**Тезисы лекций по темам учебной дисциплины**

**Лекция 1. Введение в веб-технологии: структура и принципы Веб.**

**План лекции:**

Хронология развития Интернета

Стандартизации в Интернет

Стек протоколов TCP/IP

Система доменных имен DNS

Структура и принципы WWW

Прокси-серверы

Протоколы Интернет прикладного уровня

Интернет, это самая большая в мире сеть, не имеющая единого центра управления, но работающая по единым правилам и предоставляющая своим пользователям единый набор услуг. Интернет можно рассматривать как «сеть сетей», каждая из которых управляется независимым оператором – поставщиком услуг Интернета (ISP, Internet Service Provider).

С точки зрения пользователей Интернет представляет собой набор информационных ресурсов, рассредоточенных по различным сетям, включая ISP-сети, корпоративные сети, сети и отдельные компьютеры домашних пользователей. Каждый отдельный компьютер в данной сети называется *хостом* (от английского термина host).

Сегодняшний Интернет обязан своему появлению объединенной сети ARPANET, которая начиналась как скромный эксперимент в новой тогда технологии коммутации пакетов (табл. 1). Сеть ARPANET была развернута в 1969 г. и состояла поначалу всего из четырех узлов с коммутацией пакетов, используемых для взаимодействия горстки хостов и терминалов. Первые линии связи, соединявшие узлы, работали на скорости всего 50 Кбит/с. Сеть ARPANET финансировалась управлением перспективного планирования научно-исследовательских работ ARPA (Advanced Research Projects Agency) министерства обороны США и предназначалась для изучения технологии и протоколов коммутации пакетов, которые могли бы использоваться для кооперативных распределенных вычислений.

**Таблица 1**. *Хронология развития Интернета (с 1966 по 2000 г.)*

|  |  |
| --- | --- |
| **Год** | **Событие** |
| 1966 | Эксперимент с коммутацией пакетов управления ARPA |
| 1969 | Первые работоспособные узлы сети ARPANET |
| 1972 | Изобретение распределенной электронной почты |
| 1973 | Первые компьютеры, подключенные к сети ARPANET за пределами США |
| 1975 | Сеть ARPANET передана в ведение управления связи министерства обороны США |
| 1980 | Начинаются эксперименты с TCP/IP |
| 1981 | Каждые 20 дней к сети добавляется новый хост |
| 1983 | Завершен переход на TCP/IP |
| 1986 | Создана магистраль NSFnet |
| 1990 | Сеть ARPANET прекратила существование |
| 1991 | Появление Gopher |
| 1991 | Изобретение Всемирной паутины. Выпущена система PGP. Появление Mosaic |
| 1995 | Приватизация магистрали Интернета |
| 1996 | Построена магистраль ОС-3 (155 Мбит/с) |
| 1998 | Число зарегистрированных доменных имен превысило 2 млн. |
| 2000 | Количество индексируемых веб-страниц превысило 1 млрд. |

Возможно ли централизованное управление в такой глобальной сети? Ответ на данный вопрос будет отрицательным, поскольку, во-первых, данная сеть является транснациональной и, во-вторых, в силу исторических предпосылок ее формирования.

Тем не менее, в Интернете могут проявляться опосредованные формы централизации в форме единой технической политике, согласованном наборе технических стандартов, назначении имен и адресов компьютеров и сетей, входящих в Интернет.

То есть Интернет является децентрализованной сетью, что имеет свои достоинства и недостатки.

1. Достоинства:

* Легкость наращивания Интернета путем заключения соглашения между двумя ISP.

1. Недостатки:

* Сложность модернизации технологий и услуг Интернета, поскольку требуются согласованные усилия всех поставщиков услуг.
* Невысокая надежность услуг Интернета.
* Ответственность за работоспособность отдельных сегментов этой сети возлагается на поставщиках услуг Интернета.

Существуют различные типы поставщиков услуг Интернета:

* просто поставщик услуг Интернета выполняет транспортную функцию для конечных пользователей – передачу их трафика в сети других поставщиков услуг Интернета;
* поставщик интернет-контента имеет собственные информационно-справочные ресурсы, предоставляя их содержание в виде веб-сайтов;
* поставщик услуг хостинга предоставляет свои помещения, каналы связи и серверы для размещения внешнего контента;
* поставщик услуг по доставке контента занимается только доставкой контента в многочисленные точки доступа с целью повышения скорости доступа пользователей к информации;
* поставщик услуг по поддержке приложений предоставляет клиентам доступ к крупным универсальным программным продуктам, например,SAP R3;
* поставщик биллинговых услуг обеспечивает оплату счетов по Интернету;

**О роли стандартизации в Интернет**

Как следует из всего вышеизложенного, Интернет является очень сложной сетью, и соответственно такой же сложной является задача организации взаимодействия между устройствами сети. Для решения такого рода задач используется *декомпозиция*, т.е. разбиение сложной задачи на несколько более простых задач-модулей. Одной из концепций, реализующих декомпозицию, является многоуровневый подход. Такой подход дает возможность проводить разработку, тестирование и модификацию каждого отдельного уровня независимо от других уровней. *Иерархическая декомпозиция*позволяет, перемещаясь в направлении от более низких к более высоким уровням переходить к более простому представлению решаемой задачи.

Специфика многоуровневого представления сетевого взаимодействия состоит в том, что в процессе обмена сообщениями участвую как минимум две стороны, для которых необходимо обеспечить согласованную работу двух иерархий аппаратно-программных средств. Каждый из уровней должен поддерживать *интерфейс*с выше- и нижележащими уровнями собственной иерархии средств и интерфейс со средствами взаимодействия другой стороны на том же уровне иерархии. Данный тип интерфейса называется *протоколом* (см. рисунок 2).



**Рисунок 2**. *Организация взаимодействия между уровнями иерархии при иерархической декомпозиции в сети Интернет*.

Иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети, называется *стеком протоколов*.

В начале 80-х годов международные организации по стандартизации ISO (International Organization for Standardization), ITU (International Telecommunications Union) и другие разработали стандартную модель взаимодействия открытых систем OSI (Open System Interconnection). Назначение данной модели состоит в обобщенном представлении средств сетевого взаимодействия. Ее также можно рассматривать в качестве универсального языка сетевых специалистов (*справочной модели*).

Поскольку сеть – это соединение разнородного оборудования, актуальной является проблема совместимости, что в свою очередь, требует согласования всеми производителями общепринятых стандартов. *Открытой* является система, построенная в соответствии с *открытыми спецификациями*.

*Спецификация* представляет собой формализованное описание аппаратных (программных) компонентов, способов их функционирования, взаимодействия с другими компонентами, условий эксплуатации, особых характеристик. Под *открытыми спецификациями* понимаются опубликованные, общедоступные спецификации, соответствующие стандартам и принятые в результате достижения согласия после всестороннего обсуждения всеми заинтересованными сторонами. Использование открытых спецификаций при разработке систем позволяет третьим сторонам разрабатывать для этих систем аппаратно-программные средства расширения и модификации, а также создавать программно-аппаратные комплексы из продуктов разных производителей.

Если две сети построены с соблюдением принципов открытости, это дает следующие преимущества:

* Возможность построения сети из аппаратных и программных средств различных производителей, придерживающихся стандарта;
* Безболезненная замена отдельных компонентов сети другими, более совершенными;
* Легкость сопряжения одной сети с другой.

В рамках модели OSI средства взаимодействия делятся на семь уровней: прикладной, представления, сеансовый, транспортный, сетевой, канальный и физический. В распоряжение программистов предоставляется прикладной программный интерфейс, позволяющий обращаться с запросами к самому верхнему уровню, а именно, - уровню приложений.

Сеть Интернет строилась в полном соответствии с принципами открытых систем. В разработке стандартов этой сети принимали участие тысячи специалистов-пользователей сети из вузов, научных организаций и компаний. Результат работы по стандартизации воплощается в документах *RFC*.

*RFC* (англ.RequestforComments) — документ из серии пронумерованных информационных документов Интернета, содержащих технические спецификации и Стандарты, широко применяемые во Всемирной сети. В настоящее время первичной публикацией документов *RFC* занимается *IETF* под эгидой открытой организации Общество Интернета (*ISOC*). Правами на *RFC* обладает именно Общество Интернет. Формат *RFC* появился в 1969 г. при обсуждении проекта *ARPANET*. Первые RFC распространялись в печатном виде на бумаге в виде обычных писем, но уже с декабря 1969 г., когда заработали первые сегменты ARPANET, документы начали распространяться в электронном виде. В таблице 2 приведены некоторые из наиболее известных документов RFC.

**Таблица 2.***Примеры популярных RFC-документов*.

|  |  |
| --- | --- |
| **Номер RFC** | **Тема** |
| RFC 768 | UDP |
| RFC 791 | IP |
| RFC 793 | TCP |
| RFC 822 | Формат электронной почты, заменен RFC 2822 |
| RFC 959 | FTP |
| RFC 1034 | DNS — концепция |
| RFC 1035 | DNS — внедрение |
| RFC 1591 | Структура доменных имен |
| RFC 1738 | URL |
| RFC 1939 | Протокол POP версии 3 (POP3) |
| RFC 2026 | Процесс стандартизации в Интернете |
| RFC 2045 | MIME |
| RFC 2231 | Кодировка символов |
| RFC 2616 | HTTP |
| RFC 2822 | Формат электронной почты |
| RFC 3501 | IMAP версии 4 издание 1 (IMAP4rev1) |

Основным организационным подразделением, координирующим работу по стандартизации Интернет, является *ISOC* (Internet Society), объединяющее порядка 100 тысяч участников, которые занимаются различными аспектами развития данной сети. *ISOC* курирует работу *IAB* (Internet Architecture Board), включающую две группы:

* *IRTF* (Internet Research Task Force). Координирует долгосрочные исследовательские проекты, относящиеся к TCP/IP;
* *IETF* (Internet Engineering Task Force). Инженерная группа, определяющая спецификации для последующих стандартов Интернет.

Разработкой стандартов для сети Веб, начиная с 1994 года, занимается Консорциум W3C (World Wide Web Consortium), основанный и до сих пор возглавляемый Тимом Бернерсом-Ли.

Консорциум W3C — организация, разрабатывающая и внедряющая технологические стандарты для Интернета и WWW. Миссия W3C формулируется следующим образом: «Полностью раскрыть потенциал Всемирной паутины путём создания протоколов и принципов, гарантирующих долгосрочное развитие Сети». Две другие важнейшие задачи Консорциума — обеспечить полную «интернационализацию Сети» и сделать ее доступной для людей с ограниченными возможностями.

W3C разрабатывает для WWW единые принципы и стандарты, называемые «*Рекомендациями*», которые затем внедряются разработчиками программ и оборудования. Благодаря *Рекомендациям* достигается совместимость между программными продуктами и оборудованием различных компаний, что делает сеть WWW более совершенной, универсальной и удобной в использовании.

Все *Рекомендации* W3C открыты, то есть, не защищены патентами и могут внедряться любым человеком без каких-либо финансовых отчислений Консорциуму.

Для удобства пользователей Консорциумом созданы специальные *программы-валидаторы* (англ. Online Validation Service), которые доступны по сети и могут за несколько секунд проверить документы на соответствие популярным Рекомендациям W3C. Консорциумом также созданы многие другие утилиты для облегчения работы веб-мастеров и программистов. Большинство утилит — это программы с открытым исходным кодом, все они бесплатные. В последнее время, повинуясь мировым тенденциям, Консорциум, в целом, гораздо больше внимания уделяет проектам с открытым исходным кодом.

В российском сегменте Интернета имеется своя организация - Российский НИИ Развития Общественных Сетей *РОСНИИРОС* (RussianInstituteforPublicNetworks, RIPN). *РОСНИИРОС* занимается координацией российских исследований и разработок в Интернете.

Прежде чем перейти к описанию структуры, принципов работы и основных протоколов сети Веб, рассмотрим основной стек протоколов сети Интернет - стек TCP/IP.

**Стек протоколов TCP/IP**

Эти протоколы изначально ориентированы на глобальные сети, в которых качество соединительных каналов не идеально. Он позволяет создавать глобальные сети, компьютеры в которых соединены друг с другом самыми разными способамиот высокоскоростных оптоволоконных кабелей и спутниковых каналов до коммутируемых телефонных линий. TCP/IP соответствует модели OSI достаточно условно и содержит 4 уровня. Прикладной уровень стека соответствует трем верхним уровням модели OSI: прикладному, представления и сеансовому.

В сети данные всегда передаются блоками относительно небольшого размера. Каждый блок имеет префиксную часть (заголовок), описывающую содержимое блока, и суффиксную, содержащую, например, информацию для контроля целостности передаваемого блока данных.

Название стека протоколов TCP/IP состоит из названий двух разных протоколов. Протокол IP (Internet Protocol) представляет собой протокол нижнего (сетевого) уровня и отвечает за передачу пакетов данных в сети. Он относится к так называемым протоколам *датаграмм* и работает без подтверждений. Последнее означает, что при его использовании доставка пакетов данных не гарантируется и не подтверждается. Не гарантируется также и то, что пакеты достигнут пункта назначения в той последовательности, в которой они были отправлены.

К протоколам сетевого уровня относится также протокол межсетевых управляющих сообщений *ICMP* (Internet Control Message Protocol), предназначенный для передачи маршрутизатором источнику информации об ошибках при передаче пакета.

Очевидно, что намного удобнее передавать данные по каналу, который работает корректно, доставляя все пакеты по порядку. Поэтому поверх протокола IP работает протокол передачи данных более высокого (транспортного) уровня — TCP (Transmission Control Protocol). Посылая и принимая пакеты через протокол IP, протокол TCP гарантирует доставку всех переданных пакетов данных в правильной последовательности.

Следует отметить, что при использовании протокола IP обеспечивается более быстрая передача данных, так как не тратится время на подтверждение приема каждого пакета. Есть и другие преимущества. Одно из них заключается в том, что он позволяет рассылать пакеты данных в широковещательном режиме, при котором они достигают всех компьютеров физической сети. Что же касается протокола TCP, то для передачи данных с его помощью необходимо создать канал связи между компьютерами. Он и создается с использованием протокола IP.

Для идентификации сетевых интерфейсов используются 3 типа адресов:

* аппаратные адреса (или MAC-адреса);
* сетевые адреса (IP-адреса);
* символьные (доменные) имена.

В рамках IP протокола для создания глобальной системы адресации, не зависящей от способов адресации узлов в отдельных сетях, используется пара идентификаторов, состоящая из номера сети и номера узла. При этом IP-адрес идентифицирует не отдельный компьютер или маршрутизатор, а одно сетевое соединение в составе сети, в которую он входит; то есть конечный узел может входить в несколько IP-сетей.

**Система доменных имен DNS**

Несмотря на то, что аппаратное и программное обеспечение в рамках TCP/IP сетей для идентификации узлов использует IP-адреса, пользователи предпочитают *символьные имена* (*доменные имена*).

Первоначально в локальных сетях из небольшого числа компьютеров применялись плоские имена, состоящие из последовательности символов без разделения их на отдельные части, например,*MYCOMP*. Для установления соответствия между символьными именами и числовыми адресами использовались широковещательные запросы. Однако для больших территориально распределенных сетей, работающих на основе протокола TCP/IP такойспособ,оказался неэффективным. Поэтому для установления соответствия между доменным именем и IP-адресом используется специальная система доменных имен (DNS, DomainNameSystem), которая основана насоздаваемых администраторами сети таблиц соответствия.

В сетях TCP/IP используется доменная система имен, имеющая иерархическую (в виде дерева) структуру. Данная структура имен напоминает иерархию имен, используемую во многих файловых системах. Запись доменного имени начинается с самой младшей составляющей, затем после точки следует следующая по старшинству символьная часть имени и так далее. Последовательность заканчивается корневым именем, например,*company.yandex.ru.*

Построенная таким образом система имен позволяет разделять административную ответственность по поддержке уникальности имен в пределах своего уровня иерархии между различными людьми или организациями.

Совокупность имен, у которых несколько старших составных частей совпадают, образуют *домен* имен.

Корневой домен управляется центральными органами Интернета: *IANA*и *Internic*.

Домены верхнего уровня назначаются для каждой страны, а также для различных типов организаций. Имена этих доменов должны следовать международному стандарту *ISO 3166*. Для обозначения стран используются двухбуквенные аббревиатуры, например,*ru* (Российская Федераця), *kz* (Казахстан), *us* (США), *it* (Италия), *fr* (Франция).

Для различных типов организаций используются трехбуквенные аббревиатуры:

* net – сетевые организации;
* org – некоммерческие организации;
* com - коммерческие организации;
* edu – образовательные организации;
* gov – правительственные организации.

Администрирование каждого домена возлагается на отдельную организацию, которая делегирует администрирование поддоменов другим организациям.

Для получения доменного имени необходимо зарегистрироваться в соответствующей организации, которой организция *InterNIC* делегировала свои полномочия по распределению доменных имен.

В TCP/IP сетях соответствие между доменными именами и IP-адресами может устанавливаться как локальными средствами, так и централизованными службами. Первоначально соответствие задавалось с помощью создаваемого вручную на хосте файла hosts.txt, состоящего из строк, содержащих пару вида «доменное имя – IP-адрес». Однако с активным ростом Интернета такое решение оказалось немасштабируемым.

Альтернативное решение – централизованная служба DNS, использующая распределенную базу отображений «доменное имя – IP-адрес». Сервер домена хранит только имена, которые заканчиваются на следующем ниже по дереву уровне. Это позволяет распределять более равномерно нагрузку по разрешению имен между всеми DNS-серверами. Каждый DNS-сервер помимо таблицы отображения имен содержит ссылки на DNS-серверы своих поддоменов.

Существуют две схемы разрешения DNS-имен.

*Нерекурсивная процедура:*

1. DNS-клиент обращается к корневому DNS-серверу с указанием полного доменного имени;
2. DNS-сервер отвечает клиенту, указывая адрес следующего DNS-сервера, обслуживающего домен верхнего уровня, заданный в следующей старшей части имени;
3. DNS-клиент делает запрос следующего DNS-сервера, который отсылает его к DNS-серверу нужного поддомена и т.д., пока не будет найден DNS-сервер, в котором хранится соответствие запрошенного имени IP-адресу. Сервер дает окончательный ответ клиенту.

*Реккурсивная процедура:*

1. DNS-клиент запрашивает локальный DNS-сервер, обслуживающий поддомен, которому принадлежит клиент;
2. Далее
3. Если локальный DNS-сервер знает ответ, он возвращает его клиенту
4. Если локальный сервер не знает ответ, то он выполняет итеративные запросы к корневому серверу. После получения ответа сервер передает его клиенту.

Таким образом, при рекурсивной процедуре клиент фактически перепоручает работу своему серверу. Для ускорения поиска IP-адресов DNS-серверы широко применяют кэширование (на время от часов до нескольких дней) проходящих через них ответов.

**Структура и принципы WWW**

Сеть WWW образуют миллионы *веб-серверов*, расположенных по всему миру. *Веб*-*сервер* является программой, запускаемой на подключённом к сети компьютере и передающей данные по протоколу HTTP.

Для идентификации ресурсов (зачастую файлов или их частей) в WWW используются идентификаторы ресурсов *URI* (Uniform Resource Identifier). Для определения местонахождения ресурсов в этой сети используются локаторы ресурсов *URL* (Uniform Resource Locator). Такие URL-локаторы представляют собой комбинацию URI и системы DNS.

Доменное имя (или IP-адрес) входит в состав URL для обозначения компьютера (его сетевого интерфейса), на котором работает программа веб-сервер.

На клиентском компьютере для просмотра информации, полученной от веб-сервера, применяется специальная программа — *веб-браузер*. Основная функция веб-браузера - отображение гипертекстовых страниц (веб-страниц). Для создания гипертекстовых страниц в WWW изначально использовался язык HTML. Множество веб-страниц образуют *веб-сайт*.

**Прокси-серверы**

*Прокси-сервер* (proxy-server) — служба в компьютерных сетях, позволяющая клиентам выполнять косвенные запросы к другим сетевым службам.

Сначала клиент подключается к прокси-серверу и запрашивает какой-либо ресурс, расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного *кеша* (если имеется). В некоторых случаях запрос клиента или ответ сервера может быть изменен прокси-сервером в определённых целях. Также прокси-сервер позволяет защищать клиентский компьютер от некоторых сетевых атак.

Чаще всего прокси-серверы применяются для следующих целей:

* обеспечение доступа с компьютеров локальной сети в Интернет;
* кеширование данных: если часто происходят обращения к одним и тем же внешним ресурсам, то можно держать их копию на прокси-сервере и выдавать по запросу, снижая тем самым нагрузку на канал во внешнюю сеть и ускоряя получение клиентом запрошенной информации.
* сжатие данных: прокси-сервер загружает информацию из Интернета и передаёт информацию конечному пользователю в сжатом виде.
* защита локальной сети от внешнего доступа: например, можно настроить прокси-сервер так, что локальные компьютеры будут обращаться к внешним ресурсам только через него, а внешние компьютеры не смогут обращаться к локальным вообще (они «видят» только прокси-сервер).
* ограничение доступа из локальной сети к внешней: например, можно запретить доступ к определённым веб-сайтам, ограничить использование интернета каким-то локальным пользователям, устанавливать квоты на трафик или полосу пропускания, фильтровать рекламу и вирусы.
* анонимизация доступа к различным ресурсам. Прокси-сервер может скрывать сведения об источнике запроса или пользователе. В таком случае целевой сервер видит лишь информацию о прокси-сервере, например, IP-адрес, но не имеет возможности определить истинный источник запроса. Существуют также искажающие прокси-серверы, которые передают целевому серверу ложную информацию об истинном пользователе.

**Протоколы Интернет прикладного уровня**

Самый верхний уровень в иерархии протоколов Интернет занимают следующие протоколы прикладного уровня:

* DNS - распределённая система доменных имён, которая по запросу, содержащему доменное имя хоста сообщает IP адрес;
* HTTP - протокол передачи гипертекста в Интернет;
* HTTPS - расширение протокола HTTP, поддерживающее шифрование;
* FTP (File Transfer Protocol - RFC 959) - протокол, предназначенный для передачи файлов в компьютерных сетях;
* Telnet (TELecommunication NETwork - RFC 854) - сетевой протокол для реализации текстового интерфейса по сети;
* SSH (Secure Shell - RFC 4251) - протокол прикладного, позволяющий производить удалённое управление операционной системой и передачу файлов. В отличие от Telnet шифрует весь трафик;
* POP3 – протокол почтового клиента, который используется почтовым клиентом для получения сообщений электронной почты с сервера;
* IMAP - протокол доступа к электронной почте в Интернет;
* SMTP – протокол, который используется для отправки почты от пользователей к серверам и между серверами для дальнейшей пересылки к получателю;
* LDAP - протокол для доступа к службе каталогов X.500, является широко используемым стандартом доступа к службам каталогов;
* XMPP (Jabber) - основанный на XML расширяемый протокол для мгновенного обмена сообщениями в почти реальном времени;
* SNMP - базовый протокол управления сети Internet.

Рассмотрим более подробно некоторые из этих протоколов.

1. **FTP**

FTP позволяет подключаться к серверам FTP, просматривать содержимое каталогов и загружать файлы с сервера или на сервер; кроме того, возможен режим передачи файлов междусерверами; FTP позволяет обмениваться файлами и выполнять операции над ними через TCP-сети. Данный протокол работает независимо от операционных систем. Исторически протокол FTP предложил открытую функциональность, обеспечивая прозрачный перенос файлов с одного компьютера на другой по сети. Это не так тривиально, как может показаться, так как у разнотипных компьютеров могут различаться размеры слов, биты в словах могут храниться в неодинаковом порядке или использоваться разные форматы слов.

1. **Telnet**

Название «telnet» имеют также некоторые утилиты, реализующие клиентскую часть протокола. Протокол *telnet* работает в соответствии с принципами архитектуры «клиент-сервер» и обеспечивает эмуляцию алфавитно-цифрового терминала, ограничивая пользователя режимом командной строки. Приложение *telnet* предоставило язык для общения терминалов с удаленными компьютерами. Когда появилась сеть ARPANET, для каждой компьютерной системы требовались собственные терминалы. Приложение *telnet* стало общим знаменателем для терминалов. Достаточно было написать для каждого компьютера программное обеспечение, поддерживающее «терминал *telnet*», чтобы один терминал мог взаимодействовать с компьютерами всех типов.

1. **SSH**

Сходен по функциональности с протоколами telnet и rlogin, но, в отличие от них, шифрует весь трафик, включая и передаваемые пароли. SSH-клиенты и SSH-серверы имеются для большинства операционных систем.

1. **Почтовые протоколы.**

Хотя *telnet* и FTP были (и остаются) полезными, первым приложением, совершившим переворот в сознании пользователей компьютеров сети ARPANET, стала электронная почта. До сети ARPANET существовали системы электронной почты, но все они были однокомпьютерными системами. В 1972 г. *Рэй Томлинсон* (Ray Tomlinson) из компании BBN написал первый пакет, предоставляющий распределенные почтовые услуги в компьютерной сети из нескольких компьютеров. Уже к 1973 г. исследования управления ARPA показали, что три четверти всего трафика сети ARPANET составляла электронная почта. Польза электронной почты оказалась столь велика, что все больше пользователей стремилось подключиться к сети ARPANET, в результате чего возрастала потребность в добавлении новых узлов и использовании высокоскоростных линий. Таким образом, появилась тенденция, сохраняющаяся и по сей день.

* POP3 (Post Office Protocol Version 3 - RFC 1939) — протокол, который используется почтовым клиентом для получения сообщений электронной почты с почтового сервера;
* IMAP (Internet Message Access Protocol - RFC 3501) — протокол доступа к электронной почте. Аналогичен POP3, однако предоставляет пользователю богатые возможности для работы с почтовыми ящиками, находящимися на центральном сервере. Электронными письмами можно манипулировать с компьютера пользователя (клиента) без необходимости постоянной пересылки с сервера и обратно файлов с полным содержанием писем.
* SMTP (Simple Mail Transfer Protocol — RFC 2821) — протокол, предназначенный для передачи электронной почты. Используется для отправки почты от пользователей к серверам и между серверами для дальнейшей пересылки к получателю. Для приёма почты почтовый клиент должен использовать протоколы POP3 или IMAP.

**Лекция 2. Основы World Wide Web (WWW)**

**План лекции:**

Общие принципы организации и функционирования сервиса WWW.

Сервис www

Адресация веб-ресурсов. URL, URN, URI

Язык гипертекстовой разметки HTML

Протокол HTTP

Общий шлюзовый интерфейс CGI

Программное обеспечение сервиса www

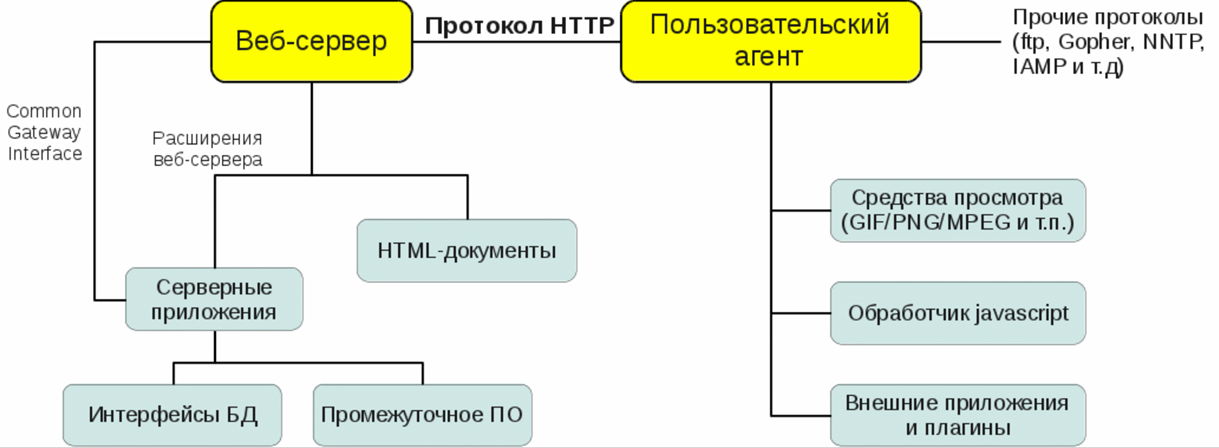
Браузеры

**Сервис www**

**Интернет** — это глобальная компьютерная сеть, объединяющая сотни миллионов компьютеров в общее информационное пространство. Интернет представляет свою инфраструктуру для прикладных сервисов различного назначения, самым популярным из которых является Всемирная Паутина – World Wide Web (www).

**World Wide Web (www, web, рус.: веб, Всемирная Паутина)** — распределенная информационная система, предоставляющая доступ к гипертекстовым документам по протоколу HTTP.

**WWW** — сетевая технология прикладного уровня [стека TCP/IP](http://www.4stud.info/networking/lecture2.html), построенная на [клиент-серверной архитектуре](http://www.4stud.info/networking/lecture5.html) и использующая инфраструктуру Интернет для взаимодействия между сервером и клиентом ([рис. 1](http://www.4stud.info/web-programming/lecture1.html#www)).



*Рис. 1. Архитектура сервиса WWW*

**Серверы www (*веб-серверы*)** — это хранилища гипертекстовой (в общем случае) информации, управляемые специальным программным обеспечением.

Документы, представленные в виде гипертекста называются *веб-страницами*. Несколько веб-страниц, объединенных общей тематикой, оформлением, связанных гипертекстовыми ссылками и обычно находящихся на одном и том же веб-сервере, называются *веб-сайтом*.

Для загрузки и просмотра информации с веб-сайтов используются специальные программы — *браузеры*, способные обрабатывать гипертектовую разметку и отображать содержимое веб-страниц.

В основе WWW — взаимодействие между веб-сервером и браузерами по протоколу HTTP (HyperText Transfer Protocol). **Веб-сервер** — это программа, запущенная на сетевом компьютере и ожидающая клиентские запросы по протоколу HTTP. Браузер может обратиться к веб-серверу по доменному имени или по ip-адресу, передавая в запросе идентификатор требуемого ресурса. Получив запрос от клиента, сервер находит соответствующий ресурс на локальном устройстве хранения и отправляет его как ответ. Браузер принимает ответ и обрабатывает его соответствующим образом, в зависимости от типа ресурса (отображает гипертекст, показывает изображения, сохраняет полученные файлы и т.п.).

Основной тип ресурсов Всемирной паутины — гипертекстовые страницы. **Гипертекст** — это обычный текст, размеченный специальными управляющими конструкциями — тегами. Браузер считывает теги и интерпретирует их как команды форматирования при выводе информации. Теги описывают структуру документа, а специальные теги, якоря и гиперссылки, позволяют установить связи между веб-страницами и перемещаться как внутри веб-сайта, так и между сайтами.

**Т. Дж. Бернерс-Ли — «отец» Всемирной паутины**

Сэр [Тимоти Джон Бернерс-Ли](http://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%80%D0%BD%D0%B5%D1%80%D1%81-%D0%9B%D0%B8,_%D0%A2%D0%B8%D0%BC) — британский учёный-физик, изобретатель Всемирной паутины (совместно с Робертом Кайо), автор URI, HTTP и HTML. Действующий глава Консорциума Всемирной паутины (W3C). Автор концепции семантической паутины и множества других разработок в области информационных технологий. 16 июля 2004 года Королева Великобритании Елизавета II произвела Тима Бернерса-Ли в Рыцари-Командоры за «службу во благо глобального развития Интернета».

**Компоненты сервиса**

Функционирование сервиса обеспечивается четырьмя составляющими:

* URL/URI — унифицированный способ адресации и идентификации сетевых ресурсов;
* HTML — язык гипертекстовой разметки веб-документов;
* HTTP — протокол передачи гипертекста;
* CGI — общий шлюзовый интерфейс, представляющий доступ к серверным приложениям.

**Адресация веб-ресурсов. URL, URN, URI**

Для доступа к любым сетевым ресурсам необходимо знать где они размещены и как к ним можно обратиться. Во Всемирной паутине для обращения к веб-документам изначально используется стандартизованная схема адресации и идентификации, учитывающую опыт адресации и идентификации таких сетевых сервисов, как e-mail, telnet, ftp и т.п. — URL, Uniform Resource Locator.

URL ([RFC 1738](http://tools.ietf.org/html/rfc1738)) — унифицированный локатор (указатель) ресурсов, стандартизированный способ записи адреса ресурса в www и сети Интернет. Адрес URL имеет гибкую и расширяемую структуру для максимально естественного указания местонахождения ресурсов в сети. Для записи адреса используется ограниченный набор символов ASCII. Общий вид адреса можно представить так:

<схема>://<логин>:<пароль>@<хост>:<порт>/<полный-путь-к-ресурсу>

Где:

Схема - схема обращения к ресурсу: http, ftp, gopher, mailto, news, telnet, file, man, info, whatis, ldap, wais и т.п.

логин:пароль - имя пользователя и его пароль, используемые для доступа к ресурсу

хост - [доменное имя](http://www.4stud.info/networking/dns.html) хоста или его IP-адрес.

Порт - порт хоста для подключения

полный-путь-к-ресурсу - уточняющая информация о месте нахождения ресурса (зависит от протокола).

Примеры URL:

http://example.com #запрос стартовой страницы по умолчанию

http://www.example.com/site/map.html #запрос страницы в указанном каталоге

http://example.com:81/script.php #подключение на нестандартный порт

http://example.org/script.php?key=value #передача параметров скрипту

ftp://user:pass@ftp.example.org #авторизация на ftp-сервере

http://192.168.0.1/example/www #подключение по ip-адресу

file:///srv/www/htdocs/index.html #открытие локального файла

gopher://example.com/1 #подключение к серверу gopher

mailto://user@example.org #ссылка на адрес эл.почты

В августе 2002 года  стандарт RFC 3305 анонсировал устаревание URL в пользу URI (Uniform Resource Identifier), еще более гибкого способа адресации, вобравшего возможности как URL, так и URN (Uniform Resource Name, унифицированное имя ресурса). URI позволяет не только указавать местонахождение ресурса (как URL), но и идентифицировать его в заданном пространстве имен (как URN). Если в URI не указывать местонахождение, то с его помощью можно описывать ресурсы, которые не могут быть получены непосредственно из Интернета (автомобили, персоны и т.п.). Текущая структура и синтаксис URI регулируется стандартом RFC 3986, вышедшим в январе 2005 года.

**Язык гипертекстовой разметки HTML**

HTML ([HyperText Markup Language](http://www.4stud.info/web-programming/html.html)) — стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц созданы при помощи языка HTML. Язык HTML интерпретируется браузером и отображается в виде документа, в удобной для человека форме. HTML является приложением SGML (стандартного обобщённого языка разметки) и соответствует международному стандарту ISO 8879.

HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. Для этого он представляет небольшой (сравнительно) набор структурных и семантических элементов — тегов. С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). В идеале, текст с разметкой HTML должен единообразно воспроизводиться на различном оборудовании (монитор ПК, экран органайзера, ограниченный по размерам экран мобильного телефона, медиа-проектор). Однако современное применение HTML очень далеко от его изначальной задачи. Со временем основная идея платформонезависимости языка HTML стала жертвой коммерциализации www и потребностей в мультимедийном и графическом оформлении.

**Протокол HTTP**

HTTP ([HyperText Transfer Protocol](http://www.4stud.info/web-programming/protocol-http.html)) — протокол передачи гипертекста, текущая версия HTTP/1.1 ([RFC 2616](http://tools.ietf.org/html/rfc2616)). Этот протокол изначально был предназначен для обмена гипертекстовыми документами, сейчас его возможности существенно расширены в сторону передачи двоичной информации.

HTTP — типичный клиент-серверный протокол, обмен сообщениями идёт по схеме «запрос-ответ» в виде ASCII-команд. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является символьно-ориентированным.

HTTP — протокол прикладного уровня, но используется также в качестве «транспорта» для других прикладных протоколов, в первую очередь, основанных на [языке XML](http://www.4stud.info/web-programming/xml.html) (SOAP, XML-RPC, SiteMap, RSS и проч.).

**Общий шлюзовый интерфейс CGI**

CGI ([Common Gateway Interface](http://www.4stud.info/web-programming/cgi.html)) — механизм доступа к программам на стороне веб-сервера. Спецификация CGI была разработана для расширения возможностей сервиса www за счет подключения различного внешнего программного обеспечения. При использовании CGI веб-сервер представляет браузеру доступ к исполнимым программам, запускаемым на его (серверной) стороне через стандартные потоки ввода и вывода.

Интерфейс CGI применяется для создания динамических веб-сайтов, например, когда веб-страницы формируются из результатов запроса к базе данных. Сейчас популярность CGI снизилась, т.к. появились более совершенные альтернативные решения (например, модульные расширения веб-серверов).

**Программное обеспечение сервиса www**

**Веб-серверы**

**Веб-сервер** — это сетевое приложение, обслуживающее HTTP-запросы от клиентов, обычно веб-браузеров. Веб-сервер принимает запросы и возвращает ответы, обычно вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Веб-серверы — основа Всемирной паутины. С расширением спектра сетевых сервисов веб-серверы все чаще используются в качестве шлюзов для [серверов приложений](http://www.4stud.info/networking/application-server.html) или сами представляют такие функции (например, [Apache Tomcat](http://www.4stud.info/networking/work10.html)).

Созданием программного обеспечения веб-серверов занимаются многие разработчики, но наибольшую популярность (по статистике [http://netcraft.com](http://news.netcraft.com/archives/category/web-server-survey/)) имеют такие программные продукты, как Apache ([Apache Software Foundation](http://httpd.apache.org/)), IIS ([Microsoft](http://www.iis.net/)), Google Web Server (GWS, Google Inc.) и [nginx](http://nginx.org/).

**Apache** — [свободное программное обеспечение](http://www.4stud.info/oss/lecture1.html), распространяется под совместимой с GPL лицензией. Apache уже многие годы является лидером по распространенности во Всемирной паутине в силу своей надежности, гибкости, масштабируемости и безопасности.

**IIS** (Internet Information Services) — проприетарный набор серверов для нескольких служб Интернета, разработанный Майкрософт и распространяемый с серверными операционными системами семейства Windows. Основным компонентом IIS является веб-сервер, также поддерживаются протоколы FTP, POP3, SMTP, NNTP.

**Google Web Server (GWS)** — разработка компании Google на основе веб-сервера Apache. GWS оптимизирован для выполнения приложений сервиса Google Applications.

**nginx** [e**ngin**e **x**] — это HTTP-сервер, совмещенный с кэширующим прокси-сервером. Разработан [И. Сысоевым](http://sysoev.ru/) для компании Рамблер. Осенью 2004 года вышел первый публично доступный релиз, сейчас nginx используется на 9-12% веб-серверов.

**Браузеры**

**Браузер, веб-обозреватель** (web-browser) — клиентское приложение для доступа к веб-серверам по протоколу HTTP и просмотра веб-страниц. Как правило браузеры дополнительно поддерживают и ряд других протоколов (например ftp, file, mms, pop3).

Первые HTTP-клиенты были консольными и работали в текстовом режиме, позволяя читать гипертекст и перемещаться по ссылкам. Сейчас консольные браузеры (такие, как lynx, w3m или links) практически не используются рядовыми посетителями веб-сайтов. Тем не менее такие браузеры весьма полезны для веб-разработчиков, так как позволяют «увидеть» веб-страницу «глазами» поискового робота.

Исторически первым браузером в современном понимании (т.е. с графическим интерфейсом и т.д.) была программа NCSA Mosaic, разработанная Марком Андерисеном и Эриком Бина. Mosaic имел довольно ограниченные возможности, но его открытый исходный код стал основой для многих последующих разработок.

Существует *множество различных программ-браузеров*, но наибольшей популярностью на дату написания этой статьи пользуются следующие:

* Internet Explorer / (Edge с версии 11) (IE) — браузер, разработанный компанией Майкрософт и тесно интегрированный c ОС Windows.
* Firefox — свободный кроссплатформенный браузер, разрабатываемый Mozilla Foundation и распространяемый под тройной лицензией GPL/LGPL/MPL.
* Safari — проприетарный браузер, разработаный корпорацией Apple и входящий в состав операционной системы Mac OS X.
* Opera — кроссплатформенный многофункциональный веб-браузер, впервые представленный в 1994 году группой исследователей из норвежской компании Telenor. Дальнейшая разработка ведется Opera Software ASA.

**Роботы-«пауки»**

Наряду с браузерами, ориентированными на пользователя, существуют и специализированные клиенты-роботы («пауки», «боты»), подключающиеся к веб-серверам и выполняющие различные задачи автоматической обработки гипертекстовой информации. Сюда относятся, в первую очередь, роботы поисковых систем, таких как google.com, yandex.ru, yahoo.com и т.п., выполняющие обход веб-сайтов для последующего построения поискового индекса.

**Лекция 3. Введение в клиент-серверные технологии Веб. Протокол HTTP.**

**План лекции:**

Протокол HTTP

Программное обеспечение для работы с протоколом HTTP

HTTP-запрос. Структура запроса клиента

Версии протокола HTTP

Единообразный идентификатор ресурса URI

Обеспечение безопасности передачи данных HTTP

Механизм Cookie

**Протокол HTTP**

*HTTP* (HyperTextTransferProtocol - RFC 1945, RFC 2616) - протокол прикладного уровня для передачи гипертекста.

Центральным объектом в HTTP является *ресурс*, на который указывает *URI* в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя изначально данный протокол предназначен для передачи символьной информации. На первый взгляд это может показаться излишней тратой ресурсов. Действительно, данные в символьном виде занимают больше памяти, сообщения создают дополнительную нагрузку на каналы связи, однако подобный формат имеет много преимуществ. Сообщения, передаваемые по сети, удобочитаемы, и, проанализировав полученные данные, системный администратор может легко найти ошибку и устранить ее. При необходимости роль одного из взаимодействующих приложений может выполнять человек, вручную вводя сообщения в требуемом формате.

В отличие от многих других протоколов, HTTP является протоколом без памяти. Это означает, что протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера. Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами. Например, клиентское веб-приложение, посылающее запросы, может отслеживать задержки ответов, а веб-сервер может хранить IP-адреса и заголовки запросов последних клиентов.

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

* *Серверы* - поставщики услуг хранения и обработки информации (обработка запросов).
* *Клиенты* - конечные потребители услуг сервера (отправка запросов).
* *Прокси-серверы* для поддержки работы транспортных служб.

Основными клиентами являются *браузеры*например: *InternetExplorer*, *Opera*, *MozillaFirefox*, *NetscapeNavigator* и другие. Наиболее популярными реализациями веб-серверов являются: *InternetInformationServices* (*IIS*), *Apache*, *lighttpd*, *nginx*. Наиболее известные реализации прокси-серверов: *Squid*, *UserGate*, *Multiproxy*, *Naviscope*.

"Классическая" схема HTTP-сеанса выглядит так.

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

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

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

* Строка состояния (иногда для ее обозначения используют также термины строка-статус, или строка запроса).
* Поля заголовка.
* Пустая строка.
* Тело запроса.

*Строку состояния* вместе с *полями заголовка* иногда называют также *заголовком запроса*.



Рисунок 1. *Структура запроса клиента.*

**Строка состояния** имеет следующий формат:

*метод\_запроса URL\_pecypca версия\_протокола\_НТТР*

Рассмотрим компоненты строки состояния, при этом особое внимание уделим методам запроса.

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

* GET. Согласно формальному определению, метод GET предназначается для получения ресурса с указанным URL. Получив запрос GET, сервер должен прочитать указанный ресурс и включить код ресурса в состав ответа клиенту. Ресурс, URL которого передается в составе запроса, не обязательно должен представлять собой HTML-страницу, файл с изображением или другие данные. URL ресурса может указывать на исполняемый код программы, который, при соблюдении определенных условий, должен быть запущен на сервере. В этом случае клиенту возвращается не код программы, а данные, сгенерированные в процессе ее выполнения. Несмотря на то что, по определению, метод GET предназначен для получения информации, он может применяться и в других целях. Метод GET вполне подходит для передачи небольших фрагментов данных на сервер.
* POST. Согласно тому же формальному определению, основное назначение метода POST - передача данных на сервер. Однако, подобно методу GET, метод POST может применяться по-разному и нередко используется для получения информации с сервера. Как и в случае с методом GET, URL, заданный в строке состояния, указывает на конкретный ресурс. Метод POST также может использоваться для запуска процесса.
* Методы HEAD и PUT являются модификациями методов GET и POST.

**Версия протокола HTTP**, как правило, задается в следующем формате:

*HTTP/версия.модификация*

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

*Имя\_поля: Значение*

Назначение поля определяется его именем, которое отделяется от значения двоеточием.

Имена некоторых наиболее часто встречающихся в запросе клиента полей заголовка и их назначение приведены в таблице 1.

Таблица 1. *Поля заголовка запроса HTTP.*

|  |  |
| --- | --- |
| **Поля заголовка HTTP-запроса** | **Значение** |
| Host | Доменное имя или IP-адрес узла, к которому обращается клиент |
| Referer | URL документа, который ссылается на ресурс, указанный в строке состояния |
| From | Адрес электронной почты пользователя, работающего с клиентом |
| Accept | MIME-типы данных, обрабатываемых клиентом. Это поле может иметь несколько значений, отделяемых одно от другого запятыми. Часто поле заголовка Accept используется для того, чтобы сообщить серверу о том, какие типы графических файлов поддерживает клиент |
| Accept-Language | Набор двухсимвольных идентификаторов, разделенных запятыми, которые обозначают языки, поддерживаемые клиентом |
| Accept-Charset | Перечень поддерживаемых наборов символов |
| Content-Type | MIME-тип данных, содержащихся в теле запроса (если запрос не состоит из одного заголовка) |
| Content-Length | Число символов, содержащихся в теле запроса (если запрос не состоит из одного заголовка) |
| Range | Присутствует в том случае, если клиент запрашивает не весь документ, а лишь его часть |
| Connection | Используется для управления TCP-соединением. Если в поле содержится Close, это означает, что после обработки запроса сервер должен закрыть соединение. Значение Keep-Alive предлагает не закрывать TCP-соединение, чтобы оно могло быть использовано для последующих запросов |
| User-Agent | Информация о клиенте |

Во многих случаях при работе в Веб тело запроса отсутствует. При запуске CGI-сценариев данные, передаваемые для них в запросе, могут размещаться в теле запроса.

Ниже представлен пример HTML-запроса, сгенерированного браузером

*GET http://oak.oakland.edu/ HTTP/1.0*

*Connection: Keep-Alive*

*User-Agent: Mozilla/4.04 [en] (Win95; I)*

*Host: oak.oakland.edu*

*Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, \*/\**

*Accept-Language: en*

*Accept-Charset: iso-8859-l,\*,utf-8*

Получив от клиента запрос, сервер должен ответить ему. Знание структуры ответа сервера необходимо разработчику веб-приложений, так как программы, которые выполняются на сервере, должны самостоятельно формировать ответ клиенту.

Подобно запросу клиента, ответ сервера также состоит из четырех перечисленных ниже компонентов.

* Строка состояния.
* Поля заголовка.
* Пустая строка.
* Тело ответа.

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

*Версия\_протокола Код\_ответа Пояснительное\_сообщение*

* **Версия\_протокола** задается в том же формате, что и в запросе клиента, и имеет тот же смысл.
* **Код\_ответа** - это трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером.
* **Пояснительное\_сообщение** дублирует код ответа в символьном виде. Это строка символов, которая не обрабатывается клиентом. Она предназначена для системного администратора или оператора, занимающегося обслуживанием системы, и является расшифровкой кода ответа.

Из трех цифр, составляющих код ответа, первая (старшая) определяет класс ответа, остальные две представляют собой номер ответа внутри класса. Так, например, если запрос был обработан успешно, клиент получает следующее сообщение:

HТТР/1.0 200 ОК

Как видно, за версией протокола HTTP 1.0 следует код 200. В этом коде символ 2 означает успешную обработку запроса клиента, а остальные две цифры (00) — номер данного сообщения.

В используемых в настоящее время реализациях протокола HTTP первая цифра не может быть больше 5 и определяет следующие классы ответов.

* 1 - специальный класс сообщений, называемых информационными. Код ответа, начинающийся с 1, означает, что сервер продолжает обработку запроса. При обмене данными между HTTP-клиентом и HTTP-сервером сообщения этого класса используются достаточно редко.
* 2 - успешная обработка запроса клиента.
* 3 - перенаправление запроса. Чтобы запрос был обслужен, необходимо пред­принять дополнительные действия.
* 4 - ошибка клиента. Как правило, код ответа, начинающийся с цифры 4, возвра­щается в том случае, если в запросе клиента встретилась синтаксическая ошибка.
* 5 - ошибка сервера. По тем или иным причинам сервер не в состоянии вы­полнить запрос.

Примеры кодов ответов, которые клиент может получить от сервера, и поясняю­щие сообщения приведены в таблице 2.

Таблица 2. *Классы кодов ответа сервера.*

|  |  |  |
| --- | --- | --- |
| **Код** | **Расшифровка** | **Интерпретация** |
| 100 | Continue | Часть запроса принята, и сервер ожидает от клиента продолжения запроса |
| 200 | OK | Запрос успешно обработан, и в ответе клиента передаются данные, указанные в запросе |
| 201 | Created | В результате обработки запроса был создан новый ресурс |
| 202 | Accepted | Запрос принят сервером, но обработка его не окончена. Данный код ответа не гарантирует, что запрос будет обработан без ошибок. |
| 206 | Partial Content | Сервер возвращает часть ресурса в ответ на запрос, содержавший поле заголовка Range |
| 301 | Multiple Choice | Запрос указывает более чем на один ресурс. В теле ответа могут содержаться указания на то, как пра¬вильно идентифицировать запрашиваемый ресурс |
| 302 | Moved Permanently | Затребованный ресурс больше не располагается на сервере |
| 302 | Moved Temporarily | Затребованный ресурс временно изменил свой адрес |
| 400 | Bad Request | В запросе клиента обнаружена синтаксическая ошибка |
| 403 | Forbidden | Имеющийся на сервере ресурс недоступен для данного пользователя |
| 404 | Not Found | Ресурс, указанный клиентом, на сервере отсутствует |
| 405 | Method Not Allowed | Сервер не поддерживает метод, указанный в запросе |
| 500 | Internal Server Error | Один из компонентов сервера работает некорректно |
| 501 | Not Implemented | Функциональных возможностей сервера недоста¬точно, чтобы выполнить запрос клиента |
| 503 | Service Unavailable | Служба временно недоступна |
| 505 | HTTP Version not Supported | Версия HTTP, указанная в запросе, не поддержи¬вается сервером |

В ответе используется такая же структура полей заголовка, как и в запросе клиента. Поля заголовка предназначены для того, чтобы уточнить ответ сервера клиенту. Описание некоторых из полей, которые можно встретить в заголовке ответа сервера, приведено в таблице 3.

Таблица 3. *Поля заголовка ответа веб-сервера.*

|  |  |
| --- | --- |
| **Имя поля** | **Описание содержимого** |
| Server | Имя и номер версии сервера |
| Age | Время в секундах, прошедшее с момента создания ресурса |
| Allow | Список методов, допустимых для данного ресурса |
| Content-Language | Языки, которые должен поддерживать клиент для того, чтобы корректно отобразить передаваемый ресурс |
| Content-Type | *MIME*-тип данных, содержащихся в теле ответа сервера |
| Content-Length | Число символов, содержащихся в теле ответа сервера |
| Last-Modified | Дата и время последнего изменения ресурса |
| Date | Дата и время, определяющие момент генерации ответа |
| Expires | Дата и время, определяющие момент, после которого информация, переданная клиенту, считается устаревшей |
| Location | В этом поле указывается реальное расположение ресурса. Оно используется для перенаправления запроса |
| Cache-Control | Директивы управления кэшированием. Например, *no-cache* означает, что данные не должны кэшироваться |

В теле ответа содержится код ресурса, передаваемого клиенту в ответ на запрос. Это не обязательно должен быть HTML-текст веб-страницы. В составе ответа могут передаваться изображение, аудиофайл, фрагмент видеоинформации, а также любой другой тип данных, поддерживаемых клиентом. О том, как следует обрабатывать полученный ресурс, клиенту сообщает содержимое поля заголовка *Content-type*.

Ниже представлен пример ответа сервера на запрос, приведенный в предыдущем разделе. В теле ответа содержится исходный текст HTML-документа.

|  |
| --- |
| HTTP/1.1 200 OK  Server: Microsoft-IIS/5.1  X-Powered-By: ASP.NET  Date: Mon, 20 OCT 2008 11:25:56 GMT  Content-Type: text/html  Accept-Ranges: bytes  Last-Modified: Sat, 18 Oct 2008 15:05:44 GMT  ETag: “b66a667f948c92:8a5”  Content-Length: 426  <html>  <body>  <form action='http://localhost/Scripts/test.pl'>  <p>Operand1: <input type='text' name='A'></p>  <p>Operand2: <input type='text' name='B'></p>  <p>Operation:<br>  <select name='op'>  <option value='+'>+</option>  <option value='-'>-</option>  <option value='\*'>\*</option>  <option value='/'>/</option>  <select></p>  <input type='submit' value='Calculate!'>  </from>  </body>  </html> |

Поля заголовка и тело сообщения могут отсутствовать, но строка состояния является обязательным элементом, так как указывает на тип запроса/ответа.

Поле с именем *Content*-*type* может встречаться как в запросе клиента, так и в ответе сервера. В качестве значения этого поля указывается *MIME*-тип содержимого запроса или ответа. *MIME*-тип также передается в поле заголовка *Accept*, присутствующего в запросе.

Спецификация *MIME* (Multipurpose Internet Mail Extension — многоцелевое почтовое расширение Internet) первоначально была разработана для того, чтобы обеспечить передачу различных форматов данных в составе электронных писем. Однако применение MIME не исчерпывается электронной почтой. Средства MIME успешно используются в WWW и, по сути, стали неотъемлемой частью этой системы.

Стандарт *MIME* разработан как расширяемая спецификация, в которой подразумевается, что число типов данных будет расти по мере развития форм представления данных. Каждый новый тип в обязательном порядке должен быть зарегистрирован в *IANA* (Internet Assigned Numbers Authority).

До появления *MIME* компьютеры, взаимодействующие по протоколу HTTP, обменивались исключительно текстовой информацией. Для передачи изображений, как и для передачи любых других двоичных файлов, приходилось пользоваться протоколом FTP.

В соответствии со спецификацией *MIME*, для описания формата данных используются *тип* и *подтип*. *Тип* определяет, к какому классу относится формат содержимого HTTP-запроса или HTTP-ответа. *Подтип* уточняет формат. Тип и подтип отделяются друг от друга косой чертой:

*тип*/*подтип*

Поскольку в подавляющем большинстве случаев в ответ на запрос клиента сервер возвращает исходный текст HTML-документа, то в поле *Content-type* ответа обычно содержится значение *text/html*. Здесь идентификатор *text* описывает тип, сообщая, что клиенту передается символьная информация, а идентификатор *html* описывает подтип, т.е. указывает на то, что последовательность символов, содержащаяся в теле ответа, представляет собой описание документа на языке HTML.

Перечень типов и подтипов *MIME* достаточно велик. В таблице 4 приведены примеры *MIME*-типов, наиболее часто встречающиеся в заголовках HTML-запросов и ответов.

**Таблица 4**. *MIME типы данных.*

|  |  |  |
| --- | --- | --- |
| **Тип/подтип** | **Расширение файла** | **Описание** |
| application/pdf | .pdf | Документ, предназначенный для обработки Acrobat Reader |
| application/msexcel | .xls | Документ в формате MicrosoftExcel |
| application/postscript | .ps, .eps | Документвформате PostScript |
| application/x-tex | .tex | Документ в формате ТеХ |
| application/msword | .doc | Документ в формате MicrosoftWord |
| application/rtf | .rtf | Документ в формате RTF, отображаемый с помощью Microsoft Word |
| image/gif | .gif | Изображениевформате GIF |
| image/ jpeg | .jpeg, .jpg, | Изображение в формате JPEG |
| image/tiff | .tiff, .tif | Изображение в формате TIFF |
| image/x-xbitmap | .xbm | Изображение в формате XBitmap |
| text/plain | .txt | ASCII-текст |
| text/html | . html , . htm | Документ в формате HTML |
| audio/midi | .midi, .mid | Аудиофайл в формате MIDI |
| audio/x-wav | .wav | Аудиофайл в формате WAV |
| message/rfc822 |  | Почтовое сообщение |
| message/news |  | Сообщение в группы новостей |
| video /mpeg | .mpeg, .mpg, .mpe | Видеофрагмент в формате MPEG |
| video/avi | .avi | Видеофрагмент в формате AVI |

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

Единообразный идентификатор ресурса URI (UniformResourceIdentifier) представляет собой короткую последовательность символов, идентифицирующую абстрактный или физический ресурс. URI не указывает на то, как получить ресурс, а только идентифицирует его. Это даёт возможность описывать с помощью RDF (ResourceDescriptionFramework) ресурсы, которые не могут быть получены через Интернет (имена, названия и т.п.). Самые известные примеры URI - это URL и URN.

* URL (UniformResourceLocator) - это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса.
* URN (UniformResourceName) - это URI, который идентифицирует ресурс в определённом пространстве имён, но, в отличие от URL, URN не указывает на местонахождение этого ресурса.

URLимеет следующую структуру:

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

где:

* схема - схема обращения к ресурсу (обычно сетевой протокол);
* логин - имя пользователя, используемое для доступа к ресурсу;
* пароль - пароль, ассоциированный с указанным именем пользователя;
* хост - полностью прописанное доменное имя хоста в системе DNS или IP-адрес хоста;
* порт - порт хоста для подключения;
* URL-путь - уточняющая информация о месте нахождения ресурса.

Общепринятые схемы (протоколы) *URL* включают протоколы: *ftp, http, https, telnet*, а также:

* gopher — протокол Gopher;
* mailto — адрес электронной почты;
* news — новости Usenet;
* nntp — новости Usenet через протокол NNTP;
* irc — протокол IRC;
* prospero — службакаталогов Prospero Directory Service;
* wais — база данных системы WAIS;
* xmpp — протокол XMPP (часть Jabber);
* file — имя локального файла;
* data — непосредственные данные (Data: URL);

**Обеспечение безопасности передачи данных HTTP**

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

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

*SSL* (SecureSocketsLayer) - криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет. При его использовании создаётся защищённое соединение между клиентом и сервером. SSL изначально разработан компанией NetscapeCommunications. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший название TLS. Этот протокол использует шифрование с открытым ключом для подтверждения подлинности передатчика и получателя. Поддерживает надёжность передачи данных за счёт использования корректирующих кодов и безопасных хэш-функций. На нижнем уровне многоуровневого транспортного протокола (например, TCP) он является протоколом записи и используется для инкапсуляции различных протоколов (POP3, IMAP, SMTP или HTTP). Для каждого инкапсулированного протокола он обеспечивает условия, при которых сервер и клиент могут подтверждать друг другу свою подлинность, выполнять алгоритмы шифрования и производить обмен криптографическими ключами, прежде чем протокол прикладной программы начнет передавать и получать данные.

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

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

* *Basic* - базовая аутентификация, при которой имя пользователя и пароль передаются в заголовках http-пакетов. Пароль при этом не шифруется и присутствует в чистом виде в кодировке base64. Для данного типа аутентификации использование SSL является обязательным.
* *Digest* - дайджест-аутентификация, при которой пароль пользователя передается в хешированном виде. По уровню конфиденциальности паролей этот тип мало чем отличается от предыдущего, так как атакующему все равно, действительно ли это настоящий пароль или только хеш от него: перехватив удостоверение, он все равно получает доступ к конечной точке. Для данного типа аутентификации использование SSL является обязательным.
* *Integrated* - интегрированная аутентификация, при которой клиент и сервер обмениваются сообщениями для выяснения подлинности друг друга с помощью протоколов NTLM или Kerberos. Этот тип аутентификации защищен от перехвата удостоверений пользователей, поэтому для него не требуется протокол SSL. Только при использовании данного типа аутентификации можно работать по схеме *http*, во всех остальных случаях необходимо использовать схему *https*.

**Cookie**

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

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

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

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

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

Для того чтобы в общих чертах представить себе, как происходит обмен данными *cookie*, рассмотрим следующий пример. Предположим, что клиент передает запросы на серверы***А***, ***В*** и ***С***. Предположим также, что сервер ***В***, в отличие от ***А*** и ***С***, передает клиенту команду записать *cookie*. Последовательность запросов клиента серверу и ответов на них будет выглядеть приблизительно следующим образом.

1. Передача запроса серверу ***А***.
2. Получение ответа от сервера ***А***.
3. Передача запроса серверу ***В***.
4. Получение ответа от сервера ***В***. В состав ответа входит поле заголовка *SetCookie*. Получив его, клиент записывает *cookie* на диск.
5. Передача запроса серверу ***С***. Несмотря на то что на диске хранится запись *cookie*, клиент не предпринимает никаких специальных действий, так как значение *cookie* было записано по инициативе другого сервера.
6. Получение ответа от сервера ***С***.
7. Передача запроса серверу ***А***. В этом случае клиент также никак не реагирует на тот факт, что на диске хранится *cookie*.
8. Получение ответа от сервера ***А***.
9. Передача запроса серверу ***В***. Перед тем как сформировать запрос, клиент определяет, что на диске хранится запись *cookie*, созданная после получения ответа от сервера ***В***. Клиент проверяет, удовлетворяет ли данный запрос некоторым требованиям, и, если проверка дает положительный результат, включает в заголовок запроса поле *Cookie*.

Таким образом, процедуру записи и получения *cookie* можно представить себе как своеобразный "запрос" сервера, инкапсулированный в его ответе клиенту. Соответственно получение *cookie* также можно представить себе как ответ клиента, инкапсулированный в составе запроса тому же серверу.

Рассмотрим подробнее, какие данные передаются в поле заголовка *Set-cookie* и как они влияют на поведение клиента.

Поле *Set-cookie* имеет следующий формат:

***Set-cookie: имя = значение; expires = дата; path = путь; домен = имя\_домена, secure***

где

* + Пара *имя = значение* – именованные данные, сохраняемые с помощью механизм *cookie*. Эти данные должны храниться на клиент-машине и передаваться серверу в составе очередного запроса клиента.
  + Дата, являющаяся значением параметра *expires*, определяет время, по истечении которого информация *cookie* теряет свою актуальность. Если ключевое слово *expires* отсутствует, данные *cookie* удаляются по окончании текущего сеанса работы браузера.
  + Значение параметра *domain* определяет домен, с которым связываются данные *cookie*. Чтобы узнать, следует ли передавать в составе запроса данные *cookie*, браузер сравнивает доменное имя сервера, к которому он собирается обратиться, с доменами, которые связаны с записями *cookie*, хранящимися на клиент-машине. Результат проверки будет считаться положительным, если сервер, которому направляется запрос, принадлежит домену, связанному с *cookie*. Если соответствие не обнаружено, данные *cookie* не передаются.
  + Путь, указанный в качестве значения параметра *path*, позволяет выполнить дальнейшую проверку и принять окончательное решение о том, следует ли передавать данные *cookie* в составе запроса. Помимо домена с записью *cookie* связывается путь. Если браузер обнаружил соответствие имени домена значению параметра *domain*, он проверяет, соответствует ли путь к ресурсу пути, связанному с *cookie*. Сравнение считается успешным, если ресурс содержится в каталоге, указанном посредством ключевого слова *path*, или в одном из его подкаталогов. Если и эта проверка дает положительный результат, данные *cookie* передаются серверу. Если параметр *path* в поле *Set*-*Cookie* отсутствует, то считается, что запись *cookie* связана с URL конкретного ресурса, передаваемого сервером клиенту.
  + Последний параметр, *secure*, указывает на то, что данные *cookie* должны передаваться по защищенному каналу.

Для передачи данных *cookie* серверу используется поле заголовка *Cookie*. Формат этого поля достаточно простой:***Cookie*: *имя=значение; имя=значение*; ...**

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

**Лекция 4. Клиентские сценарии и приложения**

**План лекции:**

Программы, выполняющиеся на клиент-машине

Программы, выполняющиеся на сервере

Насыщенные интернет-приложения

Введение в JScript

Краткая характеристика VBScript

Java-апплеты

ActionScript – общая характеристика

XAML и Microsoft Silverlight

Понятиео DOM

DHTML

Регулярные выражения

1. Программы, выполняющиеся на клиент-машине

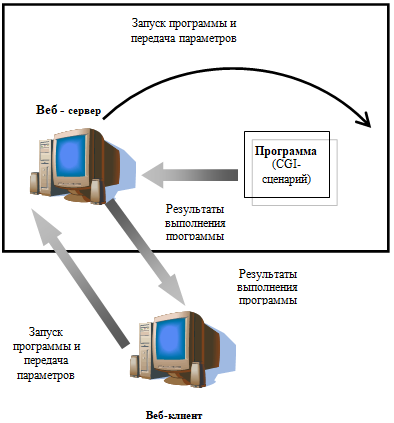
Одним из типов программ, предназначенных для выполнения на клиент-машине, являются*сценарии*, например, JavaScript (VBScript). Исходный текст сценария представляет собой часть веб-страницы, поэтому сценарий JavaScript передается клиенту вместе с документом, в состав которого он входит. Обрабатывая HTML-документ, браузер обнаруживает исходный текст сценария и запускает его на выполнение.

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

2. Программы, выполняющиеся на сервере

Код программы, работающей на сервере, не передается клиенту. При получении от клиента специального запроса, предполагающего выполнение такой программы, сервер запускает ее и передает параметры, входящие в состав запроса. Средства для генерации подобного запроса обычно входят в состав HTML-документа.

Результаты своей работы программа оформляет в виде HTML-документа и передает их веб-серверу, а последний, в свою очередь, дополняет полученные данные HTTP-заголовком и передает их клиенту. Взаимодействие клиен­та и сервера в этом случае показано на рисунке 1.



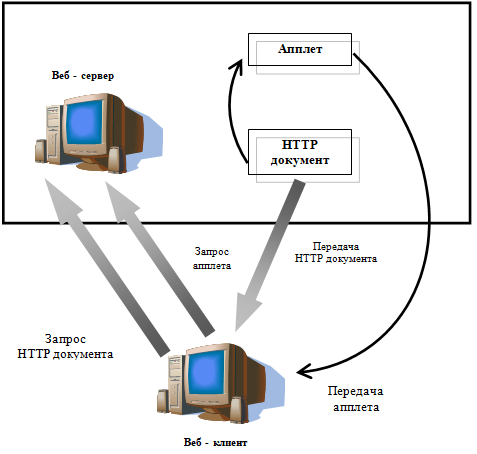
**Рисунок 1.** *Взаимодействие клиента с программой, выполняющейся на сервере.*

**3. Насыщенные интернет-приложения**

*Насыщенное интернет-приложение* (Rich Internet application) – еще один подход, который заключается в использовании *Adobe Flash* или *Java-апплетов* для полной или частичной реализации пользовательского интерфейса, поскольку большинство браузеров поддерживает эти технологии (как правило, с помощью *плагинов*).

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

При использовании Java-апплетов в состав HTML-документа включается специальный дескриптор, описывающий расположение файла, содержащего код *апплета*, на сервере. После того как клиент получает HTML-код документа, включающего *апплет*, он генерирует дополнительный запрос серверу. После того как сервер пересы­лает клиенту код *апплета*, сам *апплет* запускается на выполнение. Взаимодействие между клиентом и сервером при получении *апплета* показано на рисунке 2.



**Рисунок 2.** *Передача клиенту Java-апплета.*

При использовании насыщенных интернет-приложений приходится сталкиваться со следующими проблемами:

* необходимость обеспечения безопасной среды выполнения («песочница»);
* для исполнения кода должно быть разрешено исполнение сценариев;
* потеря в производительности (т.к. выполняется на клиентской стороне);
* требуется много времени на загрузку;

Для разработки насыщенных интернет-приложений используются пакеты *Curl*, *AdobeFlex* и *Microsoft Silverlight*.

**4. Введение в JScript.**

JavaScript - интерпретируемый язык программирования, стандартизированный международной организацией ECMA в спецификации ECMA-262. Языки JavaScript, JScript и ActionScript являются расширением стандарта ECMA-262.

Название "ECMAScript" явилось фактически компромиссом между организациями, вовлеченными в процесс стандартизации, в частности Netscape и Microsoft. Хотя JavaScript и JScript стремились к совместимости с ECMAScript, они имеют ряд дополнительных возможностей, не предусмотренных спецификацией ECMA.

Синтаксис JScript во многом аналогичен языку JavaScript, однако, помимо добавления клиентских скриптов на веб-страницы и некоторых других функций, JScript может использоваться и для других целей, например:

* автоматизация администрирования систем Microsoft Windows;
* создание страниц ASP.

Язык JScript получил дальнейшее развитие в виде языка JScript.NET, который ориентирован на работу в рамках платформы Microsoft.NET

JScript - интерпретируемый, объектно-ориентированный язык. Хотя он имеет существенно меньшее количество возможностей, чем такие объектно-ориентированные языки как C++ и Java.

Возможности языка существенно ограничены:

* язык не позволяет разрабатывать самостоятельные приложения;
* сценарии на JScript могут выполняться только при помощи интерпретатора, в частности веб-браузером.
* JScript - язык без строгого контроля типов. Поэтому не требуется объявлять тип переменных явно. Кроме того, во многих случаях JScript исполняет преобразования автоматически, когда они необходимы. Например, при сложении строки и числа, число будет преобразовано в строку.

Код на JScript пишется в текстовом формате, и организован в *инструкции*, *блоки*, состоящие из связанных наборов инструкций, и *комментариев*. В пределах инструкции можно использовать *переменные* и *данные*, такие как *строки*, *числа* и *выражения*. Для объявления *конца инструкции* используется точка с запятой (**;**). Группа JScript-инструкций, заключенная в фигурные скобки **{}**, называется *блоком*.

*Комментарием* в JScript является текст, расположенный после **//** до конца строки. *Многострочныйкомментарий* начинается с **/\***, и заканчивается **\*/**.

Знак равенства (**=**) используется в JScript как *присваивание*. Следующий код

***Pi = 3.14;***

подразумевает "Присвоить значение *3.14* переменной *Pi*".

При сравнении двух значений на равенство применяется двойной знак равенства (**==**).

JScript *выражения* можно разделить на*логические* или *числовые*. *Выражения* содержат некоторые особенности, к примеру, символ "**+**" означает "добавить к...". Любая допустимая комбинация значений, переменных, операторов, и других выражений является *выражением*.

Объявление *переменной* перед использованием является необязательным. Для этого используется инструкция ***var***. Инструкция *var* является обязательной при объявлении локальной переменной внутри функции. Разрешается объявление переменной неявно - без инструкции var. Однако, в выражениях применять необъявленные переменные не допускается. JScript различает регистр в имени переменной. *Name* и *name* рассматриваются как различные переменные.

**Типы данных**

JScript - язык с *нестрогим контролем типов*, переменные в JScript не имеют строго фиксированного типа. *Переменные* имеют *тип*, эквивалентный типу значения, которое они содержат. Однако, в некоторых случаях, необходимо принудительное преобразование переменной в определенный тип. *Числа* могут быть объявлены как *строки*, а *строки* необходимо преобразовать в *числовой* тип. Для этого применяют функции *parseInt*() и *parseFloat*().

В JScript используется шесть типов *данных*. Основные из них - *числа*, *строки*, *объекты*, *логический*. Остальные два - *null* и *undefined* (т.е. неопределенный).

*Строки* объявляются при помощи *двойных кавычек* или *апострофов*. Строка может состоять из нуля или более символов *unicode*. Когда количество символов равно нулю, это называется пустой строкой (**""**).

JScript поддерживает *числа* как *целые*, так и с *плавающей запятой*. Также существуют специальные представления чисел, например,*NaN* (не число).

Примеры чисел:

3.14 // Вещественное число

15 // Целое число

0177 // Восьмеричное число 177

0XA8 // Шестнадцатиричное число A8

*Логический* тип допускает значения - *true* и *false*. Любое выражение, равное 0, считается эквивалентным *false*, а любое выражение, равное числу, отличному от 0 будет эквивалентным *true*.

*Undefined* – означает, что тип не определен. Значение *undefined* имеет переменная после ее объявления и до присвоения ей какого-либо определенного значения.

Переменная типа *null* - не имеет никакого определенного значения.

**Операторы**

Язык поддерживает *условныевыражения****if*** и ***if*...*else***. При использовании нескольких условий одновременно можно использовать операторы **||** (ИЛИ ) или **&&** (И).

В JScript поддерживается несколько типов циклов: *for*, *for...in*, *while*, *do...while* и *switch*. Также существует инструкция *остановки* выполнения цикла. Оператор завершения *break* может использоваться, чтобы остановить цикл, при выполнении какого-либо условия. Инструкция *continue* используется, чтобы немедленно перейти к выполнению следующей итерации, пропуская остальную часть выполнения кода текущей итерации, но обновляя переменную-счетчик.

**Функции и объекты**

В JScript имеется два вида *функций*: встроенные и определяемые. Программист имеет возможность создавать собственные функции. Определение функции состоит из *объявления параметров* и *блока инструкций* JScript.

Объекты в JScript, по-сути, являются совокупностями *методов* и *свойств*. Все объекты можно разделить на три вида: *встроенные*, *созданные* и *браузерные*. Обработка *объектов* и *массивов* идентична. Можно обратиться к любой части объекта (его свойствам и методам) либо по *имени*, либо по *индексу*. Нумерация индексов в JScript начинается с нуля.

**5. Краткая характеристика VBScript**

*Visual Basic Scripting Edition* (обычно просто VBScript) — сценарный язык программирования, интерпретируемый компонентом *Windows Script Host*. Он широко используется при создании *скриптов* в операционных системах семейства Microsoft Windows.

Язык был создан компанией Microsoft как замена устаревшему пакетному языку, интерпретируемому приложением *command.com*. Синтаксис VBScript является упрощённой версией синтаксиса языка Visual Basic.

Сценарии на языке VBScript чаще всего используются в следующих областях, использующих программные продукты Microsoft:

* автоматизация администрирования систем Windows;
* серверный программный код в страницах ASP;
* клиентские сценарии в браузере Internet Explorer.

**6. Java-апплеты**

Java-апплет - это программа, написанная на языке Java и откомпилированная в байт-код. Выполняется в браузере с использованием виртуальной Java-машины (*JVM*). Апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не возможны в HTML. Так как байт-код Java платформо-независим, то Java-апплеты могут выполняться браузерами на многих операционных платформах.

Java-сервлеты являются серверными приложениями, но они отличаются от апплетов языком, функциями и другими характеристиками.

Предназначены Java-апплеты для выполнения в безопасной среде с целью предотвращения их доступа к локальным ресурсам клиентского компьютера.

Код апплета загружается с веб-сервера, и браузер

* либо вставляет апплет в веб-страницу;
* либо открывает отдельное окно с собственным пользовательским интерфейсом апплета.

Апплет может быть внедрен в веб-страницу с помощью использования HTML тэга **<applet>**, или (что рекомендуется) тэга **<object>**.

Можно назвать следующие преимущества Java-апплетов:

* работают практически на большинстве операционных платформ;
* поддерживаются большинством браузеров;
* кэшируются в большинстве браузеров, что существенно ускоряет их загрузку при возвращении на веб-страницу;
* после первого запуска апплета, когда Java-машина уже выполняется и быстро запускается, выполнение апплетов происходит существенно быстрее;
* загружаются со скоростью сопоставимой с програмами на других компилируемых языках, например, C++, но во много раз быстрее чем на JavaScript.

При этом у Java-апплетов имеются и недостатки:

* требуется установка Java-расширения, которые доступны по умолчанию не во всех браузерах;
* проблемы реализации Java-расширений для 64-разрядных процессоров;
* не могут запускаться до первой загрузки виртуальной Java-машина, что может занимать значительное время;
* разработка пользовательского интерфейса с использованием апплетов является более сложной задачей по сравнению с HTML;
* не имеют прямого доступа к локальным ресурсам клиентского компьютера;
* некоторые апплеты привязаны к использованию определенной среды времени выполнения Java (*JRE*).

**7. ActionScript – общая характеристика.**

*ActionScript* — объектно-ориентированный язык программирования, один из диалектов *EcmaScript*, который добавляет интерактивность, обработку данных и многое другое в содержимое Flash-приложений. ActionScript исполняется виртуальной машиной (ActionScript Virtual Machine), которая является составной частью приложения *Flash Player*. ActionScript компилируется в байткод, который включается в *SWF*-файл.

*SWF*-файлы исполняются *Flash Player*. Сам *Flash Player* существует в виде *плагина* к веб-браузеру, а также как самостоятельное исполняемое приложение. Во втором случае возможно создание исполняемых *exe*-файлов, когда *swf*-файл включается во *Flash Player*.

С помощью ActionScript можно создавать интерактивные мультимедиа-приложения, игры, веб-сайты и многое другое.

**8. XAML и Microsoft Silverlight**

*XAML* (eXtensible Application Markup Language) –язык интерфейсов платформы *Windows Vista*.

Модель приложений *Vista* включает объект *Application*. Его набор свойств, методов и событий позволяет объединять веб-документы в связанное приложение. Объект *Application* контролирует выполнение программы и генерирует события для пользовательского кода. *Документы приложения* создаются с помощью языка *XAML*, который описывает, прежде всего, пользовательский интерфейс. *Логика приложения* управляется процедурным кодом (С#, VB и др.). XAML включает основные четыре категории элементов: *панели*, *элементы управления*, элементы, связанные с *документом* и *графические фигуры*.

*Microsoft Silverlight* является официальным названием основанной на XML и .NET технологии под кодовым именем *WPF/E* (Windows Presentation Foundation Everywhere), являющейся альтернативной *Adobe Flash*. Представляет собой подмножество *Windows Presentation Foundation*, в котором реализованы векторная графика, анимация и средства воспроизведения видео. В версии 1.1 включает в себя полную версию *.NET CLR* - называемую *CoreCLR*, что позволит разрабатывать *Silverlight* приложения на любом из языков .NET. *Silverlight v.1.0* содержит подключаемый модуль браузера для обработки *XAML* и кодеки для воспроизведения мультимедийного содержимого в форматах *WMV*, *WMA* и *MP3*.

*Microsoft Silverlight* представляет браузеру внутреннюю модель *DOM*, управляемую из *JavaScript* кода. Поскольку язык XAML основан на XML, то документ, определяющий загружаемый клиенту пользовательский интерфейс - текстовый и потому вполне пригоден для индексирования поисковыми системами. Используя модель *DOM*, *JavaScript* может динамически обновлять содержимое *Silverlight*, аналогично *DHTML*.

Также можно вызывать методы управления презентацией (запуска анимации или приостановки воспроизведения видео, например).

*Silverlight*-приложение начинается с вызова объекта *Silverlight* из HTML страницы, загружающего *XAML* файл. *XAML* файл содержит объект *Canvas*, выступающий подложкой для других элементов. Объекты *XAML* способны генерировать события, перехватываемые из *JavaScript*.

**9. Понятие о DOM.**

DOM (Document Object Model) - объектная модель документа. Это независящий от платформы и языка программный интерфейс, позволяющий программам получать доступ к содержимому документов, а также изменять содержимое, структуру и вид документов.

В рамках DOM любой документ представляется в виде дерева узлов. Каждый узел представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы между собой находятся в отношении «родитель-потомок».

Изначально различные браузеры имели собственные модели DOM, не совместимые с остальными. Для того, чтобы обеспечить взаимную и обратную совместимость, консорциум W3C классифицировал эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM:

* Уровень 0. Включает в себя все специфические модели DOM, которые существовали до появления Уровня 1, например *document*.*images*, *document*.*forms*. Эти модели формально не являются спецификациями DOM, опубликованными W3C, а скорее отражают то, что существовало до начала процесса стандартизации.
* Уровень 1. Базовые функциональные возможности DOM (HTML и XML) в документах, такие как получение дерева узлов документа, возможность изменять и добавлять данные.
* Уровень 2. Поддержка пространства имён XML, filtered views и событий.
* Уровень 3. Состоит из шести различных спецификаций:
  + DOM Level 3 *Core*;
  + DOM Level 3 *Load and Save*;
  + DOM Level 3 *XPath*;
  + DOM Level 3 *Views and Formatting*;
  + Level 3 *Requirements*;
  + DOM Level 3 *Validation*.

Текущим уровнем спецификаций DOM является Уровень 2, но, тем не менее, некоторые части спецификаций Уровня 3 являются рекомендуемыми W3C.

**10. DHTML.**

*Динамический HTML* или *DHTML* представляет собой набор технологий, которые совместно позволяют создавать интерактивные веб-сайты на основе статического языка разметки (*HTML*), языка создания клиентских сценариев (*JavaScript*), языка описания представления документа (*CSS*) и документной объектной модели (*DOM*).

DHTML позволяет сценарным языкам изменять переменные языка описания представления документа, таким образом, изменяя вид и поведение прежде статического содержимого HTML документа уже после полной загрузки документа и в процессе просмотра его пользователем. Таким образом, динамичность, привносимая DHTML, проявляет себя в процессе просмотра страницы, но не имеет никакого отношения к генерации содержимого страницы при каждой ее загрузке.

В противоположность DHTML,  *динамически генерируемая страница* - более широкое понятие, подразумевающее, например генерацию содержимого веб-страницы индивидуально для каждого пользователя. Это достигается созданием страниц с помощью клиентских или серверных (например, на *PHP* или *Perl*) сценариев.

**11. Регулярные выражения**

Регулярные выражения — система поиска текстовых фрагментов в электронных документах, основанная на специальной системе записи образцов для поиска. Образец, задающий правило поиска, называется «*шаблоном*». Применение регулярных выражений принципиально преобразило технологии электронной обработки текстов.

Многие языки программирования поддерживают регулярные выражения для работы со строками либо в виде отдельных функций, либо имеют уже встроенный в их синтаксис механизм обработки регулярных выражений, например, *Perl* и *Tcl*. Популяризации понятия регулярных выражений способствовали утилиты, поставляемые в дистрибутивах *Unix*.

С помощью регулярных выражений можно задавать структуру искомого шаблона и его позицию внутри строки (например, в начале или в конце строки, на границе или не на границе слова).

При описании структуры шаблона используются:

* гибкая система квантификаторов (операторов повторения);
* операторы описания наборов символов и их типа (числовые, нечисловые, специальные).

**Лекция 5. Серверные веб-приложения.**

**План лекции:**

Стандарт CGI

Разработка приложений на стороне веб-сервера в форме сценариев

Наиболее известные языки разработки сценариев для веб-приложений: Python, Ruby, ASP, ISAPI

**Стандарт CGI**

Круг задач, решаемых Web-сервером, ограничен. В основном он сводится к поддержке НТТР-взаимодействия и доставке клиенту Web-документов. Любые "нестандартные" действия реализуются с помощью специальной программы, которая взаимодействует с веб-сервером и клиентом. Это взаимодействие подчиняется определенным правилам.

Основной набор таких правил - *стандарт CGI* (Common Gateway Interface - ин­терфейс общего шлюза), который определяет порядок запуска программы на компью­тере-сервере, способы передачи программе параметров и доставки результатов ее вы­полнения клиенту. Программа, написанная по правилам CGI, называется CGI-сценарием (script CGI), хотя это не означает, что на сервере не может выполняться двоичный файл.

Благодаря этому интерфейсу для разработки приложений можно использовать любой язык программирования, который располагает средствами взаимодействия со стандартными устройствами ввода/вывода. Такими возможностями обладают в также сценарии для встроенных командных интерпретаторов операционных систем.

Выполнение любой программы (в том числе CGI-сценария) можно условно разде­лить на пять этапов.

1. Запуск программы.
2. Инициализация и чтение выходных данных.
3. Обработка данных.
4. Вывод результатов выполнения.
5. Завершение программы.

Различия между CGI-сценарием и консольным приложением касаются первого, второго и четвертого этапов выполнения.

Каждый раз, когда *веб-сервер* получает запрос от *клиента*, он анализирует содержимое *запроса* и возвращает соответствующий *ответ*:

* Если запрос содержит указание на *файл*, находящийся на жестком диске, то сервер возвращает в составе ответа этот *файл*;
* Если запрос содержит указание на *программу* и необходимые для нее *аргументы*, то сервер исполняет программу и *результат* ее работы возвращает клиенту.

*CGI*определяет:

* каким образом *информация о сервере* и *запросе клиента* передается программе в форме *аргументов* и *переменныхокружения*;
* каким образом программа может передавать назад дополнительную информацию о результатах (например о типе данных) в форме *заголовков* ответа сервера.

В подавляющем большинстве случаев запуск CGI-сценария осуществляется щелчком на кнопке *Submit*, сформированной с помощью дескриптора <*input тyре* = "*submit*">, который находится на HTML-странице между *<form>*и *</form>*. Не зная назначения атрибутов *action* и *method*, невозможно понять, как происходит вызов программы и передача параметров.

Значением атрибута *action* дескриптора <form> является URL файла, содержащего код CGI-сценария. Так, приведенное ниже выражение означает, что файл с кодом CGI-сценария находится на сервере *www.myhp.edu* в каталоге cgi-bin в файле script.рl.

*<form action=”http://www.myhp.edu/cgi-bin/script.pl” method="post">*

Как веб-сервер различает, что надо сделать с файлом, на который указывает URL, — передать его содержимое клиенту или запустить файл на выполнение? Существует два способа распознавания файлов, содержащих тексты CGI-сценариев.

* + Первый способ заключается в том, что при установке веб-сервера один из каталогов специально выделяется для хранения сценариев. Обычно такой каталог получает имя *cgi-bin* (или *Scripts* для веб-сервера *IIS*)*.* В этом случае, если клиент запрашивает файл из каталога *cgi-bin*, сервер воспринимает такой запрос как команду на запуск сценария. Файлы из других каталогов интерпретируются как HTML-документы.
  + Второй способ использует расширение файла. При настройке сервера указывается, что файлы с определенными расширениями содержат коды сценариев.

Идентификация по расширению используется относительно редко. Чаще всего все сценарии помещаются в */cgi-bin*, */Scripts* или в другой каталог, специально выделенный для их хранения.

Вывод результатов выполнения CGI-сценария осуществляется чрезвычайно просто. Для того чтобы данные были переданы клиенту, достаточно вывести их в стандартный выходной поток. Однако, разрабатывая CGI-сценарий, не следует забывать о том, что он все же отличается от консольной программы и имеет следующие особенности.

Информация, передаваемая клиенту, должна соответствовать протоколу HTTP, т.е. состоять из заголовка и тела ответа. Как правило, получив данные от сценария, сервер самостоятельно добавляет первую строку заголовка.

*НТТР/1.0 200 OK*

Формирование информационных полей, входящих в состав заголовка, - задача сценария. Чтобы данные, переданные сценарием, были правильно интерпретированы клиентом, необходимо, чтобы в заголовке присутствовало как минимум поле *Content-type*. За заголовком должна следовать пустая строка. При отсутствии полей заголовка реакция браузера будет непредсказуемой. В подобных случаях браузер обычно пытается отобразить полученную информацию как текстовый файл.

Самый естественный формат для браузера - формат HTML. Результаты работы сценария обычно оформляются в виде веб-страницы, т.е. возвращаемые данные следует дополнить дескрипторами HTML. Таким образом, ответ CGI-сценария клиенту обычно выглядит так:

|  |
| --- |
| Content-type: text/html  <html>  <hеаd>  <titlе>ответсценария</titlе>  </hеаd>  <body>  ……………………  </body>  </html> |

Обратите внимание на пустую строку после выражения *Content-type: text/html*. Она обязательно должна присутствовать в ответе, в противном случае клиент воспримет все последующие данные как продолжение заголовка.

После компиляции программы необходимо скопировать исполняемый файл в каталог *cgi-bin* (или в другой каталог, предназначенный для размещения исполняемых файлов) из которого он может запускаться веб-сервером на выполнение по запросу клиента.

Для вызова данного сценария достаточно включить в веб-страницу следующий фрагмент HTML-кода:

|  |
| --- |
| <form method="post" action="/cgi-bin/hello.exe">  <input type="submit">  </form> |

Если сценарий вызывается из формы, ему передаются те данные, которые пользователь ввел с помощью интерактивных элементов, отображаемых на веб-странице - передача информации CGI-сценарию осуществляется в два этапа: сначала браузер передает данные веб-серверу, затем веб-сервер передает их сценарию.

В большинстве случаев кроме кнопки *Submit* форма содержит другие интерактивные элементы, каждый из которых имеет имя (атрибут NAME) и значение (атрибут VALUE, либо последовательность символов, введенная пользователем). Из имен элементов и их значений формируется строка параметров, которая имеет следующий формат.

*имя=значение&имя=значение& . . . &имя=значение*

Каждый параметр представляет собой имя управляющего элемента и его значение, разделенные знаком равенства, а несколько таких пар объединяют строку с помощью символа "&". Если в состав имени или значения входит символ "&" или "=", то подобные символы кодируются последовательность знака процента "%", за которым следуют две шестнадцатеричные цифры, определяющие код символа. Так, например, последовательностью "%21" кодируется восклицательный знак "!". Как правило, при передаче параметров трехсимвольными последовательностями заменяются все знаки, кроме латинских букв, цифр и символа пробела (последний заменяется знаком "+").

Таким образом, перед использованием строки параметров ее надо декодировать. Алгоритм декодирования чрезвычайно прост и включает в себя следующие действия:

* Выделить из строки параметров пары *имя = значение*.
* Выделить из каждой пары *имя* и *значение*.
* В каждом имени и каждом значении заменить символы "+" пробелами.
* Каждую последовательность из символа "%" и двух шестнадцатеричных и преобразовать в ASCII-символ.

Атрибут *method* дескриптора <*form*> имеет либо значение "GET", либо значение "POST". Значения "GET" и "POST" определяют два различных метода передачи параметров сценарию:

* Если атрибут *method* имеет значение "GET", строка параметров передается вместе с URL вызываемого сценария. Разделителем между URL и строкой параметров является символ "?".
* Если атрибут *method* имеет значение "POST", строка параметров передается в теле HTTP-запроса.

Рассмотрим, как должен вести себя CGI-сценарий, чтобы правильно обработать данные в зависимости от метода, использованного при передаче данных, строка параметров доставляется CGI-сценарию различными способами.

Если атрибут METHOD дескриптора <FORM> имел значение "GET", строка параметр передается серверу в качестве значения переменной окружения QUERY\_STRING.

При использовании метода POST данные доставляются сценарию по-другому. Они передаются через стандартный поток ввода (STDIN). Чтобы сценарий смог определить, сколько символов следует читать из стандартного ввода, веб-сервер устанавливает значение переменной окружения CONTENT\_LENGTH, равным длине строки параметров.

Получив управление, сценарий в первую очередь должен выяснить, с помощью какого метода выполнялась передача параметров. Эта информация содержится в переменной окружения REQUEST\_METHOD.

Таким образом, в простейшем случае, чтобы выполнить обработку строки параметров, достаточно знать назначение трех переменных окружения: REQUEST\_METHOD, QUERY\_STRING и CONTENT\_LENGTH.

Пример сценария на языке Perl, который возвращает клиенту строку параметров, приведен ниже. Сценарий определяет, какой метод использовался для передачи данных, читает строку параметров и передает ее клиенту, предварительно дополнив HTML-дескрипторами.

|  |
| --- |
| $method = $ENV{'REQUEST\_METHOD'};  if ($method eq "GET")  { $pars = $ENV{'QUERY\_STRING'}; }  else  { $length =$ENV{'CONTENT\_LENGTH'}; }  read (STDIN, $pars, $ length);  print "Content-type: text/html\n\n";  print "<HTML><BODY>\n";  print "<P>METHOD = ", $method;  print "<P>String of parameters: <P>\n";  print $pars;  print "</HTML></BODY>\n"; |

При разработке более сложных сценариев может потребоваться дополнительная информация. Информация о типах сервера и браузера, адресе клиент-машины и многие другие сведения передаются с помощью переменных окружения. Некоторые из них перечислены ниже

|  |  |
| --- | --- |
| REMOTE\_ADDR | IP-адрес узла, с которого поступил запрос |
| REMOTE\_HOST | Доменное имя узла, с которого поступил запрос |
| SERVER\_PORT | Номер порта, который использовался при обращении к серверу |
| SERVER\_SOFTWARE | Имя и версия сервера, посредством которого был запущен сценарий |
| SERVER\_NAME | Имя или адрес узла, на котором выполняется сервер |
| SERVER\_PSOTOCOL | Название и версия протокола, с помощью которого был передан запрос |
| HTTP\_USER\_AGENT | Клиентская программа, отправившая запрос серверу |
| HTTP\_REFERER | URL документа, отображаемого браузером при вызове сценария |

**Сценарии**

К основным достоинствам разработки приложений на стороне веб-сервера в форме сценариев можно отнести следующие:

* поскольку сценарии не компилируются, а интерпретируются, то ошибки в сценарии вызовут только диагностическое сообщение, но не приведут к дестабилизации веб-сервера или операционной системы.
* лучшие выразительные возможности. Язык сценариев как правило имеет собственный проблемно-ориентированный набор команд, и одна строка сценария может делать то же, что несколько десятков строк на традиционном языке. Как следствие, на этом языке может писать программист низкой квалификации.
* Поддержка кроссплатформенности.

Поскольку сценарии интерпретируются из исходного кода динамически при каждом исполнении, они выполняются обычно значительно медленнее готовых программ, транслированных в машинный код на этапе компиляции.

В плане быстродействия *сценарные языки* можно разделить на:

* *Языки динамического разбора* (например*command.com*). Интерпретатор считывает инструкции из файла программы минимально требующимися блоками, и исполняет эти блоки, не читая дальнейший код.
* *Предварительно компилируемые* (например*Perl*). Вначале считывается вся программа, затем компилируется либо в машинный код, либо в один из внутренних форматов, после чего получившийся код исполняется.

Рассмотрим кратко наиболее известные языки разработки сценариев для веб-приложений.

**Python**

*Python* — высокоуровневый язык программирования общего назначения с акцентом на производительность и читаемость кода. Язык *Python* сочетает в себе минимализм синтаксиса ядра и большой объём полезных функций в стандартной библиотеке.

*Python* поддерживает *структурную*, *объектно-ориентированную*, *функциональную*, *императивную* и *аспектно-ориентированную* парадигмы.

Его основные архитектурные черты:

* динамическая типизация
* автоматическое управление памятью
* полная интроспекция
* механизм обработки исключений
* поддержка многопоточных вычислений
* удобные высокоуровневые структуры данных

Код в *Python* организовывается в *функции* и *классы*, которые могут объединяться в *модули* (которые в свою очередь могут быть объединены в *пакеты*).

Для всех основных платформ *Python* имеет поддержку характерных для данной платформы технологий (например, *Microsoft COM/DCOM*). Cуществует даже специальная версия *Python* для виртуальной машины *Java* - *Jython*, что позволяет интерпретатору выполняться на любой системе, поддерживающей *Java*, при этом классы *Java* могут непосредственно использоваться из Питона и даже быть написанными на *Python*. Несколько проектов обеспечивают интеграцию с платформой *Microsoft*.*NET*, основные из которых - *IronPython* и *Python.Net*.

Стандартная библиотека *Python* имеет средства для работы со многими сетевыми протоколами и форматами интернета, например, модули для написания HTTP-серверов и клиентов, для разбора и создания почтовых сообщений, для работы с XML и т. п. Набор модулей для работы с операционной системой позволяет писать кроссплатформенные приложения. Существуют также модули для работы с регулярными выражениями, текстовыми кодировками, мультимедийными форматами, криптографическими протоколами, архивами, сериализации данных, поддержка юнит-тестирования и др.

Помимо стандартной библиотеки существует множество библиотек, предоставляющих интерфейс ко всем системным вызовам на разных платформах; Имеется большое количество прикладных библиотек для *Python* в самых разных областях (веб, базы данных, обработка изображений, обработка текста, численные методы, приложения операционной системы и т. д.).

**Ruby**

*Ruby* — интерпретируемый язык высокого уровня для быстрого и удобного объектно-ориентированного программирования. Ruby обладает независимой от операционной системы реализацией *многопоточности*, строгой *динамической типизацией*, «*сборщиком мусора*» и многими другими возможностями. Многие особенности синтаксиса и семантики языка *Perl* заимствованы в Ruby.

Первая общедоступная версия Ruby появилась в 1995 г.

Ruby - полностью объектно-ориентированный язык:

* Все данные являются объектами, в отличие от многих других языков, где существуют примитивные типы.
* Каждая *функция* является *методом*.
* Переменные Ruby содержат не сами объекты, а ссылки на них.
* Присваивание - это не передача *значения*, а копирование *ссылки* на объект.
* В Ruby можно добавлять методы не только в любые классы, но и в любые объекты. Например, можно добавить к некоторой строке произвольный метод.

*Массивы* в Ruby могут автоматически изменять размер, могут содержать любые элементы и язык предоставляет мощные средства для их обработки.

Ruby поставляется с большой стандартной библиотекой. Это, прежде всего, библиотеки для работы с различными *сетевымипротоколами* на стороне *сервера* и *клиента*, средства для работы с различными форматами представления данных (*XML*, *XSLT*, *YAML*, *PDF*, *RSS*, *CSV*, *WSDL*). Также есть библиотеки для работы с *архивами*, *датами*, *кодировками*, *матрицами*, средства для *системного администрирования*, *распределенных вычислений*, поддержки *многопоточности* и т. д.

В языке Ruby также реализован простой и удобный механизм для расширения языка с помощью библиотек, написанных на *Си*, позволяющий легко разрабатывать дополнительные библиотеки. Для унифицированного доступа к базам данных разработана библиотека *Ruby DBI*.

К недостаткам интерпретатора Ruby можно отнести следующие:

* Невысокая скорость работы.
* Отсутствие поддержки потоков операционной системы (для Unix-подобных операционных систем есть поддержка процессов ОС), есть в экспериментальной версии 1.9.
* Отсутствие встроенной поддержки юникода (возможна работа с использованием дополнительных библиотек, есть в экспериментальной версии 1.9).
* Отсутствие компиляции в байткод. (При этом есть возможность компилировать Ruby в Java и .NET байткод, используя компилятор JRuby и Ruby.NET). В экспериментальную версию 2.0 входит виртуальная машина YARV, компилирующая Ruby в байткод и существенно ускоряющая исполнение.

**ASP**

*ASP* (Active Server Pages) — технология, разработанная компанией *Microsoft*, позволяющая легко создавать приложения для Веб.

Программирование на *ASP* дает разработчикам доступ к интерфейсу программирования приложений *InternetInformationServer* с помощью языка сценариев *VBScript* и *JScript*.

*ASP* работает на платформе операционных систем линии *Windows NT* и на веб-сервере *Microsoft IIS*.

Файлы *ASP* представляют собой сценарии, интерпретируемые по мере поступления запросов. *ISAPI*-расширение *ASP.DLL* связано в *IIS* с расширениями файлов *.asp* или *.asa*.

Порядок обработки таких файлов выглядит следующим образом:

* *ASP.DLL*просматривает файлы с указанными расширениями на наличие тегов, обозначающих внедренный код для выполнения на сервер и передает найденный код в *WindowsScriptHost* (*WSH*).
* *WSH* выполняет этот код и возвращает результат файлу *ASP.DLL*.
* *ASP.DLL* передает *IIS* этот результат и содержимое самого файла *ASP*.
* *IIS*возвращает ответ клиенту, от которого поступил запрос*.*

Рассмотрим основы синтаксиса ASP.

*IIS* различает код, выполняющийся на сервере, и содержимое, отправляемое клиенту с помощью *ASP.DLL*, анализируя файл *ASP* на наличие начального “**<%**” и конечного “**%>**” тегов и выполняя код, расположенный между ними, с помощью *WSH*.

Рассмотрим пример:

|  |
| --- |
| <% Language=VBScript %>  <HTML>  <BODY>  <%  Response.Write(“<p>Hello world!</p>”)  %>  </BODY>  </HTML> |

В примере первая строка кода *<% Language=VBScript %>*сообщает о необходимости использовать интерпретатор языка *VBScript*. Для вставки строки в документ был использован метод *Write* стандартного объекта *Response*.

Событие веб-запроса в ASP обрабатывается с помощью следующих объектов:

* *Response*. Используется для записи данных в запрос *HTTP*, возвращаемый клиенту.
* *Application*. Содержит параметры и конфигурации по настройке работы *ASP* для данного веб-сайта.
* *Request*. Хранит содержимое *HTTP*-запроса и обеспечивает вспомогательные функции для обработки данных *HTTP*-запроса.
* *Server*. Содержит информацию о веб-сервере, веб-сайте, а также обеспечивает поддержку вызывающей программы.
* *Session*. Представляет собой состояние заданного веб-сеанса с заданным хостом клиентом.

**ISAPI**

Длявеб-сервера IIS (InternetInformationServer).был разработан специальный программный интерфейс для создания приложений расширяющих стандартные возможности веб-сервера.

*ISAPI* (Internet Server Application Programming Interface) – многозвенный API для IIS.

ISAPI также реализован в виде модуля *mod\_isapi* для веб-сервера *Apache*. Таким образом, серверные приложения, разработанные для MSIIS могут также выполняться в *Apache* и других веб-серверах.

В противоположность CGI - *ISAPI-приложение* загружается в том же адресном пространстве, что и веб-сервер IIS. Это позволяет повысить производительность приложений благодаря сокращению издержек на запуск отдельных процессов. Однако сбой *ISAPI-приложения* может привести к неустойчивой работе самого веб-сервера. В 6-ой версии *IIS* имеется возможность запуска приложений в рамках отдельного процесса.

*ISAPI* включает в себя 2 компоненты: *расширения* и *фильтры*.

Таким образом, все многообразие разрабатываемых *ISAPI*-приложений сводится только к этим двум типам. И *фильтры* и *расширения* компилируются в *DLL* файлы динамически запускаемые веб-сервером.

*ISAPI*приложения могут разрабатываться с помощью любых языков, поддерживающих экспорт стандартных *С*-функций, например *С*, *С++,DelphiPascal*. Для разработки имеется ограниченное число библиотек для разработки *ISAPI*приложений, например*Intraweb*-компоненты *DelphiPascal*, специальные *MFC-*классы, специальная*С++* библиотека серверных технологий *ATL*.

К наиболее важным особенностям ISAPI-расширений можно отнести следующие*:*

* *ISAPI-расширения* имеют доступ ко всем функциональным возможностям *IIS*.
* Реализуются в виде *DLL*-модулей, загружаемых в пространстве процесса, контролируемого *IIS*.
* Клиенты могут обращаться к *ISAPI-расширениям* также как к статическим *HTML* страницам.
* *ISAPI-расширения* могут быть ассоциированы с отдельными расширениями файлов, с целыми каталогами или сайтами.

*ISAPI-фильтры* необходимы для изменения или совершенствования функциональности *IIS*. Они обычно работают с IIS-сервером и фильтруют каждый запрос. Фильтры применяются для анализа и модификации входящих и исходящих потоков данных.

Фильтры также как и расширения реализуются в виде DLL файлов.

Обычно ISAPI-фильтры используются для решения следующих задач:

* Изменение данных в запросе клиента (URL или заголовков).
* Управление отображением URL в физические файлы.
* Управление именами и паролями пользователей при анонимной или базовой аутентификации.
* Анализ и модификация запросов по завершении аутентификации.
* Модификация ответа веб-сервера.
* Ведение журналов и анализ трафика.
* Реализация собственной аутентификации.
* Управление шифрацией и сжатием.

Стоит отметить, что существуют реализации в виде ISAPI-расширений для таких инструментальных средств как:

* ASP(Active Server Pages )
* ASP.NET
* ColdFusion
* Perl ISAPI (Perlis)
* PHP

**Лекция 6. Правила web-дизайна**

**План лекции:**

Основные элементы web-страницы

Правила подготовки текстовой информации

Выбор цвета и оптимизация графики

Определение навигационной структуры

Логическая и физическая структура сайта

Принципы компоновки и алгоритм создания web-сайта

Регистрация сайта в сети

В строгом смысле слова ***дизайн (design) - это разработка, конструирование. Web-дизайн включает в себя - визуальный дизайн (вообще), дизайн представления информации на экране (специфика web-дизайна).***

Web-дизайнер в зарубежном понимании - это не кто иной как web-разработчик - верстальщик (web-master/web-coder), специалист по информационной архитектуре (web-developer), программист (web-programmer), администратор (system administrator) в одном лице.

Дизайн ресурса, должен быть выдержан в едином стиле для обеспечения комфорта пользователя и создания цельного впечатления от узла, как от неделимого источника информации, а не как набора отдельно взятых страниц. Прежде всего это таблицы, фреймы, цвет и шрифт.

Дизайн играет важную роль в привлечении внимания пользователей сети Internet к определенной Web-странице. При разработке дизайна Web-страницы прежде всего нужно учитывать особенности восприятия целевой аудитории. При этом оформление должно соответствовать содержанию Web-страницы. Универсального решения в области разработки дизайна Web-страниц не существует. Возможны совершенно различные варианты исполнения Web-страницы. Все Web-страницы одного сервера должны быть оформлены в едином стиле. Это создаст дополнительное представление о фирме и ее товарах.

Дизайн Web-страниц предполагает разработку следующих элементов:

* *цвет;*
* *шрифт;*
* *графика;*
* *компоновка Web-страницы.*

При этом сеть Internet налагает некоторые ограничения на возможности оформления Web-страниц - время загрузки Web-страниц, пропускная способность канала передачи данных, размер графических файлов, совместимость браузеров, передача цветовой палитры и др.   
Единая цветовая гамма Web-страниц способствует быстрому и полному восприятию содержания. Как правило, лучшая комбинация цветов для чтения - белый фон и черный текст. Можно использовать фоновый цвет для отдельных символов текста, что дает возможность создавать интересные композиции для заголовков, подписей к рисункам. Однако не следует изменять цвета ссылок, поскольку пользователь может решить, что уже посещал их раньше и не нашел ничего интересного.  При выборе шрифта необходимо принимать во внимание следующие рекомендации:

* Не следует использовать очень крупный размер шрифта (особенно с полужирным и курсивным начертанием) для текстовых надписей, так как он чаще всего производит грубое впечатление. Тогда как шрифт обычного размера при умелой подаче кажется насыщенным информацией и располагает к вдумчивому прочтению.
* Не следует применять такие приемы, как подчеркивание и перечеркивание, так как подобные текстовые эффекты могут ассоциироваться у пользователей с гипертекстовой ссылкой.
* Навигационное меню на всех страницах должно иметь одни и те же форму, цвет, размеры и месторасположение, что обеспечивает легкость перемещения по серверу.
* Не следует заполнять страницы избыточным количеством графических материалов. Не следует включать в страницу непрерывно двигающиеся элементы - они перенапрягают периферическое зрение. Нужно сохранять баланс между текстом, изображениями и пустым пространством.   
  Закончив создание Web-страницы, нужно посмотреть на нее со стороны потенциального пользователя. Возможно разработка Web-страницы будет происходить на мощном компьютере Power Mac или Sun, однако большинство пользователей могут находиться в значительно худших условиях, и важно знать, как выглядит Web-страница с их точки зрения.

**Цвет**

Важность цвета в нашей жизни поистине огромна. Так же огромна она и в Интернете, где всё на этом и держится. Умение правильно использовать цвет позволяет задать благоприятную атмосферу сайта, погрузить посетителя в свой мир. Важно всегда сначала продумывает цветовую схему, потом только начинать создавать графику. Проблема восприятия сайта является одной из наиболее важных задач. Когда пользователь попадает на какой-либо сайт, очень важным моментом становится первое впечатление, которое он получает от ресурса. Это становится отправной точкой для решения пользователя, оставаться ли ему здесь или искать информацию в другом месте. Сайт, который обладает тяжелой психологической атмосферой, отталкивает пользователя, как следствие, у него пропадает желание находиться именно на этом сайте, что может привести к потери потенциального клиента.

Очень важно, чтобы глаз не уставал от увиденного, и при этом пользователь мог выделить всю наиболее важную для него информацию. Яркие цвета слишком давят. Блеклые не всегда воспринимаются отчетливо. Очень важно суметь передать идею, не затмив содержание. Ведь сайт — это не просто информационный ресурс, прежде всего, это лицо и имидж компании. Он отображает внутреннюю атмосферу, царящую в организации. Сайт - это рекламный продукт, информационный ресурс, дающий большие перспективы в области поиска клиентов. Сайт, который не обладает приятной и дружеской атмосферой по отношению к пользователю, не может выполнять все возложенные на него функции. Единая цветовая гамма Web-страниц способствует быстрому и полному восприятию содержания.

На страницах нужно использовать максимум три-четыре цвета, даже если видеокарта способна передать триллионы цветов. Чем проще, тем лучше, многие сайты, напичканные сотнямицветовых оттенков болезненно воспринимаются, слабо запоминаются, плохо вспоминаются. Некоторые способны сделать отличный сайт, используя всего два цвета. Если во всю использовать на сайте цветные фотографии, то можно использовать их доминирующий цвет для заголовков. Для разных типов данных желательно использовать разные оттенки. Например, заголовки делать синим цветом, предупреждения лучше делать красным, цвет основного текста должен быть тоже определён однозначно. Цвета фона и основного текста должны быть контрастными. Чем контрастнее, тем удобнее будет читать текст. Чёрный на белом, жёлтый на чёрном, гораздо удобочитаемы. На тёмном фоне лучше всего использовать белый или жёлтый цвета, голубой читается сложнее, можно использовать зелёный.

На некоторые цвета у большого количества людей преобладает негативная реакция:

Черный, Красный, Серый, Коричневый, эти цвета при доминировании на сайте вызовут отрицательную оценку сайта. При умелом сочетании цветов влияние негативной составляющей этих цветов можно нивелировать (пример: Синий-черный) или даже превратить в плюсы. Такого эффекта зачастую можно достичь при плавных переходах цветов.

Если посетителя нужно зарядить энергией, взбодрить его, мотивировать к чему-либо, нужно использовать сильные оттенки. Если, наоборот, нужно создать мирную атмосферу, лучше использовать пастельные тона, ослабить агрессивные цвета. Если сайт посвящен фильмам ужасов, следует смело использовать гаммы черного, серого, коричневого и красного цветов.

Для молодёжи с домашним Интернетом более привычен ночной выход в Интернет. Для них лучше создавать тёмный фон, светлый текст. Если аудитория состоит из людей пользующихся Интернетом на своей работе (в офисах, учреждениях), следует делать светлый сайт с тёмным текстом. По статистике светлый фон лучше тёмного.

**Шрифты**

Очень важная область дизайна - типографика - шрифтовое решение композиции. Безусловно, шрифт — самый интересный, самый сложный и самый благодарный из всех строительных материалов дизайнера. Ни одна работа не об­ходится без шрифтовых заголовков, логотипов, надписей. К сожалению современные веб-возможности ограничены двумя группами очень похожих и не очень красивых шрифтов: рублеными (Arial, Verdana etc.) и антиквами во главе с Times New Roman. Все остальные шрифты могут оказаться не установленными на компьютере пользователя, и их использование регулярно приводит к невозможности посетителем прочесть послание, - вместо текста будут показаны закорючки. Если же необходимо написать текст каким-нибудь особенным шрифтом, то используется gif изображение для всего текста (например, для заголовков) или для каждой буквы в отдельности, что гораздо экономичнее, но чрезвычайно трудоемко. Тем не менее, использование текста в виде картинок приведет к многократному (sic) увеличению объема страницы, что крайне нежелательно.

Шрифт характеризуется гарнитурой (определяет форму и носит название шрифта, serif, sans-serif, monospace, декоративные, рукописные), кеглей (размер шрифта, измеряется в пунктах), начертанием (курсив, жирный, наклонный).

Используется единый шрифтовой тег для всех заголовков ресурса. Композиция должна содержать минимальное количество резко отличающихся шрифтов. В лучшем случае это рубленый (заголовки, небольшие тексты, хорошо выглядит при жирном начертании) и шрифт с засечками (удобен для чтения, хорошо выглядит при курсивном начертании).

Следующей задачей является подбор параметров набора:

1. кегль шрифта (чем больше шрифт тем больше его значимость, но более мелкие надписи заставляют вчитываться или просто игнорируются);
2. соотношение прописных и строчных букв (красота заглавных букв по принципу контраста с прописными наблюдается у шрифтов с засечками; мелкий кегль требует написания текста только прописными буквами; для декоративных и рукописных – хорошо выглядит выделение только первых букв, что придает повествовательность фразы; все слова начинаются с заглавных букв – неприемлемо для русского языка);
3. интервал между буквами (кернинг, трекинг) и строками (интервалы между буквами в основном рассчитаны на кегль в 10 пунктов; чем больше кегль тем больше интервал между буквами; разрядка заставляет звучать фразу медленнее и торжественнее; заголовок из прописных букв с увеличенным интервалом выглядит привычнее и строже; в заголовках иногда применяется нулевой междустрочный интервал, что приводит к наезду строк и поэтому необходимо установить достаточный контраст между перекрытыми строками по кеглю или цвету, это также приводит к возникновению сильной оси).

Главное - это единый образ ресурса. Подбор оптимального шрифта и правильность его расположения одна из важнейших задач.

**Оформление текста**

Удобочитаемость текста зависит не только от гарнитуры шрифта, во многих случаях это не самая важная составляющая. Длина строки, контрастность цветов текста и фона, соотношение текста и пустого пространства на странице, размер шрифта и, наконец, взаимодействие всех указанных факторов оказывают большее влияние на юзабилити\* веб-публикаций.

**Работа с изображениями**

Чаще всего, контентными изображениями являются семейные фотографии (походные, свадебные, институтские и пр.), иллюстрации продаваемой продукции, рисунки или картины, кадры из фильмов, скриншоты игрушек и т.д. Иллюстрации - это и благо, и бич Интернета. Благо, потому что с их помощью можем лучше донести до посетителя нужную информацию, - визуальные образы практически всегда более наглядны и понятны, нежели описательные. Фактически, без картинок Интернет бы не переживал сейчас такой бум, который мы можем наблюдать последние три-четыре года. В то же время, современные скорости работы телекоммуникационного оборудования, составляющего «железную» основу Интернета, не в состоянии предоставить веб-мастерам того комфорта при работе с изображениями, который есть у дизайнеров печатной продукции. В результате веб-мастера, все время должны помнить об объеме получающегося изображения, чтобы посетителям не пришлось полчаса ждать его.

Существует два основных формата графических данных, используемых в веб-мастеринге - это GIF и JPEG, хотя они и не единственные, но на данный момент наиболее распространенные. Формат GIF используется для презентационной (деловой) графики: схем, диаграмм, графиков, а также для анимации, однако если в распоряжении есть подлежащая выкладыванию на сайт цветная фотография, то лучше использовать JPEG.

###### Формат GIF

GIF - GraphicsInterchangeFormat, формат взаимообмена графикой. Изображение не может содержать более 256 цветов (именно поэтому он не пригоден для фотографий), файл может содержать несколько изображений и описание, отражающее последовательность их воспроизведения, то есть, может использоваться для записи анимации. Формат поддерживает прозрачный фон, что делает его чрезвычайно удобным, когда есть необходимость размещения какого-либо графического элемента на сложной текстуре. Анимированные GIF умеет делать PhotoShop версии 5.5, совмещенный с ImageReady\*. Кроме того, существует огромное количество программ сторонних разработчиков, которые поддерживают как анимированные, так и статичные GIF, в том числе и специально приспособленные к работе с изображениями для Интернета. Одной из лучших программ можно считать Ulead GifAnimator версии 2 и выше, а также Gif Construction Set версии не ниже 1.3. Все эти программы позволяют очень качественно «сжимать» полученные файлы, эффективно уменьшая их объем.

###### Формат JPEG

Содержит не менее 8 бит на канал, то есть может содержать только полноцветное или серое изображение, а, следовательно, плохо приспособлен для деловой графики. Важное отличие от GIF (как и от других графических форматов) - искажающее сжатие, которое позволяет получить изображение очень маленькое. Алгоритм сжатия основан на удалении части цветовой гаммы изображения, с учетом того, что в целом и сложном изображении человек не очень хорошо различает близкие цвета. За счет этого изображения могут быть сжаты в несколько сотен раз, однако, чем больше сжатие, тем больше полутонов удаляется и тем заметнее искажения в изображении. Формат JPEG поддерживается всеми современными графическими программами, включая и самые простые.

#### *Об иллюстрациях вообще*

Принято считать разрешение мониторов равным 72 dpi. Поэтому все изображения для web следует подготавливать именно с таким разрешением. Также важно обязательно учитывать, что в нашем распоряжении есть не более 600-700 точек по ширине экрана, иначе не избежать горизонтальной прокрутки изображения.

Для фона надо приготовить изображение форматом 100\*100 точек и браузер сам повторит его по всему полю экрана пользовательского монитора. Подогнать края можно, немного растушевав 5-6 пикселов по обрезу картинки.

В зависимости от назначения изображения стоит придерживаться следующих эмпирических размеров:

Для большого баннера 468х60 - не более 15 kb - это требование большинства баннерных сетей.

Для кнопок навигации - 1-2 kb, чтобы они не перегружали страницу.

Для основного изображения - не более 100 kb, причем, только в том случае, если изображение самое главное содержание конкретной страницы.

Для фонового изображения - 5 kb, но лучше сделать фон не более 3 kb.

Все эти размеры, теоретически должны укладываться в общую величину 70-100 килобайт на страницу, хотя, надо сказать, что и эти цифры достаточно высоки для средних страниц, обычно, если не слишком увлекаемся анимацией и Java, общий объем не будет получаться более 30-40 kb (если это не галерея картинок).

### **Распределение материала**

Деление материала, прежде всего, долж­но быть *логичным*, так чтобы каждая страница была посвящена одной теме и чтобы всякая тема занималаодну и только одну отведенную под нее страницу. Кроме то­го, нужно учитывать ограничение объема файлов. С другой стороны, каждый щелчок мышью по ссылке требует от пользователя определенной затраты сил, поэтому слишком много слишком маленьких страниц — тоже не лучший выход.

Если информационные единицы, из которых состоит сайт, слишком мелки, можно объединять на одной странице несколько таких единиц — но только в том случае, если они сравнимы по объему и важности и расположены на одном уровне иерархии. Нередко, особенно на сайтах академиче­ского толка, можно встретить гигантские страницы в сотни килобайт, собравшие всю информацию по какой-нибудь большой теме и образующие иерархию не подчиненных страниц, а подразделов в пределах одного HTML-файла. Это допустимо только в академическом стиле, имеющем достаточно уровней вложенности заголовков и не обремененном графикой и таблицами.

Некоторые сайты позволяют переходить к другим Ресурсам только в два приема: ссылка в основном тексте ведет не прямо к пункту назначения, а в специальный раздел (или на отдельную страницу) со списком всех внешних ссылок сайта. Это позволяет не загромождать текст лишней информацией и унифицировать оформление ссылок (на­пример, ставить цифры или звездочки вместо названий внешних сайтов); в списке же можно, наоборот, поместить сколь угодно подробные описа­ния, аннотации и комментарии для каждой ссылки.

Есть и еще одно обстоятельство: владельцам контент-сайтов, живущим за счет рекламы, выгодно нарезать свою инфор­мацию на как можно более мелкие кусочки, чтобы иметь право повесить сверху и снизу каждого такого кусочка по рекламному баннеру. Размещение баннеров посереди­не страницы, как и прерывание рекламными вставками фильмов по телевидению, многими справедливо счита­ется слишком раздражающим и потому неэффективным приемом.

**Определение навигационной структуры**

Навигация – это схема, по которой посетитель web-сервера просматривает гипертекстовые документы. Структура логической и качественно выполненной навигационной системы тесно связано с первой страницей. Пользователь приходит на сайт с целью получить определеннуюинформацию, и система навигации должна помочь ему добраться до нее. Желание же пользователя кликнуть по ссылке зависит от того, какона сформирована, оформлена иразмещена. Система навигации – стержень сайта, это образующая единица - то, что делает любой информационный ресурс востребованным, интересным для широких слоев публики.

В системе должно быть несколько полностью взаимозаменяющих друг друга вариантов. Некоторые предпочитают пользоваться поиском, кто-то стрелками внизу страницы, кто-то системами меню-подменю, чтобы угодить всем, нужно предусмотреть все эти варианты и сбалансировать их при размещении на сайте. Структура навигации неразрывно связана с типом сайта, его направленностью. Так для текстово-информационного сайта с объемными материалами хорошо подходит развернутое меню линейного характера. С другой стороны, для развлекательного ресурса с большим количеством мелких текстов или рисунков требуется более детальная рубрикация и, возможно, перебор материалов внутри каждой рубрики.

Способы реализации навигационной системы: текстовая навига­ция; поисковая система; панель с кнопками; меню; выпадающее список меню (вертушка); раскрываемое оглавление; карта сайта; перебор страниц.

*Расположение навигации*

Привычнее всего выглядит вертикальное (как правило, слева от основного содержимого) расположение панели ма­териалов и горизонтальное (вверху или внизу страницы) — панели инструментов. Иногда на панель инструментов доба­вляют ссылку, ведущую с подчиненных страниц на первую страницу сайта («В начало» или «К началу»), и кнопку со ссылкой на почтовый адрес автора или владельца сайта («Feedback», «Contact Us» или «Пишите нам»). Если эти две кнопки при­сутствуют, то они обычно занимают в панели крайне левое и крайне правое положение соответственно. Чаще, впрочем, почтовая ссылка включается не в навигационную панель, а в состав блока подписи внизу страницы; для ссылки же на первую страницу сайта нередко «по совместительству» используется изображение логотипа сайта или фирмы, ко­торое большинство сайтов размещают в одном и том же положении на каждой странице.

Одним из вариантов использования меню является горизонтальная линейка в самом верху сайта. Небольшое знание Java способно превратить эту строку во вполне привычную каждому пользователю систему навигации в windows-приложении. Однако эргономически, это решение не слишком удачно, так как одна строка меню уже есть в окне - это меню непосредственного самого браузера, куда и будет непроизвольно тянуться рука пользователя.    
 С этой же точки зрения, выгоднее размещение меню справа, а не слева от основного информационного поля страницы. Связано это, в первую очередь, с тем, что глаза человека естественно двигаются слева направо, поэтому, прочтя основную информацию страницы, он переводит глаза направо-вверх, чтобы перейти к следующему материалу и ничего не обнаруживает.

*Семантика*

Первое, на что стоит обратить внимание в панели навигации верхнего уровня\*, — общее количество кнопок на панели, которое не должно превышать десяти, а еще лучше семи. Как показывают исследования психологов, это «магическое число» — максимум одноуровневых объектов, в которых человек может ориентироваться с бессознатель­ной легкостью. Если количество разделов верхнего уровня на сайте превышает 10, нужно постараться свести не­которые из них в один раздел или же ввести двухуровневую иерархию ссылок на самой панели, по-разному оформив ссылки на разделы и подразделы. Другой выход — разбиение одной большой панели на не­сколько маленьких. Обычно для этого ссылки разводят на две категории: материалы и инструменты. Материалы — это тематические разделы с тем содержимым, ради ко­торого пользователи и приходят на сайт: текстами, изображениями, коллекциями сетевых адресов и т.п.. Ин­струменты — это ссылки, ведущие на различные вспомогательные страницы, облегчающие доступ к основной информации: карту сайта, страницу поиска, а для сетевых магазинов и других сай­тов со сложным интерактивным интерфейсом — страницы помощи и инструкций.

Привычнее всего выглядит вертикальное (как правило, слева от основного содержимого) расположение панели ма­териалов и горизонтальное (вверху или внизу страницы) — панели инструментов.

Панель навигации верхнего уровня\* – это панель первой страницы, обычно повторяемую в почти неизменном виде на большинстве остальных страниц сайта.

Иногда на панель инструментов доба­вляют ссылку, ведущую с подчиненных страниц на первую страницу сайта («В начало» или «К началу»), и кнопку со ссылкой на почтовый адрес автора или владельца сайта («Feedback», «Contact Us» или «Пишите нам»). Если эти две кнопки при­сутствуют, то они обычно занимают в панели крайне левое и крайне правое положение соответственно.

Чаще, впрочем, почтовая ссылка включается не в навигационную панель, а в состав блока подписи внизу страницы; для ссылки же на первую страницу сайта нередко «по совместительству» используется изображение логотипа сайта или фирмы, ко­торое большинство сайтов размещают в одном и том же положении на каждой странице.

**Разновидность навигационной панели**

*Текстовая навига­ция*

Про­стейшая разновидность навигационной панели для сайта сдревовидной топологией — список тексто­вых ссылок на подчиненные страницы, иногда содержащий краткие аннотации по каждой ссылке. Текстовая навига­ция эффективна, но недостаточно экономна — аннотации занимают много места, и из-за нелюбви пользователей к прокрутке со­держимого последние ссылки в списке могут страдать от недостатка внимания. Поэтому аннотированными ссылками обычно пользуются для немногих особо важных или недавно появившихся разделов сайта, и в сочетании с соответствующими иллю­страциями такие ссылки составляют основное содержание первой страницы на многих корпоративных и контент-сайтах. При необ­ходимости разместить на заданной площади несколько совершенно одинаковых по значимости объектов выгоднее сделать один-два из них главными, крупно и выигрышно подав их на площади, сэкономленной за счет беглого перечисления всех остальных объектов. Этот принцип искусственного создания неравенства очень часто используется для ссылок на первой странице.

*Поисковая система.*

Поиск - это вторая по популярности система навигации в интернет. Может быть она не столь часто применяется в индивидуальных сайтах, но множество людей отыскивает необходимые им ресурсы именно с помощью поисковых машин.  Поэтому у большинства из них не должно возникать трудностей при использовании поисковой системы навигации на сайте. однако поиск может быть лишь вторичным элементом навигации на сайте, а не основным; он может дополнять меню или пиктограммы, или еще что-либо, но не быть основным средством передвижения на сайте.

Это заложено в самом понятии поиск - том ассоциированном действии, которое мы под ним подразумеваем. Мы ищем, следовательно, мы что-то потеряли, однако, у пользователя не должно возникать ощущения, что он что-то потерял, так как это чувство некомфортно - очень близко к "заблудился".  Следовательно, поиск необходимая вещь для любого текстоориентированного ресурса объемом более 100 страниц.

*Панель скнопками*

Для дублирования на всех остальных страницах сайта нужно более компактное средство навигации — на­вигационная панель с кнопкам. Такие панели чаще всего вытянуты в одном измерении (по горизонтали или вертикали), хотя на первой странице иногда можно увидеть иллюстрированные и даже анимированные двумерные «карты» содержимо­го сайта. Иногда, особенно для навигации в обширных многоуровневых иерархиях, используются выпадающие ме­ню. На особо разветвленных древовидных сайтах каждая страница ссылается на всех своих «родителей» — членов цепочки, ведущей от корня сайта к данной странице.

Для сайтов или фрагментов сайтов с линейной структурой необходимый минимум навигации — пара ссылок, ведущих к предыдущему и следующему доку­менту в цепочке. Чаще, однако, линейная навигационная панель включает в себя строку ссылок на все страницы цепочки, указывая в этом ряду место текущей страницы. Сходно устроены панели управления на сайтах, входящих в тематические «кольца» (web rings), с помощью которых можно перейти на следу­ющий или предыдущий сайт кольца, посмотреть полный список сайтов и выбрать один из них.

**Меню**

Самая распространенная система навигации. Действительно, это наиболее понятная и удобная для большинства людей система, так как все умеют пользоваться оглавлением в книге и эта система, без сомнения, интуитивно понятна. Постоянно висящее на экране меню предоставляет пользователю широкие возможности для проложения маршрута по сайту.

К сожалению, действительно удобным можно считать только такое меню, где перечислены все страницы сайта, но для ресурса объемом более 50 страниц это уже предоставляется малореальным. Частично решить эту проблему помогает раскрывающиеся системы меню, основанные на относительно простом Java скрипте, но даже и в этом случае предел наступает в районе 100-120 страниц сайта.  Одним из вариантов использования меню является горизонтальная линейка в самом верху сайта. Небольшое знание Jav'ы способно превратить эту строку во вполне привычную каждому пользователю систему навигации в windows-приложении. Однако эргономически это решение не слишком удачно, так как одна строка меню уже есть в окне - это меню непосредственного самого браузера, куда и будет непроизвольно тянуться рука пользователя.

С этой же точки зрения, выгоднее размещение меню справа, а не слева от основного информационного поля страницы. Связано это, в первую очередь, с тем, что глаза человека естественно двигаются слева направо, поэтому, прочтя (или просмотрев) основную информацию страницы, он переводит глаза направо-вверх, чтобы перейти к следующему материалу и… ничего не обнаруживает. Теперь, чтобы перейти к расположенному слева меню, ему необходимо делать над собой пусть небольшое, но сознательное усилие, - комфортное состояние потеряно.

*Выпадающий\_список\_меню­\_(вертушка).*

Будучи, по сути, очень близкой к системе меню, вертушка обладает рядом своеобразных свойств, которые позволяют выделить ее в отдельную систему навигации. Выпадающий список организуется простейшим GCI скриптом.

Выпадающий список имеет ряд недостатков и главный из них - это не наглядность. Действительно, по списку, в отличие от открытого, линейного меню, не видно, что есть на сайте, и посмотреть это можно, только открыв список. Второй серьезный недостаток - это непривычность. Выпадающие списки - это единственная система навигации, которая таковой не воспринимается, то есть ею пользуются, но для того чтобы осознать, что это элемент навигации, нужно сделать над собой усилие.

Из достоинств выпадающих списков можно отметить их чрезвычайную компактность, а также простоту управления, так как CGI скрипт, которым управляется список, редактируется в одном файле, а не на всех страницах сайта. Кроме того, список довольно легко вписать в любой макет без особого ущерба для вида сайта.

*Раскры­ваемые оглавления*

Динамический HTML делает возможным еще более эффек­тивное представление двухуровневых иерархий — раскры­ваемые оглавления (expandable outlines). Сразу после загрузки страницы такое оглавление представляет собой список раз­делов верхнего уровня, каждый пункт в котором в ответ на щелчок мыши раскрывается, выводя список относящихся к нему подразделов и сдвигая вниз расположенные ниже разделы. Раскрытие другого раздела, как правило, возвраща­ет в исходное состояние предыдущий; иногда предусматри­вают также ссылку, позволяющую раскрыть одновременно все разделы. Этот прием, обеспечивающий компактное и обозримое представление большого количества информа­ции на ограниченной площади страницы, незаменим для контент-сайтов. В браузерах, не поддерживающих динами­ческий HTML, все списки подразделов будут видны сразу же в раскрытом виде.

**Карта сайта.**

Карта сайта предназначенная для суперразветвленных сайтов с большим количеством документов, эта система, порой, является единственным выходом для отчаявшегося вебмастера. Можно занять целую страницу только лишь описанием ресурса и страниц на нем, при этом не мучиться тем, как же расположить меню так, чтобы оно было достаточно компактным и при этом достаточно информативным. Следует учитывать, что, также как и поисковая система, карта сайта может быть только дополнительной системой навигации. Карта сайта может быть выполнена как в виде дерева, на котором упорядоченно выложены все страницы сайта, так и в укороченном виде.  
 В любом случае, придется мириться с тем, что организованная карта сайта не будет выглядеть такой элегантной, как того хотелось бы. Обычно получается не слишком красиво просто потому, что объем текстовой информации очень велик, а сама эта информация требует специфического расположения. Карта сайта требует тщательного продумывания дерева каталога, чтобы пользователь мог быстро найти то, что ему требуется. Существуют программы, которые умеют генерировать карту сайта по структуре каталога, где он расположен.

*Перебор\_страниц.*  
 Это одна из лучших систем навигации, но она же и самая трудоемкая. В самом низу страницы располагаются кнопки "вперед", "назад" и "наверх", которые проводят пользователя последовательно со страницы на страницу, что позволяет погружать его в материалы сайта, не отвлекая на возвращение к меню, титульной странице и пр.  Однако эта система требует либо написания специального скрипта, либо ручного размещения и изменения ссылок на каждой странице сайта. В частности, именно из-за трудоемкости и трудноконтролируемости этой системы, она не подходит для крупных сайтов, но идеальна для небольших текстовых ресурсов.  Система действительно уникально своей дружественностью к пользователю. Наличие стрелок внизу страницы увеличивает глубину изучения сайта в полтора-два раза.

*Пиктографическая\_система.*

Направление пользователя с помощью пиктограмм популярно, но несколько спорно, так как пиктограммы могут вызывать различное прочтение у пользователей. Тем не менее, это самая компактная система навигации, одновременно сочетающая в себе и наглядность, и удобство и возможность быть вписанной в страницу с любым дизайном. С точки зрения интуитивности, пиктографическое меню - это единственный случай, когда имеет смысл обратиться за картинками в общедоступные клипарты, чтобы картинки вызывали как можно меньше разночтений, так как это основное требование к пиктографическому меню

**Логическая и физическая структура сайта**

Каждый ресурс Интернета, от любительской домашней странички до большого информационного портала, содержит несколько тематических рубрик, соединенных между собой гиперсвязями. Как правило, ссылки на все разделы сайта с краткими анонсами их содержимого приводится на первой, так называемой стартовой странице, которой присваивается имя index.htm (.html). Если тематические рубрики содержат собственные подразделы, каждая из них также имеет свою стартовую страницу, называющуюся index.html.

ПРИМЕЧАНИЕ Такое имя файла рекомендуется присваивать всем стартовым документам сайта, поскольку в противном случае при обращении к какому-либо разделу посредством сокращенного URL без указания названия стартовой страницы (например, http://www.mysite.ru/photos/ вместо [http://www.mysite.ru/photos/ startpage.html](http://www.mysite.ru/photos/startpage.html)) броузер отобразит не саму web-страницу, а перечень хранящихся в данной папке файлов.

Подобный набор тематических рубрик с распределенными по соответствующим разделам документами и заранее спроектированными гиперсвязями между всеми страницами ресурса и называется логической структурой сайта. Физическая структура, напротив, подразумевает алгоритм размещения физических файлов по поддиректориям папки, в которой опубликован ваш сайт. Пример сравнения логической и физической структур одного и того же ресурса Интернета показан на рис. 1.

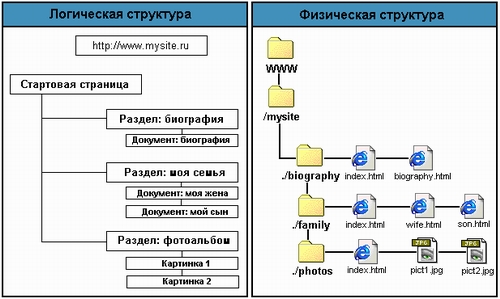


Рис. 1. Сравнение логической и физической структуры сайта

Очевидно, что логическая и физическая структуры могут не совпадать, поскольку в общем случае физическая структура ресурса разрабатывается, исходя из удобства размещения файлов. Однако более или менее точное сохранение порядка следования логических разделов в физической структуре сайта позволит вам избежать путаницы при последующем дополнении и обновлении материалов.

СОВЕТ Рекомендуется размещать все графические изображения, являющиеся элементами проекта, в отдельной папке с названием "Images", расположенной в корневой директории сайта. Такой подход позволит обновлять хранящиеся в других тематических разделах документы HTML без переноса графики, использовать одни и те же графические файлы во всех разделах сайта и при необходимости удалять целые директории.

Для того чтобы все гиперссылки на вашей домашней страничке или web-сайте работали корректно, все документы открывались правильно и броузер не выдавал ошибок при обращении к каким-либо разделам ресурса, при создании его физической структуры следует соблюдать несколько простых правил.

СОВЕТ Назначайте имена директорий, имена и расширения документов HTML и графических файлов с использованием символов только латинского алфавита и только в строчном регистре. Старайтесь, чтобы имена созданных вами файлов и директорий не превышали по длине восьми символов.

СОВЕТ При присвоении имен файлов документам HTML старайтесь следить за тем, чтобы эти имена были "смысловыми": впоследствии вы легко можете забыть содержимое и назначение какой-либо web-страницы, если имена файлов будут выглядеть, например, как l.htm, 2.htm, 3.htm и т. д.

Для того чтобы облегчить процесс обновления web-страниц, дополнения разделов или создания новых рубрик, заведите средство документирования проекта - любую электронную таблицу, созданную, например, в Microsoft Excel, или просто разграфленную тетрадку, в которую записывайте соответствие элементов физической структуры вашего проекта его логической структуре. До тех пор пока количество составляющих ваш сайт файлов относительно мало, это может показаться излишним, когда же оно перевалит за первые два десятка, в обилии html-документов и графических элементов будет легко запутаться, особенно если вы создаете несколько проектов одновременно. Пример оформления такого средства документирования показан в табл. 2.

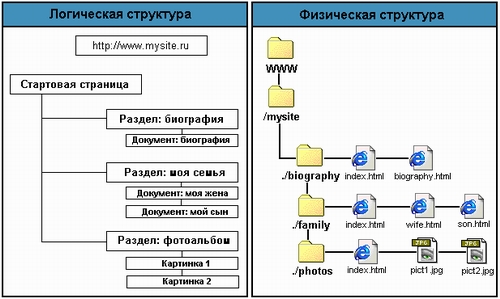


Таблица.2. Пример оформления средства документирования проекта

|  |  |  |  |
| --- | --- | --- | --- |
| Имя файла | Директория | Описание | Дата создания/ последнего изменения |
| index.html | /mysite | Стартовая страница сайта http://www.mysite.ru | 1.07.2000 |
| index.html | /mysite/family | Стартовая страница раздела <моя семья> | 5.07.2000 |
| wife.htm | /mysite/family | Рассказ о моей жене | 5.07.2000 |
| son.htm | /mysite/family | Рассказ о моем сынишке | 5.07.2000 |
| pid-jpg | /mysite/photos | Моя фотография на пляже в Сочи | 6.07.2000 |

Из всего сказанного становится очевидным, что физическая структура сайта скрыта от посетителей вашего ресурса: они могут наблюдать только логическую структуру, причем именно так, как она представлена при помощи элементов навигации. Отсюда следует вполне логический вывод: строение системы навигации должно если не полностью повторять, то хотя бы максимально соответствовать разработанной вами логической структуре сайта.

*Заглавная страница*

Один из критериев, руководствуясь которым можно разделить различные web-сайты на две основные категории, - это наличие заглавной страницы (splash) или отсутствие таковой.

Заглавная страница представляет собой html-документ, который не включает в себя какую-либо содержательную информацию и элементы навигации. Файлу заглавной страницы присваивается имя index.html, при этом стартовая страница называется иначе и вызывается посредством организации гиперссылки с заглавной страницы, загружающейся при обращении к сайту первой. Заглавная страница содержит, как правило, логотип компании-владельца данного ресурса, счетчик посещений и предложение выбора кодировки кириллицы, либо выбора между английской и русской версиями сайта. Пример web-ресурса, оснащенного такой страницей, показан на рис. 3.

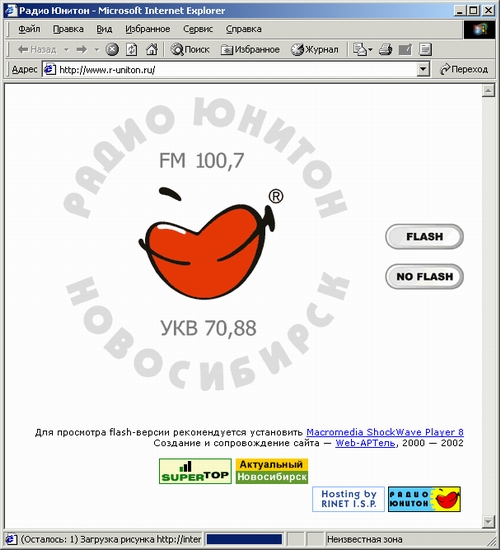


Рис. 3. Пример сайта, оснащенного заглавной страницей

При обращении к сайтам, не оснащенным заглавной страницей, первой отображается стартовый документ, включающий какое-либо информационное наполнение, элементы навигации и иногда анонсы составляющих данный ресурс тематических рубрик. На рис. 4 показан пример сайта, не оснащенного заглавной страницей.

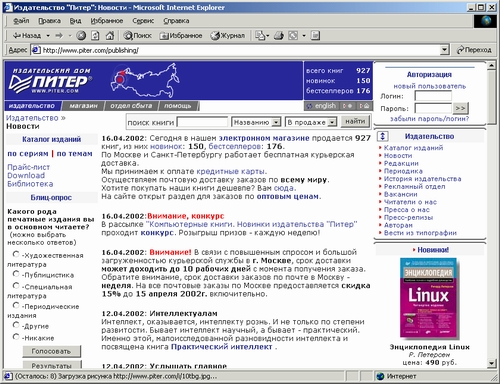


Рис. 4. Пример сайта, не оснащенного заглавной страницей

Использовать или не использовать заглавную страницу при создании собственного проекта в сети Интернет - дело вкуса каждого web-мастера. Дать какие-либо исчерпывающие рекомендации на этот счет трудно, поскольку окончательное решение зависит прежде всего от ваших художественных предпочтений ииногда - от желания заказчика, оплачивающего вашу работу.

**Верстка web страниц**

Веб-страница запускается на компьютере под управлением клиентской программы называемой браузером. Понятно, что операционная система, ее настройки и собственно сам браузер отличается от компьютера к компьютеру. Из чего напрашивается банальный вывод, что один и тот же документ сайта по-разному отображается у каждого пользователя.

Напрашивается вопрос, можно ли сделать так, чтобы веб-страница отображалась одинаково? Учитывая, сколько придется принять во внимание разных неоднозначных факторов, следует сказать, что нет, нельзя. Поэтому задача верстки веб-страниц формулируется так: сформировать документ, который бы корректно отображался с небольшими различиями на основных платформах и в браузерах. Корректно означает, что соблюден исходный замысел автора, воплощен требуемый дизайн документа, и он показывается в браузере без ошибок.

Чтобы реализовать указанную задачу надо понимать, как вообще происходит верстка веб-страниц и соизмерять свои идеи с их исполнением.

**Особенности верстки документов сайта**.

## Ширина документа

Изначально разработчику сайта ширина окна браузера пользователя неизвестна, поскольку она может меняться в самых широких пределах. Ширина зависит от разрешения монитора, длины его диагонали, размера окна и еще некоторых варьируемых данных. Иными словами, предугадать ее заранее простыми средствами не представляется возможным. С учетом этой особенности утвердилось два способа верстки: фиксированный и «резиновый».

### Фиксированный макет

В данном случае действуем от обратного и устанавливаем общую ширину макета жестко заданной и равной определенной величине. Если взять некоторую обобщенную статистику посетителей сайтов и посмотреть, какое разрешение монитора они преимущественно используют, то узнаем, что это 800 х 600 и 1024 х 768 пикселов. Получается, что ширина монитора пользователей в основном 800 и 1024 пиксела. Возьмем за ориентир 800 пикселов, тогда общая ширина макета за вычетом вертикальной полосы прокрутки и рамки браузера окажется 770–790 пикселов. На этот размер ориентируемся и устанавливаем ширину макета, например, 770 пикселов.

Преимущество такой схемы следующее. Раз общая ширина макета точно известна, то мы можем легко подгонять под нее дизайн и делать изображения уже известной ширины. В целом подобная верстка приближается к верстке печатного буклета, и в том и другом случае ширина носителя информации строго задана, за счет чего верстка хоть частично, но упрощается.

Недостаток, который инкриминируют этому виду верстки фактически один — недостаточно эффективное использование свободной площади. Действительно, для монитора с большо-о-о-й диагональю или высоким разрешением экрана документ будет смотреться по-другому, чем на предполагаемых 800 пикселах. На рис. 2.5 показано, как выглядит макет в таком случае.

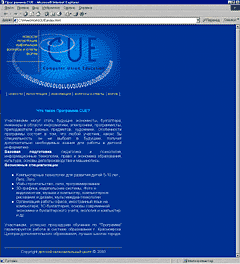


Рис. 5. Макет фиксированной ширины с размещением по левому краю

Справа появляется широкая пустая полоса, размер которой зависит от разрешения экрана пользователя и диагонали его монитора. Чтобы хоть как-то уменьшить пустое пространство, макет обычно помещают по центру окна браузера (рис. 6).

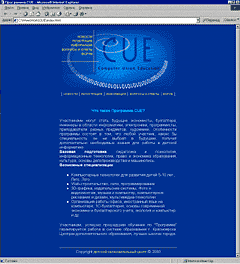


Рис. 6 Макет фиксированной ширины с размещением по центру«Резиновый» макет

Этот вид макета основывается на том, что в качестве одной из единиц измерения выступают проценты. Общая рабочая ширина окна браузера — 100%, и колонки макета в сумме не должны ее превышать, поэтому для удобства, как правило, везде применяют процентную запись. При изменении размеров окна происходит переформатирование данных страницы, чтобы они вписались в новую ширину (рис. 7).

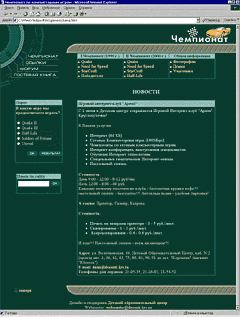


Рис. 7. Веб-страница занимает всю отведенную ширину

Этот вид верстки набирает все большую популярность и практически все известные сайты выбрали именно его в силу того, что эффективно задействуется вся площадь веб-страницы. Но следует сразу отметить некоторые особенности и недостатки присущие «резиновой» верстке.

* Хотя веб-страница и подстраивается под ширину окна браузера, при достижении некоторой величины читать текст становится неудобно — строки слишком длинные и глаза устают по ним бегать. Впрочем, браузер можно свернуть в окно, подобрав его комфортный размер.
* Верстать «резиновый» макет сложнее, чем аналогичный, но фиксированной ширины. Это связано с тем, что приходится учитывать множество дополнительных факторов и знать некоторые приемы верстки. К тому же, популярные браузеры неоднозначно трактуют некоторые параметры и в них «резиновый» макет может отображаться по-разному. Говоря проще, знать о правилах верстки надо больше.
* Любой макет имеет некоторую минимальную ширину, при достижении которой веб-страница «рассыпается» или появляется горизонтальная полоса прокрутки. Например, если в документ вставлен рисунок шириной 600 пикселов, то при уменьшении окна до этой величины браузер начнет отображать полосу прокрутки.
* «Резиновый» дизайн характеризуется активным использованием фоновых изображений, которые по горизонтали собираются без швов встык. Действительно, изменить без потери качества ширину рисунков не получится, а вот «подложить» под них фон — всегда пожалуйста.

## Высота документа

Исторически сложилось, что пролистывание большого документа на компьютере происходит сверху вниз. Для удобства листания предназначены вертикальные полосы прокрутки, клавиатурные комбинации, колесо прокрутки мыши. А вот перемещение по горизонтали происходит не так удобно, поэтому горизонтальной полосы прокрутки быть не должно. Из чего следует, что веб-страница должна вписываться в окно браузера по ширине, но никак не по высоте, которая может изменяться в очень широком диапазоне. На рис. 8 видно, что реальная высота документа не ограничена рамками браузера, но в окне показывается лишь часть страницы.

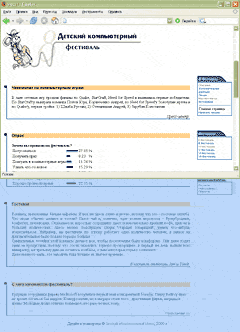


Рис. 8. Отображение документа в окне браузера по высоте

В принципе, чем больше на странице информации, и, следовательно, больше высота документа, тем сложнее находить нужные данные. Поэтому текст структурируют, разбивают на блоки и каждому из них дают свой заголовок, чтобы взгляду читателя было за что зацепиться.

Также следует учесть, что объем статей на сайте может достаточно сильно различаться между собой. При этом будет меняться и высота страницы, поэтому следует заранее побеспокоиться о том, чтобы макет отображался без ошибок, несмотря на различное значение высоты.

## Объекты веб-страницы прямоугольны

Все объекты на веб-странице имеют прямоугольную форму. Этот простой постулат не всегда согласуется с тем, что мы видим в действительности, поэтому следует сделать пояснения. В отличие от традиционной верстки (речь идет о полиграфических материалах), где в документ можно вставлять любые объекты, в том числе и векторные фигуры произвольной формы, верстка веб-страниц имеет ряд ограничений. К числу ограничений относится и то, что добавляемые объекты прямоугольны. Причем это относится к их форме, но не к содержимому, благодаря чему требуемый нам дизайн можно конструировать с помощью набора изображений. На рис. 6 показана картинка весьма неправильной формы. Но из-за того, что фон у этого рисунка совпадает с фоном веб-страницы, исходная прямоугольность изображения не видна.



Рис. 6. Изображение на рисунке может иметь произвольную форму

Однако стоит добавить вокруг рисунка рамку, как становится понятно, что изображение все-таки прямоугольно (рис.7). Например, если включить обтекание картинки текстом, то он будет обходить ее именно по границе рамки.



Рис. 7. Но сам рисунок остается прямоугольным

Данная особенность породила некоторые техники, связанные с версткой, которые перечислены далее.

### Активное использование рисунков

Рисунки не только применяются для иллюстрации текста, но и выполняют на сайте много различных ролей, например, используются для создания привлекательного дизайна, служат распоркой между ячейками таблицы, создают градиентные заливки, фоновые изображения и т.д.

### Разрезание изображения на фрагменты

Один рисунок может занимать слишком большую исходную площадь. Разрезав его на прямоугольники, получим замечательный конструктор, в котором одни фрагменты изображения допускается заменять текстом, другие анимацией, а третьи модифицировать «на лету». Таким образом, имеем в своем распоряжении средство для обхода прямоугольной природы изображений, ведь в «склеенном» рисунке может недоставать некоторых фрагментов, уголков, например.

### Применение фонового рисунка

Фоновый рисунок удобен тем, что он может заполнять всю отведенную ширину или высоту под блоком. Это позволяет создавать линии или другие декоративные элементы, которые привязываются к ширине или высоте текста и не зависят от размеров окна. К тому же поверх фона можно накладывать текст, что также расширяет возможности по дизайну веб-страниц.

### Картинки вместо текста

Если средства верстки имеют определенные ограничения, то почему бы не создать текст в графическом редакторе и не вставить его в качестве картинки или Flash? Это гарантирует, что текст сохранит свой вид и начертание несмотря ни на какие внешние условия. Однако здесь имеется и оборотная сторона — рисунки занимают больший объем, чем рядовой текст, их сложнее править, они не индексируются поисковыми машинами, их показ пользователи могут отключить. В общем, недостатков масса, поэтому изображения на сайте хотя и применяют вместо текста, но достаточно ограниченно. Например, для заголовка сайта.

**Модульная сетка**

Модульная сетка представляет собой набор невидимых направляющих, вдоль которых располагаются элементы веб-страницы. Это облегчает размещение данных в документе, обеспечивает визуальную связь между отдельными блоками и сохраняет преемственность дизайна при переходе от одной страницы к другой.

Веб-страница фактически рассматривается как набор прямоугольных блоков, которые выкладываются в определенном порядке. При этом, как правило, данные располагаются по колонкам, поэтому при верстке применяют термин одно-, двух-, трехколонный макет и т.д. Для примера рассмотрим главную страницу сайта deviantart.com (рис. 8).

Каждый блок этой страницы четко отделен от других с помощью пустого пространства, рамки или разделителя, в качестве которого выступает цветной прямоугольник с текстом заголовка. Напрашивается только вопрос, действительно ли мы имеем дело с тремя колонками? В некоторых случаях сразу определить, сколько колонок содержит макет и впрямь затруднительно. В таком случае следует понимать, что колонки могут объединяться, а также содержать не только сплошной текст, но и графические вставки. Если представить основные блоки страницы в виде однотонных прямоугольников, то получим наглядную модульную сетку, по которой сразу становится понятно, как сверстан документ (рис. 9).

По данному рисунку видно, что верхний блок с названием сайта, формой поиска и кнопками навигации занимает всю ширину страницы. Далее идут три колонки, причем первые две колонки предварительно объединены в одну для удобного представления необходимой информации. Завершает макет блок с контактной и правовой информацией.

Перейдем к принципам построения модульной сетки. Вначале макет веб-страницы разрабатывают на листе бумаги. Это позволяет, не тратя зря времени, быстро сделать серию набросков и уже из них выбрать подходящий эскиз. Сами посудите, сколько вариантов можно создать за десять минут в графическом редакторе и сколько за это же время с помощью карандаша и бумаги? При этом не имеет значения степень владения программой, все равно на бумаге выйдет быстрее. Тем более что набросок может быть и корявым, главное чтобы автор сам понял, что он нарисовал. Обычно вместо текста и рисунков применяют схематические значки. Например, текст обозначается несколькими горизонтальными линиями (рис. 10), а рисунки изображаются затемненными блоками или перечеркнутыми прямоугольниками (рис. 11).



Рис. 8. Главная страница deviantart.com

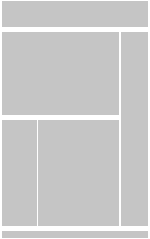


Рис. 9. Модульная сетка для главной страницы deviantart.com

http://www.htmlbook.ru/images/079_3.gif

Рис. 10. Обозначение текста в макетах

http://www.htmlbook.ru/images/079_4.gif

Рис. 11. Обозначение изображений в макетах

Далее эти обозначения будем применять при рассмотрении наиболее распространенных модульных сеток.

## Одноколонная сетка

Текст в одну колонку чаще всего встречается в академическом дизайне, при фиксированном макете и публикации большого текста.

Замечание

Академический дизайн характеризуется минимализмом оформления и даже аскетизмом. Основной упор делается на содержательную часть, а дизайну как таковому практически не уделяется внимания. Преимущественно академический дизайн встречается в научной среде.

На рис. 12 показана типичная схема одноколонной модульной сетки. Как правило, наблюдается четыре основных блока — заголовок страницы, набор ссылок на другие страницы сайта (навигация), собственно сам текст и в самом низу контактная информация. Если высота страницы достаточно велика, то блок навигации дублируют внизу или делают ссылку «Наверх», которая перемещает к началу документа.

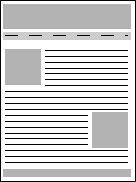


Рис.12. Одноколонная модульная сетка

Иллюстрации в тексте встречаются по ходу, при этом текст их обычно обтекает по контуру. При активном применении изображений на сайте удобнее воспользоваться фиксированным макетом, ширина которого точно известна. Тогда рисунки можно готовить заданного размера, которые точно впишутся в макет страницы фиксированной ширины и практически полностью состоит из набора рисунков.

Модульная сетка для этого сайта представлена на рис. 13. Вначале идет заголовок сайта с формой поиска, ниже следует навигация, а отдельные фотографии представляют собой саморекламу со ссылками на соответствующие разделы сайта. В самом низу расположены ссылки на информацию о сайте.

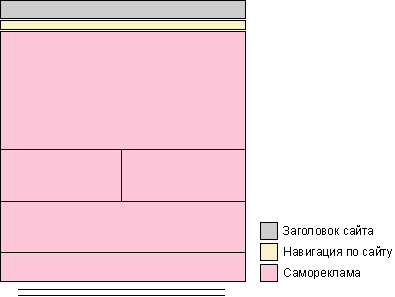


Рис. 13. Модульная сетка для главной страницы сайта victoriassecret.com

## Двухколонная сетка

Это один из самых распространенных вариантов при использовании на сайтах. При такой модульной сетке используется две колонки — одна отводится под основной текст, а вторая используется для навигации и другой полезной информации (рис. 14).

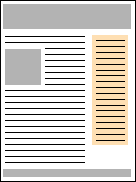


Рис. 14. Двухколонная модульная сетка

Принципиального значения не имеет, слева или справа располагается колонка с навигацией, встречается и тот и другой вариант.

В каком-то смысле двухколонная сетка стала стандартом де-факто для информационных сайтов из-за своего удобства. Действительно, все «под рукой» — и текст и ссылки, к тому же данная сетка не исключает подключения горизонтальной навигации, как это принято в одноколонной сетке.

Двухколонные сетки удобны при создании самых разнообразных сайтов и не требуют особых знаний по верстке веб-страниц. Единственный недостаток, который им вменяют, что подобные сайты выглядят достаточно однообразно. Но с другой стороны пользователям удобнее работать с сайтом привычного вида, без лишних «наворотов».

## Трехколонная сетка

Такие сетки часто применяются на главных страницах сайтов, где одновременно требуется показать пользователю множество возможностей, которые он обнаружит на данном сайте. Также трехколонная сетка используется и на внутренних страницах, если для размещения различной информации двух колонок уже не хватает (рис. 15).

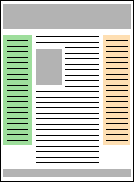


Рис. 15. Трехколонная модульная сетка

Одна из колонок отдается под навигацию, вторая, самая широкая — под основной текст, а в третью колонку добавляют рекламу, ссылки, текст и т.д.

Трехколонная сетка обеспечивает больше простора для дизайна, ведь в некоторых местах можно объединять колонки, разбивать материал на отдельные фрагменты и визуально отделять один блок от другого. Макет при этом может получиться достаточно сложным, но результат обычно того стоит. Возвращаясь к главной странице сайта deviantart.com, приведем более детальную модульную сетку (рис. 16). Отдельные блоки выделены разным цветом.

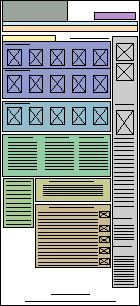


Рис. 16. Модульная сетка для главной страницы deviantart.com

На сайте deviantart.com применяется три колонки, две из них часто объединяются для получения более широкой области. Это оправданно, поскольку требуется разместить 5 фотографий или три колонки с текстом.

К недостаткам трехколонной сетки относится достаточная сложность верстки макета. Чтобы получить нужный результат приходится иной раз затратить много времени на создание стилевого файла и отладку документа в разных браузерах.

 Конечно, существуют и другие виды модульных сеток, например, многоколонные. Однако они практикуются достаточно редко из-за ограниченной ширины окна браузера. При таком раскладе текст приходится делать мелким или применять другие способы, чтобы не возникло горизонтальной полосы прокрутки. От этого может пострадать удобство восприятия информации пользователем.

Модульная сетка не является единственным верным вариантом при верстке веб-страниц. Можно воспользоваться перспективой, хаотичностью или другой структурой, главное, чтобы она была эстетичной и привлекательной. Отказ от правил верстки характерен для дизайн-студий, задача которых состоит в том, чтобы поразить своим сайтом воображение посетителей и привлечь к себе их внимание. Но даже они делают для заказчика сайты на основе формальных сеток, поскольку так проще добавлять информацию и вносить разнообразие в дизайн макета.

Модульные сетки позволяют упростить верстку сайта, поскольку все материалы разбиваются на отдельные блоки, которые выравниваются по невидимым направляющим линиям. Такие блоки хотя и взаимосвязаны друг с другом, но обычно дозволяют независимое редактирование данных, что упрощает оформление элементов.

Прежде чем переходить к созданию веб-страницы в HTML-редакторе, сделайте ее набросок на листе бумаги. Это позволит сократить время на выбор модульной сетки, которая будет применяться в дальнейшем. В зависимости от целей сайта и количества материала применяют одно-, двух, трех или многоколонную модульную сетку. Чем больше колонок применяется, тем шире возможности по управлению видом дизайна, но за это приходится платить сложностью верстки документа. Не стоит также забывать, что наиболее популярными на сайтах являются двух и трехколоночные сетки.

**Принципы компоновки и алгоритм создания web-сайта**

**Принципы компоновки web-сайта**

Современные видеокарты поддерживают несколько произвольно устанавливаемых видеорежимов, характеризующихся экранным разрешением и количеством цветов, используемых при отображении информации на экране компьютера. С помощью встроенных функций операционной системы пользователь может установить несколько стандартных значений экранного разрешения, например 640x480, 800x600, 1024x768, 1152x864, 1280x1024 или 1600x1200 точек. Разумеется, при открытии в одном и том же браузере какой-либо web-страницы она может отображаться не одинаковым образом в зависимости от используемого посетителем сайта экранного разрешения.

Для того чтобы избежать "съезжания" элементов html-документа друг относительно друга и, как следствие, деформации web-страницы в целом при изменении параметров экрана, применяется достаточно простой и действенный прием: все компоненты web-страницы заключаются в соответствующие ячейки невидимой таблицы, при этом каждому объекту назначается одно, строго определенное положение. Таким образом, появляется второй критерий, по которому можно разделить все существующие web-сайты на две условные категории. Данной таблице можно назначить строго определенную ширину в пикселях, например, 640 точек, после чего жестко позиционировать ее по центру экрана или "прижать" к левому его краю.

*Для построения сложных таблиц можно применять вложение одной таблицы в другую: например, первичная таблица шириной 640 точек задает горизонтальное разбиение страницы, в каждую из ее ячеек вкладывается еще одна, равная ей по ширине невидимая таблица, обеспечивающая вертикальное разбиение. Такой подход позволяет дробить различные участки web-страницы независимо друг от друга, создавая, к примеру, в одном ее поле четыре вертикальные колонки различной ширины, а в другом — две, оперируя при этом различными параметрами вложенных таблиц в пределах одного и того же документа — размером пространства между колонками, толщиной границ и т. д*.

Такой вариант компоновки сайта можно назвать статическим, поскольку ширина таблицы не меняется в зависимости от экранного разрешения. Разумеется, при изменении параметров экрана не происходит ни малейшего смещения элементов дизайна страницы.

Иной подход - когда ширину невидимой таблицы, содержащей фрагменты web-страницы, задают в процентах от текущей ширины экрана. При увеличении экранного разрешения таблица "растягивается" по горизонтали, и все размещенные в ее ячейках элементы, позиционированные либо по центру, либо по краям столбцов, смещаются согласно установленному алгоритму. В силу того, что параметры таблицы изменяются в зависимости от настроек экрана, такой принцип компоновки html-документа можно назвать динамическим. И тот и другой подход обладает как достоинствами, так и недостатками, которые перечислены ниже.

**Статическая компоновка страницы**

Достоинства. Простота алгоритма верстки документа. Оптимизировав страницу для отображения с экранным разрешением 640x480 точек, вы можете быть уверены, что при изменении пользовательских экранных настроек элементы дизайна не "поплывут". Кроме того, данный вариант компоновки сайта в большинстве случаев (при соблюдении ряда дополнительных условий) позволяет добиться идентичности отображения html-документа в броузерах MicrosoftInternet Explorer и Netscape Navigator.

Недостатки. При отображении документа на компьютере с высоким экранным разрешением по краям экрана или с правой его стороны остается заметное пустое поле.

**Динамическая компоновка страницы**

Достоинства. Документ растягивается по всей ширине экрана, не остается не использованных пустых полей.

Недостатки. Сложность верстки и отладки страницы, весьма часто проявляется неадекватность отображения таких документов в броузерах Microsoft Internet Explorer и Netscape Navigator.

ПРИМЕЧАНИЕ Используемые в командах HTML значения пикселов (условных точек)подразумевают физические размеры точки экрана пользователя и зависят от типа его монитора.

Примеры реализации сайтов со статической и динамической компоновкой страниц приведены на рис. 3.1 и 3.2. Данные изображения сделаны с фиксированным экранным разрешением 800x600 точек.

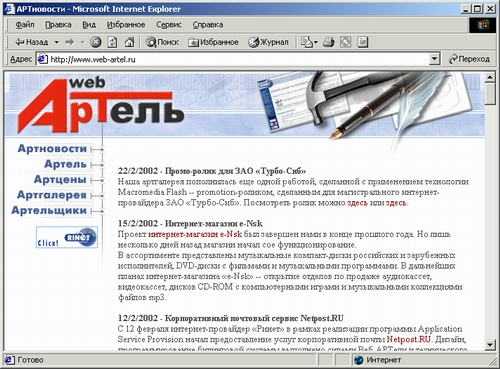


Рис. 3.1. Пример сайта со статической компоновкой страниц



Рис. 3.2. Пример сайта с динамической компоновкой страниц

***Элементы web-страницы***

Любая web-страница содержит определенный набор стандартных элементов, являющихся обязательными компонентами каждого ресурса Интернета. Безусловно, ассортимент и количество подобных объектов могут варьироваться в зависимости от тематической направленности сайта, объема опубликованных на нем материалов, а также от целей и задач, которые ставит перед собой создатель данного ресурса. Компоновка таких элементов, проектирование их взаимного расположения и составляет одну из главных задач web-мастера.

Первым элементом web-страницы, который нам предстоит рассмотреть, является ее заголовок. Он может быть выполнен как в текстовом, так и в графическом варианте, однако и в том и в другом случае он должен располагаться в верхней части документа. Иногда с заголовком совмещают меню выбора кодировки кириллицы и кнопки для перехода с русскоязычной на англоязычную версию сайта, если данный web-ресурс представлен на двух языках. Непосредственно подзаголовком документа, как правило, располагается пространство, отведенное для размещения рекламного баннера. Включение баннера именно в верхнюю часть web-страницы в большинстве случаев является обязательным условием регистрации сайта в службах баннерного обмена - системах, рекламирующих созданный вами ресурс в обмен на показ на страницах вашего сайта рекламы других участников баннерообменной сети. Стандартный размер баннеров, публикуемых под заголовком документа, составляет обычно 468x60 точек. Если вы используете статический принцип компоновки страницы, ширина заголовка вашего документа будет составлять приблизительно 640 пикселов: это значение обусловлено, прежде всего, необходимостью обеспечить корректное отображение документа на мониторах с экранным разрешением 640x480 точек и избежать появления горизонтально полосы прокрутки, затрудняющей его просмотр. Очевидно, что ширина баннера в этом случае будет значительно меньше ширины заголовка, благодаря чему в той части страницы, где вы планируете отвести место под рекламу, образуется незанятое пространство, которое можно заполнить логотипом компании-владельца данного сайта или ссылкой на сервер, осуществляющий web-хостинг. Разумеется, логотип необходим далеко не всегда: как правило, он включается в состав web-страницы лишь в случае, если сайт имеет коммерческую направленность.

Основную часть документа занимает так называемое текстовое поле - участок, где и размещается смысловое наполнение страницы: содержательный информационный текст и иллюстрации. Перечисленные элементы еще называют "контент" (от англ, content - содержание). Расположение текстового поля зависит в первую очередь от того, каким образом web-дизайнер разместит остальные элементы документа.

Следующей обязательной составляющей частью web-страницы являются элементы навигации - гиперссылки, связывающие данный документ с другими разделами сайта. Элементы навигации могут быть выполнены в виде текстовых строк, графических объектов, то есть кнопок, либо активных компонентов, например Java-апплетов. Последние представляют собой те же кнопки, которые, в отличие от своих "традиционных" сестер, умеют реагировать на движения мыши, выполняя при наведении на них курсора какие-либо несложные действия (включение подсветки, создание эффекта "нажатия", изменение формы и т. д.). Как я уже упоминал в разделе "Основные "постулаты" web-дизайна", располагать элементы навигации следует таким образом, чтобы они всегда были "на виду", "под рукой", то есть так, чтобы пользователю не приходилось "отматывать" страницу назад, если текстовое поле занимает по высоте несколько физических экранов, после чего подолгу искать ссылки на другие разделы. Наиболее устоявшимся подходом является размещение элементов навигации у левой границы страницы.

В нижней части документа принято публиковать информацию о разработчиках сайта и адрес электронной почты, по которому посетители ресурса могут направить владельцам странички свои отклики, предложения и пожелания. Если web-страница является стартовым документом, в нижней ее части также размещают счетчик посещений - небольшой сценарий, вызывающий установленный на сервере CGI-скрипт, который фиксирует каждое открытие документа в броузере пользователей, изменяя значение индикатора счетчика. Благодаря этому web-мастер без труда определит количество посетителей, навестивших его страничку в течение какого-либо времени. Отмечу, что счетчик посещений устанавливается только на первой странице, вызываемой при обращении к сайту, в остальных документах ресурса он отсутствует. Не рекомендуется также размещать на одной странице несколько разных счетчиков.

Итак, мы разобрали все основные компоненты web-страницы и их возможное расположение относительно друг друга. Пример компоновки сайта, содержащего полный набор описанных выше составляющих, показан на рис. 3.3. В нем выбрано позиционирование элементов навигации по левой границе документа.



Рис. 3.3. Пример компоновки web-страницы с левым позиционированием элементов навигации

На практике часто встречаются web-сайты, в дизайне которых элементы навигации позиционированы по правой границе экрана. В этом случае текстовое поле смещается налево, остальные компоненты документа располагаются, исходя из принципа максимальной эстетичности их сочетания. Пример такого исполнения сайта показан на рис. 3.4.



Рис. 3.4. Пример компоновки web-страницы с правым позиционированием элементов навигации

Как видно из рисунка, логотип в этом случае помещен на один уровень с заголовком документа, а рекламный баннер позиционирован относительно центра страницы. При таком подходе рекомендуется выдерживать графическое оформление заголовка, логотипа и поля для размещения рекламы в едином цветовом и художественном стиле - тогда несимметричность положения данных объектов будет не столь очевидна и не станет "резать глаз" сторонникам строгой, "табличной" эстетики дизайна.

Элементы навигации можно разместить не только вблизи правой и левой границ страницы, но и в верхней части документа. Такой вариант компоновки наиболее подходит, на мой взгляд, при создании домашних страничек: в этом случае все объекты страницы гармонично "вписываются" в заданную ширину невидимой таблицы, при этом подготовка самой таблицы значительно упрощается. Единственным недостатком подобного подхода является необходимость продублировать элементы навигации в нижней части документа, поскольку при вертикальной прокрутке страницы они исчезают за верхней границей экрана, и, чтобы добраться до них, пользователю придется "отматывать" экран назад, что, согласитесь, весьма неудобно. Пример оформления страницы с верхним расположением элементов навигации показан на рис. 3.5.



Рис. 3.5. Пример компоновки web-страницы с верхним позиционированием элементов навигации

Безусловно, все, что было сказано в данном разделе, является не панацеей, а руководством к действию. Я пытаюсь изложить лишь общие принципы, которые применяются при компоновке структуры сайта, окончательное же решение всегда остается за web-мастером. В конечном счете, какой бы дизайн вы не заложили в основу своего будущего проекта, результат ваших трудов все равно будет правильным: в Интернете нет ни цензуры, ни каких-либо регламентов, загоняющих создателя сайта в те или иные жесткие рамки. Примером дизайнерского решения, не попадающим ни в одну из указанных выше категорий, может служить так называемая смешанная компоновка, примерная схема которой приведена на рис. 3.6.

Как видно из рисунка, в данном примере часть управляющих элементов встроена непосредственно в заголовок страницы - речь идет о кнопках переключения между русской и английской версиями сайта, а также о кнопках навигации: это может быть ссылка на адрес электронной почты создателей ресурса, продублированная в нижней части документа, и ссылка на один из тематических разделов, например, страницу новостей. Основной блок элементов навигации позиционирован относительно левой границы документа, однако меню выбора кодировки кириллицы расположено непосредственно под рекламным баннером в верхней части страницы. Текстовое поле разделено на две несимметричные колонки, причем в правой размещены краткие анонсы составляющих ресурс тематических рубрик, включая ссылки на эти разделы.

Очевидно, что вариантов "смешанной компоновки" web-страницы может быть великое множество: конкретные решения зависят от количества составляющих ресурс разделов, объема подготовленного для размещения на сайте текста и, наконец, от фантазии самого дизайнера. Важно лишь, чтобы внешний вид сайта не вызывал нареканий у посетителей. В конце концов, только вы как разработчик, вы и никто другой, вправе проявить все свои способности и таланты и скомпоновать страницу по собственному вкусу. Создатели некоторых домашних страничек, не мучаясь сомнениями, размещают счетчик посещений в верхнем правом углу документа, название сайта пишут мелким убористым шрифтом и публикуют его под рекламным баннером, а элементы навигации почему-то неожиданно обнаруживаются прямо в середине текстового блока, между рассказом о себе и фотографиями любимой собаки автора проекта. На вкус и цвет, как говорится, товарищей нет. Но лично мне эта болезнь кажется неизлечимой.



Рис. 9

**Регистрация сайта в сети**











**Лекция 7. Основы HTML**

**План лекции:**

Введение в HTML

Структура HTML-документа

Секция DOCTYPE

Теги управления форматированием: Абзац, Заголовки, Начертание, Маркированный и Нумерованные списки, Таблицы, Ссылки, Рисунки, Карты изображения, Фреймы

HTML — простой инструмент для создания гипертекстовых документов. Правильно оформленный HTML-документ одинаково отображается на разных аппаратных и программных платформах.

**Введение в HTML**

**HTML** (HyperText Markup Language — «язык гипертекстовой разметки») — стандартный язык разметки документов во Всемирной паутине

**HTML -** это язык разметки, который представляет простые правила оформления и компактный набор структурных и семантических элементов разметки (тегов). HTML позволяет описывать способ представления логических частей документа (заголовки, абзацы, списки и т.д.) и создавать веб-страницы разной сложности.

Изначально язык HTML (*HyperText Markup Language*) был задуман и создан как средство структурирования и форматирования документов без привязки к средствам отображения. В идеале, гипертекстовый документ должен одинаково выглядеть на различных устройствах (монитор ПЭВМ, экран ПДА или мобильного телефона, принтер, медиа-проектор и т.п.).

Разработкой спецификаций языка HTML и утверждением их в качестве официальных стандартов занимается Консорциум всемирной паутины (W3C). Помимо W3C, в развитии языка участвуют IT-компании и сообщество разработчиков.

Версии языка HTML

•HTML 0.9

• HTML 2.0, одобренный как стандарт 22 сентября 1995 года;

• HTML 3.2— 14 января 1997 года;

• HTML 4.0 — 18 декабря 1997 года;

• HTML 4.01 — 24 декабря 1999 года;

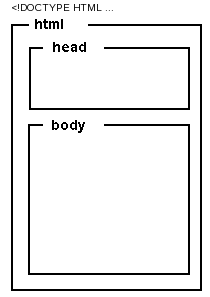
• HTML 5— 28 октября 2014 года

• HTML 5.1 начал разрабатываться примерно 19 декабря 2012 года.

Для создания web-страниц нам понадобиться обычный текстовый редактор, например, Блокнот или NotePad++. Отличие в том, что создаваемым файлам дается расширение html.HTML не является языком программирования, но веб-страницы могут содержать встроенные или загружаемые программы на скриптовых языках (в первую очередь *Javascript*) и программы-апплеты на языке *Java*.

**Структура HTML-документа**

HTML-документ состоит из текста, который представляет собой информационное содержимое и специальных средств языка HTML — тегов разметки, которые определяют структуру и внешний вид документа при его отображении браузером. Структура HTML-документа (рис. 1) довольно проста:

****

*Рис. 1. Общая структура веб-страницы*

1. Описание документа начинается с указания его типа (секция DOCTYPE).

Наличие секции DOCTYPE позволяет указать браузеру, какой тип документа ему предстоит разбирать, т.е, какие требования нужно выполнять при обработке гипертекста.

Примеры DOCTYPE:

* <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">  
  Гипертекстовый документ в формате HTML 4.01, содержащий фреймы.
* <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  
  Гипертекстовый документ в формате HTML 4.01 со строгим синтаксисом (т.е. не использованы устаревшие и не рекомендованные теги).
* <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
  Гипертекстовый документ в формате HTML 4.01 с нестрогим («переходным») синтаксисом (т.е. использованы устаревшие или не рекомендованные теги и атрибуты).
* <!DOCTYPE HTML>

Пока не стандартизованное объявление для документов HTML5.

Стандарт требует, чтобы секция DOCTYPE присутствовала в документе, т.к. это позволяет ускорить и улучшить обработку гипертекста. Это достигается за счет того, что браузер может не делать предположений о том, как интерпретировать теги, а свериться со стандартным определением (файлом .dtd).

1. Текст документа заключается втег<html>. Текст документа состоит из заголовка и тела, которые выделяются соответственно тегами <head> и <body>.
   * В заголовке (<head>) указывают название HTML-документа и другие параметры, которые браузер будет использовать при отображении документа.
   * Тело документа (<body>) — это та часть, в которую помещается собственно содержимое HTML-документа. Тело включает предназначенный для отображения текст и управляющую разметку документа (теги), которые используются браузером.

Заголовок предназначен для размещения метаинформации, описывающей веб-документ как таковой.

Блок <body> содержит то, что нужно показать пользователю: текст, изображения, внедренные объекты и пр.

**html** — элемент верхнего уровня — ограничивает начало и конец документа. Элемент html является элементом верхнего уровня в языке html. Использование элемента html не сказывается на выводе документа. Если элемент html используется, начальный и конечный теги html должны идти вокруг всего документа, сразу после объявления DOCTYPE.

<html>

заголовок и тело документа

</html>

head (заголовок) — заголовок документа

Основная структура html-документа должна состоять из заглавия и тела. Явное вложение заголовка в элемент head не обязательно. Использование элемента head не сказывается на вывод документа. Атрибуты отсутствуют. В документе должен присутствовать только один элемент head, и он должен появиться до элемента body.

Синтаксис использования

<head>

элемент title

</head>

Заметим, что начальный и конечный теги могут отсутствовать.

Тег <head>должен содержать один TITLE элемент и возможно следующие элементы:

•isindex;

•meta;

•link;

•style;

•script.

**body** (тело) — тело документа. Основная структура документа HTML всегда состоит из заголовка и тела. Нет необходимости явно помещать тело в элемент BODY, однако делая так, можно специфицировать атрибуты, влияющие на представление документа в целом (например, установить фоновое изображение или цвет).

Если элемент body не содержит атрибуты, использование его не дает явного эффекта в непосредственном отображении документа. Элемент html может быть задан явно или подразумеваться. только один элемент body разрешен в документе, и он должен находится после элементаhead (который тоже может быть, как подразумеваемым, так и явным).

Основной синтаксис

<body>тело документа</body>

*Возможные атрибуты тега BODY*

**Имя атрибута Возможные значения Смысл**

BGCOLOR Цвет Фоновый цвет документа

TEXT Цвет Цвет для текста документа

LINK Цвет Цвет для непосещенной гипертекстовой связи

VLINK Цвет Цвет для посещенной гипертекстовой связи

ALINK Цвет Цвет для активной гипертекстовой связи; используется для выделения текста , когда пользователь нажал на связь

BACKGROUND URL URL фонового образа

**Теги управления форматированием**

**Абзац** формируется при помощи тега <p>…</p>. Изначально предполагалось использовать вместе с тегами параметры. Параметры тега <p> - способы выравнивания (left, right, center, justify). Например, если необходимо сделать абзац с выравниванием по ширине, то надо написать:

<palign=”justify”>Содержаниеабзаца</p>

**Заголовки** предназначены, во-первых, показать важность раздела, к которому относятся, а во-вторых, с помощью различных заголовков легко регулировать размер текста. Чем выше уровень заголовка, тем больше размер шрифта. Самым верхним уровнем является уровень 1 <h1>, а самым нижним — уровень 6 (<h6>).

Также следует отметить, что использование заголовков приветствуется поисковыми системами. Пример использования заголовков.

Пример использования заголовков.

<body>

<h1>Заголовок первого уровня</h1>

<h2>Заголовок второго уровня</h2>

<h3>Заголовок третьего уровня</h3>

<h4>Заголовок четвертого уровня</h4>

<h5>Заголовок пятого уровня</h5>

<h6>Заголовок шестого уровня</h6>

</body>

Для **полужирного начертания** применяется два тега: <b>, <strong>.

<b>Жирное начертание шрифта</b>

<strong>Сильное выделение текста</strong>

Для *курсивного начертания* используются два тега: <i>,<em>.

<i>Курсивное начертание шрифта</i>

<em>Выделение текста</em>

Следует отметить, что теги <b> и <strong>, также как <i> и <em> хотя и похожи по своему действию, являются не совсем эквивалентными и заменяемыми. Первый тег <b> - является тегом физической разметки и устанавливает жирное начертание текста, а тег <strong> - тегом логической разметки и выделяет помеченный текст.

**Маркированный список** формируется с помощью контейнера <ul>, а каждый пункт списка начинается с тега <li>, как показано ниже.

<ul>

<li>Значение №1</li>

<li>Значение №2</li>

<li>Значение №3</li>

</ul>

В списке непременно должен присутствовать закрывающий тег </ul>, иначе возникнет ошибка. Закрывающий тег </li> хотя и не обязателен, но советуем всегда его добавлять, чтобы четко разделять элементы списка.

**Нумерованные списки** формируются так:

<ol>

<li>Значение №1</li>

<li>Значение №2</li>

<li>Значение №3</li>

</ol>

Бегущая строка задается тегом <marquee>.

**Таблицы**

Таблицы могут использоваться не только для более наглядного предоставления информации, но и для верстки веб-страниц. Таблица с невидимой границей представляет собой словно модульную сетку, в блоках которой удобно размещать элементы веб-страницы. Однако в настоящее время верстка страниц в основном осуществляется с помощью слоев. Рассмотрим пример добавления таблицы, показанной ниже.

|  |  |
| --- | --- |
| 1 | 2 |
| 3 | 4 |

Для этого надо написать следующий код:

<table border="1">

<tr>

<td>1</td>

<td>2</td>

</tr>

<tr>

<td>3</td>

<td>4</td>

</tr>

</table>

Для объединения двух и более ячеек в одну используются атрибуты colspan и rowspan тега <td>.

Рассмотрим пример добавления таблицы с объединенными ячейками как показано ниже.

|  |  |
| --- | --- |
| 1 | |
| 3 | 4 |

Для этого надо написать следующий код:

<table border="1">

<tr>

<td colspan=2>1</td>

</tr>

<tr>

<td>3</td>

<td>4</td>

</tr>

</table>

**Ссылки**

Ссылки являются основой гипертекстовых документов и позволяют переходить с одной веб-страницы на другую. Особенность их состоит в том, что сама ссылка может вести не только на HTML-файлы, но и на файл любого типа, причем этот файл может размещаться совсем на другом сайте Общий синтаксис создания ссылок следующий:

<a href="URL">ключевая фраза</a>

Атрибут **href** определяет URL (Universal Resource Locator, универсальный указатель ресурса). Текст, расположенный между тегами <a> и </a>, по умолчанию становится синего цвета и подчеркивается.

Адрес ссылки может быть абсолютным или относительным. Абсолютные адреса должны начинаться с указания протокола (обычно http://) и содержать имя сайта. Относительные ссылки ведут отсчет от корня сайта или текущего документа.

С тегом <a> можно использовать параметры target и title.

**target** указывает окно, в котором нужно открыть ссылку. По умолчанию, при переходе по ссылке документ открывается в текущем окне или фрейме. Синтаксис следующий:

<a target="имя окна">…</a>

В качестве значения используется имя окна или фрейма, заданное атрибутом name. Если установлено несуществующее имя, то будет открыто новое окно. В качестве зарезервированных имен применяются следующие.

• \_blank — загружает страницу в новое окно браузера.

• \_self — загружает страницу в текущее.

• \_parent — загружает страницу во фрейм-родитель, если фреймов нет, то это значение работает как \_self.

• \_top — отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то это значение работает как \_self.

Параметр **title** добавляет поясняющий текст к ссылке в виде всплывающей подсказки, которая отображается при наведении мыши на ссылку. Например,

<a href=”http://enu.kz” title=”ЕНУ”>Университет</a>

Для создания ссылки на адрес электронной почты в параметре href надо указать mailto:адрес электронной почты. Например,

<ahref=”mailto:admin@enu.kz”>администратор</a>

Кроме внешних ссылок можно использовать внутренние ссылки. Для этого вначале необходимо сделать метку в соответствующем месте страницы и дать ей имя при помощи атрибута name тега <a>. Для создания метки используется символ решетки (#).

Пример. Создадим в конце web-страницы ссылку, которая введет в начало документа. Для этого в начале документа, задаем метку:

<body>

<a name="ttt"></a>

Затем в конце страницы задаем ссылку:

< a href="#ttt"> В начало </a>

Также можно сделать ссылку на метку другого документа. Например,

< a href="text.html#ttt"> Как написано на сайте…</a>

**Рисунки**

Для добавления изображения используется тег <img>, атрибут src которого определяет адрес графического файла. Общий синтаксис добавления изображения будет следующий.

<img src="URL рисунка">

Полезными являются атрибуты alt, width, height. Атрибут alt предназначен для создания альтернативного текста. При помощи атрибутов width и height задается размер рисунка. Следует отметить, что заданный размер не влияет на скорость загрузки.

**Карты изображения**

Карты изображения позволяют привязывать ссылки к разным областям одного изображения. По сути это графические изображения с размеченными областями. Для создания карты изображения нужно дать команду

<img src="map.gif" usemap="#map1">

Далее необходимо определить области на карте следующим образом:

<map name="map1">

< area shape="форма" coords="список координат" href="ссылка">

…

</map>

Возможны следующие варианты форм:

**Форма Название Задаваемые координаты**

rect прямоугольник координаты двух вершин

poly многоугольник координаты всех вершин

circle окружность координаты центра и радиус

Например,

<area shape="poly" coords="282,2, 222,1, 203,6, 210,27, 212,33, 282,31, 301,33, 301,28" href="1.html">

**Фреймы**

Окно браузера может быть разделено на фреймы, т.е. на самостоятельные области. В каждую из этих областей можно загружать свои html страницы. Рассмотрим следующий пример (строки для удобства комментирования пронумерованы):

1) <HTML>

2) <HEAD>

3) <TITLE>ФРЕЙМЫ</TITLE>

4) </HEAD >

5) <FRAMESET ROWS="45%,45%,\*">

6) <FRAME SRC="B611.HTM">

7) <FRAME SRC="B612.HTM">

8) <FRAME SRC="B613.HTM">

9) </FRAMESET >

10) </HTML>

В 5-й строке вместо тега <body> используется тег FRAMESET. Параметр ROWS указывает на то, что страница должна быть разбита на горизонтальные фреймы. Если же надо было разбить на горизонтальные фреймы, то надо указать COLS. В 5-й же строке указано, что фреймов должно быть 3 – на 1-й и 2-й отводится 45% высоты экрана, на третий – все, что осталось.

В 6-й, 7-й, 8-й строках непосредственно подключается содержимое фреймов, которое берется из файлов B611.HTM, B612.HTM, B613.HTM.

В настоящее время фреймы практически не используются, так как более удобным является использование слоев. Тем не менее активно используются плавающие фреймы, для задания которых используется тэг <IFRAME>. Плавающие фреймы в поддерживаются не всеми браузерами. Плавающий фрейм можно расположить в любом месте страницы, для этого надо указать код:

<iframe src="111.html" name="iframe1" width="300" height="300" align="left"><iframe>

Здесь будет подключен файл 111.html.

## Лекция 8. Каскадные таблицы стилей (CSS)

**План лекции:**

[Что такое CSS?](http://www.4stud.info/web-programming/css.html#css)

[Общий синтаксис таблиц стилей](http://www.4stud.info/web-programming/css.html#%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81)

[Правила CSS](http://www.4stud.info/web-programming/css.html#%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D0%B0%20css)

[Классы](http://www.4stud.info/web-programming/css.html#%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B%20css)

[Идентификаторы](http://www.4stud.info/web-programming/css.html#id)

[Группировка свойств](http://www.4stud.info/web-programming/css.html#groups)

[Использование в веб-страницах](http://www.4stud.info/web-programming/css.html#%D0%B8c%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20css)

[Встроенные стили](http://www.4stud.info/web-programming/css.html#%D0%B2%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D1%82%D0%B8%D0%BB%D0%B8)

[Внедренные стили](http://www.4stud.info/web-programming/css.html#%D0%B2%D0%BD%D0%B5%D0%B4%D1%80%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D1%82%D0%B8%D0%BB%D0%B8)

[Связанные таблицы стилей](http://www.4stud.info/web-programming/css.html#%D1%81%D0%B2%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D1%82%D0%B8%D0%BB%D0%B8)

[Аппаратно-зависимые стили](http://www.4stud.info/web-programming/css.html#media)

[Свойства CSS](http://www.4stud.info/web-programming/css.html#%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0%20%D1%81%D1%82%D0%B8%D0%BB%D0%B5%D0%B9)

[Позиционирование элементов](http://www.4stud.info/web-programming/css.html#%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

Каскадные таблицы стилей — это технология разделения содержания и оформления гипертекстового документа.

**CSS** (англ. Cascading Style Sheets — каскадные таблицы стилей) — технология описания внешнего вида документа, оформленного языком разметки.

Преимущественно используется как средство оформления веб-страниц в формате HTML и XHTML, но может применяться с любыми видами документов в формате XML, включая SVG и XUL.

Каскадные таблицы стилей используются создателями веб-страниц для задания цветов, шрифтов, расположения и других аспектов представления веб-документа. Основной целью разработки CSS являлось разделение содержимого (написанного на HTML или другом языке разметки) и оформления документа (написанного на CSS). Это разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печать, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля.

**Что такое CSS?**

*Каскадные таблицы стилей* (*Cascading Style Sheets, CSS*) — это стандарт, определяющий представление данных в браузере. Если HTML предоставляет информацию о структуре документа, то таблицы стилей сообщают как он должен выглядеть.

*Стиль* — это совокупность правил, применяемых к элементу гипертекста и определяющих способ его отображения. Стиль включает все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице.

*Таблица стилей* — это совокупность стилей, применимых к гипертекстовому документу.

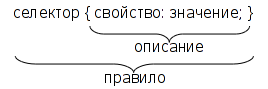
*Каскадирование* — это порядок применения различных стилей к веб-странице. Браузер, поддерживающий таблицы стилей, будет последовательно применять их в соответствии с приоритетом: сначала связанные, затем внедренные и, наконец, встроенные стили. Другой аспект каскадирования — *наследование (inheritance)*, — означает, что если не указано иное, то конкретный стиль будет применен ко всем дочерним элементами гипертекстового документа. Например, если вы примените определенный цвет текста в теге <div>, то все теги внутри этого блока будут отображаться этим же цветом.

Использование каскадных таблиц дает возможность разделить содержимое и его представление и гибко управлять отображением гипертекстовых документов путем изменения стилей.

Официальная информация о спецификации Cascading Style Sheets всегда доступна по адресу <http://www.w3.org/Style/CSS/>

**Общий синтаксис таблиц стилей**

Таблицы стилей строятся в соответствии с определенным порядком (синтаксисом), в противном случае они не могут нормально работать. Таблицы стилей составляются из определенных частей (рис. 1):

**Рис. 1. Синтаксис описания стиля CSS

* *Селектор (Selector)*. Селектор — это элемент, к которому будут применяться назначаемые стили. Это может быть тег, класс или идентификатор объекта гипертекстового документа.
* *Свойство (Property)*. Свойство определяет одну или несколько характеристик селектора. Свойства задают формат отображения селектора: отступы, шрифты, выравнивание, размеры и т.д.
* *Значение (Value)*. Значения — это фактические числовые или строковые константы, определяющие свойство селектора.
* *Описание (Declaration)*. Совокупность свойств и их значений.
* *Правило (Rule)*. Полное описание стиля (селектор + описание).

Таким образом, таблица стилей — это набор правил, задающих значения свойств селекторов, перечисленных в этой таблице. Общий синтаксис описания правила выглядит так:

селектор[, селектор[, ...]] {свойство: значение;}

Регистр символов значения не имеет, порядок перечисления селекторов в таблице и свойств в определении не регламентирован.

**Правила CSS**

Итак, каскадная таблица стилей — это набор правил форматирования тегов HTML. Приведем несколько примеров написания таких правил:

1. Основной текст с выравниванием по ширине, абзацный отступ 30px, гарнитура (шрифт) — Serif, кегль (размер шрифта) — 14px:

p {

text-align: justify;

text-indent: 30px;

font-family: Serif;

font-size: 14px;

}

Это правило будет применено ко всем тегам <p>.

1. Синий цвет для заголовков с первого по третий уровень:

h1, h2, h3 {

color: blue; /\* тоже самое, что и #0000FF \*/

}

1. Таблицы и изображения выводить без обрамления:

table, img {border: none;}

1. Ссылки в элементах списков показывать без подчеркивания:

li a {text-decoration: none;}

1. Внутренние отступы слева и справа для блоков (<div>), заголовков таблиц и ячеек таблиц установить в 10px и залить фон желтым цветом:

div, th, td {

padding-left: 10px;

padding-right: 10px;

background-color: yellow;

}

1. Все ссылки в документе отображать черным цветом и полужирным шрифтом, а в основном тексте и списках — обычным, а также выделять их зеленым цветом и подчеркивать только при наведении курсора (в описании правил использован псевдоэлемент a:hover).

a {color: black; font-weight: bold;}

p a, li a {font-weight: normal; text-decoration: none;}

p a:hover, li a:hover {

color: #00FF00; text-decoration: underline;

}

**Классы**

Стандарт CSS представляет возможности создания именованных стилей — стилевых классов. Это позволяет ответить на такой, например, вопрос: Как применить разные стили к одному и тому же селектору?

Предположим, что в документе вам нужны два различных вида основного текста — один без отступа, второй — с левым отступом и шрифтом красного цвета. Для этого нужно создать правила для каждого из них, например, так:

p {margin-left: 0;}

p.warn {margin-left: 40px; color: #FF00;}

Для применения созданного класса его имя нужно указать в атрибуте class для выбранных абзацев:

<p class=”warn”>Красный текст с отступом слева</p>

Общий синтаксис описания класса:

селектор**.имя\_класса** {описание}

При создании класса селектор можно не указывать, тогда это правило можно применять к любому селектору, поддерживающему тот же набор свойств.

Вот несколько примеров:

Правило:

.solid\_blue {color: blue;}

Использование:

<pclass=”solid\_blue”>Синийтекстабзаца</p>

<li class=”solid\_blue”>Синий текст элемента списка</li>

Правило:

h1.bigsans {font-family: Sans; font-size: 1.5em;}

h1.smallserif {font-family: Serif; font-size: .84em;}

Использование:

<h1 class=”bigsans”>Большой, но рубленый</h1>

<h1 class=”smallserif”>Маленький, но с засечками</h1>

Идентификаторы

В качестве селектора может выступать идентификатор элемента гипертекста, указанный в атрибуте id. Для назначения стилей таким элементам используется синтаксис, аналогичный описанию классов, но вместо точки ставится знак # (“решетка”). Например:

div#content {

position: absolute;

top: 10px;

left: 10%;

right: 10%;

border: solid 1px silver;

}

...

<div id="content">Текст</div>

Следует помнить, что идентификаторы элементов должны быть уникальны в пределах документа.

**Группировка свойств**

*Группировка (grouping)* состоит в объединении значений родственных свойств. При этом таблица стилей становится более компактной, но предъявляются более жесткие требования к описанию правил. Ниже приведен пример обычного стиля, задающего отступы:

div {

margin-left: 10px;

margin-top: 5px;

margin-right: 40px;

margin-bottom: 15px;

}

Это же правило можно переписать с группировкой в следующем виде:

div {margin: 5px 40px 15px 10px;} /\*порядок: top right bottom left\*/

Оба стиля будут отображаться одинаково.

Группировка может применяться для таких свойств, как padding, font, border, background и еще некоторых (см. документацию CSS)

**Использование в веб-страницах**

Существует три способа применения таблицы стилей к документу HTML:

* *Встраивание (Inline)*. Этот метод позволяет применить стиль к заданному тегу HTML.
* *Внедрение (Embedded)*. Внедрение позволяет управлять стилями страницы целиком.
* *Связывание (Linked или External)*. Связанная таблица стилей позволяет вынести описание стилей во внешний файл, ссылаясь на который можно контролировать отображение всех страниц сайта.

**Встроенные стили**

Встраивание стилей предоставляет максимальный контроль над всеми элементами веб-страницы. Встроенный стиль применяется к любому тегу HTML с помощью атрибута *style*следующим образом:

<p style="font: 12pt Courier">Это текст с кеглем 12 точек и гарнитурой Courier</P>

Пример:

<div style="font-family: Garamond; font-size: 18 pt;>"

Весь текст в этом разделе имеет размер 18 точек и шрифт Garamond.

<span style="color:#ff3300;">

А этот фрагмент еще и выделен красным цветом.</span>

</div>

Встроенные стили полезны, когда необходима тонкая настройка отображения некоторого элемента страницы или небольшой веб-страницы.

**Внедренные стили**

Внедренные стили используют тег <style>, который обычно размещают в заголовке HTML-документа (<head>...</head>):

<html>

<head>

...

<style>

правила CSS

</style>

...

</head>

<body>

...

**Связанные таблицы стилей**

*Связанные (linked)*, или *внешние (external)* таблицы стилей — наиболее удобное решение, когда речь идет об оформлении целого сайта. Описание правил помещается в отдельный файл (обычно, но не обязательно, с расширением *.css*). С помощью тега <link> выполняется связывание этой таблицы стилей с каждой страницей, где ее необходимо применить, например так:

<link rel=stylesheet href="sample.css" type="text/css">

Любая страница, содержащая такую связь, будет оформлена в соответствии со стилями, указанными в файле sample.css. Следует отметить, что файл со стилями физически может находиться на другом веб-сервере, тогда в href нужно указать абсолютный путь к нему.

**Проблемы с браузерами**

Обязательно просматривайте страницы с таблицами стилей в различных браузерах. Это связано с тем, что разные браузеры могут по-разному интерпретировать одно и то же правило, а некоторые свойства и/или значения и вовсе не поддерживать. Следует также тестировать страницы с отключенными стилями (например, в текстовых браузерах), чтобы убедиться, что страница читабельна.

**И снова каскадирование**

Если вам нужна сотня-другая-третья страниц HTML — используйте внешнюю, глобальную, таблицу стилей. Если некоторые из этих страниц требуют корректировки общего оформления — используйте внедренный стиль. А если на странице нужно явно изменить оформление одного-двух элементов, то применяйте встроенные стили. Именно в таком порядке происходит перекрытие стилей при каскадировании, схематично это можно представить так: *связанные стили -> внедренные стили -> встроенные стили*

**Аппаратно-зависимые стили**

Таблицы стилей могут применяться для управления отображением содержимого в зависимости от используемого устройства вывода (монитор, проектор, устройство печати, звуковой синтезатор и т.п.). Для этого в описание стилей включить тип устройства, например так:

@media print {/\* печатающее устройство \*/

BODY { font-size: 10pt; }

}

@media screen { /\* монитор \*/

BODY { font-size: 12pt; }

}

@media screen, print {

BODY { line-height: 1.2; }

}

@media all {

BODY { margin: 1pt; }

}

Как видно из примера, вся таблица разбивается на секции, каждая из которых начинается со слова @media, за которым следует название класса устройств и далее, в фигурных скобках, непосредственно описание стилей.

Можно разделить таблицы стилей иначе, указав тип устройства в теге <link>:

<link rel=stylesheet href="sample.css" type="text/css" **media=”screen”**>

**Свойства CSS**

В [табл.1](http://www.4stud.info/web-programming/css.html#css-properties) перечислены некоторые часто используемые свойства CSS и их назначение.

Таблица 1. Свойства элементов CSS

|  |  |  |
| --- | --- | --- |
| **Имя** | **Значения** | **Описание** |
| background | [background-color || background-image || background-repeat || background-attachment || background-position] | inherit | Управление фоном элемента |
| background-color | <color> | transparent | inherit | Цвет фона |
| background-image | <uri> | none | inherit | Фоновое изображение |
| background-position | [ [<percentage> | <length> ]{1,2} | [ [top | center | bottom] || [left | center | right] ] ] | inherit | Положение фоновой картинки |
| background-repeat | repeat | repeat-x | repeat-y | no-repeat | inherit | Повторение фоновой картинки |
| border | [ border-width || border-style || <color> ] | inherit | Границы элемента |
| border-collapse | collapse | separate | inherit | Объединение/разделение смежных границ |
| border-color | <color>{1,4} | transparent | inherit | Цвет границы |
| border-style | <border-style>{1,4} | inherit | Стиль линии границы |
| border-top border-right border-bottom border-left | [ border-top-width || border-style || <color> ] | inherit | Управление стилем заданной границы |
| border-width | <border-width>{1,4} | inherit | Толщина линии границы |
| bottom | <length> | <percentage> | auto | inherit | Низ элемента |
| clear | none | left | right | both | inherit | Запрет заполнения свободного пространства рядом с элементом |
| clip | <shape> | auto | inherit | Обрезка содержимого элемента |
| color | <color> | inherit | Цвет содержимого |
| cursor | [ [<uri> ,]\* [ auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help ] ] | inherit | Форма курсора |
| display | inline | block | list-item | run-in | compact | marker | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit | Способ отображения элемента |
| empty-cells | show | hide | inherit | Отображение пустых ячеек таблицы |
| float | left | right | none | inherit | Свободное размещение элемента |
| font | [ [ font-style || font-variant || font-weight ]? font-size [ / line-height ]? font-family ] | caption | icon | menu | message-box | small-caption | status-bar | inherit | Управление шрифтом |
| font-family | [[ <family-name> | <generic-family> ],]\* [<family-name> | <generic-family>] | inherit | Гарнитура |
| font-size | <absolute-size> | <relative-size> | <length> | <percentage> | inherit | Кегль |
| font-style | normal | italic | oblique | inherit | Стиль шрифта |
| font-variant | normal | small-caps | inherit | Варианты отображения шрифта |
| font-weight | normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit | Толщина шрифта |
| height | <length> | <percentage> | auto | inherit | Ширина элемента |
| left | <length> | <percentage> | auto | inherit | Положение левой границы элемента |
| line-height | normal | <number> | <length> | <percentage> | inherit | Высота строки |
| list-style | [ list-style-type || list-style-position || list-style-image ] | inherit | Стиль списка |
| margin | <margin-width>{1,4} | inherit | Внешний отступ |
| margin-top margin-right margin-bottom margin-left | <margin-width> | inherit | Внешний отступ по заданной стороне |
| padding | <padding-width>{1,4} | inherit | Внутренний отступ |
| padding-top padding-right padding-bottom padding-left | <padding-width> | inherit | Внутренний отступ по заданной стороне |
| position | static | relative | absolute | fixed | inherit | Позиционирование элемента |
| right | <length> | <percentage> | auto | inherit | Положение правой границы |
| text-align | left | right | center | justify | <string> | inherit | Выравнивание текстового блока |
| text-decoration | none | [ underline || overline || line-through || blink ] | inherit | Текстовые эффекты |
| text-indent | <length> | <percentage> | inherit | Абзацный отступ |
| text-transform | capitalize | uppercase | lowercase | none | inherit | Начертание текста |
| top | <length> | <percentage> | auto | inherit | Положение верхней границы элемента |
| vertical-align | baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit | Вертикальное выравнивание в пределах блока |
| visibility | visible | hidden | collapse | inherit | Управление видимостью элемента |
| white-space | normal | pre | nowrap | inherit | Управление пробелами между словами |
| width | <length> | <percentage> | auto | inherit | Ширина элемента |
| z-index | auto | <integer> | inherit | Порядок перехода по клавише Tab |

**Размещение каскадных таблиц**

1. Свойства можно определять непосредственно с тегом. Например, вставка следующего тега сделает текст красным на синем фоне

<span style="color:red; background:blue; font: bold 1.8em Arial ">красныйнасинем</span>

2. Таблицу можно разместить между тегами <style> и</style>. Например,

<head>

<style>

a { color: #33ADDB;

background-color: inherit;}

a:hover { color: #575757;

background-color: inherit;}

h1 { font: bold 1.8em Arial, Sans-Serif;

letter-spacing: -1px;

margin: 0;

padding: 0;}

h1 a { text-decoration: none;}

</style>

</head>

И теперь гиперссылки и заголовки первого уровня будет определяться в соответствии с указанными правилами.

3. Таблицу стилей можно разместить в отдельном файле. А подключение затем выполняется так:

<link rel="stylesheet" type="text/css" href="style.css">

**Верстка**

Наиболее популярным является деление макетов по ширине и количеству колонок. Выделяют следующие типы макетов, связанных с шириной:

• фиксированные;

• резиновые;

• эластичные;

• адаптивные;

• комбинированные.

*Фиксированный макет* располагается по центру окна браузера, а его ширина ограничивается заданными размерами в пикселах.

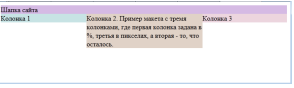
При создании резинового макета задается в процентах ширина колонок таким образом, что макет занимает всю свободную ширину окна браузера.

*Эластичный макет* по своему виду может не отличаться от фиксированного или резинового макета. Размер элементов задается в em, привязанных к размеру шрифта. Значение em можно использовать не для всех элементов, оставляя ширину некоторых фиксированной.

*Адаптивный мак*ет подстраивается под разрешение монитора и окна браузера, меняя при необходимости ширину макета, число колонок, размеры изображений и текста. Для этого заготавливается несколько стилевых правил или файлов под разный диапазон разрешений, выбор правил происходит через скрипты или CSS3, которые и определяют нужную для этого информацию о пользователе.

*Комбинированный ма*кет предполагает использование разной ширины для отдельных частей страницы, например, шапку и подвал делают резиновыми, а контент фиксированным.

Пример макета с тремя колонками, где первая колонка задана в %, третья в пикселах, а вторая – то, что осталось.



<html>

<head>

<style>

.header { background: #D5BAE4; }

.layout { position: relative;}

.layout DIV { position: absolute; }

.col1 { background: #C7E3E4; width: 30%; }

.col2 { background: #E0D2C7; left: 30%; right: 200px; }

.col3 { background: #ECD5DE; right: 0; width: 200px; }

</style>

<title>Триколонки</title>

</head>

<body>

<div class="header">Шапкасайта</div>

<div class="layout">

<div class="col1">Колонка 1</div>

<divclass="col2">Колонка 2. Пример макета с тремя колонками, где первая колонка задана в %, третья в пикселах, а вторая - то, что осталось.</div>

<div class="col3">Колонка 3</div>

</div>

</body>

</html>

Параметры CSS, управляющие положением на странице

position: absolute | fixed | relative | static | inherit

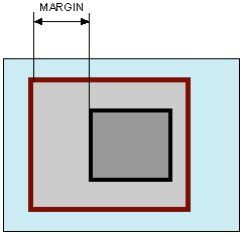
Устанавливает способ позиционирования элемента относительно окна браузера или других объектов на веб-странице. absolute указывает, что используются абсолютные координаты. relative указывает, что используются относительные координаты. inherit наследует значение родителя.

**left** задает положение относительно левого края контейнера. top задает положение относительно верхнего края контейнера. Задаются в процентах или пикселях.

**z-index** указывает на то, какой элемент должен располагаться выше при перекрытии. Измеряется в единицах.

**visibility** определяет видимость элемента. Принимает значения visible (видимый), hidden (скрытый), inherit (наследуется от родительского элемента).

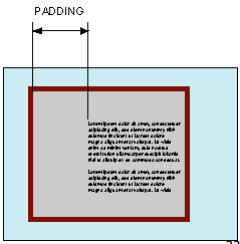
**margin** устанавливает величину отступа от каждого края элемента.



Отступом является пространство от границы текущего элемента до внутренней границы его родительского элемента.

Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом. Если задано одно значение, то будут установлены четыре одинаковых отступа. Если два, то первое значение – отступ сверху и снизу, второе – слева и справа. Если определены три значения, то первое значение задает отступ от верхнего края, второе — одновременно от левого и правого края, а третье — от нижнего края. Четыре параметра определяют отступ от верхнего, правого, нижнего и левого края.

**padding** устанавливает значение полей вокруг содержимого элемента. Полем называется расстояние от внутреннего края рамки элемента до воображаемого прямоугольника, ограничивающего его содержимое. Также как и у margin возможны несколько вариантов значений.



**Лекция 9. Язык JavaScript.**

**План лекции:**

Обзор возможностей языка JavaScript

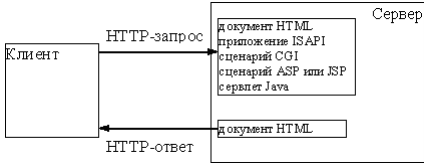
Общий обзор языка. Основные определения

Язык ядра JavaScript

Управляющие конструкции языка JavaScript: Операторы выбора, Операторы цикла, Создание и вызов функций в JavaScript

**1. Обзор возможностей языка JavaScript**

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



Его основу составляют HTTP-сообщения, подразделяемые на:

• запрос (request) клиента к серверу;

• ответ (response) сервера клиенту.

Стандартный язык разметки HTML позволяет легко создавать статичные Web–страницы. Пользователь не может менять их содержимое, не может взаимодействовать с ними. Для того чтобы сделать страницу понастоящему интерактивной, нужен еще один язык, выполняемый в контексте браузера, - скриптовый язык.

Исследования работы приложений интернета показали, что для выполнения определенных действий пользователя нет необходимости постоянно обращаться к серверу - эти действия можно реализовать на стороне клиента, если бы он позволял каким-то образом их запрограммировать. Так появился встроенный в программу просмотра Web-страниц (браузер) язык JavaScript, который расширил возможности языка разметки HTML, предоставляя разработчику возможность встраивать в документ HTML код программы, выполняющейся на клиенте.

Скриптовый язык используется для создания интерактивных страниц. Обычно он не содержит всех возможностей настоящих языков программирования, таких, например, как работа с файлами или управление графикой. Созданные с помощью скриптовых языков программы не могут выполняться самостоятельно - они работают только в контексте браузера, поддерживающего выполнения скриптовых программ. Создаваемые на скриптовых языках программы, называются сценариями или скриптами, включаются в состав Web-страниц и распознаются и обрабатываются браузером отдельно от остального HTML - кода.

Язык программирования JavaScript - объектно-ориентированный язык разработки встраиваемых приложений, выполняющихся как на стороне клиента, так и на стороне сервера.

Веб-обозреватель, работающий на компьютере-клиенте, обеспечивает среду, в которой JavaScript имеет доступ к объектам, которые представляют собой окна, меню, диалоги, текстовые области и т. д. Кроме того, обозреватель позволяет присоединить сценарии на языке JavaScript к 7 таким событиям, как загрузка и выгрузка страниц и графических образов, нажатие клавиш и движение мыши, выбор текста и пересылка форм. При этом программный код сценариев только реагирует на события и поэтому не нуждается в главной программе. Набор объектов, предоставляемых обозревателем, известен под названием DocumentObjectModel (DOM).

Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра HTML-страницы пользователем. При этом перезагрузки страницы не происходит. Основные области использования JavaScript при создании интерактивных HTML- страниц:

• Динамического создания содержимого страницы во время ее загрузки или уже после того, как она полностью загружена;

• Отображения диалоговых панелей и сообщений в статусной строке браузера;

• Оперативная проверка достоверности заполняемых пользователем полей форм HTML до передачи их на сервер;

• Создание динамических HTML-страниц совместно с каскадными таблицами стилей и объектной моделью документа (DHTML);

* 1. **Общий обзор языка. Основные определения**

Любая программа оперирует некими данными: именем стилевого класса, размерами элемента, цветом шрифта и прочие. JavaScript может манипулировать данными, относящимися к разным типам.

Тип данных описывает их возможные значения и набор применимых к ним операций. Типы данных бывают простыми и сложными. Сущность, относящаяся к простому типу данных может хранить только одно значение (это строковые, числовые и логические типы данных). Сущность сложного типа данных может хранить сразу несколько значений. Например – массивы. Другой пример сложного типа данных – объекты.

Для создания механизма управления страницами на клиентской стороне было предложено использовать объектную модель документа. Суть модели в том, что каждый HTML-контейнер - это объект, который характеризуется тройкой:

* Свойства
* Методы
* События

Существование программных объектов самих по себе не имеет никакого смысла. Они дадут преимущества при программировании тогда, когда можно организовать их взаимодействие.

1. Объекты – это сложные сущности, позволяющие хранить сразу несколько значений разных типов данных, они представляют собой блоки, из которых строится JavaScript. Применяются для возвращения значений и изменения состояния форм, страниц, браузера и определенных программистом переменных. Объекты можно сопоставить с существительными. Кошка, автомобиль, дом, компьютер, форма – все это существительные, они могут быть представлены как объекты.
2. Экземпляры объекта – сущности, хранящие реальные данные и созданные на основе этого объекта. То есть конкретный, реально существующий дом, находящийся по заданному адресу можно рассматривать, как экземпляр объекта типа дом.
3. Свойства – набор внутренних параметров объекта. Используются для того, чтобы различать экземпляры одного объекта – например, все экземпляры типа дом. Свойства сравнимы с прилагательными и ссылаются на уникальные для каждого экземпляра объекта особенности. Один и тот же объект может обладать многими свойствами: дом может быть большим и маленьким, синим и красным. Разные объекты могут обладать одинаковыми свойствами: дерево, так же, как и дом, может быть большим и маленьким, синим и красным... Большинство свойств объекта мы можем изменять, воздействуя на них через методы.
4. Методы - это действие или способ, при помощи которого мы можем изменять определенные свойства объекта, то есть управлять этими объектами, а также в некоторых случаях менять их содержимое.
5. События – это очень важное в программировании на JavaScript понятие. События главным образом порождаются пользователем, являются следствиями его действий. Если пользователь нажимает кнопку мыши, то происходит событие, которое называется Click. Если экранный указатель мыши движется по ссылке HTML-документа, происходит событие MouseOver. Существует несколько различных событий.
6. Оператор - это команда, инструкция для компьютера. Встретив в программе тот или иной оператор, машина четко его выполняет.
7. Функция - это определенная последовательность операторов, то есть набор команд, последовательное выполнение которых приводит к какому-то результату. Например, выполнение кем-то заданной Вами функции (процедуры) "возьми стакан, открой кран, набери в него воды и принеси мне" приведет к результату: Вы получите стакан воды из-под крана.
8. Переменная - в языках программирования переменные используются для хранения данных определенного типа, например параметров свойств объекта. Каждая переменная имеет свое имя (идентификатор) и хранит только одно значение, которое можетменяться в ходе выполнения программы. Данные могут быть разных типов: целое число, десятичная дробь, логическая константа, текстовая строка.

*Понятие объектной модели применительно к JavaScript*

При загрузке HTML-страницы в браузер интерпретатор языка создает объекты со свойствами, определенными значениями тэгов страницы. Для правильного использования объектных моделей следует четко понимать, как браузер компонует страницы и, тем самым, создает иерархию объектов. При загрузке страницы просматриваются сверху вниз, тем самым последовательно происходит компоновка страницы и ее отображение в окне браузера. А это означает, что и объектная модель страницы также формируется последовательно, по мере ее обработки. Поэтому невозможно обратится из сценария, расположенного ранее какойлибо формы на странице, к элементам этой формы. Всегда следует помнить о том, что браузер последовательно сверху вниз интерпретирует содержимое HTML-страницы.

Еще один аспект работы с объектами языков сценариев заключается в том, что нельзя изменить свойства объектов. Браузер обрабатывает страницу только один раз, компонуя и отображая ее. Поэтому попытка в сценарии изменить свойство отображенного элемента страницы, обречена на провал. Только повторная загрузка страницы приведет к желаемому результату.

*Размещение операторов языка JavaScript на странице*

Встроить сценарий JavaScript в HTML-страницу можно несколькими способами.

1. Задание операторов языка внутри тэга,<script>языка HTML.

Для внедрения в HTML-страницу сценария JavaScript в спецификацию языка HTML был введен тэг-контейнер<script>…</script>, внутри которого могут располагаться операторы языка JavaScript. Обычно браузеры, не поддерживающие какие-нибудь тэги HTML, просто их игнорируют, анализируя, однако, содержимое пропускаемых тэгов с точки зрения синтаксиса языка HTML, что может приводить к ошибкам при отображении страницы. Во избежание подобной ситуации следует помещать операторы языка JavaScript в контейнер комментария< !-- ... //-->как показано ниже:

<script (language="javascript")>

<!--

операторы javascript

//-->

</script>

Параметр language задает используемый язык сценариев. В случае языка JavaScript его значение задавать не обязательно, так как этот язык используется браузерами по умолчанию.

*Примечание:* символы // перед закрывающим тэгом комментария -->являются оператором комментария JavaScript. Он необходим для правильной работы интерпретатора.

Документ может содержать несколько тэгов <script>, расположенных в любом месте документа. Все они последовательно обрабатываются интерпретатором JavaScript по мере отображения частей документа в окне браузера. В связи с этим ссылка на переменную, определенную в сценарии, размещенном в конце документа, может привести к генерации ошибки интерпретатора при обращении к такой переменной из сценария в начале документа.

2. Задание файла с кодом JavaScript.

Тег <script> имеет параметр src, позволяющий связать встраиваемый сценарий с внешним файлом, содержащим программный код на языке JavaScript. В качестве значения параметра задается полный или относительный URL-адрес ресурса. Задание закрывающего тэга обязательно, независимо от того, заданы или нет операторы внутри тэга. Следующий фрагмент кода связывает документ HTML с файломисточником, содержащим некоторый набор функций:

<script language="JavaScript" src="http://url/file.js">

операторы javascript

</script>

*Примечание:* связываемый внешний файл не должен содержать тэгов HTML и должен иметь расширение .js.

3. Использование выражений JavaScript в качестве значений параметров тэгов HTML.

Переменные и выражения JavaScript можно использовать в качестве значений параметров тэгов HTML. Например:

<a href="javascript: window.open('name.htm', '\_self')">

<imgsrc="images/ch1.png" border="0">

</a>

4. Определение обработчика событий в тэге HTML.

**1.2. Язык ядра JavaScript**

*Синтаксис языка*

Язык JavaScript чувствителен к регистру.

Приложение JavaScript представляет собой набор операторов языка (команд), последовательно обрабатываемых встроенным в браузер интерпретатором. Каждый оператор можно располагать в отдельной строке. В этом случае разделитель ‘;’, отделяющий один оператор от другого, не обязателен. Его используют только в случае задания нескольких операторов на одной строке. Любой оператор можно расположить в нескольких строках без всякого символа продолжения. Например, следующие два вызова функции alert эквивалентны:

…

alert("Подсказка");

alert(

"Подсказка"

);

…

Нельзя перемещать на другую строку единый строковый литерал - он должен располагаться полностью на одной строке текста программы или разбит на два строковых литерала, соединенных операцией конкатенации ‘+’:

…

alert("Подсказка"); // правильно

alert("Под сказка"); // не правильно

alert("Под" + "сказка"); // правильно (но браузер выведет текст одной строкой!)

…

Пробельные символы в тексте программы являются незначащими, если только они не используются в строковых литералах.

В JavaScript строковые литералы можно задавать двумя равноправными способами - последовательность символов, заключенная в двойные или одинарные кавычки:

"Анна"

'Анна'

В строковых литералах можно использовать ESC-последовательности, которые начинаются с символа обратной наклонной черты, за которой следует обычный символ. Некоторые подобные комбинации трактуются как один специальный символ.

Esc-последовательности Символ

\b Возврат на один символ

\f Переход на новую страницу

\n Переход на новую строку

\r Возврат каретки

\t Горизонтальная табуляция Ctrl-I

\' Апостроф

\" Двойные кавычки

\\ Обратная наклонная черта

ESC-последовательности форматирования используются при отображении информации в диалоговых окнах, отображаемых функциями alert(), prompt() и confirm(), а также, если методом document.write() записывается содержимое элемента pre. Комментарии в программе JavaScript двух видов - однострочные и многострочные:

// комментарий, расположенный на одной строке.

/\*

комментарий, расположенный

на нескольких строках.

\*/

Ссылка на объект осуществляется по имени, заданному параметром name тэга HTML, с использованием точечной нотации. Например, пусть в документе задана форма с двумя полями ввода:

<form name="form1">

Фамилия: <input type = "text" name = "student" size = 20>

Курс:<input type = "text" name = "course" size = 2>

</form>

Для получения фамилии студента, введенного в первом поле ввода, в программе JavaScript следует использовать ссылку document.form.student.value, a чтобы определить курс, на котором обучается студент, необходимо использовать ссылку document.form.course.value.

*Переменные и литералы в JavaScript*

В JavaScript все переменные вводятся с помощью одного ключевого слова var. Синтаксическая конструкция для ввода в программе новой переменной с именем name1 выглядит следующим образом:

var name1;

Объявленная таким образом переменная name1 имеет значение 'undefined' до тех пор, пока ей не будет присвоено какое-либо другое значение, которое можно присвоить и при ее объявлении:

var name1 = 5;

var name1 = "новая строковая переменная";

JavaScript поддерживает четыре простых типа данных:

• Целый

• Вещественный

• Строковый

• Логический (булевый)

Для присваивания переменным значений основных типов применяются литералы – буквальные значения данных соответсвующих типов.

*Выражения JavaScript*

Выражение – комбинация переменных, литералов и операторов, в результате вычисления которой получается одно единственное значение. Переменные в выражениях должны быть инициализированы.

*1. Присваивание*

Оператор присваивания (=) рассматривается как выражение присваивания, которое вычисляется равным выражению правой части, и в то же время он присваивает вычисленное значение выражения переменной, заданной в левой части:

var name2=10;

*2. Арифметическое выражение*

Вычисляемым значением арифметического выражения является число. Создаются с помощью арифметических операторов.

Оператор Действие

+ Сложение

- Вычитание

\* Умножение

/ Деление

% Остаток от деления целых чисел

++ Увеличение значения на единицу

-- Уменьшение значения на единицу

*3. Логическое выражение*

Вычисляемым значением логического выражения может быть true или false. Для создания используются операторы сравнения или логические операторы, применяемые к переменным любого типа.

Операторы сравнения Значение

= = Равно

! = Не равно

>= Больше или равно

<= Меньше или равно

> Строго больше

< Строго меньше

Логические Операторы Значение

&& логическое И

|| логическое ИЛИ

! логическое НЕ

*4. Строковые выражения*

Вычисляемым значением строкового выражения является число. В JavaScript существует только один строковый оператор – оператор конкатенации (сложения) строк:

string1 = “Моя ” + “строка”

**1.3. Управляющие конструкции языка JavaScript**

*Операторы JavaScript*

Операторы служат для управления потоком команд в JavaScript. Блоки операторов должны быть заключены в фигурные скобки.

*1. Операторы выбора*

*• условный оператор if*

Эта управляющая структура используется, когда необходимо выполнить некий программный код в зависимости от определенных условий. Также предусмотрена конструкция if-else (если-тогда-иначе).

if (условие\_1)

{

оператор\_1; // эти операторы выполняются, если условие\_1 верно

оператор\_2;

}

else

{

оператор\_3; // эти операторы выполняются, если условие\_1 ложно оператор\_4;

}

Условие для проверки (вопрос компьютеру) записывается сразу после слова if в круглых скобках. После этого в фигурных скобках пишется то, что будет предприниматься в случае выполнения условия. Далее else и снова в фигурных скобках то, что выполнится в случае, если условие не сработает. Количество различных действий между фигурными скобками неограниченно, фактически можно выполнить две различные программы. При сравнении можно использовать логические выражения. Например:

<scriptlanguage="JavaScript">

varx = 5;

vary = 10;

if (x>y)

{

alert('x - максимальное число')

}

else

{

alert('y - максимальное число')

}

</script>

*• оператор выбора switch*

Это фактически несколько условных операторов, объединенных в одном. В данном операторе вычисляется одно выражение и сравнивается со значениями, заданными в блоках case. В случае совпадения выполняются операторы соответствующего блока case.

switch (выражение) {

case значение1:

оператор\_1;

break;

case значение2:

оператор\_2;

break;

…..

default:

оператор;

}

Если значение выражения не равняется ни одному из значений, заданных в блоках case, то вычисляется группа операторов блока default, если этот блок задан, иначе происходит выход из оператора switch. Необязательный оператор break, задаваемый в блоках case, выполняет безусловный выход их оператора switch.

*2. Операторы цикла*

Оператор цикла повторно выполняет последовательность операторов JavaScript, определенных в его теле, пока не выполниться некоторое заданное условие.

*• цикл for (цикл со счетчиком)*

for (i=1; i <10 ; i++){

<телоцикла>

}

Первый параметр (i=1) определяет счетчик и указывает его начальное значение. Этот параметр называется начальным выражением, поскольку в нем задается начальное значение счетчика (начальное значение в данном 16 случае равно единице). Это выражение инициализации выполняется самым первым и всего один раз.

Второй параметр (i<10) - это условие, которое должно быть истинным, чтобы цикл выполнялся, как только условие цикла становится ложным, работа цикла завершается. Он называется условием цикла. Проверка условия цикла осуществляется на каждом шаге; если условие истинно, то выполняется тело цикла (операторы в теле цикла). Цикл в данном случае выполнится только девять раз, так как задано условие i<10.

<script language="JavaScript" type="text/JavaScript">

for (var i=1; i< 10; i++){ document.write("<hr align='center' width='100> ");

}

< /script>

*• цикл while (цикл с предусловием)*

while (условие)

{

< тело цикла>

}

Пока значение условия - true (истинно), выполняется тело цикла. Тело цикла может быть представлено простым или составным оператором.

Оператор while содержит в скобках все необходимые параметры условия цикла (логическое выражение). После определения всех параметров цикла вводится открывающая фигурная скобка, символизирующая начало тела цикла. Закрывающая фигурная скобка вводится в конце тела цикла. Все операторы, введенные в скобках, выполняются при каждом прохождении цикла.

<script language="JavaScript">

var i=1; while(i<=10){

document.write('число='+i+'<br>');

i=i+2;

}

</script>

*• прерывание и перезапуск цикла*

Оператор прерывания break позволяет прервать выполнение цикла и перейти к следующему за ним выражению:

a = 10;

i = 1;

while (a<100){

a = a \* i;

if (i>4) break;

++i;

}

Если значение i превысит 4, то прерывается выполнение цикла.

Оператор перезапуска continue позволяет перезапустить цикл, т.е. оставить невыполненными все последующие выражения, входящие в тело цикла, и запустить выполнение цикла с самого начала.

a = 10;

i = 1;

while (a<100){

++i;

if (i>2 && i<11) continue;

a = a \* i;

}

*Создание и вызов функций в JavaScript*

В JavaScript функцией называется именованная часть программного кода, которая выполняется только при обращении к ней посредством указания ее имени. Функции создаются с помощью ключевого слова function. Обычно функции располагают в секции <head>. Такое расположение функций в HTML-документе гарантирует их полную загрузку до того момента, когда их можно будет вызвать из секции <body>.

После названия функции (func\_name) ставятся двойные круглые скобки, программный код при этом заключается в фигурные скобки:

<script language="JavaScript">

functionfunc\_name()

{

программный код функции (тело функции)

}

</script>

Для того, чтобы вызвать функцию в нужном месте, необходимо просто указать ее имя в тексте:

<script language="JavaScript">

func\_name();

</script>

Второй вариант вызова функции непосредственно в HTML теге:

<a href="javascript:func\_name()">Текстссылки</a>

Ниже приведен код страницы HTML, после загрузки которой каждые три секунды будет появляться сообщение, генерируемое вызовом функции myMessage():

<script>

functionmyMessage()

{

alert("My Message")

}

</script>

<body onload='setTimeout ("myMessage()",3000)'>

<p>Каждые три секунды будет появляться сообщение</p>

</body>

Метод setTimeout() запускает выполнение кода JavaScript, задаваемого первым строковым параметром, через определенный промежуток времени после выполнения метода. Интервал задается в миллисекундах (1000 соответствует 1 секунде).

**Лекция 10. Стандартные объекты и функции ядра JavaScript**

**План лекции:**

Стандартные объекты: Объект Array, Объект Date, Объект Math

Стандартные функции верхнего уровня

Объекты клиента: Объект navigator, Объект window, Объект document, Методы write() / writeln(), Объект location, Объект form

Иерархия объектов

Обработка событий: Атрибут onClick, Работа с меню

Управление логикой программного кода при помощи событий

Определение событий формы

Вставка звука

**Стандартные объекты и функции ядра JavaScript**

*Объект Array*

Массив - упорядоченный набор однородных данных, к элементам которого можно обращаться по имени и индексу. Язык JavaScript не имеет встроенного типа данных для создания массивов, поэтому для решения используется объект Array и его методы. Для создания объекта Array вызывается оператор new и конструктор массива - системная функция (ее имя совпадает с именем объекта), инициализирующая элементы массива:

m=newArray();

Заполнение массива происходит позже. Например:

<scriptlanguage="JavaScript">

//создание нового массива

m=newArray();

//заполнение массива

m[0]=1;

m[1]=2;

m[2]=4;

m[3]=56;

</script>

В приведенном выше примере с помощью команды new создается массив m, а затем происходит его заполнение - каждому элементу присваивается определенное значение.

m=newArray(1,2,4,56);

Вызывается команда new и сразу задаются значения всех элементов массива.

<scriptlanguage="JavaScript">

//создание нового массива и его заполнение

m=newArray(1,2,4,56)

</script>

Объявление строковых массивов проводится тем же способом, что и объявление числовых массивов.

Методы

объекта Array Действие

join() Объединяет все элементы массива в одну строку с указанием разделителя.

reverse() Изменяет порядок элементов в массиве - первый элемент становится последним, последний - первым

sort() Выполняет сортировку элементов массива

split() Разделяет строку на составные части

concat() Объединяет два массива в один

slice() Выделяет часть массивы

toString() Возвращает строку - результат конкатенации всех элементов массива. Элементы массива в строке разделены запятой. Свойство

length Возвращает длину массива (число элементов в нем).

Пусть определены два массива:

array1 = newArray("Первый","Второй","Третий");

array2 = newArray("Один","Два","Три");

Тогда метод join() первого массива array1.join() возвратит строку:

"Первый,Второй,Третий"

Метод sort() первого массива array1.sort() упорядочит элементы массива array1 (переставив их местами непосредственно в самом массиве array1) в алфавитном порядке:

array1[0] = "Второй";

array1[1] = "Первый";

array1[2] = "Третий";

Поскольку некоторые методы массива возвращают массив, то к нему можно сразу же применить какой-либо метод, продолжив "точечную" нотацию. Например, array1.concat(array2).sort() объединит два массива в один новый и отсортирует его.

*Объект Date*

Используется для представления дат в программах JavaScript. Время храниться в виде числа миллисекунд, прошедших от 1 января 1970 года.

Данный объект создается также, как и любой объект в JavaScript – с помощью оператора new и конструктора, в данном случае Date():

date1 = newDate(); // значением переменной date1 будет текущая дата

Параметром конструктора может быть строка, в которой записана нужная дата:

date1 = newDate(“january 14, 2000, 12:00:00”);

Можно задать список параметров:

date1 = newDate(2000, 1, 14, 12, 0, 0);

*Объект Math*

В свойствах данного объекта хранятся основные математические константы, а его методы вычисляют основные математические функции. При обращении к данному объекту, создавать его не надо, но необходимо явно указывать его имя Math. Например:

p = Math.PI; // хранится значение числа пи.

Объект String

Можно явно создавать строковый объект, используя оператор new и конструктор:

myString = newString(“Hello!”);

Данный объект имеет единственное свойство length, хранящее длину строки, содержащейся в строковом объекте, и два типа методов: одни непосредственно влияют на содержание самой строки, вторые возвращают отформатированный HTML-вариант строки.

*Стандартные функции верхнего уровня*

В JavaScript существуют несколько функций, для вызова которых не надо создавать никакого объекта, она находятся вне иерархии объектов.

Функция parseFloat(parameter) анализирует значение переданного ей строкового параметра на соответствие представлению вещественного числа.

Функция parseInt(parameter, base) пытается возвратить целое число по основанию, заданному вторым параметром. Эти функции полезны при анализе введенных пользователем данных в полях формы до их передачи на сервер.

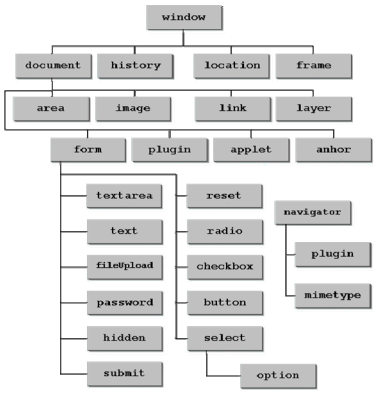
Функции Number(object) и String(object) преобразуют объект, заданный в качестве его параметра в число или строку.

**1.5. Объекты клиента**

При интерпретации страницы HTML браузером создаются объекты JavaScript, свойства которых представляют значения параметров тэгов языка HTML.

*Иерархия объектов*

Созданные объекты существуют в виде иерархической структуры, отражающей структуру самой HTML-страницы. На верхнем уровне расположен объект window, представляющий собой активное окно браузера. Далее вниз по иерархической лестнице следуют объекты frame, document, location и history и т.д. Значения свойств объектов отражают значения соответствующих параметров тэгов страницы или установленных системных параметров. На рисунке показана структура объектов клиента (браузера).



Особняком стоит объект navigator с двумя дочерними (подчиненными) объектами. Он относится к самому браузеру, и его свойства позволяют определить характеристики программы просмотра.

Каждая страница в добавление к объекту navigator обязательно имеет еще четыре объекта:

 window — объект верхнего уровня, свойства которого применяются ко всему окну, в котором отображается документ;

 document — свойства которого определяются содержимым самого документа: связи, цвет фона, формы и т. д.;

 location — свойства которого связаны с url-адресом отображаемого документа;

 history — представляет адреса ранее загружавшихся HTML-страниц.

Кроме указанных объектов страница может иметь дополнительные объекты, зависящие от ее содержимого, которые являются дочерними объектами объекта document. Если на странице расположена форма, то все ее элементы являются дочерними объектами этой формы. Для задания точного имени объекта используется точечная нотация с полным указанием всей цепочки наследования объекта. Наиболее общий объект высшего уровня находится слева в выражении, и слева направо происходит переход к более частным объектам, являющимся при этом наследниками высших в иерархии объектов.

Кроме этих классов объектов пользователь может создавать и свои собственные. Но обычно большинство программ используют эту систему классов и не создают новых.

*Объект navigator*

Этот объект применяется для получения информации о версиях.

Синтаксис:

navigator.name\_properties

Методы и события, догадаться не определены для этого объекта. Да и свойства только для чтения, так как ресурс с информацией о версии недоступен для редактирования.

Свойства:

• appCodeName - кодовое имя браузера;

• appName - название браузера;

• appVersion - информация о версии браузера;

• userAgent - кодовое имя и версия браузера;

Ниже приведен пример использования объекта navigator.

<html>

<head>

<title>navigator</title>

<script language="javascript">

varfirstn = window.prompt("Введитевашеимя: ","вашеимя")

function welcome(){

varappname=navigator.appName

varappver=navigator.appVersion

window.alert("Привет, " + firstn + ". Выиспользуете " + appname + ". Версия " + appver + ". Спасибозавизит.");

}

functionwritename(){

document.write(firstn + ".");

}

</script>

</head>

<body onLoad="welcome()">

Добропожаловать,

<script language="JavaScript">

writename();

</script>

</body>

</ html>

*Объект window*

Объект window создается автоматически при запуске браузера, так как для отображения документа необходимо окно. Одно из назначений объекта окна - это создание нового окна. Новое окно браузера создается с помощью метода window.open(). Метод window.open() имеет ряд дополнительных аргументов, которые позволяют задать местоположение окна, его размер и тип, а также указывают, должно ли окно иметь полосы прокрутки, полосу команд и т. п. Помимо этого можно задавать и имя окна.

В общем виде данный метод можно представить следующим образом:

window.open('url', 'name', 'parameters')

Рассмотрим синтаксис более подробно:

• Первый параметр метода window.open() - это url документа, загружаемого в окне. Если его не заполнить, то окно останется пустым.

• Второй параметр определяет название окна (name). Это имя может использоваться для обращения к созданному окну.

• Третий параметр представляет список необязательных опций, разделенных запятой. С их помощью Вы определяете вид нового окна: наличие в нем панелей инструментов, строки состояния и других элементов.

Приведем таблицу с описанием параметров нового окна, задаваемого третьим параметром (parameters) метода open().



Объект window использует три метода отображения сообщений:

 метод prompt() – выводит диалоговое окно с полем ввода, куда пользователь может ввести информацию

 метод alert() – выводит на экран окно - сообщение с кнопкой ОК и определенным программистом текстом

 метод confirm() – выводит диалоговое окно с кнопками ОК и Cancel.

Дает возможность пользователю продолжить или отменить предложенную операцию.

Сообщение, которое вы хотите вывести на экран, набирается в кавычках внутри круглых скобок.

Данный скрипт запрашивает имя посетителя и выдает приветствие с введенным именем.

<scriptlanguage="JavaScript" >

name=window.prompt ("Введите, пожалуйста, свое имя", "Ваше имя"); window.alert ("Вас зовут, " + name);

</script>

В этом фрагменте кода метод prompt имеет следующие параметры: текст запроса и значение, заполняющее поле ввода по умолчанию; переменная name - имя переменной, куда сохраняется введенная информация (имя может быть любым).

*Объект document*

Объект document имеет дело прежде всего с телом HTML-страницы. Он имеет несколько дочерних объектов (коллекций): all, images, link, anchor и form. Пользуясь объектной моделью построения документа можно, например, обратиться к любой картинке на странице через следующий синтаксис:

document.images.name.src

Для document не существует никаких событий. Некоторые свойства и методы перечислены в таблице, из методов наиболее употребимыwrite и writeln.

**Свойства** **Назначение**

**bgColor** Устанавливает цвет фона текущего документа. Этот цвет может иметь шестнадцатеричное представление #rrggbb или соответствующее название. Синтаксис: document.bgColor="#e7e6d8"

**fgColor** Устанавливает цвет текста документа. Аналогичен по функциям свойству bgColor

**referrer**  Указывает url документа, на который ссылается пользователь в настоящее время. Например, если кто-то обратился по адресу: http://www.nm.org/welcome.htm с сервера 26 http://www.someplace.com, то свойством referrer будет: http://www.someplace.com, если это свойство было в странице вышеупомянутого расположения; в противном случае оно обращается в null

**location** Соответствует адресу url текущего документа

**Методы Назначение**

**write()**

**writeln()** Записывает HTML-текст в текущий документ и должен вызываться, когда документ открывается для записи. Синтаксис: document.write('somestring'), где somestring может быть одной строкой, переменной или же несколькими связанными строками в формате HTML, которые выводятся на экран

**lastModified()** Показывает дату последней модификации документа: date1 = document. lastModified

**open()** Открывает документ для записи дополнительных строк в формате HTML: document.**open()**

**close**() Закрывает документ, который вызывался методом document.open(): document.close().

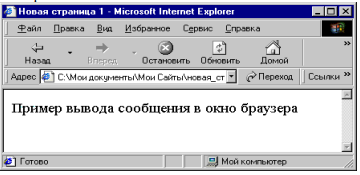
*Методы write() / writeln().*

Вызов метода document.write() с указанием определенных параметров приводит к отображению текста в окне браузера. В качестве параметра при вызове метода document.write() мы указываем строку, которую хотели бы увидеть на экране.

<scriptlanguage="JavaScript">

document.write('Пример вывода сообщения в окно браузера')

</script>

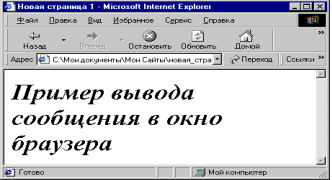


Выводимая строка может содержать и тэги языка HTML. В этом случае браузер выведет данную строку точно так же, как если бы она была размещена непосредственно в HTML документе.

<scriptlanguage="JavaScript">

document.write('<h1><b><i>Пример вывода сообщения в окно браузера</i></b></h1>')

</script>



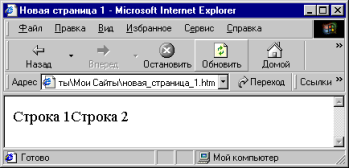
При написании скрипта, содержащего несколько команд document.write() подряд, при выводе в браузер текст окажется на одной строке

<script language="JavaScript">

document.write('Строка 1') ;

document.write('Строка 2') ;

</script>



Для размещения каждого куска текста в новом абзаце можно использовать 2 способа:

1. либо тег <p>, либо тег <br>, который включается в состав выводимой строки;

2. используя метод document.writeln().

Следующие примеры приведут к одинаковому результату:

<scriptlanguage="JavaScript">

document.write('Строка 1') ;

document.write('Строка 2') ;

</script>

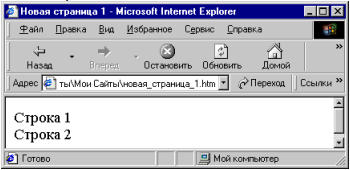
И

<script language="JavaScript">

document.writeln('Строка 1') ;

document.writeln('Строка 2') ;

</script>



*Объект location*

Объект location содержит информацию о местонахождении текущего документа, т.е. его интернет-адрес. Его также можно использовать для перехода на другой документ и перезагрузки текущего документа.

**Свойство Описание**

**Hash** Имя "якоря" в интернет-адресе документа, если оно есть.

**host** Имя компьютера в сети интернет вместе с номером порта, если он указан.

**hostname** Имя компьютера в сети Интернет.

**href** Полный интернет-адрес документа.

**pathname** Путь и имя файла, если они есть.

**port** Номер порта. Если не указан, возвращает номер 80 - стандартный порт, через который работает протокол http.

**protocol** Идентификатор протокола. Если не указан, возвращается "http:".

**search** Строка параметров, если она есть.

**Метод Описание**

**assign(url)** Загружает документ, адрес которого передан в качестве параметра. Поддерживается только IE начиная с 4.0

**reload()** Перезагружает документ с Web-сервера.

**replace(url)** Загружает документ, адрес которого передан в качестве параметра, и заменяет в списке истории Web-обозревателя адрес предыдущего документа адресом нового.

Пользуясь объектом location, можно загрузить другой документ на место текущего. Для этого просто необходимо присвоить значение нового интернет-адреса свойству href.

document.location.href = "http://www.---.ru";

Если вы хотите полностью заменить текущий документ, чтобы даже адрес его не появлялся в списке истории, воспользуйтесь методом replace:

document.location.replace("http://www.--.ru");

*Объект form*

Каждая форма в документе, определенная тегомНачало формы

, создает объект <form>, порождаемый объектом document. Cсылка на этот объект осуществляется с помощью переменной, определенной в атрибуте name тега<form>Начало формы

. В документе может быть несколько форм, поэтому для удобства ссылок и обработки в объект document введено свойство-массив forms, в котором содержатся ссылки на все формы документа. Ссылка на первую форму задается как document.forms[0], на вторую - document.forms[1] и т.д. Вместо индекса в массиве forms можно указывать имя формы. Например, если в документе присутствует единственная форма со значением атрибута name=form1, то любой из следующих операторов JavaScript содержит ссылку на эту форму:

document.forms[0];

document.forms["form1"];

document.form1;

Последний оператор возможен в силу того, что объект document порождает объект form (как и все остальные объекты, соответствующие элементам HTML страницы) и ссылку на него можно осуществлять по обычным правилам наследования языка JavaScript.

Все элементы формы порождают соответствующие объекты, подчиненные объекту родительской формы. Таким образом, для ссылки на объект text (с параметром name = text1) формы form1 можно пользоваться любым из нижеприведенных операторов:

document.forms[0].text1;

document.forms["form1"].text1;

document.form1.text1;

Кроме имени элементы формы, имеют свойство value, значение которого определяется смыслом атрибута value элемента формы. Например, для элементов text и textarea значением этого свойства будет строка содержимого полей ввода этих элементов; для кнопки подтверждения - надпись на кнопке и т.д. Обратиться к свойству value можно по тому же принципу, например:

document.form1.text1.value

**1.6. Обработкасобытий**

Использование языка JavaScript при обработке событий значительно расширило возможности языка HTML. Чаще всего программы создаются для обработки информации, вводимой пользователем в поля форм. Возможности управления элементами форм обеспечиваются главным образом за счет функций обработки событий, которые могут быть заданы для всех элементов формы. События делятся на несколько категорий:

• события, связанные с документами (события документа) - загрузка и выгрузка документов;

• события, связанные с гиперсвязью (события гиперсвязи) - помещение указателя мыши на гиперсвязь и активизация гиперсвязи;

• события, связанные с формой (события формы) –

o щелчки мыши на кнопках;

o получение и потеря фокуса ввода и изменение содержимого полей ввода, областей текста и списков;

o выделение текста в полях ввода и областях текста.

События, связанные с документами, возникают при загрузке и выгрузке документа, в то время как события гиперсвязей возникают при их активизации или при помещении на них указателя мыши. Чтобы обеспечить перехват события, необходимо написать функцию-обработчик события. В качестве обработчиков событий могут быть заданы целые функции языка JavaScript или только группы из одного или нескольких операторов. В таблице перечислены имена событий и условия их возникновения:

**Имя Атрибут HTML Условие возникновения события**

**события**

Blur onBlur Потеря фокуса ввода элементом формы

Change onChange Изменение содержимого поля ввода или области

текста, либо выбор нового элемента списка

Click onClick Щелчок мыши на элементе формы или

гиперсвязи

Focus onFocus Получение фокуса ввода элементом формы

Load onLoad Завершение загрузки документа

Unload onUnload Выгрузка текущего документа и начало загрузки

нового

MouseOver onMouseOver Помещение указателя мыши на гиперсвязь

MouseOut onMouseOut Помещение указателя мыши не на гиперсвязь

Select onSelect Выделение текста в поле ввода или области

текста

Submit onSubmit Передача данных формы

*Атрибут onClick*

Атрибут onClick может использоваться в следующих тегах HTML:

* *<*a herf=”url” onClick=" function()">…</a>
* *<*input type="checkbox" onClick=" function()"*>*
* *<*input type="radio" onClick="function()"*>*
* *<*input type="reset" onClick=" function()"*>*
* *<*input type="submit" onClick=" function()"*>*
* *<*input type="button" onClick=" function()"*>*

Операторы языка JavaScript, заданные в атрибуте onClick, выполняются при щелчке мыши на таких объектах как гиперсвязь, кнопка перезагрузки формы или контрольный переключатель. Для контрольных переключателей и селекторных кнопок событие Click возникает не только при выборе элемента, но и при разблокировании.

Разберем пример использования атрибута onClick для кнопок, определенных тегами

<input type="button">вконтейнере<form>…</form>:

…

<scriptlanguage="JavaScript">

functionbut1() {

alert("Вынажалипервуюкнопку");

}

functionbut2() {

alert("Вы нажали вторую кнопку");

}

</script>

…

<form>

<input type="button" value="Перваякнопка" onClick="but1()">

<input type="button" value="Втораякнопка" onClick="but2()">

</form>

…

*Работа с меню*

Список в форме задается с помощью объекта select, обработка событий выполняется с помощью следующих параметров: onChange - вызывается при изменении выбора; onBlur - вызывается при снятии фокуса с объекта; onFocus - вызывается при перемещении фокуса на объект.

Рассмотрим следующий пример:

…

<scriptlanguage="JavaScript">

functionselectBlur()

{

document.myForm7.myText.value="Вы нажали поле вне списка ";

}

functionselectFocus()

{

document.myForm7.myText.value="Вы нажали ту же кнопку ";

}

functionselectChange()

{

document.myForm7.myText.value="Вы нажали другую кнопку ";

}

</script>

…

<form name="myForm7">

<input type="text" name="myText" size=40 value="Город"><br>

<select name="script" multiple onBlur="selectBlur()" onFocus="selectFocus()" onChange="selectChange()">

<option value="town1" selected>Париж

<option value="town2">Лондон

<option value="town3">Рим

<option value="town4">Берлин

</select>

</form>

…

*Управление логикой программного кода при помощи событий*

В объектно-ориентированном программировании нет единой структуры управления работой программы. Есть независимые друг от 33 друга объекты. Когда пользователь щелкает, например, по ссылке на экране, браузер передает событие Click объекту, тега <a>. Для события “щелчок мыши” в этом объекте предусмотрен стандартный обработчик — он загружает в окно новый документ. Давайте попробуем “перехватить” это событие:

<ahref=”page1.htm” onClick="alert('Ходанет?')">документpage1</a>

Если щелкнуть по ссылке, на экране возникнет надпись: “Хода нет?”. Событие перехвачено, но, при закрытии окна alert, видим, что браузер по-прежнему грузит документ page1.htm. При помощи атрибута onClick мы установили в объекте, “отвод” на собственный обработчик. Но когда скрипт нашего обработчика выполнен, управление возвращается к стандартному обработчику, и это вызывает загрузку документа page1.htm.

Отключение стандартной обработки кодируется так:

<a href=page1.htm onClick="alert('Ходанет!');return false">документ page1</a>

Оператор return указывает возвращаемое функцией значение. Если ее операнд true, то документ загружается, если false, нет.

Подтверждение активизации гиперсвязи.

Аналогичный пример управления логикой программного кода при помощи событий рассмотрен и в следующем примере. Гиперссылка обычно всегда срабатывает по клику мыши, но иногда нужно, чтобы пользователь был уверен, что хочет перейти по ссылке в следующий документ. Для этого существует метод confirm(), который отображает на экране окно сообщения с кнопками "Ok" и "Cancel". Для перехвата события в теге <a href= ...> мы применим событие onClick. Рассмотрите пример подтверждения активизации гиперсвязи:

<a href="form.htm" onClick="returnconfirm('Вы действительно хотите перейти по ссылке?')"> Подтверждение активизации гиперсвязи</a>

*Определение событий формы*

Объект form имеет два обработчика событий: onSubmit и onReset. В эти обработчики событий, задаваемые в пределах дескриптораНачало формы

, добавляется группа операторов JavaScript или функция, управляющая формой.

Если вы добавите оператор (или функцию) вобработчикonSubmit, то он (или она) вызывается до отправки данных в сценарий CGI. Для того чтобы отменить отправку данных на обработку сценарием CGI, обработчик событий onSubmit должен возвратить значение false. Если же он возвращает значение true, то данные отправляются на сервер. В некоторых случаях необходимо добавить в форму кнопку reset, запускающую обработчик событий onReset.

Для формы одним из важных действий на странице является проверка правильности заполнения полей пользователем на машине клиента до пересылки их на сервер. В следующем примере разъясняется, как выполнять эту процедуру.

Рассмотрим скрипт, который будет проверять правильность заполнения формы. Необходимо проверить нет ли пустых строк и правильно ли введен e-mail:

<html>

<head>

<title>примерформы</title>

<script language="JavaScript">

functiondoSend(){

var v=document.user.e.value.indexOf("@",1)

if(document.user.f.value==""){

alert('Вы должны заполнить поле ФИО')

document.user.f.focus()

}

if(document.user.a.value==""){

alert('Вы должны заполнить поле адреса')

document.user.a.focus()

}

if(document.user.e.value==""){

alert('Вы должны заполнить поле e-mail')

document.user.e.focus()

}

if(v==-1){

alert('Адрес e-mail указан неверно')

document.user.e.select()

document.user.e.focus()

}

else

document.user.submit()

}

</script >

</head>

<body>

<p align="center"><font size=6>Данныеопользователе</font>

<formname="user">

<b> Пожалуйста, укажите данные о себе:</b>

<br>

ФИО<input type="text" name="f" size="30"><br>

Адрес<input type="text" name="a" size="35"><br>

e-mai<input type="text" name="e" size="30"><br>

<input type="button" value="Послать" onClick="doSend()">

<input type="reset" value="Отменить">

</ form>

</p>

</body>

</html>

*Вставка звука*

Если вам необходимо озвучить страницу, вот простейшая инструкция:

<bgsoundsrc="music/gimn.mid" loop=infinite>

С помощью JavaScript можно разнообразить страницы сайта. Пример скрипта проигрывания музыки при наведении на заголовок текста:

…

<script>

functionplayHome() {

document.all.sound.src = "music/file.mid"}

</script >

…

<bgsound id=sound>

<h1 onMouseover=playHome()>Заголовоксмузыкой</h1>

…

**Лекция 11. Объектная модель Dynamic HTML (динамический HTML)**

**План лекции:**

Объектная модель документа (DocumentObjectModel, DOM)

Дерево объектов в программной модели: Window, document, Объект navigator, Объект location, Объект history, Объект screen, Объект event

Обращение к свойствам таблицы стилей

Объединение JavaScript и CSS

Создание анимационных объектов

Слои. Позиционирование слоя

Свойство z-index

Свойства visibility и display

Динамическое управление слоями

Динамическое изменение цвета фона ячеек

DHTML (динамический HTML) – это набор средств, которые позволяют создавать более интерактивные Web-страницы без увеличения загрузки сервера. Другими словами, определенные действия посетителя ведут к изменениям внешнего вида и содержания страницы без обращения к серверу.

DHTML построен на объектной модели документа (DocumentObjectModel, DOM), которая расширяет традиционный статический HTML-документ. DOM обеспечивает динамический доступ к содержимому документа, его структуре и стилям. В DOM каждый элемент Web-страницы является объектом, который можно изменять. DOM не определяет новых тэгов и атрибутов, а просто обеспечивает возможность программного управления всеми тэгами, атрибутами и каскадными таблицами стилей (CSS).

Каждой гиперссылке, заголовку или текстовому параграфу можно присвоить имя, атрибуты стиля или цвета текста и указать это имя в сценарии, имеющемся на данной странице. Сценарий может быть написан на любом существующем скриптовом языке, но здесь подразумевается использование языка JavaScript. Элементы страницы впоследствии могут изменяться в результате определенного события, например, при наведении курсора мыши, при щелчке, нажатии клавиши на клавиатуре либо после истечения заданного промежутка времени. Изменяться может не только стиль и цвет текста, но и весь объект, текст или рисунок.

DHTML является продуктом взаимодействия трех технологий: языка HTML, каскадных таблиц стилей (CSS) и языка сценариев (JavaScript). Для предоставления сценариям доступа к HTML и CSS содержимое документа представляется в виде дерева объектов в программной модели.

**1. Window**

Объект window является вершиной иерархии объектной модели. Все остальные объекты являются дочерними или более отдаленными потомками объекта window. Браузер создает, как правило, единственный объект window, когда открывает документ в окне, однако если документ содержит фреймы (элементы frame и iframe), то дочерние объекты window создаются также для каждого фрейма. Доступ к дочерним окнам возможен через коллекцию frames родительского окна. Полный список свойств, методов, событий объекта window (как и у других объектов) можно получить так:

<script>

sprops="<h2>Свойстваобъекта window</h2>

for(props in window)

sprops+="<b>"+props+"</b><br>”

document.write(sprops);

</script>

Свойстваобъект window: parent;

• self;

• top;

• document;

• event;

• history;

• location;

• navigator;

• screen.

Методы:

•alert – выводсообщения;

•prompt – ввод данных;

•confirm – подтверждение;

•close – закрыть окно;

•open – открыть окно;

•blur – убрать фокус;

•focus – установить фокус;

• 40 navigate – загрузить страницу;

•scroll – развернуть страницу на указанный размер;

•setTimeout() – загрузить функцию через указанное время;•cleartimeout() – удалить загрузчик функции;

•showModalDialog – открыть модальное окно.

**2. document**

Рассмотрим некоторые свойства, коллекции и методы документа, загруженного в окно браузера.

Свойства:

•linkColor – цветссылок;

•fgColor – цветщрифта;

•bgColor – цвет фона.

Коллекции:

• all – все теги на странице;

• anchors – все ссылки;

• forms – коллекция форм.

Методы:

clear – очистить документ;

close – закрыть;

open – открыть;

write – записать в документ.

**3. Объект navigator**

Объект navigator содержит информацию о браузере.

Свойства:

•userAgent содержит строку, которая передается в HTTP заголовке на вебсервер с каждым запросом. Эта строка содержит информацию о самом браузере и его версии, операционной системе и ее версии и некоторую дополнительную информацию.

•appName содержит имя.

•appCodeName – "Mozilla" для большинства браузеров

•appVersion, appMinorVersion, platform, systemLanguage и другие содержат информацию о версии браузера и операционной системы, языковые настройки и другое.

**4. Объект location**

Объект location содержит полную информацию об адресе страницы.

Свойства:

•href содержит всю строку URL;

•protocol – протокол передачи данных;

•host – имя сервера + порт ( [www.ya.ru:80](http://www.ya.ru:80));

•hostname – имя сервера ( [www.ya.ru](http://www.ya.ru));

•port – порт;

•pathname – путь;

•search – строка запроса (?a=html&page=1);

•hash – закладка, ссылка внутри страницы ( #label1).

**5. Объект history**

Объект history содержит информацию об адресах, посещенных пользователем с момента открытия браузера.

Методы:

•back() – переход на предыдущую страницу. Эквивалентно нажатию кнопки "Назад" в браузере.

• forward() – переход на следующую страницу. Эквивалентно нажатию кнопки "Вперед" в браузере.

•go(n) – переход на n позиций в индексе

**6. Объект screen**

Объект screen содержит информацию о мониторе: ширина – width, высота – height, глубина цвета – colorDepth

**7. Объект event**

*Событие* – это сообщение, отправляемое в ответ на некоторое действие, например, завершение загрузки документа, перемещение или щелчок мыши, нажатие клавиши. Обработчик события – это, как правило, функция, написанная на языке сценариев (например, JavaScript), которая выполняет действия, когда произошло соответствующее событие.

Объект window.event создается браузером автоматически в момент возникновения. Следует подчеркнуть, что модель объекта event в Internet Explorer существенно отличается от используемой в других браузерах. Тем не менее, целый ряд свойств оказывается одинаковым во всех современных браузерах.

•type – тип события без приставки "on". Например, событие onclick имеет тип click.

•button – нажатая кнопка мыши (0 – ни одна, 1 – левая, 2 – правая, 3 – левая и правая, 4 – средняя, 5 – левая и средняя, 6 – правая и средняя, 7 – все три). Это свойство имеет смысл для событий мыши. В случае остальных событий имеет значение 0 независимо от действительного состояния кнопок.

clientX, clientY – координаты указателя мыши относительно верхнего левого угла клиентской области окна браузера.

• screenX, screenY - координаты указателя мыши относительно верхнего левого угла дисплея.

• keyCode – целочисленный код клавиши, которая вызвала событие клавиатуры. Коды клавиш различаются для различных аппаратных платформ (PC или Macintosh).

shiftKey, ctrlKey, altKey – истина (true), если во время события нажата также клавиша Shift, Ctrl или Alt, соответственно.

cancelBubble – запрещает (true ) или разрешает ( false, по умолчанию) передачу события вверх по иерархии от элемента-источника к родительскому элементу. Каждое событие происходит на единственном элементеисточнике. Если элементу-источнику не сопоставлена функцияобработчик, событие передается вверх по дереву элементов, пока не встретит функцию-обработчик или не достигнет корневого элемента. Если функция-обработчик найдена, но не содержит cancelBubble=true, то передача вверх продолжается, если содержит, то прерывается.

Отметим некоторые свойства, доступные в Internet Explorer:

•элемент-источник события доступен в IE как свойство srcElement, в Mozilla Firefox и других как target.

•cобытия onmouseover и onmouseout в InternetExplorer показывают свойства-объекты откуда ( fromElement ) и куда ( toElement). В Mozilla Firefox эти объекты являются свойством relatedTarget.

**8 Обращение к свойствам таблицы стилей**

Если известен идентификатор объекта (например, id1) и необходимо обратиться к свойству left, то можно обратиться так: id1.style.left=‖100 px‖

9. Свойства и методы управления содержимым

Свойства:

•innerText – заменяет текст, содержащийся в элементе;

• outerText – заменяет весь элемент;

• innerHTML – заменяет текст и код элемента;

• outerHTML заменяет весь текст и код.

Пример. Имеется заголовок третьего уровня

<h3 id="heading1">Старый заголовок</h3>

Далее в скрипте JavaScript даем команду

objHead1=document.all["Heading1"]

objHead1.innerText="Новыйзаголовок "

Методы:

•insertAdjacentText(положение, текст) – вставляет только текст;

•insertAdjacentHTML(положение, текст) – вставляет текст и HTML.

Вариантыположения: BeforeBegin, AfterBegin, BeforeEnd, AfterEnd.

Пример. objHead1.incertAdjacentText(―AfterBegin‖, ―Привет‖)

Пример реализации раскрывающихся списков с использованием приведенных свойств и методов.

<html>

<head>

<title>

Раскрывающийся список с использованием Inner

</title>

<script>

i1t=0

i2t=0

i3t=0

function qwe()

{

Obj=window.event.srcElement

if(Obj.id.length==2)

{ if(eval(Obj.id+"t==0"))

{

Obj.insertAdjacentHTML("BeforeEnd",document.all[Obj.id+"1"].innerHTML)

eval(Obj.id+"t=1") }

else

{

Obj.children[0].outerHTML=""

eval(Obj.id+"t=0")}

}

}

</script>

</head>

Раскрывающийся список с использованием Inner

< div id=i11 style="display:none"><ul><li>1.1<li>1.2<ul></div>

< div id=i21 style="display:none"><ul><li>2.1<li>2.2<ul></div>

< div id=i31 style="display:none"><ul><li>3.1<li>3.2<ul></div>

<ul onClick="qwe()">

<li id=i1>1

<li id=i2>2

<li id=i3>3

</ul>

</html>

**Объединение JavaScript и CSS**

1. *Пример изменения цвета текста.*

<html>

<head>

<title>Простая страница</title>

</head>

<body>

<h1 style="color:red">Дo6po пожаловать на нашу страницу!</h1>

<p>3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации. </p>

</body>

</html>

Данный пример применения CSS позволяет сделать заголовок красного цвета. Допустим, вы хотите, чтобы текст заголовка только тогда становился красным, когда пользователь наводит на него курсор. Этого можно добиться с помощью CSS и JavaScript.

Шаг 1. Удаление существующей информации о стиле

Это действие может показаться вам шагом назад, но оно действительно необходимо:

<html>

<head>

<title>Простаястраница</title>

</head>

<body>

<h1>Дo6po пожаловать на нашу страницу! </h1>

<p>3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации. </p>

</body>

</html>

Шаг 2. Добавление идентификатора

Поскольку вам нужно как-то обращаться к элементу, с которым будут производиться манипуляции, необходимо в тэг <h1> добавить атрибут id - это краткое обозначение, позволяющее указать нужный элемент:

<html>

<head>

<title>Простаястраница</title>

</head>

<body>

<h1 id="head1"> Дo6po пожаловать на нашу страницу! </h1>

<p>3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации. </p>

</body>

</html>

Шаг 3. Добавление обработчика событий

Следующий шаг — добавление обработчика событий. Этому действию соответствует событие onMouseover. Также следует указать имя функции, которая будет вызываться при выполнении события:

<html>

<head>

<title>Простаястраница</title>

</head>

<body>

<h1 id="head1" onMouseover ="colorchange ()">Дo6po пожаловатьнанашустраницу! </h1>

<p>3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации. </p>

</body>

</html>

Шаг 4. Написание сценария JavaScript

Вам потребуется единственная строка, состоящая из следующих частей:

 имя объекта на странице, с которым должен выполняться ваш сценарий - в данном случае head1;

 применяемый аспект JavaScript - в данном случае style;

 атрибут стиля, который будет изменяться - color;

 новое значение, принимаемое атрибутом стиля - red.

Соедините это, и получится следующая строка: Head1.style.color = "red"

Добавьте ее в функцию и сохраните файл. В окончательном варианте страница должна выглядеть так:

<html>

<head>

<title>Простаястраница</title>

<script language="JavaScript">

functioncolorchange()

{

head1.style.color = "red";

}

</script>

</head>

<body>

<h1 id="head1" onMouseover ="colorchange ()">Дo6po пожаловатьнанашустраницу! </h1>

<p>3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

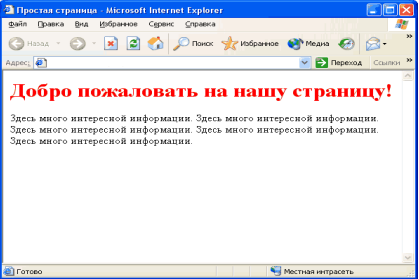
3десь много интересной информации. </p>

</body>

</html>

Откройте эту страницу в браузере и посмотрите, что происходит, когда вы наводите курсор на заголовок. Если все было сделано правильно, то цвет заголовка изменится.

Но обратите внимание, что цвет текста не становится прежним, когда вы убираете курсор с заголовка.



*2. Пример использования атрибута text-decoration.*

Используя в сценариях JavaScript атрибуты, название которых пишется через дефис, убирайте дефис и пишите оба слова слитно, причем второе слово должно начинаться с заглавной буквы. Таким образом, text-decoration в сценариях должно выглядеть как textDecoration.

<html>

<head>

<title>Простаястраница</title>

<script language="JavaScript">

functionaddunderline()

{

head.style.textDecoration = "underline";

}

functionremoveunderline()

{

head.style.textDecoration = "none";

}

</script>

</head>

<body>

<h1 id="head" onMouseover="addunderline()"

onMouseout="removeunderline()">Добро пожаловать на нашу страницу! </h1>

<p>3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации.

3десь много интересной информации. </p>

</body>

</html>

Необходимо обратить внимание на прописную букву в слове textDecoration — если все слово набрать в нижнем регистре, сценарий выполняться не будет. Теперь при наведении курсора заголовок станет подчеркнутым, а затем, если убрать курсор, вернется в прежнее состояние.

*3. Пример точного позиционирования текста*

Сначала необходимо рассмотреть, каким образом осуществляется позиционирование текста.

Наиболее часто применяются следующие атрибуты позиционирования: 

- position - имеет два интересующих нас значения: absolute и relative (по умолчанию значение static). Для значения absolute в качестве точки отсчета используется верхний левый угол окна браузера, и все параметры местоположения отмеряются от него. В свою очередь, для relative точкой отсчета является то место, в котором разместился бы элемент страницы, если бы не было представлено никакой информации о местоположении; 

- top - используется для указания вертикального смещения элемента от точки отсчета. Величина смещения может выражаться в различных единицах (пикселы, дюймы, сантиметры, миллиметры и т.п.). В наших примерах используются пикселы. Положительное значение top соответствует смещению элемента страницы вниз, в то время как отрицательное - по направлению к верхней границе окна браузера; 

- left - подобен атрибуту top, но применяется для указания горизонтального направления. Положительное значение соответствует сдвигу элемента вправо, отрицательное - влево.

<html>

<head>

<title>Простаястраница</title>

</head>

<body>

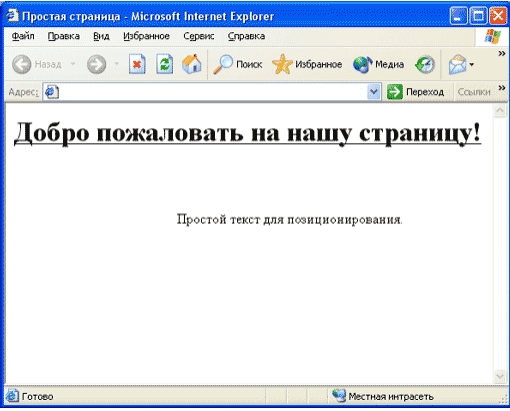
<h1 style=" text-decoration: underline">Дo6po пожаловатьнанашустраницу! </h1>

<pstyle="position:absolute; top:125px; left:200рх">Простойтекстдляпозиционирования.</p>

</body>

</html>

С помощью CSS текст расположен со значением absolute, то есть его положение отсчитывается от верхнего левого угла окна браузера. Значение атрибута top равно 125рх, таким образом, текст будет расположен на 125 пикселов ниже верхнего края страницы. Значение атрибута left равно 200рх, то есть текст будет сдвинут на 200 пикселов от левого края окна браузера.



**1.8. Создание анимационных объектов**

Анимация - это процесс «оживления» объекта

Анимация включает в себя две составляющие: расстояние между соседними кадрами, называемое скачком (jump), и временной промежуток между двумя последовательными скачками, называемый интервалом (interval). При больших скачках и длительных интервалах анимация выглядит медленной и грубой. Движение объектов кажется неестественным и воспринимается как мелькание. При малых скачках и кратких интервалах анимация выглядит более плавной, хотя, если чересчур увлечься, движение покажется нарочитым.

*1. Пример перемещения текста слева направо*

Сначала следует ввести текст в тэге, ограничивающем текст, добавить идентификатор id.

<html>

<head>

<title>Простаястраница</title>

</head>

<body>

<divid="anim">

Текст, шагоммарш!

</div>

</body>

</html>

Затем воспользуемся CSS, чтобы поместить текст в начальное положение:

<html>

<head>

<title>Простаястраница</title>

</head>

<body>

<div id="anim" style="position:absolute; left:10; top:10">

Текст, шагом марш!

</div>

</body>

</html>

Далее начинаем работать над сценарием JavaScript. Поскольку не нужно, чтобы текст вечно двигался вправо, надо предусмотреть возможность контролирования этого процесса. Чтобы запустить сценарий на выполнение только при условии, если текст находится, например, менее чем в 500 пикселах от левой границы экрана, удобнее всего воспользоваться оператором if. Для этого понадобится атрибут CSS pixelLeft.

<html>

<head>

<title>Простая страница</title>

<scriptlanguage="JavaScript">

functionmoveTxt()

{

if (anim.style.pixelLeft< 500)

{

}

}

</script>

</head>

<body>

<div id="anil" style="position:absolute; left:10; top:10">

Текст, шагом марш!

</div>

</body>

</html>

Теперь рассмотрим операторы, управляющие анимацией. Прежде всего нужно задать скачок. Каждый раз текст будет перемещаться вправо на 50 пикселей. Атрибут pixelLeft используется не только для определения положения текста, но и для изменения положения на 50 пикселей:

<html>

<head>

<title>Простаястраница</title>

<script language="JavaScript">

functionmoveTxt()

{

if (anim.style.pixelLeft< 500)

{

anim.style.pixelLeft +=50;

}

}

</script>

</head>

<body>

<div id="anim" style="position:absolute; left:10; top:10">

Текст, шагом марш!

</div>

</body>

</html>

Далее речь пойдет об интервале. Он задается с помощью метода setTimeout, позволяющего вновь запустить функцию после истечения определенного промежутка времени. Давайте установим интервал до повторного запуска функции moveTxt(), равным 5000 мс:

<html>

<head>

<title>Простаястраница</title>

<script language="JavaScript">

<!--Маскируемся!

functionmoveTxt()

{

if (anim.style.pixelLeft< 500)

{

anim.style.pixelLeft +=50;

setTimeout("moveTxt()", 5000);

}

}

</script>

</head>

<body>

<div id="anim" style="position:absolute; left:10; top:10">

Текст, шагом марш!

</div>

</body>

</html>

Процесс будет повторяться до тех пор, пока условие оператора if не станет ложным. Последнее, что нужно сделать, - запустить сценарий на выполнение. Для этого следует воспользоваться событием onLoad:

<html>

<head>

<title>Простаястраница</title>

<script language="JavaScript">

functionmoveTxt()

{

if (anim.style.pixelLeft< 500)

{

anil.style.pixelLeft +=2;

setTimeout("moveTxt()", 50);

}

}

</script>

</head>

<bodyonLoad="moveTxt()">

<div id="anil" style="position:absolute; left:10; top:10">

Текст, шагом марш!

</div>

</body>

</html>

*2. Пример движения текста по диагонали*

<html>

<head>

<title>Простаястраница</title>

<script language="JavaScript">

functionmoveTxt()

{

if (anim.style.pixelLeft< 500)

{

anim.style.pixelTop + = 2;

anim.style.pixelLeft +=2;

setTimeout("moveTxt()", 50);

}

}

</script>

</head>

<body onLoad="moveTxt()">

<div id="anim" style="position:absolute; left:10; top:10">

Текст, шагом марш!

</div>

</body>

</html>

**1.9. Слои**

*Позиционирование слоя*

Для создания слоев следует использовать тег<div> или <span>. Эти теги взаимозаменяемы и различаются лишь внешним видом в браузере. Если требуются отступы до и после текста, следует использовать элемент<div>. При размещении текста внутри параграфа применяется тег<span>.

Ниже перечислены наиболее важные атрибуты.

id Имя слоя, используемое для указания его в<script>

eft Позиция слоя по x координате

top Позиция слоя по у координате

position Задает относительную или абсолютную позицию относительно других объектов

z-index Позиция слоя при наложении нескольких объектов друг на друга

width Ширина слоя в пикселах или %

height Высота слоя в пикселах или %

bgColor Цвет фона слоя

background Картинка фона

src Внешний html документ, содержащийся в слое

Пример наложения текста:

<html>

<body>

Слой1 наверху

<div style="position:relative; font-size:50px; z-index:2; color: navy">

Слой1

</div>

<div style="position:relative; top:-55; left:5; color:orange; font-size:80px; z-index:1">

Слой 2

</div>

Слой 2 наверху

<div style="position:relative; font-size:50px; z-index:3; color: navy">

Слой 1

</div>

<div style="position:relative; top:-55; left:5; color:orange; font-size:80px; z-index:4">

Слой2

</div>

</body>

</html>

Тип позиционирования слоя определяется параметром position, положение элемента - двумя координатами top и left.

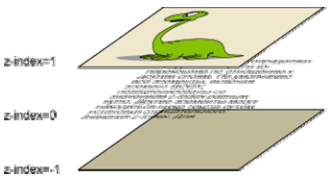
Кроме тегов<div> и <span>абсолютное позиционирование поддерживают следующие элементы:<applet>, <input>, <button>, <object>, <select>, <fieldset>, <iframe>, <table>, <img>, <textarea>.

Параметр position:relative используется для смещения слоя относительно родительского элемента. Установка этого значения не изменяет размещение элемента, но если установлены значения свойств top или left, то слой смещается от своего нормального положения в документе.

В то время как свойство position указывает тип системы координат, параметры top и left определяют точную позицию слоя. Значения этих параметров могут определяться в процентном отношении или пикселах, принимать положительные и отрицательные величины. Это дает возможность размещать контент выше или ниже на странице независимо от физической позиции кода HTML. То есть, в верхней части страницы можно поместить слой, который описан внизу HTML-документа.

*Свойство z-index*

Свойство z-index определяет порядок слоев, или их перекрытие по отношению к другим слоям. По умолчанию все слои позиционированы со значением z-index равным нулю. Другие слои могут размещаться ниже путем установки отрицательного значения z-index. Для слоев, у которых zindex не установлен, это значение назначается неявно в соответствии с их положением в документе. Поэтому слой, который помещен в документ позже, размещается выше остальных элементов, позиционированных ранее.



*Свойства visibility и display*

Для отображения или скрытия слоя используется свойство visibility. Он может принимать значения visible, установленное по умолчанию, для показа слоя, и hidden, которое его прячет. Например, скрытый блок текста можно оформить следующим образом:

<div style="visibility: hidden">Спрятанный слой</div>

При этом, когда используется данное свойство для скрытия элемента, соотвествующий данному элементу блок занимает прежнее положение на странице, но сома содержимо не отображается. Чтобы на странице не оставалось пустого блока, соотвествующего скрываемому элементу, можно использовать свойство display со значением none. Для отображения элемента display равно block.

*Динамическое управление слоями*

Сценарии JavaScript позволяют динамически управлять параметрами установленных слоев. Это позволяет получить такие эффекты, как скрытие и отображение слоя, изменение порядка отображения, перемещение слоя в окне браузера. Все эти эффекты достигаются с помощью изменения соответствующих стилевых параметров установленных слоев.

Для обращения к слоям из сценариев JavaScript, удобнее всего каждому слою дать собственное имя при помощи параметра id. Например:

<divid="div1">

…

</div>

Для того, чтобы скрыть отображение слоя div1, можно использовать следующую команду:

div1.style.visibility='hidden';

Для повторного отображения слоя следует выполнить следующее присвоение:

div1.style.visibility='visible';

Пример динамической смены слоев: в данном примере для отображения некоторого слоя следует нажать на соответствующую ссылку. Эту идею можно применить и для организации выпадающих меню.

<html>

<head>

<style type="text/css">

div {

position: absolute;

top: 20; left: 160;

visibility: hidden;

}

</style>

<script language="JavaScript">

functionshow\_d(d)

{

div1.style.visibility='hidden';

div2.style.visibility='hidden';

div3.style.visibility='hidden';

div4.style.visibility='hidden';

div5.style.visibility='hidden';

d.style.visibility='visible';

}

</script>

</head>

<body>

<a href="javascript:void(0)" onClick="show\_d(div1);">

показатьслой 1

</a><br>

<a href="javascript:void(0)" onClick="show\_d(div2);">

показатьслой2

</a><br>

<a href="javascript:void(0)" onClick="show\_d(div3);">

показатьслой3

</a><br>

<a href="javascript:void(0)" onClick="show\_d(div4);">

показатьслой4

</a><br>

<a href="javascript:void(0)" onClick="show\_d(div5);">

показать слой 5

</a><br>

<div id="div1">

<h3>Слой номер один</h3>

Некоторый текст, на слое расположенный. Его можно скрыть и показать. Текст может содержать несколько строк.

</div>

<div id="div2">

<h3>Слой номер два</h3>

Содержит свой текст. Если показывается, то текст на других слоях не виден.

</div>

<div id="div3">

<h3>Слой номер три</h3>

Тоже текст. При работе со слоями надо следить, чтобы текст одного слоя не "выглядывал" из-под другого слоя при самых различных размерах окна браузера и используемых шрифтах.

</div>

<div id="div4">

<h3>Слой номер четыре</h3>

Здесь нет текста.

</div>

<div id="div5">

<h3>Слой номер пять</h3>

И тут тем более нет.

</div>

</body>

</html>

*Динамическое изменение цвета фона ячеек*

Использование стилей и управление ими с помощью JavaScript позволяет менять вид ячейки "на ходу", при выполнении определенных условий, таких как наведение курсора на ссылку или саму ячейку.

Рассмотрим самый простой прием - цвет фона ячейки меняется, когда курсор мыши наводится на нее. Наведение мыши на область отслеживается событием onMouseover, а вывод мыши за ее пределы - событием onMouseout. Поскольку цвет фона меняется у той же самой ячейки, на которую наводим курсор мыши, то изменение стиля делается с помощью метода this.style.background.

…

<table width=60% border=1 cellspacing=0 cellpadding=4 bordercolor=#333333 align=center>

<tr>

<td align=center bgcolor=#cccccc

onMouseover="this.style.background='#ffcc33'"

onMouseout="this.style.background='#cccccc'"><a href="#">Пункт 1</a></td>

<td align=center bgcolor=#cccccc><a href="#"><Пункт 2></a></td>

</tr>

</table>

…

**Лекция 12. Модель DOM. Библиотека jQuery**

**План лекции:**

Структура документа DOM

Навигация по дереву документа

Создание новых узлов

Редактирование дерева документов

Работа с массивами элементов

Библиотека jQuery

Обработчик готовности документа

Создание элементов DOM

Работа с полученным набором значений

Манипулирование объектами на странице

Обработка событий

Скрытие и отображение объектов

Создания слайдера

**Структура документа DOM**

DOM – DocumentObjectModel – программный интерфейс XML (и HTML) документов.

Уровни DOM:

* Уровень 0. Включает в себя все специфические модели DOM, которые существовали до появления (см., например, лекцию 6) Необходимо обратить внимание, что эти модели формально не являются спецификациями DOM, опубликованными W3C, а скорее являются информацией о том, что существовало до начала процесса стандартизации.
* Уровень 1 (1998). Появление понятия узел.
* Уровень 2 (2000-2004). Поддержка пространства имен.
* Уровень 3 (с 2004).

DOM представляет документ как иерархию узлов (node). На вершине иерархии узел – document, который представляет собой весь документ. В качестве узлов представлено все содержимое документа: HTML-элементы, атрибуты и текст.

Для обсуждения свойств и методов будем использовать следующий фрагмент:

<div>

<ul id="components">

<li>HTML</li>

<li>CSS</li>

<li>JavaScript</li>

</ul>

</div>

**Навигация по дереву документа**

Навигацию можно начать с любого узла, у которого известен идентификатор.

var oList=document.getElementById("components")

Также в случае корректной страницы можно указать document.documentElement – самый верхний уровень.

Ссылка на родительский элемент:

var oParent=oList.parentNode

Дочерние элементы представляют собой коллекцию, на элемент которой ссылаются так:

var oItem1=oList.childNodes[1]

В данном случае это будет элемент <li>CSS</li>.

На первый и последний дочерний элемент ссылаются так: firstChild, lastChild.

Кроме этого можно сослаться на соседние узлы одного уровня:

var oPreviousSibling=oList.previousSibling

var oNextSibling=oList.nextSibling

**Созданиеновыхузлов**

Методы: createElement(), createTextNode().

Рассмотрим, как добавить к нашему списку элемент <li>XML</li>.

Первым делом создаем узел LI. Для этого даем команду

var oItem=document.createElement("LI")

Рекомендуется писать название тегов заглавными буквами.

Далее создаем содержимое узла:

var oText=document.createTextNode("XML")

Обращаю внимание на то, что узлы созданы, но еще не связаны ни между собой, ни с основным списком.

**Редактирование дерева документов**

Приведем некоторые методы для редактирования дерева документов.

Вставка: appendChild(), insertBefore().

Копирование: cloneNode().

Замещение: replaceChild().

Удаление: removeChild().

Присоединимузел oText кузлу oItem: oItem.appendChild(oText)

Теперь в памяти имеется веточка <li>XML</li>.

Добавим ее в конец узла oList:

oList.appendChild(oItem)

После последней команды в списке должен появиться элемент XML.

Для вставки в произвольное место можно использовать метод insertBefore(), например, так:

oList.firstChild.nextSibling.insertBefore(oItem)

Для копирования узла используется методb cloneNode(). Пример:

var oClone=oList.cloneNode(true)

Если установлен параметр true, то узел копируется со всеми потомками.

Далее скопированный узел можно добавить к списку: oList.appendChild(oClone)

Замена последнего элемента:

varoItem=document.createElement("LI")

oItem.appendChild(document.createTextNode("JS"))

oList.replaceChild(oItem,oList.lastChild)

Примерудаления:

var oRemovedItem=oList.removedChild(oList.lastChild).

**Работа с массивами элементов**

Свойство getElementById позволяет обращаться к элементу, у которого известен идентификатор. В случае, когда надо работать с определенными тегами, полезным является свойство getElementsByTagName, которое возвращает все указанные теги в виде массива.

Пример. Выделение красным цветом каждой второй строки таблицы с использованием метода getElementsByTagName.

<body>

<table border="1">

<tr><td>1</td><td>1</td><td>1</td></tr>

<tr><td>1</td><td>1</td><td>1</td></tr>

<tr><td>1</td><td>1</td><td>1</td></tr>

</table>

</body>  
<script>

a=document.getElementsByTagName("tr")

n=a.length

for(i=0;i<n;i=i+2)

{a[i].style.color="red"}

</script>

**Библиотека jQuery**

jQuery – библиотека JavaScript, которая позволяет получать доступ к любому элементу модели DOM, обращаться к атрибутам и содержимому, а также манипулировать ими. Кроме этого библиотека предоставляет удобный API для работы с технологией AJAX.

Библиотека была создана в 2006 году. В настоящее время практически каждый сайт использует данную библиотеку.

Подключив библиотеку jQuery вместо десятков команд на JavaScript можно написать несколько команд. Команды основаны на селекторах и классах CSS.

Библиотеку jQuery можно скачать с сайта http://jquery.com/. Размер библиотеки в **минимальном варианте составляет примерно 60 Кб.**

**Перед началом работы библиотеку надо подключить:**

**<script src="jquery.js" type="text/javascript" ></script>**

Вся работа с библиотекой ведется с использованием функции $. Общая идея использования: 1) выбирается элемент или группа элементов, 2) выполняются действия над выделенными элементами.

Пример. Выделение красным цветом каждой второй строки таблицы.

<script src="jquery.js" type="text/javascript" ></script>

<style>

.rrr{color:red}

</style>

<body>

<table border="1">

<tr><td>1</td><td>1</td><td>1</td></tr>

<tr><td>1</td><td>1</td><td>1</td></tr>

<tr><td>1</td><td>1</td><td>1</td></tr>

</table>

<script>

$("table tr:even").addClass("rrr")

</script>

Обращение к элементам

Рассмотрим на примерах варианты обращения. Обратите внимание, что все основано на синтаксисе CSS.

1. $(pa) выбирает все ссылки, расположенные в абзацах.

2. $(' #myId ') – выбор элемента с указанным идентификатором.

3. $('myClass') – использование классов.

4. $('body>div') – выбор элементов div, являющихся прямыми потомка-ми элемента body.

5. $(p:even) – четные абзацы.

6. Использование ссылок.

* a[href^=http://] – этому селектору соответствуют все ссылки, которые на-чинаются с символов http://, на это указывает символ ^/
* a[href$=.pdf] – ссылки заканчиваются на «pdf»
* a[href\*=ya.ru] – ссылка содержит упоминание ya.ru в произвольном месте.

7. Имеется возможность выбора определенных элементов при условии,

что они содержат другие элементы (в примере ниже двоеточие как в математике означает “такое что”). Например, селектор

li:has(a)

выбирает элементы<li>, которые содержат элемент <a>.

8. Выбор по позиции (некоторые варианты)

:first – первое совпадение на странице;

:last – последнее совпадение на странице;

:first-child – первый дочерний элемент;

:last-child – последний дочерний элемент;

:nth-child(n) – n-йэлемент;

:nth-child(even|odd) – четные или нечетные дочерние элементы;

:even – четные элементы;

:odd – нечетные элементы.

9. Выбор элементов на основе характеристик, не предусмотренных спецификацией CSS

:input – выбирает элементы формы;

:selected – выбранные элементы option;

:visible – выбор невидимых элементов.

**Обработчик готовности документа**

Первый способ – запуск обработчика, когда загрузилась страница

window.onload=function(){

$("table tr:nth-child(even) ").addClass("even");

};

Недостаток данного метода заключается в том, что ничего не будет из-менено, пока все не загрузится, включая картинки, т.е. пользователь может усмотреть как все изменяется, т.е. увидеть и исходный вариант, и преобразованный.

Второй способ – дождаться только загрузки структуры документа, а именно:

$(documen).ready(function(){

$("table tr:nth-child(even) ").addClass("even");

});

Т.е. читаем буквально – “Запуск функции после загрузки”. У этой конструкции имеется сокращенная форма:

$(function(){

$("table tr:nth-child(even) ").addClass("even");

});

**Создание элементов DOM**

Создание элемента

$(‘<div>Hi</div>’)

Пустой элемент можно создать так:

$(‘<div>’)

Можно использовать кавычки, можно апострофы. Но наличие тегов обязательно. Задавать просто текст нельзя. Сама по себе эта команда ничего не выведет, надо эту веточку опять же привязать к родительскому элементу. Делается это так:

$(‘<div>Hi</div>’).insertAfter('#a1')

Здесь a1 – это идентификатор объекта, после которого необходимо

вставить элемент.

**Работа с полученным набором значений**

В терминах jQuery эти наборы называют обернутыми.

1. Определение размера. Для этого можно использовать свойство length, а можно метод size().

Пример. Замена содержимого с идентификатором a1 на количество элементов класса b2

$("#a1").html($(".b2").size())

2. Полученный набор значений рассматривается как массив, и с ним мы соответственно и работаем. Получить любой элемент в обернутом наборе можно по индексу. Например, получить второй элемент в наборе всех гиперссылок:

$('a')[1].

Вместо индексов можно использовать метод get(), т.е. последнюю ко-манду можно записать так:

$('a').get(1).

При помощи метода get() можно получить обычный массив JavaScript, содержащий все обернутые элементы. Пример:

varallanchors=$('a').get().

3. Существуют специальные команды библиотеки, которые позво-ляют объединять различные полученные наборы, убирать определенные зна-чения по некоторым правилам

**Манипулирование объектами на странице**

Команды jQuery позволяют манипулировать свойствами, атрибутами, стилями и содержанием элементов.

Для обращения к свойствам и их значениям используются методы JavaScript, нет методов непосредственно библиотеки.

Пример. Изменяет свойство title у всех элементов класса b2

$('.b2').each(function(n){this.title="New "+n})

Здесь использована команда each(функция). Она выполняет обход всех элементов в наборе и вызывает для них функцию. В качестве параметра функции передается индекс элемента в наборе.

**Обработка событий**

1. Различные браузеры по-своему могут обрабатывать события, jQuery пытается сгладить эти неприятности. Поэтому обращаемся только к методам jQuery, а библиотека уже сама смотрит, что за браузер и применяет то или иное свойство.

Модель событий jQuery обладает следующими свойствами:

* поддерживает единый метод установки событий;
* позволяет устанавливать несколько обработчиков для события;
* использует стандартные названия типов событий;
* предоставляет единые методы отмены события и блокирования действий по умолчанию.

2. Подключение обработчиков. Рассмотрим пример функции, которая будет срабатывать при щелчке по любому рисунку:

$('img').blind('click',fuction(event){alert('Приветствую!');});

Для удаления обработчика используется команда unbind()

**Скрытие и отображение объектов**

1. Функции hide() и show(). С ними есть некоторые нюансы (напри-мер, showпоказывает изначальное состояние свойства display, поэтому при загрузке рекомендуют скрыть объект при помощи hide()), но в целом все как в примере.

Пример. $('#b3').hide()

А затем, скажем, по щелчку, $('#b3').show()

2. Имеются различные эффекты – раскрывающиеся списки, слайдеры,

увеличение рисунков (когда-нибудь надо разобрать подробнее)

**Создания слайдера**

Рассмотрим пример создания слайдера – блока с контентом, выпол-зающего по нажатию на конец закладки.

1. В текст вставляем следующий фрагмент:

<div id="container">

<div id="panel">

Содержимое панели

</div>

<p class="button"><a href="#" class="buttontext">Справка</a></p>

</div>

2. При помощи стилей изначально скрываем панель. Остальные параметры просто описательные – размер, цвет и т.д.

<!--Из-за наличия auto заодно и центрует в Mozila, т.к. auto Указывает, что размер отступов будет автоматически рассчитан браузером.-->

<!--buttontext можно вообще не задавать-->

<style>

.rrr {color:red}

#container {

margin: 0 auto;

width: 152px;

}

#panel {

background: #1ca8f6;

height: 230px;

display: none;

}

.button {

width: 152px;

height: 40px;

border-top: #333 dotted 1px;

text-align: center;

}

.buttontext {

font-weight: bold;

font-size: 1.2em;

text-shadow: 1px 1px 1px #666;

}

</style>

1. Описываем функцию

<script type="text/javascript">

$(document).ready(function(){

$(".buttontext").click(function(){

$("#panel").slideToggle("normal"); return false;

});

});

</script>

Комментарии

* Команда slideToggle позволяет чередовать два простых действия: показать и скрыть. normal– это скорость, означает, что панель выедет за 400 миллисекунд, есть еще fast, slow.
* Команда returnfalseозначает, что мы не даем перейти по ссылке, в качестве которой используется кнопка-закладка. То есть, она просто работает в качестве удобной ручки, за которую дергают, но не как непосредственно ссылка.

**Лекция 13. Язык PHP**- **средство создания динамических web-сайтов**

**План лекции:**

# Основы PHP, создание и выполнение простейших скриптов

Обработка данных HTML-форм средствами PHP

Работа с массивами и функциями в PHP

Работа со строками и регулярными выражениями в PHP

PHP (англ. Preprocessor оf Hypertext, произносится "пи-эйч-пи") представляет собой скриптовый язык программирования, созданный для реализации серверной части web-приложений, в частности, для генерации web-страниц, написания web-сервисов и организации работы с базами данных.

В настоящее время PHP является самым распространенным средством создания динамических web-сайтов, и популярность эта вполне заслужена. Благодаря своей простоте, богатой функциональности, достаточно высокой скорости выполнения, открытости исходных кодов и кроссплатформенности, за 20 лет PHP прошел путь от скромного шаблонизатора HTML-документов, написанного одним человеком, до мощного процедурного и объектно-ориентированного языка, стабильно входящего в пятерку ведущих средств программирования в мире и имеющего огромное сетевое сообщество поддержки.

На данный момент PHP поддерживается практически всеми хостинг-провайдерами в мире, входя в "стандартный набор" для создания сайтов "LAMP" (Linux, Apache, MySQL, PHP).

# 1. Основы PHP, создание и выполнение простейших скриптов

В целом синтаксис PHP подобен синтаксису языков C и C++. Некоторые элементы, такие как ассоциативные массивы и цикл foreach, позаимствованы из Perl. Для работы программы не требуется описывать какие-либо переменные, подключать используемые модули и т.п. Любая программа может начинаться непосредственно с оператора PHP. При этом интерпретатор исполняет код, находящийся внутри тега <?php ?>. Все, что находится вне ограничителей, выводится без изменений. Обычно это используется для вставки PHP-кода в HTML-документ:

<html>

<head>

<title>Тестируем PHP</title>

</head>

<body>

<?php

echo 'Hello, world!';

?>

</body>

</html>

Существует еще несколько вариантов написания тега <?php ?>, но они используются реже и зависят от настроек, сделанных в файле php.ini.

В документе может быть произвольное количество тегов <?php ?>, вложение этих тегов не допускается, в остальном же каких-либо ограничений нет, например, открывающая и закрывающая операторные скобки не обязаны находиться в одном   
теге <?php ?>:

<?php

$expression = true;

if ($expression == true) { ?>

Текст отображен, если условие истинно

<?php } else { ?>

А этот - если ложно

<?php } ?>

Код PHP не обязательно должен быть встроен в разметку HTML, разметку можно выводить в браузер стандартным оператором echo или функцией print:

<?php

echo '<!DOCTYPE html PUBLIC

"-//W3C//DTD HTML 4.01//EN">

<html><head>

<meta http-equiv="Content-Type"

content="text/html; charset=Windows-1251" />

<title>Пример</title>

</head>

<body>';

echo

'Формируем документ полностью программно';

echo '</body></html>';

?>

Этот код также показывает, какие теги HTML обязательно следует включать в формируемый программой на PHP документ. Как правило, это декларация <!doctype>, теги <html>, <head>, <title>, <body> и мета-тег, указывающий кодировку содержимого. В дальнейшем мы не везде будем использовать этот шаблон, но предполагается, что вы всегда делаете подобное оформление.

Как в C++ или Perl, в PHP требуется завершение операторов символом "точка с запятой" после каждой инструкции.

PHP предоставляет богатые возможности комментирования кода - действуют как однострочные и многострочные комментарии языков C/C++, так и комментарии в стиле языка Perl:

<?php

echo "Строка1";

// однострочный комментарий c++

/\* многострочный комментарий

c++ \*/

echo "Строка2"; # комментарий в стиле Perl

?>

Однострочные комментарии действуют до конца строки или до завершающего тега ?>, если он находится в той же строке.

Следует также понимать, что блоки HTML-разметки, находящиеся вне тега <?php ?>, таким способом не комментируются. В формате HTML есть собственный тег многострочного комментария <!-- -->.

Язык PHP предоставляет 4 скалярных типа данных:

* boolean - логический, принимает значения true (истина) и false (ложь);
* integer - целочисленный, обычно 4-байтовый, но размерность зависит от платформы;
* float - число с плавающей точкой, обычно представлено с двойной точностью, соответствующей типу double в Си-подобных языках;
* string - строка, на большинстве платформ размер ограничен 2 гигабайтами, по умолчанию символу соответствует 1 байт, т.е. в текущих версиях PHP нет встроенной поддержки Юникода.

Также определены два смешанных типа:

* array - массив пар "ключ" - "значение", если ключи не указаны, соответствует массиву в языке C++ (нумерация элементов с нуля);
* object - объект, используется в объектно-ориен­ти­рованном программировании с помощью классов.

Два специальных типа, resource и null служат, соответственно, для создания ссылок на внешние ресурсы и представления переменной без значения.

Существенно то, что программист на PHP*не устанавливает* тип переменной каким-либо специальным оператором; обычно это делает сам интерпретатор PHP во время выполнения программы в зависимости от контекста, в котором используется величина. Кроме того, имена всех переменных начинаются с символа $:

$a = 1; //целочисленная величина

$b = 'String'; /\* строка \*/

$f = 3.14; #вещественное число

echo "a=$a,b=$b,f=$f";

В остальном правила именования переменных похожи на правила из других Си-подобных языков: имя переменной чувствительно к регистру символов, начинается с буквы или символа подчеркивания и состоит из букв, цифр и символов подчеркивания в любом количестве.

По умолчанию, как и в примерах выше, переменные присваиваются *по значению*, т.е. при выполнении оператора присваивания значение оригинального выражения, стоящее справа от знака "=", копируется в эту переменную. Однако в PHP есть и присваивание *по ссылке*, позволяющее записать в одну переменную адрес другой:

$var = 'test';

$var2 = &$var;

Теперь $var2 *указывает* на $var и изменение значения одной переменной означает изменение значения другой:

$var = 'quest';

echo "$var2"; //quest

Кроме того, в PHP существует понятие "переменная переменных", позволяющее использовать значение одной переменной в качестве имени другой и создающее дополнительные возможности косвенной адресации данных:

$var=1;

$list = 'var';

$$list = 2;

echo "$var"; //2

Таким образом, конструкция $$Имя\_переменной означает "взять переменную, имя которой получено из переменной $Имя\_переменной".

В PHP нет необходимости инициализировать переменные, хотя это желательно.

Для описания *констант* служит конструкция define, в которой нужно указать имя и значение константы:

define ("TITLE","Мойблог");

define ("LIMIT",'10');

define('DB\_HOST', 'localhost');

Имя константы должно соответствовать тем же правилам, что и другие имена в PHP. По принятому соглашению, имена констант пишутся в верхнем регистре символов.

Арифметические операции и оператор присваивания реализованы в PHP аналогично языку C++.

Определен оператор присваивания (=) и 5 бинарных арифметических операций - умножение (\*), деление (/), сложение (+), вычитание (-), взятие остатка от деления (%). В отличие от C++, деление целых чисел не дает целого результата:

<?php

$a= 3/5;

echo "$a"; //0.6

?>

Как и в C++, разрешены префиксный и суффиксный инкремент и декремент, присваивание "цепочкой" и "на лету":

<?php

$a=$b=0;

echo "$a,$b "; //0,0

$b=6+($a=5);

echo "$a,$b"; //5,11

?>

Результаты операций сравнения интерпретируются и как значения 1/0, и как true/false:

<?php

$a=$b=0;

echo $a==$b; //1

if (($a==$b)==true) echo "1"; //1

?>

Кроме обычного сравнения ==, есть *сравнение идентичности* ===, возвращающее true, если операнды равны по значению и имеют один тип:

<?php

$a=1; $b='1';

echo $a==$b?"1":"0"; //1

echo $a===$b?"1":"0"; //0

?>

Сравнение идентичности "не равно по типу и значению" записывается как !==, в отличие от обычного отношения неравенства, которое обозначается != или <>.

Как видно из последнего листинга, в языке PHP также определена тернарная условная операция, соответствующая своему аналогу из C++.

Логические операции, служащие для записи составных условий, также соответствуют Си-подобным языкам: унарный оператор отрицания (знак !), логическое "и" (&&), логическое "или" (||), но можно использовать и логические операции and, or, xor ("и", "или", "исключающее или"), отличающиеся более низким приоритетом (табл. 2).

Поразрядные операции, выполняемые над отдельными битами своих операндов, традиционны для Си-подобных языков и приведены в табл. 1.

Таблица 1

Побитовые операторы PHP

|  |  |  |
| --- | --- | --- |
| Пример | Название | Результат |
| $a & $b | И | Устанавливаются только те биты, которые установлены и в $a, и в $b |
| $a | $b | ИЛИ | Устанавливаются те биты, которые установлены в $a или в $b |
| $a ^ $b | Исключающее ИЛИ | Устанавливаются только те биты, которые установлены либо только в $a, либо только в $b, но не в обоих одновременно |
| ~ $a | Отрицание | Устанавливаются те биты, которые не установлены в $a, и наоборот |
| $a<< $b | Сдвиг влево | Все биты переменной $a сдвигаются на $b позиций влево (каждая позиция - умножение на 2) |
| $a>> $b | Сдвиг вправо | Все биты переменной $a сдвигаются на $b позиций вправо (каждая позиция - деление на 2) |

Разрешена операция списка (символ "запятая").

Определены операторы new и -> для создания экземпляра класса и доступа к членам класса.

Теперь отметим основные операции PHP, не имеющие аналогов в C++.

Унарный оператор подавления ошибки @ может использоваться перед любым выражением, имеющим значение. Например, код:

<?php

$x=0; $a=1;

@($a/=$x);

echo "$a"; //$a не будет выведено

?>

не приведет к генерации сообщения об ошибке, хотя значение $a окажется не определено. При подобном отключении стандартных сообщений об ошибках, для собственной их обработки можно использовать сообщение из предопределенной переменной PHP с именем $php\_errormsg, а в файле php.ini следует включить настройку track\_errors = On (по умолчанию установлено значение Off).

Унарная операция выполнения `командная строка` заставляет PHP выполнить все, заключенное между обратными апострофами, как команду, вводимую в командной строке текущего сервера. Вывод команды будет значением выражения.

<?php

$dir=`dir \\*.\*`;

echo "<pre>$dir</pre>";

?>

Специально для работы со строковыми данными определены два оператора конкатенации (сцепления) строк:

* оператор . (точка) возвращает строку, представляющую собой соединение левого и правого аргумента;
* оператор присваивания с конкатенацией .= присоединяет свой правый аргумент к левому.

Обычно эти операторы используются с командой echo или print для формирования сложных строк, добавляемых в формируемую HTML-страницу:

<?php

$pi = round(pi(),2);

$string = 'Число pi='.$pi.'<br>';

$e = round (exp(1),2);

$string .= 'Число e='.$e;

echo $string;

?>

Обратите внимание, что в выводимую строку $string добавлен тег разрыва строки <br>, ведь обычный перевод строки в HTML-коде не имеет значения для браузера.

В табл. 2 приведены приоритеты (старшинство) операторов PHP в порядке возрастания. Операторы, размещенные в одной строке таблицы, имеют одинаковые приоритеты, и порядок их выполнения определяется, исходя из их ассоциативности. Как и в языке C++, при необходимости приоритеты операторов в арифметическом или логическом выражении изменяются с помощью круглых скобок.

Таблица 2. Приоритеты операторов PHP по возрастанию

|  |  |
| --- | --- |
| Ассоциативность | Операторы |
| Cлева направо | , |
| Cлева направо | or |
| Cлева направо | xor |
| Cлева направо | and |
| Справа налево | = += -= \*= /= .= %= &= |= ^= ~= <<= >>= => |
| Cлева направо | ?: |
| Cлева направо | || |
| Cлева направо | && |
| Cлева направо | | |
| Cлева направо | ^ |
| Cлева направо | & |
| Не определена | == != === |
| Не определена | <<= >>= |
| Cлева направо | <<>> |
| Cлева направо | + - . |
| Cлева направо | \* / % |
| Справа налево | ! |
| Справа налево | ~ ++ -- (int) (float) (string) (array) (object) (bool) @ |
| Cлева направо | [] |
| Не определена | new |
| Не определена | () |

Приведение типов данных можно выполнить и в стиле языка C++, например:

<?php

$d = 123.45;

$i=(int)$d;

echo "$i"; //123

?>

Часто применяются стандартные функции для проверки существования переменных:

* bool isset ($var) - вернет true, если переменная с именем $var существует;
* void unset ($var) - удаляет переменную с именем $var;
* bool empty($var) - вернет false, если объект $var существует и не пуст, иначе true. Пустыми считаются значения "" (строка, не содержащая символов), 0, 0.0, "0", NULL, false, array() и var $var; (переменная объявлена, но не имеет значения в классе).

Рассмотрим основные инструкции (операторы) языка PHP, акцентируя внимание на тех, что отличаются от языка C++.

Инструкцией может быть присваивание, вызов функции, повтор кода (цикл), сравнение, пустой оператор. После инструкции ставится точка с запятой. Инструкции могут быть объединены в блоки заключением их в фигурные скобки. Блок инструкций также является инструкцией.

Все виды условного оператора if реализованы аналогично языку C++. При использовании фигурных скобок для выделения ветвей алгоритма, в составном условном операторе ключевое слово elseif может быть записано как без пробела, так и с пробелом в виде elseif. Работает оператор-переключатель switch в стиле языка C++.

Кратко напомним об основных циклах в C++:

* while (условие) { тело\_цикла; }
* do { тело\_цикла; } while (условие);
* for (выражение1; условие; выражение2)

{ тело\_цикла; }

Все циклы работают в PHP так же, как их аналоги в C++, в теле циклов разрешены операторы break и continue.

В отличие от C++, оператор break принимает необязательный натуральный аргумент, показывающий, выполнение какого количества вложенных структур необходимо прервать:

<?php

$k=0;

for ($i=1; $i<=10; $i++) {

echo '<br>$i='.$i;

for ($j=1; $j<=10; $j++) {

echo ' $j='.$j;

$k++;

if ($k==30) break 2;

//выход сразу из 2 циклов!

}

}

?>

Аналогичный аргумент оператора continue позволяет пропустить оставшуюся часть итерации на нескольких уровнях вложенных циклов:

<?php

for ($i=1; $i<=4; $i++) {

echo '<br>$i='.$i;

for ($j=1; $j<=4; $j++) {

if ($j>$i) continue 2;

echo ' $j='.$j;

}

}

?>

Этот код напечатает текст:

$i=1 $j=1

$i=2 $j=1 $j=2

$i=3 $j=1 $j=2 $j=3

$i=4 $j=1 $j=2 $j=3 $j=4

Цикл foreach предназначен для работы с ассоциативными массивами.

PHP предлагает альтернативный синтаксис для некоторых управляющих структур, а именно, if, while, for, foreach и switch. В каждом случае основной формой альтернативного синтаксиса является изменение открывающей фигурной скобки на двоеточие (:), а закрывающей скобки на конструкции endif;, endwhile;, endfor;, endforeach; или endswitch; соответственно. Приведемпример:

<?php

for ($i = 1; $i <= 10; $i++) :

echo 'Число '.$i.'<br>';

endfor;

if ($i == 11) :

echo '$i==11';

endif;

?>

Рекомендуемый по итогам главы раздел для чтения в стандартной справке - "Справочник языка", до главы "Управляющие конструкции" включительно.

**Пример 1.** Реализуем скрипт, выводящий в браузер таблицу умножения с гибкой настройкой ширины ячеек $w, зависящей от заданного числа столбцов таблицы $cols.

<?php

$rows = 9; $cols = 9;

$w = floor(100/$cols).'%';

echo '<table border="1" cellpadding="2"

cellspacing="0">'."\n";

for ($i=1; $i<=$rows; $i++) {

echo '<tr>'."\n";

for ($j=1; $j<=$cols; $j++) {

echo '<td align="center" '.

'width="'.$w.'">'.($i\*$j).

'</td>'."\n";

}

echo '</tr>'."\n";

}

echo '</table>';

?>

Примечания:

1. В коде не выводятся необходимые заголовки файла HTML, добавьте их самостоятельно.

2. Использование одинарных и двойных кавычек в операторе echo не случайно. Оно будет подробно разобрано в следующей главе.

# 2. Обработка данных HTML-форм средствами PHP

Нашим первым скриптам не хватает возможностей по обмену данными с пользователем. Перечислим основные способы, с помощью которых PHP-скрипт может получить или передать данные.

1. Через URL-адрес HTTP-запроса к файлу скрипта (метод GET). Например, в строке запроса [http://www.myserver.ru/ search.php?name=Ivan&n=5](http://www.myserver.ru/%20search.php?name=Ivan&n=5) к скрипту с именем search.php, находящемуся в корневой папке web-сервера с адресом [www.myserver.ru](http://www.myserver.ru/), передаются две переменных:

* переменная с именем name, имеющая значение Ivan;
* переменная с именем n, имеющая значение 5.

Из кода скрипта все данные, переданные методом GET, содержатся в глобальном массиве с именем $\_GET и непосредственно доступны для обработки. В нашем случае элемент глобального массива $\_GET['name'] имеет значение Ivan, а элемент $\_GET['n'] - значение 5:

<?php

echo $\_GET['name']; //выведет Ivan

$m = $\_GET['n']+1;

echo '<br>'.$m; //выведет с новой строки 6

?>

Для выполнения кода примера следует создать в корневой папке сервера файл get.php и вызвать его из браузера URL-адресом вида <http://localhost/get.php?name=Ivan&n=5>.

Непривычная конструкция со строкой 'name' вместо числового индекса элемента массива, как мы узнаем в главе 4, используется в PHP повсеместно.

2. Данные могут быть переданы скрипту через тело HTTP-запроса (метод POST). В этом случае пользователь не видит переданных данных непосредственно. Как правило, методом POST передаются достаточно большие объемы данных, которые пользователь ввел или выбрал в форме, созданной средствами HTML этим же или другим скриптом. Максимальный размер данных, которые можно передать через форму методом POST, определяется значением post\_max\_size в файле php.ini. В коде скрипта все данные, переданные методом POST, хранятся в глобальном массиве с именем $\_POST и также непосредственно доступны для обработки.

Приведем пример, в котором скрипт с именем form1.php передает данные (имя пользователя) скрипту form2.php, показывающему, какое имя было введено в форму первого скрипта. Для простоты оба файла не формируют заголовки HTML и содержат только латиницу.

**Файл form1.php**

<form method="post" action="form2.php">

Your name:

<input type="text" name="name" value="">

<input type="submit" value="OK">

</form>

Обратите внимание, что в теге HTML, открывающем форму, как минимум должен быть указан метод передачи данных (GET или POST) и адрес скрипта, принимающего данные. Также все элементы формы, передающие данные, должны быть поименованы атрибутом name="имя\_переменной". Элемент массива $\_GET или $\_POST с индексом, соответствующим указанному имени, будет доступен скрипту, принимающему данные.

**Файл form2.php**

<?php

echo 'Your name='.$\_POST['name'].

'<br><a href="form1.php">Back</a>';

?>

Кроме переданного имени, скрипт выводит с новой строки ссылку "Back", позволяющую вернуться к первому скрипту (если оба они находятся в одной папке на сервере).

Сразу же отметим, что такой подход - *непосредственное* отображение или использование скриптом данных, переданных через форму HTML, в реальных приложениях абсолютно недопустим и крайне опасен! Попробуйте, например, ввести в форму первого скрипта значение <b>Ivan (перед именем открывается, но не закрывается тег жирного начертания текста). Весь вывод скрипта form2.php, следующий за $\_POST['name'], окажется отображен с жирным начертанием текста - а ведь это еще самое безобидное "нарушение", которое может допустить пользователь при вводе данных. Поэтому существенная часть главы будет посвящена приемам правильной *фильтрации* данных пользователя, делающей их относительно безопасными.

Для полноты картины перечислим другие способа обмена данными между скриптом и "внешним миром". Они будут изучены позднее.

3. Чтение и запись данных с помощью текстовых или бинарных файлов (в терминологии PHP - "плоских", flat) будут изучены в главе 6.

4. Обмену данными с СУБД MySQL посвящена глава 8. Отметим, что в PHP возможна поддержка альтернативных СУБД, мало отличающаяся от MySQL.

5. Скрипт может сохранять и читать информацию с помощью cookie-файлов на стороне клиента. Данные будут доступны через глобальный массив $\_COOKIES, работы с которым мы коснемся в главе 9.

6. Скрипт, при включенных в браузере cookies, может использовать сессии браузера, доступные через глобальный массив $\_SESSION. Работа с сессиями также будет рассмотрена в главе 9.

Прежде, чем перейти к подробному разбору обработки пользовательских данных, внесем окончательную ясность в вопрос с апострофами ' и двойными кавычками " в строковых константах, например, выводимых оператором echo.

Основное правило очень простое:

* строки в двойных кавычках " *оцениваются* - т.е. в них производится подстановка имен переменных и спецсимволов, заимствованных из языка C++;
* строки в апострофах ' печатаются "как есть", без подстановки имен переменных и управляющих последовательностей.

Сравним вывод этих двух операторов echo:

<?php

$a=1e3; $s = "String<br>\n";

echo "$a,$s"; //выведет 1000,String

echo '$a,$s'; //выведет $a,$s

?>

Обратите внимание, что первый оператор добавит разрыв строки как в отображаемый пользователю на экране код HTML (за счет добавления тега <br>), так и в исходный текст документа, увидеть который можно из контекстного меню "Просмотр кода страницы" вашего браузера (за счет физического разрыва строки управляющей последовательностью сим­волов \n).

К сожалению, при небрежном форматировании кода, PHP не всегда способен корректно подставить значения переменных или элементов массива в строки, заключенные в двойные кавычки:

<?php

$a='10 apples';

echo "<br>I have $atoday";

echo "<br>I have {$a}today";

?>

Первый оператор echo не сработает корректно и приведет к выводу в браузер предупреждения (если эта возможность не отключена в настройках php.ini). Суть дела в том, что если интерпретатор встречает знак доллара ($) внутри строки в двойных кавычках, он захватывает настолько много символов, насколько возможно, чтобы сформировать правильное имя переменной, а символ 't', без пробела следующий за значением переменной $a, является допустимым в имени объекта. Если вы хотите точно определить конец имени, заключайте имя переменной в фигурные скобки, как и сделано во втором операторе echo.

Если применение фигурных скобок кажется неудобным, во многих случаях можно заменить строку, заключенную в двойные кавычки, несколькими строками в апострофах, соединенных оператором сцепления строк "точка":

<?php

$a='10 apples';

echo '<br>I have '.$a.'today';

?>

Невнимательность с кавычками приводит к ошибке Parseerror, поэтому привыкайте следить за ними тщательно.

В табл. 3 приведены основные управляющие последовательности PHP, применимые к строкам в двойных кавычках.

Таблица 3 Управляющие последовательности в строках

|  |  |
| --- | --- |
| 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) |
| \v | Вертикальная табуляция (VT или 0x0B (11) в ASCII) (с версии PHP 5.2.5) |
| \f | подача страницы (FF или 0x0C (12) в ASCII) (с версии PHP 5.2.5) |
| \\ | Обратная косая черта |
| \$ | Знак доллара |
| \" | Двойная кавычка |
| \[0-7]{1,3} | Последовательность символов, соответствующая регулярному выражению символа в восьмеричной системе счисления |
| \x[0-9A-Fa-f]{1,2} | Последовательность символов, соответствующая регулярному выражению символа в шестнадцатеричной системе счисления |

Вторая возможность PHP, которая понадобится нам здесь и далее, касается разделения кода на несколько файлов, ведь большие файлы, содержащие и формы ввода, и код для их обработки, и вывод результатов обработки данных в браузер, крайне неудобны в отладке. На "базовом" PHP без классов было бы затруднительно в полной мере реализовать современную парадигму программирования MVC (модель - вид - контроллер), но привыкнуть разделять большой скрипт на небольшие файлы, решающие конкретные задачи или обрабатывающие конкретные запросы, нужно обязательно, тем более что PHP предоставляет для этого удобнейшие средства.

Оператор include*включает* в код и *выполняет* указанный файл.

Файлы включаются, исходя из пути, указанного в операторе, или, если путь не указан, используется путь, заданный в директиве include\_path файла php.ini. Если файл не найден в include\_path, оператор include попытается проверить папку, в которой находится включающий скрипт и текущую рабочую папку перед тем, как выдать ошибку.

Конструкция include выдаст предупреждение (Warning), если не сможет найти файл. Путь может быть указан как *абсолютным* (начинающимся с буквы диска или с символа "\" в Windows или с символа "/" в Unix-системах), так и *относительным* к текущей папке, в которой находится скрипт, использующий оператор include. Относительный путь может также начинаться со ссылки на текущую папку (./) или родительскую (../). В случае явного указания пути к файлу, директива include\_path будет проигнорирована.

Когда файл включается, его код наследует ту же область видимости переменных, что и строка, на которой произошло включение. Все переменные, доступные на этой строке во включающем файле будут также доступны во включаемом файле. Однако все функции и классы, объявленные во включаемом файле, будут доступны в глобальной области видимости. Если включение происходит внутри функции включающего файла, тогда весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции.

В качестве простейшего примера приведем мини-скрипт, состоящий из двух файлов.

**Файл config.php**

<?php

$n = 5;

?>

**Файл index.php**

<?php

include 'config.php';

for ($i=1; $i<=$n; $i++) {

echo '<br>'.$i.' fruit(s)';

}

?>

Поскольку имя файла index.php обычно принято по умолчанию для главного файла скрипта, документы следует разместить в отдельной папке fruits, вложенной в корневую папку сервера, и выполнить в браузере URL-адресом вида <http://localhost/fruits/>

Выражение include\_once также включает и выполняет указанный параметром оператора файл. Его поведение идентично include, с той разницей, что если код из файла уже был включен в скрипт, он не будет включаться повторно.

Выражение include\_once применятся в тех случаях, когда некоторый файл может быть включен и выполнен более одного раза. Оператор поможет избежать проблем с переопределением функций, переменных и т.д., что является предпочтительным для сложных скриптов.

Выражение require работает идентично include, за исключением того, что если файл не найден или ошибочен, оно сгенерирует фатальную ошибку выполнения скрипта уровня E\_COMPILE\_ERROR и остановит выполнение скрипта. Выражение include только выдало бы предупреждение E\_WARNING, которое позволяет скрипту продолжить выполнение.

Выражение require\_once идентично require за исключением того, что PHP проверит, включался ли уже данный файл, и, если да, не будет включать его еще раз. Этот оператор представляется наиболее надежным и полезным в сложных скриптах.

Перейдем к задаче фильтрации пользовательских данных, полученных методом GET или POST. Обычно требуются следующие этапы обработки:

1. Удаление лишних разделителей в пользовательском вводе (пробелов, переводов строки). Реализуется стандартной функцией trim и с помощью регулярных выражений.

2. Удаление из ввода пользовательской разметки HTML-тегами (функция strip\_tags) или превращение ее в печатаемые символы HTML (функция htmlspecialchars; при ее применении, например, символ "<", прочитанный из поля ввода формы, превратится в HTML-сущность "&lt;" и не сможет изменить разметку создаваемого скриптом документа).

3. В PHP версий ниже 5.4.0 есть несколько настроек, отвечающих за отображение апострофов и кавычек (традиционное название - "магические кавычки"). При неудачном стечении обстоятельств, например, двойная кавычка из пользовательского ввода может "экранироваться" и превратиться в \", испортив внешний вид текста. Стандартной функции, решающей все проблемы с кавычками, нет, напишем для этой цели собственную функцию с именем magic.

4. Если это уместно, над полученными из массива $\_GET или $\_POST элементами формы следует выполнить явное преобразование типов данных, например, для целого значения - функцией intval, для строки, передаваемой в SQL-запрос к базе данных - функцией mysql\_real\_escape\_string и т.д.

5. В профессиональных скриптах чаще всего необходимо проверить URL-адрес, с которого была отправлена форма - ведь ничто не мешает злоумышленнику написать скрипт, который будет отправлять данные нашему скрипту программно, перегружая таким образом сервер или стремясь подобрать пароль пользователя. Простейшие проверки такого рода включают в себя контроль значения $\_SERVER['HTTP\_REFERER'] и/или IP-адреса пользователя. К сожалению, HTTP - открытый протокол, и подделать можно все поля его заголовка, включая адрес страницы, с которой отправлена форма и IP-адрес отправителя.

Поэтому достаточно надежной защитой от программной атаки на HTML-формы в настоящее время можно считать лишь отправку вместе с данными формы некоторой "капчи" (captcha) - задачи, которую с легкостью может решить человек, но со значительно меньшей вероятностью может решить программа (например, распознавание текста, выведенного поверх изображения). О генерации captcha-защиты говорится в главе 7.

Из сказанного можно сделать вывод, что обработка данных, полученных скриптом из HTML-формы - непростая задача. Например, в показанном выше скрипте form2.php, следовало бы, как минимум, выполнить следующее:

<?php

$name = "";

if (isset($\_POST['name']))

$name = trim(htmlspecialchars(

$\_POST['name']));

echo 'Your name='.$name.

'<br><a href="form1.php">Back</a>';

?>

и это без учета возможных проблем с кавычками, так что ввод имени '"Ivan"' вполне может привести к выводу Your name=\'\"Ivan\"\'.

Реализуем скрипт по обработке введенных в форму данных, который учитывает высказанные соображения.

Чтобы не писать в каждом модуле однотипного и громоздкого кода по обработке параметров, напишем "универсальный" модуль-обработчик параметров params.php, а в конкретных модулях для работы с ним нужно будет определить в массиве с именем $params разрешенные имена получаемых "извне" переменных, например:

$params = array('action','name','number');

и подключить сам модуль строкой кода

require\_once ('params.php');

Все остальное он сделает сам, в том числе создаст "пустые" переменные для случаев, если какая-то из разрешенных величин не передана в скрипт.

**Файл params.php**

<?php

if (isset($params) && !empty($params)) {

while (list($num,$var) = each($params)) {

if (isset($\_POST[$var]))

$$var = trimall(htmlspecialchars(

magic($\_POST[$var])));

else if (isset($\_GET[$var]))

$$var = trimall(htmlspecialchars(

magic($\_GET[$var])));

else if (isset($\_SESSION[$var])) {

$$var = trimall(htmlspecialchars(

magic($\_SESSION[$var])));

unset ($\_SESSION[$var]);

}

else $$var = '';

}

}

?>

Для создания объектов с именами из массива $params использована *переменная переменных* $$var, разрешена передача данных как через методы GET и POST, так и через сессию. Передача данных через массив $\_SESSION разрешена однократно, переданные данные сразу удаляются. Это может пригодиться в будущем, например, при "возврате" в форму введенных в другом модуле данных.

Функция trimall будет удалять из ввода все лишние разделители - лишние пробелы в начале и конце строки (стандартной функцией trim), а остальное, в том числе, переводы строк в многострочном поле или лишние пробелы между словами, выполнят регулярные выражения в стандартной функции preg\_replace (см. главу 5).

Функция magic разберется с настройками кавычек, две из них можно установить в "безопасные" для внешнего вида данных нулевые значения стандартной функцией ini\_set, а третья обрабатывается стандартной функцией get\_magic\_ quotes\_gpc, при необходимости снимая "экранирование" кавычек методом stripslashes (с версии PHP 5.4.0 значение, возвращаемое методом get\_magic\_quotes\_gpc, всегда равно нулю, так как "магические кавычки" упразднены).

Кроме того удобно, когда все служебные функции объединены в один модуль, который другие модули подключают через директиву require\_once. Мы свой модуль служебных функций назовем function.php и пока включим туда реализации функций trimall и magic.

**Файл functions.php**

<?php

function trimall ($string) {

//Удалить лишние пробелы и переводы строк

return preg\_replace("/(^\s\*)|(\s\*$)/","",

preg\_replace("/\s+/"," ",trim($string)));

}

functionmagic($path) {

//Убрать возможные проблемы с кавычками

ini\_set('magic\_quotes\_runtime', '0');

ini\_set('magic\_quotes\_sybase', '0');

return (@get\_magic\_quotes\_gpc()=='1'?

stripslashes($path):$path);

}

?>

Основной файл скрипта с именем index.php будет как выводить форму, так и обрабатывать полученные от пользователя данные. Для экономии места, исключим из листинга заголовки и завершающие теги HTML-файла, также придется пожертвовать нормальным выравниванием строк, чтобы листинг поместился по ширине печатной страницы пособия.

С другой стороны, форму сделаем такой, чтобы в ней были реализованы все основные элементы HTML, служащие для ввода данных, т.е. однострочное (<inputtype="text">) и многострочное (<textarea>) поля ввода, группа радиокнопок (<inputtype="radio">), выпадающий список (<select>), переключатели (<inputtype="checkbox">) и кнопка для отправки данных (<inputtype="submit">).

Внимательное изучение этого листинга также поможет вам понять, как выровнять элементы формы с помощью табличных тегов и как обеспечить "возврат" в форму набранных пользователем значений. Одна из самых недопустимых вещей (особенно в громоздких формах) - заставлять пользователя повторно обрабатывать несколько полей ввода, если ранее он неправильно заполнил (или пропустил) одно из них и скрипт-обработчик "не принял" информацию.

**Файл index.php**

<?php

require\_once 'functions.php';

$params = array ('name','gender','education',

'php','cpp','java','text','action');

require\_once 'params.php';

?>

<form action="index.php" method="post">

<table align="center" border="0"

cellpadding="4" cellspacing="0" width="70%">

<caption>Заполнитеанкету</caption>

<tr>

<td>Имя:</td>

<td>

<input type="text" name="name" maxlength="40"

size="40" value="<?php echo $name; ?>">

</td>

</tr>

<tr>

<td>Пол:</td>

<td>

<input name="gender" value="m" type="radio"

<?php echo $gender=='m'?'checked':''; ?>

>Мужской

<input name="gender" value="f" type="radio"

<?php echo $gender=='f'?'checked':''; ?>

>Женский

</td></tr><tr>

<td>Образование:</td><td>

<select name="education" size="1">

<option value="none">Невыбрано

<option value="middle"

<?php if ($education=='middle')

echo 'selected'; ?>>

Среднее или ниже

<option value="high"

<?php if ($education=='high')

echo 'selected'; ?>>

Высшее или ученая степень

</select></td></tr><tr>

<td>Какие языки программирования Вы знаете:

</td><td>

<input type="checkbox" name="php" value="1"

<?php if ($php==1) echo 'checked'; ?>>PHP

<input type="checkbox" name="cpp" value="1"

<?php if ($cpp==1) echo 'checked'; ?>>C++

<input type="checkbox" name="java" value="1"

<?php if ($java==1) echo 'checked'; ?>>Java

</td></tr><tr>

<td>Ваш комментарий:</td>

<td>

<textarea name="text" rows="4" cols="40">

<?php if (!empty($text)) echo "$text"; ?>

</textarea></td></tr><tr><td>&nbsp;</td>

<td>

<input type="submit" name="action"

value="Отправить">

<input type="reset" value="Отмена">

</td></tr></table></form>

<?php

if (!empty($action)) {

//Блок обработки данных,

//у нас они просто выводятся под формой

echo 'Имя="'.$name.'"<br>'.

'Пол="'.$gender.'"<br>'.

'Образование="'.$education.'"<br>'.

'PHP="'.$php.'"<br>'.

'C++="'.$cpp.'"<br>'.

'Java="'.$java.'"<br>'.

'Текст=<br>'.nl2br($text);

}

?>

Внешний вид заполненной формы приложения показан на рис. 2.

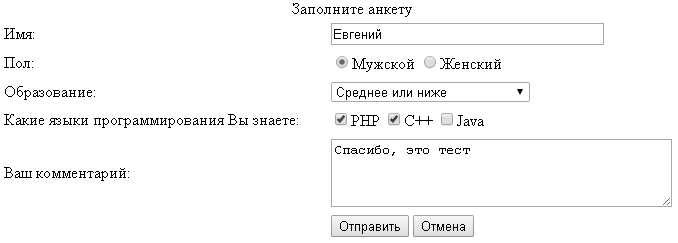


Рис. 2. Форма обработки данных пользователя

Отдельно рассмотрим типовую ситуацию, когда обработчиком данных является *другой* скрипт, например, имеющий имя action.php (это имя следует прописать в атрибуте action тега <form>), а не тот же самый, в котором реализован код формы. В этом случае следует либо обработать пользовательские данные и вернуться на страницу исходного скрипта директивой header:

header('Location: index.php');

либо, если данные содержат ошибки, вывести соответствующее сообщение и ссылку для возврата к исходному скрипту:

if (!isset($name) or empty($name)) {

echo '<div align="center">'.

'Ошибка! Вы не ввели имя<br>'.

<a href="index.php">Вернуться</a></div>';

}

Обратите внимание, что эти две возможности *несовместимы* - директива header (отправка HTTP-заголовка) работает только если клиенту еще не передавались никакие данные, т.е. она должна быть первой в выводе, перед ее вызовом не должно быть выведено никаких HTML-тегов, пустых строк, других заголовков и т.п. Таким образом, применять ее можно, например, внутри ветви условного оператора, соответствующей правильным значениям данных:

if (isset($name) and !empty($name)) {

/\* обработкаданных \*/

header('Location: index.php');

}

else {

/\* сообщения об ошибках \*/

}

В директиве header можно использовать не только фиксированное имя адресуемого скрипта, но и запись $\_SERVER['HTTP\_REFERER'], соответствующую адресу страницы, с которой была отправлена форма.

Остается открытым вопрос - как в этом случае обеспечить возврат введенных пользователем данных из action.php в скрипт index.php? Для этого существуют, по крайней мере, три возможности:

1. В случае правильной обработки данных некоторые из них можно вернуть как параметры метода GET по ссылке, переданной директиве header:

header( 'Location: index.php?name='.

urlencode($name));

URL-кодирование строки методом urlencode необходимо во избежание проблем с не латинскими символами в данных, например, с кириллицей в имени.

Аналогично можно вернуть данные, "зашив" их в атрибут href тега <a> ссылки "Вернуться".

2. При большом объеме данных бывает целесообразней сделать форму со скрытыми полями <inputtype="hidden"> и возвращать ее данные методом POST:

if (!isset($name) or empty($name) or

strlen($name)<4) {

echo '<div align="center">

Ошибка! Вы не ввели имя от 4 символов

<form action="index.php" method="post">

<input type="hidden" name="name"

value="'.$name.'">

<input type="submit" value="Вернуться">

</form>';

3. Если оба варианта неприемлемы, используйте глобальный массив $\_SESSION и функцию session\_start() в начале всех скриптов (см. главу 9).

Для закрепления материала главы почитайте следующие разделы стандартной справки: "Приступая к работе" - "Простой учебник" - "Работа с формами"; "Безопасность" - "Волшебные кавычки".

**Пример 2.** Скрипт calcme.php проверяет правильность выполнения пользователем несложных арифметических расчетов на сложение, вычитание и умножение чисел от 1 до 10. Скрипт контролирует правильность выполнения последнего задания, если таковое было, и предлагает следующее. Попробуйте модифицировать скрипт так, чтобы количество предлагаемых заданий было ограничено, например, значением 10, и дополнительно накапливалась статистика о количестве правильных решений примеров.

<?php $result = $solution = '';

if (isset($\_POST['result']) and

isset($\_POST['action'])) {

$result = intval($\_POST['result']);

$solution = intval($\_POST['solution']);

echo '<p>'.$solution.'? '.

($result == $solution ?

'Верно!' : 'Неверно!').'</p>';

} ?>

<form action="calcme.php" method="post">

<p><?php

$n1 = rand (1,10); $n2 = rand (1,10);

$ops = array ('\*','+','-');

$op = rand (0,count($ops)-1);

$res = $n1 \* $n2;

if ($ops[$op]=='+') $res = $n1+$n2;

else if ($ops[$op]=='-') $res = $n1-$n2;

echo 'Сколько будет '.$n1.' '.

$ops[$op].' '.$n2.'?';

?></p>

<input type="text" name="solution"

maxlength="4" size="4" value="">

<input type="hidden" name="result"

value="<?php echo $res; ?>">

<input type="submit" name="action"

value="Отправить"></form>

Примечания:

1. В коде не выводятся необходимые заголовки файла HTML, добавьте их самостоятельно.

2. Примеры на деление не реализованы во избежание получения вещественных ответов, однако, генерация значения $n2, кратного $n1, устранила бы эту проблему.

Вид формы скрипта после решения очередного примера показан на рис. 3.

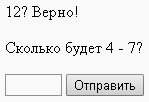


Рис. 3. Форма скрипта для решения примеров

# 3. Работа с массивами и функциями в PHP

Концепция массивов в PHP существенно глубже и мощнее, чем в C++, что естественно для интерпретируемого языка. Прежде всего, все массивы в PHP *ассоциативны*, т.е. являются хэшами, парами "ключ"=>"значение". Ключ можно понимать как аналог индекса элемента в классических языках, но, в отличие от них, ключ не обязан быть целочисленным значением. В примере ниже создаются несколько массивов:

<?php

$a1 = array ();

$a2 = array (1,2,3);

$a3 = array (0=>1, 'test'=>2, 100=>3);

?>

Здесь массив $a1 пуст (не содержит элементов), массив $a2 содержит элементы, для которых не заданы ключи, а только значения - в этом случае ключами, как и в C++, будут целые числа 0, 1 и 2, а массив $a3 содержит как числовые ключи 0 и 100, так и строковый ключ 'test'. При этом размерность массива равна 3, а его элементы могут быть записаны как $a3[0], $a3[100] и $a3['test'] со значениями 1, 3 и 2 соответственно. Запись последнего элемента напоминает нам обращение к элементам глобальных массивов $\_GET и $\_POST из главы 3, и действительно, $\_GET и $\_POST точно так же ассоциативны, как любые другие массивы в PHP.

При создании массива возможно указать ключ только для некоторых элементов и пропустить для других:

<?php

$array = array( "a", 'b', 6 => "c" );

echo $array[0].' '.$array[1].' '.$array[6];

?> //abc

В этом коде элементы массива будут иметь ключи 0, 1 и 6. Из примера также видно, что доступ к элементам массива осуществляется синтаксисом $массив[ключ], напоминающим нам о C++.

Из сказанного следует, что типовой цикл for применим только к массивам без явно указанных ключей, таким, как $a2:

<?php

$a2 = array (1,2,3);

for ($i=0; $i<count($a2); $i++)

echo $a2[$i].' '; //1 2 3

?>

Стандартная функция count позволяет узнать текущее число элементов в любом массиве, как с явно указанными ключами, так и без них.

В общем случае сканирование массива выполняется в специально предназначенном для этой цели цикле foreach:

foreach ($массив as $ключ=>$значение) {

/\* используем переменные $ключ и $значение,

полученные из очередного элемента массива \*/

}

Цикл foreach работает только с массивами и объектами и будет генерировать ошибку при попытке использования с переменными других типов или неинициализированными переменными. В качестве примера распечатаем значения элементов массива $a3 в виде $a3[ключ]=значение:

<?php

$a3 = array (0=>1, 'test'=>2, 100=>2);

foreach ($a3 as $key=>$value) {

echo '$a3['.$key.']='.$value.' ';

} //$a3[0]=1 $a3[test]=2 $a3[100]=2

?>

Если ключи элементов при обработке не нужны, заголовок цикла можно записать и в виде

foreach ($a3 as $value)

Аналогично можно обращаться и со встроенными массивами. Выведем все элементы глобального массива $\_SERVER, содержащего информацию о сервере и среде исполнения:

<?php

foreach ($\_SERVER as $key=>$value)

echo "key=$key,value=$value<br>";

?>

Другим способом задать массив было бы присваивание имени массива с квадратными скобками нужных значений элементов:

$a4[] = "Fisrt";

$a4[] = "Second";

//$a4[0] === "First", $a4[1] === "Second"

Многомерные массивы, например, матрицы, формируются как "массивы массивов", т.е. каждый элемент массива также является массивом. В качестве примера покажем создание и вывод в табличном виде матрицы целых чисел:

<?php

$matrix = array (

array (1,2,3),

array (4,5,6),

array (7,8,9,10)

);

for ($i=0; $i<count($matrix); $i++) {

echo '<br>';

for ($j=0; $j<count($matrix[$i]); $j++)

echo $matrix[$i][$j].' ';

}

?>

Число элементов в третьей строке не совпадает с остальными, но это не приведет к возникновению ошибки, просто элемент $matrix[2][3] будет существовать, а, например, $matrix[1][3]- нет:

echo isset($matrix[2][3]); //1

echo isset($matrix[1][3]); //пусто

Как и переменную, массив (или его элемент) можно удалить функцией unset, после чего он должен стать неопределенным.

Существенно, что к массивам применимы операции отношения и оператор сложения:

<?php

$a = array (1,2); $b = array (2,1);

$c = array (1,2);

echo $a==$b?'Yes':'No'.'<br>'; //No

echo $a==$c?'Yes':'No'; //Yes

?>

При сложении массивов оператором "+", если в левом массиве есть тот же ключ, что в правом, будет использовано значение из левого массива, иначе будет добавлено значение из правого с сохранением ключа:

<?php

$a = array (1,2,3);

$b = array (1=>4, 2=>5, 3=>array (6, 7));

$c = $a + $b;

foreach ($c as $i) echo $i.' ';

?> //1 2 3 Array

В PHP определено множество удобных функций для работы с массивами, они решают многие из типовых задач, которые в других языках нужно отдельно программировать. Полный перечень функций можно изучить по стандартной справке, а здесь укажем лишь несколько основных возможностей, приведя имена стандартных функций без описания параметров:

* in\_array - проверяет, присутствует ли в массиве значение;
* array\_chunk - разбивает массив на несколько частей указанного размера;
* array\_slice - выбирает указанную часть массива;
* array\_merge - сливает элементы нескольких массивов в один;
* array\_push - добавляет один или несколько элементов в конец массива;
* array\_shift - извлекает первый элемент массива;
* array\_pop - извлекает последний элемент массива;
* array\_unshift - добавляет один или несколько элементов в начало;
* array\_diff - возвращает разницу между массивами;
* sort - сортирует массив;
* usort - сортирует массив, с возможностью подключения собственной функции сравнения элементов;
* array\_search - осуществляет поиск указанного значения в массиве;
* array\_replace - замена элементов массива элементами других переданных массивов.

Вторая часть главы будет посвящена приемам разработки собственных функций на PHP. В главе 3 мы уже применяли пользовательские функции, работа с которыми похожа на использование подпрограмм-функций в языках C и C++. С другой стороны, есть и очевидные отличия - в соответствии с общей идеологией языка, для функций и их формальных параметров типы данных не указываются в явном виде:

<?php

function my\_sum ($a,$b) {

$result = $a+$b; return($result);

}

?>

Вызвать эту функцию мы могли бы кодом

$result = my\_sum (2,3);

По сути дела, функции - это участки кода, ассоциированные с определенным именем. Как и в других языках, они нужны для эффективного повторного использования кода или отделения части кода по соображениям его лучшего структурирования.

Имена функций следуют тем же правилам, что и другие метки в PHP. Корректное имя функции начинается с буквы или знака подчеркивания, за которым следует любое количество букв, цифр или знаков подчеркивания. Существенное отличие от C++ - имя функции должно быть уникальным с точностью до регистра букв. Это означает, что f1 и F1 будут считаться одинаковыми именами функций, и следующий код приведет к ошибке:

<?php

function f1 () { return 1; }

function F1 () { return 2; }

echof1();

?>

С другой стороны, мы можем давать функциям такие же имена, как и переменным в программе - конечно, без знака $ в начале. Список аргументов, если он указан, состоит из нескольких перечисленных через запятую переменных, каждую из которых мы должны будем задать при вызове функции. Если у функции не должно быть аргументов, то, как и в C++, следует оставить пустые скобки после ее имени.

Функции не обязаны быть определены до их использования, исключая тот случай, когда они определяются условно.

В случае, когда функция определяется в зависимости от какого-либо условия, обработка описания функции должна предшествовать ее вызову. Показанный ниже код приведет к ошибке в случае, если скрипту не передана методом GET величина с именем ok:

<?php

if (isset($\_GET['ok'])) {

function my\_function() {

return 'Data posted';

}

}

echomy\_function();

?>

В примере ниже показано, как одна и та же функция getversion() может возвращать различные результаты в зависимости от версии PHP, полученной как значение переменной $phpver:

<?php

$phpver = phpversion();

if ($phpver[0]==="5") {

function getversion() { return "PHP5"; }

}

if ($phpver[0]==="4") {

function getversion() { return "PHP4"; }

}

if ($phpver[0]==="3") {

function getversion() { return "PHP3"; }

}

echo getversion();

?>

Наконец, в третьем примере предварительно проверяется отсутствие в пространстве имен скрипта ранее определенной функции с именем my\_function:

<?php

if (!function\_exists("my\_function")) {

function my\_function () { return 1; }

}

echo my\_function();

?>

Для успешного применения функций важно понимать правила, связанные с *областью видимости* различных объектов в PHP-скрипте.

Область видимости переменной - это контекст, в котором она определена. В большинстве случаев все переменные PHP имеют одну *глобальную* область видимости, охватывающую также включаемые (include) и требуемые (require) файлы. Например:

<?php

$a = 1;

include 'b.inc';

?>

Здесь переменная $a будет доступна внутри включенного скрипта b.inc.

Однако определение (тело) пользовательской функции задает *локальную* область видимости данной функции. Любая используемая внутри функции переменная по умолчанию ограничена локальной областью видимости функции:

<?php

$a = 1; //глобальная область видимости

functiontest() {

echo $a;

$a = 2; //создастся локальная переменная $a

}

test(); //пусто

echo $a; //1

?>

Вызов функции test() не сгенерирует никакого вывода, кроме предупреждения "Undefined variable: a", поскольку выражение echo указывает на локальную версию переменной $a, а в пределах этой области видимости ей еще не было присвоено значение. А вот оператор echo, находящийся в последней строке программы, напечатает значение 1, сохранившееся у глобальной переменной.

Таким образом, переменная, описанная вне функции, *невидима* внутри нее, в отличие от большинства других языков. На самом деле, эта "странность" приносит больше выгоды, чем неудобств, так как является своего рода "защитным механизмом", не позволяющим случайно перепутать локальную и глобальную переменные.

Если глобальная переменная будет использоваться внутри функции, она должна быть объявлена глобальной внутри определения функции с помощью ключевого слова global:

<?php

$a = 1; //глобальная область видимости

functiontest() {

global $a;

echo $a;

$a = 2; //присвоим 2 глобальной переменной

}

test(); //1

echo $a; //2

?>

Использование ключевого слова global вне тела функции, часто применявшееся в ранних версиях PHP, в настоящее время не рекомендуется (кроме применения в файле, который включается внутри функции). Вместо этого переменную можно добавить в специальный *суперглобальный* массив с именем $GLOBALS. Ключом элемента массива будет имя, а значением - содержимое глобальной переменной:

<?php

$GLOBALS['a'] = 1;

function test() {

echo $GLOBALS['a'];

$GLOBALS['a'] = 2;

}

test(); //1

echo $GLOBALS['a']; //2

?>

Массив $GLOBALS существует и доступен в любой области видимости.

Другой важной особенностью области видимости переменной является *статическая переменная*. Подобно своему аналогу из C++, она существует только в локальной области видимости функции, но не теряет своего значения, когда выполнение программы выходит из этой области:

<?php

function cnt() {

static $a = 0; echo $a; $a++;

}

for($i=0; $i<3; $i++) echo cnt(); //012

?>

Локальная переменная $a будет проинициализирована только при первом вызове функции, а каждый последующий вызов будет выводить значение $a и увеличивать его на еди­ницу.

Сами функции (как и классы) в PHP имеют глобальную область видимости - они могут быть вызваны вне функции, даже если были определены внутри и наоборот. PHP не поддерживает перегрузку функций, также отсутствует возможность переопределить или удалить объявленную ранее функцию.

Требования к списку параметров функции типичны для процедурно-ориентированных языков программирования - функция может принимать информацию в виде списка аргументов, который является списком разделенных запятыми выражений. Аргументы вычисляются слева направо. PHP поддерживает передачу аргументов по значению (по умолчанию), по ссылке, а также значения аргументов по умолчанию:

<?php

function test($a,$b=1) {

$c = $a + $b;

echo $c;

}

test (1); //2

test (1,2); //3

?>

При первом вызове функции test параметру $b будет назначено значение по умолчанию 1, второй вызов задает явное значение этого параметра. Ошибкой был бы вызов функции с тремя параметрами или совсем без параметров. Параметры, имеющие значения по умолчанию, следует располагать правее в списке, чем параметры, таких значений не имеющие.

Функции также могут работать со списками аргументов переменной длины, для обращения к ним служат стандартные методы func\_num\_args() и func\_get\_args():

<?php

function sum() {

$numargs = func\_num\_args();

$summa = 0;

$arglist = func\_get\_args();

for ($i=0;$i<$numargs;$i++)

$summa += $arglist[$i];

return $summa;

}

echo sum (1,2,3).' '; //6

echo sum (4,5).' '; //9

echosum (); //0

?>

По умолчанию аргументы функций передаются *по значению* (изменение значение аргумента внутри функции теряется по выходе из нее). Для передачи аргумента *по ссылке*, разрешающей функции его модифицировать, достаточно указать оператор "адрес" (символ амперсанда &) перед именем аргумента в описании функции:

<?php

function cnt(&$a) {

echo $a; $a++;

}

$cnt = 0;

for($i=0; $i<3; $i++) echo cnt($cnt); //012

?>

Как видно из примера вычисления суммы, значения, вычисляемые внутри функций, возвращаются при помощи необязательного оператора возврата return. Возвращаемые значения могут быть любого типа данных, в том числе, массивами или объектами. Возврат приводит к завершению выполнения функции и передаче управления обратно к той строке кода, в которой была вызвана данная функция. Функция не может вернуть несколько значений, но аналогичного результата можно добиться, возвращая массив:

<?php

function small\_numbers() {

return array (0, 1, 2);

}

list ($zero, $one, $two) = small\_numbers();

echo "$zero,$one,$two"; //0,1,2

?>

При вызове из глобальной области видимости оператор return завершит выполнение всего файла скрипта. Если текущий файл скрипта был подключен с помощью include или require, тогда управление возвращается к файлу, который вызывал текущий. Более того, если текущий файл скрипта был подключен с помощью include, то значение, переданное оператором return, будет возвращено в качестве значения вызова include.

Для того чтобы функция возвращала результат по ссылке, нужно использовать & и при описании функции, и при присвоении переменной возвращаемого значения:

<?php

function & series ($item) {

$value = $item.' '.$item;

return $value;

}

$a = &series (1); echo $a; //1 1

$a = &series ($a); echo '<br>'.$a; //1 1 1 1

?>

Возвращаемой величиной $value в этом случае может быть только переменная, но не выражение, поскольку требуется получение ссылки.

Как и для переменных, для функций возможны *косвенные* вызовы, для чего PHP поддерживает концепцию *переменных функций*. Это означает, что если к имени переменной присоединены круглые скобки, интерпретатор ищет функцию с тем же именем, что и результат вычисления переменной, и пытается ее выполнить. Это дает возможность реализации обратных вызовов, таблиц функций и т.п. программных решений:

<?php

function f1() { return 1; }

function f2() { return 2; }

$f='f1';

echo $f(); //1

$f[1]='2';

echo $f(); //2

?>

Переменные функции не будут работать с языковыми конструкциями echo, print, unset(), isset(), empty(), include, require, однако, это возможно при реализации собственных функций или классов-"оберток".

Как и в других языках, функции могут быть *рекурсивными*, т.е. вызывать сами себя. Различают прямую и косвенную рекурсии. Функция называется прямо рекурсивной, если содержит в теле вызов самой себя. Если же функция вызывает другую функцию, которая, в свою очередь, вызывает первую, то такая функция называется косвенно рекурсивной. В качестве примера приведем прямую рекурсию, вычисляющую наибольший общий делитель (НОД) для пары натуральных значений $x и $y:

<?php

function nod ($x,$y) {

return $x==0 ? $y : nod($y%$x,$x);

}

echonod(24,36); //12

?>

*Анонимные* функции позволяют создавать методы, не имеющие определенных имен. Их основное применение - организация callback-параметров, но возможны и другие. Например, анонимные функции могут быть использованы в качестве значений переменных, причем, присваивание анонимной функции переменной использует тот же синтаксис, что и для любого другого присваивания, включая завершающую точку с запятой:

<?php

$fmt = function ($n) {

return sprintf ("%03d",$n).'%';

};

echo $fmt(100); //100%

$n = $fmt(55.5); //055%

echo '<br>'.$n;

?>

Наконец, в самом PHP содержится большое количество встроенных функций и языковых конструкций. Также есть функции, которые требуют, чтобы PHP был сконфигурирован с определенными расширениями, в противном случае вы получите сообщение о фатальной ошибке, вызванной использованием неизвестной функции. Например, чтобы использовать функции для работы с изображениями, нужно подключить библиотеку GD, а для работы функции mysql\_connect() необходима поддержка модуля MySQL.

Вызвав стандартные функции phpinfo() или get\_loaded\_extensions(), мы можем узнать, поддержка каких модулей есть в нашей версии PHP. С помощью стандартного метода get\_defined\_functions() можно получить и полный список всех определенных функций, встроенных и пользовательских. Приведем пример использования этого ме­тода:

<?php

$my\_1 = function () { return 1; };

function my\_2 () { return 2; }

$a = get\_defined\_functions();

echo 'Internal functions:<br>';

foreach ($a['internal'] as $item)

echo $item.'<br>';

echo 'User functions:<br>';

foreach ($a['user'] as $item)

echo $item.'<br>';

?>

Анонимной функции, заданной в переменной $my\_1, в списке пользовательских функций не будет. Подтвердить ее существование мы могли бы, распечатав массив всех определенных в среде переменных, полученный стандартным методом get\_defined\_vars:

<?php

$my\_1 = function () { return 1; };

$a=get\_defined\_vars();

print\_r ($a);

?>

Служебная функция print\_r не всегда красиво форматирует свой вывод, но бывает очень удобна при отладке скриптов, так как умеет выводить информацию о любом объекте, переданном ей в качестве аргумента.

В большинстве же случаев тип аргумента стандартной функции имеет значение. Если в стандартную функцию передаются аргументы неверных типов, например, массив (array) вместо строки (string), возвращаемое значение функции не определено. Вероятнее всего, в этом случае будет возвращен NULL, но это не правило, а соглашение.

Рекомендуемые по итогам главы для чтения разделы стандартной справки следующие: "Справочник языка" - "Типы" - "Массивы", "Справочник языка" - "Функции", "Справочник функций" - "Расширения, относящиеся к переменным и типам" - "Функции для работы с массивами".

**Пример 3.** Составить и вывести массив простых чисел, не превышающих заданного натурального значения N.

<?php

$n=10;

if (isset($\_GET['n']))

$n= abs(intval($\_GET['n']));

function simple ($n) {

if ($n<2) return false;

else if ($n<4) return true;

for ($i=2; $i<=floor(sqrt($n)); $i++)

if ($n%$i==0) return false;

return true;

}

$a = array ();

for ($i=1; $i<=$n; $i++) {

if (simple($i)) $a[] = $i;

}

echo '<p>';

foreach ($a as $i) echo $i.' ';

echo '</p>';

?>

Примечания:

1. В коде не выводятся необходимые заголовки файла HTML, добавьте их самостоятельно.

2. Значение переменной $n принимается равным 10 по умолчанию или может быть передано скрипту через URL-адресом методом GET, например, если скрипт сохранен в папке www под именем lab4.php, вызовите его из браузера адресной строкой <http://localhost/lab4.php?n=100>.

# 4. Работа со строками и регулярными выражениями в PHP

Строка в PHP - это набор символов в однобайтовой кодировке. Как уже отмечалось, это означает, что встроенной поддержки Юникода в языке нет. К сожалению, разработка шестой версии PHP, где таковая предполагалась, на момент написания пособия приостановлена.

Из материалов главе 3 мы усвоили основное отличие между определением строки в одинарных кавычках (апострофах) и двойных кавычках: в последнем случае PHP распознает большее количество управляющих последовательностей для специальных символов. К символам \' и \\, определенным для строк в апострофах, добавляются \n, \r, \t, \v и т.п.

Кроме того, для строк определен так называемый *heredoc-синтаксис*, для которого определение строки начинается с символов <<<, после которых следует идентификатор, а затем перевод строки. После этого записан текст строки, а затем повтор идентификатора, закрывающий определение:

<?php

$name = 'User';

$str1 = <<<HEREDOC1

Привет, $name, я -

heredoc-строка

HEREDOC1;

echo $str1;

?>

Важно, что закрывающий идентификатор не должен вводиться с отступом, при этом до или после точки с запятой не должно быть лишних разделителей. Идентификатор heredoc-строки соответствует тем же правилам именования, что и остальные метки PHP: содержит буквенно-цифровые символы и знак подчеркивания, не начинаясь с цифры. Как видно из примера, внутри heredoc-строки также как внутри строки в двойных кавычках подставляются значения переменных.

С версии PHP 5.3 определен также *nowdoc-синтасксис* - это то же самое для строк в апострофах, что heredoc для строк в двойных кавычках. Внутри nowdoc-строки не осуществляется никаких подстановок, так что конструкция подходит для внедрения PHP-кода или других больших блоков текста без необходимости его экранирования.

Чаще всего значение переменной может быть преобразовано в строку, с помощью оператора приведения типа (string) или функции strval(). В выражениях, где необходима строка, преобразование происходит автоматически, например, при использовании функции echo или print, либо преобразование выполняется, когда значение переменной сравнивается со строкой. Большинство значений в PHP может быть преобразовано в строку для постоянного хранения. Этот метод называется *сериализацией* и может быть выполнен при помощи функции serialize().

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля. Для этого, как и в C++, служат квадратные скобки после строки, но можно применять и фигурные скобки:

<?php

$s = 'Мама мыла раму мылом.';

for ($i=0; $i<strlen($s); $i++)

if ($s[$i]==' ') $s{$i}='\_';

echo $s;

?>

Таким образом, строку PHP можно рассматривать как массив символов, если речь идет об однобайтовых кодировках, а не о Юникоде. Строку в Юникоде требуется сначала конвертировать в однобайтную кодировку функцией iconv или utf8\_decode.

Смещение (индекс символа) в строке должно задаваться целым числом или строкой, содержащей цифры. Попытка записи в смещение за границами строки не приведет к ошибке, а дополнит строку пробелами до этого смещения.

Внутренняя реализация строкового типа представляет собой массив байт и целое число, содержащее длину буфера. Она не содержит никакой информации о способе преобразования этих байт в символы, предоставляя задачу программисту. Однако стандартные функции вынуждены делать некоторые предположения о кодировке строк. Одна из типовых проблем программирования на PHP - учет того, что разные группы стандартных функций исходят из различных предположений о кодировках:

* ряд функций предполагает, что строка закодирована в однобайтовой кодировке, в частности, так поступают функции-аналоги строковых методов C++ и Паскаля - substr(), strpos(), strlen(), strcmp();
* некоторые функции ожидают передачу информации о кодировке в виде параметра, при этом, они могут предполагать наличие кодировки по умолчанию, если параметр не был указан. Так ведет себя htmlentities() и большинство функций из расширения mbstring (Multi-Byte String);
* некоторые функции используют текущие установки локали, например, strcasecmp(), strtoupper();
* ряд функций подразумевает, что строка использует определенную кодировку, обычно Юникод (UTF-8).

Как правило, написание корректных программ на PHP, работающих с Юникодом, означает использование альтернативных строковых функций вместо стандартных, обычно из расширений intl (Internationalization Functions) и mbstring.

Кратко рассмотрим основные группы стандартных функций, работающих со строками. Их более подробное описание вы можете найти в стандартной справке.

Для работы с символами и получения общих свойств строки полезны следующие функции:

* chr - возвращает символ по его коду;
* ord - возвращает ASCII-код символа;
* ltrim, rtrim, trim - удаляют пробелы и другие разделители из начала, конца и начала+конца строки;
* strlen - возвращает длину строки в байтах.

Для работы с форматированными строками служат функции printf (вывод в консоль) и sprintf (вывод в строку), позволяющие форматировать свой вывод аналогично соответствующим функциям языка C++. Также определены аналоги большинства других функций C++ для работы с форматированными строками: sscanf, sprintf, fprintf, fscanf, vsprintf.

Форматные строки строятся по правилам, аналогичным языку C++, но с исключением некоторых значений спецификаторов. Спецификация определяется символом "%", за которым может следовать до пяти спецификаторов в следующем порядке:

* *спецификатор заполнения* - устанавливает символ, которым строка заполняется до заданного размера;
* *спецификатор выравнивания* - по умолчанию дополняет строку до указанной минимальной ширины с левого края (т.е. строка выравнивается по правому краю). Если добавлен символ дефиса, то строка выравнивается по левому краю (символы добавляются справа);
* *спецификатор минимальной ширины* представляет собой целое число, задающее минимальный размер форматированной строки. Если переданная строка меньше, то она дополняется символами, указанными в спецификаторе заполнения;
* *спецификатор точности* предназначен для указания количества десятичных знаков в представлении чисел с плавающей точкой. Он также задается целым числом, отделенным от спецификатора минимальной ширины символом точки. При применении этого спецификатора для форматирования строк, он определяет максимальное количество символов, которое нужно взять из переданной строки;
* *спецификатор типа* предназначен для указания типа данных, переданных в качестве аргумента.

Спецификатор типа может принимать одно из следующих значений:

* % - экранирует следующий за ним %;
* b - целое число, представляемое в двоичном виде;
* с - целое число, представляемое в виде символа с тем же ASCII-кодом;
* d - целое число, представляемое в десятичном виде;
* f - число с плавающей точкой, представляемое в виде десятичной дроби;
* o - целое число, представляемое в восьмеричном виде;
* s - строка;
* x - целое число, представляемое в шестнадцатеричном виде в нижнем регистре;
* X - целое число, представляемое в шестнадцатеричном виде в верхнем регистре.

Приведем пример, показывающий работу функций форматирования:

<?php

$f = 1234.567;

printf ("\n<br>f=%10.2f",$f); //f= 1234.57

$n = 5090;

printf ("\n<br>n=%05d",$n); //n=05090

$s = "\*\*\*";

printf ("\n<br>s=%'#-5s",$s); //s=\*\*\*##

printf ("\n<br>%d,%f,%s",$n,$f,$s);

//5090,1234.567000,\*\*\*

?>

Для объединения и разбиения строк чаще всего применяют функции explode и implode (join):

* array explode (string $delimiter, string $string) - возвращает массив строк, полученных разбиением строки string с использованием delimiter в качестве разделителя;
* string implode (string $glue, array $pieces) - функция возвращает строку, содержащую строковое представление всех элементов массива в указанном порядке, со строкой $glue между каждой парой элементов.

Приведем пример, иллюстрирующий работу данных функций:

<?php

$s = "1 1.5 2 -3 ";

$a = explode(' ',$s);

$a = array\_diff($a,array(''));

foreach ($a as $v) echo $v.' ';

//1 1.5 2 -3

echo '<br>'.implode ('\_',$a);

//1\_1.5\_2\_-3

?>

Похожиезадачирешаетцелыйрядальтернативныхфункций, например, parse\_str, str\_getcsv, str\_split, chunk\_split, str\_word\_count, strtok, str\_repeat, str\_pad. Также широко применяется разбор и объединение строк с помощью регулярных выражений.

Для сравнения однобайтовых строк можно применять как непосредственное сравнение строковых переменных, так и стандартную функцию int strcmp (string $s1, string $s2), которая возвращает отрицательное число, если $s1 меньше (предшествует по алфавиту) $s2, положительное число, если $s1 больше, чем $s2, и 0, если строки совпадают:

<?php

$s1 = 'abc'; $s2 = 'abd';

echo $s1<$s2?'Yes':'No'; //Yes

echo strcmp($s1, $s2); //-1

?>

Альтернативныефункции - strcasecmp, preg\_match, substr\_compare, strncmp.

Базовыми функциями для поиска и замены строк служат следующие:

* intstrpos (string $haystack, mixed$needle) - ищет позицию первого вхождения подстроки $needle в строку $haystack;
* mixedstr\_replace (mixed $search, mixed $replace, mixed $subject) - возвращает строку или массив, в котором все вхождения $search в $subject заменены на $replace:

<?php

$s = 'Hello, world!';

echo strpos($s,','); //5

echo str\_replace('l','L',$s); //HeLLo, worLd!

?>

Дляпохожихцелейможноприменятьидругиефункции: stripos, strrpos, strripos, strstr, strpbrk, substr, preg\_match, str\_ireplace, substr\_replace, preg\_replace, strtr.

Дляактуальнойв PHP задачи обработки специальных символов внутри строк применяются функции htmlspecialchars, htmlentities, nl2br, htmlspecialchars\_decode, html\_entity\_decode, strip\_tags, addslashes, stripslashes, quotemeta.

Выполнить преобразование регистра символов в различных ситуациях помогут стандартные методы strtolower, strtoupper, lcfirst, ucfirst, ucwords.

Для шифрования и дешифрования строк, а также работы с кодировками пригодятся методы md5, sha1, str\_rot13, str\_shuffle, convert\_uuencode, convert\_uudecode, base64\_encode, base64\_decode, crc32, crypt, bin2hex, pack, unpack, convert\_cyr\_string.

Наконец, отдельно отметим основные функции, предназначенные для работы с многобайтными строками. Почти все они имеют в начале имени префикс mb\_ и являются аналогами стандартных функций, предназначенных для однобайтовых кодировок:

* mb\_detect\_encoding - определение кодировки символов;
* mb\_strcut (или mb\_substr) - получение части строки;
* mb\_strstr (или mb\_strpos), mb\_strrchr - поиск первого и последнего вхождений одной строки Юникода в другую;
* mb\_strlen - получает длину строки;
* mb\_substr\_count - возвращает количество вхождений подстроки;
* mb\_split - разделение строк в многобайтных кодировках, используя регулярное выражение;
* mb\_ereg - совпадение с регулярным выражением с поддержкой многобайтовых символов;
* mb\_ereg\_replace - осуществляет замену по регулярному выражению с поддержкой многобайтовых символов;
* mb\_ereg\_search\_pos - возвращает позицию и длину совпавшего с регулярным выражением участка многобайтной строки.

*Регулярные выражения*, значительно облегчают разбор и форматирование сложных строковых данных.

Р**егулярные выражения** (англ. regular expressions, сокращенно regexp, regex) - это шаблоны, которые можно сопоставлять со строками. В отличие от обычного определения вхождения подстроки в строку, которое в PHP можно делать, например, с помощью функции strpos, шаблоны позволяют включать в искомое или заменяемое строковое выражение *групповые символы* (англ. wildcards) - именно эта особенность делает их столь широко применимыми.

Регулярные выражения доступны не только в PHP, но и в большинстве современных языков и пакетов, работающих с текстом - от Perl, Javascript и .NET Framework до Microsoft Office, так что знать их полезно любому грамотному пользователю, не говоря уж о программистах. Однако начинающих регулярные выражения нередко пугают, а официальные руководства достаточно сложны для изучения (например, классическая книга Mastering Regular Expressions автора Jeffrey Friedl).

В PHP доступно 2 типа регулярных выражений:

* **POSIX-совместимые** (англ. POSIX означает Portable Operating System Interface for Unix). В PHP под Windows они доступны по умолчанию, под Unix их можно установить при настройке конфигурации PHP с помощью ключа - with-regex
* **Perl-совместимые** или **PCRE** (англ. Perl Compatible Regular Expressions). Имеют синтаксические отличия, но считаются более быстрыми и мощными. Под Windows и Unix регулярные выражения PCRE доступны по умолчанию, именно на них будем ориентироваться в дальнейшем.

Сначала опишем основные **специальные символы** (**символьные классы**) регулярных выражений, приведя примеры по каждому из них:

* символ **^** в регулярном выражении обозначает начало строки. Он ставится перед символом или шаблоном, на который действует: шаблон ^а означает, что строка начинается с символа "а";
* символ $ в регулярном выражении обозначает конец строки. Он ставится после символа или шаблона, на который действует. Так, регулярное выражение а$ означает, что строка заканчивается символом "а";
* в квадратных скобках [] указываются альтернативные символы, в строке может присутствовать любой из них. Например, выражение PHP[345] может означать "PHP3", "PHP4" или "PHP5";
* символ - (дефис) внутри квадратных скобок разделяет первый и последний символы последовательности, в строке может присутствовать любой символ от первого до последнего включительно. Разумеется, "последовательности символов" имеют смысл лишь относительно текущей кодировки. Например, в UTF-8 (Юникоде) и Windows-1251 (стандартная русифицированная Windows) русские буквы (кроме "ё") гарантированно имеют идущие подряд коды, в кодировках DOS и КОИ-8R это не так. В Юникоде или русскоязычной кодировке Windows строка регулярного выражения PHP[3-5] может означать "PHP3", "PHP4" или "PHP5";
* символ ^ внутри квадратных скобок означает, что в строке допустимы любые символы, кроме указанных этим знаком. Обратите внимание, что вне квадратных скобок этот символ имеет другое значение (начало строки). Например, выражение [^A-C] означает любую цифру, или букву от "D" и далее по алфавиту;
* символ | (вертикальная черта) используется как разделитель для нескольких альтернативных шаблонов. Например, PHP4|PHP5 означает "PHP4" или "PHP5";
* круглые скобки () определяют *подшаблон*, независимую составную часть основного шаблона. Полезно отдельные "части" формата разбираемой строки определять как отдельные подшаблоны. В простейшем случае (а)(б) означает "аб", но как 2 подшаблона, а и б;
* символ . (точка) обозначает любой символ. Так, под выражение PHP. подходят строки "PHP3", "PHP4", "PHP5" или "PHPA" и т.д.;
* символ \ (обратный слэш). Следующий после "\" символ не считается специальным, иногда говорят, что "\" *экранирует* идущий за ним символ. Такое же поведение обратного слэша в строках характерно и для C++. Например, выражение \. означает символ точки, но не любой символ, а выражение \\ означает символ "\", но не управляющий символ;

Перечислим еще несколько полезных специальных символов, которые есть в PCRE-совместимых регулярных выраже­ниях:

* \s - пробел или табуляция (от англ. "space");
* \S - видимые символы, можно считать, что это все, что не совпадает с \s;
* \w - в этот спецсимвол включены все символы, которые могут входить в слово (англ. word), обычно [a-zA-Z\_], хотя это зависит от установленной локали, поддержки Юникода и ряда других настроек;
* \W - все, что не входит в определение \w, т.е. [^a-zA-Z\_];
* \d - цифра (от англ. "digit"; специальный символ позволяет не писать символьный класс [0-9]);
* \D - все, что не является цифрой.

Важно, что один символьный класс описывает только один символ, а когда символов много, применяются *квантификаторы*, управляющие числом допустимых повторений символа в шаблоне.

Например, если мы хотим описать строки, которые состоят из двух любых строчных букв латинского алфавита, после которых следуют две любые цифры, мы можем сделать это так:

[a-z][a-z][0-9][0-9]

Если мы укажем квантификаторы, запись можно сократить:

[a-z]{2}[0-9]{2}

Перечислим основные квантификаторы регулярных выражений и приведем примеры:

* квантификатор ? (вопросительный знак) означает ноль или одно повторение предыдущего символа или выражения, сокращенно "ноль или один". Например, аб? может означать "а" или "аб";
* квантификатор \* (звездочка) означает ноль или более повторений предыдущего символа или выражения ("ноль или много"). Так, аб\* может означать "а", или "аб", или "абб" и т.д. Часто используется управляющая последовательность .\* для обозначения любого количества любых символов между двумя частями регулярного выражения;
* квантифкатор + (плюс) означает одно или более повторений предыдущего символа или выражения ("хотя бы один", "один или много"). Например, аб+ может означать "аб", или "абб", или "аббб" и т.д.;
* квантифкатор {n}, где n - натуральное число, указывает, что нужно искать ровно n вхождений предыдущего символа или шаблона. Например, [0-9]{3} - это три любых идущих подряд цифры;
* квантифактор {min,max}, где min, max - натуральные числа и min<max, означает минимальное (min) и максимальное (max) допустимое число вхождений предшествующего символа или шаблона. Если опущен минимум min (но не запятая), он предполагается равным нулю, если опущен максимум max (но не запятая), он предполагается равным infinite (бесконечности). Например, запись a{1,3} может означать "а" или "аа" или "ааа".Выражение a{1,} может означать цепочку из букв "а" любой длины, но не менее одного символа. Строка a{,3} может означать цепочку из букв "а" в количестве не более трех (обратите внимание, ноль букв "а" тоже попадает под это выражение).

Манипулируя указанными выше символьными классами, специальными символами и квантификаторами, можно составить много полезных примеров регулярных выражений, приведем несколько из них:

* [^\s] - любой символ, который не является пробелом;
* [^\s]+ - минимум один символ, который не является пробелом;
* \s+ - минимум один пробел;
* ^\d+$ - строка является числом из одной и более цифр (обратите внимание, что здесь *вся* проверяемая строка должна соответствовать шаблону - так как в него включены метки начала ^ и конца $ строки);
* ^[a-zA-Z0-9]+$ - латинские буквы и цифры, минимум один символ;
* ^[a-zа-я0-9\_]{1,8}$ - строка только из латинских или русских букв, цифр и подчеркивания от 1 до 8 символов длиной (игнорирование регистра обычно можно указать с помощью описанных ниже *модификаторов шаблонов*);
* [^(\x7F-\xFF)|(\w)|(\s)] - исключаем символы с кодами 127 и больше, разрешаем печатаемые и пробельные символы. Из примера видно, что в регулярных выражениях можно писать 16-ричные коды символов в стиле языка C++.

Для работы с регулярными выражениями средствами PHP нам понадобится всего несколько стандартных функций:

* preg\_match - выполняет проверку на соответствие строки регулярному выражению;
* preg\_match\_all - выполняет глобальный поиск шаблона регулярного выражения в строке;
* preg\_replace - выполняет поиск и замену по регулярному выражению;
* preg\_split - разбивает строку на части по регулярному выражению.

**Пример 4.** Полный код приложения для проверки введенного набора слов на соответствие одной из допустимых форм записи вещественного числа.

<!DOCTYPE html PUBLIC

"-//W3C//DTD HTML 4.01 Transitional//EN">

<html><head>

<meta content=

"text/html; charset=Windows-1251"

http-equiv="content-type">

<title>Числа</title>

</head><body>

<p><small>Проверить с помощью регулярного

выражения, являются ли значения из поля

ввода допустимыми формами записи

вещественных чисел, например,

<br> -1.7E+03 (да) -15e (нет)

1 (да) +1.5 (да)</small></p>

<?php

$data='';

if (isset($\_GET['data']))

$data = strip\_tags(trim($\_GET['data']));

if (!isset($\_GET['submit'])) {

echo '<form method="get"

action="'.$\_SERVER['PHP\_SELF'].'">

<input type="text" name="data"

maxlength="128" size="128"

value="'.$data.'">

<input type="submit" name="submit"

value="OK"></form>';

}

else {

$array = preg\_split ("/[\s,]+/",$data);

print\_r ($array);

echo '<br>Всего элементов: '.

count($array);

$k=0;

foreach ($array as $d) {

echo '<br>Элемент '.++$k.

' (<font color="red">'.$d.'</font>): ';

$exp = "/^(\+|\-)?((\d+)|(\d+\.)|(\.\d+)|(\d+\.\d+)){1}((e|E)(\+|\-)?\d+)?$/";

$res = preg\_match ($exp,$d);

if ($res===false) echo

'ошибка в регулярном выражении!';

elseif ($res===0) echo 'не число';

else echo 'число '.$d;

}

echo '<br><a href="'.

$\_SERVER['PHP\_SELF'].

'?data='.

urlencode($data).'">Вернуться</a>';

}

?>

</body></html>

Примечание. Вводимые в форму приложения числа разделяются хотя бы одним символом пробела или запятой.

**Лекция 14. Работа с файловой системой и использование СУБД MySQL средствами PHP**

**План лекции:**

Работа с файловой системой средствами PHP

Загрузка файлов и динамическое формирование изображений средствами PHP

Использование СУБД MySQL в PHP

Управление сеансами пользователя и безопасность скриптов

Объектно-ориентированное программирование на PHP

Установка и модификация системы управления содержимым на PHP/MySQL

# 1. Работа с файловой системой средствами PHP

Как и приложение на других языках программирования, скрипт на PHP может читать и записывать текстовые или бинарные файлы. Для этого существует несколько основных групп стандартных функций.

Во-первых, существуют высокоуровневые функции, способные прочитать или записать файл одним оператором. Перечислим основные из них:

* string file\_get\_contents (string $file­name) - читает содержимое файла в строку и возвращает эту строку или значение false в случае ошибки;
* array file (string $filename) - читает содержимое файла и помещает его в массив, в случае ошибки возвращает false;
* int readfile (string $filename) - читает файл и записывает его в буфер вывода;
* int file\_put\_contents(string $filename, mixed $data) - пишетобъект $data, которыйможетиметьтипданных string, array илиресурса. Возвращает количество записанных в файл байт, или false в случае ошибки. Если файл с именем $filename не существует, он будет создан. Иначе, существующий файл будет перезаписан, за исключением случая, если указан флаг FILE\_APPEND.

У всех функций есть дополнительные параметры, узнать о которых можно в стандартной справке.

Показанный ниже код выводит на экран содержимое файла data.txt, находящегося в текущей папке. При этом переводы строк в тексте с помощью стандартной функции nl2br заменяются на теги разрыва строки <br>.

<?php

echo nl2br(file\_get\_contents("data.txt"));

?>

В следующем примере функция file загружает все содержимое файла в индексируемый массив, затем делается построчный вывод файла вместе с номерами строк:

<?php

$a=file('data.txt');

$n=1;

foreach ($a as $line)

echo $n++.'. '.$line.'<br>';

?>

Во-вторых, в PHP существует большое количество функций для чтения и записи отдельных символов, строк или форматированных данных, а также для работы с файлами и папками. Часть из них напоминает аналогичные средства языка C++, а часть специфична для PHP. Рассмотрим основные действия с файлами и папками.

Для проверки существования файла обычно используются две функции: file\_exists и is\_file.

Функция с прототипом bool file\_exists(string файл) проверяет, существует ли заданный файл. Если файл существует, функция возвращает true, в противном случае возвращается false.

Функция boolis\_file(string файл) проверяет не только существование заданного файла, но и возможность выполнения с ним операций чтения/записи.

Функция intfilesize(string файл) возвращает размер в байтах файла с заданным именем или false в случае ошибки:

<?php

$file='data.txt';

if (is\_file($file))

echo 'Файл доступен, размер='.

filesize($file).' байт';

else

echo 'Файл не существует или недоступен';

?>

Как и в стандартной библиотеке ввода/вывода языка C++, прежде чем выполнять операции с файлом, его нужно открыть и связать с файловой переменной, а после завершения работы с файлом его следует закрыть.

Функция int fopen (string файл, string режим [, int включение\_пути]) открывает файл (если он существует) и возвращает целое число - так называемый *файловый дескриптор* (file handle).

Открываемый файл может находиться в локальной файловой системе, существовать в виде стандартного потока ввода/вывода или представлять собой файл в удаленной системе, принимаемый средствами протокола HTTP или FTP.

Параметр файл может задаваться в нескольких формах, перечисленных ниже.

Если параметр содержит имя локального файла, функция fopen открывает этот файл и возвращает дескриптор. На платформе Windows, необходимо экранировать (удваивать) все   
обратные слеши '\' в пути к файлу или использовать прямые слеши '/'.

Если параметр задан в виде php://stdin, php://stdout или php://stderr, открывается соответствующий стандартный поток ввода/вывода.

Если параметр начинается с префикса http://, функция открывает подключение HTTP к серверу и возвращает дескриптор для указанного файла.

Если параметр начинается с префикса ftp://, функция открывает подключение FTP к серверу и возвращает дескриптор для указанного файла. В этом случае следует обратить внимание на два обстоятельства: если сервер не поддерживает пассивный режим FTP, в котором сервер ожидает подключения со стороны клиентов, то вызов fopen завершается неудачей. Более того, FTP-файлы открываются либо для чтения, либо для записи.

Параметр режим определяет возможность выполнения чтения и записи в файл. В табл. 6 перечислены некоторые значения, определяющие режим открытия файла.

Таблица 6 Режимы открытия файла

|  |  |
| --- | --- |
| Режим | Описание |
| r | Только чтение. Указатель текущей позиции устанавливается в начало файла |
| r+ | Чтение и запись. Указатель текущей позиции устанавливается в начало файла |
| w | Только запись. Указатель текущей позиции устанавливается в начало файла, а все содержимое файла уничтожается. Если файл не существует, функция пытается создать его |
| w+ | Чтение и запись. Указатель текущей позиции устанавливается в начало файла, а все содержимое файла уничтожается. Если файл не существует, функция пытается создать его |
| a | Только запись. Указатель текущей позиции устанавливается в конец файла. Если файл не существует, функция пытается создать его |
| a+ | Чтение и запись. Указатель текущей позиции устанавливается в конец файла. Если файл не существует, функция пытается создать его |

Если необязательный третий параметр включение\_пути равен значению true, то путь к файлу определяется по отношению к каталогу включаемых файлов, указанному в файле php.ini директивой include\_path.

После того, как работа с файлом завершена, его следует закрыть функцией int fclose(int дескриптор). При успешном закрытии возвращается значение true, при неудаче false. Функция успешно закрывает только те файлы, которые были ранее открыты функциями fopen или fsockopen.

С открытыми файлами выполняются две основные операции - запись и чтение.

Функция bool is\_writable (string файл) позволяет убедиться в том, что файл существует и для него разрешена операция записи. Возможность записи проверяется как для файла, так и для папки.

Функция int fwrite (int дескриптор, string переменная [, int длина]) записывает содержимое строковой переменной в файл, заданный файловым дескриптором. Если при вызове функции передается необязательный параметр длина, запись останавливается либо после записи указанного количества символов, либо при достижении конца строки. Синоним функции с теми же параметрами - fputs.

Записать в файл отформатированные данные можно с помощью функции int fprintf (int дескриптор, string формат [, mixed $args [, mixed $... ]] ). Строка формата и последующие аргументы формируются по правилам, аналогичным таковым для языка C++.

Функция bool is\_readable (string файл) позволяет убедиться в том, что файл существует и для него разрешена операция чтения. Возможность чтения проверяется как для файла, так и для папки.

Функция intfread(int дескриптор, int длина) читает из файла, заданного файловым декскриптором, указанное количество байт. Дескриптор должен ссылаться на открытый файл, доступный для чтения. Чтение прекращается после прочтения заданного количества байт или при достижении конца файла. Используя функцию int filesize (string файл) для определения размера нужного файла в байтах, вы гарантируете, что функция fread прочитает все содержимое файла.

Функция string fgetc (int дескриптор) возвращает строку, содержащую один символ из файла в текущей позиции указателя, или значение false при достижении конца файла. Дескриптор должен ссылаться на открытый файл, доступный для чтения.

Функция string fgets (int дескриптор [, int длина]) возвращает строку, прочитанную от текущей позиции указателя в файле, определяемом файловым дескриптором. Файловый дескриптор должен ссылаться на открытый файл, доступный для чтения. Чтение прекращается при выполнении одного из следующих условий:

* указан параметр длина и из файла прочитано длина-1 байт;
* из файла прочитан символ новой строки (включается в возвращаемую строку);
* из файла прочитан признак конца файла (EOF).

Простейший путь организовать построчное чтение файла - не указывать второй параметр, тогда чтение из потока будет продолжаться до достижения конца строки.

Функция string fgetss (int дескриптор, int длина [, string разрешенные\_теги]) полностью аналогична fgets за одним исключением - она пытается удалять из прочитанного текста все теги HTML и РНР, кроме разрешенных, если таковые указаны третьим параметром.

Для форматного чтения данных из файла удобна функция fscanf (int дескриптор, string формат [, mixed &$... ] ). Функция берет данные для обработки из файла, ассоциируемого с дескриптором, и интерпретирует их согласно строке формата.

Если в функцию переданы только два аргумента, обработанные значения будут возвращены в виде массива. В ином случае, если были переданы необязательные аргументы, функция вернет количество присвоенных значений. Необязательные аргументы должны быть переданы по ссылке. Каждый вызов fscanf читает одну строку из файла, таким образом, функция *не работает* аналогично одноименной функции из C++ и не применима, например, к задаче чтения из файла произвольного количества числовых значений, расположенных в разных строках.

Функция bool feof (int дескриптор) проверяет, достигнут ли конец файла. Дескриптор должен быть корректным и указывать на файл, успешно открытый функциями fopen или fsockopen и еще не закрытый функцией fclose. Функция возвращает true, если указатель файла указывает на EOF или произошла ошибка (в том числе таймаут сокета), иначе возвращает false.

Ниже приведен пример открытия файла функцией fopen с последующим чтением данных (целых или вещественных чисел) в массив $a. Вызов die, используемый в сочетании с fopen, обеспечивает вывод сообщения об ошибке в том случае, если открыть файл не удастся.

<?php

$file = 'data.txt';

$f = fopen($file, "r") or

die("File $file does not exist!");

$a = Array ();

while (($data = fgets($f))!==false) {

$a0 = explode (" ",$data);

foreach ($a0 as $item)

if (is\_numeric(trim($item))) $a[] = $item;

}

fclose ($f);

foreach ($a as $item) echo $item.' ';

?>

Числа в файле data.txt могут располагаться произвольно, например, следующим образом:

1 2 3.5

4 5.5 6

3

Пустые строки и нечисловые значения будут проигнорированы скриптом.

Обратите внимание на синтаксис записи условия, определяющего, достигнут ли конец файла, а также на необходимость удаления из прочитанных элементов файла $item лишних разделителей функцией trim, в противном случае, например, могут быть не прочитаны числа, завершающие строку файла.

В РНР существуют функции для просмотра и выполнения различных операций с файлами на сервере. Информация об атрибутах серверных файлов (местонахождение, владелец и привилегии) часто бывает полезной. В табл. 7 приводится краткое описание основных функций для работы с файловой системой, более подробную информацию о них можно найти в справочной системе.

Таблица 7 Функции для работы с файловой системой

|  |  |
| --- | --- |
| Имя функции | Описание |
| basename | Выделяет имя файла из переданного полного пути к файлу |
| getlastmod | Возвращает дату и время последней модификации страницы, из которой вызывается функция |
| stat | Возвращает индексируемый массив с подробной информацией о файле с заданным именем |
| chgrp | Позволяет сменить группу, которой принадлежит заданный файл |
| filegroup | Возвращает идентификатор группы владельца файла с заданным именем |
| chmod | Изменяет разрешения файла с заданным именем |
| fileperms | Возвращает разрешения файла с заданным именем |
| chown | Позволяет сменить владельца файла |
| fileowner | Возвращает идентификатор пользователя для владельца файла с заданным именем |
| сору | Позволяет скопировать файл при выполнении сценария РНР |
| rename | Позволяет переименовать файл при выполнении сценария РНР |
| unlink | Позволяет удалить файл при выполнении сценария РНР |

Ряд стандартных функций, основные из которых описаны в табл. 8, позволяют работать с папками (каталогами) операционной системы.

Таблица 8Функции для работы с папками

|  |  |
| --- | --- |
| Имя функции | Описание |
| getcwd | Получает имя текущего рабочего каталога |
| scandir | Получает список файлов и каталогов, расположенных по указанному пути |
| dirname | Извлекает путь из полного имени файла |
| is\_dir | Проверяет, является ли каталогом файл с заданным именем |
| mkdir | Позволяет создать новый каталог при выполнении сценария РНР |
| opendir | Открывает и возвращает дескриптор для работы с заданным каталогом |
| closedir | Закрывает ранее открытый дескриптор для работы с каталогом |
| readdir | Возвращает очередной элемент заданного каталога |
| chdir | Осуществляет переход в каталог, заданный параметром |
| chroot | Изменяет корневой каталог текущего процесса на переданный в качестве параметра |
| rewinddir | Сбрасывает поток каталога, переданный в параметре таким образом, чтобы  тот указывал на начало каталога |

Наконец, PHP позволяет работать с файлами на уровне *сокетов*, т.е. интерфейсов для обмена данными между процессами как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью.

Функция int fsockopen (string узел, int порт [, int код\_ошибки [, string текст\_ошибки [, int тайм-аут]]]) устанавливает сокетное соединение с сервером в Интернете через протокол TCP или UDP. Необязательные параметры код\_ошибки и текст\_ошибки содержат информацию, которая будет выводиться в случае неудачи при подключении к серверу. Оба параметра должны передаваться по ссылке. Третий необязательный параметр, тайм-аут, задает продолжительность ожидания ответа от сервера (в секундах).

Функция stream\_set\_blocking (int дескриптор, int режим) позволяет установить контроль над тайм-аутом для операций с сервером. Параметр режим равен 1 для блокирующего режима или 0 для неблокирующего.

Следующий код показывает пример получения информации о сервере с помощью указанных функций.

<?php

function gethost ($host,$path) {

$fp=fsockopen($host,80,&$errno,&$errstr,10);

socket\_set\_blocking($fp,1);

//Отправитьзаголовки HTTP

fputs ($fp,"GET $path HTTP/1.1\r\n");

fputs ($fp, "Host: $host\r\n\r\n");

//Получитьзаголовки HTTP

do {

$headers = fgets ($fp, 4096);

print $headers.'<br>';

} while (trim($headers)!='');

fclose($fp);

}

gethost("www.google.com", "/");

?>

**Пример 5.** Скрипт для работы с базой данных, хранящейся в текстовом файле.

В качестве развернутого примера напишем скрипт, поддерживающий работу с простейшей текстовой "базой данных", каждая строка которой содержит одну запись. При написании приложения учтем основные типовые проблемы начинающего разработчика.

Прежде всего, определимся, что все файлы будут в кодировке Юникод (UTF-8). Использовать в качестве текстового редактора стандартный Блокнот нельзя из-за добавляемых им в начало файла специальных символов (*меток BOM*), а использовать Notepad++ нужно с осторожностью - на пустом файле сразу же выберите пункт верхнего меню Кодировки - Кодировать в UTF-8 (без BOM), затем Синтаксис - PHP, затем уже пишите или вставляйте текст и выполняйте команду меню Файл - Сохранить. Подойдет также встроенный редактор файл-менеджера Far 3, в котором вы нажали комбинацию клавиш Shift+F8 и выбрали кодовую страницу 65001 (UTF-8).

В противном случае, из-за лишней и невидимой метки BOM в начале файла мы будем получать следующую распространенную у начинающих ошибку:

Warning: session\_start() [function.session-start]: Cannot send session cache limiter - headers already sent (output started at ...) in ...\function.php on line 2

Определим требования к скрипту. Он должен:

* состоять из отдельных коротких модулей, чтобы можно было легко читать и модифицировать код;
* корректно отображаться, например, сразу "понимать" Юникод;
* не быть чувствительным к настройкам кавычек в PHP и позволять хранить в текстовом файле любые строки, в том числе, со спецсимволами PHP/HTML;
* разумно фильтровать ввод пользователя, например, избавлять его от лишних разделителей;
* уметь добавлять, удалять, редактировать и сортировать записи.

В стороне пока оставляем следующие вопросы:

* надежную защиту от обращения к формам "со стороны";
* защиту от переписывания текстового файла при одновременном обращении к базе нескольких пользователей;
* авторизацию для доступа к редактированию файла;
* разбиение вывода базы на "порции" для более удобного редактирования;
* стили оформления и дизайн проекта.

Формат записи базы определим тоже простейшим - в одной строке текстового файла будут содержаться некое Имя (произвольная строка длиной до 30 символов включительно) и Число (целое значение, занимающие до 6 знакомест). Разделитель записей внутри строки назначим определенной в файле конфигурации константой DIV.

Напишем файлы head.php и foot.php - общие верх и низ всех страниц, подключать их будем оператором include.

**Файл head.php**

<?php

echo '<!DOCTYPE HTML PUBLIC

"-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html><head>

<meta http-equiv="Content-Type"

content="text/html; charset=utf-8">

<title>МИНИ-БД</title>

</head><body>'."\n";

?>

**Файл foot.php**

<?php

echo "\n".'</body></html>';

?>

Мы включили только минимально необходимые мета-теги - тип документа и кодировку.

Файл config.php будет содержать общие настройки скрипта, пока это только имя файла с данными (предполагается, что он в текущей папке, той же, где и все остальные файлы) и разделитель записей в строке данных. Назначим этим разделителем табуляцию, а защиту от случайного или намеренного ввода пользователем табуляции внутри строк "Имя" и "Число" (например, даже в однострочное поле <input type="text"> можно вставить символ табуляции из системного Буфера Обмена) возложим на функции разбора параметров, которые все разделители в строке сделают пробелами.

**Файл config.php**

<?php

define ('FILENAME','data.txt');

define ('DIV',"\t");

?>

Перейдем к функционалу. Как и любой скрипт, позволяющий пользователю вводить данные в HTML-формы, наш продукт должен будет фильтровать пользовательский ввод, передаваемый методами GET, POST, и, возможно, через глобальный массив $\_SESSION. Для решения типовых проблем фильтрации пользовательского ввода, описанных в главе 3, добавим в папку проекта приведенный там файл params.php.

Обратите внимание, что передача данных через массив $\_SESSION в коде params.php разрешена однократно, переданные данные сразу удаляются. Это может пригодиться, например, при "возврате" в форму введенных в другом модуле данных.

В модуль служебных функций function.php, кроме приведенных в главе 3 методов trimall и magic, включим следующие функции:

* read() - функция будет читать текущую базу и возвращать массив записей;
* write($a) - функция будет записывать массив записей $a в файл базы данных;
* get\_index\_by\_name ($a,$name) - функция будет искать по имени $name соответствующую запись и возвращать ее номер (начиная с нуля) или значение -1, если запись не найдена. Это пригодится, чтобы отличать добавление новой записи от редактирования существующей.

Кроме того, файл функций подключит файл конфигурации и запустит сессию - эти возможности могут понадобиться любому модулю, который его подключает.

**Файл function.php**

<?php

session\_start();

require\_once ('config.php');

//Добавитьсюдаметоды trimall и magic

function read () {

return array\_filter (explode ("\n",

@file\_get\_contents (FILENAME)),

function ($var) { return(!empty($var)); });

}

function write ($a) {

@file\_put\_contents(FILENAME,implode("\n",$a));

}

function get\_index\_by\_name ($a,$name) {

$name = mb\_strtolower( trimall($name),

'UTF-8');

foreach ($a as $index=>$item) {

list($myname,$mynumber)=explode(DIV, $item);

if($name==mb\_strtolower(trimall($myname),

'UTF-8')) return $index;

}

return -1;

}

?>

Обратите внимание, что функция read на всякий случай дополнительно фильтрует массив записей от пустых строк, которых, вообще-то, не должно возникать.

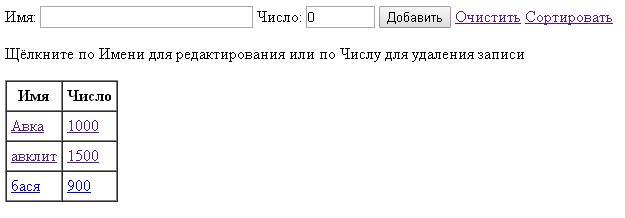
Также важно корректное приведение строки в Юникоде к нижнему регистру (см. mb\_strtolower в коде). Альтернативной показанному в листинге подходу могла бы стать настройка и использование локали на сервере.

Основной файл приложения index.php будет решать следующие задачи:

* выводить форму для добавления новой записи, которую обрабатывает модуль add.php;
* рядом с формой добавления выведем дополнительные команды - очистка формы "самовызовом" скрипта без параметров (кнопка <input type="reset"> здесь не подойдет, так как не передает на сервер данных) и ссылку для обращения к модулю сортировки записей по имени sort.php;
* получать от других модулей результаты их работы в виде числовой переменной $status и выводить соответствующие сообщения (массив $status\_msg). Значение $status, равное нулю, будет принято по умолчанию, ему соответствует вывод краткой справки о работе программы;
* если база непуста, модуль должен показать ее записи и обеспечить переход к редактированию или удалению.

Чтобы не перегружать таблицу дополнительными кнопками и ссылками, сделаем щелчок *по имени* переходом к редактированию записи модулем edit.php, а щелчку *по полю "Число"* будет соответствовать удаление записи модулем del.php.

На рис. 4 показан внешний вид приложения при нескольких добавленных записях.

  
Рис. 4. Внешний вид скрипта для работы с текстовым файлом

**Файл index.php**

<?php

require\_once ('function.php');

$params = array('name','number','status');

require\_once ('params.php');

include 'head.php';

$a = read ();

echo '<form action="add.php" method="post">

Имя: <input type="text" name="name"

value="'.$name.'" size="30" maxlength="30">

Число: <input type="text" name="number"

value="'.intval($number).'" size="6"

maxlength="6">

<input type="submit" value="Добавить">

<a href="index.php">Очистить</a>

<a href="sort.php">Сортировать</a></form>';

$status\_msg = array (

'Щелкните по Имени для редактирования

или по Числу для удаления записи',

'Запись уже существует, исправлено

значение в ней',

'Не удалось найти запись с

указанным номером',

'Не переданы данные для добавления'

);

if (empty($status)) $status = 0;

echo '<p>'.$status\_msg[$status].'</p>';

if (count($a)>0) {

echo '<table cellpadding="4"

cellspacing="0" border="1">'."\n".

'<tr><th>Имя</th><th>Число</th></tr>'."\n";

foreach ($a as $index=>$item) {

echo '<tr>'."\n";

list ($name, $number)=explode (DIV,$item);

echo '<td><a href="edit.php?id='.$index.'">

<span title="Править">'.

htmlspecialchars(trimall($name)).

'</span></a></td>'."\n".

'<td><a href="del.php?id='.$index.'">

<span title="Удалить">'.

intval(trimall($number)).

'</span></a></td>'."\n".

'</td></tr>'."\n";

}

echo "</table>"."\n";

}

include 'foot.php';

?>

Теперь займемся недостающими модулями. У файла add.php, кажется, простая задача - получить от index.php переменные $name и $number и записать их в файл. Однако модуль должен проверить, что ему переданы непустые данные, а также уметь отличать ситуацию, когда введено уже существующее в базе имя от ввода новой записи (см. if ($id>-1)   
{ ... } else { ... } в коде). В последнем случае запись всегда добавляется в конец, так как будет реализован модуль сортировки строк по алфавиту.

Также важно, что строка $name, "пропущенная" через обработчик параметров, уже лишена "критичных" для разметки символов вроде ', ", < и >, а в массиве $a, прочитанном из файла данных, все строки лежат "как есть", и строка <>123 не будет найдена, если $name после обработки превратилась &lt;&gt;123. Поэтому функции поиска записи, названной нами get\_index\_by\_name, передается строка, преобразованная "обратно" к первоначальному виду с помощью функции htmlspecialchars\_decode (доступна, начиная с версии PHP 5.1). В том же виде строка возвращается обратно в index.php через массив $\_SESSION. Это обеспечит некоторое удобство работы - после ввода новой записи ее данные останутся в форме и можно будет внести еще одну запись, мало отличающуюся по имени ("Иванова" после "Иванов").

Сам возврат из модуля в модуль реализован типовым образом через стандартную функцию header. Помните, что ее можно применять, только если модуль еще ничего не выводил в браузер.

**Файл add.php**

<?php

require\_once ('function.php');

$params = array('name','number');

require\_once ('params.php');

$status = 0;

if (!empty($name) && isset($number)) {

$number = intval ($number);

$a = read();

$id = get\_index\_by\_name

($a, htmlspecialchars\_decode($name));

$new\_string =

htmlspecialchars\_decode($name).DIV.

intval($number);

if ($id>-1) {

list($name,$number0)=explode (DIV,$a[$id]);

$a[$id] = $new\_string;;

$status=1;

}

else {

$id = count($a)+1;

array\_push ($a,$new\_string);

}

write ($a);

$\_SESSION['id'] = intval($id);

}

else $status=3;

$\_SESSION['name'] = htmlspecialchars\_decode($name);

$\_SESSION['number'] = intval($number);

header ('Location:

index.php'.($status==0?'':'?status='.$status

));

?>

Редактирование записей будет реализовано в модуле edit.php. Модуль сам будет обработчиком данных, переданных через форму редактирования, а значит, должен отличать ситуацию, когда он только вызван, от той, когда пользователь нажал "Сохранить". Последней задаче служит проверка

if (!empty($\_POST['submit']) && !empty($name)

&&isset($number) &&isset($id)) { ... }

определяющая, была ли нажата кнопка и переданы все данные. Втораяветка -

else if (isset($a[$id])) { ... }

предназначена для ситуации, когда в массиве $a есть запись, номер которой передан скрипту, и она должна быть отредактирована. Номер записи сохраняется в скрытом HTML-поле <input type="hidden">.

**Файл edit.php**

<?php

require\_once ('function.php');

$params = array('id','name','number');

require\_once ('params.php');

$a = read ();

$status = 0;

if (!empty($\_POST['submit']) && !empty($name)

&& isset($number) && isset($id)) {

$a = read ();

$a[$id] =

htmlspecialchars\_decode($name).DIV.

intval($number);

write ($a);

}

else if (isset($a[$id])) {

list ($name,$number)=explode(DIV, $a[$id]);

include 'head.php';

echo '<form action="edit.php" method="post">

<input type="hidden" name="id"

value="'.$id.'">

Имя: <input type="text" name="name"

value="'.htmlspecialchars(trimall($name)).

'" size="30" maxlength="30">

Число: <input type="text" name="number"

value="'.intval(trimall($number)).'"

size="6" maxlength="6">

<input type="submit" name="submit"

value="Сохранить">

<a href="index.php">Назад</a></form>';

include 'foot.php';

exit (0);

}

else $status = 2;

header ('Location:

index.php'.($status==0?'':'?status='.$status

));

?>

Модуль удаления записи del.php будет достаточно простым, все, что ему понадобится - получить допустимый $id записи (номер элемента в массиве $a), убрать соответствующий элемент из массива, переписать файл и вернуться на страницу главного модуля.

**Файл del.php**

<?php

require\_once ('function.php');

$params = array('id');

require\_once ('params.php');

$status = 0;

if (isset($id)) {

$a=read();

if (isset($a[$id])) unset ($a[$id]);

else $status = 2;

write ($a);

}

header ('Location:

index.php'.($status==0?'':'?status='.$status

));

?>

Наконец, модуль сортировки sort.php потребует решения новой проблемы - как сортировать строки в Юникоде по алфавиту, не различая строчных и прописных букв? "Прямая" сортировка с помощью стандартной функции sort подойдет едва ли - она считает строчную и прописную букву разными символами. Локаль мы не устанавливали, тем более, что для ее установки единой формы записи для всех операционных систем не существует.

Ограничимся тем, что из всего многообразия функций сортировки массивов выберем usort с пользовательской функцией сравнения элементов.

**Файл sort.php**

<?php

require\_once ('function.php');

$a = read();

function cmp ($a0,$b0) {

$a = mb\_strtolower(trimall($a0),'UTF-8');

$b = mb\_strtolower(trimall($b0),'UTF-8');

$alen = mb\_strlen ($a,'UTF-8');

$blen = mb\_strlen ($b,'UTF-8');

for ($i=0; $i<min($alen,$blen); $i++) {

$ca = mb\_substr ($a, $i, 1, 'UTF-8');

$cb = mb\_substr ($b, $i, 1, 'UTF-8');

if ($ca<$cb) return -1;

else if ($ca>$cb) return 1;

}

if ($alen<$blen) return -1;

else if ($alen>$blen) return 1;

else return 0;

}

usort ($a,'cmp');

write ($a);

header ('Location: index.php');

?>

Нам остается создать в папке файл с именем .htaccess, где мы пропишем кодировкой по умолчанию Юникод и укажем директивы настройки кавычек для приложения.

**Файл .htaccess**

AddDefaultCharsetutf-8

php\_flag magic\_quotes\_gpc off

php\_flag magic\_quotes\_runtime off

php\_flag magic\_quotes\_sybase off

В этой же папке создадим пустой (0 байт) файл с именем data.txt (необязательно, если все права настроены).

На практике не следует забывать о теоретической возможности потери данных при удаленной работе с файлами, особенно при записи в них. Например, если два клиента пытаются записать файл одновременно, изменения, сделанные как минимум одним из них, могут пропасть... а в худшем случае пропадает весь файл!

Уменьшить вероятность "падения" может установка клиентом исключительных прав на файл в то время, когда происходит запись. Перед записью файла следует блокировать его стандартной функцией PHP flock, а по окончании записи разблокировать, например, так:

$fp = fopen("/tmp/lock.txt", "w+");

if (flock($fp, LOCK\_EX)) {

//Если удалось "запереть" файл

fwrite($fp, "Что-нибудь пишем\n");

flock($fp, LOCK\_UN); // "Отпираем" файл

}

else {

echo "Ошибка блокировки файла на запись!";

}

fclose($fp);

Однако проблемы может создать и предшествующее записи чтение - представьте, что скрипт открывает файл, читает оттуда информацию в массив (точка 1), что-то добавляет в этот массив и пишет все обратно в файл (точка 2). Блокируется только запись. Клиент 2 находится в точке 2 и собирается записать информацию. Клиент 1 из-за блокировки не смог получить содержимое файла, поэтому прочитал пустой массив. Но потом-то он тоже дойдет до точки 2! Результат - содержимое файла поте­ряно.

В принципе, данной ситуации можно избежать, если написать функцию, которая "не пускает" пользователя дальше по скрипту до тех пор, пока файл не прочитается:

function read\_file($path){

if (!is\_file ($path)) { return false; }

elseif (!filesize ($path)) {

return array (); }

elseif ($array= file ($path)) {

return $array; }

else {

while (!$array= file ($path)) {

sleep (1);

}

return $array; }

}

Суть этого кода - пока файл блокирован на чтение, пользователь, который хочет прочитать из него данные, находится в цикле с функцией sleep, т.е. как бы "стоит на месте", ожидая, когда файл разблокируется на чтение.

Нужно заметить, что при действительно большой загруженности сервера эти приемы могут как не сработать, так и создать большую нагрузку на сервер, которая, в конце концов, его "повесит". Но для больших сайтов и не подойдет хранение информации в текстовых файлах, лучше поискать решение, основанное на базах данных.

Рекомендуемые по итогам главы разделы стандартной справки: права, "Справочник функций" - "Расширения для работы с файловой системой".

# 

# 2. Загрузка файлов и динамическое формирование изображений средствами PHP

Одна из самых привлекательных сторон web-приложения - возможность для пользователя загружать на сервер собственные файлы, которыми могут быть как изображения или видеофрагменты, так и архивы, офисные документы и т.п. Основой для загрузки файла служит управляющий элемент HTML<inputtype="file">, помещаемый в тег <form>. Конкретный вид элемента зависит от браузера, обычно он представляет собой однострочное текстовое поле для отображения имени загружаемого файла и кнопку, вызывающую стандартный диалог открытия файла. При этом обязательны следующие требования к HTML-форме:

* форма передается на сервер методом POST, т.е., в теге <form> указан атрибут method="post";
* втеге<form>указанатрибут enctype="multi­part/form-data".

Всю работу по проверке формата файла и его размещению на сервере должно выполнять web-приложение, при этом, некоторые настройки, сделанные в файле php.ini, могут помешать успешной загрузке пользовательского файла:

* настройка post\_max\_size определяет максимальный разрешенный размер файла для передачи методом POST;
* настройка upload\_max\_filesize задает максимальный размер загружаемого файла.

Критичными могут оказаться и настройки прав на доступ к папке, где размещаются загруженные файлы, а также к папке для временных файлов, определяемой переменной upload\_tmp\_dir файла php.ini. С настройкой прав тесно связана проблема безопасности скрипта. Следует понимать, что любая форма для загрузки пользователем файлов на ваш сайт потенциально опасна. Например, в одном из худших случаев, пользователь может загрузить shell-файл и получить доступ к вашему серверу. Чтобы такого не произошло, надо убрать права "остальным пользователям", т.е. [поставить права 666](http://pers.narod.ru/perl/iis_perl_setup.html#problems) на папку, в которую пользователи загружают файлы. По умолчанию там обычно установлены права 755. Остановимся на настройке прав на папки и файлы подробнее.

Предполагается, что Вы закачиваете скрипты на сервер Интернет через службу передачи файлов FTP любым FTP-клиентом, имеющим необходимый минимум настроек. В большинстве случаев для этой задачи достаточно клиента, встроенного в менеджер файлов [Far Manager](http://nickolay.info/text/coding_and_far2.html), рассмотрим работу с FTP его средствами. Нажав в нужной панели Far комбинацию клавиш Alt+F1 или Alt+F2, выбираем под списком дисков пункт меню FTP. Если нужного соединения в панели еще нет, создаем его нажатием комбинации клавиш Shift+F4. Обычно достаточно ввести в первое поле окна "Ввод адреса FTP" строку вида ftp://пользователь:пароль@сервер, например,ftp://misha314:pg0\_xE1@ftp.narod.ru для пользователя сервиса Народ.Ру с логином misha314 и паролем pg0\_xE1. Платные хостинги, как правило, также предоставляют данные для доступа к FTP. Для многих серверов также нужно выставить опции "Пассивный режим", в целях безопасности рекомендуется также "Использовать Firewall", опия "Расширенное окно команд" включена по умолчанию. Введя данные, нажимаем "Сохранить" и соединение сохраняется в панели FTP. Перейдя по его ссылке, теперь мы можем закачивать файлы на сервер, создавать там папки и менять права доступа к этим объектам.

Для смены прав в панели FTP Far Manager выделите нужные файлы клавишей Ins и нажмите комбинацию клавиш Ctrl+A. Появится окно, изображенное на рис. 5.

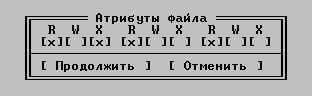


Рис. 5. Окно настройки атрибутов файла FarManager

Права доступа к файлам действуют во всех Unix-сов­мести­мых операционных системах, которые обычно установлены на серверах Интернет и разделяются на три категории:

* права владельца файла (u);
* права группы (g);
* права всех остальных пользователей (o).

Именно в этом порядке на рисунке показаны три группы переключателей. В каждой категории можно установить или снять три вида прав:

* R - право на чтение;
* W - право на запись;
* X - право на исполнение.

Права доступа обозначаются цифрами:

* 4 - читать (R);
* 2 - записывать (W);
* 1 - исполнять (X);
* 0 - ничего нельзя делать.

Эти цифры представляют собой двоичные коды комбинаций RWX, а их сумма и показывает права доступа. Вот основные варианты:

* 7 = 4 + 2 +1 (полные права, все чекбоксы включены);
* 6 = 4 + 2 (чтение и запись, включены R, W, выключен X);
* 5 = 4 + 1 (чтение и выполнение, включены R, X, выключен W);
* 4 = 4 + 0 + 0 (только чтение)

и т.д. для каждой из трех категорий. В результате получается, что права 777 обозначают, что и владелец, и группа пользователей, к которой он относится, и любой сторонний пользователь имеют полный доступ к этим файлам, как в Windows с правами администратора. Поэтому ваш скрипт таких прав требовать не должен, хотя при размещении приложения на локальном хосте и локальном сервере Windows допустимо установить права 777 для гостевой учетной записи интернета и учетной записи для запуска сервера.

Как правило, на реальном хостинге корневая папка Вашего сервера уже создана и права на нее настроены. Обычно требуются права 755 (положение переключателей XXX X0X X0X) для папок и исполняемых файлов, и 644 (X0X X00 X00) для всех остальных файлов, в том числе, скриптов с расширением .php. На рис. 5 показаны права 644 доступа к файлу.

Следует помнить, что для успешной загрузки файлов должны быть такие же права на временную папку PHP, узнать которую можно из настройки upload\_tmp\_dir файла php.ini. На хостинге обычно все уже настроено и настройка пуста.

Для работы с безопасной версий протокола FTP, называемой SFTP, существует расширение WinSCP для FarManager, работа с ним аналогична описанной.

Вся информация о загруженном на сервер файле сохраняется в глобальном массиве $\_FILES. Предположим, что файл загружен с помощью HTML-элемента <inputtype="file" name="myfile">. Тогда массив $\_FILES  имеет следующие элементы:

* $\_FILES['myfile']['name'] - имя, которое имел файл на машине клиента;
* $\_FILES['myfile']['type'] - mime-тип отправленного файла, например text/html, если браузер предоставил эту информацию;
* $\_FILES['myfile']['size'] - размер загруженного файла в байтах;
* $\_FILES['myfile']['tmp\_name'] - временное имя файла, под которым он был сохранен на сервере;
* $\_FILES['myfile']['error'] - код ошибки, появившейся при загрузке.

При загрузке в PHP выделяют нескоько типов ошибок, соответственно, значение $\_FILES['myfile']['error'] может быть одним из следующих:

* 0 - ошибки не произошло, файл загружен успешно;
* 1 - загружаемый файл превышает размер, установленный директивой upload\_max\_filesize в файле настроек php.ini;
* 2 - загружаемый файл превышает размер, установленный скрытым элементом MAX\_FILE\_SIZE формы HTML (например, <inputtype="hidden" name="MAX\_FILE\_SIZE" value="300000">, возможность работает не во всех браузерах);
* 3 - файл был загружен частично;
* 4 - файл загружен не был;
* 6 - отсутствует временная папка;
* 7 - не удалось записать файл на диск;
* 8 - одно из PHP-расширений остановило загрузку файла.

По умолчанию загруженные файлы сохраняются во временной директории сервера, если другая директория не указана с помощью опции upload\_tmp\_dir в файле настроек php.ini. Переместить загруженный файл в нужную директорию можно с помощью функции bool move\_up­loaded\_file (временное\_имя\_файла, место\_назна­чения). Эта функция проверяет, действительно ли файл, обозначенный строкой временное\_имя\_файла, был загружен через механизм загрузки HTTP методом POST. Если это так, то файл перемещается в файл, заданный параметром место\_назначения (этот параметр содержит как путь к новой директории для хранения, так и новое имя файла).

Если временное\_имя\_файла задает неправильный загруженный файл, то никаких действий произведено не будет, и move\_uploaded\_file вернет false. То же самое произойдет, если файл по каким-то причинам не может быть перемещен. В этом случае интерпретатор выведет соответствующее предупреждение. Если файл, заданный параметром место\_наз­начения, существует, то функция move\_uploaded\_file перезапишет его.

**Пример 6.** Реализуем законченное приложение для загрузки на сервер файлов с изображениями. Как минимум, такой код должен выполнять следующие функции:

* загружать файлы в выбранную настройками папку;
* проверять ограничения на размер файла, установленные в настройках скрипта и в настройках PHP;
* отличать картинки от прочих файлов;
* проверять ограничения на ширину и высоту картинок;
* при необходимости сжимать картинки до указанных в настройках размеров, используя библиотеку GD;
* выдавать осмысленные сообщения об ошибках.

Сначала все задаваемые пользователем настройки соберем в файл config.php:

<?php

define ("MAX\_WIDTH","800");

//Макс. ширина картинки в пикселах

define ("MAX\_HEIGHT","800");

//Макс. высота картинки в пикселах

define ("FOLDER","./uploads");

//Путь к папке для картинок,

//без слеша в конце, латинские буквы

define ("MAX\_SIZE","1048576");

//Максимальный размер файла, байт

define ("USE\_GDLIB","1");

//Использовать библиотеку GD

//(php\_gd2.dll) для масштабирования больших

//изображений: 1 - включить, 0 - выключить

?>

Так как нашему скрипту параметры будут передаваться только методом POST, для обработки параметров достаточно следующего кода (файл params.php):

<?php

while (list($num,$var) = each($params)) {

if (!empty($\_POST[$var]))

$$var = htmlspecialchars($\_POST[$var]);

else $$var = '';

}

?>

Список разрешенных параметров будет задан в главном файле index.php, пока таковой понадобится только один - имя кнопки "Загрузить".

Напишем несколько полезных для поставленной задачи функций и включим их в файл functions.php.

Во-первых, это функция gdVersion для определения версии библиотеки GD. Для надежности потребуем версию не ниже 2, что соответствует PHP 5 версии.

Функция tumbmaker будет заниматься обработкой и масштабированием изображений, ее заголовок таков: function tumbmaker ($src, $dest, $width, $height, $rgb=0xFFFFFF, $quality=65).

Параметры $src и $dest - ссылки на исходный и целевой файл, $width и $height - требуемые ширина и высота рисунка в пикселах, параметр $rgb задает фоновый цвет нового изображения (по умолчанию принят белый), а параметр $quality определяет качество генерируемого изображения в формате JPEG (по умолчанию 65%).

Эта функция преобразует размерности к тем, что мы указали. Чтобы вычислить "правильные" размерности и пропорционально сжать рисунок, напишем маленькую вспомогательную функцию function get\_new\_size($width, $height, $max\_width, $max\_height).

Она пропорционально преобразует размеры $width и $height с учетом размеров максимального прямоугольника, куда разрешено вписывать рисунок. Размеры заданы параметрами $max\_width и $max\_height.

Четвертая функция - let\_to\_num - будет переводить размерности, хранящиеся в файле php.ini, в количество байт, поскольку, 2 мегабайта, например, там обозначены как 2M.

Наконец, функция cyr2lat будет заменять в переданном ей тексте символы кириллицы на аналогичные латинские буквы, поскольку далеко не каждый хостинг сможет записать файл с кириллицей в имени.

Ниже приведен код файла functions.php:

<?php

require\_once "config.php";

function gdVersion() {

$gdv=@gd\_info();

$ver=$gdv['GD Version'];

$v=0;

if (preg\_match(

"/.\*([0-9]+)\.([0-9]+)\.([0-9]+).\*/",

$ver, $r)) $v=$r[1];

return $v;

}

function get\_new\_size ($width, $height,

$max\_width, $max\_height) {

$w=$width; $h=$height;

$dw=$max\_width/$width;

$dh=$max\_height/$height;

if ($width>$max\_width and

$height>$max\_height) {

if ($dw<$dh) { $

w=$max\_width; $h=floor($height\*$dw);

}

else {

$h=$max\_height; $w=floor($width\*$dh);

}

}

else if ($width>$max\_width and

$height<=$max\_height) {

$w=$max\_width; $h=floor($height\*$dw);

}

else if ($width<=$max\_width and

$height>$max\_height) {

$h=$max\_height; $w=floor($width\*$dh);

}

return array ($w, $h);

}

function tumbmaker ($src, $dest, $width, $height, $rgb=0xFFFFFF, $quality=65) {

if (!file\_exists($src)) return false;

$size = getimagesize($src);

if ($size === false) return false;

$format =

strtolower(substr($size['mime'],

strpos($size['mime'], '/')+1));

$icfunc = "imagecreatefrom" . $format;

if (!function\_exists($icfunc))

return false;

$x\_ratio = $width / $size[0];

$y\_ratio = $height / $size[1];

$ratio = min($x\_ratio, $y\_ratio);

$use\_x\_ratio = ($x\_ratio == $ratio);

$new\_width = $use\_x\_ratio ? $width :

floor($size[0] \* $ratio);

$new\_height = !$use\_x\_ratio ? $height :

floor($size[1] \* $ratio);

$new\_left = $use\_x\_ratio ? 0 :

floor(($width - $new\_width) / 2);

$new\_top = !$use\_x\_ratio ? 0 :

floor(($height - $new\_height) / 2);

$isrc = $icfunc($src);

$idest = imagecreatetruecolor

($width, $height);

imagefill ($idest, 0, 0, $rgb);

imagecopyresampled ($idest, $isrc,

$new\_left, $new\_top, 0, 0,$new\_width,

$new\_height, $size[0], $size[1]);

imagejpeg($idest, $dest, $quality);

imagedestroy($isrc);

imagedestroy($idest);

return true;

}

function let\_to\_num($v){

$l = substr($v, -1);

$ret = substr($v, 0, -1);

switch(strtoupper($l)){

case 'P': $ret \*= 1024;

case 'T': $ret \*= 1024;

case 'G': $ret \*= 1024;

case 'M': $ret \*= 1024;

case 'K': $ret \*= 1024;

break;

}

return $ret;

}

function cyr2lat ($text) {

$cyr2lat\_replacements = array (

"А"=>"a","Б"=>"b","В"=>"v","Г"=>"g","Д"=>"d",

"Е"=>"e","Ё"=>"yo","Ж"=>"dg","З"=>"z","И"=>"i",

"Й"=>"y","К"=>"k","Л"=>"l","М"=>"m","Н"=>"n",

"О"=>"o","П"=>"p","Р"=>"r","С"=>"s","Т"=>"t",

"У"=>"u","Ф"=>"f","Х"=>"h","Ц"=>"ts","Ч"=>"ch",

"Ш"=>"sh","Щ"=>"csh","Ъ"=>"","Ы"=>"i","Ь"=>"",

"Э"=>"e","Ю"=>"yu","Я"=>"ya",

"а"=>"a","б"=>"b","в"=>"v","г"=>"g","д"=>"d",

"е"=>"e","ё"=>"yo","ж"=>"dg","з"=>"z","и"=>"i",

"й"=>"y","к"=>"k","л"=>"l","м"=>"m","н"=>"n",

"о"=>"o","п"=>"p","р"=>"r","с"=>"s","т"=>"t",

"у"=>"u","ф"=>"f","х"=>"h","ц"=>"ts","ч"=>"ch",

"ш"=>"sh","щ"=>"sch","ъ"=>"","ы"=>"i","ь"=>"",

"э"=>"e","ю"=>"yu","я"=>"ya",

"-"=>"\_"," "=>"\_" );

return strtr ($text, $cyr2lat\_replacements);

}

?>

Осталось написать сам скрипт и поместить его в файл index.php. После формирования обычной титульной части html-файла:

<html>

<head>

<meta http-equiv="Content-Type"

content="text/html; charset=windows-1251">

<title>Сервис для загрузки файла</title>

</head>

<body><div align="center">

подключим файл с функциями и наш обработчик параметров:

<?php

require\_once "functions.php";

$params = array ('ok');

require\_once ("params.php");

?>

Форма для загрузки файла будет включать только соответствующий элемент HTML и кнопку "Загрузить":

<form action="index.php" method="post"

enctype="multipart/form-data">

<input type="file" name="url">

<input type="submit" name="ok"

value="Загрузить"></form>

Вся остальная часть файла, если не считать закрывающих тегов </div></body></html> в самом конце, будет состоять из одного блока PHP-кода, основные действия которого мы сейчас напишем.

Если форма была отправлена кнопкой "Загрузить", в параметрах должна присутствовать переменная $ok, если ее нет - вернуться:

if (empty($ok)) return;

Далее установим служебные переменные - сообщение об ошибке $error, временное имя переданного файла $file, флаг загрузки файла $load, флаг $need\_move для функции загрузки временного файла move\_uploaded\_file (дело в том, что при масштабировании рисунка он может быть загружен и функцией tumbmaker, тогда стандартная move\_up­loaded\_file не нужна):

$error = $file = '';

$load = true;

$need\_move= true;

Все остальное поместим в блок условия, проверяющего, что файл был передан:

if (isset($\_FILES['url'])

and !empty($\_FILES['url']['tmp\_name'])) {

//здесь будет остальной код

}

else {

$error .= '<br>Файл не передан.';

if (!empty($\_FILES['url']['error']))

$error.=' Код ошибки: '.

$\_FILES['url']['error'];

}

Получаем временное имя файла и его размер в байтах, а также ограничение на размер - это минимальная из величин MAX\_SIZE (наше собственное ограничение из конфигурации), post\_max\_size и upload\_max\_filesize. Если размер превышен, формируем соответствующее сообщение:

$file = $\_FILES['url']['tmp\_name'];

$file\_size = $\_FILES['url']['size'];

$php\_max\_size =

min(let\_to\_num(ini\_get('post\_max\_size')),

let\_to\_num(ini\_get('upload\_max\_filesize')));

$max\_size=min(MAX\_SIZE,$php\_max\_size);

if ($file\_size > $max\_size) {

$error .= '<br>Слишком большой файл'.

'Ограничение в настройках '.

($max\_size==MAX\_SIZE?'скрипта':'PHP').

': '.$max\_size.' байт';

$load=false;

}

if ($\_FILES['url']['error']!=UPLOAD\_ERR\_OK) {

$error .= '<br>Сервер вернул ошибку. Код: '.

$\_FILES['url']['error'];

$load=false;

}

Извлекаем тип переданного файла, формируем путь к записываемому файлу $filename, при этом имя файла, возможно, будет изменено методом cyr2lat:

$path\_parts =

pathinfo($\_FILES['url']['name']);

$filetype =

strtolower($path\_parts['extension']);

$filename = FOLDER.'/'.

cyr2lat(strtolower($path\_parts['basename']));

По типу файла проверяем, загружается ли картинка. Следует понимать, что этот способ - не самый надежный. Дляпроверкивведемфлаг $is\_pic:

$is\_pic = true;

switch ($filetype) {

case "jpeg":

case "jpg":

case "gif":

case "png": break;

default: $is\_pic=false;

}

Если это картинка, проверяем ее размер, при необходимости пытаемся сжать, если включена настройка USE\_GDLIB и библиотека доступна. Если сжать рисунок удалось, ставим $need\_move=false, чтобы не грузить рисунок повторно методом move\_uploaded\_file:

if ($is\_pic) {

$size = getimagesize($file);

$width = $size[0]; $height = $size[1];

if ($width>MAX\_WIDTH or $height>MAX\_HEIGHT) {

$ver = gdVersion();

if (USE\_GDLIB=='1') {

if ($ver>1) {

$new\_size = get\_new\_size($width, $height,

MAX\_WIDTH,MAX\_HEIGHT);

$r=tumbmaker ($file,$filename,

$new\_size[0],$new\_size[1]);

if (!$r) {

$error .=

'<br>Не удалось уменьшить изображение';

$load=false;

}

else $need\_move=false;

}

else {

$error .=

'<br>Библиотека GD недоступна';

$load=false;

}

}

else {

$error .= '<br>Слишкомбольшиеразмеры';

$load=false;

}

}

}

После этого, если установлены флаги $load и $need\_move, можно загрузить картинку стандартным методом и установить на нее права 644:

if ($load) {

if ($need\_move) {

if (move\_uploaded\_file ($file,$filename)) {

chmod ($filename,0644);

}

else {

$error .= '<br>Неудалосьзакачать '.

$filename.' извременного '.

$file.'<br>Информация для отладки: ';

print\_r($\_FILES);

}

}

}

В конце основного блока кода можно проверить, удален ли временный файл (функция move\_uploaded\_file должна удалить его сама после успешной загрузки), проверить, не нужно ли вывести сообщение об ошибке, если нет - дать ссылку на загруженный файл:

if (file\_exists($file)) @unlink ($file);

if (!empty($error)) {

print 'Возникли проблемы!'.$error;

}

else {

print 'Файл загружен, вот он: '.

' <a href="'.$filename.

'" target="\_blank">'.$filename.'</a>';

}

Внешний вид приложения после загрузки файла в браузере Google Chrome показан на рис. 6.

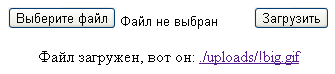


Рис. 6. Внешний вид приложения для загрузки файла

Загружать не графические файлы можно теми же приемами кодирования, исключив код для масштабирования рисунков.

В реальном скрипте к возможностям приложения следует добавить авторизацию пользователя, контроль лимита общего объема загруженных файлов, возможность работать с их списком и т.п. сервисные функции. Помещать файлы на сервер можно "как обычно", но под именем, полученным скриптом (например, функцией uniqid), а настоящее имя файла (если оно нужно) и MIME-тип хранить в базе данных. При этом "отдавать" файлы тоже лучше не "напрямую" по ссылке, а скриптом, написав для этой цели соответствующий класс.

Второй актуальной задачей, связанной с библиотекой GD, является генерация изображений на стороне сервера. Ее основные применения следующие:

* защита web-приложений от программного обращения к файлу скрипта, минующего форму. Поскольку любые данные, передаваемые по открытому протоколу HTTP, можно подделать, такая защита необходима в большинстве случаев. Самый распространенный ее способ - генерация вместе с HTML-формой специального изображения - "капчи" (англ. captcha), которое может быть распознано человеком, но не программой. Обычно пользователь-человек должен ввести в форму сочетание символов, прочитанное им на специально "зашумленной", чтобы уменьшить вероятность машинного распознавания, картинке;
* генерация изображений по заданным правилам, применяемая, например, для создания фотоэффектов, градиентов, динамических картинок-информеров для web-страниц и т.п.

Для решения этих задач используются функции библиотеки GD, работу с которыми мы рассмотрим на примерах.

**Пример 7.** Показанный ниже код source.php генерирует простую графическую капчу, сохраняет ее код в сессии, а потом передает файлу-приемнику destination.php. При этом используется "библиотечный" файл img.php, который, собственно, и генерирует картинку.

Для работы приложения требуется подключенная библиотека GD на сервере и включенный прием cookie-файлов в барузере.

**Файл source.php**

<?php

$alpha =

"ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";

//символыдлякапчи

$secret = ""; //ключдлякапчи

for($i=0;$i<5;$i++)

$secret.=$alpha[rand(0,strlen($alpha)-1)];

//Делаем 5 символов

session\_id(md5(microtime()\*rand()));

//и новый случайный SESSIONID

session\_start(); //Запускаем сессию

$\_SESSION['secret'] = $secret;

//и сохраняем в ней ключ

?>

<!DOCTYPE html PUBLIC

"-//W3C//DTD HTML 4.01 Transitional//EN">

<html><head>

<meta content=

"text/html; charset=Windows-1251"

http-equiv="content-type">

<title>Отправка формы с капчей</title>

</head><body>

<form action="destination.php" method="post">

<p>Имя: <input type="text" name="name"></p>

<p>Пароль: <input type="text" name="pass">

</p><p>Число на картинке :

<input type="text" name="secret"></p>

<p><input type="submit" name="send"

value="Отправить"></p>

</form><img

src="img.php?sid=<?php echo session\_id()?>">

</body></html>

**Файл destination.php**

<?php

session\_start();

?>

<!DOCTYPE html PUBLIC

"-//W3C//DTD HTML 4.01 Transitional//EN">

<html><head>

<meta content=

"text/html; charset=Windows-1251"

http-equiv="content-type">

<title>Прием формы</title>

</head><body>

<?php

if (strtoupper($\_POST['secret']) ==

$\_SESSION['secret']) {

echo '<p>Имя = '.$\_POST['name'].

'<br>Пароль = '.$\_POST['pass'].'</p>';

}

else {

echo '<p>Неверный источник данных

или неправильный ввод кода</p>';

}

?>

</body></html>

**Файл img.php**

<?php

session\_id($\_GET['sid']);

//Запусксессиисполученным SESSIONID

session\_start();

$img = imagecreate(88,31);

//Пустая картинка указанного размера

imageColorAllocate($img,204,204,204);

//Фон картинки

$color2 = imageColorAllocate($img,0,0,0);

//Цвет линий

imageline($img,10,0,87,31,$color2);

//Дополнительные линии,

//чтоб затруднить автораспознавание

imageline($img,0,20,60,0,$color2);

imageline($img,40,31,87,11,$color2);

imageline($img,0,31,87,0,$color2);

$color = imageColorAllocate($img,0,128,0);

//Цвет букв

imagestring ($img,6,15,8,

$\_SESSION['secret'],$color); //Вывод текста

imageGif($img); //Показать картинку

header("Content-Type: image/gif");

//и вернуть нужный тип контента

?>

**Пример 8.** Здесь функции библиотеки GD применяются для генерации квадратного рисунка-градиента заданного размера. Код функции gdVersion следует добавить из примера 6.

<?php

if (gdVersion()<2)

exit ("Need GD 2 or higher");

$img = imagecreatetruecolor(255, 255);

for ($i=0; $i<255; $i++)

for ($j=0; $j<255; $j++) {

$r=255-round(sqrt(pow($i-127,2)+

pow($j-127,2)));

$color=imagecolorallocate($img, $r, $r, $r);

imagesetpixel ($img, $i, $j, $color);

}

header('Content-Type: image/jpeg');

imagejpeg($img);

imagedestroy($img);

?>

Рекомендуемые по итогам главы разделы стандартной справки: "Справочник функций" - "Другие службы" - "FTP", "Справочник функций" - "GD и функции для работы с изображениями".

# 3. Использование СУБД MySQL в PHP

Одной из сильных сторон PHP является легкая интеграция написанных на нем приложений с базами данных. При этом интерпретатор, подключая различные написанные для этой цели расширения, способен работать с СУБД различных производителей. Самой распространенной СУБД для PHP является открытая реляционная система управления базами данных MySQL, созданная корпорацией Oracle. В отличие от более мощных версий SQL, MySQL не имеет курсоров, хранимых процедур и некоторых других возможностей, но для типовой задачи хранения базы данных сайта СУБД MySQL подходит хорошо. Принципы работы с СУБД остальных производителей отличаются незначительно.

Для работы с MySQL требуется установленное на жестком диске и подключенное в файле php.ini расширение mysql. Например, если php.ini находится в папке c:\WINDOWS, а PHP установлен в папку c:\PHP, строка подключения расширения в файле конфигурации может быть такой:

extension=../php/ext/php\_mysql.dll

В общем случае лучше полагаться не на относительные пути к файлам, а на настройку директивы extension\_dir в файле php.ini.

Как правило, для управления базами данных также   
нужно приложение phpMyAdmin, напомню, что при установленном пакете Denwer оно доступно по URL-адресу <http://localhost/tools/phpmyadmin/>.

После установки в приложение phpMyAdmin можно зайти с логином root и аналогичным паролем (в некоторых сборках Denwer пароль может быть пустым). Выбрав в меню пункт "Базы данных" введем в поле "Новая база данных" название базы my и укажем в списке "Сравнение" тип сопоставления строк utf8\_general\_ci. Это означает, что наши данные будут сохраняться в кодировке Юникод, а при сравнении строк средствами MySQL игнорируется регистр символов.

Будет создана пустая база данных с указанным именем. Выбрав ее слева в списке доступных баз, щелкнем по пункту меню SQL и выполним следующие запросы на создание таблиц базы данных (в текстовое поле можно вставить все запросы   
сразу):

create table users (

id int primary key auto\_increment,

login varchar(32),

password varchar(32)

);

create table messages (

id int primary key auto\_increment,

title varchar(80),

text text

);

insert into users (id,login,password) values

(1,'Admin','202cb962ac59075b964b07152d234b70');

insert into messages (id,title,text) values

(1,'Запись 1','Текст записи 1'),

(2,'Запись 2','Текст записи 2')

Как видно из текста запросов, они создают две таблицы - users и messages, предназначенные, соответственно, для хранения информации о пользователях и сообщениях, а также вносят два тестовых сообщения и вписывают в таблицу пользователей логин и пароль для входа администратора в будущее приложение. Для удобства обычно сохраняют все нужные для создания базы запросы в отдельном файле с расширением .sql.

Пароль администратора, в учебных целях равный строке 123, хранится в зашифрованном виде, который получен следующим кодом:

<?php

echomd5('123');

?>

Для работы с базой данных требуется задать в скрипте следующие настройки:

* имя хоста (сервера баз данных): обычно localhost, но на реальном хостинге возможно и другое значение, указанное провайдером;
* логин пользователя БД: на локальном хосте обычно root, на удаленном сервере - логин, полученный от провай­дера;
* пароль пользователя БД: на локальном хосте обычно пуст или также root, на удаленном сервере - пароль, полученный от провайдера;
* имя базы данных: в нашем случае, задано строкой my;
* имя таблицы или общий префикс всех таблиц, если БД достаточно сложна: в нашем случае, соответствует имени таблицы messages, содержащей сообщения.

Обычный порядок работы с этими данными таков:

* подключиться к базе данных стандартной функцией mysql\_connect, передав ей параметрами имя хоста, логин и пароль;
* выбрать нужную таблицу, передав ее имя параметром функции mysql\_select\_db;
* сформировать строку SQL-запроса, записав ее, например, в переменную $sql;
* передать эту строку параметром функции mysql\_query, сохранив результат, возвращенный функцией, например, в переменной $result;
* если переменная $result пуста, запрос не выполнен, в противном случае узнать количество записей базы, возвращенных запросом, можно стандартной функцией mysql\_num\_rows ($result);
* в цикле поочередно извлечь и обработать записи одной из двух функций: mysql\_fetch\_assoc ($result), если запись запроса нужно вернуть как ассоциативный массив, или mysql\_fetch\_array ($result), если достаточно массива с числовыми индексами. Например, при выполнении запроса select \* from messages первая функция будет возвращать записи сообщений в виде элементов массива с индексами id, title и text, а вторая - с индексами 0, 1 и 2.

Приведем законченный пример кода, извлекающего записи из таблицы messages и выводящего их на страницу:

<?php

define ('DB\_HOST','localhost'); //имя хоста

define ('DB\_LOGIN','root'); //логин

define ('DB\_PASSWORD','root'); //пароль

define ('DB\_NAME','my'); //имя БД

define ('TAB\_NAME','messages'); //имя таблицы

mysql\_connect

(DB\_HOST, DB\_LOGIN, DB\_PASSWORD);

mysql\_select\_db (DB\_NAME);

$sql =

'select \* from '.TAB\_NAME.' order by title';

$result = mysql\_query ($sql);

if ($result) {

$n = mysql\_num\_rows ($result);

if ($n>0) {

$i=1;

while ($row=mysql\_fetch\_assoc($result)) {

echo '<br>'.($i++).'.

<b>'.$row['title'].'</b>'.

': '.$row['text'];

}

}

else { echo '<br>Void result!'; }

}

else { echo '<br>SQL error'; }

?>

Текст этого примера следует сохранить в кодировке Юникод. Нумерация выводимых записей производится с помощью переменной $i.

**Пример 9.** Показанный выше код вполне работоспособен, но не слишком удобен при написании реального приложения, состоящего из множества модулей, работающих с базой данных. Поэтому напишем развернутый пример, решающий следующие типовые задачи:

* корректная обработка параметров, переданных методами GET и POST с удалением специальных символов и лишних пробелов (переводов строк), а также корректная обработка кавычек (см. главу 3);
* наличие отдельного файла настроек, в котором будут прописаны параметры соединения с MySQL и, возможно, другие настройки;
* наличие файла-инсталлятора, который создаст базу данных и потом удалится (для простоты не будем удалять его программно);
* наличие файла-"обертки", предназначенного для выполнения операций с БД, по возможности, избегающего длинных имен функций и лишних переподключений к базе;
* наличие главного файла, предоставляющего форму для заполнения БД с простейшим контролем (например, не разрешать добавлять ту же самую запись по обновлению страницы после отправки данных).

К файлам проекта подключать будем только модуль functions.php, а он позаботится об остальных служебных файлах:

<?php

require\_once ("config.php");

require\_once ("params.php");

require\_once ("db.php");

function trimall ($string) {

$string=preg\_replace("/\r/","",trim($string));

$string=preg\_replace("/ +/"," ",$string);

$string=preg\_replace("/ \n/","\n",$string);

$string=preg\_replace("/\n\n+/","\n",$string);

return $string;

}

//включить код функции magic из главы 3

?>

Функция magic приведена в главе 3, а реализация trimall изменена так, чтобы в полученной параметром строке убирались только лишние разрывы строк (т.е. больше одного разрыва подряд).

Файл настроек config.php содержит все необходимые настройки приложения:

<?php

define ('DB\_HOST','localhost'); //хост

define ('DB\_LOGIN','root'); //логин

define ('DB\_PASS','root'); //пароль

define ('DB\_NAME','test'); //имя БД

define ('DB\_TABLE','messages'); //имя таблицы

define ('DB\_CODEPAGE','utf8');

//кодовая страница БД

?>

Из кода видно, что данные будут храниться в новой базе с именем test, которую создадим программно скриптом-инстал­лятором. Файл обработки параметров params.php может быть подключен из главы 3.

Файл операций с базой данных db.php позволит, во-первых, использовать один и тот же идентификатор соединения при последовательном выполнении нескольких запросов (статическая переменная $conid), а не переподключаться к базе каждый раз, во-вторых, предоставит более короткие имена для функций работы с базой:

<?php

if (!isset($conid)) {

function dbconnect() {

$mysql=

mysql\_connect(DB\_HOST, DB\_LOGIN, DB\_PASS);

mysql\_select\_db(DB\_NAME);

return $mysql;

}

function dbquery ($sql){

$result=mysql\_query($sql,get\_conid());

return $result;

}

function dbfetcha ($result){

if ($row=mysql\_fetch\_assoc ($result))

return $row;

else return false;

}

function dbfetch ($result){

if ($row=mysql\_fetch\_array($result))

return $row;

else return false;

}

function dbrows ($result){

$num=mysql\_num\_rows($result);

return $num;

}

function dbfree ($result){

mysql\_free\_result($result);

}

function dbclose ($conid) {

mysql\_close(get\_conid());

}

function get\_conid () {

static $conid=0;

if ($conid==0) {

$conid=dbconnect();

@mysql\_set\_charset (DB\_CODEPAGE,$conid);

}

return $conid;

}

get\_conid ();

}

?>

Основной демо-файл index.php, во-первых, проверит, присутствует ли в текущей папке скрипт-установщик install.php (и откажется запускаться, если этот файл не удален), во-вторых, предоставит форму для добавления сообщений и покажет имеющиеся записи БД.

<?php

$params = array('title','message');

require\_once ("functions.php");

//Формирование верхней части страницы:

print '<html><head>

<meta http-equiv="Content-Type"

content="text/html; charset=UTF-8">

<title>Демо</title></head>

<body bgcolor="#E5E5E5" text="#000000"

link="#006699" vlink="#5493B4">';

if (file\_exists('install.php')) {

print '<p>Выполните install.php '.

'из папки скрипта или удалите его, '.

'если выполнили его ранее.'.

'<a href="'.$\_SERVER['PHP\_SELF'].'">'.

'Обновить эту страницу</a>';

exit;

}

if (!empty($title) and !empty($message)) {

//Нужные параметры непусты?

$sql = 'select title,message from '.

DB\_TABLE.' where title="'.$title.

'" and message="'.$message.'"';

$result = dbquery($sql)

or die (mysql\_error());

if (dbrows($result)<1) {

//Добавить сообщение в базу:

$sql = 'insert into '.DB\_TABLE.

' (title,message) values ("'.$title.

'", "'.$message.'")';

$result=dbquery($sql) or die (mysql\_error());

}

else {

echo '<p>Такое сообщение уже есть!</p>';

}

}

//Вывод формы для добавления сообщений:

print '<form method="post"

action="'.$\_SERVER['PHP\_SELF'].'">

<p>Заголовок: <input type="text" size="40"

maxlength="80" name="title" value="'.

$title.'"></p>

<p>Сообщение: <textarea name="message"

rows="4" cols="72">'.

$message.'</textarea></p>

<p><input type="submit" value="Добавить">

</form>';

//Получить и показать имеющиеся записи БД:

$sql = 'select \* from '.DB\_TABLE.

' order by id desc';

//Сортируем по № записи в обратном порядке

$result=dbquery($sql) or die (mysql\_error());

if ($result and dbrows($result)) {

//Если есть непустой результат,

while ($data = dbfetcha($result)) {

//то пока есть записи,

print '<p><b>'.$data['title'].

'</b>: '; //вывестизаголовок

print $data['message'].'</p>'."\n";

//и сообщение с переводом строки

}

}

print '</body></html>';

?>

Пример простого, хотя и недостаточно безопасного вариант скрипта-установщика (инсталлятора) представляет собой показанный ниже файл-инсталлятор install.php. Он при своем запуске создаст базу данных и таблицу в ней (если база уже существовала, она удалится).

<?php

$params = array('action');

require\_once ("functions.php");

$sqls = array (

'DROP DATABASE IF EXISTS '.DB\_NAME.';',

'CREATE DATABASE '.DB\_NAME.

' CHARACTER SET utf8'.

' COLLATE utf8\_general\_ci;',

'USE '.DB\_NAME.';',

'DROP TABLE IF EXISTS '.DB\_TABLE.';',

'CREATE TABLE '.DB\_TABLE.

' (id int PRIMARY KEY auto\_increment, '.

'title varchar(80), message text);'

);

print '<html><head>

<meta http-equiv="Content-Type"

content="text/html; charset=UTF-8">

<title>ФормированиеБД</title></head>

<body bgcolor="#E5E5E5" text="#000000"

link="#006699" vlink="#5493B4">';

if (empty($action)) {

print '<p>Нажмите кнопку для создания БД,

после успешного завершения

удалите файл install.php

<form method="post"

action="'.$\_SERVER['PHP\_SELF'].'">

<input type="hidden" name="action" value="1">

<input type="submit" value="OK">

</form>';

}

else {

$error='';

for ($i=0; $i<count($sqls); $i++) {

$result = dbquery($sqls[$i])

or exit (mysql\_error());

if (!$result) {

$error.='<br>Не могу выполнить запрос: '.

'<font color="red">'.$sqls[$i].'</font>';

}

}

if (!empty($error)) {

print '<p>Возниклиошибки!'.$error.'</p>';

}

else {

print '<p>БДсоздана! '.

'<a href="index.php">Запустить</a></p>';

}

}

print '</body></html>';

?>

Предполагается, что все файлы проекта закодированы в Юникоде и находятся в одной папке на сервере, а приложение выполняется в версии PHP не ниже 5.3.0.

# 

# 4. Управление сеансами пользователя и безопасность скриптов

Большинство сложных web-приложений позволяют регистрацию и авторизацию пользователей на сайте. Основой для управления механизмами авторизации служат *сессии*. Сессии позволяют создавать и использовать переменные, сохраняющие свое значение в течение всего времени работы пользователя с сайтом. При этом у каждого пользователя сайта данные переменные будут собственными, т.е. их область видимости (англ. variable scope) распространяется на все время нахождения на сайте конкретного пользователя, причем для каждого захода пользователя на ваш сайт эти переменные будут различными.

В основе всего механизма сессий лежит решение задачи об идентификации того, от кого именно пришел запрос на сервер. Если это будет точно известно, то уже не возникнет большой проблемы в том, чтобы предоставить скрипту информацию, относящуюся именно к этому конкретному пользователю.

Данная задача решается путем присвоения каждой сессии уникального идентификатора SID (англ. Session IDentifier), который создается в тот момент, когда пользователь заходит на сайт, и уничтожается, когда пользователь уходит с сайта. Он представляет собой строку из 32 символов, например, ac4f4a45bdc893434c95dcaffb1c1811. SID передается на сервер вместе с каждым запросом клиента и возвращается обратно вместе с ответом сервера. Алгоритм генерации SID позволяет гарантировать его уникальность, поэтому исключена возможность того, что две сессии будут иметь один и тот же идентификатор сессии.

PHP может использовать два различных механизма в качестве транспортного средства для передачи SID:

* файлы cookies;
* дополнительный параметр URL-адреса.

Cookies (от англ. cookie - печенье) - это небольшие текстовые файлы с зашифрованной информацией, отправляемые веб-сервером и хранимые на компьютере пользователя в назначенной браузером для этой цели системной папке. Браузер всякий раз при попытке открыть страницу соответствующего сайта пересылает его cookie серверу в составе HTTP-запроса, при этом SID сохраняется "внутри" браузера и остается незаметным для пользователя. Поддержка cookies - необязательное условие для браузера, она может отсутствовать или быть отключена.

Все cookies, как минимум, имеют имя и значение, а отправить их программно можно функцией bool setcookie (string $name [, string $value]), имеющей еще несколько необязательных параметров. Прочитать имеющиеся для данной страницы cookies можно как элементы ассоциативного массива $\_COOKIE.

Функции для удаления cookie не существует, это происходит либо по истечении срока ее хранения, либо по явному вызову setcookie с указанием третьим параметром времени хранения, которое уже истекло:

setcookie ('my\_cookie','',time()-14\*24\*3600);

Следует также учесть, что вначале сервер направляет cookie клиенту как часть отклика HTTP, потом клиент, если он готов принять cookie, возвращает ее серверу. Поэтому для проверки того, подключены ли cookies в браузере клиента, скрипту может понадобиться программная перезагрузка страницы:

<?php

$my\_cookie = '';

if (!isset($\_GET['step'])) { //Первыйвызов

setcookie ('my\_cookie','ok');

header ('Location: '.$\_SERVER['PHP\_SELF'].

'?step=1');//Перезагружаем с параметром step=1

}

else { //Это повторный вызов?

$res=@$\_COOKIE['my\_cookie']=='ok'?

'Yes':'No';

setcookie ('my\_cookie','',

time()-14\*24\*3600);

echo '<br>Cookie test: '.$res;

}

?>

При проверке этого кода после включения/выключения в браузере поддержки cookies для некоторых обозревателей может понадобиться их перезапуск.

Менее "красивый" альтернативный способ работы с SID - его передача через параметр URL-адреса. PHP имеет возможность автоматически добавлять SID ко всем ссылкам в генерируемых HTML страницах, поэтому вам, как правило, не нужно будет заботиться о том, чтобы добавлять этот идентификатор к каждой ссылке вручную. Если же вы по каким-либо причинам хотите сами передавать идентификатор сессии - вы всегда можете получить его из константы SID или из функции session\_id(). Данный способ неудобен тем, что SID будет отображаться во всех URL-адресах приложения, что может создать, например, проблему с идентификацией URL-адресов поисковыми машинами. Отключить использование номера сессии в URL можно настройкой session.use\_trans\_sid = 0 в файле php.ini. На практике примерно у 99,5 % пользователей cookies включены и поддерживаются, поэтому данный способ можно считать не очень актуальным.

Для того чтобы иметь возможность использовать сессионные переменные в своей программе, необходимо сначала создать сессию:

session\_start();

Нужно вызывать эту функцию на каждой странице, где требуется использовать сессионные переменные.

Для наглядности сессии можно задать имя с помощью функции session\_name(имя\_сессии). Делать это нужно еще до инициализации сессии. Получить имя текущей сессии можно с помощью этой же функции, вызванной без параметров: session\_name();

Регистрация сессионных переменных производится путем вызова следующей функции:

session\_register('var1','var2',...);

Зарегистрировать переменную также можно, просто записав ее значение в ассоциативный массив $\_SESSION:

$\_SESSION['имя'] = 'значение';

В этом массиве хранятся и открыты для программного доступа все зарегистрированные (глобальные) переменные сессии.

Получить идентификатор текущей сессии можно с помощью функции session\_id().

Функция session\_unregister(имя\_переменной) удаляет указанную глобальную переменную из текущей сессии.

Для того чтобы сбросить значения всех переменных сессии, можно использовать функцию session\_unset().

Уничтожить текущую сессию целиком можно функцией session\_destroy(). Она не сбрасывает значения глобальных переменных сессии и не удаляет cookies, но уничтожает все данные, ассоциируемые с текущей сессией.

Следует понимать, что использование механизма сессий не гарантирует полной безопасности системы. Во-первых, данные, передающиеся по открытому протоколу HTTP, могут быть перехвачены, во-вторых, так как переменные сессии глобальны, они сохраняются в зашифрованном виде в cookie-файлах на компьютере пользователя. Более надежным решением представляется хранение ценных данных, таких как лоигны и пароли, только в базе и с шифрованием паролей, а также использование в важных приложениях защищенных версий протоколов, таких как HTTPS.

С учетом сказанного, добавим к возможностям примера 9 несложный механизм регистрации и авторизации пользователей с использованием сессий.

Часть кода, доступная только для авторизованных пользователей, может быть реализована в файле index.php и других модулях системы следующим образом:

$login1 = check\_user();

if (!empty($login1)) {

echo '<p>Вы вошли как '.$login1.

'. <a href="logout.php">Выход</a></p>';

//код для авторизованного пользователя

}

else {

include "logform.php";

}

Функция check\_user, добавленная в модуль functions.php, просто проверяет, выполнялась ли авторизация, связанная с установкой определенной переменной сессии:

function check\_user () {

if (!isset($\_SESSION['my\_inside']))

return '';

else return $\_SESSION['current\_user'];

}

Включаемый модуль logform.php содержит код для вывода формы авторизации, отправки логина и пароля модулю login.php, непосредственно отвечающему за вход в систему, а также ссылки на модуль регистрации register.php:

<form method="post" action="login.php">

<table width=100% align=center border=0>

<tr><td>Логин:</td>

<td><input type=text size=32 maxlength=32

name=login></td></tr>

<tr><td>Пароль:</td>

<td>

<input type=password size=32 maxlength=32

name=password></td></tr>

<tr><td align=center>

<a href="register.php">Регистрация</a></td>

<td align=center>

<input type=submit value="OK"></td></tr>

</table></form>

<?php /\*другойкод \*/ ?>

Модуль login.php сравнивает данные, полученные из формы входа, с логином и зашифрованным паролем, хранящимися в базе. При совпадении данных устанавливаются нужные переменные сессии и происходит перенаправление на главную страницу:

<?php

$params = array('login','password');

require\_once ("functions.php");

function login ($login1, $password1) {

$sql = 'select login, password from '.

'users where login="'.$login1.'" limit 0,1';

$result = dbquery ($sql);

if ($result and dbrows($result)>0) {

$data = dbfetcha ($result);

if ($data['password']==md5($password1)) {

$\_SESSION['my\_inside']=1;

$\_SESSION['current\_user']=$login1;

}

}

}

login ($login,$password);

header('Location: index.php');

?>

Модуль logout.php, также вызываемый по ссылке из index.php, позволяет выйти из системы:

<?php

require\_once ("functions.php");

$current\_user=check\_user();

if (!empty($current\_user)) {

$\_SESSION = array();

session\_destroy ();

}

header('Location: index.php');

?>

Наконец, модуль register.php предоставляет форму для регистрации, проверяет введенный логин на уникальность и добавляет новую запись о пользователе, если регистрация успешна:

<?php

$params = array('login', 'password1',

'password2');

require\_once ("functions.php");

include "head.php";

$error='';

if (!empty($password1) and

$password1!=$password2)

$error.='Пароли не совпадают';

if (!empty($login)) {

$sql='select login from users '.

'where login="'.$login.'" limit 0,1';

$result=dbquery($sql);

if ($result and dbrows($result)) {

$data=dbfetcha($result);

if ($data['login']==$login) {

$error.='Такой логин уже есть';

}

}

}

if (!empty($error)) {

echo '<p>'.$error.

', <a href="register.php">попробуйте '.

'еще раз</a></p>';

include "foot.php";

exit;

}

if (!empty($login) and !empty($password1)) {

$sql='insert into users (login,password)'.

' values ("'.$login.'","'.

md5($password1).'")';

$result=dbquery($sql);

if (!$result) { echo "Bad sql $sql"; }

else {

echo '<p>Регистрация успешна, войдите '.

'в систему с <ahref="index.php">'.

'главной страницы</a></p>';

}

}

else {

?>

<form method="post">

<table width=100% align=center border=0>

<tr><td>Новый логин:</td>

<td><input type=text size=32 maxlength=32

name=login></td></tr>

<tr><td>Новыйпароль:</td>

<td><input type=password size=32 maxlength=32

name=password1></td></tr>

<tr><td>Новыйпарольещераз:</td>

<td><input type=password size=32 maxlength=32

name=password2></td></tr>

<tr><td colspan="2" align=center>

<input type=submit value="OK"></td></tr>

</table></form>

<?php

}

include "foot.php";

?>

Предполагается, что включаемые файлы с именами head.php и foot.php содержат общие для всех файлов проекта дизайнерские "шапку" и "подвал" страницы.

Рекомендуемые по итогам главы разделы стандартной справки: "Справочник языка" - "Предопределенные переменные" - "$\_COOKIE", "Справочник функций" - "Функции для работы с сессиями".

# 5. Объектно-ориентированное программирование на PHP

Начиная с пятой версии, язык PHP в полном объеме поддерживает классы и парадигму объектно-ориентированного программирования. Хотя значительная часть написанного на PHPкода остается процедурно-ориентированной, в будущем следует ожидать все более активного применения разработчиками ООП-подхода.

Во-первых, web-страница естественным образом состоит из объектов, а экземпляры классов как раз и призваны описывать объекты. Во-вторых, написание не просто процедурных модулей, а классов, позволяет удобно хранить настройки скриптов и обращаться к ним извне. Модификатор доступа private может, к тому же, защитить некоторые переменные и функции от изменения извне класса. В-третьих, конструктор класса позволяет при создании объекта установить значения тех свойств, которые нельзя просто инициализировать по умолчанию (значения по умолчанию не могут быть переменными или зависеть от вызова функций). Наконец, классы хорошо решают проблему с пространствами имен, мы уже не рискуем перепутать переменную $i из текущего скрипта со свойством $myclass->i из класса $myclass.

Как правило, продуманный класс на PHP корректно описывает моделируемый объект, содержит достаточно понятных настроек и удобен в обращении. Предполагая знакомство читателя с основными концепциями ООП, кратко опишем особенности их реализации в PHP.

**Инкапсуляция.** Определение класса начинается с ключевого слова class, затем следует имя класса, и далее пара фигурных скобок, которые заключают в себе определение свойств и методов этого класса. Класс может содержать собственные константы, переменные, традиционно называемые *свойствами* и функции (*методы*).

Подобно C++, ссылка на объект, которому принадлежит вызванный метод, доступна через ключевое слово $this, а для создания экземпляра класса используется директива new. Рекомендуется определять классы до создания их экземпляров.

Копия объекта создается с использованием ключевого слова clone (который вызывает метод \_\_clone() объекта, если это возможно). Вызов метода \_\_clone() не может быть осуществлен непосредственно:

$copy\_of\_object = clone $object;

**Наследование.** Класс может наследовать методы и свойства другого класса используя ключевое слово extends при его описании. Невозможно наследовать несколько классов, один класс может наследовать только один базовый класс.

**Полиморфизм.** Наследуемые методы и свойства могут быть переопределены (за исключением случаев, когда метод класса объявлен как final) путем объявления их с теми же именами, как и в родительском классе. Существует возможность доступа к переопределенным методам или статическим методам путем обращения к ним через ключевое слово parent:: (ссылку на родительский класс).

Когда переопределяются методы, количество аргументов и их типы должны оставаться такими же, как и были, иначе PHP выведет ошибку. Это не относится к конструктору, который можно переопределить с другими параметрами.

Свойства, добавленные программистом в класс, определяются с помощью ключевых слов public, protected или private, следуя правилам описания переменных. Это описание может содержать инициализацию, но инициализация должна применяться для константных значений - т.е. переменные должны быть вычислены во время компиляции и не должны зависеть от информации программы во время выполнения для их вычисления:

class A {

private $n1=1;

public $n2=2,$n3=3;

const CONSTANT = 'значение константы';

}

Объявление свойств и методов класса статическими выполняется с помощью ключевого слова static. Это позволяет обращаться к ним без создания экземпляра класса. Атрибут класса, объявленный статическим, не можеть быть доступен посредством экземпляра класса (но статический метод может быть вызван). Как и в C++, статическое свойство существует "в единственном числе" для всех экземпляров класса.

В пределах методов класса доступ к нестатическим свойствам может быть получен с помощью объектного оператора ->: $this->property, где property - имя свойства. Обратите внимание, что в отличие от всех остальных переменных, перед именем свойства property не написан символ "$". Доступ к статическим свойствам может быть получен с помощью оператора :: (двойного двоеточия): class::$property, где class имя класса, и здесь символ "$" перед property необходим!

Для обработки классов и объектов существует ряд стандартных функций, основные из них перечислены в табл. 9.

Более подробную информацию можно получить в справочной системе.

PHP 5 позволяет объявлять методы-конструкторы. Классы, в которых объявлен метод-констуктор, будут вызывать этот метод при каждом создании нового объекта оператором new,   
так что это может оказаться полезным, например, для инициализации какого-либо состояния объекта перед его использованием.

Для создания конструктора достаточно определить в классе функцию с именем \_\_construct. В отличие от C++, в классе может быть объявлен только один конструктор. Конструктор не может возвращать значения, как обыкновенный метод класса, но может принимать параметры.

Таблица 9. Функции для работы с классами и объектами

|  |  |
| --- | --- |
| Имя функции | Описание |
| class\_alias | Создает псевдоним для указанного класса |
| class\_exists | Проверяет, был ли объявлен класс |
| get\_class | Возвращает имя класса, к которому принадлежит объект |
| get\_class\_methods | Возвращает массив имен методов класса |
| get\_class\_vars | Возвращает объявленные по умолчанию свойства класса |
| get\_declared\_classes | Возвращает массив с именами объявленных классов |
| get\_object\_vars | Возвращает свойства указанногообъекта |
| get\_parent\_class | Возвращает имя родительского класса для объекта или класса |
| is\_a | Проверяет, принадлежит ли объект к данному классу или содержит данный класс в числе своих предков |
| is\_subclass\_of | Проверяет, содержит ли объект в своем дереве предков указанный класс |
| method\_exists | Проверяет, существует ли метод в данном классе |
| property\_exists | Проверяет, содержит ли объект или класс указанное свойство |

Конструкторы в классах-родителях не вызываются автоматически, если класс-потомок определяет собственный конструктор. Чтобы вызвать конструктор, объявленный в родительском классе, следует обратиться к методу parent::\_\_construct() внутри конструктора класса-потомка.

PHP 5 предоставляет концепцию деструкторов, сходную с той, что применяется в C++. Деструктор будет вызван при освобождении всех ссылок на определенный объект или при завершении скрипта. Однако порядок выполнения деструкторов не гарантируется и не стоит ожидать немедленного освобождения памяти при явном вызове деструктора.

Чтобы создать деструктор, включите в класс функцию с именем \_\_destruct. Деструктор не может возвращать значения или принимать параметры.

Как и в случае с конструкторами, деструкторы, объявленные в родительском классе, не будут вызваны автоматически. Для вызова деструктора, объявленного в классе-родителе, следует обратиться к методу parent::\_\_destruct() в теле деструктора-потомка. Также класс-потомок может унаследовать деструктор из родительского класса, если он не определен в нем.

Существует еще ряд методов с "предустановленными" именами, например, методы \_\_set и \_\_get позволяют динамически создавать и читать свойства и методы класса, а метод \_\_toString дает возможность описать, как класс должен реагировать на попытку его преобразования в строку. В целях совместимости, рекомендуется не называть свои методы классов именами, начинающимися с символов \_\_ (два подчеркивания).

Все остальные методы класса должны быть определены через модификаторы public (по умолчанию), private, или protected. Объекты одного типа имеют доступ к элементам с модификаторами private и protected друг друга, даже если не являются одним и тем же экземпляром. Это объясняется тем, что реализация видимости элементов известна внутри этих объектов.

**Пример 10.** Реализуем класс A, предназначенный для хранения числового свойства $n и вывода его в виде целого числа. Класс-наследник B переопределяет метод вывода числа show, позволяя печатать и дробную часть числа. При этом класс B пользуется конструктором и деструктором родительского класса, и оба класса поддерживают статическое свойство-счетчик $count, хранящее информацию о том, сколько экземпляров классов A и B создано.

Обратите внимание на комментарии в коде, показывающие, что напечатает функция вывода show, а также на то, что явный вызов деструктора, в отличие от стандартной функции unset, не привел к физическому удалению объекта из памяти.

<?php

class A {

protected $n=0; //свойство

static protected $count=0;

//статический счетчик

function \_\_construct ($n=0) { //конструктор

$this->n = $n; A::$count++;

}

function \_\_destruct() { //деструктор

A::$count--;

}

private function int () { //private функция

return round($this->n);

}

function show() { //public функция

echo '<br>Число '.$this->int($this->n).

', всего '.A::$count."\n";

}

}

class B extends A { //класс-потомок

function \_\_construct ($n=0) { //конструктор

parent::\_\_construct ($n);

}

function \_\_destruct () { //деструктор

parent::\_\_destruct ();

}

function show() { //public функция

echo '<br>Число '.$this->n.

', всего '.A::$count."\n";

}

}

$a=new A(5); $a->show(); //Число 5, всего 1

$b=new B(5.5); $b->show();//Число 5.5, всего 2

$c=new A(10.5); $c->show();//Число 11, всего 3

$b->\_\_destruct();

$b->show();

//Число 5.5, всего 2 - после явного вызова

//деструктора объект не удален,

//хотя счетчик уменьшился!

unset ($b);

if (isset($b)) $b->show();

else echo '<br>Пусто';

//Пусто - объект $b удален!

$a->show(); //Число 5, всего 1

$c->show(); //Число 11, всего 1

//Деструктор для $b

//все равно вызывался дважды!

?>

Такие стандартные для ООП возможности, как абстрактные методы, интерфейсы, финализация, сериализация объектов и обработка исключений, предлагается изучить самостоятельно по рекомендуемым разделам стандартной справки.

Рекомендуемые по итогам главы разделы стандартной справки: "Справочник языка" - "Классы и объекты", "Справочник функций" - "Расширения, относящиеся к переменным и типам" - "Классы/Объекты", "Справочник языка" - "Исключения".

# 6. Установка и модификация системы управления содержимым на PHP/MySQL

Чаще всего разработка больших web-приложений выполняется не "с нуля", а на основе готовой *системы управления содержимым*, также написанной на PHP (англ. Content Management System, CMS, русск. простореч. "движок" сайта). Для типовых сайтов бывает достаточно установить и настроить на сервере CMS, после чего она наполняется информацией с помощью встроенных средств администрирования. При этом, имея доступ к исходным кодам модулей системы, разработчик может модифицировать CMS под свои нужды, если это позволяет лицензия, а также написать для системы дополнительные модули или плагины, если в ней предусмотрено их подключение. Таким образом, хорошее знание PHP способно существенно облегчить труд по разработке большого web-приложения, если мы берем за основу одну из существующих CMS.

На PHP разработано множество готовых CMS-приложений, которые широко используются на различных сайтах, форумах, в сетевых сообществах и блогах. Перечислим только самые известные из них:

* Drupal, Typo3, Joomla, Wordpress - универсальные CMS для личного или корпоративного сайта;
* Magento, osCommerce, OpenCart - системы для Интернет-коммерции;
* Coppermine, Gallery Project, 4images - галереиизображений;
* MediaWiki, DokuWiki - CMS дляподдержкисетевыхэнциклопедий, такихкак Wikipedia;
* phpBB, SMF, vBulletin, Invision Power Board - CMS дляподдержкиинтернет-форумов;
* phpMyAdmin, phpPgAdmin, Adminer - утилитыадминистрированияСУБД;
* eyeOS - удаленная операционная система, основанная на принципе Desktop Operating System.

## Лекция 15. Перспективы развития веб-технологий

**План лекции:**

Развитие веб-технологий: основные тенденции и перспективы, такие как увеличение доли мобильных пользователей, расширение числа веб-сервсов и семантическая паутина (web 3.0).

#### AJAX — подход к построению пользовательских интерфейсов веб-приложений

## Семантическая паутина  Semantic Web

# Развитие веб-технологий: основные тенденции и перспективы

Небольшой (по историческим меркам) срок существования сервиса WWW показал его востребованность все возрастающему числу пользователей. Это стало хорошим стимулом для развития веб-ориентированных концепций и технологий, увеличивающих возможности пользователей. Массовое внедрение и использование этих решений - причина качественных изменений во Всемирной паутине, своего рода смена «версии» Web. На текущий момент аналитики Интернет выделяют три таких «версии» — Web 1.0, Web 2.0 и Web 3.0 (стоит отметить, что деление это условное и часто критикуемое).

## Web 1.0

Понятие Web 1.0 — это общий термин, описывающий состояние Всемирной Паутины за первое десятилетие ее существования. Для 90-х годов XX века были характерны низкая компьютерная грамотность пользователей, медленные типы подключения и ограниченное число сервисов Интернет. Веб-сайтам того времени были присущи следующие основные черты:

* Статичное содержимое веб-страниц, контент создавался и поддерживался разработчиками веб-сайта.
* Фреймовая и/или табличная верстка.
* Низкое качество разметки (зачастую контент был представлен в виде обычного текста, заимствованного из конференций UseNet и подобных источников, и заключенного в тег <pre>).
* Широкое использование нестандартных тегов, поддерживаемых только конкретным браузером (см. «войны браузеров» самостоятельно).
* Использование физических или внедренных стилей, редко - встраиваемых и, тем более, связанных таблиц стилей.
* Указание информации о рекомендованной версии браузера и разрешении монитора, при которых дизайн сайта отображается корректно.
* Гостевые книги, форумы или чаты — как инструменты обратной связи и придания интерактивности.
* Использование графических и текстовых информеров (погода, курс доллара и т.п.) для агрегирования информации.

Условное окончание эпохи «Web 1.0» датируется 2001 годом, когда произошел обвал акций интернет-компаний (об этом говорят так: «лопнул пузырь доткомов» (от англ. .com)). Собственно, существовавшие сайты никуда не делись, но вот вновь создаваемые сайты все больше и больше отличались от типичных «веб-один-нольных».

## Web 2.0

Web 2.0 — совокупность веб-технологий, ориентированная на активное участие пользователей в создании контента веб-сайтов.

Появление названия Web 2.0 принято связывать со статьёй Тима О’Рейли [«What Is Web 2.0»](http://oreilly.com/web2/archive/what-is-web-20.html) от 30 сентября 2005 года

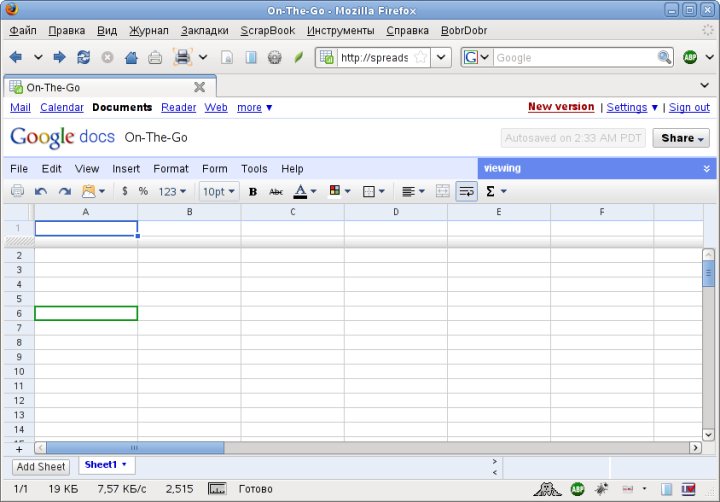
### *Особенности web 2.0:*

* Привлечение «коллективного разума» для наполнения сайта;
* Взаимодействие между сайтами с использованием веб-сервисов;
* Обновление веб-страниц без перезагрузки;
* Агрегирование и синдикация информации;
* Объединение различных сервисов для получения нового функционала;
* Дизайн с применением стилевой разметки и акцентом на юзабилити.

### *Основные элементы web 2.0*

*Веб-сервисы (веб-службы)* — это сетевые приложения, доступные по протоколу HTTP, в качестве протоколов взаимодействия использующие основанные на XML форматы данных (RPC, SOA и подобные). В результате программное обеспечение может использовать веб-службы вместо того, чтобы самостоятельно реализовывать требуемый функционал (например, проверить введенный в форме почтовый адрес). В отличие от обычных динамических библиотек, такой подход обладает рядом плюсов:

* Веб-служба находится на серверах компании, которая её создала ([рис. 1](http://www.4stud.info/web-programming/lecture9.html#web-service)). Поэтому в любой момент пользователю доступна самая свежая версия программы и ему не приходится заботиться об обновлениях и вычислительных мощностях, требуемых для выполнения операции.
* Инструменты для работы с HTTP и XML есть в любом современном языке программирования, поэтому веб-службы являются платформонезависимыми.

**

*Рис. 1. Google Docs как пример веб-сервиса.*

#### AJAX

Asynchronous JavaScript and XML — подход к построению пользовательских интерфейсов веб-приложений, при котором веб-страница, не перезагружаясь, асинхронно загружает нужные пользователю данные. Использование Ajax стало наиболее популярно после того, как Google начала активно использовать его при создании своих сайтов, таких как Gmail и Google Maps. Часто Ajax считают синонимом Веб 2.0, что совершенно не так. Веб 2.0 не привязан к какой-то одной технологии или набору технологий, с тем же успехом ещё в 1999 году возможность асинхронного обновления страницы уже предоставлял Flash 4.

#### Веб-синдикация

Одновременное распространение информации в том числе аудио- и видео- на различные страницы или web-сайты, как правило, с использованием технологий RSS или Atom. Принцип заключается в распространении заголовков материалов и ссылки на них (например, последние сообщения форумов, и т. п.). Первоначально эта технология использовалась на новостных ресурсах и в блогах, но постепенно сфера применения расширилась.

#### Mash-up

Веб mash-up (дословный перевод — «смешение») — сервис, который полностью или частично использует в качестве источников информации другие сервисы, предоставляя пользователю новую функциональность для работы. В результате такой сервис может становиться также новым источником информации для других веб mash-up сервисов. Таким образом образуется сеть зависимых друг от друга сервисов, интегрированных друг с другом.

Например, сайт транспортной фирмы может использовать карты сервиса Google Maps для отслеживания местонахождения перевозимого груза.

#### Метки (теги)

Ключевые слова, описывающие рассматриваемый объект, либо относящие его к какой-либо категории. Это своего рода метки, которые присваиваются объекту, чтобы определить его место среди других объектов. С понятием меток тесно связано понятие фолксономии — термина, о котором широко заговорили именно в связи с ростом сервисов Веб 2.0, таких как Flickr, del.icio.us, и, в дальнейшем, Wink.

Появление и быстрое распространение блогов тоже вписывается в концепцию Веб 2.0, создавая так называемую «редактируемую Паутину» (writable web).

Возможность пометить документ ключевыми словами существует и в языке HTML (англ. keywords), однако этот способ был полностью скомпрометирован широким его использованием в целях поискового спама.

#### Социализация

Использование разработок, которые позволяют создавать сообщества пользователей.

* В понятие социализация сайта можно также включить возможность индивидуальных настроек сайта и создание личной зоны (личные файлы, изображения, видео, блоги) для пользователя, чтобы пользователь чувствовал свою уникальность.
* Поощрение, поддержка и доверие «коллективному разуму».
* При формировании сообщества большое значение имеет соревновательный элемент, Репутация или Карма, которые позволяют сообществу саморегулироваться и ставить пользователям дополнительные цели присутствия на сайте.

#### Дизайн

Понятие Веб 2.0 также отразилось и в дизайне. Предпочтительными стали округлость, имитация выпуклых поверхностей, имитация отражений на манер глянцевого пластика современных hi-end устройств (к примеру, плееры). В целом, восприятие внешнего вида на глаз кажется более приятным. Графика таких сайтов занимает больший объём, нежели при использовании аскетичного дизайна. Отчасти эта тенденция связана с совпавшим по времени выходом новых версий операционных систем использующих вышеупомянутые идеи.

Однако однообразие таких сайтов явно и в последнее время считается, графический облик классического дизайна веб 2.0, устаревшим и не креативным. Особенно это отражается в современной тенденции создания информативных сайтов, где главную роль играет простота, изящество, графичность и юзабилити. В дизайне не должно быть ограничений, но веб 2.0 их прививает.

### Недостатки Веб 2.0

При использовании технологий web 2.0 вы становитесь арендатором сервиса и/или дискового пространства у какой-то сторонней компании. Возникающая при этом зависимость формирует ряд недостатков новых сервисов:

* зависимость сайтов от решений сторонних компаний, зависимость качества работы сервиса от качества работы многих других компаний;
* слабая приспособленность нынешней инфраструктуры к выполнению сложных вычислительных задач в браузере;
* уязвимость конфиденциальных данных, хранимых на сторонних серверах, для злоумышленников (известны случаи хищения личных данных пользователей, массовых взломов учётных записей блогов).

Фактически сайт эпохи Веб 2.0 на первый взгляд интерактивен и дружелюбен, позволяет себя легко настраивать. Однако сбор статистики о пользователях, их предпочтениях и интересах, личной жизни, карьере, круге друзей могут помочь владельцу сайта манипулировать сообществом. По самым пессимистичным прогнозам многочисленные сайты Веб 2.0 вкупе с другими современными технологиями дают прообраз тоталитарной системы «Большого брата».

## Web 3.0

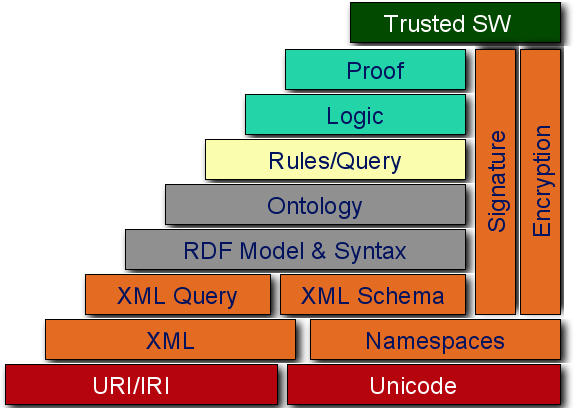
Web 3.0 — это принципиально новый подход к обработке информации, представленной во Всемирной паутине. Web 3.0 в первую очередь подразумевает под собой иной подход к обработке информации сообществом пользователей. Если Web 1.0 предполагает веб-мастера в качестве поставщика контента, а Web 2.0 — сообщество равноправных пользователей, генерирующих контент в рамках тематического проекта, то Web 3.0 уже позволяет этим самым равноправным пользователям "выбирать" экспертов в заданной области (или в нескольких областях) и "наделять его властью". Такие общепризнанные эксперты-выдвиженцы постепенно расширяют свое влияние на сообщество, могут выступать его модераторами, управлять сообществом при помощи дополнительных прав и расширенных возможностей в рамках интернет-проекта. Это исключает возможность управления сообществом некомпетентных и малозначимых его участников, что является очень важным вещью — достаточно вспомнить «вебдванольное» равноправие на проекте Wikipedia, из-за которого ее создатель должен был более десятка раз исправлять свою собственную (!) биографию – пользователи считали, что в ней есть неточности и честно ее исправляли.

Также термином Web 3.0 часто называют концепцию семантической паутины (Semantic Web).

## Semantic Web

*Семантическая паутина* (англ. Semantic Web) — часть глобальной концепции развития сети Интернет, целью которой является реализация возможности машинной обработки информации, доступной во Всемирной паутине. Основной акцент концепции делается на работе с метаданными, однозначно характеризующими свойства и содержание ресурсов Всемирной паутины, вместо используемого в настоящее время текстового анализа документов. Термин впервые введён Тимом Бернерсом-Ли в мае 2001 года в журнале «Scientific American», и называется им «следующим шагом в развитии Всемирной паутины». В семантической паутине предполагается повсеместное использование, во-первых, унифицированных идентификаторов ресурсов (URI), а во-вторых — онтологий и языков описания метаданных.

Эта концепция была принята и продвигается Консорциумом W3 ([Рис. 2](http://www.4stud.info/web-programming/lecture9.html#semantic-web)). Для её внедрения предполагается создание сети документов, содержащих метаданные о ресурсах Всемирной паутины и существующей параллельно с ними. Тогда как сами ресурсы предназначены для восприятия человеком, метаданные используются машинами (поисковыми роботами и другими интеллектуальными агентами) для проведения однозначных логических заключений о свойствах этих ресурсов.

*[](http://www.4stud.info/web-programming/img/semantic_web_stack.jpg)*

*Рис. 2. Стек протоколов Semantic Web*

### *Основная идея*

Семантическая паутина — это надстройка над существующей Всемирной паутиной, которая призвана сделать размещённую в ней информацию более понятной для компьютеров. Машинная обработка возможна в семантической паутине благодаря двум её важнейшим характеристикам:

* Повсеместное использование унифицированных идентификаторов ресурсов (URI). Традиционная схема использования таких идентификаторов в современном Интернете сводится к установке ссылок, ведущих на объект, им адресуемый. Очевидным свойством такой ссылки является возможность «загрузки» объекта, на который она указывает. Таким объектом может быть веб-страница, файл произвольного содержания, фрагмент веб-страницы, а также неявное указание на обращение к реально существующему физическому ресурсу по протоколу, отличному от HTTP (например, ссылки mailto:). Концепция семантической паутины расширяет это понятие, включая в него ресурсы, недоступные для скачивания. Адресуемыми с помощью URI ресурсами могут быть, например, отдельные люди, города и другие географические сущности, художественные артефакты и т. д. К идентификатору предъявляются несколько простых требований: он должен быть уникальной строкой определённого формата, адресующей реально существующий объект.
* Повсеместное использование онтологий и языков описания метаданных. Современные методы автоматической обработки данных, доступных в Интернете, как правило, основаны на частотном и лексическом анализе текстового содержимого, которое прежде всего предназначено для восприятия человеком. В семантической паутине предлагается использовать форматы описания, доступные для машинной обработки (например, семейство форматов, часто упоминаемое в литературе как «Semantic Web family»: RDF, RDF Schema или RDF-S, и OWL), в свою очередь, использующие URI для адресации описываемых и описывающих объектов, а также онтологии и дескрипционные логики в качестве базовых математических формализмов.

### *Критика*

Несмотря на все преимущества, предоставляемые семантической паутиной в случае её внедрения, существуют сомнения в возможности её полной реализации.

#### Практическая нереализуемость. Разные комментаторы высказывают различные причины, которые могут быть препятствием к этому, начиная с человеческого фактора (люди склонны избегать работы по поддержке документов с метаданными, открытыми остаются проблемы истинности метаданных, и т. д.), и заканчивая сложностью определения онтологии верхнего уровня (корня иерархии), критической для семантической паутины.

#### Дублирование информации.Необходимость описания метаданных так или иначе приводит к дублированию информации. Каждый документ должен быть создан в двух экземплярах: размеченным для чтения людьми, а также в машинно-ориентированном формате.

#### Невозможность получения коммерческой выгоды.Известно, что основное финансирование современных интернет-ресурсов (за исключением строго некоммерческих проектов) обеспечивают рекламодатели. Главный критерий, от которого зависит стоимость рекламного места — посещаемость сайта. Однако в случае реализации семантических поисковых систем, которые будут сами отбирать и сразу предоставлять нужную пользователю информацию, отпадает необходимость посещать сайт — источник материала, а значит пользователь не увидит рекламу, и как следствие прекратится финансирование интернет-проектов.

### *Реализация*

#### Языки описания.

Техническую часть семантической паутины составляет семейство стандартов на языки описания, включающее XML, XML Schema, RDF, RDF Schema, OWL, а также некоторые другие. Располагая их в порядке повышения уровня абстракции, реализуемого тем или иным языком, получаем:

* XML предоставляет синтаксис для определения структуры документа, подлежащего машинной обработке. Синтаксис XML не несёт семантической нагрузки.
* XML Schema определяет ограничения на структуру XML-документа. Стандартный синтаксический анализатор языка XML в состоянии проверить произвольный XML-документ на соответствие его структуры так называемой схеме документа, описанной в XML Schema.
* RDF представляет собой простой способ описания экземплярных данных в формате субъект-отношение-объект, в котором в качестве любого элемента этой тройки используются только идентификаторы ресурсов. Существует стандартизованное отображение этих троек на XML-документы предопределённой структуры (т. е. консорциумом W3 определена схема XML-документов, содержащих RDF-описания), а также на другие форматы представления (например, в нотацию N3).
* RDF Schema описывает набор атрибутов (здесь их точнее назвать отношениями), таких, как rdfs:Class, для определения новых типов RDF-данных. Языком поддерживается также отношение наследования типов rdfs:subClassOf.
* OWL расширяет возможности по описанию новых типов (в частности, добавлением перечислений), а также позволяет описывать новые типы данных RDF Schema в терминах уже существующих (например, определять тип, являющийся пересечением или объединением двух существующих).

#### Логический вывод

Форматы описания метаданных в семантической паутине предполагают проведение логического вывода на этих метаданных, и разрабатывались с оглядкой на существующие математические формализмы в этой области. Математическое обоснование тех или иных конструкций языка описания необходимо для проведения заключений о свойствах программ, обрабатывающих данные в этом формате.

Особенно сильно это относится к языку OWL. Базовым формализмом для него являются дескрипционные логики, а сам язык разбит на три вложенных подмножества (в порядке вложенности): OWL Lite, OWL DL и OWL Full. Доказано, что логический вывод на метаданных с выразительностью OWL Lite выполняется за полиномиальное время (другими словами, задача вывода принадлежит к классу P). OWL DL описывает максимальное разрешимое в данный момент подмножество дескрипционных логик, но некоторые запросы по таким данным могут требовать экспоненциального времени выполнения. OWL Full реализует все существующие конструкторы дескрипционных логик, но не каждый запрос в этом подмножестве языка может быть разрешён (слово «разрешён» здесь употребляется в значении, основанном на корне «решать»).

Простая структура предикатов языка RDF, в свою очередь, позволяет использовать при его обработке опыт из теорий логических баз данных, логики предикатов, и т. д.

### Проекты

#### Дублинское ядро

Одним из первых серьёзных и популярных проектов, основанным на принципах семантической паутины, стал проект «Дублинское ядро» (англ. DublinCore), реализуемыйинициативнойорганизациейDublinCoreMetadataInitiative (DCMI). Это открытый проект, цель которого — разработать стандарты метаданных, которые были бы независимы от платформ и подходили бы для широкого спектра задач. Конкретнее, DCMI занимается разработкой словарей метаданных общего назначения, стандартизирующих описания ресурсов в формате RDF.

#### RSS (версий 0.90 и 1.0)

Версии 0.90 и 1.0 формата RSS основаны на RDF. Информация в нём представляется как и в RDF, тройками субъект-отношение-объект. Необходимо отметить, что несмотря на то, что ему присущи многие недостатки семантической паутины (дублирование информации, например), этот простейший формат быстро стал чрезвычайно популярным за счёт узкой категоризации подмножества используемых метаданных. Отличие RSS от RDF состоит в том, что субъектом тройки всегда является сайт-источник RSS-файла, а в качестве отношений используются самые очевидные свойства документов, имеющие отношение к часто обновляющимся источникам информации: дата написания, автор, постоянная ссылка, и т. д. Другими словами, RSS — узкоспециализированное подмножество RDF.

Помимо недостатков, RSS унаследовал и все достоинства форматов из семейства семантической паутины: гибкость RSS позволяет использовать его не только для проверки на наличие новой информации на регулярно обновляющихся сайтах, но и для подкастов, и торренттрекеров.

Заметим, что формат RSS версии 2.0, хотя и не является форматом, основанным на RDF, позволяет внедрение произвольного XML-содержимого, находящегося в собственных пространствах имён XML. Это позволяет использовать RDF-описания также и в нём (используя пространство имён rdf).

#### FOAF

Проект «Friend of a Friend» («Друг друга») позволяет описывать отношение знакомства с помощью RDF. Любой его участник может идентифицировать себя уникальным образом с помощью URI (например, mailto-адресом электронной почты, адресом блога, и т. п.), создать свой профиль, используя предопределённые для FOAF отношения на языке RDF, и перечислить идентификаторы людей, которых этот участник знает. Это описание может обрабатываться автоматически; на его основе можно строить сети доверия, анализировать структуру социальных групп, и т. д.

### *Семантические веб-сервисы*

В то время как совокупность ресурсов и их метаданных можно считать статической частью семантической паутины, её динамическую часть представляют т. н. семантические веб-сервисы — законченные элементы программной логики с однозначно описанной семантикой, доступные через Интернет и пригодные для поиска, композиции и выполнения.

Технически, семантический веб-сервис отличается от обычного веб-сервиса наличием не только описания интерфейса (обычно на языке WSDL) в терминах типов данных, передаваемых сервису, возвращаемых значений и генерируемых ошибок, но и семантического описания всех его характеристик. Заметим, что дублирования данных, упомянутого в числе недостатков семантической паутины, здесь не происходит: WSDL-описания изначально были предназначены для машинной обработки.

Потенциальная выгода от использования семантических веб-сервисов заключается в возможности автоматического поиска (а также композиции) программными агентами подходящих сервисов для решения поставленных задач. Тем не менее, сложность этой задачи в её общей формулировке пока позволяет добиваться некоторых положительных результатов только в узкоспециализированных отраслях, явным образом выигрывающих от внедрения сервисно-ориентированной архитектуры, например в интеграции корпоративных приложений.

## Вместо эпилога

Для более полного представления о том, какие перспективы несет семантический веб выделим ряд факторов сравнения и выведем сводную таблицу, характеризующую прошлое, настоящее и будущее Всемирной паутины ([Табл.1](http://www.4stud.info/web-programming/lecture9.html#web-differences)).

Таблица 1. Сравнение технологий web 1.0, web 2.0 и web 3.0

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Web 1.0** | **Web 2.0** | **Web 3.0** |
| **Обобщенное описание** | Интерактивная паутина | Программируемая паутина | Паутина связанных данных |
| **Единица представления** | Веб-страница | Веб-приложение | Пространство данных |
| **Значимая единица обмена данными** | Адрес страницы (URL) | Адрес веб-приложения (URL) | URI ресурса, сущности, объекта |
| **Зернистость данных** | Низкая (HTML-страница) | Средняя (XML-тег) | Высокая (объекты RDF) |
| **Представляемые услуги** | Поиск (возможность ИСКАТЬ ИНФОРМАЦИЮ, результаты поиска не точны) | Сообщества (блоги в социальных сетях) | Поиск (способ НАХОДИТЬ ИНФОРМАЦИЮ, результаты поиска точны ) |
| **Фактор участия пользователя** | низкий | средний | высокий |
| **Фактор удовлетворенности пользователя от использования сайта** | низкий | средний | высокий |
| **Фактор ссылаемости на данные** (обращения по ссылкам) | низкий (документы) | средний (документы) | высокий (документы и их отдельные части) |
| **Фактор субъективности** | высокий | средний (возможность выбора партнеров (friend lists) или установления ограничений на доступ к данным в блогах) | низкий (каждый может обратиться к ресурсу через URI) |
| **Уровень трансклюзивности содержимого** | низкий | средний ("смешивание" данных, управляемое кодом приложения) | высокий ("смешивание" данных, управляемое данными) |
| **Уровень соответствия видимого предпочитаемому** (What You See Is What You Prefer (WYSIWYP)) | низкий | средний | высокий (настраиваемое описание представления ресурсов) |
| **Доступность данных** (открытый доступ к данным) | низкая | средняя (доступ через бункеры данных — серверные приложения) | высокая (прямой доступ) |
| **Средства идентификации пользователей** | слабые | средние (OpenID) | сильные (FOAF+SSL) |
| **Модель развертывания системы** | Централизованная | Централизованная, с делегированием части полномочий пользователем (регистрация нового пользователя автоматически приводит к созданию среды для него ) | Распределенная, с выделенными централизованными функциями |
| **Модель данных** | Логическая (иерархическая, на основе DOM) | Логическая (иерархическая, на основе XML) | Концептуальная (графы RDF) |
| **Пользовательский интерфейс** | Динамически генерируемый (server-side) статичный интерфейс (client-side) | Динамически генерируемый (server-side), с возможностью частичного изменения на стороне клиента (XSLT, XQuery/XPath) | Полностью динамический интерфейс, представляемый возможностью самоописания RDF |
| **Возможности запросов данных** | Полнотекстовый поиск | Полнотекстовый поиск | Полнотекстовый поиск + поиск в графовых структурах с помощью SPARQL (Structured Graph Pattern Query Language) |
| **Web как средство массовой информации** | Представляет мнение автора/издателя | Отражает мнение социальной группы, состоящей из равноправных авторов и комментаторов | Представляет мнение социальной группы, подкрепленное экспертными оценками |