МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

ТЕХНОЛОГИИ СЕТИ ИНТЕРНЕТ

Электронный учебно-методический комплекс
по дисциплине в LMS Moodle

Работа выполнена по мероприятию блока 1 «Совершенствование
образовательной деятельности» Программы развития СГАУ
на 2009 – 2018 годы по проекту «Разработка контента для системы электронного и дистанционного
обучения по основным образовательным программам факультета информатики»
Соглашение № 1/34 от 3.06.2013 г.

САМАРА
2013

УДК 004.738.5(075) ББК 32.973.202
Т 384

Автор-составитель: **Котенева Светлана Эдуардовна**

Технологии сети Интернет [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle / Мин-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост. С.Э. Котенева. - Электрон. текстовые и граф. дан. - Самара, 2013. – 1 эл. опт. диск (CD-ROM).

В состав учебно-методического комплекса входят:
1. Курс лекций.
2. Лабораторные работы. Задания и методические указания.
3. Рекомендованная литература.
4. Контрольные вопросы.
5. Методические указания к курсовой работе.
6. Билеты. Итоговое тестирование.
7. Рабочая программа курса.

УМКД «Технологии сети Интернет» предназначен для студентов факультета информатики, обучающихся по направлению подготовки бакалавров 010300.62 «Фундаментальная информатика и информационные технологии» в 7 семестре.

УМКД разработан на кафедре Программных систем.

1. ВВЕДЕНИЕ В ИНТЕРНЕТ ТЕХНОЛОГИИ

1.1. Принципы построения и структура сети Интернет

1.1.1. Маршрутизация в сети Интернет

Маршрутизация в сети Интернет - пакетная (сравните: канальная, как в телефонии, АТС-маршрутизатор).

Все локальные компьютеры объединены в сеть имеющую IP-адресацию. Пакеты с такой адресацией "путешествовать" в глобальной сети не смогут, т.к. маршрутизаторы их не пропустят.

Поэтому существует шлюз, который преобразовывает пакеты с локальными IP-адресами, давая им свой внешний адрес. И дальше ваши пакеты путешествуют с адресом шлюза.

Схема прохождения пакетов из локальной сети к Интернет-серверу.

Маршрутизаторы объединяют отдельные сети в общую составную сеть (см. рисунок ниже). К каждому маршрутизатору могут быть присоединены несколько сетей (по крайней мере две).

Маршрут - это последовательность маршрутизаторов, которые должен пройти пакет от отправителя до пункта назначения.

1.2. Схема объединения отдельных сетей в общую сеть

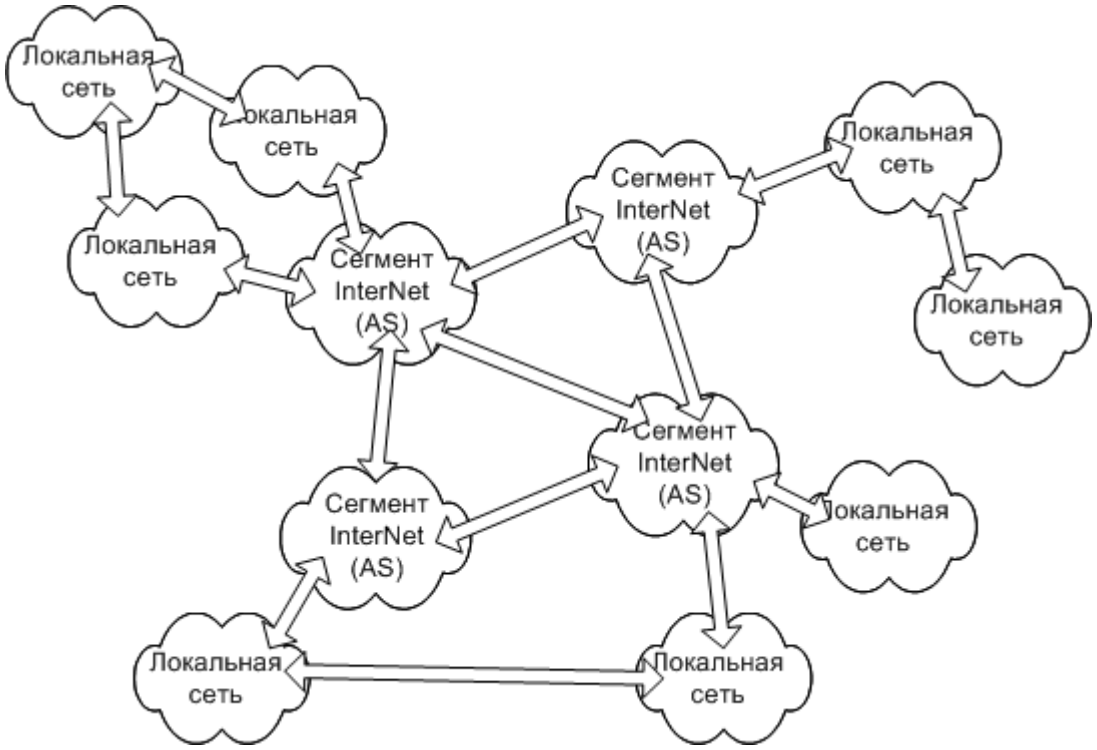


Схема объединения отдельных сетей в общую составную сеть

Локальных сетей много, поэтому реально объединяют автономные системы.

Автономная система – сеть, находящаяся под одним административным контролем, это может быть несколько компьютеров или большая сеть (понятие в достаточной степени условное)

1.3. Официальная документация по Internet

RFC (Request for Comments) - официальная документация по Интернет, можно найти по адресу http://www.rfc-editor.org/ или http://www.ietf.org/rfc.html

Все разработчики должны придерживаться этой документации, но на практике, не всегда так происходит.

2. АДРЕСАЦИЯ В СЕТИ ИНТЕРНЕТ

2.1. Типы адресов:

1. Физический (MAC-адрес)
2. Сетевой (IP-адрес)
3. Символьный (DNS-имя)

Компьютер в сети на базе протокола TCP/IP может иметь адреса трех уровней (но не менее двух):

Локальный адрес компьютера. Для узлов, входящих в локальные сети - это MAC-адрес сетевого адаптера. Эти адреса назначаются производителями оборудования и являются уникальными адресами.

IP-адрес, состоящий из 4 байт, например, 109.26.17.100. Этот адрес используется на сетевом уровне. Он назначается администратором во время конфигурирования компьютеров и маршрутизаторов.

Символьный идентификатор-имя (DNS), например, www.ssau.ru.

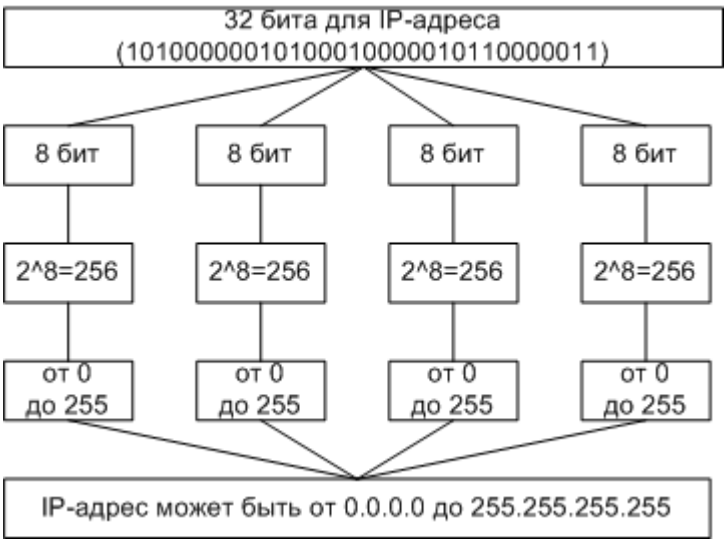
2.2. IP-адреса

IPv4 - адрес является уникальным 32-битным идентификатором IP-интерфейса в Интернет.

IPv6 - адрес является уникальным 128-битным идентификатором IP-интерфейса в Интернет, иногда называют **Internet-2,** адресного пространства IPv4 уже стало не хватать, поэтому постепенно вводят новый стандарт.

IP-адреса принято записывать разбивкой всего адреса по октетам (8), каждый октет записывается в виде десятичного числа, числа разделяются точками. Например, адрес

10100000010100010000010110000011 записывается как
10100000.01010001.00000101.10000011 = 160.81.5.131



Перевод адреса из двоичной системы в десятичную

IP-адрес хоста состоит из номера IP-сети, который занимает старшую область адреса, и номера хоста в этой сети, который занимает младшую часть.

160.81.5.131 - IP-адрес 160.81.5. - номер сети 131 - номер хоста

3. УНИВЕРСАЛЬНЫЙ ИДЕНТИФИКАТОР РЕСУРСОВ (URI), ЕГО НАЗНАЧЕНИЕ И СОСТАВНЫЕ ЧАСТИ

3.1. Что такое URI?

URI (Uniform Resource Identifier, Универсальный идентификатор ресурса) (RFC 2396, August 1998) - компактная строка символов для идентификации абстрактного или физического ресурса. Под ресурсом понимается любой объект, принадлежащий некоторому пространству. Включает и переопределяет определенные ранее URL (RFC 1738/RFC 1808) и URN (RFC 2141, RFC 2611).

URI предназначен для уникальной идентификации любого ресурса.

Некоторые подмножества URI:

URL (Uniform Resource Locator, Универсальный указатель ресурса), - подмножество схем URI, который идентифицирует ресурс по способу доступа к нему (например, его "местонахождению в сети") вместо того, чтобы идентифицировать его по названию или другим атрибутам этого ресурса.

Примеры URL:
http://www.ipm.kstu.ru/index.php ftp://www.ipm.kstu.ru/

В HTML записывается так:

URN (Uniform Resource Name, Универсальное имя ресурса) - частная URI-схема "urn:" с подмножеством "пространства имен", который должен быть уникальным и неизменным даже в том случае, когда ресурс уже не существует или недоступен.

Предполагается что, например браузер, знает, где искать этот ресурс.

Синтаксис:
urn:namespace: data1.data2,more-data, где namespace (пространство имен) определяет, каким образом используются данные, указанные после второго ":".

Пример URN:
urn:ISBN: 0-395-36341-6
ISBN - тематический классификатор для издательств
0-395-36341-6 - конкретный номер тематики книги или журнала

При получении URN клиентская программа обращается к ISBN (каталогу "тематический классификатор для издательств" в Интернете). И получает расшифровку номера тематики "0-395-36341-6" (например: "квантовая химия").

URN массово используется в P2P сетях (подобных edonkey).

Пример URN указывающего на образ диска Adobe Photoshop v8.0 в сети edonkey:
urn:ed2k://|file|Adobe Photoshop v8.0.iso|940769280|b34c101c90b6dedb4071094cb1b9f2d3|/

где
ed2k - указывает на сеть file – файл
Adobe Photoshop v8.0.iso - название файла 940769280 - размер в байтах
b34c101c90b6dedb4071094cb1b9f2d3 - идентификатор файла (вычисляется с помощью хеш-функции)

4. Подробнее про универсальный указатель ресурса URL

4.1. Синтаксис

URL - Uniform Resource Locators явно описывает, как добраться до объекта.

Синтаксис:
<scheme>:<scheme-specific-part> где:
scheme = "http" | "ftp" | "gopher" | "mailto" | "news" | "telnet" | "file" | "man" | "info" | "whatis" | "ldap" | "wais" | ... - имя схемы
scheme-specific-part - зависит от схемы
Имя схемы - последовательность символов [a-z0-9+.-].

В scheme-specific-part можно использовать шестнадцатеричные значения в виде: %5f. Обязательно должны кодироваться непечатные октеты: 00-1F, 7F, 80-FF. Также всегда кодируются "небезопасные" символы: " ", "<", ">", """, "#", "%", "{", "}", "|", "\", "^", "~", "[", "]", "`". Некоторые схемы резервируют и другие символы: ";","/","?",":","@","=" и "&". Их также необходимо кодировать, если хочется "обойти" их специальное трактование. Таким образом остаются [a-z0-9$-_.+!*'(),] и резервированные символы в их специальном значении для данной схемы.

4.2. Некоторые схемы URL

4.2.1. Схема HTTP

В схеме указывается ее идентификатор, адрес машины, TCP-порт, путь в директории сервера, переменные и их значения, метка.

Синтаксис:
http://[<user>[:<password]>@]<host>[:<port>][/[<url-path>][?<query>]] http - название схемы
user - имя пользователя password - пароль пользователя host - имя хоста
port - номер порта
url-path - путь к файлу и сам файл
query (<имя-поля>=<значение>{&<имя-поля>=<значение>) - строка запроса Определен в RFC 2068. По умолчанию, port=80.

Примеры:
http://ipm.kstu.ru/internet/index.php

Это наиболее распространенный вид URI, применяемый в документах WWW. Вслед за именем схемы (http) следует путь, состоящий из доменного адреса машины и полного адреса HTML-документа в дереве сервера HTTP.

В качестве адреса машины допустимо использование и IP-адреса:
http://195.208.44.20/internet/index.php

Если сервер протокола HTTP запущен на другой, отличный от 80 порт TCP, то это отражается в адресе:

http://195.208.44.20:8080/internet/index.php

При указании адреса ресурса возможна ссылка на метку внутри файла HTML. Для этого вслед за именем документа может быть указана метка внутри документа:

http://195.208.44.20/internet/index.php#metka1

Символ "#" отделяет имя документа от имени метки.

Переменные и их значения передаются следующим образом:
http://ipm.kstu.ru/internet/index.php?var1=value1&vard2=value2

Значения "var1" и "var2" - это имена переменных, а "value1" и "value2" - их значения.

4.2.2. Схема FTP

Данная схема позволяет адресовать файловые архивы FTP.

Синтаксис:
ftp://[<user>[:<password]>@]<host>[:<port>][/<url-path>]

где
ftp - название схемы
user - имя пользователя password - пароль пользователя host - имя хоста
port - номер порта
url-path - путь к файлу и сам файл

Определен в RFC 1738. По умолчанию, port=21, user=anonymous, password=email-адрес, если имя указано, а пароль нет, то он запрашивается в диалоге.

<url-path> имеет вид:
<cwd1>/<cwd2>/.../<cwdN>/<name>[;type=<typecode>], где <typecode>:

<url-path> преобразуется клиентской прораммой в набор команд CWD <cwd1>
...
CWD <cwdN> TYPE <typecode> RETR <name>

Примеры:
ftp://ipm.kstu.ru/students/name/

Чтобы указать имя пользователя и его пароль, надо записать так: ftp://name:password@ftp://ipm.kstu.ru/students/name/

В данном случае эти параметры отделены от адреса машины символом "@", а друг от друга двоеточием.

4.2.3. Схема MAILTO

Данная схема предназначена для отправки почты.

Синтаксис:
mailto:[<e-mail-1>{,<e-mail-2>,...}][?<query>] mailto - название схемы
e-mail-1 (<user>@<host>)- первый адрес электронной почты user - имя пользователя
host - имя хоста
e-mail-2 - второй адрес электронной почты
query (<имя-поля-заголовка>=<значение>{&<имя-поля-заголовка>=<значение>) - строка запроса

Примеры:
mailto:name@ipm.kstu.ru

В этой схеме передаются поля и их значения:

Пример:
mailto:name@ipm.kstu.ru?subject=Тема_письма&body=Текст_который_будет_вставлен_в_письмо

Адрес получателя можно также записывать в виде значения поля to: mailto:?to=name@ipm.kstu.ru?subject=Тема_письма&body=Текст_который_будет_вставлен_в_письмо

4.2.4. Схема NEWS

Данная схема используется для просмотра сообщений системы Usenet.

Синтаксис:
news:[<article>@<group>] news - название схемы article - номер статьи
group - название группы

Пример:
news:comp.infosystems.gopher

В данном случае можно получить статьи из группы "comp.infosystems.gopher" в режиме уведомления. Можно получить и текст статьи, но в этом случае указывают ее идентификатор:
news:086@comp.infosystems.gopher

Это означает что заказана 86 статья из группы.

4.2.5. Схема NNTP

Это еще одна схема получения доступа к ресурсам Usenet.

Синтаксис:
nntp:[<group>/<article>]

где
nntp - название схемы
group - название группы
article - номер статьи

Пример:
В данной схеме обращение к группе comp.infosystems.gopher для получения статьи 86 будет выглядеть так:
nntp:comp.infosystems.gopher/086

Следует обратить внимание на то, что адрес сервера Usenet не указан. Программа-клиент должна быть предварительно сконфигурирована на работу с одним из серверов Usenet. Сама служба Usenet является распределенным информационным ресурсом, и группа comp.infosystems.gopher на серверах содержит одни и те же сообщения.

4.2.6. Схема TELNET

По этой схеме осуществляется доступ к ресурсу в режиме удаленного терминала. При использовании этой схемы необходимо указывать имя пользователя и пароль.

Синтаксис:
telnet://[<user>[:<password]>@]<host>[:<port>]/
telnet - название схемы
user - имя пользователя password - пароль пользователя host - имя хоста
port - номер порта

По умолчанию, port=23.
Пример: telnet://name:password@ipm.kstu.ru

4.2.7. Схема FILE

Для локального режима используют схему FILE.

Синтаксис:
file://<host>/<path>
file - название схемы
host - имя хоста
port - номер порта
path - путь к файлу и сам файл
В качестве <host> обычно указывается localhost

Пример:
file:///C|/text/html/index.htm
C - диск файловой системы (для Windows)

Клиент запускает только программы просмотра на основе MIME-типов из заголовка сообщений сервера или по расширению файла.

4.3. Административная структура раздачи IP-адресов InterNet

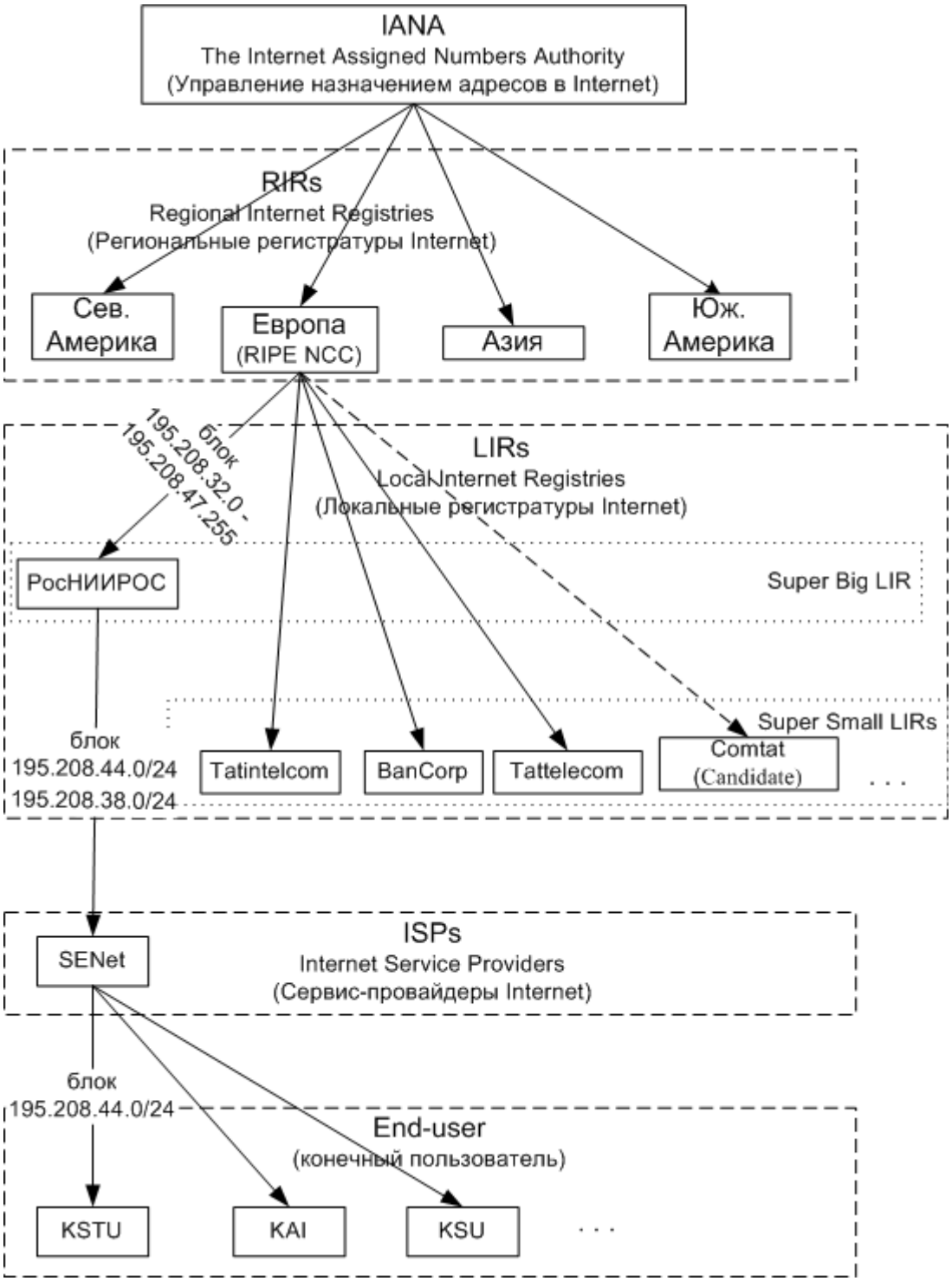
Раздача IP-адресов осуществляется регистраторами.

Основные понятия:

1. **IANA - The Internet Assigned Numbers Authority** (Управление назначением адресов в Интернет) - организация, осуществляющая контроль за распределением всего пространства Интернет адресов, включая IP-адреса. IANA выделяет адресное пространство Региональным регистратурам в соответствии с их потребностями. Сервер http://www.iana.org/ .

Главная база данных, по сетям, поддерживается Регистрационной службой Интернет (InterNIC - www.internic.net).

Базу можно посмотреть по адресу rs.internic.net и whois.internic.net. Через WWW-интерфейс http://www.internic.net/whois.html



Пример получения IP-адресов KSTU (до смены адресов в 2004г.)

2. **RIR - Regional Internet Registry** (Региональная регистратура Интернет) - организация, занимающаяся распределением адресного пространства в пределах одного из 4-х регионов (Северная Америка, Латинская Америка, Европа, Азия). Региональные регистратуры осуществляют координацию деятельности Локальных регистратур.

3. **LIR - Local Internet Registries** (Локальная регистратура Интернет) - организация, занимающаяся распределением адресного пространства пользователям сетей (сервис-провайдерам и их абонентам) и оказанием

сопутствующих регистрационных услуг. Как правило, Локальными регистратурами управляют крупные сервис-провайдеры и корпоративные сети.

4. **ISP** - I**nternet Service Provider** (сервис-провайдер Internet) - поставщик услуг Internet.

5. **End-user** (конечный пользователь) - организация, которая использует выделенное ей адресное пространство для работы своих сетей и подключенная к сети Интернет.

Запрос информации на сервере whois.nic.ru для СГАУ.

```
по данным WHOIS.NIC.RU:

% By submitting a query to RU-CENTER's Whois Service
% you agree to abide by the following terms of use:
% http://www.nic.ru/about/servpol.html (in Russian)
% http://www.nic.ru/about/en/servpol.html (in English).

domain:      SSAU.RU
nserver:     mb.ssau.ru
nserver:     stream.ssau.ru
state:       REGISTERED, DELEGATED
admin-contact:https://www.nic.ru/cgi/whois_webmail.cgi?domain=SSAU.RU
org:         Samara State Aerospace University
registrar:   RU-CENTER-REG-RIPN
created:     2004.09.14
paid-till:   2014.09.01
source:      RU-CENTER
```

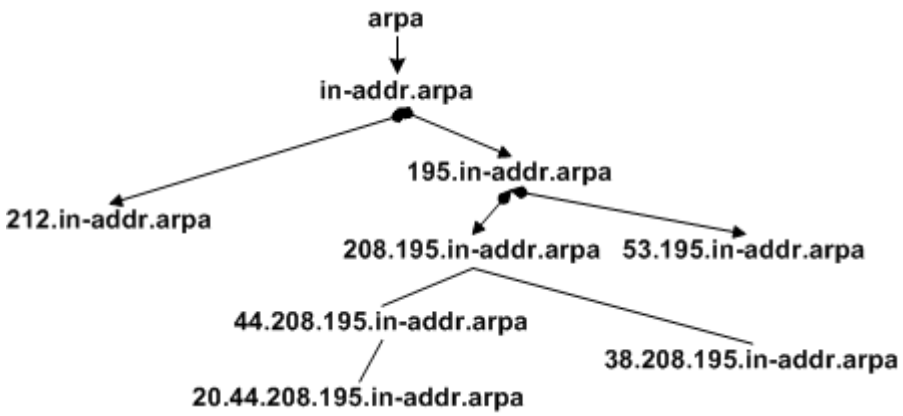
4.4. Обратные зоны

Что бы знать какой адрес кому принадлежит (имя хоста), нужно вести базу данных, для этого стали использовать службу DNS.

Заведен домен in-addr.arpa.

IP адрес 195.208.44.20 будет записан так 20.44.208.195.in-addr.arpa

Тип записи в запросах **PTR.**



Дерево в домене in-addr.arpa

Более подробную информацию можно получить на серверах whois.

Запрос whois для обратной зоны 208.195.in-addr.arpa (сети 195.208.44) , которой пользуется КГТУ.

5. Программа Ping

Программа для проверки соединения с удаленным хостом.

Можно использовать как с командной строки, так и в таких программах как CyberKit (официальный сервер - http://www.cyberkit.net), скачать с локального сервера cyberkit-v2.5.zip.

6. TraceRoute

Программа TraceRoute -- позволяет проверить маршрут до удаленного хоста.

Можно использовать как с командной строки, так и в таких программах как CyberKit

PingPloter - рисует график маршрута (официальный сервер - http://www.pingplotter.com).

VisualRoute - рисует маршрут на карте Земли, и делает запросы whois (официальный сервер -http://www.visualroute.com/).

7. NetScanner

NetScanner - позволяет посмотреть ответ определенного порта.

Можно использовать в таких программах как CyberKit.

Работу порта, также можно проверить с помощью telnet.

Использованные источники:

1. Храмцов П. Б. Администрирование сети и сервисов Internet. Учебное пособие., Центр Информационных Технологий, 1997

2. Храмцов П. Б Лабиринт Internet. Электронинформ, 1996: http://www.lib.ru/LABIRINT/content.htm .

3. Семенов Ю.А. Телекоммуникационные технологии. (v4.11, 17 мая 2013 года): http://book.itep.ru/

4. Принципы построения и организационная структура Интернет: http://www.ipm.kstu.ru/internet/lec/1.php

1. АДРЕСАЦИЯ В СЕТИ ИНТЕРНЕТ

1.1. Типы адресов:

Компьютер в сети на базе протокола TCP/IP может имеет адреса трех уровней (но не менее двух):

1. Локальный адрес компьютера (физический или MAC-адрес)

   Локальный адрес компьютера. Для узлов, входящих в локальные сети - это MAC-адрес сетевого адаптера. Эти адреса назначаются производителями оборудования и являются уникальными адресами так как управляются централизовано, например, 11-A0-17-3D-BC-01. Для всех существующих технологий локальных сетей MAC-адрес имеет формат 6 байтов: старшие 3 байта - идентификатор фирмы производителя, а младшие 3 байта назначаются уникальным образом самим производителем. Для узлов, входящих в глобальные сети, такие как X.25 или frame relay, локальный адрес назначается администратором глобальной сети.

2. Сетевой (IP-адрес)

   IP-адрес, состоящий из 4 байт, например, 109.26.17.100. Этот адрес используется на сетевом уровне. Он назначается администратором во время конфигурирования компьютеров и маршрутизаторов. IP-адрес состоит из двух частей: номера сети и номера узла. Номер сети может быть выбран администратором произвольно, либо назначен по рекомендации специального подразделения Internet (Network Information Center, NIC), если сеть должна работать как составная часть Internet. Обычно провайдеры услуг Internet получают диапазоны адресов у подразделений NIC, а затем распределяют их между своими абонентами.

3. Символьный (DNS-имя)

   Символьный идентификатор-имя, (DNS) например, SERV1.IBM.COM или www.ssau.ru. Этот адрес назначается администратором и состоит из нескольких частей, например, имени машины, имени организации, имени домена. Такой адрес, называемый также DNS-именем, используется на прикладном уровне, например, в протоколах FTP или telnet

1.2. IP-адреса

IPv4 - адрес является уникальным 32-битным идентификатором IP-интерфейса в Интернет.

IPv6 - адрес является уникальным 128-битным идентификатором IP-интерфейса в Интернет, иногда называют **Internet-2,** адресного пространства IPv4 уже стало не хватать, поэтому постепенно вводят новый стандарт.

IP-адреса принято записывать разбивкой всего адреса по октетам (8), каждый октет записывается в виде десятичного числа, числа разделяются точками.

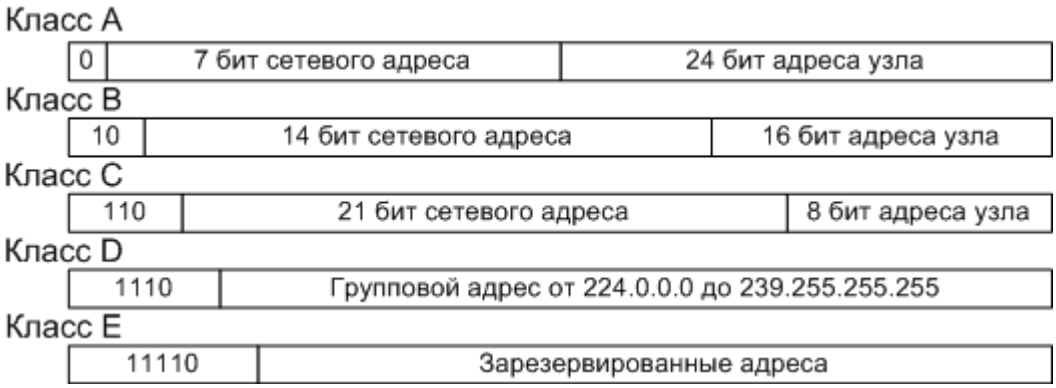
Например, адрес 10100000010100010000010110000011 записывается как 10100000.01010001.00000101.10000011 = 160.81.5.131

1.3. Основные классы IP-адресов

В самом начале развития сети было принято решение раздавать IP-адреса блоками, было создано три вида блока, и зарезервировано под эти блоки адреса:

Классы адресов

Класс	Длина сетевой части адреса в байтах	Первое число	Количество IP - адресов в блоке	Пример блока
A	1	0-127	16 777 216	122.0.0.0/255.0.0.0 или 122.0.0.0/8
B	2	128-191	65 536	152.126.0.0/255.255.0.0 или 122.126.0.0/16
C	3	192-223	256	83.149.236.0/255.255.255.0 или 122.149.236.0/24
Специальные адреса				
D	-	224-239	групповые адреса	
E	-	240-255	для экспериментальных целей	



В таблице приведены диапазоны номеров сетей, соответствующих каждому классу сетей.

Класс	Наименьший адрес	Наибольший адрес
A	01.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0

C	192.0.1.0.	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Как показала практика, такое распределение оказалось не эффективным. Самая большая потребность именно в сетях класса С, а не в В и А. Но классы В и А "съели" большую часть адресов, и их стало не хватать.

Поэтому была принята **бесклассовая** раздача адресов, которая позволяет, например, дать блок в несколько адресов, либо блок из 256 и еще 64 адресов.

Это пока спасает ситуацию до перехода на IPv6.

1.4. Маска подсети

Для того, что бы выделить четыре адреса в блок, используют **маску** подсети, например:

выделим адреса с 83.149.236.0 по 83.149.236.31 в отдельный блок, чтоб задать 32 адреса нужно 5 бит (2^5=32)

256-32=224

83.149.236.0/255.255.255.224 или 83.149.236.0/27 (32-5=27)

Почему называется маска?

01010011 10010101 11101100 00000000 - сеть 83.149.236.0

Наложим маску на адрес (логическое И, 1и1=1, 1и0=0, 0и0=0)

01010011 10010101 11101100 00010000 - проверяемый адрес 83.149.236.16

11111111 11111111 11111111 11100000 - маска 255.255.255.224

01010011 10010101 11101100 00000000 - получаем сеть 83.149.236.0 (в двоичном, см. выше)

Это был адрес, принадлежащий сети.

Возьмем адрес, не принадлежащий сети - 83.149.236.64

01010011 10010101 11101100 01000000 - проверяемый адрес 83.149.236.64

11111111 11111111 11111111 11100000 - маска 255.255.255.224

01010011 10010101 11101100 01000000 - не получаем сеть 83.149.236.0, значит адрес не принадлежит сети.

Адреса, зарезервированные для закрытых локальных сетей (в Интернете их не видно):

10.0.0.0/8

172.16.0.0 - 172.31.255.255

192.168.0.0 - 192.168.255.255

1.5. Соглашения о специальных адресах: broadcast, multicast, loopback

В протоколе IP существует несколько соглашений об особой интерпретации IP-адресов:

- если IP-адрес состоит только из двоичных нулей,

0 0 0 0 ..... 0 0 0 0

то он обозначает адрес того узла, который сгенерировал этот пакет;

- если в поле номера сети стоят 0,

0 0 0 0 0	Номер узла

то по умолчанию считается, что этот узел принадлежит той же самой сети, что и узел, который отправил пакет;

- если все двоичные разряды IP-адреса равны 1,

1 1 1 1	1 1

то пакет с таким адресом назначения должен рассылаться всем узлам, находящимся в той же сети, что и источник этого пакета. Такая рассылка называется ограниченным широковещательным сообщением (**limited broadcast**);

- если в поле адреса назначения стоят сплошные 1,

Номер сети	1111.....11

то пакет, имеющий такой адрес рассылается всем узлам сети с заданным номером. Такая рассылка называется широковещательным сообщением (**broadcast**);

Принято назначать широковещательным - последний адрес сети.

Например:

для сети 83.149.236.0/24
83.149.236.255 - broadcast адрес сети 83.149.236.0/24.
83.149.236.0 - адрес сети, unicast-адреса с таким номером быть не должно.
Т.е. минимальный размер подсети может быть в 4 адреса.

Кроме случая, когда используется маска 32, для указания одного unicast-адреса, например -83.149.236.36/32

- адрес 127.0.0.1 зарезервирован для организации обратной связи (**loopback**) при тестировании работы программного обеспечения узла без реальной отправки пакета по сети. Этому адресу по умолчанию назначают имя localhost.

Групповой адрес (multicast)

Предназначен для группы хостов.

Уже упоминавшаяся форма группового IP-адреса - **multicast** - означает, что данный пакет должен быть доставлен сразу нескольким узлам, которые образуют группу с номером, указанным в поле адреса. Узлы сами идентифицируют себя, то есть определяют, к какой из групп они относятся. Один и тот же узел может входить в несколько групп. Такие сообщения в отличие от широковещательных называются мультивещательными. Групповой адрес не делится на поля номера сети и узла и обрабатывается маршрутизатором особым образом.

В протоколе IP нет понятия широковещательности в том смысле, в котором оно используется в протоколах канального уровня локальных сетей, когда данные должны быть доставлены абсолютно всем узлам. Как ограниченный широковещательный IP-адрес, так и широковещательный IP-адрес имеют пределы распространения в интерсети - они ограничены либо сетью, к которой принадлежит узел - источник пакета, либо сетью, номер которой указан в адресе назначения. Поэтому деление сети с помощью маршрутизаторов на части локализует широковещательный шторм пределами одной из составляющих общую сеть частей просто потому, что нет способа адресовать пакет одновременно всем узлам всех сетей составной сети.

Например, адрес **224.0.0.5** - адрес OSPF - маршрутизаторов, т.е. все OSPF - маршрутизаторы обязаны принимать пакеты с адресом назначения **224.0.0.5.**

IP-адрес хоста состоит из номера IP-сети, который занимает старшую область адреса, и номера хоста в этой сети, который занимает младшую часть.

160.81.5.131 - IP-адрес 160.81.5. - номер сети 131 - номер хоста

255.255.255.255 - широковещательный адрес, для всех сетей. Используется для DHCP.

2. СЛУЖБА DNS

2.1. Назначение службы DNS

Числовая IP-адресация не неудобна для человека. Запомнить наборы цифр гораздо труднее, чем слова. Для облегчения стали использовать соответствия числовых адресов именам машин.

Например, для нашего сервера существуют следующие соответствия:

127.0.0.1 localhost
91.221.128.17 ssau.ru
91.221.128.17 www.ssau.ru

Длина имени не более 63 символов

Сначала такие соответствия просто сами записывали в файл, брали у знакомых и копировали с FTP-серверов.

Это файлы имеет название hosts, и находится в каталогах:

Для UNIX - /etc/hosts
Для Windows - C:\windows\system32\drivers\etc\hosts

Можете сами изменить его, например, внеся запись:

91.221.128.17 ssau

Теперь вы можете, просто набрав в браузере ipm попасть на сервер www.ipm.kstu.ru. Таким образом, вы можете записать любой сервер.

Однако такой способ присвоения символьных имен был хорош до тех пор, пока Internet был маленьким. По мере роста сети стало затруднительным поддерживать большие списки имен на каждом компьютере. Для того, что бы решить эту проблему, были придумана служба DNS (Domain Name System).

2.2. Принципы организации DNS

Первый стандарт DNS определен в RFC0883 (Domain names: Implementation specification P.V. Mockapetris Nov-01-1983) и RFC0882 (Domain names: Concepts and facilities P.V. Mockapetris Nov-01-1983)

Последняя версия RFC1034 (Domain names - concepts and facilities P.V. Mockapetris Nov-01-1987) и RFC1035 (Domain names - implementation and specification P.V. Mockapetris Nov-01-1987)

Система доменных адресов строится по иерархическому принципу. Администрирование начинается с доменов верхнего, или первого, уровня.

Первые домены верхнего уровня были рассчитаны на США:

- gov - государственные организации
- mil - военне учреждения
- edu - образовательные учреждения
- com - коммерческие организации
- net - сетевые организации

Позднее, когда сеть перешагнула национальные границы США появились национальные домены типа:

- uk - Объединенное королевство
- jp - Япония
- su - СССР ru - Россия и т.п.

IANA - The Internet Assigned Numbers Authority (Управление назначением адресов в Internet) -организация, осуществляющая контроль над распределением доменов первого уровня. Сервер http://www.iana.org/ .

Базу можно посмотреть по адресу http://whois.iana.org .

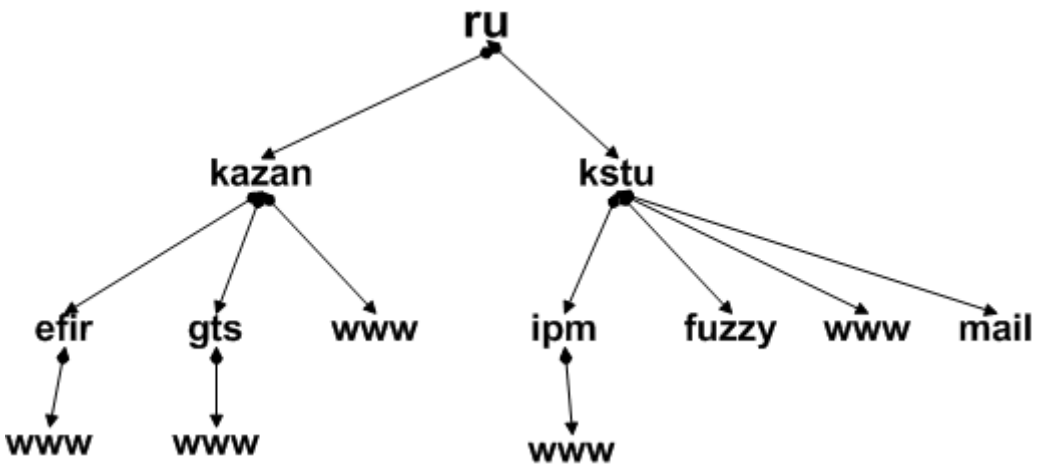
Через WWW-интерфейс http://whois.iana.org/

Вслед за доменами первого уровня следуют домены, либо географические (kazan.ru, tatarstan.ru), либо организации (kstu.ru). В настоящее время практически любая организация или физическое лицо может получить свой собственный домен второго уровня (сервер РосНИИРОС - www.ripn.net).

Далее идут домены третьего уровня, например :

- gosuslugi.samara.ru
- pogoda.63.ru - домены третьего уровня.

Систему доменной адресации можно представить следующим образом:



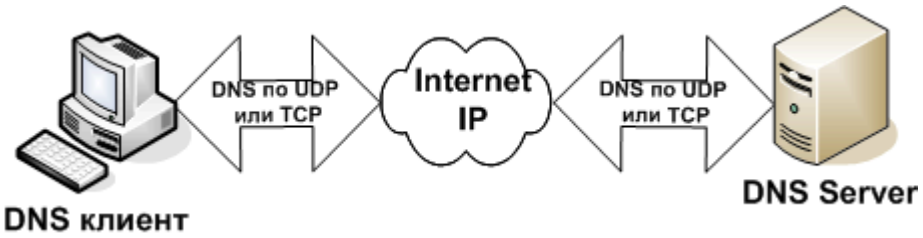
Дерево доменных имен.

Служба доменных имен работает как распределенная база, данные которой распределены по DNS-серверам.

Система доменных имен - это сервис прикладного уровня, значит, использует транспорт **TCP** и **UDP**.

Порт по умолчанию - 53.

Сервис DNS строится по схеме "клиент-сервер". В качестве клиентской части выступает процедура разрешения имен - **resolver,** а в качестве сервера DNS-сервер (**BIND ...**).

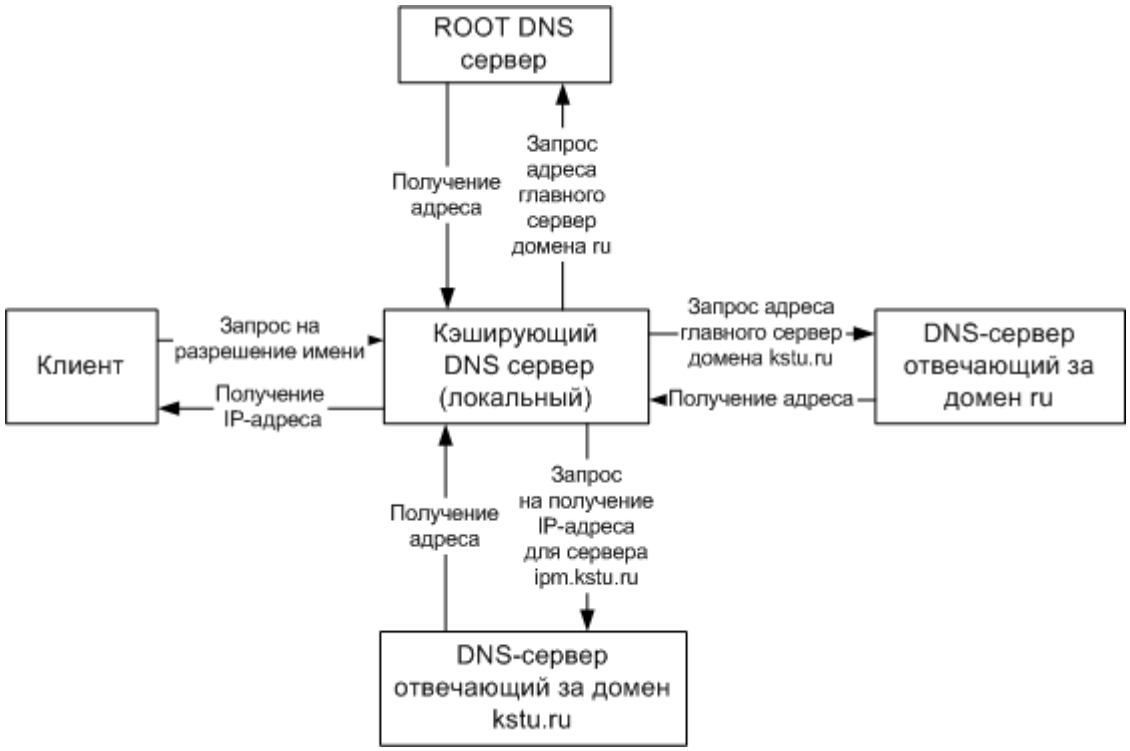


Взаимодействие клиент и сервера по протоколу DNS.

Например, когда мы хотим обратиться к серверу ipm.kstu.ru, ваш браузер, используя **resolver**, поступает следующим образом:

1. ищет запись ssau.ru в файле hosts, если не находит, то,
2. посылает запрос на известный DNS-кэширующий сервер (как правило, локальный), если на этом сервере запись не найдена, то,
3. сервер DNS-кэширующий обращается к DNS-ROOT серверу с запросом адреса DNS сервера отвечающего за домен первого уровня ru, если получает адрес, то,
4. сервер DNS-кэширующий обращается к DNS серверу, отвечающего за домен первого уровня ru, с запросом адреса DNS сервера отвечающего за домен второго уровня ssau.ru, если получает адрес, то,
5. сервер DNS-кэширующий посылает запрос на DNS сервер, отвечающий за домен второго уровня ssau.ru, если получает адрес, то,
6. сервер DNS-кэширующий адрес кэширует и передает клиенту
7. клиент обращается по IP адресу - 91.221.128.17

На схеме это выглядит так:



Алгоритм разрешения имен.

2.3. Некоторые типы DNS-серверов

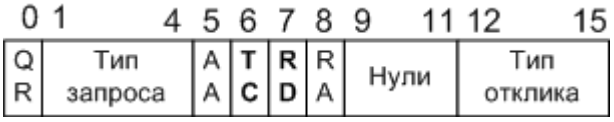
Первичный - сервер, содержащий полную информацию о зоне.

Вторичный - сервер, содержащий копию полной информации о зоне, полученную с первичного сервера.

Кэширующий - содержит записи, которые уже были запрошены

2.4. Формат DNS-сообщения

Формат DNS-сообщения. Слова по 32 бита.



Флаги DNS-сообщения (подробно поле флаги)

QR - Операция:
0 Запрос
1 Отклик

Тип запроса:
0 стандартный
1 инверсный
2 запрос состояния сервера

AA:
Равен 1 при ответе от сервера, в ведении которого находится домен, упомянутый в запросе.
TC:
Равен при укорочении сообщения. Для UDP это означает, что ответ содержал более 512 байт, но прислано только первые 512.
RD:
Равен 1, если для получения ответа желательна рекурсия.
RA:
Равен 1, если рекурсия для запрашиваемого сервера доступна.
Нули - Зарезервировано на будущее..

Тип отклика
0 нет ошибки
1 ошибка в формате запроса
2 сбой в сервере
3 имени не существует

2.5. Некоторые виды записей в DNS

SOA (Start Of Authorisation) - Начало Полномочий.

Запись SOA, обозначает начало зоны. Для каждой зоны должна быть только одна запись SOA.

Синтаксис:
[name] [ttl] addr-class SOA Origin Person in charge

Пример для зоны kstu.ru:
kstu.ru IN SOA ns.kstu.ru. admin.kstu.ru. (19970315 ; Serial
3600 ; Refresh 300 ; Retry 3600000 ; Expire 3600) ; Minimum

Пояснения:
kstu.ru ({name}) - имя зоны

IN (addr-class) - Запись относится к InterNet. SOA - вид записи.
ns.kstu.ru (Origin) - первичный сервер зоны.
admin.kstu.ru. (Person in charge) - ответственный за зону. Почтовый адрес лица (admin@kstu.ru), ответственного за зону.

В скобки заключаются параметры, растянутые на несколько строк: 19970315 ; Serial - серийный номер версии; должен увеличиваться при каждом изменении в зоне - по нему вторичный сервер обнаруживает, что надо обновить информацию. Обычно пишется в виде <год><месяц><число><номер>.

3600 ; Refresh - временной интервал в секундах, через который вторичный сервер будет проверять необходимость обновления информации.

300 ; Retry - временной интервал в секундах, через который вторичный сервер будет повторять обращения при неудаче.

3600000 ; Expire - временной интервал в секундах, через который вторичный сервер будет считать имеющуюся у него информацию устаревшей.

3600) ; Minimum - значение времени жизни информации на кэширующих серверах ((ttl) в последующих записях ресурсов).

NS - Сервер Имен

Описывает вторичные DNS-сервера или DNS-сервер которому делегируется подзона.

Синтаксис:
{name} {ttl} addr-class NS Name-servers-name

Пример описания вторичных DNS-серверов:
IN NS ns1.kstu.ru.
IN NS ns2.kstu.ru.
IN NS ns3.kstu.ru.

Пример описания DNS-сервера (ns.ipm.kstu.ru), которому делегируется подзона ipm.kstu.ru: ipm.kstu.ru. IN NS ns.ipm.kstu.ru.

A - Адрес

Описывает имя указанного IP-адреса

Синтаксис:
{name} {ttl} addr-class A address

Пример:
ipm.kstu.ru. IN A 195.208.44.20 www.ipm.kstu.ru. IN A 195.208.44.20

HINFO - Информация о Хосте

Содержит некоторую информацию о машине, обычно - тип процессора и операционной системы.

Синтаксис:
{name} {ttl} addr-class HINFO Hardware OS
Hardware - тип процессора
OS - операционная система

Пример:
ipm.kstu.ru. IN HINFO "Celeron-600" "RedHat linux"

CNAME (Canonical Name) - Каноническое имя

Указывает псевдоним для официального имени хоста

Синтаксис:
alias {ttl} addr-class CNAME Canonical-name

Пример псевдонимов для хоста ipm.kstu.ru:
www.ipm.kstu.ru. IN CNAME ipm.kstu.ru.
fuzzy.kstu.ru. IN CNAME ipm.kstu.ru.
www.fuzzy.kstu.ru. IN CNAME ipm.kstu.ru.

MX (Mail Exchange) - Почтовый Коммутатор

Записи используются для обозначения списка хостов, которые сконфигурированы для приема почты посланной на это доменное имя.

Синтаксис:
name {ttl} addr-class MX preference value mail exchange

preference value - значение приоритета, более низкие числа показывают более высокий приоритет, а приоритеты одинаковые отправители должны использовать в произвольном порядке хосты MX для равномерного распределения нагрузки.

Пример:
mail.kstu.ru. IN MX 0 mail1.kstu.ru.
mail.kstu.ru. IN MX 20 mail2.kstu.ru.
mail.kstu.ru. IN MX 10 mail3.kstu.ru.
0, 10, 20 - указывают приоритет для отправки почты.

Если mail1.kstu.ru не доступен, то почта посылается на mail3.kstu.ru, если и он не доступен то на mail2.kstu.ru.

TXT - Текст

Запись TXT содержит текстовые данные любого вида.

Синтаксис:
name {ttl} addr-class TXT string

Пример:
ipm.kstu.ru. IN TXT "Текстовая информация"

RP - Ответственная Персона

Ответственный за хост. Почтовый адрес лица (admin@ipm.kstu.ru), ответственного за хост.

Синтаксис:
owner {ttl} addr-class RP mbox-domain-name TXT-domain-name
mbox-domain-name - это доменное имя, определяющее почтовый ящик ответственного человека
TXT-domain-name - это имя домена, для которого существует запись TXT

Пример:
www.ipm.ru. IN RP admin.ipm.kstu.ru. ipm.kstu.ru.
ipm.kstu.ru. IN TXT "tel: 34-54-34, fax: 34-54-34"

Пояснение:
www.ipm.ru. - хост за который он отвечает admin.ipm.kstu.ru - его e-mail (admin@ipm.kstu.ru)
ipm.kstu.ru - это имя записи TXT, которая может содержать телефоны этого лица.

Использованные источники:

1. Храмцов П. Б. Администрирование сети и сервисов Internet. Учебное пособие., Центр Информационных Технологий, 1997

2. Семенов Ю.А. Телекоммуникационные технологии. (v4.11, 17 мая 2013 года): http://book.itep.ru/

1. СТЕК ПРОТОКОЛОВ TCP/IP

1.1. Семиуровневая модель сетевого обмена OSI

При рассмотрении процедур межсетевого взаимодействия всегда опираются на стандарты, разработанные International Standard Organization (ISO). Эти стандарты получили название "Семиуровневой модели сетевого обмена" или в английском варианте "Open System Interconnection Reference Model" (OSI Ref.Model). В данной модели обмен информацией может быть представлен в виде стека, представленного на рисунке ниже. Как видно из рисунка, в этой модели определяется все - от стандарта физического соединения сетей до протоколов обмена прикладного программного обеспечения. Дадим некоторые комментарии к этой модели.

Семиуровневая модель протоколов межсетевого обмена OSI

Физический уровень данной модели определяет характеристики физической сети передачи данных, которая используется для межсетевого обмена. Это такие параметры, как: напряжение в сети, сила тока, число контактов на разъемах и т.п. Типичными стандартами этого уровня являются, например RS232C, V35, IEEE 802.3 и т.п.

К канальному уровню отнесены протоколы, определяющие соединение, например, SLIP (Strial Line Internet Protocol), PPP (Point to Point Protocol), NDIS, пакетный протокол, ODI и т.п. В данном случае речь идет о протоколе взаимодействия между драйверами устройств и устройствами, с одной стороны, а с другой стороны, между операционной системой и драйверами устройства. Такое определение основывается на том, что драйвер - это, фактически, конвертор данных из оного формата в другой, но при этом он может иметь и свой внутренний формат данных.

К сетевому (межсетевому) уровню относятся протоколы, которые отвечают за отправку и получение данных, или, другими словами, за соединение отправителя и получателя. Вообще говоря, эта терминология пошла от сетей коммутации каналов, когда отправитель и получатель действительно соединяются на время работы каналом связи. Применительно к сетям TCP/IP, такая терминология не очень приемлема. К этому уровню в TCP/IP относят протокол IP (Internet Protocol). Именно здесь определяется отправитель и получатель, именно здесь находится необходимая информация для доставки пакета по сети.

Транспортный уровень отвечает за надежность доставки данных, и здесь, проверяя контрольные суммы, принимается решение о сборке сообщения в одно целое. В Internet транспортный уровень представлен двумя протоколами TCP (Transport Control Protocol) и UDP (User Datagramm Protocol). Если предыдущий уровень (сетевой) определяет только правила доставки информации, то транспортный уровень отвечает за целостность доставляемых данных.

Уровень сессии определяет стандарты взаимодействия между собой прикладного программного обеспечения. Это может быть некоторый промежуточный стандарт данных или правила обработки информации. Условно к этому уровню можно отнеси механизм портов протоколов TCP и UDP и Berkeley Sockets. Однако обычно, рамках архитектуры TCP/IP такого подразделения не делают.

Уровень обмена данными с прикладными программами (Presentation Layer) необходим для преобразования данных из промежуточного формата сессии в формат данных приложения. В Internet это преобразование возложено на прикладные программы.

Уровень прикладных программ или приложений определяет протоколы обмена данными этих прикладных программ. В Internet к этому уровню могут быть отнесены такие протоколы, как: FTP, TELNET, HTTP, GOPHER и т.п.

1.2. Структура стека протоколов TCP/IP

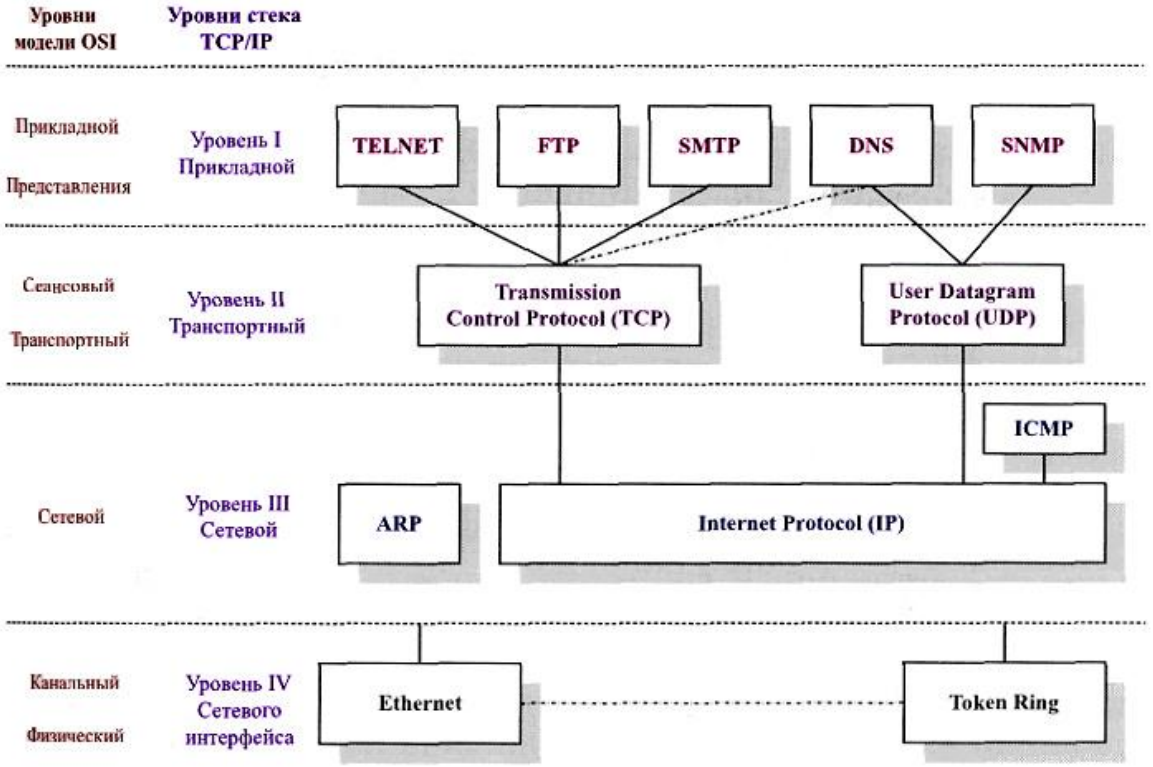
TCP/IP - собирательное название для набора (стека) сетевых протоколов разных уровней, используемых в Интернет.

Особенности TCP/IP:

- Открытые стандарты протоколов, разрабатываемые независимо от программного и аппаратного обеспечения;
- Независимость от физической среды передачи;
- Система уникальной адресации;
- Стандартизованные протоколы высокого уровня для распространенных пользовательских сервисов.

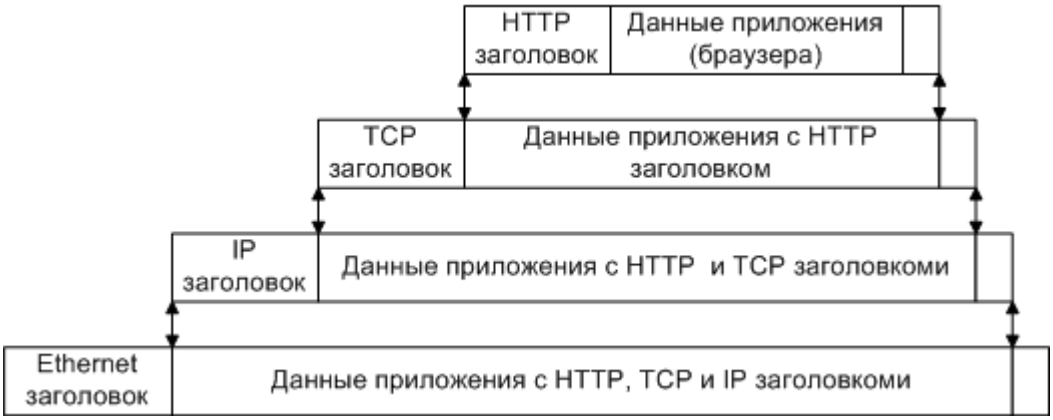
Базируясь на классификации OSI (Open System Integration) всю архитектуру протоколов семейства TCP/IP попробуем сопоставить с эталонной моделью:

Так как стек протоколов TCP/IP был разработан до появления эталонной модели OSI, то соответствие его уровней уровням модели OSI достаточно условно. Структура стека протоколов TCP/IP приведена на рис.



Структура стека протоколов TCP/IP.

Данные передаются в пакетах. Пакеты имеют заголовок и окончание, которые содержат служебную информацию. Данные, более верхних уровней вставляются, в пакеты нижних уровней.



Пример инкапсуляции пакетов в стеке TCP/IP

Самый нижний — уровень сетевого интерфейса (уровень IV) - соответствует физическому и канальному уровню модели OSI. В стеке протоколов TCP/IP этот уровень не регламентирован. Он поддерживает стандарты физического

и канального уровня популярных локальных сетей: Ethernet, Token Pang, FDDI и т.д. Для распределенных сетей поддерживаются проколы соединений PPP и SLIP, а для глобальных сетей - протокол X.25.. Обычной практикой стало включение в стек протоколов TCP/IP новых технологий локальных или распределенных сетей и регламентация их новыми документами RFC.

Сетевой уровень (уровень III) — это уровень межсетевого взаимодействия. Уровень управляет взаимодействием между пользователями в сети. Он принимает от транспортного уровня запрос на посылку пакета от отправителя вместе с указанием адреса получателя. Уровень инкапсулирует пакет в дейтаграмму, заполняет ее заголовок и при необходимости использует алгоритм маршрутизации. Уровень обрабатывает приходящие дейтаграммы и проверяет правильность поступившей информации. На стороне получателя программное обеспечение сетевого уровня удаляет заголовок и определяет, какой из транспортных протоколов будет обрабатывать пакет.

В качестве основного протокола сетевого уровня в стеке TCP/IP используется протокол IP, который и создавался с целью передачи информации в распределенных сетях.

К этому уровню относятся все протоколы, которые создают, поддерживают и обновляют таблицы маршрутизации. Кроме того, на этом уровне функционирует протокол обмена информацией об ошибках между маршрутизаторами в сети и отправителями.

Следующий уровень — **транспортный (уровень II).** Транспортный уровень управляет потоком информации с обеспечением надежной передачи. Для этого использован механизм подтверждения правильного приема с дублированием передачи утерянных или пришедших с ошибками пакетов. Транспортный уровень принимает данные от нескольких прикладных программ и посылает их более низкому уровню. При этом он добавляет дополнительную информацию к каждому пакету, в том числе и значение вычисленной контрольной суммы.

На этом уровне функционирует протокол управления передачей данных TCP (Transmission Control Protocol) и протокол передачи прикладных пакетов дейтаграммным методом UDP (User Datagram Protocol.

Самый верхний уровень (уровень I) — прикладной. На нем реализованы широко используемые сервисы прикладного уровня. К ним относятся: протокол передачи файлов между удаленными системами, протокол эмуляции удаленного терминала, почтовые протоколы и т.д. Прикладная программа передает данные транспортному уровню в требуемой форме.

Рассмотрение принципов функционирования стека протоколов TCP/IP целесообразно проводить, начиная с протоколов третьего уровня.

Для понимания проблем маршрутизации в распределенных сетях изучение протоколов рекомендуется проводить в следующей последовательности: IP, ARP, ICMP, UDP и TCP. Это связано с тем, что для доставки информации между

удаленными системами в распределенной сети используется в той или иной степени все семейство стека протоколов TCP/IP.

Стек протоколов TCP/IP включает в свой состав большое число протоколов прикладного уровня. Эти протоколы выполняют различные функции, в том числе: управление сетью, передачу файлов, оказание распределенных услуг при использовании файлов, эмуляцию терминалов, доставку электронной почты и т.д. Протокол передачи файлов (File Transfer Protocol — FTP) обеспечивает перемещение файлов между компьютерными системами. Протокол Telnet обеспечивает виртуальную терминальную эмуляцию. Простой протокол передачи почты (Simple Mail Transfer Protocol — SMTP) обеспечивает механизм передачи электронной почты. Эти протоколы и другие приложения используют услуги стека TCP/IP для обеспечения пользователей базовыми сетевыми услугами.

1.3. Уровень IV – сетевого интерфейса

Стек TCP/IP не подразумевает использования каких-либо определенных протоколов уровня доступа к среде передачи и физических сред передачи данных. От уровня доступа к среде передачи требуется наличие интерфейса с модулем IP, обеспечивающего передачу IP-пакетов. Также требуется обеспечить преобразование IP-адреса узла сети, на который передается IP-пакет, в MAC-адрес. Часто в качестве уровня доступа к среде передачи могут выступать целые протокольные стеки, тогда говорят об IP поверх ATM, IP поверх IPX, IP поверх X.25 и т.п.

1.4. Сетевой уровень и протокол IP

Первый стандарт IPv4 определен в RFC-760 (DoD standard Internet Protocol J. Postel Jan-01-1980)

Последняя версия IPv4 - RFC-791 (Internet Protocol J. Postel Sep-01-1981).

Первый стандарт IPv6 определен в RFC-1883 (Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden December 1995)

Последняя версия IPv6 - RFC-2460 (Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden December 1998).

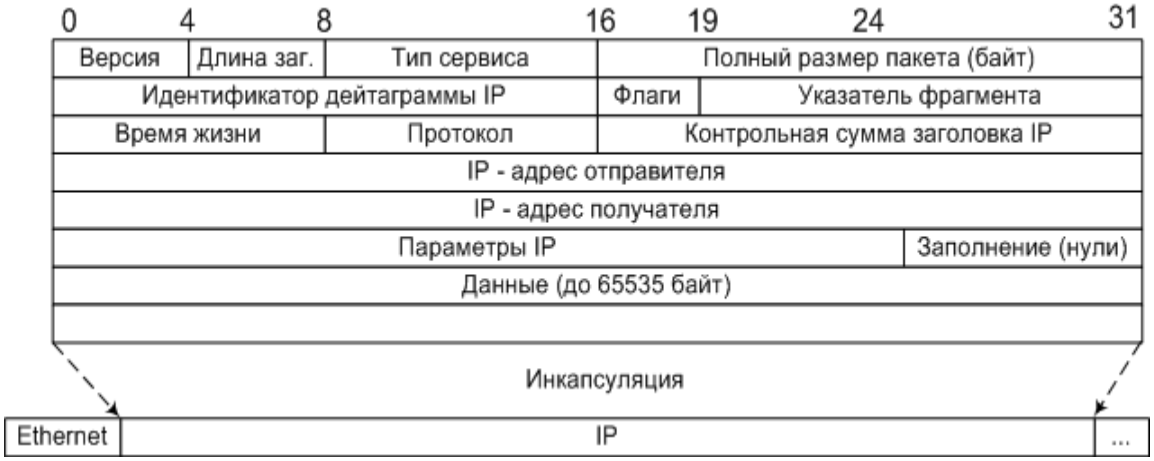
Основные задачи:

Адресация
Маршрутизация
Фрагментация дейтаграмм
Передача данных

Протокол IP доставляет блоки данных от одного IP-адреса к другому.

Программа, реализующая функции того или иного протокола, часто

называется модулем, например, “IP-модуль”, “модуль TCP”.

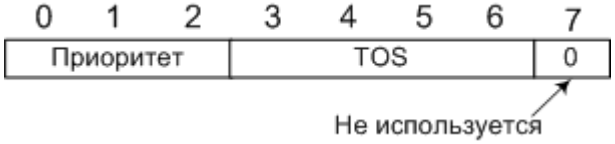


Структура дейтограммы IP. Слова по 32 бита.

Версия - версия протокола IP (например, 4 или 6)

Длина заг. - длина заголовка IP-пакета.

Тип сервиса (TOS - type of service) - это поле делится на шесть подполей.



Подробно поле "Тип сервиса"

Приоритет - присваивает код приоритета каждой дейтограмме.

Значения приоритетов:

111 - Network Control - Сетевое управление
110 - Internetwork Control - Межсетевое управление 101 - CRITIC/ECP - critic/ecp
100 - Flash Override - Экстренный 011 - Flash – Срочный
010 - Immediate - Немедленный 001 - Priority – Приоритетный
000 - Routine - Обычный уровень

Формат поля TOS определен в документе RFC-1349.

Коды типа сервиса (TOS)

TOS-код	**Значения** (RFC-1349)	**Значения (перевод)**
0000	normal service	Обычный сервис
0001	minimize monetary cost	Минимальная денежная стоимости

0010	maximize reliability	Максимальная надежность
0100	maximize throughput	Максимальная пропускная способность
1000	minimize delay	Минимальная задержка

TOS играет важную роль в маршрутизации пакетов. Интернет не гарантирует запрашиваемый TOS, но многие маршрутизаторы учитывают эти запросы при выборе маршрута (протоколы OSPF и IGRP).

Идентификатор дейтаграммы, флаги (3 бита) и указатель фрагмента - используются для распознавания пакетов, образовавшихся путем фрагментации исходного пакета.

Поле флаги состоит из 3-х бит:

Bit 0: reserved, must be zero - зарезервирован, равен нулю.
Bit 1: 0 = May Fragment, 1 = Don't Fragment. - разрешена/запрещена фрагментация.
Bit 2: 0 = Last Fragment, 1 = More Fragments - является ли данный фрагмент последним, последний/будут еще.

Время жизни (TTL - time to live) - каждый маршрутизатор уменьшает его на 1, что бы пакеты не блуждали вечно.

Протокол - Идентификатор протокола верхнего уровня указывает, какому протоколу верхнего уровня принадлежит пакет (например: TCP, UDP).

1.4.1. Статическая маршрутизация

Для маршрутизации нужна маршрутная информация, куда (на какой интерфейс) пакет отправлять, зная адрес назначения.

Статическая маршрутизация - это когда таблица маршрутизации формируется "вручную".

1.4.2. Таблицы маршрутизации

Для получения таблицы маршрутизации используется команда route.

Рассмотрим таблицы маршрутизации самой простой сети.

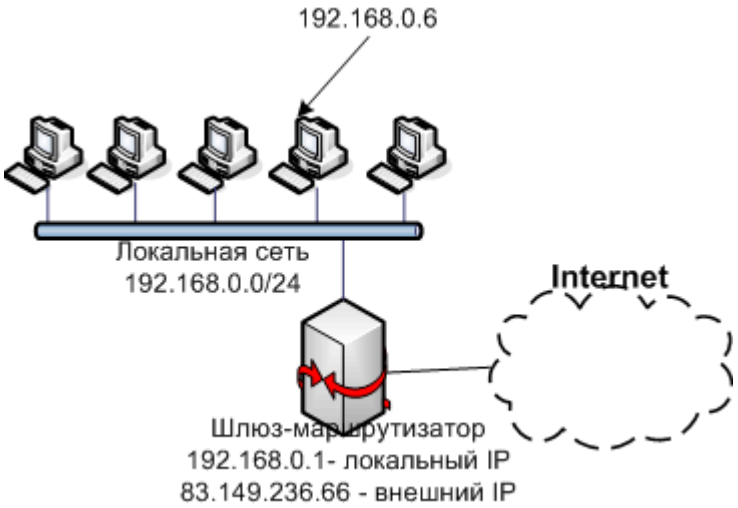


Схема сети

Столбцы таблицы маршрутизации:

Destination - адрес сети назначения Gateway - адрес шлюза
Genmask - маска сети назначения Flags – флаги

U - показывает, что маршрут активен.

G - показывает, что маршрут проходит через промежуточный маршрутизатор (Gateway).

H - специфический маршрут, маршрут к этому хосту отличается от маршрута ко всей этой сети.

Metric - метрика, если для отправки можно использовать несколько маршрутов, позволяет делать выбор.
Метрика =0 обозначает, что эта сеть непосредственно подключена к данному интерфейсу.
Ref - сколько раз ссылались на данный маршрут при обработке пакетов Use - количество пакетов, переданное по данному маршруту.
Iface - интерфейс для отправки пакетов.

```
# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.0.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.0.1 0.0.0.0 UG 0 0 0 eth0
#
```

Таблица (linux) маршрутизации на 192.168.0.6

В таблице прописаны два правила:

1. Пакеты для сети 192.168.0.0/255.255.255.0 посылать напрямую, т.е. не через маршрутизатор.

2. Пакеты для всех остальных сетей посылать на default-маршрутизатор с адресом 192.168.0.1.

```
# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.0.0 * 255.255.255.0 U 0 0 0 eth1
83.149.236.64 * 255.255.255.224 U 0 0 0 eth0
default 83.149.236.92 0.0.0.0 UG 0 0 0 eth0
#
```

Таблица (linux) маршрутизации на 192.168.0.1

В таблице прописаны три правила:

1. Пакеты для сети 192.168.0.0/255.255.255.0 посылать напрямую, через интерфейс eth1.

2. Пакеты для сети 83.149.236.64/255.255.255.224 посылать напрямую, через интерфейс eth0.

3. Пакеты для всех остальных сетей посылать на default-маршрутизатор с адресом 83.149.236.92, через интерфейс eth0.

В разных системах таблицы немного отличаются, но в целом очень похожи.

```
>route PRINT
===========================================================================
Активные маршруты:
Сетевой адрес Маска сети Адрес шлюза Интерфейс Метрика
0.0.0.0 0.0.0.0 10.85.241.18 10.85.241.63 1
10.85.240.0 255.255.254.0 10.85.241.63 10.85.241.63 20
10.85.241.63 255.255.255.255 127.0.0.1 127.0.0.1 20
10.255.255.255 255.255.255.255 10.85.241.63 10.85.241.63 20
127.0.0.0 255.0.0.0 127.0.0.1 127.0.0.1 1
224.0.0.0 240.0.0.0 10.85.241.63 10.85.241.63 20
255.255.255.255 255.255.255.255 10.85.241.63 10.85.241.63 1
Основной шлюз: 10.85.241.18
===========================================================================
```

Пример таблицы маршрутизации в Windows

Источники информации для таблицы маршрутизации:

Ручная настройка сетевых интерфейсов.

Ручная настройка default маршрутизатора. Ручная настройка настройка маршрутов. Динамические протоколы - RIP, OSPF и т.д.

1.4.3. Маршрутизация без маски (на классах)

```
# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.0.0 * 255.255.255.0 U 0 0 0 eth0
192.168.1.0 * 255.255.255.0 U 0 0 0 eth0
192.168.2.0 * 255.255.255.0 U 0 0 0 eth0
192.168.3.0 * 255.255.255.0 U 0 0 0 eth0
192.168.4.0 * 255.255.255.0 U 0 0 0 eth0
192.168.5.0 * 255.255.255.0 U 0 0 0 eth0
192.168.6.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.0.1 0.0.0.0 UG 0 0 0 eth0
#
```

Таблица (linux) маршрутизации без использования маски

1.4.4. Маршрутизация с маской (CIDR).

Это позволяет создавать непрерывное адресное пространство, и маршрут к нему, что уменьшает записей в таблице.

```
# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.0.0 * 255.255.248.0 U 0 0 0 eth0
default 192.168.0.1 0.0.0.0 UG 0 0 0 eth0
#
```

Таже таблица (linux) маршрутизации, но с использованием маски.

1.4.5. Протокол ICMP

ICMP (Internet Control Message Protocol) - расширение протокола IP, позволяет передавать сообщения об ошибке или проверочные сообщения.

1.4.6. Другие служебные IP-протоколы

IGMP (Internet Group Management Protocol) - позволяет организовать многоадресную рассылку средствами IP.

RSVP (Resource Reservation Protocol) - протокол резервирования ресурсов.

ARP (Address Resolution Protocol) - протокол преобразования IP-адреса и адреса канального уровня.

1.5. Протокол надежной доставки сообщений TCP

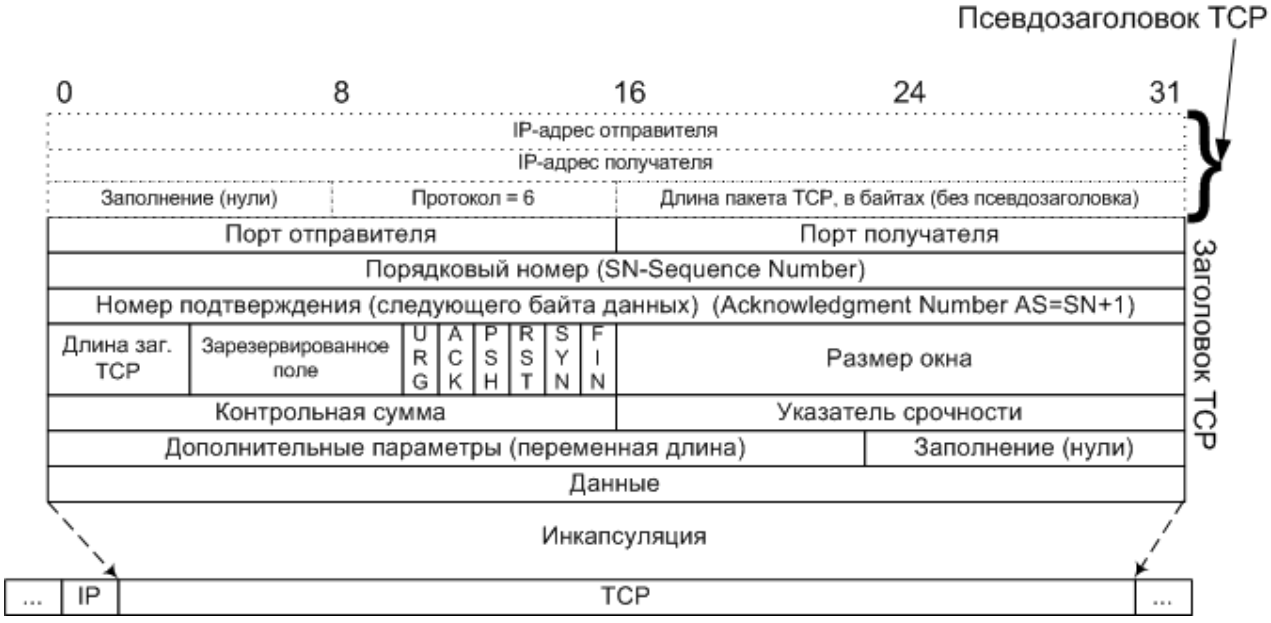
TCP (Transfer Control Protocol) – протокол контроля передачи, протокол TCP применяется в тех случаях, когда требуется гарантированная доставка сообщений.

Первая и последняя версия TCP - RFC-793 (Transmission Control Protocol J. Postel Sep-01-1981).

Основные особенности:

- Устанавливается соединение.
- Данные передаются **сегментами**. Модуль TCP нарезает большие сообщения (файлы) на пакеты, каждый из которых передается отдельно, на приемнике наоборот файлы собираются. Для этого нужен **порядковый номер (Sequence Number - SN)** пакета.
- Посылает запрос на следующий пакет, указывая его номер в поле **"Номер подтверждения" (AS).** Тем самым, подтверждая получение предыдущего пакета.
- Делает проверку целостности данных, если пакет битый посылает повторный запрос.

.



Структура дейтограммы TCP. Слова по 32 бита.

Длина заголовка - задается словами по 32бита.

Размер окна - количество байт, которые готов принять получатель без подтверждения.

Контрольная сумма - включает псевдо заголовок, заголовок и данные.

Указатель срочности - указывает последний байт срочных данных, на которые надо немедленно реагировать.

URG - флаг срочности, включает поле "Указатель срочности", если =0 то поле игнорируется. **ACK -** флаг подтверждение, включает поле "Номер подтверждения, если =0 то поле

игнорируется.

PSH - флаг требует выполнения операции push, модуль TCP должен срочно передать пакет программе.

RST - флаг прерывания соединения, используется для отказа в соединении

SYN - флаг синхронизация порядковых номеров, используется при установлении соединения. **FIN -** флаг окончание передачи со стороны отправителя

В дополнительных параметрах, может быть:

MSS (maximum segment size) - максимальный размер сегмента. Если MSS не задан, то устанавливается, по умолчанию, 536 байт. С помощью этого параметра можно увеличить скорость передачи, подбирая оптимальные MSS и MTU (размер пакета канального уровня).

1.5.1. Понятие сокета

Чтобы идентифицировать отдельные потоки данных введено понятие **сокет(socket)** - это уникальное число для каждого процесса, для получения числа используется IP-адрес хоста и порт приложения

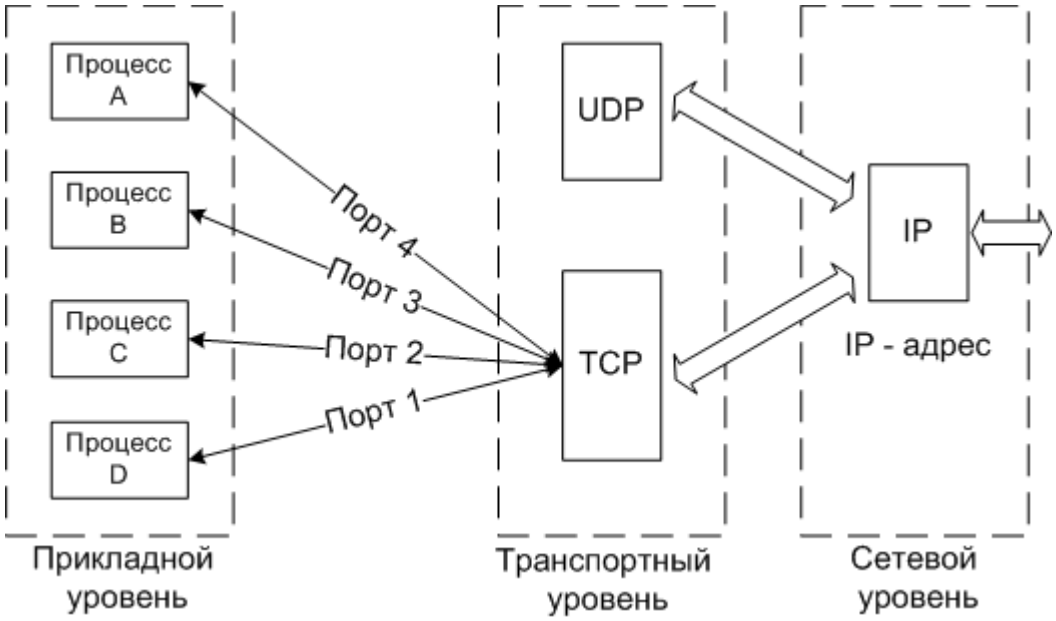
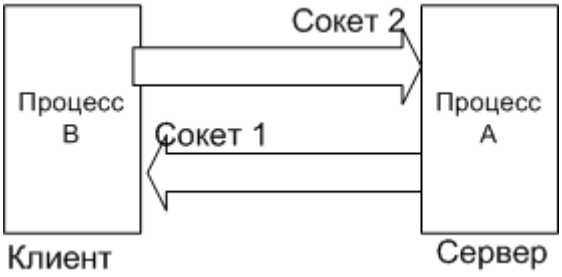


Схема мультиплексирования информационных потоков

6.4.2 Соединения TCP:

Соединение TCP является полнодуплексным, т.е. существуют два потока, прямой и обратный.

Поэтому каждое соединение определяется двумя сокетами. Локальный сокет может принимать участие во многих соединениях с различными чужими сокетами.



Полнодуплексное соединение, используется 2 сокета.

Все необходимые переменные для соединения хранятся в блоке управления передачей **TCB** (Transmission Control Block).

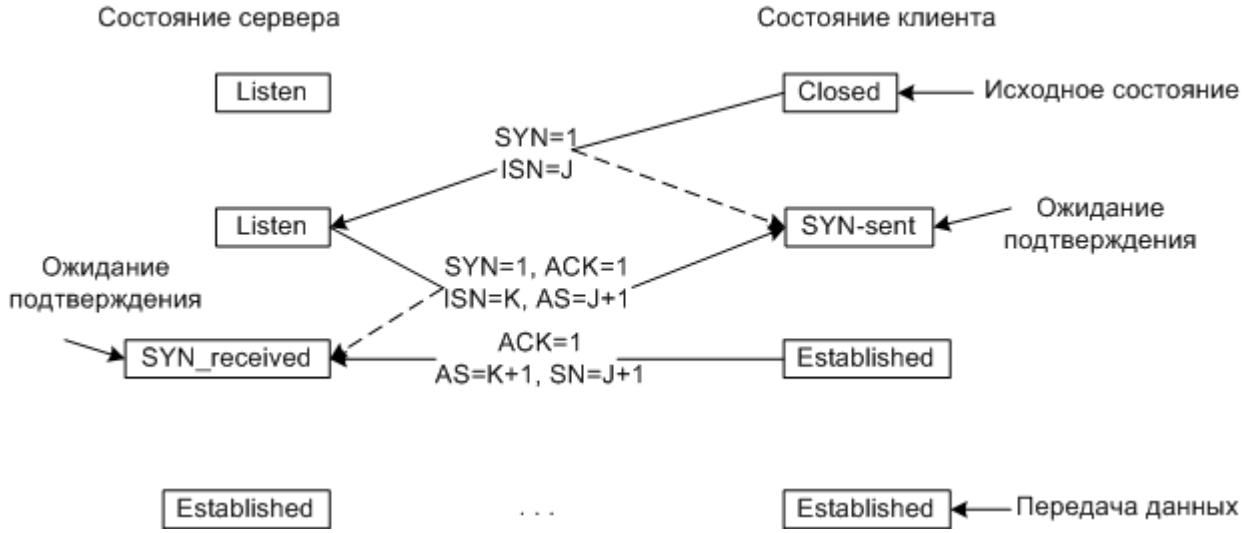
В TCB могут содержаться:

сокеты
номера очередей
флаги безопасности и приоритета и т.д.

Состояния TCP соединения:

LISTEN - ожидает запроса на соединение от удаленного TCP-модуля, чужой сокет равен нулям.
SYN-SENT - ожидание ответа на соединение, после посылки запроса на

соединение.
SYN-RECEIVED - ожидание подтверждения на соединение, после посылки обоих предыдущих запросов.
ESTABLISHED - состояние установленного соединения, стадия передачи данных.
FIN-WAIT-1 - ожидание запроса завершения соединения от удаленного TCP-модуля, или подтверждения запроса завершения соединения, предварительно посланного.
FIN-WAIT-2 - ожидание запроса завершения соединения от удаленного TCP-модуля. **CLOSE-WAIT** - ожидание запроса завершения соединения от локального пользователя.. **CLOSING** - ожидание подтверждения запроса завершения соединения от удаленного TCP-модуля.
LAST-ACK - ожидание подтверждения запроса завершения соединения, предварительно посланного удаленному TCP-модулю (который включает подтверждение его запроса завершения соединения).
TIME-WAIT - время ожидания, что удаленный TCP-модуль получил подтверждение его запроса завершения соединения.
CLOSED - TCP-модуль закрыт для любого подключения.



Алгоритм установки TCP-соединения

Алгоритм установки соединения:

Клиент посылает SYN-сегмент (SYN=1), и порядковый номер ISN=J (initial sequence number - **первоначальный порядковый номер**). Раньше ISN просто приравнивали 0, но сейчас, как правило, это случайное число, это сделано, что бы усложнить атаки с помощью подмены IP-адреса и исключить попадания пакетов с одинаковыми номерами. Сервер откликается, посылая свой SYN-сегмент (SYN=1), содержащий свой ISN=K. И AS=ISN +1=J+1.

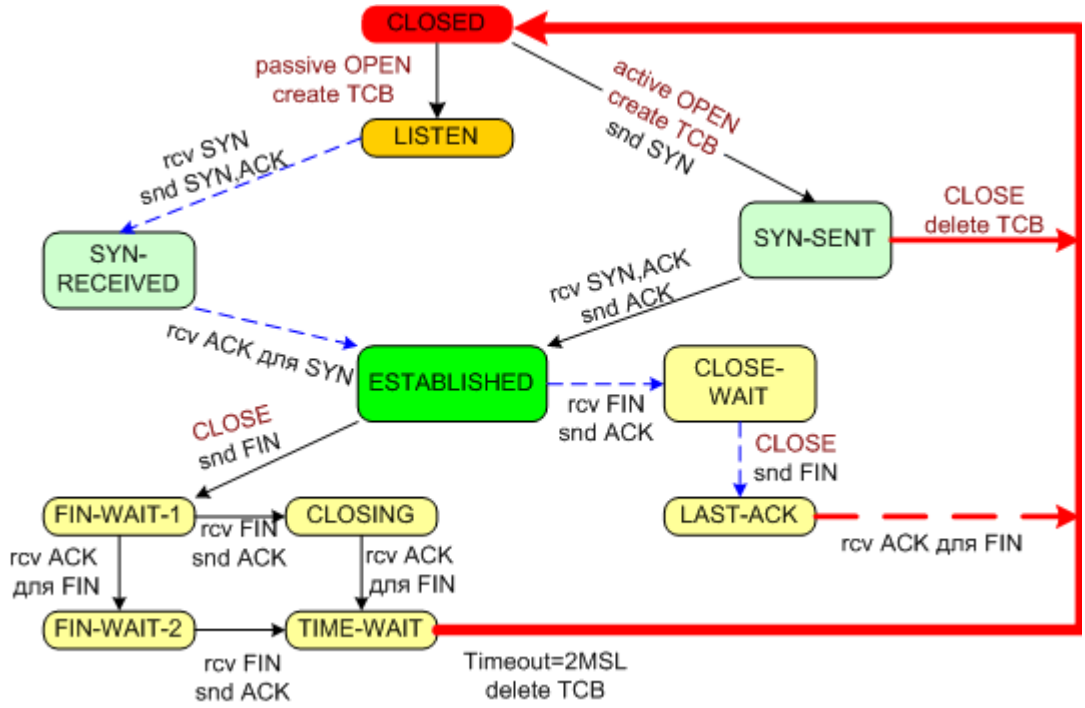
Клиент отправляет подтверждение получения SYN-сегмента от сервера с AS=ISN +1=K+1 и SN=J+1.

Начинается передача данных. При передачи данных в серийном номере .

Это один из вариантов соединения остальные можно посмотреть в RFC-793.

1.5.2. Диаграмма состояний TCP:

Для более простого восприятия сделана диаграмма состояний TCP, но ею нельзя пользоваться как спецификацией.



Упрощенная диаграмма состояний TCP (полная в RFC-793)

1.6. Протокол UDP

UDP (Universal Datagram Protocol) - универсальный протокол передачи данных, более облегченный транспортный протокол, чем TCP.

Первая и последняя версия UDP - RFC-768 (User Datagram Protocol J. Postel Aug-28-1980).

Основные отличия от TCP:

- Отсутствует соединение между модулями UDP. Не разбивает сообщение для передачи
- При потере пакета запрос для повторной передачи не посылается

UDP используется если не требуется гарантированная доставка пакетов , например, для потокового видео и аудио, DNS (т.к. данные небольших размеров). Если проверка контрольной суммы выявила ошибку или если процесса, подключенного к требуемому порту, не существует, пакет игнорируется (уничтожается). Если пакеты поступают быстрее, чем модуль UDP успевает их обрабатывать, то поступающие пакеты также игнорируются.

Структура дейтограммы UDP. Слова по 32 бита.

Не все поля UDP-пакета обязательно должны быть заполнены. Если посылаемая дейтаграмма не предполагает ответа, то на месте адреса отправителя могут помещаться нули.

1.6.1. Протокол реального времени RTP

RTP (Real Time Protocol) - транспортный протокол для приложений реального времени. **RTCP (Real Time Control Protocol)** - транспортный протокол обратной связи для приложения RTP..

2. Назначение портов

По номеру порта транспортные протоколы определяют, какому приложению передать содержимое пакетов.

Порты могут принимать значение от 0-65535 (два байта 2^16).

Номера портам присваиваются таким образом: имеются стандартные номера (например, номер 21 закреплен за сервисом FTP, 23 - за telnet, 80 - за HTTP), а менее известные приложения пользуются произвольно выбранными локальными номерами (как правило, больше>1024), некоторые из них также зарезервированы.

Некоторые заданные порты RFC-1700 (1994)

Порт	**Служба**	**Описание**
0	-	Зарезервировано
13	Daytime	Синхронизация времени
20	ftp-data	Канал передачи данных для FTP
21	ftp	Передача файлов
23	telnet	Сетевой терминал
25	SMTP	Передача почты

37	time	Синхронизация времени
43	Whois	Служба Whois
53	DNS	Доменные имена
67	bootps	BOOTP и DHCP - сервер
68	bootps	BOOTP и DHCP - клиент
69	tftp	Упрощенная передача почты
80	HTTP	Передача гипертекста
109	POP2	Получение почты
110	POP3	Получение почты
119	NNTP	Конференции
123	NTP	Синхронизация времени
137	netbios-ns	NETBIOS - имена
138	netbios-dgm	NETBIOS Datagram Service
139	netbios-ssn	NETBIOS Session Service
143	imap2	Получение почты
161	SNMP	Протокол управления
210	z39.50	Библиотечный протокол

3. Служба WWW

Служба WWW (World Wide Web) - предназначена для обмена гипертекстовой информацией.

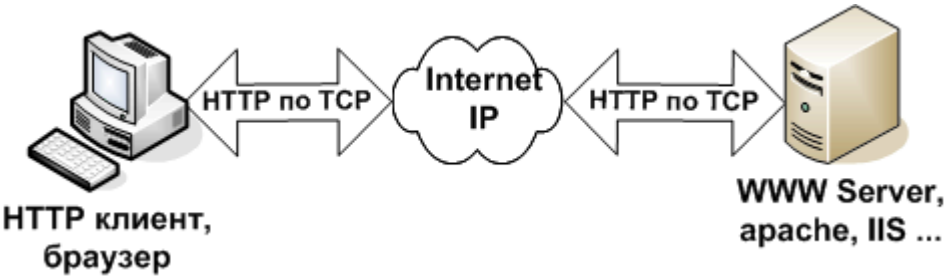
Проект был предложен в 1989 году. В 1993 появился первый браузер.

WWW построена по схеме "клиент-сервер".

Браузер (Internet Explorer, Opera ...) является мультипротокольным клиентом и интерпретатором HTML. И как типичный интерпретатор, клиент в зависимости от команд (тегов) выполняет различные функции. В круг этих функций входит не только размещение текста на экране, но обмен информацией с сервером по мере анализа полученного HTML-текста, что наиболее наглядно происходит при

отображении встроенных в текст графических образов.

Сервер HTTP (Apache, IIS ...) обрабатывает запросы клиента на получение файла (в самом простом случае).



Взаимодействие клиент и сервера по протоколу HTTP.

В начале служба WWW базировалась на трех стандартах:

HTML (HyperText Markup Language) - язык гипертекстовой разметки документов;

URL (Universal Resource Locator) - универсальный способ адресации ресурсов в сети ;

HTTP (HyperText Transfer Protocol) - протокол обмена гипертекстовой информацией.

Позже добавили:

CGI (Common Gateway Interface) - универсальный интерфейс шлюзов. Создан для взаимодействия HTTP - сервера с другими программами, установленными на сервере (например, СУБД).

4. Протокол HTTP

Первый документ (но не стандарт) - RFC1945 (Hypertext Transfer Protocol -- HTTP/1.0 T. Berners-Lee, R. Fielding, H. Frystyk May 1996)

Последняя версия - RFC2616 (Hypertext Transfer Protocol -- HTTP/1.1 R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee June 1999)

Hypertext Transfer Protocol - протокол передачи гипертекста, протокол высокого уровня (а именно, уровня приложений).

.Используется службой WWW для передачи Web-страниц.

Протокол HTTP определяет запрос-ответный способ взаимодействия между программой-клиентом и программой-сервером в рамках технологии World Wide Web. Ниже приведены примеры запроса клиента и ответа сервера:

```
I 03.12.2002 13:03:36 ---------- Attempt 1 ----------
P 03.12.2002 13:03:36 Connecting to ipm.kstu.ru ...
P 03.12.2002 13:03:36 Connected to ipm.kstu.ru [195.208.44.20]
S 03.12.2002 13:03:36 GET /internet/index.php HTTP/1.1
S 03.12.2002 13:03:36 Connection: close
S 03.12.2002 13:03:36 Host: ipm.kstu.ru
S 03.12.2002 13:03:36 Accept: */*
S 03.12.2002 13:03:36 Pragma: no-cache
S 03.12.2002 13:03:36 Cache-Control: no-cache
S 03.12.2002 13:03:36 Referer: http://ipm.kstu.ru/internet/
S 03.12.2002 13:03:36 User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
S 03.12.2002 13:03:36 Cookie: user=old; PHPTEST=YToyOntzOjE2OiJjb29raWVfcV9zd
S 03.12.2002 13:03:36
R 03.12.2002 13:03:36 HTTP/1.0 200 OK
R 03.12.2002 13:03:36 Date: Tue, 03 Dec 2002 10:22:04 GMT
R 03.12.2002 13:03:36 Server: Apache/1.3.26 (Unix) PHP/4.1.2 rus/PL30.15
R 03.12.2002 13:03:36 X-Powered-By: PHP/4.1.2
R 03.12.2002 13:03:36 Content-Type: text/html; charset=windows-1251
R 03.12.2002 13:03:36 Expires: Thu, 01 Jan 1970 00:00:01 GMT
R 03.12.2002 13:03:36 Last-Modified: Tue, 03 Dec 2002 10:22:04 GMT
R 03.12.2002 13:03:36 X-Cache: MISS from proxy.kfti.knc.ru
R 03.12.2002 13:03:36 Connection: close
R 03.12.2002 13:03:36
P 03.12.2002 13:03:36 Data transfer started
P 03.12.2002 13:03:36 Connection closed by remote server
I 03.12.2002 13:03:36 Received 1 755 bytes in 0:00:00 (25 071 bytes/s)
I 03.12.2002 13:03:37 JOB COMPLETED SUCCESSFULLY
```

Листинг запроса и ответа HTTP

Установка соединения
Connecting to ipm.kstu.ru ...
Connected to ipm.kstu.ru [195.208.44.20]

Запрос клиента:
GET /internet/index.php HTTP/1.1 - (запрос файла и указание протокола HTTP/1.1)
Connection: close - (закрыть соединение после отправки файла)
Host: ipm.kstu.ru - (указание адреса сервера)
Accept: */* - (предпочтение типов данных)
Cache-Control: no-cache - (не кешировать)
Referer: http://ipm.kstu.ru/internet/ - (от куда пришел клиент)
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 95) - (название программы клиента)

Ответ сервера
HTTP/1.0 200 OK - (какой протокол используется, 200 - означает, что файл найден)
Date: Wed, 23 Oct 2002 08:32:31 GMT - (дата и время ответа)
Server: Apache/1.3.26 (Unix) PHP/4.1.2 rus/PL30.15 - (название, версия и модули http-сервера) X-Powered-By: PHP/4.1.2 - (чем создана страница)
Connection: close - (закрыть соединение после получения файла)
Content-Type: text/html; charset=windows-1251 - (тип данных html, кодировка windows-1251)

Дальше идет содержимое файла (тело запроса).

Работа по протоколу HTTP происходит следующим образом: программа-клиент устанавливает TCP-соединение с сервером (стандартный номер порта-80) и выдает ему HTTP-запрос. Сервер обрабатывает этот запрос и выдает HTTP-ответ клиенту.

HTTP-запрос состоит из заголовка запроса и тела запроса, разделенных пустой строкой. Тело запроса может отсутствовать.

Заголовок запроса состоит из главной (первой) строки запроса и последующих строк, уточняющих запрос в главной строке. Последующие строки также могут отсутствовать.

Запрос в главной строке состоит из трех частей, разделенных пробелами:

1) Метод (иначе говоря, команда HTTP):

GET - Метод GET служит для получения любой информации, в соответствии URI-запроса.

HEAD - запрос заголовка документа. Отличается от GET тем, что выдается только заголовок запроса с информацией о документе. Сам документ не выдается.

POST - этот метод применяется для передачи данных CGI-скриптам. Сами данные следуют в последующих строках запроса в виде параметров.

PUT - поместить документ на сервере. Запрос с этим методом имеет тело, в котором передается сам документ.

DELETE - используется для удаления ресурсов, идентифицированных с помощью URI-запроса

2) Ресурс - это путь к определенному файлу на сервере (называется URI), который клиент хочет получить (или разместить - для метода PUT). Если ресурс - просто какой-либо файл для считывания, сервер должен по этому запросу выдать его в теле ответа. Если же это путь к какому-либо CGI-скрипту, то сервер запускает скрипт и возвращает результат его выполнения. Кстати, благодаря такой унификации ресурсов для клиента практически безразлично, что он представляет собой на сервере.

3) Версия протокола - версия протокола HTTP, с которой работает клиентская программа.

Строки после главной строки запроса имеют следующий формат: Параметр: значение.

Таким образом, задаются параметры запроса. Это является необязательным, все строки после главной строки запроса могут отсутствовать;

в этом случае сервер принимает их значение по умолчанию или по результатам предыдущего запроса (при работе в режиме Keep-Alive).

4.1. Некоторые параметры HTTP-запроса:

Connection (соединение) - может принимать значения Keep-Alive и close. Keep-Alive ("оставить в живых") означает, что после выдачи данного документа соединение с сервером не разрывается, и можно выдавать еще запросы. Большинство браузеров работают именно в режиме Keep-Alive, так как он позволяет за одно соединение с сервером "скачать" html-страницу и рисунки к ней. Будучи однажды установленным, режим Keep-Alive сохраняется до первой ошибки или до явного указания в очередном запросе Connection: close.
close ("закрыть") - соединение закрывается после ответа на данный запрос.

User-Agent - значением является "кодовое обозначение" браузера, например:

Mozilla/4.0 (compatible; MSIE 5.0; Windows 95; DigExt)

Accept - список поддерживаемых браузером типов содержимого в порядке их предпочтения данным браузером, например:

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*

Это, очевидно, нужно для случая, когда сервер может выдавать один и тот же документ в разных форматах.

Значение этого параметра используется в основном CGI-скриптами для формирования ответа, адаптированного для данного браузера.

Referer - URL, с которого перешли на этот ресурс.

Host - имя хоста, с которого запрашивается ресурс. Полезно, если на сервере имеется несколько виртуальных серверов под одним IP-адресом. В этом случае имя виртуального сервера определяется по этому полю.

Accept-Language - поддерживаемый язык. Имеет значение для сервера, который может выдавать один и тот же документ в разных языковых версиях.

4.2. Формат HTTP-ответа

Формат ответа очень похож на формат запроса: он также имеет заголовок и тело, разделенное пустой строкой.

Заголовок также состоит из основной строки и строк параметров, но формат основной строки отличается от таковой в заголовке запроса.

Основная строка запроса состоит из 3-х полей, разделенных пробелами:

Версия протокола - аналогичен соответствующему параметру запроса.

4.3. Код возврата (ошибки, состояния)

Код возврата - кодовое обозначение "успешности" выполнения запроса.

Например, код 200 означает "все нормально" (OK).

Значения кодов возврата по первой цифре:

1xx: Информационный - Не используется, но зарезервирован для использования в будущем

2xx: Успех - Запрос был полностью получен, понят, и принят к обработке.

3xx: Перенаправление - Клиенту следует предпринять дальнейшие действия для успешного выполнения запроса. Необходимое дополнительное действие иногда может быть выполнено клиентом без взаимодействия с пользователем, но настоятельно рекомендуется, чтобы это имело место только в тех случаях, когда метод, использующийся в запросе безразличен (GET или HEAD).

4xx: Ошибка клиента - Запрос, содержащий неправильные синтаксические конструкции, не может быть успешно выполнен. Класс 4xx предназначен для описания тех случаев, когда ошибка была допущена со стороны клиента. Если клиент еще не завершил запрос, когда он получил ответ с Статус-Кодом- 4xx, он должен немедленно прекратить передачу данных серверу. Данный тип Статус-Кодов применим для любых методов, употребляющихся в запросе.

5xx: Ошибка Сервера - Сервер не смог дать ответ на корректно поставленный запрос. В этих случаях сервер либо знает, что он допустил ошибку, либо не способен обработать запрос. За исключением ответов на запросы HEAD, сервер посылает описание ошибочной ситуации и то, является ли это состояние временным или постоянным, в Содержание-Ответа. Данный тип Статус-Кодов применим для любых методов, употребляющихся в запросе.

Наиболее часто встречающиеся:
"200"; OK - документ отправлен.
"404"; Not Found - документ не найден (ошибка URL), клиент должен проверить правильность написания URL, если не помогает, значит, документ уже удален.
"500"; Internal Server Error - внутренняя ошибка сервера, клиент исправить не может, только администратор сервера.

Словесное описание ошибки - "расшифровка" предыдущего кода. Например, для 200 это OK, для 500 - Internal Server Error.

Таблица кодов возврата

Код	**Название**	**Описание**
100	Продолжайте	Клиент должен продолжать передачу запроса.
101	Переключение протоколов	Сервер предлагает изменить протокол на указанный в поле заголовка отклика Update. Обычно это предложение о переходе на более новую версию HTTP.
110	Отклик мог устареть	Отклик является устаревшим (используется в заголовке Warning).
111	Обновление не удалось	Отклик от кэша не является свежим, т. к. попытка обновить его закончилась неудачей (используется в заголовке Warning).
112	Разрыв соединения	Кэш был умышленно отсоединен от остальной сети на некоторое время (используется в заголовке Warning).
113	Эвристическое устаревание	Кэш эвристически выбрал период обновления, больший 24 часов, и возраст отклика более 24 часов (используется в заголовке Warning).
199	Различные предупреждения	Дополнительные предупреждения, не содержащиеся в данном списке (используется в заголовке Warning).
200	**OK**	Запрос был успешно обработан. Содержимое отклика зависит от типа запроса.
201	Создано	Запрос был успешно обработан и в результате был создан новый ресурс. Его URI указан в поле заголовка отклика Location.
202	Принято	Запрос был принят, и его обработка началась другим асинхронным процессом, поэтому сервер не сможет сообщить о ее завершении.
203	Неавторитетная информация	Возвращаемая метаинформация получена не от сервера ее происхождения, а из локальной копии.
204	Нет содержимого	Сервер выполнил запрос, но ему нечего возвращать клиенту. Обозреватель не должен изменять отображение документа.
205	Сброс содержимого	Сервер выполнил запрос, и обозреватель должен сбросить отображение документа.
206	Частичное содержимое	Сервер выполнил частичный запрос GET для ресурса.
214	Применено преобразование	Применено преобразование, изменившее кодировку или тип MIME отклика (используется в заголовке Warning).
299	Различные настойчивые предупреждения	Дополнительные предупреждения, не содержащиеся в данном списке (используется в заголовке Warning).

300	Несколько вариантов	Запрошенный ресурс имеет несколько представлений, и клиент должен выбрать одно из них.
301	Ресурс перенесен	Запрошенный ресурс сменил свой URI. Его новый URI указан в поле заголовка отклика Location
302	Найдено	Запрошенный ресурс временно сменил свой URI.
303	Смотри другое	Отклик на данный запрос может быть найден под другим URI, указанным в поле заголовка отклика Location.
304	Не изменено	Клиент выполнил условный запрос GET, доступ разрешен, но документ не был изменен.
305	Используйте прокси	Доступ к запрошенному ресурсу возможен только через прокси-сервер, указанный в поле заголовка отклика Location.
306	зарезервирован	
307	Временное перенаправление	Запрошенный ресурс временно находится под другим URI, указанном в поле заголовка отклика Location.
400	Неверный запрос	Запрос не был понят сервером из-за его неверного синтаксиса.
401	Нет права доступа	Запрос требует авторизации доступа, тип которой указан в поле заголовка отклика WWW-Authenticate.
402	Требуется платеж	Зарезервировано для следующих версий HTTP.
403	Запрещено	Сервер понял запрос, но отказался его выполнять.
404	Не найдено	Ресурс, заданный в URI запроса, не найден.
405	Недопустимый метод	Данный тип запроса не применим к ресурсу, заданному в URI запроса.
406	Неприемлемо	Ресурс, заданный в URI запроса, может генерировать только отклики, не приемлемые для клиента.
407	Прокси требует авторизации доступа	Прокси-сервер требует авторизации доступа, тип которой указан в поле заголовка отклика Proxy-Authenticate.
408	Таймаут запроса	Клиент не послан ни одного запроса в течение отведенного ему интервала.
409	Конфликт	Запрос не может быть выполнен из-за конфликта с текущим состоянием ресурса.
410	Ресурс исчез	Запрошенного ресурса больше нет на сервере, и сервер не знает его нового URI.
411	Требуется длина запроса	В запросе не задано поле заголовка Content-Length.
412	Условие ложно	Условие, заданное в заголовке условного запроса, не может быть выполнено.
413	Слишком длинное тело запроса	Тело запроса длиннее, чем допускает сервер.

414	Слишком длинное URI запроса	URI запроса длиннее, чем допускает сервер.
415	Не поддерживаемый тип устройства	Формат тела запроса не поддерживается данным ресурсом для данного типа запроса.
416	Запрошенный диапазон пуст	Запрошенный ресурс не содержит значений в диапазоне, заданном в поле заголовка запроса Range.
417	Предположение не оправдалось	Предположение, указанное в поле заголовка запроса Expect, отвергнуто сервером.
500	Внутренняя ошибка сервера	Запрос не исполнен из-за неожиданной ошибки сервера.
501	Не реализовано	Сервер не поддерживает данный тип запросов.
502	Плохой шлюз	Сервер, выступающий в роли шлюза или прокси-сервера, получил неверный отклик от следующего сервера.
503	Служба недоступна	Сервер временно не может исполнить запрос из-за перегрузки.
504	Таймаут шлюза	Сервер, выступающий в роли шлюза или прокси-сервера, не получил своевременного отклика от следующего сервера.
505	Версия HTTP не поддерживается	Сервер не поддерживает версию HTTP, указанную в заголовке запроса.

4.4. Некоторые параметры http-ответа:

Connection - этот ответ аналогичен соответствующему параметру запроса.

Если сервер не поддерживает Keep-Alive (есть и такие серверы), то значение Connection в ответе всегда close.

Content-Type - содержит обозначение типа содержимого ответа в MIME.

В зависимости от значения Content-Type браузер воспринимает ответ как HTML-страницу, картинку gif или jpeg, как файл, который надо сохранить на диске, или как что-либо еще и предпринимает соответствующие действия.

Некоторые типы содержимого:
text/html - текст в формате HTML (веб-страница);
text/plain - простой текст (аналогичен "блокнотовскому");
image/jpeg - картинка в формате JPEG;
image/gif - то же, в формате GIF;

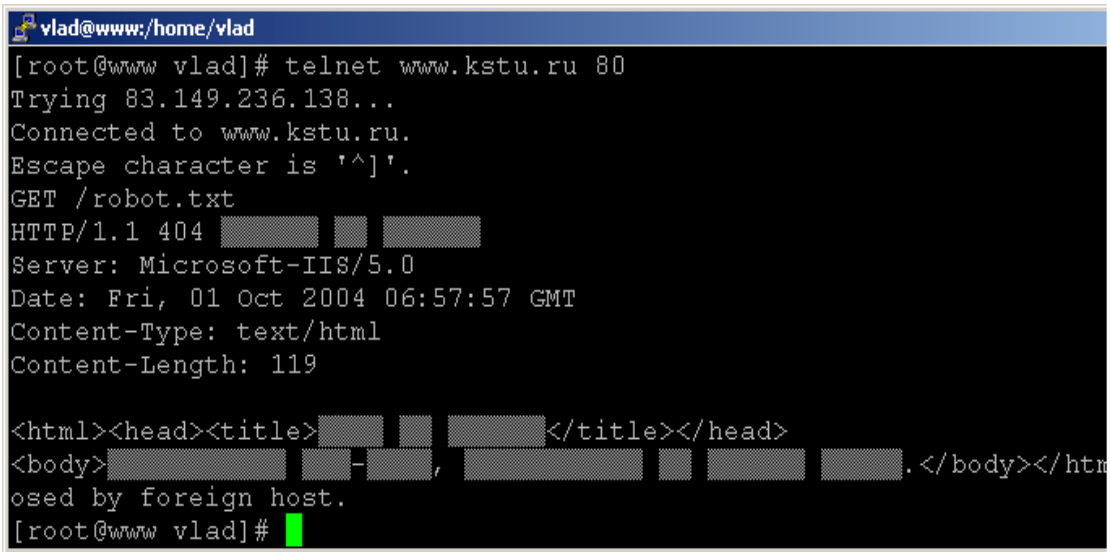
Также может передавать кодировку для текстовых данных.

Например:
charset=windows-1251
charset=koi8-rus

Content-Length - длина содержимого ответа в байтах (размер файла).
Last-Modified - дата и время последнего изменения документа.

4.5. Соединение к HTTP с помощью Telnet

Подсоединимся к командному порту HTTP (80), и запросим файл robots.txt:



Использованные источники:

1. **Администрирование сети и сервисов Internet. Учебное пособие.** *П. Б. Храмцов, Центр Информационных Технологий, 1997*

2. Стек протоколов TCP/IP http://www.agpu.net/fakult/ipimif/fpiit/kafinf/umk/el_lib/calc_system/lab_work_net/kulgin_3.htm

1. HTML – ЯЗЫК ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ

1.1. Основные понятия HTML

HTML – язык гипертекстовой разметки документов (HyperText Markup Language). С помощью HTML создаются Web-страницы или HTML-страницы, которые размещают в сети Интернет. HTML – это не язык программирования в традиционном смысле, он является языком разметки. С помощью HTML текстовый документ разбивают на блоки смысловой информации (заголовки, параграфы, таблицы, рисунки и т.п.).

Мы рассматриваем язык HTML с некоторыми ограничениями. Введенные нами ограничения позволят избежать несовместимости со следующей версией стандарта языка -- HTML5 (пока находится в разработке).

Гипертекстовый документ – это документ, содержащий переходы (гиперссылки) на другие документы. При использовании гиперссылки происходит перемещение от одного документа к другому (как по цепочке) в Интернете.HTML-документ является гипертекстовым документом.

Особенности HTML-документа:

1. HTML-документ может содержать текст, графику, видео и звук.

2. В общем случае HTML-документ – это один или несколько текстовых файлов, имеющих расширение .htm или .html.

3. Создавать HTML-документ можно как с помощью специальных программ – редакторов HTML (например, Notepad++), так и с помощью любого текстового редактора (например, блокнота Windows).

4. Для просмотра HTML-документов существуют специальные программы-браузеры. Они интерпретируют HTML-документы, т.е. переводят текст документа и отображают ее на экране пользователя. Существует много различных браузеров, но наиболее распространенными браузерами являются Chrome, Firefox, Microsoft Internet Explorer и Opera. Познакомиться со статистикой использования браузеров можно по адресу: http://www.w3schools.com/browsers/browsers_stats.asp

5. Если при интерпретации HTML-документа браузер чего-то не понимает, то сообщения об ошибке не возникает, а это место в HTML-документе игнорируется и не отображается браузером.

HTML-документ состоит из HTML-элементов.

HTML-элемент– это чаще всего два тэга (открывающий и закрывающий) и часть документа между ними. Кроме того, существуют элементы HTML, состоящие только из одного тэга.

Тэг – в переводе с английского – ярлык, этикетка. Тэг определяет тип выводимого элемента HTML (например, заголовок, таблица, рисунок и т.п.). Сам тэг не отображается браузером. Тэг представляет собой последовательность элементов:

- символ левой угловой скобки (<) – начало тэга;
- необязательный символ слеша (/) – символ используется, чтобы обозначить закрывающий тэг;
- имя тэга;
- необязательные атрибуты в открывающем тэге;
- символ правой угловой скобки (>)

Как правило тэги парные, то есть существует открывающий и закрывающий тэги. Однако, существуют, так называемые пустые HTML элементы, они не имеют содержимого. Например, тэг
 является пустым и не имеет закрывающего тэга (тэг
 определяет разрыв строки). Такие тэги рекомендуется использовать в написании
.

Атрибуты – необязательный набор параметров, определяющих дополнительные свойства элемента HTML (например, цвет или размер).

Атрибут состоит:

- из имени атрибута;
- знака равенства (=);
- значения атрибута – строки символов, заключенной в кавычки

Пример элемента HTML:

<h1 align= "center">ГЛАВА 1</h1>

в этом примере:
<h1 align= "center"> – открывающий тэг
</h1> – закрывающий тэг
h1 – имя тэга
align= "center" – атрибут
align – имя атрибута
"center" – значение атрибута
ГЛАВА 1 – содержание элемента HTML h1

Правила создания HTML-документов:

1. Тэги и атрибуты рекомендуется записывать в нижнем регистре, например, </h1>, а не <H1>.

2. Значения атрибутов обязательно заключаются в кавычки.

3. Каждый HTML-документ, отвечающий спецификации HTML какой-либо версии, обязан начинаться со строки декларации версии HTML <!DOCTYPE>, которая обычно выглядит примерно так:
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">. Если эта строка не указана, то добиться корректного отображения документа в браузере становится труднее.
Для HTML5 данная строка имеет вид: <!DOCTYPE html>

4. Несколько пробелов подряд, символы табуляции и перевода строки при интерпретации браузером заменяются на один пробел. Это позволяет писать хорошо структурированные исходные тексты файлов HTML.

5. Рекомендуется давать имена файлам HTML строчными английскими буквами. Длина имени – до восьми символов. В принципе, можно не придерживаться данной рекомендации, но тогда пользователи, работающие в операционных системах, отличных от Windows, не смогут воспользоваться данными HTML-документами.

6. Большинство HTML-элементов могут быть вложены (могут содержать вложенные элементы). HTML-документ состоит из вложенных HTML-элементов.

1.2. Структура HTML-документа

Каждый HTML-документ должен начинаться тэгом <html> и заканчиваться тэгом </html>. Эти тэги обозначают, что находящиеся между ними строки представляют единый HTML-документ. Кроме того, можно заметить, что файл HTML в целом является элементом языка HTML.

Также в HTML-документе должны присутствовать элементы HEAD (информация о документе) и BODY (тело документа).

1.2.1. Раздел документа HEAD

Раздел документа HEAD определяет его заголовок, а также содержит дополнительную информацию о документе для браузера.

Раздел заголовка начинается тэгом <head> и следует сразу за тэгом <html>. Между открывающим и закрывающим тэгами элемента HEAD располагаются другие элементы заголовка.

Название документа TITLE

Для того чтобы дать название HTML-документу, предназначен тэг <title>. Это название будет выведено в заголовок окна браузера. Это же название по умолчанию дается файлу при сохранении его пользователем.

Название записывается между тэгами <title> и </title> и представляет собой строку текста. Длина этой строки может быть любой, но рекомендуется делать ее не больше 60 символов. Элемент TITLE может находиться только в разделе HEAD.

1.2.2. Раздел документа BODY

В этом разделе документа располагается та информация, которая отображается в окне браузера. Раздел BODY должен начинаться тэгом <body> и завершаться тэгом </body>, между которыми располагаются другие элементы HTML, из которых и состоит содержание документа.

Спецификация элемента BODY

Тэг <body> имеет ряд атрибутов, определяющих внешний вид документа. Ниже приводится спецификация тэга <body>.

```
<body
text="цвет текста"
bgcolor="цвет фона"
background="адрес фонового рисунка"
link="цвет непосещенной гиперссылки"
vlink="цвет посещенной гиперссылки"
alink="цвет активной гиперссылки"
>
```

Атрибут text задает цвет шрифта для всего документа в формате RGB или в символьной нотации. По умолчанию (если не указан этот атрибут) используются настройки браузера.

Атрибут bgcolor задает цвет фона окна браузера документа в формате RGB или в символьной нотации. По умолчанию используются настройки браузера.

Атрибут background позволяет указать адрес и имя рисунка, используемого в качестве фона. Этот рисунок будет размножен и распределен на заднем плане документа.

Атрибуты link, vlink и alink задают цвета гиперссылок в формате RGB или в символьной нотации. По умолчанию используются настройки браузера. Непосещенная гиперссылка – гиперссылка, которая еще не использовалась для перехода к другому документу. Посещенная гиперссылка – гиперссылка, которая уже использовалась для перехода к другому документу. Активная гиперссылка – гиперссылка на документ, к которому в данный момент происходит переход.

Советы по использованию атрибутов тэга BODY

Хотя можно задавать значения атрибутов в тэге BODY настоятельно рекомендуется выносить эти определения в CSS (Cascading Style Sheets – Каскадные таблицы стилей), тем более что они дают гораздо большие возможности для оформительских эффектов.

Пример простого HTML-документа

```
<!DOCTYPE html>
<html>
<body>
<p>Это мой первый абзац</p>
</body>
</html>
```

1.3. Цветовые спецификации

Для определения цвета различных элементов HTML-документа необходимо указать значение соответствующих атрибутов. Указывать значения этих атрибутов можно двумя способами:

- определять цвет в формате RGB (в десятичной и шестнадцатеричной системах счисления);
- определять цвет, используя символьную нотацию

1.3.1. Формат RGB

Формат RGB – это система указания цвета, которая базируется на смешении трех основных цветов: красном (RED), зеленом (GREEN) и синем (BLUE). Итоговый цвет определяется цифрами в шестнадцатеричном коде. Для каждого цвета задается шестнадцатеричное значение в пределах от 0 до FF, что соответствует диапазону 0-255 в десятичном исчислении. Затем эти значения объединяются в одно число, перед которым ставится символ #. Например, число #800080 обозначает фиолетовый цвет. Указывая цвет в формате RGB, можно определить более шестнадцати миллионов цветовых оттенков.

1.3.2. Символьная нотация

Задание цвета в формате RGB имеет один недостаток – необходимо помнить совокупности цифр для указания цвета. Этого недостатка лишена символьная нотация. Можно указывать название цвета на английском языке..

1.4. Соответствие формата RGB и символьной нотации

Ниже приведена таблица из нескольких соответствий указания цвета в символьной нотации и формате RGB.

Символьная нотация	Формат RGB (шестнадцатеричный)	Формат RGB (десятичный)	Цвет
Black	#000000	rgb(0,0,0)	Черный
Silver	#C0C0C0	rgb(192,192,192)	Серебро

Lime	#00FF00	rgb(0,255,0)	Лайм
Red	#FF0000	rgb(255,0,0)	Красный

Таким образом, строка text="#00FF00" и строка text="Lime" в тэге <body> одинаково определяют цвет шрифта – лайм (светло зеленый).

1.5. Вывод текстовой информации

Любые тексты, будь то школьные сочинения, заметка в газету или техническое описание устройства, имеют определенную структуру. Элементами такой структуры являются заголовки, подзаголовки, абзацы, списки и др.

Разбиение всего текста на структурные элементы называется логическим форматированием. В HTML-документе логическое форматирование достигается с

1.5.1. Абзацы

Одним из первых правил составления любых документов является разбиение его текста на отдельные абзацы, выражающие законченную мысль. В HTML-документе разделение на абзацы производится с помощью специального тэга <p>

Синтаксис этого тэга таков:

<p align="выравнивание">

Атрибут align определяет способ выравнивания абзаца. Он может иметь следующие значения:

left – текст выравнивается по левому краю окна браузера. Это значение используется по умолчанию, т.е. когда атрибут не указан.

center – текст выравнивается по центру окна браузера.

right – текст выравнивается по правому краю окна браузера.

Пример использования тэга <p>:

```
<p> Это мой первый абзац</p>
<p align="center"> Это мой второй абзац</p>
<p align="rigth"> Это мой третий абзац</p>
```

Этот документ отобразится в браузере так:

Это мой первый абзац

Это мой второй абзац

Это мой третий абзац

Браузер автоматически формирует абзацы в зависимости от ширины окна браузера или размера шрифта, перенося слова из строки в строку и отделяя абзацы друг от друга пустой строкой.

Управление переводом строки

Так как браузер автоматически определяет места переноса строк, иногда возникают ситуации запретить перевод строки в каком-нибудь месте или, наоборот, принудительно сделать перевод строки в каком-то определенном месте. Для этого существуют особые тэги, управляющие переводом строк.

Когда необходимо сделать принудительный перевод строки, используют тэг
. Этот тэг не имеет атрибутов и закрывающего тэга. Пример использования принудительного перевода строки:

```
<p>Добро<br>пожаловать!</p>
```

При использовании тэга
 пустая строка не образуется, т.е. абзац не прерывается.

В некоторых случаях, наоборот, бывает необходимо сделать так, чтобы браузер не производил перевода строки. Например, не рекомендуется отрывать буквы инициалов от фамилии. В таких случаях рекомендуется вместо пробелов использовать нерасширяемый пробел - .

1.5.2. Заголовки

Почти в каждом тексте используются заголовки для отдельных частей документа. Эти заголовки представляют собой фрагменты текста, которые выделяются на экране при отображении страницы браузером.

Для разметки заголовков используются тэги <h1>, <h2>, <h3>, <h4>, <h5> и <h6>. Эти тэги требуют соответствующего закрывающего тэга. Заголовок с номером 1 является самым крупным (заголовок верхнего уровня), а с номером 6 – самым мелким. Тэги заголовка нельзя использовать для выделения отдельных слов текста с целью увеличения их размера. При использовании тэгов заголовков происходит вставка пустой строки до и после заголовка, поэтому тэгов абзаца и перевода строки здесь не требуется.

1.5.3. Основные тэги форматирования текста

 … — логическое ударение (обычно отображается *курсивным шрифтом*)

 … — усиленное логическое ударение (обычно отображается **жирным шрифтом**)

 … — выделение текста *курсивом*

 … — выделение текста **жирным шрифтом**

<u> … </u> — подчёркивание текста

_… — подстрочный текст. Например, H₂O создаст текст H_2O.

[…] — надстрочный текст. Например, E=mc² создаст текст $E=mc^2$.

 … — задание параметров шрифта. У этого тэга есть следующие параметры:

color=*color* — задание цвета. Цвет может быть задан в шестнадцатеричной форме как #rrggbb (первые 2 шестнадцатеричные цифры задают красную компоненту, следующие 2 — зелёную, последние 2 — синюю) или названием.

face=*шрифт* задание гарнитуры шрифта

size=*размер* задание размера шрифта. Размер от 1 до 7: стандартный по умолчанию 3. Есть много способов изменить стандартный размер.

size=+*изменение* или size=-*изменение* — изменение размера шрифта от стандартного. Например, +2 означает размер на 2 больше стандартного.

1.5.4. Контейнер DIV

Иногда бывает необходимо произвести выравнивание большого блока документа, содержащего не только текст, но и рисунки, таблицы и т.п. Для этих целей используется элемент-контейнер **DIV**.

Спецификация элемента DIV:

Атрибут align определяет тип выравнивания содержимого и может иметь те же значения, что и элемент P.

1.6. Списки

В языке HTML предусмотрен специальный набор тэгов для представления информации в виде списков. Списки являются одним из наиболее часто употребляемых форм представления данных как в электронных документах, так и печатных. В языке HTML предусмотрены маркированные, нумерованные списки и списки определений.

1.6.1. Маркированный список

Этот список еще называется ненумерованным или неупорядоченным. В маркированном списке для выделения его элементов используются специальные символы, называемые маркерами списка. Вид маркеров определяется браузером, причем при создании вложенных списков браузеры автоматически разнообразят вид маркеров различного уровня вложенности.

Для создания маркированного списка необходимо использовать тэг-контейнер <ul> , внутри которого располагаются все элементы списка. Открывающий и закрывающий тэги списка обеспечивают перевод строки до и после списка, отделяя, таким образом, список от основного содержимого документа, поэтому нет необходимости использовать тэги абзаца или принудительного перевода строки.

Каждый элемент списка должен начинаться тэгом <li> и заканчиваться тэгом .

Спецификация элемента UL:

<ul type="вид маркера">

Атрибут type задает вид маркера, которым выделяются элементы списка. Он может имеет следующие значения:

disk – маркеры отображаются закрашенными кружочками, это значение используется по умолчанию;

circle – маркеры отображаются не закрашенными кружочками;

square – маркеры отображаются квадратиками.

1.6.2. Нумерованный список

Нумерованные списки иногда называют упорядоченными. Списки данного типа представляют собой упорядоченную последовательность отдельных элементов. Отличием от маркированных списков является то, что в нумерованном списке перед каждым его элементом автоматически проставляется порядковый номер. Вид нумерации зависит от браузера и может задаваться атрибутами тэгов списка. В остальном реализация нумерованного списка во многом похожа на реализацию маркированного списка.

Для создания нумерованного списка следует использовать тэг-контейнер <ol> , внутри которого располагаются все элементы списка.

Каждый элемент списка должен начинаться тэгом <li> и заканчиваться тэгом .

Спецификация элемента OL:

<ol type="вид нумерации" start="начальная позиция">

Атрибут type задает вид нумерации, которой выделяются элементы списка. Он может иметь следующие значения:

A – маркеры в виде прописных латинских букв;

a – маркеры в виде строчных латинских букв;

I- маркеры в виде больших римских цифр;

i- маркеры в виде маленьких римских цифр;

1- маркеры в виде арабских цифр, это значение используется по умолчанию.

Атрибут start определяет позицию, с которой начинается нумерация списка. Используя этот атрибут, можно начать нумерацию, например, с цифры 5 или буквы E, в зависимости от вида нумерации. Значением атрибута START является число, вне зависимости от вида нумерации.

Пример:

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
    <li>Black tea</li>
    <li>Green tea
      <ul>
      <li>China</li>
      <li>Africa</li>
      </ul>
    </li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

1.7. Гиперссылки

Как уже было сказано, гиперссылки – переходы к другим документам. Они являются основой HTML. Гиперссылки можно использовать для перехода не только к другим HTML-документам, но и другим объектам, которые можно разместить на компьютере, например, к видео- и аудиофайлам, архивам, рисункам и т.п.

Каждая ссылка состоит из двух частей. Первая – это то, что отображается браузером. Она называется указателем ссылки. Вторая часть – адресная, содержащая адрес объекта, к которому должен происходить переход.

Когда вы щелкаете мышью по указателю ссылки, браузер загружает документ, адрес которого указан в URL.

Указателем ссылки может быть слово, группа слов или рисунок. Если указатель текстовый, то он обычно отображается браузером подчеркнутым синим шрифтом. При наведении курсора мыши на указатель курсор принимает форму руки, указывая, что это ссылка и можно произвести переход. Если указатель графический, внешне он не отличается от других рисунков, но при наведении курсора мыши на такой указатель, курсор также принимает форму руки.

1.7.1. Правила записи ссылок

Для организации ссылки необходимо указать браузеру, что является указателем ссылки, а также определить адрес документа, на который происходит ссылка. Оба этих действия выполняются с помощью тэга <a>.

Тэг <a> имеет следующую спецификацию:

<a href="url-адрес" name="имя ссылки" target="объект для вывода">

Атрибут HREF используется для задания адреса файла, к которому производится переход. Значением этого атрибута является текстовая строка, содержащая абсолютный или относительный URL-адрес.

Атрибут NAME предназначен для задания ссылке имени. Значением этого атрибута является короткая текстовая строка. Этот атрибут используется для ссылок внутри одного HTML-документа.

Атрибут TARGET позволяет определить, куда будет выводиться документ, на который происходит переход. Этот атрибут может иметь значение _blank – это означает, что документ будет выводиться в новом окне браузера.

Пример ссылки:

Документ 1

Браузер отобразит эту строку так: Документ 1

При нажатии мышью на этой строке браузер загрузит и отобразит файл doc1.htm, т.е. “Документ 1” – это указатель ссылки, а “doc1.htm” – URL-адрес.

1.7.2. Внутренние ссылки

Кроме ссылок на другие документы, часто бывает полезно включить ссылки на различные части текущего документа. Например, большой документ читается лучше, если он имеет оглавление со ссылками на соответствующие разделы.

Для построения внутренней ссылки сначала нужно создать указатель, определяющий место назначения. Для этого в месте, куда потом будет производиться ссылка, надо поместить тэг <a> с атрибутом name, определив этим атрибутом имя указателя.

Например:

Обратите внимание, что в этом примере отсутствует содержимое тэга <a>. Обычно так и делают, поскольку здесь нет необходимости как-то выделять текст, а требуется лишь указать местоположение.

После того как место назначения определено, можно приступить к созданию ссылки на него. Для этого в атрибуте HREF тэга <a> помещается имя ссылки с префиксом #, говорящим о том, что это внутренняя ссылка.

Например:

Глава 5

Теперь, если пользователь щелкнет кнопкой мыши на словах “Глава 5”, то браузер выведет соответствующую часть документа в окне просмотра.

Можно совмещать внутренние ссылки со ссылками на другие документы. Например:

Глава 5 Документа 1

При нажатии на эту ссылку браузер откроет файл doc1.htm, найдет в этом файле указатель glava5 и выведет в окне просмотра соответствующую информацию.

1.8. Таблицы

Одним из наиболее мощных и широко применяемых в HTML средств являются таблицы. Они используются не только традиционно как метод представления данных, но и как средство разметки Web-страниц. Документ HTML может содержать произвольное число таблиц, причем допускается вложенность таблиц друг в друга.

Каждая таблица начинается тэгом <table> и заканчивается тэгом </table>. Внутри этой пары тэгов располагается описание содержимого таблицы. Любая таблица состоит из одной или нескольких строк, в которых задаются данные для отдельных ячеек.

Каждая строка начинается тэгом <tr> и заканчивается тэгом </tr>. Отдельная ячейка в строке обрамляется парой тэгов <td> и </td> или <th> и </th>. Тэг <TH>

используется для ячеек заголовка таблицы, а <td> – для ячеек данных. Отличие этих тэгов в том, что в заголовке по умолчанию используется полужирный шрифт, а для данных – обычный.

Тэги <td> и <th> не могут появляться вне описания строки таблицы <tr>.

Пример таблицы:

```
<table>
<tr>
 <td>ячейка 1</td>
 <td>ячейка 2</td>
</tr>
<tr>
 <td>ячейка 3</td>
 <td>ячейка 4</td>
</tr>
</table>
```

Спецификация тэга <table>:

<table align="выравнивание" border="толщина рамки" cellpadding="расстояние" cellspacing="расстояние" height="высота" valign="вертикальное выравнивание" width="ширина" >

Атрибут align определяет выравнивание таблицы в окне просмотра браузера. Он может иметь одно из двух значений: left (по левому краю) и right (по правому краю). По умолчанию – left.

Атрибут border управляет толщиной рамки. Значением этого атрибута является число. Это число определяет толщину рамки таблицы в пикселях. По умолчанию толщина рамки – 1.

Атрибут cellpadding определяет расстояние в пикселях между рамкой и содержимым ячейки. По умолчанию – 1.

Атрибут cellspacing определяет расстояние в пикселях между ячейками таблицы. По умолчанию – 2.

Атрибут height определяет высоту таблицы в пикселях.

Атрибут valign определяет вертикальное выравнивание содержимого таблицы. Он может иметь следующие значения: top (по верхнему краю), middle (посередине) и bottom (по нижнему краю). По умолчанию – middle.

Атрибут width определяет ширину таблицы в пикселях или процентах от ширины окна браузера.

Спецификация тэга <tr>

<tr align="выравнивание" bgcolor="цвет фона" valign="вертикальное выравнивание" >

Атрибут align определяет выравнивание содержимого всех ячеек строки. Он может иметь одно из трех значений:

left (по левому краю),

right (по правому краю) и

center (по центру). По умолчанию – center.

Атрибут bgcolor определяет цвет фона для всех ячеек строки. Его значение можно указывать в символьной нотации или в формате RGB.

Атрибут valign определяет вертикальное выравнивание содержимого всех ячеек строки. Он может иметь следующие значения: top (по верхнему краю), middle (посередине) и bottom (по нижнему краю). По умолчанию – middle.

Спецификация тэга <td>:

<td align="выравнивание" bgcolor="цвет фона" colspan="количество ячеек" height="высота ячейки" rowspan=" количество ячеек " valign="вертикальное выравнивание" width="ширина ячейки" >

Атрибут align определяет выравнивание содержимого ячейки. Он может иметь одно из трех значений:

left (по левому краю),

right (по правому краю) и

center (по центру). По умолчанию – center.

Атрибут bgcolor определяет цвет фона для ячейки. Его значение можно указывать в символьной нотации или в формате RGB.

Атрибут colspan позволяет объединить несколько соседних ячеек по горизонтали. Значение этого атрибута – количество объединяемых ячеек.

Атрибут height определяет высоту ячейки в пикселях.

Атрибут rowspan позволяет объединить несколько соседних ячеек по вертикали. Значение этого атрибута – количество объединяемых ячеек.

Атрибут valign определяет вертикальное выравнивание содержимого ячейки. Он может иметь следующие значения: top (по верхнему краю), middle (посередине) и bottom (по нижнему краю). По умолчанию – middle.

Атрибут width определяет ширину ячейки в пикселях.

1.9. Графические элементы

Одним из достоинств HTML-документа является возможность использования графических элементов в оформлении. Можно выделить три элемента, чаще всего используемых в HTML-документах: горизонтальные линии, таблицы и рисунки.

1.10. Горизонтальные линии

Горизонтальные линии визуально подчеркивают законченность той или иной области документа. Сейчас часто используют рельефную, вдавленную линию, чтобы обозначить “объемность” документа.

Тэг <hr> позволяет провести рельефную горизонтальную линию в окне большинства браузеров. Этот тэг не является контейнером, поэтому не требует закрывающего тэга. До и после линии автоматически вставляется пустая строка.

Спецификация тэга <hr>:

<hr align="выравнивание" width="длина линии" size="толщина линии" noshade >

Атрибут align определяет способ выравнивания линии. Он может иметь следующие значения:

left – линия выравнивается по левому краю окна браузера. Это значение используется по умолчанию.

center – линия выравнивается по центру окна браузера.

right – линия выравнивается по правому краю окна браузера.

Атрибут width задает длину линии. Значением данного атрибута является число. Это число означает длину линии в пикселях. Если после числа стоит знак %, то это означает длину в процентах от ширины окна. Например:

<hr width="400"> – линия длиной 400 пикселей.

<hr width="50%"> – линия длиной 50 процентов от ширины окна.

Атрибут size задает толщину линии. Значением этого атрибута является число. Это число означает толщину линии в пикселях.

Атрибут noshade отменяет “трехмерность” линии.

1.11. Рисунки

Без иллюстраций документ скучен, вял и однообразен. HTML позволяет использовать рисунки в формате JPG и GIF для оформления HTML-документов. Для вставки рисунков используется тэг <img>.

Спецификация тэга <img>:

<img src="адрес рисунка" align="выравнивание" height="высота рисунка" width="ширина рисунка" border="толщина рамки" >

Атрибут src определяет URL-адрес рисунка, который будет отображаться браузером.

Атрибут align определяет способ выравнивания рисунка. Он может иметь следующие значения:

top – рисунок выравнивается по верхнему краю текущей строки.

middle – рисунок выравнивается серединой по текущей строке.

bottom – рисунок выравнивается по нижнему краю текущей строки.

left – рисунок прижимается к левому краю окна браузера и обтекается текстом.

right – рисунок прижимается к правому краю окна браузера и обтекается текстом.

Атрибут height определяет высоту рисунка в пикселях.

Атрибут width определяет ширину рисунка в пикселях.

Используя атрибуты height и width можно увеличивать или уменьшать рисунок. Если указать только один из этих атрибутов, то рисунок будет увеличен или уменьшен пропорционально и по ширине, и по высоте.

Атрибут border позволяет задавать рамку вокруг рисунка. Значение этого атрибута – толщина рамки в пикселях. По умолчанию – 1.

Пример выравнивания рисунков:

```
<p>Выравнивание<img src="ris1.jpg" align="top">по верхнему краю</p>
<p>Выравнивание<img src="ris1.jpg" align="bottom"> по нижнему краю</p>
<p>Выравнивание<img src="ris1.jpg" align="middle"> по середине</p>
```

1.11.1. Рисунок-ссылка

Можно использовать рисунки в качестве гиперссылок. Для этого нужно включить тэг IMG в содержание элемента A.

Например:

<img src="ris1.jpg">

1.12. HTML формы

HTML формы могут содержать такие элементы ввода как:

текстовые поля
флажки
переключатели
кнопки отправления и др.

Формы также могут содержать списки выбора, многострочные текстовые поля, метки и др.

Для того, чтобы создать форму в HTML используется элемент <form>.

Пример:
<form>
Введите ваше имя:
<input type="text" />
Введите пароль:
<input type="password" />
</form>

1.12.1. Элементы ввода

Элементы ввода используются для приема пользовательских данных.

Элементы ввода сильно отличаются друг от друга в зависимости от значения атрибута type.

Текстовое поле

<input type="text" /> определяет однострочное текстовое поле, в которое пользователь может вводить различную информацию.

<form>
 Введите ФИО в поля ниже:
Имя: <input type="text" name="firstname" />

```
Фамилия: <input type="text" name="lastname" /><br />
Отчество: <input type="text" name="lastname" />
</form>
```

Обратите внимание: по умолчанию текстовое поле вмещает 20 знаков.

Поле пароля

<input type="password" /> определяет поле для ввода пароля. Содержимое вводимое в данное поле закрывается черными кружками позволяя вводить пароли даже в присутствии посторонних.

```
<form>
Введите пароль: <input type="password" name="pass" />
</form>
```

Флажок

<input type="checkbox" /> определяет флажок. Флажки позволяют пользователям выбирать несколько пунктов с предварительно заполненной информацией из группы.

```
<form>
<p> Как вы относитесь к полетам в космос? </p>
<input type="checkbox" name="space" value="1" />
Положительно, всегда хотел полететь в космос<br />
<input type="checkbox" name="space" value="2" />
Безразлично, никогда не думал об этом серьезно <br />
<input type="checkbox" name="space" value="3" />
Отритцательно, меня с детства отталкивают мысли о космосе <br />
</form>
```

Переключатель

<input type="radio" /> определяет переключатель. Переключателипозволяют пользователям выбрать только один пункт с предварительно заполненной информацией из группы.

```
<form>
<p> Укажите Ваш пол: </p>
<input type="radio" name="s" value="m" /> Мужской<br />
<input type="radio" name="s" value="f" /> Женский
</form>
```

Кнопка отправления

<input type="submit" /> определяет кнопку отправления. После нажатия на данную кнопку данные введенные пользователем будут отправлены на сервер.

Адрес, на который будут пересылаться данные формы указывается в атрибуте тэга form - action. Если данный атрибут отсутствует данные будут отправлены на текущую страницу.

Обратите внимание: обработка данных полученных сервером в результате отправки форм будет производиться с помощью серверных языков (таких как ASP? PHP, Ruby или Python) и поэтому в данном учебнике рассмотрена не будет.

```
<form name="input" action="form.php" method="get">
Введите Ваше имя: <input type="text" name="name" />
<input type="submit" value="Отправить" />
</form>
```

Выпадающий список

Для создания выпадающих списков используется тэг <select>, а элементы выпадающего списка определяются с помощью тэга <option>.

Пользователи не любят вводить данные вручную, поэтому если Вы можете заменить обычное текстовое поля выпадающим списком, радио кнопками, или флажками обязательно сделайте это.

```
<p> Выберите ваш пол </p>
<form>
<select name="sex" >
<option value="m"> Мужской </option>
<option value="f"> Женский </option>
</select>
</form>
```

С помощью атрибута multiple Вы можете указать, что в выпадающем списке могут быть выбраны одновременно несколько элементов.

```
<p>В данном списке может быть одновременно выбрано несколько значений
(для этого зажмите клавишу Ctrl и щелкайте на необходимые элементы): </p>
<form>
<select name='city' multiple='multiple'>
<option value='london'>Лондон</option>
<option value='moskva'>Москва</option>
<option value='newyork'>Нью Йорк</option>
</select>
</form>
```

Заголовки в формах

Для того, чтобы озаглавить группу элементов формы, Вы должны с помощью тэга <fieldset> сгруппировать желаемую часть формы и затем с помощью тэга <legend> установить желаемое заглавие.

```
<form>
<fieldset>
<legend>Данные о пользователе</legend>
Имя: <br /><input type="text" name="firstname" /><br />
Фамилия:<br /> <input type="text" name="lastname" /><br />
Отчество: <br /><input type="text" name="lastname" /><br />
<p>Укажите Ваш пол:</p>
<input type="radio" name="s" value="m" /> Мужской<br />
<input type="radio" name="s" value="f" /> Женский
</fieldset>
<fieldset>
<legend> Анкета: </legend>
<p>Как Вы относитесь к полетам в космос:</p>
<input type="checkbox" name="space" value="1" />
Положительно, всегда хотел полететь в космос<br />
<input type="checkbox" name="space" value="2" />
Безразлично, никогда не думал об этом серьезно <br />
<input type="checkbox" name="space" value="3" />
Отрицательно, меня с детства отталкивают мысли о космосе <br />
</fieldset>
</form>
```

Использованные источники:

1. Учебники и справочники для веб-разработчиков: http://www.wisdomweb.ru/JS/bomdet.php.

1. CSS – КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ

1.1. Введение

CSS расшифровывается **C**ascading **S**tyle **S**heets (Каскадные Таблицы Стилей).

С помощью CSS Вы можете оформлять HTML документы.

CSS было представлено вместе с HTML 4.0 и создавалось для разрешения проблем с оформлением, возникающих в предыдущих версиях HTML.

Во время создания HTML в 1991 году не предполагалось, что он будет содержать тэги позволяющие оформлять документы.

Изначально HTML был создан только для группировки текста:

<h1> Заголовок 1 </h1>
 Какой-то текст
<h1> Заголовок 2 </h1>
 Какой-то текст

Когда в HTML были добавлены такие тэги как <font> оформление и разметка смешались в единое целое. Это было неудобно так как оформление элементов стало занимало большое количество времени.

Для того, чтобы разрешить эту проблему в конце 1996 года W3C (W3C - Консорциум Всемирной Паутины занимается разработкой веб-стандартов) представил CSS.

CSS позволяет хранить информацию об оформлении HMTL документа в отдельном внешнем файле с расширением .css. Редактируя лишь один этот файл Вы можете легко изменять оформление целого веб-сайта.

На данный момент CSS является стандартом оформления HTML документов и поддерживается всеми современными браузерами.

1.2. CSS синтаксис

Таблицы стилей состоят из набора правил(1). Каждое правило состоит из одного или нескольких селекторов(3) и блока определения(2) выделяющегося фигурными скобками.

Блок определения может содержать одно или несколько свойств (4) отделенных точкой с запятой (;) (после последнего свойства точка с запятой необязательна). Каждое свойство должно иметь значение (5) отделенное двоеточием (:).

1.3. CSS комментарии

Таблицы стилей могут содержать комментарии. Комментарии используются для создания пояснений.

Комментарии полностью игнорируются браузером при разборе таблиц стилей.

В CSS комментарии начинаются с "/*", и заканчиваются "*/"

Например:

```
/* Правило перекрасит абзацы HTML документа в зеленый цвет */
p {color:green;}
```

1.4. Селекторы CSS

С помощью селекторов Вы можете выбирать элементы на странице, которые хотите оформить.

В CSS существуют следующие виды селекторов:

- селекторы тэгов;
- селектор id;
- селектор class.

1.4.1. Селекторы тэгов

Вы можете выбирать элементы на странице для оформления по названию тэга.

Пример

```
p
{
color:green;
```

```
}
h2
{
color:red;
}
```

1.4.2. Селектор id

Данный вид селекторов позволяет производить более точную выборку и используется, когда необходимо выбрать только один определенный элемент на странице, с предварительно заданным идентификатором.

Идентификатор для элемента задается с помощью атрибута id (текст).

Для того, чтобы затем оформить данный элемент необходимо обратиться к идентификатору в таблицах стилей добавив перед ним символ "#" (#идентификатор {color:red}).

Пример:

```
/* Оформим элемент с id="test1" */
#test1
{
color:green;
font-family:verdana;
font-size:1.2em;
}
```

1.4.3. Селектор class

Данный вид селекторов позволяет выбирать для оформления не единственный элемент, а группу элементов.

С помощью атрибута class можно задать, что элемент относится к группе (текст).

Для того, чтобы затем оформить эту группу необходимо в таблицах стилей обратится к имени группы добавив перед ней символ "." (.имя_группы {color:red}).

Пример:

```
/* Свойства будут применены ко всем элементам с class="test1" */
.test1
{
color:green;
font-family:verdana;
font-size:1.2em;
}
```

Обратите внимание: имя группы и идентификатор может состоять только из латинских букв и не может начинаться с цифр.

1.4.4. Селекторы атрибутов

Элементы на странице могут быть выбраны по их атрибутам.

Пример
```
/* Оформит все элементы имеющие атрибут src */
[src]
{
border:solid green 3px;
}
```

Пример ниже оформит все элементы, которые ссылаются на главную страницу нашего сайта:

Пример
```
/* Оформит все элементы, атрибут href которых имеет значение
http://www.wisdomweb.ru */
[href="http://www.wisdomweb.ru"]
{color:green;}
```

1.4.5. Комбинирование селекторов

Для более точного выбора элементов в CSS может использоваться комбинирование селекторов.

Например, Вы можете комбинировать селекторы тэгов с селекторами class:

Пример
```
/* Свойства будут применены только к тем элементам с class="test1", которые
являются заголовками */
h2.test1
{
color:green;
font-family:verdana;
font-size:1.2em;
}
```

Также Вы можете комбинировать селекторы тэгов:

Пример
```
/* Свойства будут применены только к тем элементам p, которые находятся внутри
элементов div */
div p
{
color:green;
font-family:verdana;
```

```
font-size:1.2em;
}
```

Символ "+" позволяет выбирать элементы, которые идут сразу после указанного.

Пример

```
/* Свойства будут применены только к тем элементам p, которые идут сразу после элементов div */
div+p
{
color:green;
font-family:verdana;
font-size:1.2em;
}
```

1.4.6. Группировка селекторов

Часто при оформлении HTML документов с помощью CSS приходится применять одинаковые свойства для разных селекторов.

Например:

```
h1
{
font-family:verdana;
color:green;
}
h2
{
font-family:verdana;
color:green
}
p
{
font-family:verdana;
color:green;
}
```

Для того, чтобы сократить размер кода Вы можете группировать селекторы с одинаковыми свойствами разделяя их запятой:

Пример

```
h1,h2,p
{
font-family:verdana;
color:green;
}
```

1.5. Типы стилей

Существует три типа включения таблиц стилей в документ:

- Внешние таблицы стилей;
- Внутренние таблицы стилей;
- Встроенные стили.

1.5.1. Внешние таблицы стилей

Внешнее объявление стилей используется в случаях, когда оформление задается для группы связанных HTML документов (например, для целого веб-сайта).

В этом случае все оформление выносится в один внешний файл, на который должны ссылаться все документы веб-сайта.

Внешнее объявление стилей очень удобно так как позволяет редактируя лишь один файл изменять оформление целого веб-сайта.

Для того, чтобы подключить внешний файл стилей необходимо в секции head каждой страницы веб-сайта указать ссылку на него с помощью элемента <link>:

Пример:

```
<head>
<link rel="stylesheet" type="text/css" href="адрес_внешнего_файла_стилей" />
</head>
```

Внешний файл стилей является обычным текстовым файлом с расширением .css.

Пример содержимого внешнего файла стилей:

```
h1 {color:red;}
p {margin-right:38px;}
div {float:left;}
```

1.6. Внутренние таблицы стилей

Внутреннее объявление стилей используются в случаях, когда стиль нужно задать только для одного отдельного HTML документа.

В этом случае оформление определяется в секции head с помощью тэга style:

Пример:

```
<head>
<style type='text/css'>
h1 {color:red;}
p {margin-right:38px;}
div {float:left;}
```

```
</style>
</head>
```

1.6.1. Встроенные стили

Строковое объявление стилей используется в случаях, когда необходимо оформить только один определенный элемент в HTML документе.

Для того, чтобы оформить элементы этим способом Вы должны воспользоваться атрибутом style соответствующего элемента.

Атрибут style может содержать любые CSS свойства.

Пример:

```
<p style="font-size:1.3em"> Абзац оформленный с помощью CSS.</p>
```

1.6.2. Приоритет стилей

Стили подключенные разным способом имеют разный приоритет:

В общем, мы можем сказать, что все стили образуют новую "виртуальную" таблицу стилей по следующим правилам, из которых номер четыре имеет наивысший приоритет:

1. Используемые браузером по умолчанию
2. Внешние таблицы стилей
3. Внутренние таблицы стилей (в разделе head)
4. Встроенные стили (внутри HTML-элемента)

Если один HTML документ имеет несколько стилей присоединенных разными способами и в них заданы разные свойства оформления для одного и того же элемента, то в итоге элемент будет оформлен согласно содержимому стиля с более высоким приоритетом.

Пример

```
/* Внутренние стили: */
h1 {color:red}
/* Встроенные стили: */
h1 {color:green}
/* В результате заголовок будет зеленого цвета так как встроенные стили имеют
более высокий приоритет */
```

1.7. Текст в CSS

1.7.1. Цвет текста

С помощью свойства color можно изменять цвет текста HTML элементов.

Цвет может быть задан следующими способами:

- названия цвета – например, 'red' задаст красный цвет;
- значения RGB – например, 'rgb(255,255,255)' задаст белый цвет;
- шестнадцатеричного значения – например, '#00ff00' задаст зеленый цвет.

Пояснения:

Первый способ в основном используется для задания основных цветов, названия которых хорошо известны. Например red определит красный, blue - синий, white - белый.

Второй способ может использоваться для задания любых цветов и оттенков.

Синтаксис:
rgb (красный**,**зеленый**,**голубой**)**

красный число от 0 до 255 указывающее как много красного будет в итоговом оттенке.
зеленый число от 0 до 255 указывающее как много зеленого будет в итоговом оттенке.
голубой число от 0 до 255 указывающее как много голубого будет в итоговом оттенке.

Например, rgb(255,0,0) задаст красный цвет, а rgb(0,255,0) зеленый. Смешивая красный с зеленым rgb(255,255,0) мы получим желтый.

Третий способ по функциональности эквивалентен второму, но более компактен. На практике в основном используют именно этот способ.

Синтаксис:
#**красныйзеленыйголубой**

красный шестнадцатеричное число от 0 до ff указывающее как много красного будет в итоговом оттенке.
зеленый шестнадцатеричное число от 0 до ff указывающее как много зеленого будет в итоговом оттенке.
голубой шестнадцатеричное число от 0 до ff указывающее как много голубого будет в итоговом оттенке.

Например, #ff0000 задаст красный цвет, а #00ff00 зеленый. Смешивая красный с зеленым #ffff00 мы получим желтый.

Теперь попробуем перекрасить абзацы в зеленый цвет всеми перечисленными выше способами:

```
p {color:green;}
p {color:rgb(0,255,0);}
p {color:#00ff00;}
```

Удобно выбирать необходимые оттенки цвета можно с помощью подборщика цветов.

1.7.2. Выравнивание текста

С помощью свойства text-align можно выровнять текст элемента по горизонтали.

Текст может быть выровнен:
center -- по центру,
left -- по левому краю;
right -- по правому краю;
justify -- по ширине.

По ширине (justify) текст выравнивается путем добавления пробелов в строках до достижения одинаковой длины. Этот метод выравнивания часто используется в газетах и журналах.

Пример

```
p.ta1 {text-align:center;}
.ta2 {text-align:left;}
p.ta3 {text-align:right;}
p.ta4 {text-align:justify;}
```

1.7.3. Оформление текста - свойство text-decoration

С помощью свойства text-decoration можно оформить текст HTML элемента:

underline – подчеркнутым
line-through – перечеркнутым
overline -- отобразить над текстом элемента линию

Пример

```
p.td1 {text-decoration:underline;}
p.td2 {text-decoration:line-through;}
p.td3 {text-decoration:overline;}
```

Свойство text-decoration со значением none "очищает" текст от всех вышеперечисленных эффектов. Это может использоваться для создания не подчеркнутых ссылок.

Пример

```
a:link {text-decoration:none;}
a:visited {text-decoration:none;}
```

Не рекомендуется подчеркивать обычный текст, так как пользователи могут перепутать его со ссылкой.

1.7.4. Отступ между словами и буквами в тексте

С помощью свойства letter-spacing можно увеличивать или уменьшать отступ между буквами в тексте HTML элементов.

Пример
p.ls1 {letter-spacing:10px;}

С помощью свойства word-spacing можно увеличивать или уменьшать отступ между словами в тексте HTML элементов.

Пример
p.ws1 {word-spacing:15px;}

1.7.5. Остальные свойства оформления текста CSS

Свойство	Описание	Значения
direction	Устанавливает направление текста.	ltr rtl
line-height	Устанавливает высоту строки.	normal пиксели %
text-indent	Устанавливает величину отступа первого символа текста.	пиксели %
text-transform	Устанавливает регистр букв текста элемента.	none capitalize uppercase lowercase
vertical-align	Устанавливает вертикальное выравнивание элемента.	sub super top middle bottom пиксели %
white-space	Устанавливает как должны обрабатываться пробелы внутри элемента.	normal pre nowrap

1.8. Шрифт

1.8.1. CSS Font Families

В CSS существует два типа имен гарнитур шрифтов:

generic family - семейство шрифтов, выглядящих похоже (например, "Serif" или "Monospace")

font family – конкретный гарнитура шрифта (например,"Times New Roman" или "Arial")

Generic family	Font family	Описание
Serif	Times New Roman Georgia	Шрифты с засечками имеют маленькие линии на концах некоторых символов.
Sans-serif	Arial Verdana	"Sans" означает «без» - эти шрифты не имеют засечек.
Monospace	Courier New Lucida Console	Все символы имеют одинаковую ширину.

1.8.2. Задание начертания шрифта

Свойство font-family позволяет задавать шрифт для текста HTML элементов.

Если название шрифта состоит из нескольких слов, то оно обязательно должно заключаться в кавычки.

Браузер обрабатывает сначала первое значение, если такой шрифт не установлен в системе, то следующее. Если последнее значение соответствует групповому имени, а предшествующие не найдены, то выбирается имеющийся шрифт из группы.

Пример
p{font-family:"Times New Roman", Georgia, serif;}

1.8.3. Безопасные шрифты

Безопасными шрифтами называют шрифты, вероятность поддержки которых на любом компьютере с любой установленной ОС близка к 100%.

Список безопасных шрифтов:
Arial
Arial Black
Courier New
Comic Sans MS
Georgia
Impact
Times New Roman

Trebuchet MS
Verdana

Пример
```
<p style='font-family:Arial;'>Демонстрация шрифта Arial.</p>
<p style='font-family:"Arial Black";'>Демонстрация шрифта Arial Black.</p>
<p style='font-family:"Comic Sans MS";'>Демонстрация шрифта Comic Sans MS.</p>
<p style='font-family:"Courier New";'>Демонстрация шрифта Courier New.</p>
<p style='font-family:Georgia;'>Демонстрация шрифта Georgia.</p>
<p style='font-family:Impact;'> Демонстрация шрифта Impact.</p>
<p style='font-family:"Times New Roman";'>Демонстрация шрифта Times New
Roman.</p>
<p style='font-family:"Trebuchet MS";'>Демонстрация шрифта Trebuchet MS.</p>
<p style='font-family:Verdana;'>Демонстрация шрифта Verdana.</p>
```

1.8.4. Размер шрифта

CSS свойство **font-size** устанавливает размер шрифта HTML элементов. Размер шрифта можно задать двумя способами:

Абсолютный способ

Этот способ по сравнению со следующим удобнее, но его использование может привести к несоответствию при отображении одной и той же страницы в различных браузерах.

Данный способ позволяет задавать размер шрифта в абсолютных единицах таких как: пиксели (px) или проценты (%).

Пример
```
p.fz1 {font-size:20px;}
p.fz2 {font-size:30px;}
p.fz3 {font-size:13px;}
```

Относительный способ

Данный способ помогает избежать проблем с несоответствием при отображении страницы в разных браузерах, так как все размеры устанавливаются относительно.

Для задания размера шрифта данным способом используются единицы em. 1em эквивалентен размеру шрифта в браузере по умолчанию и равен 16px.

W3C рекомендует для задания шрифта использовать именно этот способ.

Пример
```
p.fz1 {font-size:1.2em;}
p.fz2 {font-size:1.5em;}
p.fz3 {font-size:1.15em;}
```

Не используйте свойство font-size для того, чтобы оформлять абзацы как заголовки. Всегда используйте для определения абзацев тэги p, а для заголовков h1-h6.

1.8.5. Стиль шрифта

Свойство font-style позволяет сделать шрифт HTML элемента курсивным.

Свойство font-weight позволяет изменять толщину шрифта.

Пример
```
p.italic {font-style:italic;}
p.fz1 {font-weight:bold;}
```

1.9. Оформление фона в CSS

В CSS существует группа свойств для оформления фона HTML элементов:

background-attachment
background-color
background-image
background-position
background-repeat

1.9.1. Цвет фона

CSS свойство background-color позволяет установить цвет фона для выбранного элемента.

Пример ниже делает цвет фона страницы зеленым:

Пример
```
body
{
background-color:green;
}
```

Цвет может быть задан следующими способами:

указав имя (например green задаст зеленый цвет);
указав RGB значения ('rgb(255,0,0)' задаст красный цвет);
задав шестнадцатеричного значения ("#0000ff" задаст синий цвет).

1.9.2. Изображение в качестве фона

Свойство background-image позволяет вставить произвольное изображение в качестве фона.

По умолчанию, изображение будет повторяться пока не заполнит все содержимое элемента.

Пример
```
body
{
background-image:url('http://www.wisdomweb.ru/editor/spider2.gif');
}
```

Обратите внимание: всегда тщательно подбирайте цвета. Фоновое изображение не должно сливаться с текстом.

Повторяющееся изображение в качестве фона это не всегда, то что нужно.

Свойства background-repeat позволяет определить как должно повторяться фоновое изображение при вставке:

background-repeat:repeat-x - изображение будет повторятся только по горизонтали;
background-repeat:repeat-y - изображение будет повторятся только по вертикали;
background-repeat:no-repeat - изображение не будет повторятся.

Пример
```
/* Фоновое изображение будет повторятся только по горизонтали */
body
{
background-image:url('http://www.wisdomweb.ru/editor/spider2.gif');
background-repeat:repeat-x;
}
```

Свойство background-position позволяет задавать местоположение фонового изображения.

В качестве первого значения данного свойства должна задаваться величина смещения изображения по горизонтали, а в качестве второго величина смещения по вертикали.

Величина смещения может быть указана как с помощью пикселей (px), процентов (%) и сантиметров (cm) (background-position:50px 30px;), так и с помощью предопределенных ключевых слов (background-position:right top;).

Пример
```
body
{
background-image:url('http://www.wisdomweb.ru/editor/spider2.gif');
background-repeat:no-repeat;
background-position:40px 60px;
}
```

Свойство background-attachment позволяет прикреплять фоновые изображения к определенным местам.

Прикрепленное изображение будет оставаться данном месте даже при прокрутке страницы.

Пример

```
/* Фоновая картинка будет оставаться на одном месте даже при прокрутке страницы
*/
body
{
background-image:url('http://www.wisdomweb.ru/editor/spider2.gif');
background-repeat:no-repeat;
background-attachment:fixed;
background-position:right top;
}
```

1.9.3. Краткая форма записи

Для того, чтобы сократить итоговый размер кода в CSS была предусмотрена краткая форма записи некоторых свойств.

Для краткой записи оформления фона элементов в CSS предусмотрено свойство background.

Пример

```
/* Фоновое изображение будет размещено в правом верхнем углу (в предыдущем
примере нам приходилось использовать 4 различных CSS свойства для того, чтобы
сделать тоже самое) */
body
{
background:url('http://www.wisdomweb.ru/editor/spider2.gif') no-repeat fixed right top;
}
```

При использовании стенографической формы записи соблюдайте следующий порядок следования свойств:

background-color
background-image
background-repeat
background-attachment
background-position

Обратите внимание: Вы можете пропускать неиспользуемые значения.

Список всех свойств оформления фона

Для того, чтобы узнать больше о желаемом свойстве щелкните по его названию.

Свойство	Описание	Значение
background	Устанавливает все возможные свойства фона	background-color background-image

	за одно определение	background-repeat background-attachment background-position
background-attachment	Указывает будет ли фоновая картинка привязана к одному месту, или будет прокручиваться вместе с текстом.	scroll fixed
background-color	Устанавливает фоновый цвет элемента.	rgb(x,x,x) #xxx название_цвета
background-image	Устанавливает фоновую картинку для элемента	url(*URL*) none
background-repeat	Указывает как фоновая картинка будет повторятся	repeat repeat-x repeat-y no-repeat
background-position	Указывает координаты расположения фоновой картинки.	left top left center left bottom center top center center center bottom right top right center right bottom x% y% xpos ypos

1.10.Оформление ссылок

Ссылки могут находится в четырех различных состояниях и каждое из этих состояний может быть отдельно оформлено с помощью специальных **псевдо-классов**.

Псевдо-класс должен добавляться к селектору элемента, отделяясь от него двоеточием (:).

Оформление для псевдо-классов должно задаваться в порядке перечисленном ниже.

- **a:link** - определяет оформление обычной не посещенной ссылки.
- **a:visited** - определяет оформление посещенной пользователем ссылки.

- **a:hover** - определяет оформление ссылки, на которую наведен курсор мыши.
- **a:active** - определяет оформление ссылки, на которую щелкнули мышкой.

Пример

```
a:link
{
text-decoration:none;
color:green;
}
a:visited
{
text-decoration:none;
color:green;
}
a:hover
{
text-decoration:underline;
color:red;
font-size:1.1em;
}
a:active
{
text-decoration:none;
color:red;
font-size:1.1em;
}
```

1.11.Оформление списков

CSS свойство list-style-type позволяет оформлять списки.

Всего существует два вида списков:

- Неупорядоченные - маркер таких списков имеет вид закрашенного круга.
- Упорядоченные - элементы таких списков нумеруется арабскими цифрами.

Ниже представлены таблицы с возможными значениями для оформления списков:

1.11.1. Значения для неупорядоченных списков:

none	Нет маркера.
disc	*Значение по умолчанию.* Маркер имеет вид закрашенного черного круга.
circle	Маркер принимает вид не закрашенного круга.
square	Маркер принимает вид закрашенного квадрата.

1.11.2. Значения для упорядоченных списков:

armenian	Элементы списка нумеруются Армянскими числами.
decimal	Значение по умолчанию. Элементы списка нумеруются Арабскими числами.
decimal-leading-zero	Элементы списка нумеруются Арабскими числами начинающимися с нуля (01, 02, 03 и т.д.).
georgian	Элементы нумеруются традиционной Грузинской нумерацией.
lower-greek	Элементы нумеруются традиционной Греческой нумерацией (alpha, beta, gamma).
lower-latin	Элементы нумеруются маленькими буквами латиницы (a, b, c, d).
lower-roman	Элементы нумеруются маленькими римскими цифрами (i, ii, iii, iv).
upper-latin	Элементы нумеруется большими буквами латиницы (A, B, C ,D).
upper-roman	Элементы нумеруется большими римскими цифрами (I, II, III, IV).

Пример
```
ul.lis1
{
list-style-type:square;
}
ol.lis2
{
list-style-type:upper-roman;
}
```

1.11.3. Использование маркера-картинки

Свойство list-style-image позволяет заменить маркер списка на произвольное изображение.

Пример
```
ul.lis1
```

```
{
list-style-image:url('marimg.gif');
}
```

1.12.Оформления таблицы

Пример

```
table, th, td
{
border-style:solid;
border-width:1px;
border-collapse:collapse;
padding:2px;
}
th
{
height:28px;
background-color:#f892dc;
color:white;
border-color:black;
}
.ts1
{
background-color:#ffeffb;
}
```

1.12.1. Ширина и Высота таблицы

С помощью CSS свойства width Вы можете устанавливать ширину таблицы.

В основном ширина устанавливается в пикселях или %, но можно также можете использовать cm и em.

Пример

```
.tab1
{
width:100%;
}
.tab2
{
width:70%;
}
.tab3
{
width:300px;
}
```

CSS свойство height позволяет установить высоту таблицы. Высота в основном указывается в пикселях, но можно также использовать cm и em.

Пример
```
.tab1
{
height:200px;
}
.tab2
{
height:7cm;
}
```

1.12.2. Оформление границ

Для оформления табличных границ в CSS используется свойство border.

Пример
```
table, th, td
{
border-style:solid;
border-color:green;
border-width:1px;
}
```

Обратите внимание: свойство border не является уникальным свойством таблиц оно может использоваться с любыми элементами. Данное свойство будет подробно рассмотрено далее.

1.12.3. Свойство border-collapse

Таблица в примере выше имеет двойную границу, потому что и сама таблица и ее ячейки (элементы th и td) имеют собственные границы.

Свойство border-collapse позволяет соединить границы таблицы и ячеек. Соединенные границы обычно смотрятся более аккуратно и красиво.

Пример
```
table, th, td
{
border-style:solid;
border-color:green;
border-width:1px;
border-collapse:collapse;
}
```

1.12.4. Выравнивание текста в таблице

С помощью свойства text-align Вы можете выравнивать текст табличных ячеек по горизонтали. Текст может быть выравнен:

По левому краю (значение left)

По правому краю (right)

По центру (center)

Пример
```
.tab1
{
text-align:right;
}
.tab2
{
text-align:left;
}
.tab3
{
text-align:center;
}
```

С помощью свойства vertical-align можно выравнивать текст в ячейках по вертикали. Текст может быть выравнен:

- По верхней границе (top)
- По центру (middle)
- По нижней границе (bottom)

Пример
```
.top
{
vertical-align:top;
}
.mid
{
vertical-align:middle;
}
.bot
{
vertical-align:bottom;
}
```

1.12.5. Свойство padding

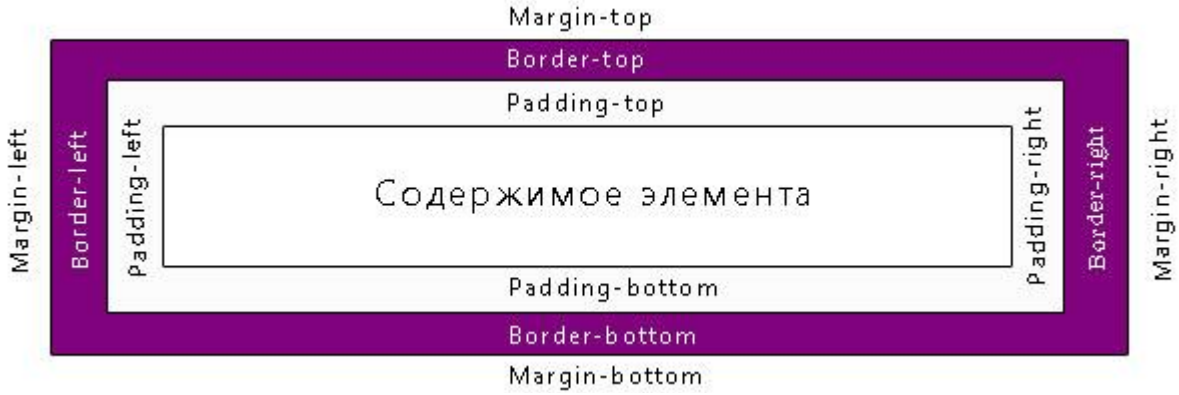
С помощью свойства padding Вы можете устанавливать величину отступа между границей ячейки и ее содержимым.

Пример
```
#tab1 td
{
padding:10px;
}
```

```
#tab2 td
{
padding:5px;
}
#tab3 td
{
padding:0px;
}
```

1.13.Блоковая модель

Все элементы в CSS являются прямоугольными блоками. Каждый такой блок имеет зону content, в которой располагается содержимое элемента (т.е. текст, изображения и т.д.). Вокруг зоны content могут располагаться необязательные зоны: padding, border и margin.



Зона padding окружает зону content. Данная зона используется для задания величины отступа содержимого элемента (зона content) от его границы (зона border). Зона может быть разбита на четыре части, которые могут оформляться независимо от друг друга: padding-top, padding-right, padding-bottom, padding-left.

Зона border окружает зону padding. Данная зона позволяет задать элементу границу произвольной ширины, стиля и цвета. Зона может быть разбита на: border-top, border-right, border-bottom, border-left.

Зона margin окружает зону border. Данная зона позволяет задать величину внешнего отступа данного элемента от окружающих. Зона может быть разбита на: margin-top, margin-right, margin-bottom, margin-left.

Пример использования блоковой модели

```
Пример
.ex1
{
border:10px red solid;
```

```
margin-left:50px;
margin-right:10px;
padding:15px;
}
.ex2
{
border:5px brown solid;
margin-top:30px;
margin-left:250px;
margin-right:70px;
padding:15px;
}
```

Правильное задание ширины и высоты элемента

Свойства width и height устанавливают ширину и высоту только блока content, а не элемента целиком.

Итоговый размер элемента помимо размеров content будет включать в себя еще и размеры padding, border и margin.

Пример

```
.ex1
{
width:400px;
}
.ex2
{
width:250px;
padding-left:85px;
padding-right:85px;
border-left:5px brown solid;
border-right:5px brown solid;
}
```

1.14.Границы

1.14.1. Стиль границ

Свойство border-style позволяет установить стиль для границ HTML элемента.

Значения для установки стилей границ:

- solid границы будут нарисованы сплошной линией;
- dashed границы будут нарисованы пунктирной линией;
- dotted границы будут нарисованы точками;
- double границы будут нарисованы двойной сплошной линией.

Пример

```
.bor1
{
border-style:solid;
}
.bor2
{
border-style:dashed;
}
.bor3
{
border-style:dotted;
}
```

Обратите внимание: по умолчанию граница элементов всегда невидима (значение none).

1.14.2. Цвет границы

С помощью CSS свойства border-color Вы можете задать цвет границы HTML элемента.

Цвет может быть задан следующими способами:

С помощью имени (например 'red' задаст красный цвет);

С помощью RGB значения (например 'rgb(255,255,255)' задаст белый цвет);

С помощью шестнадцатеричного значения (например '#00ff00' задаст голубой цвет).

Пример

```
.bor1
{
border-style:solid;
border-color:red;
}
.bor2
{
border-style:solid;
border-color:green;
}
.bor3
{
border-style:solid;
border-color:blue;
}
```

1.14.3. Толщина границы

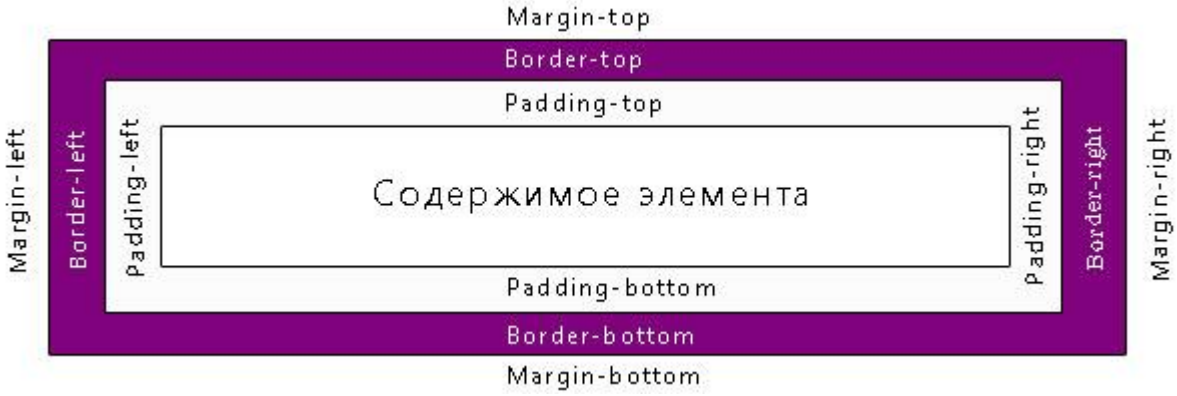
С помощью CSS свойства border-width Вы можете задать толщину границы HTML элемента.

Толщина может быть указана либо в пикселях, либо с помощью предопределенных значений: thin, medium, thick (тонкая, средняя, толстая).

Пример
```
.bor1
{
border-style:solid;
border-width:4px;
}
.bor2
{
border-style:solid;
border-width:2px;
}
.bor3
{
border-style:solid;
border-width:thin;
}
```

1.14.4. Задание стилей для отдельных сторон

Рассмотренные ранее свойства могут также применяться к отдельным сторонам границы. Их названия перечислены на рисунке ниже:



Пример
```
.bor1
{
border-top-style:solid;
border-width:2px;
}
.bor2
{
border-bottom-style:solid;
```

```
border-width:2px;
}
.bor3
{
border-left-style:solid;
border-right-style:solid;
border-width:2px;
}
```

Существует способ быстрого задания стилей границ. Например в результате применения border-style:dashed double solid groove; К верхней границе будет применено dashed, к правой double, к нижней solid, а к левой groove.

Если Вы укажете только два значения, например border-style:dashed double, то верхняя и нижняя граница будут оформлены как dashed, а левая и правая граница будут оформлены как double.

1.14.5. Последовательное задание свойств

Для того чтобы сократить длину кода, в CSS предусмотрен последовательную запись.

Такая форма записи объединяет все свойства оформления границ в одном свойстве border.

Пример

```
/* Вокруг элемента с классом bor1 будет отображена сплошная граница зеленого
цвета,  толщиной 2 пикселя */
.bor1
{
border:2px solid green;
}
```

Порядок следования свойств

border-width
border-style
border-color

Обратите внимание: Вы можете пропускать неиспользуемые свойства.

Пример

```
/* Вокруг элемента с классом bor1 будет отображена сплошная граница толщиной 1
пиксель */
.bor1
{
border:1px solid;
}
```

1.14.6. Внешняя граница

Помимо обычной границы элементы могут иметь еще и внешнюю границу, которая задается с помощью CSS свойства outline.

Пример
```
.out1
{
outline-style:dashed; /* определяет стиль внешней границы */
outline-color:green; /* определяет цвет внешней границы */
outline-width:4px; /* определяет ширину внешней границы */
border-style:solid;
}
```

1.15.Отступы

1.15.1. CSS Padding

В CSS существует два вида отступов: внутренние и внешние.

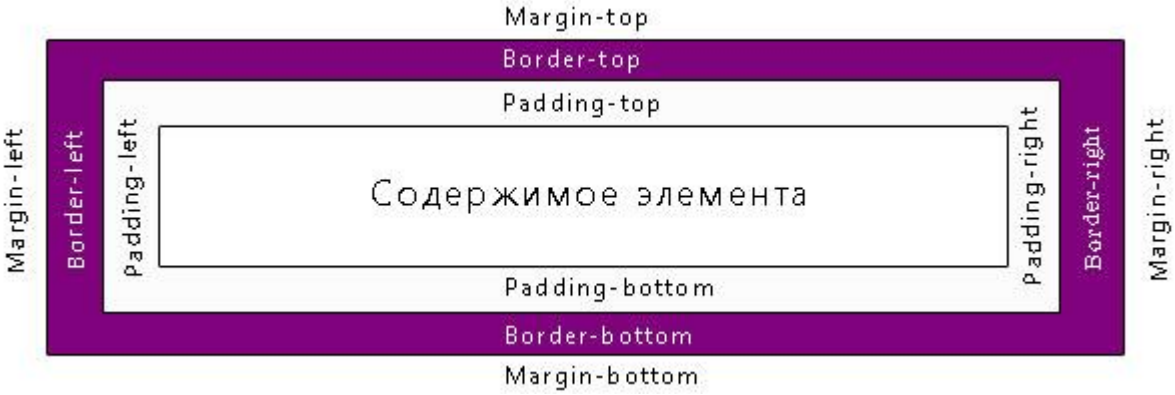
Внутренний отступ - это расстояние между содержимым элемента и его границей. Внешний отступ - это расстояние отступаемое с внешней стороны границы элемента.

Величина отступов может определяться в пикселях (px), сантиметрах (cm), процентах (%), em.

CSS свойство padding позволяет установить величину внутреннего отступа.

Пример
```
.pad1
{
border-style:solid;
padding:20px;
}
.pad2
{
border-style:solid;
padding:10px;
}
.pad3
{
border-style:solid;
padding:5px;
}
```

Величина отступа может быть независимо определена для каждой стороны элемента:



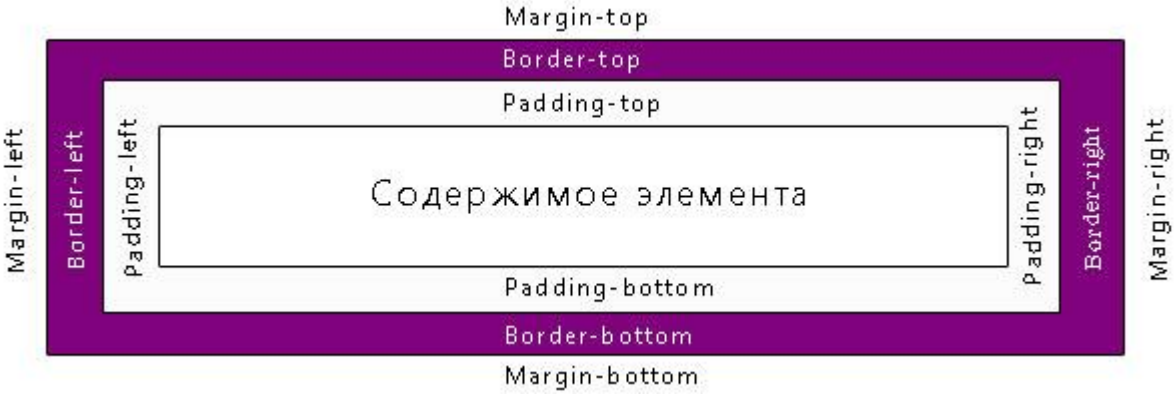
Пример
/* Отступ от содержимого элемента до его верхней границы равен 30, до левой 20, до нижней 10 и до правой 40 пикс. */
.pad1
{
padding-top:30px;
padding-left:20px;
padding-bottom:10px;
padding-right:40px;
}

1.15.2. CSS Margin

С помощью CSS свойства margin Вы можете задать величину внешнего отступа.

Пример
.mar1
{
margin:25px;
}
.mar2
{
margin:10px;
}
.mar3
{
margin:5px;
}

Величина внешнего отступа также может быть задана отдельно для каждой стороны элемента:



Пример
/* Отступ с верхней стороны элемента равен 20, снизу 40, слева 70, а справа 10 пикселям */
.mar1
{
margin-top:20px;
margin-bottom:40px;
margin-left:70px;
margin-right:10px;
}

1.15.3. Краткая форма записи

Для того, чтобы сократить размер кода Вы можете определять величину отступа для каждой стороны элемента используя краткую форму записи свойств.

Пример
/* Отступ от содержимого до границы элемента сверху будет равен 60, справа 20, снизу 40, а слева 50 пикселям */
.pad1
{
border:2px solid;
padding:60px 20px 40px 50px;
}
/* Отступ от содержимого до границы элемента сверху будет равен 40, слева и справа 30, а снизу 10 */
.pad2
{
border:2px solid;
padding:40px 30px 10px;
}
/* Отступ от содержимого до границы элемента сверху и снизу будет равен 40, а слева и справа 30 */
.pad3
{
border:2px solid;
padding:40px 30px;
}

Краткая форма записи может также использоваться с внешними отступами:

Пример
```
/* Внешний отступ сверху элемента будет равен 50, справа 20, снизу 40 и слева 50 пикселям */
.mar1
{
margin:50px 20px 40px 50px;
}
/* Внешний отступ сверху элемента будет равен 30, слева и справа 40, а снизу 50 пикселям */
.mar2
{
margin:30px 40px 50px;
}
/* Внешний отступ сверху и снизу элемента будет равен 30, а слева и справа 50 пикселям */
.mar3
{
margin:30px 50px;
}
```

1.16.Отображение

1.16.1. Скрытие элементов в CSS

В CSS скрыть элементы можно двумя способами:

С помощью свойства visibility:hidden;

С помощью display:none.

Элемент скрытый первым способом будет невидим, но будет по прежнему занимать место на странице.

Пример
```
/* Абзац стал невидим, но все еще занимает место на странице */
p.dis1
{
visibility:hidden;
}
```

Второй способ позволяет полностью скрыть элемент, чтобы он больше не занимал места на странице.

Пример
```
/* Абзац стал невидим и не занимает места */
```

```
p.dis1
{
display:none;
}
```

1.16.2. Блочные и строковые элементы в CSS

В CSS встречаются два типа элементов по способу отображения:

Блочные элементы полностью занимают всю ширину их родительского элемента. Примеры блочных элементов: p, h1-h6, div.

Строковые элементы занимают только необходимую им ширину. Примеры строковых элементов: a, span.

С помощью CSS Вы можете изменять способ отображения элементов. В следующем примере элемент p отображен как строковый:

Пример

```
/* Превратим блочный элемент p в строковый */
#dis1
{
display:inline;
}
```

Также Вы можете отображать строковые элементы как блочные:

Пример

```
/* Превратим строковый элемент a в блочный */
a.dis1
{
display:block;
}
```

1.17.Размещение элементов

С помощью свойств позиционирования Вы можете размещать элементы где пожелаете.

Пример

```
#pos1
{
position:absolute;
top:10px;
left:200px;
}
#pos2
{
position:absolute;
```

```
top:1px;
left:0px;
}
#pos3
{
position:absolute;
top:100px;
right:70px;
}
```

Местоположение элементов задается с помощью следующих CSS свойств:

- top - устанавливает величину смещения текущего элемента от верхнего края родительского элемента;
- bottom - устанавливает величину смещения текущего элемента от нижнего края родительского элемента;
- left - устанавливает величину смещения текущего элемента от левого края родительского элемента;
- right - устанавливает величину смещения текущего элемента от правого края родительского элемента.

Описанные выше свойства не вступят в силу пока Вы не зададите способ размещения. Также способ размещение определяет поведение данных свойств.

В CSS существуют 4 различных способа размещения элементов:

1.17.1. Статическое размещение

Статичные элементы всегда отображаются там, где они были объявлены. CSS свойства top, bottom, left и right не работают со статичными элементами.

Все элементы по умолчанию размещаются данным способом, но Вы также можете явно объявить элемент статичным с помощью position:static.

Пример

```
#pos1
{
position:static;
top:40px;
left:17px;
}
```

1.17.2. Фиксированное размещение

Фиксировано размещенные элементы не изменяют своего местоположения даже при прокрутке окна браузера. К фиксировано размещенным элементам могут применяться CSS свойства top, bottom, left, right.

Элемент может быть объявлен фиксировано размещенным с помощью position:fixed.

Пример
```
#pos1
{
position:fixed;
right:40px;
top:17px;
}
```

1.17.3. Относительное размещение

Относительно размещенные элементы размещаются относительно их обычной позиции.

Элемент может быть объявлен относительно размещенным с помощью position:relative.

Обратите внимание: элемент будет по прежнему занимать свою изначальную позицию.

Пример
```
#pos1
{
position:relative;
top:40px;
left:170px;
}
```

1.17.4. Абсолютное размещение

Абсолютно размещенные элементы располагаются относительно первого родительского элемента, способ размещения которого отличен от статического. Если такие элементы не были найдены, то элемент будет расположен относительно корневого элемента (html).

Иногда для того, чтобы добиться желаемого эффекта родительский элемент специально определяется как относительно размещенный с нулевым смещением.

Вы можете объявить элемент абсолютно размещенным с помощью position:absolute.

Пример
```
#pos1
{
position:absolute;
top:10px;
left:200px;
}
```

```
#pos2
{
position:absolute;
top:1px;
left:0px;
}
#pos3
{
position:absolute;
top:100px;
right:70px;
}
```

1.17.5. Наложение элементов

При применении свойств позиционирования элементы могут накладываться друг на друга. Свойство z-index позволяет установить какой элемент в случае наложения будет сверху, а какой снизу.

Элементы с большим значением свойства z-index располагаются выше других.

Обратите внимание: свойство z-index может принимать отрицательные значения.

Пример

```
#pos1
{
z-index:10;
}
#pos2
{
z-index:5;
}
#pos3
{
z-index:-1;
}
```

1.17.6. Связанные с размещением CSS свойства

Свойства	Описание	Значения
clip	Обрезает элемент размещенный абсолютно.	rect auto inherit
cursor	Задает вид, который будет принимать курсор при наведении на элемент.	auto crosshair default pointer

		move e-resize ne-resize nw-resize n-resize se-resize sw-resize s-resize w-resize text wait help
overflow	Устанавливает как должно быть отображено содержимое вышедшее за границы элемента.	auto hidden scroll visible inherit

1.18.Выравнивание

1.19.Центрирование с помощью margin

Блоковые элементы могут быть выравнены по центру установкой margin с левой и правой стороны значения auto.

Обратите внимание: если ширина блокового элемента будет равна 100%, то он не будет выравнен (т.к. доступного margin нет).

Пример
```
.ali1
{
margin-left:auto;
margin-right:auto;
width:50%;
}
```

1.19.1. Выравнивание с помощью свойств позиционирования

Вы можете выравнивать элементы с помощью свойств позиционирования.

Пример
```
.pos
{
position:absolute;
width:400px;
left:0px;
background-color:green;
}
```

1.19.2. Свойство float

Элементы в CSS также могут быть выравнены с помощью свойства float.

Элемент выравненный с помощью float будет прижат к левой или правой границе родительского элемента (в зависимости от заданного значения) и заставит следующие за ним элементы "обтекать" его с противоположной стороны.

Свойство float часто используют с картинками (как в примере ниже), но оно также бывает полезно при обычном выравнивании.

Пример
```
.fl1
{
float:right;
}
```

1.19.3. Очищение элементов от float

HTML элементы следующие за элементом с заданным float будут его "обтекать", чтобы избежать этого используйте свойство clear.

Значения переданные данному свойству (left или right) указывают с какой стороны находится элемент с float. Если Вы не уверены или элементы с float находятся с двух сторон воспользуйтесь свойством both.

Пример
```
.fl2
{
clear:both;
}
```

Использованные источники:

1. Учебники и справочники для веб-разработчиков: http://www.wisdomweb.ru/JS/bomdet.php.

1. JAVASCRIPT – ЯЗЫК СОЗДАНИЯ ИНТЕРАКТИВНЫХ ДОКУМЕНТОВ

1.1. Введение в JavaScript

JavaScript используется на множестве Web страниц для улучшиня дизайна, проверки правильности ввода данных в формы и многого другого. JavaScript разработан корпорацией Netscape и является самым популярным языком написания сценариев в Интернете.

JavaScript поддерживается всеми основными браузерами.

Что такое JavaScript?

- JavaScript был создан для придания интерактивности HTML страницам.
- JavaScript - это язык написания сценариев. Язык написания сценариев – это облегченный язык программирования.
- JavaScript представляет собой строки исполняемого компьютером кода.
- JavaScript обычно вставляется прямо в HTML страницы.
- JavaScript является интерпретируемым языком (означает что сценарии исполняются без предварительной компиляции).
- Для использования JavaScript не надо приобретать лицензию.
- JavaScript поддерживается всеми основными браузерами.

Java и JavaScript?

Java и JavaScript два совершенно разных языка!

Java (разработка Sun Microsystems) мощный и очень сложный язык программирования – относится к той же категории, что и C и C++.

Возможности JavaScript?

- **JavaScript дает HTML дизайнерам инструмент программирования –** создатели HTML страниц обычно не программисты, но язык JavaScript очень прост и имеет простой синтаксис! Почти каждый может добавить несколько строк кода на свою HTML страницу.
- **JavaScript позволяет динамически выводить текст на HTML страницу –** оператор JavaScript типа такого: document.write("<h1>" + name + "</h1>") может выводить различные текст на HTML страницу.
- **JavaScript позволяет реагировать на события -** JavaScript можно использовать для выполнения определенных действий на конкретные

события, например, когда страница заканчивает загружаться или когда пользователь щелкает HTML элемент.

- **JavaScript позволяет считывать или записывать HTML элементы –** с помощью JavaScript можно считывать и изменять содержимое HTML элементов.
- **JavaScript может использоваться для проверки правильности данных -** JavaScript может использоваться для проверки правильности вводимых в форму данных перед их передачей на сервер, что снижает нагрузку на сервер.

1.2. Как вставить сценарий на языке JavaScript

Для вставки сценариев на языке JavaScript в HTML страницу используется HTML тэг <script>.

1.3. Как вставить сценарий на языке JavaScript

В HTML тэг <script> используется для вставки JavaScript-сценариев в HTML-страницу.

Пример: «Как вывести на станице текст»

```
<html>
<body>
<script type="text/javascript">
document.write("Привет!")
</script>
</body>
</html>
```

Пояснение примера:
Атрибут type тэга <script> указывает браузеру какой скриптовый язык мы встраиваем. В случае если это JavaScript атрибут type должен иметь значение text/javascript;
document.write() - это команда JavaScript позволяющая выводить произвольное содержимое на страницу;
Разместив document.write() между тэгами <script> и </script> мы сообщаем браузеру обрабатывать ее как команду JavaScript, поэтому после загрузки страницы браузер выведет: *"Привет!"*

Пример: «Вывод форматированного текста с тегами HTML»

```
<html>
<body>
<script type="text/javascript">
document.write("<h1>Добро пожаловать!</h1>")
</script>
</body>
</html>
```

1.4. Где вставить сценарий на языке JavaScript

Сценарии на JavaScripts в разделе body будут выполняться по мере загрузки страницы.

Сценарии на JavaScripts в разделе head будут выполнены при вызове на страницы.

Сценарии в разделе HEAD

Сценарии содержащие функции располагаются в разделе HEAD HTLM-документа. Таким образом, можно быть уверенным, что сценарии уже загружены к моменту вызова функции.

Пример:

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("Это окно-предупреждение вызывается событием onload")
}
</script>
</head>
<body onload="message()">
</body>
</html>
```

Сценарии в разделе BODY

Выполняются сценарий, расположенный в разделе BODY.

Пример:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
document.write("Это сообщение выводится во время загрузки страницы")
</script>
</body>
</html>
```

Внешние сценарии

Как сделать доступным внешний сценарий.

Пример:

```
<html>
<head>
</head>
<body>
<script src="xxx.js">
</script>
<p>Рабочий сценарий находится во внешнем файле сценариев "xxx.js".</p>
</body>
</html>
```

1.5. Переменные в JavaScript

Переменная является "контейнером" для хранения данных.

Каждая JavaScript переменная должна иметь собственное уникальное имя, которое может начинаться с латинской буквы или символа "_".

Имя переменных в JavaScript не может начинаться с цифр.

Так как JavaScript чувствителен к регистру переменные с одинаковыми именами написанными в разном регистре (например var и VAR) будут являться разными переменными.

Создание переменных в JavaScript часто называют ″объявлением″ переменных.

Переменные в JavaScript объявляются с помощью команды var.

Пример

```
<html>
<body>
<script type="text/javascript">
var name = "Hege"
document.write(name)
document.write("<h1>"+name+"</h1>")
</script>
<p>В этом примере переменная объявляется, инициализируется (приписывается значение) и выводится на экран.</p>
<p>Далее переменная выводится еще раз, но в качестве заголовка.</p>
</body>
</html>
```

1.6. Условные операторы

1.6.1. Оператор if...else в JavaScript

Условные операторы в JavaScript используются для выполнения различных действий в зависимости от различных условий.

iIf оператор

Синтаксис:
```
if (условие)
{
код, выполняемый если условие истинно
}
```

Пример
```
<html>
<body>

<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time < 10)
{
document.write("<b>Доброе утро!</b>")
}
</script>
<p>Пример демонстрирует оператор If.</p>
<p>Если время на вашем компьютере ранее 10 часов, вы получите приветствие "Доброе утро!".</p>
</body>
</html>
```

if...else оператор

Синтаксис:
```
if (условие)
{
код, выполняемый если условие истинно
}
else
{
код, выполняемый если условие ложно
}
```

Пример:
```
<html>
```

```
<body>
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time < 10)
{
document.write("<b>Доброе утро!</b>")
}
else
{
document.write("<b>Добрый день!</b>")
}
</script>
<p>Пример демонстрирует оператор If...Else.</p>
<p>Если время на вашем компьютере ранее 10 часов, вы получите приветствие
"Доброе утро!". В противном случае приветствие" Добрый день!".</p>
</body>
</html>
```

1.6.2. if..else if...else оператор

Синтаксис:

```
if (условие1)
{
код, выполняемый если условие1 истинно
}
else if (условие2)
{
код, выполняемый если условие2 истинно
}
else
{
код, выполняемый если условие1 и условие2 ложны
}
```

Пример:

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Доброе утро!</b>")
}
else if (time>10 && time<16)
{
```

```
document.write("<b>Добрый день!</b>")
}
else
{
document.write("<b>Привет!</b>")
}
</script>

<p>Пример демонстрирует оператор if..else if...else.</p>

</body>
</html>
```

Пример сценария «Случайная ссылка»

Этот пример демонстрирует случайную ссылку, когда вы кликаете на ссылке происходит переход или на www.w3schools.com или на www.microsoft.ru. Выбор ссылки происходит с вероятностью 50% для каждой.

Пример:

```
<html>
<body>
<script type="text/javascript">
var r=Math.random()
if (r>0.5)
{
document.write("<a href='http://www.w3schools.com'>Учитесь Web
программированию!</a>")
}
else
{
document.write("<a href='http://www.microsoft.ru'>Узнай больше о
Microsoft!</a>")
}
</script>
</body>
</html>
```

1.6.3. Оператор switch

Условный оператор в JavaScript используемый для выполнения различных действий в зависимости от различных условий.

Синтаксис:

```
switch(n)
{
case 1:
```

```
  выполнение блока кода 1
  break
case 2:
  выполнение блока кода 2
  break
default:
  выполнение блока кода если n отличается
  от case 1 and 2
}
```

Пример:

```
<html>
<body>

<script type="text/javascript">
var d = new Date()
theDay=d.getDay()
switch (theDay)
{
case 5:
  document.write("<b>Наконец пятница!</b>")
  break
case 6:
  document.write("<b>Ура, суббота!</b>")
  break
case 0:
  document.write("<b>Отоспимся в воскресенье</b>")
  break
default:
  document.write("<b>I'm really looking forward to this weekend!</b>")
}
</script>
<p>Это сценарий JavaScript генерирует различные приветствия в зависимости от дня недели. Замечание Воскресенье=0, Понедельник=1, Вторник=2 и т.д.</p>
</body>
</html>
```

1.7. Циклы в JavaScript

Циклы в JavaScript используются для выполнения одного и того же блока кода или заданное количество раз или пока выполняется определенное условие.

Цикл For

Создание цикла FOR. Цикл FOR используется, когда нужно выполнить блок кода заданное количество раз.

Синтаксис**:**

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
   исполняемый код
}
```

Пример

```
<html>
<body>
<script type="text/javascript">
for (i = 0; i <= 5; i++)
{
document.write("Число " + i)
document.write("<br>")
}
</script>
<p>Пояснения:</p>
<p>Цикл начинается с i=0.</p>
<p>До тех пор пока <b>i</b> меньше чем, или равно 5, цикл продолжает
выполняться.</p>
<p><b>i</b> увеличивается на 1 при каждом проходе цикла.</p>
</body>
</html>
```

Пример «Циклический перебор HTML заголовков»

```
<html>
<body>
<script type="text/javascript">
for (i = 1; i <= 6; i++)
{
document.write("<h" + i + ">Это заголовок " + i)
document.write("</h" + i + ">")
}
</script>
</body>
</html>
```

Цикл while

Используйте цикл WHILE, когда нужно выполнить блок кода пока выполняется определенное условие.

Синтаксис:

```
while (var<=endvalue)
{
   исполняемый код
}
```

Пример

```
<html>
<body>
<script type="text/javascript">
i = 0
while (i <= 5)
{
document.write("The number is " + i)
document.write("<br>")
i++
}
</script>
<p>Пояснения:</p>
<p><b>i</b>  равен 0.</p>
<p>Пока <b>i</b> менее или равен 5, цикл продолжает выполняться.</p>
<p><b>i</b> будет увеличиваться на 1 при каждом проходе цикла.</p>
</body>
</html>
```

Цикл do while

Используется DO...WHILE цикл для выполнения блока кода пока выполняется определенное условие. Этот цикл выполняется хотя бы один раз, даже если условие ложно, поскольку операторы выполняются прежде чем проверяется истинность условия.

Синтаксис:

```
do
{
   исполняемый код
}
while (var<=endvalue)
```

Пример

```
<html>
<body>
<script type="text/javascript">
i = 0
do
{
document.write("Число " + i)
document.write("<br>")
i++
}
while (i <= 5)
</script>
<p>Пояснения:</p>
<p><b>i</b>  равно 0.</p>
```

```
<p>Цикл выполняется.</p>
<p><b>i</b> будет увеличиваться на 1 при каждом проходе цикла.</p>
<p>Пока <b>i</b> меньше чем или равно 5, цикл будет продолжать
выполняться.</p>
</body>
</html>
```

1.8. Функции в JavaScript

Функция это многократно используемый блок кода. Функция исполняется, когда происходит некое событие или вызывается функция.

Функции без аргументов.

Синтаксис:

```
function functionname()
{
код
}
```

Пример

```
<html>
<head>

<script type="text/javascript">
function myfunction()
{
alert("Привет!")
}
</script>
</head>
<body>
<form>
<input type="button"
onclick="myfunction()"
value="Вызов функции">
</form>
<p>Функция вызывается при нажатии кнопки. Функция выводит окно-
предупреждение.</p>
</body>
</html>
```

Функция с аргументами

Синтаксис:

```
function functionname(var1,var2,...,varX)
{
```

```
код
}
```

Пример:

```
<html>
<head>
<script type="text/javascript">
function myfunction(txt)
{
alert(txt)
}
</script>
</head>
<body>
<form>
<input type="button"
onclick="myfunction('Привет')"
value="Вызов функции">
</form>
<p>При нажатии кнопки вызывается функция с аргументами. Функция выводит
окно-предупреждение с передаваемым аргументом.</p>
</body>
</html>
```

Функция с аргументами

Пример:

```
<html>
<head>
<script type="text/javascript">
function myfunction(txt)
{
alert(txt)
}
</script>
</head>
<body>
<form>
<input type="button"
onclick="myfunction('Доброе утро!')"
value="Утром">
<input type="button"
onclick="myfunction('Добрый вечер!')"
value="Вечером">
</form>
<p>При щелчке мышью на одной из кнопок вызывается функция. Функция
выводит окно-предупреждение с передаваемым аргументом.</p>
```

```
</body>
</html>
```

Функция без аргументов, которая возвращает значение

Синтаксис**:**

```
function functionname(var1,var2,...,varX)
{
returnvalue=…
return returnvalue
}
```

Пример:

```
<html>
<head>
<script type="text/javascript">
function myFunction()
{
return ("Привет! Чудесный день!")
}
</script>
</head>
<body>
<script type="text/javascript">
document.write(myFunction())
</script>
<p>Сценарий в разделе BODY вызывает функцию.</p>
<p>Функция возвращает текст.</p>
</body>
</html>
```

Функция с аргументами, возвращающая значение

Пример показывает как создать функцию вычисляющую сумму двух аргументов и возвращающую результат.

Пример:

```
<html>
<head>
<script type="text/javascript">
function total(a,b)
{
return a*b
}
</script>
</head>
<body>
<script type="text/javascript">
```

```
document.write(total(4,3))
</script>
<p>Сценарий в разделе вызывает функцию с двумя аргументами (4 и 3).</p>
<p>Функция возвращает произведение этих двух аргументов.</p>
</body>
</html>
```

1.9. Всплывающие окна JavaScript

В JavaScript можно создать три вида всплывающих окон: окно предупредительных сообщений (Alert box), окно подтверждения действия (Confirm box) и окно ввода информации (Prompt box).

Окно-предупреждение (Alert box)

Пример

```
<html>
<head>
<script type="text/javascript">
function disp_alert()
{
alert("Я окно - предупреждение!!")
}
</script>
</head>
<body>
<form>
<input type="button" onclick="disp_alert()" value="Вывод окна-предупреждения">
</form>
</body>
</html>
```

Окно-подтверждение (Confirm box)

```
<html>
<head>
<script type="text/javascript">
function disp_confirm()
{
var name=confirm("Нажмите кнопку")
if (name==true)
{
document.write("Вы нажали кнопку OK!")
}
else
{
document.write("вы нажали кнопку Cancel!")
```

```
}
}
</script>
</head>
<body>
<form>
<input type="button" onclick="disp_confirm()" value="Вывод окна-
подтверждения">
</form>
</body>
</html>
```

Окно ввода (prompt box)

Пример:

```
<html>
<head>
<script type="text/javascript">
function disp_prompt()
{
var name=prompt("Введите ваше имя","")
if (name!=null && name!="")
{
document.write("Привет " + name + "! Как дела?")
}
}
</script>
</head>
<body>
<form>
<input type="button" onclick="disp_prompt()" value="Вывод окна ввода">
</form>
</body>
</html>
```

1.10. Прерывание (Break) и продолжение (Continue) в JavaScript

Существует два специальных оператора, которые используются внутри циклов: break (прервать) и continue (продолжить).

Break оператор

Используется для прерывания цикла.

Пример

```
<html>
<body>
```

```
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3){break}
document.write("Число " + i)
document.write("<br />")
}
</script>
</table>
<p>Пояснения: Цикл будет прерван при i=3.</p>
</body>
</html>
```

Оператор continue

Используется оператор CONTINUE что бы прервать текущий цикл и продолжить со следующим значением параметра цикла.

Пример

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3){continue}
document.write("Число" + i)
document.write("<br />")
}
</script>
</table>
<p>Пояснения: Цикл будет прерван при i=3 и будет продолжен со следующим значением i.</p>
</body>
</html>
```

1.11. Особенности JavaScript

Несколько важных особенностей о написании сценариев на JavaScript.

JavaScript чувствителен к регистру: функция названная "myfunction" не та же самая функция что "myFunction" и переменная названная "myVar" не та же самое что "myvar".

Символы

Открывающие символы, такие как ({ [" ', должны всегда иметь соответствующий закрывающий символ, такой как ' "] }).

Пробелы

В JavaScript игнорируются дополнительные пробелы. Можно добавлять пробелы для большей удобочитаемости сценариев.

Прерывание строки кода

Можно разорвать строку кода только в пределах текстовой строки. Для этого используйте обратную косую черту.

Пример:
```
document.write("Hello \
World!")
```

Тем не менее, нельзя разорвать строку кода вот так:
```
document.write \
("Hello World!")
```

В этом случае будет сгенерирована ошибка!

Вставка специальных символов

Специальные символы (такие как " ' ; &) могут быть вставлены с помощью обратной косой черты:

Например:
```
document.write ("You \& I sing \"Happy Birthday\".")
```

В этом случае будет выведена строка:
You & I sing "Happy Birthday".

Комментарий

Однострочный комментарий начинается с двух наклонных черт "//":

```
sum=a + b  //это комментаририй
```

Для многострочного комментария используются /* и */:

```
/* This is a comment
block. It contains
several lines*/
```

2. Объектно-ориентированное программирование в JavaScript. Основные объекты JavaScript

Объектно ориентированное программирование - это стиль программирования ориентированный на работу с объектами.

В реальном мире мы можем рассматривать как объекты все, что нас окружает. Объектом может быть: компьютер, машина, самолет, дом, гитара. В JavaScript объектами являются строки, массивы, регулярные выражения и т.д.

Объекты могут иметь свойства и методы.

Обратите внимание: в данной главе будут рассмотрены только встроенные объекты.

Свойства объектов в JavaScript

Свойства являются значениями, которые связаны с объектами.

Объекты в JavaScript также могут иметь свойства, например объект массив имеет свойство length позволяющее узнать количество элементов в этом массиве.

При обращении к свойству объекта необходимо отделить его точкой от названия объекта (объект.свойство). Если название свойства состоит из двух (или более) слов необходимо удалить пробел между ними и начать второе слово с заглавной буквы или заменить этот пробел на «_» (знак нижнего подчеркивания).

Пример
```
document.write('Массив содержит '+x.length+' элемента.')
```

Методы объектов в JavaScript

Методы являются действиями, которые могут быть совершены над объектами.

Объекты в JavaScript также могут иметь методы, например объект массив имеет метод reverse() позволяющий изменять порядок следования элементов в массиве на противоположный:

2.1. Массивы в JavaScript

Массивы представляют собой контейнеры, в которых может быть сохранено неограниченное количество переменных.

Каждому значению, которое заносится в массив присваивается уникальный идентификатор, по которому Вы затем сможете обращаться к данному элементу внутри массива.

2.1.1. Создание массивов

Вы можете создать массивы тремя разными способами:

Первый способ:

```
// Создаем список вещей с помощью массива
item = new Array();
item[0] = "Автомобиль";
item[1] = "Микроволновая печь";
item[2] = "Стиральная машина";
item[3] = "Пылесос";
```

Второй способ:

```
var item = new Array("Автомобиль", "Микроволновая печь", "Стиральная машина", "Пылесос");
```

Третий способ:

```
var item = ["Автомобиль", "Микроволновая печь", "Стиральная машина", "Пылесос"];
```

2.1.2. Доступ к переменным в массиве

Для того, чтобы обратиться к элементу сохраненному в массиве необходимо указать имя массива и индекс желаемого элемента в квадратных скобках (массив[индекс]).

Обратите внимание: нумерация индексов в массивах начинается не с 1, а с 0.

Пример

```
//Создадим массив
var item = ["Автомобиль","Микроволновая печь","Стиральная машина", "Пылесос"];
//Выведем значение первого элемента массива
document.write(item[0]+'<br />');
//Выведем значение третьего элемента массива
document.write(item[2]+'<br />');
```

Значения элементов в массивах могут быть изменены.

Пример

```
//Создадим массив
var item = ["Автомобиль", "Микроволновая печь", "Стиральная машина", "Пылесос"];
//Изменим значение третьего элемента
item[2] = "Снегоход";
//Выведем измененное значение
document.write(item[2]);
```

2.1.3. Свойства объекта Array

С помощью свойства length Вы можете узнать количество элементов в массиве.

Пример
```
//Создадим массив
var item = ["Автомобиль", "Микроволновая печь", "Стиральная машина",
"Пылесос"];
//Выведем количество элементов массива
document.write(item.length);
```

2.1.4. Некоторые методы объекта Array

Метод concat()

С помощью метода concat**()** Вы можете объединить два и более массива в один.

Пример
```
//Создадим 1 массив
var item1=["Автомобиль", "Микроволновая печь"];
//Создадим 2 массив
var item2=["Стиральная машина"];
//Создадим 3 массив
var item3=["Пылесос"];
//Объединим массивы и запишем результат в переменную item
item=item1.concat(item2,item3);
//Выведем содержимое массива item на страницу.
var i;
for (i in item) {
  document.write(i + " элемент массива: " + item[i] + "<br />");
}
```

Метод sort()

С помощью метода sort() Вы можете отсортировать значения массива.

Пример
```
//Создадим массив
var item=["Автомобиль", "Микроволновая печь", "Стиральная машина",
"Пылесос"];
//Отсортируем массив
item.sort();
//Выведем отсортированное содержимое массива item на страницу.
var i;
for (i in item) {
  document.write(i + " элемент массива: " + item[i] + "<br />");
}
```

Метод pop()

С помощью метода pop() Вы можете удалить последний элемент массива.

Пример

```
//Создадим массив
var item=["Автомобиль", "Микроволновая печь", "Стиральная машина",
"Пылесос"];
//Удалим последнее значение массива и сохраним ее в переменной del
del=item.pop();
//Выведем удаленный элемент
document.write("Удаленный элемент: " + del + "<br />");
//Выведем содержимое массива item на страницу.
var i;
for (i in item) {
  document.write(i+ " элемент массива: " + item[i] + "<br />");
}
```

Метод slice (начало, конец)

Позволяет извлечь (вырезать) элементы из массива и создать из извлеченных элементов новый массив.

начало указывает индекс элемента, с которого начнется извлечение.

конец указывает индекс элемента массива, на котором закончится извлечение. Если данный параметр не будет задан извлечение будет проведено до конца строки.

Пример

```
//Создадим массив
var item=["Автомобиль", "Микроволновая печь", "Стиральная машина",
"Пылесос"];
//Извлечем два последних элемента массива и создадим из них массив sl1
sl1=item.slice(2);
//Выведем содержимое sl1
document.write(sl1 + "<br />");
//Извлечем два первых элемента массива и создадим из них массив sl2
sl2=item.slice(0,2);
//Выведем содержимое sl2
document.write(sl2 + "<br />");
//Извлечем последний элемент массива и и создадим из него массив sl3
sl3=item.slice(-1);
//Выведем содержимое sl3
document.write(sl3);
```

2.1.5. Некоторые часто используемые методы объекта Array

Метод	Описание
concat()	Объединяет массивы и возвращает их объединенную копию.
join()	Соединяет все элементы массива в строку.
pop()	Удаляет последний элемент массива и возвращает его значение.
push()	Добавляет новый элемент в конец массива и возвращает его новую длину (массива).
reverse()	Изменяет порядок следования элементов в массиве на противоположный (последний элемент становится первым, первый последним и т.д.).
shift()	Удаляет первый элемент массива и возвращает его значение.
slice()	Извлекает часть элементов массива и создает из них новый массив.
sort()	Сортирует элементы массива.
toString()	Преобразует массив в строку.
valueOf()	Возвращает содержимое массива.

2.2. Объект String

Объект String (строковый объект) используется для хранения и обработки текстовой информации.

Синтаксис:
```
// Создание объекта String (первый вариант)
x=new String("произвольный текст");
// Создание объекта String (второй вариант)
x="произвольный текст";
// или
x='произвольный текст';
```

2.2.1. Свойства объекта String

С помощью свойства length Вы можете узнать длину строки.

Пример
```
//Создадим строку
var x=new String('Это строка произвольного текста.');
// Узнаем длину строки (количество символов учитывая пробелы) и выведем ее на страницу
document.write(x.length);
```

2.2.2. Методы объект String

Методы toUpperCase() и toLowerCase()

С помощью метода toUpperCase() Вы можете перевести все символы текста в верхний регистр, а с помощью toLowerCase() в нижний.

Пример

```
//Создадим строку
x='ЭтО СтрОка';
//Выведем строку без изменений
document.write(x + '<br />');
//Переведем все буквы строки x в верхний регистр и выведем результат на страницу
document.write(x.toUpperCase()+'<br />');
//Переведем все буквы строки x в нижний регистр и выведем результат на страницу
document.write(x.toLowerCase()+'<br />')
```

Не забудьте о важности регистра букв в JavaScript при обращении к методам, а также не забывайте о круглых скобках (), которые должны добавляться после их названия.

Метода concat()

С помощью метода concat() Вы можете объединить две и более строки и вывести результат на страницу.

Пример

```
//Создадим несколько строк
x='Эти строки текста  ';
y='будут объединены ';
z='с помощью метода <b>concat</b>.';
//Объединим эти строки и выведем результат на страницу
document.write(x.concat(y,z));
```

С помощью метода replace() Вы можете заменить одно произвольное слово в строке на другое.

Пример

```
//Создаем строку
x='<b>HTML</b> - это скриптовый объектно ориентированный язык программирования. ';
//Заменим слово 'HTML' на 'JavaScript' в строке x и выведем результат на страницу
document.write(x.replace('HTML','JavaScript'));
```

2.2.3. Некоторые часто используемые методы объекта String

Метод	Описание
charAt()	Возвращает символ находящийся на указанной позиции в строке.
charCodeAt()	Возвращает Юникод символа на указанной позиции в строке.
concat()	Объединяет две и более строки и возвращает результат.
indexOf()	Возвращает позицию первого найденного совпадения выражения в методе со строкой текста.
fontcolor()	Возвращает строку заданного цвета
lastIndexOf()	Возвращает позицию последнего найденного совпадения выражения в методе со строкой текста.
match()	Ищет совпадение между регулярным выражением переданным в методе и строкой текста и возвращает найденное совпадение (если оно имеется).
replace()	Ищет совпадение между регулярным выражением и строкой текста и заменяет найденное совпадение (если оно имеется) новой строкой.
search()	Ищет совпадение между регулярным выражением и строкой текста и возвращает позицию найденного совпадения (если оно имеется).
slice()	Извлекает часть существующей строки и возвращает новую строку.
split()	Разделяет строку на массив подстрок.
substr()	Извлекает указанное количество символов из строки начиная с указанного места.
substring()	Извлекает символы из строки между указанными координатами.
toLowerCase()	Делает все символы строки строчными.
toUpperCase()	Делает все символы строки заглавными.

2.3. Объект Date

Объект Date позволяет производить различные операции с датой и временем.

Синтаксис:

```
//Определим текущую дату и запишем ее в переменную x
x=new Date();
//При выводе на страницу текущая дата (24 Декабря 2010) будет выглядеть
//следующим образом:
Fri Dec 24 2010 22:15:31 GMT+0600
/* Fri = Friday (Пятница) - обозначает текущий день недели
   Dec = December (Декабрь) - обозначает текущий месяц
   24 - обозначает день месяца
   2010 - обозначает год
   22:15:31 - текущее время
   GMT+0600 - смещение времени от Гринвича +6 часов
*/
```

Помимо определения текущей даты Вы можете определить произвольную дату.

Синтаксис:
// Первый способ:
x=new Date(год, месяц, день, час, минута, секунда, миллисекунда);
// 20 Декабря 1999 этим способом можно определить так (нумерация месяцев начинается с 0):
d=new Date(1999,11,20);
// Второй способ:
x=new Date(миллисекунды);
// Задать 20 Декабря 1999 этим способом можно так:
d=new Date(945624000000);
где
миллисекунды -- количество прошедших с 00:00:00 1 Января 1970 миллисекунд (1000 миллисекунд = 1 секунда) до задаваемой даты.

После того, как объект будет создан Вы можете с помощью доступных методов производить над ним различные операции.

2.3.1. Методы объекта Date

Метод getDate() позволяет извлечь из объекта день месяца.

Пример
```
//Определим текущую дату и запишем результат в x
x=new Date();
//Извлечем день месяца из объекта x и выведем результат на страницу
document.write(x.getDate());
```
Метод getFullYear() позволяет извлечь из объекта заданный год.

Пример
```
//Определим текущую дату и запишем результат в x
x=new Date();
//Извлечем год из объекта x и выведем результат на страницу
document.write(x.getFullYear());
```

Метод setFullYear(год, месяц, число_месяца) позволяет изменить дату, заданную в объекте, на желаемую.

Пример
```
//Определим текущую дату и запишем результат в d
d=new Date();
//Выведем текущую дату на страницу
document.write(d + '<br />');
//Изменим дату заданную в объекте
d.setFullYear(1990,04,12);
//Выведем новую дату на страницу
document.write(d);
```

2.3.2. Некоторые часто используемые методы объекта Date

Метод	Описание
getDate()	Возвращает день месяца (может принимать значения от 1-31) заданной даты.
getDay()	Возвращает день недели (может принимать значения от 0-6, причем 0-Воскресенье, а 6-Суббота) заданной даты.
getFullYear()	Возвращает год (4 числа, например 2011) заданной даты.
getHours()	Возвращает час (может принимать значения от 0-23) заданной даты.
getMinutes()	Возвращает минуту (может принимать значения от 0-59) заданной даты.
getMonth()	Возвращает месяц (может принимать значения от 0-11) заданной даты.
getTime()	Возвращает количество миллисекунд, которые прошли с полночи 1 Января 1970 года до заданной в объекте даты.
parse()	Возвращает количество прошедших с полночи 1 Января 1970 года миллисекунд до заданной даты.
setDate()	Устанавливает день месяца (может принимать значения от 0-31) заданной дате.
setFullYear()	Устанавливает год (4 числа, например 2011) заданной дате.
setHours()	Устанавливает час (может принимать значения от 0-23) заданной дате.
setMinutes()	Устанавливает минуту (может принимать значения от 0-59) заданной дате.
setMonth()	Устанавливает месяц (может принимать значения от 0-11) заданной дате.
setTime()	Устанавливает дату и время путем прибавления или вычитания указанного количества миллисекунд от полночи 1 Января 1970.
toDateString()	Преобразует часть объекта содержащую дату в строку.
toString()	Преобразует объект в строку.
toTimeString()	Преобразует часть объекта содержащую время в строку.

2.4. Объект Math

Используя свойства и методы объекта Math Вы можете выполнять в JavaScript различные математические операции.

Обратите внимание: для того, чтобы обращаться к свойствам и методам математического объекта его не нужно (в отличие от остальных встроенных объектов JavaScript) предварительно создавать.

2.4.1. Свойства объекта Math

Свойства данного объекта содержат значения часто используемых математических констант:

Пример
```
//Выведем значение числа Пи на страницу
document.write(Math.PI+'<br />');
//Выведем значение экспоненты
document.write(Math.E+'<br />');
//Выведем значение натурального логарифма 10
document.write(Math.LN10+'<br />');
//Выведем значение квадратного корня 2
document.write(Math.SQRT2);
```

2.4.2. Методы объекта Math

С помощью методов объекта Вы можете производить над числами различные математические операции.

Метод round() позволяет округлять числа до ближайшего целого.

Пример
```
//Округлим число 25.34 до ближайшего целого и выведем значение на страницу
document.write(Math.round(25.34)+'<br />');
//Округлим число 25.88 до ближайшего целого и выведем значение на страницу
document.write(Math.round(25.88));
```

Метода random() позволяет генерировать случайные числа между 0 и 1.

Пример
```
//Сгенерируем случайное число между 0 и 1 и выведем его на страницу
document.write(Math.random()+'<br />');
/* Вы также можете генерировать случайные числа в произвольном промежутке.
Например для того, чтобы сгенерировать случайное число в промежутке от 0 до
100 нужно умножить число, полученное с помощью метода random(), на 100 и
затем округлить его до ближайшего целого */
document.write(Math.round(Math.random()*100));
```

Метод pow(число,степень) позволяет возводить числа в степень.

Пример
```
//Возведем 3 в 3 степень
document.write(Math.pow(3,3) + '<br />');
//Возведем 4 в 0.5 степень (найдем корень из 4)
document.write(Math.pow(4,0.5) + '<br />');
//Возведем 5 в -1 степень
document.write(Math.pow(5,-1));
```

Методы max (максимум) и min (минимум) выбирают максимальные и минимальные числа из предложенных.

Пример

```
//Выберем из чисел 10 68 35 12 44 максимальное и выведем его на страницу
document.write(Math.max(10,68,35,12,44) + '<br />');
//Выберем из чисел 10 68 35 12 44 минимальное и выведем его на страницу
document.write(Math.min(10,68,35,12,44));
```

Использованные источники:

1. Учебники и справочники для веб-разработчиков: http://www.wisdomweb.ru/JS/bomdet.php.

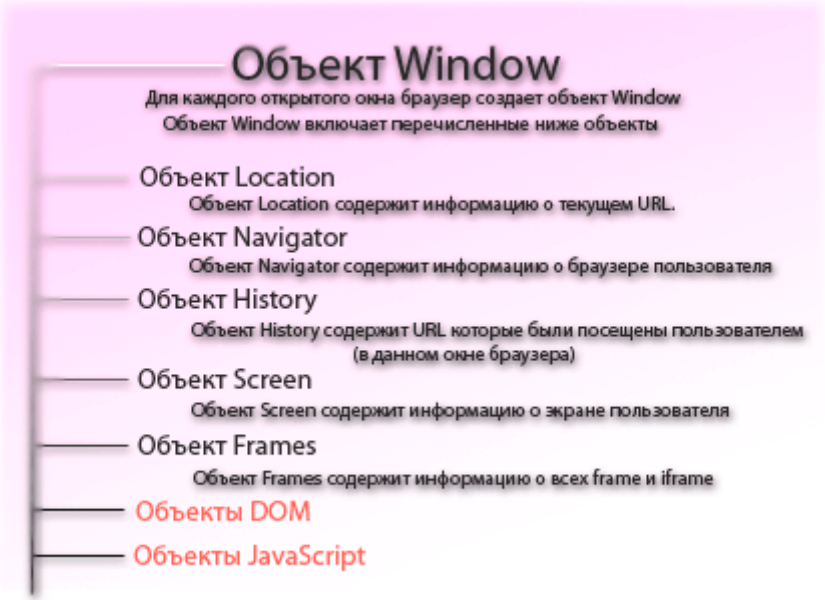
2. Учебные материалы W3 Консорциума: http://www.w3shools.org.

1. BOM – ОБЪЕКТНАЯ МОДЕЛЬ БРАУЗЕРА

BOM это сокращение от Browser Object Model (Объектная модель браузера).

С помощью объектной модели браузера (BOM) Вы можете управлять поведением браузера с пмощию JavaScript.

BOM включает в себя несколько объектов:



1.1. Объект window

Объект window является корневым объектом JavaScript. Все объекты JavaScript, а также переменные и функции определяемые пользователем хранятся в объекте window.

Если Вы знаете имя другого открытого окна Вы можете обращаться к объектам созданным в другом окне.

Писать *"window."* при обращении к объектам и переменным необязательно так как JavaScript подставляет его автоматически.

Пример

```
//Обратимся к объекту navigator
document.write(window.navigator.appName+'<br />');
//Теперь обратимся к объекту navigator опустив window
document.write(navigator.appName);
//Создадим переменную a (то же самое что a=10)
window.a=10;
//Выведим значение переменной a на экран
document.write(a+'<br />');
document.write(window.a);
```

1.1.1. Свойства объекта window

С помощью свойства **length** Вы можете узнать сколько фрэймов (включая iframes) присутствует в данном окне.

Пример
```
//Узнаем количество фрэймов на странице
document.write(window.length);
```

1.1.2. Методы объекта window

С помощью метода **alert()** Вы можете вывести окно оповещения.

Пример
```
//Выведем окно оповещения
alert('Это окно оповещения');
```

С помощью метода **open()** Вы можете открыть новое окно.

Пример
```
//Откроем новое пустое окно
nw=open();
//Выведем сообщение в новое окно
nw.document.write('Этот текст был выведен с помощью JavaScript.');
```

С помощью метода **close()** Вы можете закрыть окно.

Пример
```
<script type=text/javascript>
//Откроем новое окно
nw=open();
//Выведем сообщение в новое окно
nw.document.write('Этот текст был выведен с помощью JavaScript.');
/* Создадим функцию cl() закрывающую окно nw
которая будет вызываться после нажатия на кнопку */
function cl(){
  nw.close();
}
</script>
<input type='button' value='Закрыть окно' onclick='cl()' />
```

С помощью метода **print()** Вы можете распечатать содержимое окна на принтере.

Пример
```
<script type=text/javascript>
/* Создадим функцию pr() печатающую содержимое данного окна
после нажатия на кнопку */
function pr(){
  print();
```

```
}
</script>
<form>
<input type='button' value='Напечатать содержимое данной страницы' onclick='pr()' />
</form>
```

1.2. Объект navigator

С помощью объекта **navigator** Вы можете определить какой браузер использует пользователь.

Так же с помощью navigator Вы можете проверить может ли пользователь принимать cookie и включена ли у него поддержка Java.

1.2.1. Свойства

С помощью свойства **userAgent** Вы можете узнать всю информацию о браузере пользователя.

Пример

```
//Выведем информацию о Вашем браузере на страницу
document.write('Информация о Вашем браузере: '+navigator.userAgent);
```

С помощью свойства **cookieEnabled** Вы можете узнать включена ли возможность использования cookie в браузере пользователя. Свойство возвращает true если возможность использования cookie включена и false если нет.

Пример

```
//Проверим может ли Ваш браузер принимать cookie
if (navigator.cookieEnabled)
  document.write('Ваш браузер может принимать cookie!');
else
  document.write('Ваш браузер не может принимать cookie.');
```

С помощью свойства **appVersion** Вы можете узнать версию браузера.

Пример

```
//Узнаем версию Вашего браузера и выведем ее на экран
document.write(navigator.appVersion);
```

С помощью свойства **appName** Вы можете узнать название браузера, а с помощью **appCodeName** кодовое название браузера.

Пример

```
//Узнаем название Вашего браузера и выведем его на страницу
document.write(navigator.appName + '<br />');
//Узнаем кодовое название браузера и выведем его на страницу
document.write(navigator.appCodeName +'<br />');
```

С помощью свойства **platform** Вы можете узнать платформу, под которую скомпилирован браузер (т.е. узнать ОС, которую использует пользователь).

Пример
```
//Узнаем платформу, под которую скомпилирован Ваш браузер и выведем ее на страницу
document.write(navigator.platform);
```

1.2.2. Методы

С помощью метода **javaEnabled()** Вы можете проверить включена ли поддержка Java в браузере пользователя или нет. Метод вернет true если поддержка включена и false если нет.

Пример
```
//Проверим включена ли в Вашем браузере поддержка Java
document.write(navigator.javaEnabled());
```

1.3. Объект Screen

Объект **Screen** содержит информацию об экране пользователя.

С помощью свойств данного объекта Вы можете узнать какое разрешение, а также какая глубина цвета установлена на экране пользователя.

Свойство **width** определяет ширину экрана пользователя, а свойство **height** высоту.

Пример
```
//Узнаем разрешение Вашего экрана и выведем его на на страницу
document.write('Разрешение экрана: <b>'+screen.width+'X'+
screen.height+'</b>');
```

С помощью свойства **colorDepth** Вы можете узнать глубину цвета (измеряется в битах на пиксель) установленную у пользователя:

Пример
```
//Узнаем глубину цвета Вашего экрана и выведем ее на страницу
document.write('Глубина цвета: ' + screen.colorDepth);
```

1.4. Объект History

Объект History содержит список URL которые были посещены в данном окне браузера.

С помощью свойства **length** Вы можете узнать количество посещенных URL хранящихся в списке.

Пример
```
//Выведем количество посещенных в данном окне браузера URL
document.write(history.length);
```

Вы можете перемещаться по списку посещенных URL с помощью метода **go(смещение)**.

Например, с помощью go(2) Вы можете переместится на два URL вперед, а go(-3) переместит Вас на три URL назад.

Пример
```
function bc2() {
  window.history.go(-2);
}
```

1.5. Объект Location

С помощью объекта **location** Вы можете узнать любую информацию о URL текущего документа.

1.5.1. Свойства

Свойство **href** хранит URL текущего документа целиком.

Пример
```
document.write(location.href);
```

С помощью свойства **pathname** Вы можете узнать путь к загруженному документу.

Пример
```
document.write(location.pathname);
```

Свойство **host** содержит имя домена загруженного документа.

Пример
```
document.write(location.host);
```

1.5.2. Методы

С помощью метода **assign()** Вы можете загрузить новый документ в данное окно браузера (изменить текущий URL на желаемый).

Пример
```
//После нажатия на кнопку будет загружена главная страница нашего сайта
function newdoc (){
  location.assign('http://www.wisdomweb.ru');
```

```
}
<input type='button' value='Загрузить главную страницу' onclick='newdoc()' />
```

2. HTML DOM – ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА

2.1. Что такое HTML DOM

HTML DOM расшифровывается HTML Document Object Model (Объектная Модель HTML Документа).

HTML DOM позволяет читать и изменять содержимое HTML элементов страницы из скриптов.

HTML DOM может использоваться со многими языками программирования, но наиболее часто DOM используют в связке с JavaScript.

HTML DOM является W3C стандартом, поэтому все современные браузеры поддерживают данную технологию.

2.2. Объектная структура

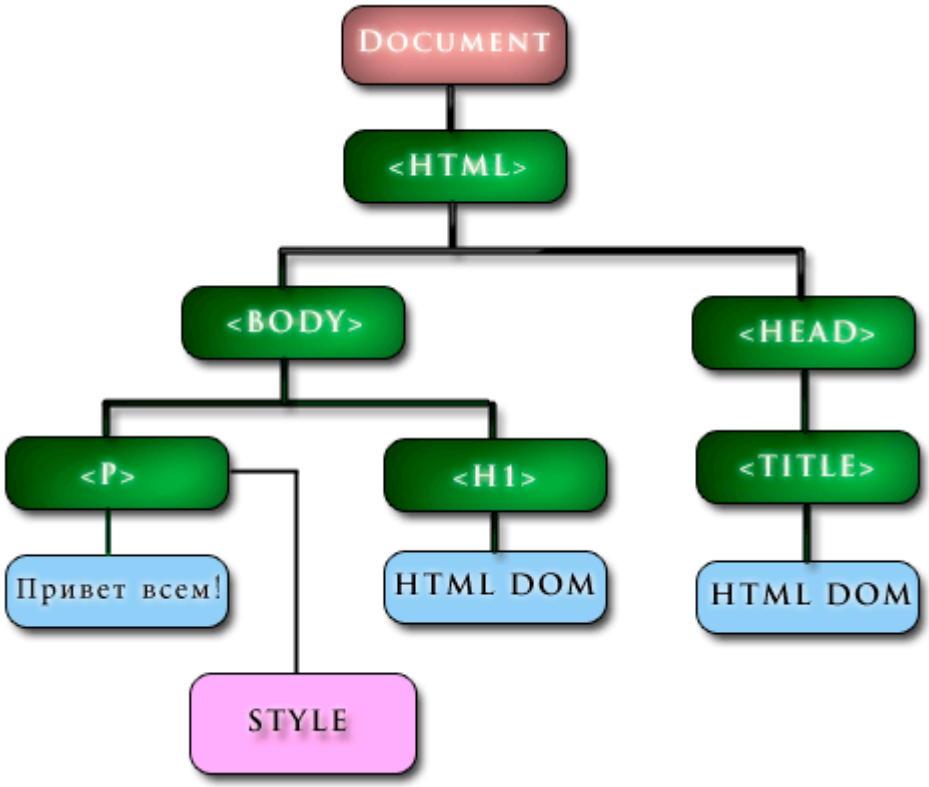
При открытии любого HTML документа браузер предварительно производит разбор его содержимого и на основе этого разбора создает объектную модель HTML документа или более коротко DOM.

DOM состоит из вложенных в друг друга в иерархическом порядке объектов, которые называются узлами. Каждый узел в структуре представляет располагающийся на странице HTML элемент.

Используя DOM Вы можете взаимодействовать (*считывать, изменять, удалять*) с содержимым HTML документов из скриптов.

Ниже располагается код HTML документа и DOM, которая бы была создана браузером на основе этого кода:

```
<html>
<head>
<title>HTML DOM</title>
</head>
<body>
<h1>HTML DOM.</h1>
<p style="color:green">Привет всем.</p>
</body>
</html>
```



Все прямоугольники изображенные на картинке являются объектами (или узлами). Разным цветом на изображение отмечены узлы разного типа.

Красным цветом отмечен узел Document. Любое обращение к DOM должно начинаться с обращения к данному узлу.

Зеленым цветом отмечены элементные узлы. Для каждого HTML элемента на странице браузер создает соответствующий элементный узел.

Содержимое элементов хранится в текстовых узлах. Текстовые узлы отмечены на нашей схеме синим цветом.

Для каждого атрибута HTML элемента создается атрибутный узел. Атрибутный узел отмечен на схеме розовым цветом.

Обратите внимание: не забывайте, что текст всегда хранится в текстовых узлах, а не является свойством элемента. Т.е. для того, чтобы обратится к содержимому HTML элемента Вы должны обратится к свойству его текстового узла.

2.2.1. Отношения между узлами

Узлы в объектной структуре связаны друг с другом. Существует несколько специальных терминов для описания отношений между узлами:

Родительский узел (*parent node*) - родительским узлом по отношению к рассматриваемому объекту является узел, в который вложен рассматриваемый объект. На

нашей схеме по отношению к узлам <h1> и <p> <body> является родительским. Для узла <title> родительским является узел <head>.

Узлы потомки (*child node*) - узлом потомком по отношению к рассматриваемому объекту является узел, который вложен в рассматриваемый объект. На нашей схеме по отношению к узлу <body> <h1> и <p> являются потомками. Для узла <head> потомком является <title>.

Узлы братья (*sibling node*) - узлы находящиеся на одинаковом уровне вложенности по отношению к их родительскому узлу. На нашей схеме узлами братьями являются <body> и <head>, <p> и <h1>.

Самый верхний узел в DOM называется корневым. На нашей схеме корневым является <html> (т.к. объект document не является частью DOM).

Пример
```
/* Прочитаем содержимое элемента p,
перемещаясь по объектной и выведем результат на страницу
1. Обратимся к первому объекту потомку элемента body (h1).
2. Перейдем к следующему узлу брату. (p)
3. Обратимся к текстовому узлу соответствующего элемента.
4. Обратимся к содержимому текстового узла */
document.write(document.body.childNodes[0].nextSibling.childNodes[0].nodeValue);
```

2.3. Обращение к элементам

Условно можно сказать что обращаться к элементам в DOM можно двумя различными способами:

Использовать последовательное перемещение по объектной структуре до нахождения необходимого элемента.

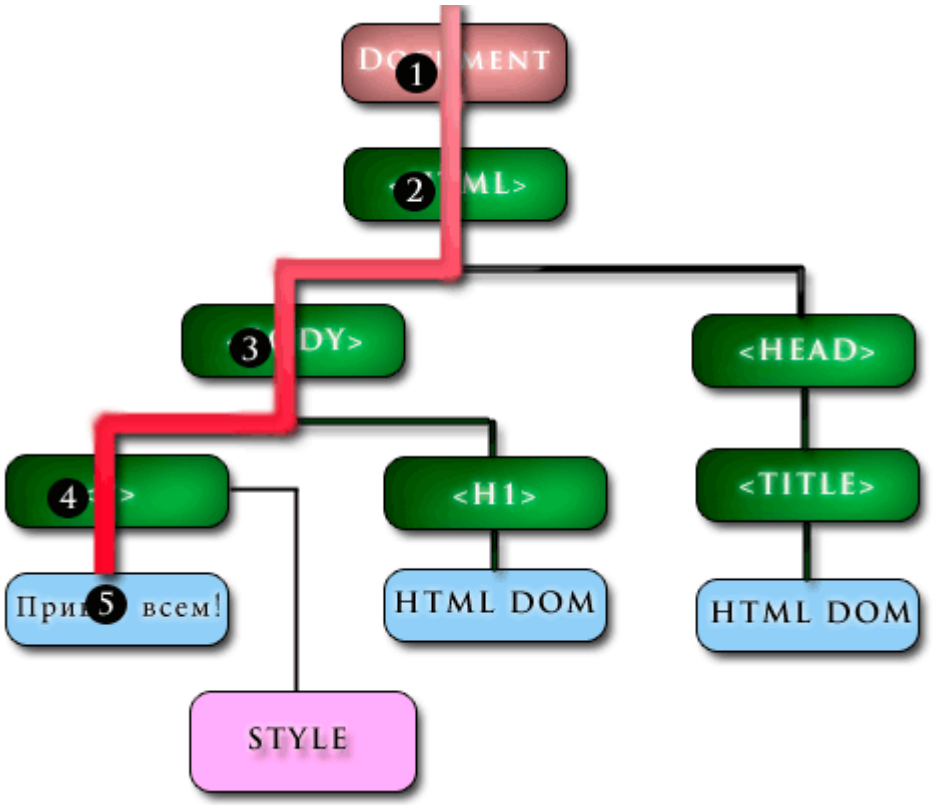
Использовать прямое обращение к элементу по его идентификатору или имени тэга.

Второй способ безусловно проще и удобнее и в повседневной практике всегда используют именно его. Однако в учебных целях полезно разобрать оба способа.

2.3.1. Последовательное перемещение

Вернемся к DOM из второй главы и представим, что нам нужно прочитать текстовое содержимое ее элемента p.

Красной стрелкой и черными кружками с цифрами отмечено как последовательно будет выглядеть перемещение по объектной структуре:



Разберем процесс перемещения подробнее:

1. Обращаемся к объекту document, в котором находится DOM. Код на данном шаге имеет вид: document.

2. Обращаемся к корневому узлу (т.е. тэгу <html>) который находится внутри объекта document. Код на данном шаге имеет вид: document.documentElement.

3. Обращаемся ко второму потомку (так как в коде страницы body располагается после head) корневого узла. Код на данном этапе будет иметь вид: document.documentElement.childNodes[1]. Вы также можете напрямую обратится к body используя следующий код (далее будем полагать что Вы выбрали этот вариант): document.body.

4. Обращаемся ко второму потомку body (элемент p задан в коде после h1). Код на данном этапе будет иметь вид: document.body.childNodes[1].

1. Обращаемся к текстовому узлу который является первым потомком p и узнаем значение его свойства. Код на данном этапе будет иметь вид: document.body.childNodes[1].childNodes[0].nodeValue.

Пример

```
//Выведем содержимое абзаца на страницу
document.write(document.body.childNodes[1].childNodes[0].nodeValue);
```

2.3.2. Прямое обращение к элементу

С помощью метода **getElementById** Вы можете напрямую обращаться к элементам по их идентификатору (*атрибут id*), а с помощью свойства **innerHTML** можно быстро считывать их текстовое содержимое.

Пример
```
//Выведем содержимое второго абзаца на страницу
document.write(document.getElementById("par").innerHTML);
```

2.4. DOM Манипуляции

Для каждого элемента на странице браузер создает в DOM элементный узел.

Каждый элементный узел имеет набор предопределенных **свойств** и **методов**. С помощью этих свойств и методов Вы можете производить различные манипуляции над содержимым элементных узлов объектной структуры

2.4.1. Свойства

С помощью свойства innerHTML Вы можете получить доступ к содержимому текстового узла DOM объекта.

Пример
```
function change(){
  document.getElementById("par").innerHTML="Я изучаю HTML DOM на
  <b>wisdomweb.ru</b>!";
}
function changeback(){
  document.getElementById("par").innerHTML="Я изучаю JavaScript.";
}
```

С помощью свойства **nodeName** Вы можете узнать имя узла.

Пример
```
//Узнаем имя элемента с id=par и сохраним результат в переменную x
x=document.getElementById("par").nodeName;
//Выведим результат на страницу
document.write(x+"<br />");
//Узнаем имя элемента с id=header и сохраним результат в переменную y
y=document.getElementById("header").nodeName;
//Выведим результат на страницу
document.write(y);
```

С помощью **свойства nodeValue Вы можете** узнать значения узла.

Пример
```
//Узнаем значение свойства первого потомка элемента с id=par и сохраним результат
```

```
в x
x=document.getElementById("par").childNodes[0].nodeValue;
//Выведим результат на страницу
document.write(x+"<br />");
```

С помощью свойства parentNode Вы можете обратится к родительскому узлу элемента.

```
Пример
//Узнаем имя родительского узла элемента с id=par и сохраним результат в x
x=document.getElementById("par").parentNode.nodeName;
//Выведим результат на страницу
document.write(x+"<br />");
```

С помощью свойства childNodes Вы можете обратится к узлам потомкам элемента.

Для того, чтобы обратится к первому узлу потомку используйте childNodes[0], ко второму childNodes[1] и т.д.

```
Пример
//Узнаем свойство первого потомка элемента с id=par и запишем результат в x
x=document.getElementById("par").childNodes[0].nodeValue;
//Выведим результат на страницу
document.write(x+"<br />");
```

С помощью свойства attributes Вы можете обратится к атрибутам узла.

Нумерация атрибутов идет в обратном порядке. Для того, чтобы обратится к последнему атрибуту используйте attributes[0], к предпоследнему attributes[1] и т.д.

```
Пример
//Узнаем значение первого атрибута элемента и запишем результат в x
x=document.getElementById("ww").attributes[0].nodeValue;
//Выведим результат на страницу
document.write(x+"<br />");
```

С помощью свойства nodeType Вы можете узнать тип узла.

Элементный узел имеет тип 1, атрибутный узел имеет тип 2, текстовый узел имеет тип 3.

```
Пример
//Узнаем тип узла и запишем результат в x
x=document.getElementById("par").nodeType;
//Выведим результат на страницу
document.write(x+"<br />");
```

2.4.2. Методы HTML DOM

С помощью метода getElementById() Вы можете обратится к элементу с указанным id.

Пример

```
function change(){
  document.getElementById("par").innerHTML="Я изучаю HTML DOM на
  <b>wisdomweb.ru</b>!";
}
function changeback(){
  document.getElementById("par").innerHTML="Я изучаю JavaScript.";
}
```

С помощью метода getElementsByTagName() Вы можете обратится ко всем элементам с указанным именем тэга.

Метод возвращает массив элементов. Нумерация элементов в массиве начинается с 0.

Пример

```
//Узнаем содержимое первого абзаца на странице и запишем результат в x
x=document.getElementsByTagName("p")[0].innerHTML;
//Выведим результат на страницу
document.write(x+"<br />");
//Узнаем содержимое второго абзаца и запишем результат в y
y=document.getElementsByTagName("p")[1].innerHTML;
//Выведим результат на страницу
document.write(y);
```

С помощью метода createElement() Вы можете создать элемент.

С помощью метода appendChild() Вы можете вставить созданный элемент в произвольный узел.

Пример

```
var i=0;
function elCreate(){
  //Создадим новый абзац
  var newpar = document.createElement("p");
  i++;
  //Запишем созданный абзац в body
  document.body.appendChild(newpar).innerHTML="<b>Я созданный элемент номер
"+i+"</b>.";
}
```

С помощью метода removeChild() Вы можете удалить узел потомок из элемента.

Данный метод возвращает удаленный узел.

Пример
```
function mvParFrom(){
  var rcont = document.getElementById("container");
  var rpar= document.getElementById("par");
  /* Удалим из элемента с id=container элемент с id=par и вернем
  удаленный элемент в rel */
  var rel=rcont.removeChild(rpar);
  //Выведем содержимое текстового узла удаленного элемента на страницу
  document.getElementById("res").innerHTML=rel.innerHTML;
  }
function mvParTo(){
  document.getElementById("cont").style.display="none";
  document.getElementById("container").appendChild(document.getElementById("res"));
```

2.5. Объект style

Оформление любого элемента можно изменять из скриптов с помощью объекта **style**.

Пример
```
//Установим желтый цвет фона для элемента elp
document.getElementById("elp").style.backgroundColor="yellow";
//Узнаем цвет фона элемента elh и выведем его на страницу
document.write("<b>"+document.getElementById("elh").style.backgroundColor+"</b>");
```

2.5.1. Изменение стилей после событий

Пример
```
<html>
<head>
<script type='text/javascript'>
function start()
{
document.getElementById('elp').style.color='red';
document.getElementById('elp').style.fontWeight='bold';
}
</script>
</head>
<body>
<p id='elp' onclick='start()'>Щелкните на мне мышкой.</p>
</body>
</html>
```

2.6. События

Код переданный событиям будет выполнен после того, как произойдут действия, которые активируют данное событие. У каждого события разные *активирующие действия*.

Примеры активирующих действий:

- Щелчок мыши (*событие onclick*) и двойной щелчок мыши (*событие ondblclick*).
- Нажатие клавиши (*onkeypress*).
- Отправление формы (*onsubmit*).
- Наведение курсора мыши на элемент (*onmouseover*) или выведение курсора мыши за пределы его границ (*onmouseout*).
- Полная загрузка страницы или картинки (*onload*).
- Изменение содержимого элемента, например содержимого текстового поля формы (*onchange*).

События могут непосредственно содержать код, который будет выполнятся:

Пример
```
<input type="button" value="Нажмите на меня" onclick='alert("Вы нажали на кнопку");'/>
```

Либо вызывать функцию, которая содержит необходимый код:

Пример
```
function dispMes(){
  alert("Вы нажали на кнопку");
}
....
<input type="button" value="Нажмите на меня" onclick="dispMes()"/>
```

2.6.1. Событие onclick

Код переданный событию onclick будет исполнен, когда пользователь щелкнет мышкой на элементе.

Пример
```
function go(){
  document.location.assign("http://www.wisdomweb.ru");
}
....
<input type="button" value="Перейти на главную страницу нашего сайта"
onclick="go()"/>
```

2.6.2. Событие onmouseover и onmouseout

Код переданный событию onmouseover будет выполнен при наведении курсора мыши на элемент.

Код переданный событию onmouseout будет выполнен при выводе курсора из границ элемента.

Пример
```
function start(){
  document.getElementById("par1").style.color="red";
}
function stop(){
  document.getElementById("par1").style.color="black";
}
....
<p id="par1" onmouseover="start()" onmouseout="stop()">Наведите на меня курсор мыши.</p>
```

2.6.3. Чтения кода событий

Так как события являются методами DOM объектов. Имеется возможность прочитать содержимое кода событий из скрипта.

Пример
```
/* Выведем код события onclick кнопки с id=show */
document.write(document.getElementById("show").onclick+"<br />");
/* Теперь выведем код события onclick кнопки с id=hide */
document.write(document.getElementById("hide").onclick+"<br />");
/* И напоследок выведем код события onclick кнопки с id=pers */
document.write(document.getElementById("pers").onclick+"<br />");
```

2.7. Анимированные кнопки

Комбинирование HTML DOM, JavaScript и HTML называют DHTML.

Использование DHTML предоставляет множество новых возможностей при создании страниц, например создание анимированных кнопок:

Пример
```
function start(){
  document.getElementById("elp").src="button2.gif";
}
function stop(){
  document.getElementById("elp").src="button1.gif";
}
function go(){
```

```
  document.location.assign("http://www.wisdomweb.ru");
}
....
<img src="button1.gif" id="elp" onmouseover="start()" onmouseout="stop()"
onclick="go()" />
```

2.8. Использоваие

2.8.1. Анимированные кнопки

Комбинирование HTML DOM, JavaScript и HTML называют DHTML.

Использование DHTML предоставляет множество новых возможностей при создании страниц, например создание анимированных кнопок:

Пример

```
function start(){
   document.getElementById("elp").src="button2.gif";
}
function stop(){
   document.getElementById("elp").src="button1.gif";
}
function go(){
   document.location.assign("http://www.wisdomweb.ru");
}
....
<img src="button1.gif" id="elp" onmouseover="start()"
onmouseout="stop()"
onclick="go()" />
```

2.8.2. Анимация

Используя выполнение кода по расписанию и возможности DHTML Вы можете создавать настоящую анимацию на своих страницах.

Пример

```
setTimeout("document.getElementById("movesq").style.top="10px"",1000);
setTimeout("document.getElementById("movesq").style.top="20px"",2000);
setTimeout("document.getElementById("movesq").style.top="30px"",3000);
setTimeout("document.getElementById("movesq").style.top="40px"",4000);
setTimeout("document.getElementById("movesq").style.left="100px"",5000);
setTimeout("document.getElementById("movesq").style.top="60px"",6000);
setTimeout("document.getElementById("movesq").style.backgroundColor="green"",7000
);
setTimeout("document.getElementById("movesq").style.top="90px"",8000);
setTimeout("document.getElementById("movesq").style.left="120px"",9000);
```

```
setTimeout("document.getElementById("movesq").style.left="30px"",10000);
setTimeout("document.getElementById("movesq").style.top="0px"",11000);
setTimeout("document.getElementById("movesq").style.backgroundColor="red"",12000);
```

2.8.3. Выпадающие меню

Также комбинируя эти возможности Вы можете создавать выпадающие меню.

Выпадающее меню является очень удобным и часто используемым элементом сайта.

Пример

```
function starteng(){
  document.getElementById("eng").style.display="block";
}
function stopeng(){
  document.getElementById("eng").style.display="none";
}
function startde(){
  document.getElementById("de").style.display="block";
}
function stopde(){
  document.getElementById("de").style.display="none";
}
function startrus(){
  document.getElementById("rus").style.display="block";
}
function stoprus(){
  document.getElementById("rus").style.display="none";
}
function startdia(){
  document.getElementById("dia").style.display="block";
}
function stopdia(){
  document.getElementById("dia").style.display="none";
}
```

2.9. Специальные свойства и методы

Некоторые объекты HTML элементов в DOM помимо набора стандартных свойств и методов (таких как innerHTML, childNode, nodeType, nodeValue) имеют еще и специальные свойства и методы имеющиеся только у них.

Пример

```
//Обратимся к специальному свойству href элемента a и выведем результат на
страницу
document.write(document.getElementById("link").href+"<br />");
/* Обратимся к специальным свойствам type и value элемента input button и
```

```
 выведем результат на страницу */
document.write(document.getElementById("button").type+"<br />");
document.write(document.getElementById("button").value+"<br />");
//Обратимся к специальному свойству src элемента img и
//выведем результат на страницу
document.write(document.getElementById("image").src+"<br />");
```

Ниже приведем список HTML элементов, которые имеют в DOM специальные свойства и методы:

Элемент <a>
Элемент <area>
Элемент <button>
Элемент <form>
Элемент <iframe>
Элемент <image>
Элемент <input>
Элемент <link>
Элемент <meta>
Элемент <option>
Элемент <select>
Элемент <table>
Элемент <td>
Элемент <tr>
Элемент <textarea>

2.10. Объект Document

Для каждого документа, который загружается в окне браузера создается DOM объект document.

Объект document обеспечивает доступ из скрипта ко всем HTML элементам, которые находятся на странице.

Вы можете обращаться из скриптов к содержимому документов открытых в других окнах браузера используя: имя_окна.document.

2.10.1. Свойства объекта document

С помощью свойства referrer Вы можете узнать URL документа, из которого был совершен переход на текущую страницу.

Пример

```
document.write(document.referrer);
```

С помощью свойства URL Вы можете узнать URL текущего документа.

Пример
```
document.write(document.URL);
```

С помощью свойства title Вы можете узнать значение заголовка (тэга title) текущего документа.

Пример
```
document.write(document.title);
```

2.10.2. Методы объекта document

Метод **write** выводит переданный в него текст на страницу.

Пример
```
document.write("Я текст выведенный с помощью метода write.");
```

Метод writeln делает тоже самое что и write, но вставляет после каждого вывода символ перевода строки.

Пример
```
document.writeln('Первое предложение');
document.writeln('Второе предложение');
document.writeln('Третье предложение');
document.write('<hr />');
document.write('Первое предложение');
document.write('Второе предложение');
document.write('Третье предложение');
```

С помощью метода getElementsByName Вы можете получить доступ к содержимому всех элементов на странице с указанным именем.

Метод возвращает массив элементов с указанным именем. Нумерация элементов в массиве начинается с 0.

Пример
```
<input type='button' name='but1' value='Кнопка 1' />
<input type='button' name='button' value='Кнопка 2' />
<input type='button' name='button' value='Кнопка 3' />
....
//Узнаем значение атрибута value первого элемента с именем but1
document.write(document.getElementsByName('but1')[0].value+'<br />');
//Узнаем значение атрибута value первого элемента с именем button
document.write(document.getElementsByName('button')[0].value+'<br />');
//Узнаем значение атрибута value второго элемента с именем button
document.write(document.getElementsByName('button')[1].value+'<br />');
```

Обратите внимание: разобранные ранее методы getElementById и getElementsByTagName также являются методами объекта document.

2.10.3. Работа с Cookie

Cookie являются важным элементом для реализации многих полезных возможностей во всемирной паутине.

Cookie - это небольшой фрагмент идентификационных данных который:

1. Создается сервером;
2. Передается сервером браузеру пользователя;
3. Сохраняется браузером на компьютере пользователя в виде файла;
4. При повторном посещении сервера отправляется ему браузером;
5. Сервер получает cookie анализирует их содержание и на основе этого производит какие-либо действия.

С помощью cookie реализуется:

- Аутентификация пользователя на сайте (запоминание логина и пароля);
- Сохранение информации о личных настройках внешнего вида сайта;
- Ведение статистики о пользователях.

Свойство cookie позволяет узнать cookie, которые браузер отправил при запросе данной страницы.

Пример
document.write(document.cookie);

Использованные источники:

1. Учебники и справочники для веб-разработчиков: http://www.wisdomweb.ru .

1. Проверка данных, предоставленных пользователем на основе JavaScript

Автор: Пучкова Д.М.

1.1. Возможности JavaScript по проверке данных пользователя

Нет ничего более проблематичного, чем обработка заказов, записей в гостевой книге или любых других данных в незаконченном виде, отсылаемых на сервер пользователями. Т.е. это обработка так называемых «плохих» данных. Частично этих проблем можно избежать путем обработки данных средствами JavaScript. Эта техника называется «обработка формы».

Идея проверки данных пользователя средствами JavaScript заключается в проверке информации, вводимой пользователем в поля ввода ещё до того, как они успеют переслать информацию на сервер. JavaScript также позволяет выводить сообщения пользователю об ошибках заполнения формы и рекомендациях, как их можно исправить.

Если сравнить возможности по проверке данных, вводимых пользователем, средствами JavaScript по сравнению с теми же самыми возможностями скриптов PHP. Вообще говоря, данные должны проверяться и самим PHP-скриптом. Но если сайт высокопосещаемый, то это создает дополнительную нагрузку на сервер. JavaScript проверяет данные на стороне клиента, поэтому нагрузка на сервер при этом минимальна. Так как 95% все пользователей используют браузеры, поддерживающие JavaScript, то скрипты можно смело писать.

1.2. Определение списка задач по обработке данных пользователя, наиболее эффективно решаемых средствами JavaScript

Определим список задач по обработке данных пользователя средствами JavaScript на примере обработки полей формы.

Рассмотрим форму:

```
<form name=“myform” method=“post” action=“test.php”>
<input type=“text” name=“sender” maxlength=“20” />
<input type=“text” name=“email” />
<input type=“text” name=“icq” maxlength=“9” />
<textarea=“msg”></textarea>
<input type=“submit” value=“Послать” />
</form>
```

Итак, мы создали форму, в которой присутствует четыре поля: поле для указания имени пользователя, поле для указания его email и поле для указания его icq, а также поле с содержанием пользовательского сообщения. Все четыре поля необходимо проверить на предмет правильности заполнения следующим образом:

- поле для указания имени пользователя не должно содержать от 3 до 20 символов.;
- email пользователя должен быть введен в корректной форме (т.е. содержать ‘@’ и ‘.’);
- icq пользователя должен состоять только из цифр и содержать в себе от 5 до 9 символов;
- проверка объекта «textarea» состоит в том, чтобы определить, что сообщение не пустое и содержит не менее 10 символов.

Итак, рассмотрим коды реализации всех четырех сценариев.

1.2.1. Проверка правильности заполнения поля ввода имени пользователя

```
p_sender = document.myform.sender.value.toString();
if (p_sender!=’’)
{
if (p_sender.length<3 || p_sender.length>20)
{
alert(‘Укажите Ваше имя (3-20 символов): ’);
document.myform.sender.focus();
return false;
}
}
else
{
alert(‘Необходимо ввести имя!’);
document.myform.sender.focus();
return false;
}
```

1.2.2. Проверка на корректность ввода email пользователя

```
p_email = document.myform.email.value.toString();
if (p_email!=’’)
{
t=p_email.indexOf(‘@’);
if ((p_email.indexOf(‘.’)==-1)||(t==-1)||(t<1)||(t>p_email.length-5)||(p_email.charAt(t-
1)==‘.’)||(p_email.charAt(t+1)==‘.’))
{
alert(‘Некорректный email!’);
document.myform.email.focus();
return false;
}
}
```

1.2.3. Проверка на корректность ввода icq пользователя.

```
<input type="text" name="icq" maxlength="9" onkeypress="if ((event.keyCode
<48)||(event.keyCode>57)) event.returnValue=false;" />
p_icq = document.myform.icq.value.toString();
if (p_icq!='')
{
if (p_icq.length<5||p_icq.length>9)
{
alert('Длина номера icq 5-9 символов!');
document.myform.icq.focus();
return false;
}
}
```

1.2.4. Проверка объекта textarea

```
p_msg = document.myform.msg.value.toString();
if (p_msg.length<10)
{
alert('Необходимо ввести текст сообщения (не менее 10 символов)!');
document.myform.msg.focus();
return false;
}
```

Существует и другой способ защиты от некорректных данных. Суть его заключается в «отключении» кнопки «Submit» и подключении ее только тогда, когда заполнены необходимые поля.

```
<input type="submit" value="Послать" disabled="disabled">
<input type="text" name="sender" maxlength="20" onkeypress="checkreq()" onkeyup
="checkreq()" onblur="checkreq()" />
<textarea name="msg" onkeypress="checkreq()" onkeyup="checkreq()"
onblur="checkreq()"></textarea>
```

Определим функцию checkreq():

```
function checkreq()
{
path = document.myform;
tmp = (path.sender.value == '');
if (!tmp&&(path.sender.value.length<3))
{tmp = true;}
path.submit.disabled = tmp;
if (tmp) {return;}
tmp = (path.msg.value='');
if (!tmp&&(path.sender.value.length<10))
{tmp = true;}
```

```
path.submit.disabled = tmp;
}
```

Далее рассмотрим полный код примера проверки формы средствами JavaScript.

```
<html>
<head>
<script language="javascript">
function checkreq()
{
path = document.myform;
tmp = (path.sender.value == '');
if (!tmp&&(path.sender.value.length<3))
{tmp = true;}
path.submit.disabled = tmp;
if (tmp) {return;}
tmp = (path.msg.value='');
if (!tmp&&(path.sender.value.length<10))
{tmp = true;}
path.submit.disabled = tmp;
}
</script>
</head>
<body>
<form method="post" name="myform" action="test.php" onsubmit="return checkreq();">
<input type="text" name="sender" onkeypress="checkreq()" onkeyup="checkreq()"
onblur ="checkreq() />
<input type="text" name="email" />
<input type="text" name="icq" onkeypress="if
((event.keyCode<48)||(event.keyCode>57)) event.returnValue=false;" />
<textarea name="msg" onkeypress="checkreq()" onkeyup="checkreq()"
onblur="checkreq()"></textarea>
<input type="submit" value="Послать">
</form>
</body>
</html>
```

В заключении хотелось бы подытожить, что традиционным способом использования веб-форм является пересылка всех пользовательских запросов на сервер, где запущена программа обработки, к примеру, CGI-скрипт. Основной задачей большинства CGI-скриптов является верификация того, что пользователь ввел в форму правильные данные. Задачей JavaScript является проверка заполнения формы на предмет ошибок ещё до отправки данных на сервер.

Таким образом, JavaScript помогает снизить объем работ CGI-скрипта и уменьшить общую нагрузку на сервер.

Литература.

1. Гарнаев А.Ю., Гарнаев С.Ю., Web-программирование на Java и JavaScript. – СПб.:БХВ-Петербург, 2005.

2. К.Л.Мордок, JavaScript: наглядный курс создания динамических Web-страниц.: Пер. с англ.: Уч. пос. – М.: Издательский дом “Вильямс”, 2001.

3. Э. Кингсли-Хью и К. Кингсли-Хью (Adrian Kingsley-Hughes, Kathie Kingsley-Hughes) “JavaScript 1.5: Учебный курс”. - СПб.:Питер, 2001.__

Источник: http://www.webbcare.org/lectures/poks/Lecture27.pdf

1. Основы VBScript

1.1. Типы данных

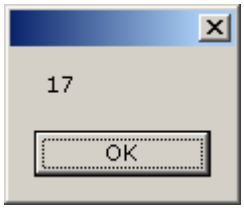
В языке VBScript используется единственный тип данных – Variant (Вариант), который позволяет хранить в переменной число, строку, дату, булевское значение, ссылку на объект и другую информацию. Определить тип содержимого переменной можно с помощью набора функций: VarType, TypeName, IsArray, IsDate, IsEmpty, IsNull, IsNumeric, IsObject, которые будут рассмотрены ниже. Тип содержащейся информации еще называется подтипом варианта. Полный список подтипов приведен в следующей таблице:

Подтип	Описание
Empty	Переменной не присвоено значение. При использовании неинициализированной переменной в числовых выражениях, будет подставляться 0, а в строковых – пустая строка.
Null	Переменная не содержит данных.
Boolean	Булевская переменная может принимать значения True, или False.
Byte	Целое число в диапазоне от 0 до 255.
Integer	Целое число в диапазоне от -32 768 до 32 767.
Currency	Число с фиксированной точкой в диапазоне от -922 337 203 685 477.5808 до 922 337 203 685 477.5807.
Long	Целое число в диапазоне от -2 147 483 648 до 2 147 483 647.
Single	Число с плавающей точкой одинарной точности. Для отрицательных значений допустимый диапазон от -3.402823E38 до -1.401298E-45. Для положительных – от 1.401298E-45 до 3.402823E38.
Double	Число с плавающей точкой двойной точности. Для отрицательных значений допустимый диапазон от -79769313486232E308 до -4.94065645841247E-324. Для положительных -- от 4.94065645841247E-324 до 1.79769313486232E308.
Date (Time)	Содержит число, представляющее дату в диапазоне от 1-го января 100 года, до 31 декабря 9999 года.
String	Последовательность символов. Максимальная длина в районе 2-х миллиардов знаков.
Object	Объект.
Error	Номер ошибки.

В зависимости от выражения, в котором участвует переменная, ее содержимое будет автоматически приведено к нужному типу. Рассмотрим такой пример:

```
Option Explicit
Sub TestVBScript
   Dim A, B
   A = 5
   B = "12"
   Application.MessageBox A + B, "", vbOkOnly
End Sub
```

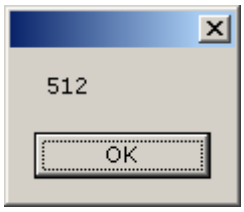
Так как в выражении участвует числовая переменная **A**, интерпретатор преобразует значение переменной **B** из строки **"12"** в число и просуммирует их:



Изменим макрос так, чтобы переменная **A** тоже содержала строку:

```
Option Explicit
Sub TestVBScript
   Dim A, B
   A = "5"
   B = "12"
   Application.MessageBox A + B, "", vbOkOnly
End Sub
```

Запустим его на выполнение. Теперь на экране появится результат слияния (конкатенции) двух строк, а не сумма их числовых представлений:



Во избежании путаницы с автоматическим приведением типов, рекомендуется использовать функции конверсии: CBool, CByte, CCur, CDate, CDbl, CInt, CLng, CSng, CStr.

Если результатом выражения должно быть именно слияние строк, а не сумма их числовых представлений, то следует использовать оператор & вместо +.

1.2. Переменные

Переменная – это удобное символьное обозначение области памяти, где приложение хранит некоторые данные. В процессе выполнения приложения значение переменной может изменяться. Перед использованием переменную следует объявить с помощью оператора Dim.

```
Dim A
```

С помощью одного оператора можно объявить сразу несколько переменных, если перечислить их имена через запятую:

```
Dim Left, Right, Top, Bottom
```

При объявлении нет необходимости в указании типа данных, так как все переменные имеют тип Variant.

Если в первой строке текста скрипта не указано Option Explicit, то использовать переменные можно без объявления. Но, такой путь может привести к трудно выявимым ошибкам. Достаточно один раз ошибиться в написании имени переменной в тексте программы, чтобы получить непредсказуемый результат. Мы рекомендуем всегда указывать Option Explicit и объявлять переменные.

Имя переменной должно соответствовать следующим требованиям:

- Начинаться с символа латинского алфавита;
- Состоять только из символов латинского алфавита или из символов латинского алфавита и цифр;
- Не превышать 255 символов в длину;
- Быть уникальным в пределах своей области видимости.

1.3. Область видимости и время жизни

Область видимости переменной определяется тем, где она была объявлена. Если внутри тела процедуры, то такая переменная называется локальной и доступна только в пределах этой процедуры. Если переменная объявлена в тексте скрипта, то она будет видима для всех процедур или функций определенных в этом скрипте. Локальные переменные могут иметь одинакове имена, если объявлены в разных процедурах.

Интерпретатор выделяет память для локальных переменных в момент их объявления и высвобождает по выходу из процедуры. Глобальные переменные существуют с момента их объявления и пока скрипт не закончит свое выполнение. Применительно к Гедымину это означает, что глобальные переменные существуют на протяжении всего времени выполнения программы.

1.4. Присваивание значения переменной

Значение объявленной переменной присваивается с помощью оператора =. Имя переменной указывается слева от оператора, новое значение – справа. Например:

```
A = 200
B = "Наименование"
```

1.5. Скалярные переменные и массивы

Переменная содержащая единственное значение называется скалярной. Иногда, возникает необходимость хранить несколько значений в одной переменной. В этом случае следует объявить массив. Синтаксис объявления идентичен объявлению скалярной переменной за тем исключением, что после имени в круглых скобках мы задаем размерность массива. Следующее объявление создаст массив из 12 элементов:

```
Dim Monthes(11)
```

В языке VBScript левая граница индекса массива всегда 0. Таким образом размер массива вычисляется, как число указанное в скобках плюс один. При присваивании значения элементу массива следует указать его индекс в круглых скобках:

```
Monthes(0) = "Январь"
Monthes(1) = "Февраль"
Monthes(2) = "Март"
...
Monthes(10) = "Ноябрь"
Monthes(11) = "Декабрь"
```

Аналогично, при обращении к значению элемента мы используем его индекс:

```
MonthName = Monthes(5)
```

Массив не обязательно должен быть одномерным. VBScript позволяет нам задать до 60 размерностей при объявлении массива. Например, следующий оператор создаст двумерный массив из 12 строк и двух колонок:

```
Dim MonthDays(11, 1)
```

При обращении к элементам многомерного массива следует указывать все индексы:

```
MonthDays(0, 0) = "Январь"
MonthDays(0, 1) = 31
MonthDays(1, 0) = "Февраль"
MonthDays(1, 1) = 28
...
```

Выше мы объявляли массивы, размер которых не меняется в процессе работы программы. Если заранее не известно сколько элементов понадобится, то можно объявить динамический массив:

```
Dim A()
```

Перед использованием следует установить размер динамического массива с помощью оператора ReDim:

```
ReDim A(25)
```

В процессе выполнения можно вызывать оператор ReDim многократно, каждый раз изменяя размер массива. Опция Preserve сохраняет значения элементов массива при изменении размера. Например, следующий код увеличит объявленный выше массив на пять элементов, оставив существующие нетронутыми:

```
ReDim Preserve A(30)
```

Помните, что при уменьшении размера массива, значения удаленных элементов будут безвозвратно утеряны. С помощью оператора Erase можно очистить элементы фиксированного массива или освободить память, занимаемую динамическим массивом.

```
Dim A
ReDim A(25)
...
Erase A
```

1.6. Константы

Правилом хорошего тона является объявление констант для многократно используемых в тексте программы значений. Грамотно присвоенное имя константы улучшает читабельность, а само использование -- упрощает процесс внесения изменений в код. В отличие от переменных, значение константы нельзя изменить в процессе выполнения программы. Создание константы происходит с помощью оператора Const:

```
Const CountryName = "Belarus"
Const CountryCode = 375
```

Несколько констант могут быть объявлены в рамках одного оператора, через запятую. Как и переменная, константа обладает своей областью видимости в зависимости от того, где (в процедуре или за ее пределами) и как (Public или Private) она была объявлена. Константы, созданные оператором Const без указания Public или Private являются общедоступными по умолчанию.

Значения строковых констант заключаются в двойные кавычки.

Значения типа Дата следует обрамлять символами решетки (#) и использовать американский формат: месяц/день/год. Например:

```
Const Public IndependenceDay = #03/25/1918#
```

Во избежание путаницы между константами и переменными рекомендуется использовать единый префикс для всех констант, например "con", или набирать имя константы в верхнем регистре.

Для облегчения труда программиста VBScript содержит набор предопределенных констант.

1.7. Операторы

Операторы VBScript подразделяются на пять категорий: арифметические, сравнения, слияния, логические и присваивания.

Арифметические операторы

Оператор	Пример использования	Описание
^	number ^ exponent	Возводит number в степень exponent. Number может быть меньше нуля только в случае целочисленной степени. Если один из операндов Null, все выражение принимает значение Null. Если несколько возведений в степень выполняются подряд, результат вычисляется слева направо.
*	number1 * number2	Произведение двух чисел. Если операнд имеет значение Empty, то он принимается равным нулю. Если хотя бы один из операндов Null, все выражение принимает значение Null.
/	number1 / number2	Вещественное деление двух чисел. Для операндов действуют правила аналогично оператору умножения.
\	number1 \ number2	Целочисленное деление. Перед вычислением оба операнда приводятся к типу Byte, Integer или Long. В остальном действуют правила как для оператора деления.
Mod	number1 Mod number2	Остаток от целочисленного деления. Приведение операндов к целому, а также правила обращения с Empty и Null, как у целочисленного деления.
+	expression1 + expression2	Если оба операнда числа, результатом является их арифметическая сумма. Если оба операнда строки – слияние (конкатенция) двух строк. Если один операнд число, а другой строка, то строковый операнд будет преобразован в число и прибавлен к числовому. Если хотя бы один из операндов Null, все выражение принимает значение Null. Если оба операнда Empty, результат имеет целочисленное значение 0. Если только один оператор Empty, в качестве результата возвращается значение второго операнда.
-	number1 – number2 или - number	В первом случае возвращает разность двух чисел. Во втором – инвертирует знак числа. Правила для операндов со значениями Null и Empty, как для оператора умножения.

1.7.1. Операторы сравнения

Формат использования операторов сравнения:

```
result = expression1 comparisonoperator expression2
```

где используются следующие операторы сравнения: < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), = (равно), <> (не равно).

В зависимости от типов и значений операндов, сравнение осуществляется следующим образом:

Если	**То**
Оба операнда числа.	Выполняется сравнение двух чисел.
Оба операнда строки.	Выполняется сравнение двух строк.
Один из операндов число, а второй строка.	Строковый операнд приводится к числу и выполняется сравнение двух чисел.
Один из операндов Empty, а второй число.	Операнд со значением Empty принимается равным 0.
Один из операндов Empty, а второй строка.	Операнд со значением Empty принимается равным пустой строке "". Осуществляется сравнение двух строк.
Оба операнда Empty.	Операнды считаются равными.
Хотя бы один из операндов Null.	Результат принимает значение Null.

Специальный оператор Is применяется для сравнения двух объектных переменных и возвращает Истину, если обе переменных ссылаются на один и тотже экземпляр объекта.

1.7.2. Операторы конкатенции

В данной категории находятся два оператора: + и &. Первый подробно описан в разделе "Арифметические операторы" выше. Рассмотрим использование оператора &.

```
result = expression1 & expression2
```

Если операнд не является строкой, он приводится к строковому типу. Если оба операнда Null, то результат также принимает значение Null, однако, в отличие от остальных операторов, если только один операнд Null, то он принимается равным пустой строке. Операнд, имеющий значение Empty, также воспринимается как пустая строка "".

1.7.3. Логические операторы

VBScript предоставляет нам следующие логические операторы:

- Логическое отрицание, инверсия (Not);
- Логическое умножение, коньюнкция (And);
- Логическое сложение, дизьюнкция (Or);
- Логическое исключение (Xor);
- Логический эквивалент (Eqv);
- Логическая импликация (Imp).

В качестве операндов логических операторов могут выступать булевские выражения или числовые значения. В первом случае результатом будет булевская константа, во втором – число. В зависимости от оператора подача на вход одного или двух значений Null может приводить к Null результату. Оператор Not является унарным и возвращает логическое отрицание выражения. Над числовым операндом оператор Not производит побитовую инверсию. Остальные логические операторы являются бинарными. В таблице ниже приведены результаты выполнения каждого из операторов в зависимости от значения операндов Exp1 и Exp2:

Exp1	**Exp2**	**And**	**Or**	**Xor**	**Eqv**	**Imp**
True	True	True	True	False	True	True
True	False	False	True	True	False	False
False	True	False	True	True	False	True
False	False	False	False	False	True	True
True	Null	Null	True	Null	Null	Null
False	Null	False	Null	Null	Null	True
Null	True	Null	True	Null	Null	True
Null	False	False	Null	Null	Null	Null
Null	Null	Null	Null	Null	Null	Null

В жизни чаще всего используются операторы And и Or и гораздо реже – Xor. Нам не приходилось сталкиваться с использованием на практике операторов Eqv и Imp. Если вам тяжело разбираться с приведенной выше таблицей резюмируем действие данных операторов:

And принимает значение Истина только если оба операнда Истина. В любом другом случае – это либо Ложь, либо Null.

Or принимает значение Истина, если хотябы один из операндов Истина.

Xor принимает значение Истина, если значения операндов различаются и Ложь, если они одинаковы.

При побитовом выполнении над числовыми операндами результат логического оператора определяется по следующей таблице:

Exp1	Exp2	And	Or	Xor	Eqv	Imp
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	0
1	1	1	1	0	1	1

1.7.4. Оператор присваивания

Оператор присваивания (=) подробно описан в разделе "Переменные".

1.7.5. Очередность применения операторов

Если выражение содержит несколько операторов, то они применяются в соответствии с установленным порядком, который называется приоритетом операторов. Изменить порядок по умолчанию можно с помощью круглых скобок. Выражение внутри скобок всегда вычисляется в первую очередь.

В выражении, содержащем операторы разных категорий, арифметические действия выполняются в первую очередь, затем выполняются операторы сравнения и, в последнюю очередь, -- логические операторы. Все операторы сравнения имеют одинаковый приоритет и вычисляются слева-направо. Арифметические и логические операторы вычисляются в следующем порядке:

1. Возведение в степень (^);
2. Смена знака числа, унарный минус (-);
3. Умножение (*) и деление (/);
4. Целочисленное деление (\);
5. Остаток от целочисленного деления (Mod);
6. Сложение (+) и вычитание (-);
7. Слияние строк (&).

Если умножение и деление встречаются в одном выражении, то операции выполняются в порядке следования слева-направо. Аналогичное правило действует в случае одновременного присутствия операторов сложения и вычитания.

Оператор слияния строк (&) не является арифметическим и по приоритету располагается между арифметическими операторами и операторами сравнения.

Очередность для логических операторов установлена следующая:

1. Логическое отрицание, инверсия (Not);
2. Логическое умножение, коньюнкция (And);
3. Логическое сложение, дизьюнкция (Or);
4. Логическое исключение (Xor);
5. Логический эквивалент (Eqv);
6. Логическая импликация (Imp).

1.8. Условные выражения

Условные выражения применяются для управления порядком выполнения команд программы и позволяют организовать переходы (ветвления) и повторения команд. Как правило, операторы сравнения используются вместе с условными выражениями.

1.8.1. Выражение If..Then..Else

Выражение условного перехода If позволяет выполнить ту или иную группу команд в зависимости от результата логического выражения или значения булевской переменной.

Для выполнения единственной команды при выполнении заданного условия используется однострочный синтаксис выражения:

```
Dim S
If DatePart("w", Now) = vbMonday Then S = "Понедельник"
Application.MessageBox S, "", vbOkOnly
```

Обратите внимание, что секция Else в этом случае опущена. Для выполнения группы операторов следует заключить их между ключевыми словами Then и End If.

```
Dim S
If DatePart("w", Now) = vbMonday Then
  S = "Сегодня понедельник"
  Application.MessageBox S, "", vbOkOnly
End If
```

Если при выполнении условия требуется выполнить один код, а при невыполнении – другой, то используется синтаксис выражения с секцией Else:

```
Dim S
If DatePart("w", Now) = vbMonday Then
   S = "Сегодня понедельник"
 Else
   S = "Сегодня не понедельник"
End If
Application.MessageBox S, "", vbOkOnly
```

При необходимости выбора из нескольких альтернатив подойдет синтаксис с конструкцией ElseIf:

```
Dim S, D
D = DatePart("w", Now)
If D = vbMonday Then
  S = "Понедельник"
ElseIf D = vbTuesday Then
  S = "Вторник"
ElseIf D = vbWednesday Then
  S = "Среда"
...
End If
Application.MessageBox S, "", vbOkOnly
```

Выражения If могут быть вложенными:

```
Dim S, D
D = DatePart("w", Now)
If D = vbMonday Then
  S = "Понедельник"
Else
  If D = vbTuesday Then
    S = "Вторник"
  Else
    If D = vbWednesday Then
      S = "Среда"
    Else
      ...
    End If
  End If
End If
```

Хотя количество секций ElseIf в условном выражении не ограничено, интенсивное их использование может привести к запутанному, неудобочитаемому коду. В случае выбора одной альтернативы из множества возможных в зависимости от значения некоторого селектора рекомендуется использовать выражение Select Case.

1.8.2. Выражение Select..Case

Перепишем пример с днями недели с использованием выражения выбора:

```
Dim S
Select Case DatePart("w", Now)
  Case vbMonday
    S = "Понедельник"
  Case vbTuesday
```

```
      S = "Вторник"
    Case vbWednesday
      S = "Среда"
    ...
    Case Else
      Err.Raise 32000, "", "Неизвестный день недели"
End Select
```

Так как выражение селектора вычисляется только один раз, использование Select..Case приводит к более эффективному коду. Рекомендуется всегда использовать секцию Case Else для отлавливания некорректных или необработанных значений селектора.

1.9. Операторы цикла

Довольно часто возникает ситуация, когда код требуется запустить повторно. Для этого следует написать оператор цикла, который повторяет определенные команды снова и снова. Операторы цикла используются во многих ситуациях: при вычислении итоговой суммы по списку чисел, перемещении по записям набора данных или для запуска блока кода для нескольких объектов. Существует несколько циклов, описанных в следующих разделах. Некоторые из них выполняются, пока условие имеет значение Истина, некоторые – пока Ложь. И, наконец, есть такие, которые выполняются заданное число раз.

1.9.1. Оператор Do..Loop

Данный оператор предназначен для выполнения группы команд пока заданное условие Истинно или до тех пор, когда оно не станет Истинным. Проверка условия может осуществляться как в начале цикла:

```
Do [{While | Until} condition]
   [statements]
   [Exit Do]
   [statements]
Loop
```

так и в конце:

```
Do
 [statements]
 [Exit Do]
 [statements]
Loop [{While | Until} condition]
```

Команда Exit Do может встречаться неограниченное число раз в теле цикла. Обычно она используется вместе с условным выражением If..Then и позволяет передать управление на оператор, следующий непосредственно за циклом. При

использовании Exit Do внутри вложенного цикла, управление перейдет во внешний цикл.

Следующий код позволяет заменить игральный кубик:

```
Dim Resp, Num
Do
  Num = Int(6 * Rnd + 1)
  Resp = Application.MessageBox(Num & " Еще число?", "",_
    vbYesNo or vbQuestion)
Loop Until Resp = vbNo
```

1.9.2. While..Wend

Представляет собой усеченную версию оператора Do..Loop и позволяет выполнять группу команд пока условие Истинно. Синтаксис оператора:

```
While condition
  [statements]
Wend
```

Обратите внимание, что Exit Do не действует внутри данного цикла. Циклы While..Wend могут быть вложенными.

1.9.3. For..Next

Данный цикл повторяет заданный набор команд указанное число раз. Синтаксис оператора имеет вид:

```
For counter = start To end [Step step]
  [statements]
  [Exit For]
  [statements]
Next
```

Перед стартом цикла переменной counter присваивается значение start. Далее проверяется выполнение условия counter <= end, при step >= 0, или counter >= end, при отрицательном шаге. После выполнение блока команд переменная counter увеличивается на значение step и все повторяется сначала.

Изменение счетчика в теле цикла не запрещено, но настоятельно не рекомендуется, так как затрудняет понимание логики программы и ее отладку.

Exit For может встречаться в теле цикла произвольное число раз. Циклы могут быть вложенными. Например, такой цикл инициализирует трехмерный массив:

```
Dim A(9, 9, 9)
Dim I, J, K
```

```
For I = 0 To 9
  For J = 0 To 9
    For K = 0 To 9
      A(I, J, K) = 1
    Next
  Next
Next
```

1.9.4. For Each..Next

Оператор цикла For Each..Next повторяет заданный набор команд для каждого элемента массива или коллекции и имеет следующий синтаксис:

```
For Each element In group
  [statements]
  [Exit For]
  [statements]
Next
```

Цикл выполняется, если в массиве или коллекции присутствует хотябы один элемент. Exit For может встречаться в теле цикла произвольное количество раз.

Проиллюстрируем использование For Each..Next на примере следующего кода, который выводит на экран список файлов из корневого каталога диска c:\

```
Dim fso, f, f1, fc, s
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.GetFolder("c:\")
Set fc = f.Files
For Each f1 in fc
   s = s & f1.name & vbNewLine
Next
Application.MessageBox s, "Файлы на c:\", vbOkOnly
```

1.10. Процедуры

Для экономии памяти и структурирования программы, фрагмент кода, вызываемый многократно в разных местах, может быть оформлен ввиде процедуры. В языке VBScript существуют два вида процедур: подпрограммы (Sub) и функции (Function). Подпрограмма – это последовательность операторов, обрамленная ключевыми словами Sub и End Sub. Подпрограмма может принимать на вход параметры, но не возвращает значения. Функция – последовательность операторов, заключенная между Function и End Function, -- возвращает результат и поэтому может быть использована в выражении. Каждая процедура должна иметь имя, уникальное в пределах модуля. Имена процедур, объявленных в глобальном модуле, должны быть уникальны в рамках всего проекта.

Определение подпрограммы и функции имеет следующий синтаксис:

```
[Public [Default] | Private] Sub name [(arglist)]
  [statements]
  [Exit Sub]
  [statements]
End Sub
[Public [Default] | Private] Function name [(arglist)]
  [statements]
  [Exit Function]
  [statements]
End Function
```

Public процедуры являются глобальными и доступны во всех скриптах программы. Private процедуры доступны только в том скрипте, где они были объявлены. Если не указано иное, объявленная процедура является общедоступной. Ключевое слово Default может быть использовано только в теле класса и служит для указания метода по-умолчанию этого класса.

Список параметров имеет следующий синтаксис:

```
[ByRef | ByVal] varname [, ...]
```

Параметры могут передаваться по значению (ByVal) или по ссылке (ByRef). По-умолчанию все параметры передаются по значению. Константы, результаты вычисления выражений могут быть переданы только по значению. Изменение параметра, переданного по ссылке, приведет к изменению значения наружной переменной. Поясним передачу параметров внутрь процедуры на следующем примере:

```
Sub DoCalculation(ByRef A, ByVal B, ByVal C)
  A = C * 2
  B = C / 2
End Sub

Sub TestVar
  Dim V1, V2
  V1 = 1
  V2 = 2
  DoCalculation V1, V2, 10
  ' После выполнения процедуры DoCalculation
  ' V1 = 20
  ' V2 = 2
End Sub
```

Переменные, объявленные внутри тела процедуры, являются локальными и уничтожаются по завершении ее выполнения. Значения локальных переменных не сохраняются.

Список параметров указывается в круглых скобках при вызове функции или при вызове подпрограммы с помощью оператора Call. Так, вызов процедуры DoCalculation в приведенном выше примере мы могли написать следующим образом:

```
Call DoCalculation(V1, V2, 10)
```

Использованные источники:

1. Основы VBScript: http://gsbelarus.com/gs/wiki/index.php/VBScript._%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B_VBScript

1. Введение в ASP

1.1. Общие сведения

ASP (Active Server Pages) – это мощная технология от Microsoft, позволяющая легко разрабатывать приложения для WWW. ASP работает на платформе Windows NT и IIS (Internet Information Server), начиная с версии 3, хотя вроде есть реализации на других платформах. ASP – это не язык программирования, это внутренняя технология, позволяющая подключать программы к Web-страницам. Основа успеха ASP – простой скриптовый язык (Visual Basic Script или Java Script) и возможность использования внешних COM-компонент.

Как это все происходит?

Вы пишете программу и складываете в файл на сервере. Браузер клиента запрашивает файл. Файл сначала интерпретируется сервером, на выходе производится HTML-код. Этот HTML посылается клиенту. Файлы с программами имеют расширение .asp. Файлы asp – это обычные текстовые файлы, содержащие исходные тексты программ. Файлы делаются с помощью любого текстового редактора. Каталог, в котором размещены файлы asp должен иметь права на выполнение, так как сервер исполняет эти файлы, когда браузер их запрашивает. Собственно программы пишутся на любом скриптовом языке, который установлен в системе. По умолчанию поддерживаются VBScript и JavaScript. Можно доустановить другие (например, Perl). Если ничего специально не указывать используется VBScript. В дальнейшем будем ссылаться только на него. Программные фрагменты заключаются в скобки <% %>. Можно ставить открывающую скобку в начале файла, закрывающую – в конце, все что между ними – программа на Visual Basic'e.

Какие средства есть для программирования?

Web – нормальная среда программирования, если правильно понять, что есть что. В VBScript есть все нормальные конструкции структурного программирования (if, while, case, etc). Есть переменные (описывать не обязательно, тип явно не задается). Поддерживаются объекты. Работа с ними обычная – Object.Property, Object.Method. Есть ряд встроенных объектов (Request, Response, Session, Server, Connection, Recordset). Можно доустанавливать другие компоненты (скачивать, покупать, программировать), например для работы с электронной почтой.

Вывод

Понятия "экран", куда можно выводить данные нет. Все, что надо показать пользователю, выбрасывается в выходной поток на языке HTML. Браузер пользователя интерпретирует этот HTML. Для упрощения вывода существует объект **Response**. Вывод осуществляется с помощью метода Write.

```
Response.Write("<h2>Hello, world!</h2>").
```

Так производится запись во внутренний буфер объекта Response. Когда скрипт заканчивает работу, весь буфер выдается клиенту. Надо заметить, что клиент получает "чистый" HTML, таким образом программы на ASP не зависят от клиентского ПО, что очень важно. Если внутри выводимой строки нужно использовать кавычку, кавычка удваивается. Другие методы и свойства Response позволяют управлять выводом. Так Response.Buffer регулирует, получает ли клиент данные по мере из записи в Response, или все сразу по завершении исполнения страницы. Метод Response.Redirect перенаправляет браузер на другую страницу. Чтобы им пользоваться, нельзя до него на странице использовать Response.Write.

Ввод

Программа на ASP не может явно спросить пользователя о чем-то. Она получает данные из других страниц, либо через URL. Передаваемые параметры помещаются во входной поток и доступны через объект **Request**. Чтобы передать переменную **var** в программу **test.asp**, надо написать:

```
test.asp?var=abc
```

Чтобы из программы получить значение этой переменной, надо написать:

```
var = Request("var")
```

Несколько переменных разделяется знаком &:

```
test.asp?var1=abc&var2=def
```

Кроме того, чтобы задавать параметры в URL, можно воспользоваться формами HTML. В вызывающей странице пишем так:

```
<form method="get" action="test.asp">
<input type=text name="var1" value="default">
<input type=hidden name="var2" value="var2value">
<input type=submit value="Submit Form">
</form>
```

При этом пользователь увидит форму из одного поля ввода (var1), в нем будет значение по умолчанию "default". Второе поле (var2) будет невидимо и будет передавать всегда фиксированное значение "var2value". Кнопка "Submit Form" завершает заполнение формы и передает все переменные на test.asp (action). Если method="get", переменные передаются через URL (test.asp?var1=default&var2=var2value). Если method="post", передаются вместе с запросом так, что внешне передача переменных не заметна. В вызываемой программе безразлично, какой метод изпользовался (почти). Если у вас нет специальных аргументов за метод GET, используйте метод POST.

Формы

Формы HTML используются для организации диалога с пользователем. Поддерживаются стандартные элементы управления. Все многообразие задается немногими тэгами:

- INPUT (с параметром TYPE=)
- SELECT
- TEXTAREA

Описание – в документации по HTML.

Взаимосвязь между отдельными страницами

Обычно сервер WWW не хранит состояние приложения, т.е. все запросы взаимонезависимы, и нет стандартного способа понять, что несколько запросов пришли от одного и того же пользователя. Но это очень нужно для разработки полноценных приложений и является одной из главных проблем разработки Web-приложений.

Один из методов решения этой проблемы - cookies. Пользователю при первом обращении выдается специальный идентификатор, после этого браузер пользователя предъявляет этот идентификатор при каждом обращении, и сервер может распознать, что это тот же самый пользователь. Пользователь может отключить cookies, в этом случае этот метод не работает.

ASP, используя cookies, предоставляет программисту более простое средство - объект Session (сессия). Сессия стартует, когда новый пользователь обращается к любому asp-файлу приложения. Сессия заканчивается при отсутствии активности пользователя в течение 20 минут, либо по явной команде. Специальный объект Session хранит состояние сессии. Туда можно записывать переменные, которые доступны из любой страницы в этой сессии. Записать данные в этот объект можно просто:

```
Session("var") = var
```

Считать потом еще проще:

```
var = Session("var")
```

Сессия, таким образом, – это еще один метод передачи данных между страницами. Одна страница пишет данные в сессию, другая – берет потом оттуда.

Наряду с объектом Session существует объект Application. Если сессия создается для каждого нового пользователя, до Application существует в единственном экземпляре, и может использоваться всеми страницами приложения.

```
Application("var") = var
var = Application("var")
```

Управление приложением

Программисту предоставляется возможность реагировать на 4 события: старт/стоп приложения и старт/стоп каждой сессии. Для реализации этих событий предназначен файл global.asa, который должен располагаться в корневом каталоге приложения. Вот его примерный скелет:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Application_OnStart
END SUB
SUB Application_OnEnd
END SUB
SUB Session_OnStart
END SUB
SUB Session_OnEnd
END SUB
</SCRIPT>
```

Нужно "просто" вписать Ваш код на соответствующее место. Нужно заметить, что отлаживать код для global.asa довольно непросто, так как он выполняется при очень специфических обстоятельствах (к примеру при старте или остановке сервера).

Использование внешних компонент

Если на сервере установлены дополнительные компоненты, их можно использовать из ASP. Стандартные объекты (например из библиотек ADO (Connection и Recordset) и Scripting (Dictionary, FileSystemObject)) доступны всегда. Установка новой компоненты обычно состоит в копировании dll-файла в каталог на сервере и ее регистрации с помощью программы regsvr32.exe. [В COM+ используется своя процедура инсталляции объектов, это однако не влияет на использования объектов.]

Создать экземпляр объекта можно так:

```
Set var  = Server.CreateObject("Class.Object")
```

Class.Object указываются в документации на компоненту. В переменной var запоминается ссылка на созданный экземпляр объекта. Когда объект не нужен, ссылку нужно обнулить с помощью команды:

```
Set var = Nothing
```

Пожалуйста всегда обнуляйте все ссылки на объекты, когда они больше не нужны. Теоретически это должно происходить автоматически при завершении процедуры/страницы, однако в стандартной сборке мусора есть определенные "проблемы".

В остальном использование компоненты зависит от самой этой компоненты.

Работа с базами данных

Из ASP можно легко и просто работать с любыми базами данных. Это делается через две промежуточные технологии: ODBC и ADO.

ODBC позволяет организовать доступ к любым базам данных через унифицированный интерфейс с помощью языка SQL. Специфика конкретных СУБД учитывается при помощи специальных драйверов БД. Такие драйверы существуют для всевозможных СУБД (в частности SQL Server, Oracle, Access, FoxPro). Поддержка ODBC обеспечивается на уровне операционной системы Windows (NT). Настройка – через Control Panel/ODBC. Базовым понятием является источник данных или data source. Источник данных – это совокупность сведений о базе данных, включая ее драйвер, имя компьютера и файла, параметры. Чтобы пользоваться базой надо создать источник данных для нее. Важно, чтобы источник данных был "системным", в отличии от "пользовательского". После этого надо лишь знать имя источника данных. [В настоящее время ODBC отступает перед натиском технологии OLE DB. На практике это однако практически ничего не изменяет. Вместо имени источника данных нужно использовать Connection String, в которой указывается имя ODBC-драйвера и все его параметры.]

ADO – это совокупность объектов, доступных из ASP, позволяющих обращаться к источнику данных ODBC [или OLE DB]. Фактически нужны лишь 2 объекта – Connection, представляющий соединение с базой данных и Recordset, представляющий набор записей, полученный от источника. Сначала необходимо открыть соединение, потом к нему привязать Recordset, потом, пользуясь методами Recordset'a, обрабатывать данные. Вот пример:

```
<%
Dim Conn, RS, strSQL, strOut
strOut = ""
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Data-Source-Name"
Set RS = Server.CreateObject("ADODB.Recordset")
strSQL = "SELECT * FROM AGENTS ORDER BY USER_ID"
RS.Open strSQL, Conn
RS.MoveFirst
strOut = strOut & "<P>Here is the data:"
strOut = strOut & "<TABLE BORDER=""1"">"
strOut = strOut & "<TR><TH>USER_ID</TH><TH>Name</TH></TR>"
Do While Not RS.EOF
   strOut = strOut & "<TR>"
   strOut = strOut & "<TD>" & RS.Fields("USER_ID") & "</TD>"
   strOut = strOut & "<TD>" & RS.Fields("NAME") & "</TD>"
   strOut = strOut & "</TR>"
   RS.MoveNext
Loop
strOut = strOut & "</TABLE>"
strOut = strOut & "<HR>"
strOut = strOut & "That's it!"
RS.Close
```

```
Set RS = Nothing
Conn.Close
Set Conn = Nothing
Response.Write strOut
%>
```

Если команда SQL не возвращает данных, recordset не нужен, надо пользоваться методом Conn.Execute(SQL_COMMAND).

Если Вы хотите вызывать хранимые процедуры сервера БД с параметрами, нужно воспользоваться объектом Command, который в свою очередь содержит объекты Parameter.

1.2. Советы

1.2.1. Описание переменных

VBScript - очень нетребовательный к программисту язык. Так он не требует описывать переменные и не содержит явных типов данных. Все переменные принадлежат одному типу Variant. Из-за отсутствия описаний могут произойти очень трудно обнаруживаемые ошибки. Одна опечатка может стоить полдня поисков.

Однако, есть возможность явно потребовать описания переменных. Для этого первой строкой в ASP-файле нужно написать Option Explicit. После этого обращение к переменной, которая не была объявлена с помощью Dim, вызывает ошибку с указанием номера строки.

Кстати, где расположены описания Dim в процедуре - совершенно не важно. Они могут стоять как до использования переменной, так и после, и даже в цикле. Видимо они отрабатываются препроцессором. Явно задать тип переменной с помощью Dim **Var as Typ**, как в Visual Basic, все равно нельзя.

1.2.2. Чередование ASP/HTML

Если нужно выдать большой кусок HTML, можно не пользоваться Response.Write. Если в asp-файле встречается кусок текста вне скобок <% %>, он трактуется просто как HTML, который надо вывести. Пример:

```
<% If var="VAL" Then %>
   Дорогой друг!
   …
<% Else %>
   Извините, вам сюда нельзя
   …
<% End If %>
```

1.2.3. Обработка ошибок

Для отслеживания ошибок используется специальный объект Err. Он устанавливается в ненулевое значение, если предыдущая команда породила ошибку. Ее можно проверять с помощью If, и таким образом реагировать на ошибки. Чтобы из-за ошибки не прерывалось выполнение программы, в начале нужно включить команду

```
On Error Resume Next
```

1.2.4. Включение других файлов

Можно выносить повторяющийся код в отдельный файл, и подключать к разным другим по мере необходимости с помощью команды **include**. Это очень удобно, если вы хотите вынести повторяющийся код в отдельный файл и использовать снова и снова в разных страницах:

```
<!--#include file="filename.inc" -->
```

Важно**:** все includes в тексте отрабатываются **до** исполнения файла. Т.е. даже если include стоит внутри if, то сначала будут включены все includes во всех ветках, и только потом, во время исполнения, будет принятно решение, какую ветку выполнять. Т.е. следующий код НЕ дает условного включения файлов:

```
<% If var="VAL" Then %>
   <!--#include file="filename1.inc" -->
<% Else %>
   <!--#include file="filename2.inc" -->
<% End If %>
```

1.2.5. Обработка форм

Если надо что-то спросить у пользователя и на основании этого что-то сделать, в простейшем случае создается два файла: один с формой, второй – с ее обработчиком. Обработчик выполняет все действия. Пример:

```
form.htm:
<form method="post" action="run.asp">
<input type=text name="var1" value="default">
<input type=hidden name="var2" value="var2value">
<input type=submit value="Обработать">
</form>

run.asp:
var1 = Request("var1")
var2 = Request("var2")
…
что-то делаем
```

```
…
Response.Write("все хорошо!<br>”)
```

1.2.6. Рекурсивная обработка форм

Удобный метод состоит в том, чтобы сбор данных и обработку осуществлял один и тот же файл. Для этого пишется asp, в котором есть разные разделы. Специальная переменная отвечает за выбор раздела при запуске. Пример:

```
test.asp:
    <%
    Action = Request("Action")
    If Action = "" Then
       ' Action не заполнен – значит надо показать форму
       Action = "FORM"
    End If
    Select Case Action
    Case "FORM"
       ' Показываем форму
       ' action- сам на себя, Action="PROCESS", т.е.
       ' второй раз пойдет по другой ветке
    %>
    <form method="post" action="test.asp">
    <input type=hidden name="Action" value="PROCESS">
    <input type=text name="var1" value="default">
    <input type=submit value="Обработать">
    </form>
    <%
    Case "PROCESS"
       ' Обрабатываем
    var1 = Request("var1")
    var2 = Request("var2")
    …
    что-то делаем
    …
    Response.Write("все хорошо!<br>")
    End Select
    %>
```

1.2.7. Переменные HTTP

Запрос от браузера, кроме запрашиваемой страницы несет еще некоторые данные. Эти данные, например, IP-адрес клиента, доступны через специальные переменные объекта Request. IP-адрес – Request("REMOTE_ADDR"). Другие - см.документацию (ASPSamp\Samples\srvvar.asp).

1.2.8. Переадресация

Очень легко написать на ASP скрипт, который будет производить некоторые расчеты, и в зависимости от результатов переадресовывать браузер на разные URL (например, подставлять нужный баннер). Делается это так:

```
Response.Redirect "somewhere.html"
```

Только надо следить, чтобы до выполнения команды redirect ничего не было записано в Response (даже коментарии HTML).

1.2.9. Электронная почта

Одна из часто встречающихся задач – отправить электронную почту с Web-страницы. На первый взгляд, можно просто написать

```
<FORM ACTION="mailto:user@host.com">
```

Но это приводит к тому, что при отправке формы делается попытка на клиентской машине запустить почтовую программу и создать новое сообщение с данными формы. Если это не получается (почтовая программа не настроена, пользователь не отправил почту, и т.д.) - письмо и не будет отправлено. Гораздо надежнее работает серверное решение.

Для этого существуют внешние компоненты, есть и бесплатные. Например, компонента Jmail от Dimac. Все, что для нее нужно – это адрес SMTP-сервера. Вот пример ее использования:

```
On Error Resume Next
Set Mailer = Server.CreateObject("JMail.SMTPMail")
If err<>0 Then
   Response.Write("Невозможно создать объект")
Else
   Mailer.ServerAddress = SMTP_SERVER
   Mailer.Sender = EMAIL
   Mailer.Subject = "Service: Delivery order"
   Mailer.AddRecipient ADMIN_EMAIL
   Mailer.Body = "-----------------------------------" &
chr(10)
   Mailer.AppendText "body text"
   Mailer.Execute
   If err<>0 Then
      Response.Write("Mail error")
   Else
      Response.Write("Mail send Ok")
   End If
End If
Set Mailer = Nothing
```

Использованные источники:

1. Григорий Грин. Введение в ASP: http://citforum.ru/internet/asp/asp_intro.shtml

1. Введение в БД

1.1. Основные понятия БД

Базы данных могут содержать различные объекты, но основными объектами любой базы данных являются ее таблицы. Простейшая база данных имеет хотя бы одну таблицу. Соответственно, структура простейшей базы данных тождественно равна структуре ее таблицы, рис. 11.1.

Мы знаем, что структуру двумерной таблицы образуют столбцы и строки. Их аналогами в структуре простейшей базы данных являются *поля* и *записи*. Если записей в таблице пока нет, значит, ее структура образована только набором полей. Изменив состав полей базовой таблицы (или их свойства), мы изменяем структуру базы данных и, соответственно, получаем новую базу данных.

Рис. 11.1. Простая таблица базы данных

1.1.1. Свойства полей базы данных

Поля базы данных не просто определяют структуру базы — они еще определяют групповые свойства данных, записываемых в ячейки, принадлежащие каждому из полей. Ниже перечислены основные свойства полей таблиц баз данных на примере СУБД Microsoft Access.

- **Имя поля** — определяет, как следует обращаться к данным этого поля при автоматических операциях с базой (по умолчанию имена полей используются в качестве заголовков столбцов таблиц).
- **Тип поля** — определяет тип данных, которые могут содержаться в данном поле.
- **Размер поля** — определяет предельную длину (в символах) данных, которые могут размещаться в данном поле.

- **Формат поля** — определяет способ форматирования данных в ячейках, принадлежащих полю.
- **Маска ввода** — определяет форму, в которой вводятся данные в поле (средство автоматизации ввода данных).
- **Подпись** — определяет заголовок столбца таблицы для данного поля (если подпись не указана, то в качестве заголовка столбца используется свойство Имя поля).
- **Значение по умолчанию** — то значение, которое вводится в ячейки поля автоматически (средство автоматизации ввода данных).
- **Условие на значение** — ограничение, используемое для проверки правильности ввода данных (средство автоматизации ввода, которое используется, как правило, для данных, имеющих числовой тип, денежный тип или тип даты).
- **Сообщение об ошибке** — текстовое сообщение, которое выдается автоматически при попытке ввода в поле ошибочных данных (проверка ошибочности выполняется автоматически, если задано свойство Условие на значение).
- **Обязательное поле** — свойство, определяющее обязательность заполнения данного поля при наполнении базы;
- **Пустые строки** — свойство, разрешающее ввод пустых строковых данных (от свойства Обязательное поле отличается тем, что относится не ко всем типам данных, а лишь к некоторым, например к текстовым).
- **Индексированное поле** — если поле обладает этим свойством, все операции, связанные с поиском или сортировкой записей по значению, хранящемуся в данном поле, существенно ускоряются. Кроме того, для индексированных полей можно сделать так, что значения в записях будут проверяться по этому полю на наличие повторов, что позволяет автоматически исключить дублирование данных.

Здесь нужно обратить особое внимание на то, что поскольку в разных полях могут содержаться данные разного типа, то и свойства у полей могут различаться в зависимости от типа данных. Так, например, список вышеуказанные свойств полей относится в основном к полям текстового типа. Поля других типов могут иметь или не иметь эти свойства, но могут добавлять к ним и свои. Например для данных, представляющих действительные числа, важным свойством является количество знаков после десятичной запятой. С другой стороны, для полей, используемых для хранения рисунков, звукозаписей, видеоклипов и других объектов OLE, большинство вышеуказанных свойств не имеют смысла.

1.2. Механизмы доступа к данным. OLE DB и ADO

Рассмотрим универсальный механизм доступа к данным — Microsoft ADO (ActiveX Data Objects), ныне широко применяемый не только в средствах разработки

фирм Microsoft и Borland, но и во многих «пользовательских» продуктах, таких как Microsoft Office, Microsoft Internet Explorer, в ASP-приложениях и др.

ADO становится все более и более популярным способом доступа к данным, так как не только входит в состав Microsoft Office 2000 и Microsoft Internet Explorer 5.0, но и является частью ядра операционных систем семейства Windows 2000. В предыдущей статье цикла мы упоминали, что OLE DB и ADO, являясь частью универсального механизма доступа к данным фирмы Microsoft (Microsoft Universal Data Access), позволяют осуществить доступ как к реляционным, так и к нереляционным источникам данных, таким как файловая система, данные электронной почты, многомерные хранилища данных и др. Эту статью мы хотим целиком посвятить универсальному механизму доступа к данным фирмы Microsoft.

1.3. Microsoft Universal Data Access

Универсальный механизм доступа к данным (Universal Data Access) являет собой стратегию предоставления доступа к любому типу информации предприятия. Он обеспечивает высокопроизводительный доступ к различным источникам информации (включая реляционные и нереляционные данные), в том числе к данным, хранящимся на мэйнфреймах, данным электронной почты и файловой системы, текстовым, графическим и географическим данным и др. Для многих современных приложений, использующих данные, характерно подобное разнообразие их источников. Более того, вполне очевидно, что могут появляться новые форматы данных и способы их хранения, поэтому разумным требованием к универсальному механизму доступа к данным была бы возможность поддержки не только существующих в настоящее время форматов и источников данных, но и форматов данных, которые будут созданы в будущем.

Назначение универсального механизма доступа к данным фирмы Microsoft — предоставить доступ к перечисленным источникам данных с помощью единой модели доступа к данным.

В настоящее время универсальный механизм доступа к данным фирмы Microsoft поддерживает все наиболее популярные настольные и серверные СУБД, о которых мы писали в предыдущих статьях данного цикла.

Далее мы рассмотрим основные компоненты архитектуры универсального механизма доступа к данным Microsoft и обсудим их более детально:

- **Microsoft ActiveX Data Objects (ADO)** представляет собой программный интерфейс для доступа к данным из приложений. С точки зрения программирования ADO и его расширения являются упрощенным высокоуровневым объектно-ориентированным интерфейсом к OLE DB;
- **OLE DB** — это низкоуровневый интерфейс для доступа к данным. ADO использует OLE DB, но можно использовать OLE DB и напрямую, минуя ADO;

- **Open Database Connectivity (ODBC)**, уже обсуждавшийся в одной из предыдущих статей данного цикла, — стандартный способ доступа к реляционным данным. Этот компонент универсального механизма доступа к данным оставлен с целью обеспечения совместимости с прежними версиями программного обеспечения. В современных приложениях применению ODBC-драйверов предпочитают использование OLE DB-провайдеров.

Архитектура универсального механизма доступа к данным Microsoft схематически представлена на рис.1.

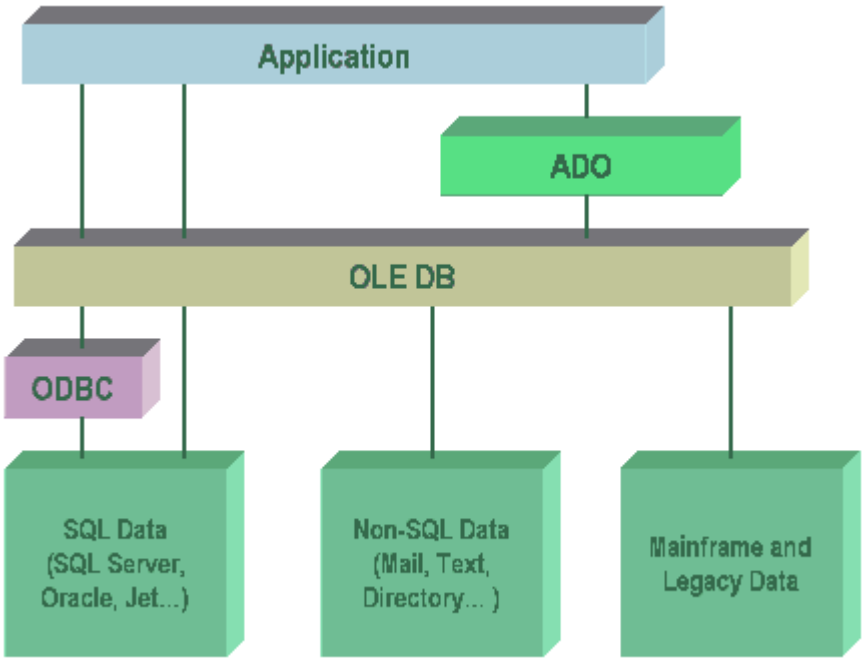


Рис. 1

Так как OLE DB является низкоуровневым интерфейсом доступа к данным, мы начнем рассмотрение составных частей универсального механизма доступа к данным фирмы Microsoft именно с него. Ниже мы обсудим архитектуру и интерфейсы OLE DB, а затем покажем, как OLE DB взаимодействует с ADO.

1.4. OLE DB

Итак, OLE DB представляет собой программный интерфейс для доступа к различным источникам данных, таким как реляционные и нереляционные данные, текстовые, графические и географические данные, архивы электронных писем, файловая система, бизнес-объекты. В спецификации OLE DB определен набор COM-интерфейсов (COM, Component Object Model, компонентная модель объектов Microsoft, являющаяся составной частью 32-разрядных версий Windows), инкапсулирующих различные сервисы управления данными и предоставляющих однотипный доступ к перечисленным выше данным. Эти интерфейсы могут быть использованы в приложениях, предоставляющих доступ к данным.

1.4.1. Компоненты OLE DB

На самом верхнем уровне можно отметить три главных компонента OLE DB: потребители (consumers), провайдеры данных (**data** providers) и сервисные компоненты (**service** components).

Любой компонент программного обеспечения, применяющий интерфейсы OLE DB, является потребителем. Это может быть какое-либо офисное приложение или иное бизнес-приложение, средство разработки типа Visual Basic или Delphi либо даже COM-объекты для доступа к данным, применяющие интерфейсы OLE DB. Потребители могут обращаться к данным посредством ActiveX Data Objects, представляющих собой высокоуровневый интерфейс к OLE DB, или применять OLE DB непосредственно, используя OLE DB-провайдер.

Провайдер — это часть программного обеспечения, в которой реализованы интерфейсы OLE DB. С точки зрения OLE DB существуют два типа OLE DB-провайдеров — провайдеры данных и сервисные компоненты.

Провайдер данных — это компонент программного обеспечения, манипулирующий данными. Он располагается между потребителем данных и базой данных. В OLE DB все провайдеры представляют данные в табличном формате (аналогичном тому, в котором хранятся данные в реляционных СУБД и файлах электронных таблиц), в виде виртуальных таблиц. Провайдер данных выполняет следующие функции:

- получение от потребителя запросов на получение или модификацию данных;
- получение данных из базы данных или их модификацию в базе данных;
- возвращение данных потребителю.

Примером провайдеров данных является провайдер Microsoft Jet 4.0 OLE DB Provider, который используется для доступа к данным Microsoft Access, а также к данным I-ISAM (Installable Indexed Sequential Access Method), файлам рабочих книг Excel, хранилищ данных Microsoft Outlook и Microsoft Exchange, таблиц dBase и Paradox, текстовым файлам, файлам в формате HTML и др. Еще один пример OLE DB-провайдера — Microsoft OLE DB Provider for SQL Server, применяемый для доступа к базам данных Microsoft SQL Server 6.5 и 7.0.

Провайдер сервисов (или сервисный компонент) реализует расширенную функциональность, не поддерживаемую обычными провайдерами данных, например сортировку и фильтрацию данных, обработку транзакций и SQL-запросов, управление курсором и др. Сервисный компонент может обращаться к хранилищу данных непосредственно или с помощью соответствующего провайдера данных — в этом случае провайдер сервисов является одновременно и провайдером, и потребителем. Например, сервисные компоненты, такие как Microsoft Cursor Service for OLE DB и Microsoft Data Shaping Service for OLE DB, могут использоваться совместно с провайдерами данных OLE DB для расширения их функциональности.

В табл. 1 приведен список провайдеров, доступных в составе набора MDAC (Microsoft Data Access Components), поставляемого с рядом продуктов Microsoft.

Провайдер	Описание
Microsoft OLE DB Provider for ODBC Drivers	Позволяет осуществить доступ к любому ODBC-источнику
Microsoft Jet 4.0 OLE DB Provider	Применяется для доступа к данным Microsoft Access и некоторых других СУБД
Microsoft OLE DB Provider for SQL Server	Используется для доступа к данным Microsoft SQL Server
Microsoft OLE DB Provider for Oracle	Используется для доступа к данным Oracle
Microsoft OLE DB Provider for Internet Publishing	Используется для доступа к Web-серверам и ресурсам, обслуживаемым Microsoft FrontPage или Microsoft Internet Information Server
OLE DB Provider for Microsoft Directory Services	Применяется для доступа к гетерогенным службам каталогов с помощью Microsoft Active Directory Service Interfaces (ADSI)
Microsoft OLE DB Provider for Microsoft Index Server	Предоставляет доступ «только для чтения» к файловой системе и данным Web, индексированным с помощью Microsoft Indexing Service
Microsoft OLE DB Simple Provider	Используется для доступа к текстовым файлам
Cursor Service for OLE DB	Расширяет функциональность курсора
Data Shaping Service for OLE DB	Применяется для создания связей между наборами данных и получения иерархических наборов данных
OLE DB Persistence Provider	Используется для сохранения наборов данных в форматах ADTG (Advanced Data Table Gram) или XML (eXtensible Markup Language)
OLE DB Remoting Provider	Позволяет обращаться к провайдерам данных на удаленных компьютерах
Microsoft OLE DB Provider for OLAP Services	Применяется с расширением ADO Multi-Dimensional (ADO MD) для доступа к многомерным данным, созданным с помощью Microsoft SQL Server 7.0 OLAP Services

Как мы уже писали, OLE DB представляет собой набор COM-интерфейсов. На рис. 3 схематически представлены четыре основных объекта и их интерфейсы, а также методы, с помощью которых они взаимодействуют.

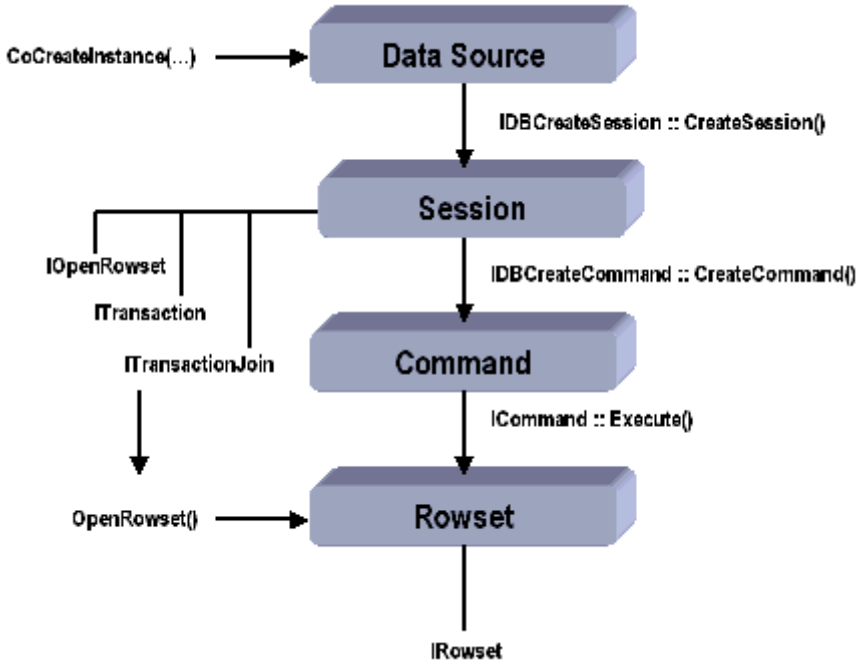


Рис. 3

Отметим, что каждый OLE DB-провайдер должен содержать реализацию объектов DataSource, Session и Rowset. Помимо трех основных объектов могут быть и другие.

Рассмотрим эти объекты более подробно.

1.4.2. Объекты OLE DB

Объектная модель OLE DB содержит четыре ключевых объекта:

- DataSource;
- Session;
- Command;
- Rowset.

1.4.3. Объект DataSource

Объект DataSource, применяемый потребителями данных для соединения с провайдером, может быть создан различными способами, включая вызов функции CoCreateInstance с идентификатором класса (CLSID, Class Identifier) OLE DB-провайдера, использование объекта Enumerator (см. ниже), который занимается поиском источников данных, и пр. Объект DataSource инкапсулирует информацию, связанную с соединением (включая имя пользователя и пароль). Основное назначение этого объекта — предоставлять данные из источника данных потребителю.

Для создания новой сессии (объект Session) потребитель должен вызвать метод CreateSession интерфейса IDBCreateSession объекта DataSource.

1.4.4. Объект Session

Объект Session предоставляет контекст для транзакций, может генерировать наборы данных (rowsets) из источников данных, а также команды для запросов к источнику данных. Объект Session может выполнять роль фабрики классов для объектов Command и Rowset (см. ниже) и объекта Transaction, применяемого для управления вложенными транзакциями. Объекты Command и Rowset могут быть использованы для создания или модификации таблиц и индексов. Интерфейс IOpenRowset используется потребителями данных для работы с отдельными таблицами и индексами в хранилище данных.

С одним объектом DataSource может быть связано несколько объектов Session.

Если OLE DB-провайдер поддерживает команды или запросы, он должен уметь порождать объект Command. С одним объектом DataSource может быть связано несколько объектов Command. Для создания нового объекта Command применяется метод CreateCommand интерфейса IDBCreateCommand.

1.4.5. Объект Command

Объект Command используется для выполнения команд, представляющих собой строки, передаваемые от потребителя данных объекту Data Source для выполнения. В большинстве случаев такая команда представляет собой SQL-предложение SELECT, однако в общем случае это может быть любое другое SQL-предложение (например, DDL-предложение). Команды могут содержать параметры — в этом случае применяется интерфейс ICommandWithParameters. Одна сессия может порождать несколько команд. Результатом выполнения команды (с помощью метода Execute интерфейса ICommand) обычно является новый объект Rowset.

1.4.6. Объект Rowset

Объект Rowset (набор данных) позволяет OLE DB-провайдеру данных представлять данные из источников данных в табличном формате, то есть в виде набора строк, содержащих одну или несколько колонок. Этот объект может быть результатом выполнения команды или может быть сгенерирован непосредственно провайдером данных, если провайдер не поддерживает команд. Все провайдеры данных «умеют» создавать наборы данных напрямую. Объект Rowset может быть также использован для обновления, добавления или удаления строк — это зависит от функциональности провайдера данных.

С помощью интерфейса IRowset из объекта Rowset потребители могут перемещаться по набору данных вперед и, если набор данных позволяет, назад. Некоторые провайдеры могут предоставлять дополнительные функции наподобие

непосредственного доступа или определения примерной позиции данной строки в наборе.

Частным случаем объекта Rowset является объект Index, предоставляющий набор строк, использующий соответствующий индекс для получения набора данных в упорядоченном виде.

Существуют также специальные объекты типа Rowset — schema rowsets, содержащие метаданные (то есть сведения о структуре данных), и view rowsets, содержащие подмножество строк и столбцов объекта Rowset.

Помимо четырех основных объектов, перечисленных выше, существуют и другие объекты OLE DB. Они нужны для перечисления источников данных, управления транзакциями, обработки ошибок и др. Некоторые из них мы рассмотрим ниже.

1.4.7. Объект Enumerator

Объект Enumerator необходим для получения списка доступных объектов, обеспечивающих доступ к источникам данных (OLE DB-провайдеров). Этот объект используется потребителями данных для поиска соответствующих объектов. В большинстве случаев сведения, возвращаемые объектом Enumerator, извлекаются из системного реестра. Этот объект реализует интерфейс ISourceRowset и возвращает объект Rowset с описанием всех источников данных и других доступных с его помощью объектов Enumerator. Для этой цели используется метод GetSourcesRowset интерфейса ISourceRowset.

1.4.8. Объект Transaction

Объект Transaction поддерживает транзакции в источнике данных. Транзакции, как мы уже знаем из первой статьи данного цикла, позволяют определить группу операций, которые либо все вместе выполняются, либо все вместе отменяются.

Транзакции бывают локальными и распределенными. Локальные транзакции - это транзакции, выполняемые в контексте единого провайдера данных. Провайдер, поддерживающий локальные транзакции, должен реализовать интерфейс ITransactionLocal. Транзакция начинается с вызова метода StartTransaction, завершается с помощью метода Commit или откатывается с помощью Abort. Способность провайдера поддерживать транзакции может быть определена с помощью интерфейса IDBProperties.

Распределенные транзакции выполняются в контексте нескольких провайдеров данных. Для выполнения таких транзакций потребители используют интерфейс TtransactionJoin, доступный только если провайдер данных поддерживает распределенные транзакции. В этом случае потребитель вызывает метод JoinTransaction для регистрации сессии в распределенной транзакции. После

присоединения к распределенной транзакции потребитель использует интерфейс ITransaction для завершения или отката транзакции.

1.4.9. Объект Error

В дополнение к кодам возврата и информации о состоянии, свидетельствующей об успехе или неуспехе вызова любого из методов OLE DB, OLE DB-провайдеры могут предоставлять расширенную информацию об ошибках с помощью объекта Error. Потребители данных могут использовать интерфейс ISupportErrorInfo для того, чтобы определить, может ли данный объект возвратить объект Error, и если да, то каковы эти интерфейсы.

Подробности об OLE DB можно найти на Web-сайте компании Microsoft:

http://www.microsoft.com/data/oledb

или в следующих группах новостей на http://msnews.microsoft.com:

microsoft.public.oledb

microsoft.public.oledb.sdk

Выше уже были перечислили ряд стандартных OLE DB-провайдеров, доступных в составе Microsoft Data Access Components и некоторых других продуктов Microsoft. Далее мы выясним, где можно найти OLE DB-провайдеры сторонних производителей и даже инструменты для создания собственных провайдеров.

1.5. Microsoft ActiveX Data Objects

ADO, как мы знаем, представляет собой высокоуровневый программный интерфейс для доступа к OLE DB-интерфейсам. Он позволяет манипулировать данными с помощью любых OLE DB-провайдеров, как входящих в состав Microsoft Data Access Components некоторых других продуктов Microsoft, так и произведенных сторонними производителями. ADO содержит набор объектов, используемых для соединения с источником данных, для чтения, добавления, удаления и модификации данных.

Объект ADO Connection применяется для установки связи с источником данных. Он представляет единственную сессию. Этот объект позволяет изменить параметры соединения с базой данных, а также начать или завершить транзакцию. Используя объект Connection, мы можем выполнять команды (например, SQL-запросы) с помощью метода Execute. Если команда возвращает набор данных, автоматически создается объект Recordset, который возвращается в результате выполнения этого метода.

Объект Error используется для получения сведений об ошибках, возникающих в процессе выполнения.

Объект Command представляет собой команду, которую можно выполнить в источнике данных. Команда может содержать SQL-предложение или вызов хранимой процедуры. В последнем случае для определения параметров процедуры может быть использована коллекция Parameters объекта Command.

Объект Recordset - это набор записей, полученных из источника данных, и может быть использован для добавления, удаления, изменения, просмотра записей. Данный объект может быть открыт непосредственно или создан с помощью объектов Connection или Command.

Объект Field - это колонка в наборе данных, представленных объектом Recordset. Он может быть использован для получения значений конкретного поля, его модификации, извлечения метаданных, таких как имя колонки и тип данных.

На рис. 4 изображена объектная модель ADO.

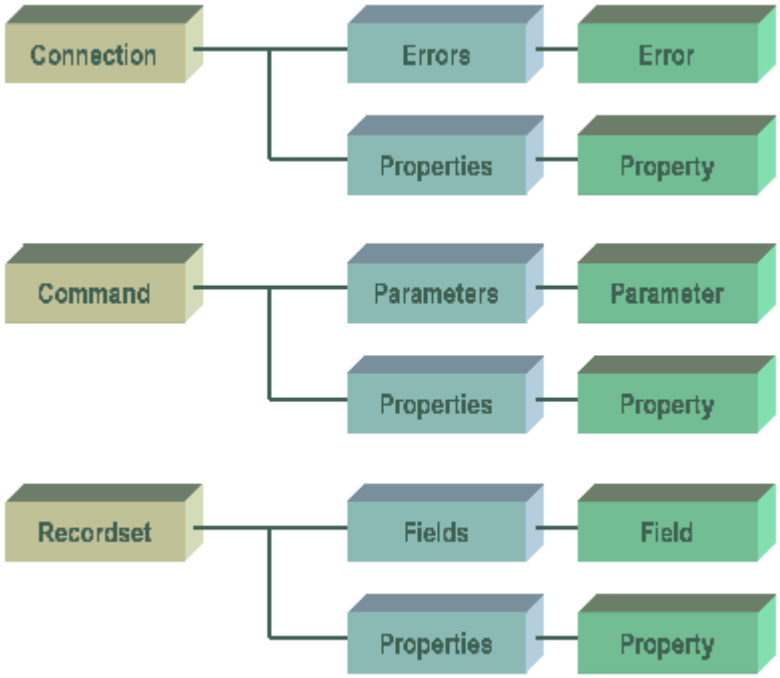


Рис. 4

Библиотека ADO 2.5, являющаяся составной частью операционной системы Windows 2000, содержит два новых объекта - Record и Stream.

Объект Record представляет одну запись внутри объекта Recordset и может быть использован для работы с гетерогенными и иерархическими данными.

Объект Stream представляет двоичные данные, связанные с объектом Record. Например, если объект Record представляет собой файл, то его объект Stream должен содержать данные внутри этого файла.

1.5.1. OLE DB и ADO

Ранее в этой статье мы уже рассмотрели некоторые основные объекты OLE DB. Теперь настало время показать, как объекты ADO используют OLD DB:

- объект ADO Connection использует объекты OLE DB DataSource и Session. Транзакции поддерживаются с помощью интерфейсов ITransaction и ITransactionLocal, метод Execute использует метод Execute интерфейса ICommand или метод OpenRowset интерфейса IOpenRowset. Большинство свойств доступно с помощью IDBProperties. Объект Error поддерживается с помощью интерфейса IErrorRecords;
- объект ADO Command использует объект OLE DB Command и интерфейс ICommand. Например, его метод Execute вызывает непосредственно одноименный метод объекта OLE DB Command, его свойство CommandText доступно с помощью методов GetCommandText и SetCommandText интерфейса ICommandText. Коллекция Parameters объекта ADO Command доступна с помощью интерфейса ICommandWithParameters;
- объект ADO Recordset использует объект OLE DB Rowset. Он применяет интерфейсы IRowset, IRowsetLocate и IRowsetInfo для реализации большинства методов, свойств и коллекций. Объект Field использует интерфейс IColumnsInfo.

Подробную информацию можно найти в Microsoft OLE DB Programmer's Reference, являющемся составной частью Platform SDK.

1.5.2. Расширения ADO

Начиная с версии 2.1, ADO содержит несколько расширений, на которых мы кратко остановимся ниже.

1.5.3. Где найти MDAC

Microsoft Data Access Components можно найти на Web-сайте компании Microsoft: http://www.microsoft.com/data, а также в составе Delphi 5 и C++Builder 5 Enterprise, Microsoft Office 2000, Internet Explorer 5.0. Рекомендуется периодически обновлять версию MDAC, получая ее с сайта компании Microsoft, - это гарантирует наличие последних версий всех компонентов MDAC.

ADO 2.5 является частью Windows 2000 и не требует отдельной установки. MDAC Software Development Kit, который может потребоваться разработчикам ADO-приложений, входит в состав Microsoft Platform SDK. Он может быть заказан на CD-ROM или получен на Web-сайте компании Microsoft.

1.6. Заключение

В данной статье мы рассмотрели основные части Microsoft Data Access Components (MDAC) - набора технологий, реализующих универсальный механизм доступа к данным Microsoft. Мы узнали, что универсальный механизм доступа к данным (Universal Data Access) представляет собой стратегию предоставления доступа к любому типу информации предприятия, включая реляционные и нереляционные данные.

Далее мы рассмотрели основные компоненты архитектуры универсального механизма доступа к данным Microsoft. Мы узнали, что:

- OLE DB - это низкоуровневый интерфейс для доступа к данным. ADO использует OLE DB, но можно использовать OLE DB и напрямую, минуя ADO;
- Microsoft ActiveX Data Objects (ADO) - это программный интерфейс для доступа к данным из приложений. С точки зрения программирования ADO и его расширения представляют собой упрощенный высокоуровневый объектно-ориентированный интерфейс к OLE DB.

Кроме того, мы рассмотрели основные объекты OLE DB и ADO и выяснили, как они взаимодействуют друг с другом.

Перечислив стандартные OLE DB-провайдеры, доступные в составе Microsoft Data Access Components и некоторых других продуктов Microsoft, мы обсудили, какие OLE DB-провайдеры сторонних производителей доступны в настоящее время и какие в данный момент имеются инструменты для производства собственных провайдеров.

Мы также рассмотрели расширения ADO, доступные начиная с версии 2.1:

- ADO Extensions for Data Definition and Security (ADOX) - это набор объектов, позволяющих манипулировать метаданными в базах данных и управлять объектами, отвечающими за безопасность данных;
- ADO Multi-Dimensional Extensions (ADOMD) - это набор объектов, позволяющих использовать в ADO-приложениях многомерные данные, управляемые OLAP-серверами;
- Jet and Replication Objects - это набор объектов для управления репликациями, специально предназначенных для использования совместно с Microsoft Jet OLE DB Provider.

Несколько слов хотелось бы сказать об универсальных механизмах доступа к данным в целом. Как мы видим, механизм, основанный на применении OLE DB и ADO, имеет все основания претендовать на роль ключевой технологии доступа к данным для Windows-приложений, ибо:

- он входит в состав не только многих популярных продуктов Microsoft, но и семейства операционных систем Windows 2000, что позволяет безболезненно

решить ряд проблем, связанных с поставкой таких приложений и их сопровождением;

- OLE DB-провайдеры выпускаются не только автором технологии OLE DB, но и многими сторонними производителями (вспомним, например, что ситуация с BDE-драйверами как раз обратная);
- доступ к нереляционным данным, предусмотренный данной технологией, является весьма привлекательной ее особенностью, и на данный момент другие универсальные механизмы доступа к данным не могут предложить ничего подобного;
- данная технология базируется на использовании COM-интерфейсов, поэтому она поддерживается всеми средствами разработки, позволяющими создание COM-клиентов с их помощью. Кроме того, многие средства разработки, даже не принадлежащие Microsoft, поддерживают ADO на уровне собственных классов и компонентов (например, Delphi 5 и C++Builder 5), несмотря на наличие других встроенных механизмов доступа к данным.

Отметим также, что расширяемая объектная модель ADO позволяет придавать последним реализациям этой технологии новые возможности, что, скорее всего, приведет к повышению популярности этой технологии.

Этой статьей мы закончили обзор универсальных механизмов доступа к данным. Следующая статья данного цикла будет посвящена языку SQL, широко применяемому ныне для манипуляции данными.

2. Работа с базами данных из ASP

Из ASP можно легко и просто работать с любыми базами данных. Это делается через две промежуточные технологии: ODBC и ADO.

Пример:

```
<%
Dim Conn, RS, strSQL, strOut
strOut = ""
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Data-Source-Name"
Set RS = Server.CreateObject("ADODB.Recordset")
strSQL = "SELECT * FROM AGENTS ORDER BY USER_ID"
RS.Open strSQL, Conn
RS.MoveFirst
strOut = strOut & "<P>Here is the data:"
strOut = strOut & "<TABLE BORDER=""1"">"
strOut = strOut & "<TR><TH>USER_ID</TH><TH>Name</TH></TR>"
Do While Not RS.EOF
   strOut = strOut & "<TR>"
```

```
        strOut = strOut & "<TD>" & RS.Fields("USER_ID") & "</TD>"
        strOut = strOut & "<TD>" & RS.Fields("NAME") & "</TD>"
        strOut = strOut & "</TR>"
        RS.MoveNext
    Loop
    strOut = strOut & "</TABLE>"
    strOut = strOut & "<HR>"
    strOut = strOut & "That's it!"
    RS.Close
    Set RS = Nothing
    Conn.Close
    Set Conn = Nothing
    Response.Write strOut
    %>
```

Если команда SQL не возвращает данных, recordset не нужен, надо пользоваться методом Conn.Execute(SQL_COMMAND).

Если Вы хотите вызывать хранимые процедуры сервера БД с параметрами, нужно воспользоваться объектом Command, который в свою очеред содержит объекты Parameter.

Использованные источники:

1. Алексей Федоров А, Елманова Н. Введение в базы данных: КомпьютерПресс 8'2000.

2. Григорий Грин. Введение в ASP: http://citforum.ru/internet/asp/asp_intro.shtml

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к Лабораторной работе 1
«Изучение протокола FTP»

1. Порядок выполнения лабораторной работы

Перед выполнением лабораторной работы студенты должны ознакомиться с методическими указаниями по ее выполнению и рекомендованной литературой.

Для получения зачета по лабораторной работе студент сдает преподавателю полностью оформленный отчет в электронном виде, а также демонстрирует работу на компьютере.

Во время защиты/сдачи лабораторной работы студент обязан продемонстрировать знания по теме работы, теоретическому материалу, методам выполнения работы, содержанию основных разделов разработанного отчета с демонстрацией результатов.

Для самопроверки при подготовке к выполнению работы студент должен ответить на контрольные вопросы, приведенные в конце лабораторной работы.

Общий зачет студент получает после выполнения и сдачи всех лабораторных и контрольных работ курса работы.

2. Задание для выполнения

1. Провести сеансы работы в активном и пассивном режимах с помощью стандартного FTP-клиента Windows со следующими FTP-серверами:

   ftp.rfbr.ru
   ftp.ncsa.uiuc.edu
   ftp.w3.org
   ftp.microsoft.com
   ftp.symantec.com

2. Сохранить протоколы обмена для обоих случаев, пояснить используемые команды FTP-клиента и ответы FTP-сервера.

3. Сеансы работы должны включать следующие действия:

   - получение списка файлов каталога,
   - смена каталога
   - скачивание файла.

4. Найти FTP-сервера, открытые для свободного доступа

5. Подготовить отчет о работе.

3. Содержание отчета

1. Титульный лист
2. Задание
3. Примеры FTP-транзакций с комментариями.
4. Список FTP-серверов, открытых для свободного доступа
5. Выводы по работе.

4. Теоретический материал

Протокол пересылки файлов FTP

FTP (RFC-959) обеспечивает файловый обмен между удаленными пользователями. Протокол FTP формировался многие годы. Первые реализации в МТИ относятся к 1971. (RFC 114 и 141). RFC 172 рассматривает протокол, ориентированный на пользователя и предназначенный для передачи файлов между ЭВМ. Позднее в документах RFC 265 и RFC 281 протокол был усовершенствован. Заметной переделке протокол подвергся в 1973, и окончательный вид он обрел в 1985 году. Таким образом, данный протокол является одним из старейших.

Здесь, так же, необходима идентификация, но многие депозитарии допускают анонимный вход (имя пользователя ANONYMOUS, RFC-1635), который не требует пароля или разрешает ввод вашего почтового адреса вместо него. Работа FTP на пользовательском уровне происходит в несколько этапов:

Идентификация (ввод имени-идентификатора и пароля)
Выбор каталога
Определение режима обмена (поблочный, поточный, ASCII или двоичный)
Выполнение команд обмена (get, mget, dir, mdel, mput, put и т.д.)
Завершение процедуры (quit или close)

FTP довольно необычная процедура, так как поддерживает две логические связи между компьютерами:

- одна связь служит для удаленного доступа и использует протокол Telnet для передачи команд.
- другая связь предназначена для обмена данными. Сервер производит операцию passive open для порта 21 и ждет соединения с клиентом. В качестве транспорта применяется протокол TCP. Клиент осуществляет операцию active open для порта 21. Канал остается активным до завершения процедуры FTP. TOS (тип IP-сервиса) соответствует минимуму задержки, так как этот канал используется для ручного ввода команд. Канал для передачи данных формируется каждый раз для пересылки файлов. Канал открывается перед началом пересылки и закрывается по коду end_of_file (конец файла). IP-тип сервиса (TOS) в этом случае ориентирован на максимальную пропускную способность.

Конечный пользователь взаимодействует с протокольным интерпретатором, в задачи которого входит управление обменом информацией между пользователем и файловой системой, как местной, так и удаленной. Схема взаимодействия различных частей Internet при работе FTP изображена на рисунке.

Рис. 1.2. Схема работы протокола FTP

Сначала по запросу клиента формируется канал управления, который в дальнейшем используется для передачи команд от клиента и откликов от сервера. Информационный канал формируется сервером по команде клиента, он не должен существовать постоянно на протяжении всей FTP -сессии и может формироваться и ликвидироваться по мере необходимости. Канал управления может быть закрыт только после завершения информационного обмена. После того как управляющий канал сформирован, клиент может посылать по нему команды. Сервер воспринимает, интерпретирует эти команды и передает отклики.

Пример FTP -сессии. Для этого выдадим команду:

FTP -d ns.primer.ru (флаг -d означает установку отладочного режима, при котором выдаются все сообщения и внутренние команды на экран терминала)
FTP Trying...Open
220- *** Welcome at FTP-Server ftp.ITEP.RU ***
220-
220 ns.primer.ru FTP server ready.
Userid for logging in on ns.itep.ru (SEMENOV)? Semenov
FTP command: USER Primerov
FTP response: 331 Password required for primerov.
331 Password required for primerov.
Password for logging in as primerov on ns.primer.ru? XXXXXXXX
PASS XXXXXXXX (ввод пароля не отображается на экране)
FTP response: 230 User primerov logged in.
230 User primerov logged in.
ftp:ns.itep.ru> help
…(список команд)
ftp:ns.itep.ru> quit

FTP command: QUIT
FTP response: 221 Goodbye.

Ниже приведен список базовых команд FTP. Следует разделять внутренний набор команд FTP, которыми обмениваются клиент и сервер по командному каналу, и набор команд, доступный пользователю. Служебные команды содержат три или четыре заглавные буквы. Эти наборы команд перекрываются лишь частично. Служебные команды унифицированы (они выделены в приведенном выше примере FTP -сессии жирным шрифтом, в помещенной ниже таблице эти команды представлены в ее верхней части), пользовательский же набор команд может варьироваться от реализации к реализации. Если выдать команду FTP без аргументов, система обычно откликается приглашением FTP > и вы можете выполнить некоторые из приведенных ниже команд (весь набор становится доступным только после идентификации).

Внутренние команды FTP:

Субкоманды FTP	Описание
ABOR	Прерывание исполнения предыдущей FTP - команды и связанного с ней обмена
ACCT<SP> <accountinformation>	Ввод идентификатора пользователя (ID)
ALLO <SP> <десятичное целое> [<SP> R <SP> <десятичное целое>]	Зарезервировать достаточно места (в байтах) для пересылки файла. Для файлов с постраничной структурой после символа R указывается число записей
APPE <SP> <проход>	Присовокупить передаваемые данные к файлу, указанному в параметре проход
CDUP	Переход в каталог-прародитель
CWD <SP> <проход>	Изменить рабочий каталог (CD)
DELE <SP> <проход>	Стереть файл (del)
HELP	Выдать справочную информацию о выполнимых командах
HELP [<SP> <строка>]	Выдать описание работы данной команды
LIST [<SP> <проход>]	Вывод списка файлов или каталогов (dir)
MKD <SP> <проход>	Создать каталог
MODE <SP> <код режима>	Режим обмена = поток, блоки или со сжатием
NLST [<SP> <проход>]	Переслать оглавление каталога от сервера к клиенту
NOOP	Пустая команда
PASS <SP> <пароль>	Слово-пропуск (пароль) пользователя, заполняется пользователем
PASV	Перевести сервер в режим прослушивания информационного порта на предмет установления соединения
PORT <SP> <порт ЭВМ>	IP-адрес и номер порта клиента
PWD	Выдать имя текущего каталога
QUIT	Уход из FTP
REIN	Завершение сессии и открытие новой

REST <SP> <маркер>	Возобновление обмена, начиная с места, указанного маркером
RETR <SP> <проход>	Переслать копию файла (get) другому адресату
RMD <SP> <проход>	Удалить каталог
RNFR <SP> <проход>	Начало процедуры переименования файла (Rename From)
RNTO <SP> <проход>	Указание нового имени файла при переименовании (Rename To)_
SITE <SP> <строка>	Используется сервером для реализации локально специфических команд
SMNT <SP> <проход>	Позволяет пользователю смонтировать нужную файловую систему
STAT	Выдать текущие значения параметров (STATUS)
STOR <SP> <проход>	Сервер должен запомнить полученные данные в виде файла
STOU	Аналог команды STOR, но записывает файл в текущий каталог и присваивает файлу уникальное имя
STRU <SP> <код структуры>	Структура файла = файл, запись или страница
SYST	Сервер сообщает тип системы
TYPE <SP> <код типа>	Специфицирует тип информации, часто для этой цели используются команды binary и ASCII
USER <SP> < [имя [пропуск]] >	Идентифицирует пользователя, запрашивается сервером
?	То же, что и HELP;
lcd	Изменить локальный каталог (на вашей ЭВМ)
!	Выйти временно из FTP и уйти в Shell (UNIX)
! команда	Исполнить команду Shell (UNIX)
close	Прервать связь с удаленным сервером, оставаясь в FTP
open [имя_ЭВМ]	Установить связь с указанным удаленным сервером
dir	Выдать содержимое удаленного каталога

<SP> — пробел; все команды завершаются последовательностью <CRLF> возврат каретки + перевод строки. В квадратных скобках записан опционный аргумент. Выполнение любой команды можно прервать с помощью Ctrl-C.

Пользовательские команды FTP (перечень не полон):

! [команда]	Исполняется команда интерпретатора shell вашей ЭВМ (UNIX). Если имя команды явно не введено, система переходит в интерактивный режим shell
$ имя-макро [аргументы]	Выполняется макро, имя которого введено, аргументы используются этим макро

account [пароль]	Позволяет ввести пароль, необходимый для доступа в удаленный сервер
append имя_местного_файла [имя_удаленного_файла]	Добавить местный файл к файлу на удаленном сервере
bye	Завершает FTP -сессию.
case	Переключает регистр символов, которыми записаны имена файлов на удаленной ЭВМ, в процессе выполнения команды MGET. Если case включен (по умолчанию выключен), все прописные буквы в именах файлов на удаленной ЭВМ меняются при переносе в вашу ЭВМ на строчные
close	Завершает FTP -сессию и возвращает систему в интерактивный командный режим. Все описанные ранее макро стираются
debug [debug-value]	Включает/выключает режим отладки. Значение debugvalue определяет отладочный уровень. Если отладка включена, FTP отображает на экране каждую команду, посылаемую удаленной ЭВМ. Эта информация помечается символом '-->'
dir [удаленный каталог] [местный файл]	Выдает на экран содержимое удаленного каталога. Если в качестве параметра указано имя местного файла, результат заносится в него. Если имя удаленного каталога не указано, команда выполняется для текущего каталога
disconnect	синоним close
hash	включает/выключает знак (#). Во включенном состоянии отмечается пересылка каждого блока, что позволяет визуально контролировать процесс обмена
macdef macro-name	Определяет макро. Последующие строки запоминаются в качестве текста макро с именем macro-name. Нулевая строка (двойное нажатие клавиши RETURN) завершает ввод текста макро. Можно ввести до 16 макро с суммарным объемом до 4096 символов
mdelete [имена_файлов_на удаленной_ЭВМ]	удаляет файлы на удаленной ЭВМ
open имя-ЭВМ [port]	устанавливает связь с указанным FTP -сервером (ЭВМ) через специфицированный порт

prompt	включает/выключает интерактивные запросы со стороны ЭВМ. Это бывает полезным при выполнении групповых команд MPUT, MGET или MDELETE и позволяет проводить соответствующие операции над файлами выборочно
proxy	ftp -команда выполняет FTP -команду на вторичной удаленной ЭВМ. Эта команда позволяет связать два удаленных FTP -сервера и осуществить пересылку файлов между ними. Первой proxy-командой должна быть команда open, необходимая для связи со вторичным сервером. Введите команду proxy ?, чтобы проверить выполнимость этих команд на данном сервере
quit	синоним bye
recv	удаленный_файл [местный_файл] синоним команды get
remotehelp [имя_команды]	Запрашивает справочную информацию от удаленного FTP -сервера. Если имя_команды задано, запрашивается информация о конкретной команде
send local-file [remote-file]	Синоним команды put
status	Отображает текущее состояние ftp

Обратите внимание, что большие и маленькие буквы в именах файлов различаются.

Анонимные FTP серверы позволяют вам войти в них под именем пользователя 'anonymous' или 'ftp', например,

ftp> login: anonymous
ftp> password: [ваш полный E-mail адрес]
ftp> cd <имя_каталога > (смена каталога)
ftp> binary (если текст, например, архивирован,
в противном случае команду выдавать не нужно)
ftp> get <имя_файла> (копирование файла)
ftp> quit (выход из процедуры)

Ввод адреса электронной почты не необходим, но является "правилом хорошего тона" при работе с FTP.

Вместо имени FTP сервера вы можете использовать его IP адрес, например 198.105.232.1 для того же ftp.microsoft.com.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к Лабораторной работе 2
«Процедуры Интернет: ping и tracert»

1. Порядок выполнения лабораторной работы

Перед выполнением лабораторной работы студенты должны ознакомиться с методическими указаниями по ее выполнению и рекомендованной литературой.

Для получения зачета по лабораторной работе студент сдает преподавателю полностью оформленный отчет в электронном виде, а также демонстрирует работу на компьютере.

Во время защиты/сдачи лабораторной работы студент обязан продемонстрировать знания по теме работы, теоретическому материалу, методам выполнения работы, содержанию основных разделов разработанного отчета с демонстрацией результатов.

Для самопроверки при подготовке к выполнению работы студент должен ответить на контрольные вопросы, приведенные в конце лабораторной работы.

Общий зачет студент получает после выполнения и сдачи всех лабораторных и контрольных работ курса работы.

2. Задание для выполнения

1. С помощью утилиты ipconfig определить IP адрес и физический адрес основного сетевого интерфейса компьютера, IP адрес шлюза, IP адреса DNS-серверов и используется ли DHCP. Результаты представить в виде таблицы.

2. Проверить состояние связи с любыми двумя узлами (работоспособными). Число отправляемых запросов должно составлять не менее 20. В качестве результата отразить для каждого из исследуемых узлов в виде таблицы:

   процент потерянных пакетов;
   среднее время приема-передачи;
   количество маршрутизаторов (с учетом шлюза) до опрашиваемого узла;
   IP адрес узла.
   класс сети, к которой принадлежит данный узел;
   имя узла, полученное по IP-адресу узла.

   В отчёте необходимо пояснить, как были определены значения.

Варианты заданий (любые два рабочих и один не рабочий)

www.informika.ru ; www.gpntb.ru ; www.rusmedserv.com ; www.keldysh.ru ; www.rfbr.ru ; www.ras.ru ; www.nsc.ru ; www.inauka.ru ; www.viniti.ru www.sostav.ru ; www.gramota.ru; www.ripn.net ; www.shpl.ru ; sai.msu.su www. symantec.com ; ncsa.uiuc.edu ; www.w3.org ; microsoft.com.

3. Произвести трассировку двух работоспособных узлов из пункта 2 задания к лабораторной работе. Результаты запротоколировать в таблице.

№ узла	Время прохождения пакета №1	Время прохождения пакета №2	Время прохождения пакета №3	среднее время прохождения пакета	DNS-имя маршрутизатора	IP-адрес маршру-тизатора

Если значения времени прохождения трёх пакетов отличаются более, чем на 10 мс, либо если есть потери пакетов, то для соответствующих узлов среднее время прохождения необходимо определять с помощью утилиты ping по 20 пакетам.

По результатам таблицы в отчете привести график изменения среднего времени прохождения пакета. В отчёте привести одну копию окна с результатами команды tracert.

Для каждого опрашиваемого узла определить участок сети между двумя соседними маршрутизаторами, который характеризуется наибольшей задержкой при пересылке пакетов.

Для найденных маршрутизаторов с помощью сервиса Whois определить название организации и контактные данные (тел., email). Полученную информацию необходимо указать в отчёте.

4. Подготовить отчет о работе.

3. Содержание отчета

1. Титульный лист
2. Задание
3. Результат работы утилиты ipconfig.
4. Таблицу с результатами исследованных узлов с указание их работоспособности (см. п.3 Задания).
5. Результаты работы службы Whois.
6. Выводы по работе.

4. Теоретический материал

Типы адресов: физический (MAC-адрес), сетевой (IP-адрес) и символьный (DNS-имя)

Каждый компьютер в сети TCP/IP имеет адреса трех уровней:

- Локальный адрес узла, определяемый технологией, с помощью которой построена отдельная сеть, в которую входит данный узел. Для узлов, входящих в локальные сети, это MAC-адрес сетевого адаптера или порта маршрутизатора,

например, 11-A0-17-3D-BC-01. Эти адреса назначаются производителями оборудования и являются уникальными адресами, так как управляются централизовано. Для всех существующих технологий локальных сетей MAC-адрес имеет формат 6 байтов: старшие 3 байта - идентификатор фирмы производителя, а младшие 3 байта назначаются уникальным образом самим производителем.

- IP-адрес, состоящий из 4 байт, например, 109.26.17.100. Этот адрес используется на сетевом уровне. Он назначается администратором во время конфигурирования компьютеров и маршрутизаторов. IP-адрес состоит из двух частей: номера сети и номера узла. Номер сети может быть выбран администратором произвольно, либо назначен по рекомендации специального подразделения Internet (Network Information Center, NIC), если сеть должна работать как составная часть Internet. Обычно провайдеры услуг Internet получают диапазоны адресов у подразделений NIC, а затем распределяют их между своими абонентами. Номер узла в протоколе IP назначается независимо от локального адреса узла. Деление IP-адреса на поле номера сети и номера узла - гибкое, и граница между этими полями может устанавливаться весьма произвольно. Узел может входить в несколько IP-сетей. В этом случае узел должен иметь несколько IP-адресов, по числу сетевых связей. Таким образом, IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.
- • Символьный идентификатор-имя, например, SERV1.IBM.COM. Этот адрес назначается администратором и состоит из нескольких частей, например, имени машины, имени организации, имени домена. Такой адрес, называемый также DNS-именем, используется на прикладном уровне, например, в протоколах FTP или telnet.

Основные классы IP-адресов

IP-адрес имеет длину 4 байта и обычно записывается в виде четырех чисел, представляющих значения каждого байта в десятичной форме, и разделенных точками, например:

128.10.2.30 - традиционная десятичная форма представления адреса,
10000000 00001010 00000010 00011110 - двоичная форма представления этого же адреса.

Далее показана структура IP-адреса в зависимости от класса сети.

Класс A
0 N сети N узла

Класс B
1 0

Класс C
1 1 0 N сети N узла

Класс D
1 1 1 0 адрес группы multicast

Класс E
1 1 1 1 0 зарезервирован

Адрес состоит из двух логических частей - номера сети и номера узла в сети. Какая часть адреса относится к номеру сети, а какая к номеру узла, определяется значениями первых битов адреса:

Если адрес начинается с 0, то сеть относят к классу A, и номер сети занимает один байт, остальные 3 байта интерпретируются как номер узла в сети. Сети класса A имеют номера в диапазоне от 1 до 126. (Номер 0 не используется, а номер 127 зарезервирован для специальных целей, о чем будет сказано ниже.) В сетях класса A количество узлов должно быть больше 2^{16} , но не превышать 2^{24}.

Если первые два бита адреса равны 10, то сеть относится к классу B и является сетью средних размеров с числом узлов 2^8 - 2^{16}. В сетях класса B под адрес сети и под адрес узла отводится по 16 бит, то есть по 2 байта.

Если адрес начинается с последовательности 110, то это сеть класса C с числом узлов не больше 2^8. Под адрес сети отводится 24 бита, а под адрес узла - 8 бит.

Если адрес начинается с последовательности 1110, то он является адресом класса D и обозначает особый, групповой адрес - multicast. Если в пакете в качестве адреса назначения указан адрес класса D, то такой пакет должны получить все узлы, которым присвоен данный адрес.

Если адрес начинается с последовательности 11110, то это адрес класса E, он зарезервирован для будущих применений.

В таблице приведены диапазоны номеров сетей, соответствующих каждому классу сетей.

Класс	**Наименьший адрес**	**Наибольший адрес**
A	0.1.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.1.0	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Отображение символьных адресов на IP-адреса: служба DNS

DNS (Domain Name System) - это распределенная база данных, поддерживающая иерархическую систему имен для идентификации узлов в сети Internet. Служба DNS предназначена для автоматического поиска IP-адреса по известному символьному имени узла. Спецификация DNS определяется стандартами RFC 1034 и 1035. DNS

требует статической конфигурации своих таблиц, отображающих имена компьютеров в IP-адрес.

Протокол DNS является служебным протоколом прикладного уровня. Этот протокол несимметричен - в нем определены DNS-серверы и DNS-клиенты. DNS-серверы хранят часть распределенной базы данных о соответствии символьных имен и IP-адресов. Эта база данных распределена по административным доменам сети Internet. Клиенты сервера DNS знают IP-адрес сервера DNS своего административного домена и по протоколу IP передают запрос, в котором сообщают известное символьное имя и просят вернуть соответствующий ему IP-адрес.

Если данные о запрошенном соответствии хранятся в базе данного DNS-сервера, то он сразу посылает ответ клиенту, если же нет - то он посылает запрос DNS-серверу другого домена, который может сам обработать запрос, либо передать его другому DNS-серверу. Все DNS-серверы соединены иерархически, в соответствии с иерархией доменов сети Internet. Клиент опрашивает эти серверы имен, пока не найдет нужные отображения. Этот процесс ускоряется из-за того, что серверы имен постоянно кэшируют информацию, предоставляемую по запросам. Клиентские компьютеры могут использовать в своей работе IP-адреса нескольких DNS-серверов, для повышения надежности своей работы.

База данных DNS имеет структуру дерева, называемого доменным пространством имен, в котором каждый домен (узел дерева) имеет имя и может содержать поддомены. Имя домена идентифицирует его положение в этой базе данных по отношению к родительскому домену, причем точки в имени отделяют части, соответствующие узлам домена.

Корень базы данных DNS управляется центром Internet Network Information Center. Домены верхнего уровня назначаются для каждой страны, а также на организационной основе. Имена этих доменов должны следовать международному стандарту ISO 3166. Для обозначения стран используются трехбуквенные и двухбуквенные аббревиатуры, а для различных типов организаций используются следующие аббревиатуры:

com - коммерческие организации (например, microsoft.com);
edu - образовательные (например, mit.edu);
gov - правительственные организации (например, nsf.gov);
org - некоммерческие организации (например, fidonet.org);
net - организации, поддерживающие сети (например, nsf.net).

Каждый домен DNS администрируется отдельной организацией, которая обычно разбивает свой домен на поддомены и передает функции администрирования этих поддоменов другим организациям. Каждый домен имеет уникальное имя, а каждый из поддоменов имеет уникальное имя внутри своего домена. Имя домена может содержать до 63 символов. Каждый хост в сети Internet однозначно определяется своим *полным доменным именем (fully qualified domain name, FQDN)*, которое включает имена всех доменов по направлению от хоста к корню.

4.1. Системные утилиты сетевой диагностики

4.1.1. Утилита ipconfig

Утилита ipconfig (IP configuration) предназначена для настройки протокола IP для операционной системы Windows. В данной лабораторной работе эта утилита будет использоваться только для получения информации о соединении по локальной сети. Для получения этой информации выполните «Пуск» → «Выполнить» → cmd и в командной строке введите:

```
ipconfig /all
```

4.1.2. Утилита ping

Утилита ping (Packet Internet Groper) является одним из главных средств, используемых для отладки сетей, и служит для принудительного вызова ответа конкретной машины. Она позволяет проверять работу программ TCP/IP на удаленных машинах, адреса устройств в локальной сети, адрес и маршрут для удаленного сетевого устройства. В выполнении команды ping участвуют система маршрутизации, схемы разрешения адресов и сетевые шлюзы. Это утилита низкого уровня, которая не требует наличия серверных процессов на проверяемой машине, поэтому успешный результат при прохождении запроса вовсе не означает, что выполняются какие-либо сервисные программы высокого уровня, а говорит о том, что сеть находится в рабочем состоянии, питание проверяемой машины включено, и машина не отказала ("не висит").

В Windows утилита ping имеется в комплекте поставки и представляет собой программу, запускаемую из командной строки.

Запросы утилиты ping передаются по протоколу ICMP (Internet Control Message Protocol). Получив такой запрос, программное обеспечение, реализующее протокол IP у адресата, посылает эхо-ответ. Если проверяемая машина в момент получения запроса была загружена более приоритетной работой (например, обработкой и перенаправлением большого объема трафика), то ответ будет отправлен не сразу, а как только закончится выполнение более приоритетной задачи. Поэтому следует учесть, что задержа, рассчитанная утилитой ping, вызвана не только пропускной способностью канала передачи данных до проверяемой машины, но и загруженностью этой машины.

Эхо-запросы посылаются заданное количество раз (ключ -n). По умолчанию передается четыре запроса, после чего выводятся статистические данные.

Обратите внимание: поскольку с утилиты ping начинается хакерская атака, некоторые серверы в целях безопасности могут не посылать эхо-ответы (например, www.microsoft.com). Не ждите напрасно, введите команду прерывания (CTRL+C).

Формат команды:

```
ping [-t][-a][-n][-l][-f][-i TTL][-v TOS]
 [-r][][имя машины][[-j списокУзлов]|[-k списокУзлов]][-w]
```

Параметры утилиты ping

Ключи	Функции
-t	Отправка пакетов на указанный узел до команды прерывания
-a	Определение имени узла по IP-адресу
-n	Число отправляемых запросов
-l	Размер буфера отправки
-f	Установка флага, запрещающего фрагментацию пакета
-i TTL	Задание времени жизни пакета (поле "Time To Live")

На практике большинство опций в формате команды можно опустить, тогда в командной строке может быть: ping имя узла (для зацикливания вывода информации о соединении используется опция –t; для вывода информации n-раз используется опция –n количество раз).

Пример:

ping –n 20 peak.mountin.net
Обмен пакетами с peak.mountin.net [207.227.119.2] по 32 байт:
Превышен интервал ожидания для запроса.
Ответ от 207.227.119.2: число байт=32 время=734мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=719мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=688мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=704мс TTL=231
Превышен интервал ожидания для запроса.
Ответ от 207.227.119.2: число байт=32 время=719мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=1015мс TTL=231
Превышен интервал ожидания для запроса.
Ответ от 207.227.119.2: число байт=32 время=703мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=688мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=782мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=688мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=688мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=688мс TTL=231
Превышен интервал ожидания для запроса.
Ответ от 207.227.119.2: число байт=32 время=687мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=735мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=672мс TTL=231
Ответ от 207.227.119.2: число байт=32 время=704мс TTL=231

Статистика Ping для 207.227.119.2:
Пакетов: отправлено = 20, получено = 16, потеряно = 4 (20% потерь),
Приблизительное время передачи и приема:
наименьшее = 672мс, наибольшее = 1015мс, среднее = 580мс

Пример определения имени узла по IP-адресу

```
ping –a 194.67.57.26
Обмен пакетами с mail.ru [194.67.57.26] по 32 байт: …
```

4.1.3. Утилита tracert

Утилита tracert позволяет выявлять последовательность маршрутизаторов, через которые проходит IP-пакет на пути к пункту своего назначения.

Формат команды:

`tracert имя_машины`

`имя_машины` может быть именем узла или IP-адресом машины.

Выходная информация представляет собой список машин, начиная с первого шлюза и заканчивая пунктом назначения.

Пример:

tracert peak.mountin.net

Трассировка маршрута к peak.mountin.net [207.227.119.2]

с максимальным числом прыжков 30:

№	Пакет 1	Пакет 2	Пакет 3	DNS-имя узла и (или) его IP-адрес
1	<10 мс	<10 мс	<10 мс	SLAVE [192.168.0.1]
2	<10 мс	<10 мс	<10 мс	gw.b10.tpu.edu.ru [195.208.164.2]
3	<10 мс	<10 мс	<10 мс	195.208.177.62
4	<10 мс	<10 мс	<10 мс	news.runnet.tomsk.ru [195.208.160.4]
5	<10 мс	<10 мс	16 ms	ra.cctpu.tomsk.su [195.208.161.34]
6	781 ms	563 ms	562 ms	spb-2-gw.runnet.ru [194.85.33.9]
7	547 ms	594 ms	578 ms	spb-gw.runnet.ru [194.85.36.30]
8	937 ms	563 ms	562 ms	20.201.atm0-201.ru-gw.run.net [193.232.80.105]
9	1125 ms	563 ms	547 ms	fi-gw.nordu.net [193.10.252.41]
10	906 ms	1016 ms	578 ms	s-gw.nordu.net [193.10.68.41]
11	844 ms	828 ms	610 ms	dk-gw2.nordu.net [193.10.68.38]
12	578 ms	610 ms	578 ms	sl-gw10-cop-9-0.sprintlink.net [80.77.65.25]
13	610 ms	968 ms	594 ms	sl-bb20-cop-8-0.sprintlink.net [80.77.64.37]
14	641 ms	672 ms	656 ms	sl-bb21-msq-10-0.sprintlink.net [144.232.19.29]
15	671 ms	704 ms	687 ms	sl-bb21-nyc-10-3.sprintlink.net [144.232.9.106]
16	985 ms	703 ms	765 ms	sl-bb22-nyc-14-0.sprintlink.net [144.232.7.102]
17	719 ms	734 ms	688 ms	144.232.18.206
18	891 ms	703 ms	734 ms	p1-0.nycmny1-nbr1.bbnplanet.net [4.24.8.161]
19	719 ms	985 ms	703 ms	so-6-0-0.chcgil2-br2.bbnplanet.net [4.24.4.17]
20	688 ms	687 ms	703 ms	so-7-0-0.chcgil2-br1.bbnplanet.net [4.24.5.217]
21	719 ms	703 ms	672 ms	p1-0.chcgil2-cr9.bbnplanet.net [4.24.8.110]
22	687 ms	719 ms	687 ms	p2-0.nchicago2-cr2.bbnplanet.net [4.0.5.242]
23	781 ms	703 ms	672 ms	p8-0-0.nchicago2-core0.bbnplanet.net [4.0.6.2]

№	Пакет 1	Пакет 2	Пакет 3	DNS-имя узла и (или) его IP-адрес
24	672 ms	703 ms	687 ms	fa0.wcnet.bbnplanet.net [207.112.240.102]
25	734 ms	687 ms	688 ms	core0-s1.rac.cyberlynk.net [209.100.155.22]
26	1188 ms	*	890 ms	peak.mountin.net [207.227.119.2]

Трассировка завершена.

Пакеты посылаются по три на каждый узел. Для каждого пакета на экране отображается величина интервала времени между отправкой пакета и получением ответа. Символ * означает, что ответ на данный пакет не был получен. Если узел не отвечает, то при превышении интервала ожидания ответа выдается сообщение «Превышен интервал ожидания для запроса». Интервал ожидания ответа может быть изменен с помощью опции –w команды tracert.

Команда tracert работает путем установки поля времени жизни (числа переходов) исходящего пакета таким образом, чтобы это время истекало до достижения пакетом пункта назначения. Когда время жизни истечет, текущий шлюз отправит сообщение об ошибке на машину-источник. Каждое приращение поля времени жизни позволяет пакету пройти на один маршрутизатор дальше.

Для вывода информации в файл используйте символ перенаправления потока вывода «>». Данный символ справедлив и для утилит ping и tracert.

Пример:
tracert 195.208.164.1 > tracert.txt

Отчет о трассировке маршрута до указанного узла будет помещен в файл tracert.txt.

4.1.4. Сервис Whois

При регистрации доменных имен второго уровня обязательным условием является предоставление верных сведений о владельце этого домена: для юридических лиц - название организации, для физических лиц - ФИО и паспортные данные. Также обязательным является предоставление контактной информации. Часть этой информации становится свободно доступной для любого пользователя сети Интернет через сервис Whois. Получить интересующую информацию о владельце домена можно через Whois-клиент, например, в Unix это консольная команда whois, в ОС Windows - это приложение SmartWhois. Но проще всего отправить запрос можно через веб-форму on-line сервиса Whois, например через форму на странице http://www.nic.ru/whois/.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к Лабораторной работе 3
«HTML – гипертекстовый язык разметки документа»

1. Порядок выполнения лабораторной работы

Перед выполнением лабораторной работы студенты должны ознакомиться с методическими указаниями по ее выполнению и рекомендованной литературой. Во время занятий каждый студент получает индивидуальный вариант задания.

Для получения зачета по лабораторной работе студент сдает преподавателю полностью оформленный отчет в электронном виде, а также демонстрирует работу на компьютере.

Во время защиты/сдачи лабораторной работы студент обязан продемонстрировать знания по теме работы, теоретическому материалу, методам выполнения работы, содержанию основных разделов разработанного отчета с демонстрацией результатов.

Для самопроверки при подготовке к выполнению работы студент должен ответить на контрольные вопросы, приведенные в конце лабораторной работы.

Общий зачет студент получает после выполнения и сдачи всех лабораторных и контрольных работ курса работы.

2. Задание для выполнения

1. Создание web-сайта согласно вариантам в Приложении 1. Вариант из Приложения 1 демонстрируем макет сайта. По аналогии с главной страницей, необходимо разработать 6-7 связанных между собой страниц с работающими гиперссылками на эти страниц.

2. Страницы сайта должна обязательно содержать следующие элементы:
   список;
   заголовки;
   таблицы;
   изображения;
   ссылки;
   текст, разбитый на параграфы, выровненные по-разному;
   шрифт различных цветов и размеров.

3. Верстку сайта можно выполнить с помощью
   вложенных таблиц
   или
   блоков.

3. Содержание отчета

1. Титульный лист
2. Задание.
3. Структура страницы в виде рисунка.
4. Текст HTML-документов.
5. Выводы по работе.

4. Теоретический материал

Разметка с помощью таблиц

Изначально в HTML не было предусмотрено возможности разметки страниц, поэтому для этих целей стали использовать таблицы.

Если посмотреть, большинство сайтов сделано именно с использованием таблиц для разметки.

Макетирование страниц таким способом позволяет создать поля страницы, разделить текст на колонки, окрасить отдельные области в определенный фоновый цвет и даже использовать локальные фоновые изображения.

Для представления информации в виде колонок текст и изображения размещают внутри ячеек таблицы. Внутрь ячеек можно вкладывать дополнительные таблицы.

Наиболее часто используемая структура приведена ниже (см. рис.):

Пример разбивки страницы

Сейчас самое распространенное разрешение — более 1024x768, это означает, что ширину основной таблицы нужно задавать несколько меньше, чтобы она была полностью видна на экране.

Размеры таблицы можно задать в процентном отношении к окну браузера, например 80% (если задать 100%, то часть таблицы не будет видна, и не будет полей).

Для ускорения загрузки страницу делают из трех, одинаковых по ширине, таблиц - верхнюю, основную и нижнюю. Также это помогает поддерживать общий стиль всего сайта, когда верхняя часть и нижняя повторяется на всех страницах.

Чтобы увидеть скрытую табличную структуру можно посмотреть код щелкнув на странице правой кнопкой и выбрав пункт «Кодировка» или аналогичный.

Недостатки использования таблиц:

- "Медленная загрузка". Пока вся таблица не загрузится, информация на экране не появится.
- Излишний код. Приходится создавать много ячеек и строк, которые в общем-то, не нужны.
- Отсутствие возможности с точностью до пикселов расположить элемент на экране.
- Отсутствие возможности "надвинуть" один элемент на другой.
- Отсутствие возможности управлять отображением элементов (при переполнении, таблица просто увеличивается в размерах).

Разметка с помощью блоков

Блочная верстка (с помощью тэга div) лишена перечисленных выше недостатков и очень популярна сегодня.

Допускается использовать возможность обтекания блоков (свойство float), различные способы отображения блоков (свойства display), управлять порядком наложения слоев с помощью свойства z-index.

Тэг - служит для группирования элементов в блок. К сгруппированным элементам можно применить стили.

Выделение блока бордюром

Границу можно легко разместить вокруг заголовка, списка, абзаца или их группы, поместив их в элемент div.

Прописываем стили для этого блока

Это можно использовать с разметкой следующим образом:

Пример:

```
<div style="border-color:#ff00ff; border-style:dotted; ">
Содержимое этого элемента div будет заключено в прерывистую рамку.
</div>
```

Позиционирование и задание размеров блоков

<table>
<tr><td colspan="4"><div style="background-color: #aaaaff">
Содержимое этого элемента DIV будет позиционироваться здесь.
</div></td></tr>
<tr><td><div style="float: left; width: 200; height: 200; background-color: #00ffff">
Содержимое этого элемента div будет позиционироваться здесь.
</div></td><td><div style="width: 200; height: 200; position: relative; left: 20; background-color: #0ffd02">
Содержимое этого элемента div будет позиционироваться здесь.
</div></td><td></td><td><div style="float: right; width: 200; height: 200; background-color: #ffff00">
Содержимое этого элемента div будет позиционироваться здесь.
</div></td></tr>
<tr><td colspan="4"><div style="background-color: #fcc403">
Содержимое этого элемента div будет позиционироваться здесь.
</div></td></tr>
</table>

5. Литература

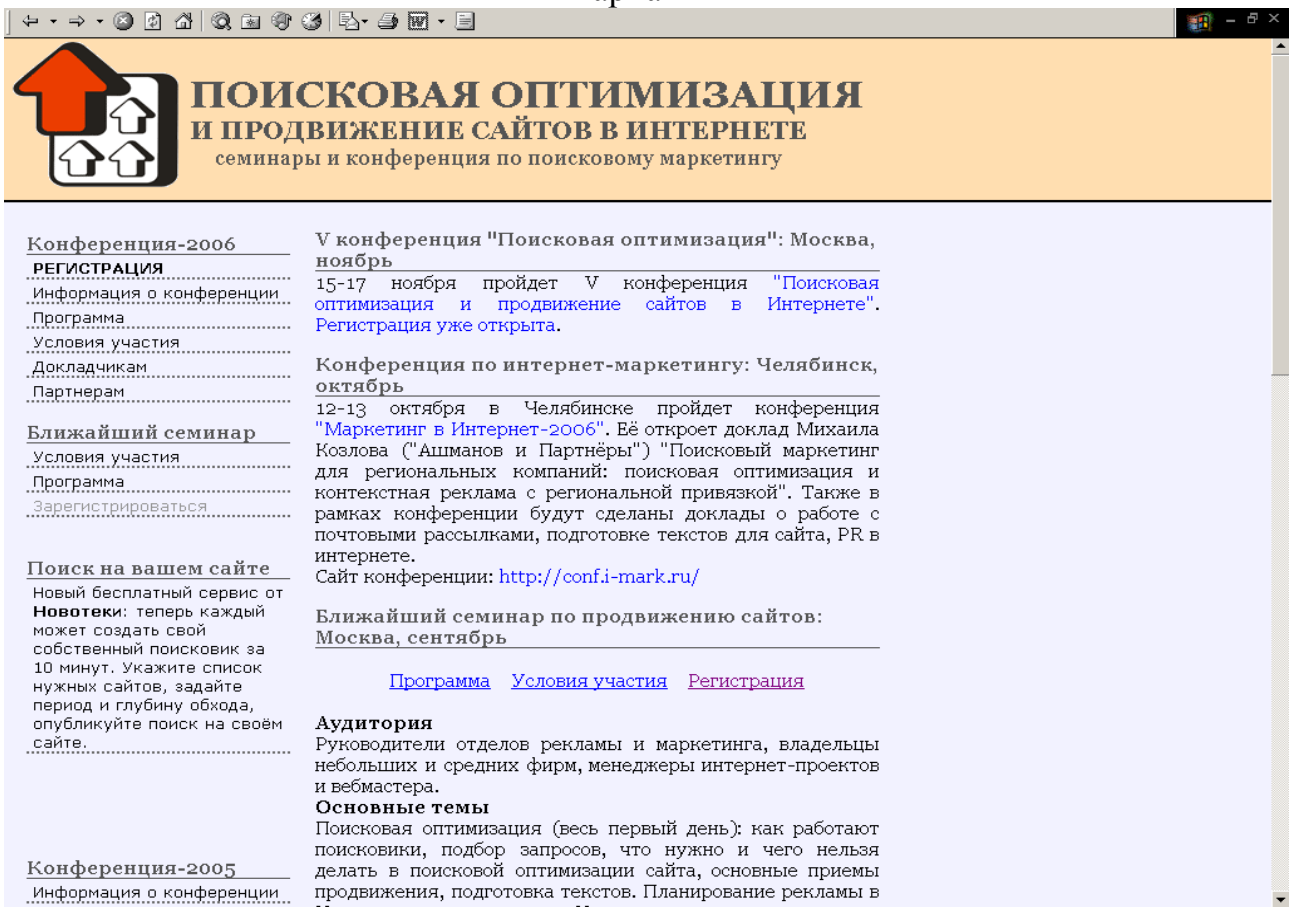
1. .Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Учебник по HTML (оригинальная версия на английском от World Wide Web Consortium'a (W3C)) http://www.w3schools.com/html/default.asp. С работающими примерами «Try it yourself».

3. HTML 4.01 Specification (english) http://www.w3.org/TR/1999/REC-html401-19991224. Спецификация HTML 4.01. Переводчик Пирамидин А. http://www.intuit.ru/department/internet/html/1/

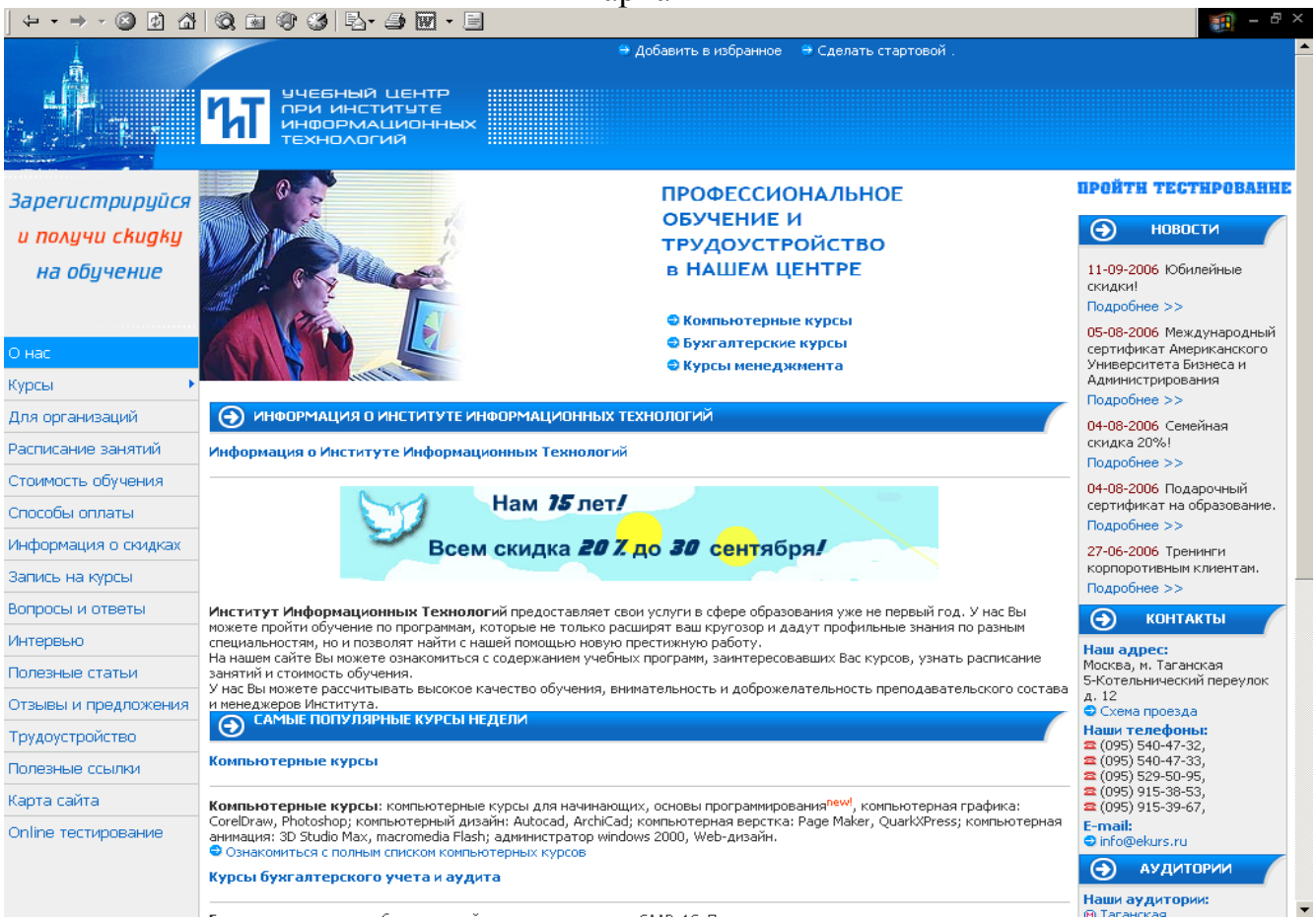
Приложение 1

Вариант 1



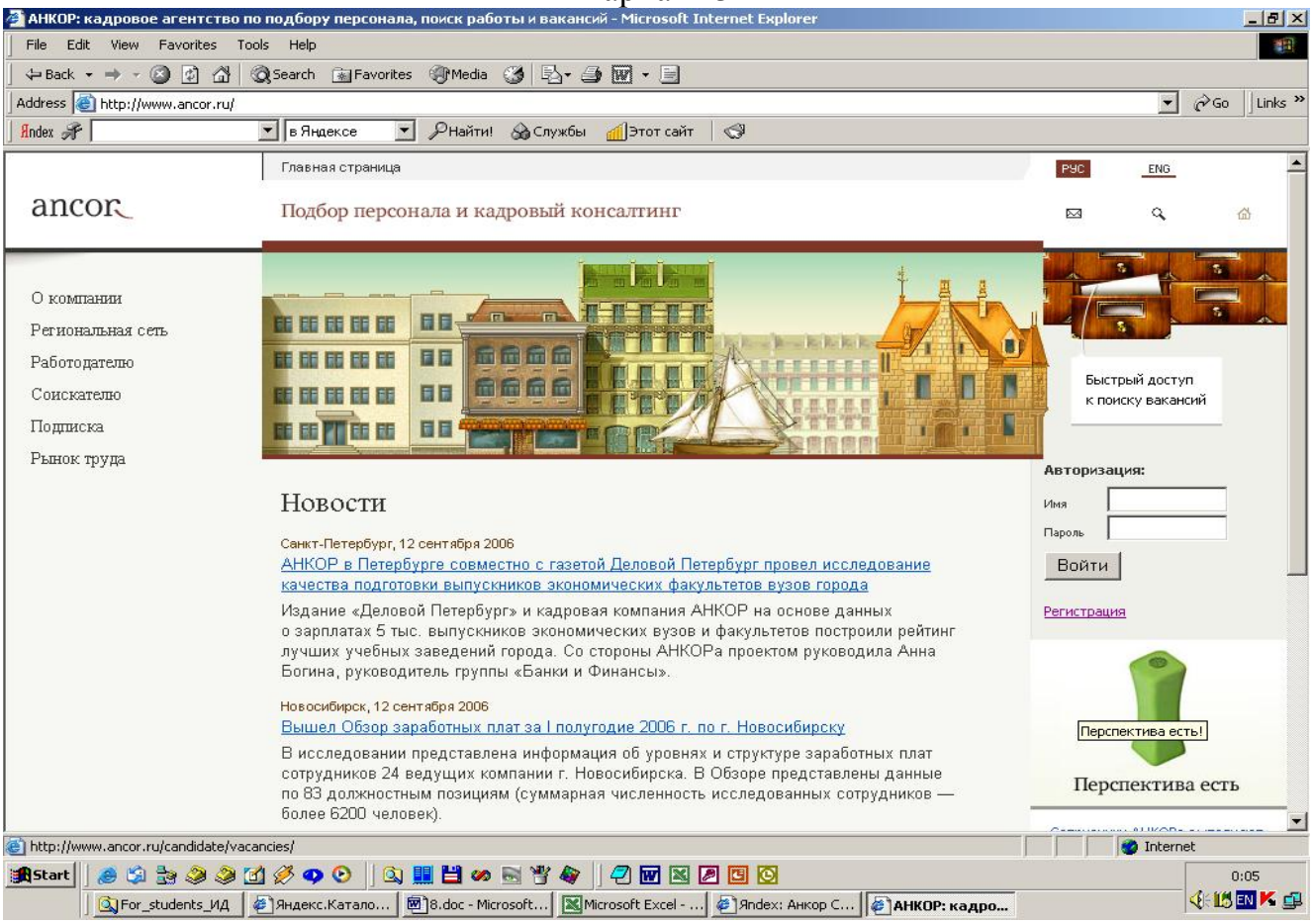
Главная страница

Вариант 2



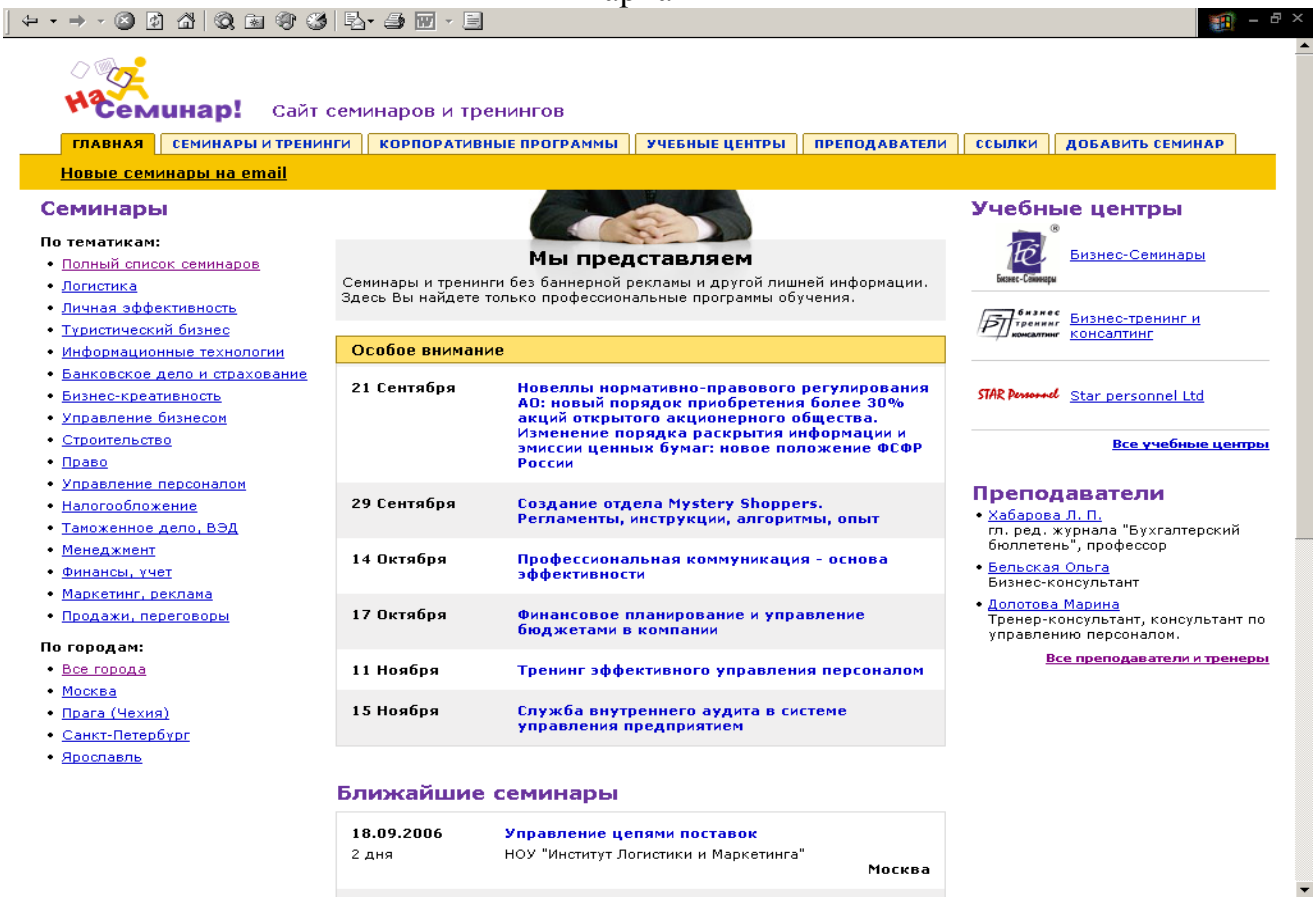
Главная страница

Вариант 3



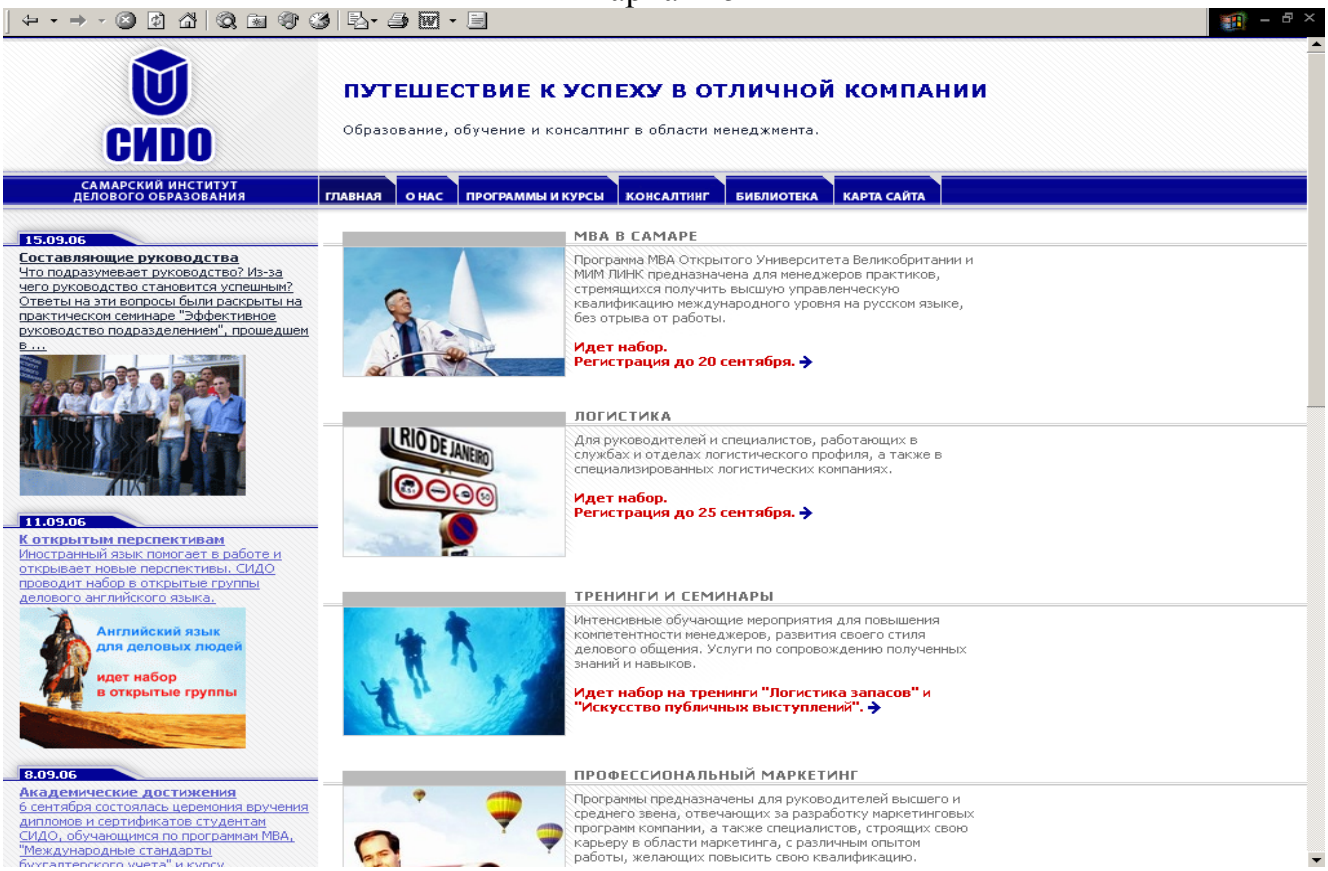
Главная страница

Вариант 4



Главная страница

Вариант 5



Главная страница

Вариант 6

Главная страница

Вариант 7

Главная страница

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к Лабораторной работе 4
«CSS – каскадные таблицы стилей»

1. Порядок выполнения лабораторной работы

Перед выполнением лабораторной работы студенты должны ознакомиться с методическими указаниями по ее выполнению и рекомендованной литературой. Во время занятий каждый студент продолжает работу со своим и индивидуальным вариантом задания.

Для получения зачета по лабораторной работе студент сдает преподавателю полностью оформленный отчет в электронном виде, а также демонстрирует работу на компьютере.

Во время защиты/сдачи лабораторной работы студент обязан продемонстрировать знания по теме работы, теоретическому материалу, методам выполнения работы, содержанию основных разделов разработанного отчета с демонстрацией результатов.

Для самопроверки при подготовке к выполнению работы студент должен ответить на контрольные вопросы, приведенные в конце лабораторной работы.

Общий зачет студент получает после выполнения и сдачи всех лабораторных и контрольных работ курса работы.

2. Задание для выполнения

1. Создайте каталог /css/

2. Создайте в нем два файла
   main.css (основную таблицу стилей для сайта) и
   print.css (таблицу стилей для вывода html-страницы на печать).

3. Подключите его ко всем страницам сайта с помощью тега <link>.

4. В таблице стилей в соответствии с вариантом, задайте по умолчанию следующие параметры для всех страниц (переопределив, тег <body>):
   - цвет фона (в соответствии с вариантом)
   - размер шрифта
   - цвет шрифта
   - семейство шрифта (например, Arial)

5. В соответствии с вариантом задайте по умолчанию следующие параметры для всех абзацев (переопределив, тег <p> :
   - выравнивание абзаца
   - отступ красной строки
   - отступ после абзаца.

6. Задайте по умолчанию следующие свойства ссылок для всех страниц:
   - цвет и оформление ссылки
   - цвет и оформление посещенной ссылки
   - цвет и оформление активной ссылки
   - цвет и оформление ссылки, в момент нахождения курсора мыши над ней

7. В комментариях к таблице стилей поясните эти параметры.

3. Содержание отчета

1. Титульный лист
2. Задание.
3. Структура страницы в виде рисунка.
4. Текст HTML-документов.
5. Выводы по работе.

4. Теоретический материал

4.1. Для чего нужны таблицы стилей?

Вы создавали предыдущие страницы, так как их создавали раньше до появления каскадных таблиц стилей или CSS (Cascading Style Sheets).

Основные проблемы, с которыми сталкивались разработчики сайтов до появления CSS:

- Проблема позиционирования (разметка страниц). Хотя для этого стали использовать таблицы (они небыли для этого предназначены изначально), они не решили всех проблем. Невозможно осуществить следующее:
  - Нельзя задать фиксированные размеры какого-нибудь блока. Например, текстовый блок будет разъезжаться при переполнении (даже если он в таблице).
  - Нельзя задать фиксированные координаты положения блока на странице.
  - Нельзя наложить один блок на другой. Например, надвинуть картинку поверх таблицы или другой картинки.
- Вторая проблема заключалась в том, что приходилось каждый раз задавать на страницах размер и цвет шрифта, свойства ячеек таблицы и т.д. Это сильно увеличивало размеры страницы, а значит, она медленнее загружается. В случае если вам захотелось изменить цвет или размер шрифта, вам пришлось бы редактировать все страницы.
- Третья проблема заключалась в том, что все браузеры имеют индивидуальные настройки. Например, в браузере можно увеличить шрифт, что приведет к искажению всей страницы.

С помощью CSS эти проблемы можно решить.

4.2. Способы применения CSS

Существует три способа применения таблиц стилей:

Встроенные таблицы стилей (Inline Style Sheets) - при помощи специального атрибута помещаются прямо в HTML теги.

Пример HTML:
 Пример

Пример CSS:
 Пример

Как можно заметить, код встроенных стилей получается больше чем HTML. Поэтому их следует использовать, только если необходимо задать определенному элементу свой индивидуальный стиль.

При помощи дополнительного атрибута style мы можем определить нужные нам стилевые параметры в любом теге. Это самый легкий способ, и действует он в пределах лишь одного тега. Но представьте, насколько вырастет размер файла, и насколько неудобно будет его исправлять, если мы будем указывать стиль у каждого тега.

Внутренние таблицы стилей - определяют стиль элементов во всем документе. Для этого используется тег <style type="text/css">. Он размещается в заголовке документа <head> .

Пример:
<html>
<head>
<STYLE type="text/css">
h1 { color:red; font-style:italic; font-size:32px} - переопределение стандартного тега.
.blue { color:blue} - определение нового класса
</STYLE>
</head>
<body>
Теперь эти стили можно применять в любом месте html-кода. Для этого используются следующие конструкции:
<h1> Этот заголовок написан крупным красным курсивом </h1>
Вот это слово - синие.
</body>
</html>

В данном примере все элементы h1 будут написаны крупным красным курсивом, все элементы с указанным классом blue будут синими.

Внешние таблицы стилей - могут быть использованы для нескольких документов сразу и хранятся во внешнем файле.

Пример внешнего файла:
Файл main.css

```
body {background:black; font-size:9pt; color:red; font-family:Arial Black}
.base{color:blue; font-style:italic}
h1 {color:white}
#bold {font-weight:bold}
```

В самих же HTML документах делается ссылка на этот файл при помощи тега <link>. Выглядит это так:

```
<link rel="stylesheet" type="text/css" href="путь до файла">
```

Пример:
Файл Index.html

```
<html>
<head>
<LINK rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
Содержание Документа
</body>
</html>
```

Это самый удобный способ, и для основной таблицы стилей рекомендуется пользоваться именно им.

4.3. Приоритет стилей

Стили подключенные разным способом имеют разный приоритет:

В общем, мы можем сказать, что все стили образуют новую "виртуальную" таблицу стилей по следующим правилам, из которых номер четыре имеет наивысший приоритет:

1. Используемые браузером по умолчанию
2. Внешние таблицы стилей
3. Внутренние таблицы стилей (в разделе head)
4. Встроенные стили (внутри HTML-элемента)

Если один HTML документ имеет несколько стилей присоединенных разными способами и в них заданы разные свойства оформления для одного и того же элемента, то в итоге элемент будет оформлен согласно содержимому стиля с более высоким приоритетом.

Пример

```
/* Внутренние стили: */
h1 {color:red}
/* Встроенные стили: */
h1 {color:green}
/* В результате заголовок будет зеленого цвета так как встроенные стили имеют более
высокий приоритет */
```

4.4. Наследование

Некоторые значения наследуются дочерними элементами (тегами).

Предположим, что имеется элемент h1, в котором расположен элемент выделения em:

<h1>Данный заголовок очень важен!</h1>

Если для элемента em не задан цвет, то слово "очень" унаследует цвет родительского элемента. Таким образом, если для h1 определен синий цвет, то и элемент em будет представлен, синим цветом.

Определение свойств стиля, используемого в документе по умолчанию, можно осуществить в корневом элементе дерева данного документа. Например, в языке HTML эту возможность можно реализовать с помощью элементов html или body.

Пусть задана следующая таблица стиля:

```
body { font-size: 10pt }
h1 { font-size: 120% }
```

и фрагмент документа:

```
<body>
<h1>Некоторый <em>крупный</em> заголовок</h1>
</body>
```

Свойство “font-size” элемента h1 будет иметь вычисленное значение “12pt” (120% от 10pt, являющегося значением свойства родительского элемента). Так как вычисляемое значение свойства “font-size” является наследуемым, то элемент em также будет иметь вычисленное значение “12pt”.

4.5. Синтаксис CSS

ТЕГ:псевдокласс.класс { характеристика1: значение1; характеристика2: значение2}

Сначала указывается имя класса - оно может быть произвольным, но желательно все-таки давать осмысленное название. Далее, в фигурных скобках {} перечисляются все необходимые параметры для данного класса. Параметры отделяются друг от друга точкой с запятой. Вот еще один пример, в котором используется более длинное описание:

```
.small { font-size: 9pt; color: #eeeeee; text-align: center; }
```

Универсальный класс - может быть применен к любому тегу, имя класса начинается с точки.

Пример:
Накладываем стиль на этот текст
<td class=small>Накладываем стиль на этот текст</td>

Теговые классы - может быть применен к конкретному тегу, имя класса начинается с указания тега, после точки записывается имя класса.

Пример:
p.small { font-size: 9pt; }

Класс, определенный таким образом, сработает только в том теге, для которого он предназначен, а для всех остальных будет проигнорирован:

Этот текст будет выведен стилем small
<td class=small>А этот останется неизменным</td>

Можно определять параметры не только для одного тега, но и сразу для нескольких. Для этого в определении стиля достаточно перечислить их через запятую:

P, TD, LI { font-size: 9pt; color:green;}

В случае переопределения существующих тегов, в описании стиля можно указывать не все параметры, а лишь те из них, которые мы хотим изменить. Все остальные параметры примут значения по умолчанию, которые для разных тегов различны.

Псевдоклассы

В CSS есть такое понятие как псевдокласс. В отличие от обычного класса, действие псевдокласса распространяется не на весь текст, к которому применен данный стиль, а лишь на его часть и возможно лишь в определенном состоянии. Чтобы было понятнее, давайте разберем эффект, при котором ссылки подчеркиваются лишь при наведении на них курсора. Эффект достаточно распространенный, и его можно наблюдать в том числе и на этом сайте. Вот фрагмент таблицы стилей, который позволяет достигать вышеописанного эффекта:

a { text-decoration: none; }
a:hover { text-decoration: underline; }

Верхняя строчка - это переопределение стандартного тега <a>, которое запрещает подчеркивать ссылки, а вот нижняя - это определение стиля для псевдокласса hover, который описывает стиль ссылки в момент, когда курсор находится над ней.

А вот и другой пример псевдокласса - определение буквицы вначале абзаца:

p:first-letter { font-size: 200%; font-weight: bold; }

Заметьте, что и в том, и в другом случае действие стиля распространяется либо на определенное состояние (пользователь собирается щелкнуть по ссылке), либо на фрагмент текста (изменяется только первая буква абзаца). В этом и заключается смысл псевдоклассов.

Комментарии

Как и в любом достаточно сложном языке, при создании таблицы стилей можно пользоваться комментариями.

Пример:
/* Этот текст является комментарием */

Для небольших сайтов эта возможность вряд ли пригодится, а вот при создании сложных, многоуровневых таблиц стилей комментарии могут пригодиться. Кстати, здесь будет уместно привести золотое правило - чем понятнее названа переменная (в данном случае имя класса), тем меньше комментариев необходимо.

4.6. Некоторые параметры CSS

Некоторые параметры шрифта
font-weight: [bold | normal | number] - жирность шрифта
font-style: [normal | italic | oblique] - наклон шрифта
font-size: number - размер шрифта
font-family: name - гарнитура шрифта
color: number - цвет шрифта
background-color: number - цвет подложки
background: url - текстурная подложка

Некоторые параметры абзаца
text-align: [left | right | center | justify] - выравнивание
text-indent: number - отступ красной строки
line-height: number - интерлиньяж
letter-spacing: number – трекинг

padding-left: number - отступ от текста слева
padding-right: number - отступ от текста справа
padding-top: number - отступ от текста сверху
padding-bottom: number - отступ от текста снизу

margin-left: number - отступ от границы слева
margin-right: number - отступ от границы справа
margin-top: number - отступ от границы сверху
margin-bottom: number - отступ от границы снизу

Единицы измерения в CSS

В свойствах, которым требуется указание размеров, можно использовать несколько способов для их задания:

- относительный размер в процентах (%)

- относительный размер при помощи словесного описания (larger, smaller, xx-small, x-small, small, medium, large, x-large, xx-large)
- абсолютный размер в типографских единицах - размер может задаваться в пунктах (pt), пиках (pc), средней шириной буквы "m" (em), средней шириной буквы "x" (ex)
- абсолютный размер в стандартных единицах длины - размер может задаваться в сантиметрах (cm), миллиметрах (mm), дюймах (in)
- абсолютный в пикселях (px)

4.7. Пример использования CSS

Назначим для всех состояний ссылок цвета и сделаем так, чтобы ссылки во всех состояниях не имели подчеркивания, а в момент нахождения курсора над ссылкой появлялось подчеркивание.

```
A:link {COLOR: #0000CC; text-decoration: none}
A:visited {COLOR: black; text-decoration: none}
A:active {COLOR: red; text-decoration: none}
A:hover {COLOR: #6666FF; text-decoration: underline}
```

Пояснения:
- ссылка (link) - синий цвет (#0000CC);
- посещенная ссылка (visited) - черный цвет (black);
- активная ссылка (active) - красный цвет (red);
- ссылка, в момент нахождения курсора мыши над ней (hover) - светло-синий (#6666FF);

Используя CSS, мы может отменить или переназначить эффект для ссылок. Для этого используем свойство text-decoration, которое может иметь следующие значения:

none - без оформления
underline - подчеркивание
overline - черта сверху
line-through - перечеркивание
blink - мигание

5. Литература

1. .Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Учебник по CSS (перевода на русский нет) http://www.w3schools.com/css/default.asp. С работающими примерами «Try it yourself»

3. Cascading Style Sheets, level 2. CSS2 Specification http://www.w3.org/TR/1998/REC-CSS2-19980512. Спецификация CSS2. Перевожчик Пирамидин А. http://www.intuit.ru/department/internet/css2/

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к Лабораторной работе 5
«Создание интерактивных документов»

1. Порядок выполнения лабораторной работы

Перед выполнением лабораторной работы студенты должны ознакомиться с методическими указаниями по ее выполнению и рекомендованной литературой. В приложении 1 приведены формы, корректность ввода данных в которые надо будет проверить. Варианты соответствуют полученным ранее вариантам.

Для получения зачета по лабораторной работе студент сдает преподавателю полностью оформленный отчет в электронном виде, а также демонстрирует работу на компьютере.

Во время защиты/сдачи лабораторной работы студент обязан продемонстрировать знания по теме работы, теоретическому материалу, методам выполнения работы, содержанию основных разделов разработанного отчета с демонстрацией результатов.

Для самопроверки при подготовке к выполнению работы студент должен ответить на контрольные вопросы, приведенные в конце лабораторной работы.

Общий зачет студент получает после выполнения и сдачи всех лабораторных и контрольных работ курса работы.

2. Задание для выполнения

1. Добавить к сайту страницу с формой (см. Приложение 1).

2. Написать сценарии проверки правильности вода данных в различные поля формы:
   ФИО – должно содержать только буквы и не быть короче 2-х символов;
   email – должен содержать слева направо
   (англ.буквы, символы) -- @ -- (англ.буквы, символы) -- (англ.буквы, от 4-х);.
   индекс – 6 цифр;
   адрес – буквы и цифры.
   Остальные поля в зависимости от варианта.

3. Проверку корректности ввода привязать к
   событию «потеря фокуса» проверяемым элементом
   или
   событию «отправка формы».

4. Пользователю вывести сообщение какие поля заполнены не верно.

3. Содержание отчета

1. Титульный лист
2. Задание.
3. Структура страницы в виде рисунка.
4. Текст HTML-документов.
5. Выводы по работе.

4. Теоретический материал

Объектная модель браузера

Объектная модель браузера содержит 12 объектов:

1. Document - предоставляющий возможность доступа к компонентам документа HTML.
2. Event - предоставляющий возможность доступа к свойствам событий, когда последние происходят.
3. History - предоставляет информацию об адресах, которые клиент посетил.
4. Location - предоставляет информацию об адресе текущего документа.
5. Navigator - позволяет обращаться к свойствам браузера.
6. Screen - предоставляет информацию о мониторе и системе вывода, информации клиента.
7. Window - предоставляет свойства, методы и события, связанные с окном браузера.

которые позволяют обращаться к части браузера или части страницы с помощью языка JavaScript.

Кроме объектов в объектной модели вводятся следующие понятия:

Методы - способы работы с объектами. Например: закрыть окно. По сути это функция, ассоциированная с объектом.

object.methodname

События - объект сообщает нам, что нечто произошло. Например: элемент становиться активным.

Свойства - свойства объекта. Например: имя и размеры окна.

Окна диалога. Объект Window

Открытие окна

Синтаксис:

window.open ("URL или URI","имя окна","свойства окна")

Следующий оператор создаёт окно, которое отображает содержимое страницы http://www.kstu.ru:

window.open("http://www.kstu.ru")

При создании окна вы можете также предоставить имя, в данном случае - kstuWindow, для обращения к окну как к цели/target при отправке формы или при переходе по гиперссылке.

window.open("http://www.kstu.ru", "kstuWindow")

Имя окна не требуется при создании окна. Но окно обязано иметь имя, если вы хотите обратиться к нему из другого окна.

При открытии окна вы можете специфицировать атрибуты, такие как высота/height и ширина/width, панель утилит/toolbar, адресная строка/location field или полосы прокрутки/scrollbars. Следующий оператор создаёт окно без панели утилит, но с полосами прокрутки:

window.open ("http://www.kstu.ru","wwwWindow","toolbar=no,scrollbars=yes")

Некоторые свойства окна:

directories - Если yes, создаются стандартные кнопки директорий браузера, такие как What's New и What's Cool.

height - Специфицирует высоту окна в пикселах.

innerHeight - Специфицирует высоту области содержимого окна в пикселах. Это свойство заменило height, которое оставлено для обеспечения обратной совместимости.

innerWidth - Специфицирует ширину области содержимого окна в пикселах. Это свойство заменило width, которое оставлено для обеспечения обратной совместимости.

location - Если yes, создаёт поле ввода Location.

menubar - Если yes, создаёт строку меню в верхней части окна.

outerHeight - Специфицирует размер по вертикали в пикселах внешней границы окна.

resizable - Если yes, даёт пользователю возможность изменять размеры окна.

screenX - Специфицирует расстояние, на котором новое окно помещается от левого края экрана.

screenY - Специфицирует расстояние, на котором новое окно помещается от верха экрана.

scrollbars - Если yes, создаются вертикальная и горизонтальная полосы прокрутки, если документ становится больше размеров окна.

status - Если yes, создаётся статусная строка внизу окна.

titlebar - Если yes, создаётся окно со строкой заголовка.

toolbar - Если yes, создаётся стандартная панель браузера с кнопками, такими как Back и Forward.

width - Специфицирует ширину окна в пикселах.

Для того чтобы функция отрабатывалась при нажатии мышкой на элементе документа, будем использовать событие **onClick** объекта Document.

Пример:

Откроем ссылку в новом окне

<a href=""

onClick="window.open('http://www.kstu.ru','kstuWindow','left=300,top=300,width=200,hight=400,toolbar=no,menubar=no,location=no,directories=no')">

Открыть новое окно

Исполнение: Открыть новое окно

Тоже самое можно сделать с помощью кнопки:

<input type="button" value="Открыть новое окно"

onClick="window.open('http://www.kstu.ru','kstuWindow','left=300,top=300,width=200,hight=400,toolbar=no,menubar=no,location=no,directories=no')">

Исполнение:

Закрытие окна

Следующий оператор закрывает текущее окно:

```
window.close()
```

Пример:

Проверка правильности заполнения форм с помощью JavaScript.

С помощью JavaScript можно контролировать правильность заполнения форм.

Попробуйте нажать на "Отправить" в форме расположенной ниже, не заполняя ее.

Имя
E-mail
Коментарий

Отправить Сбросить

Функции для этого примера:

```
<script language="JavaScript">
<!--

function CheckForm(UserForm)
{
   var is_ok = true;

   if (UserForm.name.value == '')
   {
   is_ok = false;
   alert("Введите ваше имя!!");
   UserForm.name.focus();
   }
   if (UserForm.Comment.value == '')
   {
   is_ok = false;
   alert("Введите ваши комменiарии!");
   UserForm.Comment.focus();
   }
   if (UserForm.email.value == '')
   {
   is_ok = false;
   alert("Введите ваш e-mail!");
   UserForm.email.focus();
}

return is_ok;
}
// -->
</script>
```

Вызов функции в теге <FORM> при нажатии на кнопку "Отправить"

```
<form onSubmit = "return CheckForm(this)">
```

Имена полей

```
<input type="text" name="name" size="20">
<input type="text" name="email"  size="20">
<textarea rows="3" name="Comment" cols="20">
```

5. Литература

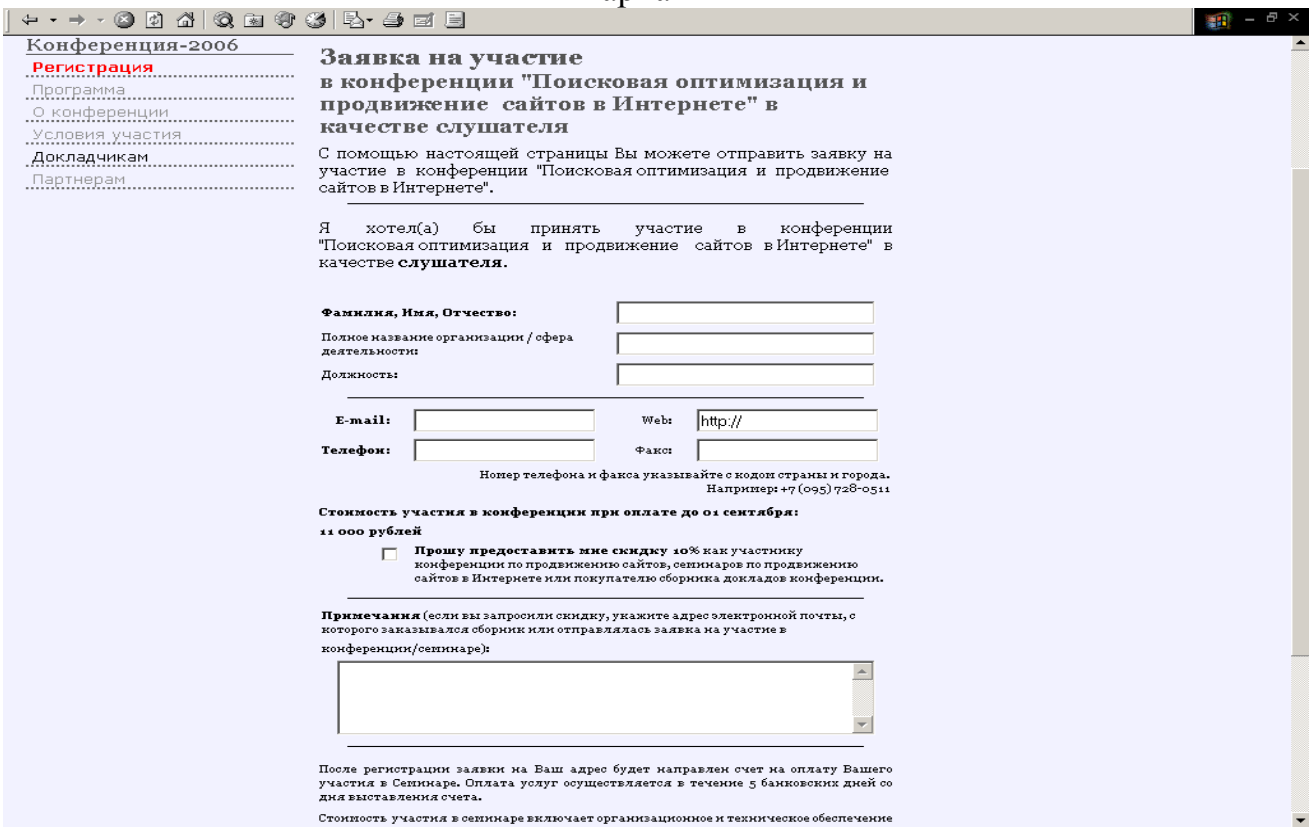
1. .Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Учебник по HTML (оригинальная версия на английском от World Wide Web Consortium'a (W3C)) http://www.w3schools.com/html/default.asp. С работающими примерами «Try it yourself».

3. HTML 4.01 Specification (english) http://www.w3.org/TR/1999/REC-html401-19991224. Спецификация HTML 4.01. Переводчик Пирамидин А. http://www.intuit.ru/department/internet/html/1/

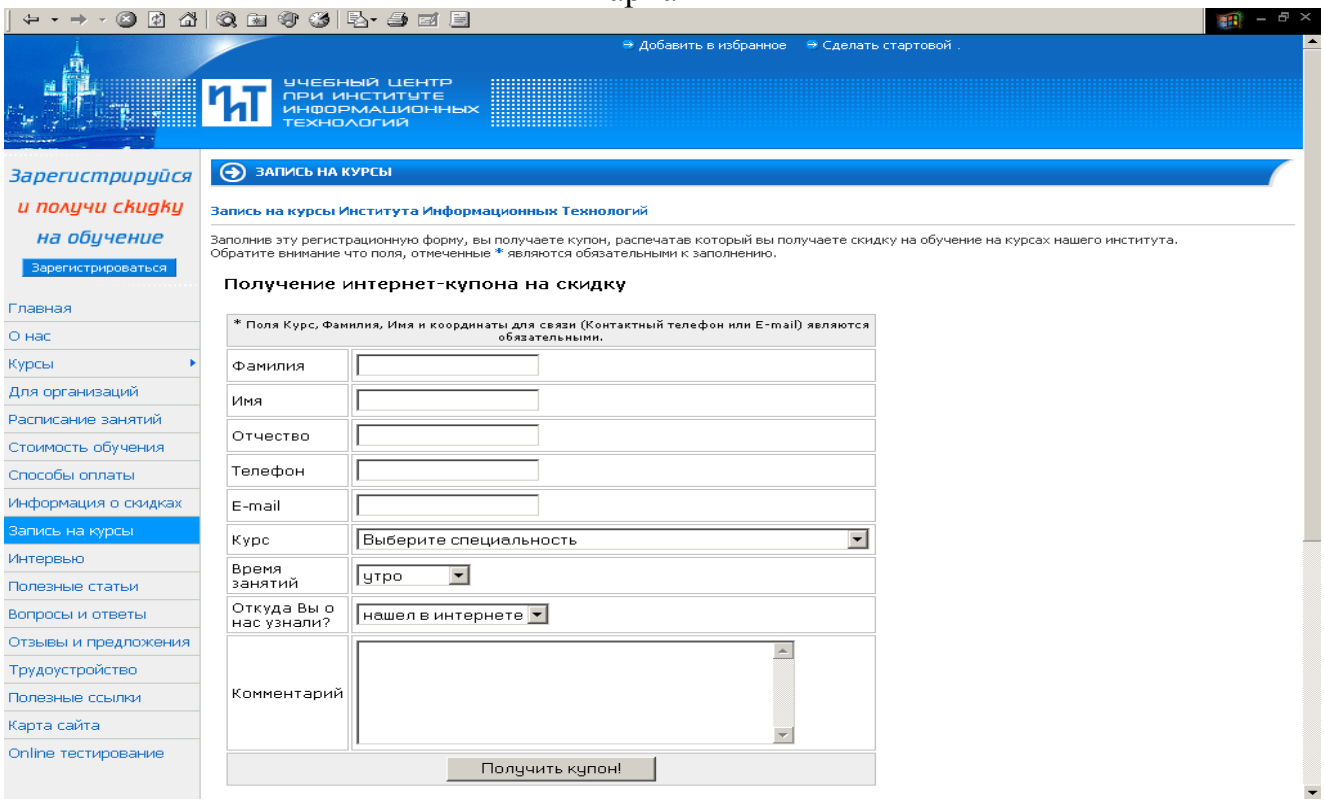
Приложение 1

Вариант 1



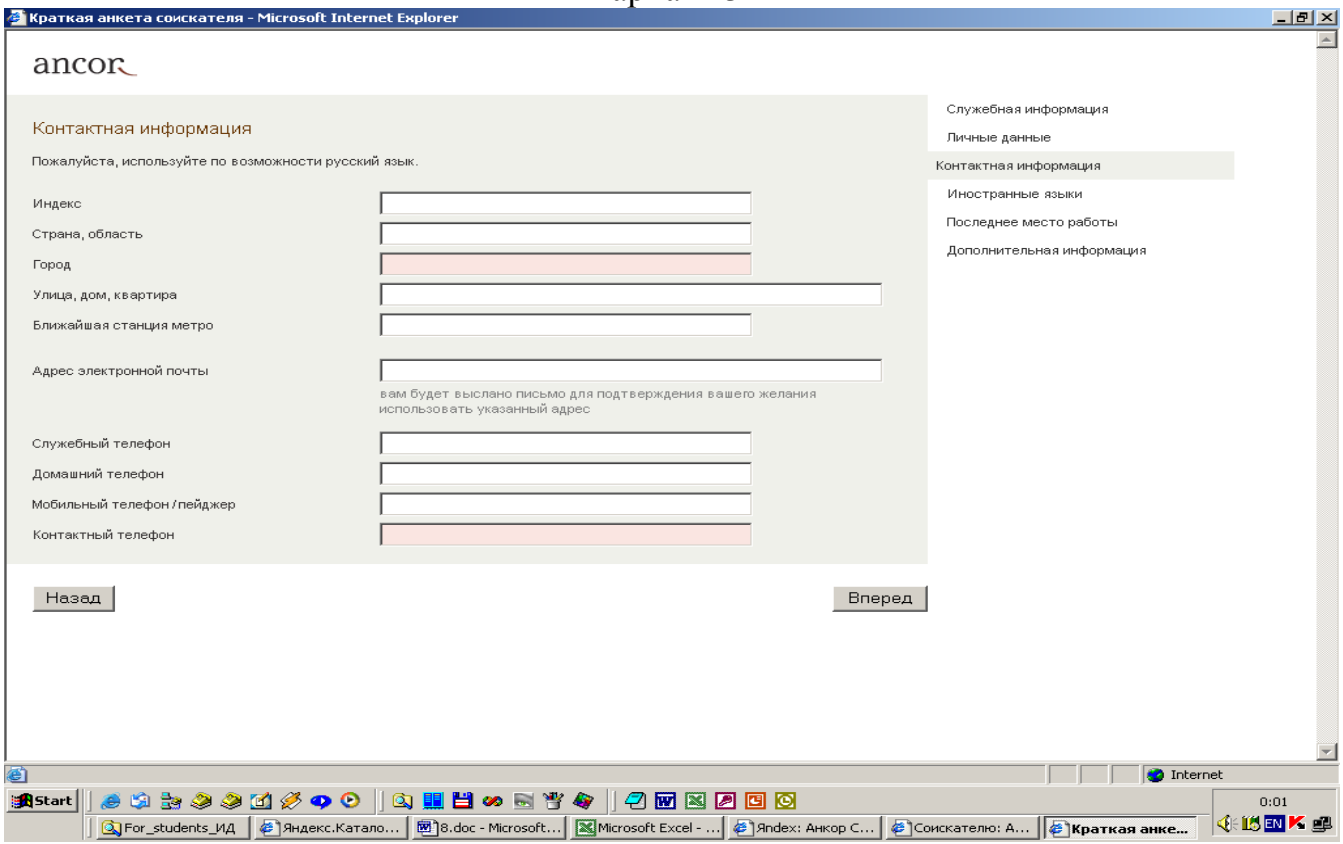
Страница «Форма: Заявка на участие»

Вариант 2



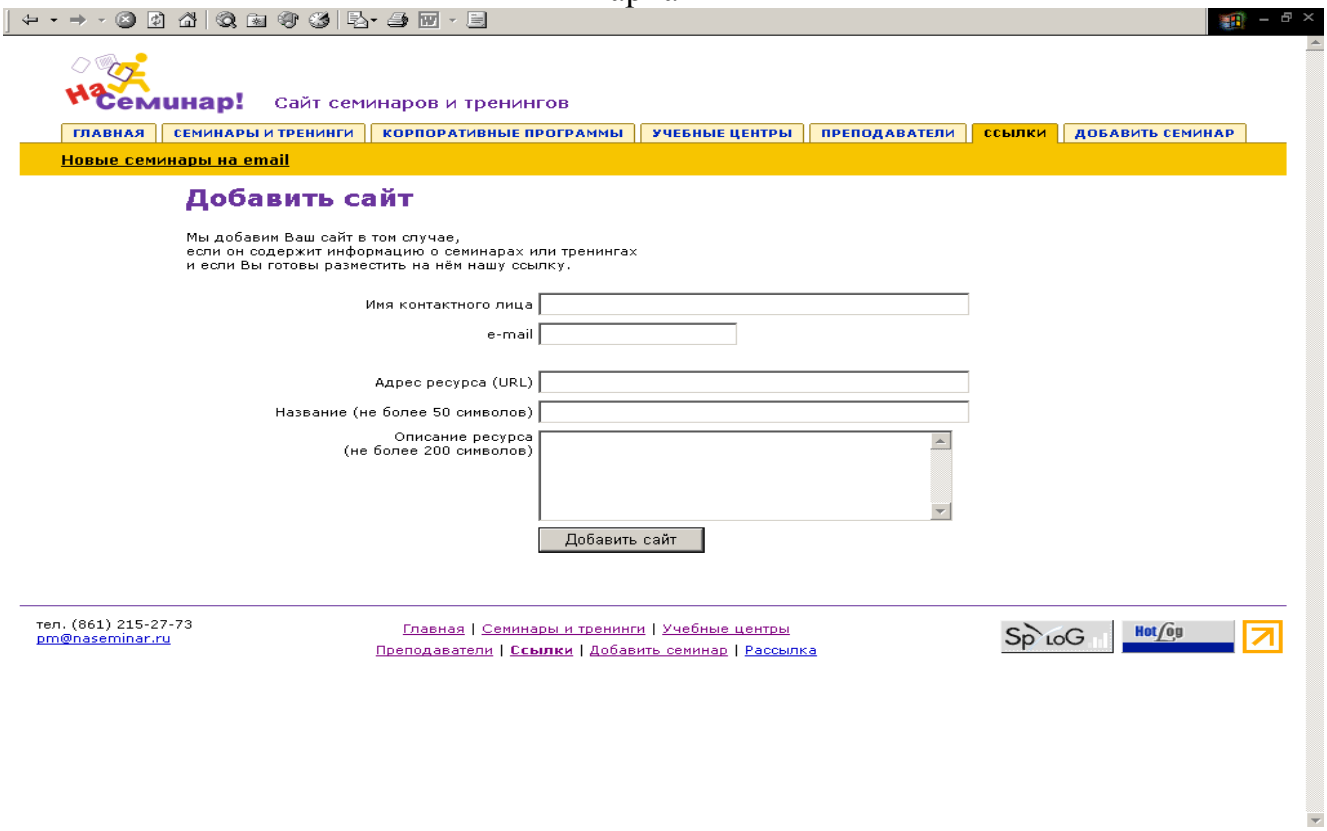
Страница «Форма :Получение купона на скидку»

Вариант 3



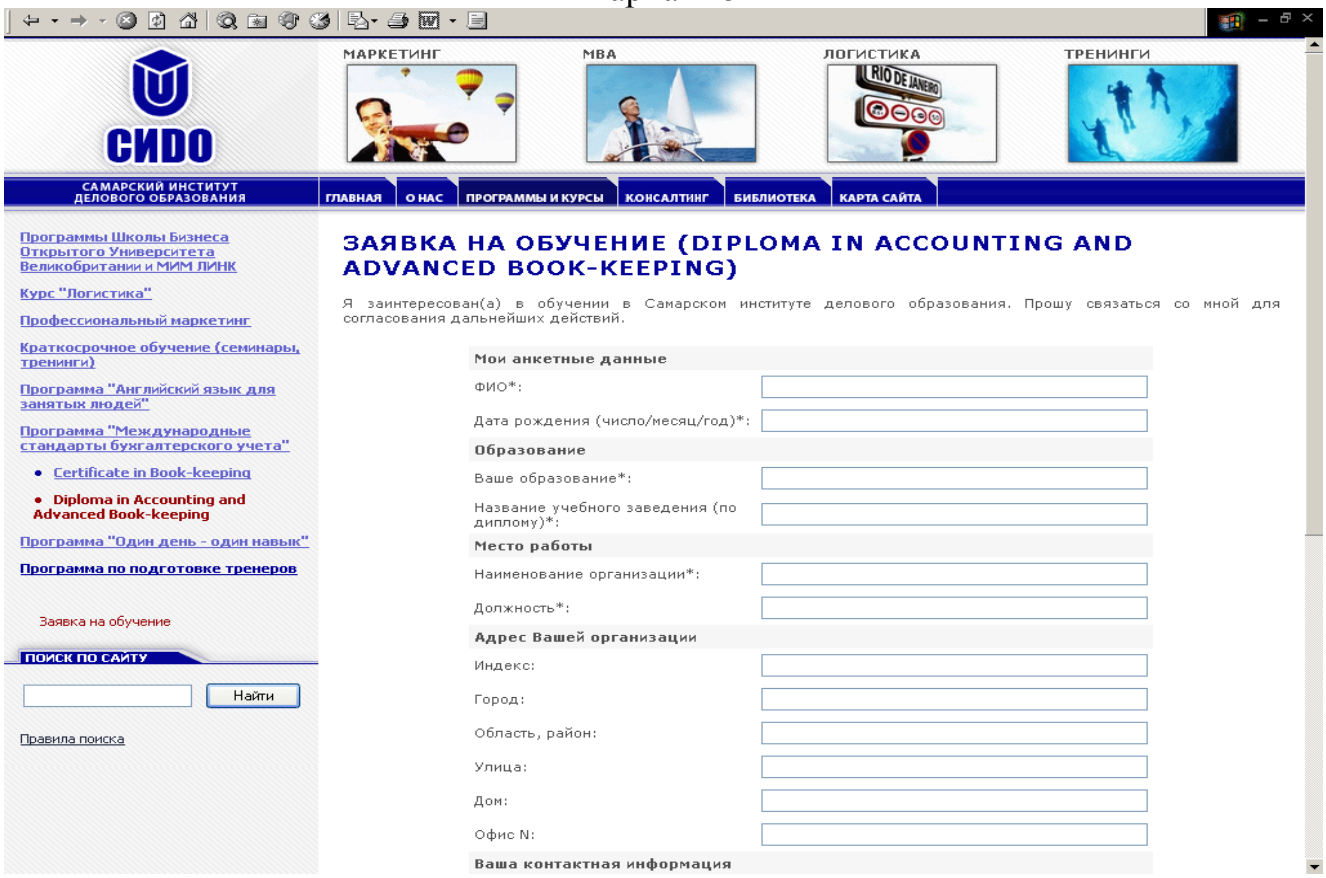
Страница «Форма Контактная информация»

Вариант 4



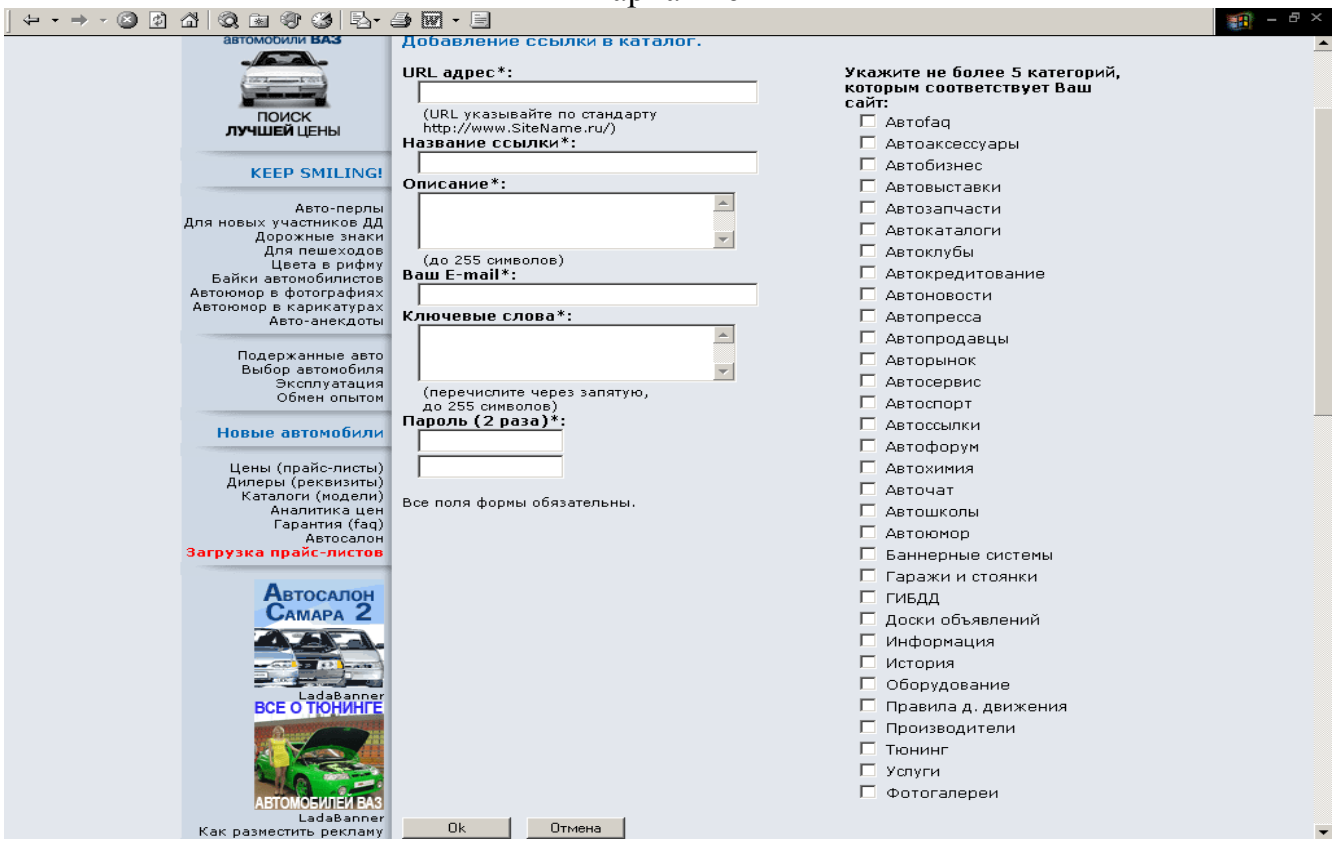
Страница «Форма: Добавить сайт»

Вариант 5



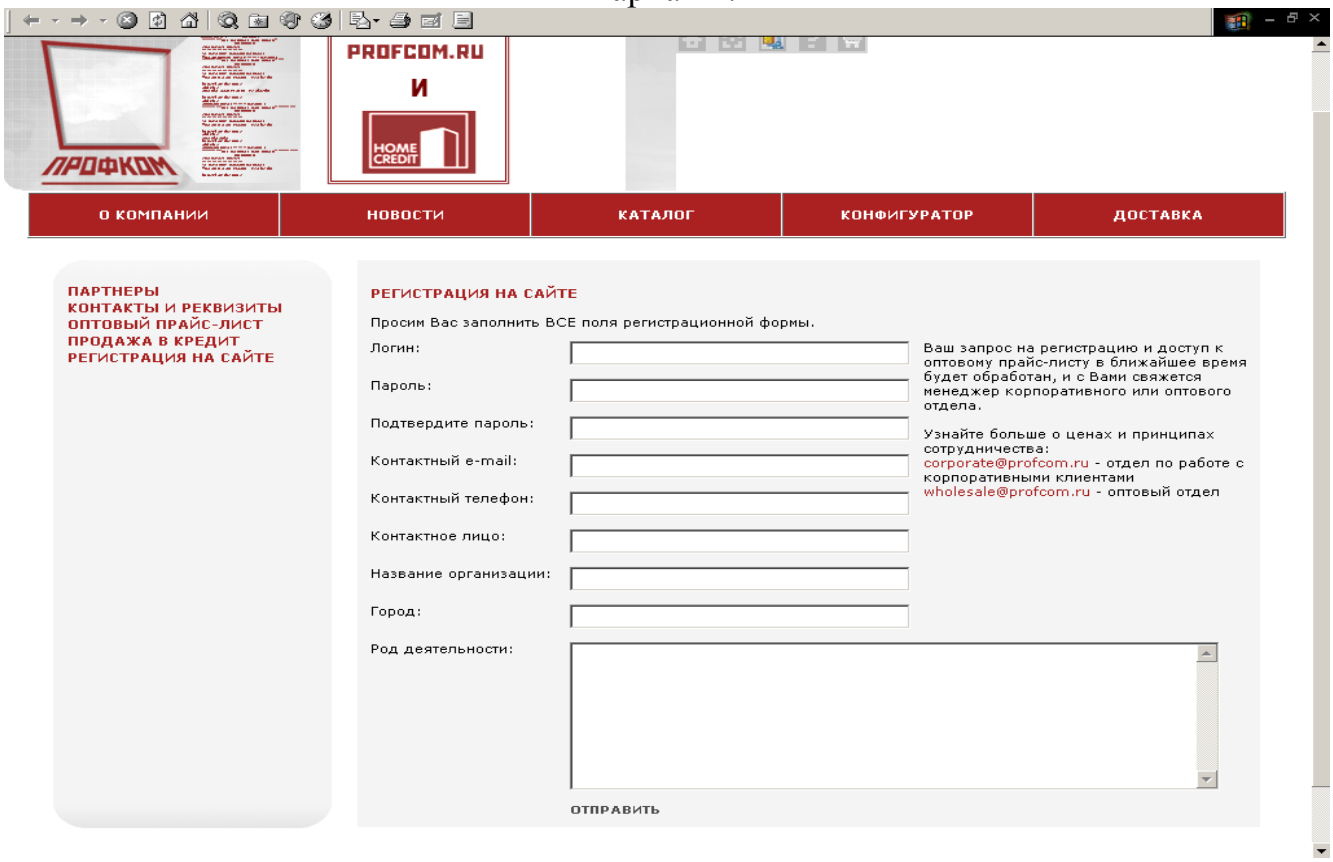
Страница «Форма: Заявка на обучение»

Вариант 6



Страница «Форма: Добавление ссылки в каталог»

Вариант 7



Страница «Форма: Регистрация на сайте»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к Лабораторной работе 6
«Создание Web-интерфейса к базе данных»

1. Порядок выполнения лабораторной работы

Перед выполнением лабораторной работы студенты должны ознакомиться с методическими указаниями по ее выполнению и рекомендованной литературой. В приложении 1 приведены формы, корректность ввода данных в которые надо будет проверить. Варианты соответствуют полученным ранее вариантам.

Для получения зачета по лабораторной работе студент сдает преподавателю полностью оформленный отчет в электронном виде, а также демонстрирует работу на компьютере.

Во время защиты/сдачи лабораторной работы студент обязан продемонстрировать знания по теме работы, теоретическому материалу, методам выполнения работы, содержанию основных разделов разработанного отчета с демонстрацией результатов.

Для самопроверки при подготовке к выполнению работы студент должен ответить на контрольные вопросы, приведенные в конце лабораторной работы.

Общий зачет студент получает после выполнения и сдачи всех лабораторных и контрольных работ курса работы.

2. Задание для выполнения

1. Данные, которые пользователь вводит в форму (см лабораторную работу 5) необходимо записывать в БД на сервере с использование технологии ASP и языка VBScript, используемом для программирования на стороне сервера. Рекомендуется использовать БД MS Access.
2. Дополнительно необходимо создать страницу, где пользователь может посмотреть данные, хранящиеся в БД. Пользователь также должен иметь возможность посмотреть или все записи, или воспользоваться несколькими (не менее чем 2-я) фильтрами. Записи, отображаемые пользователю, должны быть отсортированы по алфавиту.
3. Данные должны отображаться в форме таблицы.
4. Результаты работы разместить на сайт http://www.parking.ru/.

3. Содержание отчета

1. Титульный лист

2. Задание.
3. Структура БД.
4. Код на для выполнения на стороне сервера.
5. Выводы по работе.

4. Теоретический материал

ASP файлы, как правило, представляют собой HTML-файлы с интегрированными в них скриптами. ASP скрипт заключается в специальные тэги “<% %>”.

Напишем первую программу на ASP. Наша программка будет выводить любимое всеми программистами словосочетание “ Hello World ”.

```
<%@ Language=VBScript %>
<html>
<head>
<title> Пример 1</title>
</head>
<body>
<% Response.write "Hello world!" %>
</body>
</html>
```

После интерпретации на сервере клиент получит следующую HTML страничку:

```
<html>
<head>
<title> Пример 1</title>
</head>
<body>
 Hello world!
</body>
</html>
```

Рассмотрим, что происходит в этом примере. “Response” – один из нескольких встроенных ASP объектов.
“Write (arg)” – его метод, производящий запись аргумента(arg) клиенту.
В нашем случае аргументом была строка “Hello world!”.
Для Jscript пример будет аналогичен, за исключением двух строк.
Первая строка, объявляющая используемый язык меняется с “<%@ Language=VBScript %> ” на “<%@ Language=javascript %> ”.
Вызов объекта “Response” поменяется с “<% Response.write "Hello world!" %> ” на “<% Response.Write ("Hello world!"); %>”.
Здесь следует обратить внимание на регистр, точку с запятой после вызова и скобки.

Примеры

Отображение в форме таблицы:

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT Companyname, Contactname FROM Customers",
conn
%>

<table border="1" width="100%">
<%do until rs.EOF%>
  <tr>
  <%for each x in rs.Fields%>
    <td><%Response.Write(x.value)%></td>
  <%next
  rs.MoveNext%>
  </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

Таблица с заголовками:

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers"
rs.Open sql, conn
%>
```

```
<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close
  %>
</table>

</body>
</html>
```

Добавление записи в базу данных:

Вот такая форма:

```
<html>
<body>

<form method="post" action="demo_add.asp">
<table>
<tr>
<td>CustomerID:</td>
<td><input name="custid"></td>
</tr><tr>
<td>Company Name:</td>
<td><input name="compname"></td>
</tr><tr>
<td>Contact Name:</td>
<td><input name="contname"></td>
</tr><tr>
<td>Address:</td>
<td><input name="address"></td>
</tr><tr>
<td>City:</td>
<td><input name="city"></td>
</tr><tr>
```

```
<td>Postal Code:</td>
<td><input name="postcode"></td>
</tr><tr>
<td>Country:</td>
<td><input name="country"></td>
</tr>
</table>
<br><br>
<input type="submit" value="Add New">
<input type="reset" value="Cancel">
</form>

</body>
</html>
```

Передает данные на страницу demo_add.asp и пишет из в БД:

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

sql="INSERT INTO customers (customerID,companyname,"
sql=sql & "contactname,address,city,postalcode,country)"
sql=sql & " VALUES "
sql=sql & "('" & Request.Form("custid") & "',"
sql=sql & "'" & Request.Form("compname") & "',"
sql=sql & "'" & Request.Form("contname") & "',"
sql=sql & "'" & Request.Form("address") & "',"
sql=sql & "'" & Request.Form("city") & "',"
sql=sql & "'" & Request.Form("postcode") & "',"
sql=sql & "'" & Request.Form("country") & "')"

on error resume next
conn.Execute sql,recaffected
if err<>0 then
  Response.Write("No update permissions!")
else
  Response.Write("<h3>" & recaffected & " record
added</h3>")
end if
conn.close
%>
```

```
</body>
</html>
```

Литература
к модулю 1 «Введение в Интернет. Протоколы»

Основная литература

1. Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Основы Web-технологий [Текст]: курс лекций : специальность "Интернет-технологии" : [Для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов [и др.]. - М. : ИНТУИТ. ру, 2003. - 510 с.(8 экз. НТБ СГА)

Дополнительная литература

1. Столлингс, Вильям. Компьютерные сети, протоколы и технологии Интернета : [пер. с англ.] / Вильям Столлингс. - СПб. : БХВ-Петербург, 2005. - 817 с. - Библиогр. в конце глав. (1 экз.)

Электронные источники и интернет ресурсы

1. Администрирование сети и сервисов Internet. Учебное пособие**.** П. Б. Храмцов, Центр Информационных Технологий, 1997. http://citforum.ru/nets/services/index.shtml

2. Интернет университет информационных технологий. Официальный сайт: http://www.intuit.ru

3. Официальный сайт WWW-консорциума (англ.) (стандарты и рекомендации): http://www.w3.org/Protocols/.

4. Спецификация протокола IP (перевод): http://citforum.ru/nets/tcp/ipspec.shtml

5. Спецификация протокола UDP (перевод): http://citforum.ru/nets/tcp/udpspec.shtml

6. Спецификация протокола TCP (перевод): http://citforum.ru/nets/tcp/tcpspec.shtml

Литература к модулю 2 «Создание HTML-документов»

Основная литература

1. Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Основы Web-технологий [Текст]: курс лекций : специальность "Интернет-технологии" : [Для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов [и др.]. - М. : ИНТУИТ. ру, 2003. - 510 с.(8 экз. НТБ СГА)

3. Матросов А. HTML 4.0: Наиболее полное руководство. – СПб.: BHV-Петербург, 2005. – 671 с. (5 экз. НТБ СГАУ)

Дополнительная литература

1. Дейтел, Харви М. Как программировать для Internet & WWW [Текст] / Х. М. Дейтел, П. Д. Дейтел, Т. Р. Нието ; пер. с англ. под ред. А. В. Козлова. - М. : Бином, 2002. - 1177 с(5 экз. НТБ СГАУ)

2. Хейз, Дидре. Освой самостоятельно HTML и XHTML [Текст] : [пер. с англ.] / Дидре Хейз. - 3-е изд. - М. [и др.]: Вильямс, 2003, 2004. - 223 с. (5 экз. НТБ СГАУ)

Электронные источники и интернет ресурсы

1. Учебные и справочные материалы WWW-консорциума. Официальный сайт (англ.): http://www.w3schools.com.

2. Учебные и справочные материалы http://www.wisdomweb.ru/.

3. Официальный сайт WWW-консорциума (англ.) (стандарты и рекомендации): http://www.w3.org.

4. Спецификация языка HTML 401 (en): http://www.w3.org/TR/html5/ (руск.): http://pyramidin.narod.ru/

5. Спецификация языка CSS2 (англ.): http://www.w3.org/Style/CSS/ (руск.): http://pyramidin.narod.ru/

Литература
к модулю 3 «Создание интерактивных документов»

Основная литература

1. Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Основы Web-технологий [Текст]: курс лекций : специальность "Интернет-технологии" : [Для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов [и др.]. - М. : ИНТУИТ.ру, 2003. - 510 с.(8 экз. НТБ СГА)

Дополнительная литература

1. Дейтел, Харви М. Как программировать для Internet & WWW [Текст] / Х. М. Дейтел, П. Д. Дейтел, Т. Р. Нието ; пер. с англ. под ред. А. В. Козлова. - М. : Бином, 2002. - 1177 с(5 экз. НТБ СГАУ)

Электронные источники и интернет ресурсы

1. Учебные и справочные материалы WWW-консорциума. Официальный сайт (англ.): http://www.w3schools.com.

2. Учебные и справочные материалы http://www.wisdomweb.ru/.

3. Официальный сайт WWW-консорциума (англ.) (стандарты и рекомендации): http://www.w3.org.

4. Спецификация языка JavaScript - Стандарт ECMA-262, 3я редакция (англ): http://www.ecma-international.org/publications/standards/Ecma-262.htm , (рус.): http://javascript.ru/ecma

5. Спецификация DOM (англ.): http://www.w3.org/DOM/

6. Справочник по языку JavaScript:http://msdn.microsoft.com/ru-ru/library/ie/d1et7k7c(v=vs.94).aspx

Литература
к модулю 4 «Программирование на стороне сервера»

Основная литература

1. Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. - 374 с. (30 экз. НТБ СГАУ)

2. Основы Web-технологий [Текст]: курс лекций : специальность "Интернет-технологии" : [Для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов [и др.]. - М. : ИНТУИТ.ру, 2003. - 510 с.(8 экз. НТБ СГА)

Дополнительная литература

1. Дейтел, Харви М. Как программировать для Internet & WWW [Текст] / Х. М. Дейтел, П. Д. Дейтел, Т. Р. Нието ; пер. с англ. под ред. А. В. Козлова. - М. : Бином, 2002. - 1177 с(5 экз. НТБ СГАУ)

Электронные источники и интернет ресурсы

1. Учебные и справочные материалы WWW-консорциума. Официальный сайт (англ.): http://www.w3schools.com.
VBScript: http://www.w3schools.com/vbscript/default.asp
ASP 3.0: http://www.w3schools.com/asp/default.asp
ADO (разделы Display[Отображение], Queries[Запросы], Sort[Сортировка])
http://www.w3schools.com/ado/ado_examples.asp

2. Администрирование web-серверов в IIS (Лекция 12: Программирование на ASP) http://www.intuit.ru/department/internet/websadmin/12/

3. Visual Basic Script. Основы программирования: http://www.citforum.ru/internet/vbscript/vbscript1.shtml

4. Введение в ASP: http://www.citforum.ru/internet/asp/asp_intro.shtml

Контрольные вопросы по модулю 1 «Введение в Интернет. Протоколы»

1. В чем состоит основная функция семейства сетевых протоколов TCP/IP?
2. Для чего используется IP-адрес и чему он присваивается?
3. Что такое доменный адрес, и чем его использование удобнее использования IP-адреса? Можно ли обойтись без доменных адресов?
4. Какой компонент технологии Интернета выполняет функцию преобразования доменного адреса в IP-адрес?
5. Что такое URI?
6. Можно ли по MAC-адресу компьютера опѐеделить его принадлежность к определенной сети?
7. Могут ли разные устройства иметь одинаковый MAC-адрес?
8. Сколько всего может быть различных MAC-адресов?
9. Можно ли по IP-пакету определить, данные каких приложений передаются?
10. Сколько бит отводится под сеть при следующих масках сети: 255.255.255.0, 255.240.0.0, 255.255.8.0?
11. По какому адресу можно судить о производителе сетевого адаптера:
12. Что такое IP-адрес и маска подсети? В чем заключается отличие статического адреса от динамического?
13. Опишите особенности уровней стека протоколов TCP/IP? Как они связаны с семиуровневой моделью OSI?
14. Объясните, что означают свойства «платформонезависимость» и «открытость» применительно к стеку протоколов TCP/IP
15. Поясните, для чего предназначена модель OSI? Где она применяется?
16. Назовите функции канального, сетевого и транспортного уровней модели OSI..
17. Поясните принцип работы утилиты ping.
18. Поясните принцип работы утилиты tracert.
19. Перечислите виды и примеры адресов, используемых в стеке TCP/IP.
20. Из каких частей состоит IP-адрес?

21. Как определяется номер подсети в IP-адресе?
22. Каков диапазон возможных адресов у сети класса C?
23. Определите номер подсети на основе маски: 116.98.04.39/27.
24. Для чего необходимы доменные имена?
25. Для чего нужна служба DNS?

Контрольные вопросы по модулю 2 «Создание HTML-документов»

1. Чем различаются локальные и глобальные гипертекстовые ссылки?
2. В чем состоит основное отличие документов Web от других электронных документов?
3. В чем разница между обычной страницей и гипертекстовой страницей?
4. Для чего предназначен HTML?
5. Как браузер обрабатывает HTML-документ?
6. Что такое тег и для и для чего предназначен?
7. Объясните назначение тегов гипертекстовой разметки: <HEAD>, <A HREF=” “>.
8. Чем отличаются текстовые и графические гиперссылки?
9. Предположим, что на Web-странице опубликован очень длинный документ (повесть). Для удобства пользователя автор ввел в начало документа содержание, состоящее из 20 пунктов, соответствующих главам повести. Что он должен предусмотреть, чтобы мог перейти к любой главе щелчком на соответствующем пункте содержания?
10. Какие виды списков вы знаете? Привести пример.
11. Какими средствами создаются списки на Web-страницах?
12. Что такое вложенные списки Web-документа?
13. Какие три типа CSS Вы знаете?
14. В каком месте Web-страницы записываются встроенные CSS?
15. Где располагается описание внешних CSS?
16. В каком месте страницы описываются внедренные CSS?
17. Какое основное достоинство каскадных таблиц стилей?
18. Какое расширение должен иметь файл с описанием внешних CSS?
19. Для чего используются классы?
20. Какие программы используются для автоматизации создания таблиц стилей?
21. Какова зона действия встроенных CSS?

22. Какова зона действия внедренных CSS?
23. Какова зона действия внешних CSS?
24. Как с помощью CSS создать псевдографическое изображение?
25. Что такое стиль?
26. Почему таблицы стилей называют каскадными?
27. Какой тип CSS следует использовать, если предполагается изменение формата сразу нескольких страниц?
28. Какой из трех типов CSS следует считать наименее перспективным?
29. Как выполнить группировку селекторов?

Контрольные вопросы по модулю 3 «Создание интерактивных документов»

1. Как обратиться к Форме, содержащейся в документе?
2. Метод window.alert() ?
3. Метод window.confirm() ?
4. Метод window.prompt() ?
5. Метод document.write() ?
6. Однострочный комментарий в языке JavaScript?
7. Создание одномерного массива?
8. Присвоение значений элементам одномерного массива?
9. Свойство, в котором хранится размер массива ?
10. Синтаксис полного и сокращённого операторов if ?
11. В каком случае можно не использовать операторные скобки в операторе if?
12. Синтаксис оператора цикла for ?
13. Области видимости переменных ?
14. Метод document.write() ?
15. Метод getElementById()
16. Операции языка JavaScript?
17. В чём различия вывода данных с помощью метода alert(), с помощью свойства innerHTML и с помощью метода document.write() ?
18. Свойство объекта Math, содержащее число "пи" ?
19. Ключевое (служебное или зарезервированное) слово, предназначенное для образования переменных ?
20. Многострочный комментарий в языке JavaScript ?
21. Примеры вызова (обращения к) функции ?
22. Свойство innerHTML ?
23. Синтаксис оператора цикла while ?
24. Синтаксис оператора цикла do ... while ?

25. Для чего нужны тэги <script></script> ?
26. Для чего в открывающем script-тэге используется атрибут type ?
27. Является ли код языка JavaScript чувствительным к регистру букв ?
28. Какое расширение имени обычно используется для внешнего Javascript-файла?
29. Какой атрибут используется для ссылки на внешний JavaScript-файл ?
30. Пример использования операции конкатенации ?
31. Пример присвоения переменной значения, вводимого в текстовое поле формы, если форма имеет имя "forma", а имя текстового поля - "input" ?
32. Какова область видимости переменной, создаваемой в функции, если при её создании используется зарезервированное слово var ?
33. Какова область видимости переменной, создаваемой в функции, если при её создании не используется зарезервированное слово var ?
34. Можно ли использовать в качестве имён функций наименования открывающих HTML-тэгов ?

Контрольные вопросы по модулю 4 «Программирование на стороне сервера»

1. Что такое ASP?
2. Какой язык используется по умолчанию для программирования на ASP?
3. Как в HTML-документе можно указать, что код выполняется на стороне сервера?
4. Какие языки программирования могут использоваться для программирования на ASP?
5. Для чего используется объект Request?
6. Объект Response – что это такое?
7. Web – приложение. Что это такое?
8. Что такое сессия пользователя?
9. Какой объект ASP используется для идентификации пользователя?
10. Какие файлы имеют расширение ".inc"? Для чего они используются?
11. Назначение файла global.asa.
12. Какие события являются стандартными для файла global.asa?

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к курсовой работе по дисциплине
«Технологии сети Интернет»

1. Общие положения

Дисциплина «Технологии сети Интернет» включена в учебный план специальности 010300 Фундаментальная информатика и информационные технологии (квалификация (степень) «бакалавр»).

Темы курса изложены в курсе лекций [1], далее были закреплены при выполнении лабораторных работ [1], на которых студенты поэтапно создают небольшие сайты, работая в команде из двух-трех человек. Все это служит основой для выполнения курсовой работы в соответствии с предъявляемыми требованиями. Цель курсовой работы дисциплине «Технологии сети Интернет» подготовить студентов к выполнению итоговой выпускной работы.

Тему курсовой работы выдает преподаватель. В соответствии с ней студент разрабатывает техническое задание. Техническое задание в дальнейшем является основным документом, по которому студент выполняет курсовую работу. Любые изменения технического задания на систему должны быть согласованы спреподователем.

Документация по проекту разрабатывается в соответствии с существующими стандартами [2].

Завершающая стадия работы создание макета программы, его отладка, тестирование и сдача. Студент предъявляет преподавателю для проверки на ПК завершенную реализацию программы и при необходимости производит ее доработки и защищает.

2. Примерные темы курсовых работ

Примерные темы курсовых работ приведены ниже и могут быть изменены:

1. Разработка макета сайта биржи труда.

2. Разработка макета сайта дистанционного обучения по курсу «Технологии сети Интернет».

3. Разработка макета сайта по формированию и ведению банка вакансий.

4. Разработка макета сайта для организации дистанционной работы по договору.

5. Разработка макета сайта для информационного взаимодействия кадровой службы судоходной компании с внешним рынком труда.

6. Разработка макета сайта для информационного взаимодействия кадровой службы бюджетной организации с внешним рынком труда.

7. Разработка макета Интранет-сайта организации по делопроизводству и регистрации исполнения поручений.

8. Разработка макета Интранет-сайта по управлению проектной деятельностью на предприятии.

9. Разработка макета Интранет-сайта по управлению проектной деятельностью на предприятии.

10. Разработка макета сайта – хранилища документов на предприятии.

11. Разработка интерактивного сайта «Интернет-газета

12. Разработка базы данных архива фотодокументов и технологии ее представления в Internet.

13. Разработка макета информационно-телекоммуникационной системы «Виртуальный университет»

3. Задание на программную систему

Задание на курсовую работу выдается преподавателем, на его основании студент разрабатывает техническое задание, которое подписывается преподавателем. С момента утверждения задания студент считается приступившим к выполнению курсовой работы. Оформление технического задания производится с учетом требований стандарта [2] по следующим разделам:

1. Содержание задания (анализ и описание предметной области).

2. Характеристики объекта (элементы объекта и условия их функционирования, количественные и качественные показатели объекта, накладывающие ограничениями, информационные характеристики (объемы и интенсивности потоков данных в существующих условиях)).

3. Требования к информационному, техническому и программному обеспечению (исходные документы, перечень исходных и выходных данных, ограничения по составу технических средств, перечень используемых системных и прикладных программных средств, требования по совместимости разрабатываемого программного обеспечения с существующими системами).

4. Общие требования к проектируемой системе/подсистеме (перечень функций, которые должна выполнять проектируемая система или подсистема в процессе ее эксплуатации, способы ввода, обработки, передачи и хранения данных)

5. Перечень дополнительных работ (если необходимо).

4. Содержание курсовой работы

В рамках курсовой работы студент разрабатывает небольшой сайт (возможна доработка задания на лабораторную работу).

Основные шаги работы над системой.

1. Анализ предметной области, где выделяются основные объекты, участвующие в функционировании системы, определяются их наиболее

существенные характеристики, взаимосвязи в рамках решаемой задачи, а также определяются основные информационные потоки в системе.

2. Постановка задачи. На данном этапе формулируются все требования, которым должна удовлетворять система, перечисляются функции, выполняемые системой.

3. Построение структурной схемы программной системы. На данном этапе система по функциональному признаку разделяется на основные подсистемы, между ними указываются информационные связи или связи по управлению, описывается основное назначение подсистем. При необходимости структура подсистем детализируется.

4. Разработка функциональной спецификации системы:

- Входные и выходные данные;
- Описание внешней информационной среды, которое должно быть представлено в виде контекстной диаграммы потоков данных:
- Перечень исключительных ситуаций и реакцию системы на их возникновение, при необходимости приводится перечень ошибок, которые могут возникать в системе и соответствующие им системные сообщения;

5. Реализация проекта. Преподаватель проверяет полноту и качество реализации функций, соответствие системы техническому заданию. Для демонстрации работоспособности системы необходимо подготовить нескольких тестовых примеров. При необходимости производится доработка реализации с повторным предъявлением системы, после доработки система выносится на защиту.

6. Полное оформление документации курсовой работы. После приемки реализации студент оформляет пояснительную записку к курсовой работе со всеми требуемыми приложениями.

5. Оформление отчета

Пояснительная записка оформляется в соответствии со стандартом СГАУ [2] и должна содержать:

1. титульный лист
2. задание
3. реферат
4. содержание
5. введение
6. основная часть
7. заключение
8. перечень принятых сокращений (при наличии)
9. перечень принятых терминов (при наличии)
10. список использованных источников
11. приложения

Рекомендуемый объем пояснительной записки 15-25 страниц машинописного текста (без приложений).

6. Список дополнительных источников

1. Технологии сети Интернет [Электронный ресурс]. - Мин-во обр. и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). - Самара, 2013. – 1 эл. опт. диск (CD-ROM).

2. СТО СГАУ 02068410-004-2007. Общие требования к учебным текстовым документам. Самара, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т) - Самара, 2007. - 30 с.

Студент: _____

БИЛЕТ №1

1. HTML-документ содержит код, приведенный выше. По вашему мнению, допущена ли здесь какая-либо ошибка?

```
<TABLE border="1">
<TR><TD>1<TD>2<TD>3
<TR><TD>4<TD ROWSPAN="2">5<TD>6
<TR><TD COLSPAN="2">7<TD>8</TABLE>
```

   - ☐ Да, ошибка в структуре таблицы
   - ☐ Да, отсутствует обязательный заголовок таблицы, оформленный тегом <CAPTION>
   - ☐ Да, ошибка заключается в отсутствии некоторых закрывающих (парных) тегов
   - ☐ Нет, код записан без ошибки

2. Атрибут SIZE тега <INPUT> устанавливает:
   - ○ Размер поля ввода по горизонтали в символах
   - ○ Размер поля ввода по вертикали в символах
   - ○ Размер поля ввода по горизонтали в пикселах
   - ○ Размер поля ввода по вертикали в пикселах
   - ○ Максимальное значение вводимых символов

3. Атрибуты WIDTH и HEIGHT тега <IMG> могут быть использованы для:
   - ☐ Установки для изображения размеров в пикселях, отличных от его действительных размеров
   - ☐ Установки для изображения размеров в процентах, отличных от его действительных размеров
   - ☐ Установки горизонтальных и вертикальных полей вокруг изображения
   - ☐ Указания браузеру зарезервировать данные размеры для изображения в процессе загрузки HTML документа еще на этапе вывода текстовой части

4. В какой цветовой модели задается цвет в HTML?
   - ○ RGB
   - ○ HSB
   - ○ CMYK
   - ○ LAB
   - ○ WEB

5. Какой тэг используется для увеличения шрифта на 1 пункт?
   - ○ <SMALL>
   - ○ <BIGFONT>
   - ○ <FONT size=3>
   - ○ <H1>
   - ○ <BIG>

6. Какое значение из представленных НЕ является значением свойства "text-transform"?
   - ○ Small-caps
   - ○ None
   - ○ Lowercase
   - ○ Capitalize
   - ○ Uppercase

7. Укажите характеристику НЕ относящуюся к каскадным таблицам стилей:
   - ○ Расширение возможностей форматирования
   - ○ Возможность создания новых элементов разметки (тэгов)
   - ○ Возможность использования одного внешнего листа стилей для нескольких документов
   - ○ Подготовка документов для отображения на экране, принтере, проекторе, звуковое воспроизведение и т.д.
   - ○ Возможность точного позиционирования текстовых блоков

8. Какова область действия переменной, объявленной при помощи оператора var в теле сценария вне какой-либо функции (язык JavaScript)?
   - ○ Только внутри всех функций документа
   - ○ Только внутри данного JavaScript-сценария, ограниченного одним контейнером <script> </script>
   - ○ Во всех JavaScript-сценариях данного документа

9. Каково будет значение переменной a после выполнения приведенного JavaScript кода?

```
var a = 1;
function myFunc(a)
{
a = 2;
return a;
}
myFunc(a);
```

   - ○ a = 0;
   - ○ a = 1;
   - ○ a = 2;
   - ○ a = undefined;

10. Как оформляются комментарии в ASP при использовании языка VBScript?
   - ○ <!-- комментарий -->
   - ○ < комментарий >
   - ○ ' комментарий
   - ○ <% комментарий %>

11. Пользователь передает данные из формы. В чем заключается ошибка примера их приема?

```
<%
If Request.ServerVariables(“REQUEST_METHOD”) = “GET” Then
UserName=Request.Form(“name”)
End If
%>
```

   - ○ Имя элемента формы должно быть записано заглавными буквами
   - ○ Метод Form не соответствует методу передачи GET
   - ○ Серверные переменные возвращаются с помощью объекта Server
   - ○ Ошибок нет

12. ActiveX Data Objects (ADO) устанавливается вместе с ASP и позволяет с ваших страниц легко получать доступ к базе данных. Какие два объекта ADO используются для открытия и соединения и доступа к базе данных?
   - ○ объекты Connection и RecordingSet
   - ○ объекты Connection и Recordset
   - ○ объекты Connect и RecordingSet
   - ○ объекты Connect и Recordset

13. В asp-коде необходимо считать данные из поля формы "<input type=text name=ff>" Какой фрагмент кода позволяет это сделать?
   - ☐ Request.Cookies("ff")
   - ☐ Request.ServerVariables("forms[0].ff")
   - ☐ Request.QueryString("ff")
   - ☐ Request.Form("ff")
   - ☐ Session(“ff”)
   - ☐ Session.SessionID ”ff”

14. Какому объекту принадлежит метод CreateObject?
   - ○ Server
   - ○ Response
   - ○ Application
   - ○ Session

Студент: _____

БИЛЕТ №2

1. HTML-документ содержит несколько таблиц. Как задать цвет текста для одной из таблиц? Выберите правильные варианты:
   - ☐ Окружить таблицу тегами <FONT>
   - ☐ Использовать тег <FONT> для текста внутри каждой ячейки таблицы
   - ☐ И разделе <STYLE> создать селектор TABLE и для него соответствующую декларацию
   - ☐ В разделе <STYLE> создать селектор TD и соответствующую декларацию
   - ☐ В разделе <STYLE> создать описание класса и указать его имя в атрибуте CLASS соответствующего тега <TABLE>
   - ☐ В разделе <STYLE> создать описание класса и указать его имя в атрибуте CLASS соответствующих тегов <TD>

2. В каком варианте правильно установлено начальное значение списка, если список необходимо начать с латинской буквы “c”?
   - ○ <OL TYPE="a" START="3">
   - ○ <OL START="c">
   - ○ <OL TYPE="c">
   - ○ <OL TYPE="a" START="c">

3. В качестве содержимого гиперссылок могут выступать:
   - ☐ Текст
   - ☐ Графический объект
   - ☐ Ячейки таблиц, оформленные пустыми тегами <TD></TD>
   - ☐ Поля ввода форм
   - ☐ Содержимое элементов списка, созданных тегами <LI>

4. В каком разделе HTML-документа содержится тело страницы?
   - ○ <HEAD>
   - ○ <BODY>
   - ○ <FRAMESET>
   - ○ <MAIN>
   - ○ <SCRIPT>
   - ○ <STYLE>

5. Какой атрибут тега <TABLE> управляет зазором между ячейками?
   - ○ CELLSPACING
   - ○ CELLPADDING
   - ○ SPACE
   - ○ BORDER

6. Какое свойство используется для установки для текстового блока величины отступа сверху?
   - ○ Margin-top
   - ○ Margin-bottom
   - ○ Spacing-top
   - ○ Padding-top
   - ○ Border-top

7. Какая функция JavaScript позволяет преобразовать строку, содержащую и числовые и алфавитные символы в целое число?
   - ○ Number()
   - ○ parseInt()
   - ○ parseFloat()
   - ○ parseNumber()
   - ○ Integer()

8. В каких строках нижеприведенного фрагмента есть ошибки?

```
1.  <STYLE type="text/css">
2.  P {font-family: Tahoma;
3.     font-style: italic;
4.     font-weight: 1000;
5.     font-size: medium;
6.     color: green;}
7.  </STYLE>
```

- ○ 2, 5, 6
- ○ 3
- ○ 4
- ○ 5
- ○ 1, 7

9. В каких операторах допущена ошибка? (язык JavaScript)
- ☐ myStr = "Изучаем "JavaScript"";
- ☐ myStr = "Изучаем "JavaScript";
- ☐ myStr = "Изучаем 'JavaScript'";
- ☐ myStr = "Изучаем \"JavaScript\"";

10. Переменные используются для хранения как числовых значений и текста. Какие имена переменных НЕ ЯВЛЯЮТСЯ корректными?
- ○ X
- ○ X1
- ○ 1X
- ○ x1X

11. Какие операторы VBScript можно использовать для организации циклов?
- ☐ For…Next.
- ☐ While…wend
- ☐ If…then…else
- ☐ ничего из вышеперечисленного

12. Как в ASP вывести пользователю на экран текст "ASP – технология Microsoft"?
- ○ Response.Write("ASP – технология Microsoft")
- ○ "ASP – технология Microsoft"
- ○ "ASP – технология Microsoft"
- ○ Document.Write("ASP – технология Microsoft")

13. Вы расположили на Web - странице изображение в формате gif большого объема. Как сообщить пользователю, чтобы он подождал окончания загрузки?
- ○ Response.Write("interesting image. Please, wait")
- ○ Server.HTMLEncode("interesting image. Please, wait")
- ○ Response.Status "interesting image. Please, wait"
- ○ Response.ClientCertificate("interesting image. Please, wait")

14. На странице page1.asp есть ссылка: Перейти
Как на странице page2.asp извлечь значение параметра "color"?
- ○ Response.QueryString("color")
- ○ Get("color")
- ○ Response.Parameter("color")
- ○ Request.Form("color")
- ○ Request.QueryString("color")

Студент: _____

БИЛЕТ №3

1. В чем ошибка при следующем написании ссылки: <A HREF="MAIL TO: AAAA@DA.RU"> ПОЧТА </A>?
   - ○ Нельзя использовать заглавные буквы
   - ○ Нельзя внутри названия протокола делать пробел
   - ○ Нельзя после двоеточия делать пробел
   - ○ Нельзя в электронном адресе использовать название протокола
   - ○ Ошибки нет

2. Выберите правильный вариант задания цвета фона страницы:
   - ○ <BODY BACKGROUND=BLACK>
   - ○ <BODY BGCOLOR=BLACK>
   - ○ <BODY COLOR=BLACK>
   - ○ цвет фона определяется настройками цветовой палитры WINDOWS

3. Каким образом иллюстрацию можно отобразить в центре окна браузера (по горизонтали)?
   - ☐ С помощью атрибута ALIGN="CENTER" тега <IMG>
   - ☐ С помощью атрибута ALIGN="MIDDLE" тега <IMG>
   - ☐ С помощью атрибута ALIGN="CENTER" окружающего тега <P>
   - ☐ С помощью атрибута ALIGN="CENTER" окружающего тега <DIV>
   - ☐ Это невозможно сделать никакими тегами HTML

4. Какое значение атрибута “TARGET” надо использовать при написании гиперссылки, чтобы по щелчку она загружалась в новом окне?
   - ○ _SELF
   - ○ _BLANK
   - ○ _TOP
   - ○ NEW WINDOW

5. Укажите неверный способ применения шрифта "Arial" ко всему документу.
   - ○ <BODY> <FONT face=Arial> </FONT> </BODY>
   - ○ <HEAD> <STYLE> BODY {font-face: Arial;}</STYLE> </HEAD> <BODY>
   - ○ <HEAD> <STYLE> BODY {font-family: Arial;}</STYLE> </HEAD> <BODY>
   - ○ <HEAD> <BASEFONT face="Arial"> </HEAD>
   - ○ <BODY style="font-family: Arial;">

6. Укажите номер строки с ошибкой.

```
1.  DIV {background-color: green;
2.       background-image: rifg.jpg;
3.       background-repeat: no-repeat;
4.       background-position: 50% 50%;
5.       background-attachment: fixed}
```

   - ○ 5
   - ○ 4
   - ○ 1
   - ○ 3
   - ○ 2

7. Где внутри HTML-документа могут при помощи тега <script> размещаться JavaScript-сценарии?
   - ○ Только в разделе head
   - ○ Только в разделе body
   - ○ В разделах head и body
   - ○ Вне разделов head и body

8. Каково будет значение переменной x после выполнения приведенного JavaScript кода?

```
x = 1;
switch(x)
{
case 0: break;
case 1: ++x;
case 2: x = 10;
break;
}
```

- ○ x = 0
- ○ x = 1
- ○ x = 2
- ○ x = 3
- ○ x = 4
- ○ x = 5
- ○ x = 10

9. Какое свойство объекта document позволяет получить доступ к цвету основного текста документа?
   - ○ document.text
   - ○ document.textColor
   - ○ document.fgColor
   - ○ document.color

10. Какие имена переменных верные?
    - ○ employee salary
    - ○ 2nd_employee
    - ○ employee_hire_date
    - ○ date-of.birth

11. Для чего предназначено свойство Response.Status?
    - ○ Возвращает статус клиенту
    - ○ Возвращает статус соединения клиент/сервер
    - ○ Указывает статус текущего ответа, основанного на статусе запроса
    - ○ Возвращает статус WEB сервера
    - ○ Используется для изменения значения статусной строки возвращаемой сервером

12. Каким образом можно использовать язык JavaScript на ASP странице?
    - ○ В конце документа указать: <% language="javascript" %>
    - ○ Начать документ с: <%@ language="javascript" %>
    - ○ Начать документ с: <% language="javascript" %>
    - ○ В ASP язык JavaScript является языком, используемым по умолчанию

13. Вы хотите завершить сессию пользователя. Какие методы можно использовать?
    - ○ Session.Abandon
    - ○ Application.Lock
    - ○ Response.Redirect
    - ○ Response.Expires

14. Включаемые файлы обязательно должны иметь расширение ".inc"
    - ○ Верно
    - ○ Неверно

Студент: _____

БИЛЕТ №4

1. Выберите правильные утверждения относительно фона страницы:
   - ☐ Элементом фонового узора должен быть графический объект.
   - ☐ Элементом фонового узора может быть любой объект, в том числе и файл HTML.
   - ☐ Для элемента фонового узора можно использовать параметры масштабирования.
   - ☐ Если фоновый узор имеет прозрачные области, то они будут заполнены цветом, указанным в соответствующем параметре тега <BODY>.
   - ☐ Управлять размножением фонового узора в окне браузера невозможно (даже с помощью CSS).
   - ☐ По умолчанию фоновый узор заполняет все пространство окна браузера.

2. Внутри каких контейнеров (парных тегов) нельзя использовать никакие другие теги
   - ☐ <TITLE>
   - ☐ <CAPTION>
   - ☐ <A>
   - ☐ <OPTION>
   - ☐ <TEXTAREA>
   - ☐ <FORM>
   - ☐ <HEAD>
   - ☐ <PRE>

3. Для тега <CAPTION> справедливы следующие утверждения:
   - ☐ Тегом <CAPTION> оформляется название таблицы
   - ☐ Использование закрывающего тега </CAPTION> не является обязательным
   - ☐ Использование тегов разметки на уровне текста (inline elements) сразу за открывающим тегом <CAPTION> невозможно
   - ☐ Тег <CAPTION> необходимо располагать сразу за тегом <TABLE>

4. Для каких целей может использоваться тег <META>?
   - ☐ Для указания любой информации об HTML-документе, которая не отображается в окне браузера
   - ☐ Для указания ключевых слов и краткой аннотации содержимого HTML-документа для поисковых систем
   - ☐ Для формирования содержимого HTTP-заголовка
   - ☐ Для указания используемой кодировки символов HTML-документа
   - ☐ Для связи данного документа с внешним стилевым файлом CSS
   - ☐ Для запрещения поисковым роботам индексировать данный HTML-документ

5. Какой из тегов позволяет получить самый крупный текст при отображении в браузере?
   - ○ <H1>
   - ○ <FONT SIZE="7">
   - ○ <H6>
   - ○ <FONT SIZE="1">

6. Выберите такие варианты, которые дадут одинаковый цвет текста.
   - ☐ {color: red}
   - ☐ {color: #ff0000}
   - ☐ {color: #F00}
   - ☐ {color: rgb(100,0,0)}
   - ☐ {color: rgb(red)}

7. Укажите корректное правило для позиционирования фонового рисунка по центру:
   - ○ Background-attachment: center center
   - ○ Background: 50%, 50%
   - ○ Background-repeat: top
   - ○ Background-position:50% 50%
   - ○ Background-position: center; center

8. Каково будет значение переменной j после выполнения приведенного JavaScript кода?

```
var i, j;
for (i = 0; i <= 10; i+=2)
{
j = i;
}
```

- ○ 1
- ○ 8
- ○ 9
- ○ 10
- ○ 11
- ○ 12

9. Какой метод объекта Date позволит в JavaScript-сценарии получить текущий день недели?
   - ○ getDay()
   - ○ setDay()
   - ○ Day()
   - ○ currentDay()

10. Какой из представленных методов объекта document позволяет осуществлять динамическое создание HTML-документа?
    - ○ clear()
    - ○ write()
    - ○ read()
    - ○ dynamic()
    - ○ html()

11. Некоторые утверждают, что JavaScript проще, чем ASP и не зависит от операционной системы пользователя (Unix, Linux, Windows 2000). Другие утверждают, что ASP имеет следующие преимущества перед JavaScript, такие как:
    - ☐ ASP не зависит от браузера пользователя.
    - ☐ ASP не отображается при просмотре кода страницы, а значит код защищен.
    - ☐ ASP не загружается в браузер пользователя.
    - ☐ С помощью ASP легко организовать взаимодействие с базой данных.

    Отметьте правильные утверждения.

12. Необходимо создать страницу UserProfile.asp, на которой пользователь сможет проходить процедуру регистрации. ID пользователя (UserID) для извлечения информации из БД передается WEB-серверу следующим образом: http://server/UserProfile.asp?UserID=idxxx.
    Каким образом правильно получить значение UserID, переданное WEB серверу из URL запрашиваемой страницы?
    - ○ Request.Form("UserID")
    - ○ Request.ServerVariables("UserID")
    - ○ Request.QueryString("UserID")
    - ○ Response.QueryString("UserID")
    - ○ Session("UserID")

13. Как правильно включить файл "time.inc" в обрабатываемый файл?
    - ○ <!--#include file="time.inc"-->
    - ○ <% #include file="time.inc" %>
    - ○ <% include file="time.inc" %>
    - ○ <include file="time.inc">

14. Сценарии ASP ограничиваются разделителями. Какими?
    - ○ <%>...</%>
    - ○ <script>...</script>
    - ○ <%...%>
    - ○ <&>...</&>

Студент: _____

БИЛЕТ №5

1. Выберите правильные утверждения:
   - ☐ Несколько подряд идущих пробелов интерпретируются браузером как один пробел, даже если они окружены тегом <PRE>
   - ☐ Использование в названии тега символов кириллицы недопустимо, даже если начертание символов совпадает с латиницей
   - ☐ Заглавные и строчные буквы в названиях тегов интерпретируются браузером одинаково
   - ☐ Между названием тега и параметром тега пробел необязателен
   - ☐ Между символом "<" и названием тега пробел недопустим
   - ☐ Любой параметр, указанный в кавычках, можно с таким же успехом указать и без кавычек

2. Для каких тегов не существует закрывающих тегов в стандарте HTML 4.0?
   - ☐ <TABLE>
   - ☐ <IMG>
   - ☐ <A>
   - ☐
   - ☐ <H1>
   - ☐ <HR>
   - ☐ <FONT>

3. Как можно задать шрифт для элемента списка формы?
   - ○ <OPTION ....><FONT ...>ТЕКСТ</FONT></OPTION>
   - ○ <FONT ...><OPTION ....>ТЕКСТ</OPTION></FONT>
   - ○ Это можно сделать только при помощи CSS

4. Как можно при открытии страницы выполнить автоматическую переадресацию на другой URL?
   - ○ это невозможно
   - ○ с помощью тега <A>
   - ○ с помощью тега <META>
   - ○ с помощью тега <LINK>

5. Какому значению параметра SIZE в теге FONT соответствует размер текста по умолчанию?
   - ○ <FONT SIZE=2>
   - ○ <FONT SIZE=3>
   - ○ <FONT SIZE=4>
   - ○ <FONT SIZE=5>

6. Выберите те строки примера, которые содержат ошибки:
   - ○ строка 1: <STYLE TYPE = CSS/TEXT>
   - ○ строка 2: <!--
   - ○ строка 3: P {FONT FACE: ARIAL}
   - ○ строка 4: DIV:HOVER { TEXT-DECORATION: UNDERLINE}
   - ○ строка 5: DIV {MARGIN: 0 0 10 0; PADDING:10}
   - ○ строка 6: -->
   - ○ строка 7: </STYLE>

7. В разделе <STYLE> заданы правила, приведенные в примере. Какие свойства наследует тег <FONT> при использовании далее в конструкции <TABLE><TR><TD><P><FONT>?

```
P {COLOR: RED}
TD {FONT-WEIGHT: BOLD}
P FONT {FONT-SIZE: 20PX}
```

   - ○ {COLOR: RED}
   - ○ {COLOR: RED; FONT-WEIGHT: BOLD}
   - ○ {COLOR: RED; FONT-SIZE: 20PX}
   - ○ {FONT-WEIGHT: BOLD; FONT-SIZE: 20PX}
   - ○ {COLOR: RED; FONT-WEIGHT: BOLD; FONT-SIZE: 20PX}

8. Каково будет значение переменной result после выполнения приведенного JavaScript кода?

```
function myFunc()
{
var a = 1;
a++;
return a;
++a;
}
var result = myFunc();
```

- ○ result = 0
- ○ result = 1
- ○ result = 2

9. Какой метод объекта String позволяет проводить поиск символов в строке? (язык JavaScript)
   - ○ search()
   - ○ searchChar()
   - ○ test()
   - ○ indexOf()

10. С помощью какого свойства объекта window можно осуществить ссылку на окно браузера, из которого было открыто текущее окно?
    - ○ self
    - ○ opener
    - ○ main
    - ○ top

11. Для чего используется директива Option Explicit?
    - ○ делает обязательным объявления переменных
    - ○ делает необходимым вызов метода
    - ○ делает необходимым объявления статических переменных
    - ○ делает необходимым объявления всех глобальных переменных

12. При создании ASP страницы решено использовать JScript в качестве языка сценария.
    Какая директива ASP указывает на используемый язык сценария?
    - ○ @ LANG
    - ○ @ LANGUAGE
    - ○ @ LCID
    - ○ @ CODEPAGE
    - ○ @ TRANSACTION

13. Отметьте стандартные объекты ASP?
    - ☐ Server
    - ☐ Response
    - ☐ Session
    - ☐ Cookies

14. С помощью какой коллекции можно получить значения параметров, переданных методом GET ASP странице?
    - ○ Request.QueryString
    - ○ Response.QueryString
    - ○ Request.Form
    - ○ Request.ServerVariables
    - ○ Request.Parametrs

Студент: _____

БИЛЕТ №6

1. Выберите утверждения, которые Вы считаете правильными по отношению к формам в HTML:
   - ☐ Список, созданный тегом <SELECT> и состоящий из нескольких элементов, можно сразу отобразить в развернутом виде и без полосы прокрутки.
   - ☐ В одном HTML документе может быть только одна форма
   - ☐ Кнопка "послать" (<INPUT TYPE=SUBMIT>) позволяет инициировать процесс отправки данных формы.
   - ☐ При использовании метода POST для пересылки данных формы, эти данные передаются в строке адреса
   - ☐ При вводе в поле, созданное тегом <INPUT TYPE=HIDDEN>, вводимые символы отображаются в виде звездочек

2. Как правильно вложить одну таблицу внутрь другой таблицы?
   - ○ <TABLE><TR><TD><TABLE><TR><TD>.....
   - ○ <TABLE><TABLE><TR><TD>....
   - ○ <TABLE><TR><TD <TABLE><TR><TD>...
   - ○ ни один из вариантов не является правильным

3. Как проще всего задать вертикальное выравнивание по верхней границе для всех ячеек таблицы, состоящей из 2 рядов и 20 колонок?
   - ○ задать вертикальное выравнивание для каждой ячейки: <TD VALIGN=TOP>
   - ○ задать вертикальное выравнивание для каждого ряда: <TR VALIGN=TOP>
   - ○ задать вертикальное выравнивание для всей таблицы: <TABLE VALIGN=TOP>

4. Как фон страницы сделать непрокручиваемым (для Internet Explorer)?
   - ○ <BODY BGPROPERTIES="FIXED" BACKGROUND="1.JPG">
   - ○ <BODY BACKGROUND="1.JPG">
   - ○ <BODY BACKGROUND="1.JPG" NOSCROLL>
   - ○ сделать это средствами HTML невозможно

5. Что произойдет по щелчку мыши на гиперссылке <A HREF="P1">ТЕКСТ</A>?
   - ○ Будет выполнен переход на документ "P1.HTM"
   - ○ Будет выполнен переход на документ "P1.HTML"
   - ○ Будет выполнен переход на метку "P1" текущего документа
   - ○ Будет выполнен переход на документ "P1"

6. Какая строка примера ошибочна?

```
1. P    {background-image: url(mytile.gif);
2.       background repeat: repeat-x;
3.       background-position: center center}
4. BODY {background-image: url(mytile.gif);
5.       background-repeat: no-repeat}
```

   - ○ 1
   - ○ 3
   - ○ 5
   - ○ 2
   - ○ 4

7. Какое свойство используется для установки интервалов между словами?
   - ○ word-spacing
   - ○ letter-distance
   - ○ word-interval
   - ○ word-distance
   - ○ letter-spacing

8. Каково будет значение переменной x после выполнения приведенного JavaScript кода?

```
y = 0;
++y;
x = --y;
```

- ○ x = 0
- ○ x = 1
- ○ x = 2
- ○ x = undefined

9. Какой оператор используется в JavaScript для объединения строковых значений?
   - ○ &
   - ○ .
   - ○ +
   - ○ @

10. С помощью какого метода объекта window можно переместить окно браузера на несколько пикселей относительно его текущего положения на экране?
    - ○ moveTo()
    - ○ moveBy()
    - ○ resizeBy()
    - ○ resizeTo()

11. Форма документа HTML, содержащая элемент INPUT с именем "FirstName", методом POST передает данные файлу Profile.asp.
    Какой фрагмент кода верно решает поставленную задачу?
    - ○ Request.QueryString("FirstName")
    - ○ Session("FirstName")
    - ○ Request.Form("FirstName")
    - ○ Server.HTMLEncode("FirstName")
    - ○ Response.Form("FirstName")
    - ○ Request.ServerVariables("FirstName")

12. Что происходит, когда пользователь вводит URL, запрашивающий ASP-страницу?
    - ○ Браузер запрашивает ASP-код, сервер возвращает код, браузер обрабатывает код.
    - ○ Браузер запрашивает ASP-код, сервер обрабатывает ASP код и возвращает HTML документ.
    - ○ Браузер запрашивает ASP-код, сервер возвращает код, MS Windows обрабатывает код, так как ASP является технологией Microsoft

13. Укажите местоположение событий Session_onStart и Session_onEnd.
    - ○ В файле Session.asa
    - ○ В .inc файлах VBScript
    - ○ В домашней странице приложения
    - ○ В файле Global.asa
    - ○ В любом ASP файле приложения

14. Где должны располагаться asp файлы на сервере?
    - ○ В папке C:\Inetpub\wwwroot\asp
    - ○ В папке C:\Inetpub\wwwroot\cgi-bin
    - ○ В любой доступной папке IIS сервера
    - ○ В любой папке диска, к которой открыт доступ IIS серверу и даны соответствующие разрешения

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

СОГЛАСОВАНО	**УТВЕРЖДАЮ**
Управление образовательных программ	Проректор по учебной работе
_____/ А.В. Дорошин /	_____/ Ф.В. Гречников /
"_____"_____ 20___ г.	"____" _____ 20___ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Наименование модуля (дисциплины)

Технологии сети Интернет

Цикл, в рамках которого происходит освоение модуля (дисциплины)	Б3. Профессиональный цикл
Часть цикла	Вариативная
Код учебного плана	010300.2.62-2011-О-П-4г00м
Факультет	6
Кафедра	"Программные системы"
Курс	4
Семестр	7
Лекции (СЛ)	36
Семинарские и практические занятия (СП)	0
Лабораторные занятия (СЛР)	36
Контроль самостоятельной работы / Индивидуальные занятия (КСР / ИЗ)	18
Самостоятельная работа (СРС)	18
Всего (Всего с экзаменами)	108

Экзамен

Зачет 7

Согласовано: 110 FAQ

Наименование стандарта, на основании которого составлена рабочая программа:

010300 Фундаментальная информатика и информационные технологии (квалификация (степень) «бакалавр»)

Соответствие содержания рабочей программы, условий ее реализации, материально-технической и учебно-методической обеспеченности учебного процесса по дисциплине всем требованиям государственных стандартов подтверждаем.

Составители:

Котенева Светлана Эдуардовна, ассистент _____ (подпись)

Заведующий кафедрой:

Коварцев Александр Николаевич, д.т.н., профессор _____ (подпись)

Рабочая программа обсуждена на заседании кафедры

"Программные системы"

Протокол №___ от "____" _____ 20___ г.

Наличие основной литературы в фондах научно-технической библиотеки (НТБ) подтверждаем:

Директор НТБ _____ (подпись) /_____/ (расшифровка подписи)

Согласовано:

Декан _____ (подпись) /_____/ (расшифровка подписи)

1 Цели и задачи модуля (дисциплины), требования к уровню освоения содержания

1.1 Перечень развиваемых компетенций

ОК-9, ОК-12, ОК-14 – общекультурные компетенции.
ПК-1, ПК-2, ПК-17, ПК-18, ПК-13, ПК-26 – профессиональные компетенции.

1.2 Цели и задачи изучения модуля (дисциплины)

Дать понимание принципов построения и функционирования сети Интернет, а также базирующихся на ней информационных технологий и сформировать основные навыки создания клиент-серверных Web-приложений.

1.3 Требования к уровню подготовки студента, завершившего изучение данного модуля (дисциплины)

В результате изучения курса студенты должны знать:
- основы программирования на языке JavaScript;
- основы программирования на языке VBScript;
- основы использования технологии ASP;
- основы создания клиент-серверных Web-приложений.

В результате изучения курса студенты должны будут уметь:
- разрабатывать простые и сложные HTML-документы;
- разрабатывать сценарии, исполняемые на стороне клиента, на языке JavaScript;
- программировать серверную часть Web-приложения на языке VBScript сиспользованием технологии ASP.

1.4 Связь с предшествующими модулями (дисциплинами)

Для успешного освоения курса студенты должны изучить следующиедисциплины:
1. Основы программирования.
2. Сети ЭВМ и телекоммуникаций.

1.5 Связь с последующими модулями (дисциплинами)

Знания, полученные студентами при изучении курса «Технологии сети Интернет» будут востребованы при прохождении следующих курсов:
- Программирование сетевых приложений
- НИР бакалавра;
и при подготовке курсовых и выпускных квалификационных работ.

2 Содержание рабочей программы (модуля)

Семестр 1		
СЛ 0,3333 36 часов 0,9999 ЗЕТ	Активные 1	Тема 1. Введение в Интернет-технологии.
		Тема 2. Адресация и именование в сети Интернет.

		Тема 3. Протоколы сети Интернет.
		Тема 4. Язык разметки HTML: теги и атрибуты, списки, таблицы, ссылки, изображения, формы и фреймы.
		Тема 5. Каскадные таблицы стилей (CSS): спецификация каскадных таблиц стилей, основные понятия и определения, создание стилей и классов, принципы каскадирования и принципы группировки.
		Тема 6. Программирование на стороне клиента. Основы JavaScript. Объекты, свойства, методы и коллекции объектов языка JavaScript.
		Тема 7. Объектная модель документа (DOM) и объектная модель браузера (BOM). Обращение к свойствам объектов. Взаимодействие программы на языке JavaScript с объектной моделью документа.
		Тема 8. Основные принципы создания интерактивных Web-приложений. Динамическое обновление полей HTML-форм и их значений. Проверка данных, введенных пользователем в HTML-форму, перед их отправкой.
		Тема 9. Основы VBScript: характеристика, синтаксические конструкции, встроенные функции и объекты языка.
		Тема 10. ASP - технология программирования на стороне сервера: введение в ASP, использование объектов ASP, основные сведения о IIS 5.0, настройка web-сервера MS IIS 5.0, понятие WEB-приложения.
		Тема 11. Создание WEB-интерфейса к базе данных. Основные понятия теории реляционных баз данных. Составление запроса к БД в ASP сценарии.
	Интерактивные 0	
	Традиционные 0	
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	

СЛР 0,3333 36 часов 0,9999 ЗЕТ	Активные 0	
	Интерактивные 1	Лабораторная работа 1. «Исследование FTP-протокола».
		Лабораторная работа 2. «Процедуры Интернет - ping и tracert».
		Лабораторная работа 3. «Создание комплекса простых взаимосвязанных HTML документов».
		Лабораторная работа 4. «Создание внешних каскадных таблиц стилей для html-страниц, созданных в лабораторной работе 1».
		Лабораторная работа 5. «Создание формы ввода данных пользователем и проверка корректности их ввода на стороне клиента».
		Лабораторная работа 6. «Создание Web-интерфейса к базе данных (добавление записей пользователем, отбор по запросу пользователя)».
	Традиционные 0	
КСР 0,1667 18 часов 0,5001 ЗЕТ	Активные 0	
	Интерактивные 1	Выполнение работы в соответствии с заданием.
	Традиционные 0	
СРС 0,1667 18 часов 0,5001 ЗЕТ	Активные 1	Подготовка к лабораторной работе 1.
		Подготовка к лабораторной работе 2.
		Подготовка к лабораторной работе 3.
		Подготовка к лабораторной работе 4.
		Подготовка к лабораторной работе 5.
		Подготовка к лабораторной работе 6.
	Интерактивные 0	
	Традиционные 0	

3 Инновационные методы обучения

1. Для автоматизации учебного процесса преподавания курса «Технологии сети Интернет» на кафедре программных систем используется система дистанционного обучения (СДО), построенная на основе стандарт Moodle, где использовались широкие возможности системы для организации тестирования промежуточных

знаний студентов.
2. Выполнение лабораторных работ с элементами исследования.
3. Использование мультимедиа проектора в процессе чтения лекций сделает представление информации более наглядным; позволит демонстрировать студентам работу программ, сложные диаграммы и схемы; улучшит и облегчит восприятие информации студентами.
5. Использование белой доски и маркеров сухого и влажного стирания.
6. Использование Интернет позволит студентам получить доступ к актуальной справочной информации и стандартам.
7. Использование серверов бесплатного on-line тестирования с большим набором тестов по разным темам и множественным выбором ответов.
8. Выполняемые студентами лабораторные работы взаимосвязаны между собой – каждая последующая основана на предыдущей работе. В результате студенты получает свой собственный небольшой Web-сайт.

4 Технические средства и материальное обеспечение учебного процесса

1. Современный компьютерный класс и доступом к сети Интернет.

5 Учебно-методическое обеспечение

5.1 Основная литература

1. Основы Web-технологий [Текст]: учеб. пособие : [для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов, С. А. Брик, А. М. РусакБ А. И. Сурин. - 2-е изд., испр. - М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний , 2007. -

374 с. (30 экз.)
2. Основы Web-технологий [Текст]: курс лекций : специальность "Интернет-технологии" : [Для вузов по специальности 351400 "Прикладная информатика"] / П. Б. Храмцов [и др.]. - М. : ИНТУИТ. ру, 2003. - 510 с.(8 экз.) 3. Матросов А. HTML 4.0: Наиболее полное руководство. – СПб.: BHV-Петербург, 2005. – 671 с. (5 экз.)
4. Дмитриева, М. JavaScript (Экспресс-курс). – СПб.: BHV-Петербург, 2005. – 328 с. (5 экз.)

5.2 Дополнительная литература

1. Столлингс, Вильям. Компьютерные сети, протоколы и технологии Интернета : [пер. с англ.] / Вильям Столлингс. - СПб. : БХВ-Петербург, 2005. - 817 с. - Библиогр. в конце глав. (1 экз.)
2. Хейз, Дидре. Освой самостоятельно HTML и XHTML [Текст] : [пер. с англ.] / Дидре Хейз. - 3-е изд. - М. [и др.] : Вильямс, 2003, 2004. - 223 с. (5 экз.)
3. Бранденбау, Джерри. JavaScript: сборник рецептов [Текст] : [Пер. с англ.] / Д. Бранденбау. - СПб. [и др.] : Питер, 2000. - 414 с. - (Для профессионалов).(1 экз.)
4. Дунаев, Сергей Б. Технологии Интернет-программирования : [практ. руководство] / С. Б. Дунаев. - СПб. : БХВ-Петербург, 2001. - 472 с. - (Мастер)(2 экз.)
5. Дейтел, Харви М. Как программировать для Internet & W W W [Текст] / Х. М. Дейтел, П. Д. Дейтел, Т. Р. Нието ; пер. с англ. под ред. А. В. Козлова. - М. : Бином, 2002. - 1177 с(5 экз.)

5.3 Электронные источники и интернет ресурсы

1. Электронный каталог НТБ СГАУ (lib.ssau.ru)
2. Научная электронная библиотека eLibrary (www.elibrary.ru)
3. Портал дистанционного обучения кафедры программных систем СГАУ (www.do-ps.ssau.ru)

4. Учебные и справочные материалы WWW-консорциума. Официальный сайт: http://www.w3schools.com.
5. Официальный сайт WWW-консорциума (стандарты и рекомендации): http://www.w3.org.
6. Интернет университет информационных технологий. Официальный сайт: http://www.intuit.ru

5.4 Методические указания и рекомендации

Промежуточный контроль знаний студентов проводят в семестре в виде тестирования и выполнения и отчета студентов по лабораторным работам. Невыполненнение лабораторных работ и не прохождение тестов является основанием для не допуска студента к зачету.
Получение зачета ставится на основании ответов студента на вопросы с учетом оценок за лабораторные работы и проходжение тестов.