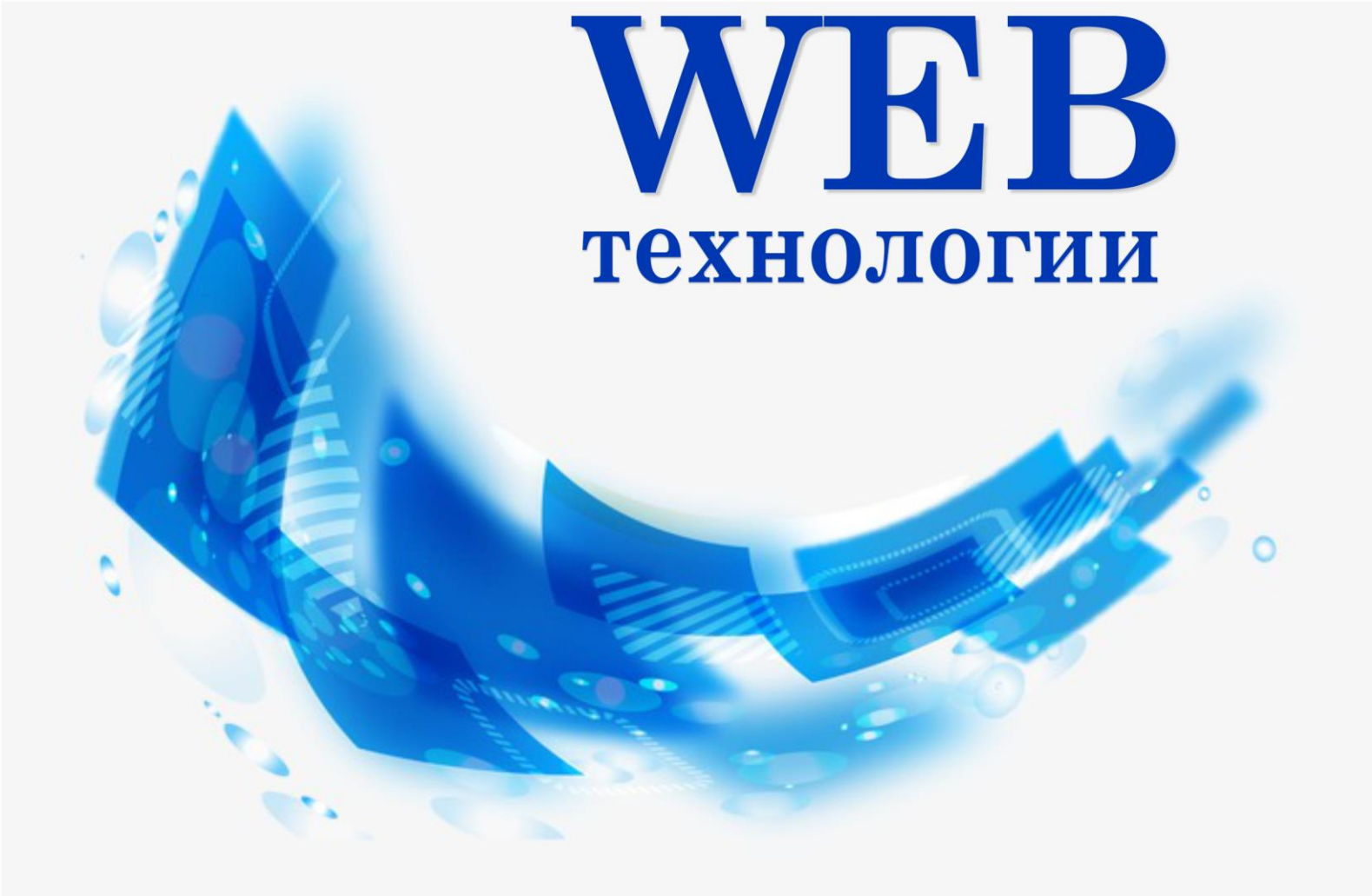
Ниязмаметова С.А.

УЧЕБНОЕ ПОСОБИЕ

Для студентов специальностей: 220206 - «Автоматизированные системы обработки информации и управления»;
230109 - «Программное обеспечение средств вычислительной техники»

WEB технологии



Бишкек -2018 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

КЫРГЫЗСКОЙ РЕСПУБЛИКИ

Кыргызский государственный университет им. И. Арабаева

Колледж КГУ им. И. Арабаева

Секция «Информатика»

Учебное пособие

Современные web-технологии

Для студентов специальностей:

220206 - «Автоматизированные системы обработки информации и управления»

230109 - «Программное обеспечение средств вычислительной техники»

Бишкек -2018 г.

УДК 004.738.52(072)

Современные web-технологии: учебное пособие для студентов специальностей: 220206 – «Автоматизированные системы обработки информации и управления», 230109 – «Программное обеспечение средств вычислительной техники».

Составитель: Ниязмаметова С.А.

Рекомендовано:

- на заседании Ученого Совета КГУ им. И. Арабаева, протокол № 73 от 02.07.2018

Рецензенты:

- к.т.н., доцент кафедры «ИСТ» КГУСТА им. Н. Исанова Т.Т. Каримбаев
- зав.ИО КГУ имени И.Арабаева Ж.К.Мокешов

Содержание

Пояснительная записка

Учебное пособие составлено для студентов, обучающихся по специальностям: 220206 – «Автоматизированные системы обработки информации и управления», 230109 «Программное обеспечение средств вычислительной техники».

Данный курс дает систематическое изложение основных понятий современных web-технологий.

Предметом данного курса являются технологии глобальной сети World Wide Web (сокращенно WWW или просто Web). Или по другому Web-технологии.

В пособии будут рассмотрены:

1. Структура и принципы Web (базовые понятия, архитектура, стандарты и протоколы);
2. Технологии сети Web (языки разметки и программирования web-страниц, инструменты разработки и управления web-контента и приложений для Web, средства интеграции web-контента и приложений в Web).

Сеть Web представляет собой глобальное информационное пространство, основанное на физической инфраструктуре Интернета и протоколе передачи данных HTTP. Зачастую, говоря об Интернете, подразумевают именно сеть Web.

1. Введение в web-технологии: структура и принципы Web.

Развитие Интернет

Интернет — всемирная система объединенных компьютерных сетей, построенная на базе IP и маршрутизации IP-пакетов. Интернет образует глобальное информационное пространство, служит физической основой для Всемирной паутины (World Wide Web, WWW) и множества других систем передачи данных.

Интернет можно рассматривать как «сеть сетей», каждая из которых управляется независимым оператором – поставщиком услуг Интернета (ISP, Internet Service Provider – интернет провайдер).

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 нескольких 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 OC-3 (155 Мбит/с)
1998	Число зарегистрированных доменных имен превысило 2 млн.
2000	Количество индексируемых web-страниц превысило 1 млрд.

Интернет сегодня не принадлежит какому-либо определенному государству.

Но тем не менее, в Интернете проявляются определенные формы централизации в 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.
2. 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 web-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;

Стандартизация в интернет

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).

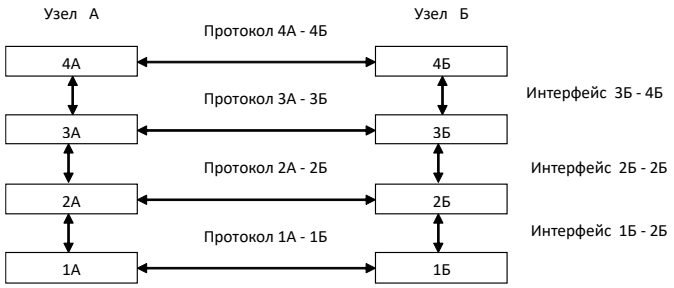


Рисунок 2. *Организация взаимодействия между уровнями иерархии при иерархического декомпозиции в сети Интернет*.

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. Request for Comments) — 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 Web, 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 web-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.

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 способами от 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.

DNS – система доменных имен

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 такой способ оказался неэффективным. Поэтому для установления соответствия между доменным именем и IP-адресом используется специальная система доменных имен (DNS, Domain Name System), которая основана на создаваемых 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), *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.

С 1997 года компания **«АзияИнфо»** является является официальным регистратором доменов в зоне KG. Данное право было делегировано компании Международной Корпорацией по Регистрации Имен и Номеров в Интернет (ICANN).

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.

World Wide Web (WWW)

Сеть WWW образуют миллионы *web-серверов*, расположенных по всему миру. *Web-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 web-сервер.

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

Понятие прокси-сервер

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 *кэша* (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;
- кэширование 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 web-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 между серверам; 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.

2. **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.

3. **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.

4. **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.

2. Клиент-серверные технологии Web. Протокол HTTP.

Базовым протоколом сети гипертекстовых ресурсов Web является протокол HTTP. В его основу положено взаимодействие «*клиент-сервер*», то есть предполагается, что:

1. Потребитель-*клиент* инициировав соединение с поставщиком-сервером посылает ему запрос;
2. Поставщик-*сервер*, получив запрос, производит необходимые действия и возвращает обратно клиенту ответ с результатом.

При этом возможны два способа организации работы компьютера-клиента:

- *Тонкий клиент* - это компьютер-клиент, который переносит все задачи *по обработке информации* на сервер. Примером тонкого клиента может служить компьютер с браузером, использующийся для работы с web-приложениями.
- *Толстый клиент*, напротив, производит обработку информации независимо от сервера, использует последний в основном лишь для *хранения данных*.

Прежде чем перейти к конкретным клиент-серверным web-технологиям, рассмотрим основные принципы и структуру базового протокола HTTP.

Протокол HTTP

HTTP (HyperText Transfer Protocol - 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 web-приложение, посылающее запросы, может отслеживать задержки ответов, а web-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: *Internet Explorer*, *Opera*, *Mozilla Firefox*, *Netscape Navigator* и другие. Наиболее популярными реализациями web-серверов являются: *Internet Information Services* (*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 стро¬ка-статус, или строка запроса).
- Поля заголовка.
- Пустая строка.
- Тело запроса.

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

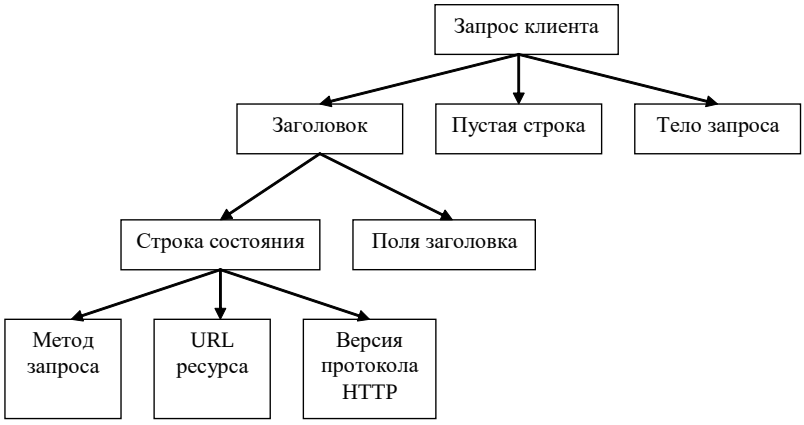


Рисунок 1. *Структура запроса клиента.*

Строка состояния имеет следующий формат:

метод_запроса URL_ресурса версия_протокола_HTTP

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 web-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 Web 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 web-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:

HTTP/1.0 200 OK

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 web-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 web-страницы. В составе ответа могут передаваться изображение, аудио-файл, 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 Microsoft Excel
application/postscript	.ps, .eps	Документ в формате PostScript
application/x-tex	.tex	Документ в формате TeX
application/msword	.doc	Документ в формате Microsoft Word
application/rtf	.rtf	Документ в формате RTF, отображаемый с помощью Microsoft Word
image/gif	.gif	Изображение в формате 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 Web используются уникальные идентификаторы URL.

Единообразный идентификатор ресурса URI (Uniform Resource Identifier) 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 (Resource Description Framework) ресурсы, которые не могут быть получены через Интернет (имена, названия и т.п.). Самые известные примеры URI - это URL и URN.

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

URL имеет следующую структуру:

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

где:

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

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* — служба каталогов 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 web-сервер для обработки HTTPS соединений, администратор должен получить и установить в систему сертификат для этого web-сервера.

SSL (Secure Sockets Layer) - криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет. При его использовании создаётся защищённое соединение между клиентом и сервером. SSL изначально разработан компанией Netscape Communications. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший название TLS. Этот протокол использует шифрование с открытым ключом для подтверждения подлинности передатчика и получателя. Поддерживает 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 web-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 Web 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 Web, вспомните, как вел себя клиент после того, как вы создали для себя почтовый ящик на сервере. Когда вы в следующий раз обратились с того же компьютера к *mail.ru*, вы, вероятно, заметили, что после загрузки web-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 ***A***, ***B*** и ***C***. Предположим также, что сервер ***B***, в отличие от ***A*** и ***C***, передает клиенту команду записать *cookie*. Последовательность запросов клиента серверу и ответов на них будет выглядеть приблизительно следующим образом.

1. Передача запроса серверу ***A***.
2. Получение ответа от сервера ***A***.
3. Передача запроса серверу ***B***.
4. Получение ответа от сервера ***B***. В состав ответа входит поле заголовка *SetCookie*. Получив его, клиент записывает *cookie* на диск.
5. Передача запроса серверу ***C***. 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 ***C***.
7. Передача запроса серверу ***A***. В этом случае клиент также никак не реагирует на тот факт, что на диске хранится *cookie*.
8. Получение ответа от сервера ***A***.

9. Передача запроса серверу ***B***. Перед тем как сформировать запрос, клиент определяет, что на диске хранится запись *cookie*, созданная после получения ответа от сервера ***B***. 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*; ...**

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

3. Сценарии и приложения выполняющиеся на стороне клиента

Как правило, Web-приложение - приложение, в котором *клиентом* выступает *браузер*, а *сервером* - *web-сервер*.

Рассмотрим типы программ, обеспечивающих работу Web и использующих HTTP-протокол.

Никакой HTTP-обмен невозможен без клиента и сервера. Однако помимо клиента и сервера в web-сеансе могут участвовать и другие программы, которые и являются объектом web-программирования.

Результатом работы web-приложения является web-страница, отображаемая в окне браузера. При этом само web-приложение может выполняться как на компьютере клиента, так и на компьютере сервера.

Рассмотрим подробнее обе схемы.

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 web-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.

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 web-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.

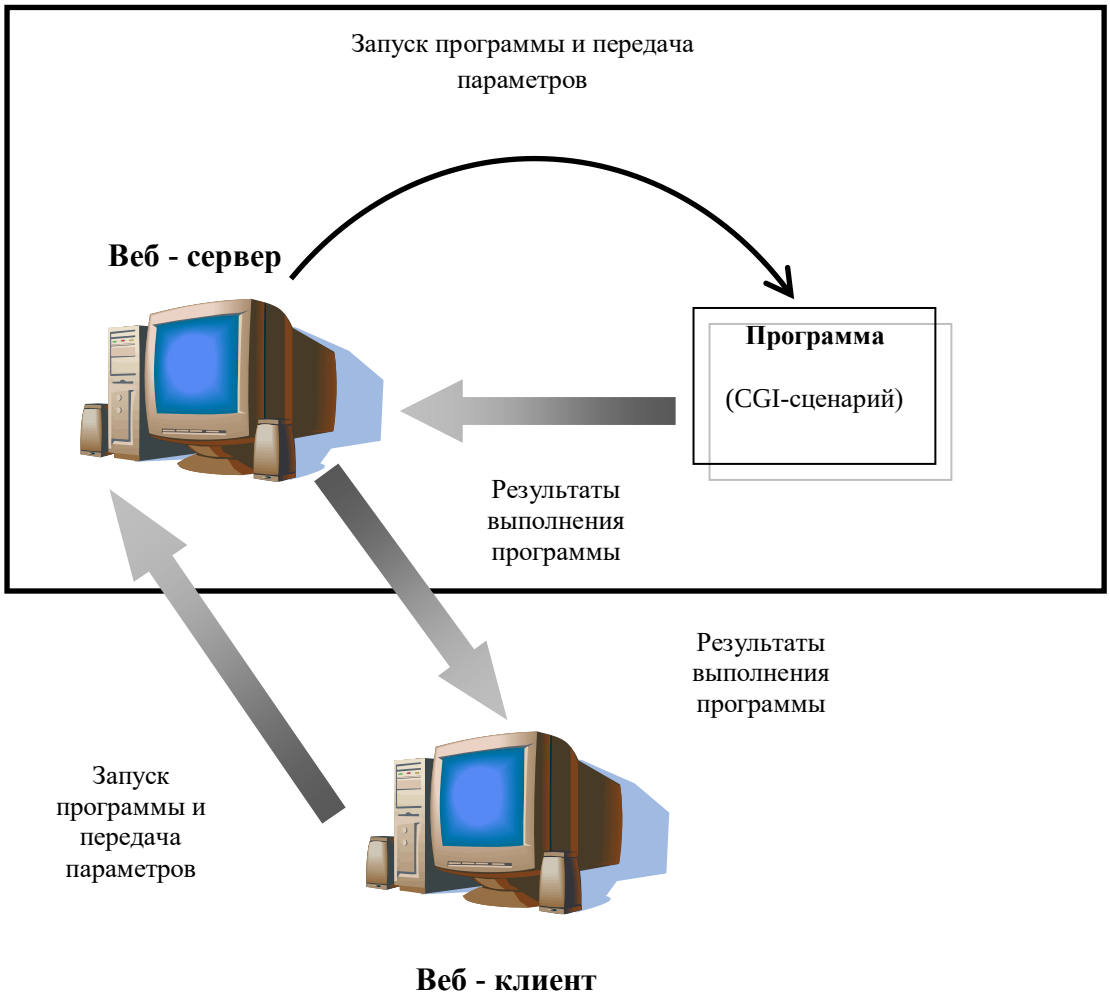


Рисунок 1. *Взаимодействие клиента с программой, выполняющейся на сервере.*

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 web-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.

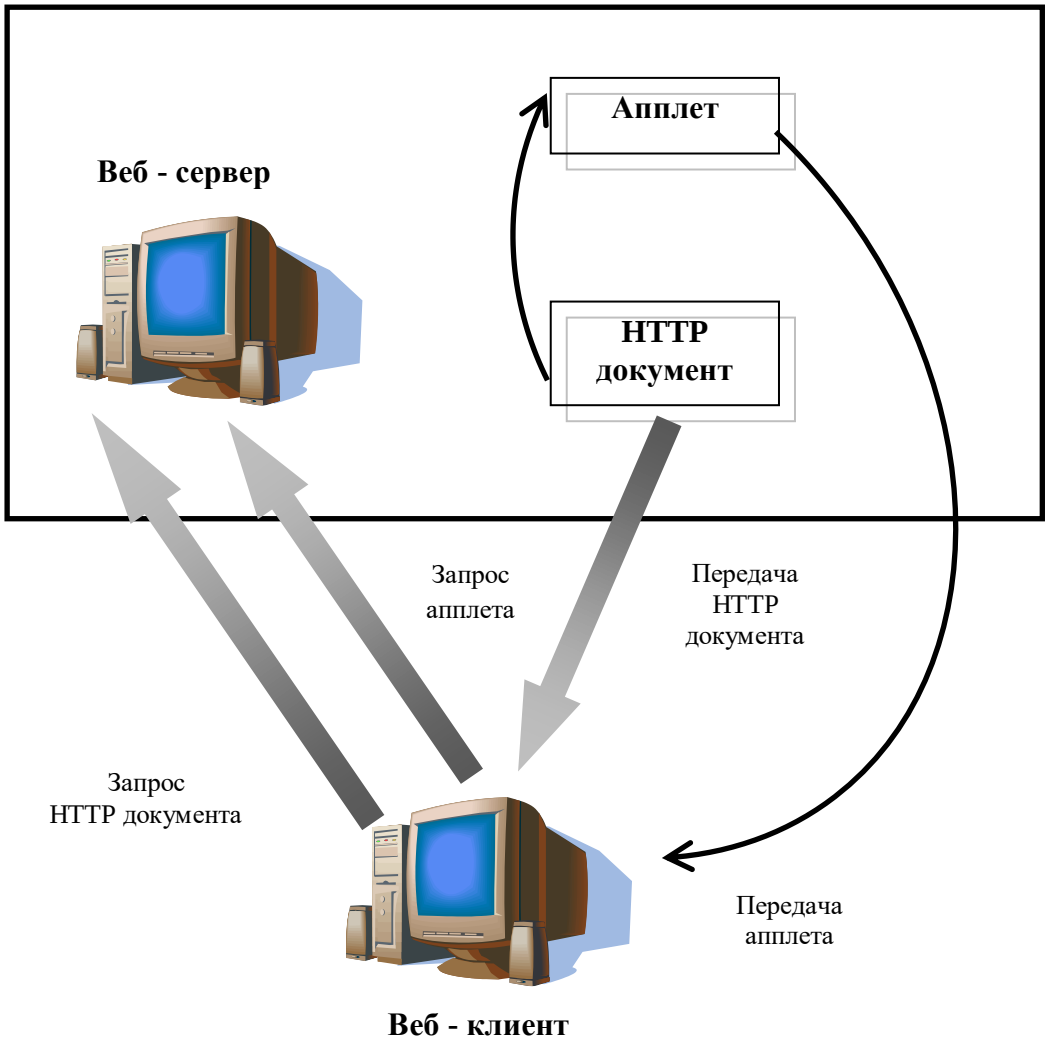


Рисунок 2. *Передача клиенту Java-апплета.*

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. выполяется на клиентской стороне);
- Требуется много времени на загрузку;

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

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 web-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 web-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. *Многострочный комментарий* 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 *условные выражения* ***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.

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.

Java-апплеты

Java-апплет - это программа, написанная на языке Java и откомпилированная в байт-код. Выполнется в браузере с использованием виртуальной Java-машины (*JVM*). Апплеты используются для предоставления интерактивных возможностей web-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 web-сервера, и браузер

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

Апплет может быть внедрен в web-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 web-страницу;
- после первого запуска апплета, когда Java-машина уже выполняется и быстро запускается, выполнение апплетов происходит существенно быстрее;

– загружаются со скоростью сопоставимой с программами 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*).

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 web-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, web-сайты и многое другое.

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 web-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 (C#, 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*.

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.

DHTML.

Динамический HTML или *DHTML* представляет собой набор технологий, которые совместно позволяют создавать интерактивные web-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

web-страницы индивидуально для каждого пользователя. Это достигается созданием страниц с помощью клиентских или серверных (например, на *PHP* или *Perl*) сценариев.

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).

4. Серверные web-приложения.

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

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

Сценарий (*скрипт*, *script*) - программа, автоматизирующая некоторую задачу, которую пользователь обычно выполняет вручную, используя интерфейсы программы.

Стандарт CGI

Круг задач, решаемых Web-сервером, ограничен. В основном он сводится к поддержке HTTP-взаимодействия и доставке клиенту Web-документов. Любые "нестандартные" действия реализуются с помощью специальной программы, которая взаимодействует с web-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 *web-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 *переменных окружения*;
- 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 mype* = "*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.pl.

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

Как web-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 web-сервера один из каталогов специально выделяется для хранения сценариев. Обычно такой каталог получает имя *cgi-bin* (или *Scripts* для web-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.

HTTP/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 web-страницы, т.е. возвращаемые данные следует дополнить дескрипторами HTML. Таким образом, ответ CGI-сценария клиенту обычно выглядит так:

```
Content-type: text/html

<html>
<head>
<title>ответ сценария</title>
</head>
<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 web-сервером на выполнение по запросу клиента.

Для вызова данного сценария достаточно включить в web-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 web-странице - передача информации CGI-сценарию осуществляется в два этапа: сначала браузер передает данные web-серверу, затем web-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, web-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 web-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 web-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.

В рассмотрим кратко наиболее известные языки разработки сценариев для web-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*). Существует 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 кросс-платформенные 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 (web, 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 *сетевыми протоколами* 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 *Cи*, 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.

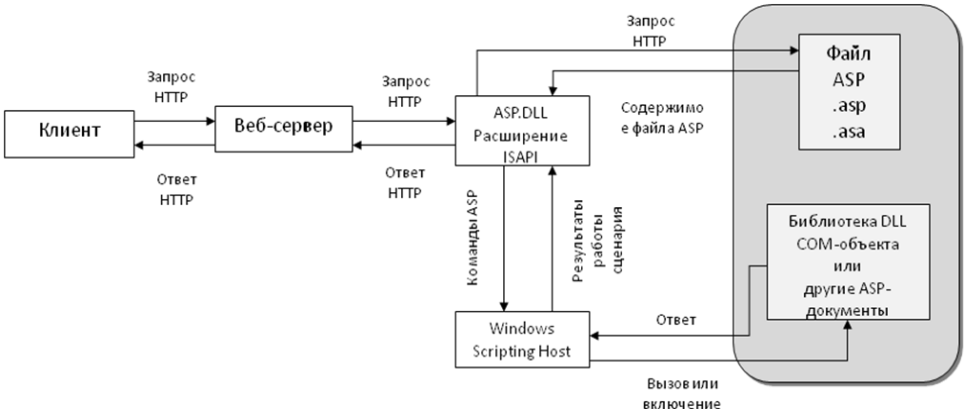
ASP

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

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

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

Архитектура ASP представлена ниже.



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 *Windows Script Host* (*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 web-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 web-сайта.
- *Request*. Хранит содержимое *HTTP*-запроса и обеспечивает вспомогательные функции для обработки данных *HTTP*-запроса.
- *Server*. Содержит информацию о web-сервере, web-сайте, а также обеспечивает поддержку вызывающей программы.
- *Session*. Представляет собой состояние заданного web-сеанса с заданным хостом клиентом.

ISAPI

Для web-сервера IIS (Internet Information Server) Был разработан специальный программный интерфейс для создания приложений расширяющих стандартные возможности web-сервера.

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

ISAPI также реализован в виде модуля *mod_isapi* для web-сервера *Apache*. Таким образом, серверные приложения, разработанные для MS IIS могут также выполняться в *Apache* и других web-серверах.

В противоположность CGI - *ISAPI-приложение* загружается в том же адресном пространстве, что и web-сервер IIS. Это позволяет повысить производительность приложений благодаря сокращению издержек на запуск отдельных процессов. Однако сбой *ISAPI-приложения* может привести к неустойчивой работе самого web-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 web-сервером.

ISAPI приложения могут разрабатываться с помощью любых языков, поддерживающих экспорт стандартных *C*-функций, например *C, C++, Delphi*

Pascal. Для разработки имеется ограниченное число библиотек для разработки *ISAPI* приложений, например *Intraweb*-компоненты *Delphi Pascal*, специальные *MFC*-классы, специальная *C++* 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 web-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

5. Языки разработки сценариев Perl и PHP

Язык Perl

Язык *Perl* (Practical Extraction and Report Language) — это язык программирования, сильными сторонами которого считаются его богатые возможности для работы с текстом, в том числе реализованные при помощи регулярных выражений. Также язык известен тем, что имеет огромную коллекцию дополнительных модулей CPAN.

Чтобы запустить программу на языке Perl на выполнение, ее компиляция не требуется, она вполне может выполняться под управлением интерпретатора. Чтобы файл с исходным текстом Perl можно было запускать на выполнение, надо чтобы первая его строка выглядела так:

#!путь_к_интерпретатору_Perl

Основными типами данных в языке являются: скаляры, массивы (скалярные), хеш-таблицы (ассоциативные массивы), функции, файловые дескрипторы и константы.

Переменные разных типов отличаются знаком, который стоит перед именем переменной:

$a - *скаляр или указатель*
@b - *скалярный массив*
%c - *ассоциативный массив (хеш-таблица)*
&d - *функция*
F - *дескриптор ввода-вывода или константа*

Скалярные переменные используются для хранения одиночных значений. Они могут содержать *числа, строки и ссылки* на другие объекты. Перед именем скалярной переменной необходимо ставить знак доллара '$'. *Тип* скалярной переменной не фиксирован и определяется динамически в зависимости от контекста.

Скалярный массив является упорядоченным списком скаляров. Каждый элемент массива имеет порядковый номер (индекс), с помощью которого к нему можно получить доступ. Нумерация элементов начинается с нуля.

Перед именем переменной типа *скалярный массив* указывается знак **@**, а для доступа к *определенному элементу* массива необходимо ставить знак **$**, так как определенный элемент массива является скаляром:

```
@winter = ("декабрь", "январь", "февраль");
print "Второй месяц зимы ", $winter[1], "\n";
```

Хеш-таблица представляет собой ассоциативный массив, позволяющий ассоциировать строку (*ключ*) со скаляром (*значение*). Строка при этом называется *ключом*, а скаляр в хеш-таблице - *значением*. Перед именем переменной-списка необходимо ставить знак процента **%**, а для доступа к определенному элементу массива ставят знак **$**.

Фактически хеш-таблица представляет собой массив, где в *нечетных* позициях находятся *ключи*, а на *четных* - *значения*.

Использование ассоциативных массивов напоминает использование массивов скалярных значений, однако индексация производится не целыми числами, а ключевыми словами. Кроме того, индексы заключаются не в квадратные, а в фигурные скобки.

Так, например, для того чтобы присвоить значение трем элементам массива *%dict* с индексами *first*, *second* и *third*, можно воспользоваться одним из двух способов, указанных ниже.

```
$dict {  'first' } = "первый";
$dict {  'second' } = "второй";
$dict {  'third' } = "третий";
```

или

```
%dict { 'first', 'second', 'third' } = "первый", "второй", "третий";
```

Кроме того, существует способ одновременно записать в ассоциативный массив и ключевые слова, и их значения. Сделать это можно с помощью следующего выражения:

```
имя_массива = ( ключ 1, значение 1, ключ 2, значение 2, ... );
```

Для примера, приведенного выше, это выражение будет выглядеть так:

```
%dict = ("first", "первый", "second", "второй", "third", "третий");
```

Рассмотрим приведенный следующий фрагмент программы на языке Perl.

```
while (<STDIN>)
{ print; }
```

Несмотря на то, что формально программа составлена правильно, она на первый взгляд может показаться бессмысленной. Однако при запуске она ведет себя точно так же, как и одна из программ "эхо". В языке Perl существует предопределенная скалярная переменная *$_*, используемая по умолчанию. Именно в нее помещаются данные, прочитанные из стандартного ввода, и из нее берется значение для вывода в STDOUT.

Кроме *$_* в Perl имеются и другие предопределенные переменные:

- *$]* - номер версии Perl.
- *$.* - номер строки, прочитанной из файла последней.

- *$!* - сообщение об ошибке.
- *$$* - идентификатор текущего процесса.
- *$^T* - время в секундах с начала 1970 года до запуска данной программы.
- *$O* - имя файла, в котором содержится выполняемая программа.
- *$1...$9* - фрагменты текста, отмеченные при выполнении операции сопоставления с шаблоном.

Подобно предопределенным скалярным переменным, в Perl существуют массивы, имеющие специальное значение. Наиболее важный из них - ассоциативный массив *%ENV*, содержащий текущие значения переменных окружения. Чтобы получить значение переменной окружения, надо обратиться к элементу данного массива, указав в качестве индекса имя переменной окружения. Так, приведенное ниже выражение записывает в скалярную переменную *$path_string* значение переменной окружения *PATH*.

$path_string = $ENV { 'PATH' };

Одна из первых строк CGI-сценария на Perl, может выглядеть так

$method = $ENV { 'REQUEST_METHOD' };

Для работы с файлами и потоками в Perl предусмотрены специальные *файловые дескрипторы*.

Файловые дескрипторы представляет собой указатель на файл, устройство или PIPE канал, открытые для записи, чтения или для записи и чтения. Оператор “<>” в Perl называется *бриллиантовым оператором* (diamond operator). Он определяет операцию чтения строки из потока, дескриптор которого содержится в угловых скобках:

$str=<STDIN>; #чтение строки из дескриптора STDIN (стандартного потока ввода)

@lines=<F>; #чтение всех строк из связанного с дескриптором файла F.

print STDOUT $str; #печать в STDOUT (стандартный поток вывода)

Для связывания файла с файловым дескриптором используется функция *open*. Ниже приводятся варианты использования этой функции:

open дескриптор_потока > имя_файла	файл открывается для вывода данных. Если файл с указанным именем отсутствует, создается новый файл.
open дескриптор_потока >> имя_файла	файл открывается в режиме, позво-ляющем записывать данные в конец

	файла.
open дескриптор_потока +> имя_файла	открытый файл становится доступным для чтения и для записи.

Функция

close дескриптор_файла

закрывает файл, связанный с указанным дескриптором.

В состав языка Perl входят средства поиска и замены, причем, задавая шаблон для поиска, можно использовать регулярные выражения. Это значит, что сложные операции, встречающиеся в специализированных приложениях, можно легко реализовать в любой Perl-программе.

Оператор поиска **m//** записывается следующим образом:

m/шаблон/

Если значение переменной *$_* содержит подстроку, соответствующую указанному шаблону, оператор поиска возвращает значение *true*.

Рассмотрим следующий пример:

```
$_ = <INPUT>;
if (m/Scripts/)
  { print "B URL есть каталог Scripts \n"; }
else
  { print " B URL нет каталога Scripts \n"; }
```

Оператор замены **s///** записывается следующим образом:

s/шаблон поиска/выражение для замены/[набор модификаторов]

При выполнении оператора s/// производится поиск соответствия шаблону, и если поиск завершается успешно, найденная подстрока заменяется указанным выражением. Подобно оператору m//, оператор s/// использует переменную $_. Ниже приведен простейший пример применения оператора s///.

```
$_ = "CGI-сценарий написан на языке C";
s/C$/Perl/;
print;
```

В результате выполнения сценария на консоль будет выведена следующая строка:

CGI-сценарий написан на языке Perl

Поскольку символ *C* содержится в аббревиатуре CGI, поэтому в шаблоне поиска указано, что он должен быть последним в строке.

За последним разделителем в операторе **s///** могут следовать один или несколько модификаторов. Назначение некоторых модификаторов приведено ниже.

- **g** - глобальный поиск. Если этот модификатор не указан, после обнаружения первого соответствия оператор **s///** закончит свою работу. Поэтому при отсутствии модификатора *g* будет произведено не более одной замены.
- **i** - указывает, что при поиске следует игнорировать регистр символов.
- **e** - указывает, что последовательность символов для замены следует интерпретировать не как подстроку, а как выражение Perl.

В выражении для подстановки могут присутствовать переменные *$1 - $9*, и в этом случае необходимо указать модификатор *e*. Так, например, если требуется интерпретировать десятичное число как код символа, можно воспользоватъся следующим выражением:

s/([0-9]+)/chr($1)/e;

Чтобы поиск или замена производились в строке, содержащейся в нужной переменной, надо использовать следующее выражение:

Переменная =~ оператор_поиска_или_замены

Так, например, для преобразования шестнадцатеричных чисел, содержащихся в переменной $*string*, в десятичное представление можно использовать инструкцию:

$string =~ s/([0-9A-Fa-f]+)(H|h)/hex($1)/ge;

Язык PHP

Язык PHP (*PHP:Hypertext Preprocessor*) - один из наиболее популярных сценарных языков ввиду своей простоты, скорости выполнения, богатой функциональности и распространенности исходных кодов на основе лицензии PHP.

PHP состоит из ядра и набора подключаемых расширений: для работы с базами данных, сокетами, динамической графикой, криптографическими библиотеками, документами формата PDF и др. Возможна разработка своих собственных расширений с их последующим подключением. Хотя и существуют сотни расширений, однако в стандартную поставку входит лишь несколько десятков хорошо зарекомендовавших себя расширений.

Интерпретатор PHP подключается к web-серверу либо через *DLL модуль*, созданный специально для этого сервера, либо в виде *CGI-приложения*.

В настоящее время PHP используется сотнями тысяч разработчиков. Порядка 20 миллионов сайтов сообщают о работе с PHP, что составляет более пятой доли доменов Интернета

Синтаксис PHP подобен синтаксису языка Си. При этом некоторые элементы, как например *ассоциативные массивы* и цикл *foreach*, заимствованы из языка *Perl*.

Для работы программы на PHP не требуется описывать какие-либо переменные, используемые модули, и т.п. Любая программа может начинаться непосредственно с оператора PHP.

```
<?php
    echo 'Hello, world!'; ?>
```

Помимо ограничителей *<?php ?>*, допускается использование дополнительных вариантов, таких как *<? ?>* и *<script language="php"> </script>*. Кроме того, до версии 6.0 допускается использование ограничителей языка программирования ASP *<% %>*.

Имена переменных начинаются с символа *$*, тип переменной объявлять не требуется. В отличие от имён функций и классов, имена переменных *чувствительны к регистру*. Переменные обрабатываются в строках, заключённых в двойные кавычки.

Инструкции завершаются точкой с запятой (**;**)

PHP поддерживает два типа *комментариев*:

- в стиле языка *C* (ограниченные **/* */**),
- C++ (начинающиеся с **//** и идущие до конца строки)

PHP является языком программирования с *динамической типизацией*, не требующим указания типа при *объявлении* переменных, равно как и самого объявления переменных. *Преобразования* между скалярными типами может осуществляться *автоматически* (хотя и имеются возможности для явного преобразования типов).

К *скалярным* типам данных относятся

- целый тип (*integer*),
- вещественный тип данных (*float, double*),
- логический тип (*boolean*),
- строковый тип (*string*)
- специальный тип *NULL*.

К *нескалярным* типам относится

- «ресурс» (*resource*),

- массив (*array*)
- и объект (*object*).

Тип *NULL* предназначен для переменных без определённого значения. Значение *NULL* принимают неинициализированные переменные, переменные инициализированные константой *NULL*, а также переменные, удалённые при помощи конструкции *unset()*.

Ссылки на внешние ресурсы имеют тип *resource*. Переменные данного типа, как правило, представляют собой дескриптор, позволяющий управлять внешними объектами, такими как файлы, динамические изображения, результирующие таблицы базы данных и т. п.

Массивы поддерживают *числовые* и *строковые* ключи и являются гетерогенными. Массивы могут содержать значения любых типов, включая другие массивы. Суперглобальными массивами (*superglobal arrays*) в *PHP* называются предопределённые массивы, которые видны в любом месте исходного кода без использования ключевого слова *global*.

- $GLOBALS - массив всех глобальных переменных (в том числе и пользовательских).
- $_SERVER - содержит множество информации о текущем запросе и сервере.
- $_ENV - текущие переменные среды. Их набор специфичен для каждой конкретной платформы, на которой выполняется сценарий.
- $_GET - ассоциативный массив с параметрами *GET*-запроса. В исходном виде эти параметры доступны в *$_SERVER['QUERY_STRING']* и в *$_SERVER['REQUEST_URI']* в составе *URI*.
- $_POST - ассоциативный массив значений полей *HTML-формы* при отправки методом *POST*.
- $_FILES - ассоциативный массив со сведениями об отправленных методом *POST* файлах. Каждый элемент имеет индекс идентичный значению атрибута «*name*» в форме и, в свою очередь, также является массивом со следующими элементами:
    - ['*name*'] — исходное имя файла на компьютере пользователя.
    - ['*type*'] — указанный агентом пользователя *MIME-тип* файла.
    - ['*size*'] — размер файла в байтах.
    - ['*tmp_name*'] — полный путь к файлу во временной папке.
    - ['*error*'] — код ошибки.
- $_COOKIE - ассоциативный массив с переданными агентом пользователя значениями *cookie*.

- $_REQUEST - общий массив вводных данных запроса пользователя как в массивах *$_GET*, *$_POST*, *$_COOKIE*. Начиная с версии PHP 4.1 включается и содержимое *$_FILES*.
- $_SESSION - информация о текущей сессии пользователя.

PHP поддерживает широкие *объектно-ориентированные* возможности, полная поддержка которых была введена в пятой версии языка. Класс в PHP объявляется с помощью ключевого слова *class*. Методы и поля класса могут быть общедоступными (*public*, по умолчанию), защищёнными (*protected*) и скрытыми (*private*). PHP поддерживает все три основные механизма ООП — *инкапсуляцию*, *полиморфизм* и *наследование* (родительский класс указывается с помощью ключевого слова *extends* после имени класса). Поддерживаются интерфейсы (ставятся в соответствие с помощью *implements*). Разрешается объявление *финальных*, *абстрактных* методов и классов. Множественное наследование классов не поддерживается, однако класс может реализовывать несколько интерфейсов. Для обращения к методам родительского класса используется ключевое слово *parent*. Экземпляры класса создаются с помощью ключевого слова *new*, обращение к полям и методам объекта производится с использованием символов ->. Для доступа к членам класса из его методов используется переменная *$this*.

Среди наиболее часто используемых возможностей PHP стоит отметить следующие:

- имеется большой набор функций для работы со строками;
- работа с регулярными выражениями *PCRE*.
- работа с базами данных, осуществляемая посредством модулей:
  - *php5-mysql* для MySQL,
  - *php5-pgsql* для PostgreSQL
  - и др.
- для PHP разработаны средства шаблонирования web-страниц, позволяющие эффективно разделить представление от модели, например *Smarty*;
- имеется библиотека для работы с графическими изображениями *GD*, позволяющая производить преобразования с графическими файлами, и создавать изображения «на лету».

6. Введение в C# и платформу Visual Studio.Net

Microsoft Visual Studio .NET - это интегрированная среда разработки для создания, документирования, запуска и отладки программ, написанных на языках .NET.

Эта среда разработки является открытой языковой средой. Наряду с языками программирования, изначально включенными в среду - *C++, C#, J#, Visual Basic,* - в нее могут добавляться любые языки программирования, компиляторы которых создаются сторонними разработчиками. Необходимым условием для включения языков в среду Visual Studio .Net является использование единого каркаса – платформы *Framework.Net*.

Платформа *Framework.Net* позволяет:

- Легко использовать компоненты, разработанных на различных языках;
- Разрабатывать единое приложение из нескольких частей на разных языках;

Платформа *Framework .Net* содержит две основных компоненты:

- *FCL* (Framework Class Library) - библиотеку классов каркаса;
- *CLR* (Common Language Runtime) - общеязыковую исполнительную среду.

В рамках данной платформы используется стандартная система типов *Common Type System* (CTS), которая полностью описывает все типы данных, поддерживаемые средой выполнения, определяет взаимодействие типов данных и их представление в формате метаданных .NET.

Набор правил, определяющих подмножество общих типов данных, в отношении которых гарантируется, что они небезопасны при использовании во всех языках .NET, описывается в рамках спецификации *Common Language Specification* (CLS). Для того чтобы классы, разработанные на разных языках, можно было совместно использовать в рамках одного приложения, они должны удовлетворять определенным ограничениям, задаваемым *CLS*. Класс, удовлетворяющий *CLS*, называется *CLS-совместимым*. Он доступен для использования в других языках, классы которых могут быть клиентами или наследниками совместимого класса.

Платформа .NET предоставляет в распоряжение программиста библиотеку базовых классов, доступную из любого языка программирования .NET. Поскольку число классов библиотеки FCL достигает нескольких тысяч, то в целях структуризации функционально близкие классы объединяются в группы, называемые *пространством имен* (Namespace).

Основным пространством имен библиотеки FCL является пространство *System*, содержащее как классы, так и другие вложенные пространства имен. Например, в пространстве *System.Collections* находятся классы и интерфейсы, поддерживающие работу с коллекциями объектов - списками, очередями, словарями. Пространство *System.Windows.Forms* содержит классы, используемые при создании windows-приложений.

Следует помнить, что C# генерирует код, предназначенный для выполнения только в среде выполнения .NET (управляемый код). Сам двоичный файл, содержащий управляемый файл, называется сборкой. Сборка содержит код на промежуточном языке MSIL (Microsoft Intermediate Language) или просто IL. Аналогично байт-коду Java IL-код компилируется в платформенно-специфические инструкции при непосредственном обращении среды выполнении .NET к блоку IL-инструкций. Двоичные модули .NET в дополнение к инструкциям содержат также *метаданные*. *Метаданные* описывают не только типы, используемые в сборке, но и саму сборку. Данная часть метаданных называется *манифестом*.

В большинстве случаев между двоичным файлом .NET и сборкой существует отношение «*один-к-одному*». Однако сборка может состоять как из одного, так и из нескольких двоичных файлов.

Сборка из одного файла содержит и манифест, и метаданные, и инструкции IL.

Двоичные файлы, образующие совместно общую сборку называются *модулями*. При этом один из двоичных файлом должен содержать манифест сборки. Остальные модули могут содержать только метаданные типов и инструкции IL.

Многофайловые сборки позволяют среде исполнения избирательно загружать только те сборки, которые в данный момент работы приложения действительно необходимы, что позволяет сокращать сетевой трафик и увеличивать скорость работы программ.

Основы C#

Что нового принес язык C# по сравнению с уже существовавшими до него языками?

- В программах на C#, как правило, нет необходимости в работе с указателями (при сохранении этой возможности), поскольку в нем реализовано автоматическое управление памятью.
- Предусмотрены встроенные синтаксические конструкции для работы с перечислениями, структурами и свойствами классов.

- Имеется полная поддержка программных интерфейсов. Использование двоичных модулей .NET позволяет передавать объекты (по ссылке или по значению) через границы программных модулей.
- Полная поддержка объектно-ориентированных технологий.

Для разработки приложений в Visual Studio.Net используются *проекты*.

Проект (Project) - это основная единица, с которой имеет дело разработчик. Сначала он должен выбрать тип проекта, после чего Visual Studio создает каркас проекта в соответствии с выбранным типом. Проект состоит из *классов*, собранных в одном или нескольких пространствах имен. *Пространства имен* (Namespaces) позволяют структурировать проекты, содержащие большое число классов, объединяя в одну группу близкие классы.

Несколько проектов могут объединяться в *решение* (Solution), которое также может включать ресурсы, необходимые этим проектам.

С точки зрения разработчика конечным результатом его работы, получаемым после компиляции исходного программного кода, является решение, а с точки зрения CLR (Common Language Runtime - общеязыковой среды исполнения) – сборка (assembly), содержащая PE файл, т.е. модуль в формате исполняемого файла PE (Portable Executable) для 32-разрядной ОС Windows либо DLL (Dynamic Link Library) файл.

Visual Studio.Net предлагает большое разнообразие возможных типов проектов.

Типы данных C#

Стандарт языка C++ включает следующий набор фундаментальных типов.

- Логический тип (*bool*).
- Символьный тип (*char*).
- Целые типы. Они могут отличаться размером: *short, int, long*, а также могут быть знаковыми (*signed*) или беззнаковыми (*unsigned*).
- Типы с плавающей точкой. Они также могут отличаться размерами: *float, double* и *long double*.

Тип *void* указывает на отсутствие информации.

К *конструируемым* типам относятся следующие:

- Указатели (например, *char**).
- Ссылки (например, *char&*).
- Массивы (например, *char[]*).

Также язык позволяет разработчику конструировать собственные типы:

- Перечислимые типы (*enum*).
- Структуры (*struct*).
- Классы.

В языке C# все типы можно рассматривать и под другим ракурсом, разделив их на четыре категории:

- Типы-значения (*value*).
- Ссылочные (*reference*).
- Указатели (*pointer*).
- Тип *void*.

Для ссылочного типа значение задает ссылку на область памяти в "куче" (heap), где расположен соответствующий объект. Для типа-значения значением являются собственно данные, а память для них выделяется в стеке.

Логический, арифметический, структуры, перечисление относятся *типам-значениям*. Массивы, строки и классы относятся к *ссылочным* типам.

И ссылочные, и обычные типы являются производными от базового класса *object*. В тех случаях, когда обычный тип должен вести себя как объект, создается оболочка (wrapper), которую можно рассматривать как ссылочный объект, помещенный в кучу, и в нее копируется значение переменной обычного типа. Оболочка автоматически помечается таким образом, что система знает, какое значение она содержит. Этот процесс называется *упаковкой* (boxing), а обратный процесс - *распаковкой* (unboxing).

Упаковка происходит автоматически, для этого нужно только присвоить значение обычного типа переменной типа *object*. Упаковка и распаковка позволяют обрабатывать любой тип как объект. Например, в выражении

```
7.ToString();
```

целое число 7 упаковывается путем вызова функции *Int32.ToString()*.

Массивы в C# могут быть *многомерными* (multidimensional) или *невыровненными* (jagged). Более сложные структуры данных такие, как *стек* и *хеш-таблица* определены в пространстве имен *System.Collections*.

В языке C# определен класс *char[]*, и его можно использовать для представления строк постоянной длины. Однако массив *char[]* - это обычный массив, поэтому его нельзя инициализировать строкой символов. В C# не определено преобразование из класса *char[]* в класс *String*. У *String* есть динамический метод *ToCharArray*, задающий подобное преобразование в *char[]*.

Класс *String* не разрешает изменять существующие объекты. Класс *StringBuilder* позволяет исправить этот недостаток. Этот класс принадлежит к изменяемым классам и его можно найти в пространстве имен *System.Text*.

Классы и методы в C#.

В Visual Studio.Net, и в C# в частности, любая программная система рассматривается как *совокупность классов*, объединенных в проекты, пространства имен, решения.

Описание класса имеет следующий синтаксис*:*

*[атрибуты][модификаторы]**class** имя_класса[:список_родителей]*
{тело_класса}

В теле класса могут быть объявлены:

- константы;
- поля;
- конструкторы и деструкторы;
- методы;
- события;
- делегаты;
- классы (структуры, интерфейсы, перечисления).

Поля класса синтаксически являются обычными переменными (объектами) языка. Их описание удовлетворяет обычным правилам объявления переменных. Поля характеризуют *свойства объектов* класса.

Методы класса синтаксически являются обычными процедурами и функциями языка. Методы содержат описания операций, доступных над объектами класса. Методы, называемые *свойствами* являются специальной синтаксической конструкцией, предназначенной для обеспечения эффективной работы с классами.

Конструктор представляет собой специальный метод класса, позволяющий создавать объекты класса. Его имя должно совпадать с именем класса. Если разработчик не определяет конструктор класса, то к классу автоматически добавляется конструктор по умолчанию - конструктор без аргументов.

Делегат в C# представляет собой описание случая класса и задает определение функционального типа (класса) данных. Экземплярами класса являются функции. Каждый *делегат* описывает множество функций с заданной сигнатурой. Каждая функция (метод), сигнатура которого совпадает с сигнатурой *делегата*, может рассматриваться как экземпляр класса, заданного *делегатом*. Синтаксис объявления *делегата* имеет следующий вид:

[<спецификатор доступа>] ***delegate*** *<тип результата> <имя класса> (<список аргументов>);*

Выражения и операторы C#.

Выражения строятся из операндов - констант, переменных, функций, - объединенных знаками операций и скобками. При вычислении выражения определяется его *значение* и *тип*.

В таблице ниже приведен список операций C#.

Категория операций	Операции
Арифметические	+ – * / %
Логические (*boolean* и побитовые)	& | ^ ! ~ && ||
Строковые	+
Инкремент и декремент	++ --
Сдвиг	>> <<
Сравнение	== != < > <= >=
Присвоение	= += –= *= /= %= &= |= ^= <<= >>=
Обращение к члену класса	.
Индексация	[]
Приведение типа (Cast)	()
Условие	?:
Создание объекта	new()
Информация о типе	is sizeof typeof
Управление исключениями	checked unchecked
Косвенности и адресации	* –> [] &

Имя и *тип* переменной задаются при ее объявлении и остаются неизменными в течение всего времени ее жизни. Особенностью языка C# является требование *обязательной инициализации переменной до начала ее использования*. Попытка использовать неинициализированную переменную приводит к ошибкам, обнаруживаемым еще на этапе компиляции.

По используемым выражениям и операторам C# похож на C++. Так в программах на C# используются такие операторы как:

- Оператор присваивания (=)
- Составной оператор (***{}***)
- Операторы выбора: ***if-else*** и ***switch***
- Операторы цикла: ***for***, ***while***, ***оператор***
- Операторы ***break*** и ***continue***
- Оператор ***return***
- Оператор перехода ***goto***

Кроме того, введены несколько новых инструкций. Например, оператор ***foreach*** позволяет получить доступ ко всем элементам массива или коллекции поочередно, в порядке возрастания индексов. Его синтаксис:

foreach (*тип идентификатор* **in** *контейнер*) *оператор*

Интерфейсы C#

Интерфейс представляет собой полностью абстрактный класс, все методы которого абстрактны. Однако методы *интерфейса* объявляются без указания модификатора доступа, и класс, наследующий *интерфейс*, обязан полностью реализовать все методы *интерфейса*. В этом - отличие от класса, наследующего абстрактный класс, где потомок может реализовать лишь некоторые методы родительского абстрактного класса, оставаясь абстрактным классом.

Интерфейс позволяет описывать некоторые желательные свойства, которыми могут обладать объекты разных классов.

Среди интерфейсов, встроенных в библиотеку базовых классов .NET, можно особо выделить такие как:

- *IEnumerable* (для работы с наборами объектов, в т.ч. с использованием оператора *foreach*).
- *IClonable* (копирование объектов).
- *IComparable* (для сравнения и сортировки объектов).

Пространство имен System.Collections, предназначенное для работы с наборами объектов, поддерживает интерфейсы:

- *ICollection* (определяет общие характеристики класса набора элементов).
- *IComparer*, *IDictionary* (позволяет представлять содержимое объекта в виде пар имя-значение).
- *IDictionaryEnumerator* (нумерация содержимого объекта, поддерживающего IDictionary).
- *IEnumerable*, *IEnumerator*.
- *IHashCodeProvider* (возвращает хэш-код с помощью выбранного алгоритма хэширования).
- *IList* (обеспечивает методы добавления, удаления и индексирования элементов в списке объектов).

Сериализация объектов

При работе с программной системой зачастую возникает необходимость в сериализации объектов. Под *сериализацией* понимают процесс сохранения

объектов в долговременной памяти (файлах) в период выполнения системы. Под десериализацией понимают обратный процесс - восстановление состояния объектов, хранимых в долговременной памяти.

Механизмы сериализации C# и Framework.Net поддерживают два формата сохранения данных - в бинарном файле и XML файле. В первом случае данные при сериализации преобразуются в бинарный поток символов, который при десериализации автоматически преобразуется в нужное состояние объектов. Другой возможный преобразователь запоминает состояние объекта в формате *XML*.

Если класс объявить с атрибутом [*Serializable*], то в него встраивается стандартный механизм сериализации, поддерживающий глубокую сериализацию. Если по каким-либо причинам стандартная сериализация разработчика не устраивает, то класс следует объявить наследником интерфейса *ISerializable*, реализация методов которого позволит управлять процессом сериализации.

7. Архитектура web-приложений ASP.NET. Разработка web-приложений на платформе .NET.

Платформа .NET Framework предоставляет возможность разработки и интеграции web-приложений. ASP.NET является одной из составляющих инфраструктуры .NET Framework и фактически является платформой для создания web-приложений и web-сервисов, работающих под управлением IIS.

ASP.NET внешне во многом напоминает более старую технологию ASP, но в то же время внутреннее устройство ASP.NET существенно отличается от ASP. Компания Майкрософт ASP.NET построила на базе CLR (Common Language Runtime), который является основой всех приложений .NET. Разработчики могут создавать код для ASP.NET, используя языки программирования, входящие в .NET Framework: C#, Visual Basic.NET, JScript.NET и другие.

Рассмотрим более подробно, чем отличается ASP.NET от ASP.

Классический ASP имеет следующие недостатки**:**

- Используются только *языки сценариев*, которые дают большой проигрыш в производительности (из-за их интерпретируемости) и не поддерживают многие возможности объектно-ориентированного программирования.
- *Логика представления* (в виде кода HTML) не отделена от *бизнес-логики* (исполняемого кода), что приводит перемешиванию в одном файле кода HTML с кодом сценария.
- Невозможно *повторно использовать готовые решения* в других проектах (возможно только копирование кода сценариев).

В файлах ASP.NET включается код на таких языках программирования как C#, JScript.NET, VisualBasic.NET, что позволяет применять непосредственно в web-приложениях возможности объектно-ориентированного программирования. Также существенно сокращается объем кода, написанного вручную за счет применения серверных объектов, автоматически генерирующих код элементов управления HTML. Возможно использование стандартной среды разработки *Visual Studio.NET*, т.е. ASP.NET имеет преимущество в скорости по сравнению со сценарными технологиями, так как при первом обращении код компилируется и помещается в специальный кеш, а впоследствии только исполняется, не требуя затрат времени на парсинг, оптимизацию, и т. д.

Несмотря на возможность совместной работы *ASP* и *ASP.NET* на одном web-сервере, они не могут использовать общий сеанс. Файлы *ASP.NET*

обрабатываются библиотекой *aspnet_isapi.dll* (а не *asp.dll*), которая, в свою очередь, использует для выполнения кода технологию *.NET*.

Библиотека базовых классов .NET содержит пространства имен 3 основных групп:

- элементы *web-приложений* (протоколы, безопасность и др.);
- элементы *графического интерфейса* (WebForms) ;
- *web-службы*.

Как уже указывалось ранее, ASP.NET использует возможности стандартной среды разработки *Visual Studio.Net,* и в частности классы библиотеки FCL (Framework Class Library).

Разработчику web-приложений на ASP.NET доступны классы, входящие в следующие пространства имен:

Пространство имен	**Содержание**
System.Web	Организация взаимодействия web-клиента (браузера) с web-сервером (запрос-ответ, cookie и и др.)
System.Web.Caching	Поддержка кэширования при работе web-приложений
System.Web.Configuration	Настройка web-приложения в соответствии с файлами конфигурации проекта
System.Web.Security	Реализация системы безопасности web-приложений
System.Web.Services	Организация работы web-сервисов
System.Web.Services.Description	
System.Web.Services.Discovery	
System.Web.Services.Protocols	
System.Web.UI	Построение графического интерфейса пользователей web-приложений
System.Web.UI.WebControls	
System.Web.HtmlControls	

В свою очередь пространство имен *System.Web* включает в себя пространства имен, названия которых знакомы разработчикам web-приложений на ASP:

Пространство имен	Содержание
HttpApplication	Данный класс определяет общие для всех web-приложений члены
HttpApplicationState	В данном классе содержится общая информация web-приложения для множества запросов, сеансов и каналов передачи данных
HttpBrowserCapabilities	Этот класс используется для получения информации о возможностях клиентского браузера, обращающегося к web-серверу
HttpCookie	Поддержка механизма безопасной работы с объектами HTTP cookie
HttpRequest	Предоставляет доступ к информации, переданной web-клиентом
HttpResponse	Используется для формирования HTTP-ответа сервера

В основу разработки web-приложений на ASP.NET положена модель *разделения кода представления и кода реализации*, рекомендуемая Майкрософт при создании динамических документов на с помощью программных кодов. Это делается путем размещения программного кода либо в отдельный файл, либо внутри специального тэга для сценариев. Файл такого рода обычно имеет расширение **.aspx.cs* (*.aspx.vb) и имеет имя, совпадающее с именем основного ASPX файла. В принципе такой подход позволяет web-дизайнеру сконцентрироваться работе с кодом разметки документа с минимальными изменениями программного кода, в обычном ASP внедряемого непосредственно в код разметки.

Взаимодействие пользователя с web-приложением, реализованном на ASP.NET включает в себя следующие процессы:

- При запросе страницы *ASPX* инициируется событие *Page_Init*, производящее начальную инициализацию страницы и ее объекта.
- Далее инициируется событие *Page_Load*, которое может быть использовано, например для установки начальных значений для элементов управления. При этом также можно определить была ли загружена страница впервые или обращение к ней осуществляется повторно в рамках обратной отсылки в ответ на события, связанные с элементами управления, размещенными на странице; т.е. проверить свойство *Page.IsPostBack*.

- Далее выполняется проверка валидности элементов страницы с точки зрения корректности введенных пользователем данных.
- И, наконец, следует обработка всех событий, связанных с действиями пользователя с момента последней обратной отсылки.

Для сохранения данных web-страницы в промежутках между обращениями к ней в ASP.NET используются состояния отображения (view state).

Если данные, введенные в web-форму, необходимо сделать доступными другим web-формам того же приложения, эти данные необходимо сохранить в объектах *Application* и *Session*. Объекты *Application* доступны всем пользователям приложения и могут рассматриваться как глобальные переменные, обращение к которым возможно из любых сеансов. Объекты *Session* доступны только в рамках одного сеанса, и поэтому они оказываются доступными только одному пользователю.

Серверные элементы управления ASP.NET

Важной особенностью ASP.NET является использование *серверных элементов управления* на web-странице (элементы *WebForm*), которые являются фактически тэгами, понятными web-серверу. Эти элементы определены в пространстве имен *System.Web.UI.WebControls*.

Принято выделять три типа серверных элементов управления:

- Серверные *элементы управления HTML* – обычные HTML тэги.
- *Элементы управления web-сервера* – новые тэги ASP.NET.
- Серверные *элементы управления для проверки данных (валидации)* – применяются для валидации входных данных от клиентского приложения (обычно web-браузера).

Преимущества от использования таких элементов при разработке web-приложений:

- Сокращается количество кода, написанного вручную (что особенно заметно в для сложных элементов документа). Элемент просто «перетаскивается» из панели инструментов, после чего выполняется настройка его параметров в специальном окне. При этом все изменения автоматически заносятся непосредственно в **.aspx* файл.
- С программной точки зрения каждому из этих элементов управления соответствует определенный класс в библиотеке базовых классов .NET, что позволяет писать для них такой же код как и для любых других классов.

- Для любого элемента управления WebForm определен набор событий, обрабатываемых на web-сервере.
- Для любого элемента управления WebForm предоставляется возможность для проверки ввода данных пользователем.

По умолчанию серверные *элементы управления HTML* в ASP.NET файлах рассматриваются как текст. Для их программирования требуется добавление атрибута *runat="server"* в соответствующий HTML элемент. Кроме того, все серверные элементы управления HTML должны быть размещены внутри области действия тэга <*form*>, также имеющего атрибут *runat="server"*.

Подобно серверным элементам управления HTML *элементы управления web-сервера* также создаются на web-сервере и предполагают добавление атрибута *runat="server"*. Однако они могут и не соответствовать конкретным элементам HTML, но представлять более сложные элементы.

Общий синтаксис для описания таких элементов:

<asp:*тип_элемента* id="*идентификатор*" runat="server"/>

Серверные элементы валидации применяются для проверки вводимых пользователем данных.

Имеют следующий синтаксис:

<asp:*тип_элемента* id="*идентификатор*" runat="server" />

Работа с источниками данных в ASP.NET

В ASP.NET используются два элемента управления WebForm для управления отображением данных, получаемых из источника данных:

- *DataGrid* - элемент управления, отображающий содержимое объекта ADO.NET *DataSet* в виде таблицы.
- *DataList* - элемент управления для выбора значений, заполняемых из источника данных.

Если необходимо отобразить данные, полученные по запросу пользователя из источника данных, в виде таблицы на web-странице, то ASP.NET предоставляет в распоряжение web-программиста удобный элемент управления *DataGrid*.

8. Интерфейсы взаимодействия web-приложений с СУБД.

Сегодня большинство информационных систем в той или иной степени используют базы данных. Не составляют исключение и системы, основанные на web-технологиях. Поэтому организация взаимодействия web-приложений с СУБД является неотъемлемой составной частью web-технологий.

До начал 90-х годов существовало несколько разных поставщиков баз данных, каждый из которых имел собственный интерфейс. Если приложению было необходимо обмениваться данными с несколькими источниками данных, для взаимодействия с каждой из баз данных было необходимо написать отдельный код. С целью решения этой проблемы Майкрософт и ряд других компаний создали *стандартный интерфейс* для получения и отправки данных источникам данных различных типов. Этот интерфейс получил название *open database connectivity* (ODBC).

С помощью ODBC прикладные программисты смогли разрабатывать приложения с использованием единого интерфейса доступа к данным, не учитывая тонкости взаимодействия с различными источниками данных. Это достигается благодаря тому, что поставщики различных баз данных разрабатывают *драйверы*, учитывающие специфику конкретных источников данных при реализации стандартных функций из ODBC API. При этом приложения используют функции такого API, реализованные в соответствующем конкретному источнику данных драйвере.

По-сути, интерфейс ODBC является обычным *процедурным* API. *ODBC* поддерживается большим количеством операционных систем.

Имеются также ODBC-драйверы и для нереляционных данных, таких как *электронные таблицы*, *текст* и *XML* файлы.

Типичный сценарий работы web-приложения с источником данных выглядит следующим образом:

1. Установление соединение и подключение к источнику данных.
2. Выполнение запросов, необходимых для выборки, вставки или изменения наборов данных источника.
3. Отключение от источника данных.

Компанией Майкрософт был предложен интерфейс программирования приложений для доступа к данным, разработанный и основанный на технологии компонентов *ActiveX* - ADO (ActiveX Data Objects), который позволяет представлять данные из разнообразных источников (реляционных баз данных, текстовых файлов и т. д.) в объектно-ориентированном виде.

Компоненты ADO нашли применятся при разработке приложений на таких языках как VBScript в ASP и Visual Basic.

В рамках Microsoft .NET основной моделью доступа приложений к источникам данных является ADO.NET. Она не является развитием *ADO* и представляет собой совершенно самостоятельную технологию.Компоненты *ADO.NET* входят в поставку *.NET Framework.*

ADO.NET включает в себя две основные части:

- ***Data provider*** - набор классов для доступа к источникам данных. Каждый из источников данных имеет свой собственный набор объектов, однако все они имеют общее множество классов: *Connection, Command, Parameter, DataAdapter, DataReader*.
- ***DataSets*** объекты - группа классов, описывающих простые реляционные базы данных, размещаемы в памяти. Содержит иерархию таких классов как: *DataTable, DataView, DataColumn, DataRow, DataRowView, DataRelation, Constraint.*

Объект *DataSet* заполняется данными из БД с помощью объекта *DataAdapter*, у которого заданы свойства *Connection* и *Command. DataSet* может сохранять свое содержимое также в *XML* (опционально вместе с *XSD* схемой) или получать данные из *XML*.

ADO.NET поддерживает работу с *отсоединенными наборами данных*, что крайне важно при использовании масштабируемых web-приложений. Такая возможность реализуется с помощью класса *DataSet* совместно с классом *DataAdapter*.

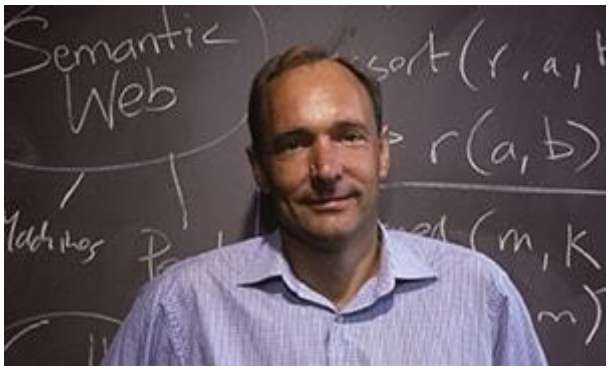
Одной из важнейших составляющих технологии ADO.NET является *поставщик данных*. По-сути, это набор классов, предназначенных для взаимодействия с источником данных определенного типа. Использование разных поставщиков данных делает ADO.NET очень гибкой и расширяемой.

9. Язык разметки гипертекста HTML

Предпосылки создания HTML

Технология гипертекста, зародилась в Европейской организации по ядерным исследованиям (сокр. CERN). С хранением больших объемов научных данных системные администраторы справлялись отлично, но когда дело доходило до их использования, вставал вопрос - «как быстро получить доступ к данным, передавать их от компьютера к компьютеру в структурированном виде и добиться одинакового отображения на любом мониторе?».

Решать эту проблему взялся британский ученый **Тим Бернерс-Ли**. В стенах CERN на базе другого языка разметки SGML он разработал новый. Произошло это ориентировочно в 1986-1991 годах (точная дата не известна). В основу создания и дальнейшего развития HTML были заложены три принципа: простота использования, универсальность и расширяемость.



Принципы работы с гипертекстовой разметкой:

1. Простота использования — реализация минимума структурных элементов и понятного синтаксиса, позволяющие любому человеку, потратив немного времени, научиться правильно составлять и размечать ***документы***.
2. Универсальность — правильно составленный документ должен одинаково отображаться на любом устройстве, независимо от его комплектации или программного обеспечения.
3. Расширяемость — при создании HTML невозможно было учесть всех существующих требований и тех, что могли возникнуть в будущем. Поэтому Бернерс-Ли заложил в него возможность к расширению (добавлению новых структурных элементов, атрибутов и т.д).

HTML – язык разметки гипертекста.

Гипертекст – информационная структура, позволяющая устанавливать смысловые связи между элементами текста на экране компьютера таким

образом, чтобы можно было легко осуществлять переходы от одного элемента к другому. На практике в гипертексте некоторые слова выделяют путем подчеркивания или окрашивания в другой цвет (гиперссылки). Выделение слова говорит о наличии связи этого слова с некоторым документом, в котором тема, связанная с выделенным словом рассматривается более подробно.

Такие страницы как правило имеют расширение htm или html. Отдельный документ, выполненный в формате HTML, называется:

HTML-документом;

Web-документом;

Web-страницей.

Гиперссылка — фрагмент текста, который является указателем на другой файл или объект. Гиперссылки необходимы для того, чтобы обеспечить возможность перехода от одного документа к другому.

Группа Web-страниц, принадлежащих одному автору или одному издателю и взаимосвязанных общими гиперссылками, образует структуру, которая называется Web-узлом или Web-сайтом. Каждая HTML-страница имеет свой уникальный URL-адрес в Интернете.

Семантический элемент — передает особую смысловую нагрузку своему содержимому. Например, адрес организации, главное содержание статьи, заголовок статьи и др. Это имеет большую значимость при сканировании сайтов поисковыми системами либо программами чтения экрана, которыми пользуются люди с ограниченными возможностями.

Структурный элемент — используются только для стилизации оформления и удобной разбивки содержимого страницы на блоки

Заметка

Изначально этот язык разметки имел различные рабочие имена. Официальное и всеми используемое название Hyper Text Markup Language (сокр. HTML) закрепилось в момент создания всемирной паутины.

Появление всемирной паутины

Работая в CERN, Тим Бернерс-Ли занимался не только развитием HTML. В его задачи входило построение внутренней сети организации. Концепции, реализованные в ней, были доработаны и переросли в проект под названием «Всемирная паутина».

Проект подразумевал публикацию документов в открытом доступе, размеченных при помощи HTML, и имеющих ***гиперссылки*** друг на друга, что позволяло реализовать их связанность. Такие ссылки в прямом смысле образуют формацию, похожую на паутину.

Для реализации своей идеи Бернерс-Ли создал специальные программы: HTTP-сервер и WEB-браузер. Первый в мире веб-сайт был размещён 6 августа 1991 года по адресу **http://info.cern.ch/** (вы можете посмотреть на его архивную версию **здесь**). В его содержании описывается принцип работы сети, как установить веб-сервер и создать простую страницу.

Дальнейшее развитие языка

Концепция «всемирной паутины» взорвала умы людей. В октябре 1994 года был создан регулирующий орган World Wide Web Consortium (сокр. W3C).

С тех пор HTML прошел большой путь и стал зрелой фундаментальной технологией. В процессе развития HTML приобрел элементы пользовательских форм, произошло разделение на ***семантические*** и ***структурные*** элементы, а также появилась возможность подключения к страницам скриптов и внешних стилей.

На данный момент последней версией языка является HTML5, принятая 28 октября 2014 года.

Структура HTML-документа

Элемент — конструкция языка HTML. Это контейнер, содержащий данные и позволяющий отформатировать их определенным образом. Любая Web-страница представляет собой набор элементов. Одна из основных идей гипертекста возможность вложения элементов.

Тег — начальный или конечный маркеры элемента. Теги определяют границы действия элементов и отделяют элементы друг от друга. В тексте Web-страницы теги заключаются в угловые скобки, а конечный тег всегда снабжается косой чертой. Например: элемент, содержащий некоторый текст, ограничен начальным тегом (маркером) и конечным тегом (маркером) . Т.е. текст помещен между тегами как в контейнер. Здесь же можно увидеть, как осуществляется возможность вложения элементов. Тег вложен внутрь тега , поэтому конечный тег стоит перед . В данном примере тег указывает на то, что текст является отдельным абзацем, а тег задает, например, формат шрифта.

 Этот текст будет расположен в отдельном абзаце и выполнен зеленым цветом шрифта.

В результате такого форматирования на экране компьютера мы увидим текст зеленого цвета в отдельном абзаце.

Атрибут — параметр или свойство элемента. Это, по сути, переменная, которая имеет стандартное имя и которой может присваиваться определенный набор значений: стандартный или произвольный. Атрибуты располагаются внутри начального тега и отделяются друг от друга пробелами.

 Этот текст будет выровнен по центру экрана . В данном примере атрибут align (выравнивание) расположен внутри тега , следовательно он задает выравнивание этого абзаца. Значение атрибута равно "center", т.е. выравнивание абзаца будет по центру экрана.

Ниже приведена структура типичного Web-документа.

<HTML>	Этот тег указывает на начало HTML-документа
<HEAD>	Этот тег указывает на начало области заголовка Web-страницы. Служит для формирования общей структуры документа.
<TITLE>Название Web-страницы</title>	Элемент для размещения заголовка Web-страницы. Строка отображается в заголовке окна браузера.
<META http-equiv="Content-Type" content="text/html; charset=windows-1251">	Этот тег несет служебную информацию и не отображается на экране браузера. В данном случае идет речь о кодировке Web-страницы. Вам достаточно лишь каждый раз вставлять этот тег в таком виде на свою страничку. Тогда ваша страничка будет использовать кодировку windows-1251, наиболее распространенную на сегодняшний день.
<META name="Author" content="Ivanov Ivan">	Имя автора Web-страницы.
<META name="Keywords" content="WWW, HTML, docoment, страничка, структура">	Набор ключевых слов для поиска. Раньше этими словами пользовались поисковые машины, для отбора сайтов по тематике запроса. Сегодня эти слова поисковыми машинами практически не используются, однако полезно вставить этот тег на свою страничку и указать в нем ключевые слова, отражающие содержание вашего сайта.
</head>	Конец области заголовка Web-

	страницы.
<BODY bgcolor="blue">	Начало собственно содержимого Web-страницы. Тег <BODY> включает в себя атрибут bgcolor, который задает цвет вашей страницы. В данном случае голубой. Если не использовать этот атрибут, то по умолчанию цвет страницы будет белым.
<H2> Здесь расположен заголовок вашей странички </h2>	<H2> </h2> Элемент заголовка
<P> Здесь расположен текст первого абзаца вашей странички</p>	<P> </p> Элемент абзаца.
<P> Здесь расположен текст второго абзаца вашей странички</p>	
</body>	Конец содержимого Web-страницы.
</html>	Конец HTML-документа.

Теги можно записывать как строчными, так и заглавными латинскими символами.

Правила синтаксиса

1. Взаимное расположение элементов HTML, HEAD, TITLE, BODY должно быть стандартным на любой странице.

```
<HTML>
<HEAD>
<TITLE>.....</title>
</head>
<BODY>
.....
</body>
</html>
```

2. Необходимо всегда использовать конечные теги (не забывать </p>, </h1>, </table> и др.).

3. Не нарушать правила вложения тегов. Правильно: <H1>Заголовок крупный <H2> Заголовок поменьше </h2> </h1>. Не правильно: <H1>Заголовок крупный <H2> Заголовок поменьше </h1> </h2>

4. Любая полезная информация должна находится между начальным и конечным тегами, указывающими ее формат.

5. Все атрибуты располагаются в начальном теге.

Форматирование текста

Элемент	Тег	Атрибуты	Пример
Абзац	<P> </p>	<P align="left"> </p> — выравнивание текста по левому краю экрана. <P align="center> </p> — выравнивание текста по центру экрана. <P align="right"> </p> — выравнивание текста по правому краю экрана. <P align="justify"> </p> — выравнивание текста по ширине страницы.	<P align="center"> Текст этого абзаца выровнен по центру экрана </p>
Принудительный переход на новую строку	 		Уронили мишку на пол. Оторвали мишке лапу. Все равно его не брошу, Потому что он хороший.
Выделение текста полужирным шрифтом	<B> 		Этот текст имеет обычное начертание, <B> а этот выделен полужирным шрифтом .
Выделение текста курсивом	<I> </i>		Этот текст имеет обычное начертание, <I> а этот выделен курсивом</i>.
Определение типа,	<FONT> 	<FONT size=3>	<FONT size=1> Это

размера и цвета шрифта.		 - абсолютный размер шрифта (возможные значения от 1 до 7). <FONT color="blue"> — цвет шрифта <FONT face="arial"> — определение названия шрифта. <FONT size=3 color="blue" face="arial"> — все атрибуты могут быть использованы совместно внутри данного тега.	шрифт 1 <FONT size=2> Это шрифт 2 <FONT size=3> Это шрифт 3 <FONT size=4> Это шрифт 4 <FONT size=5> Это шрифт 5 <FONT size=6> Это шрифт 6 <FONT size=7> Это шрифт 7 <FONT color="blue"> Это шрифт синего цвета <FONT face="arial" size=3 color="blue" > Это шрифт arial размером 3, цвет синий.
Цитата	<BLOCKQUOTE > </blockquote>		Это обычный текст абзаца. <BLOCKQUOTE> А это текст цитаты. </blockquote> А это снова обычный текст.
Маркированный список	<UL> <LI> <LI> <LI> 	Имеет атрибут type, задающий вид маркера в списке: “disc”, “square”, “circle” (по умолчанию “disc”). А каждый элемент списка определяется тегом <LI>	<UL> <LI> Первый пункт списка; <LI> Второй пункт списка; <LI> Третий пункт списка.
Нумерованный список	<OL> <LI> <LI> <LI> 	Имеет атрибут type, задающий тип нумерации: арабские цифры, римские цифры, буквы (по	<OL> <LI> Первый пункт списка; <LI> Второй пункт списка;

		умолчанию арабские цифры). Каждый элемент списка определяется тегом <LI>	<LI> Третий пункт списка.

Управление цветом

Кодирование цвета используется для раскрашивания шрифтов, горизонтальных линий, фона и других элементов страницы. Цвета обозначаются английскими названиями или числовыми шестнадцатеричными кодами.

Стандартные цвета

Аквамарин		aqua	#00FFFF
Белый		white	#FFFFFF
Желтый		yellow	#FFFF00
Зеленый		green	#008000
Золотистый		gold	#FFD700
Индиго		indigo	#4B0080
Каштановый		maroon	#800000
Красный		red	#FF0000
Оливковый		oliv	#808000
Пурпурный		purple	#800080
Светло-зеленый		lime	#00FF00
Серебристый		silver	#C0C0C0
Серый		gray	#808080
Сизый		teal	#008080
Синий		blue	#0000FF
Ультрамарин		navy	#000080
Фиолетовый		violet	#EE80EE
Фуксиновый		fuchsia	#FF00FF
Черный		black	#000000

Градации красного

Код	Цвет	Код	Цвет
#010000		#800000	

#100000		#900000	
#200000		#A00000	
#300000		#B00000	
#400000		#C00000	
#500000		#D00000	
#600000		#E00000	
#700000		#FF0000	

Градации зеленого

Код	Цвет	Код	Цвет
#000100		#008000	
#001000		#009000	
#002000		#00A000	
#003000		#00B000	
#004000		#00C000	
#005000		#00D000	
#006000		#00E000	
#007000		#00FF00	

Градации синего

Код	Цвет	Код	Цвет
#000001		#000080	
#000010		#000090	
#000020		#0000A0	
#000030		#0000B0	
#000040		#0000C0	
#000050		#0000D0	
#000060		#0000E0	
#000070		#0000FF	

Градации оранжевого

Код	Цвет
#FFB000	1
#FFA800	2

#FFA000	3
#FF9800	4
#FF9000	5
#FF8800	6
#FF8000	7
#FF7800	8
#FF7000	9
#FF6800	10
#FF6000	11
#FF5800	12

Компьютерная радуга:

К

О

Ж

З

Г

С

Ф

Использование цвета при оформлении страницы

Цвет шрифта можно задать с помощью атрибута color в теге <FONT>, например:

```
<FONT color="FF5800"> Это цветной текст 1 </font>
<FONT color="blue"> Это цветной текст 2 </font>
```

Чтобы задать цвет фона страницы используется атрибут color внутри тега <BODY>, например:

```
<BODY color=" red">
```

Тэги

Тэги списков

Существует три основных вида списков в HTML-документе:

- пронуменрованный;
- непронуменрованный;
- список описаний;

Вы можете создавать вложенные списки, используя различные тэги списков или повторяя одни внутри других. Для этого просто необходимо разместить одну пару тэгов (стартовый и завершающий) внутри другой. Будут

ли элементы вложенного списка иметь те же маркеры, обозначающие элемент списка — зависит от браузера. Более подробно смотри в разделе "Вложенные списки".

Пронумерованные списки

В пронумерованном списке браузер автоматически вставляет номера элементов по порядку. Это означает, что если Вы удалите один или несколько элементов пронумерованного списка, то остальные номера автоматически будут пересчитаны.

Пронумерованный список начинается стартовым тэгом <OL> и завершается тэгом </OL>. Каждый элемент списка начинается с тэга <LI> и заканчивается тегом </LI>. Например:

```
<OL>
  <LI>Программирование
  <LI>Алгоритмизация
  <LI>Проектирование
</OL>
```

Тэг <OL> может иметь параметры:

<OL TYPE=A|a|I|i|1 START=n> где: TYPE - вид счетчика:

- A — большие латинские буквы (A,B,C...)
- a — маленькие латинские буквы (a,b,c...)
- I — большие римские цифры (I,II,III...)
- i — маленькие римские цифры (i,ii,iii...)
- 1 — обычные цифры (1,2,3...)

START=n - число, с которого начинается отсчет. Например:

```
<OL TYPE=I START=15>
  <LI> Программирование
  <LI> Алгоритмизация
  <LI> Проектирование
</OL>
```

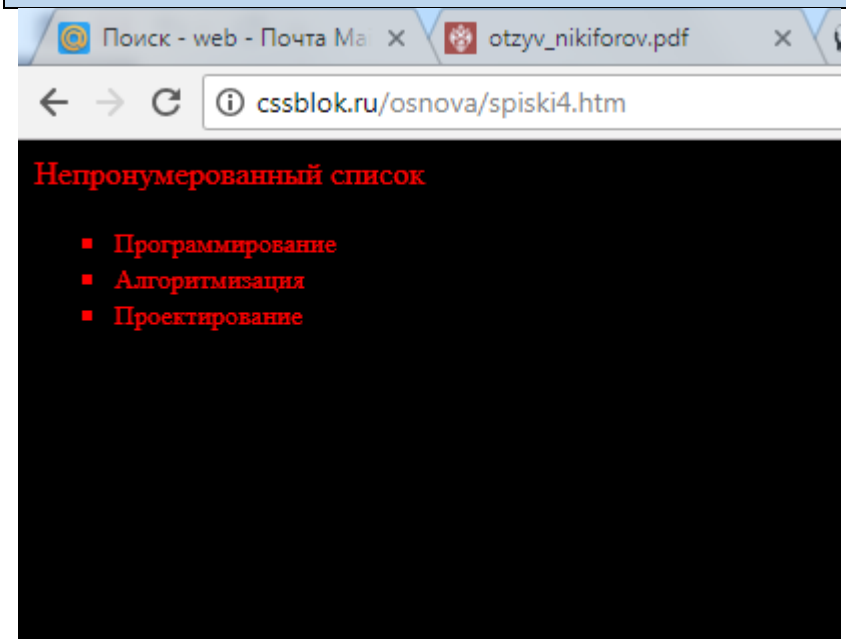
Непронумерованные списки.

Для непронумерованных списков браузер обычно использует маркеры для пометки элемента списка. Вид маркера, как правило, настраивает пользователь браузера.

Список начинается стартовым тэгом <UL> и завершается тэгом </UL>. Каждый элемент списка начинается с тэга <LI>. Например:

```
<UL>
  <LI>Программирование
```

```
    <LI>Алгоритмизация
    <LI>Проектирование
</UL>
```



Тэг <UL> может иметь параметр: TYPE=disc|circle|square>. Тип тэга <UL> определяет внешний вид маркера — по умолчанию (disc), круглый (circle) или квадратный (square). Например:

```
<UL TYPE=square>
    <LI>Программирование
    <LI>Алгоритмизация
    <LI>Проектирование
</UL>
```

Вложенные списки

Дадим пример вложенных списков:

```
<HTML>
    <HEAD>
        <TITLE> Список сотрудников </TITLE>
    </HEAD>
    <BODY>
        <H2> Список сотрудников нашей фирмы </H2>
        <H3> Составлено : 30 июля 2006 года </H3>
<p>Данный список содержит фамилии, имена и отчества всех сотрудников нашей компании. <P>
<p>Список может быть использован только в служебных целях. <P>
        <OL>
            <LI> Дирекция
                <UL>
                    <LI> Иванов И.И.
                    <LI> Петров К.В.
```

```
            </UL>
          </LI>
          <LI> Отдел маркетинга
            <UL>
              <LI> Варшавская Е.Л.
              <LI> Самсонов Д.М.
            </UL>
          </LI>
        </OL>
      </BODY>
    </HTML>
```

Элемент списка <LI>

Тэг <UL> может иметь параметры: TYPE=disc|circle|squade> или <OL TYPE=A|a|I|i|1 VALUE=n>, в зависимости от того, в списке какого вида находится данный элемент.

Таким образом атрибут TYPE определяет вид маркера (см. <UL>) или счетчика (см.OL), а VALUE=n — значение для элемента пронумерованного списка (его номер). Все дальнейшие номера элементов списка будут отсчитываться от этого номера, а каждый элемент списка задаётся тегом <LI>. Например:

```
<OL TYPE=I START=15>
  <LI> Программирование
  <LI TYPE=i VALUE=25> Алгоритмизация
  <LI> Проектирование
</OL>
```

XV. Программирование
XVI. Алгоритмизация
XVII. Проектирование

Список определений

Список определений начинается с тэга <DL> и завершается тэгом </DL>. Данный список служит для создание списков типа "термин"-"описание". Каждый термин начинается тэгом <DT>, а описание — тэгом <DD>. Например:

```
<DL>
  <DT> <B> Отдел маркетинга </B>
    <DD> Данный отдел занимается продвижением продуктов и услуг
  <DT> <B> Финансовый отдел </B>
    <DD> Данный отдел занимается всеми финансовыми операциями
  <DT> <B> Отдел кадров </B>
```

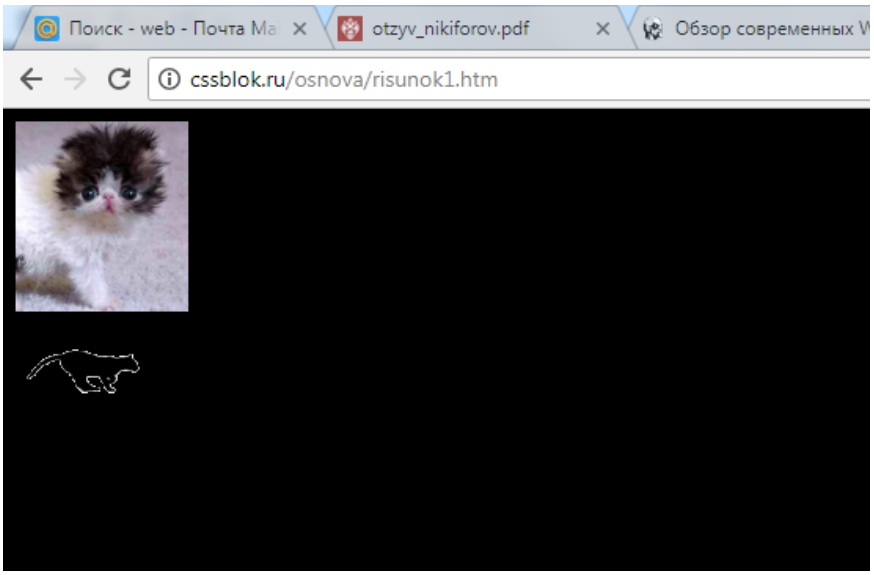
<DD> Данный отдел занимается учетом и набором новых сотрудников фирмы, распределением отпусков, отслеживанием больничных листов и т.д.
</DL>

Рисунки на WEB-страничке

<IMG> — элемент для создания ссылки на графический файл (image). Он не содержит конечного тега — вся необходимая информация задается при помощи атрибутов. Этот элемент является универсальным: с его помощью можно использовать изображения в гиперссылках, вставлять картинки в таблицы, просто размещать рисунки на Web-странице, решать задачи дизайна и т.д.

Необходимым атрибутом является src — указатель на файл графики:

src="Ссылка на файл". Например: <IMG src="cat5-web.jpg> — обычный рисунок, <IMG src="1c5.gif"> — анимированный рисунок.



Очень полезным и обязательным атрибутом является атрибут alt. Он позволяет выводить текст в тех местах, где должны располагаться рисунки. Страница может загружаться достаточно долго, и пока графические файлы на подходе, пользователь должен видеть, какие изображения он сможет получить.

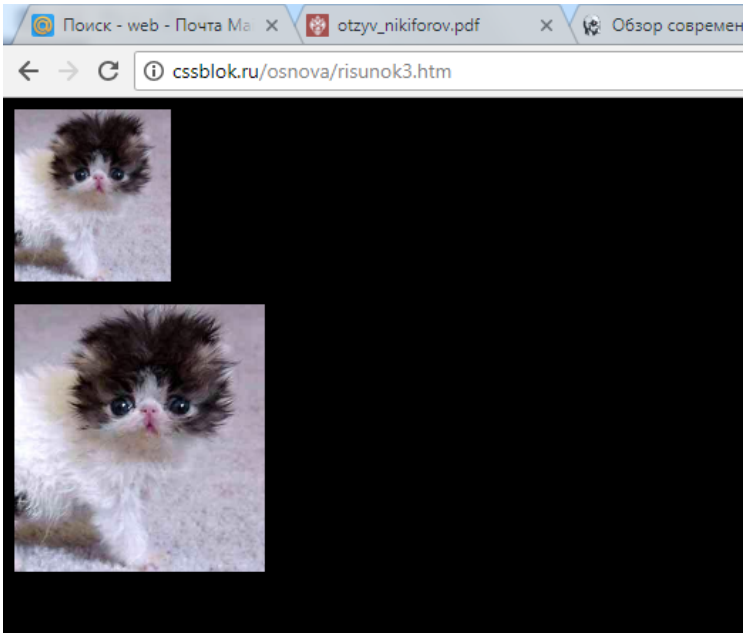
Например: <IMG src="cat5-web.jpg alt="Фотография маленького котенка">. <IMG src="1c5.gif" alt="Большая черная кошка, которая гуляет сама по себе">

Высоту и ширину области, в которой демонстрируется рисунок, задают при помощь атрибутов height — высота и width — ширина.

Например: <IMG src="bos2.gif" width="76" height="121">

<img src="bos2.gif" width="176">. Обратите внимание, что во втором случае изменен размер рисунка (мы изменили ширину, высота будет изменена

автоматически). При этом происходит потеря качества изображения, поэтому желательно задавать эти атрибуты в соответствии с реальными размерами рисунка.



Атрибут border задает размер рамки вокруг объекта, например, border="4" ширина рамки задается в пикселях и в нашем примере равна 4.

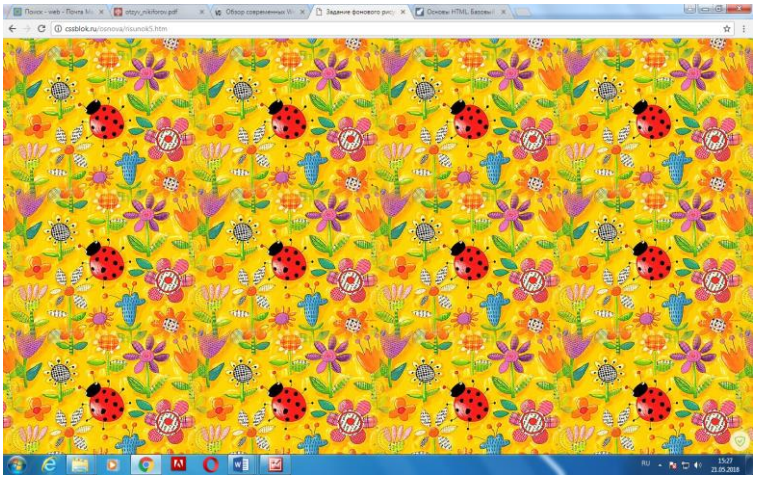
<IMG src="cat5-web.jpg" alt="Фото котёнка в рамке" width="176" style="border: 4px solid #9A2EEB;">. Размер рамки и её цвет зададим, например, с помощью стилей.

Полностью тег IMG должен выглядеть следующим образом:

<IMG src="bos2.gif" width="76" height="121" alt="Портрет маленького джентельмена">

Если вы хотите использовать рисунок в качестве обоев странички, то в теге <BODY> используем атрибут background с указанием адреса рисунка обоев.

Например: <BODY background="wood.jpg">



Гиперссылки

<A> — один из самых важных элементов языка, обеспечивающий создание гиперссылок. Чаще всего используется такой шаблон:

Произвольный текст <A href="адрес ссылки"> Текст для щелчка

Например, так выглядит гиперссылка в тексте: Если вы хотите вернуться к материалам урока "Гиперссылки", то вам сюда , при этом надо предварительно поставить имя закладки перед тем местом в документе, на которое происходит переход. Закладка в документе задаётся в следующем виде: . В наше случае это имя zakl2.

А так задается шаблон в том случае, когда видимая часть гиперссылки представляет собой рисунок:

<A href="Адрес ссылки"> <IMG src="Ссылка на рисунок">

Например: Чтобы вернуться к уроку "Гиперссылки" нажмите на кнопку <A href="index.html#zakl2"> <IMG src="knopka.gif" width="30" height="20">

Внутри тега <BODY> используются атрибуты link — задает цвет ссылок на всей Web-странице, vlink — задает цвет посещенных ссылок, alink — задает цвет активных ссылок (цвет появляется при нажатии мыши).

Пример: <BODY link="0000FF" vlink="CC0066" alink="FF0000">

По умолчанию используется ссылка на файлы текущей папки (той, где расположен файл Web-страницы). В этом случае просто указывается имя файла, например: page2.htm, photo35.gif и т.д.

Часто используются относительные ссылки на папки: это позволяет легко менять местоположение комплекса страниц на диске. Если в текущей папке есть другая, в которой размещены необходимые файлы, ссылка строиться по такому шаблону:

href="./Папка/файл.тип". Здесь на структуру вложенных папок указывает точка перед наклонной чертой. Если необходимо указать папку, которая находится на том же уровне вложенности, что и текущая, добавляют еще одну точку: href="../Папка/файл.тип".

Если ссылка указывает на какой-либо Web-ресурс, то она выглядит следующим образом, например: href="http://www.netscape.com".

Когда гиперссылка используется для указания адреса электронной почты, ее выбор обеспечивает не переход к новому документу, а запуск почтовой программы на компьютере для отправки сообщения указанному адресату. Обычно такую ссылку размещают в конце страницы для обеспечения связи с Web-мастером или автором страницы, например:

<A href="mailto:goncharov@online.ru>Алексей Гончаров

Таблицы

Таблицы являются очень удобным средством форматирования данных на Web-станице. Они позволяют решать чисто дизайнерские задачи: выравнивать части страницы друг относительно друга, размещать рядом рисунки и текст, управлять цветовым оформлением и т.д.

При создании таблиц используется принцип вложения: внутри основного элемента таблицы <TABLE> создается ряд элементов, определяющих строки <TR>, а внутри этих элементов размещаются элеметы для описания каждой ячейки в с строке <TD>.

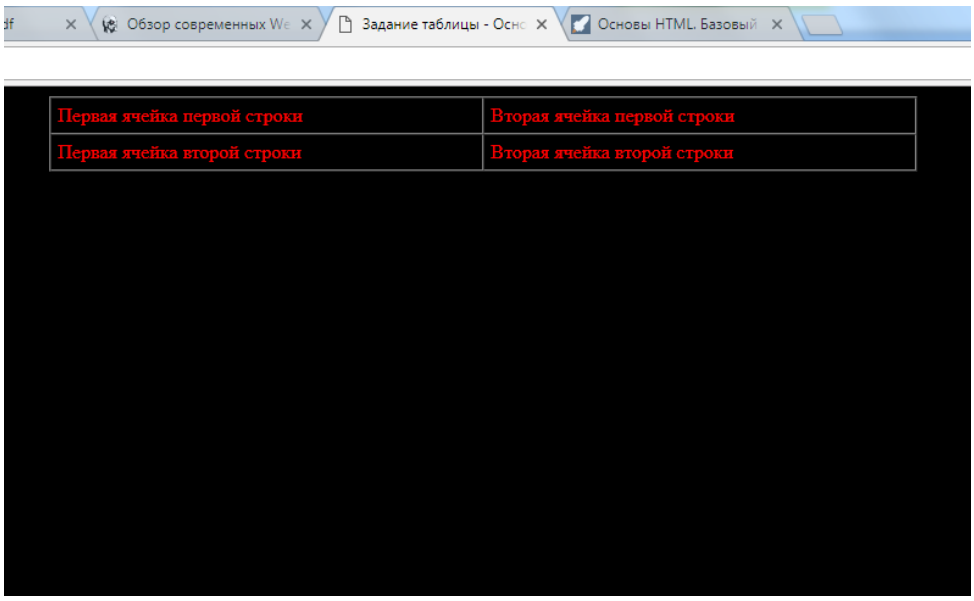
<TABLE> </table> — внешний элемент таблицы.

<TR> </tr> — элемент, задающий строку таблицы. Конечный тег можно не использовать, т.к. строка заканчивается там, где начинается следующая строка.

<TD> </td> — элемент, задающий ячеку таблицы. Конечный тег также можно не использовать.

Для примера опишем таблицу, которая будет состоять из двух строк и двух столбцов:

<TABLE>	Начало таблицы
<TR>	Начало первой строки
<TD> Первая ячейка первой строки</td>	Первая ячейка первой строки
<TD> Вторая ячейка первой строки</td>	Вторая ячейка первой строки
</tr>	Конец первой строки
<TR>	Начало второй строки
<TD>Первая ячейка второй строки</td>	Первая ячейка второй строки
<TD>Вторая ячейка второй строки</td>	Вторая ячейка второй строки
</tr>	Конец второй строки
</table>	Конец таблицы



Ширину таблицы можно задавать точно в пикселах или в процентном отношении к ширине страницы в окне браузера.

Например, если нам нужно создать таблицу определенного размера, то мы укажем:

```
<TABLE width="500">
  <TR>
    <TD> Ширина этой таблицы 500 пикселей и она состоит из одной строки и одного столбца.</td>
  </tr>
</table>
```

Если мы хотим получить таблицу на всю ширину экрана, не заботясь при этом, какое разрешение монитора (800x600, 1024 x 768, 1280 x 1024) у того, кто будет просматривать нашу Web-страницу, то мы зададим ширину страницы как 100%.

```
<TABLE width="100%">
  <TR>
    <TD> Ширина этой таблицы 100%.</td>
    <TD> и она состоит из одной строки и двух столбцов </td>
  </tr>
</table>
```

Для всей таблицы может быть задан цвет фона:

bgcolor="Цвет" или bgcolor="#RRGGBB", например:

```
 <TABLE width="100%" bgcolor="#00CC99">
  <TR>
    <TD> Ширина этой таблицы 50%.</td>
  </tr>
  <TR>
    <TD> и она состоит из двух строк и одного столбца </td>
```

```
    </tr>
  </table>
```

Можно задавать отдельно цвет ячеек таблицы.

<table width="600" border="1" cellspacing="0" cellpadding="5" align="center">

```
    <tr>
      <td bgcolor="#00FFFF"></td>
      <td bgcolor="#CCFF00"></td>
      <td bgcolor="#FF6633"></td>
    </tr>
    <tr>
      <td bgcolor="#0000FF"></td>
      <td bgcolor="#33FF66"></td>
      <td bgcolor="#FF00FF"></td>
    </tr>
    <tr>
      <td bgcolor="#CCCCCC"></td>
      <td bgcolor="#9933FF"></td>
      <td bgcolor="#FFFFCC"></td>
    </tr>
  </table>
```

Шириной боковой грани управляет атрибут border. Можно задать ширину боковой грани таблицы в пикселах.

```
  <TABLE width="100%" bgcolor="#00CC99" border="3" >
    <TR>
      <TD> </td>
      <TD> Ширина этой таблицы 300 пикселей</td>
      <TD> </td>
    </tr>
    <TR>
      <TD> и она состоит из двух строк и трех столбцов</td>
      <TD> </td>
      <TD></td>
    </tr>
  </table>
```

Можно сделать грани таблицы невидимыми, для этого ширину бордюра таблицы нужно задать равной 0:

```
  <TABLE width="100%" bgcolor="#00CC99" border="0" >
    <TR>
      <TD> </td>
      <TD> Ширина этой таблицы 300 пикселов</td>
      <TD> </td>
```

```
    </tr>
    <TR>
      <TD> и она состоит из двух строк и трех столбцов</td>
      <TD> </td>
      <TD></td>
    </tr>
  </table>
```

Существует набор атрибутов, предназначенных для выравнивания данных в ячейках таблиц. Атрибут align позволяет выравнивать данные в ячейках по горизонтали. Он принимает следующие значения:

left — выравнивание влево;

right — выравнивание вправо;

center — центрирование.

Атрибут valign позволяет выравнивать текст по вертикали. Значения могут быть такие:

top — выравнивание по верхнему краю ячейки

center — выравнивание по центру

baseline — выравнивание по первой строке.

```
<table width="100%" border="1" cellspacing="0" cellpadding="5" align="center">
  <tr> <td width="257">Выравнивание по горизонтали</td>
    <td width="233" align="center"> По центру </td>
    <td width="217" align="left">По левому краю </td>
    <td width="246" align="right"> По правому краю </td>
  </tr>
  <tr>
    <td width="257" height="112">Выравнивание по вертикали</td>
    <td width="233" height="112" valign="top">По верхнему краю</td>
    <td width="217" height="112" valign="middle">По центру</td>
    <td width="246" height="112" valign="baseline">По нижнему краю</td>
  </tr>
</table>
```

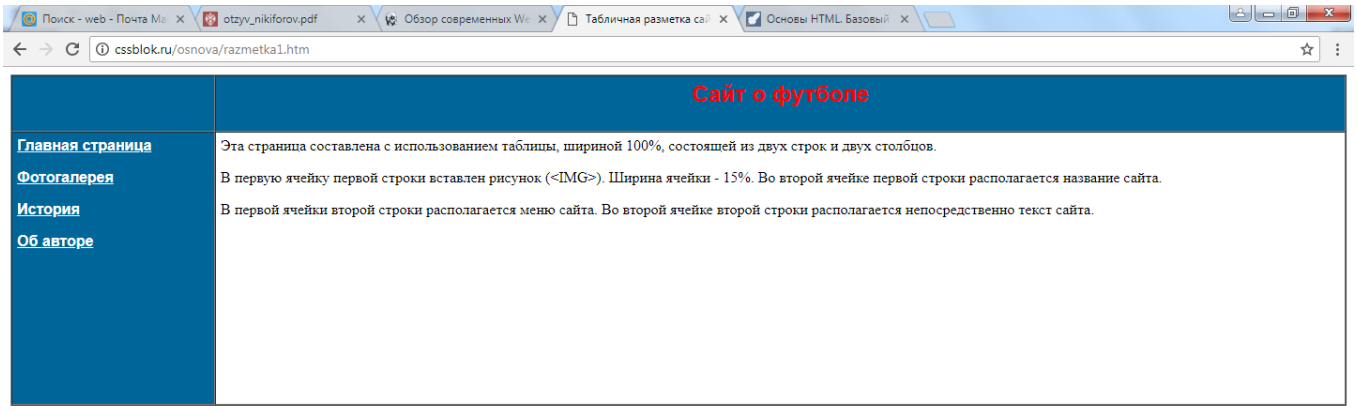
Разметка Web-страницы с использованием таблицы

Разметку Web-странцы удобно производить с использованием таблицы. Возможны различные варианты разметки. Рассмотрим некоторые из них.

1 вариант

Разметка страницы производится с использованием таблицы шириной на весь экран, независимо от того, каково разрешение экрана (width=100%). В данном случае удобно создать таблицу, состоящую из двух строк и двух

столбцов. Верхняя строка будет отведена под заголовок странички, левый столбец будет отведен под меню Web-сайта.

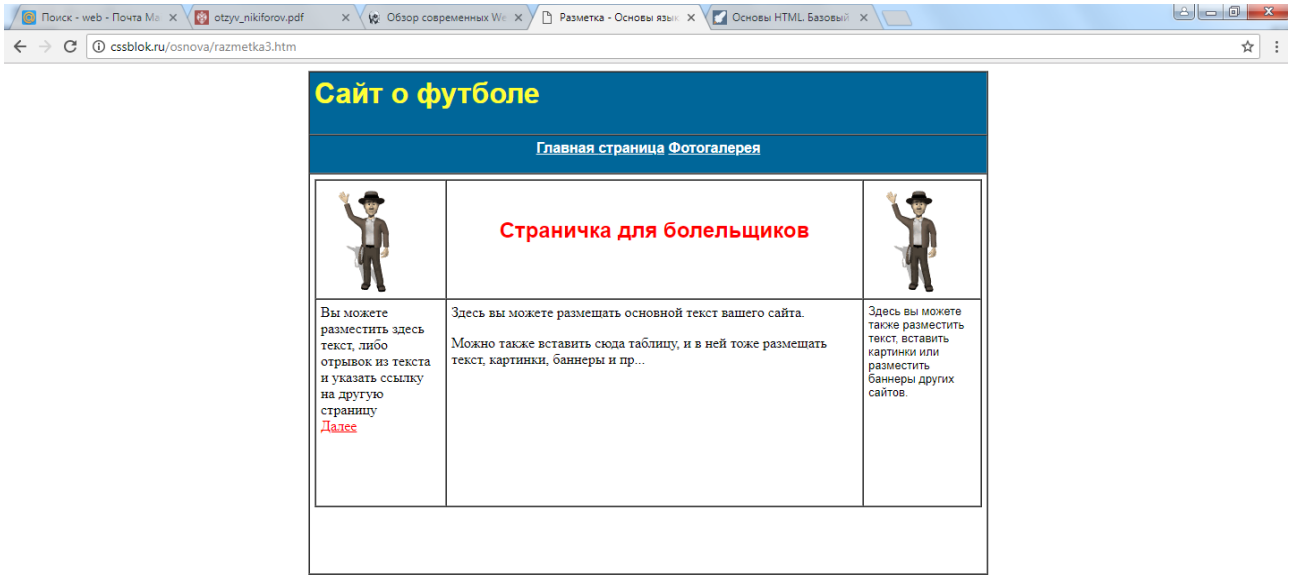


Вернуться к вариантам разметки

2 вариант

Разметка страницы производится с использованием таблицы шириной 760 пикселей, выравненной по центру экрана. В данном случае удобно создать таблицу, состоящую из трех строк и одного столбца. Верхняя строка будет отведена под заголовок странички, вторая строка будет отведена под меню Web-сайта, а третья строка отведена непосредственно под содержание сайта.

Если необходимо разместить внутри текста странички какие-либо иллюстрации, фотографии и пр., то в этом случае также могут использоваться таблицы. В приведенном ниже примере во вторую ячейку второй строки вставлена таблица, состоящая из из двух строк и трех столбцов. В первую и в третью ячейки первой строки вставлены рисунки, а во вторую ячейку первой строки — название страницы. Во все ячейки второй строки введен текст странички.



Вернуться к вариантам разметки

Фреймы

Используя фреймы, позволяющие разбивать Web-страницы на множественные скроллируемые подокна, Вы можете значительно улучшить внешний вид и функциональность информационных систем и Web-приложений. Каждое подокно, или фрейм, может иметь следующие свойства:

- Каждый фрейм имеет свой URL, что позволяет загружать его независимо от других фреймов.
- Каждый фрейм имеет собственое имя (параметр name), позволяющее переходить к нему из другого фрейма.
- Размер фрейма может быть изменен пользователем прямо на экране при помощи мыши (если это не запрешено указанием специального параметра).

Данные свойства фреймов позволяют создавать продвинутые интерфейсные решения, такие как:

- Размещение статической информации, которую автор считает необходимым постоянно показывать пользователю, в одном статическом фрейме. Это может быть графический логотип фирмы, copyright, набор управляющих кнопок.
- Помещение в статическом фрейме оглавления всех или части Web-документов, содержащихся на Web-сервере, что позволяет пользователю быстро находить интересующую его информацию.
- Создавать окна результатов запросов, когда в одном фрейме находится собственно запрос, а в другом результаты запроса.
- Создавать формы типа "мастер-деталь" для Web-приложений, обслуживающих базы данных.

Синтаксис фреймов

Формат документа, использующего фреймы, внешне очень напоминает формат обычного документа, только вместо тэга BODY используется контейнер FRAMESET, содержащий описание внутренних HTML-документов, т. е. собственно информацию, размещаемую во фреймах.

```
<HTML>
   <HEAD>...</HEAD>
   <FRAMESET>...</FRAMESET>
   </HTML>
```

Однако, фрейм-документ является специфичным видом HTML-документа, поскольку не содержит элемента BODY и какой-либо информационной нагрузки соответственно. Он описывает только фреймы, которые будут

содержать информацию (кроме случая двойного документа, который мы рассмотрим позже).

Представим общий синтаксис фреймов:

```
<FRAMESET COLS="value" | ROWS="value">
<FRAME SRC="url1">
<FRAME...>
...
</FRAMESET>
```

Общий контэйнер FRAMESET описывает все фреймы, на которые делится экран. Вы можете разделить экран на несколько вертикальных или несколько горизонтальных фреймов. Тэг FRAME описывает каждый фрейм в отдельности. Рассмотрим более детально каждый компонент.

FRAMESET

<FRAMESET [COLS="value" | ROWS="value"]>

Тэг <FRAMESET> имеет завершающий тэг </FRAMESET>. Все, что может находиться между этими двумя тэгами, это тэг <FRAME>, вложенные тэги <FRAMESET> и </FRAMESET>, а также контейнер из тэгов <NOFRAME> и </NOFRAME>, который позволяет строить двойные документы для броузеров, поддерживающих фреймы и не поддерживающих фреймы.

Данный тэг имеет два взаимоисключающих параметра: ROWS и COLS.

ROWS="список-определений-горизонтальных-подокон". Данный атрибут содержит описания некоторого количества подокон, разделенных запятыми. Каждое описание представляет собой числовое значение размера подокна в пикселах, процентах от всего размера окна или связанное масштабное значение. Количество подокон определяется количеством значений в списке. Общая сумма высот подокон должна составлять высоту всего окна (в любых измеряемых величинах). Отсутствие атрибута ROWS определяет один фрейм, величиной во все окно броузера.

Синтаксис используемых видов описания величин подокон:

value — простое числовое значение определяет фиксированную высоту подокна в пикселах. Это далеко не самый лучший способ описания высоты подокна, поскольку различные браузеры имеют различный размер рабочего поля, не говоря уже о различных экранных разрешениях у пользователя. Если Вы, все же, используете данный способ описания размера, то настоятельно рекомендуется сочетать его с каким-либо другим, чтобы в результате Вы точно получили 100%-ное заполнение окна броузера вашего пользователя.

value% — значение величины подокна в процентах от 1 до 100. Если общая сумма процентов описываемых подокон превышает 100, то размеры всех

фреймов пропорционально уменьшаются до суммы 100%. Если, соответственно, сумма меньше 100, то размеры пропорционально учеличиваются.

value* — вообще говоря, значение value в данном описании является необязательным. Символ "*" указывает на то, что все оставшееся место будет принадлежать данному фрейму. Если указывается два или более фрейма с описанием "*" (например "*,*"), то оставшееся пространство делится поровну между этими фреймами. Если перед звездочкой стоит цифра, то она указывает пропорцию для данного фрейма (во сколько раз одно будет больше аналогично описанного чистой звездочкой). Например, описание "3*,*,*", говорит, что будет создано три фрейма с размерами 3/5 свободного пространства для первого фрейма и по 1/5 для двух других.

COLS="список-определений-горизонтальных-подокон". То же самое, что и ROWS, но делит окно по вертикали, а не по горизонтали. Внимание! Совместное использование данных параметров может привести к непредствазуемым результатам. Например, строка: <FRAMESET ROWS="50%,50%" COLS "50%,50%"> может привести к ошибочной ситуации.

Примеры:

<FRAMESET COLS="50,*,50"> — описывает три фрейма, два по 50 точек справа и слева, и один внутри этих полосок.

<FRAMESET ROWS="20%,3*,*"> — описывает три фрейма, первый из которых занимает 20% площади сверху экрана, второй 3/4 оставшегося от первого фрейма места (т.е. 60% всей площади окна), а последний 1/4 (т.е. 20% всей площади окна).

<FRAMESET ROWS="*,60%,*"> — аналогично предыдущему примеру.
Тэги <FRAMESET> могут быть вложенными, т.е. например:

```
<FRAMESET ROWS="50%,50%">
<FRAMESET COLS="*,*"
</FRAMESET>
</FRAMESET>
```

Результат данного примера мы рассмотрим позже.

FRAME

<FRAME SRC="url" [NAME="frame_name"] [MARGINWIDTH="nw"] [MARGINHEIGHT="nh"]

[SCROLLING=yes|no|auto] [NORESIZE]>. Данный тэг определяет фрейм внутри контейнера FRAMESET.

SRC="url" описывает URL документа, который будет отображен внутри данного фрейма. Если он отсутствует, то будет отображен пустой фрейм.

NAME="frame_name". Данный атрибут описывает имя фрейма. Имя фрейма может быть использовано для определения действия с данным фреймом из другого HTML-документа или фрейма (как правило, из соседнего фрейма этого же документа). Имя обязательно должно начинаться с символа. Содержимое поименованных фреймов может быть задействовано из других документов при помощи специального атрибута TARGET, описываемого ниже.

MARGINWIDTH="value". Этот атрибут может быть использован, если автор документа хочет указать величину разделительных полос между фреймами сбоку. Значение value указывается в пикселах и не может быть меньше единицы. По умолчанию данное значение зависит от реализации поддержки фреймов используемым клиентом браузером.

MARGINHEIGHT="value". То же самое, что и MARGINWIDTH, но для верхних и нижних величин разделительных полос.

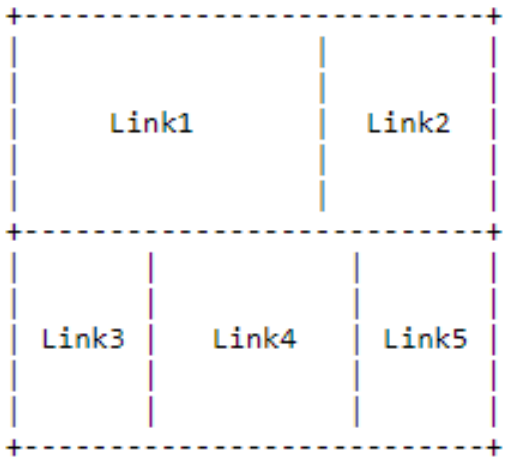
SCROLLING="yes | no | auto". Этот атрибут позволяет задавать наличие полос прокрутки у фрейма. Параметр yes указывает, что полосы прокрутки будут в любом случае присутствовать у фрейма, параметр no наоборот, что полос прокрутки не будет. Auto определяет наличие полос прокрутки только при их необходимости (значение по умолчанию).

NORESIZE. Данный атрибут позволяет создавать фреймы без возможности изменения размеров. По умолчанию, размер фрейма можно изменить при помощи мыши так же просто, как и размер окна Windows. NORESIZE отменяет данную возможность. Если у одного фрейма установлен атрибут NORESIZE, то у соседних фреймов тоже не может быть изменен размер со стороны данного.

NOFRAMES данный атрибут используется в случае, если Вы создаёте документ, который может просматриваться как браузерами, поддерживающими фреймы, так и браузерами, их не поддерживающими. Данный тэг помещяется внутри контейнера FRAMESET, а все, что находится внутри тэгов <NOFRAMES> и </NOFRAMES> игнорируется браузерами, поддерживающими фреймы.

Примеры

Рассмотрим реализацию фреймов для подобного разбиения окна:



<FRAMESET ROWS="*,*" <NOFRAMES> <H1>Ваша версия WEB-браузера не поддерживает фреймы!</H1> </NOFRAMES>

```
<FRAMESET COLS="65%,35%">
  <FRAME SRC="link1.html">
  <FRAME SRC="link2.html">
</FRAMESET>
<FRAMESET COLS="*,40%,*">
  <FRAME SRC="link3.html">
  <FRAME SRC="link4.html">
  <FRAME SRC="link5.html">
</FRAMESET>
</FRAMESET>
```

Планирование фреймов и взаимодействия между фреймами

С появлением фреймов сразу возникает вопрос: "А как сделать так, чтобы нажимая на ссылку в одном фрейме инициировать появление информации в другом?"

Ответом на данный вопрос является планирование взаимодействия фреймов (далее — планирование). Каждый фрейм может иметь собственное имя, определяемое параметром NAME при описании данного фрейма. Существует, также, специальный атрибут — TARGET, позволяющий определять, к какому фрейму относится та или иная операция. Формат данного атрибута следующий:

TARGET="windows_name"

Данный атрибут может встречаться внутри различных тэгов: TARGET в тэге <A> — это самое прямое использование TARGET. Обычно, при активизации пользователем ссылки соответствующий документ появляется в том же окне (или фрейме), что и исходный, в котором была на него сделана ссылка. Добавление атрибута TARGET позволяет произвести вывод

документа в другой фрейм. Например: <A HREF="mydoc.html" TARGET="Frame1"> Переход в фрейм № 1 </A>

Размещение TARGET в тэге BASE позволит Вам не указывать при описании каждой ссылки фрейм — приемник документов, вызываемых по ссылкам. Это очень удобно, если в одном фрейме у Вас находится меню, а в другой — выводится информация. Например:

Документ № 1.

```
<FRAMESET ROWS="20,*">
  <FRAME SRC="doc2.htm" NAME="Frame1">
  <FRAME SRC="doc3.htm" NAME="Frame2">
</FRAMESET>
Документ № 2 (doc2.htm).
<HTML>
  <HEAD>
  <BASE TARGET="Frame2">
  </HEAD>
<BODY>
    <A HREF="url1"> Первая часть</A>
    <A HREF="url2"> Вторая часть</A>
</BODY>
</HTML>
```

Таже можно включать параметр TARGET в описание ссылки при создании карты изображения в теге <AREA>. Например:

<AREA SHAPE="circle" COORDS="100,100,50" HREF="http://www.softexpress.com" TARGET="Frame1">

То же относится и к определению формы. В данном случае, после обработки переданных параметров формы результирующий документ появится в указанном фрейме.

<FORM ACTION="url" TARGET="window_name">. Внимание! Имя окна (фрейма) в параметре TARGET должно начинаться с латинской буквы или цифры. Также необходимо помнить, что существуют зарезервированные имена для разрешения специальных ситуаций.

Зарезервированные имена фреймов

Зарезервированные имена фреймов служат для разрешения специальных ситуаций. Все они начинаются со знака подчёркивания. Любые другие имена фреймов, начинающиеся с подчеркивания будут игнорироваться браузером.

TARGET="_blank" — данное значение определяет, что документ, полученный по ссылке будет отображаться в новом окне броузера.

TARGET="_self" — данное значение определяет, что документ, полученный по ссылке будет отображаться в том же фрейме, в котором

находится ссылка. Это имя удобно для переопределения окна назначения, указанного ранее в тэге BASE.

TARGET="_parent" — данное значение определяет, что документ, полученный по ссылке будет отображаться в родительском окне, вне зависимости от параметров FRAMESET. Если родительского окна нет, то данное имя аналогично "_self".

TARGET="_top" — Данное значение определяет, что документ, полученный по ссылке будет отображаться на всей поверхности окна, вне зависимости от наличия фреймов. Использование данного параметра удобно в случае вложенных фреймов.

Формы

Некоторые WWW-браузеры позволяют пользователю, заполнив специальную форму, возвращающую полученное значение, выполнять некоторые действия на Вашем WWW-сервере. Когда форма интерпретируется WEB-браузером, создается специальные экранные элементы GUI, такие, как поля ввода, checkboxes, radiobuttons, выпадающие меню, скроллируемые списки, кнопки и т.д. Когда пользователь заполняет форму и нажимает кнопку "Подтверждение" (SUBMIT - специальный тип кнопки, который задается при описании документа), информация, введённая пользователем в форму, посылается HTTP-серверу для обработки и передаче другим программам, работающим под сервером, в соответствии с CGI (Common Gateway Interface) интерфейсом.

Когда Вы описываете форму, каждый элемент ввода данных имеет тэг <INPUT>. Пользователь, помещает данные в элемент формы, и инфоромация размещается в разделе VALUE данного элемента.

Синтаксис

Все формы начинаются тэгом <FORM> и звершаются тэгом </FORM>.

Обший вид:

<FORM METHOD="get|post" ACTION="URL"> Элементы_формы_и_другие_элементы_HTML </FORM>

В зависимости от используемого метода посылки сообщения с данными из формы, Вы можете посылать результаты ввода данных в форму двумя путями:

- GET: Информация из формы добавляется в конец URL, который был указан в описании заголовка формы. Ваша CGI-программа (CGI-скрипт) получает данные из формы в виде параметра переменной среды QUERY_STRING. Использование метода GET не рекомендуется.
- POST: Данный метод передает всю информацию о форме немедленно после обращения к указанному URL. Ваша CGI-программа получает

данные из формы в стандартный поток ввода. Сервер не будет пересылать вам сообщение об окончании пересылки данных в стандартный поток ввода; вместо этого используется переменная среды CONTENT_LENGTH для определения, какое количество данных Вам необходимо считать из стандартного потока ввода. Данный метод рекомендуется к использованию.

ACTION описывает URL, который будет вызываться для обработки формы. Данный URL почти всегда указывает на CGI-программу, обрабатывающую данную форму.

Тэги Формы

Тэг <TEXTAREA> используется для того, чтобы позволить пользователю вводить более одной строки информации (свободный текст). Вот пример использовани тэга <TEXTAREA>:

<TEXTAREA NAME="address" ROWS=10 COLS=50> Бишкек, улица Советская, д.9Б, офис 48 </TEXTAREA>

Атрибуты, используемые внутри тэга <TEXTAREA> описывают внешний вид и имя вводимого значения. Тэг </TEXTAREA> необходим даже тогда, когда поле ввода изначально пустое. Описание атрибутов:

- NAME — имя поля ввода.
- ROWS — высота поля ввода в символах.
- COLS — ширина поля ввода в символах.

Если вы хотите, чтобы в поле ввода по умолчанию выдавался какой-либо текст, то необходимо вставить его внутри тэгов <TEXTAREA> и </TEXTAREA>.

Тэг <INPUT> используется для ввода одной строки текста или одного слова. Атрибуты тэга:

- CHECKED — означает, что CHECKBOX или RADIOBUTTON будет выбран.
- MAXLENGTH — определяет количество символов, которое пользователи могут ввести в поле ввода. При превышении количества допустимых символов браузер реагирует на попытку ввода нового символа звуковым сигналом и не дает его ввести. Не путать с атрибутом SIZE. Если MAXLENGTH больше чем SIZE, то в поле осуществляется скроллинг. По умолчанию значение MAXLENGTH равно бесконечности.

- NAME — имя поля ввода. Данное имя используется как уникальный идентификатор поля, по которому, впоследствии, Вы сможете получить данные, помещенные пользователем в это поле.
- SIZE — определяет визуальный размер поля ввода на экране в символах.
- SRC — URL, указывающий на картинку (используется совместно с атрибутом IMAGE).
- TYPE — определяет тип поля ввода. По умолчанию это простое поле ввода для одной строки текста. Остальные типы должны быть явно указаны: CHECKBOX. Используется для простых логических (BOOLEAN) значений. Значение, ассоциированное с именем данного поля, которое будет передаваться в вызываемую CGI-программу, может принимать значение ON или OFF.
- HIDDEN — поля данного типа не отображаются браузером и не дают пользователю изменять присвоенные данному полю по умолчанию значение. Это поле используется для передачи в CGI-программу статической информации, как то ID прользователя, пароля или другой информации.
- IMAGE — данный тип поля ввода позволяет Вам связывать графический рисунок с именем поля. При нажатии мышью на какую-либо часть рисунка будет немедленно вызвана ассоциированная форме CGI-программа. Значения, присвоенные переменной NAME будут выглядеть так — создается две новых переменных: первая имеет имя, обозначенное в поле NAME с добавлением .x в конце имени. В эту переменную будет помещена X-координата точки в пикселах (считая началом координат левый верхний угол рисунка), на которую указывал курсор мыши в момент нажатия, а переменная с именем, содержащимся в NAME и добавленным .y, будет содержать Y-координату. Все значения атрибута VALUE игнорируются. Само описание картинки осуществляется через атрибут SRC и по синтаксису совпадает с тэгом <IMG>.
- PASSWORD — то же самое, что и атрибут TEXT, но вводимое пользователем значение не отображается браузером на экране.
- RADIO — данный атрибут позволяет вводить одно значение из нескольких альтернатив. Для создания набора альтернатив Вам необходимо создать несколько полей ввода с атрибутом TYPE="RADIO" с разными значениями атрибута VALUE, но с одинаковыми значениями атрибута NAME. В CGI-программу будет передано значение типа NAME=VALUE, причем VALUE примет значение атрибута VALUE

того поля ввода, которое в данный момент будет выбрано (будет активным). При выборе одного из полей ввода типа RADIO все остальные поля данного типа с тем же именем (атрибут NAME) автоматически станут невыбранными на экране.

- RESET — данный тип обозначает кнопку, при нажатии которой все поля формы примут значения, описанные для них по умолчанию.
- SUBMIT — данный тип обозначает кнопку, при нажатии которой будет вызвана CGI-программа (или URL), описанная в заголовке формы. Атрибут VALUE может содержать строку, которая будет высвечена на кнопке.
- TEXT — данный тип поля ввода описывает однострочное поле ввода. Используйте атрибуты MAXLENGTH и SIZE для определения максимальной длинны вводимого значения в символах и размера отображаемого поля ввода на экране (по умолчанию принимается 20 символов).
- VALUE — присваивает полю значение по умолчанию или значение, которое будет выбрано при использовании типа RADIO (для типа RADIO данный атрибут обязателен).

Под меню выбора в формах понимают такой элемент интерфейса, как LISTBOX. Существует три типа тэгов меню выбора для форм:

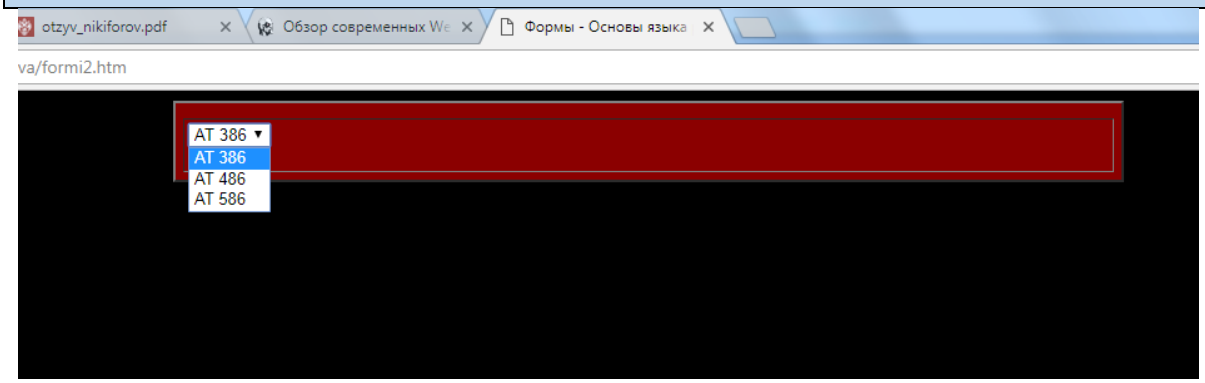
- SELECT — пользователь выбирает одно значение из фиксированного списка значений, представленных тэгами OPTION. Данний вид представляется как выпадающий LISTBOX.
- SELECT SINGLE — то же самое, что и SELECT, но на экране пользователь видит одновременно три элемента выбора. Если их больше, то предоставляется автоматический вертикальный скроллинг.
- SELECT MULTIPLE — позволяет выбрать несколько элементов из LISTBOX.

Тэг SELECT позволяет пользователю выбрать значение из фиксированного списка значений. Обычно это представлено выпадающим меню.

Тэг SELECT имеет один или более параметров между стартовым тэгом <SELECT> и завершающим </SELECT>. По умолчанию, первый элемент отображается в строке выбора. Вот пример тэга <SELECT>:

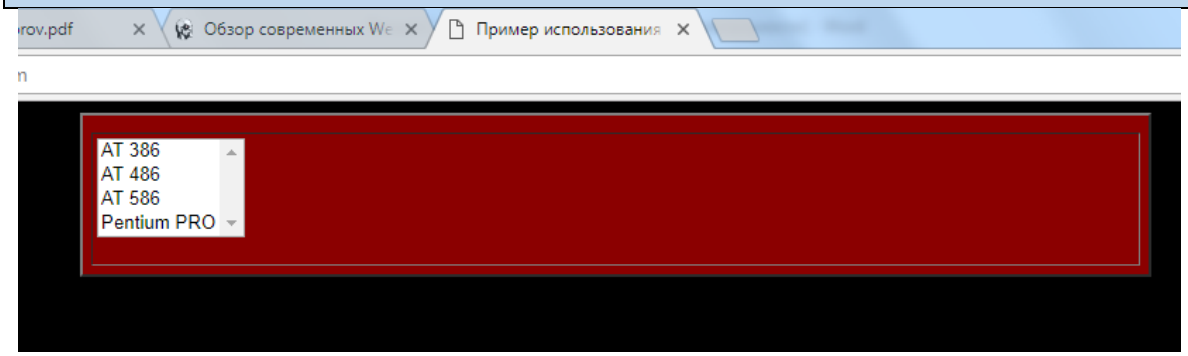
```
<FORM>
<SELECT NAME=group>
<OPTION> AT 386
<OPTION> AT 486
<OPTION> AT 586
</SELECT>
```

```
</FORM>
```



Тэг SELECT SINGLE — это то же самое, что и SELECT, но на экране пользователь видит одновременно несколько элементов выбора (три по умолчанию). Если их больше, то предоставляется автоматический вертикальный скроллинг. Количество одновременно отображаемых элементов определяется атрибутом SIZE. Пример:

```
<FORM>
<SELECT SINGLE NAME=group SIZE=4>
<OPTION> AT 386
<OPTION> AT 486
<OPTION> AT 586
<OPTION> Pentium PRO
</SELECT>
</FORM>
```



Тэг SELECT MULTIPLE похож на тэг SELECT SINGLE, но пользователь может одновременно выбрать более чем один элемент списка. Атрибут SIZE определяет количество одновременно видимых на экране элементов, атрибут MULTIPLE - максимальное количество одновременно выбранных элементов. Пример:

```
<FORM>
<SELECT SINGLE NAME=group SIZE=4 MULTIPLE=2>
<OPTION> AT 386
<OPTION> AT 486
<OPTION> AT 586
<OPTION> Pentium PRO
```

```
</SELECT>
</FORM>
```

Если выбрано одновременно несколько значений, то серверу передаются соответствующее выбранному количество параметров NAME=VALUE с одинаковыми значениями NAME, но разными VALUE.

Отправление файлов при помощи форм

Формы можно использовать для отправки не только небольших информационных сообщений ввиде параметров, а также и для отправки файлов. Внимание! Поскольку данная возможность требует поддержки получения файлов WEB-сервером, то, соответственно, необходимо, чтобы сервер поддерживал получение файлов!

Например:

```
<FORM ENCTYPE="multipart/form-data" ACTION="url" METHOD=POST>
<INPUT TYPE="submit" VALUE="Отправить файл">
</FORM>
```

10. Организация процесса разработки web-контента.CMS/CMF.

Система управления контентом (Content management system, CMS) - компьютерная программа, используемая для создания, редактирования, управления и публикации контента некоторым систематическим образом.

Обычно такие системы используются для хранения и публикации большого количества документов, изображений, музыки или видео.

Система управления web-контентом (Web content management system, WCMS или Web CMS) - програмное обеспечения *CMS* класса, реализованное обычно в виде web-приложения, и предназначенное для создания, и управления HTML содержимым. *WCMS* обычно используется для управления и контроля большими, динамически изменяемыми коллекциями web-материала (*HTML* документами и связанными с ними картинками). Такая система упрощает процесс создания, управления, редактирования контента и многие другие важные задачи, связанные с поддержкой этих процессов.

WCMS предоставляет следующие возможности:

- *Применение автоматических шаблонов отображения* (в *HTML* формате), автоматически применяемых к новому или существующему контенту. Тем самым вид всех документов может задаваться из одного места.
- *Простота редактирования контента*. Пользователю достаточно легко создавать и управлять контентом, поскольку ему либо вообще не требуется знания языков программирования или языков разметки, либо требуется минимальное знание таковых.
- *Масштабируемость*. Возможность расширения функциональности существующего сайта путем установки поставляемых с дистрибутивом *WCMS плагинов* и *модулей*.
- *Управление документами*. Имеются средства управления жизненным циклом документов с момента создания до удаления.
- *Визуализация контента*. Любой пользователь может работать с виртуальной копией всего web-сайта, множества документов или кодами программ, что позволяет *увидеть* все *изменения* множества взаимосвязанных ресурсов перед их окончательным применением.

В зависимости от способа применения шаблонов для генерации web-страниц принято выделять три основные типа WCMS-систем: *с автономной обработкой*, *он-лайн обработкой* и *гибридные системы*.

- Автономные системы обрабатывают все содержимое путем применением шаблонов перед публикацией *web-страниц*.
- On-line системы применяют шаблоны в момент посещения сайта пользователями (либо извлекают страницы и кэша).
- Гибридные системы комбинируют первые два подхода. Некоторые из них вместо статических *HTML* страниц генерируют исполняемые коды (*JSP, PHP, Perl*), избавляя от необходимости установки WCMS-системы на каждом web-сервере.

В качестве примера системы рассмотрим WCMS Drupal**.**

Drupal – это WCMS система, разработанная на языке PHP и использующая в качестве хранилища данных реляционную базу данных (поддерживаются MySQL, PostgreSQL и другие). Архитектура *Drupal* позволяет применять его для построения различных типов сайтов - от блогов и форумов, до информационных архивов или сайтов новостей.

Функциональность обеспечивается подключаемыми модулями, обращающимися к общему *API Drupal*. Стандартный набор модулей включает, например, такие функции как новостная лента, блог, форум, загрузка файлов, сборщик новостей, голосования, поиск и др.

Наиболее важные функции, предоставляемые модулями входящими в поставку Drupal:

- единая категоризация всех видов содержимого (таксономия) - от форумных сообщений до блогов и новостных статей;
- широкий набор свойств при построении рубрикаторов: плоские списки, иерархии, иерархии с общими предками, синонимы, родственные категории;
- вложенность категорий любой глубины;
- поиск по содержимому сайта, в том числе поиск по таксономии и пользователям;
- разграничение доступа пользователей к документам;
- динамическое построение меню;
- поддержка сменных тем оформления сайта с предоставлением нескольких готовых вариантов;
- переводы интерфейса сайта на разные языки, а также поддержка ведения разноязычного контента;
- возможность создания сайтов с пересекающимся содержимым (например общей базой пользователей или общими настройками);

- раздельные конфигурации сайта для различных виртуальных хостов (в том числе собственные наборы модулей и тем оформления для каждого подсайта);
- механизм для ограничения нагрузки на сайт (автоматическое отключение при высокой посещаемости части информационных блоков и модулей).

Существует огромное множество систем как коммерческих, так и бесплатных. Например Майкрософт предлагает реализацию WCMS системы на базе *Windows SharePoint Services*.

Каркасная система управления содержимым (Content Management Framework, CMF) — это инструментарий для создания систем управления содержимым, а также отдельных web-приложений. Некоторые CMS, предоставляющие API для расширения своей функциональности, можно рассматривать как CMF, например WCMS *Drupal* .

11. Синдикация и аггрегирование web-контента

Web-синдикация - форма синдикации при которой содержимое web-сайта предоставляется другим многочисленным web-сайтам. Иначе говоря, *web-синдикация* означает создание доступных с сайта *web-потоков* (feed), предоставляющих всем пользователям в форме краткой сводки информацию о новом содержимом, появившемся на сайте (это могут быть новости, сообщения из форума и др.).

Web-поток - формат данных, используемый для предоставления пользователям часто обновляемого контента. Распространители контента *объединяют* (*синдицируют*) web-потоки, давая пользователям возможность *подписаться* на них. Другое название для web-потока - *синдицированный поток*. Создание набора web-потоков, которые доступны одновременно в одном месте называется *аггрегированием*. Для этого используются специальные *аггрегаторы*.

Аггрегатор потоков (feed aggregator) - клиентское web-приложение, собирающее синдицированный web-контент такой как новостные *заголовки*, *блоги*, *подкасты* и другие в одном месте для более удобного просмотра.

Для *принимающего сайта* web-синдикация является эффективным способом размещения более исчерпывающей и своевременной информации на своих страницах.

Для *сайта передающего* синдицируемую информацию выгода заключается в большей степени его представленности среди различных он-лайн платформ. Кроме того, порождается дополнительный трафик, что, по-сути, является простой и бесплатной формой рекламы сайта в сети web.

Взаимодействие web-потоков и аггрегаторов происходит в следующем порядке:

- *Провайдер* контента *публикует ссылку* на поток со своего сайта.
- *Пользователь* может *зарегистрировать* эту ссылку с помощью *программы-аггрегатора* на своем компьютере.
- *Программа-аггрегатор* затем опрашивает все серверы, входящие в список зарегистрированных потоков, с целью получения нового контента.
- При наличии нового контента *программа-аггрегатор* либо *информирует* пользователя о наличии такового либо сразу же *загружает* его.

Контент web-потока обычно представляет собой *web-страницы*, *гиперссылки* либо *мультимедиа*. Извлечение контента с сайта в форме web-

потока обычно производится средствами самого web-сайта. Однако, не все web-сайты могут иметь web-поток. В этом случае могут быть использованы средства сторонних агентов. Web-поток является удобным инструментом для доставки структурированной информации. Пользователи могут подписываться на web-потоки с помощью *аггрегаторов* или программ для чтения потоков, которые комбинируют содержимое нескольких web-потоков для отображения на одной странице (или нескольких последовательных страницах).

Некоторые из web-браузеров содержат встроенные возможности для аггрегирования потоков. Это делается путем простого ввода URL web-потока или кликом на гиперссылке в браузере. Формат web-потоков не предназначен для непосредственного чтения пользователем, поскольку позволяет автоматически переносить контент с сайт на сайт.

Если сравнивать web-поток с более традиционной почтовой технологией доставки часто обновляемой информации, то можно указать на следующие преимущества первого:

- Поскольку при подписке пользователь не указывает свой адрес электронной почты, эта технология лишена таких потенциальных угроз как спам, вирусы, фишинг и кража личной информации.
- При отказе от использования web-потока нет необходимости отправлять запрос на отказ от подписки; пользователь просто исключает данный поток из своего аггрегатора.
- Имеются широкие возможности для автоматической сортировки сообщений от web-потоков вплоть до использования сложных правил и регулярных выражений.
- Браузеры *Internet Explorer*, *Opera*, *Safari*, *Firefox* и другие могут работать с web-потоками через инструменты панели *Закладок*, *Избранного* и других. Имеются также специализированные программы для чтения web-потоков, например *FeedDemon, Thunderbird, Outlook* и другие.

Аггрегатор позволяет объединить информацию из разных потоков в одном окне web-браузера или web-приложения.

Многие языки программирования имеют библиотеки функций, позволяющие загружать, обрабатывать, генерировать и выполнять удаленную загрузку *каналов*.

12. Web-порталы. Классификация web-порталов.

Портал предоставляет *единую точку безопасного доступа* часто в форме *web-интерфейса*, и предназначен для *аггрегирования* и *персонализации* информации с помощью подходящих *портлетов*.

Например, корпоративный портал (корпоративный информационный портал) - среда интеграции информации, людей и процессов из различных организационных подразделений. Важным признаком корпоративных порталов является децентрализованное распределение и управление контентом, что позволяет информации постоянно обновляться.

Портлеты - подключаемые программные компоненты пользовательского интерфейса, управляемые и отображаемые в *web-портале*. *Портлеты* генерируют фрагменты кода разметки, которые внедряются на страницу портала. Страница портала представляет собой набор непересекающихся окон портлетов. Например, окно электронной почты, окно погоды, окно форума или новостей. *Стандартизация* портлетов нацелена на предоставление в распоряжение программиста портлетов, которые могут быть встроены в любой портал, поддерживающий стандарты.

WSRP (Web Services for Remote Portlets) – протокол, который можно рассматривать как стандарт для *web-сервисов*, позволяющий автоматически встраивать удаленно запущенные *портлеты* из совершенно разных источников.

Спецификация *Java-портлетов* (JSR168, JSR286) даёт возможность для *портлетов* из разных web-порталов взаимодействовать между собой. Эта спецификация определяет множество *API* для взаимодействия *контейнера портлета* с *портлетом* и рассматривает такие вопросы как *персонализация*, *представление* и *безопасность*.

Web-портал - это web-сайт, предоставляющий пользователю различные интерактивные сервисы, работающие в рамках одного web-сайта (web-страницы). Web-порталы зачастую выполняют роль единой точки доступа к информации в WWW. Порталы представляют информацию из разных источников единообразным способом. Наряду с стандартной функцией поиска, web-порталы предлагают и другие сервисы такие как электронная почта, новости, форумы, голосования и другие.

Первичная классификация порталов включает два основных класса:

- Горизонтальные порталы.
- Вертикальные порталы.

Кроме того, можно рассматривать и другие классы порталов:

- Персональные.
- Академические.
- Правительственные.
- Корпоративные.
- Тематические.

Общедоступные (или горизонтальные) порталы (называемые иногда мегапорталами), такие как Lycos, Excite, Rambler, Yandex и др. Эти порталы предназначены для самой широкой аудитории, что отражается на содержании предоставляемой ими информации и услуг — обычно они носят общий характер (например, новости о политических событиях и культурной жизни, электронная почта, новостные рассылки и т.д.). Сфера деятельности таких порталов пересекается со сферой деятельности средств массовой информации, поэтому в последнее время наблюдаются процессы слияния общедоступных порталов и средств массовой информации в рамках одной компании.

Вертикальные порталы предназначены для специфических видов рынка и обслуживают аудиторию, пользующуюся услугами этого рынка или работающую на нем. Примерами таких порталов могут служить приложения B2C (*Business-to-consumer*), например туристические агентства, предоставляющие услуги по бронированию мест в гостиницах, заказу и доставке билетов, доступу к картам и т.п., либо порталы типа B2B (*business-to-business*), позволяющие своим клиентам реализовывать совместные бизнес-операции (например, выбирать поставщиков и осуществлять закупку товаров, проводить аукционы и т.п.). Число подобных порталов в последнее время быстро растет, поскольку всё новые рынки товаров и услуг перемещаются в Интернет.

Корпоративные порталы предназначены для сотрудников, клиентов и партнеров одного предприятия (иногда они называются B2E-порталы - *Business-to-employees*). Пользователи такого портала получают доступ к предназначенным им сервисам и приложениям в зависимости от их роли и персонального профиля, и это наиболее интересная категория порталов в плане реализации корпоративной инфраструктуры и интеграции приложений.

Корпоративный портал предназначен для:

- предоставления внешним и внутренним пользователям возможности персонифицированного доступа ко всем корпоративным данным и приложениям (включая неструктурированные и разнородные данные),

- объединения изолированных моделей бизнеса, интеграция различных корпоративных приложений (в том числе приложений бизнес-партнеров),
- обеспечения полноценного круглосуточного доступа всех пользователей (включая и мобильных) к ресурсам компании независимо от их места пребывания.

Первое поколение корпоративных порталов имеет следующие характеристики:

- поиск и индексирование широкого набора информационных репозитариев;
- категоризация информационного наполнения;
- управление информационным наполнением и его агрегация;
- персонализация;
- высокоэффективная разработка приложений и возможности интеграции с другими приложениями.

Для второго поколения корпоративных порталов, применяемых в качестве составляющей части электронного бизнеса, характерны:

- надежная среда реализации приложений;
- мощные и гибкие инструменты разработки приложений;
- широкие возможности в области интеграции приложений;
- соответствие требованиям к информационным системам масштаба предприятия;
- поддержка интеграции с другими приложениями и информационными системами партнеров;
- поддержка мобильного/беспроводного доступа к данным.

В составе типичного корпоративного портала условно можно выделить три основных функциональных слоя:

- *Слой базовой инфраструктуры*, отвечающий за базовые сервисы, такие как управление транзакциями, система безопасности, управление порталом и др. Технически он содержит, как правило, сервер приложений, сервер баз данных и web-сервер, либо несколько подобных серверов.
- *Слой интеграции приложений*, отвечающий за взаимодействие портала со всеми существующими в компании приложениями, такими как СУБД, CRM- и ERP-системы, унаследованные приложения и др.
- *Слой интерфейсов*, включающий в себя средства управления информационным наполнением (CMS – Content Management System),

интерфейсы для обмена данными с информационными системами бизнес-партнеров, средства для работы с мобильными и беспроводными устройствами и др. К этому же слою относятся визуальные и невизуальные компоненты порталов, называемые обычно портлетами, но иногда имеющие и другие названия (Pagelets, Gadgets, iViews и т.д.).

Из числа наиболее распространенных средства создания порталов можно назвать следующие:

Microsoft SharePoint Server.

WebSphere Portal Server - компании IBM.

Oracle 9iAS Portal - компании Oracle Corporation.

Enterprise Portal - компании SAP Portals.

iPlanet Portal Server - компании Sun Microsystems.

Sybase Enterprise Portal - компании Sybase.

InfoExchange Portal - компании BroadVision.

13. Введение в Web 2.0.

Термин *Web 2.0* используется для обозначения новых тенденций в использовании технологий WWW, направленных на расширение творческих возможностей пользователей, более безопасный обмен информацией и взаимодействие между ними.

При этом больший акцент делается на формирование web-сообществ и социально-ориентированных сайтов таких как, например, *блоги* и *видеоблоги*, *фолксономии*, *википедии* и др.

Термин получил распространение после конференции по Web 2.0 Медиа в 2004 году с подачи Тима О'Рейли для выражения нового способа взаимодействия разработчиков ПО и конечных пользователей через Web. Ключевой принцип идеологии Web 2.0 был сформулирован как: ”Интернет - как платформа”.

Web 2.0 можно рассматривать и как подход к построению систем, при котором они становятся *тем лучше, чем больше людей* ими пользуются в процессе *сетевых взаимодействий*. Фактически Web 2.0 означает переход web-сайтов от изолированных накопителей информации к взаимосвязанным программным платформам, воспринимаемым пользователями так как будто они испольняются локально на его компьютере.

Сайты *Web 2.0* предоставляют для пользователей пользователи возможность:

- Не просто получать информацию, но и выполнять программы исключительно через браузер;
- Размещать и управлять своими данными.

Еще одна важная концепция Web 2.0 - «Архитектура участия», которая поощряет пользователей повышать ценность ПО, путем его использования.

В качестве ключевых для Web 2.0 рассматриваются следующие технологии:

- Web-сервисы — это программы, доступ к которым осуществляется через протокол *HTTP*, а обмен данными происходит в формате *XML* (или производном от него). *Web-сервис* реализуется на серверах компании, её создала создавшей. В любой момент пользователю доступны самые свежие данные; *Web-сервисы* являются платформо-независимыми, поскольку инструменты для работы с *HTTP* и *XML* есть в любом современном языке программирования.
- AJAX. Использование Ajax стало наиболее популярно после того как *Google* начала активно использовать его при создании своих сайтов, таких как *Gmail* и *Google Maps*.

- Web-синдикация (на основе технологий RSS или Atom) - одновременное распространение информации в том числе аудио- и видео- на различные страницы или web-сайты.
- *Web mash-up* - сервис, который полностью или частично использует в качестве источников информации другие сервисы, предоставляя пользователю новую функциональность для работы. *Web mash-up* сервис может становиться также новым источником информации для других *web mash-up* сервисов. Таким образом, образуется сеть зависимых друг от друга сервисов, интегрированных друг с другом.
- *Теги* - ключевые слова, описывающие рассматриваемый объект, либо относящие его к какой-либо категории. Теги можно рассматривать как метки, которые присваиваются объекту, для определения его места среди других объектов.
- *Фолксономия* — популярная классификация, практика совместной категоризации информации (ссылок, фото, видео клипов и др.) посредством произвольно выбираемых меток (тегов). Примеры использования *фолксономии*: Flickr, del.icio.us.
- *Социальное ПО* - широкий диапазон ПО, предназначенного для обмена и совместного доступа к информации пользователей сети Web.

К недостаткам Web 2.0 можно отнести следующие:

- Зависимость от наличия постоянного соединения;
- Зависимость работоспособности сайтов от решений сторонних компаний;
- Зависимость качества работы сервиса от качества работы многих других компаний;
- Недостаточные возможности существующей web-инфраструктуры для выполнения сложных вычислительных задач в браузере;
- Уязвимость конфиденциальных данных, хранимых на сторонних серверах, при несанкционированном доступе.

Мэшапы

Мэшап (*Mashup*) - гибридное web-приложение, объединяющее данные из нескольких источников в рамках единого интегрированного инструмента. Контент *мэшап* обычно получает извне с помощью открытых интерфейсов, web-сервисов, web-источников (например *RSS* или *Atom*) или анализа документов, генерируемых другими программами.

Наиболее часто *мэшапы* используют программные интерфейсы представляемые *Amazon*, *eBay*, *Flickr*, *Google*, *Microsoft*, *Yahoo* и *YouTube*.

Архитектура *мэшапа* включает в себя 3 части:

- *провайдер контента* - источник данных.
- собственно *мэшап* – web-приложение, предлагающее новую функциональность с использованием различных источников, не принадлежащих ему.
- *клиент* - обычно web-браузер, отображающий web-страницу мэшапа.

Различные типы *мэшапов* могут генерировать RSS, web-сервисы, мгновенные и почтовые сообщения.

Следует отличать *мэшапы* от простого *внедрения данных* с других сайтов с образованием сложных документов. *Мэшап* самостоятельно получает внешние данные через программные интерфейсы и, обрабатывая их определенным образом, придает им дополнительную ценность.

В качестве примера можно привести использование картографических данных *Google Maps* для добавления к ним данных о недвижимости в *Cragislist* (в результате создается новый уникальный web-сервис, изначально не предлагаемый ни одним из источников).

Среди мэшапов можно выделить следующие группы:

- *Потребительские мэшапы* - комбинируя данные различных типов из нескольких источников, предоставляют доступ к ним с помощью единого графического интерфейса. Наиболее известный пример - многочисленные приложения *Google Maps*,
- *Мэшапы данных* - смешивают данные близкие по типу из нескольких источников, предоставляет доступ к ним с помощью единого графического интерфейса. Пример - *Yahoo!Pipes* позволяет пользователям получать потоки информации из разных источников и создавать правила по управлению полученным контентом (например, используя фильтры).
- *Бизнес-мэшапы* - акцент делают на различных способах аггрегирования и представления данных, предоставляя новые возможности для совместной работы представителей бизнеса и разработчиков.

Хотя *мэшапы* также как и *порталы* являются технологиями аггрегирования контента, имеется принципиальное отличие между ними. *Порталы* - более ранняя технология, являющаяся, по-сути, расширением традиционных динамических web-приложений, в которых процесс преобразования данных в гипертекстовые документы разбивается на два этапа: генерация отдельных частей разметки и объединение их на одной странице. За генерацию каждой из

частей отвечает соответствующие портальные приложения, исполняемые на портальном сервере или другом сервере. *Портальная* технология является серверной технологией, отвечающей за *аггрегирование* данных *только на уровне представления*.

Компания Майкрософт предоставляет пользователям специальный сайт ***Microsoft Popfly***, коотрый позволяет пользователям создавать web-страницы, фрагменты программного кода (для повторного использования) и мэшапы с помощью пакета разработки приложений с поддержкой *Microsoft Silverlight* .

На сайте имеются четыре инструменты, основанные на *Microsoft Silverlight*:

- *Разработчик игр*
- *Разработчик мэшапов*
- *Разработчик web-страниц*
- *Popfly Space* – пространство для размещения готовых мэшапов и web-страниц, доступных пользователям.

14. Приложения для социальных сетей

Понятие "*Социальный Web*" (*Social Web*) используется для описания того, как происходит социализация пользователей и их взаимодействие друг с другом с помощью сети *WWW*. Основой для объединения пользователей служат самые разнообразные общие интересы.

Термин "*Социальный Web*" может быть использован для выражения двух понятий.

- Первое понятие связано с описанием технологий Web 2.0, которые фокусируются в первую очередь на социальных взаимодействиях и сообществах.
- Второе понятие используется для описания сети будущего, аналогичной WWW.

Можно рассматривать как *сообщество* множество людей, связанных и взаимодействующих между собой посредством контента в режиме диалога и совместных действий через Интернет.

Будучи нацеленными на стимулирование взаимодействия между людьми, *социальное ПО* для Web оперирует со следующими *социальными атрибутами*:

- идентичность,
- репутация,
- присутствие,
- связи,
- групповая принадлежность,
- общение,
- разделяемый контент.

Социально программное обеспечение - это широкий диапазон программных систем, позволяющмх пользователям взаимодействовать и обмениваться данными. Этот способ компьютерно-опосредованного взаимодействия стал популярным с появлением таких социальных сайтов как MySpace, Facebook, Одноклассники, медиа сайтов Flickr и YouTube, коммерческих сайтов eBay.

Многие из из этих приложений имеют такие общие характеристики как: открытые API, сервис-ориентированный дизайн, возможность удаленного размещения данных и медиа-файлов. Такие приложения принято относить к системам Web 2.0.

Внутри социального ПО можно выделить две группы программных инструментов: *коммуникационные* и *интерактивные*.

- *Коммуникационные* инструменты применяются для записи, хранения и представления коммуникационных данных, чаще всего в текстовом виде, но все больше в аудио и видео форматах.
- *Интерактивные* инструменты применяются для поддержки опосредованного данными (различных медиаформатов) взаимодействия между отдельными пользователями и их группами. В отличие от коммуникационных инструментов акцент делается на поддержке связности пользователей и механизмов общения между ними.
- В противоположность *коммуникационным* инструментам, которые являются обычно асинхронными, *интерактивные* инструменты преимущественно синхронны, позволяя взаимодействовать пользователям в режиме реального времени (как в случае интернет-телефонии, видеочатов и т.п.) либо почти синхронно (службы мгновенных сообщений, текстовые чаты и т.п.).

Можно назвать следующие примеры программных систем, которые относится к социальному ПО:

- Системы обмена мгновенными сообщениями (*IM - Instant messaging*) позволяют общаться с другим пользователем через сеть в режиме реального времени (в относительно защищенном режиме). К наиболее популярным из них можно отнести Skype, ICQ, Yahoo! Messenger, MSN Messenger, AOL Instant Messenger, Miranda IM. К системам, ориентированным на бизнес, можно отнести IBM Lotus Sametime, Microsoft Messenger и Jabber.
- Интернет-чаты (*IRC - Internet Relay Chat*) позволяют одновременно нескольким пользователям общаться в режиме реального времени.
- Интернет-форумы пришли на смены электронным конференциям (возникшим до появления WWW). Пользователь форума может создавать новую "тему", доступную для других. Другие пользователи могут просматривать тему и оставлять свои комментарии в режиме последовательной записи.
- Web-блоги (*web logs*), или кратко блоги, можно рассматривать как личные он-лайн журналы отдельных пользователей. Владелец блога может размещать сообщения в своем журнале, в то время как другие пользователи (читатели) могут оставлять к ним свои комментарии.
- Вики-справочники (*wiki*), или просто вики, - по-сути, web-сайты, содержимое которых может редактироваться посетителями сайта. Наиболее известный пример - Википедия.

- Сервисные социальные сети позволяют пользователям объединяться в онлайн режиме вокруг общих для них интересов, увлечений или по различным поводам. Например, некоторые сайты предоставляют сервисы, с помощью которых пользователи могут размещать для общего доступа персональную информацию, необходимую для поиска партнеров. Примеры: LinkedIn, В контакте.
- Социальные сети прнятия решений - web-сети для обсуждения с целью принятия решений. Используются для установления постоянной связи отдельных людей с правительством.
- Коммерческие социальные сети ориентированы на поддержку бизнес-транзакций и формирование доверия людей к брендам на основе учета их мнений о продукте, о том как сделать его лучше и т.п., тем самым позволяя потребителям участвовать в продвижении продукта и расширяя их осведомленность.
- Социальные закладки (*social bookmarking*). Некоторые webсайты позволяют пользователям предоставлять в распоряжение других список закладок или популярных webсайтов. Такие сайты также могут использоваться для поиска пользователей с общими интересами. Пример: del.icio.us.
- Социальные каталоги (*social cataloging*) напоминают социальные закладки, но ориентированы на использование в академической сфере, позволяя пользователям работать с базами данных цитат из научных статей. Примеры: *Academic Search Premier, LexisNexis Academic University*, CiteULike, Connotea.
- Социальные библиотеки представляю собой приложения, позволяющие посетителям оставлять ссылки на их коллекции, книги, аудиозаписи и т.п., доступные другим. Предусмотрена поддержка системы рекомендаций, рейтингов и т.п. Примеры: discogs.com, imdb.com.
- Многопользовательские сетевые игры (*Massively Multiplayer Online Games*) имитируют виртуальные миры с различными системами подсчета очков, уровней, состязательности, победителей и проигравших. Пример: World of Warcraft.

Фолксономия

Фолксономия (*folksonomy*) — практика и методика совместной категоризации контента (ссылок, фото, видео клипов и т.п.) посредством произвольно выбираемых меток (тегов). Она основана на спонтанном

сотрудничестве группы людей с целью организации контента и полностью отличается от традиционных формальных методов классификации на основе *индексных терминов*. Как правило, этот феномен возникает только в неиерархических сообществах, например на общедоступных web-сайтах. Так как участники *фолксономии* контента обычно являются и основными же ее потребителями, использование методики фолксономии приводит к результатам, более точно отражающим *совместную концептуальную модель контента* всей группы.

Основными проблемами фолксономии, приводящими к ненадежности и несогласованности результатов, являются:

- Наличие форм множественного числа.
- Полисемия.
- Синонимия.
- Глубина (специфичность) использования меток.

Фолксономию можно рассматривать в качестве одного из ключевых элементов в развитии *Семантической web-сети*, в рамках которой все web-страницы содержат машинно-ориентированные метаданные, описывающие содержимое страниц. Эти метаданные должны значительно улучшать точность работы поисковых систем.Однако во избежание трудностей, вызванных ненадежностью и несогласованностью в работе больших сообществ авторов страниц, им рекомендуется использовать стандарты метаданных, например *Дублинское ядро* (*Dublin Core*).

Семантическая web-сеть

Семантическая web-сеть (Semantic Web) — часть глобальной концепции развития сети Интернет, целью которой является реализация возможности машинной обработки информации, доступной в сети WWW. Основной акцент в этой концепции делается на работе с *метаданными*, однозначно характеризующими свойства и содержание ресурсов WWW, вместо используемого в настоящее время *текстового анализа* документов.

Термин был введен Тимом Бернерсом-Ли в мае 2001 года.

В семантической web-сети предполагается повсеместное использование

- универсальных идентификаторов ресурсов (URI),
- *онтологий* и языков описания метаданных.

Концепция семантической web-сети была принята и продвигается W3C. Для её внедрения предполагается создание сети документов, содержащих метаданные о ресурсах WWW, и существующей параллельно с ними. Тогда

как сами ресурсы предназначены для восприятия человеком, метаданные используются машинами (поисковыми роботами и другими интеллектуальными агентами) для получения однозначной информации о свойствах этих ресурсов с помощью механизмов логического вывода.

Техническую часть семантической паутины составляет семейство стандартов на языки описания, включающее *XML*, *XML Schema*, *RDF*, *RDF Schema*, *OWL* и др. 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.

Онтология

Онтология - это попытка всеобъемлющей и детальной формализации некоторой области знаний с помощью концептуальной схемы. Обычно такая схема состоит из иерархической структуры данных, содержащей все релевантные классы объектов, их связи и правила (теоремы, ограничения), принятые в этой области.

Современные онтологии обычно состоят из *экземпляров*, *понятий*, *атрибутов* и *отношений*.

Для описания онтологий Web был разработан специальный язык - *OWL* (Web Ontology Language), построенный на основе XML. Язык *OWL* может быть использован для описания классов и отношений между ними. В основе языка — представление действительности в модели данных «объект — свойство». Язык применим не только для описания web-страниц, но и любых объектов действительности и рассматривается в качестве одной из фундаментальных технологий, необходимых для построения Семантической web-сети.

Семантические web-сервисы

В то время как совокупность ресурсов и их метаданных можно считать статической частью *семантической паутины*, её динамическую часть представляют *семантические web-сервисы* - законченные элементы программной логики с однозначно описанной семантикой, доступные через Web и пригодные для поиска, композиции и выполнения.

Технически, *семантический web-сервис* отличается от обычного web-сервиса наличием не только описания интерфейса (обычно на языке *WSDL*) в терминах типов данных, передаваемых сервису, возвращаемых значений и

генерируемых ошибок, но и наличием семантического описания всех его характеристик.

Потенциальная выгода от использования семантических web-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.

Список литературы

1. П.Б. Храмцов, С.А.Брик, А.М. Русак, А.И.Сурин Основы WEB-технологий. – М.: ИТУИТ.РУ, 2003. – 512 с.
2. Д.Роджерс Программирование на Microsoft JScript.NET. “Вильямс”, 2002. - 352 с.
3. В.В. Дунаев JavaScript.— СПб. : “Питер”, 2003 .— 394 с.
4. Б. Форта Освой самостоятельно регулярные выражения. 10 минут на урок. – М.: “Вильямс”, 2005. – 184 с.
5. К.Мельтцер Разработка CGI-приложений на Perl. М.: “Вильямс”, 2001.— 395 с.
6. И.Шапошников PHP 5.1: учебный курс — СПб [и др.]: “Питер”, 2007.— 192 с.
7. Д.Шеперд Освой самостоятельно XML за 21 день. – М.: “Вильямс”, 2002. – 432 с.
8. А.Старыгин XML: разработка Web-приложений. —СПб.: “БХВ-Петербург”, 2003 .— 585 с.
9. Э.Троелсен C# и платфома .NET. Библиотека программиста. – СПб.: “Питер”, 2007.- 796 с.
10. М.Беллиньясо Разработка Web-приложений в среде ASP.NET 2.0: с примерами на C#. “Диалектика”, 2007. - 640 с.
11. С.Шорт Разработка XML Web-сервисов средствами Microsoft .NET. — СПб.: “БХВ-Петербург”, 2003 .— 480 с.
12. Э.Ньюкомер Web-сервисы: XML, WSDL, SOAP и UDDL. — СПб.: Питер, 2003.
13. И.Салмре Программирование мобильных устройств на платформе .NET Compact Framework. “Вильямс”, 2006. - 736 с.
14. Д.Крейн, Э.Паскарелло, Д. Джеймс AJAX в действии: технология - Asynchronous JavaScript and XML = AJAX in Action. — М.: “Вильямс”, 2006. — С. 640.
15. Wikipedia, the free encyclopedia (http://en.wikipedia.org/wiki/Main_Page).
16. Википедия (http://ru.wikipedia.org/wiki/)
17. W3CSchools Online Web Tutorials (http://www.w3schools.com/)
18. Интернет Университет информационных технологий (http://www.intuit.ru/catalog/)

Учебное пособие

Современные web-технологии

для студентов специальностей:

220206 «Автоматизированные системы обработки информации и управления»

230109 «Программное обеспечение средств вычислительной техники»

Колледжа КГУ имени И.Арабаева

Составитель:

Ниязмаметова Сахинур Абликимовна