МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНО-СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

Самаркандский государственный университет

Учебно-методический комплекс по предмету

“Создание интернет приложения ”

для студентов 3-курса по специальности

“Прикланая математика и информатика”

Бустанов Х.А.

Самарканд-2019

Министерство высшего и среднего специального образования республики Узбекистан
Самаркандский государственный университет

Факультет прикладной математики и информатики

Кафедра «Информационных технологий»

Утверждаю:

Проректор по учебной части

_____ проф. А. Солеев

«___» _____ 2019 г.

Учебно-методический комплекс по предмету “Создание интернет приложения”

для студентов 3-курса по специальности
“Прикланая математика и информатика”
Бустанов Х.А.

Область знаний:	100000 – Гуманитарная сфера
Область образования:	130000 – Математика
Направление образования:	5130200 – Прикладная математика и информатика

Самарканд - 2019

Учебно-методический комплекс предмета "Создание интернет приложений" разработана согласно с учебно-рабочей программой.

Составили :
Доцент кафедры «Информационных технологий» СамГУ доц. Ф.Ш. Намозов и старший преподователь Х.А.Бустанов.

Рецезденты:

Доцент кафедры «Информатика» Самаркандского филиала УТИТ А.Каршиев.
Доцент кафедры «Информационных технологий» СамГУ Абдуллаев А.

Учебно-методический комплекс предмета обсуждена и утверждена на собрании кафедры *«Информационных технологий» « ___ » августа 2019 года* ***№1.***

Заведующей кафедрой: проф. И. И. Жуманов

Учебно-методический комплекс предмета обсуждена и утверждена на учебно-метедическом Совете факультета *прикладной математики и информатики « ___ »_____ 2019 года* ***№1.***

Председатель методического совета факультета Ш.Маматов

Передседатель Совета факультета: доц. А. Бобоеров

Согласовал:
Начальник учебно-методического
управлений : Аликулов Б.

СОДЕРЖАНИЕ

1. Основы компьютерных сетей и телекоммуникационных систем а также области их приминения.....6
2. Технические и программное обеспечение компьютерныхсетей (Провайдеры и браузеры Интернет) ……..……..
167
3. Использование и назначение сайтов , парталов Интернент а также их типы и т.д.28
4. Основы web-программирования, Структура HTML-документа. Создание и форматирование текста в HTML-документе..... 80
5. Создание гипер показателей, списки, таблицы, фреймы, формы и др. 88
6. Синтаксис и применение CSS в HTML – документе. Свойства шрифты, тексты, цвет и т.д..... 88
7. Введение в PHP. История PHP. 97
8. Основы синтаксиса PHP 100
9.Управляющие конструкции PHP..... 111
10. Обработка запросов с помощью PHP..... 102
11. Функции в PHP..... 143
12. Объекты и классы в PHP..... 154
13. Работа с массивами данных 160
14. Работа со строками. Работа с файловой системой.. 174
15. Взаимодействие PHP и XML. Использование шаблонов в PHP. 186
16. Литература……199

Календарный тематический план лекционных занятий по предмету «Создание приложение в среде Интернет»

№	Название темы лекции	Кол. часов
1	2	3
1	Основы компьютерных сетей и телекоммуникационных систем а также области их приминения.	2
2	Технические и программное обеспечение КС (Провайдеры и браузеры Интернет).	2
3	Использование и назначение сайтов , парталов Интернент а также их типы и т.д.	2
4	Основы web-программирования, Структура HTML-документа. Создание и форматирование текста в HTML-документе.	2
5	Создание гипер показателей, списки, таблицы, фреймы, формы и др.	2
6	Синтаксис и применение CSS в HTML – документе. Свойства шрифты, тексты, цвет и т.д.	2
7	Введение в PHP. История PHP	2
8	Основы синтаксиса PHP	2
9	Управляющие конструкции PHP	2
10	Обработка запросов с помощью PHP	2
11	Функции в PHP	2
12	Объекты и классы в PHP	2
13	Работа с массивами данных	2
14	Работа со строками. Работа с файловой системой.	2
15	Взаимодействие PHP и XML. Использование шаблонов в PHP.	2
ВСЕГО:		**30**

Тема1. ОСНОВЫ КОМПЬЮТЕРНЫХ СЕТЕЙ И ИХ ТИПЫ, А ТАКЖЕ ОБЕСПЕЧЕНИЕ

КОМПЬЮТЕРНЫЕ СЕТИ.

Компьютерная сеть - объединение нескольких ЭВМ для совместного решения информационных, вычислительных, учебных и других задач.

Все компьютерные сети без исключения имеют одно назначение-обеспечение совместного доступа к общим ресурсам. Слово ***ресурс*** очень удобное. Ресурсы бывают трех видов: ***аппаратные, программные, информационные.***

Аппаратные ресурсы – это, когда все участники компьютерной сети пользуются одним аппаратом, на котором хранят свои архивы и результаты работы.

Компьютерные сети позволяют совместно использовать ***программные ресурсы***.

Данные, хранящиеся на удаленных компьютерах, образуют ***информационный ресурс***, например, Интернет.

По способу организации сети подразделяются на ***реальные и искусственные***.

Искусственные сети позволяют связывать компьютеры вместе через ***последовательные или параллельные порты*** и ***не нуждаются в дополнительных устройствах***.

Реальные сети позволяют связывать компьютеры с помощью ***специальных устройств коммутации и физической среда передачи данных***.

По территориальной распространенности сети могут быть ***локальными, глобальными, региональными и городскими.***

По скорости передачи информации компьютерные сети делятся на низко-, средне- и высокоскоростные.

низкоскоростные (до 10 Мбит/с),

среднескоростные (до 100 Мбит/с),

высокоскоростные (свыше 100 Мбит/с);

Городская сеть (MAN - Metropolitan Area Net Work) - сеть, которая обслуживает информационные потребности большого города.

Региональные - расположенные на территории города или области.

Глобальные сети - WAN (World-Wide Area Net Work) объединяют сотни, тысячи узлов во многих странах мира.

2. ***ЛОКАЛЬНЫЕ КОМПЬЮТЕРНЫЕ СЕТИ.***

Локальная компьютерная сеть - это совокупность компьютеров, соединенных линиями связи, обеспечивающая пользователям сети потенциальную возможность совместного использования ресурсов всех компьютеров.

Конфигурация локальной сети называется топологией.

1. Наиболее простой вид топологии — ***шина***. В такой сети все компьютеры подключены к одному кабелю.

2. На шину похожа и структура, которую называют ***кольцо.***

3. Для локальных сетей, основанных на файловом сервере, может применяться схема ***звезда***.

4. От схемы зависит состав оборудования и программного обеспечения. Топологию выбирают, исходя из потребностей предприятия.

Характерная особенность локальных сетей - наличие связывающего всех абонентов высокоскоростного канала связи для передачи информации в цифровом виде. Существуют ***проводные и беспроводные каналы***. Каждый из них характеризуется определенными значениями существенных с точки зрения организации локальных сетей ***параметров:***

- скорости передачи данных;

- максимальной длины линии;

- помехозащищенности;

- механической прочности;

- удобства и простоты монтажа;

- стоимости.

Существуют ***проводные и беспроводные*** каналы. В настоящее время обычно применяют четыре типа ***сетевых кабелей:***

- коаксиальный кабель;

- незащищенная витая пара;

- защищенная витая пара;

- волоконно-оптический кабель.

Первые три типа кабелей передают электрический сигнал по ***медным*** проводникам. Волоконно-оптические кабели передают свет по ***стеклянному волокну***.

Беспроводная связь на радиоволнах СВЧ диапазона может использоваться для организации сетей в пределах больших помещений типа ангаров или павильонов, там, где использование обычных линий связи затруднено или нецелесообразно.

Протоколы – это наборы ***правил и процедур***, регулирующих порядок осуществления некоторой связи.

Протоколы – это правила и технические процедуры, позволяющие нескольким компьютерам при объединении в сеть общаться друг с другом.

Существует множество протоколов. Протоколы работают на разных уровнях модели взаимодействия открытых систем OSI/ISO. Среди множества протоколов наиболее распространены следующие:

- NetBEUI;
- XNS;
- IPX/SPX и NWLmk;
- Набор протоколов OSI.
- И другие

3. ГЛОБАЛЬНЫЕ СЕТИ.

Глобальная сеть (ГВС или WAN - World Area NetWork) - сеть, соединяющая компьютеры, удалённые географически на большие расстояния друг от друга. Отличается от локальной сети более протяженными коммуникациями (***спутниковыми, кабельными*** и др.). Глобальная сеть объединяет локальные сети.

Internet - глобальная компьютерная сеть, охватывающая весь мир. ***Internet*** образует как бы ядро, обеспечивающее связь различных информационных сетей, принадлежащих различным учреждениям во всем мире, одна с другой. Как и во всякой другой сети в Internet существует *7* ***уровней*** взаимодействия между компьютерами: ***физический, логический, сетевой, транспортный, уровень сеансов связи, представительский и прикладной уровень.***

Протоколы физического уровня определяют вид и характеристики линий связи между компьютерами.

Для каждого типа линий связи разработан соответствующий ***протокол логического уровня***, занимающийся управлением передачей информации по каналу. К протоколам логического уровня для телефонных линий относятся протоколы ***SLIP*** (Serial Line Interface Protocol) и ***PPP*** (Point to Point Protocol).

Протоколы сетевого уровня отвечают за передачу данных между устройствами в разных сетях, то есть занимаются маршрутизацией пакетов в сети. К протоколам сетевого уровня принадлежат ***IP*** (Internet Protocol) и ***ARP*** (Address Resolution Protocol).

Протоколы транспортного уровня управляют передачей данных из одной программы в другую. К протоколам транспортного уровня принадлежат ***TCP*** (Transmission Control Protocol) и ***UDP*** (User Datagram Protocol).

Протоколы уровня сеансов связи отвечают за установку, поддержание и уничтожение соответствующих каналов. В Internet этим занимаются уже упомянутые ***TCP и UDP*** протоколы, а также протокол ***UUCP*** (Unix to Unix Copy Protocol).

Протоколы представительского уровня занимаются обслуживанием прикладных программ. К таким программам относятся: ***telnet-сервер***, ***FTP-***сервер, ***Gopher***-сервер, ***NFS***-сервер, ***NNTP*** (Net News Transfer Protocol), ***SMTP*** (Simple Mail Transfer Protocol), ***POP2 и POP3*** (Post Office Protocol) и т.д.

К протоколам прикладного уровня относятся сетевые услуги и программы их предоставления.

ОСНОВНЫЕ ПОНЯТИЯ

Сеть ЭВМ - комплекс ***аппаратного и программного обеспечения***, поддерживающий функции обмена информацией между отдельно расположенными (на расстояниях от нескольких метров до тысяч километров) компьютерами.

Соответственно ***программное обеспечение компьютерных сетей*** - комплекс программ, поддерживающий функции обмена информацией между отдельно расположенными ЭВМ. В настоящее время программное обеспечение компьютерных сетей обычно является составной часть операционных систем.

Локальная вычислительная сеть (***ЛВС***) - система связи отдельно расположенных ЭВМ на относительно небольшом расстоянии, физическая линия связи - двухпроводной кабель или коаксиальный кабель [3].

Корпоративная вычислительная сеть - сеть, работающая по протоколу ***TCP/IP*** и не обязательно подключенная к Internet, но использующая коммуникационные стандарты Internet'a и сервисные приложения, обеспечивающие доставку данных пользователям сети; эксплуатируется в пределах (крупной) организации.

Глобальная вычислительная сеть объединяет множество локальных сетей и сотни тысяч - миллионы разнотипных ЭВМ по всему миру, физическая линия связи - оптокабель или космическая радиолиния связи.

Рабочая группа (workgroup) - набор компьютеров, объединенных для удобства при просмотре сетевых ресурсов одним именем.

Домен (domain) - определенная администратором сети совокупность компьютеров, использующих общую базу данных и систему защиты; каждый домен имеет уникальное имя.

Узел (host) - подключенное к сети устройство (обычно компьютер), идентифицируемое собственным адресом.

Скорость передачи данных по компьютерной сети измеряется в битах в секунду (bps - bit per second) или бодах (boud).

Трафик (traffic) - поток сообщений в разделяемой среде передачи данных, часто используется для грубой оценки уровня использования передающей среды (тяжелый, средний, легкий трафик).

Серверная ЭВМ - компьютер (обычно обладающий высоким быстродействием и значительным объемом оперативной и дисковой памяти) и выполняющий запросы, поступающие с клиентских ЭВМ.

Клиентская ЭВМ - пользовательский компьютер (обычно обладающий ограниченными ресурсами), выдающий запросы для исполнения серверу.

Файл-сервер - выделенная ЭВМ, выполняющая функции хранения данных и программ, используемых пользователями на клиентских ЭВМ.

Клиентское приложение - приложение, обращающееся с целью выполнения отдельных функций к другому приложению-серверу (и обычно инициирующее начало его выполнения и завершение).

Протокол (коммуникационный) - набор правил и соглашений, согласно которому взаимодействуют два (или более) компьютеров.

Топология (topology) сети - физическая конфигурация машин в сети.

Маршрутизация - процесс определения пути доступа к объектам сети.

Пакет (датаграмма) - определенное количество байт, сгруппированное вместе и посылаемое одновременно (практически все сети коммуникаций передают данные небольшими частями - пакетами или датаграммами).

WWW - это распределенная информационная система мультимедиа, основанная на гипертексте. Информация хранится на множестве Web-серверов и

пользователь получает к ней доступ при помощи программ просмотра или Web-браузеров. Взаимодействие между клиентом и сервером осуществляется при помощи протоколов ***HTTP*** или ***FTP.***

Мультимедиа - это информация, включающая в себя не только текст, но и двух и трехмерную графику, звук и видео.

Гипертекст - это информация в WWW, представленная в виде Web-страниц, в которых содержатся перекрестные ссылки на другие документы в Internete.

Для воплощения всех этих идей был разработан формат представления документов для Web-браузеров ***HTML*** (Hyper Text Markup Language) - это язык гипертекстовой разметки.

ТЕХНОЛОГИИ СЕТЕВЫХ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ПРИЛОЖЕНИЙ

«Файл-серверная» и «клиент-серверная» архитектуры

Сервер - логический процесс, который обеспечивает некоторый сервис по запросу от клиента.

Клиент - процесс, который запрашивает обслуживание от сервера.

В «***клиент-серверной***» системе программа сервера стартует первой и пассивно ожидает запросов от клиента и при получении обрабатывает их при помощи различных механизмов.

Таким образом, в модели «клиент-сервер» различают запросы и ответы.

Сетевое приложение «***файл-серверной***» архитектуры отличается от предыдущей тем, где происходит обработка данных. Данные в виде одного или нескольких файлов размещаются на файловом сервере. Сервер принимает запросы от ПК в сети и передает им требуемые данные. Основная обработка данных

происходит в процессе клиента. Сервер следит лишь за тем, чтобы не возникало конфликтов при одновременном обращении к файлам.

В «***клиент-серверной***» архитектуре сервер не только обеспечивает пересылку необходимой информации, но и берет на себя часть или всю обработку этих данных.

«***Клиент-серверная***» архитектура позволяет обработки информации путем распределения вычислительной нагрузки между клиентом и сервером.

Основная задача клиентского приложения - это обеспечение интерфейса с пользователем, т.е. ввод данных и предоставление результатов в удобном для пользователя виде и управление сценариями работы приложения.

Основная задача серверного приложения - это обеспечение надежности, согласованности и защищенности данных, управление запросами клиентов, быстрая обработка запросов и предоставление ответов.

Сетевые операционные системы

Сетевые операционные системы (Network Operating System -NOS) - это комплекс программ, обеспечивающих обработку, хранение и передачу данных в сети.

NOS определяет взаимосвязанную группу протоколов верхних уровней, обеспечивающих выполнение ***основных функций*** сети. К ним, в первую очередь, относятся:

адресация объектов сети;

функционирование сетевых служб;

обеспечение безопасности данных;

управление сетью.

Структура сетевой операционной системы

Коммуникационные средства ОС, с помощью которых происходит обмен сообщениями в сети.

Клиентское программное обеспечение-это программное обеспечение обеспечивает доступ к ресурсам, расположенным на сетевом сервере.

Редиректори. Редиректор - сетевое программное обеспечение, которое принимает запросы ввода/вывода для удаленных файлов, именованных каналов или почтовых слотов и затем переназначает их сетевым сервисам другого компьютера.

Редиректор перехватывает все запросы, поступающие от приложений, и анализирует их.

Фактически существуют ***два типа редиректоров***, используемых в сети:

клиентский редиректор (client redirector)

серверный редиректор (server redirector).

Распределители

Распределитель (designator) представляет собой часть программного обеспечения, управляющую присвоением букв накопителя (drive letter) как локальным, так и удаленным сетевым ресурсам или разделяемым дисководам, что помогает во взаимодействии с сетевыми ресурсами.

Серверное программное обеспечение - часть сетевой операционной системы, которая позволяет поддерживать ресурсы и распространять их среди сетевых клиентов.

Клиентское и серверное программное обеспечение.

Это позволяет компьютерам поддерживать и использовать сетевые ресурсы и преобладает в одноранговых сетях.

Выбор сетевой операционной системы

При выборе сетевой операционной системы необходимо учитывать:

совместимость оборудования;

тип сетевого носителя;

размер сети;

сетевую топологию;

требования к серверу;

операционные системы на клиентах и серверах;

сетевая файловая система;

соглашения об именах в сети;

организация сетевых устройств хранения.

Сетевые операционные системы имеют разные свойства в зависимости от того, предназначены они для сетей масштаба рабочей группы (отдела), для сетей масштаба кампуса или для сетей масштаба предприятия.

Сети отделов используются небольшой группой сотрудников, решающих общие задачи. Главной целью сети отдела является разделение локальных ресурсов, таких как приложения, данные, лазерные принтеры и модемы. Сети отделов обычно не разделяются на подсети.

Сети кампусов соединяют несколько сетей отделов внутри отдельного здания или одной территории предприятия.

Сети предприятия (корпоративные сети) объединяют все компьютеры всех территорий отдельного предприятия. Они могут покрывать город, регион или даже континент.

ПО ДЛЯ РАБОТЫ В ИНТЕРНЕТ-БРАУЗЕРЫ

Начиная с первой половины 1990 годов для работы с Интернет, и, прежде всего, со Всемирной Паутиной, создаются специальные ***программы-браузеры*** (от английского "browse" - просмотр). В российской практике название "браузер" закрепилось достаточно прочно, хотя в некоторых случаях употребляется и прямой перевод на русский язык - "программа-просмотрщик" или "обозреватель".

Все браузеры имеют свои достоинства и недостатки, которые складываются из скорости работы, способности открывать те или иные приложения, соответствия требованиям безопасности и множества других параметров. Конкуренция заставляет производителей постоянно совершенствовать свои продукты, добиваясь повышения потребительских свойств. Лидеры в постоянной гонке браузеров периодически сменяют друг друга и подчас безоговорочные законодатели мод постепенно переходят в разряд аутсайдеров.

Так, например, Netscape Navigator, единолично лидировавший в 1994 - 1997 годах, в начале 2000 годов практически полностью уступил свои позиции продукту Microsoft, который, в свою очередь, начинает испытывать давление со стороны Opera и Mozilla.

Освоение работы с браузерами - сугубо прикладная задача и может быть реализована непосредственно на месте, с учетом выбранного программного решения. Существует огромное число источников, помогающих освоить все типы браузеров.

К числу наиболее распространенных ***требований*** относятся, обязательная ***установка антивирусной*** программы, контролирующей содержание входящих писем и загружаемых web-страниц.

Доменами первого уровня. В их число вошли: .biz, .info, .pro, .aero, .coop, .museum, .name. Распределение этих имен было произведено следующим образом:

.biz - коммерческие компании и проекты;

.info - учреждения, для которых информационная деятельность является ведущей (библиотеки, средства массовой информации);

.pro - сайты сертифицированных профессионалов таких областей деятельности как врачи, юристы, бухгалтеры, а также представители других профессий, в которых персональный аспект имеет ключевое значение (pro от слов profession, professional);

.aero - компании и персоны, непосредственно связанные с авиацией;

.coop - корпорации, использующие совместный капитал (от слова cooperative);

.museum - только музеи, архивы, выставки;

.name - персональные сайты, состоящие, как правило, из двух частей: имени и фамилии: www.bruce.edmonds.name.

НАЗНАЧЕНИЕ СЛУЖБЫ ДОМЕННЫХ ИМЕН DNS

Из всех многочисленных сетевых служб, работающих на прикладном уровне и обрабатывающие сетевые запросы клиентов, существует одна без которой и всем остальным работать очень тяжело, если вообще возможно. Эта служба называется DNS (Domain Name Service - служба доменных имен).

Организация системы доменных имен.

Мировая система доменных имен организована следующим образом: существует несколько корневых серверов, которые ни в коем случае не знают все имена всех компьютеров в сети (и, более того, не обязаны этого знать). Вместо этого они содержат информацию о так называемых зонах DNS, состоящих из одного имени. Например, существуют зоны ru, com, info, de и так далее. Во-первых, они знают о том, какие имена первого уровня существуют в принципе, поскольку их относительно немного. Недавно их было совсем мало - в старых справочниках по интернету указывались только com, net, org, edu, gov и еще несколько зон, связанных с государствами, по двухбуквенным кодам. В какой-то момент было принято решение сильно расширить диапазон, а недавно приняли решение, что для записи имен зон будет использоваться кодировка юникод, что позволит создать имена зон на национальных алфавитах.

Поэтому для того, чтобы эту процедуру сделать более эффективной, есть три идеи, которые сильно облегчают нагрузку.

1. Каждая таблица с записью соответствия IP-адреса доменному имени содержит в себе информацию о том, в течение какого времени эта таблица, во-первых, действительна, а во-вторых, может не меняться, то есть ее время жизни и задержка на изменение.

2. Чем больше в распределенной системе узлов, тем ниже совокупная надёжность самой системы.

3. Некоторое ограничение свободы обычного пользователя.

Литература

1. Якубайтис Э.А. Информационные сети и системы: Справочная книга. - М.: Финансы и статистика, 1996.

2. Бэрри Нанс. Компьютерные сети пер. с англ. - М.: БИНОМ, 1996.

3. Основы современных компьютерных технологий под редакцией А.Д. Хомоненко- СПб КОРОНА принт, 1998.

Тема 2. ТЕХНИЧЕСКОЕИ ПРОГРАММНОЕОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХСЕТЕЙ

1.1 Общие сведения о сетях

1.2 Локальные сети

1.3 Топология сетей

1.1 Общие сведения о сетях

Компьютерные сети, называемые также вычислительными сетями, или **сетями передачи данных**, являются логическим результатом эволюции двух важнейших научно-технических отраслей современной цивилизации - компьютерных и телекоммуникационных технологий.

Компьютерная (вычислительная) сеть - совокупность компьютеров и терминалов, соединённых с помощью каналов связи в единую систему, удовлетворяющую требованиям распределённой обработки данных.

Основное назначение любой компьютерной сети - предоставление информационных и вычислительных ресурсов подключенным к ней пользователям [6].

В зависимости от территориального расположения абонентских систем вычислительные сети можно разделить на три основных класса:

- глобальные сети (WAN -- Wide Area Network);

- региональные сети (MAN -- Metropolitan Area Network);

- локальные сети (LAN -- Local Area Network).

Глобальная вычислительная сеть объединяет абонентов, расположенных в различных странах, на различных континентах.

Региональная вычислительная сеть связывает абонентов, расположенных на значительном расстоянии друг от друга.

Локальная вычислительная сеть объединяет абонентов, расположенных в пределах небольшой территории.

1.2 Локальные сети

Локальные сети (Local Area Networks, LAN) -- это объединение компьютеров, сосредоточенных на небольшой территории, обычно в радиусе не более 2-2,5 км, хотя в отдельных случаях локальная сеть может иметь и более протяженные размеры, например в несколько десятков километров.

Основной назначение любой компьютерной сети - предоставление информационных и вычислительных ресурсов подключенным к ней пользователям.

Сервер - компьютер, подключенный к сети и обеспечивающий ее пользователей определенными услугами.

Рабочая станция - персональный компьютер, подключенный к сети, через который пользователь получает доступ к ее ресурсам.

Клиент - задача, рабочая станция или пользователь компьютерной сети.

Одноранговая сеть, в которой нет единого центра управления взаимодействием рабочих станций и нет единого центра для хранения данных.

Достоинства одноранговых сетей:

- низкая стоимость;

- высокая надежность.

Недостатки одноранговых сетей:

- зависимость эффективности работы сети от количества станций;

- сложность управления сетью;

- сложность обеспечения защиты информации;

- трудности обновления и изменения программного обеспечения станций.

В **сети с выделенным сервером** один из компьютеров выполняет функции хранения данных, предназначенных для использования всеми рабочими станциями, управления взаимодействием между рабочими станциями и ряд сервисных функций т.е. сервером сети.

Достоинства сети с выделенным сервером:

- надежна система защиты информации;

- высокое быстродействие;

- отсутствие ограничений на число рабочих станций;

- простота управления по сравнению с одноранговыми сетями.

Недостатки сети:

- высокая стоимость из-за выделения одного компьютера на сервер;

- зависимость быстродействия и надежности от сервера;

- меньшая гибкость по сравнению с одноранговыми сетями.

1.3 Топология сетей

Топология типа «звезда»

Концепция топологии сети в виде звезды пришла из области больших ЭВМ, в которой головная машина получает и обрабатывает все данные с периферийных устройств как активный узел обработки данных. Вся информация между двумя периферийными рабочими местами проходит через центральный узел вычислительной сети.

Кольцевая топология

При кольцевой топологии сети рабочие станции связаны одна с другой по кругу, т.е. рабочая станция 1 с рабочей станцией 2, рабочая станция 3 с рабочей станцией 4 и т.д. Последняя рабочая станция связана с первой. Коммуникационная связь замыкается в кольцо.

Прокладка кабелей от одной рабочей станции до другой может быть довольно сложной и дорогостоящей, особенно если географическое расположение рабочих станций далеко от формы кольца (например, в линию).

Шинная топология

При шинной топологии среда передачи информации представляется в форме коммуникационного пути, доступного дня всех рабочих станций, к которому они все должны быть подключены. Все рабочие станции могут непосредственно вступать в контакт с любой рабочей станцией, имеющейся в сети.

Рабочие станции в любое время, без прерывания работы всей вычислительной сети, могут быть подключены к ней или отключены. Функционирование вычислительной сети не зависит от состояния отдельной рабочей станции.

Отключение и особенно подключение к такой сети требуют разрыва шины, что вызывает нарушение циркулирующего потока информации и зависание системы.

ТЕМА 3. ИСПОЛЬЗОВАНИЕ И НАЗНАЧЕНИЕ САЙТОВ , ПАРТАЛОВ ИНТЕРНЕНТ А ТАКЖЕ ИХ ТИПЫ И Т.Д.

3.1 Программное обеспечение информационно-вычислительных сетей

3.2 Основные протоколы обмена в компьютерных сетях

3.3 Установка и настройка протоколов сети

3.4 Аутентификация и авторизация. Система Kerberos

3.1 Программное обеспечение информационно-вычислительных сетей

Сетевое программное обеспечение состоит из трех частей:

- общего программного обеспечения;
- системного программного обеспечения;
- специального программного обеспечения.

Общее программное обеспечение образуется из компонентов базового программного обеспечения отдельных компьютеров, входящих в состав сети, и включает в себя *операционные системы*, системы автоматизации *программирования* и системы *технического обслуживания.*

Системное программное обеспечение представляет собой комплекс программных средств, поддерживающих и координирующих взаимодействие всех ресурсов сети как единой системы.

Специальное программное обеспечение предназначено для максимального удовлетворения пользователей программами часто решаемых задач и, соответственно, *содержит прикладные программы пользователя, ориентированные на специфику его предметной области.*

Сетевая операционная система (СОС) включает в себя набор управляющих и обслуживающих программ, обеспечивающих:

- координацию работы всех звеньев и элементов сети;

- оперативное распределение ресурсов по элементам сети;

- потоков заданий между узлами вычислительной сети;

- установление последовательности решения задач и обеспечение их общесетевыми ресурсами;

- контроль работоспособности элементов сети и обеспечение достоверности входной и выходной информации;

- защиту данных и вычислительных ресурсов от несанкционированного доступа;

- выдачу справок об использовании информационных, программных и технических ресурсов сети [2].

3.2 Основные протоколы обмена в компьютерных сетях

Для обеспечения согласованной работы в сетях передачи данных используются различные коммуникационные протоколы передачи данных - наборы правил, которых должны придерживаться передающая и принимающая стороны для согласованного обмена данными.

Протоколы - это наборы правил и процедур, регулирующих порядок осуществления некоторой связи. ***Протоколы*** - это правила и технические процедуры, позволяющие нескольким компьютерам при объединении в сеть общаться друг с другом.

Компьютер-отправитель в соответствии с протоколом выполняет следующие действия: разбивает данные на небольшие блоки, называемыми пакетами, с которыми может работать протокол, добавляет к пакетам адресную информацию, чтобы компьютер-получатель мог определить, что эти данные предназначены именно ему, подготавливает данные к передаче через плату сетевого адаптера и далее - по сетевому кабелю.

Компьютер-получатель в соответствии с протоколом выполняет те же действия, но только в обратном порядке: принимает пакеты данных из сетевого кабеля; через плату сетевого адаптера передает данные в компьютер; удаляет из пакета всю служебную информацию, добавленную компьютером-отправителем, копирует данные из пакета в буфер - для их объединения в исходный блок, передает приложению этот блок данных в формате, который оно использует.

И компьютеру-отправителю, и компьютеру-получателю необходимо выполнить каждое действие одинаковым способом, с тем чтобы пришедшие по сети данные совпадали с отправленными.

Протоколы, которые поддерживают передачу данных между сетями по нескольким маршрутам, называются маршрутизируемыми протоколами.

Среди множества протоколов наиболее распространены следующие:

- NetBEUI;

- XNS;

- IPX/SPX и NWLmk;

- Набор протоколов OSI [1].

3.3 Установка и настройка протоколов сети

NetBEUI - расширенный интерфейс NetBIOS. Первоначально NetBEUI и NetBIOS были тесно связаны и рассматривались как один протокол, затем производители их обособили и сейчас они рассматриваются отдельно.

NetBEUI - небольшой, быстрый и эффективный протокол транспортного уровня, который поставляется со всеми сетевыми продуктами фирмы Microsoft. К преимуществам NetBEUI относятся небольшой размер стека, высокая скорость передачи данных и совместимость со всеми сетями Microsoft. Основной недостаток - он не поддерживает маршрутизацию, это ограничение относится ко всем сетям Microsoft.

3.4 Аутентификация и авторизация. Система Kerberos

Kerberos - это сетевая служба, предназначенная для централизованного решения задач аутентификации и авторизации в крупных сетях.

В сетях, использующих ***систему безопасности Kerberos***, все процедуры аутентификации между клиентами и серверами сети выполняются через посредника, которому доверяют обе стороны аутентификационного процесса, причем таким авторитетным арбитром является сама система Kerberos.

Сайты Интернети их типы

Типы и основные виды сайтов

В сети Интернет существуют десятки, а возможно и сотни миллионов сайтов. Зная это, даже самому неосведомленному в данной сфере человеку, не трудно предположить, что при таком огромном количестве, сайты делятся на ***типы и виды***.

На какие же именно **типы *и* виды** принято подразделять сайты?

Начнем с типов. О них не придется долго говорить. Типов сайтов существует всего **два** – все сайты делятся на **информационные** и **сервисные**.

Видов сайтов гораздо больше. По некоторым оценкам, их насчитывается около сотни. В рамках данной статьи мы не будем перечислять все известные нам виды сайтов, а остановимся только на наиболее популярных и востребованных.

Таковых на наш взгляд тринадцать.

Назовем их и дадим каждому краткое описание.

1. Персональные сайты или персональные страницы

Персональный сайт – это сайт, с содержанием, описывающим сферу интересов какого-либо человека. Обычно он создается самим владельцем с целью заявить о себе, найти друзей, единомышленников, людей со схожими взглядами и т.д.

2. Личные блоги

Личный блог (от англ. blog, web log) – это web-сайт, основное содержимое которого – регулярно добавляемые записи (посты), содержащие текст, изображения, мультимедиа.

3. Сайты-визитки

Сайт-визитка – это наиболее распространенный вид сайтов. Его название говорит само за себя. По сути, сайт-визитка – это электронный аналог традиционной бумажной визитки. Основное назначение сайта – представить своего владельца (реализуемые им товары, предоставляемые услуги) и дать максимум сведений необходимых для контакта с ним – адрес, телефоны, e-mail и т.п.

4. Сайты-галереи

Сайт-галерея – это, прежде всего, интернет-выставка каких-либо собственноручно выполненных работ. Он может быть полезен дизайнеру, художнику, фотографу, мастеру или группе какого-либо прикладного творчества – любому автору стремящемуся заявить о себе, а также, пытающемуся найти клиентов, заказчиков, покупателей на свои произведения.

5. **Официальные сайты компаний** (организаций)

Официальный сайт компании – это представительство компании в глобальной сети, место публикации всех новостей и информации, которую руководство компании хочет донести до общественности.

6. Тематические сайты

Тематический сайт – это web-ресурс посвященный какой-то одной теме.

Обычно, тематический сайт – это достаточно большой виртуальный массив информации, своего рода, специализированный тематический журнал, в котором авторы сайта рассматривают избранную ими тему достаточно подробно и разносторонне.

7. Сайты-каталоги (или сайты-витринаы)

Сайт-каталог – это удобное средство информирования потенциальных покупателей и партнеров компании обо всех предлагаемых ей товарах. В общем виде, структура сайта-каталога может быть следующей:

- данные о компании;
- каталог товаров (или услуг);
- компонент скачивания прайс-листов;
- контактные данные (адрес офиса, телефон, адрес электронной почты).

Дизайн сайта не имеет строго определённых рамок, он может быть и простым и сложным – здесь так же все зависит только от пожеланий заказчика.

8. Сайты Интернет-магазины

Сайт Интернет-магазин – это современный торговый канал, дающий возможность реализовывать товары через Интернет.

9. Промо-сайты

Промо-сайт – это почти всегда неосновной сайт компании – сайт, который создается специально для продвижения какого-либо товара (группы товаров) или услуги.

Промо-сайт – это, прежде всего, ***рекламный*** инструмент. В общем виде, она может включать в себя:

- данные о компании;
- описание продвигаемых товаров (услуги);
- контактные данные: телефон, адрес офиса, адрес электронной почты.

10. Новостные сайты

Новостной сайт – это мощный информационный ресурс посвященный новостям из какой-либо области. Это могут быть как новости политики, науки, культуры, или спорта, так и новинки сферы высоких технологий, моды и даже новые кулинарные рецепты.

11. Корпоративные сайты

Корпоративный сайт – это фирменный коммерческий сайт компаний. Солидное интернет-представительство компании. Оптимальное решение для всех компаний, которые хотят стать лидерами в своей сфере бизнеса.

Хотя у разных компаний структуры их корпоративных сайтов могут сильно отличаться – опыт показывает, что в общем виде, такие сайты включают в себя:

- информацию о компании (историю, новости, вакансии и т.д.);
- деловую и полезную информацию (статьи, обзоры, советы, вопросы-ответы, др.);
- каталоги товаров, услуг, их описания, др.;
- прайс-листы;
- раздел для клиентов, партнеров, посетителей;
- закрытую часть для разных групп сотрудников (персональные разделы, личные кабинеты);
- контактные данные и форму обратной связи.

12. Сайты-порталы

Сайт-портал – это наиболее мощный и самый сложный вид сетевого ресурса, который может быть посвящен как одной теме, так и нескольким.

Сайты порталы очень популярны в сети Интернет, однако изготовление сайта-портала – это достаточно трудоёмкая задача, которая требует много времени и привлечения к работе специалистов разных направлений.

13. Контент-проекты

Контент-проект – это сайт, который представляет собой обширное собрание каких-либо тематических материалов – книг, статей, аудио и видео файлов, др.

Основная задача такого сайта – привлечь посетителей интересующихся определенной тематикой и перенаправить их на сайты партнеров.

Сайт

Сайт, или **веб-сайт** (от англ. *website*: *web* — «паутина, сеть» и *site* — «место», буквально «место, сегмент, часть в сети»), — одна или несколько логически связанных между собой веб-страниц; также место расположения контента сервера.

Обычно сайт в Интернете представляет собой массив связанных данных, имеющий уникальный адрес и воспринимаемый пользователем как единое целое.

Веб-сайты называются так, потому что доступ к ним происходит по протоколу HTTP[1].

Веб-сайт, как система электронных документов (файлов данных и кода) может принадлежать частному лицу или организации и быть доступным вкомпьютерной сети под общим доменным именем и IP-адресом или локально на одном компьютере.

Все сайты в совокупности составляют Всемирную паутину, где коммуникация (паутина) объединяет сегменты информации мирового сообщества в единое целое — базу данных и коммуникации планетарного масштаба. Для прямого доступа клиентов к сайтам на серверах был специально разработан протокол HTTP.

Содержание. История.

Первый[3] в мире сайт **info.cern.ch** появился 6 августа 1991 года. Его создатель, Тим Бернерс-Ли, опубликовал на нём описание новой технологии World Wide Web, основанной на протоколе передачи данных HTTP, системе адресации URI и языке гипертекстовой разметки HTML. Также на сайте были описаны принципы установки и работы серверов и браузеров. Сайт стал и первым в мире интернет-каталогом, так как позже Тим Бернерс-Ли разместил на нём список ссылок на другие сайты.

Все инструменты, необходимые для работы первого сайта, Бернерс-Ли подготовил ещё раньше — в конце 1990 года появились первый гипертекстовый браузер World Wide Web с функционалом веб-редактора, первый сервер на базе NeXTcube и первые веб-страницы.

«Отец» веба считал, что гипертекст может служить основой для сетей обмена данными, и ему удалось претворить свою идею в жизнь. Ещё в 1980 году Тим Бернерс-Ли создал гипертекстовое программное обеспечение Enquire, использующее для хранения данных случайные ассоциации. Затем, работая в Европейском центре ядерных исследований в Женеве (CERN), он предложил коллегам публиковать гипертекстовые документы, связанные между собой гиперссылками. Бернерс-Ли продемонстрировал возможность гипертекстового доступа к внутренним поисковику и документам, а также новостным ресурсам Интернета. В результате, в мае 1991 года в CERN был утверждён стандарт WWW.

Типы интернет-ресурсов.

- Открытые — все сервисы полностью доступны для любых посетителей и пользователей.
- Полуоткрытые — для доступа необходимо зарегистрироваться (обычно бесплатно).
- Закрытые — полностью закрытые служебные сайты организаций (в том числе корпоративные сайты), личные сайты частных лиц. Такие сайты доступны для

узкого круга пользователей. Доступ новым пользователям обычно даётся через так называемые ***инвайты*** (приглашения).

По физическому расположению

- Общедоступные сайты сети Интернет.
- Локальные сайты — доступны только в пределах локальной сети. Это могут быть как корпоративные сайты организаций, так и сайты частных лиц в локальной сети провайдера.

По схеме представления информации, её объёму и категории решаемых задач можно выделить следующие типы веб-ресурсов

- **Интернет-портал** — многокомпонентная разветвлённая структура, скомпонованная из функционально самодостаточных сайтов самостоятельных организаций или подразделений корпоративной структуры.
- **Информационные ресурсы**:
- **Тематический** сайт — сайт, предоставляющий специфическую узкотематическую информацию по какой-либо теме.
- **Тематический портал** — это очень большой веб-ресурс, который предоставляет исчерпывающую информацию по определённой тематике. Порталы похожи на тематические сайты, но дополнительно содержат средства взаимодействия с пользователями и позволяют пользователям общаться в рамках портала (форумы, чаты) — это среда существования пользователя.
- **Интернет-представительства** владельцев бизнеса (торговля и услуги, не всегда связанные напрямую с Интернетом):
- **Сайт-визитка** — содержит самые общие данные о владельце сайта (организация или индивидуальный предприниматель): вид деятельности, история, прейскурант, контактные данные, реквизиты, схема проезда. Специалисты размещают своё резюме (то есть подробная визитная карточка).

- **Представительский сайт** — так иногда называют сайт-визитку с расширенной функциональностью: подробное описание услуг, портфолио, отзывы, форма обратной связи и т. д.
- **Корпоративный сайт** — содержит полную информацию о компании-владельце, услугах/продукции, событиях в жизни компании.
- **Каталог продукции** — в каталоге присутствует подробное описание товаров/услуг, сертификаты, технические и потребительские данные, отзывы экспертов и так далее. На таких сайтах размещается информация о товарах/услугах, которую невозможно поместить в прейскурант.
- **Интернет-магазин** — сайт с каталогом продукции, с помощью которого клиент может заказать нужные ему товары. Используются различные системы расчётов: от пересылки товаров наложенным платежом или автоматической пересылки счета по факсу до расчётов с помощью пластиковых карт.
- **Промосайт** — сайт о конкретной торговой марке или продукте, на таких сайтах размещается исчерпывающая информация о бренде, различных рекламных акциях (конкурсы, викторины, игры и т. п.).
- **Сайт-квест** — интернет-ресурс, на котором организовано соревнование по разгадыванию последовательно взаимосвязанных логических загадок.
- **Веб-сервис** — сайт, созданный для выполнения каких-либо задач или предоставления услуг в рамках сетиWWW:
- **Доска объявлений** представляет собой ресурс, на котором есть возможность размещения публичного объявления о продаже или покупке товаров и услуг, также возможно оставить какую-либо информацию краткого содержания.
- **Каталог сайтов** — это ресурс, на котором размещаются сайты и блоги, например, Open Directory Project. Каталоги бывают платные и бесплатные.

Также каталоги могут способствовать продвижению ресурса, который размещается в каталоге сайтов.

- Поисковые сервисы — например, Yahoo!, Google, Bing, Яндекс.

- Почтовый сервис — например, Mail.Ru и Gmail.
- Веб-форумы
- Блоговый сервис
- Файлообменный пиринговый сервис — например, Bittorrent.
- Облачное хранилище данных — например, OneDrive.
- Сервис редактирования данных — например, Google Docs.
- Фотохостинг — например, Picnik, ImageShack, Panoramio, Photobucket.
- Видеохостинг — например, YouTube, Dailymotion.
- Социальные медиа.
- Комбинированные веб-сервисы (Социальные сети)- например, Facebook, Twitter.
- Комбинированные веб-сервисы (Специализированные социальные сети) — например, MySpace, Flickr.

Безопасность

Наиболее распространённые проявления взлома сайта:

- несанкционированное изменение злоумышленниками отображения сайта (дефейсинг, хакеры)
- подделка сайта (дизайн и содержимое сайта может быть скопировано и у пользователя такого сайта могут украсть пароли)
- снижение числа пользователей сайта из-за воровства пользователей, перешедших на сайт с поисковой системы или мобильных устройств
- появление ссылок на внешние ресурсы (чёрное seo)
- появление порно-баннеров и другой назойливой рекламы

Вторичные последствия взлома сайта:

- блокировка сайта как «вредоносного» поисковыми системами Google и Яндекс
- блокировка сайта браузерами Google Chrome, Opera, Яндекс. Браузер и другими

- блокировка сайта антивирусами
- блокировка сайта хостинг-провайдером, на котором он расположен
- снижение позиций сайта в поисковой выдаче поисковых систем
- снижение количества ежедневных посетителей сайта

Наиболее популярными мотивами взлома сайта являются:

- подорвать продажи или имидж конкурирующего сайта
- получить выгоду: рассылать за деньги спам с сайта; перенаправлять за деньги пользователей сайта на другие сайты и страницы приложения Google Play и AppStore; использовать сайт для DDoS-атак; использовать сайт для размещения на нём ссылок на внешние сайты; размещение вредоносного кода, заражающего компьютеры посетителей сайта
- шантаж: воровство с целью возврата владельцу за деньги
- реклама: размещение на сайте дефейсинга с целью рекламы хакерских услуг
- политические мотивы: с целью показать позицию в отношении того или иного политического строя или организации

По данным, проведённого сервисом по защите сайтов SiteSecure, исследования безопасности коммерческих сайтов в России за 1 квартал 2015 года[8] каждый 10-й сайт заражён или имеет высокий риск заражения и блокировки за вредоносность.

Литература

1. ↑ *Воройский Ф. С.* Информатика. Энциклопедический систематизированный словарь-справочник. — М.: Физматлит, 2006. — С. 432. — 945 с.
2. ↑ Бобкова О., Давыдов С. К вопросу о соотношении понятий «доменное имя» и «название сайта» // Хозяйство и право. — М., 2014, № 6. — С. 102—106.
3. Хаген Граф. Создание веб-сайтов с помощью Joomla! 1.5. — Издательский дом «Вильямс», 2009. — 312 р. — ISBN 978-5-8459-1506-1.
4. *Виктор Ромашев.* CMS Drupal: Система управления содержимым сайта. — Питер, 2010. — 255 р. — ISBN 978-5-49807-241-8.

ТЕМА 4. ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ (4 ЧАСОВ)

План.

1.1. Структура HTML-документа.

1.2. Создание и форматирование текста в HTML-документе.

1.3. Создание гипер показателей, списки, таблицы, фреймы, формы и др.

1.1. Структура HTML-документа.

Структура. Web-документ должен содержать в себе следующие разделы: ***заглавие, название компании, навигационную панель, собственно содержание, контактную информацию, дату и время обновления, авторские права и статус документа.***

Логотип. Создавая Web-страницу, необходимо позаботиться о том, чтобы название фирмы всегда присутствовало на экране.

Навигационная панель. Одним из наиболее важных разделов Web-документа является навигационная панель или панель управления.

Содержание. Прежде всего, следует отметить, что содержание Web-документов должно в полной мере отвечать всем требованиям, предъявляемым к обычным газетным или журнальным публикациям: грамматическая и орфографическая корректность, точность и достоверность предлагаемых материалов и многое другое.

Графика. При разработке Web-страницы нужно очень внимательно выбирать оптимальное соотношение графических и текстовых материалов.

Пропускная способность каналов. Чтобы вашим посетителям не пришлось слишком долго ждать загрузки страниц, провайдер должен обладать надежным высокоскоростным соединением порядка 1-2 Мбит в секунду.

Поддержка сервером провайдера SSI (Server Side Includes, вставки на стороне сервера). Использование SSI позволяет Web-серверу вставлять небольшие объемы динамических данных непосредственно в пересылаемый пользователю *HTML*-документ.

Поддержка сервером провайдера CGI-сценариев. CGI (Common Gateway Interface, общий шлюзовой интерфейс) — спецификация, позволяющая Web-серверу выполнять произвольные прикладные программы. Создать CGI-сценарий можно с помощью популярного языка программирования: Perl, Basic, C, C++, Pascal и т.п.

Поддержка моментальной перекодировки. Чтобы пользователю было легко просматривать страницы, Web-сервер провайдера должен уметь автоматически перекодировать документы в зависимости от поступившего запроса.

Способ обновления страниц. Обычно страницы обновляются по протоколу FTP (File Transfer Protocol, протокол передачи файлов). Некоторые FTP-клиенты позволяют работать с файлами на компьютере провайдера так же, как с собственным диском, — копировать, удалять, переименовывать и т.п.

1.1.1. История развития HTML

В 1989 году Тим Бернерс-Ли предложил руководству международного центра высоких энергий (CERN) проект распределенной гипертекстовой системы, которую он назвал *World Wide Web* (WWW), Всемирная паутина. Глобальной компьютерной сети Internet ассоциируется с тремя основными информационными технологиями:

- электронная почта (e-mail);
- файловые архивы FTP;
- *World Wide Web*.

Язык HTML позволяет размечать электронный документ, который отображается на экране с полиграфическим уровнем оформления; результирующий документ

может содержать самые разнообразные метки, иллюстрации, аудио- и видеофрагменты и так далее.

1.1.2. Принципы гипертекстовой разметки

HTML является описательным *языком разметки* документов, в нем используются указатели разметки (***теги***).

Теговая модель описывает документ как совокупность контейнеров, каждый из которых начинается и заканчивается *тегами*, то есть документ HTML представляет собой не что иное, как обычный ASCII-файл, с добавленными в него управляющими *HTML*-кодами (*тегами*).

Теги HTML-документов в большинстве своем просты и понятны, ибо они образованы с помощью общеупотребительных слов английского языка, понятных сокращений и обозначений.

HTML-*тег* состоит из имени, за которым может следовать необязательный список атрибутов *тега*.

Текст *тега* заключается в угловые скобки ("<" и ">"). Простейший вариант *тега* — имя, заключенное в угловые скобки, например, <HEAD> или <I>. Для ряда *тегов* характерно наличие атрибутов, которые могут иметь конкретные значения, устанавливаемые автором для изменения функции *тега*.

Атрибуты *тега* следуют за именем и отделяются друг от друга одним или несколькими знаками табуляции, пробелами или символами возврата к началу строки. Например, особенно важно использовать нужный регистр при вводе URL (Uniform Resource Locator, унифицированный указатель ресурса), других документов в качестве значения атрибута HREF.

Например, *тег* изображения <IMG>, который служит для вставки в документ графического изображения, конечного компонента не требует. К автономным

элементам разметки также относятся разрыв строки (
), горизонтальная линейка (<HR>) и *теги*, содержащие такую информацию о документе, которая не влияет на его отображаемое содержимое, например, *теги* <META> и <BASE>.

Общая схема построения контейнера в формате *HTML* может быть записана в следующем виде:

"контейнер"=
<"имя тега" "список атрибутов">
содержание контейнера
</"имя тега">

Кроме *тегов*, элементами *HTML* являются CER (Character Entity Reference), они предназначены для представления специальных символов в документе *HTML*, которые могут быть неверно обработаны браузером. Например, чтобы представить символ "<" в документе *HTML*, нужно заменить его на <, а символ ">" — на >. То есть, если указать в тексте *HTML* строку <BODY>, она будет выглядеть на экране как текст <BODY>.

Числовой код	**Именная замена**	**Символ**	**Описание**
"	"	"	Кавычка
&	&	&	Амперсанд
<	<	<	Меньше
>	>	>	Больше
			Неразрывный пробел
¡	¡	¡	Перевернутый восклицательный знак
¢	¢	¢	Цент
£	£	£	Фунт
¤	¤	¤	Валюта
¥	¥	¥	Йена

¨	¨	¨	Умляут
©	©	©	Копирайт
«	«	«	Левая угловая кавычка
®	®	®	Зарегистрированная торговая марка
±	±	±	Плюс или минус
»	»	»	Правая угловая кавычка

1.1.3. Группы тегов HTML

Все *теги* HTML по их назначению и области действия можно разделить на следующие основные группы:

- определяющие структуру документа;
- оформление блоков гипертекста (параграфы, списки, таблицы, картинки);
- гипертекстовые ссылки и закладки;
- формы для организации диалога;
- вызов программ.

Структура гипертекстовой сети задается гипертекстовыми ссылками.

Гипертекстовая ссылка — это адрес другого *HTML*-документа или информационного ресурса Internet, который тематически, логически или каким-либо другим способом связан с документом, в котором ссылка определена.

Реальный механизм интерпретации идентификатора ресурса, опирающийся на URI (Uniform Resource Identifier, универсальный идентификатор ресурса), называется URL, и пользователи WWW имеют дело именно с ним.

Гипертекстовые ссылки в **HTML** ***делятся на два класса:*** контекстные гипертекстовые ссылки и общие. ***Контекстные*** ссылки вмонтированы в тело документа, как это было продемонстрировано в предыдущем примере, в то время как

общие ссылки связаны со всем документом в целом и могут использоваться при просмотре любого фрагмента документа.

Структура **HTML**-документа позволяет задействовать вложенные друг в друга контейнеры. Собственно, сам документ — это один большой контейнер, который начинается с *тега* <HTML> и заканчивается *тегом* </HTML>.

В заключение отметим, что при подготовке документов *HTML* используется идентификатор текста DTD (Document Type Declaration, определение типа документа) в качестве первой строки. Это позволяет браузеру идентифицировать документ как соответствующий стандарту *HTML*.

Обычно (но не обязательно) каждый документ *HTML* начинается со строки типа: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

Здесь содержится информация о том, что документ соответствует версии *HTML 4.0*; разработанной W3C; используемый язык — английский.

HTML-документ — это один большой контейнер, который начинается с тега <HTML> и заканчивается тегом </HTML>:

```
<HTML>Содержание документа</HTML>
```

Контейнер HTML или гипертекстовый документ состоит из двух других вложенных контейнеров: *заголовка* документа (HEAD) и тела документа (BODY). Рассмотрим простейший пример классического документа.

```
<HTML>
<HEAD>
<TITLE>Простейший документ</TITLE>
</HEAD>
<BODY TEXT=#0000ff BGCOLOR=#f0f0f0>
<H1>Пример простого документа</H1>
```

```
<HR>
Формы HTML-документов
<UL>
<LI>Классическая
<LI>Фреймовая
</UL>
<HR>
</BODY>
</HTML>
```

1.2. Создание и форматирование текста в HTML-документе

1.2.1. Назначение заголовка

Заголовок HTML-документа является необязательным элементом разметки. Современная практика HTML-разметки такова, что почти в каждом документе есть HTML-*заголовок*.

Первоначально существование *заголовка* определялось необходимостью именования окна браузера. Это достигалось за счет элемента разметки *TITLE*:

```
<HTML>
 <HEAD>
  <TITLE>Это заголовок</TITLE>
 ...
 </HEAD>
 <BODY>
 ...
 </BODY>
</HTML>
```

Отображение содержания элемента TITLE

Однако задумывался *заголовок* для несколько иных целей. Все гипертекстовые связи информационных узлов принято разделять на контекстные и общие.

Контекстные гипертекстовые связи соответствуют определенному месту документа — контексту. ***Общие*** гипертекстовые связи определяются не частью документа (контекстом), а всем документом целиком.

Основные контейнеры заголовка. Основные контейнеры *заголовка* — это элементы HTML-разметки, которые наиболее часто встречаются в *заголовке* HTML-документа, т.е. внутри элемента разметки *HEAD*. Мы рассмотрим только восемь элементов разметки, включая сам элемент разметки *HEAD*:

- *HEAD* (элемент разметки *HEAD*);
- *TITLE* (заглавие документа);
- *BASE* (база URL);
- ISINDEX (поисковый шаблон);
- *META* (метаинформация);
- *LINK* (общие ссылки);
- *STYLE* (описатели стилей);
- *SCRIPT* (скрипты).

Элемент разметки HEAD

Элемент разметки ***HEAD*** содержит *заголовок* HTML-документа. При наличии тега начала элемента разметки желательно использовать и тег конца элемента разметки. Синтаксис контейнера *HEAD* в общем виде выглядит следующим образом:

<HEAD profile="http://www.intuit.ru/help">
Это пример из документации по сайту Интернет-Университета Информационных Технологий
</HEAD>

Контейнер *заголовка* служит для размещения информации, относящейся ко всему документу в целом.

Элемент разметки TITLE

Элемент разметки ***TITLE*** служит для именования документа в World Wide Web. Состоит контейнер из тега начала, содержания и тега конца. В различных браузерах алгоритм отображения элемента *TITLE* может отличаться. Так, в некоторых руководствах предлагается создать бегущую строку в *заголовке* документа, указав несколько последовательных контейнеров *TITLE*:

```
<TITLE>И</TITLE>
<TITLE>Ин</TITLE>
<TITLE>Инт</TITLE>
<TITLE>Инте</TITLE>
<TITLE>Интер</TITLE>
...
<TITLE>Интернет</TITLE>
<TITLE>Интернет</TITLE>
```

Синтаксис контейнера *TITLE* в общем виде выглядит следующим образом:

```
<TITLE>название документа</TITLE>
```

Элемент разметки BASE

Элемент разметки ***BASE*** служит для определения базового URL для гипертекстовых ссылок документа, заданных в неполной (частичной) форме.

```
<A HREF=../next_level/document.html>...</A>
```

Контейнер *BASE* можно использовать вне документа, в *заголовке* или теле документа. При этом область действия базового адреса определяется от места размещения контейнера до следующего контейнера *BASE*.

```
<BASE HREF=http://intuit.ru/start/>
<HTML>
<HEAD>
<BASE HREF=http://intuit.ru/cgi-bin/>
 ... </HEAD>
<BODY>
 <BASE HREF=http://intuit.ru/start/>
 ... </BODY>
</HTML>
```

Элемент разметки ISINDEX

Элемент разметки **ISINDEX** используется для указания поискового шаблона и унаследован от ранних версий HTML. В HTML 4.0 этот контейнер не определен. **ISINDEX** подходит для документов с компоновкой в стиле HTML 2.0.

```
<HTML>    <HEAD>
 <ISINDEX>
</HEAD>       <BODY>
 ...
</BODY>         </HTML>
```

Элемент разметки META

Это наиболее популярный элемент разметки *заголовка*, более распространен только элемент *TITLE*. Такое положение дел объясняется назначением данного элемента разметки. ***META*** содержит управляющую информацию, которую браузер использует для правильного отображения и обработки содержания тела документа.

Элемент разметки LINK

Элемент разметки *LINK* – это результат давно предпринятой попытки придать HTML академический вид. Согласно теории гипертекстовых систем, все

гипертекстовые связи разделяют на два типа: ***контекстные и общие***. Такое деление чисто условное и определяется тем, что контекстную связь можно привязать к определенному месту документа, а ***общую*** — отнести только ко всему документу целиком.

Элемент разметки STYLE

Элемент разметки *STYLE* предназначен для размещения описателей стилей.

Элемент разметки SCRIPT

Элемент разметки *SCRIPT* служит для размещения кода JavaScript, VBScript или JScript. Вообще говоря, *SCRIPT* можно использовать не только в *заголовке* документа, но и в его теле. Это можно сделать непосредственно в самом контейнере *SCRIPT*:

```
<SCRIPT LANGUAGE="JavaScript"
    SRC=script.code>
```

1.3. Создание гипер показателей, списки, таблицы, фреймы, формы и др.

Теги тела документа. *Теги тела* **документа предназначены для управления отображением информации в программе интерфейса пользователя.**

Тело документа состоит из:

- иерархических контейнеров и заставок;
- заголовков (от H1 до H6);
- *блоков* (параграфы, списки, формы, таблицы, картинки и т.п.);
- горизонтальных отчеркиваний и адресов;
- текста, разбитого на области действия стилей (подчеркивание, выделение, курсив);
- математических описаний, графики и гипертекстовых ссылок.

Тело документа – контейнер BODY Описание тегов тела документа следует начать с тега BODY. В отличие от тега HEAD, тег BODY имеет атрибуты. Атрибут BACKGROUND определяет фон, на котором отображается текст документа.

Таблица 3.1. Атрибуты	
Атрибут	**Значение**
BGCOLOR=#FFFFFF	Цвет фона
TEXT=#0000FF	Цвет текста
VLINK =#FF0000	Цвет пройденных гипертекстовых ссылок
LINK =#00FF00	Цвет гипертекстовой ссылки

В данной таблице строка #XXXXXX определяет цвет.

Таблица 3.2. Цвета			
Название	**Код**	**Название**	**Код**
aqua	#00FFFF	navy	#000080
black	#000000	olive	#808000
blue	#0000FF	purple	#800080
fuchsia	#FF00FF	red	#FF0000
gray	#808080	silver	#C0C0C0
green	#008000	teal	#008080
lime	#00FF00	white	#FFFFFF
maroon	#800000	yellow	#FFFF00

Теги управления разметкой. Заголовки.

Заголовок обозначает начало раздела документа. В стандарте определено 6 уровней заголовков: от H1 до H6. Текст, окруженный тегами <H1></H1>, получается большим — это основной заголовок. Если текст окружен тегами <H2></H2>, то он

выглядит несколько меньше (подзаголовок); текст внутри <H3></H3> еще меньше и так далее до <H6></H6>.

Ниже на рисунке показан результат использования следующих заголовков: (открыть)

<H1>Заголовок 1</H1>
<H2>Заголовок 2</H2>

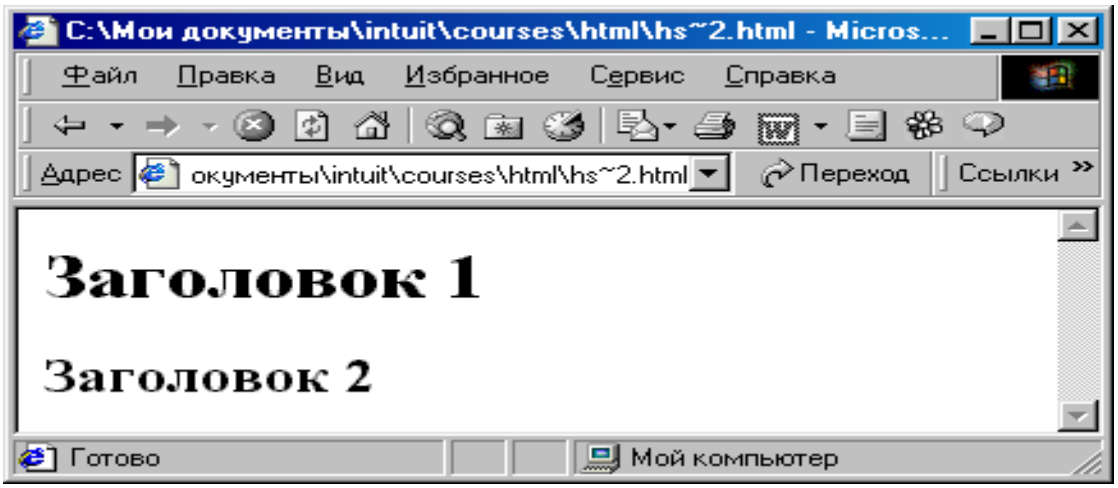


Рис. 3.1.

Тег <P>. Тег <P> применяется для разделения текста на параграфы. В нем используются те же атрибуты, что и в заголовках.

Атрибут ALIGN

Атрибут ALIGN позволяет выровнять текст по левому или правому краю, по центру или ширине. Далее приведены возможные значения атрибута ALIGN:

ALIGN=justify выравнивание по левому и правому краям.

ALIGN=left выравнивание по левому краю.

ALIGN=right выравнивание по правому краю.

ALIGN=center центрирование текста и графики.

С помощью атрибута ALIGN можно заставить текст "обтекать" графический объект. Для этого следует поместить тег <IMG SRC="../../../../путь/файл.gif"> туда, где должен быть графический объект, и добавить атрибут ALIGN=left, ALIGN=right или ALIGN=center.

*Использование тега
. Принудительный перевод строки используется для того, чтобы нарушить стандартный порядок отображения текста.*

<BR CLEAR=left> Текст будет продолжен, начиная с ближайшего пустого левого поля.

<BR CLEAR=right> Текст будет продолжен, начиная с ближайшего пустого правого поля.

<BR CLEAR=all> Текст будет продолжен, как только и левое, и правое поля окажутся пустыми.

Элемент разметки <NOBR>

Тег <NOBR> (No Break, без обрыва) дает браузеру команду отображать весь текст в одной строке, не обрывая ее.

Теги управления отображением символов

Все эти теги можно разбить на ***два класса***: теги, управляющие формой отображения (font style), и теги, характеризующие тип информации (information type).

Теги, управляющие формой отображения

Курсив, усиление, подчеркивание, верхний индекс, нижний индекс, шрифт большой, маленький, красный, синий, различные комбинации.

Теги <BIG> и <SMALL> — изменение размеров шрифта

Текст, расположенный между тегами <BIG></BIG> или <SMALL> </SMALL>, будет, соответственно, больше или меньше стандартного.

Верхние и нижние индексы

С помощью тегов <SUP> и <SUB> можно задавать верхние и нижние индексы, необходимые для записи торговых знаков, символов копирайта, ссылок и сносок.

Атрибут SIZE Атрибут SIZE тега <FONT> позволяет задавать размер текста в данной области.

Атрибут COLOR Если вы хотите сделать свою страницу более красочной, можете воспользоваться атрибутом COLOR в теге FONT, и тогда единственным ограничением будет цветовая палитра на компьютере пользователя.

Теги, управляющие формой отображения, приведены в таблице.

Таблица 3.3. Теги, управляющие формой отображения

Тег	Значение
<I>...</I>	Курсив (Italic)
<B>...</B>	Усиление (Bold)
<TT>...</TT>	Телетайп
<U>...</U>	Подчеркивание
<S>...</S>	Перечеркнутый текст
<BIG>...</BIG>	Увеличенный размер шрифта
<SMALL>...</SMALL>	Уменьшенный размер шрифта
_{...}	Подстрочные символы
^{...}	Надстрочные символы

Таблица 3.4. Теги, характеризующие тип информации

Тег	Значение
<EM>...</EM>	Типографское усиление
<CITE>...</CITE>	Цитирование

<STRONG>...</STRONG>	Усиление
<CODE>...</CODE>	Отображает примеры кода (например, "коды программ")
<SAMP>...</SAMP>	Последовательность литералов
<KBD>...</KBD>	Пример ввода символов с клавиатуры
<VAR>...</VAR>	Переменная
<DFN>...</DFN>	Определение
<Q>...</Q>	Текст, заключенный в двойные кавычки

Блоки цитат — элемент <BLOCKQUOTE>

Тег добавляет поля слева и справа от текста. Это полезный тег, поскольку он позволяет компактно расположить текст в центре страницы. При неоднократном использовании <BLOCKQUOTE> текст все больше сжимается к центру.

Создание списков в HTML

Списки являются важным средством структурирования текста и применяются во всех языках разметки. В HTML имеются следующие виды списков: ненумерованный список (неупорядоченный) (Unordered Lists <UL>), нумерованный список (упорядоченный) (Ordered Lists <OL>) и список определений.

Неупорядоченные списки — тег <UL>

Ненумерованный список. Ненумерованный список предназначен для создания текста типа:

- первый элемент списка;
- второй элемент списка;
- третий элемент списка.

Записывается данный список в виде последовательности:

<UL>

<LI>первый элемент списка
<LI>второй элемент списка
<LI>третий элемент списка
</UL>

Теги <UL> и </UL> — это теги начала и конца ненумерованного списка, тег <LI> (List Item) задает тег элемента списка. Помимо этих тегов, существует тег, позволяющий именовать списки — <LH> (List Header).

Упорядоченные списки — тег <OL>

Нумерованные списки. Тег <OL> вместе с атрибутом TYPE= в *HTML* 3.2 позволяет создавать нумерованные списки, используя в качестве номеров не только обычные числа, но и строчные и прописные буквы, а также строчные и прописные римские цифры. При необходимости можно даже смешивать эти типы нумерации в одном списке:

<OL TYPE=l> Тег создает список с нумерацией в формате 1., 2., 3., 4. и т.д.
<OL TYPE=A> Тег создает список с нумерацией в формате A., B., C., D. и т.д.
<OL TYPE=a> Тег создает список с нумерацией в формате a., b., c., d. и т.д.
<OL TYPE=I> Тег создает список с нумерацией в формате I., II., III., IV. и т.д.

Список определений — тег <DL>

Теги списка (Definition List: <DL>, <DT>, <DD>) используют для создания списка терминов и их определений. Схема использования тега следующая.

<DL><DT>Термин</DT> <DD>Определение</DD></DL>

Горизонтальные линейки — тег <HR>

Горизонтальное отчеркивание (Horizontal Rule) применяется для разделения документа на части.

Преформатированный вывод — тег <PRE>

Применение этого тега позволяет отобразить текст "как есть" (без форматирования), теми же символами и с тем же разбиением на строки.

Применение тега <BLINK>

Текст, помещенный между тегами <BLINK> и </BLINK>, мерцает. Данный тег поддерживается только браузером Netscape Navigator. Пользоваться им следует с большой осторожностью.

Комментарии в языке HTML

Комментарии *HTML* начинаются с символа "<!--" и оканчиваются символом "-->".

Гипертекстовые ссылки

Для записи гипертекстовой ссылки используется тег <A>, который называют "якорь" (anchor). Якорь имеет несколько атрибутов, главным из которых является HREF.

Использование графики в HTML

Для того чтобы вставить в Web-страницу изображение, необходимо либо нарисовать его, либо взять уже готовое.

Приведем наиболее распространенные случаи применения изображений:

- логотип компании на деловой странице;
- графика для рекламного объявления;
- различные рисунки;
- диаграммы и графики;
- художественные шрифты;
- подпись автора страницы;

- применение графической строки в качестве горизонтальной разделительной линии;
- применение графических маркеров для создания красивых маркированных списков.

Атрибут SRC

Указывает файл изображения и путь к нему; изображение должно быть загружено в браузер и размещено в том месте документа, где расположен тег изображения.

Атрибут ALT

Позволяет указать текст, который будет выводиться вместо изображения браузерами, неспособными представлять графику.

Атрибут ALIGN

Определяет положение изображения относительно окружающего его текста. Возможные значения аргумента — ["top" | "middle" | "bottom"] (соответственно, "вверху", "посередине", "внизу").

ALIGN="top" выравнивает верх изображения по верхнему краю самого высокого элемента в строке окружающего текста.

ALIGN="middle" выравнивает центр изображения по базовой линии строки окружающего текста.

ALIGN="bottom" выравнивает нижний край изображения по базовой линии строки окружающего текста.

ALIGN="left" определяет огибаемое текстом изображение. Изображение располагается вдоль левой границы документа, а последующие строки текста огибают его справа.

ALIGN="right" определяет огибаемое текстом изображение. Изображение располагается вдоль правой границы документа, а последующие строки текста огибают его слева.

ALIGN="top" выравнивает верх изображения по верхнему краю самого высокого элемента в строке окружающего текста точно так же, как при использовании стандартного набора атрибутов.

ALIGN="texttop" выравнивает верх изображения по верхнему краю самого высокого текстового символа в строке окружающего текста. Действие этого аргумента в большинстве случаев, но не всегда, подобно действию аргумента ALIGN="top".

ALIGN="middle" выравнивает центр изображения по базовой линии строки окружающего текста точно так же, как при использовании стандартного набора атрибутов.

ALIGN="absmiddle" выравнивает центр изображения по центру строки окружающего текста.

ALIGN="baseline" выравнивает нижний край изображения по базовой линии строки окружающего текста, то есть производит такое же действие, как и ALIGN="bottom".

ALIGN="bottom" выравнивает нижний край изображения по базовой линии строки окружающего текста точно так же, как при использовании стандартного набора атрибутов.

ALIGN="absbottom" выравнивает нижний край изображения по нижнему краю строки окружающего текста.

Атрибут USEMAP

Если присутствуют атрибут USEMAP и тег <MAP>, изображение становится чувствительной картой, или "графическим меню".

Атрибут BORDER

Целочисленное значение аргумента определяет толщину рамки вокруг изображения. Если значение равно нулю, рамка отсутствует.

Атрибут HSPACE

Целочисленное значение этого атрибута задает горизонтальное расстояние между вертикальной границей страницы и изображением, а также между изображением и огибающим его текстом.

Атрибут VSPACE

Целочисленное значение этого атрибута задает вертикальное расстояние между строками текста и изображением.

Атрибуты WIDTH и HEIGHT

Оба атрибута задают целочисленные значения размеров изображения по горизонтали и по вертикали соответственно.

Форматы графических файлов

Самыми распространенными графическими форматами в Web являются *GIF* и *JPEG*. ***GIF*** — наиболее подходящий формат для обмена изображениями между системами.

Активные изображения

Активные изображения (***image maps***), или изображения, чувствительные к щелчкам мыши, позволяют создать на узле графические меню произвольной формы. ***Активное изображение*** — это изображение с так называемыми активными областями (hot spots), которые ссылаются на URL других страниц или узлов.

Средства описания таблиц в HTML

Создание таблиц в HTML

Для описания ***таблиц*** используется тег <TABLE>. Тег <TABLE>, как и многие другие, автоматически переводит строку до и после *таблицы*.

Создание строки таблицы - тег <TR>

Тег <TR> (Table Row, строка *таблицы*) создает строку *таблицы*. Весь текст, другие теги и атрибуты, которые требуется поместить в одну строку, должны размещаться между тегами <TR></TR>.

Определение ячеек таблицы - тег <TD>

Внутри строки *таблицы* обычно размещаются ячейки с данными. Каждая ячейка, содержащая текст или изображение, должна быть окружена тегами <TD></TD>. Число тегов <TD></TD> в строке определяет число ячеек

Заголовки столбцов таблицы - тег <TH>

Заголовки для столбцов и строк *таблицы* задаются с помощью тега заголовка <TH></TH> (Table Header, заголовок *таблицы*).

Использование заголовков таблицы - тег <CAPTION>

Тег <CAPTION> позволяет создавать заголовки *таблицы*.

Атрибут NOWRAP

Обычно любой текст, не помещающийся в одну строку ячейки *таблицы*, переходит на следующую строку. Однако при использовании атрибута NOWRAP с тегами <TH> или <TD> длина ячейки расширяется настолько, чтобы заключенный в ней текст поместился в одну строку.

Атрибут COLSPAN

Теги <TD> и <TH> модифицируются с помощью атрибута COLSPAN (Column Span, соединение столбцов). Если вы хотите сделать какую-нибудь ячейку шире, чем верхняя или нижняя, можно воспользоваться атрибутом COLSPAN, чтобы растянуть ее над любым количеством обычных ячеек.

Атрибут ROWSPAN

Атрибут ROWSPAN, используемый в тегах <TD> и <TH>, подобен атрибуту COLSPAN=, только он задает число строк, на которые растягивается ячейка. Если вы указали в атрибуте ROWSPAN=s число, большее единицы, то соответствующее количество строк должно находиться под растягиваемой ячейкой. Внизу *таблицы* ее поместить нельзя.

Атрибут WIDTH

Атрибут WIDTH применяется в двух случаях. Можно поместить его в тег <TABLE>, чтобы дать ширину всей *таблицы*, а можно использовать в тегах <TD> или <TH>, чтобы задать ширину ячейки или группы ячеек.

Применение пустых ячеек

Если ячейка не содержит данных, она не будет иметь границ.

Атрибут CELLPADDING

Данный атрибут определяет ширину пустого пространства между содержимым ячейки и ее границами, то есть задает поля внутри ячейки.

Атрибуты ALIGN и VALIGN

Теги <TR>, <TD> и <TH> можно модифицировать с помощью атрибутов ALIGN и VALIGN. Атрибут ALIGN определяет выравнивание текста и *графики* по горизонтали, то есть по левому или правому краю, либо по центру. Горизонтальное выравнивание может быть задано несколькими способами:

ALIGN=bleedleft прижимает содержимое ячейки вплотную к левому краю.

ALIGN=left выравнивает содержимое ячейки по левому краю с учетом отступа, заданного атрибутом CELLPADDING.

ALIGN=center располагает содержимое ячейки по центру.

ALIGN=right выравнивает содержимое ячейки по правому краю с учетом отступа, заданного атрибутом CELLPADDING.

VALIGN=top выравнивает содержимое ячейки по ее верхней границе.

VALIGN=middle центрирует содержимое ячейки по вертикали.

VALIGN=bottom выравнивает содержимое ячейки по ее нижней границе.

Атрибут BORDER

В теге <TABLE> часто определяют, как будут выглядеть рамки, то есть линии, окружающие ячейки *таблицы* и саму *таблицу*. Если вы не зададите рамку, то получите *таблицу* без линий, но пространство под них будет отведено. Того же результата можно добиться, задав <TABLE BORDER=0>.

Атрибут CELLSPACING

Атрибут CELLSPACING определяет ширину промежутков между ячейками в пикселах. Если этот атрибут не указан, по умолчанию задается величина, равная двум пикселам. С помощью атрибута CELLSPACING= можно размещать текст и *графику* там, где вам нужно.

Атрибут BGCOLOR

Данный атрибут позволяет установить цвет фона. В зависимости от того, с каким тегом (TABLE, TR, TD) он применяется, цвет фона может быть установлен

для всей *таблицы*, для строки или для отдельной ячейки. Значением данного атрибута является RGB-код или стандартное название цвета.

Атрибут BACKGROUND

Данный атрибут задает фоновое изображение для *таблиц*. Применим к тегам TABLE и TD. Его значением является URL файла с фоновым изображением. Применение этого атрибута рассматривается ниже.

Использование таблиц в дизайне страницы

Создание разноцветных таблиц

Некоторые браузеры позволяют отображать цвета. Есть несколько способов раскрасить *таблицу*, в основном они зависят от используемого браузера.

HTML-формы

Формы были созданы и используются в WWW для получения отклика пользователя на предоставленную информацию и сбора данных о пользователе.

Задание формы — элемент FORM

Элемент FORM обозначает документ как *форму* и определяет границы использования других тегов, размещаемых в *форме*.

<FORM METHOD=post ACTION=mailto:yourname@your.email.address>

Определение элементов управления формы — тег <INPUT>

Данный тег используют для определения области внутри *формы*, куда вводятся данные. Он формирует *поле для ввода информации* пользователем. Это может быть текстовое поле, опция, изображение или кнопка. Вид *поля ввода* определяется значением атрибута TYPE.

Атрибут TYPE=text

Когда пользователю необходимо ввести небольшое количество текста (одну или несколько строк), используется тег <*INPUT*>, и атрибут TYPE устанавливается в значение text.

Создание многострочных областей ввода текста — тег <TEXTAREA>

В зависимости от типа *формы* может потребоваться организовать ввод большого количества текста. В таких случаях используется тег ***<TEXTAREA>*** для создания текстового поля из нескольких строк. Данный тег использует три атрибута: COLS, NAME и ROWS.

Атрибут COLS

Указывает (число символов) число колонок, содержащихся в текстовой области.

Атрибут NAME Определяет наименование поля.

Атрибут ROWS Задает количество видимых строк текстовой области.

<TEXTAREA NAME=тема COLS=38 ROWS=3> </TEXTAREA>

Использование списков в форме — тег <SELECT>

Когда формы HTML становятся более сложными, в них часто включают списки с прокруткой и выпадающие *меню*. Для этого используют тег ***SELECT*** с атрибутом TYPE=select. Для определения списка пунктов используют тег <*OPTION*>. Тег <*SELECT*> поддерживает три необязательных атрибута: MULTIPLE, NAME и SIZE.

Атрибут MULTIPLE Позволяет выбрать более чем одно наименование.

Атрибут NAME Определяет наименование объекта.

Атрибут SIZE Определяет число видимых пользователю пунктов списка. Если в *форме* установлено значение атрибута SIZE=1, то браузер выводит на экран список в

виде выпадающего *меню*. В случае SIZE > 1 браузер представляет на экране обычный список.

Атрибут SELECTED Используется для первоначального выбора значения элемента по умолчанию.

Атрибут VALUE

Указывает на значение, возвращаемое *формой* после выбора пользователем данного пункта. По умолчанию значение *поля* равно значению тега <*OPTION*>

Как работают фреймы

На первый взгляд, *фреймы* — это нечто сложное, но их легче понять, если провести аналогию с ячейками таблицы. Расположение *фреймов* на экране и ячеек в таблице задается почти одинаково: теги и атрибуты работают так же, как их табличные "родственники".

Создание простой страницы с фреймами

Построим страницу с двумя *фреймами*. Зададим слева *фрейм* оглавления с заголовками статей, а справа поместим страницу с самими статьями. Сделаем так, что когда пользователь щелкает мышкой на ссылке в той части экрана, где находится оглавление, сама статья появляется в правом *фрейме*. Это основной, наиболее распространенный способ использования *фреймов*.

Задание фреймовой структуры

Для начала мы должны представить себе общий вид страницы – где расположить *фреймы* и какого они будут размера. Затем можно подумать об их содержании. Ниже приводится код простой фреймовой *структуры* с использованием тега <*FRAMESET*>. Обратите внимание: страница с фреймовой *структурой* не содержит тега <BODY>.

Подготовка содержимого фрейма

Теперь загрузим *фреймы* с содержимым. Зададим страницу menu.html в левом *фрейме*, где мы собираемся щелкать мышью, переключаясь между двумя страницами в правом *фрейме*. menu.html — это обычная HTML-страница, построенная как оглавление.

Подготовка фрейма main

Правый *фрейм* main будет содержать сами HTML-страницы. Ваша задача — спроектировать их так, чтобы они хорошо смотрелись в меньшем, чем обычно, окне, потому что часть экрана будет занята левым кадром оглавления.

Использование тега *<NOFRAMES>*

Если читатель с устаревшим браузером окажется на вашей странице с фреймовой *структурой*, все, что находится на ней между тегами ***<NOFRAMES>*** и *</NOFRAMES>*, будет выглядеть отлично — браузер просто проигнорирует *фреймы*. Вот почему обязательно нужно использовать теги <BODY> </BODY>. Возможно, экран без *фреймов* придется организовать иначе.

Макетирование фреймов — тег *<FRAMESET>*

Теги *<FRAMESET>* обрамляют текст, описывающий компоновку *фреймов*. Здесь размещается информация о числе *фреймов*, их размерах и ориентации (горизонтальной или вертикальной).

Атрибуты ROWS и COLS

Для каждой строки и столбца, упомянутых в теге *<FRAMESET>*, необходим свой набор тегов <FRAME>.

Атрибут ROWS

Атрибут ROWS тега *<FRAMESET>* задает число и размер строк на странице.

Атрибут COLS

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

Задание содержимого *фрейма* — элемент FRAME

Тег <FRAME> определяет внешний вид и поведение *фрейма*. Этот тег не имеет закрывающего тега, поскольку в нем ничего не содержится. Вся суть тега <FRAME> в его атрибутах. Их шесть: NAME, MARGINWIDTH, MARGINHEIGHT, SCROLLING, NORESIZE и SRC.

Атрибут NAME

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.

Атрибут MARGINWIDTH

Атрибут MARGINWIDTH действует аналогично атрибуту таблиц CELLPADDING. Он задает горизонтальный отступ между содержимым кадра и его границами.

Атрибут MARGINHEIGHT

Атрибут MARGINHEIGHT действует так же, как и MARGINWIDTH. Он задает поля в верхней и нижней частях *фрейма*.

Атрибут SCROLLING

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.

Атрибут NORESIZE

Как правило, пользователь может, перемещая границу *фрейма* мышкой, изменить его размер. Это удобно, но не всегда. Иногда требуется атрибут NORESIZE.

Атрибут SRC

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

Атрибут TARGET

Чтобы разобраться с атрибутом TARGET, необходимо вернуться к простому примеру с кадром оглавления.

Вложенные и множественные кадровые структуры

Вложенные *фреймы* не очень способствуют навигации. И все же бывают случаи, когда возникает потребность разместить одни *фреймы* внутри других. *Фреймы* сами по себе — необычное средство навигации.

Литература

1. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. Основы web-технологий БИНОМ.
2. Erik Wilde: Wilde's WWW, technical foundations of the World Wide Web. Springer 1998,ISBN:3 – 540 – 64285 – 4 [CSS1].

ТЕМА 6. ВВОД В CSS (4 ЧАСОВ)

План.

2.1. Синтаксис CSS.

2.2. Применение CSS в HTML – документе.

2.3. Свойства шрифты, тексты, цвет и т.д..

2.1. Синтаксис CSS.

Синтаксис

Формально стиль отображения *элементов разметки* задается ссылкой в *элементе разметки* на *селектор* стиля. *Синтаксис* описания стилей в общем виде представляется следующим образом:

selector[, selector[, ...]]

{ attribute:value;

[attribute:value;...] }

или

selector selector [selector ...]

{ attribute:value;

[attribute:value;...] }

В первом варианте перечислены *селекторы*, для которых действует данное описание стиля. Второй вариант задает иерархию вложенности *селекторов*, для совокупности которых определен стиль. Напомним, что речь в данном случае идет об описаниях стилей в нотации `text/css`. Описания стилей размещаются либо внутри элемента `STYLE`, либо во *внешнем файле*.

В качестве *селектора* можно использовать имя *элемента разметки*, *имя класса* и *идентификатор объекта* на HTML-странице.

Атрибут `(attribute)` определяет свойство отображаемого элемента, например левый отступ параграфа `(margin-left)`, а значение `(value)` — значение этого атрибута, например, 10 типографских пунктов (10 pt).

Селектор — имя элемента разметки

Когда автор Web-узла хочет определить общий стиль всех страниц, он просто прописывает стили для всех *элементов HTML-разметки*, которые будут использоваться на страницах. Это дает возможность скомпоновать страницы из логических элементов, а стиль отображения элементов описать во *внешнем файле*. Такой способ создания сайта позволяет автору изменять внешний вид всех страниц путем внесения изменений в файл описания стилей, а не в файлы HTML-страниц.

Внешний файл при этом может выглядеть следующим образом:

```
I, EM {color:#003366;font-style:normal}
A I {font-style:normal;font-weight:bold;
    text-decoration:line-through}
```

В первой строке этого описания перечислены селекторы-элементы, которые будут отображаться одинаково:

```
<I>Это курсив</I> и это тоже <EM>курсив</EM>
```

Последняя строка определяет стиль отображения вложенного в гипертекстовую ссылку курсива:

```
<A NAME=empty><I>intuit</I></A>
```

В данном случае *переопределение* состоит в том, что текст отображается внутри гипертекстовой ссылки перечеркнутым, причем жирным шрифтом.

Селектор — имя класса

Имя класса не является каким-либо стандартным именем *элемента HTML-разметки*. Оно определяет описание класса *элементов разметки*, которые будут отображаться одинаково. Для того, чтобы отнести *элемент разметки* к тому или иному классу, нужно воспользоваться его атрибутом `CLASS`:

Лидирующую точку в *имени класса* можно опустить. Она задается из соображений сохранения единства описания. Например, можно определить классы отображения однотипных *элементов разметки*:

```
a.menu { color:red;background-color:white;
      text-decoration:none; }
a.paragraph { color:navy;
         text-decoration:underline; }
```

Класс гипертекстовых ссылок `menu` имеет одно описание стиля, а класс гипертекстовых ссылок `paragraph` — совершенно другое. При этом каждый из этих классов нельзя применить к другим *элементам разметки*, например, параграфу или списку. Если имя *элемента разметки* не задано, это означает, что класс можно отнести к любому *элементу разметки* — корневой класс описания стилей. Это очень похоже на обозначение имени корневого домена в системе доменных имен. Собственно ничего удивительного здесь нет, т.к. система классов объектов на HTML-странице представляет собой дерево. *Элементы разметки* — это узлы дерева.

Селектор — идентификатор объекта

Объектная модель документа (Document Object Model) описывает документ как дерево объектов. Объектами являются: сам документ, его разделы (элемент `DIV`), картинки, параграфы, приложения и т.п. Каждый из объектов можно поименовать и обращаться к нему по имени. Данная возможность используется при программировании страниц на стороне клиента.

Применение *идентификатора объекта* оправдано еще и в случае модификации атрибута описания стиля для данного объекта в его CSS-описании. Вместо двух описаний классов, которые отличаются только одним из параметров, можно создать одно описание класса и описание *идентификатора объекта*. Описание стиля для объекта задается строкой, в которой *селектор* представляет собой имя этого объекта с лидирующим символом `"#"`:

```
a.mainlink { color:darkred;
        text-decoration:underline;
        font-style:italic; }
#blue { color:#003366 }
...
<A CLASS=mainlink>основная гипертекстовая
ссылка</A>
<A CLASS=mainlink ID=blue>модифицированная
гипертекстовая ссылка</A>
```

Следует отметить, что интерпретация *идентификаторов объектов* в Internet Explorer и Netscape Navigator различна. Существует еще атрибут name у *элемента разметки*. При идентификации объекта Netscape Navigator обычно имеет дело именно с этим атрибутом, а Internet Explorer — с атрибутом `ID`.

Различия в интерпретации `ID` в браузерах при декларативном использовании CSS не очень страшны. Другое дело, если автор решится программировать стили, т.е. изменять значения атрибутов описателей стилей. В этом случае разница *объектных моделей документов* в Netscape Navigator и Internet Explorer проявится в полной мере. Фактически, придется для каждого из браузеров разрабатывать совершенно разные страницы.

Наследование и переопределение

При обсуждении технических спецификаций часто бывает полезно вникнуть в смысл названия. В названии принято точно определять суть и назначение стандарта или спецификации. Описание стилей отображения *элементов HTML-разметки* носит название *"Каскадные таблицы стилей"*. Со словом "стилей" все более-менее понятно. Под словом "таблицы" следует понимать набор свойств *элемента разметки*, который можно представить в виде строки в таблице свойств, т.е. *элементы разметки* — строки, а свойства — столбцы. А вот слово "каскадные" требует пояснения.

Во-первых, существует иерархия *элементов разметки* (дерево объектов на странице). Во-вторых, свойства этих объектов могут наследоваться. Таким образом в дереве объектов образуется ветвь, которая ведет к листу дерева — *элементу разметки*, например, элементу списка или параграфу. Его свойства определяются *элементами разметки*, в которые вложен элемент, и описателями стиля для данного элемента.

Когда объяснение некоторого феномена HTML-разметки растягивается на несколько параграфов, имеет смысл воспользоваться приведенной ниже графической схемой построения страницы.

При использовании стилей действуют следующие правила ***старшинства стилей***:

Рис. 8.3.

- сначала применяются стили браузера по умолчанию;
- стили браузера по умолчанию переопределяются прилинкованными стилями (элемент LINK заголовка документа);
- прилинкованные стили переопределяются описаниями стилей в элементе STYLE;
- стили элемента STYLE переопределяются атрибутом STYLE в любом из *элементов разметки*.

Не все атрибуты стиля могут наследоваться. Например, "набивка" (отступ содержания элемента от его границ) элемента BODY не наследуется вложенными в него элементами и определяется по умолчанию или прописывается для каждого элемента отдельно. Алгоритмы *наследования* в Internet Explorer и в Netscape Navigator разные, поэтому для единства отображения элементов следует прописывать стиль по максимуму атрибутов.

Блочные и строковые элементы

В описании элементов разметки языка HTML существует понятие *строкового* (in-line) элемента разметки и *блочного* (block) элемента разметки. Формально они определены в DTD SGML-описания языка HTML. Объяснить различие между *блочным* и *строковым элементами* можно на примере:

- **параграф** — это *блочный элемент* разметки;
- **выделение курсивом** — это *строковый элемент* разметки.

Блочные элементы можно вкладывать друг в друга, но они не должны пересекаться. *Строковые элементы* можно как вкладывать, так и пересекать (согласно DTD и практике старых версий браузеров), но последнее делать не рекомендуется.

Очевидно, что по набору атрибутов управления отображением (атрибуты описания стиля) *строковые* и *блочные элементы* отличаются. Упрощенно можно сказать, что атрибуты описания стиля *строкового элемента* являются подмножеством атрибутов описания стиля *блочного элемента*.

Обобщениями *блочного* и *строкового элементов* , с точки зрения стилей, являются элементы DIV и SPAN, соответственно.

Элемент DIV

DIV играет роль универсального блока. *Блочный элемент* всегда отделен от прочих элементов страницы (контекста) пустой строкой. DIV не несет никакой смысловой нагрузки. Часто говорят, что DIV — это раздел страницы. Но на самом деле его применение имеет смысл только в контексте CSS. Никаких правил по умолчанию для отображения DIV не существует. Это просто новая строка текста.

Если текст будет просматриваться браузерами, не поддерживающими CSS, элемент DIV использовать не рекомендуется. В этом случае лучше применить параграф или другой подходящий по смыслу элемент разметки из стандартного набора HTML.

Элемент SPAN

Элемент разметки SPAN — это обобщенный *строковый элемент* разметки, применение которого не приводит к образованию блока текста. Он может заменить элементы FONT, I, B, U, SUB, SUP и т.п. Приведем примеры таких соответствий:

Таблица 9.1.

HTML-элемент	**CSS-аналог**
<FONT COLOR=red> ...</FONT>	<SPAN STYLE="color:red; ">...</SPAN>
<I>...</I>	<SPAN STYLE="font-style:italic; ">...</SPAN>
<B>...</B>	<SPAN STYLE="font-weight:bold; ">...</SPAN>
<U>...</U>	<SPAN STYLE="text-decoration:underline; ">...</SPAN>
	и т.п.

Применение элемента SPAN ограничено браузерами, которые поддерживают CSS. При этом не все атрибуты спецификации CSS поддерживаются в браузерах.

Например, атрибут `vertical-align`, который призван заменить элементы `SUP` и `SUB`, не поддерживается ни одним из браузеров.

Свойства блоков

Блочные элементы (блоки текста или box) позволяют оперировать с текстом в терминах прямоугольников, которые этот текст занимает. При этом блок текста становится элементом дизайна страницы с теми же свойствами, что и картинка, таблица или прямоугольная область приложения.

Блок текста обладает свойствами: *высоты* (height), *ширины* (width), *границы* (border), *отступа* (margin), *набивки* (padding), *произвольного размещения* (float), *управления обтеканием* (clear).

Графически свойства можно представить следующим образом:

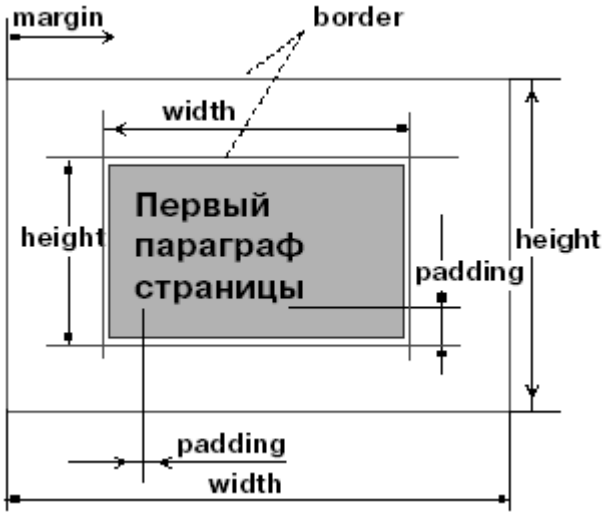


Рис. 9.2.

С *шириной* и *высотой блока* текста все более или менее понятно. Задаваться они могут в типографских пунктах (pt), пикселах (px) и условных единицах (em).

С *высотой блока* текста следует быть осторожным, т.к. в четвертой версии Netscape Navigator многие из атрибутов CSS не поддерживаются, в том числе высота обычного *блочного элемента*.

Расстояние от *границы блочного элемента* до *границы* вложенного в него *блочного элемента* называется padding. В рамках данного курса лекций для обозначения этого свойства используется слово ***"набивка"*** или словосочетание ***"внутренний отступ"***.

Отступ от *"набивки"* внешнего блочного элемента до *границы* вложенного элемента называется margin. Для его обозначения мы будем употреблять термин ***"отступ"*** или словосочетание ***"внешний отступ"***.

Таким образом padding и margin характеризуют *отступы блочного элемента* относительного начала его содержания и относительно границы охватывающего его элемента разметки, соответственно:

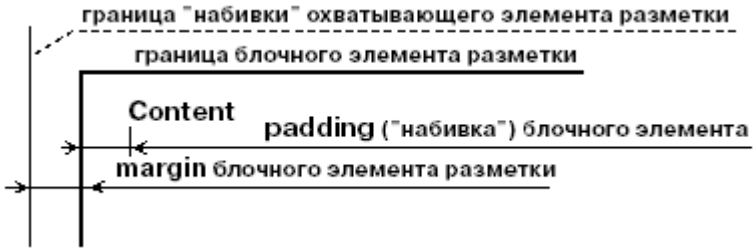


Рис. 9.4.

Отступы и *"набивка"* могут быть левыми, правыми, верхними и нижними. CSS позволяет изменять любые из них. Более подробно отступы рассматриваются в разделе *"Отступы"*(margin)", а *"набивка"* в разделе *"Набивка"* (padding)".

При отображении *блока текста* можно показать его видимую *границу*. CSS позволяет определить ее стиль, ширину и цвет. При использовании видимой *границы* следует учитывать специфику браузеров. Одним из возможных способов применения *границы* является видимое ограничение "плавающих" блоков текста.

"Плавающий" текстовый блок позволяет реализовать возможность *обтекания этого блока текстом*.

Прижмем блок текста вправо. Слева его будет обтекать другой текст.

Обтекание одного текста другим происходит в том же ключе, что и Обтекание текстом картинки или таблицы. Текст охватывающего блока стремится "втиснуться" на свободное место, оставленное "плавающим" блоком. Когда *граница* "плавающего" блока кончается, охватывающий блок распространяется на всю ширину отведенного для текста пространства.

CSS позволяет выравнивать блок текста не только по краю страницы, но и по центру (только в Netscape Navigator).

Отцентрируем блок текста.

Блок размещен по центру страницы, если страница просматривается в Netscape Navigator. CSS не поддерживает значение `center` для атрибута `float`.

Таким образом, блок текста с точки зрения размещения на странице равноценен картинкам или прямоугольным областям приложений.

Отступы (margin)

При отображении блока текста на бумаге вокруг него обычно оставляют поля. Поля можно задавать либо относительно границы страницы, либо относительно самого блока текста. В первом случае мы имеем дело с *"набивкой"* (padding), а во втором — с *отступом* (margin). Собственно, ширина поля будет определяться суммой ширины *"набивки"* и ширины *отступа*:

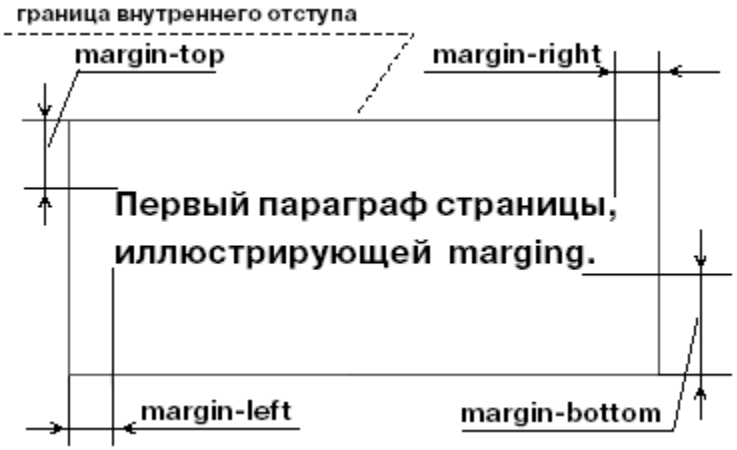


Рис. 9.5.

Обычно пунктирная линия и *граница блока* являются невидимыми линиями. Они угадываются по выравненному краю текста. Вернее, угадывается суммарная ширина полей. Стрелки указывают направление отсчета отступа. Padding отсчитывается от внешней *границы блока* внутрь блока, в то время как margin — от внешней *границы блока* в область охватывающего его блока (наружу).

Внешний отступ (margin) может отсчитываться по любому направлению относительно сторон блока:

margin-left — левый *внешний отступ*. Определяет расстояние от левой *границы блока* текста до левой границы *внутреннего отступа* (*"набивки"*, padding) охватывающего элемента;

margin-right — правый *внешний отступ*. Определяет расстояние от правой *границы блока* текста до правой границы *внутреннего отступа* (*"набивки"*, padding) охватывающего элемента;

margin-top — верхний *внешний отступ*. Определяет расстояние от верхней *границы блока* текста до верхней границы *внутреннего отступа* (*"набивки"*, padding) охватывающего элемента;

margin-bottom — нижний *внешний отступ*. Определяет расстояние от нижней *границы блока* текста до нижней границы *внутреннего отступа* (*"набивки"*, padding) охватывающего элемента;

margin — задает общий *внешний отступ* от всех сторон блока текста. Применяется в том случае, если блок текста равноудален от всех границ *внутреннего отступа* охватывающего элемента.

Графически эти отступы можно представить следующим образом:

Рис. 9.6.

В данном случае для параграфа использовалось следующее описание стиля:

```
P{margin-left:50px;margin-right:5px;margin-top:15px;margin-bottom:50px; padding:0px; text-align:left;}
```

Нужно иметь в виду, что браузеры могут отображать эти параметры по-разному. Netscape Navigator 4.x довольно неуклюже обрабатывает `margin`, оптимизируя представление стиля там, где этого делать не нужно.

Если размер всех *внешних отступов* одинаковый, то можно просто воспользоваться атрибутом `margin`:

```
P { margin:5px; }
```

При применении *внешнего отступа* следует помнить, что он отсчитывается от границы элемента до границы *внутреннего отступа* (*"набивки"*, padding) охватывающего элемента. Если этот факт не учитывать, то общая ширина видимых полей может оказаться больше, чем указано во *внешнем отступе*.

Набивка (padding)

Текст внутри блока начинается не от самой его *границы*. Между *границей* и содержанием блока есть свободное пространство. Оно называется *внутренний отступ* текстового блока или padding. Совместно с *внешним отступом* (margin) текстового блока padding образует общее поле отступа от границы охватывающего блок элемента разметки.

Padding можно проиллюстрировать на примере левого *внутреннего отступа* текста в параграфе:

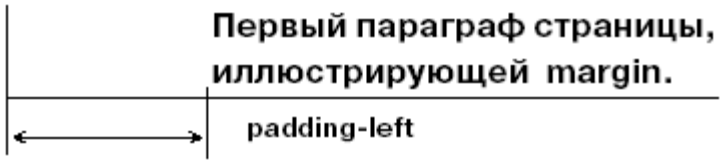


Рис. 9.7.

Для этого примера при описании параграфа использовался стиль:

```
P { padding-left:100px;text-align:left; border-width:1px; }
```

Чтобы браузер правильно отображал стили, не следует размещать описание стиля на нескольких строчках, как это сделано в примере. Для Internet Explorer это не имеет значения, а Netscape Navigator может "споткнуться".

У блока текста существует четыре стороны. Соответственно, padding может быть:

padding-left — левый *внутренний отступ*, который определяет расстояние от левого края блока до его содержания;

padding-right — правый *внутренний отступ*, который определяет расстояние от правого края блока до его содержания;

padding-top — верхний *внутренний отступ*, который определяет расстояние от верхнего края блока до его содержания;

padding-bottom — нижний *внутренний отступ*, который определяет расстояние от нижнего края блока до его содержания;

padding — определяет единый размер *внутреннего отступа* блока. Этот параметр задается в случае одинакового размера *отступа* от всех сторон блока.

Проиллюстрируем применение padding на примере:

Рис. 9.8.

```
P { padding-left:100px;padding-right:50px; padding-top:20px;padding-bottom:10px;
  text-align:left; border-width:1px; }
```

При установке padding следует помнить, что этот параметр задает размер *отступа* от *границы блока* до границы *внешнего отступа* (margin) содержания блока. По этой причине общий размер поля может оказаться больше, чем задано в параметре padding.

Граница (border)

У каждого блочного элемента разметки есть *граница*. От *границы* отсчитываются отступы (margin и padding). Вдоль границы "плавающего" блока его обтекает текст.

Для описания границ блоков применяются следующие атрибуты:

border-top-width — ширина верхней *границы блока*;

border-bottom-width — ширина нижней *границы блока*;

border-left-width — ширина левой *границы блока*;

border-right-width — ширина правой *границы блока*;

border-width — ширина *границы блока*. Задается в том случае, если ширина *границы блока* одинаковая по всему периметру блока;

border-color — цвет *границы блока*. Согласно спецификации CSS1 может быть задан для каждой из *границ блока*. Например, `border-right-color:red`. Может

задаваться как мнемоникой (red, blue, navy и т.п.), так и в нотации RGB (`border-color:#003366`). Указание цвета для каждой из *границ* поддерживается не всеми браузерами;

border-style — тип линии *границы блока*. Может принимать значения: `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`. Согласно спецификации CSS1, может быть задан для каждой из *границ блока*. Например, `border-right-style:dotted`. Указание типа линии *границы* поддерживается не всеми браузерами.

Для описания *границы* нет необходимости указывать в стиле все атрибуты. Существует сокращенная запись атрибутов. Например, для описания верхней линии *границы* можно использовать запись типа:

P { border-top:1px dotted red; }

атрибут: ширина_линии тип_линии цвет_линии код

Обтекание блока текста

Под ***обтеканием блока текстом*** понимают тот же эффект, который можно реализовать для графики, когда картинка не разрывает блок текста, а встраивается в него. Текст в этом случае "обтекает" картинку с одной стороны — там, где есть свободное поле между границей страницы (элемента) и картинкой. Обтекание картинки текстом от обычного встраивания картинки в текст документа отличается тем, что вдоль вертикальной границы картинки располагается несколько строк текста, а не одна.

Обтеканием блока текста другим текстом управляют два атрибута CSS: `float` и `clear`.

Атрибут `float` определяет "плавающий" блок текста. Он может принимать значения:

left — блок прижат к левой границе охватывающего элемента;

right — блок прижат к правой границе охватывающего элемента;

both — текст может обтекать блок с обеих сторон.

2.2. Применение CSS в HTML–документе. Свойства шрифты, тексты, цвет.

Управление цветом в CSS

Каскадные таблицы стилей (CSS) в первую очередь описывают свойства текста. Это касается как текстовых блоков, так и строковых элементов разметки содержания страницы. В данном разделе речь пойдет об управлении отображением *цвета текста* (*color*) и *цвета фона* (`background-color`), на котором отображается текст.

Кроме *цвета текста* и *цвета фона* CSS позволяет определять цвет границы текстового блока (`border-color`).

Атрибуты стилей, которые мы собираемся рассмотреть, согласно спецификации Microsoft, относятся к группе атрибутов *Color* and Background Properties. Всего в эту группу входит семь атрибутов, шесть из которых определяют свойства фона. Кроме *цвета фона* и его прозрачности, можно управлять фоновой картинкой (координатами ее размещения и способами повторения). К сожалению, Netscape Navigator большинство из этих атрибутов не поддерживает, поэтому мы не будем рассматривать их подробно.

Интерпретация атрибутов цвета в Netscape Navigator и Internet Explorer различна. В Netscape Navigator фоновый цвет отображается только там, где есть текст, а в Internet Explorer фоновый цвет заливает весь блок или строковый элемент вне зависимости от наличия в нем текста.

Цвет текста

В HTML для управления цветом отображаемого текста используется элемент `FONT`. Его аналогом в CSS является атрибут *color*. Этот атрибут можно применять как для блочных, так и для строковых элементов разметки.

Цвет фона текста

В HTML *цветом фона* можно управлять только для конкретного блочного элемента разметки. Таким элементом может быть вся страница:

<BODY BGCOLOR=...>...</BODY>

Или, например, таблица:

<TABLE BGCOLOR=...>...</TABLE>

Шрифт. *Шрифтам* в компьютерной графике всегда уделялось много внимания, и World Wide Web не является исключением. Но все богатство и разнообразие существующих *шрифтов* для русского языка ограничено фактически тремя *шрифтами*: `serif` (обычно `Times` или другой *шрифт* с засечками), `sans-serif` (`Arial`, `Helvetica` или другой *шрифт* без засечек) и `monospace` (`Courier`). Если быть точным, то здесь перечислены семейства *шрифтов*. Обычно каждое из этих семейств представлено только одним кириллическим *шрифтом*.

Автор документа для управления отображением букв может применить несколько атрибутов, влияющих на *шрифт*:

font-family - семейство *начертаний шрифта* (*гарнитура*);

font-style - прямое *начертание* или курсив;

font-weight - "усиление" (насыщенность) *шрифта*, "жирность" букв;

font-size - *размер шрифта* (***кегль***). Задается в *пикселах (px)* и *типографских пунктах (pt)*.

font-variant - вариант *начертания* (обычный или мелкими буквами - капитель).

Все эти параметры можно совместить в одном атрибуте `font`:

```
font:bold 12pt sans;
```

Правда, нет никакой уверенности в том, что последнее определение *шрифта* будет работать во всех браузерах.

При использовании различных *гарнитур* (`font-family`) следует помнить, что наличие или отсутствие необходимой автору *гарнитуры* всецело зависит от предпочтений пользователя. Для кириллицы это может вылиться в появление абракадабры там, где автор применяет отсутствующие у пользователя *шрифты*.

Самое неприятное, с чем можно столкнуться при использовании *шрифтов* - это несоответствие моноширинных *шрифтов*, которые применяются в HTML-формах. Обратная связь с пользователем в этом случае невозможна.

Спецификация CSS предусматривает перечисление *шрифтов* в описаниях стилей, что позволяет частично решить проблему подбора *шрифта*. К сожалению, в Unix и Windows *шрифты* не согласованы. Фактически, при разработке страниц в CSS используются только классы *шрифтов* (`serif`, `sans-serif` и `monospace`).

Гарнитура (font-family)

Гарнитура *шрифта* - это набор *начертаний* одного *шрифта*. *Шрифт* может иметь "прямое" *начертание* (`normal`), курсив (`italic`), "скошенное" (`oblique`), усиленное по насыщенности ("жирное", `bold`), "мелкое" (капитель, `small-caps`) и т.п.

Наиболее распространенные *гарнитуры* в российской части Web - это `Times`, `Arial`, `Courier`. Причем все они принадлежат к разным группам *шрифтов*. `Times` - это пропорциональный *шрифт* "с засечками" (`serif`), `Arial` - это пропорциональный *шрифт* "без засечек" (`sans-serif`), а `Courier` - это моноширинный *шрифт* (`monospace`). В Unix вместо `Arial` чаще применяется `Helvetica`.

Кегль (font-size)

Кегль - это, если говорить упрощенно, *размер шрифта*. Более подробное объяснение следует искать в специальной литературе. Нам достаточно знать, что CSS через параметр `font-size` позволяет управлять размером букв.

Размер шрифта можно задавать в *типографских пунктах (pt, 0,35 мм)* или *пикселах (px)*. При установке *кегля* следует помнить, что `font-size` задает не высоту буквы, а размер "очка" под букву, который больше самой буквы.

Начертание

У каждой *гарнитуры* (`font-family`) имеется несколько *начертаний*. Каждое из них определяется в CSS тремя параметрами стиля: `font-style`, *font-variant*, `font-weight`.

Текст

В этом разделе мы рассмотрим те свойства текстового фрагмента, которые остались без внимания в разделах, посвященных блокам *текста* и шрифтам.

При обсуждении свойств блочных элементов разметки речь шла о параметрах, относящихся к блоку как целому. Мы не рассматривали внутренние характеристики *текста*.

Рассказывая о шрифтах, мы акцентировали внимание на начертаниях символов как таковых, а не на их соотношении.

Межбуквенные расстояния

Расстояние между буквами автоматически регулируется размером шрифта — кеглем. Чем больше размер шрифта, тем больше расстояние между буквами

Выравнивание

По умолчанию все слова в параграфе прижаты влево. Левый край параграфа таким образом оказывается выравненным. Точно так же может быть выравнен правый край параграфа или блока *текста*, и даже оба края вместе.

В обычной HTML-разметке такой эффект достигается за счет применения атрибута `ALIGN`, как это сделано на страницах данного пособия:

```
<P ALIGN=justify>...</P>
```

Преобразование шрифта

Преобразование шрифта подразумевает капитализацию слов, перевод всех "больших" и "маленьких" букв в большие, или, наоборот, получение одних строчных.

Первая строка параграфа

При оформлении параграфов в технологии CSS автор может воспользоваться "красной" строкой, такую возможность предоставляет ему атрибут `text-indent`.

Межстрочное расстояние

В CSS ***межстрочное расстояние*** определяется параметром `line-height`. Он задает расстояние не между строками, а между базовыми линиями строк. Проще говоря, если, например, взять букву "н" и напечатать ее последовательно друг под другом, то `line-height` будет равно расстоянию между двумя одинаковыми точками букв.

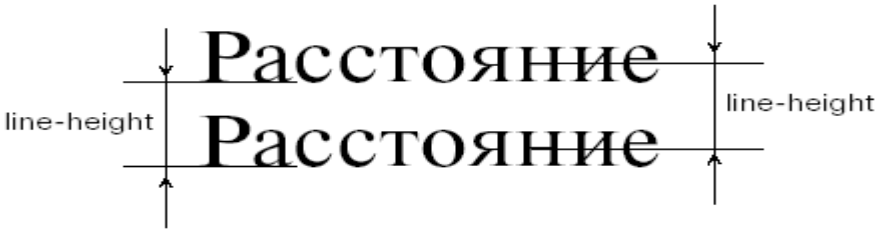


Рис. 11.9.

Списки

При отображении *списков* CSS позволяет управлять формой и *изображением "пулек" (bullets)* списка. "Пулька" (bullet) — это символ, стоящий перед элементом списка. Например, в неупорядоченном списке (`unordered list`) перед элементом списка ставится "жирная" точка:

Первый элемент списка

Второй элемент списка

Третий элемент списка

CSS позволяют управлять формой "пулек" и заменять "пульки" картинками.

Любопытно, что управление отображением элементов списка отнесено к набору свойств, в который входит атрибут `display`. У этого атрибута может быть только одно значение — `none`. Если элемент в своем описании имеет атрибут `display`, и этот атрибут равен `none`, то он не отображается браузером вообще:

```
<UL STYLE="display:none;">
<LI>Первый элемент списка
<LI>Второй элемент списка
<LI>Третий элемент списка
</UL>
```

Если посмотреть HTML-код данного документа, то за примером описания списка следует код, который браузер не отобразил.

Атрибут `display` управляет отображением документа на дисплее компьютера, но не распространяется на другие среды отображения документа. Например, при печати скрытый список должен быть отображен.

Однако, на самом деле он не отображается и при печати.

Форма "пулек"

Форма "пульки" в виде "жирной" точки несколько непривычна. Обычно в машинописных документах используют черту. С другой стороны, в рекламных материалах часто в качестве "пульки" применяют квадрат или другой символ типографского набора, а также графическую картинку.

CSS позволяет управлять формой "пульки" через атрибут `list-style-type.`

"Пульки"-картинки

Если стандартные формы "пулек" автора страницы не устраивают, он может использовать нестандартные. Для этого ему придется "пульку" нарисовать самому и в виде графического файла разместить на Web-узле. У такой "пульки" есть URL, который используется в CSS для обращения к ней.

```
<UL STYLE="list-style-image:url(bimage.gif);"
> <LI>Элемент списка расположен за чертой
</UL>
```

Картинка может быть и более замысловатой. Это уже зависит от фантазии автора документа. Например, можно создать картинку-стрелочку

Координаты и размеры

Стандарт *CSS-P* позволяет с точностью до пиксела разместить блочный элемент разметки в рабочем поле окна браузера. При таком подходе возникает естественный вопрос: как устроена система *координат*, в которой автор страницы производит размещение ее компонентов.

CSS-P определяет две системы *координат*: *относительную* и *абсолютную*. Это позволяет обеспечить гибкость размещения элементов как относительно границ рабочего поля окна браузера, так и относительно друг друга.

Блоки - это не абстрактные точки, которые не занимают на плоскости страницы места. Блоки представляют собой прямоугольники, которые "заметают" площадь. Текст и другие компоненты страницы под блоком становятся недоступны пользователю, поэтому *линейные размеры* блока имеют для создания HTML-страниц не меньшее значение, чем его *координаты*.

Абсолютные координаты

При использовании "***абсолютных***" *координат* точка отсчета помещается в верхний левый угол окна браузера, а оси X и Y направлены вправо по горизонтали и вниз по вертикали, соответственно:

Если в этой системе *координат* некоторый блочный элемент должен быть размещен на `10 px` ниже верхнего обреза рабочей области браузера и на `20 px` правее левого края рабочей области браузера, то его описание будет выглядеть следующим образом:

```
.example { position:absolute;top:10px; left:20px; }
```

В данной записи тип системы *координат* задан атрибутом `position` (значение - `absolute`), *координата* X задана атрибутом `left` (значение - `20 px`), *координата* Y - атрибутом `top` (значение - `10 px`).

Атрибуты `top` и `left` определяют *координаты* верхнего левого угла блока в *абсолютной* системе *координат*.

Относительные координаты

Данная *координатная система* позволяет разместить блоки на странице в *координатах* охватывающего их блока. Преимущества такой системы *координат* очевидны: она позволяет сохранять взаимное расположение элементов разметки при любом размере окна браузера и его настройках по умолчанию.

В качестве точки отсчета в этой ***системе координат*** выбрана точка размещения текущего блока по умолчанию. Ось X при этом направлена горизонтально вправо, а ось Y - вертикально вниз.

Для работы с *относительной системой координат* лучше пользоваться универсальными блоками `DIV`. Это связано с тем, что в Netscape Navigator, например, параграф не может содержать параграфов. Любой блок немедленно закрывает параграф, следовательно, вложить в него что-либо нельзя.

Следует отметить, что Netscape Navigator вообще непредсказуем в работе с *относительными координатами*, поэтому в нем следует их избегать.

В *относительной системе координат* можно пользоваться отрицательными смещениями.

Линейные размеры блока

Линейные размеры блока задаются двумя параметрами: шириной (`width`) и высотой (`height`) блока. В браузерах ширина и высота блока интерпретируется по-разному.

В Netscape Navigator ширина и высота блока - рекомендуемые параметры. Если текст выходит за эти ограничения, то блок увеличивается до необходимых размеров, а если текста нет вообще, то блок сжимается до маленького квадрата:

```
<P STYLE="width:200px;height:100px;
background-color:black;color:white;">
...
</P>
```

Без границы блок не будет залит черным цветом, а без `span` текст будет отображаться цветом данной страницы по умолчанию. Никакому разумному объяснению такое поведение браузера не поддается, поэтому строить дизайн страниц на этих атрибутах не стоит.

Литература

1. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. Основы web-технологий БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2007
2. Cascading Style Sheets, level 1 (CSS1) W3C Recommendation 17 Dec 1996, revised 11 Jan 1999.
3. Hakon Wium Lie и Bert Bos. Каскадные таблицы стилей, уровень 1 Перевод Всероссийского Клуба Вебмастеров Перевод спецификации CSS1. 2007.

ТЕМА 8-9. ВВОД В JAVASCRIPT (4 ЧАСОВ)

План.

3.1. Применение в HTML – документы скрипты JavaScript.

3.2. Типы, переменные, выражение, арифметические операторы в JavaScripte.

3.3. Управляющей операторы JavaScript. Функции и методы. Объекты и свойства.

3.1. Применение в HTML – документы скрипты JavaScript.

Размещение кода на HTML-странице

В общем случае можно выделить четыре способа функционального применения *JavaScript*:

- гипертекстовая ссылка (схема URL);
- обработчик события (handler);
- подстановка (entity) (в Microsoft Internet Explorer реализована в версиях от 5.X и выше);
- вставка (контейнер SCRIPT).

URL-схема JavaScript

Схема URL (Uniform Resource Locator) - это один из основных элементов Web-технологии. Каждый информационный ресурс в Web имеет свой уникальный URL. URL указывают в атрибуте HREF контейнера A, в атрибуте SRC контейнера IMG, в атрибуте ACTION контейнера FORM и т.п. Все URL подразделяются на схемы доступа, которые зависят от протокола доступа к ресурсу, например, для доступа к FTP-архиву применяется схема ftp, для доступа к Gopher-архиву - схема gopher, для отправки электронной почты - схема smtp. Тип схемы определяется по первому компоненту URL: http://intuit.ru/directory/page.html

В данном случае URL начинается с http - это и есть определение схемы доступа (схема http).

Основной задачей языка программирования гипертекстовой системы является программирование гипертекстовых переходов. Это означает, что при выборе той или иной гипертекстовой ссылки вызывается программа реализации гипертекстового перехода. В Web-технологии стандартной программой является программа загрузки страницы. *JavaScript* позволяет поменять стандартную программу на программу пользователя. Для того чтобы отличить стандартный переход по протоколу HTTP от перехода, программируемого на *JavaScript*, разработчики языка ввели новую схему URL - *JavaScript*:

```
<A HREF="JavaScript:JavaScript_код">...</A>
<IMG SRC="JavaScript:JavaScript_код">
```

В данном случае текст "JavaScript_код" обозначает программы-обработчики на *JavaScript*, которые вызываются при выборе гипертекстовой ссылки в первом случае и при загрузке картинки - во втором.

Например, при нажатии на гипертекстовую ссылку **Внимание!!!** можно получить окно предупреждения:

```
<A HREF="JavaScript:alert('Внимание!!!');"> Внимание!!!</A>
```

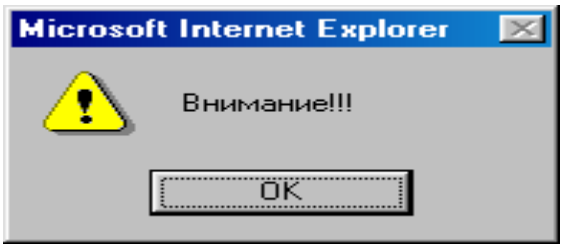


Рис. 1.

А при нажатии на кнопку типа submit в форме можно заполнить текстовое поле этой же формы:

```
<FORM NAME=f METHOD=post
```

```
    ACTION="JavaScript:window.document.f.i.VALUE='Нажали кнопку Click';
void(0);">
<TABLE BORDER=0>
<TR>
<TD><INPUT NAME=i></TD>
<TD><INPUT TYPE=submit VALUE=Click></TD>
<TD><INPUT TYPE=reset VALUE=Reset></TD>
</TABLE>
</FORM>
```

Обработчики событий

Такие программы, как обработчики событий (handler), указываются в атрибутах контейнеров, с которыми эти события связаны. Например, при нажатии на кнопку происходит событие click:

```
<FORM><INPUT TYPE=button VALUE="Кнопка"
onClick="window.alert('intuit');"></FORM>
```

Подстановки

Подстановка (entity) встречается на Web-страницах довольно редко. Тем не менее это достаточно мощный инструмент генерации HTML-страницы на стороне браузера. Подстановки используются в качестве значений атрибутов HTML-контейнеров. Например, как значение по умолчанию поля формы, определяющего домашнюю страницу пользователя, будет указан URL текущей страницы:

```
<SCRIPT>
function l()
{
  str = window.location.href;
  return(str.length);
}
```

```
</SCRIPT>
<FORM><INPUT VALUE="&{window.location.href};" SIZE="&{l()};">
</FORM>
<SCRIPT>
<!-- Это комментарий ...JavaScript-код...// -->
</SCRIPT>
<BODY>
... Тело документа ...
</BODY>
</HTML>
```

HTML-комментарии здесь вставлены для защиты от интерпретации данного фрагмента страницы HTML-парсером в старых браузерах (у высокого начальства еще встречаются). В свою очередь, конец HTML-комментария защищен от интерпретации *JavaScript*-интерпретатором (// в начале строки). Кроме того, в качестве значения атрибута LANGUAGE у тега начала контейнера указано значение "*JavaScript*". VBScript, который является альтернативой *JavaScript* - это скорее экзотика, чем общепринятая практика, поэтому данный атрибут можно опустить - значение "*JavaScript*" принимается по умолчанию.

Очевидно, что размещать в заголовке документа генерацию текста страницы бессмысленно - он не будет отображен браузером. Поэтому в заголовок помещают декларации общих переменных и функций, которые будут затем использоваться в теле документа. При этом браузер Netscape Navigator более требовательный, чем Internet Explorer. Если не разместить описание функции в заголовке, то при ее вызове в теле документа можно получить сообщение о том, что данная функция не определена.

Приведем пример размещения и использования функции:

```
<HTML>
<HEAD>
```

```
<SCRIPT>
function time_scroll()
{
  var d = new Date();
  window.status = d.getHours()+":"+d.getMinutes()+":"+d.getSeconds();
  setTimeout('time_scroll();',500);
}
</SCRIPT>
</HEAD>
<BODY onLoad=time_scroll()>
<CENTER>
<H1>Часы в строке статуса</H1>
```

В Internet Explorer 4.0 подстановки не поддерживаются, поэтому пользоваться ими следует аккуратно. Прежде чем выдать браузеру страницу с подстановками, нужно проверить тип этого браузера.

Вставка (контейнер SCRIPT - принудительный вызов интерпретатора)

Контейнер SCRIPT - это развитие подстановок до возможности генерации текста документа *JavaScript*-кодом. В этом смысле применение SCRIPT аналогично Server Side Includes, т.е. генерации страниц документов на стороне сервера. Однако здесь мы забежали чуть вперед. При разборе документа HTML-парсер передает управление интерпретатору после того, как встретит тег начала контейнера SCRIPT. Интерпретатор получает на исполнение весь фрагмент кода внутри контейнера SCRIPT и возвращает управление HTML-парсеру для обработки текста страницы после тега конца контейнера SCRIPT.

Контейнер SCRIPT выполняет две основные функции:

- **размещение кода** внутри HTML-документа;
- **условная генерация** HTML-разметки на стороне браузера.

Первая функция аналогична декларированию переменных и функций, которые потом можно будет использовать в качестве программ переходов, обработчиков событий и подстановок. Вторая - это подстановка результатов исполнения *JavaScript*-кода в момент загрузки или перезагрузки документа.

Размещение кода внутри HTML-документа

Собственно, особенного разнообразия здесь нет. Код можно разместить либо в заголовке документа, внутри контейнера HEAD, либо внутри BODY. Последний способ и его особенности будут рассмотрены в разделе "Условная генерация HTML-разметки на стороне браузера". Поэтому обратимся к заголовку документа.

Код в заголовке размещается внутри контейнера SCRIPT:

```
<HTML>
<HEAD>
<SCRIPT>
function time_scroll()
{
  d = new Date();
  window.status = d.getHours()+":"+d.getMinutes()+":"+d.getSeconds();
  setTimeout('time_scroll();',500);
}
</SCRIPT>
</HEAD>
<BODY onLoad=time_scroll()>
<CENTER>
<H1>Часы в строке статуса</H1>
<FORM>
<INPUT TYPE=button VALUE="Закрыть окно" onClick=window.close()>
</FORM>
</CENTER>
```

</BODY>
</HTML>

В этом примере мы декларировали функцию time_scroll() в заголовке документа, а потом вызвали ее как обработчик события load в теге начала контейнера BODY (onLoad=time_scroll()) .

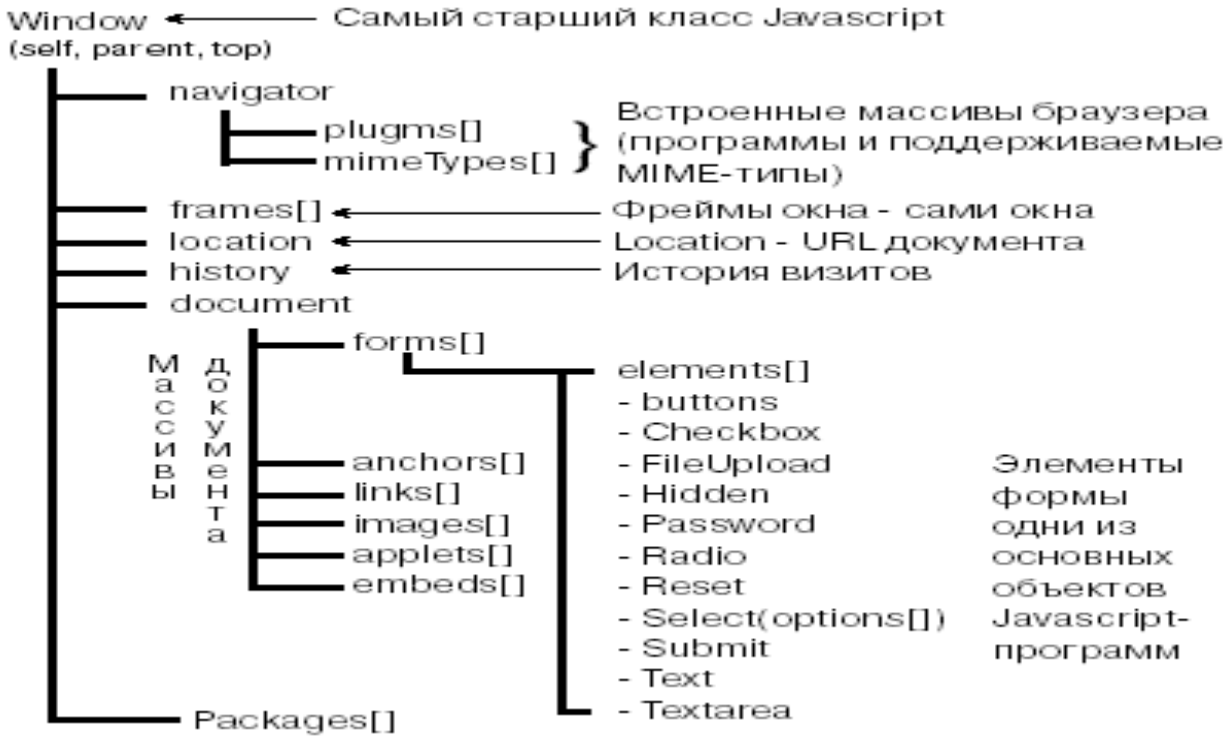
Условная генерация HTML-разметки на стороне браузера

Всегда приятно получать с сервера страницу, подстроенную под возможности нашего браузера или, более того, под пользователя. Существует только две возможности генерации таких страниц: на стороне сервера или непосредственно у клиента. *JavaScript*-код исполняется на стороне клиента (на самом деле, серверы компании Netscape способны исполнять *JavaScript*-код и на стороне сервера, только в этом случае он носит название LiveWire-код; не путать с LiveConnect), поэтому рассмотрим только генерацию на стороне клиента.

Для генерации HTML-разметки контейнер SCRIPT размещают в теле документа.

Иерархия классов

Объектно-ориентированный язык программирования предполагает наличие иерархии классов *объектов*. В *JavaScript* такая иерархия начинается с класса *объектов* Window, т.е. каждый *объект* приписан к тому или иному окну. Для обращения к любому *объекту* или его *свойству* указывают полное или частичное имя этого *объекта* или *свойства объекта*, начиная с имени *объекта* старшего в иерархии, в который входит данный *объект*:



Сразу оговоримся, что приведенная нами схема объектной модели верна для Netscape Navigator версии 4 и выше, а также для Microsoft Internet Explorer версии 4 и выше. Еще раз отметим, что объектные модели у Internet Explorer и Netscape Navigator совершенно разные, а приведенная схема составлена на основе их общей части.

Вообще говоря, *JavaScript* не является классическим объектным языком (его еще называют облегченным объектным языком). В нем нет наследования и полиморфизма. Программист может определить собственный класс *объектов* через оператор function, но чаще пользуется стандартными *объектами*, их конструкторами и вообще не применяет деструкторы классов. Это объясняется тем, что область действия *JavaScript*-программы обычно не распространяется за пределы текущего окна.

Иногда у разных *объектов* *JavaScript* бывают определены *свойства* с одинаковыми именами. В этом случае нужно четко указывать, *свойство* какого *объекта* программист хочет использовать. Например, Window и Document имеют *свойство* location. Только для Window это *объект* класса Location, а для Document -

строковый литерал, который принимает в качестве значения URL загруженного документа.

Следует также учитывать, что для многих *объектов* существуют стандартные *методы* преобразования значений *свойств* *объектов* в обычные переменные. Например, для всех *объектов* по умолчанию определен *метод* преобразования в строку символов: toString(). В примере с location, если обратиться к window.location в строковом контексте, будет выполнено преобразование по умолчанию, и программист этого не заметит:

```
<SCRIPT>
document.write(window.location);
document.write("<BR>");
document.write(document.location);
</SCRIPT>
```

Однако разница все-таки есть, и довольно существенная. В том же примере получим длины строковых констант:

```
<SCRIPT>
w=toString(window.location);
d=toString(document.location);
h=window.location.href;
document.write(w.length);
document.write(d.length);
document.write(h.length);
</SCRIPT>
```

Результат исполнения получите сами.

Как легко убедиться, при обращении к *свойству* *объекта* типа URL, а *свойство* location как раз является *объектом* данного типа, длина строки символов после преобразования будет другой.

Класс объектов Window — это самый старший класс в иерархии объектов JavaScript. К нему относятся объект Window и объект Frame. Объект Window ассоциируется с **окном** программы-браузера, а объект Frame — с окнами внутри окна браузера, которые порождаются последним при использовании автором HTML-страниц контейнеров FRAMESET и FRAME.

При программировании на JavaScript чаще всего используют следующие свойства и методы объектов типа Window:

Таблица 14.1.

Свойства	Методы	События
status	open()	Событий нет
location	close()	
history	focus()	
navigator		

Объект Window создается только в момент открытия окна. Все остальные объекты, которые порождаются при загрузке страницы в *окно*, есть свойства объекта Window. Таким образом, у Window могут быть разные свойства при загрузке разных страниц.

3.2. Типы, переменные, выражение, арифметические операторы в JavaScripte.

Типы и структуры данных

Как и любой другой язык программирования, JavaScript поддерживает встроенные *типы и структуры данных*. Все их многообразие подразделяется на:

- литералы и переменные;
- массивы, *функции* и *объекты*.

При этом все они делятся на встроенные и определяемые программистом. *Функции* и *объекты* рассматриваются в разделах "*Функции*" и "*Объекты*". Поэтому здесь мы остановимся на литералах, переменных и массивах.

Литералы

Литералом называют данные, которые используются в программе непосредственно. При этом под данными понимаются числа или строки текста. Все они рассматриваются в JavaScript как элементарные *типы данных*. Приведем примеры литералов:

```
числовой литерал: 10
числовой литерал: 2.310
числовой литерал: 2.3e+2
строковый литерал: 'Это строковый литерал'
строковый литерал: "Это строковый литерал"
```

Литералы используются в операциях присваивания значений переменным или в операциях сравнения:

```
var a=10;
var str = 'Строка';
if(x=='test') window.alert(x);
```

Два варианта строковых литералов необходимы для того, чтобы использовать вложенные строковые литералы. Вообще говоря, есть подозрение, что равноправие "..." и '...' мнимое. Если внимательно посмотреть на реализацию страниц программирования гипертекстовых ссылок (href.htm, path.htm и mouse.htm), можно заметить, что вместо прямого переназначения гипертекстовой ссылки литералом типа '...' там используется косвенное переназначение через *функцию* литералом "...":

```
...
function line(a)
```

```
{
...
window.document.main.document.links[4].href=
"javascript:data(0);void(0);";
...
}
...
<A HREF="javascript:line(0);void(0);">
<IMG SRC=image.gif BORDER=0>
</A>
```

вместо:

```
<A HREF="javascript:
window.document.main.document.links[4].href=
'javascript:data(0);void(0);';void(0);">
<IMG SRC=image.gif BORDER=0>
</A>
```

Это связано с особенностями реализации Netscape. Дело в том, что прямое переназначение неправильно отображает кириллицу в win32, а вот косвенное работает. Похоже, что "..." разрешает анализ информации внутри строкового литерала JavaScript-интерпретатором, а '...' — нет.

Если быть более точным, то следует сказать, что строка — это *объект*. У этого *объекта* существует великое множество методов. Строчный литерал и строчный *объек*т — далеко не одно и то же. При применении к строчным литералам методов строчных *объектов* происходит преобразование первых в последние.

Переменные

Переменные в JavaScript могут быть определены назначением или при помощи оператора var:

```
i=10;
var i;
var i=10;
var id = window.open();
var a = new Array();
```

Как видно из примеров, переменные могут принимать самые разные значения, при этом тип переменной определяется контекстом.

3.3. Управляющей операторы JavaScript. Функции и методы. Объекты и свойства.

Операторы языка

В этом разделе будут рассмотрены операторы JavaScript. Основное внимание при этом мы уделим операторам декларирования и управления потоком вычислений. Без них не может быть написана ни одна JavaScript-программа.

Общий перечень этих операторов выглядит следующим образом:

- var;
- {...};
- if;
- while;
- for;
- for ... in;
- break;
- continue;
- return.

Сразу оговоримся, что этот список неполный.

var

Оператор var служит для объявления переменной. При этом переменная может принимать значения любого из разрешенных типов данных. На практике довольно часто обходятся без явного использования var. Переменная соответствующего типа создается путем простого присваивания:

```
var a;
var a=10;
var a = new Array();
var a = new Image();
```

Все перечисленные выше примеры использования var верны и могут быть применены в JavaScript-программе. Область действия переменной определяется блоком (составным оператором), в котором используется переменная. Максимальная область действия переменной — страница.

{...}

Фигурные скобки определяют составной оператор JavaScript — блок. Они одновременно ограничивают область действия переменных, которые определены внутри этих скобок. За пределами блока переменные не видны:

```
{
var i=0;
}
```

Основное назначение блока — определение тела цикла и тела условного оператора.

if

Условный оператор применяется для ветвления программы по некоторому логическому условию. Общий синтаксис:

```
if (логическое выражение) оператор1;
[else оператор2;]
```

Логическое выражение — это выражение, которое принимает значение true или false. Если оно равно true, то оператор 1 исполняется. В квадратных скобках необязательная составляющая оператора if — альтернатива основной ветви вычислений:

```
if (navigator.appName=="Netscape")
{
window.location.href=
"http://intuit.ru/netscape.htm";
}
else
{
window.location.href=
"http://intuit.ru/explorer.htm";
}
```

Примеры использования условного оператора можно найти, например, в разделе "Тип браузера".

while

Оператор while определяет цикл. Определяется он в общем случае следующим образом:

```
While (логическое выражение)
оператор;
```

Оператор, в том числе и составной, — тело цикла. Тело исполняется до тех пор, пока верно логическое условие:

```
while (flag==0)
{
id=setTimeout ("test();",500);
}
```

Обычно цикл этого типа применяют при выполнении периодических действий до некоторого события.

for

Оператор for — это еще один оператор цикла. В общем случае он имеет вид:

```
for (инициализация переменных цикла;
    условие; модификация переменных цикла)
оператор;
```

Оператор в теле цикла может быть блоком. Рассмотрим типичный пример использования этого оператора:

```
for(i=0;i<document.links.length;i++)
{
document.write(document.links[i].href+"<BR>");
}
http://intuit.ru/help/index.html
http://intuit.ru/help/shop.html#choice
http://intuit.ru/help/payment.html
```

Подобные примеры разбросаны по всем разделам курса.

for ... in

Данный оператор позволяет "пробежаться" по свойствам *объекта*. Рассмотрим пример:

```
for(v in window.document)
{
document.write(v+"<BR>");
}
```

Все свойства текущего *объекта* "документ":

break

Оператор break позволяет досрочно покинуть тело цикла. Распечатаем только title документа:

```
for(v in window.document)
if(v=="title")
{
document.write(v+":"+eval('document.'+v)+"
");
break;
}
```

Результат исполнения:

```
title:Web-engineering
(Introduction to the JavaScript. Operators.).
```

В пример распечатки свойств *объекта* document мы вставили break при просмотре свойства title и получили искомый результат.

continue

Того же результата, что и при использовании break, можно было бы достичь при помощи оператора continue:

```
for(v in window.document)
```

```
{
if(v!="title") continue;
document.write(v+":"+eval('document.'+v));
break;
}
```

Результат исполнения:

```
title:Web-engineering
(Introduction to the JavaScript. Operators.)
```

Этот оператор позволяет пропустить часть тела цикла (от оператора до конца тела) и перейти к новой итерации. Таким образом мы просто пропускаем все свойства до title и после этого выходим из цикла.

return

Оператор return используют для возврата значения из *функции* или обработчика события (см. разделы "Поле статуса", "*Обмен данными*"). Рассмотрим пример:

```
<FORM>
<INPUT TYPE=submit VALUE=Submit
onClick="return false;">
</FORM>
```

В данном примере return используется для маскирования передачи данных на сервер.

Управление фокусом

Фокус — это характеристика текущего окна, фрейма или поля формы. В каждом из разделов, описывающем программирование этих *объектов*, мы, так или иначе, касаемся вопроса фокуса. Под фокусом понимают возможность активизации свойств и методов *объекта*. Например, окно в фокусе, если оно является текущим, т.е. лежит

поверх всех других окон и исполняются его методы или можно получить доступ к его свойствам.

В данном разделе мы рассмотрим управление фокусом в

- окнах;
- фреймах;
- полях формы.

Следует сразу заметить, что фреймы — это тоже *объекты* класса Window, и многие решения, разработанные для окон, справедливы и для фреймов.

Функции

Язык программирования не может обойтись без механизма многократного использования кода программы. Такой механизм обеспечивается *процедурами* или *функциями*. В JavaScript *функция* выступает в качестве одного из основных *типов данных*. Одновременно с этим в JavaScript определен *объект* Function.

В общем случае любой *объект* JavaScript определяется через *функцию*. Для создания *объекта* используется конструктор, который в свою очередь вводится через Function. Таким образом, с *функциями* в JavaScript связаны следующие ключевые вопросы:

- *функция — тип данных*;
- *функция — объект*;
- конструкторы *объектов*.

Именно эти вопросы мы и рассмотрим в данном разделе.

Функция — тип данных

Определяют *функцию* при помощи ключевого слова function:

function f_name(arg1,arg2,...)

```
{
/* function body */
}
```

Здесь следует обратить внимание на следующие моменты. Во-первых, function определяет переменную f_name. Эта переменная имеет тип "function":

```
document.write("Тип переменной f_name:"+
        typeof(f_name));
```

Тип переменной f_name: function. Во-вторых, этой переменной присваивается значение:

```
document.write("Значение i:"+i.valueOf());
document.write("Значение f_name:"+
        f_name.valueOf());
```

Значение переменной f_name: 10. Значение переменной f_name:function f_name(a) { if(a>=0) return true; else return false; }. В данном случае метод valueOf() применяется как к числовой переменной i, так и к f_name. По этой причине *функции* можно назначить синоним путем присваивания ее значения другой переменной:

```
function f_name(a)
{
if(a>=0) return true; else return false;
}
document.write("Значение переменной f_name:"+
        f_name(1)+"");
b = f_name;
document.write("Значение переменной b:"+
        b(1)+"");
Значение переменной f_name:true
Значение переменной b:true
```

Функция — объект

У любого *типа данных* JavaScript существует *объектовая* "обертка" — Wrapper, которая позволяет применять методы *типов данных* к переменным и литералам, а также получать значения их свойств. Например, длина строки символов определяется свойством length. Аналогичная "обертка" есть и у *функций — объект* Function.

Например, увидеть значение *функции* можно не только при помощи метода valueOf(), но и используя метод toString():

```
function f_name(x,y)
{
return x-y;
}
document.write(f_name.toString()+"<br>");
```

Результат распечатки:

```
function f_name(x,y) { return x-y; }
```

Свойства *функции* доступны для программиста только тогда, когда они вызываются внутри *функции*. При этом обычно программисты имеют дело с массивом аргументов *функции* (arguments[]), его длиной (length), именем *функции*, вызвавшей данную *функцию* (caller)и *прототипом* (prototype).

Объекты

Объект — это главный *тип данных* JavaScript. Любой другой *тип данных* имеет *объектовую* "обертку" — Wrapper. Это означает, что прежде чем можно будет получить доступ к значению переменной того или иного типа, происходит конвертирование переменной в *объект*, и только после этого выполняются действия над значением. *Тип данных* Object сам определяет *объекты*.

В данном разделе мы остановимся на трех основных моментах:

- понятие *объекта*;
- *прототип объекта*;
- методы *объекта* Object.

Мы не будем очень подробно вникать во все эти моменты, так как при программировании на стороне браузера чаще всего обходятся встроенными средствами JavaScript. Но поскольку все эти средства — *объекты*, нам нужно понимать, с чем мы имеем дело.

Понятие объекта

Объект — это совокупность свойств и методов, доступ к которым можно получить, только создав при помощи конструктора *объект* данного класса и использовав его контекст.

На практике довольно редко приходится иметь дело с *объектами*, созданными программистом. Дело в том, что *объект* создается *функцией*-конструктором, которая определяется на конкретной странице и, следовательно, все, что создается в рамках данной страницы, не может быть унаследовано другими страницами. Нужны очень веские основания, чтобы автор Web-узла занялся разработкой библиотеки классов *объектов* пользователя. Гораздо проще писать *функции* для каждой страницы.

Прототип

Обычно мы имеем дело со встроенными *объектами* JavaScript. Собственно, все, что изложено в других разделах курса — это обращение к свойствам и методам встроенных *объектов*. В этом смысле интересно свойство *объектов*, которое носит название prototype. Прототип — это другое название конструктора *объекта* конкретного класса.

Литератра

1. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. **Основы web-технологий** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2007
2. Капустин М.А., Капустин П.А., Копылова А.Г. **Flash MX для профессиональных программистов** Интернет-университет информационных технологий - ИНТУИТ.ру, 2006
3. Савельева Н.В. **Основы программирования на PHP** Интернет-университет информационных технологий - ИНТУИТ.ру, 2005
4. Шохирев М.В. **Язык программирования Perl 5** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006
5. Сузи Р.А. **Язык программирования Python** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006

ТЕМА 10. ОБЪЕКТНЫЕ МОДЕЛИ HTML-ДОКУМЕНТА В JAVASCRIPTE (4 ЧАСОВ)

План

4.1. Объект windows, document и т.д. свойства и методы.

4.2. Обработка случаи.

4.3. Интерактивной форма.

4.1. Объект windows, document и т.д. свойства и методы. Обработка собитии.

Поле location

В поле location отображается URL загруженного документа. Если пользователь хочет вручную перейти к какой-либо странице (набрать ее URL), он делает это в поле location. Поле располагается в верхней части окна браузера ниже панели инструментов, но выше панели личных предпочтений.

Вообще говоря, Location — это объект. Из-за изменений в версиях JavaScript класс Location входит как подкласс и в класс Window, и в класс Document. Мы будем рассматривать Location только как window.location. Кроме того, Location — это еще и подкласс класса URL, к которому относятся также объекты классов Area и Link. Location наследует все свойства URL, что позволяет получить доступ к любой части схемы URL.

Рассмотрим характеристики и способы использования объекта Location:

- свойства;
- методы;
- событий, характеризующих Location, нет.

Как мы видим, список характеристик объекта Location неполный.

Свойства

Предположим, что браузер отображает страницу, расположенную по адресу:

http://intuit.ru:80/r/dir/page?search#mark

Тогда свойства объекта Location примут следующие значения:

window.location.href = http://intuit.ru:80/r/dir/page?search#mark
window.location.protocol = http;
window.location.hostname = intuit.ru;
window.location.host = intuit.ru:80;
window.location.port = 80
window.location.pathname = /r/dir/;
window.location.search = search;
window.location.hash = mark;

Методы

Методы Location предназначены для управления загрузкой и перезагрузкой страницы. Это управление заключается в том, что можно либо перезагрузить документ (reload), либо загрузить (replace). При этом в историю просмотра страниц (history) информация не заносится:

window.location.reload(true);
window.location.replace('#top');

Метод reload() полностью моделирует поведение браузера при нажатии на кнопку Reload в панели инструментов. Если вызывать метод без аргумента или указать его равным true, то браузер проверит время последней модификации документа и загрузит его либо из кеша (если документ не был модифицирован), либо с сервера. Такое поведение соответствует простому нажатию на кнопку Reload. Если в качестве аргумента указать false, то браузер перезагрузит текущий документ с

сервера, несмотря ни на что. Такое поведение соответствует одновременному нажатию на Reload и кнопку клавиатуры Shift (Reload+Shift).

История посещений (History)

История посещений (трасса) страниц World Wide Web позволяет пользователю вернуться к странице, которую он просматривал несколько минут (часов, дней) назад. История посещений в JavaScript трансформируется в объект класса history. Этот объект указывает на массив URL-страниц, которые пользователь посещал и которые он может получить, выбрав из меню браузера режим GO. Методы объекта history позволяют загружать страницы, используя URL из этого массива.

Чтобы не возникло проблем с безопасностью браузера, путешествовать по History можно, только используя индекс URL. При этом URL, как текстовая строка, программисту недоступен. Чаще всего этот объект используют в примерах или страницах, на которые могут быть ссылки из нескольких разных страниц, предполагая, что можно вернутся к странице, из которой пример будет загружен:

```
<FORM><INPUT TYPE=button VALUE="Назад" onClick=history.back()></FORM>
```

Данный код отображает кнопку "Назад", нажав на которую мы вернемся на предыдущую страницу.

Управление окнами

Что можно сделать с окном? Открыть (создать), закрыть (удалить), положить его поверх всех других открытых окон (передать фокус). Кроме того, можно управлять свойствами окна и свойствами подчиненных ему объектов. Описанию основных свойств посвящен раздел "Программируем свойства окна браузера", поэтому сосредоточимся на простых и наиболее популярных методах управления окнами:

- alert();
- confirm();

- prompt();
- *open()*;
- *close()*;
- focus();
- *setTimeout()*;
- clearTimeout().

Здесь не указаны только два метода: scroll() и blur().

Первый позволяет прокрутить *окно* на определенную позицию. Но его очень сложно использовать, не зная координат окна. Последнее является обычным делом, если только не используется технология программирования слоев или CSS (Cascading Style Sheets).

Второй метод уводит фокус с окна. При этом совершенно непонятно, куда этот фокус будет передан. Лучше целенаправленно передать фокус, чем просто его потерять.

window.alert()

Метод alert() позволяет выдать *окно* предупреждения:

```
<A HREF="javascript:window.alert('Внимание')">
Повторите запрос!</A>
```

Все очень просто, но нужно иметь в виду, что сообщения выводятся системным шрифтом, следовательно, для получения предупреждений на русском языке нужна локализованная версия ОС.

window.confirm()

Метод confirm() позволяет задать пользователю вопрос, на который тот может ответить либо положительно, либо отрицательно:

```
<FORM>
<INPUT TYPE=button VALUE="Вы знаете JavaScript?"
onClick="if(window.confirm('Знаю все')==true)
{ document.forms[0].elements[1].value='Да'; }
else {
  document.forms[0].elements[1].value='Нет';
};"><BR>
</FORM>
```

Все ограничения для сообщений на русском языке, которые были описаны для метода alert(), справедливы и для метода confirm().

window.prompt()

Метод prompt() позволяет принять от пользователя короткую строку текста, которая набирается в поле ввода информационного окна:

```
<FORM>
<INPUT TYPE=button VALUE="Открыть окно ввода"
onClick="document.forms[0].elements[1].value=window.prompt('Введите
сообщение');">
<INPUT SIZE=30>
</FORM>
```

Введенную пользователем строчку можно присвоить любой переменной и потом разбирать ее в JavaScript-программе.

window.open()

У этого метода окна атрибутов больше, чем у некоторых объектов. Метод ***open()*** предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

open("URL","window_name","param,param,...", replace);

где: URL — страница, которая будет загружена в новое *окно*, window_name — имя окна, которое можно использовать в атрибуте TARGET в контейнерах A и FORM.

Таблица 2.	
Параметры	**Назначение**
replace	Позволяет при открытии окна управлять записью в массив History
param	Список параметров
width	Ширина окна в пикселах
height	Высота окна в пикселах
toolbar	Создает *окно* с системными кнопками браузера
location	Создает *окно* с полем location
directories	Создает *окно* с меню предпочтений пользователя
status	Создает *окно* с полем статуса status
menubar	Создает *окно* с меню
scrollbars	Создает *окно* с полосами прокрутки
resizable	Создает *окно*, размер которого можно будет изменять

window.close()

Метод ***close()*** — это оборотная сторона медали метода *open()*. Он позволяет закрыть *окно*. Чаще всего возникает вопрос, какое из окон, собственно, следует закрыть. Если необходимо закрыть текущее, то:

window.close();
self.close();

Если необходимо закрыть родительское *окно*, т.е. окно, из которого было открыто текущее, то:

```
window.opener.close();
```

Если необходимо закрыть произвольное *окно*, то тогда сначала нужно получить его идентификатор:

```
id=window.open();
...
id.close();
```

Как видно из последнего примера, закрывают *окно* не по имени (значение атрибута TARGET тут ни при чем), а используют указатель на объект.

window.focus()

Метод focus() применяется для передачи фокуса в *окно*, с которым он использовался. Передача фокуса полезна как при открытии окна, так и при его закрытии, не говоря уже о случаях, когда нужно выбирать окна. Рассмотрим пример.

Открываем *окно* и, не закрывая его, снова откроем *окно* с таким же именем, но с другим текстом. Новое *окно* не появилось поверх основного окна, так как фокус ему не был передан. Теперь повторим открытие окна, но уже с передачей фокуса:

window.setTimeout()

Метод ***setTimeout()*** используется для создания нового потока вычислений, исполнение которого откладывается на время (ms), указанное вторым аргументом:

```
idt = setTimeout("JavaScript_код",Time);
```

Типичное применение этой функции — организация автоматического изменения свойств объектов. Например, можно запустить часы в поле формы: window.clearTimeout

Метод clearTimeout() позволяет уничтожить поток, вызванный методом *setTimeout()*. Очевидно, что его применение позволяет более эффективно

распределять ресурсы вычислительной установки. Для того чтобы использовать этот метод в примере с часами, нам нужно модифицировать функции и форму:

Фреймы (Frames)

Фреймы — это несколько видоизмененные окна. Отличаются они от обычных окон тем, что размещаются 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. Существует и ряд других отличий.

Мы остановимся на:

- иерархии *фреймов*;
- именовании *фреймов*;
- передаче данных во *фрейм*.

Естественно, что иерархия определяет и правила именования *фреймов*, и способы передачи фокуса фрейму.

4.2. Интерактивной форма

Контейнер FORM

Если рассматривать программирование на JavaScript в исторической перспективе, то первыми объектами, для которых были разработаны методы и свойства, стали поля форм. Обычно контейнер FORM и поля форм именованы:

```
<FORM NAME=f_name METHOD=get
ACTION="javascript:void(0);">
<INPUT NAME=i_name SIZE=30 MAXLENGTH=30>
</FORM>
```

Поэтому в программах на JavaScript к ним обращаются по имени:

```
window.document.f_name.i_name.value="Текстовое поле";
```

Того же эффекта можно достичь, используя массив форм загруженного документа:

```
window.document.forms[0].elements[0].value="Текстовое поле";
```

В данном примере не только к форме, но и к полю формы мы обращаемся как к элементу массива.

Рассмотрим подробнее объект *Form*, который соответствует контейнеру FORM.

Свойства	Методы	События
action method target elements[] encoding	reset() submit()	onReset onSubmit

Сами по себе методы, свойства и события объекта *Form* используются нечасто. Их переопределение обычно связано с реакцией на изменения значений полей формы.

action

Свойство action отвечает за вызов скрипта (CGI-скрипта). В нем указывается его (скрипта) URL. Но там, где можно указать URL, можно указать и его схему javascript:

```
<FORM METHOD=post
    ACTION="javascript:window.alert('We use JavaScript-code as an URL');
        void(0);">
<INPUT TYPE=submit VALUE="Продемонстрировать JavaScript в action">
</FORM>
```

Обратите внимание на тот факт, что в контейнере FORM указан атрибут METHOD. В данном случае это сделано для того, чтобы к URL, заданному в action, не дописывался символ "?". Дело в том, что *методом доступа* по умолчанию является метод GET. В этом методе при обращении к ресурсу из формы создается элемент URL под названием search. Этот элемент предваряется символом "?", который дописывается к URL скрипта, а в нашем случае, к JavaScript-коду.

Конструкция вида

```
window.alert("String");void(0);?
```

провоцирует ошибку JavaScript.

Метод POST передает данные формы скрипту в теле HTTP-сообщения, поэтому символ "?" не добавляется к URL, и ошибка не генерируется. При этом применение void(0) отменяет перезагрузку документа, и браузер не генерирует событие submit, т.е. не обращается к серверу при нажатии на кнопку, как при стандартной обработке форм.

method

Свойство method определяет *метод доступа* к ресурсам HTTP-сервера из программы-браузера. В зависимости от того, как автор HTML-страницы собирается получать и обрабатывать данные из формы, он может выбрать тот или иной *метод доступа*. На практике чаще всего используются методы GET и POST. JavaScript-программа может изменить значение этого свойства. В предыдущем разделе (action) *метод доступа* в форме был указан явно. Теперь мы его переопределим в момент исполнения программы:

target

Свойство target определяет имя окна, в которое следует загружать результат обращения к CGI-скрипту. Применение значения этого свойства внутри JavaScript-программ не оправдано, так как всегда можно получить идентификатор окна или задействовать встроенный массив frames[0] и свойства окна opener, top, parent и т.п. Для загрузки внешнего файла в некоторое окно всегда можно применить метод window.open(). Но тем не менее использовать это свойство можно:

elements[]

При генерации встроенного в документ объекта *Form* браузер создает и связанный с ним массив полей формы.

encoding

Такое свойство у объекта *Form* есть, но не совсем понятно, как его использовать. Изменение свойства encoding оправдано только в том случае, когда в форме имеется поле типа file. В этом случае предполагается, что пользователю разрешена передача файла со своего локального диска на сервер. При этом если не указана кодировка multipart/form-data, то передаваться будет только имя файла, а если она указана, то и сам файл тоже.

Первое, что приходит в голову по этому поводу, — отмена передачи файла при определенном стечении обстоятельств. Сам скрипт нужно размещать во внешнем файле, чтобы пользователь не изменил его код.

reset()

Метод reset(), не путать с обработчиком события onReset, позволяет установить значения полей формы по умолчанию.

submit()

Метод submit() позволяет проинициировать передачу введенных в форму данных на сервер. При этом методом submit() инициируется тот же процесс, что и нажатием на кнопку типа Submit.

onReset

Событие reset (восстановление значений по умолчанию в полях формы) возникает при нажатии на кнопку типа Reset или при выполнении метода reset(). В контейнере FORM можно переопределить функцию обработки данного события.

onSubmit

Событие submit возникает при нажатии на кнопку типа Submit, графическую кнопку (тип image) или при вызове метода submit(). Для переопределения метода обработки события submit в контейнер FORM добавлен атрибут *onSubmit*. Функция, определенная в этом атрибуте, будет выполняться перед тем, как отправить данные на сервер.

Текст в полях ввода

Поля ввода (контейнер INPUT типа TEXT) являются одним из наиболее популярных объектов программирования на JavaScript. Это объясняется тем, что, помимо использования по прямому назначению, их применяют и в целях отладки программ, вводя в эти поля промежуточные значения переменных и свойств объектов.

Объект Text (текстовое поле ввода) характеризуется следующими свойствами, методами и событиями:

Свойства	Методы	События
• defaultValue • form • name • type • value	• blur() • focus() • select()	• onBlur • onChange • onFocus

Свойства объекта Text — это стандартный набор свойств поля формы. В полях ввода можно изменять только значение свойства value.

Обычно при программировании полей ввода решают две типовых задачи: защита поля от ввода данных пользователем и реакция на изменение значения поля ввода.

Защита поля ввода

Для защиты поля от ввода в него символов применяют метод blur() в сочетании с обработчиком события onFocus:

```
<FORM>
<INPUT SIZE=10 VALUE="1-е значение"
    onFocus="document.forms[0].elements[0].blur();">
<INPUT TYPE=button VALUE=Change
    onClick="document.forms[0].elements[0].value=
'2-е значение';">
<INPUT TYPE=reset VALUE=Reset>
</FORM>
```

В этом примере значение поля ввода можно изменить, только нажав на кнопки Change и Reset. При попытке установить курсор в поле ввода он немедленно оттуда убирается, и таким образом, значение поля не может быть изменено пользователем.

Изменение значения поля ввода

Реакция на изменение значения поля ввода обрабатывается посредством программы, указанной в атрибуте *onChange*:

```
<FORM METHOD="post" onSubmit="return false;">
<INPUT SIZE="15" MAXLENGHT="15" VALUE="Тест"
    onChange="window.alert(document.forms[0].elements[0].value);">
<INPUT TYPE="button" VALUE="Изменить"
    onClick="document.forms[0].elements[0].value='Change';">
</FORM>
```

Если установить фокус на поле ввода и нажать Enter, ничего не произойдет. Если ввести что-либо в расположенное выше поле ввода, а потом нажать на Enter, то появится окно предупреждения с введенным текстом (для Netscape Navigator) или ничего не произойдет (для Internet Explorer последних версий). Если вы используете Internet Explorer последних версий, то окно предупреждения появится только после установки фокуса вне поля ввода. Это следует прокомментировать следующим образом: во-первых, обработчик *onChange* вызывается только тогда, когда ввод в поле закончен. Событие не вызывается при каждом нажатии на кнопки клавиатуры при вводе текста в поле. Во-вторых, обработчик события не вызывается при изменении значения атрибута VALUE из JavaScript-программы. В этом можно убедиться, нажав на кнопку Change - окно предупреждения не открывается. Но если ввести что-то в поле, а после этого нажать на Change, окно появится.

Отметим, что он работает по-разному для Internet Explorer и Netscape Navigator, а именно по-разному обрабатывается событие *onChange*. Для Internet Explorer при любом изменении поля событие обрабатывается незамедлительно, для Netscape Navigator — после потери фокуса активным полем.

Списки и выпадающие меню

В данном случае речь пойдет о выпадающих меню в контексте форм, а не в контексте слоев и технологии CSS.

Одним из важных элементов интерфейса пользователя является меню. В HTML-формах для реализации меню используются поля типа select (контейнер SELECT, который, в свою очередь, вмещают в себя контейнеры OPTION). Эти поля представляют собой списки вариантов выбора. При этом список может "выпадать" или прокручиваться внутри окна. Поля типа select позволяют выбрать из списка только один вариант, либо отметить несколько вариантов. Для управления полями типа select в JavaScript существуют объекты Select и Option.

Эти объекты характеризуются следующими свойствами, методами и событиями:

Объект Select

Свойства	**Методы**	**События**
• form • length • name • options[] • selectedIndex • type	• blur() • click() • focus()	• onBlur • onChange • onFocus

Объект Option

Свойства	**Методы**	**События**
• defaultSelected • index • selected • text • selectedIndex • value	нет	нет

Мы не будем описывать все свойства, методы и события этих двух объектов. Остановимся только на типичных способах применения их комбинаций. Несмотря на то, что объект Option в нашей таблице находится ниже, что отражает его подчиненное по отношению к Select положение, начнем с описания его свойств и особенностей.

Объект Option

Объект Option интересен тем, что в отличие от многих других объектов JavaScript, имеет конструктор. Это означает, что программист может сам создать объект Option:

```
opt = new Option([ text, [ value,
    [ defaultSelected, [ selected ] ] ] ]);
```

где:

text — строка текста, которая размещается в контейнере <LI> (<LI>текст);

value — значение, которое передается серверу при выборе альтернативы, связанной с объектом Option;

defaultSelected — альтернатива выбрана по умолчанию(true/false);

selected — альтернатива выбрана(true/false).

На первый взгляд не очень понятно, для чего может понадобиться программисту такой объект, ведь создать объект типа Select нельзя и, следовательно, нельзя приписать ему новый объект OPTION. Все объясняется, когда речь заходит об изменении списка альтернатив встроенных в документ объектов Select. Делать это можно, так как изменение списка альтернатив Select не приводит к переформатированию документа. Изменение списка альтернатив позволяет решить проблему создания вложенных меню, которых нет в HTML-формах, путем *программирования обычных меню* (options[]).

При программировании альтернатив (объект Option) следует обратить внимание на то, что среди свойств Option нет свойства name. Это означает, что к объекту нельзя обратиться по имени. Отсутствие свойства объясняется тем, что у контейнера OPTION нет атрибута NAME. К встроенным в документ объектам Option можно обращаться только как к элементам массива options[] объекта Select.

options[]

Массив options[] — это свойство объекта Select. Элементы этого массива обладают теми же свойствами, что и объекты Option. Собственно, это и есть объекты Option, встроенные в документ. Они создаются по мере загрузки страницы браузером.

length

В примерах перепрограммирования options[] активно используется свойство объекта Select length. Оно определяет количество альтернатив, заданных для поля выбора. При помощи этого свойства можно удалять и восстанавливать списки.

selectedIndex

Свойство объекта Select, которое возвращает значение выбранного варианта, обозначается как selectedIndex.

onChange

Событие change наступает в тот момент, когда изменяется значение выбранного индекса в объекте Select. С изменением этого индекса в полях выбора единственного варианта на данной странице мы сталкивались неоднократно (selectedIndex и options[]). Данное событие обрабатывается JavaScript-программой, которая указывается в атрибуте *onChange* контейнера SELECT. В этом контексте интересно посмотреть, что происходит, когда мы имеем дело с multiple контейнером SELECT:

selected

Свойство selected объекта Option, на котором был построен пример с канцелярскими принадлежностями, может принимать два значения: истина (true) или ложь (false). В примере мы распечатываем индекс выбранной альтернативы, если значение свойства selected у объекта Option — true:

```
if(form.elements[0].options[i].selected==true)
 ...
```

Вообще говоря, свойство selected интересно именно в случае поля множественного выбора. В случае выбора единственного варианта его можно получить, указав на свойство selectedIndex объекта Select.

text

Свойство text представляет собой отображаемый в меню текст, который соответствует альтернативе:

```
<SELECT onChange= "form.elements[2].value=
form.elements[0].options [form.elements[0
].selectedIndex].text;">
</SELECT>
```

В данном примере свойство text выводится в текстовое поле формы.

value

При передаче данных от браузера к серверу в запросе передается текст выбранной опции, если не было указано значение в атрибуте VALUE контейнера OPTION.

Кнопки

Использование кнопок в Web вообще немыслимо без применения JavaScript. Создайте форму с кнопкой и посмотрите, что будет, если на эту кнопку нажать — кнопка продавливается, но ничего не происходит. Ни одно из стандартных событий

формы (reset или submit) не вызывается. Конечно, данное замечание не касается кнопок Submit и Reset.

Картинки

Кнопки-картинки — это те же кнопки, но только с возможностью отправки данных на сервер. Собственно, такие кнопки в JavaScript составляют две разновидности контейнера INPUT: image и submit. В JavaScript объект, связанный с данными кнопками, называется Submit.

Обмен данными

Передача данных на сервер из формы осуществляется по событию submit. Это событие происходит при одном из следующих действий пользователя:

- нажата кнопка Submit;
- нажата графическая кнопка;
- нажата клавиша Enter в форме из одного поля;
- вызван метод submit().

При описании отображения контейнера FORM на объекты JavaScript было подробно рассказано об обработке события submit. В данном разделе мы сосредоточимся на сочетании JavaScript-программ в атрибутах полей и обработчиках событий. Особое внимание нужно уделить возможности перехвата/генерации события submit.

Кнопка Submit

Кнопка Submit представляет собой разновидность поля ввода. Она ведет себя так же, как и обычная кнопка, но только еще генерирует событие submit (передачу данных на сервер).

Литератра

1. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. **Основы web-технологий** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2007
2. Капустин М.А., Капустин П.А., Копылова А.Г. **Flash MX для профессиональных программистов** Интернет-университет информационных технологий - ИНТУИТ.ру, 2006
3. Савельева Н.В. **Основы программирования на PHP** Интернет-университет информационных технологий - ИНТУИТ.ру, 2005
4. Шохирев М.В. **Язык программирования Perl 5** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006
5. Сузи Р.А. **Язык программирования Python** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006

ТЕМА 12. ВВОД В PHP. УСТАНОВКА PHP И ТЕСТТИРОВАНИЕ (4 ЧАСОВ)

План.

5.1. Виртуальные сервер Apache.

5.2. Синтаксис PHP.

5.3. Типы, переменные, выражение, операции.

5.4. Управляющей операторы программы.

5.5. Создание функции, классы и объекты.

5.6. Обработка ошибки.

5.7. Функции PHP.

5.1. Виртуальные сервер Apache.

Установка и настройка ПО

Возможности языка мы обсудили, области применения рассмотрели, *историю* изучили. Теперь можно приступать к *установке* необходимого инструментария. Поскольку в качестве практической основы курса мы будем рассматривать задачи, решаемые с помощью технологии клиент-*сервер*, и *PHP* соответственно будет использоваться для создания *скриптов*, обрабатываемых *сервером*, нам нужно *установить web-сервер* и *интерпретатор PHP*. В качестве *web-сервера* выберем, например, *Apache*, как наиболее популярный среди web-разработчиков. Для просмотра результатов работы *программ* нам понадобится web-браузер, например Internet Explorer.

Установка Apache 1.3.29 под Windows XP

Чтобы что-нибудь *установить*, нужно для начала иметь соответствующее программное обеспечение (ПО). Скачать ПО для *установки Apache* можно, например, с его официального сайта http://www.apache.org. Мы скачали файл apache_1.3.29-win3x86-no_src.exe. Это автоматический установщик (иначе – wizard) *сервера Apache* под Windows. Эта *программа* попытается почти самостоятельно (а точнее, с минимальными усилиями с вашей стороны) *установить* на компьютер какое-либо программное обеспечение, а в данном случае *сервер*.

Как теперь с ним работать? Откуда можно запускать *скрипты* и где должны находиться файлы пользователей? Файлы, которые должны быть обработаны *сервером*, можно сохранять либо в корне *сервера* (в нашем случае это c:\Program Files\Apache Group\Apache\htdocs), либо в директориях пользователей (в нашем случае это c:\Program Files\Apache Group\Apache\users\). Местоположение корня *сервера* и директорий пользователей прописано в *настройках сервера*, а точнее, в файле конфигурации *httpd.conf* (найти его можно в c:\Program Files\Apache Group\Apache\conf). Для изменения этих путей нужно изменить соответствующие переменные в файле конфигурации *сервера*. Переменная в файле конфигурации ServerRoot отвечает за корневую директорию *сервера*, а переменная UserDir – за директорию, где будут располагаться файлы пользователей *сервера* (для более безопасной работы советуем изменить переменную UserDir на что-нибудь типа c:\users\). Чтобы получить доступ к файлу test.html, находящемуся в корне *сервера*, нужно набрать в браузере http://localhost/test.html (т. е. имя хоста, имя файла). Если же файл test.html находится в директории пользователя user, то для его просмотра нужно набрать в браузере http://localhost/~user/test.html.

Установка PHP 4.3.4 под Windows

Перейдем к *установке PHP*. Скачать его дистрибутив можно с официального сайта *PHP* – http://www.php.net. Для удобства лучше выбрать автоматический инсталлятор, как и в случае с *сервером*. Самое первое окошко при такой *установке PHP* содержит приветствие и предупреждение о существовании авторских прав на этот продукт.

Это значит, что нужно настраивать *сервер Apache* для работы с *PHP* вручную.

Сначала следует выбрать, как мы хотим *установить PHP*, поскольку он поставляется в двух видах: *CGI-скрипт* (php.exe) или набор *SAPI-модулей* (например, php4isapi.dll), используемых *сервером*. Последний вариант обладает новыми возможностями, однако из-за недостаточной проработанности может функционировать недостаточно надежно, особенно на платформах старше Windows 2000 (может появляться куча ошибок 500, могут возникать сбои в других *серверных* модулях, таких как ASP). Так что, если нужна абсолютная стабильность – надо выбирать *установку PHP* в виде *CGI* выполнимого приложения.

Если мы хотим *установить PHP* как серверный модуль, то в файле конфигурации *сервера* (*httpd.conf*) нужно написать:

```
LoadModule php4_module
        c:/php/sapi/php4apache.dll
AddType application/x-httpd-php .php .phtml
AddModule mod_php4.c
```

Если мы устанавливаем *PHP* как *cgi-программу*, то в *httpd.conf* нужно написать:

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php .phtml
Action application/x-httpd-php "/php/php.exe"
```

В этом случае могут возникнуть проблемы с безопасностью. Рекомендуется исправить директорию, где лежит исполняемый файл *интерпретатора* (c:\php\), на что-нибудь менее очевидное (например, на c:\abc_php\). Мы советуем устанавливать *PHP* как серверный модуль.

Еще нужно отредактировать файл *php.ini* (в папке c:\Windows), чтобы *PHP* «знал», где находится корневая директория *сервера*, где пользовательские директории, а где его собственные библиотеки расширений. За это в файле *php.ini* отвечают соответственно переменные ***doc_root***, ***user_dir*** и ***extension_dir***. Зададим их таким образом:

```
doc_root = "c:\Program Files\Apache
        Group\Apache\htdocs"
user_dir = "c:\users"
extension_dir = "c:\php\extensions"
```

Кроме того, можно выбрать расширения, которые будут загружаться при запуске *PHP*. В реализацию *PHP* под Windows изначально входит очень мало расширений. Чтобы загрузить расширение, нужно раскомментировать в *php.ini* соответствующую ему строчку 'extension=php_*.dll'. Например, чтобы загрузить расширение для работы с MSSQL, нужно раскомментировать строку 'extension=php_mssql.dll'. Некоторые расширения требуют дополнительных

библиотек. Поэтому рекомендуется скопировать дополнительные библиотеки в системную директорию (из папки c:\php\dlls). При первой *установке* следует настроить и протестировать *PHP* без расширений.

Для того чтобы *настройки*, выполненные в конфигурационных файлах *сервера* и *PHP* вступили в силу, нужно перезапустить *сервер*.

Проверим, работает ли *PHP*. Для этого создадим тестовый файл (1.php) в директории пользователя (c:\users\nina) со следующим содержанием:

```
<?php
echo"<h1>Привет всем!</h1>";
?>
```

Запустим этот файл через браузер, набрав http://localhost/~nina/1.php. Если что-то не так, то на экран будет выведен текст этого файла. Если все хорошо, то наш *скрипт* должен обработаться *сервером* и вывести большими буквами строку «Привет всем!».

5.2. Синтаксис PHP

Основной синтаксис

Первое, что нужно знать относительно синтаксиса PHP, – это то, как он встраивается в HTML-код, как интерпретатор узнает, что это код на языке PHP. В предыдущей лекции мы уже говорили об этом. Повторяться не будем, отметим только, что в примерах мы чаще всего будем использовать вариант <?php ?>, и иногда сокращенный вариант <? ?>.

Разделение инструкций

Программа на PHP (да и на любом другом языке программирования) – это набор команд (инструкций). Обработчику программы (парсеру) необходимо как-то отличать одну команду от другой. Для этого используются специальные символы – разделители. В PHP инструкции разделяются так же, как и в Си или Perl, – каждое выражение заканчивается точкой с запятой.

Закрывающий тег «?>» также подразумевает конец инструкции, поэтому перед ним точку с запятой не ставят. Например, два следующих фрагмента кода эквивалентны:

```
<?php
echo "Hello, world!"; // точка с запятой
                    // в конце команды
                    // обязательна
?>
<?php
echo "Hello, world!" ?>
<!-- точка с запятой
   опускается из-за "?>" -->
```

Комментарии

Часто при написании программ возникает необходимость делать какие-либо ***комментарии*** к коду, которые никак не влияют на сам код, а только поясняют его. Это важно при создании больших программ и в случае, если несколько человек работают над одной программой. При наличии *комментариев* в программе в ее коде разобраться гораздо проще. Кроме того, если решать задачу по частям, недоделанные части решения также удобно *комментировать*, чтобы не забыть о них в дальнейшем. Во всех языках программирования предусмотрена возможность включать *комментарии* в код программы. PHP поддерживает несколько видов *комментариев*: в стиле Си, C++ и оболочки Unix. Символы // и # обозначают начало однострочных *комментариев*, /* и */ – соответственно начало и конец многострочных *комментариев*.

5.3. Типы, переменные, выражение, операции.

Переменные, константы и операторы

Важным элементом каждого языка являются *переменные*, *константы* и *операторы*, применяемые к этим *переменным* и *константам*. Рассмотрим, как выделяются и обрабатываются эти элементы в PHP.

Переменные

Переменная в PHP обозначается знаком доллара, за которым следует ее имя. Например:

$my_var

Имя *переменной* чувствительно к регистру, т.е. *переменные* $my_var и $My_var различны.

Имена *переменных* соответствуют тем же правилам, что и остальные наименования в PHP: правильное имя *переменной* должно начинаться с буквы или символа подчеркивания с последующими в любом количестве буквами, цифрами или символами подчеркивания.

В PHP 3 *переменные* всегда присваивались по значению. То есть когда вы присваиваете выражение *переменной*, все значения оригинального выражения копируются в эту *переменную*. Это означает, к примеру, что после присвоения одной *переменной* значения другой изменение одной из них не влияет на значение другой.

Константы

Для хранения постоянных величин, т.е. таких величин, значение которых не меняется в ходе выполнения скрипта, используются ***константы***. Такими величинами могут быть математические *константы*, пароли, пути к файлам и т.п. Основное отличие *константы* от *переменной* состоит в том, что ей нельзя присвоить значение больше одного раза и ее значение нельзя аннулировать после ее объявления. Кроме того, у *константы* нет приставки в виде знака доллара и ее нельзя определить простым присваиванием значения. Как же тогда можно определить *константу*? Для этого существует специальная функция define(). Ее синтаксис таков:

define("Имя_константы",

"Значение_константы",

[Нечувствительность_к_регистру])

По умолчанию имена *констант* чувствительны к регистру. Для каждой *константы* это можно изменить, указав в качестве значения аргумента Нечувствительность_к_регистру значение True. Существует соглашение, по которому имена *констант* всегда пишутся в верхнем регистре.

Получить значение *константы* можно, указав ее имя. В отличие от *переменных*, не нужно предварять имя *константы* символом $. Кроме того, для получения значения *константы* можно использовать функцию constant() с именем *константы* в качестве параметра.

Операторы

Операторы позволяют выполнять различные действия с *переменными*, *константами* и выражениями. Мы еще не упоминали о том, что такое выражение. Выражение можно определить как все, что угодно, что имеет значение. *Переменные* и *константы* – это основные и наиболее простые формы выражений. Существует множество операций (и соответствующих им *операторов*), которые можно производить с выражениями. Рассмотрим некоторые из них подробнее.

Таблица 2.1. Арифметические операторы

Обозначение	Название	Пример
+	Сложение	$a + $b
-	Вычитание	$a - $b
*	Умножение	$a * $b
/	Деление	$a / $b
%	Остаток от деления	$a % $b

Таблица 2.2. Строковые операторы

Обозначение	Название	Пример
.	*Конкатенация* (*сложение строк*)	$c = $a . $b (это строка, состоящая из $a и $b)

Таблица 2.3. Операторы присваивания

Обозначение	Название	Описание	Пример
=	Присваивание	*Переменной* слева от *оператора* будет присвоено значение, полученное в результате выполнения каких-либо операций или *переменной*/*константы* с правой стороны	$a = ($b = 4) +5; ($a будет равна 9, $b будет равна 4)
+=		Сокращение. Прибавляет к *переменной* число и затем присваивает ей полученное значение	$a += 5; (эквивалентно $a = $a + 5;)
.=		Сокращенно обозначает комбинацию операций *конкатенации* и присваивания (сначала добавляется строка, потом полученная строка записывается в *переменную*)	$b = "Привет "; $b .= "всем"; (эквивалентно $b = $b . "всем";) В результате: $b="Привет всем"

Таблица 2.4. Логические операторы

Обозначение	Название	Описание	Пример
and	И	$a и $b истинны (True)	$a and $b
&&	И		$a && $b
or	Или	Хотя бы одна из *переменных* $a или $b истинна (возможно, что и обе)	$a or $b
||	Или		$a || $b
xor	Исключающее или	Одна из *переменных* истинна. Случай, когда они обе истинны, исключается	$a xor $b

!	Инверсия (NOT)	Если $a=True, то !$a=False и наоборот	! $a

Таблица 2.5. Операторы сравнения

Обозначение	Название	Пример	Описание
==	Равенство	Значения *переменных* равны	$a == $b
===	Эквивалентность	Равны значения и *типы переменных*	$a === $b
!=	Неравенство	Значения *переменных* не равны	$a != $b
<>	Неравенство		$a <> $b
!==	Неэквивалентность	*Переменные* не эквивалентны	$a !== $b
<	Меньше		$a < $b
>	Больше		$a > $b
<=	Меньше или равно		$a <= $b
>=	Больше или равно		$a >= $b

Таблица 2.6. Операторы инкремента и декремента

Обозначение	Название	Описание	Пример
++$a	Пре-*инкремент*	Увеличивает $a на единицу и возвращает $a	<? $a=4; echo "Должно быть 4:" .$a++; echo "Должно быть 6:" .++$a; ?>
$a++	Пост-*инкремент*	Возвращает $a, затем увеличивает $a на единицу	
--$a	Пре-*декремент*	Уменьшает $a на	

		единицу и возвращает $a	
$a--	Пост-*декремент*	Возвращает $a, затем уменьшает $a на единицу	

Типы данных

PHP поддерживает восемь простых *типов данных*.

Четыре скалярных *типа*:

- *boolean (логический)*;
- *integer (целый)*;
- *float (с плавающей точкой)*;
- *string (строковый)*.

Два смешанных *типа*:

- *array (массив)*;
- *object (объект)*.

И два специальных *типа*:

- *resource (ресурс)*;
- *NULL*.

В PHP не принято явное объявление *типов переменных*. Предпочтительнее, чтобы это делал сам интерпретатор во время выполнения программы в зависимости от контекста, в котором используется *переменная*. Рассмотрим по порядку все перечисленные *типы данных*.

Тип boolean (булев или логический тип)

Этот простейший *тип* выражает истинность значения, то есть *переменная* этого *типа* может иметь только два значения – истина TRUE или ложь FALSE.

Чтобы определить булев *тип*, используют ключевое слово TRUE или FALSE. Оба регистронезависимы.

```
<?php
$test = True;
?>
```

Тип integer (целые)

Этот *тип* задает число из множества целых чисел Z = {..., -2, -1, 0, 1, 2, ...}. Целые могут быть указаны в десятичной, шестнадцатеричной или восьмеричной системе счисления, по желанию с предшествующим знаком «-» или «+».

Если вы используете восьмеричную систему счисления, вы должны предварить число 0 (нулем), для использования шестнадцатеричной системы нужно поставить перед числом 0x.

Размер *целого* зависит от платформы, хотя, как правило, максимальное значение около двух миллиардов (это 32-битное знаковое). Беззнаковые *целые* PHP не поддерживает.

Если вы определите число, превышающее пределы *целого типа*, оно будет интерпретировано как *число с плавающей точкой*. Также если вы используете *оператор*, результатом работы которого будет число, превышающее пределы *целого*, вместо него будет возвращено *число с плавающей точкой*.

В PHP не существует *оператора* деления *целых*. Результатом 1/2 будет *число с плавающей точкой* 0.5. Вы можете привести значение к *целому*, что всегда округляет его в меньшую сторону, либо использовать функцию round(), округляющую значение по стандартным правилам. Для преобразования *переменной* к конкретному *типу* нужно перед *переменной* указать в скобках нужный *тип*. Например, для

преобразования *переменной* $a=0.5 к *целому типу* необходимо написать (integer)(0.5) или (integer) $a или использовать сокращенную запись (int)(0.5). Возможность явного приведения *типов* по такому принципу существует для всех *типов данных* (конечно, не всегда значение одного *типа* можно перевести в другой *тип*). Мы не будем углубляться во все тонкости приведения *типов*, поскольку PHP делает это автоматически в зависимости от контекста.

Тип float (числа с плавающей точкой)

Числа с плавающей точкой (они же числа двойной точности или действительные числа) могут быть определены при помощи любого из следующих синтаксисов:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Размер *числа с плавающей точкой* зависит от платформы, хотя максимум, как правило, ~1.8e308 с точностью около 14 десятичных цифр.

Тип string (строки)

Строка – это набор символов. В PHP символ – это то же самое, что байт, это значит, что существует ровно 256 различных символов. Это также означает, что PHP не имеет встроенной поддержки Unicode. В PHP практически не существует ограничений на размер *строк*, поэтому нет абсолютно никаких причин беспокоиться об их длине.

Строка в PHP может быть определена тремя различными *способами*:

- с помощью *одинарных кавычек*;

- с помощью *двойных кавычек*;
- *heredoc-синтаксисом*.

Одинарные кавычки

Простейший *способ* определить *строку* – это заключить ее в *одинарные кавычки* «'». Чтобы использовать *одинарную кавычку* внутри *строки*, как и во многих других языках, перед ней необходимо поставить символ обратной косой черты «\», т. е. экранировать ее. Если обратная косая черта должна идти перед *одинарной кавычкой* либо быть в конце *строки*, необходимо продублировать ее «\\'».

Если внутри *строки*, заключенной в *одинарные кавычки*, обратный слэш «\» встречается перед любым другим символом (отличным от «\» и «'»), то он рассматривается как обычный символ и выводится, как и все остальные. Поэтому обратную косую черту необходимо экранировать, только если она находится в конце *строки*, перед закрывающей кавычкой.

В PHP существует ряд комбинаций символов, начинающихся с символа обратной косой черты. Их называют ***управляющими последовательностями***, и они имеют специальные значения, о которых мы расскажем немного позднее. Так вот, в отличие от двух других синтаксисов, *переменные* и *управляющие последовательности* для специальных символов, встречающиеся в *строках*, заключенных в *одинарные кавычки*, не *обрабатываются*.

Если *строка* заключена в *двойные кавычки* «"», PHP распознает большее количество *управляющих последовательностей* для специальных символов. Некоторые из них приведены в таблице 2.7.

Таблица 2.7. Управляющие последовательности

Последовательность	Значение
\n	Новая *строка* (LF или 0x0A (10) в ASCII)
\r	Возврат каретки (CR или 0x0D (13) в ASCII)

\t	Горизонтальная табуляция (HT или 0x09 (9) в ASCII)
\\	Обратная косая черта
\$	Знак доллара
\"	Двойная кавычка

Повторяем, если вы захотите экранировать любой другой символ, обратная косая черта также будет напечатана!

Самым важным свойством *строк* в *двойных кавычках* является *обработка переменных*.

Heredoc

Другой *способ определения строк* – это использование ***heredoc-синтаксиса***. В этом случае *строка* должна начинаться с символа <<<, после которого идет идентификатор. Заканчивается *строка* этим же идентификатором. Закрывающий идентификатор должен начинаться в первом столбце *строки*. Кроме того, идентификатор должен соответствовать тем же правилам именования, что и все остальные метки в PHP: содержать только буквенно-цифровые символы и знак подчеркивания и начинаться не с цифры или знака подчеркивания.

Heredoc-текст ведет себя так же, как и *строка* в *двойных кавычках*, при этом их не имея. Это означает, что вам нет необходимости экранировать кавычки в *heredoc*, но вы по-прежнему можете использовать перечисленные выше *управляющие последовательности*. *Переменные* внутри *heredoc* тоже *обрабатываются*.

Тип array (массив)

Массив в PHP представляет собой упорядоченную карту – *тип*, который преобразует *значения* в *ключи*. Этот *тип* оптимизирован в нескольких направлениях, поэтому вы можете использовать его как собственно *массив*, список (вектор), хеш-

таблицу (являющуюся реализацией карты), стэк, очередь и т.д. Поскольку вы можете иметь в качестве *значения* другой *массив* PHP, можно также легко эмулировать деревья.

Определить *массив* можно с помощью конструкции *array*() или непосредственно задавая *значения* его элементам.

Определение при помощи array()

array ([key] => value,
    [key1] => value1, ...)

Языковая конструкция ***array***() принимает в качестве параметров пары *ключ* => ***значение***, разделенные запятыми. Символ => устанавливает соответствие между *значением* и его *ключом*. *Ключ* может быть как *целым числом*, так и *строкой*, а *значение* может быть любого имеющегося в PHP *типа*. Числовой *ключ массива* часто называют индексом. Индексирование *массива* в PHP начинается с нуля. *Значение* элемента *массива* можно получить, указав после имени *массива* в *квадратных скобках ключ* искомого элемента. Если ***ключ*** *массива* представляет собой стандартную запись *целого числа*, то он рассматривается как число, в противном случае – как *строка*. Поэтому запись $a["1"] равносильна записи $a[1], так же как и $a["-1"] равносильно $a[-1].

Если использовать в качестве *ключа* TRUE или FALSE, то его *значение* переводится соответственно в единицу и ноль *типа integer*. Если использовать *NULL*, то вместо *ключа* получим пустую *строку*. Можно использовать и саму пустую *строку* в качестве *ключа*, при этом ее надо брать в кавычки. Так что это не то же самое, что использование пустых *квадратных скобок*. Нельзя использовать в качестве *ключа массивы* и *объекты*.

Определение с помощью синтаксиса квадратных скобок

Создать *массив* можно, просто записывая в него *значения*. Как мы уже говорили, *значение* элемента *массива* можно получить с помощью *квадратных*

скобок, внутри которых нужно указать его *ключ* например, $book["php"]. Если указать новый *ключ* и новое *значение* например, $book["new_key"]="new_value", то в *массив* добавится новый элемент. Если мы не укажем *ключ*, а только присвоим *значение* $book[]="new_value", то новый элемент *массива* будет иметь числовой *ключ*, на единицу больший максимального существующего. Если *массив*, в который мы добавляем *значения*, еще не существует, то он будет *создан*.

Для того чтобы изменить конкретный элемент *массива*, нужно просто присвоить ему с его *ключом* новое *значение*. Изменить *ключ* элемента нельзя, можно только *удалить элемент* (пару *ключ*/*значение*) и добавить новую. Чтобы ***удалить элемент*** *массива*, нужно использовать функцию ***unset***().

Тип object (объекты)

Объекты – *тип данных*, пришедший из объектно-ориентированного программирования (ООП). Согласно принципам ООП, класс – это набор *объектов*, обладающих определенными свойствами и методами работы с ним, а *объект* соответственно – экземпляр класса. Например, программисты – это класс людей, которые пишут программы, изучают компьютерную литературу и, кроме того, как все люди, имеют имя и фамилию. Теперь, если взять одного конкретного программиста, Васю Иванова, то можно сказать, что он является *объектом* класса программистов, обладает теми же свойствами, что и другие программисты, тоже имеет имя, пишет программы и т.п.

В PHP для доступа к методам *объекта* используется *оператор* ->. Для инициализации *объекта* используется выражение new, создающее в *переменной* экземпляр *объекта*.

Тип resource (ресурсы)

Ресурс – это специальная *переменная*, содержащая ссылку на внешний *ресурс* (например, соединение с базой данных). *Ресурсы* создаются и используются специальными функциями (например, mysql_connect(), pdf_new() и т.п.).

Тип Null

Специальное значение *NULL* говорит о том, что *переменная* не имеет значения.

Переменная считается *NULL*, если:

- ей была присвоена *константа NULL* ($var = NULL);
- ей еще не было присвоено какое-либо значение;
- она была удалена с помощью *unset*().

Существует только одно значение *типа NULL* – регистронезависимое ключевое слово NULL.

Решение задачи

Теперь вернемся к задаче, которую мы поставили в самом начале лекции. Напомним, что она состояла в *составлении письма* разным людям по поводу разных событий. Попытаемся использовать для решения этой задачи изученные средства – *переменные*, *операторы*, *константы*, *строки* и *массивы*. В зависимости от получателя изменяется событие и обращение, указанные в письме, поэтому естественно вынести эти величины в *переменные*. Более того, поскольку событий и людей много, удобно использовать *переменные типа массив*. Подпись в письме остается постоянной всегда, поэтому логично задать ее как *константу*. Чтобы не писать слишком длинные и громоздкие *строки*, используем *оператор конкатенации*. Итак, вот что получилось:

5.4. Управляющей операторы программы

Условные операторы

Оператор if

Это один из самых важных операторов многих языков, включая PHP. Он позволяет выполнять фрагменты кода в зависимости от условия. Структуру оператора *if* можно представить следующим образом:

if (выражение) блок_выполнения

Здесь выражение есть любое правильное PHP-выражение (т.е. все, что имеет значение). В процессе обработки скрипта выражение преобразуется к логическому типу. Если в результате преобразования значение выражения истинно (True), то выполняется блок_выполнения. В противном случае блок_выполнения игнорируется. Если блок_выполнения содержит несколько команд, то он должен быть заключен в фигурные скобки { }.

Правила преобразования выражения к логическому типу:

1. В FALSE преобразуются следующие значения:
    - логическое False
    - целый ноль (0)
    - действительный ноль (0.0)
    - пустая строка и строка "0"
    - массив без элементов
    - объект без переменных (подробно об объектах будет рассказано в одной из следующих лекций)
    - специальный тип NULL
2. Все остальные значения преобразуются в TRUE.

Оператор else

Мы рассмотрели только одну, основную часть оператора *if*. Существует несколько расширений этого оператора. Оператор *else* расширяет *if* на случай, если проверяемое в *if* выражение является неверным, и позволяет выполнить какие-либо действия при таких условиях.

Структуру оператора *if*, расширенного с помощью оператора *else*, можно представить следующим образом:

if (выражение) блок_выполнения

else блок_выполнения1

Эту конструкцию if...else можно интерпретировать примерно так: если выполнено условие (т.е. выражение=true), то выполняем действия из блока_выполнения, иначе – действия из блока_выполнения1. Использовать оператор *else* не обязательно.

Посмотрим, как можно изменить предыдущий пример, учитывая необходимость совершения действий и в случае невыполнения условия.

Оператор elseif

Еще один способ расширения *условного оператора if* – использование оператора *elseif*. *elseif* – это комбинация *else* и *if*. Как и *else*, он расширяет *if* для выполнения различных действий в том случае, если условие, проверяемое в *if*, неверно. Но в отличие от *else*, альтернативные действия будут выполнены, только если elseif-условие является верным. Структуру оператора *if*, расширенного с помощью операторов *else* и *elseif*, можно представить следующим образом:

if (выражение) блок_выполнения
elseif(выражение1) блок_выполнения1
...
else блок_выполненияN

Операторов *elseif* может быть сразу несколько в одном if-блоке. Elseif-утверждение будет выполнено, только если предшествующее if-условие является False, все предшествующие elseif-условия являются False, а данное elseif-условие – True.

Альтернативный синтаксис

PHP предлагает *альтернативный синтаксис* для некоторых своих управляющих структур, а именно для *if*, *while*, *for*, *foreach* и *switch*. В каждом случае

открывающую скобку нужно заменить на двоеточие (:), а закрывающую – на endif;, endwhile; и т.д. соответственно.

Например, синтаксис оператора *if* можно записать таким образом:

```
if(выражение): блок_выполнения endif;
```

Смысл остается тем же: если условие, записанное в круглых скобках оператора *if*, оказалось истиной, будет выполняться весь код, от двоеточия «:» до команды endif;. Использование такого синтаксиса полезно при встраивании php в html-код.

Если используются конструкции *else* и *elseif*, то также можно задействовать *альтернативный синтаксис*:

Оператор switch

Еще одна конструкция, позволяющая проверять условие и выполнять в зависимости от этого различные действия, – это *switch*. На русский язык название данного оператора можно перевести как «переключатель». И смысл у него именно такой. В зависимости от того, какое значение имеет переменная, он переключается между различными блоками действия. *switch* очень похож на оператор if...elseif...else или набор операторов *if*. Структуру *switch* можно записать следующим образом:

```
switch (выражение или переменная){
case значение1:
   блок_действий1
break;
case значение2:
   блок_действий2
break;
...
default:
   блок_действий_по_умолчанию
```

}

В отличие от *if*, здесь значение выражения не приводится к логическому типу, а просто сравнивается со значениями, перечисленными после ключевых слов *case* (значение1, значение 2 и т.д.). Если значение выражения совпало с каким-то вариантом, то выполняется соответствующий блок_действий – от двоеточия после совпавшего значения до конца *switch* или до первого оператора *break*, если таковой найдется. Если значение выражения не совпало ни с одним из вариантов, то выполняются действия по умолчанию (блок_действий_по_умолчанию), находящиеся после ключевого слова default. Выражение в *switch* вычисляется только один раз, а в операторе *elseif* – каждый раз, поэтому, если выражение достаточно сложное, то *switch* работает быстрее.

Циклы

В PHP существует несколько конструкций, позволяющих выполнять повторяющиеся действия в зависимости от условия. Это циклы *while*, *do..while*, *foreach* и *for*. Рассмотрим их более подробно.

while

Структура:

while (выражение) { блок_выполнения }

либо

while (выражение): блок_выполнения endwhile;

while – простой цикл. Он предписывает PHP выполнять команды блока_выполнения до тех пор, пока выражение вычисляется как True (здесь, как и в *if*, происходит приведение выражения к логическому типу). Значение выражения проверяется каждый раз в начале цикла, так что, даже если его значение изменилось в процессе

выполнения блока_выполнения, цикл не будет остановлен до конца итерации (т.е. пока все команды блока_выполнения не будут исполнены).

do... while

Циклы *do..while* очень похожи на циклы *while*, с той лишь разницей, что истинность выражения проверяется в конце цикла, а не в начале. Благодаря этому блок_выполнения цикл do...while гарантированно выполняется хотя бы один раз.

Структура:

do {блок_выполнения} while (выражение);

for

Это самые сложные циклы в PHP. Они напоминают соответствующие циклы C.

Структура:

for (выражение1; выражение2; выражение3) {блок_выполнения}

либо

for (выражение1; выражение2; выражение3): блок_выполнения endfor;

Здесь, как мы видим, условие состоит сразу из трех выражений. Первое выражение выражение1 вычисляется безусловно один раз в начале цикла. В начале каждой итерации вычисляется выражение2. Если оно является True, то цикл продолжается и выполняются все команды блока_выполнения. Если выражение2 вычисляется как False, то исполнение цикла останавливается. В конце каждой итерации (т.е. после выполнения всех команд блока_выполнения) вычисляется выражение3.

Каждое из выражений 1, 2, 3 может быть пустым. Если выражение2 является пустым, то это значит, что цикл должен выполняться неопределенное время (в этом

случае PHP считает это выражение всегда истинным). Это не так бесполезно, как кажется, ведь цикл можно останавливать, используя оператор *break*.

foreach

Еще одна полезная конструкция. Она появилась только в PHP4 и предназначена исключительно для работы с массивами.

Синтаксис:

```
foreach ($array as $value) {блок_выполнения}
```

либо

```
foreach ($array as $key => $value)
  {блок_выполнения}
```

В первом случае формируется цикл по всем элементам массива, заданного переменной $array. На каждом шаге цикла значение текущего элемента массива записывается в переменную $value, и внутренний счетчик массива передвигается на единицу (так что на следующем шаге будет виден следующий элемент массива). Внутри блока_выполнения значение текущего элемента массива может быть получено с помощью переменной $value. Выполнение блока_выполнения происходит столько раз, сколько элементов в массиве $array.

Вторая форма записи в дополнение к перечисленному выше на каждом шаге цикла записывает ключ текущего элемента массива в переменную $key, которую тоже можно использовать в блоке_выполнения.

Операторы передачи управления

Иногда в случае особых обстоятельств требуется немедленно завершить работу цикла и передать управление первой инструкции программы, расположенной за последней инструкцией цикла. Для этого используют операторы *break* и *continue*.

Break

Оператор *break* заканчивает выполнение текущего цикла, будь то *for*, *foreach*, *while*, *do..while* или *switch*. *break* может использоваться с числовым аргументом, который говорит, работу скольких управляющих структур, содержащих его, нужно завершить.

continue

Иногда нужно не полностью прекратить работу цикла, а только начать его новую итерацию. Оператор *continue* позволяет пропустить дальнейшие инструкции из блока_выполнения любого цикла и продолжить выполнение с нового круга. *continue* можно использовать с числовым аргументом, который указывает, сколько содержащих его управляющих конструкций должны завершить работу.

Заменим в примере предыдущего параграфа оператор *break* на *continue*. Кроме того, ограничим число шагов цикла четырьмя.

Операторы включения

include

Оператор *include* позволяет включать код, содержащийся в указанном файле, и выполнять его столько раз, сколько программа встречает этот *оператор*. Включение может производиться любым из перечисленных способов:

include 'имя_файла';
include $file_name;
include ("имя_файла");

Заметим, что использование оператора *include* эквивалентно простой вставке содержательной части файла params.inc в код программы include.php. Может быть, тогда можно было в params.inc записать простой текст без всяких тегов, указывающих на то, что это php-код? Нельзя! Дело в том, что в момент вставки файла происходит переключение из режима обработки PHP в режим HTML.

Поэтому код внутри включаемого файла, который нужно обработать как PHP-скрипт, должен быть заключен в соответствующие теги.

Поиск файла для вставки происходит по следующим правилам.

Сначала ведется поиск файла в *include_path* относительно текущей рабочей директории.

Если файл не найден, то поиск производится в *include_path* относительно директории текущего скрипта.

Параметр *include_path*, определяемый в файле настроек PHP, задает имена директорий, в которых нужно искать включаемые файлы.

Например, ваш *include_path* это . (то есть текущая директория), текущая рабочая директория это /www/. В основной файл include.php вы включаете файл my_dir/a.php, который в свою очередь включает b.php. Тогда парсер первым делом ищет файл b.php в директории /www/, и если такового нет, то в директории /www/my_dir/.

Если файл включен с помощью *include*, то содержащийся в нем код наследует область видимости переменных строки, где появился *include*. Любые переменные вызванного файла будут доступны в вызывающем файле с этой строки и далее. Соответственно, если *include* появляется внутри функции вызывающего файла, то код, содержащийся в вызываемом файле, будет вести себя так, как будто он был определен внутри функции. Таким образом, он унаследует область видимости этой функции. Хотя мы и не знакомились еще с понятием функции, все же приводим здесь эти сведения в расчете на интуитивное его понимание.

Кроме локальных файлов, с помощью *include* можно включать и внешние файлы, указывая их url-адреса. Данная возможность контролируется директивой url_fopen_wrappers в файле настроек PHP и по умолчанию, как правило, включена. Но в версиях PHP для Windows до PHP 4.3.0 эта возможность не поддерживается совсем, вне зависимости от url_fopen_wrappers.

include() – это специальная языковая конструкция, поэтому при использовании внутри условных блоков ее нужно заключать в фигурные скобки.

При использовании *include* возможны два вида ошибок – ошибка вставки (например, нельзя найти указанный файл, неверно написана сама команда вставки и т.п.) или ошибка исполнения (если ошибка содержится во вставляемом файле). В любом случае при ошибке в команде *include* исполнение скрипта не завершается.

require

Этот оператор действует примерно так же, как и #include в C++. Все, что мы говорили о *include*, лишь за некоторыми исключениями, справедливо и для *require*. *Require* также позволяет включать в программу и исполнять какой-либо файл. Основное отличие *require* и *include* заключается в том, как они реагируют на возникновение ошибки. Как уже говорилось, *include* выдает предупреждение, и работа скрипта продолжается. Ошибка в *require* вызывает фатальную ошибку работы скрипта и прекращает его выполнение.

Условные операторы на require() не влияют. Хотя, если строка, в которой появляется этот оператор, не исполняется, то ни одна строка кода из вставляемого файла тоже не исполняется. Циклы также не влияют на require(). Хотя код, содержащийся во вставляемом файле, является объектом цикла, но вставка сама по себе происходит только однажды.

В реализациях PHP до версии 4.0.2 использование require() означало, что интерпретатор обязательно попытается прочесть вставляемый файл.

require, как и *include*, при использовании внутри условных блоков нужно заключать в фигурные скобки.

Решение задачи

И наконец, вернемся к задаче, сформулированной в начале лекции. Мы хотим создать программу, которую можно было бы использовать для отправки писем (или просто для их генерации) с приглашениями на различные мероприятия множеству пользователей. В предыдущей лекции уже рассматривался подобный случай. Сейчас мы вынесем всю информацию о людях и событиях в отдельный файл data.php и напишем программу, не зависящую (ну, может, совсем чуть-чуть зависящую) от этой информации и ее структуры. В этом случае для того, чтобы, например, расширить список адресатов, не нужно будет изменять скрипт, генерирующий приглашения.

Кроме того, можно будет использовать информацию о людях и событиях в других скриптах. В самом скрипте, генерирующем приглашения letters.php, мы использовали *условные операторы*, циклы, *require* и другие изученные ранее конструкции.

5.5. Создание функции, классы и объекты.

Функции, определяемые пользователем

Для чего нужны функции? Чтобы ответить на этот вопрос, нужно понять, что вообще представляют собой функции. В программировании, как и в математике, функция есть отображение множества ее аргументов на множество ее значений. То есть функция для каждого набора значений аргумента возвращает какие-то значения, являющиеся результатом ее работы. Зачем нужны функции, попытаемся объяснить на примере. Классический пример функции в программировании – это функция, вычисляющая значение факториала числа. То есть мы задаем ей число, а она возвращает нам его факториал. При этом не нужно для каждого числа, факториал которого мы хотим получить, повторять один и тот же код – достаточно просто вызвать функцию с аргументом, равным этому числу.

Таким образом, когда мы осуществляем действия, в которых прослеживается зависимость от каких-то данных, и при этом, возможно, нам понадобится выполнять такие же действия, но с другими исходными данными, удобно использовать механизм функций – оформить блок действий в виде *тела функции*, а меняющиеся данные – в качестве ее параметров.

Посмотрим, как в общем виде выглядит задание (объявление) функции. Функция может быть определена с помощью следующего синтаксиса:

```
function Имя_функции (параметр1, параметр2,
  ... параметрN){
  Блок_действий
  return "значение возвращаемое функцией";
}
```

Если прямо так написать в php-программе, то работать ничего не будет. Во-первых, Имя_функции и имена *параметров функции* (параметр1, параметр2 и т.д.)

должны соответствовать правилам наименования в PHP (и русских символов в них лучше не использовать). Имена функций нечувствительны к регистру. Во-вторых, *параметры функции* – это переменные языка, поэтому перед названием каждого из них должен стоять знак $. Никаких троеточий ставить в списке параметров нельзя. В-третьих, вместо слов блок_действий в *теле функции* должен находиться любой правильный PHP-код (не обязательно зависящий от параметров). И наконец, после ключевого слова *return* должно идти корректное php-выражение (что-либо, что имеет значение). Кроме того, у функции может и не быть параметров, как и возвращаемого значения. Пример правильного объявления функции – функция вычисления факториала, приведенная выше.

Как происходит вызов функции? Указывается имя функции и в круглых скобках список значений ее параметров, если таковые имеются:

```
<?php
Имя_функции ("значение_для_параметра1",
  "значение_для_параметра2",...);
// пример вызова функции – вызов функции
// вычисления факториала приведен выше,
// там для вычисления факториала числа 3
// мы писали: fact(3);
// где fact – имя вызываемой функции,
// а 3 – значение ее параметра с именем $n
?>
```

Когда можно вызывать функцию? Казалось бы, странный вопрос. Функцию можно вызвать после ее определения, т.е. в любой строке программы ниже блока function f_name(){...}. В PHP3 это было действительно так. Но уже в PHP4 такого требования нет. Все дело в том, как интерпретатор обрабатывает получаемый код. Единственное исключение составляют функции, определяемые условно (внутри условных операторов или других функций).

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя. Несмотря на то, что имена функций нечувствительны к

регистру, лучше вызывать функцию по тому же имени, каким она была задана в определении.

Аргументы функций

У каждой функции может быть, как мы уже говорили, список аргументов. С помощью этих аргументов в функцию передается различная информация (например, значение числа, факториал которого надо подсчитать). Каждый аргумент представляет собой переменную или константу.

С помощью аргументов данные в функцию можно передавать тремя различными способами. Это передача аргументов по значению (используется по умолчанию), по ссылке и *задание значения аргументов по умолчанию*. Рассмотрим эти способы подробнее.

Когда аргумент передается в функцию по значению, изменение значения аргумента внутри функции не влияет на его значение вне функции. Чтобы позволить функции изменять ее аргументы, их нужно передавать по ссылке. Для этого в определении функции перед именем аргумента следует написать знак амперсанд «&».

В функции можно определять значения аргументов, используемые по умолчанию. Само значение по умолчанию должно быть константным выражением, а не переменной и не представителем класса или вызовом другой функции.

У нас есть функция, создающая информационное сообщение, подпись к которому меняется в зависимости от значения переданного ей параметра. Если значение параметра не задано, то используется подпись "Оргкомитет".

Списки аргументов переменной длины

В PHP4 можно создавать функции с переменным числом аргументов. То есть мы создаем функцию, не зная заранее, со сколькими аргументами ее вызовут. Для написания такой функции никакого специального синтаксиса не требуется. Все делается с помощью *встроенных функций func_num_args()*, *func_get_arg()*, *func_get_args()*.

Функция ***func_num_args()*** возвращает число аргументов, переданных в текущую функцию. Эта функция может использоваться только внутри определения

пользовательской функции. Если она появится вне функции, то интерпретатор выдаст предупреждение.

Функция ***func_get_arg*** (целое номер_аргумента) возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром номер_аргумента. Аргументы функции считаются начиная с нуля. Как и *func_num_args()*, эта функция может использоваться только внутри определения какой-либо функции.

Номер_аргумента не может превышать число аргументов, переданных в функцию. Иначе будет сгенерировано предупреждение, и функция *func_get_arg()* возвратит False.

Создадим функцию для проверки типа данных ее аргументов. Считаем, что проверка прошла успешно, если первый аргумент функции – целое число, второй – строка.

Функция ***func_get_args()*** возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции. Если функция используется вне определения пользовательской функции, то генерируется предупреждение.

Перепишем предыдущий пример, используя эту функцию.

Как видим, комбинации функций *func_num_args()*, *func_get_arg()* и *func_get_args()* используется для того, чтобы функции могли иметь переменный список аргументов. Эти функции были добавлены только в PHP 4. В PHP3 для того, чтобы добиться подобного эффекта, можно использовать в качестве аргумента функции массив. Например, вот так можно написать скрипт, проверяющий, является ли каждый нечетный *параметр функции* целым числом:

Использование переменных внутри функции

Глобальные переменные

Чтобы использовать внутри функции переменные, заданные вне ее, эти переменные нужно объявить как глобальные. Для этого в *теле функции* следует перечислить их имена после ключевого слова *global*:

global $var1, $var2;

```
<?
$a=1;
function Test_g(){
global $a;
  $a = $a*2;
  echo 'в результате работы функции $a=',$a;
}
echo 'вне функции $a=',$a,', ';
Test_g();
echo "<br>";
echo 'вне функции $a=',$a,', ';
Test_g();
?>
```

Статические переменные

Чтобы использовать переменные только внутри функции, при этом сохраняя их значения и после выхода из функции, нужно объявить эти переменные как статические. ***Статические переменные*** видны только внутри функции и не теряют своего значения, если выполнение программы выходит за пределы функции. Объявление таких переменных производится с помощью ключевого слова *static*:

```
static $var1, $var2;
```

Статической переменной может быть присвоено любое значение, но не ссылка.

Возвращаемые значения

Все функции, приведенные выше в качестве примеров, выполняли какие-либо действия. Кроме подобных действий, любая функция может возвращать как результат своей работы какое-нибудь значение. Это делается с помощью утверждения *return*. Возвращаемое значение может быть любого типа, включая списки и объекты. Когда интерпретатор встречает команду *return* в *теле функции*, он немедленно прекращает ее исполнение и переходит на ту строку, из которой была вызвана функция.

Возвращение ссылки

В результате своей работы функция также может возвращать ссылку на какую-либо переменную. Это может пригодиться, если требуется использовать функцию для того, чтобы определить, какой переменной должна быть присвоена ссылка. Чтобы получить из функции ссылку, нужно при объявлении перед ее именем написать знак амперсанд (&) и каждый раз при вызове функции перед ее именем тоже писать амперсанд (&). Обычно функция возвращает ссылку на какую-либо *глобальную переменную* (или ее часть – ссылку на элемент глобального массива), ссылку на *статическую переменную* (или ее часть) или ссылку на один из аргументов, если он был также передан по ссылке.

Переменные функции

PHP поддерживает концепцию *переменных функций*. Это значит, что если имя переменной заканчивается круглыми скобками, то PHP ищет функцию с таким же именем и пытается ее выполнить.

Внутренние (встроенные) функции

Говоря о функциях, определяемых пользователем, все же нельзя не сказать пару слов о *встроенных функциях*. С некоторыми из *встроенных функций*, такими как echo(), print(), date(), include(), мы уже познакомились. На самом деле все перечисленные функции, кроме date(), являются языковыми конструкциями. Они входят в ядро PHP и не требуют никаких дополнительных настроек и модулей. Функция date() тоже входит в состав ядра PHP и не требует настроек. Но есть и функции, для работы с которыми нужно установить различные библиотеки и подключить соответствующий модуль. Например, для использования функций работы с базой данных MySql следует скомпилировать PHP с поддержкой этого расширения. В последнее время наиболее распространенные расширения и соответственно их функции изначально включают в состав PHP так, чтобы с ними можно работать без каких бы то ни было дополнительных настроек интерпретатора.

Решение задачи

Напомним, в чем состоит задача. Мы хотим написать интерфейс, который позволял бы создавать html-формы. Пользователь выбирает, какие элементы и в

каком количестве нужно создать, придумывает им названия, а наша программа сама генерирует требуемую форму.

5.6. Функции PHP

Массивы

В одной из первых лекций мы рассказывали о том, как можно *создать массив* данных. Напомним, что *массив* можно *создать* двумя способами:

С помощью конструкции array

```
$array_name = array("key1"=>"value1",
                    "key2"=>"value2");
```

Непосредственно задавая *значения* элементам *массива*

```
$array_name["key1"] = value1;
```

Например, нам нужно хранить список документов, которые будут удалены из базы данных. Естественно хранить его в виде *массива*, *ключом* в котором будет идентификатор документа (его уникальный номер), а *значением* – название документа. Этот *массив* можно *создать* таким образом:

Операции с массивами

Массив – это тип данных, с данными этого типа должны быть определены *операции*. Какие же *операции* можно производить с *массивами*? *Массивы* можно *складывать* и *сравнивать*.

Складывают массивы с помощью стандартного оператора «+». Вообще говоря, эту *операцию* по отношению к *массивам* точнее назвать объединением. Если у нас есть два *массива*, $a и $b, то результатом их *сложения* (объединения) будет *массив* $c, состоящий из элементов $a, к которым справа дописаны элементы *массива* $b. Причем, если встречаются совпадающие *ключи*, то в результирующий *массив* включается элемент из первого *массива*, т.е. из $a. Таким образом, если *складываются массивы* в языке PHP, от перемены мест слагаемых сумма меняется.

Функция count

Не раз уже мы использовали функцию *count()*, чтобы вычислить *количество элементов массива*. На самом деле эта функция вычисляет число элементов в переменной вообще. Если применить ее к любой другой переменной, она возвратит

1. Исключение составляет переменная типа NULL – *count(NULL)* есть 0. Кроме того, применяя эту функцию к многомерному *массиву*, чтобы получить *число его элементов*, нужно использовать дополнительный параметр COUNT_RECURSIVE.

Функция in_array

in_array("искомое значение","массив",
["ограничение на тип"]);

позволяет установить, содержится ли в заданном *массиве* искомое *значение*. Если третий аргумент задан как true, то в *массиве* нужно найти элемент, совпадающий с искомым не только по *значению*, но и по типу. Если искомое *значение* – строка, то сравнение чувствительно к регистру.

Функция array_search

Это еще одна функция для *поиска значения в массиве*. В отличие от *in_array* в результате работы ***array_search*** возвращает *значение ключа*, если элемент найден, и ложь – в противном случае. А вот синтаксис у этих функций одинаковый:

array_search("искомое значение","массив",
["ограничение на тип"]);

Сравнение строк чувствительно к регистру, а если указан опциональный аргумент, то сравниваются еще и типы *значений*. До PHP 4.2.0, если искомое *значение* не было найдено, эта функция возвращала ошибку или пустое *значение* NULL.

Функция array_keys

Функция *array_keys()* выбирает все *ключи массива*. Но у нее имеется дополнительный аргумент, с помощью которого можно получить список *ключей* элементов с конкретным *значением*. Синтаксис этой функции таков:

array_keys ("массив"
[,"значение для поиска"])

Функция *array_keys()* возвращает как строковые, так и числовые *ключи массива*, организуя все *значения* в виде нового *массива* с числовыми индексами.

Функция array_unique

Функция *array_unique(массив)* удаляет повторяющиеся *значения* из *массива* и возвращает новый *массив*. Таким образом, вместо нескольких одинаковых *значений* и их *ключей* мы имеем одно *значение*. Какой у него будет *ключ*? Как из нескольких *ключей* одинаковых элементов выбирается тот, который будет сохранен в новом *массиве*? Происходит следующее. Все элементы *массива* преобразуются в строки и *сортируются*. Затем обработчик запоминает первый *ключ* для каждого *значения*, а остальные *ключи* игнорирует.

Сортировка массивов

Необходимость *сортировки* данных, в том числе и данных, хранящихся в виде *массивов*, очень часто возникает при решении самых разнообразных задач. Если в языке Си для того, чтобы решить эту задачу, нужно написать десятки строк кода, то в PHP это делается одной простой командой.

Функция sort

Функция *sort* имеет следующий синтаксис

sort (массив [, флаги])

и *сортирует массив*, т.е. упорядочивает его *значения* по возрастанию. Эта функция удаляет все существовавшие в *массиве ключи*, заменяя их числовыми индексами, соответствующими новому порядку элементов. В случае успешного завершения работы она возвращает true, иначе – false.

Функции asort, rsort, arsort

Если требуется сохранять индексы элементов *массива* после *сортировки*, то нужно использовать функцию *asort (массив [, флаги])*. Если необходимо *отсортировать массив* в обратном порядке, т.е. от наибольшего *значения* к наименьшему, то можно задействовать функцию *rsort (массив [, флаги])*. А если при этом нужно еще и сохранить *значения ключей*, то следует использовать функцию *arsort(массив [, флаги])*. Как вы, наверное, заметили синтаксис у этих функций абсолютно такой же, как у функции *sort*. Соответственно и *значения* флагов могут быть такими же, как у *sort*: *SORT_REGULAR*, *SORT_NUMERIC*, *SORT_STRING*. Кстати говоря, флаг *SORT_NUMERIC* появился только в PHP4.

Сортировка массива по ключам

Очевидно, что может возникнуть необходимость в *сортировке массива по значениям ключей*. Например, если у нас есть *массив* данных о книгах, как в приведенном выше примере, то вполне вероятно, что мы захотим *отсортировать* книги по именам авторов. Для этого в PHP также не нужно писать много строк кода – можно просто воспользоваться функцией *ksort()* для *сортировки* по возрастанию (прямой порядок *сортировки*) или *krsort()* – для *сортировки* по убыванию (обратный порядок *сортировки*). Синтаксис этих функций опять же аналогичен синтаксису функции *sort()*.

Сортировка с помощью функции, заданной пользователем

Кроме двух простых способов *сортировки значений массива* (по убыванию или по возрастанию) PHP предлагает пользователю возможность самому задавать критерии для *сортировки* данных. Критерий задается с помощью функции, имя которой указывается в качестве аргумента для специальных функций *сортировки usort()* или *uksort()*. По названиям этих функций можно догадаться, что *usort() сортирует значения* элементов *массива*, а *uksort() – значения ключей массива* с помощью определенной пользователем функции. Обе функции возвращают true, если *сортировка* прошла успешно, и false – в противном случае. Их синтаксис выглядит следующим образом:

usort (массив , сортирующая функция)

uksort (массив , сортирующая функция)

Конечно же, нельзя *сортировать массив* с помощью любой пользовательской функции. Эта функция должна удовлетворять определенным критериям, позволяющим сравнивать элементы *массива*. Как должна быть устроена сортирующая функция? Во-первых, она должна иметь два аргумента. В них интерпретатор будет передавать пары *значений* элементов для функции *usort()* или *ключей массива* для функции *uksort()*. Во-вторых, сортирующая функция должна возвращать:

- целое число, меньшее нуля, если первый аргумент меньше второго;
- число, равное нулю, если два аргумента равны;
- число большее нуля, если первый аргумент больше второго.

Как и для других функций *сортировки*, для функции *usort()* существует аналог, не изменяющий *значения ключей*, – функция *uasort()*.

Применение функции ко всем элементам массива

Функция ***array_walk(массив, функция [, данные])*** применяет созданную пользователем функцию функция ко всем элементам *массива* массив и возвращает true в случае успешного выполнения операции и false – в противном случае.

Пользовательская функция, как правило, имеет два аргумента, в которые поочередно передаются *значение* и *ключ* каждого элемента *массива*. Но если при вызове функции *array_walk()* указан третий аргумент, то он будет рассмотрен как *значение* третьего аргумента пользовательской функции, смысл которого определяет сам пользователь. Если функция пользователя требует больше аргументов, чем в нее передано, то при каждом вызове *array_walk()* будет выдаваться предупреждение.

Если необходимо работать с реальными *значениями массива*, а не с их копиями, следует передавать аргумент в функцию по ссылке. Однако нужно иметь в виду, что нельзя добавлять или удалять элементы *массива* и производить действия, изменяющие сам *массив*, поскольку в этом случае результат работы *array_walk()* считается неопределенным.

Выделение подмассива

Функция array_slice

Поскольку *массив* – это набор элементов, вполне вероятно, потребуется выделить из него какой-нибудь поднабор. В PHP для этих целей есть функция *array_slice*. Ее синтаксис таков:

array_slice (массив,
    номер_элемента [, длина])

Эта функция *выделяет подмассив* длины длина в *массиве* массив, начиная с элемента, номер которого задан параметром номер_элемента. Положительный номер_элемента указывает на порядковый номер элемента относительно начала *массива*, отрицательный – на номер элемента с конца *массива*.

Функция array_chunk

Есть еще одна функция, похожая на *array_slice()* – это *array_chunk()*. Она разбивает *массив* на несколько подмассивов заданной длины. Синтаксис ее такой:

array_chunk (массив, размер [, сохранять_ключи])

В результате работы *array_chunk()* возвращает многомерный *массив*, элементы которого представляют собой полученные подмассивы. Если задать параметр сохранять *ключи* как true, то при разбиении будут сохранены *ключи* исходного *массива*. В противном случае *ключи* элементов заменяются числовыми индексами, которые начинаются с нуля.

Пример 7.15. У нас есть список приглашенных, оформленный в виде *массива* их фамилий. У нас имеются столики на три персоны. Поэтому нужно распределить всех приглашенных по трое.

Сумма элементов массива

В этом разделе мы познакомимся с функцией, вычисляющей *сумму всех элементов массива*. Сама задача вычисления *суммы значений массива* предельно проста. Но зачем писать лишний раз один и тот же код, если можно воспользоваться специально созданной и всегда доступной функцией. Функция эта называется, как можно догадаться, *array_sum()*. И в качестве параметра ей передается только имя *массива*, *сумму значений элементов* которого нужно вычислить.

В качестве примера использования этой функции приведем решение более сложной задачи, чем просто вычисление *суммы элементов*. Этот пример также иллюстрирует применение функции *array_slice()*, которую мы обсуждали чуть раньше.

Литератра

1. Савельева Н.В. **Основы программирования на PHP** Интернет-университет информационных технологий - ИНТУИТ.ру, 2005

2. Анисимов А.Е., Пупышев В.В. **Сборник заданий по основаниям программирования** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006
3. Непейвода Н.Н. **Стили и методы программирования** Интернет-университет информационных технологий - ИНТУИТ.ру, 2005
4. Сузи Р.А. **Язык программирования Python** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006
5. Терехов А.Н. **Технология программирования** БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2007

ТЕМА 14. ОСНОВЫ MYSQL (4 ЧАСОВ)

План.

6.1. Базы данных MySQL.

6.2. Принципы разработки с базами данных MySQL.

Базы данных MySQL.

Базы данных: основные понятия

В жизни мы часто сталкиваемся с необходимостью хранить какую-либо информацию, а потому часто имеем дело и с *базами данных*. Например, мы используем записную книжку для хранения номеров телефонов своих друзей и планирования своего времени. Телефонная книга содержит информацию о людях, живущих в одном городе. Все это своего рода *базы данных*. Ну а раз это *базы данных*, то посмотрим, как в них хранятся данные. Например, телефонная книга представляет собой таблицу (табл. 10.1).

В этой таблице данные – это собственно номера телефонов, адреса и ФИО., т.е. строки «Иванов Иван Иванович», «32-43-12» и т.п., а названия столбцов этой таблицы, т.е. строки «ФИО», «Номер телефона» и «Адрес» задают смысл этих данных, их семантику.

Таблица 10.1. Пример базы данных: телефонная книга		
ФИО	Номер телефона	Адрес

Иванов Иван Иванович	32-43-12	ул. Ленина, 12, 43
Ильин Федор Иванович	32-32-34	пр. Маркса, 32, 45

Теперь представьте, что записей в этой таблице не две, а две тысячи, вы занимаетесь созданием этого справочника и где-то произошла ошибка (например, опечатка в адресе). Видимо, тяжеловато будет найти и исправить эту ошибку вручную. Нужно воспользоваться какими-то средствами автоматизации. Для управления большим количеством данных программисты (не без помощи математиков) придумали системы управления *базами данных* (*СУБД*). По сравнению с текстовыми *базами данных* электронные *СУБД* имеют огромное число преимуществ, от возможности быстрого поиска информации, взаимосвязи данных между собой до использования этих данных в различных прикладных программах и одновременного доступа к данным нескольких пользователей.

Для точности дадим определение *базы данных*, предлагаемое Глоссарий.ру

База данных – это совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. *База данных* является информационной моделью предметной области. Обращение к *базам данных* осуществляется с помощью системы управления *базами данных* (**СУБД**). *СУБД* обеспечивает поддержку создания *баз данных*, централизованного управления и организации доступа к ним различных пользователей.

Итак, мы пришли к выводу, что хранить данные независимо от программ, так, что они связаны между собой и организованы по определенным правилам, целесообразно. Но вопрос, как хранить данные, по каким правилам они должны быть организованы, остался открытым. Способов существует множество (кстати, называются они моделями представления или хранения данных). Наиболее популярные – объектная и *реляционная* модели данных.

Автором **реляционной модели** считается Э. Кодд, который первым предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение) и показал, что любое представление

данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение.

Таким образом, реляционная *база данных* представляет собой набор таблиц (точно таких же, как приведенная выше), связанных между собой. Строка в таблице соответствует сущности реального мира (в приведенном выше примере это информация о человеке).

Примеры реляционных *СУБД*: *MySql*, PostgreSql.

В основу **объектной модели** положена концепция объектно-ориентированного программирования, в которой данные представляются в виде набора объектов и классов, связанных между собой родственными отношениями, а работа с объектами осуществляется с помощью скрытых (инкапсулированных) в них методов.

Примеры объектных *СУБД*: Cache, GemStone (от Servio Corporation), ONTOS (ONTOS).

В последнее время производители *СУБД* стремятся соединить два этих подхода и проповедуют объектно-реляционную модель представления данных. Примеры таких *СУБД* – IBM DB2 for Common Servers, Oracle8.

Поскольку мы собираемся работать с *Mysql*, то будем обсуждать аспекты работы только с реляционными *базами данных*. Нам осталось рассмотреть еще два важных понятия из этой области: *ключи* и *индексирование*, после чего мы сможем приступить к изучению языка запросов *SQL*.

Ключи

Для начала давайте подумаем над таким вопросом: какую информацию нужно дать о человеке, чтобы собеседник точно сказал, что это именно тот человек, сомнений быть не может, второго такого нет? Сообщить фамилию, очевидно, недостаточно, поскольку существуют однофамильцы. Если собеседник человек, то мы можем приблизительно объяснить, о ком речь, например вспомнить поступок, который совершил тот человек, или еще как-то. Компьютер же такого объяснения не поймет, ему нужны четкие правила, как определить, о ком идет речь. В системах управления *базами данных* для решения такой задачи ввели понятие *первичного ключа*.

Первичный ключ (primary key, PK) – минимальный набор полей, уникально идентифицирующий запись в таблице. Значит, *первичный ключ* – это в первую очередь набор полей таблицы, во-вторых, каждый набор значений этих полей должен определять единственную запись (строку) в таблице и, в-третьих, этот набор полей должен быть минимальным из всех обладающих таким же свойством. Поскольку *первичный ключ* определяет только одну уникальную запись, то никакие две записи таблицы не могут иметь одинаковых значений *первичного ключа*.

Например, в нашей таблице (см. выше) ФИО и адрес позволяют однозначно выделить запись о человеке. Если же говорить в общем, без связи с решаемой задачей, то такие знания не позволяют точно указать на единственного человека, поскольку существуют однофамильцы, живущие в разных городах по одному адресу. Все дело в границах, которые мы сами себе задаем. Если считаем, что знания ФИО, телефона и адреса без указания города для наших целей достаточно, то все замечательно, тогда поля ФИО и адрес могут образовывать *первичный ключ*. В любом случае проблема создания *первичного ключа* ложится на плечи того, кто проектирует *базу данных* (разрабатывает структуру хранения данных). Решением этой проблемы может стать либо выделение характеристик, которые естественным образом определяют запись в таблице (задание так называемого логического, или естественного, PK), либо создание дополнительного поля, предназначенного именно для однозначной идентификации записей в таблице (задание так называемого суррогатного, или искусственного, PK). Примером логического *первичного ключа* является номер паспорта в *базе данных* о паспортных данных жителей или ФИО и адрес в телефонной книге (таблица выше). Для задания суррогатного *первичного ключа* в нашу таблицу можно добавить поле id (идентификатор), значением которого будет целое число, уникальное для каждой строки таблицы. Использование таких суррогатных *ключей* имеет смысл, если естественный *первичный ключ* представляет собой большой набор полей или его выделение нетривиально.

Кроме однозначной идентификации записи, *первичные ключи* используются для организации связей с другими таблицами.

Например, у нас есть три таблицы: содержащая информацию об исторических личностях (Persons), содержащая информацию об их изобретениях (Artifacts) и содержащая изображения как личностей, так и артефактов (Images) (рис 10.1).

Первичным ключом во всех этих таблицах является поле id (идентификатор). В таблице Artifacts есть поле author, в котором записан идентификатор, присвоенный автору изобретения в таблице Persons. Каждое значение этого поля является **внешним ключом** для *первичного ключа* таблицы Persons. Кроме того, в таблицах Persons и Artifacts есть поле photo, которое ссылается на изображение в таблице Images. Эти поля также являются *внешними ключами* для *первичного ключа* таблицы Images и устанавливают однозначную логическую связь Persons-Images и Artifacts-Images. То есть если значение *внешнего ключа* photo в таблице личности равно 10, то это значит, что фотография этой личности имеет id=10 в таблице изображений. Таким образом, **внешние ключи** используются для организации связей между таблицами *базы данных* (родительскими и дочерними) и для поддержания ограничений ссылочной целостности данных.

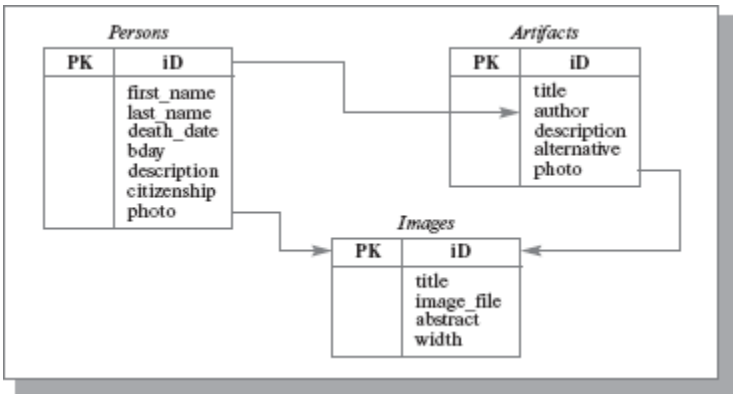


Рис. 10.1. Пример использования первичных ключей для организации связей с другими таблицами

Индексирование

Одна из основных задач, возникающих при работе с *базами данных*, – это задача поиска. При этом, поскольку информации в *базе данных*, как правило, содержится много, перед программистами встает задача не просто поиска, а эффективного поиска, т.е. поиска за сравнительно небольшое время и с достаточной точностью. Для этого (для оптимизации производительности запросов) производят **индексирование** некоторых полей таблицы. Использовать индексы полезно для

быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется по всей таблице, начиная с первой записи, пока не будут найдены соответствующие строки. Чем больше таблица, тем больше накладные расходы. Если же таблица содержит индекс по рассматриваемым столбцам, то *база данных* может быстро определить позицию для поиска в середине файла данных без просмотра всех данных. Это происходит потому, что *база данных* помещает проиндексированные поля поближе в памяти, так, чтобы можно было побыстрее найти их значения. Для таблицы, содержащей 1000 строк, это будет как минимум в 100 раз быстрее по сравнению с последовательным перебором всех записей. Однако в случае, когда необходим доступ почти ко всем 1000 строкам, быстрее будет последовательное чтение, так как при этом не требуется операций поиска по диску. Так что иногда индексы бывают только помехой. Например, если копируется большой объем данных в таблицу, то лучше не иметь никаких индексов. Однако в некоторых случаях требуется задействовать сразу несколько индексов (например, для обработки запросов к часто используемым таблицам).

Если говорить о *MySQL*, то там существует три вида индексов: *PRIMARY*, *UNIQUE*, и *INDEX*, а слово ключ (KEY) используется как синоним слова индекс (*INDEX*). Все индексы хранятся в памяти в виде *B-деревьев*.

PRIMARY – уникальный индекс (ключ) с ограничением, что все индексированные им поля не могут иметь пустого значения (т.е. они NOT NULL). Таблица может иметь только один первичный индекс, но он может состоять из нескольких полей.

UNIQUE – ключ (индекс), задающий поля, которые могут иметь только уникальные значения.

INDEX – обычный индекс (как мы описали выше). В *MySqL*, кроме того, можно индексировать строковые поля по заданному числу символов от начала строки.

6.2. Принципы разработки с базами данных MySQL.

Продолжим разговор о *СУБД MySQL*. ***MySQL*** – это реляционная система управления *базами данных*. То есть данные в ее базах хранятся в виде логически

связанных между собой таблиц, доступ к которым осуществляется с помощью языка запросов *SQL*. *MySQL* – свободно распространяемая система, т.е. платить за ее применение не нужно. Кроме того, это достаточно быстрая, надежная и, главное, простая в использовании *СУБД*, вполне подходящая для не слишком глобальных проектов.

Работать с *MySQL* можно не только в текстовом режиме, но и в графическом. Существует очень популярный визуальный интерфейс (кстати, написанный на PHP) для работы с этой *СУБД*. Называется он ***PhpMyAdmin***. Этот интерфейс позволяет значительно упростить работу с *базами данных* в *MySQL*.

В текстовом режиме работа с *базой данных* выглядит просто как ввод команд в командную строку (рис 10.2), а результаты выборок возвращаются в виде своеобразных таблиц, поля в которых налезают друг на друга, если данные не помещаются на экран (рис 10.3).

Рис. 10.2. Работа с MySQL в коммандной строке. Команда show databases — вывести все имеющиеся базы данных

PhpMyAdmin позволяет пользоваться всеми достоинствами браузера, включая прокрутку изображения, если оно не умещается на экран. Многие из базовых *SQL*-функций работы с данными в *PhpMyAdmin* сведены к интуитивно понятным интерфейсам и действиям, напоминающим переход по ссылкам в Internet. Но тем не менее стоит все же поработать и в текстовом режиме.

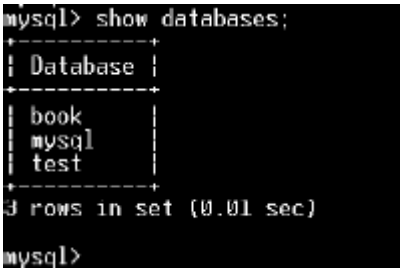


Рис. 10.3. Работа с MySQL в коммандной строке. Результат обработки команды show databases

Перед тем как переходить к детальному изучению языка *SQL*, несколько слов об установке *MySQL* и подготовке к работе. Если вы не собираетесь заниматься

администрированием сервера, то информация, приведенная ниже, пригодится вам только для общего развития. Итак, устанавливается *MySQL* очень просто – автоматически, пару раз нажмите OK, и все. После этого вы можете зайти в директорию, где лежат файлы типа mysql.exe, mysqld.exe и т.п. (у нас под Windows XP это c:\mysql\bin) Последний файл запускает Mysql-сервер. В некоторых системах сервер запускается в виде сервиса. После запуска сервера следует запустить mysql-клиент, запустив программу mysql.exe. Здесь даже пароля не спросят. Более того, если вы наберете

```
shell> mysql.exe -u root
```

или

```
shell>mysql -u root mysql
```

то получите все права администратора mysql сервера. Кстати, выполнять эти команды надо, находясь в той директории, где лежат файлы mysql.exe.

Для начала, не вдаваясь в подробности команд, исправим эти два недочета (отсутствие пароля у администратора и возможность входа анонимным пользователям):

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
    WHERE user='root';
mysql> DELETE FROM user WHERE user='';
mysql> FLUSH PRIVILEGES;
```

Все данные о пользователях *MySQL* хранит в таблице user в специальной *базе данных* mysql, доступ к которой имеет только администратор сервера. Поэтому, чтобы изменить какой-либо пароль, нужно изменить эту таблицу. Пароль задается с помощью функции PASSWORD, которая кодирует введенные данные. Кроме изменения пароля администратора, нужно еще удалить всех пользователей, не имеющих логина (команда *DELETE*). Команда Flush Privileges заставляет вступить в действие изменения, произошедшие в системной *базе данных* (mysql).

Теперь создадим *базу данных*, с которой будем работать (мы все еще работаем как администратор сервера):

```
mysql>create database book;
```

Как можно заметить, все команды в *MySQL* заканчиваются точкой с запятой. Если вы забыли поставить этот знак, то выдается приглашение его поставить до тех пор, пока это не будет сделано:

```
mysql> show tables
    ->
    ->
```

Теперь последнее действие – создадим простого пользователя, предоставим ему доступ к созданной *базе данных*, и начнем работать.

```
mysql> GRANT ALL PRIVILEGES ON book.* TO nina@localhost
    IDENTIFIED BY '123';
```

Команда GRANT наделяет пользователя nina, зашедшего на сервер с этой же машины (с localhost) и идентифицируемого паролем «123», определенными правами (в данном случае всеми) на все таблицы *базы данных* book. Теперь мы можем выйти и зайти как пользователь nina с соответствующим паролем:

```
shell>mysql -u nina -p
Enter password: ***
Welcome to the MySQL monitor!...
mysql>
```

Если вы собираетесь пользоваться *базой данных* на чужом сервере, то его администратор проделает все описанные выше действия за вас, т.е. все настроит и создаст пользователя и *базу данных*. В следующей главе описаны команды языка *SQL*, которые пригодятся для работы с данными, хранящимися в *СУБД MySQL*.

Язык SQL

Итак, мы в общих чертах познакомились с основными понятиями теории *баз данных*, установили и настроили для работы *MySQL*. Теперь самое время научиться манипулировать данными, хранящимися в *базах данных*. Для этого нам понадобится ***SQL*** – структурированный язык запросов. Этот язык дает возможность создавать, редактировать и удалять информацию, хранящуюся в *базах данных*, создавать новые *базы данных* и многое другое. *SQL* является стандартом ANSI (Американский

национальный институт стандартов) и ISO (Международная организация по стандартизации).

Немного истории

Первый международный стандарт языка *SQL* был принят в 1989 г., его часто называют *SQL/89*. Среди недостатков этого стандарта выделяют в первую очередь то, что многие важные свойства он устанавливал как определяемые в реализации. Отсюда произошло множество расхождений в реализациях языка разными производителями. Кроме того, высказывались претензии по поводу отсутствия в этом стандарте упоминаний о практических аспектах языка, таких как его встраивание в язык программирования Си.

Следующий международный стандарт языка *SQL* был принят в конце 1992 г. И стал называться *SQL/92*. Он получился гораздо более точным и полным, чем *SQL/89*, хотя и не был лишен недостатков. В настоящее время большинство систем почти полностью реализуют этот стандарт. Однако, как известно, прогресс не остановишь, и в 1999 году появился новый стандарт *SQL*:1999, также известный как SQL3. SQL3 характеризуется как «объектно-ориентированный *SQL*» и является основой нескольких объектно-реляционных систем управления *базами данных* (например, ORACLE8 компании Oracle, Universal Server компании Informix и DB2 Universal Database компании IBM). Этот стандарт является не просто слиянием SQL-92 и объектной технологии. Он содержит ряд расширений традиционного *SQL*, а сам документ составлен таким образом, чтобы добиться более эффективной работы в области стандартизации в будущем.

Если говорить о *MySQL*, то она соответствует начальному уровню SQL92, содержит несколько расширений этого стандарта и стремится к полной поддержке стандарта ANSI SQL99, но без ущерба для скорости и качества кода.

Далее, говоря об основах языка *SQL*, будем придерживаться его реализации в *СУБД MySQL*.

Основные операторы языка SQL

Функции любой *СУБД* включают:

создание, удаление, изменение *базы данных* (БД);

добавление, изменение, удаление, назначение прав пользователя;

внесение, удаление и изменение данных в БД (таблиц и записей);

выборку данных из БД.

К первым двум функциям имеют доступ только администраторы *СУБД* или привилегированные пользователи. Рассмотрим, как решаются последние две задачи (на самом деле это семь задач).

Прежде чем что-либо делать с данными, нужно создать таблицы, в которых эти данные будут храниться, научиться изменять структуру этих таблиц и удалять их, если потребуется. Для этого в языке *SQL* существуют операторы *CREATE TABLE*, *ALTER TABLE* и *DROP TABLE*.

Оператор CREATE TABLE

Оператор ***CREATE TABLE*** создает таблицу с заданным именем в текущей *базе данных*. Правила для допустимых имен таблицы приведены в документации. Если нет активной текущей *базы данных* или указанная таблица уже существует, то возникает ошибка выполнения команды.

В версии *MySQL* 3.22 и более поздних имя таблицы может быть указано как имя_базы_данных.имя_таблицы. Эта форма записи работает независимо от того, является ли указанная *база данных* текущей.

В версии *MySQL* 3.23 при создании таблицы можно использовать ключевое слово TEMPORARY. Временная таблица автоматически удаляется по завершении соединения, а ее имя действительно только в течение данного соединения. Это означает, что в двух разных соединениях могут использоваться временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем (существующая таблица скрыта, пока не удалена временная таблица).

В версии *MySQL* 4.0.2 для создания временных таблиц необходимо иметь привилегии CREATE TEMPORARY TABLES.

В версии *MySQL* 3.23 и более поздних можно использовать ключевые слова IF NOT EXISTS для того, чтобы не возникала ошибка, если указанная таблица уже существует. Следует учитывать, что при этом идентичность структур этих таблиц не проверяется.

Каждая таблица представлена набором определенных файлов в директории *базы данных*.

Синтаксис

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
    имя_таблицы [(определение_столбца,...)]
   [опции_таблицы] [select_выражение]
```

В выражении определение_столбца перечисляют, какие столбцы должны быть созданы в таблице. Каждый столбец таблицы может быть пустым (NULL), иметь значение по умолчанию, являться ключом или автоинкрементом. Кроме того, для каждого столбца обязательно указывается тип данных, которые будут в нем храниться. Если не указывается ни NULL, ни NOT NULL, то столбец интерпретируется так, как будто указано NULL. Если поле помечают как автоинкремент (AUTO_INCREMENT), то его значение автоматически увеличивается на единицу каждый раз, когда происходит добавление данных в таблицу и в это поле записывается пустое значение (NULL, т.е. ничего не записывается) или 0. Автоинкремент в таблице может быть только один, и при этом он обязательно должен быть проиндексирован. Последовательность AUTO_INCREMENT начинается с 1. Наличие автоинкремента является одной из особенностей *MySQL*. Формально описание столбца (определение_столбца) выглядит так:

```
имя_столбца тип [NOT NULL | NULL]
   [DEFAULT значение_по_умолчанию]
   [AUTO_INCREMENT][PRIMARY KEY]
   [reference_definition]
```

Тип столбца (тип в выражении определение_столбца) может быть одним из следующих:

целый: INT[(length)] [UNSIGNED] [ZEROFILL]

действительный: REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]

символьный: CHAR(length) [BINARY] и VARCHAR(length) [BINARY]

дата и время: DATE и TIME

для работы с большими объектами: BLOB

текстовый: TEXT

перечислимое множество: ENUM(value1,value2,value3,...) и SET(value1,value2,value3,...)

Полный список типов смотрите в документации *MySQL*.

Вместо перечисления столбцов и их свойств в определении_столбца можно задавать списки ключевых и индексных полей, ограничения и проверки:

PRIMARY KEY (имя_индексируемого_столбца, ...)

или

KEY [имя_индекса] (имя_индексируемого_столбца,...)

или

INDEX [имя_индекса] (имя_индексируемого_столбца,...)

или

UNIQUE [INDEX] [имя_индекса]
(имя_индексируемого_столбца,...)

или

FULLTEXT [INDEX] [имя_индекса]
(имя_индексируемого_столбца,...)

или

[CONSTRAINT symbol]
FOREIGN KEY [имя_индекса]
(имя_индексируемого_столбца,...)
[reference_definition]

или

CHECK (expr)

При задании всех этих элементов указывается список полей (столбцов), которые будут входить в индекс, ключ или ограничение, имя_индексируемого_столбца записывается следующим образом:

имя_столбца [(длина_индекса)]

FOREIGN KEY, CHECK и REFERENCES на самом деле ничего не делают в *MySQL*. Они добавлены только для совместимости с другими SQL-серверами. Поэтому на них мы останавливаться не будем.

Кроме всего перечисленного, при создании таблицы можно указать некоторые ее свойства (опции_таблицы), например такие:

тип таблицы: TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }

начальное значение счетчика автоинкремента: AUTO_INCREMENT = число

средняя длина строк в таблице: AVG_ROW_LENGTH = число

комментарии к таблице (строка из 60 символов): COMMENT = "строка"

максимальное и минимальное предполагаемое число строк: MAX_ROWS = число и MIN_ROWS = число

И последний (опять же опциональный) элемент команды CREATE – это выражение *SELECT* (select_выражение). Синтаксис такой:

```
[IGNORE | REPLACE] SELECT ...
   (любое корректное выражение SELECT)
```

Если при создании таблицы в команде CREATE указывается выражение *SELECT*, то все поля, полученные выборкой, добавляются в создаваемую таблицу.

Пример 10.1. Создадим таблицу Persons, структура которой была приведена на рисунке 10.1.

```
mysql>CREATE TABLE Persons
   (id INT PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(50), last_name
   VARCHAR(100), death_date INT,
  description TEXT, photo INT,
  citienship CHAR(50) DEFAULT 'Russia');
```

Пример 10.1. Создание таблицы Persons (html, txt)

С помощью специфичной для *MySql* команды SHOW можно просмотреть существующие *базы данных*, таблицы в *базе данных* и поля в таблице.

Показать все *базы данных*:

mysql>SHOW databases;

Сделать текущей *базу данных* book и показать все таблицы в ней:

mysql>use book;

mysql>show tables;

Показать все столбцы в таблице Persons:

mysql> show columns from Persons;

Оператор DROP TABLE

Оператор ***DROP TABLE*** удаляет одну или несколько таблиц. Все табличные данные и определения удаляются, так что при работе с этой командой следует соблюдать осторожность.

Синтаксис:

```
DROP TABLE [IF EXISTS] имя_таблицы
   [, имя_таблицы,...]
   [RESTRICT | CASCADE]
```

В версии *MySQL* 3.22 и более поздних можно использовать ключевые слова IF EXISTS, чтобы предупредить ошибку, если указанные таблицы не существуют.

Опции RESTRICT и CASCADE позволяют упростить перенос программы с других *СУБД*. В данный момент они не задействованы.

```
mysql> DROP TABLE IF EXISTS Persons,
     Artifacts, test;
```

Пример 10.2. Использование оператора DROP TABLE (html, txt)

Оператор ALTER TABLE

Оператор ***ALTER TABLE*** обеспечивает возможность изменять структуру существующей таблицы. Например, можно добавлять или удалять столбцы, создавать или уничтожать индексы или переименовывать столбцы либо саму таблицу. Можно также изменять комментарий для таблицы и ее тип.

Синтаксис:

```
ALTER [IGNORE] TABLE имя_таблицы
   alter_specification
   [, alter_specification ...]
```

Можно производить следующие изменения в таблице (все они записываются в alter_specification):

добавление поля:

ADD [COLUMN] определение_столбца
[FIRST | AFTER имя_столбца]

или

ADD [COLUMN] (определение_столбца,
определение_столбца,...)

Здесь, как и далее, определение_столбца записывается так же, как при создании таблицы.

добавление индексов:

ADD *INDEX* [имя_индекса] (имя_индексируемого_столбца,...) или ADD *PRIMARY* KEY (имя_индексируемого_столбца,...) или ADD *UNIQUE* [имя_индекса] (имя_индексируемого_столбца,...) или ADD FULLTEXT [имя_индекса] (имя_индексируемого_столбца,...)

изменение поля:

ALTER [COLUMN] имя_столбца {SET DEFAULT literal | DROP DEFAULT} или CHANGE [COLUMN] старое_имя_столбца определение_столбца или MODIFY [COLUMN] определение_столбца

удаление поля, индекса, ключа:

DROP [COLUMN] имя_столбца
DROP PRIMARY KEY
DROP INDEX имя_индекса

переименование таблицы:

RENAME [TO] новое_имя_таблицы

переупорядочение полей таблицы:

ORDER BY поле

или

опции_таблицы

Если оператор *ALTER TABLE* используется для изменения определения типа столбца, но DESCRIBE имя_таблицы показывает, что столбец не изменился, то, возможно, *MySQL* игнорирует данную модификацию по одной из причин, описанных в специальном разделе документации. Например, при попытке изменить столбец VARCHAR на CHAR *MySQL* будет продолжать использовать VARCHAR, если данная таблица содержит другие столбцы с переменной длиной.

Оператор *ALTER TABLE* во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая переименовывается. Это делается для того, чтобы в новую таблицу автоматически попадали все обновления, кроме неудавшихся. Во время выполнения *ALTER TABLE* исходная таблица доступна для чтения другими клиентами. Операции обновления и записи в этой таблице приостанавливаются, пока не будет готова новая таблица. Следует отметить, что при использовании любой другой опции для ALTER TABLE, кроме RENAME, *MySQL* всегда будет создавать временную таблицу, даже если данные, строго говоря, и не нуждаются в копировании (например, при изменении имени столбца).

Пример10.3. Добавим в созданную таблицу Persons поле для записи года рождения человека:

```
mysql> ALTER TABLE Persons
   ADD bday INTEGER AFTER last_name;
```

Пример 10.3. Добавление в таблицу Persons поля для записи года рождения человека (html, txt)

Итак, мы научились работать с таблицами: создавать, удалять и изменять их. Теперь разберемся, как делать то же самое с данными, которые в этих таблицах хранятся.

Оператор SELECT

Оператор ***SELECT*** применяется для извлечения строк, выбранных из одной или нескольких таблиц. То есть с его помощью мы задаем столбцы или выражения, которые надо извлечь (select_выражения), таблицы (table_references), из которых должна производиться выборка, и, возможно, условие (where_definition), которому

должны соответствовать данные в этих столбцах, и порядок, в котором эти данные нужно выдать.

Кроме того, оператор *SELECT* можно использовать для извлечения строк, вычисленных без ссылки на какую-либо таблицу. Например, чтобы вычислить, чему равно 2*2, нужно просто написать

```
mysql> SELECT 2*2;
```

Упрощенно структуру оператора *SELECT* можно представить следующим образом:

```
SELECT select_выражение1, select_выражение2,
  ...
  [FROM table_references
    [WHERE where_definition]
    [ORDER BY {число | имя_столбца |
        формула}
      [ASC | DESC], ...]]
```

Квадратные скобки [ ] означают, что использование находящегося в них оператора необязательно, вертикальная черта | означает перечисление возможных вариантов. После ключевого слова ORDER BY указывают имя столбца, число (целое беззнаковое) или формулу и способ упорядочения (по возрастанию – ASC, или по убыванию – DESC). По умолчанию используется упорядочение по возрастанию.

Когда в select_выражении мы пишем «*», это значит выбрать все столбцы. Кроме «*» в select_выражения могут использоваться функции типа max, min и avg.

Пример 10.4. Выбрать из таблицы Persons все данные, для которых поле first_name имеет значение 'Александр':

```
mysql> SELECT * FROM Persons
    WHERE first_name='Александр';
```

Пример 10.4. Использование оператора SELECT (html, txt)

Выбрать название и описание (title, description) артефакта под номером 10:

```
mysql> SELECT title,description
  FROM Artifacts WHERE id=10;
```

Оператор INSERT

Оператор ***INSERT*** вставляет новые строки в существующую таблицу. Оператор имеет несколько форм. Параметр имя_таблицы во всех этих формах задает таблицу, в которую должны быть внесены строки. Столбцы, для которых задаются значения, указываются в списке имен столбцов (имя_столбца) или в части SET.

Синтаксис:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
   [INTO] имя_таблицы [(имя_столбца,...)]
   VALUES (выражение,...),(...),...
```

Эта форма команды *INSERT* вставляет строки в соответствии с точно указанными в команде значениями. В скобках после имени таблицы перечисляются столбцы, а после ключевого слова VALUES – их значения.

Например:

```
mysql> INSERT INTO Persons
   (last_name, bday) VALUES
   ('Иванов', '1934');
```

вставит в таблицу Persons строку, в которой значения фамилии (last_name) и даты рождения (bday) будут заданы соответственно как «Иванов» и «1934».

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
   [INTO] имя_таблицы [(имя_столбца,...)]
   SELECT ...
```

Эта форма команды *INSERT* вставляет строки, выбранные из другой таблицы или таблиц.

Например:

```
mysql> INSERT INTO Artifacts (author)
      SELECT id FROM Persons
        WHERE last_name='Иванов'
          AND bday='1934';
```

вставит в таблицу Artifacts в поле «автор» (author) значение идентификатора, выбранного из таблицы Persons по условию, что фамилия человека Иванов.

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
```

[INTO] имя_таблицы

SET имя_столбца=выражение,

имя_столбца=выражение, ...

Например:

```
mysql> INSERT INTO Persons
        SET last_name='Петров',
            first_name='Иван';
```

Эта команда вставит в таблицу Persons в поле last_name значение «Петров», а в поле first_name – строку «Иван».

Форма *INSERT* ... VALUES со списком из нескольких значений поддерживается в версии *MySQL* 3.22.5 и более поздних. Синтаксис выражения имя_столбца=выражение поддерживается в версии *MySQL* 3.22.10 и более поздних.

Действуют следующие соглашения.

Если не указан список столбцов для *INSERT* ... VALUES или *INSERT* ... *SELECT*, то величины для всех столбцов должны быть определены в списке VALUES() или в результате работы *SELECT*. Если порядок столбцов в таблице неизвестен, для его получения можно использовать DESCRIBE имя_таблицы.

Любой столбец, для которого явно не указано значение, будет установлен в свое значение по умолчанию. Например, если в заданном списке столбцов не указаны все столбцы в данной таблице, то не упомянутые столбцы устанавливаются в свои значения по умолчанию.

Выражение expression может относиться к любому столбцу, который ранее был внесен в список значений. Например, можно указать следующее:

```
mysql> INSERT INTO имя_таблицы (col1,col2)
    VALUES(15,col1*2);
```

Но нельзя указать:

```
mysql> INSERT INTO имя_таблицы (col1,col2)
    VALUES(col2*2,15);
```

Мы еще не обсудили три необязательных параметра, присутствующих во всех трех формах команды: LOW_PRIORITY, DELAYED и IGNORE.

Параметры LOW_PRIORITY и DELAYED используются, когда с таблицей работает большое число пользователей. Они предписывают устанавливать приоритет данной операции перед операциями других пользователей. Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды *INSERT* будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. В этом случае клиент должен ожидать, пока данная команда вставки не будет завершена, что в случае интенсивного использования таблицы может потребовать значительного времени. В противоположность этому команда *INSERT* DELAYED позволяет данному клиенту продолжать операцию сразу же, независимо от других пользователей.

Если в команде *INSERT* указывается ключевое слово IGNORE, то все строки, имеющие дублирующиеся ключи *PRIMARY* или *UNIQUE* в этой таблице, будут проигнорированы и не внесены в таблицу. Если не указывать IGNORE, то данная операция вставки прекращается при обнаружении строки, имеющей дублирующееся значение существующего ключа.

Оператор UPDATE

Синтаксис:

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы
   SET имя_столбца1=выражение1
  [, имя_столбца2=выражение2, ...]
   [WHERE where_definition]
   [LIMIT число]
```

Оператор ***UPDATE*** обновляет значения существующих столбцов таблицы в соответствии с введенными значениями. В выражении SET указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В выражении WHERE, если оно присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды *UPDATE* задерживается до тех пор, пока другие клиенты не завершат чтение этой таблицы.

Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

Если в выражении, которое задает новое значение столбца, используется имя этого поля, то команда *UPDATE* использует для этого столбца его текущее значение. Например, следующая команда устанавливает столбец death_date в значение, на единицу большее его текущей величины:

```
mysql> UPDATE Persons
    SET death_date=death_date+1;
```

В версии *MySQL* 3.23 можно использовать параметр LIMIT #, чтобы убедиться, что было изменено только заданное количество строк.

Например, такая операция заменит в первой строке нашей таблицы экспонатов название title на строку «Ламповая ЭВМ»:

```
mysql> UPDATE Artifacts
    SET title='Ламповая ЭВМ' Limit 1;
```

Оператор DELETE

Оператор ***DELETE*** удаляет из таблицы имя_таблицы строки, удовлетворяющие заданным в where_definition условиям, и возвращает число удаленных записей.

Если оператор *DELETE* запускается без определения WHERE, то удаляются все строки.

Синтаксис:

```
DELETE [LOW_PRIORITY] FROM имя_таблицы
  [WHERE where_definition]
  [LIMIT rows]
```

Например, следующая команда удалит из таблицы Persons все записи, у которых поле «год рождения» (bday) больше 2003:

```
mysql> DELETE FROM Persons WHERE bday>2003;
```

Удалить все записи в таблице можно еще и с помощью такой команды:

mysql> DELETE FROM Persons WHERE 1>0;

Но этот метод работает гораздо медленнее, чем использование той же команды без условия:

mysql> DELETE FROM Persons;

Специфическая для *MySQL* опция LIMIT для команды *DELETE* указывает серверу максимальное количество строк, которые следует удалить до возврата управления клиенту. Эта опция может использоваться для гарантии того, что данная команда *DELETE* не потребует слишком много времени для выполнения.

Заключение

Итак, мы разобрались с основами реляционных *баз данных*, научились создавать простые и не очень SQL-запросы. Надеюсь, что большое количество технических деталей не помешало читателям получить представление о базовых элементах языка, поскольку все это наверняка пригодится для решения практических задач.

В дистрибутив PHP входит расширение, содержащее встроенные функции для работы с базой данных *MySQL*. В этой лекции мы познакомимся с некоторыми основными функциями для работы с *MySQL*, которые потребуются для решения задач построения web-интерфейсов с целью отображения и наполнения базы данных. Возникает вопрос, зачем строить такие интерфейсы? Для того чтобы вносить информацию в базу данных и просматривать ее содержимое могли люди, не знакомые с языком запросов SQL. При работе с web-интерфейсом для добавления информации в базу данных человеку нужно просто ввести эти данные в *html-форму* и отправить их на сервер, а наш скрипт сделает все остальное. А для просмотра содержимого таблиц достаточно просто щелкнуть по ссылке и зайти на нужную страницу.

Для наглядности будем строить эти интерфейсы для таблицы Artifacts, в которой содержится информация об экспонатах виртуального музея информатики. В предыдущей лекции мы уже приводили структуру этой коллекции, а также ее связи с коллекциями описания персон (Persons) и изображений (Images). Напомним, что

каждый экспонат в коллекции Artifacts описывается с помощью следующих характеристик:

название (title);

автор (author);

описание (description);

альтернативное название (alternative);

изображение (photo).

Название и альтернативное название являются строками менее чем 255 символов длиной (т.е. имеют тип VARCHAR(255)), описание - текстовое поле (имеет тип TEXT), а в полях "автор" и "изображение" содержатся идентификаторы автора из коллекции Persons и изображения экспоната из коллекции Images соответственно.

Построение интерфейса для добавления информации

Итак, у нас есть какая-то таблица в базе данных. Чтобы построить интерфейс для добавления информации в эту таблицу, нужно ее структуру (т.е. набор ее полей) отобразить в *html-форму*.

Разобьем эту задачу на следующие подзадачи:

установка соединения с БД;

выбор рабочей БД;

получение списка полей таблицы;

отображение полей в *html-форму*.

После этого данные, введенные в форму, нужно записать в базу данных. Рассмотрим все эти задачи по порядку.

Установка соединения

Итак, первое, что нужно сделать, - это установить соединение с базой данных. Воспользуемся функцией *mysql_connect*.

Синтаксис *mysql_connect*

```
ресурс mysql_connect ( [строка server
   [, строка username [, строка password
  [, логическое new_link
   [, целое client_flags]]]]])
```

Данная функция устанавливает соединение с сервером *MySQL* и возвращает указатель на это соединение или FALSE в случае неудачи. Для отсутствующих параметров устанавливаются следующие значения по умолчанию:

server = 'localhost:3306'

username = имя пользователя владельца процесса сервера

password = пустой пароль

Если функция вызывается дважды с одними и теми же параметрами, то новое соединение не устанавливается, а возвращается ссылка на старое соединение. Чтобы этого избежать, используют параметр new_link, который заставляет в любом случае открыть еще одно соединение.

Параметр client_flags - это комбинация следующих констант: *MYSQL_CLIENT_COMPRESS* (использовать протокол сжатия), *MYSQL_CLIENT_IGNORE_SPACE* (позволяет вставлять пробелы после имен функций), *MYSQL_CLIENT_INTERACTIVE* (ждать interactive_timeout секунд - вместо wait_timeout - до закрытия соединения).

Параметр new_link появился в PHP 4.2.0, а параметр client_flags - в PHP 4.3.0.

Соединение с сервером закрывается при завершении исполнения скрипта, если оно до этого не было закрыто с помощью функции *mysql_close()*.

Итак, устанавливаем соединение с базой данных на локальном сервере для пользователя nina с паролем "123":

```
<?
$conn = mysql_connect(
   "localhost", "nina","123")
or die("Невозможно установить
      соединение: ". mysql_error());
echo "Соединение установлено";
mysql_close($conn);
?>
```

Действие *mysql_connect* равносильно команде

```
shell>mysql -u nina -p123
```

Выбор базы данных

После *установки соединения* нужно выбрать базу данных, с которой будем работать. Наши данные хранятся в базе данных book. В *MySQL выбор базы данных* осуществляется с помощью команды use:

```
mysql>use book;
```

В PHP для этого существует функция *mysql_select_db*.

Синтаксис *mysql_select_db*:

```
логическое mysql_select_db (
   строка database_name
   [, ресурс link_identifier])
```

Эта функция возвращает TRUE в случае успешного *выбора базы данных* и FALSE - в противном случае.

Сделаем базу данных book рабочей:

```
<?
$conn = mysql_connect(
   "localhost","nina","123")
or die("Невозможно установить
      соединение: ". mysql_error());
echo "Соединение установлено";
mysql_select_db("book");
?>
```

Получение списка полей таблицы

Теперь можно заняться собственно решением задачи. Как получить список полей таблицы? Очень просто. В PHP и на этот случай есть своя команда - mysql_list_fields.

Синтаксис mysql_list_fields

```
ресурс mysql_list_fields (
   строка database_name,
   строка table_name
```

[, ресурс link_identifier])

Эта функция возвращает список полей в таблице table_name в базе данных database_name. Получается, что выбирать базу данных нам было необязательно, но это пригодится позже. Как можно заметить, результат работы этой функции - переменная типа ресурс. То есть это не совсем то, что мы хотели получить. Это ссылка, которую можно использовать для получения информации о полях таблицы, включая их названия, типы и флаги.

Функция *mysql_field_name* возвращает имя поля, полученного в результате выполнения запроса. Функция *mysql_field_len* возвращает длину поля. Функция *mysql_field_type* возвращает тип поля, а функция *mysql_field_flags* возвращает список флагов поля, записанных через пробел. Типы поля могут быть int, real, string, blob и т.д. Флаги могут быть not_null, primary_key, unique_key, blob, auto_increment и т.д.

Синтаксис у всех этих команд одинаков:

```
строка mysql_field_name (
   ресурс result, целое field_offset)
строка mysql_field_type (
   ресурс result, целое field_offset)
строка mysql_field_flags (
   ресурс result, целое field_offset)
строка mysql_field_len (
   ресурс result, целое field_offset)
```

Здесь result - это идентификатор результата запроса (например, запроса, отправленного функциями mysql_list_fields или *mysql_query* (о ней будет рассказано позднее)), а field_offset - порядковый номер поля в результате.

Вообще говоря, то, что возвращают функции типа mysql_list_fields или *mysql_query*, представляет собой таблицу, а точнее, указатель на нее. Чтобы получить из этой таблицы конкретные значения, нужно задействовать специальные функции, которые построчно читают эту таблицу. К таким функциям и относятся *mysql_field_name* и т.п. Чтобы перебрать все строки в таблице результата выполнения запроса, нужно

знать число строк в этой таблице. Команда *mysql_num_rows*(ресурс result) возвращает число строк во множестве результатов result.

А теперь попробуем получить список полей таблицы Artifacts (коллекция экспонатов).

```
<?
$conn = mysql_connect(
   "localhost","nina","123")
or die("Невозможно установить
       соединение: ". mysql_error());
echo "Соединение установлено";
mysql_select_db("book");
$list_f = mysql_list_fields (
  "book","Artifacts",$conn);
  $n = mysql_num_fields($list_f);
for($i=0;$i<$n; $i++){
  $type = mysql_field_type($list_f, $i);
  $name_f = mysql_field_name($list_f,$i);
  $len = mysql_field_len($list_f, $i);
  $flags_str =  mysql_field_flags (
             $list_f, $i);
echo "<br>Имя поля: ". $name_f;
echo "<br>Тип поля: ". $type;
echo "<br>Длина поля: ". $len;
echo "<br>Строка флагов поля: ".
   $flags_str . "<hr>";
}
?>
```

В результате должно получиться примерно вот что (если в таблице всего два поля, конечно):

Имя поля: id

Тип поля: int

Длина поля: 11

Строка флагов поля:

not_null primary_key auto_increment

Имя поля: title

Тип поля: string

Длина поля: 255

Строка флагов поля:

Отображение списка полей в html-форму

Теперь немножко подкорректируем предыдущий пример. Будем не просто выводить информацию о поле, а отображать его в подходящий элемент *html-формы*. Так, элементы типа BLOB переведем в textarea (заметим, что поле description, которое мы создавали с типом TEXT, отображается как имеющее тип BLOB), числа и строки отобразим в текстовые строки ввода <input type=text>, а элемент, имеющий метку автоинкремента, вообще не будем отображать, поскольку его значение устанавливается автоматически.

Все это решается довольно просто, за исключением выделения из списка флагов флага auto_increment. Для этого нужно воспользоваться функцией *explode*.

Синтаксис *explode*:

массив explode(строка separator,

строка string [, int limit])

Эта функция разбивает строку string на части с помощью разделителя separator и возвращает массив полученных строк.

В нашем случае в качестве разделителя нужно взять пробел " ", а в качестве исходной строки для разбиения - строку флагов поля.

Итак, создадим форму для ввода данных в таблицу Artifacts:

Листинг 11.0.1. Форма для ввода данных в таблицу Artifacts (html, txt)

Запись данных в базу данных

Итак, форма создана. Теперь нужно сделать самое главное - отправить данные из этой формы в нашу базу данных. Как вы уже знаете, для того чтобы записать данные в таблицу, используется команда INSERT языка SQL. Например:

```
mysql> INSERT INTO Artifacts
        SET title='Петров';
```

Возникает вопрос, как можно воспользоваться такой командой (или любой другой командой SQL) в PHP скрипте. Для этого существует функция *mysql_query()*.

Синтаксис *mysql_query*

```
ресурс mysql_query ( строка query
   [, ресурс link_identifier])
```

mysql_query() посылает SQL-запрос активной базе данных *MySQL* сервера, который определяется с помощью указателя link_identifier (это ссылка на какое-то соединение с сервером *MySQL*). Если параметр link_identifier опущен, используется последнее открытое соединение. Если открытые соединения отсутствуют, функция пытается соединиться с СУБД, аналогично функции *mysql_connect()* без параметров. Результат запроса буферизируется.

Замечание: строка запроса НЕ должна заканчиваться точкой с запятой.

Только для запросов SELECT, SHOW, EXPLAIN, DESCRIBE, *mysql_query()* возвращает указатель на результат запроса, или FALSE, если запрос не был выполнен. В остальных случаях *mysql_query()* возвращает TRUE, если запрос выполнен успешно, и FALSE - в случае ошибки. Значение, не равное FALSE, говорит о том, что запрос был выполнен успешно. Оно не говорит о количестве затронутых или возвращенных рядов. Вполне возможна ситуация, когда успешный запрос не затронет ни одного ряда. *mysql_query()* также считается ошибочным и вернет FALSE, если у пользователя недостаточно прав для работы с указанной в запросе таблицей.

Итак, теперь мы знаем, как отправить запрос на вставку строк в базу данных. Заметим, что в предыдущем примере элементы формы мы назвали именами полей таблицы. Поэтому они будут доступны в скрипте insert.php, обрабатывающем данные формы, как переменные вида

$_POST['имя_поля'].

Листинг 11.0.2. insert.php (html, txt)

Итак, задачу добавления данных с помощью web-интерфейса мы решили. Однако тут есть одна тонкость. При решении мы не учитывали тот факт, что значения некоторых полей (author, photo) должны браться из других таблиц (Persons, Images). Поскольку *MySQL* с внешними ключами не работает, этот момент остается на совести разработчиков системы, т.е. на нашей совести. Нужно дописать программу таким образом, чтобы была возможность вводить в такие поля правильные значения. Но мы делать этого не будем, поскольку задача лекции состоит в том, чтобы познакомить читателя с элементами технологии, а не в том, чтобы создать работающую систему. Кроме того, имеющихся у читателя знаний вполне достаточно, чтобы решить эту проблему самостоятельно. Мы же обратимся к другой задаче - отображение данных, хранящихся в базе данных СУБД *MySQL*.

 Отображение данных, хранящихся в MySQL. Чтобы отобразить какие-то данные в браузер с помощью PHP, нужно сначала получить эти данные в виде переменных PHP. При работе с *MySQL* без посредника (такого, как PHP) выборка данных производится с помощью команды SELECT языка SQL: mysql> SELECT * FROM Artifacts;

В предыдущей главе мы говорили, что любой запрос, в том числе и на выборку, можно отправить на сервер с помощью функции *mysql_query()*; Там у нас стояла немного другая задача - получить данные из формы и отправить их с помощью запроса на вставку в базу данных. Результатом работы *mysql_query()* там могло быть только одно из выражений, TRUE или FALSE. Теперь же требуется отправить запрос на выбор всех полей, а результат отобразить в браузере. И здесь результат - это целая таблица значений, а точнее, указатель на эту таблицу. Так что нужны какие-то аналоги функции *mysql_field_name()*, только чтобы они извлекали из результата запроса не имя, а значение поля. Таких функций в PHP несколько. Наиболее популярные - *mysql_result()* и *mysql_fetch_array()*.

Синтаксис *mysql_result* смешанное mysql_result (ресурс result,

целое row [, смешанное field])

mysql_result() возвращает значение одной ячейки результата запроса. Аргумент field может быть порядковым номером поля в результате, именем поля или именем поля с именем таблицы через точку tablename.fieldname. Если для имени поля в запросе применялся алиас ('select foo as bar from...'), используйте его вместо реального имени поля.

Работая с большими результатами запросов, следует задействовать одну из функций, обрабатывающих сразу целый ряд результата (например, *mysql_fetch_row()*, *mysql_fetch_array()* и т.д.). Так как эти функции возвращают значение нескольких ячеек сразу, они НАМНОГО быстрее *mysql_result()*. Кроме того, нужно учесть, что указание численного смещения (номера поля) работает намного быстрее, чем указание колонки или колонки и таблицы через точку.

Вызовы функции *mysql_result()* не должны смешиваться с другими функциями, работающими с результатом запроса.

Синтаксис *mysql_fetch_array*

массив mysql_fetch_array (ресурс result [, целое result_type])

Эта функция обрабатывает ряд результата запроса, возвращая массив (ассоциативный, численный или оба) с обработанным рядом результата запроса, или FALSE, если рядов больше нет.

mysql_fetch_array() - это расширенная версия функции *mysql_fetch_row()*. Помимо хранения значений в массиве с численными индексами, функция возвращает значения в массиве с индексами по названию колонок.

Если несколько колонок в результате будут иметь одинаковые названия, будет возвращена последняя колонка. Чтобы получить доступ к первым, следует использовать численные индексы массива или алиасы в запросе. В случае алиасов именно их вы не сможете использовать настоящие имена колонок, как, например, не сможете использовать "photo" в описанном ниже примере.

```
select Artifacts.photo as art_image,
   Persons.photo as pers_image
  from Artifacts, Persons
```

Основная литература

1. Дейт К. Руководство по реляционной СУБД DB2. - М.: Финансы и статистика, 1988. - 320 с.
2. Кириллов В.В. Основы проектирования реляционных баз данных. Учебное пособие. - СПб.: ИТМО, 1994. - 90 с.
3. Мейер М. Теория реляционных баз данных. -М.: Мир, 1987. - 608 с.
4. Ульман Дж. Базы данных на Паскале. -М.: Машиностроение, 1990. - 386 с.
5. А.И. Марченко. Программирование в среде Turbo Pascal.-Киев «ВЕК+»Москва «ДЕСС», 1999.-488 с.
6. Зуев.Программирование на языке Turbo Pascal-М:Радио и связь, Веста,1993
7. Климов Ю.С.,Касаткин А.И.,Мороз С.М.Программирование в среде Turbo Pascal-Минск:Выш.шк.,1992.
8. Н.Вирт Алгоритмы и структура данных-М.:Мир,1989.

Дополнительная литература

9. Абрамов С.А.,Гнезделова Капустина Е.Н.и др. Задачи по программированию. - М.: Наука, 1988.
10. Вирт Н. Алгоритмы + структуры данных = программа.-М.:Мир,1985.-405с.
11. Информатика. Базовой курс. Учебник для Вузов., Санк-Петербург, 2001. под редакцией С.В.Симоновича.
12. Нортон П. Программно-аппаратная организация IBM PC.-М.:Мир,1991.-327с.
13. Informatika va programmalsh.O'quv qo'llanma. Mualliflar: A.A.Xaldjigitov, Sh.F.Madraximov, U.E.Adamboev, O'zMU, 2005 yil, 145 bet.

Сайты Internet и Ziyonet

1. http://www.jetinfo.ru/1996/19/1/article19.1996.html
2. http://www. jetinfo.ru /1997/4/1/ article 1.4.1997.html
3. http://www. jetinfo.ru/1999/1/1/ article 1.1.1996.html
4. http://www. jetinfo.ru /2002/5/2/ article 2.5.2002.html
5. http://www.intuit.ru