис конструкции if-else таков:

1. if (логическое выражение) оператор_1;
else оператор_2;

2) if (логическое выражение) оператор_1;

Если вам нужно выполнить не один оператор, а целую группу, тогда эти операторы следует поместить в блок операторов {}:

3) if (логическое выражение)
{
оператор_1;
…
оператор_i }
else { оператор_i+1;
….
оператор_n;
}
Пример. if ($a>5) echo "a>5"; else echo "a<=5";

4) if (логическое выражение 1) оператор_1;
elseif (логическое выражение 2) оператор_2;
else оператор_3;

Циклы. Цикл позволяет повторить один или несколько операторов определенное, зависящее от условия цикла, количество раз. Повторяемые операторы называются телом цикла. Один проход цикла называется итерацией. PHP поддерживает три вида циклов.

1) Цикл с предусловием while

Принцип работы цикла с предусловием: вычисляется значение логического выражения. Если значение истинно, выполняется тело цикла, в противном случае переходим на следующий за циклом оператор. Синтаксис цикла с предусловием: while (логическое выражение) инструкция; Простейший пример цикла:

```
<?
$i=0;
while($i++ <10) echo $i;
?>
```

Данная программа выведет строку 12345678910. Этот же цикл можно было бы записать по-другому:

```
$i = 0;
while ($i<10)
(
$i ++; // увеличение счетчика
echo $i;
}
```

Примеры: 1. $i = 1; while ($i <= 10) {print $i++;}

1. $i = 1; while ($i<= 10): print $i; $i++;

endwhile;

2) Цикл с постусловием do-while. Цикл с постусловием отличается от цикла с предусловием тем, что сначала выполняется тело цикла, а только потом уж проверяется условие. Таким образом, тело цикла хотя бы один раз, но будет обязательно выполнено.

Синтаксис цикла такой:

```
do
    {
    // тело цикла
    }
 while (условие);
```

Попробуем с помощью цикла с постусловием получить строку 12345678910. Поскольку в этом цикле сначала выполняется тело, а потом проверяется условие, то нам нужно установить начальное значение счетчика в единицу:

```
 <?
$i = 1;
do echo $i; while ($i++ <10);
 ?>
```

3) Цикл со счетчиком for. Синтаксис цикла for:

```
for (команды_инициализации; условие; команды_после_итерации)
    {тело цикла }
```

Пример:

```
<? for ($i=0; $i<10; $i++) echo $i; ?>
for ($i = 1; $i <= 10; $i++) print $i;
for ($i = 1;;$i++)
{if ($i > 10) break;
print $i;}
```

Управлять количеством итераций цикла позволяют операторы break и continue.

Рассмотрим следующие примеры:

```
<?
$i = 0;
while ($i++ < 10)
{
if ($i==3) break;
echo "Итерация $i\n";
}
?>
```

В этом примере после второй итерации работа цикла будет прервана и программа выведет строки:

Итерация 1

Итерация 2

Цикл перебора массивов foreach. В четвертой версии PHP появился еще один вид цикла — foreach, предназначенный специально для перебора массивов. Синтаксис данного цикла таков: foreach (массив as $ключ=>$значение) операторы;

Операторы, содержащиеся в теле цикла, будут выполнены для каждого элемента массива. Переменная $ключ будет содержать имя ключа элемента, а переменная $значение — значение элемента.

Оператор switch-case (конструкция выбора). Оператор служит для выбора действий в зависимости от значения указанного выражения. Конструкция switch-case является аналогом if-else. Конструкцию выбора нужно использовать, если предполагаемых вариантов много (скажем, больше 5) и для каждого варианта нужно выполнить специфические действия. Синтаксис switch-case таков:

```
switch (выражение)
{ case значение1 : команды_1; [break;]
 …….
 case значениеN : команды_N; [break;)
 [default: команды_по_умолчанию; [break;]]
 }
```

Пример 1:

```
<? $age=21;
switch ($age)
{
 case 20 : echo "Вам двадцать лет\n";
 case 21 : echo "Вам двадцать один год\n";
 case 22 : echo "Вам двадцать два года\n";
default: echo "Вам $age лет\n";
}
 ?>
```

Если переменная $age равна 21, то будут выведены строки: Вам двадцать один год Вам двадцать два года Вам 21 год

Пример 2:

```
switch ($i) {
 case 0: print "i = 0"; break;
 case 1: print "i = 1"; break;
 case 2: print "i = 2"; break;
  default: print "i!= 0, 1 or 2";
 }
```

Инструкция require. Инструкция require позволяет включить код до выполнения нашего сценария. Общий синтаксис такой: require "имя_файла";

Очень удобно использовать эту инструкцию для включения HTML-заголовков.

Например, Файл header.html:

```
<html><head>
<title>My Company's Official Web Page</title>
</head>
<body>
```

Сценарий: require "header.html" /* тело документа */

Инструкция include вставляет и выполняет содержимое указанного файла. Это случается каждый раз, когда встречается оператор INCLUDE, так что вы можете включить этот оператор внутрь цикла, чтобы включить несколько файлов:

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++) { include($files[$i]); }
```

include() отличается от require() тем, что include выполняется каждый раз при его встрече, а require() заменяется на содержимое указанного файла безотносительно, будет ли выполнено его содержимое или нет.

3. Функции PHP

Подпрограмма — это специальным образом оформленный фрагмент программы, к которому можно обратиться из любого места внутри программы. Подпрограммы существенно упрощают жизнь программистам: они улучшают читабельность исходного кода, сокращают исходный код (не нужно писать один и тот же фрагмент несколько раз). В языке PHP подпрограммы называются функциями.

При наличии стандартных функций языка возникает необходимость создания своей собственной функции, выполняющей нужные именно нам действия. Такие создаваемые самостоятельно функции и называются пользовательскими.

Особенности использования функций в PHP:

- вы можете использовать параметры, установленные по умолчанию. Это, в частности, позволяет вызывать одну и ту же функцию с переменным числом параметров;
- функция может возвращать значение любого типа;
- область видимости переменных внутри функции, как и в других языках программирования, древовидная;
- вы можете вернуться из функции с помощью инструкции return;
- можно изменять значения переменных, переданных в качестве аргумента.

Преимущество PHP-функций. В языке C++ появился механизм перегрузки функций, но параметры по умолчанию — это более гибкое средство. Ведь в C++ для каждой перегруженной функции нам нужно заново писать ее тело. А в языке PHP этого делать не нужно. В этой главе мы обязательно рассмотрим параметры по умолчанию.

Однако у PHP-функций, кроме преимуществ, есть и недостатки, причем существенные. *Первый недостаток* связан с объявлением локальной функции. В PHP вы не можете объявить локальную функцию, как это можно делать в других языках. То есть вы не можете создать функцию внутри другой функции таким образом, чтобы первая (вложенная) функция была видна только во второй функции. В PHP вложенная функция будет видна всей программе, то есть не будет локальной.

Рассмотрим следующий пример:

```
<??
 function first ()
  { echo "First";
 function Second() {
 echo "Second"; } }
 First(); second();
  ?>
```

Сценарий напечатает «FirstSecond». Это означает, что якобы локальная функция доступна во всей программе.

Второй недостаток связан с областью видимости функций. Но сначала немного скажем о глобальных и локальных переменных.

Глобальные переменные — это переменные, которые видны и доступны всей программе со всеми входящими в нее подпрограммами (функциями).

Локальные переменные, определенные внутри подпрограмм, видны и доступны только внутри той подпрограммы (функции), в которой они определены. Так вот в PHP все объявленные и используемые в функции переменные по умолчанию локальны для функции. Это значит, что, в отличие от других языков программирования, из тела функции невозможно изменить значение глобальной переменной. Если вы в функции будете использовать переменную с именем, идентичным имени глобальной переменной, то это никакого отношения к глобальной переменной иметь не будет. Просто в функции будет создана локальная переменная с таким же именем, невидимая за пределами этой функции. Следующий пример подтверждает данный факт:

```
 <? $i = 10;
 function change()
 { $i = 5;
 echo "i = $i<br>\n";
}
change () ;
 echo "i = $i";
?>
```

Сценарий сначала напечатает число 5, а потом число 10.

Пользовательские функции. Функция может быть объявлена в любой части программы (в пределах операторов <? и ?>), но до места первого ее использования. Другими словами, не нужно никакого предварительного объявления, как в других языках программирования, в частности в С. Дойдя до определения функции, транслятор проверит корректность определения и оттранслирует определение функции во внутреннее представление, но транслировать код он не станет. Для объявления функций используется такой синтаксис:

function Имя (аргумент1 [=значение1], ..., аргумент1[=значение1])
{тело функции}

Объявление функции начинается со служебного слова function, за которым следует имя функции, после имени — список аргументов в скобках. Тело

функции заключается в фигурные скобки и может содержать произвольное число операторов.

Передача функции параметров. В большинстве языков программирования поддерживаются два способа передачи функции аргументов: по значению и по ссылке. Не является исключением и PHP. Параметр, переданный по значению, доступен внутри функции только для чтения: функция может его использовать, но не может изменить. В качестве такого параметра необязательно указывать переменную: можно передать само значение. Отсюда название — параметр-значение.

Следующий фрагмент кода выведет строку «Параметр = 3» дважды:

```
function func($a) j
echo "Параметр = $a\n" ; }
$b = 3; func (3) ; func($b);
```

Параметром, переданным по ссылке, функция может распоряжаться как обычной переменной, отсюда название: параметр-переменная. Очевидно, что в качестве такого параметра нельзя передавать значение, нужно обязательно передать саму переменную.

Для указания того, что данный параметр передается по ссылке, нужно при объявлении функции предварить имя переменной символом амперсанда.

```
<?
  $First = $Second = 5;
  echo "<PRE>";
  function fl($f, &$s)
  {
  echo "Изменяем параметры\n";
  $f = 7; $s = 10;
  echo "First = $f, Second = $s\n"; }
  echo "Значение переменных до вызова функции\n";
  echo "First = $First Second = $Second\n";
  fl($First, $Second);
  Стандартные функции echo "First = $First Second = $Second\n";
?>
```

Сначала переменные $First и $Second равны 5. Затем мы в теле функции изменяем оба параметра — обыкновенный и переданный по ссылке.

Стандартные функции

Работа с датой и временем. Практически ни один серьезный проект не обходится без обработки дат и времен, поэтому не уделить внимание функции date () просто невозможно.

Функция **date (string $format [, int $timestamp])** принимает два параметра:

- формат даты/времени (строка);
- сама дата (или время). Это целое число, представляющее собой количество секунд, прошедших с начала эры Unix — с полуночи (по Гринвичу) 1 января 1970 года.

Функция форматирует указанную временную метку (либо текущую дату, если параметр $timestamp не задан) согласно указанной строке формата и возвращает полученную строку. Строка формата может содержать обычный текст, а также

модификаторы, которые при выводе заменяются соответствующими значениями (по 2 цифры: d — номер дня; .m — номер месяца; Y — год (4 цифры) и др.).

Примеры: echo date("Сегодня d.m.Y"); echo date('Дата создания файла d.m.Y', filectime ('test.php'));

При обработке введенных параметров очень полезной окажется функция checkdate (), которая проверяет корректность введенной даты. Функции нужно передать три параметра — номер месяца, число (день) и год (именно в такой последовательности).

Математические функции

Таблица 37 — Математические функции PHP

Функция	Назначение
acos(float $x)	Вычисляет арккосинус аргумента $x
asin(float $x)	Вычисляет арксинус аргумента $x
atan(float $x)	Вычисляет арктангенс аргумента $x
atan2(float $x, float $y)	Вычисляет арктангенс числа $x/$y и возвращает результат в радианах
sin(float $x)	Вычисляет синус аргумента Sx. Аргумент задается в радианах
cos(float $x)	Вычисляет косинус аргумента $x. Аргумент задается в радианах
tan(float $x)	Вычисляет тангенс аргумента $x. Аргумент задается в радианах
pi()	Возвращает число Пи. Вместо функции pi() м. использовать M_PI
sqrt(float $x)	Возвращает квадратный корень $x.
log(float $x)	Возвращает натуральный логарифм $x
exp(float $x)	Возвращает экспоненту аргумента
pow(float $x, $a)	Возвращает $x в степени $a

Генератор случайных чисел. Чаще всего генерация случайных чисел нужна для отображения баннеров или криптографии. Для генерации случайных чисел служат функции mt_rand() и mt_srand (). Функция mt:_rand} (int $from, int $to=RAND_MAX) возвращает случайное число в диапазоне от $from до $to. Если второй параметр не задан, используется значение по умолчанию — константа RANDMAX. Значение данной константы можно узнать с помощью функции mt_getrandmax () . В качестве примера сгенерируем 10 случайных чисел от 0 до 100: for ($i=0; $i<10; $i++) echo mt_rand(0, 100)." ";

Запуск внешних программ. Большинство задач не требуют вызова внешних программ, поскольку без этого можно обойтись, используя стандартные функции PHP. Однако иногда удобнее вызвать внешнюю программу, чем использовать функции PHP. Самый простой пример — вывод оглавления каталога.

$dir = ‘ls’; // вывод оглавления текущего каталога echo "<pre> $dir </pre>";

Функции для работы с файлами.

Все функции делятся на группы.

1. Функции манипулирования файлами (copy(), rename(), unlink() — удаление). Пример: copy (string $s, string $d) — копирует файл $s в файл $d; Rename (string $oldname, string $newname).
2. Функции для работы с именами файлов file_exists (string $name) — проверяет существование файла basename (string $path) — выделяет имя

файла из полного пути dirname (string $path) — возвращает имя каталога из полного пути realpath (string $path) — преобразует относительный путь к файлу в абсолютный и др.

Пример: echo basename("/home/den/lin.txt"); //выведет lin.txt echo dirname("/home/den/lin.txt"); //выведет /home/den.

3. Функции определения типа и параметров файла filetype (string $filename) — возвращает тип файла (file, dir, link, …) в виде строки.
4. Функции определения прав доступа к файлу.
5. Специальные функции: fileatime() — возвращает время последнего доступа к файлу; filectime() — возвращает время создания файла и др.

4. Работа с формами

Формы предназначены для отправки информации CGI-приложению, то есть нашему сценарию. Тег FORM своими атрибутами указывает адрес сценария, которому будет послана форма, способ пересылки и характеристику данных, содержащихся в форме. Рассмотрим простую форму ввода имени:

```
<form action="http://localhost/1.php" method="GET">
Имя: <input type = text name=user_name><br>
<input type=submit value="riepeflaTb"> </form>
```

Что происходит, когда пользователь вводит имя и нажимает на кнопку «Передать»? Браузер передаст сценарию 1.php параметр user_name. В качестве значения параметра будет введенное пользователем имя. Так как используется метод GET, то в строке адреса вы увидите:

http://localhost/1. php?user__name=Denis

Имена элементам формы присваиваются через их атрибут NAME. Каждый элемент формы имеет начальное и конечное значения строкового типа. Начальное значение присваивается по умолчанию, конечное вводится пользователем. Начальные значения элементов не меняются, благодаря чему возможно «сбросить» форму, то есть снова присвоить всем её элементам первоначальные значения.

Элементы управления

Кнопки — задаются с помощью элементов BUTTON и INPUT. Различают:

• кнопки отправки — при нажатии на них форма посылается серверу;

• кнопки сброса — при их нажатии управляющим элементам присваиваются начальные значения;

• прочие кнопки — кнопки, для которых не указано действие, выполняемое по умолчанию при их нажатии.

Зависимые переключатели (переключатели с зависимой фиксацией) — задаются элементом INPUT и представляют собой переключатели «вкл/выкл». Если несколько зависимых переключателей имеют одинаковые имена, то они являются взаимоисключающими. Это значит, что если одна из них ставится в положение «вкл», то все остальные — автоматически в положение «выкл». Именно это и является преимуществом их использования.

Независимые переключатели (переключатели с независимой фиксацией) — задаются элементом INPUT и представляют собой переключатели «вкл/выкл». В отличие от зависимых, независимые переключатели могут при-

нимать и изменять значения независимо от остальных переключателей, даже если последние имеют такое же имя.

Меню — реализуется с помощью элементов SELECT, OPTGROUP и OPTION. Меню предоставляют пользователю список возможных вариантов выбора.

Ввод текста — реализуется элементами INPUT, если вводится одна строка, и элементами TEXTAREA — если несколько строк. В обоих случаях введенный текст становится текущим значением управляющего элемента.

Выбор файлов — позволяет вместе с формой отправлять выбранные файлы, реализуется HTML-элементом INPUT.

Скрытые управляющие элементы — создаются управляющим элементом INPUT.

5. Работа с базами данных

Язык SQL. Реляционная база данных — представляет собой совокупность связанных двумерных таблиц, хранящих данные. Базовым элементом таблицы является поле, обладающее двумя атрибутами: типом данных и хранящимся в нем значением — элементом данных. Набор полей (одного или разных типов) называется *записью*. *Таблица* — это совокупность записей одной структуры. Программное обеспечение, которое управляет базой данных, называется системой управления базой данных (СУБД). В PHP есть функции для работы с множеством СУБД (MS SQL Server, Oracle, Sybase и др.).

MySQL — это своеобразный стандарт сетевых баз данных, пригодный для реализации самых разнообразных веб-проектов. Так уж исторически сложилось, что эти веб-проекты написаны преимущественно на языке PHP.

SQLite — сравнительно молодой продукт, поддержка которого появилась только в пятой версии PHP. Тесты показывают, что SQLite работает несколько быстрее, чем MySQL, но за все в этом мире нужно платить. Ценой быстродействия стали «урезанные» возможности SQLite. Например, в отличие от MySQL, SQLite не является сетевым: если база данных управляется SQLite, то она должна находиться на том же сервере, что и PHP-сценарий, тогда как база MySQL может быть расположена на другом сервере. Для большинства веб-проектов это не слишком серьезное ограничение, потому что и их база данных, и PHP-сценарий расположены на сервере хостинг-провайдера. MySQL в узких кругах веб-программистов настолько популярен, что для него придумали даже собственное название — «мускул». Поэтому, если вы когда-нибудь услышите или увидите выражение «движок на мускуле», это означает, что сайт написан с использованием базы данных MySQL. Как работает сервер SQL? Клиент посылает запрос, в котором он указывает, какие данные он хочет получить от сервера или какую операцию над данными он хочет выполнить; сервер посылает клиенту ответ, в котором указывается, выполнил ли сервер запрос клиента и, если сервер выполнил запрос, собственно результат запроса.

Сервер баз данных MySQL. Сервер баз данных MySQL надежен, устойчив, легок в использовании, является идеальным решением для малых и средних сетевых приложений баз данных. Поскольку сервер MySQL сетевой, у него должны быть клиентская и серверная части. Клиентская часть подключается к

серверной, принимает запрос от пользователя, передает его на сервер и отображает результат — ответ от сервера. Серверная часть получает запрос от клиента, обрабатывает его и отправляет результат обработки. Логически таблица является совокупностью записей, а запись — совокупностью полей разного типа (тип полей может повторяться). Имя базы данных уникально в пределах системы, таблицы — в пределах базы данных, а поля — в пределах таблицы. Структуру базы данных можно просмотреть с помощью утилиты mysqlshow: mysql show -p mysql. Вы увидите список таблиц, которые находятся в базе данных mysql. Для самих же операций с данными служит программа mysql. Она и является клиентом сервера. Программа mysql принимает те же ключи, что и mysqlshow, а среди прочих ее ключей я хочу обратить ваше внимание только на один: «-s». Он подавляет большинство ненужных сообщений, выводимых клиентом. Создать базу данных можно с помощью программы mysqladmin. Естественно, что использовать эту программу может только администратор, например mysqladmin -u admin -p create my_db

Функции php для работы с MySQL. Основной функцией является mysql_connect (), которая подключает наш сценарий к серверу баз данных и выполняет авторизацию пользователя.

mysql_connect([string $hostname][, string $user] [, string $password])

Все параметры функции являются необязательными, поскольку значения по умолчанию можно прописать в конфигурационном файле php.ini. Если вы хотите указать другие имя MySQL-узла, имя пользователя и пароль, вы всегда можете это сделать. Параметр $hostname может быть указан в виде «узел: порт».

Функция возвращает целый идентификатор соединения, служащий для всех дальнейших обращений к этому серверу БД. Следующий вызов функции mysql_connect () с теми же параметрами не откроет нового соединения, а вернет идентификатор существующего.

Для закрытия соединения предназначена функция mysql_close ([int $connection_id]).

Функция mysql_connect () устанавливает обыкновенное соединение с базой данных. Однако PHP позволяет создавать так называемые постоянные соединения — для этого служит функция mysql_pconnect (). Параметры у этой функции такие же, как и у mysql_connect (). PHP работает с постоянными соединениями примерно так: при вызове функции mysql_pconnect () PHP проверяет, было ли раньше установлено соединение. Если да, то возвращается его идентификатор, а если нет, открывается новое соединение и также возвращается идентификатор.

Выбор базы данных. Функция mysql_select_db (string $db [, int $id]) выбирает базу данных, с которой будет работать сценарий. Если открыто только одно соединение, параметр $id можно не указывать. Пример подключения к базе данных:

```
// Пытаемся установить соединение
 if (!mysql_connect($SERVER,$USER,$PASSWD))
 {
  echo "Не могу подключиться к серверу";
  exit; }  // Выбираем базу данных mysql_select_db($DB);
```

Обработка ошибок. Если произойдет ошибка, мы получим сообщение «Не могу подключиться к серверу», и все: сценарий завершит свою работу. Во время отладки сценария для уточнения подробностей ошибки можно воспользоваться двумя функциями:

mysql_errno(int $id);

mysql_error(int $id);

Первая функция возвращает номер ошибки, а вторая — сообщение, описывающее эту ошибку. Вместо того чтобы реагировать на все ошибки одним и тем же сообщением, лучше использовать такой вызов: echo "ERROR ".mysql_errno().": ".mysql_error()."\n"; Теперь вам не нужно гадать, из-за чего произошла ошибка — вы сразу увидите сообщение об ошибке.

Обработка результата SQL-запроса. Все запросы к текущей базе данных отправляются функцией mysql_query (string $query), принимающей один параметр: текст запроса на языке SQL. Текст запроса может содержать пробельные символы и символы новой строки (\n). Пример запроса: $r = mysql_query("select * from hbd");

Контрольные вопросы

1. Для чего предназначен язык PHP?
2. Какие имеются особенности в использовании функций в PHP?
3. В каких случаях определяют тип данных в PHP?
4. Как вывести сообщение на языке PHP?
5. Что такое жесткая ссылка?
6. Что такое MySQL?

Лекция 19. Классификация технической документации согласно ЕСПД

План:

1. Краткое представление стандартов ЕСПД.
2. Стандарты ЕСПД.

1. Краткое представление стандартов ЕСПД

ЕСПД нуждается в полном пересмотре на основе стандарта ИСО/МЭК 12207-95 на процессы жизненного цикла ПС. Тем не менее до пересмотра всего комплекса многие стандарты ЕСПД могут с пользой применяться в практике документирования ПС. Эта позиция основана на следующем:

- стандарты ЕСПД вносят элемент упорядочения в процесс документирования ПС;
- предусмотренный стандартами ЕСПД состав программных документов вовсе не такой «жесткий», как некоторым кажется: стандарты позволяют вносить в комплект документации на ПС дополнительные виды;
- стандарты ЕСПД позволяют вдобавок мобильно изменять структуры и содержание установленных видов ПД исходя из требований заказчика и пользователя.

При этом стиль применения стандартов может соответствовать современному общему стилю адаптации стандартов к специфике проекта: заказчик и руководитель проекта выбирают уместное в проекте подмножество стандартов и

ПД, дополняют выбранные ПД нужными разделами и исключают ненужные, привязывают создание этих документов к той схеме ЖЦ, которая используется в проекте.

Стандарты ЕСПД (как и другие ГОСТы) подразделяют на группы, приведенные в таблице 38.

Таблица 38 — Группы стандартов

Код группы	Наименование группы
0	Общие положения
1	Основополагающие стандарты
2	Правила выполнения документации разработки
3	Правила выполнения документации изготовления
4	Правила выполнения документации сопровождения
5	Правила выполнения эксплуатационной документации
6	Правила обращения программной документации
7	Резервные группы
8	
9	Прочие стандарты

Обозначение стандарта ЕСПД строят по классификационному признаку и должно состоять из:

- числа 19 (присвоенных классу стандартов ЕСПД);
- одной цифры (после точки), обозначающей код классификационной группы стандартов, указанной в таблице;
- двузначного числа (после тире), указывающего год регистрации стандарта.

Перечень документов ЕСПД

1. ГОСТ 19.001-77 ЕСПД. Общие положения.
2. ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов.
3. ГОСТ 19.102-77 ЕСПД. Стадии разработки.
4. ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов.
5. ГОСТ 19.104-78 ЕСПД. Основные надписи.
6. ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам.
7. ГОСТ 19.106-78 ЕСПД. Требования к программным документам, выполненным печатным способом.
8. ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению.
9. ГОСТ 19.202-78 ЕСПД. Спецификация. Требования к содержанию и оформлению.
10. ГОСТ 19.301-79 ЕСПД. Порядок и методика испытаний.
11. ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению.

12. ГОСТ 19.402-78 ЕСПД. Описание программы.

13. ГОСТ 19.404-79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению.

14. ГОСТ 19.501-78 ЕСПД. Формуляр. Требования к содержанию и оформлению.

15. ГОСТ 19.502-78 ЕСПД. Описание применения. Требования к содержанию и оформлению.

16. ГОСТ 19.503-79 ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению.

17. ГОСТ 19.504-79 ЕСПД. Руководство программиста.

18. ГОСТ 19.505-79 ЕСПД. Руководство оператора.

19. ГОСТ 19.506-79 ЕСПД. Описание языка.

20. ГОСТ 19.508-79 ЕСПД. Руководство по техническому обслуживанию. Требования к содержанию и оформлению.

21. ГОСТ 19.604-78 ЕСПД. Правила внесения изменений в программные документы, выполняемые печатным способом.

22. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

23. ГОСТ 19.781-90. Обеспечение систем обработки информации программное.

2. Стандарты ЕСПД

Из всех стандартов ЕСПД остановимся только на тех, которые могут чаще использоваться на практике. Первым укажем стандарт, который можно использовать при формировании заданий на программирование.

ГОСТ (СТ СЭВ) 19.201-78 (1626-79). ЕСПД. Техническое задание. Требование к содержанию и оформлению (переиздан в ноябре 1987 г. с изм.).

Техническое задание (ТЗ) содержит совокупность требований к ПС и может использоваться как критерий проверки и приемки разработанной программы. Поэтому достаточно полно составленное (с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком ТЗ является одним из основополагающих документов проекта ПС.

Техническое задание должно содержать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки;
- в техническое задание допускается включать приложения.

Таблица 39 — Виды программных документов, разрабатываемых на разных стадиях, и их коды

Код вида документа	Вид документа	Стадии разработки			
		Эскизный проект	Технический проект	Рабочий проект	
				компонент	комплекс
-	Спецификация	-	-	!	+
05	Ведомость держателей подлинников	-	-	-	?
12	Текст программы	-	-	+	?
13	Описание программы	-	-	?	?
20	Ведомость эксплуатационных документов	-	-	?	?
30	Формуляр	-	-	?	?
31	Описание применения	-	-	?	?
32	Руководство системного программиста	-	-	?	?
33	Руководство программиста	-	-	?	?
34	Руководство оператора	-	-	?	?
35	Описание языка	-	-	?	?
46	Руководство по техническому обслуживанию	-	-	?	?
51	Программа и методика испытаний	-	-	?	?
81	Пояснительная записка	?	?	-	-
90-99	Прочие документы	?	?	?	?

Условные обозначения:

+ — документ обязательный;

! — документ обязательный для компонентов, имеющих самостоятельное применение;

? — необходимость составления документа определяется на этапе разработки и утверждения технического задания;

- — документ не составляют.

Допускается объединять отдельные виды эксплуатационных документов (за исключением ведомости эксплуатационных документов и формуляра). Необходимость объединения этих документов указывается в техническом задании. Объединенному документу присваивают наименование и обозначение одного из объединяемых документов. В объединенных документах должны быть приведены сведения, которые необходимо включать в каждый объединяемый документ.

ГОСТ 19.102-77. ЕСПД. Стадии разработки

Устанавливает стадии разработки программ и программной документации для вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Таблица 40 — Стадии разработки, этапы и содержание работ

Стадии разработки	Этапы работ	Содержание работ
Техническое задание	Обоснование необходимости разработки программы	Постановка задачи. Сбор исходных материалов. Выбор и обоснование критериев эффективности и качества разрабатываемой программы. Обоснование необходимости проведения научно-исследовательских работ
	Научно-исследовательские работы	Определение структуры входных и выходных данных. Предварительный выбор методов решения задач. Обоснование целесообразности применения ранее разработанных программ. Определение требований к техническим средствам. Обоснование принципиальной возможности решения поставленной задачи
		Определение требований к программе. Разработка технико-экономического обоснования разработки программы. Определение стадий, этапов и сроков разработки программы и документации на нее. Выбор языков программирования
	Разработка и утверждение технического задания	Определение необходимости проведения научно-исследовательских работ на последующих стадиях. Согласование и утверждение технического задания
Эскизный проект	Разработка эскизного проекта	Предварительная разработка структуры входных и выходных данных. Уточнение методов решения задачи. Разработка общего описания алгоритма решения задачи. Разработка технико-экономического обоснования
	Утверждение эскизного проекта	Разработка пояснительной записки. Согласование и утверждение эскизного проекта

Продолжение табл. 40

Стадии разработки	Этапы работ	Содержание работ
Технический проект	Разработка технического проекта	Уточнение структуры входных и выходных данных. Разработка алгоритма решения задачи. Определение формы представления входных и выходных данных. Определение семантики и синтаксиса языка. Разработка структуры программы. Окончательное определение конфигурации технических средств
	Утверждение технического проекта	Разработка плана мероприятий по разработке и внедрению программ. Разработка пояснительной записки. Согласование и утверждение технического проекта
Рабочий проект	Разработка программы	Программирование и отладка программы
	Разработка программной документации	Разработка программных документов в соответствии с требованиями ГОСТ 19.101-77
	Испытания программы	Разработка, согласование и утверждение программы и методики испытаний. Проведение предварительных государственных, межведомственных, приемо-сдаточных и других видов испытаний. Корректировка программы и программной документации по результатам испытаний
Внедрение	Подготовка и передача программы	Подготовка и передача программы и программной документации для сопровождения и/или изготовления. Оформление и утверждение акта о передаче программы на сопровождение и /или изготовление. Передача программы в фонд алгоритмов и программ

Примечания:

1. Допускается исключать вторую стадию разработки, а в технически обоснованных случаях — вторую и третью стадии. Необходимость проведения этих стадий указывается в техническом задании.

2. Допускается объединять, исключать этапы работ и/или их содержание, а также вводить другие этапы работ по согласованию с заказчиком.

ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов

Код страны-разработчика и код организации-разработчика присваивают в установленном порядке.

- Регистрационный номер присваивается в порядке возрастания, начиная с 00001 до 99999, для каждой организации-разработчика.
- Номер издания программы или номер редакции, номер документа данного вида, номер части документа присваиваются в порядке возрастания с 01 до 99. (Если документ состоит из одной части, то дефис и порядковый номер части не указывают.)

- Номер редакции спецификации и ведомости эксплуатационных документов на программу должны совпадать с номером издания этой же программы.

ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам

Настоящий стандарт устанавливает общие требования к оформлению программных документов для вычислительных машин, комплексов и систем, независимо от их назначения и области применения и предусмотренных стандартами Единой системы программной документации (ЕСПД) для любого способа выполнения документов на различных носителях данных.

Программный документ может быть представлен на различных типах носителей данных и состоит из следующих условных частей: титульной; информационной; основной.

Правила оформления документа и его частей на каждом носителе данных устанавливаются стандартами ЕСПД на правила оформления документов на соответствующих носителях данных.

ГОСТ 19.106-78 ЕСПД. Требования к программным документам, выполненным печатным способом

Программные документы оформляют:

- на листах формата А4 (ГОСТ 2.301-68) при изготовлении документа машинописным или рукописным способом;
- допускается оформление на листах формата А3;
- при машинном способе выполнения документа допускаются отклонения размеров листов, соответствующих форматам А4 и А3, определяемые возможностями применяемых технических средств; на листах форматов А4 и А3, предусматриваемых выходными характеристиками устройств вывода данных, при изготовлении документа машинным способом;
- на листах типографических форматов при изготовлении документа типографским способом.

Расположение материалов программного документа осуществляется в следующей последовательности.

Титульная часть:

- лист утверждения (не входит в общее количество листов документа);
- титульный лист (первый лист документа).

Информационная часть:

- аннотация;
- лист содержания.

Основная часть:

- текст документа (с рисунками, таблицами и т. п.);
- перечень терминов и их определений;
- перечень сокращений;
- приложения;
- предметный указатель;
- перечень ссылочных документов.

Часть регистрации изменений:

- лист регистрации изменений.

Перечень терминов и их определений, перечень сокращений, приложения, предметных указатель, перечень ссылочных документов выполняются при необходимости.

Следующий стандарт ориентирован на документирование результирующего продукта разработки.

ГОСТ 19.402-78 ЕСПД. Описание программы

Состав документа «Описание программы» в своей содержательной части может дополняться разделами и пунктами, почерпнутыми из стандартов для других описательных документов и руководств: ГОСТ 19.404-79 ЕСПД. Пояснительная записка, ГОСТ 19.502-78 ЕСПД. Описание применения, ГОСТ 19.503-79 ЕСПД. Руководство системного программиста, ГОСТ 19.504-79 ЕСПД. Руководство программиста, ГОСТ 19.505-79 ЕСПД. Руководство оператора.

Есть также группа стандартов, определяющая требования к фиксации всего набора программ и ПД, которые оформляются для передачи ПС. Они порождают лаконичные документы учетного характера и могут быть полезны для упорядочения всего хозяйства программ и ПД (ведь очень часто требуется просто навести элементарный порядок!). Есть и стандарты, определяющие правила ведения документов в «хозяйстве» ПС.

ГОСТ 19.301-79 ЕСПД. Программа и методика испытаний, которая (в адаптированном виде) может использоваться для разработки документов планирования и проведения испытательных работ по оценке готовности и качества ПС.

ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные графические и правила выполнения. Он устанавливает правила выполнения схем, используемых для отображения различных видов задач обработки данных и средств их решения, и полностью соответствует стандарту ИСО 5807:1985.

ГОСТ 19781-90. Обеспечение систем обработки информации программное. Термины и определения. Разработан взамен ГОСТ 19.781-83 и ГОСТ 19.004-80 и устанавливает термины и определения понятий в области программного обеспечения (ПО) систем обработки данных (СОД), применяемые во всех видах документации и литературы, входящих в сферу работ по стандартизации или использующих результаты этих работ.

ГОСТ 28388-89. Системы обработки информации. Документы на магнитных носителях данных. Порядок выполнения и обращения. Распространяется не только на программные, но и на конструкторские, технологические и другие проектные документы, выполняемые на магнитных носителях.

Стандарты комплекса ГОСТ 34

ГОСТ 34 в основном уделяет внимание содержанию проектных документов, распределение действий между сторонами обычно делается, отталкиваясь от этого содержания.

Наиболее популярными можно считать стандарты ГОСТ 34.601-90 (Стадии создания АС), ГОСТ 34.602-89 (ТЗ на создание АС) и методические указа-

ния РД 50-34.698-90 (Требования к содержанию документов). Стандарты предусматривают стадии и этапы выполнения работ по созданию АС, но не предусматривают сквозных процессов в явном виде.

Практика применения стандартов показала, что в них применяется по существу (но не по строгим определениям) единая система понятий, есть много общих объектов стандартизации, однако требования стандартов не согласованы между собой, имеются различия по составу и содержанию работ, различия по обозначению, составу, содержанию и оформлению документов и пр.

Государственные стандарты РФ (ГОСТ Р)

В РФ действует ряд стандартов в части документирования ПС, разработанных на основе прямого применения международных стандартов ИСО.

ГОСТ Р ИСО/МЭК 9294-93. Информационная технология. Руководство по управлению документированием программного обеспечения. Стандарт полностью соответствует международному стандарту ИСО/МЭК ТО 9294:1990 и устанавливает рекомендации по эффективному управлению документированием ПС для руководителей, отвечающих за их создание. Целью стандарта является оказание помощи в определении стратегии документирования ПС; выборе стандартов по документированию; выборе процедур документирования; определении необходимых ресурсов; составлении планов документирования.

ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению. Стандарт полностью соответствует международному стандарту ИСО/МЭК 9126:1991. В его контексте под характеристикой качества понимается «набор свойств (атрибутов) программной продукции, по которым ее качество описывается и оценивается». Стандарт определяет шесть комплексных характеристик, которые с минимальным дублированием описывают качество ПС (ПО, программной продукции): функциональные возможности; надежность; практичность; эффективность; сопровождаемость; мобильность. Эти характеристики образуют основу для дальнейшего уточнения и описания качества ПС.

ГОСТ Р ИСО 9127-94. Системы обработки информации. Документация пользователя и информация на упаковке для потребительских программных пакетов. Стандарт полностью соответствует международному стандарту ИСО 9127:1989. В контексте настоящего стандарта под потребительским программным пакетом (ПП) понимается «программная продукция, спроектированная и продаваемая для выполнения определенных функций; программа и соответствующая ей документация, упакованные для продажи как единое целое». Под документацией пользователя понимается документация, которая обеспечивает конечного пользователя информацией по установке и эксплуатации ПП. Под информацией на упаковке понимают информацию, воспроизводимую на внешней упаковке ПП. Ее целью является предоставление потенциальным покупателям первичных сведений о ПП.

ГОСТ Р ИСО/МЭК 8631-94. Информационная технология. Программные конструктивы и условные обозначения для их представления. Описывает представление процедурных алгоритмов.

Международный стандарт ISO/IEC 12207: 1995-08-01

Первая редакция ISO12207 подготовлена в 1995 году объединенным техническим комитетом ISO/IEC JTC1 «Информационные технологии, подкомитет SC7, проектирование программного обеспечения».

По определению ISO12207 — базовый стандарт процессов ЖЦ ПО, ориентированный на различные (любые!) виды ПО и типы проектов АС, куда ПО входит как часть. Стандарт определяет стратегию и общий порядок в создании и эксплуатации ПО, он охватывает ЖЦ ПО от концептуализации идей до завершения ЖЦ.

Конрольные вопросы

1. Как применяются стандарты ЕСПД в практике документирования ПС?
2. Как обозначаются ЕСПД?
3. Какие требования к содержанию и оформлению технического задания указаны в ЕСПД?
4. Какие требования к программным документам, выполненным печатным способом, указаны в ЕСПД?

Лекция 20. Методы разработки основных видов технической документации

План:

1. Понятия технической документации.
2. Классификация технической документации.
3. Техническое описание.
4. Научно-исследовательская документация.

1. Понятия технической документации

Техническое документирование — это и есть способ фиксации технической мысли на материальном носителе.

Техническая документация появляется в процессе документирования различных видов научно-технической деятельности, к числу которых относится проектирование, конструирование, разработка технологических процессов, научно-исследовательская деятельность, организация промышленного производства, а также геолого-разведочные, геодезические, картографические работы. Таким образом, *технические документы* — это обобщающее название графических и текстовых документов, отражающих техническую мысль.

В дальнейшем расширение сферы технического документирования происходило за счёт появления новых либо совершенствования старых его направлений: технологическое документирование, документирование научных исследований, стандартизации и др.

В техническом документировании используются прежде всего изобразительные средства (чертежи, схемы, диаграммы, рисунки, графики), поскольку с помощью письма зачастую сложно или невозможно передать точную информацию о предмете, объекте, явлении. Формализованный графический язык является специфической знаковой системой передачи информации. Графический

образ представляет собой «целесообразно построенную совокупность изобразительных элементов, имеющих условное значение». Графический образ включает пять элементов: точки, линии, фигуры, цвета, текстуры (штриховки). Вместе с тем в техническом документировании достаточно широко используется также словесная форма, тексты.

2. Классификация технической документации

Обычно всё многообразие технической документации разделяют на 4 группы:

1) документы, фиксирующие информацию о средствах производства (проектно-конструкторские документы);

2) результаты записи процесса труда (технологические документы);

3) зафиксированная информация о природе как объекте человеческой деятельности (документы о геологоразведке, геодезии, картографии, метеорологии и др.);

4) документы, связанные со сферой обслуживания и потребления, с использованием технических средств.

Во всех группах технической документации значительный удельный вес занимают *чертежи*. Существует множество разновидностей чертежей, обусловленных:

- характером изображаемого предмета (чертежи предметов промышленного производства, объектов капитального строительства);
- степенью подробности изображения (генеральные планы, габаритные чертежи, собственно чертежи);
- полнотой изображения (чертежи общего вида, чертежи сборочных единиц, чертежи деталей) и др.

К числу технических документов относятся *рисунки*, позволяющие рельефно представить предмет, особенно в тех случаях, когда изделие оценивается не только с технической, но и с художественной стороны (рисунки обоев, ткани, упаковочных коробок, а также архитектуры домов и т. д.).

3. Техническое описание

Технические описания содержат информацию об особенностях изделия, его основных характеристиках, назначении, устройстве, работе отдельных частей. К числу текстовых технических документов относится также *пояснительная записка*. В ней приводятся сведения об устройстве и принципе работы изделия, даётся обоснование технических и технико-экономических решений. К техническим описаниям и пояснительным запискам тесно примыкают *паспорта изделий, расчёты, инструкции, схемы, графики* и т. д.

Большую группу графических и текстовых технических документов составляют *технологические документы*, являющиеся результатом технического документирования процесса труда. Они содержат информацию не о самих предметах, изделиях и т. п., а о способах их изготовления, сборки, строительства, эксплуатации, ремонта, организации производственного процесса.

Основным технологическим документом является *технологическая карта*, содержащая подробную информацию обо всех производственных операциях, необходимых для изготовления изделия.

К технологическим документам относятся также *прикладное программное обеспечение* автоматизированных технологических процессов, *графики работ цехов и бригад; технические условия* на обработку деталей, сборку конструкций, строительство объектов и выполнение других производственных задач.

4. Научно-исследовательская документация

В процессе проведения научных исследований и выполнения научно-технических разработок создаётся обширная научно-исследовательская документация. Её основные виды:

- отчёты по научно-исследовательским, опытно-конструкторским, опытно-технологическим и экспериментально-проектным работам;
- отчёты, доклады о научных экспедициях, научных и технических командировках специалистов;
- программы научно-исследовательских работ;
- технико-экономические обоснования, обзоры, доклады, записки и др.;
- заключения, отзывы, рецензии, аннотации;
- монографии, диссертации, другие научные публикации и отзывы на них;
- первичная документация, образующаяся в ходе проведения научно-исследовательских работ (результаты анализов, журналы записей экспериментов и т. п.).

Особую группу технических документов, имеющих правовое значение, составляют стандарты.

В отдельную группу выделяются документы, способствующие использованию технических средств и охватывающие, таким образом, сферу потребления. К ним относятся: паспорта технических изделий, каталоги промышленных фирм, рекламные чертежи и рисунки и т. д.

До недавних пор техническое документирование осуществлялось преимущественно в традиционной (аналоговой) форме — в виде уже рассматривавшихся выше чертежей, карт и т. п. В настоящее время всё большее распространение получают цифровые средства и методы представления научно-технической информации.

Проектная документация — это вид технической документации, определяющий функциональные, архитектурные и технические решения проектируемого программного обеспечения (автоматизированной системы).

Как правило, документы данного вида составляют «ядро» комплекта технической документации: именно проектную документацию мы чаще всего подразумеваем, когда говорим о технической документации в целом.

Какие основные документы входят в состав проектной документации?

- техническое задание;
- пояснительная записка;
- программа и методика испытаний;

- описание программы / программного обеспечения

и т.д.

Стоит отметить, что все эти документы — это лишь начало огромного перечня проектной документации в соответствии с требованиями ГОСТ. В документировании, особенно если речь идет о ЕСКД, встречается огромное множество иных документов, многие из которых уже морально устарели, но тем не менее активно используются в силу государственных традиций и советского наследия.

В соответствии с какими отечественными стандартами разрабатывается проектная документация?

ЕСКД, Единая система конструкторской документации (ГОСТ 2);

ЕСПД, Единая система программной документации (ГОСТ 19);

КСАС, Комплекс стандартов на автоматизированные системы (ГОСТ 34).

Каждая из вышеперечисленных серий ГОСТ предъявляет свои требования к составу, порядку разработки и ведения комплектов документации, а также к содержанию и оформлению отдельных документов.

В соответствии с какими зарубежными стандартами разрабатывается проектная документация?

Данный вид документации в плоскости **зарубежных стандартов** соответствует следующим типам:

Requirements, описание требований и характеристик программного обеспечения (аналог Техническое задание);

Architecture/Design, описание архитектуры и взаимосвязей (аналог Пояснительная записка);

Technical, описание кода, алгоритмов и интерфейсов (аналог Описание программы).

В отличие от ГОСТов, за рубежом изначально несколько **иной подход к стандартам документирования**.

Контрольные вопросы

1. Как классифицируют техническую документацию?
2. Что такое технические описание?
3. Что такое проектная документация?
4. В соответствии с какими отечественными стандартами разрабатывается проектная документация?
5. В соответствии с какими зарубежными стандартами разрабатывается проектная документация?

Лекция 21. Инструментальные средства для разработки технической документации

План:

1. Средства разработки документации.
2. Автоматизированное создание документов ГОСТ.

1. Средства разработки документации

Разработка электронной документации — это обширная область практики, охватывающая принципы разработки, сопровождения, поддержания цело-

стности больших пакетов документов (например, технических заданий к большим программным системам, нормативной базы банка или крупной корпорации, свода законов), специализированные языки и средства, особенности перевода текстов на разные языки и т. д.

Простая документация разрабатывается в редакторах общего назначения, таких как Microsoft Word. Для более сложной используются специализированные методы и средства, такие как FrameMaker (издательская система компании Adobe), DocBook (open-source стандарт в области разработки Association for Computer Machinery's Special Interest Group on the Design of Communication документации для Unix/Linux-приложений), DITA (метод разработки сложной модульной документации компании IBM) и др.

Распространенным классом технических документов является пользовательская документация ПО. Ее особенности вытекают из свойств ПО — структурная сложность, изменчивость, многоверсионность, многоязычность, многоформатность (руководство пользователя, сайт поддержки, справочная система). В больших пакетах пользовательской документации много фрагментов текста, которые используются почти без изменений в различных контекстах и документах.

Повторное использование различных активов при создании ПО является бурно развивающейся областью. Усилия практиков и теоретиков сегодня сконцентрированы вокруг семейств программных продуктов (product lines, далее — СПП). Эта парадигма предлагает для набора продуктов компании, обладающих общими свойствами, единый процесс разработки на основе повторно используемых активов в рамках хорошо определенных процедур, а также общую стратегию продвижения на рынке.

Такие технологии, как FrameMaker, DocBook, DITA, поддерживают повторное использование, но не обеспечивают адаптивности, то есть повторно используемые фрагменты должны быть идентичны во всех контекстах использования. Отсутствие поддержки адаптивности сильно ограничивает возможности повторного использования.

Метод Бассета позволяет повторно использовать произвольную электронную информацию и обеспечивает адаптивность. Однако он не применялся для управления повторным использованием документации. Наконец, существует очень перспективный подход к управлению повторным использованием с помощью визуального моделирования — это диаграммы возможностей (Feature Diagrams). Однако данный метод также не применялся для управления повторным использованием документации.

DocBook — технология разработки документации, предложенная в 1991 г. компаниями HaL Computer Systems и O'Reilly&Associates. Ее основная идея — разделение содержания документа и его форматирования, что позволяет создать единое исходное представление документа (single source) и на этой основе автоматически генерировать документацию в различных форматах, например печатную документацию (PDF), справочные системы (HTMLHelp/WinHelp), электронную документацию (HTML).

Технология включает в себя язык, который позволяет выполнить полиграфическую разметку и форматирование текстов документов. Современная версия является XML-языком, описание схемы является открытым, стандартизовано консорциумом OASIS и доступно в нескольких форматах — DTD, XML Schema, Relax NC3.

Целый ряд инструментальных средств поддерживает разработку документов DocBook. В первую очередь это пакет XSLT-трансформаций, позволяющий по документам DocBook получить документы в виде HTML, HTMLHelp, FO, PDF, RTF и в некоторых других форматах. Также многие коммерческие XML-редакторы (например, XML Spy и Oxygen) предлагают поддержку редактирования DocBook-документов.

В настоящее время DocBook широко используется для разработки документации операционных систем семейства UNIX (FreeBSD, Linux), а также в мире разработки открытого ПО.

Постоянное увеличение объемов и сложности программных систем (ПС), а также рост требований к их качеству привели к активному развитию технологий, автоматизирующих процессы жизненного цикла программных средств. Потребность в таких технологиях основывается на постоянном усложнении проектов проектирования, разработки и сопровождения программных средств, за счет увеличения числа участников проекта, ужесточении требований к качеству и срокам выхода продукта на рынок.

2. Автоматизированное создание документов ГОСТ

Одной из наиболее развитых и популярных методологий, описывающих процессы ЖЦ ПС, является Rational Unified Process (RUP), разработанный компанией Rational Software и соответствующий ГОСТ Р ИСО/МЭК 12207-99. При этом необходимо отметить, что RUP ориентирован прежде всего на разработку ПС и без предварительной адаптации не может использоваться для задач процесса сопровождения. Требуется автоматизировать процесс создания документации, соответствующей требованиям ГОСТ, на основе имеющихся материалов — артефактов RUP — для того, чтобы минимизировать трудозатраты. Кроме того, такой подход позволяет не заботиться о синхронизации документации и артефактов RUP, являющихся рабочими материалами проекта.

Для решения задачи было проведено сопоставление артефактов RUP и документации, разрабатываемой по требованиям ГОСТ 19 и 34. При сопоставлении артефактов учитывались стадии ЖЦ ПС, на которых они разрабатываются. После установления соответствия между документами ГОСТ 19 и 34 и артефактами RUP на уровне разделов и компонентов является разработка специальных шаблонов для настройки автоматизированной отчетности с помощью инструментального средства Rational SoDA.

Rational SoDA имеет возможности выборки данных из различных артефактов, созданных на основе инструментальных средств Rational, — моделей данных и классов (Rational Rose), версионного хранилища (Rational ClearCase), репозитория требований (Rational RequisitePro), репозитория запросов на изменения (Rational ClearQuest), репозитория тестирования (Rational Test Manager).

Документация генерируется по заранее определенным шаблонам на основе артефактов, создаваемых в процессе выполнения работ ЖЦ ПС. Генератор отчетов Rational SoDA имеет ряд встроенных отчетов, ориентированных на работу по методологии RUP. Для генерации документов другого типа требуется разработать новые шаблоны, что и было сделано при формировании документов серии ГОСТ 19 и 34.

Преимущества представленной технологии

Представленный обобщенный опыт автоматизированного создания документов ГОСТ 19 и 34 на основе адаптации артефактов RUP к потребностям организации доведен до уровня типовой технологии, которая может быть использована при реализации аналогичных проектов и включает следующие составляющие:

- адаптированные шаблоны артефактов RUP с разделением внутренней и внешней информации на уровне компонент артефактов;
- перечень компонент артефактов RUP, на основании которых формируются разделы документов серии ГОСТ 19 и 34;
- шаблоны Rational SoDA, разработанные для автоматизированного формирования документов серии ГОСТ 19 и 34 на основании компонент артефактов RUP;
- использование инструментальных средств Rational при выполнении работ ЖЦ ПС.

Преимущества использования такой технологии основываются на следующих положениях.

1. Технология основана на широко распространенной методологии и стандартах и может использоваться в большом числе проектов при незначительном уровне доработок.
2. Применение такой технологии сокращает до минимума количество ручных операций при создании документации.
3. Использование технологии позволит косвенным образом контролировать ход проекта по уровню готовности документации, которая может автоматически создаваться на любом этапе проекта.
4. Представленная технология может быть достаточно легко применена к документации и отчетности любого типа, отличного от рассмотренных документов серии ГОСТ 19 и 34, за счет переработки шаблонов отчетов SoDA и реструктуризации артефактов RUP в соответствии с требованиями по структуре и содержанию документации.

Контрольные вопросы

1. Какие средства служат для разработки технической документации?
2. Что такое Rational Unified Process (RUP)?

Лекция 22. Автоматизация документооборота в Word

План:

1. Прикладное программирование для MS Office.
2. Запись макросов макрорекордером.

3. Объектная модель Word.
4. Доступ к документам Word с помощью VBA.
5. Управление параметрами и окнами Word.
6. Использование объектов Selection и Range.
7. Работа с текстом.

1. Прикладное программирование для MS Office

Потребность в автоматизации возникает тогда, когда данных много. А если данных много, то они, скорее всего, будут храниться в базе данных — просто потому, что более удобного способа не придумано. Приложения Office и созданные на основе их программы могут быть очень полезными и сами по себе, но их полезность увеличивается многократно при сопряжении их с базами данных. Чаще всего в реальных приложениях Word используется для генерации отчетов на основе информации из баз данных, Excel — для анализа данных из баз данных, а Access — это сама по себе система управления базами данных (которая очень часто используется для построения клиентского интерфейса для внесения информации в клиент-серверные базы данных, такие как SQL Server и Oracle).

Microsoft Word 2010 — это специализированная программа для работы с текстовыми документами. Когда другому офисному приложению требуется работать с текстом, оно практически всегда обращается к функциональным возможностям объектов, предоставляемых приложением Microsoft Word. Документ Word является идеальным местом для объединения различных элементов из других офисных приложений, например отформатированного отчета и диаграммы Excel, слайда из PowerPoint или некоторых данных из Access.

2. Запись макросов макрорекордером

Макрорекордер — это небольшая программа, встроенная в Excel, которая переводит любое действие пользователя на язык программирования VBA и записывает получившуюся команду в программный модуль. Если мы включим макрорекордер на запись, а затем начнем создавать свой еженедельный отчет, то макрорекордер начнет записывать команды вслед за каждым нашим действием, и в итоге мы получим макрос, создающий отчет, как если бы он был написан программистом. Такой способ создания макросов не требует знаний пользователя о программировании и VBA и позволяет пользоваться макросами как неким аналогом видеозаписи: включил запись, выполнил операции, перемотал пленку и запустил выполнение тех же действий еще раз. Естественно, у такого способа есть свои плюсы и минусы:

- макрорекордер записывает только те действия, которые выполняются в пределах окна Microsoft Excel. Как только вы закрываете Excel или переключаетесь в другую программу — запись останавливается;
- макрорекордер может записать только те действия, для которых есть команды меню или кнопки в Excel. Программист же может написать макрос, который делает то, что Excel никогда не умел (сортировку по цвету, например, или что-то подобное);

- если во время записи макроса макрорекордером вы ошиблись — ошибка будет записана. Однако смело можете нажимать на кнопку отмены последнего действия (Undo) — во время записи макроса макрорекордером она не просто возвращает вас в предыдущее состояние, но и стирает последнюю записанную команду на VBA.

Чтобы включить запись, необходимо выбрать команду ***Начать запись*** и настроить параметры записываемого макроса в окне **Запись макроса**:

- **Имя макроса** — подойдет любое имя на русском или английском языке. Имя должно начинаться с буквы и не содержать пробелов и знаков препинания (например, Proba);
- **Сочетание клавиш** — будет потом использоваться для быстрого запуска макроса. Если забудете сочетание или вообще его не введете, то макрос можно будет запустить через команду ***Выполнить*;**
- **Сохранить в...** — здесь задается место, куда будет сохранен текст макроса, т. е. набор команд на VBA, из которых и состоит макрос:

1) *эта книга* — макрос сохраняется в модуль текущей книги и, как следствие, будет выполнятся, только пока эта книга открыта в Excel;

2) *Новая книга* — макрос сохраняется в шаблон, на основе которого создается любая новая пустая книга в Excel, т. е. макрос будет содержаться во всех новых книгах, создаваемых на данном компьютере начиная с текущего момента;

3) *Личная книга макросов* — это специальная книга Excel с именем *Personal.xls*, которая используется как хранилище макросов. Все макросы из *Personal.xls* загружаются в память при старте Excel и могут быть запущены в любой момент и в любой книге;

- После включения записи и выполнения действий, которые необходимо записать, запись можно остановить командой ***Остановить запись.***

Запуск и редактирование макросов

Управление всеми доступными макросами производится в окне, которое можно открыть через меню ***Сервис — Макрос — Макросы***:

- любой выделенный в списке макрос можно запустить кнопкой **Выполнить;**
- кнопка **Параметры** позволяет посмотреть и отредактировать сочетание клавиш для быстрого запуска макроса;
- кнопка **Изменить** открывает редактор Visual Basic и позволяет просмотреть и отредактировать текст макроса на VBA.

Создание кнопки для запуска макросов

Чтобы не запоминать сочетание клавиш для запуска макроса, лучше создать кнопку и назначить ей нужный макрос.

Откройте панель инструментов **Формы** и выберите объект **Кнопка.** Затем нарисуйте кнопку на листе, удерживая левую кнопку мыши. Автоматически появится окно, где нужно выбрать макрос, который должен запускаться при щелчке по нарисованной кнопке.

3. Объектная модель Word

Ключевым в объектной модели Word является объект **Application** (равно как и в объектных моделях остальных приложений Microsoft Office). Этот объект содержит все остальные объекты Word. Поскольку объект **Application** занимает центральное место в объектной модели приложения, при программировании на языке VBA в Word не требуется явно указывать его имя при работе со многими другими важными объектами. Не следует забывать о роли, которую играет этот объект, поскольку он необходим при работе со свойствами и методами самого приложения, а также при обращении к некоторым другим объектам.

Непосредственно под объектом **Application** размещается коллекция **Documents,** содержащая объекты **Document** для каждого открытого документа. Объект **Document** является центром всего, что происходит в приложении Word, и большая часть настоящей главы будет посвящена именно ему. Остальные компоненты Word, с которыми придется работать, подчинены объекту **Document**.

4. Доступ к документам Word с помощью VBA

Если созданная VBA-процедура функционирует непосредственно в документе, то объект этого документа должен быть явно указан в коде. Иногда требуемые действия можно реализовать и неявным образом, используя объект **Selection,** о котором мы будем говорить ниже в этой главе, однако в большинстве случаев потребуется явно идентифицировать целевой объект.

Обычно типичная VBA-процедура в Word выполняет все свои действия в том документе, который в данный момент открыт для редактирования. В этом случае для указания *активного документа* можно использовать объект **ActiveDocument**. Например, чтобы закрыть активный документ, не требуется получать ссылку на объект редактируемого в данный момент документа, достаточно просто обратиться к объекту **ActiveDocument.**

```
ActiveDocument.Close        ' Закрытие активного документа
```

При работе с определенным документом, который в данный момент времени не активен, следует обращаться к нему как к члену коллекции **Documents,** состоящей из всех документов, открытых в настоящее время в Word. Как и в случае с любой другой коллекцией объектов в VBA, можно обратиться к отдельному документу в коллекции, используя значение его свойства **Name,** которое в данном случае совпадает с именем файла (только с именем файла, а не с полным путем к нему!).

```
Documents("Отчет за ноябрь.doc")
```

Поскольку имя файла вновь создаваемого документа неизвестно заранее и пользователь может определить или изменить его в любой момент работы программы, в этом случае удобнее создать переменную, которая будет содержать имя файла. После этого можно использовать эту переменную для указания объекта данного документа, например **Documents(strDocИмя).**

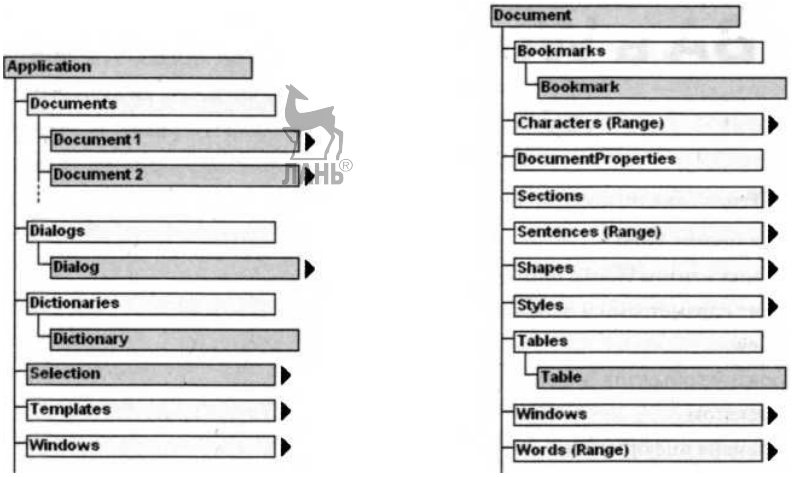


Рисунок 27 — Фрагмент объектной модели MS Word (слева). Темным выделены только объекты, а светлым — объекты и коллекции. Фрагмент структурной схемы объекта Document приложения Word (справа)

Кроме того, к документу можно обращаться по его индексному номеру: **Documents (5)** это, например, обращение к пятому документу в коллекции **Documents.** Следует отметить, что применение данного метода достаточно ограничено, поскольку редко, когда заранее известен индексный номер того документа, с которым требуется работать. Можно поступить иначе: с помощью индексных номеров узнать имена открытых документов, а затем выбрать требуемый. Например, следующий оператор помещает в переменную **strDocИмя** имя файла третьего открытого документа.

strDocИмя = Documents(3).Name

Создание, открытие и закрытие документов

Для создания нового документа используется метод **Add** коллекции объектов **Documents.** Его синтаксис следующий.

Dim MyDocl as Word.Document

Set MyDocl = Documents.Add*[Шаблон, Новый_Шаблон]*

Здесь ***Шаблон*** — необязательный аргумент, определяющий имя шаблона, на основе которого будет создан новый документ. Необязательный аргумент ***Новый_Шаблон*** указывает, будет ли создаваемый документ сам являться шаблоном (значение **True),** или он будет обычным документом (значение **False).** Если оба аргумента опущены, создается обычный документ на основании шаблона Normal. Можно одним оператором создать новый документ и сразу же присвоить ему некоторое имя.

Dim MyDoc As Word.Document

Documents.Add.SaveAs("Отчет за октябрь.doc") ’Создание документа

Set MyDoc = Documents("Отчет за октябрь.doc") ’Получение ссылки

Если документу во время его создания не присваивается имя, это имя следует присвоить при первом его сохранении.

Для открытия уже существующего документа используется метод **Open** коллекции объектов **Documents**.

Dim MyDoc As Word.Document

Set MyDoc = Documents.Open*(Имя_файла)*

Здесь аргумент ***Имя_файла*** задает полный путь и имя файла открываемого документа. Если заданный файл не существует или не является допустимым документом Word, возникает ошибка.

Для активизации уже открытого документа следует использовать метод **Activate** коллекции объектов **Documents.** Предположим, что в VBA-программе требуется активизировать определенный документ, который в момент ее запуска или открыт, или закрыт. Используйте код, подобный показанному ниже. Для активизации открытого документа или открытия его, если этот документ еще не открыт, можно использовать процедуру, приведенную в листинге 1.

Листинг 1. Активизация документа Word

```
Public Sub DocActivateOrOpen()
Dim Имя_Файла, Путь As String Dim Документ As Document Dim Открыт
As Boolean
Имя_Файла = "Отчет за октябрь.docx"   'Имя требуемого файла
Путь = "C:\Отчеты\"                   'Путь к папке с отчетами
'Проверка, открыт ли уже требуемый файл For
Each Документ In Documents
If Документ.Name = Имя_Файла Then Открыт = True
                                      ’Да, требуемый файл открыт
End If
Next Документ
        'Активизация документа с именем "Отчет за      октябрь"
If Открыт =True Then                  'Файл уже   открыт
Documents(Имя_Файла).Activate Else    'Открытие   файла
Documents.Open FileName:=Путь & Имя_Файла
End If End Sub
```

Для закрытия документа используется метод **Close** объекта **Document.** Синтаксис его следующий (здесь ***Документ*** — ссылка на объект закрываемого документа).

Документ. Close (***Сохранить, Формат***, ***Переслать)***

Здесь аргумент ***Сохранить*** указывает, должен ли документ быть сохранен перед закрытием. Возможными значениями этого аргумента являются константы: **wdDoNotSaveChanges** (не сохранять документ), **wdPromptToSaveChanges** (запросить сохранение изменений — принимается по умолчанию) и **wdSave-Changes** (сохранить изменения).

Аргумент ***Формат*** определяет тип формата, в котором будет сохранен документ. Возможными значениями этого аргумента являются константы: **wdOriginalDocumentFormat** (исходный формат документа), **wdPromptUser** (запросить пользователя) и **wdWordDocument** (формат документа Word — принимается по умолчанию).

Аргумент ***Переслать*** допускает значения **True** либо **False** (принимается по умолчанию) и определяет, должен ли документ отсылаться следующему получателю в списке рассылки (routing). Значение **True** не будет иметь смысла, если документ не содержит присоединенного к нему списка рассылки.

Все открытые документы можно одновременно закрыть, используя метод Close коллекции Documents. Он имеет те же аргументы, которые были описаны выше для метода Close объекта Document.

5. Управление параметрами и окнами Word

Иногда в процессе работы с компонентами Word недостаточно сосредоточиться исключительно на объекте **Document** и его подобъектах. В некоторых случаях потребуется иметь дело непосредственно с объектом **Application** и другими объектами, которые в иерархии объектов Word не подчинены объекту **Document.** К таким объектам, в частности, относятся глобальные параметры настройки приложения, некоторые полезные диалоговые окна, а также средства работы с окнами и находящимися в них инструментами.

При работе с приложением Word его объект **Application** будет доступен автоматически — достаточно просто указать его имя: **Application.** Если доступ к объектам Word осуществляется из другого приложения Office, потребуется сначала создать экземпляр приложения Word, а затем использовать в обращениях ссылку на него.

```
Dim MyWord As Word.Application
Set MyWord=New Word.Application
```

Переопределение параметров приложения Word

Приложение Word имеет множество параметров, контролирующих различные аспекты работы программы. При непосредственной работе с приложением доступ к этим параметрам осуществляется с помощью диалогового окна Параметры Word. Оно имеет несколько вкладок, на которых устанавливаются различные параметры, определяющие те или иные аспекты работы программы. Многие из параметров, доступные в этом диалоговом окне, являются глобальными, поскольку оказывают влияние на работу приложения в целом, тогда как другие параметры применяются только к активному документу. Глобальные параметры являются свойствами объекта **Options,** в то время как параметры, влияющие на отдельный документ, будут свойствами объекта **Document.**

Для получения доступа к параметрам уровня приложения используется объект **Options,** ссылка на который хранится в одноименном свойстве объекта **Application.** Например, для включения режима печати скрытого текста можно использовать оператор Options.PrintHiddenText = True.

В общем случае рекомендуется сохранять исходные значения всех параметров приложения, значения которых изменяются в программе, и восстанавливать их перед завершением работы приложения.

Диалоговые окна приложения Word

В VBA-программах можно использовать для своих целей множество диалоговых окон, имеющихся в приложении Word. Доступ к этим окнам осуществляется через коллекцию **Dialogs,** которая является свойством объекта **Applica-**

tion. Для использования некоторого диалогового окна прежде всего нужно создать ссылку на это диалоговое окно.

Dim Диалог As Dialog

Set Диалог = Dialogs(*Тип_окна)*

Здесь аргумент ***Тип_окна*** определяет конкретное диалоговое окно, доступ к которому требуется получить. В библиотеке типов Word содержится список заранее заданных констант в форме **wdDialogXXXX,** где ***XXXX*** определяет конкретное диалоговое окно. В большинстве случаев (но не во всех) имя ***XXXX*** связано с именем соответствующего диалогового окна в английской версии приложения. Например, **wdDialogFileOpen** определяет диалоговое окно открытия файла.

После того как ссылка на требуемое диалоговое окно будет получена, для его отображения можно использовать методы **Show** или **Display** этого объекта. Синтаксис этих двух методов практически одинаков.

Окно.Show(***Таймаут)***

Окно.Display(***Таймаут)***

Здесь аргумент ***Таймаут*** определяет время, по истечении которого данное диалоговое окно следует закрыть, если пользователь не работал с ним. Важно помнить, что единицами измерения времени здесь являются *миллисекунды.* Если этот аргумент опущен, диалоговое окно отображается до тех пор, пока пользователь сам его не закроет.

Когда диалоговое окно имеет вкладки, можно потребовать при его открытии сразу выбрать одну из них. Для ее указания используется свойство **DefaultTab** объекта **Dialog.** Требуемое значение этому свойству задается с помощью дополнительной константы, которая имеет вид **wdDialogXXXXTabyyyy,** где ***XXXX*** определяет диалоговое окно, а *yyyy* — его отдельную вкладку. Так, константа wdDialogFormatFontTabCharacterSpacing определяет вкладку **Интервал** диалогового окна **Шрифт.** При этом общая последовательность операторов будет выглядеть, например, так:

```
Dim Диалог As Dialog
Set Диалог = Dialogs(wdDialogFormatFont)'        Выбор окна
Диалог.DefaultTab = wdDialogFormatFontTabCharacterSpacing
Диалог.Show
```

Следует отметить, что при использовании метода **Show** изменения и любые действия, выполненные в данном диалоговом окне, вступают в силу автоматически, обычным способом. В противоположность этому, при использовании метода **Display** выполненные пользователем изменения и действия не вступают в силу автоматически. Ваша программа сама должна получить от объекта диалогового окна (после его закрытия) информацию о внесенных пользователем изменениях, а затем поступить с ними требуемым образом.

Оба метода возвращают значение типа **Long,** определяющее, какая кнопка использовалась для закрытия окна. Возможные значения приведены в таблице 41.

Таблица 41 — Значения, которые могут возвращаться методами Show и Display

Значение	Описание
-2	Кнопка Close
-1	Кнопка OK или ее эквивалент (например, кнопка Open в диалоговом окне открытия файла)
0	Кнопка Cancel
>0	Другая командная кнопка (1 — первая кнопка, 2 — вторая и т. д.)

Следующий пример показывает, как диалоговое окно Open можно открыть из программного кода:

Dim Открыть_файл as Dialog
Set Открыть_файл = Dialogs(wdDialogFileOpen)
Открыть_файл.Show

В этом случае, если в раскрывшемся диалоговом окне пользователь выберет некоторый файл и щелкнет на кнопке Open, указанный файл будет открыт, причем новый объект **Document** для него создается автоматически, без каких-либо дополнительных действий со стороны вашей программы. При использовании метода **Display** пользователю также будет предоставлена возможность выбрать в раскрывшемся диалоговом окне требуемый файл, однако в программный код потребуется включить процедуру, которая будет его открывать. Обычно для этого используется метод **Execute** объекта **Dialog.**

Иначе говоря, можно считать, что метод **Show** равноценен методу **Display** с последующим выполнением метода **Execute.** Использование метода **Display** для вывода встроенных диалоговых окон будет полезным в тех случаях, когда диалоговое окно выводится для получения необходимой информации от пользователя, однако прежде чем переходить к дальнейшей обработке, ее желательно проверить. В приложении Word имеется около 295 встроенных диалоговых окон.

Обращение к окнам документов из программного кода

Если в VBA-программе работа с Word осуществляется как со скрытым приложением, то обращаться к его окнам и панелям не потребуется. Однако если VBA-приложение предполагает вывод на экран окна Word, с тем чтобы пользователь мог выполнить в нем те или иные необходимые действия, обращение к окнам и панелям Word из программного кода, скорее всего, будет неизбежным. Каждый открытый документ Word содержит, как минимум, одно окно, причем пользователь может открыть столько окон, сколько ему необходимо для работы с данным документом. Каждое из подобных окон — самостоятельный объект с соответствующим набором свойств. В объектной модели Word объект **Application** содержит коллекцию **Windows**, а членами этой коллекции являются объекты **Window** — по одному для каждого открытого окна. Кроме того, каждый объект **Document** содержит свою собственную коллекцию объектов **windows** (см. рис. 28).

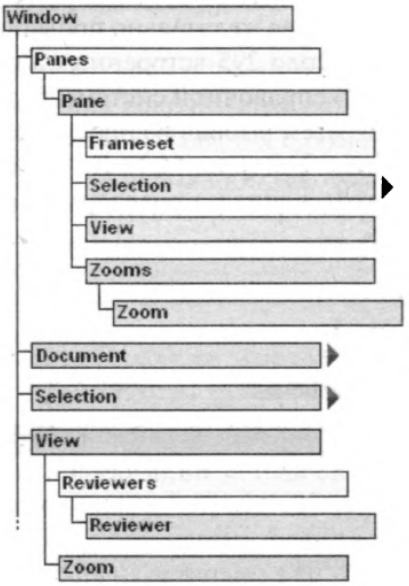


Рисунок 28 — Фрагмент структурной схемы объекта Window приложения Word

Application.Windows

Из программного кода VBA-приложения проще всего обращаться к тому окну, в котором редактируемый документ был открыт во время запуска процедуры. Для ссылок на это окно используется объект **ActiveWindow.** Для определения окна непосредственно в программном коде следует ссылаться на него как на члена одной из коллекций **Windows.**

При работе с глобальной коллекцией объектов **Windows** не нужно дополнительно указывать объект **Application.** Однако при обращении к коллекции объектов **Windows** некоторого документа указывать имя объекта этого документа нужно обязательно.

Окно в коллекции можно идентифицировать по его имени или индексному номеру. Имя окна совпадает с именем документа, который в нем отображается, за исключением тех случаев, когда для одного документа открыто сразу несколько окон.

В этом случае после имени документа следует ставить точку с запятой, а затем номер окна. Типичные примеры записи ссылок на объекты **Window** приведены в таблице 42.

Объект Window имеет множество свойств, как определяющих внешний вид соответствующего окна, так и предназначенных для работы с его содержимым. Большая часть этих свойств может принимать только одно из двух допустимых значений — True или False.

Подобные операторы можно использовать для включения или выключения различных свойств, таких как **DisplayScreenTips** (отображение подсказок) или **DisplayRulers** (отображение линеек). Полезно будет напомнить, что с помощью ключевого слова **Not** можно изменить текущее значение логической переменной или свойства на обратное.

Таблица 42 — Типичные ссылки на объекты Window

Ссылка	Комментарии
Windows(«Отчет за ноябрь»)	Ссылка на окно по его имени. Действительно, если для документа *Отчет за ноябрь* в данный момент открыто только одно окно
Windows(«Реестр; 2»)	Ссылка на окно по его имени. Указывает на второе окно документа Реестр
Document(«Реализация.doc»).Windows(4)	Указывает на четвертое окно в коллекции Window документа *Реализация*

Например, изменить на обратное текущее значение свойства, управляющего отображением в окне линеек, можно следующим образом.

ActiveWindow.DisplayRules = Not ActiveWindow.DisplayRules

С помощью свойств Left, Top, Height и Width можно изменять размер и расположение окна, которое не развернуто на весь экран.

Объект **Window** имеет несколько методов, которые позволяют управлять горизонтальной и вертикальной прокруткой документа для отображения конкретных его частей. Метод **SmallScroll** используется для медленной, а **LargeScroll** — для быстрой прокрутки документа.

Window.SmallScroll (Down, Up, ToRight, ToLeft)

Window.LargeScroll (Down, Up, ToRight, ToLeft)

Аргументы Down, Up, ToRight, ToLeft метода указывают количество единиц прокрутки в соответствующем направлении.

В методе **SmallScroll** единицей является расстояние, на которое смещается содержимое экрана при щелчке на стрелке полосы прокрутки (приблизительно одна строка).

В методе **LargeScroll** единицей прокрутки является содержимое одного экрана. Если опущены все аргументы, оба метода осуществляют прокрутку документа вниз на одну единицу.

Для постраничной прокрутки содержимого документа используется метод **PageScroll.** Например, Window.PageScroll(Down, Up).

Для прокрутки до заданной позиции документа используется метод **ScrollIntoView.**

Window.ScrollIntoView(Obj, Start)

Аргумент Obj определяет объект Range или Shape, к которому нужно переместиться. Аргумент Start не обязателен, он определяет, должен ли левый верхний угол фрагмента, указанного аргументом Obj, совпадать с левым верхним углом экрана (Start = True, принимается по умолчанию) или же должны совпадать правые нижние углы фрагмента и экрана (Start = False).

Объект View

В Word каждый объект **Window** имеет объектное свойство **View,** содержащее ссылку на объект одноименного класса. Свойства объекта **View** определяют многие аспекты отображения окна или его области (табл. 43).

Таблица 43 — Некоторые свойства объекта View

Свойство	Назначение
Fullscreen	Управляет режимом отображения окна в стандартном или полноэкранном варианте
ShowAll	Определяет отображение в документе всех непечатаемых знаков и соответствует флажку Показывать все знаки форматирования в группе Всегда показывать эти знаки форматирования на экране на вкладке Экран диалогового окна Параметры Word. Из программного кода можно также включить или отключить отображение отдельных непечатаемых символов, а также других элементов, таких как выделение текста или отображение границ. Для управления подобными элементами используются различные свойства, названия которых начинаются с **Show** — например **ShowBookmarks** (закладки) или **ShowHigh light** (выделение)
TableGridlines	Управляет отображением сетки таблицы
Type	Определяет способ представления информации в окне, который задается щелчком на кнопках Разметка страницы, Режим чтения, Структура и так далее в группе Режимы просмотра документа на вкладке Вид ленты приложения. Допустимыми значениями свойства являются следующие константы: **wdMasterView, wdNormalView, wdOutlineView, wdPrintView, wdWebView, wdReadingView** и **wdPrintPreview.** Для изменения способа представления документа в окне достаточно соответствующим образом изменить значение данного свойства. Например, оператор **ActiveWindow.View.Type = wdPrintView** вызывает переключение активного окна в режим предварительного просмотра

6. Использование объектов Selection и Range

Объект **Selection** (рис. 29) относится к области окна, а не документа, и представляет собой выделенный в документе текст. Это может быть таблица, текстовое окно, фрагмент текста, изображение или что-нибудь другое, что можно выделить с помощью мыши или клавиатуры. Если в документе ничего не выделено, то объект **Selection** представляет текущее расположение точки вставки, т. е. текстового курсора. Доступ к этому объекту осуществляется через цепочку объектов Application, Window и Pane, а не через объект Document.

Объект **Selection** может представлять содержимое различных типов, поэтому целесообразно сначала проверить, данные какого типа были выделены в документе, и лишь потом выполнять их обработку. В противном случае можно получить самые неожиданные результаты или даже сообщение об ошибке.

Сведения о типе выделенных в документе данных помещаются в свойство **Type** объекта **Selection.** По значению свойства **Type** можно определить тип текущего выделения.

Например, приведенный ниже фрагмент кода проверяет, является ли выделенная область обычным текстом, прежде чем вырезать ее из документа с помещением в буфер обмена, а затем вставить вырезанное на одну страницу выше.

```
With      Selection  'Работаем с текущим выделением в документе
If .Type = wdSelectionNormal Then 'Выделентекстовый блок
.Cut                        'Выделенныйтекст вырезается в буфер обмена
```

```
                    ' Перемещение в документе на страницу вверх
.GoTo What:=wdGoToPage, Which:=wdGoToPrevious, Count:=1
.Paste
                    ' Текст из буфера обмена вставляется в документ
End If
End With
```

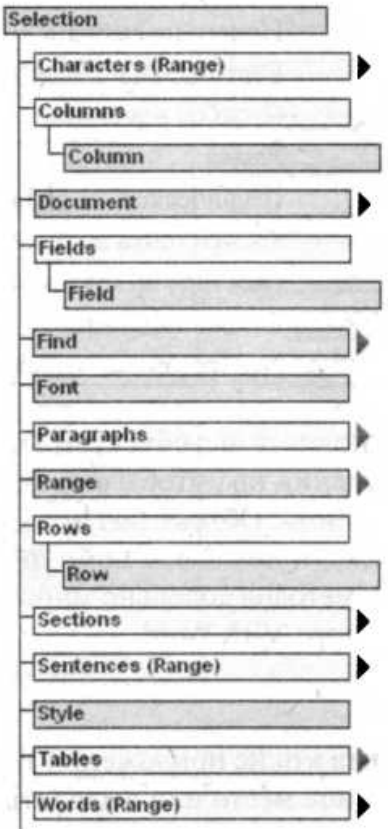


Рисунок 29 — Фрагмент структурной схемы объекта **Selection** *приложения Word*

Все допустимые значения, которые может принимать свойство **Type** объекта **Selection,** определяются глобальными константами, приведенными в таблице 44.

Таблица 44 — Допустимые значения свойства Type

Константа	Что выделено в документе
wdNoSelection	Ничего не выбрано
wdSelectionBlock	Вертикальный блок текста
wdSelectionColumn	Столбец в таблице
wdSelectionFrame	Рамка
wdSelectionInlineShape	Графическое изображение в тексте
wdSelectionIP	Только точка вставки
wdSelectionNormal	Обычное выделение фрагмента текста
wdSelectionRow	Строка таблицы
wdSelectionShape	Изображение, не помещенное в текст

Действия, которые можно выполнять над объектом **Selection,** условно разделяют на две категории — изменение выделения так, чтобы оно включало другую часть документа, и изменение текста внутри выделения. Объект **Selection** имеет свойства, содержащие информацию о выделенном тексте, и методы, с помощью которых этим текстом можно манипулировать.

Объект Range

При редактировании документа в окне приложения Word предварительно необходимо поместить указатель мыши в нужное место или выделить в нем определенный фрагмент и только после этого можно добавлять, удалять или форматировать текст. Однако при работе с Word из VBA-программ необходимость подобных действий можно исключить, если использовать объекты **Range** (диапазоны).

Объект **Range** используется для представления непрерывной области (диапазона) в документе Word. Диапазон определяется позицией первого и последнего входящего в него символа. Примером диапазона является место вставки, фрагмент текста, а также весь документ Word вместе с непечатаемыми символами, такими как символы пробела и абзаца. Использование объекта **Range** позволяет создавать более эффективный код за счет упрощенного способа обращения к требуемым элементам документа Word.

В VBA-программах с помощью объекта **Range** можно помечать требуемую часть документа, однако эта отметка будет существовать только до тех пор, пока работает та процедура, в которой она была создана. Самое важное здесь то, что объекты **Range** совершенно независимы от текущего положения курсора ввода или выделенной части документа (объект **Selection),** которые пользователь видит в окне приложения. Более того, в одном документе этих объектов может быть создано сразу несколько. По-своему назначению этот объект очень похож на объект **Selection.** После создания объекта **Range** можно манипулировать определяемым им текстом с помощью операторов языка VBA, представляющих собой полные аналоги любых команд редактирования Word, точно так же как при использовании объекта **Selection.** Внутренняя структура объекта **Range** показана на рисунке 30.

Определить объект **Range** в программном коде можно одним из двух следующих способов:

- с помощью свойства **Range** объектов различных классов, определяющих различные элементы документа (абзац, сноска, ячейка таблицы и пр.);
- с помощью метода **Range** объекта **Document.**

Открытый документ Word уже содержит объекты **Range,** соответствующие многим его элементам. Для каждого его абзаца или таблицы, отдельной ячейки таблицы, комментария или колонтитула (и это далеко не полный список) объекты **Range** создаются автоматически. Например, для доступа к объекту **Range,** соответствующему первому абзацу активного документа, достаточно указать объектную ссылку следующего вида:

ActiveDocument.Paragraphs(1).Range

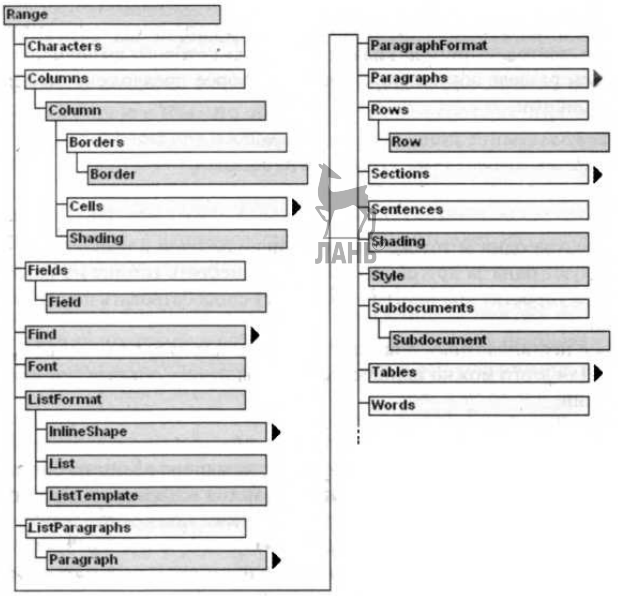


Рисунок 30 — Фрагмент структурной схемы объекта **Range** *приложения Word*

Поскольку в открытом документе Word подобные стандартные диапазоны уже существуют, можно использовать объектные ссылки на них напрямую, не применяя каких-либо дополнительных переменных. Это очень удобно в тех случаях, когда отдельная операция выполняется над определенным диапазоном в целом. Например, приведенный ниже оператор копирует вторую таблицу в документе в буфер обмена, используя метод **Copy** объекта **Range:**

```
ActiveDocument.Tables(2).Range.Copy
```

Если несколько операторов используют один и тот же диапазон, то для упрощения кода программы и ускорения ее выполнения можно использовать конструкцию **With… EndWith.** Ниже приведен пример кода, с помощью которого выполняется сортировка абзацев в третьем разделе документа, после чего второе предложение в нем выделяется полужирным шрифтом.

```
With ActiveDocument.Sections(3).Range
.Sort sortOrder:=wdSortOrderAscending
.Sentences(2).Bold = True
End With
```

Если в процедуре один и тот же диапазон используется в нескольких инструкциях, которые не следуют одна за другой, то полезно поместить ссылку на него в объектную переменную. Это упростит код программы и будет способствовать повышению скорости ее выполнения.

Для работы с нестандартным диапазоном прежде всего следует создать собственный объект **Range.** Для этого можно использовать метод **Range** объекта **Document.** Его синтаксис следующий:

Документ.Range[Start, End]

Здесь **Start** и **End** — необязательные аргументы, задающие позицию символов начала и конца диапазона. Если эти аргументы опущены, метод возвращает диапазон, в котором содержится весь документ. В любом открытом документе можно определить сколько угодно объектов **Range**, используя для этого метод **Range** документа. Требуется лишь указать начальную и конечную точки каждого диапазона в терминах расположения соответствующих символов в документе:

ActiveDocument.Range(Start:= 10, End:=20)

Это выражение представляет собой объектную ссылку на диапазон, который начинается с 11-го символа и заканчивается 20-м символом в активном документе. Числовое значение, определяющее расположение символа, на самом деле указывает на позицию справа от данного символа. Например, значение **0** соответствует первому символу в документе, а значение 10 указывает на точку между 10-м и 11-м символами. Word считает все символы в документе, включая скрытые и непечатаемые знаки.

Для создания диапазона, который будет указывать лишь место расположения в документе и не будет содержать никакого текста, присвойте начальному и конечному значениям, определяющим диапазон, одно и то же число. Для включения в объект **Range** всего документа достаточно вызвать метод **Range** этого документа без указания каких-либо аргументов или же воспользоваться свойством **Content** представляющего его объекта **Document.**

7. Работа с текстом

Объекты **Range** и **Selection** являются основой для практически любых операций, которые можно выполнять с текстом с помощью VBA в Microsoft Word. Некоторые из доступных в этом приложении операций можно применять к документам в целом, однако в общем случае, прежде чем вносить в документ какие-либо изменения, необходимо определить диапазон или выделить ту область, к которой они будут применены.

Объекты **Range** и **Selection** представляют в программе непрерывные последовательности символов, над которыми можно выполнять различные операции, поэтому у них много общих свойств и методов. Однако некоторые свойства и методы применимы только к выделенным областям **(Selection),** а другие — только к диапазонам **(Range).** Принципиальное отличие в том, что объект **Selection** соответствует выделению в области окна: тексту, графическому изображению или любому другому объекту, в то время как объекты **Range** существуют только в программе, независимо от выделенной области, и всегда содержат только текст.

Объект **Selection** имеет смысл использовать в тех случаях, когда программа зависит от действий пользователя — например, он должен указать текст, над которым следует выполнить определенные действия, или, наоборот, когда программа должна показать пользователю, какой именно текст будет изменяться. Во всех остальных случаях программной обработки документа удобнее использовать объекты **Range** — они обеспечивают большую скорость выполнения программ и работа с ними незаметна для пользователя. Это

очень важный момент, поскольку Word автоматически обновляет содержимое экрана при каждом изменении содержимого выделенной области, а при изменении текста в диапазоне (т. е. в объекте **Range)** содержимое экрана не обновляется. Более того, изменения диапазонов не отражаются и в выделенных областях, созданных пользователем.

Дополнительно следует отметить, что объекты **Selection** и **Range** можно создавать один из другого. Иногда эта возможность оказывается чрезвычайно полезной, так как некоторые функции редактирования работают только с диапазонами. И наоборот, единственный доступный способ отобразить новое содержимое диапазона пользователю — это выделить его. Для выделения диапазона, представленного объектом **Range,** используется его метод **Select.** Например, для объекта Диапазон1 соответствующий оператор имеет вид Диапазон1.Select. Аналогичным образом для создания диапазона, представляющего собой текстовое содержимое выделенной в документе области, используется свойство **Range** объекта **Selection.**

Приложение Microsoft Word предоставляет в распоряжение программиста ряд методов для перемещения и изменения размеров диапазонов или выделенных областей. Например, метод **Expand** увеличивает указанный диапазон или выделенную область, добавляя блок текста в их конец. Блок, который может представлять собой символ, слово, абзац, раздел или что-нибудь другое, определяется значением аргумента, указанного при вызове метода. Например, для добавления к выделенной области слова, которое следует непосредственно за ней, необходимо использовать такой оператор.

Selection.Expand(wdWord)

При каждом вызове метода **Expand** к выделению можно добавить только один блок, определяемый указанным аргументом. Добавление подобных блоков в начало диапазона или выделенной области не допускается.

Word позволяет переопределять начало и конец диапазона или выделенной области. Так, метод **Move** изменяет расположение диапазона или выделенной области. Работа этого метода начинается с того, что диапазон или выделение сжимаются к точке их начала или конца, в зависимости от значения аргумента. В результате точка начала и конца будет указывать на одну и ту же позицию в документе, и обрабатываемая область не будет содержать никакого текста. Далее метод **Move** перемещает «сжатый» объект соответственно заданному аргументу (на один символ, одно слово, одно предложение и т. д.) в указанном направлении (к началу или концу документа). Метод возвращает числовое значение, указывающее, на сколько единиц перемещение было действительно выполнено. По окончании перемещения можно воспользоваться методами Expand, MoveStart или MoveEnd для наполнения объекта текстом, начиная с новой позиции его начальной точки (которая в этот момент совпадает с его конечной точкой).

Методы **MoveStart** и **MoveEnd** изменяют соответственно положение начальной или конечной точки диапазона или выделенной области. При этом «сжатие» диапазона, как в методе Move, не выполняется, что позволяет использовать эти методы для расширения (или сужения) обрабатываемой облас-

ти. Приведенная ниже инструкция перемещает начало выделенной области на два слова ближе к концу документа, сокращая тем самым ее размер:

Selection.MoveStart Unit := wdWord, Count := 2

Все методы, в имени которых присутствует слово Move, изменяют лишь расположение диапазона или выделенной области, но не перемещают текст, который содержится в указанном объекте.

Еще одна пара методов, **startOf** и **EndOf**, перемещает начало или конец диапазона либо выделенной области. Метод **StartOf** перемещает начало обрабатываемой области назад, к началу текущего блока, тип которого определяется заданным параметром, тогда как метод **EndOf** перемещает конец объекта вперед, к концу текущего блока. Оба метода имеют два аргумента. Первый определяет тип блока, к началу (или концу) которого перемещается граница области. Второй аргумент может принимать одно из двух допустимых значений. Первое, именованная константа **wdMove,** определяет, что указанным образом следует переместить обе границы выделенной области — и верхнюю, и нижнюю (принимается по умолчанию). Второе, именованная константа **wdExtend,** указывает на то, что следует переместить только верхнюю **(StartOf)** или нижнюю **(EndOf)** границу выделенной области.

Selection.StartOf Unit := wdSentence, Extend := wdMove

Здесь обе границы выделенной в документе области перемещаются к началу того предложения, в котором они расположены. Если при использовании методов **StartOf** и **EndOf** перемещаемая сторона выделенной области уже находится в той позиции, к которой предпринимается попытка ее переместить, ничего не происходит.

Достаточно часто при обработке документа возникает необходимость сжать диапазон или выделенную область в одну точку, которая не содержит никакого текста. Подобное сжатие необходимо, когда в документ требуется вставить поле, таблицу или какой-либо другой элемент, поместив его до или после выделенной области или диапазона, не изменяя выделенного текста. (При вставке некоторого элемента в «несжатый» диапазон или выделенную область он заменит текст, содержавшийся в данной области.) Для сжатия диапазона или выделенной области предназначен метод **Collapse.** Область можно сжать к ее начальной или конечной точке, что определяется необязательным аргументом **Direction.** Так, оператор **Selection.Collapse** сжимает выделенную область к ее начальной точке, тогда как оператор, приведенный ниже, сожмет выделенную область к ее конечной точке.

Selection.Collapse(Direction:=wdCollapseEnd)

Удаление, копирование и вставка текста

Для удаления из документа всего текста, содержащегося в диапазоне или выделенной области, можно воспользоваться методом **Delete** соответствующего объекта. Если же необходимо удалить текст и поместить его в буфер обмена, следует использовать метод **Cut.** Метод **Copy** просто копирует в буфер обмена текст из диапазона или выделенной области, не оказывая на него никакого влияния. Метод **Paste** позволяет вставить текст, помещенный в буфер обмена, в

любой диапазон или выделенную область. Если указанный объект назначения не был предварительно сжат, вставленный текст заменит исходный текст в объекте.

Следует отметить, что использование буфера обмена для вставки или копирования текста нельзя считать эффективным приемом. В VBA-программах удобнее использовать для этой цели свойства **Text** или **FormattedText** диапазона, или выделенной области. Для копирования достаточно присвоить соответствующему свойству области назначения значение аналогичного свойства копируемой области. Не забывайте, что если вставляемый элемент не должен заменить текст, уже существующий в области назначения, то последняя должна быть предварительно сжата. Приведенный ниже фрагмент кода передает текст из выделенной области в сжатый диапазон, связанный с закладкой в документе. Поскольку здесь используется свойство **Text,** в новое место копируется только сам текст; при этом теряется любое его форматирование, присутствующее в исходной области. Для передачи вместе с текстом и его форматирования вместо свойства **Text** следует использовать свойство **FormattedText.**

```
Dim Вставка As Range      ' Объявление рабочей переменной
With ActiveDocument.Bookmarks("Закладка1")
Set Вставка = ActiveDocument.Range(Start:=.Start, End:=.Start)
                                    'Диапазон сжат
End With
Вставка.Text = Selection.Text   'Копирование текста
```

Свойства объектов диапазона или выделенной области, приведенные в таблице 45, определяют внешний вид содержащегося в них текста. Все эти свойства соответствуют аналогичным командам форматирования текста программы Word.

Таблица 45 — Свойства, определяющие форматирование текста в выделенной области

Свойство	Описание
Borders	Представляет собой коллекцию объектов **Border,** описывающих параметры отдельных элементов рамки вокруг текста в выделенной области
Font	Содержит ссылку на объект, описывающий параметры символьного форматирования текста в выделенной области. К ним относятся **Name, Size** и **Bold**
ParagraphFormat	Содержит ссылку на объект, описывающий параметры форматирования абзацев текста в выделенной области. К ним относятся **Left Indent, LineSpacing** и др.
Style	Содержит имя стиля символа или абзаца, примененного к тексту в диапазоне или выделенной области
TabStops	Тип и расположение точек табуляции. Доступ к этим свойствам можно получить только с помощью объектов **Paragraph,** а не напрямую через диапазоны или выделенные области

Поиск и замена информации

Как объект **Selection,** так и объект **Range** имеют свойство **Find,** которое содержит ссылку на объект **Find,** принадлежащий данному диапазону или выделенной области. Для отыскания с помощью объекта **Find** заданного текста или форматирования определенного типа необходимо выполнить следующие действия.

1. Получите ссылку на объект **Find** интересующего диапазона или выделенной области. Если необходимо просмотреть весь документ, воспользуйтесь свойством **Content** объекта **Document:ActiveDocument.Content.Find.**

2. Требуемым свойствам **(Text, Style, Font, Format** и т. д.) объекта **Find** присвойте необходимые значения в соответствии с тем, что именно требуется найти и как следует проводить поиск.

3. Для выполнения поиска вызовите метод **Execute** объекта **Find.** Например, чтобы найти в выделенном на экране фрагменте документа слово «поставка», можно использовать следующий код.

```
With Selection.Find
    .ClearFormatting   'Поиск без учета форматирования
    .Text ="поставка" 'Отыскиваемый текст
    .Execute                'Выполнение поиска
End With
```

В приведенном выше примере используются два метода объекта **Find,** на которых следует остановиться подробнее. Метод **ClearFormatting** удаляет из свойств объекта **Find** сведения об отыскиваемых параметрах форматирования, которые были установлены. В результате следующая операция поиска будет выполнять поиск заданного текста, без анализа особенностей его форматирования. Хорошей практикой можно считать предварительный вызов данного метода при каждом использовании объекта **Find** — это гарантирует, что любые условия учета форматирования при поиске из предыдущих операций будут удалены. Метод **Execute** выполняет поиск очередного вхождения искомого текста или форматирования в указанном диапазоне или выделенной области. После выполнения этого метода сначала следует определить, было ли найдено то, что требовалось. Для подобной проверки предназначено свойство **Found** объекта **Find,** имеющее логический тип данных. Пример использования этого свойства в операторе **If...Then** показан ниже.

```
If .Found = True Then
            'Выполнение требуемых действий с найденным текстом
...  Else
            'Отображение сообщения о том, что образец не найден
...  End If
```

Объект **Replacement** используется, когда выполняется поиск с заменой. Ссылка на этот объект присутствует в одноименном свойстве объекта **Find.** Свойства объекта **Replacement (Text, Style, Font** и пр.) содержат новые значения, на которые заменяются соответствующие характеристики текста, предварительно найденного с помощью метода **Execute** объекта **Find.**

Следующий фрагмент кода заменяет все экземпляры слова «поставка» словом «отгрузка» в части документа, выделенной на экране на данный момент.

```
With Selection.Find
.ClearFormatting      'Очистка форматирования при поиске
.Text = "поставка"          'Искомый текст With
.Replacement
.ClearFormatting            'Очистка форматирования при замене
.Text = "отгрузка"          'Текст замены
End With
'Выполнение поиска и замены всех вхождений
.Execute Replace := wdReplaceAll       'Метод объекта Find
End With
```

Обратите внимание на то, что в вызове метода **Execute** объекта **Find** может присутствовать аргумент **Replace,** который указывает, следует ли заменить все обнаруженные экземпляры искомого текста или только *первый.*

Для поиска текста с определенным форматированием используются соответствующие свойства объекта **Find.** При необходимости задать форматирование для замещающего текста можно использовать аналогичные свойства объекта **Replacement.**

Представленный ниже код проводит поиск абзацев, которым в данный момент назначен стиль **Цитата,** после чего назначает им стиль **Обычный.**

```
With Selection.Find
.ClearFormatting
.Style = "Цитата"
.Text = ""
With .Replacement
.ClearFormatting
.Style = "Обычный"
.Text = ""
End With
.Execute Replace := wdReplaceAll
.ClearFormatting
.Replacement.ClearFormatting
End With
```

Для поиска *любого* текста с определенным форматированием поместите требуемые значения в соответствующие свойства объекта **Find** и присвойте его свойству **Text** значение, равное пустой строке; т. е. укажите вместо значения только пару кавычек. Чтобы изменить форматирование найденного текста, не изменяя сам текст, поместите пустую строку в качестве значения в свойство **Text** объекта **Replacement.**

Использование переменных документа

В отличие от остальных приложений Microsoft Office, Word позволяет определять в VBA-программах специальные *переменные документа*, которые сохраняются вместе с документом. Переменные документа — удобное средство

сохранения любых используемых VBA-программой значений данных между сеансами редактирования.

Переменные документа создаются и используются как члены коллекции **Variables** в данном документе. Как и обычные документы, переменные документа характеризуются собственными именами, но могут хранить в себе только строковые значения. Приведенный ниже оператор присваивает значение переменной документа **Срок реализации** обычной переменной **Установленный_срок.**

```
Dim Установленный_срок As String.
Установленный_срок =ActiveDocument .Variable ("Срок_реализации")
.Value
```

Для создания новой переменной используется метод **Add** коллекции **Variables,** как показано ниже.

```
Documents("Отчет за ноябрь").Variables.Add _Name := "Дата_обработки",
Value := "25.05.2010"
```

При попытке создать уже существующую переменную документа будет выдано сообщение об ошибке, поэтому рекомендуется предварительно проверить существование соответствующего имени и лишь затем создавать новую переменную. Если необходимая переменная уже существует, достаточно будет считать ее текущее значение; если нет — можно создать эту переменную и присвоить ей исходное значение. Данный прием иллюстрируется следующим примером:

```
' Поиск переменной в коллекции по ее имени
Dim DocVar As Variable
Dim Текущее_число_ознакомившихся As String
For Each DocVar In ActiveDocument.Variables
  If DocVar.Name = "Число_ознакомившихся"  Then
DocIndex = DocVar.Index
Next DocVar
If DocIndex = 0 Then 'Переменная не существует Active-
Document.Variables.Add _
Name := "Число_ознакомившихся", Value := "1" Теку-
щее_число_ознакомившихся = "1"
Else        'Переменная уже существует
Текущее_число_ознакомившихся = ActiveDocu-
ment.Variables(DocIndex).Value
End If
```

Контрольные вопросы

1. Что такое макрорекордер?
2. Какие существую способы запуска макроса?
3. В каких местах можно сохранить макрос?
4. Какому объекту подчинены многие объекты, с которыми обычно ведется работа в VBA-программах в Word?
5. Какой метод используется для открытия из VBA-программы некоторого диалогового окна?
6. Чем позволяет управлять объект View?
7. Какие различия имеют объекты Selection и Range?

Лекция 23. Создание технологической документации в среде MS Excel

План:

1. Объектная модель ADO.
2. Методы и свойства объекта ADO.
3. Подключение к таблице Excel средствами ADO.
4. Обработка данных на рабочем листе.

1. Объектная модель ADO

Потребности в обращении из приложений Office к базам данных возникают практически на любом предприятии. Очень часто приложение, которое изначально предназначалось для работы с данными, которые находятся в самом приложении, по мере увеличения объема данных приходится переделывать под работу с клиент-серверными источниками. Примеры базы данных:

- клиент-серверные: Microsoft SQL Server, Oracle, IBM DB2;
- настольные: Access, FoxPro, DBase, Paradox.

ADO расшифровывается как **ActiveX Data Objects** — набор программных объектов, построенных по технологии ActiveX (COM) и позволяющих получать данные и управлять ими на самых разных источниках. В настоящее время появилась новая версия ADO — ADO.NET, которая сильно отличается от обычной ADO и предназначена для работы в .NET Framework. ADO.NET обязательно требует установленной .NET Framework; обычными средствами с ADO.NET из редактора Visual Basic работать не получится — требуется Visual Studio; отличается повышенной ресурсоемкостью.

ADO умеет работать с самыми разными драйверами для подключения к базам данных, например с драйверами OLE DB и ODBC. Поскольку ADO построен по технологии COM, эти объекты можно использовать в любых COM-совместимых языках программирования (VC++, Visual Basic, Delphi, VBA, VBScript, JScript, ActivePerl и т.п.).

Объектная модель ADO имеет всего три главных объекта:

1 <u>объект Connection</u> — позволяет установить соединение с источником данных и управлять им. Все ошибки, которые возникают в ходе работы соединения, помещаются в коллекцию Errors;

2 <u>объект Command</u> — представляет команду, при помощи которой производится выполнение определенной операции на источнике данных (выполнение запроса, хранимой процедуры, создание или изменение объекта, изменение данных и т. п.). Если источник данных — SQL-совместимый, то объект Command, скорее всего, будет представлять команду SQL. Объекту Command сопутствует коллекция Parameters — параметры, которые передаются запросу или хранимой процедуре;

3 <u>объект Recordset</u> — представляет набор записей, полученных с источника или сгенерированный другим способом. Ему сопутствует коллекция Fields, представляющая информацию о столбцах в этом наборе записей (имя, тип, размерность данных и т.п.), а также сами данные.

Для каждого из этих трех объектов предусмотрена также коллекция Properties, которая определяет соответственно свойства соединения, команды или набора за-

писей. Все объекты явно создавать необязательно — например, при создании объекта Recordset можно в автоматическом режиме создать объект Connection.

2. Методы и свойства объекта ADO

Provider — драйвер для подключения к источнику данных. Для каждого типа источника данных (SQL Server, Access, Oracle) он свой. Мы будем использовать подключение по ODBC к таблице на листе Excel. IntegratedSecurity — в данном случае это свойство используется, поскольку мы используем для подключения к SQL Server аутентификацию Windows.

DataSource — имя источника данных. В нашем случае это имя компьютера, на котором работает SQL Server. В других случаях оно могло бы быть именем экземпляра Oracle или файла базы данных Microsoft Access — все зависит от того, к какой базе данных вы подключаетесь.

InitialCatalog — имя базы данных на этом сервере. В нашем случае — Northwind.

Когда вы генерируете строку подключения автоматически при помощи файла UDL, в первый раз это может показаться долгим. На самом же деле это занимает не более минуты. Кроме того, при этом вы гарантируете, что в строке подключения не будет ошибок, и у вас есть возможность проверить подключение к базе данных прямо из свойств файла UDL (кнопка Test Connection).

Свойство Provider позволяет определить драйвер, который будет использован для подключения к базе данных. Мы с вами определили такой драйвер внутри значения ConnectionString, но можно для этой цели использовать и отдельное свойство.

Свойство ConnectionString — главное свойство объекта Connection. Оно определяет параметры подключения к источнику.

Метод Open() позволяет открыть соединение с базой данных. Метод Close() позволяет закрыть соединение.

Для этого объекта предусмотрено множество других свойств и методов, однако здесь они рассматриваться не будут. Единственное свойство, которое обязательно необходимо рассмотреть, — это свойство Errors, которое возвращает коллекцию объектов Error — ошибок. Ошибки при установке или работе соединения встречаются очень часто (неверно введен пароль или имя пользователя, у пользователя недостаточно прав на подключение, невозможно обратиться к компьютеру по сети и т.п.), поэтому настоятельно рекомендуется реализовывать в программе обработку ошибок.

Самые важные свойства объекта ADOError:

Description — описание ошибки.

Number — номер ошибки. По номеру удобно производить поиск в базе знаний и в Интернете.

Source — источник ошибки. Эта информация полезна только в том случае, если в коллекции Errors могут оказаться ошибки из разных источников.

SQLState и NativeError — информация о возникшей ошибке, которая пришла с SQL-совместимого источника данных.

3. Подключение к таблице Excel средствами ADO

Чтобы подключиться к файлу Excel, необходимо:

- создать именованный диапазон в книге Excel;
- создать источник данных ODBC;
- написать три строки кода, начиная с создания объекта Connection до вызова его метода Open.

Что такое объект Recodrset? Само слово Recodrset расшифровывается как Set of Records, то есть набор записей. Проще всего представить его как табличку (аналогичную таблицам в Excel), которая находится в оперативной памяти компьютера.

Recordset — это «строгая» таблица. В ней четко определены столбцы и строки, и разрывов она не допускает (хотя какие-то значения на пересечении строк и столбцов вполне могут быть пустыми); в таблице Excel в одном столбце без проблем можно использовать самые разные значения — числовые, даты времени, строковые, формулы и т. п.

При открытии объекта Recordset есть возможность определить еще несколько очень важных его свойств:

Первое свойство — *CursorType*, тип курсора. Это свойство определяется только перед открытием Recordset (после открытия оно доступно только на чтение). Курсор можно представить себе как указатель на записи в Recordset. В зависимости от типа курсора мы определяем возможности работы с Recordset и производительность выполняемых операций.

Второе важное свойство — *CursorLocation.* Оно определяет, где будет создан курсор — на сервере или на клиенте. По умолчанию используется значение adUseServer — создавать на сервере. Есть возможность использовать значение adUseClient — создавать на клиенте. В целом практически во всех ситуациях удобнее и производительнее использовать серверные курсоры, за одним исключением — в реализации серверных курсоров на разных источниках данных больше отличий, поэтому если вы планируете работать с разными источниками, есть смысл подумать о клиентских курсорах.

Третье важное свойство — *LockType*. Это свойство определяет тип блокировок, которые будут наложены на записи на источнике, помещенные в Recordset.

После того как объект Recordset создан, нам необходимо выполнять с ним различные операции. Рассмотрим методы Move…():

Move() — этот метод принимает два параметра: NumRecords — на сколько записей необходимо переместиться (это число может быть и отрицательным, что значит — переместиться назад), и второй параметр (необязательный) — имя закладки, с которой нужно начать перемещение. Можно использовать три встроенные закладки: для текущей, первой и последней записи. Если имя закладки не указано, то перемещение начинается с текущей позиции.

MoveFirst(), MoveLast(), MoveNext() и MovePrevious() — назначение этих методов понятно из названия: перемещение на первую, последнюю, следующую и предыдущую запись соответственно.

Если нужно напрямую перепрыгнуть на нужную запись, можно использовать методы Find() и Seek().

Метод Find() предназначен для поиска по значению одного столбца. Он принимает в качестве параметра критерий поиска, насколько нужно отступить от исходной позиции, направление поиска и откуда нужно начать поиск. Очень удобно, что при определении критерия поиска можно использовать оператор Like с подстановочными символами. При обнаружении нужной записи метод Find() переставляет курсор на найденную запись, если же запись не обнаружена, то курсор устанавливается на EOF (или BOF, если поиск был назад).

Метод Seek() отличается от метода Find() тем, что они ищет значение по индексу (объект Index для Recordset создается либо программным способом, либо автоматически — если на таблицу, на основе который был создан Recordset, было наложено ограничение Primary Key). Этот метод работает только для серверных курсоров с типом команды TableDirect и поэтому к использованию не рекомендуется.

Главное содержание Recordset — это то, что лежит в ячейках на пересечении строк (в Recordset они называются записями — records и представлены соответствующими объектами Record) и столбцов. В Recordset столбцы называются полями — объектами Field, которые сведены в коллекцию Fields. Объекты Record используются нечасто — поскольку имен у них нет, а переходить между записями проще при помощи свойств и методов самого объекта Recordset — AbsolutePosition, Find(), Move() и т. п. Коллекция же Fields и объекты Field используются практически в каждой программе.

У коллекции Fields все свойства стандартные, как у каждого объекта Collection:

Count — сколько всего столбцов в Recordset;

Item — возможность вернуть нужный столбец (объект Field) по имени или номеру.

Методы же имеются как стандартные, так и специфические:

Append() — возможность добавить новый столбец в Recordset. Delete() — соответственно удалить столбец. Обе команды разрешено выполнять только на закрытом Recordset (пока не был вызван метод Open() или установлено свойство ActiveConnection);

Update() — сохранить изменения, внесенные в Recordset (будет произведена попытка создать новый столбец на источнике данных, если источник данных по каким-то причинам принимать эти изменения отказался, возникнет ошибка), CancelUpdate() — отменить изменения, внесенные в Recordset;

Refresh() — загадочный метод, который ничего не делает (о чем честно написано в документации). Обновить структуру Recordset данными с источника можно только методами самого объекта Recordset;

Resync() — работает только для коллекции Fields объекта Record (не Recordset), обновляя значения в строке.

Намного больше интересных свойств у объекта Field:

ActualSize — реальный размер данных для текущей записи, DefinedSize — номинальный размер данных для столбца (в байтах) в соответствии с полученной с источника информацией;

Attributes — возможность определить битовую маску для атрибутов столбца (допускает ли пустые значения, можно ли использовать отрицательные значения, можно ли обновлять, используется ли тип данных фиксированной длины и т. п.);

Name — просто строковое имя столбца. Для столбцов, полученных с источника, доступно только для чтения;

NumericScale и Precision — значения, которые определяют соответственно допустимое количество знаков после запятой и общее максимальное количество цифр, которое можно использовать для представления значения;

Value — самое важное свойство объекта Field. Определяет значение, которое есть в столбце (если мы пришли через коллекцию Fields объекта Record, то для этой записи, если через Fields объекта Recordset — то для текущей записи);

ADO позволяет работать с большими двоичными данными (изображения, документы, архивы), что очень удобно.

Свойство Value — это свойство по умолчанию, поэтому эти две строки равноценны:

```
Debug.Print rs.Fields("CompanyName")
Debug.Print rs.Fields("CompanyName").Value
```

Status — значение, отличное от adFieldOK (значение 0), означает, что поле было недавно программно добавлено в Recordset или при добавлении возникла ошибка на источнике данных;

Type — тип данных в соответствии с приведенной в документации таблицей.

У объекта Field есть только два метода — AppendChunk() и GetChunk(). Оба этих метода используются только для работы с большими двоичными типами данных (изображениями, документами и т. п.), когда работать обычными способами через свойство Value не получается.

Данные в Recordset помещаются в том порядке, как они пришли из источника. Если специальный порядок сортировки в запросе не указан, то данные возвращаются в соответствии с параметрами источника данных. Можно произвести сортировку на сервере, а можно — сортировку на клиенте.

Общее правило выглядит так: если есть возможность, всегда нужно выполнять сортировку на сервере. Сервер намного лучше оптимизирован для выполнения подобных операций, на нем обычно больше оперативной памяти и процессорных ресурсов. На клиенте сортировку выполнять следует только тогда, когда это невозможно сделать на сервере.

В Recordset записи можно фильтровать. Отфильтрованные записи остаются в Recordset, но являются невидимыми при выполнении операции перемещения и поиска и курсор на отфильтрованных записях не установить. В принципе всего можно обойтись фильтрацией записей в запросе к источника или вставить дополнительные проверки при операциях перемещения, но иногда с синтаксической точки зрения удобнее использовать именно фильтрацию. Для фильтрации используется свойство Filter.

Очень часто возникает необходимость из приложения не только получать информацию о записях из источника, но и вносить на источник изменения. При этом обычно возникает множество сложностей, связанных с решением вопроса о том, в какой таблице данные изменять (если у нас набор таблиц), с блокиров-

ками, производительностью, разрешениями, каскадными обновлениями, возможностью отката внесенных пользователем изменений и т. п. Многие проблемы решаются намного проще, если вы изначально будете следовать правилу: любые изменения можно проводить только при помощи хранимых процедур (и соответственно при помощи объекта Command).

Необходимо также помнить, что значение свойства LockType при открытии Recordset по умолчанию устанавливается в adLockReadOnly, что не позволяет вносить изменения в Recordset. Вам потребуется изменить значение этого свойства перед открытием Recordset.

Общая схема внесения изменений через Recordset выглядит так: вначале вызывается один из нужных нам методов (AddNew(), Edit(), Delete()) и производится внесение изменений в оперативной памяти на клиенте. Следующая операция — вызывается метод Update() для Recordset, который и производит запись внесенных изменений на источник данных (после вызова Delete() вызывать метод Update не нужно).

Свойства AbsolutePage, PageSize, PageCount позволяют использовать группы записей — страницы — для перемещения по Recordset. По умолчанию размер страницы равен 10 записям.

Свойство ActiveCommand позволяет вернуть объект Command, представляющий собой команду, которая использовалась на источнике при создании Recordset и заполнении его записями.

Свойство ActiveConnection возвращает объект Connection, который использовался для создания Recordset. Передать (или получить) строковое значение, на основе которого будет создан объект Connection, можно при помощи свойства Source.

Свойство CacheSize позволяет определить количество записей, которое будет находиться в оперативной памяти на клиенте (остальные записи будут подкачиваться по мере необходимости с источника). Используется тогда, когда количество записей в Recordset очень большое или приходится работать с записями очень большого размера, например, большими двоичными объектами.

Свойства DataSource и DataMember используются только при применении Data Environment и здесь рассматриваться не будут.

Свойство EditMode позволяет определить состояние текущей записи — не изменялась, изменялась, но изменения еще не переданы на источник, удалена и т. п.

Свойства InsertCommand, DeleteCommand, UpdateColumn позволяют определить объекты Command, представляющие команды, которые будут использоваться на источнике при создании, удалении и изменении записей в Recordset соответственно.

MarshalOptions — позволяет определить, какие записи при изменении Recordset будут возвращаться с клиента на сервер — все (по умолчанию) или только измененные.

Свойство MaxRecords настоятельно рекомендуется указывать перед открытием для всех Recordset, для которых потенциально возможно получить с источника очень большое количество записей (что может привести к нехватке оперативной памяти на клиенте). Оно определяет максимальное количество за-

писей, которые могут быть скачаны в Recordset. Вместо этого свойства можно использовать и CacheSize.

Свойство State позволяет определить, что в настоящее время происходит с Recordset. Используется одно из 5 значений: открыт, закрыт, соединяется, выполняет команду на источнике или получает оттуда данные.

Свойство Status позволяет определить результат последней операции обновления данных.

StayInSync — свойство, которое используется только иерархических Recordset. Определяет, будут ли при перемещении родителя перемещаться в иерархии и его дети (по умолчанию) или останутся на месте.

Методы:

- Cancel() позволяет прервать открытие Recordset (например, если оно затянулось);
- CancelBatch() и CancelUpdate() позволяют отменить внесенные в Recordset изменения (до вызова команды Update()) в разных режимах;
- Clone() позволяет скопировать объект Recordset в другой объект Recordset со всеми закладками (обычно используется, когда очень надо иметь более чем одну текущую запись);
- Close() позволяет высвободить память, занимаемую данными Recordset (но не удаляет сам объект). В случае необходимости вы можете опять вызвать метод Open(), чтобы воссоздать этот объект с ранее определенными параметрами (значения свойств объекта Recordset при вызове метода Close() сохраняются);
- CompareBookmarks() позволяет сравнить две закладки и вернуть результат сравнения (указывают на ту же запись, первая выше, первая ниже и т.п.)
- GetRows() позволяет вернуть из Recordset двухмерный массив типов Variant. В качестве необязательных параметров принимает стартовую позицию, количество строк, которые нужно поместить в массив, и те столбцы, которые нужно извлечь из Recordset;
- GetString() — самая простая возможность получить из объекта Recordset строковое значение. По умолчанию разделители между столбцами — символы табуляции, между записями — перевод каретки;
- Requery() — повторно выполнить запрос, который использовался для метода Open(), и заново заполнить Recordset;
- Resync() — обновить значения уже полученных записей, скачав их заново из источника. Новые записи при этом видны не будут (в отличие от метода Requery());
- Save() — возможность сохранения Recordset в файле на диске. Можно использовать формат Microsoft Advanced Data TableGram (ADTG), XML и родной формат провайдера;
- SetAllRowsStatus() — возможность изменить значение свойства Status для всех строк Recordset;
- Supports() — возможность выполнить проверку того, что поддерживает данный Recordset (в каких направлениях можно двигаться, поддерживается ли по-

иск, закладки и т. п.). Те возможности, которые проверяются этим методом, определяются особенностями провайдера (драйвера для подключения к источнику).

Для объекта Recordset предусмотрен также набор событий (EndOfRecordset, FetchComplete, MoveComplete), но используются они нечасто и поэтому здесь рассматриваться не будут.

В самых простых вариантах, когда можно получить и изменять данные напрямую в таблицах, можно обойтись объектом Recordset. Однако во многих ситуациях возможностей только этого объекта недостаточно. Как уже говорилось, предпочтительнее производить любое внесение изменений на источник данных при помощи хранимых процедур.

Для выбора типа команды используется свойство CommandType. Значения, которые ему можно присвоить, аналогичны возможным значениям параметра Options метода Open() объекта Recordset, которое было рассмотрено выше. Например, если мы передаем команду на выполнение хранимой процедуры, то присвоить соответствующее значение можно так:

cmd.CommandType = adCmdStoredProc

Следующее действие — определить текст команды, которая будет выполняться. Делается это при помощи свойства CommandText. Например, если мы хотим запустить на выполнение хранимую процедуру CustOrderHist, то соответствующий код может выглядеть так: cmd.CommandText = "CustOrderHist".

Чаще всего хранимая процедура требует передачи ей одного или нескольких параметров. Делается это при помощи коллекции Parameters и объектов Parameter. После этого команду необходимо запустить на выполнение. Для этого используется метод Execute(). Самый простой способ его вызова выглядит так: cmd.Execute.

4. Обработка данных на рабочем листе

Способы агрегирования данных

Встроенные в Excel команды и методы позволяют эффективно работать с диапазоном: заполнять его элементами по образцу, сортировать, фильтровать и консолидировать данные, строить итоговую таблицу и создавать сценарии, решать нелинейное уравнение с одной переменной.

Excel имеет разные способы агрегирования данных.

Outline — возвращает одноименный объект, задающий структурированное представление рабочего листа. Зачастую данные, представленные на рабочем листе, можно структурировать, сжимая или разворачивая их по мере необходимости. Типичной является ситуация, когда данные, отражающие работу некоторого предприятия, представлены по дням, неделям, месяцам, кварталам и годам. При глобальном анализе деятельности предприятия нас могут интересовать только сводные результаты за каждый год, в этом случае нижние уровни структуры будут свернуты, но при необходимости их всегда можно развернуть вплоть до ежедневного анализа. Поскольку таблица двумерная, то возможны два направления свертки данных. Так, например, второе направление может отражать структуру предприятия: цеха, участки, группы, отдельного работника.

Метод **ShowLevels(RowLevels, ColumnLevels)** объекта Outline позволяет показать структуру рабочего листа, где уровни детализации по строкам и

столбцам задают параметры метода. Чаще всего для проведения подобного анализа целесообразнее использовать сводные таблицы — объект **PivotTablle**.

Другим средством являются сводные таблицы. Еще одну возможность объединения данных дает их консолидация. Как правило, консолидируются однотипные данные, построенные, например, на основе единого шаблона. Можно, например, консолидировать данные, представляющие результаты работы однотипных подразделений. Что реально скрывается за термином «консолидация», определяет функция консолидации — чаще всего это функция Sum, проводящая обычное суммирование. Но это может быть и нахождение среднего или минимального (максимального) значения.

Свойство *ConsolidationFunction*, предназначенное только для чтения, возвращает константу, задающую код функции консолидации: xlAverage, xlCount, xlCountNums, xlMax, xlMin, xlProduct, xlStDev, xlStDevP, xlSum, xlVar, или xlVarP.

Свойство *ConsolidationSources* возвращает массив строк, содержащий имена листов, служивших источниками для консолидации данных.

Свойство *ConsolidationOptions* возвращает трехэлементный массив булевых переменных, каждая из которых имеет значение True, если одна из трех соответствующих опций установлена.

Объекты PivotTablle

PivotTableWizard — создает сводную таблицу.

RefreshAll — обновляет сводные таблицы и все области, содержащие внешние данные.

PivotCaches — возвращает коллекцию областей памяти, отводимых сводным таблицам данной рабочей книги. Элементами этой коллекции являются объекты PivotCache. Каждой сводной таблице — объекту PivotTable — отводится своя память (кэш), которую и задает объект PivotCache.

Метод AutoFill. Метод AutoFill (автозаполнение) автоматически заполняет ячейки диапазона элементами последовательности. Метод AutoFill отличается от метода DataSeries тем, что явно указывается диапазон, в котором будет располагаться прогрессия. Вручную этот метод эквивалентен расположению указателя мыши на маркере заполнения выделенного диапазона (в который введены значения, порождающие создаваемую последовательность) и протаскиванию маркера заполнения вдоль диапазона, в котором будет располагаться создаваемая последовательность.

Синтаксис: объект. AutoFill(диапазон, тип)

Аргументы:

Диапазон, с которого начинается заполнение. Допустимые значения: xlFillDefault, xlFillSeries, xlFillCopy, xlFillFormats, xlFillValues,xlFillDays, xlFillWeekdays, xlFillMonths, xlFillYears, xlLinearTrend, xlGrowthTrend. По умолчанию xlFillDefault

AutoFiter — возвращает одноименный объект, позволяющий производить фильтрацию данных в специальные рода Excel-запросах.

Метод AutoFilter. Метод AutoFilter (автофильтр) представляет собой простой способ запроса и фильтрации данных на рабочем листе. Если AutoFilter активизирован, то каждый заголовок поля выделенного диапазона данных пре-

вращается в поле с раскрывающимся списком. Выбирая запрос на вывод данных в поле с раскрывающимся списком, осуществляется вывод только тех записей, которые удовлетворяют указанным условиям. Поле с раскрывающимся списком содержит следующие типы условий: Все (All), Первые десять (Top 10), Условие (Custom), конкретный элемент данных, Пустые (Blanks) и Непустые (NonBlanks). Вручную метод запускается посредством выбора команды Данные, Фильтр, Автофильтр (Data, Filter, AutoFilter).

При применении метода AutoFilter допустимы два синтаксиса.

Синтаксис 1: Объект. AutoFilter.

В этом случае метод AutoFilter выбирает или отменяет команду Данные, Фильтр, Автофильтр (Data, Filter, AutoFilter), примененную к диапазону, заданному в аргументе объект.

Синтаксис 2: Объект. AutoFilter (field, criteria1, operator, criteria2).

В этом случае метод AutoFilter выполняет команду Данные, Фильтр, Автофильтр (Data, Filter, AutoFilter) по критериям, указанным в аргументе.

Аргументы:

- field — целое, указывающее поле, в котором производится фильтрация данных;
- criteria1 — задают два возможных условия фильтрации и criteria2 поля. Допускается использование строковой постоянной, например 101, и знаков отношений >, <,>=, <=, =, <>;
- operator — допустимые значения: X1And (логическое объединение первого и второго критериев); X1or (логическое сложение первого и второго критериев).

При работе с фильтрами полезны метод showAllData и свойства FilterMode и AutoFilterMode.

Метод **ShowAllData** показывает все отфильтрованные и неотфильтрованные строки рабочего листа.

Свойство **FilterMode** — допустимые значения: True (если на рабочем листе имеются отфильтрованные данные со скрытыми строками), False (в противном случае).

Свойство **AutoFilterMode** — допустимые значения: True (если на рабочем листе выведены раскрывающиеся списки метода AutoFilter), False (в противном случае).

Метод GoalSeek

Метод GoalSeek (подбор параметра) подбирает значение параметра (неизвестной величины), являющееся решением уравнения с одной переменной. Предполагается, что уравнение приведено к виду: правая часть является постоянной, не зависящей от параметра, который входит только в левую часть уравнения. Вручную метод GoalSeek выполняется с помощью команды Сервис, Подбор параметра (Tools, Goal Seek). Метод GoalSeek вычисляет корень, используя метод последовательных приближений, результат выполнения которого, вообще говоря, зависит от начального приближения. Поэтому для корректности нахождения корня надо позаботиться о корректном указании этого начального приближения.

Синтаксис: Объект. GoalSeek(Goal, ChangingCell)

Аргументы:

Объект — Ячейка, в которую введена формула, являющаяся правой частью решаемого уравнения. В этой формуле роль параметра (неизвестной величины) играет ссылка на ячейку, указанную в аргументе ChangingCell;

Goal — значение левой части решаемого уравнения, не содержащей параметра;

ChangingCell — ссылка на ячейку, отведенную под параметр (неизвестную величину). Значение, введенное в данную ячейку до активизации метода Goalseek, рассматривается как начальное приближение к искомому корню.

Точность, с которой находится корень и предельно допустимое число итераций, используемых для нахождения корня, устанавливается свойствами Maxchange и Maxiterations объекта Application. Например, определение корня с точностью до 0,0001 максимум за 1000 итераций устанавливается инструкцией:

```
With Application
Maxiterations = 1000
MaxChange = 0.0001
End With
```

Вручную эти величины устанавливаются на вкладке Вычисления (Calculation) диалогового окна Параметры (Options), вызываемого командой Сервис, Параметры (Tools, Options).

Метод Sort

Сортировка позволяет выстраивать данные в лексикографическом порядке по возрастанию или убыванию. Метод sort осуществляет сортировку строк списков и баз данных, а также столбцов рабочих листов с учетом до трех критериев, по которым производится сортировка. Сортировка данных вручную совершается с использованием команды Данные, Сортировка (Data, Sort).

Синтаксис:

Объект. Sort(key1, order1, key2, order2, key3, order3, header, orderCustom, matchCase, orientaticn)

Аргументы:

Объект Диапазон, который будет сортироваться

Key1 — ссылка на первое упорядочиваемое поле;

Order1 — задает порядок упорядочивания. Допустимые значения: xlAscending (возрастающий порядок); xlDescending (убывающий порядок);

key2 — ссылка на второе упорядочиваемое поле;

order2 — задает порядок упорядочивания. Допустимые значения: xlAscending (возрастающий порядок); xlDescending (убывающий порядок);

header — допустимые значения: xlYes (первая строка диапазона содержит заголовок, который не сортируется); xlNo (первая строка диапазона не содержит заголовка, по умолчанию считается данное значение); xlGuess (Excel решает, имеется ли заголовок);

orderCustom — пользовательский порядок сортировки. По умолчанию используется Normal;

matchCase — допустимые значения: True (учитываются регистры) и False (регистры не учитываются);

orientation — допустимые значения: xlTopToBottom (сортировка осуществляется сверху вниз, т. е. по строкам); xlLeftToRight (слева направо, т. е. по столбцам);

Например, диапазон A1:C20 рабочего листа лист1 сортируется следующей командой в порядке возрастания так, что первоначальная сортировка происходит по первому столбцу этого диапазона, а второстепенная — по второму:

Worksheets(«Лист»).Range(«A1: C20»).Sort _
key1:=Worksheets(«Sheet1»).Range(«A1»),
key2:=Worksheets(«Sheet1»).Range («B1»)

Для приведения введенных данных к нужному типу в VBA включен обширный набор функций, одна из которых — CDBL.

Синтаксис: CDbl(выражение)

Обязательный аргумент выражение является любым строковым или числовым выражением. Для считывания информации, введенной в текстовое поле, в созданной форме вводят переменную и прописывают выражение:

A = Cdbl(textBoxN.text)

Для выведения значений непосредственно в ячейки книги Excel удобно использовать объект Range:

range(«A5»).value = a

Функцией, обратной по действию к CDbl, является функция CStr — она переводит числа в строки и удобна для вывода результата либо в ячейку на лист, либо в то или иное текстовое окно.

TextBoxN.text = CStr(.Range(«A8»).value) — считывание значения с ячейки и вывод его в текстовое окно.

Контрольные вопросы

1. Какие существуют способы доступа к удаленным источникам данных?
2. Что такое объектная модель ADO?
3. Какое назначение имеет модель ADO?
4. Какие преимущества имеет ADO по сравнению с объектными моделями DAO, ADO.NET и др.?
5. Какие способы агрегирования имеет данных Excel?
6. Как создать сводную таблицу средствами VBA?
7. Для чего предназначен метод AutoFill?
8. Для чего предназначен метод GoalSeek?
9. Для чего предназначен метод AutoFilter?

Лекция 24. Управление базами данных Access на VBA

План:

1. Модели объектов Access.
2. Процедурное и структурное программирование в DAO.
3. Процедурное и структурное программирование в ADO.
4. Визуальное программирование в Access.

1. Модели объектов Access

Access представлен двумя уровнями компонентов: ядром базы данных Jet и системой управления базой данных Access. На уровне ядра находятся данные, то есть таблицы и запросы, а также файлы, хранящие компоненты системы управления. Для организации данных используется индексно-последовательный метод (ISAM), в соответствии с которым каждая запись имеет переменную длину и хранится на странице объемом до двух килобайт. Выборка данных поддерживается механизмом запросов SQL и программным доступом на VBA.

Система управления Access обслуживает интерфейс пользователя (формы, отчеты, макросы, меню, панели, окна диалога) и процедуры VBA. Первый уровень поддерживается двумя моделями объектов. Первая из них базируется на библиотеках классов DAO (Data Access Objects), вторая на библиотеках ADO (ActiveX Data Objects ADODB, ActiveX Data Objects Extensions for DDL and Sequrity ADOX, Microsoft Jet and Replication Objects JRQ).

ADODB обеспечивает приложению доступ к источнику данных с возможностью отбора и изменения данных. ADOX позволяет программно изменять структуру объектов источника данных и систему защиты баз данных. JRQ служит для создания, модификации и синхронизации реплик баз данных Access.

Второй уровень строится на библиотеке Access. Библиотека классов DAO ориентирована на работу с данными. Базовым классом DAO является DBEngine, описывающий семейства Errors (Ошибки) и Workspaces (Рабочие области). Каждая рабочая область Workspace характеризуется классами Databases (Базы данных), Groups (Группы), Users (Пользователи).

Наиболее часто используется семейство Recordset (Результирующие наборы записей) класса Database. Каждое множество Recordset основывается на записях таблицы или на описании запроса и позволяет находить, добавлять, изменять или удалять записи. Структуры таблиц базы данных хранятся в семействе класса TableDefs, в частности в объектах его классов Fields (Поля), Indexes (Индексы). В семействе класса Relations (Связи) размещаются схемы данных таблиц. Структура запросов базы данных описывается семейством класса QueryDefs (Запросы) с объектами классов Fields (Поля), Parameters (Параметры).

На работу с данными рассчитана и модель ADO. Во главе этой модели стоит объект Connection (Соединение). Он описывает среду, в которой выполняется обмен данными. Источник данных управляется производным от Connection объектом Command (Команда), который командами SQL добавляет, удаляет, обновляет и считывает данные. Его семейство Parameters (Параметры) представляет переменные компоненты объекта Command. Другой производный от Connection объект Recordset накапливает считанные из источника данные. Его семейство Fields представляет поля таблиц Recordset. Поля характеризуются семействами свойств Properties. Встроенные свойства являются частью объекта ADO и всегда доступны, а динамические свойства существуют только в момент работы источника данных. Библиотеку классов Access возглавляет класс Application (Приложение), описывающий семейства Forms, Reports, Modules, References, DataAccessPages, Controls и такие объекты, как Screen, DoCmd, Module, Assistant, CommandBar.

Обращение к членам библиотек классов DAO и ADO и все обращения в запросах SQL выполняется по схеме:

имяКласса.имяОбъекта.Член

Имена длиной более одного слова заключаются в прямоугольные скобки. При ссылке на объект в активном окне базы данных или контейнера (объекта, содержащего адресуемый объект) имя класса иногда можно опускать. Для упрощения обращений к объектам используется оператор Set, ставящий в соответствие описываемому объекту объектную переменную.

2. Процедурное и структурное программирование в DAO

В общем случае доступ к данным библиотек DAO организуется в три этапа. Сначала создается объект методом Create его родителя. Затем устанавливаются его свойства, которые после сохранения объекта в базе данных останутся доступными только для чтения. В ряде случаев, кроме свойств, сразу же задаются и объекты-потомки, например поля таблиц. В заключение объект добавляется в состав семейства методом Append этого семейства и сохраняется в памяти. Отдельные классы объектов создаются по упрощенной схеме. Сеанс доступа к данным модели DAO описывает класс Workspace. Все действия в рамках сеанса определены правилами доступа пользователя, а их последовательность транзакция рассматривается как одно целое. В незащищенных базах данных открывается сеанс по умолчанию под именем Default Workspace для пользователя admin без пароля. При создании нового сеанса Workspace

DBEngine!CreateWorkspace _ (имяРабочейОбласти, Пользователь, Пароль [, Тип])

его не добавляют в семейство ввиду временного характера его существования.

Метод CreateDatabase используют для создания новых баз данных:

[рабочаяОбласть].CreateDatabase (имяБазыДанных, _ Язык, [Параметры])

Здесь ранее созданная рабочаяОбласть служит ссылкой на объект Workspace, имяБазыДанных длиной до 255 символов представляет короткий или полный путь к файлу .MDB, Язык задается значением констант dbLangGeneral, dbLangCirillic, определяя порядок сортировки данных, а необязательные Параметры задают формат ядра Jet и необходимость шифрования. Базы данных автоматически добавляются в соответствующие семейства и сохраняются. Для открытия базы данных используется метод OpenDatabase:

[базаДанных.] [рабочаяОбласть.]OpenDatabase (имяБазыДанных _ [, Монопольность [, толькоЧтение[, Источник]]])

Если база данных уже открыта, к ней удобно обращаться через функцию CurrentDb:

- CurrentDB!имяТаблицы!имяПоля.имяСвойства;
- CurrentDB!QueryDefs!имяЗапроса;
- CurrentDB!QueryDefs («имяЗапроса»);
- CurrentDB!QueryDefs (индекс);
- CurrentDB!QueryDefs (ссылка).

Результирующие множества записей — объекты классов TableDef и QueryDef — создаются методами CreateTableDef, CreateQueryDef:

- [рабочаяОбласть.]базаДанных.CreateTableDef («имяТаблицы»);
- [рабочаяОбласть.]базаДанных.CreateQueryDef ([запрос, SQL]).

Если при вызове метода CreateQueryDef аргументы не заданы, можно присвоить новому объекту значения свойств Name и SQLtext позднее, посредством оператора присваивания. Объекты классов TableDef и QueryDef открываются методом OpenRecordset и закрываются с удалением из семейства Databases методом Close объекта Database:

- базаДанных.OpenRecordset (Источник [,Тип, Параметры]);
- объект.OpenRecordset (Источник [,Тип, Параметры]);
- базаДанных.Close;
- объект.Close.

В отличие от режима работы через интерфейс пользователя, процедуры VBA дают возможность одновременно открывать несколько баз данных, хотя на экране отображается только одна из них. Свойство Name позволяет проверить, открылась ли база данных: полный путь к открытой базе возвращает выражение объект.Name. Поля таблиц вначале создаются, затем добавляются в семейства, после чего обновляется окно базы данных:

Set объектПоле = объектТаблица.CreateField (имяПоля, Тип, [Размер]) объектТаблица.Fields.Append объектПоле базаДанных.TableDefs.Append объектТаблица RefreshDatabaseWindow

Доступ к полям реализуется через объекты класса Fields:

- имяТаблицыИлиЗапроса.Fields!имяПоля;
- имяТаблицыИлиЗапроса.имяПоля;
- имяТаблицыИлиЗапроса!Fields!имяПоля;
- имяТаблицыИлиЗапроса! имяПоля;
- Parent!имяПоляГлавнойФормы.

Ни TableDef, ни QueryDef не содержат данных. Для доступа к данным нужны наборы записей объекты RecordSet. В модели DAO присутствуют четыре типа таких объектов. Тип Table представляет набор записей одной таблицы открытого файла базы данных. Он не обрабатывает связанные таблицы и таблицы ODBC и обслуживает только рабочие области Jet. Тип Dynaset представляет динамический набор записей таблицы открытой базы данных, связанной таблицы, результата выполнения запроса или оператора SQL SELECT. Он состоит из ссылок, поэтому обрабатывается медленнее, чем Table, и иногда не обновляется, но охватывает более широкую область данных. Тип Snapshot представляет статическую копию таблицы, запроса или оператора SQL SELECT, удобную для выборки данных и создания отчетов. Forward-Only представляет аналогичную копию, предназначенную для единовременного просмотра данных. Синтаксис:

базаДанных.OpenRecordSet (Источник [,Тип, Параметры, Блокировка])

Источник — это строка с именем таблицы, запроса или текстом SQL, далее следует тип объекта RecordSet, по умолчанию Table для таблиц и Dynaset для за-

просов и связанных таблиц. Любой объект RecordSet существует только в рамках своей процедуры, а затем уничтожается. Его можно закрыть раньше методом Close.

При создании объекта RecordSet строки данных помещаются в буфер и не выводятся на экран, а указатель позиционирует на текущей записи. При открытии набора записей активной становится первая запись. Для перемещения к другим записям используются методы MoveFirst, MoveNext, MovePrevious, MoveLast. Методом BookMark можно определять закладки и возвращаться впоследствии к запомненным в них записям:

объектЗакладка = объектНабор.BookMark

объектНабор.BookMark = объектЗакладка

Метод *Move числоСтрок [,Закладка]* смещает указатель на требуемое число записей вперед или назад. Свойства BOF и EOF объекта Recordset фиксируют выход за пределы набора записей. Для поиска определенной записи в наборах типа Table используется метод Seek, а в наборах других типов — методы FindFirst, FindNext, FindPrevious, FindLast. Найденная запись становится текущей, а свойство NoMatch устанавливается в False. При отсутствии искомой записи NoMatch = True:

- объект.Seek «оператор», списокКлючей;
- объект.Find «критерий».

Здесь используются операторы >, <, >=, <=, =, а список ключей описывает поля текущего индекса. Критерий же предствляет логичекое выражение вида «Поле оператор Значение». В модели DAO изменения в таблицы вносятся последовательно. Предварительно запись копируется в специальную область памяти буфер копирования методом Edit, а затем методом Update возвращается в объект Recordset. При необходимости буфер копирования очищается методом CancelUpdate:

объект.Move или Find или Seek объект.Edit

объект.имяПоля = Значение объект.Update объект.CancelUpdate

При необходимости Execute выполняет запрос на изменение и Update сохраняет изменения. Новые записи тоже не добавляются прямо, сначала они помещаются в буфер копирования, а затем обновляют набор методом AddNew. В наборах типа Dynaset и Table без индекса новая запись добавляется в конец, а там, где определен индекс, она вставляется в соответствии с индексом. Для добавленной записи автоматически создается закладка в свойстве LastModified. Указатель текущей записи при этом автоматически не перемещается.

Удаляемая запись помещается в буфер удаления и остается текущей. Удаление выполняется без предупреждения и отмены. Уникальный объект DoCmd реализует связь между макропрограммированием и программированием на VBA, дублируя операции интерфейса пользователя Access. Методы DoCmd не возвращают значений. В их аргументах активно используются константы Access, такие как acTable, acQuery, acForm, acReport, True, False. Имена объектов здесь не заключаются в прямоугольные скобки, а при необходимости помещаются в двойные кавычки. В частности, для выполнения запросов SQL вызывается метод RunSQL объекта DoCmd. В программах VBA допускается использование выражений SQL длиной до 32768 символов. Другой метод Clone создает копии наборов записей. При копировании организуется независимый указатель

указатель текущей записи, а закладки копии совпадают с закладками оригинала. Но копия не наследует свойств Index, Filter, Sort. Связь полей таблиц и запросов описывается методом Relation. Он используется для создания новых связей (CreateRelation), для их изменения и просмотра. При создании связи указываются соответствующие поля в первичной и внешней таблицах (запросах). В первой таблице эти поля образуют первичный ключ, однозначно определяющий запись, а во второй таблице — внешний ключ. Каждое поле первичного ключа следует добавить в семейство Fields связи с указанием внешнего ключа в свойстве ForeignName:

Set объектСвязи = базаДанных.CreateRelation ([«Имя», Таблица, внешняяТаблица, Атрибуты]) объектСвязи.Table = «Таблица» объектСвязи.ForeignTable = «внешняяТаблица» Set объектПоле = объектСвязи.CreateField(«Поле») объект Поле.ForeignName = «Поле» объектСвязи.Fields Append объектПоле базаДанных.Relations Append объектСвязи

При этом Access устанавливает параметр ссылочной целостности, но не устанавливает каскадное обновление и удаление записей.

3. Процедурное и структурное программирование в ADO

Синтаксис обращения к объектам ADO: **ADODB.имяОбъекта**

Для доступа к данным в этой модели используются объекты Connection, несущие сведения об источнике данных и его расположении. Для работы с ним открывается множество Recorddet или Command и задается строка подключения ConnectionString. Значения каждого свойства такого объекта можно задавать как аргумент метода Open или отдельно, через свойство Properties. Connection открывает сеанс обмена данными через провайдера OLEDB. Command служит для выполнения запросов. Recordset представляет набор записей, а Fields — семейство его полей. Отдельные параметры SQL-запросов и хранимых процедур образуют семейство Parameter. Ошибки сеанса связи фиксируются объектами семейства Errors. Как и в DAO, свойства объекты Properties делятся на встроенные и динамические. Можно сначала задавать свойства, а потом создавать объект, используя его метод или метод другого объекта. Или наоборот: вначале создать объект, а затем установить значения его свойств. Свойства и методы классов модели ADODB приведены в таблице.

Объекты Recordset создают тремя методами: Open класса Recordset, Exequte класса Command или Exequte класса Connection. Синтаксис метода Open: rst.Open Source — Connection, CursorType, lockType, Options, где Source — это имя таблицы или запроса, инструкция SQL, ссылка на объект Command или имя файла, Connection — ссылка на объект подключения, cursorType тип доступа: adOpenDynamic отображает все изменения, вставки и удаления, вносимые другими пользователями; adOpenKeyset показывает только изменения; adOpenStatic сохраняет исходный набор; adOpenForwardOnly то же, при одностороннем движении курсора; LockType — тип блокировки: adLockReadOnly только для чтения, adLockPessimistic пессимистическая блокировка, adLockOptimistic оптимистическая блокировка, adLockBatchOptimistic с кэшированием изменений в наборе; Options — характеристика Source: adCmdText инструкция

adCmdTable SQL-запрос ко всем записям таблицы, adCmdTableDirect таблица или запрос, adCmdUnknown без указания параметра, adCmdFile файл с набором записей. Объект Command служит для выполнения запросов. В качестве источника данных запроса может задаваться таблица или SQL-инструкция, серверное представление или хранимая процедура. Его свойство CommandType при этом принимает одно из значений: adCmdUnknown неизвестный, по умолчанию, adCmdText запрос или хранимая процедура, adCmdTable таблица для запроса, adCmdTableDirect непосредственно открываемая таблица, adCmdStoredProc хранимая процедура на сервере, adCmdFile файл с набором записей.

Для изменения данных следует сначала сделать запись текущей. В модели ADO изменения данных программируются проще, чем в DAO, так как используется обычный оператор присваивания с последующим обновлением. В модели ADO используется отличный от DAO синтаксис поиска:

- объектRecordset.Seek (значенияКлюча, параметрыПоиска);
- Find (Критерии, Пропуск, направлениеПоиска, Старт),

где аргументы указывают массив переменных искомого значения и константный тип сравнения, число пропускаемых строк и направление.

4. Визуальное программирование в Access

Для работы с формами и отчетами используется библиотека Access с базовым классом Application.

Входящее в Access семейство Forms (Формы) содержит формы и их объекты, принадлежащие классу Controls (Элементы управления). Семейство Reports (Отчеты) включает все отчеты приложения и объекты отчетов того же класса Controls. Семейство Modules (Модули) объединяет все стандартные модули и модули авторских классов, а также модули, связанные с формами и отчетами. Для работы с окнами используется класс Screen. А класс DoCmd позволяет обращаться из модулей к стандартным средствам Access. Схема обращения к членам библиотеки классов Access:

имяКласса! имяОбъекта [. имяЭлементаУправления]. Член

Открытие форм и отчетов выполняется методами OpenForm и OpenReport объекта DoCmd:

DoCmd.OpenForm Объект [, режимВывода] [, Запрос] _
[, условиеОтбора] [, режимДанных] [, режимОкна] _
[, аргументыОткрытия]
DoCmd.OpenReport Объект [, режимВывода] [, Запрос] _
[, условиеОтбора] [, режимДанных] [, режимОкна] _
[, аргументыОткрытия]

Обращение к объектам форм и отчетов выполняется по схемам:

- [Forms! Форма!] Объект;
- [Reports! Отчет!] Объект;
- Forms («имяОбъекта»);
- Reports («имяОбъекта»);

- Forms (индексОбъекта);
- Reports (индексОбъекта);
- Forms (ссылочнаяПеременная);
- Reports (ссылочнаяПеременная).

Точка отделяет объект от его свойства или метода, а также разделяет объекты подчиненной формы (отчета):

- [Forms! Форма!] [Объект].[свойствоИлиМетод];
- [Reports! Отчет!] [Объект].[свойствоИлиМетод];
- [Forms!ГлавнаяФорма!]ПодчиненнаяФорма.[Form!] Объект;
- [Forms!ГлавнаяФорма!]ПодчиненнаяФорма.eeИсточникДанных.

Обращение к активным объектам класса Screen в ряде случаев заменяют их свойствами ActiveDataSheet, ActiveForm, ActiveControl, PreviousControl, Application, Parent. А для ссылки на активную форму или отчет из модулей их классов удобно использовать конструкции:

- Me.Член;
- Me!Объект.

Закрытие объектов выполняется методом Close класса Access и объекта DoCmd, а выход из Access методом Quit:

DoCmd.Close [типОбъекта, имяОбъекта], [параметрСохранения]

- Application.Quit [параметрСохранения];
- DoCmd.Quit [параметрСохранения],

где типОбъекта и параметрСохранения выражаются встроенными константами.

Некоторые методы объектов Form:

Метод	Действие
GoToPage	Передает фокус первому элементу управления активной формы
Recalc	Обновляет вычисляемые элементы, но не функции SQL
Refresh	Обновляет записи без учета добавлений и удалений
Repaint	Обновляет экран и завершает вычисления
Requery	Обновляет источник данных формы (учет добавлений и удалений).
	Эквивалентно повторному открытию формы
SetFocus	Передает фокус элементу, ранее последним им владевшему
Undo	Отменяет изменения в форме

Метод Действие GoToPage Передает фокус первому элементу управления активной формы Recalc Обновляет вычисляемые элементы, но не функции SQL Refresh Обновляет записи без учета добавлений и удалений Repaint Обновляет экран и завершает вычисления Requery Обновляет источник данных формы SetFocus Передает фокус элементу, ранее последним им владевшему Undo Отменяет изменения в форме.

Обновление объектов, активных таблиц и запросов выполняется методом Requery:

- имяОбъекта.Requery;
- DoCmd.Requery.

Событие *До* обновления (BeforeUpdate) возникает в момент перемещения с записи на запись. Если пользователь изменяет запись, свойство формы Dirty становится истинным, и если после этого аргументу Cancel присвоить значение True, событие отменяется. Метод Undo восстанавливает исходные данные. Событие *После* обновления (AfterUpdate) используется для выполнения определенных действий в зависимости от введенных в поле значений. При передаче фокуса от одного элемента управления к другому возникают события Вход (Enter), соответствующее приему фокуса, и Выход (Exit), соответствующее его потере. Для выявления идентичных объектов можно использовать оператор Is. Он определяет логический результат как Null или Not Null. Для ссылок из формы или отчета на объект RecordSet используется свойство RecordSetClone. Значение этого свойства формы определяет копию записей базовой таблицы или запроса, указанных в свойстве Источник записей формы. В частности, если форма основана на запросе, то обращение к свойству RecordSetClone эквивалентно созданию копии объекта RecordSet с помощью того же запроса.

Контрольные вопросы

1. Какими уровнями компонентов представлен Access?
2. Опишите иерархию модели Application.
3. Как организуется доступ к данным библиотек DAO?
4. Напишите синтаксис обращения к объектам ADO.
5. Как реализуется визуальное программирование в Access?

СПИСОК ЛИТЕРАТУРЫ

Основные источники:

1. *Заика, А. А.* VBA в MS Office 2007. [Электронный ресурс] : учебный курс. — М. : Национальный открытый ун-т «Интуит», 2016. — 348 с.
2. *Колисниченко, Д. Н.* PHP и MySQL. Разработка веб-приложений. [Электронный ресурс] : учеб. курс. — СПб. : БХВ-Петербург, 2015. — 592 с.
3. *Роббинс, Д.* HTML5, CSS3 и JavaScript. Исчерпывающее руководство [Электронный ресурс] : учеб. руководство. — М. : Эксмо, 2014. — 528 с.

Дополнительные источники:

1. *Водовозов, В. М.* Управление базами данных Access на VBA [Электронный ресурс] : учебник. — СПб. : Питер, 2003. — 32 с.
2. *Гаевский, А. Ю.* Создание веб-страниц и веб-сайтов. HTML и JavaScript. Самоучитель. [Электронный ресурс] : учеб. пособие / А. Ю. Гаевский, В. А. Романовский. — М. : Технолоджи-3000, 2015. — 464 с.
3. *Глаголев, В. А.* Разработка технической документации: Руководство для технических писателей и локализаторов ПО. [Электронный ресурс] : учеб. курс. — СПб. : Питер, 2008. —192 с.
4. *Осетрова, И. С.* Microsoft Visual Basic for Application [Электронный ресурс] : учеб. пособие / И. С. Осетрова, Н. А. Осипов. — СПб. : НИУ ИТМО, 2013. — 120 с.

Интернет-ресурсы:

ИР.1 Курс лекций по VBA [Электронный ресурс]. — Режим доступа : http://mini-soft.ru/soft/vba/index.php. — Загл. с экрана.

ИР.2 «Программирование» Perl, PHP, JavaScript, HTML, XML, DHTML, CSS, C++, Pascal, Delphi, MySQL и др. [Электронный ресурс]. — Режим доступа : http://www.program.rin.ru. — Загл. с экрана

Светлана Викторовна БЕЛУГИНА
РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ
ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ
Учебное пособие

Зав. редакцией
литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *С. В. Макаров*
Корректор *Т. А. Кошелева*
Выпускающий *Н. А. Крылова*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр. Юрия Гагарина, д.1, лит. А.
Тел.: (812) 336-25-09, 412-92-72.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 19.11.19.
Бумага офсетная. Гарнитура Школьная. Формат 70×100 $^{1}/_{16}$.
Печать офсетная. Усл. п. л. 25,35. Тираж 100 экз.

Заказ № 051-20.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские технологии»
109316, г. Москва, Волгоградский пр., д. 42, к. 5.