

# Дисциплина



**WEB-технологии**

Ст. преп. кафедры ИС

Овчинников Александр Львович

# Введение

## ■ Структура курса:

### 1 семестр:

- Лекции – 34 час.
- Лабораторные работы – 34 час.
- Экзамен

### 2 семестр:

- Курсовой проект.

# Лекционный материал:

- 1. Введение в Web-технологии
  - 1.1. Краткая история создания и организация сети INTERNET
  - 1.2. Краткая история развития World Wide Web (WWW)
  - 1.3. Основные компоненты технологии World Wide Web
  - 1.4. Универсальный локатор ресурсов - URL
  - 1.5. HyperText Transfer Protocol
  - 1.6. CGI
- 2. Web-дизайн. Язык разметки гипертекста HTML
  - 2.1. История развития HTML
  - 2.2. Основы HTML
  - 2.3. Каскадные таблицы стилей CSS. CSS фреймворки
- 3. Web-программирование на стороне клиента. Язык JavaScript
  - 3.1. Использование JavaScript в HTML
  - 3.2. Основные элементы языка JavaScript
  - 3.3. Система событий языка JavaScript
  - 3.4. Объектная модель документа DOM.
  - 3.5. Объектно-ориентированное программирование на JavaScript
  - 3.6. JavaScript фреймворки. Библиотека jQuery.
- 4. Web-программирование на стороне сервера
  - 4.1. Основы построения приложений серверной стороны.
  - 4.2. Языки серверного программирования.
  - 4.3. Язык PHP.
  - 4.4. Взаимодействие WEB-приложения с СУБД.
  - 4.5. Механизм организации сессий.
- 5. Технология AJAX
  - 5.1. Введение в технологию AJAX.
  - 5.2. Форматы передачи данных. Язык XML.
  - 5.3. Реализация асинхронного взаимодействия браузера с сервером.
- 6. Web-порталы
  - 6.1. WEB-сервисы.
  - 6.2. Стандарты и протоколы(SOAP, XML-RPC, REST).
  - 6.3. Системы управления содержимым(CMS) и фреймфорки(CMF).
  - 6.4. Архитектура WEB-приложений MVC.

# Лабораторный практикум:

- **ЛАБОРАТОРНАЯ РАБОТА №1**  
*Исследование возможностей языка разметки гипертекстов HTML и каскадных таблиц стилей CSS.*
- **ЛАБОРАТОРНАЯ РАБОТА №2**  
*Исследование возможностей программирования на стороне клиента. Основы языка JavaScript.*
- **ЛАБОРАТОРНАЯ РАБОТА №3**  
*Работа с деревом документа. Исследование системы событий JavaScript. Стандарт DOM 2.*
- **ЛАБОРАТОРНАЯ РАБОТА №4**  
*Исследование возможностей библиотеки jQuery.*
- **ЛАБОРАТОРНАЯ РАБОТА №5**  
*Основы программирования на стороне сервера. Язык PHP.  
Обработка строк. Обработка данных HTML-форм.*
- **ЛАБОРАТОРНАЯ РАБОТА №6**  
*Исследование средств взаимодействия WEB-приложения с СУБД.*
- **ЛАБОРАТОРНАЯ РАБОТА №7**  
*Изучение механизма сессий в PHP.*
- **ЛАБОРАТОРНАЯ РАБОТА №8**  
*Исследование технологии AJAX.*

# Литература:

- Основы WEB-технологий [Текст] : учеб. пособие для студ. вузов, обуч. по спец. 351400 "Прикладная информатика" / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин. - 2-е изд., испр. - М. : ИНТУИТ : БИНОМ. ЛЗ, 2007. - 374 с.
- Кожемякин, А. А. HTML и CSS в примерах. Создание WEB-страниц [Текст] / А. А. Кожемякин. - М.
- Коэн, Л. И. Полный справочник по HTML, CSS и JavaScript [Текст] : справочник / Л. И. Коэн, Дж. И. Коэн. - М. : ЭКОМ, 2007. - 1168 с.
- Вайк, А. Р. JavaScript. Полное руководство [Текст] : пер. с англ. / А. Вайк, Дж. Джиллиам. - 4-е изд. - М. ; СПб. ; К. : Вильямс, 2004. - 720 с.

# Литература:

- Пауэлл, Т. Полный справочник по JavaScript [Текст] : пер. с англ. / Т. Пауэлл, Ф. Шнайдер. - 2-е изд. - М. ; СПб. ; К. : Вильямс, 2006. - 960 с.
- Будилов, В. А. JavaScript, XML и объектная модель документа [Текст] / В. А. Будилов. - СПб. : Наука и техника, 2001.
- Климов, А. П. JavaScript на примерах [Текст] / А. П. Климов. - 2-е изд., доп. и перераб. - СПб. : БХВ - Петербург, 2009. - 328 с.
- Колисниченко, Д. Н. PHP 5/6 и MySQL 6: разработка Web-приложений [Текст] / Д. Н. Колисниченко. - 2-е изд. - СПб. : БХВ - Петербург, 2010. - 546 с.
- Веллинг, Л. Разработка Web-приложений с помощью PHP и MySQL [Текст] / Л. Веллинг, Л. Томсон. - 3-е изд. - М. ; СПб. ; К. : Вильямс, 2005.

# Литература:

- Гетланд, Дж. Прагматика Ajax [Текст] : [пер. с англ.?] / Дж. Гетланд, Б. Гэлбрайт, Д. Алмаэр. - М. : ЛОРИ, 2008. - 342 с.
- Бенкен, Е. PHP, MySQL, XML программирование для Интернета [Текст] / Е. Бенкен. - СПб. : БХВ - Петербург, 2007.

# Введение в Web-технологии

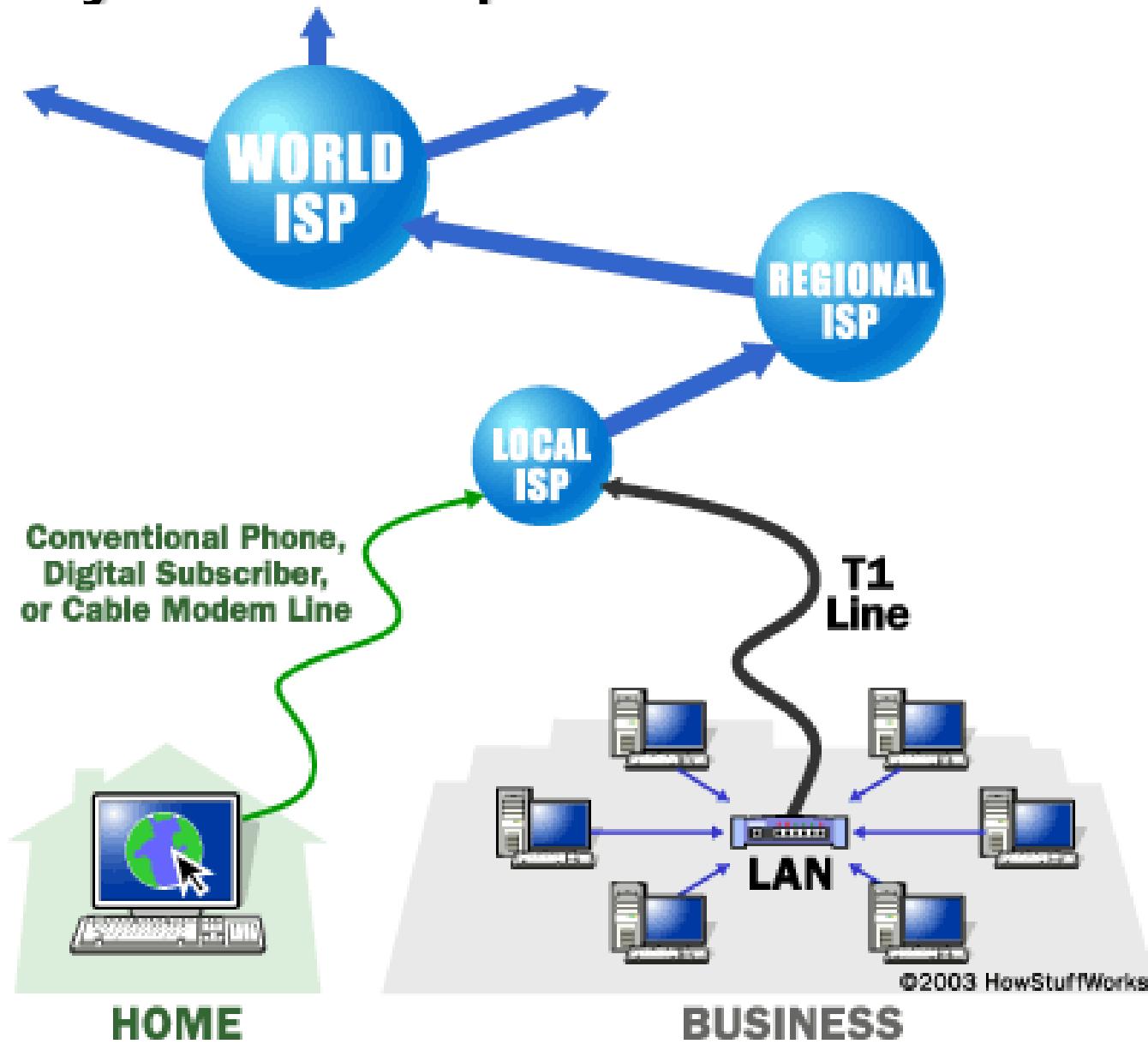
## Краткая история развития Интернет

- **1960 ARPA** - *U.S Defense Department's Advanced Research Projects Agency* - Агентства Перспективных Проектов Исследований Министерства Обороны США.
- **70-80 NSF** - *National Science Foundation* - Национальный Научный Фонд – сеть основанная на IP технологии

**Internet** - гиперсеть, состоящая из компьютерных сетей.

- Доступ в Internet, обычно, получают через поставщиков услуг (*Internet service provider*) ISP.

# Доступ к Интернет



# Краткая история развития World Wide Web (WWW)

## ■ Internet:

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

1989 год. Тим Бернерс-Ли предложил руководству Европейской организации по ядерным исследованиям (CERN) проект распределенной гипертекстовой системы, которую он назвал

**World Wide Web (WWW) ( Всемирная паутина)**

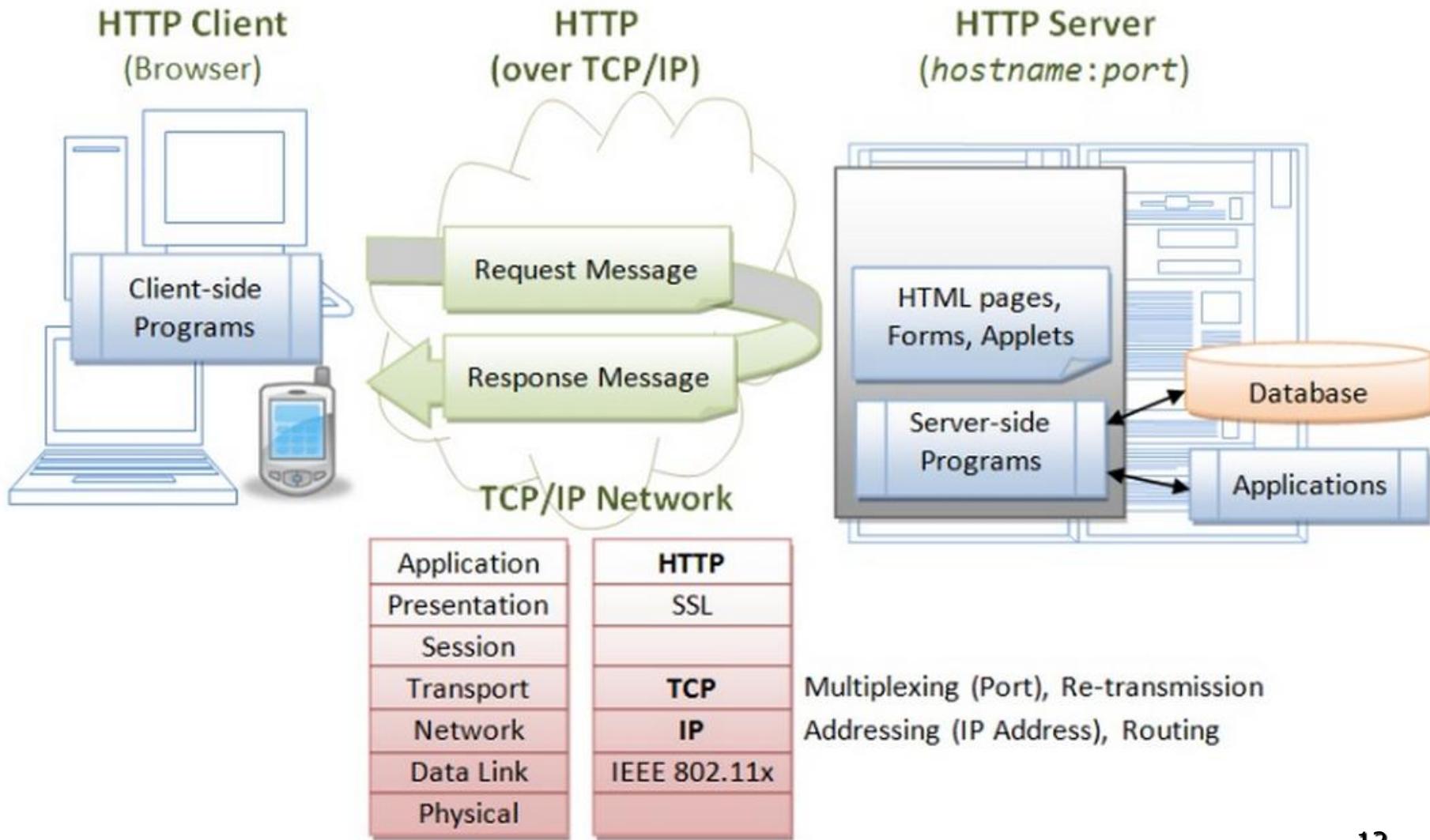
Успех технологии World Wide Web определен двумя основными факторами:

- + простота;
- + использование протоколов межсетевого обмена семейства TCP/IP - Transmission Control Protocol(протокол управления передачей)/Internet Protocol(протокол Internet);

# Основные составляющие WWW:

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

# Обобщенная схема взаимодействия по протоколу HTTP:





# Браузер:

**Браузер** - **прикладное программное обеспечение** для просмотра запроса, обработки, манипулирования и отображения содержания веб-сайтов.

Браузеры также могут использоваться для просмотра содержания файлов многих графических форматов (gif, jpeg, png, svg), аудио-видео форматов (mp3, mpeg), текстовых форматов (pdf, djvu) и других файлов.

Первый веб-браузер был создан в 1990 году Тимом Бернерсом-Ли. Он назывался WorldWideWeb и позже был переименован в Nexus. Но первым распространённым браузером с графическим интерфейсом был NCSA Mosaic. (Netscape Navigator и Internet Explorer взяли его за основу).

**Браузерный движок** (англ. layout engine) — программное обеспечение, преобразующее содержимое веб-страниц (файлы HTML, XML, цифровые изображения и т. д.) и информацию о форматировании (в форматах CSS, XSL и т. д.) в интерактивное изображение форматированного содержимого на экране.



## Браузерные движки:

В число наиболее распространённых движков входят следующие:

**Trident** — проприетарный движок Microsoft Internet Explorer; используется многими программами для Microsoft Windows (например, мини-браузерами в программах Winamp и RealPlayer).

**Gecko** — открытый движок проекта Mozilla; используется в большом числе программ, основанных на коде Mozilla (браузере Firefox, почтовом клиенте Thunderbird, наборе программ SeaMonkey).

**KHTML** — разработан в рамках проекта KDE, используется в браузере Konqueror и послужил основой для WebKit.

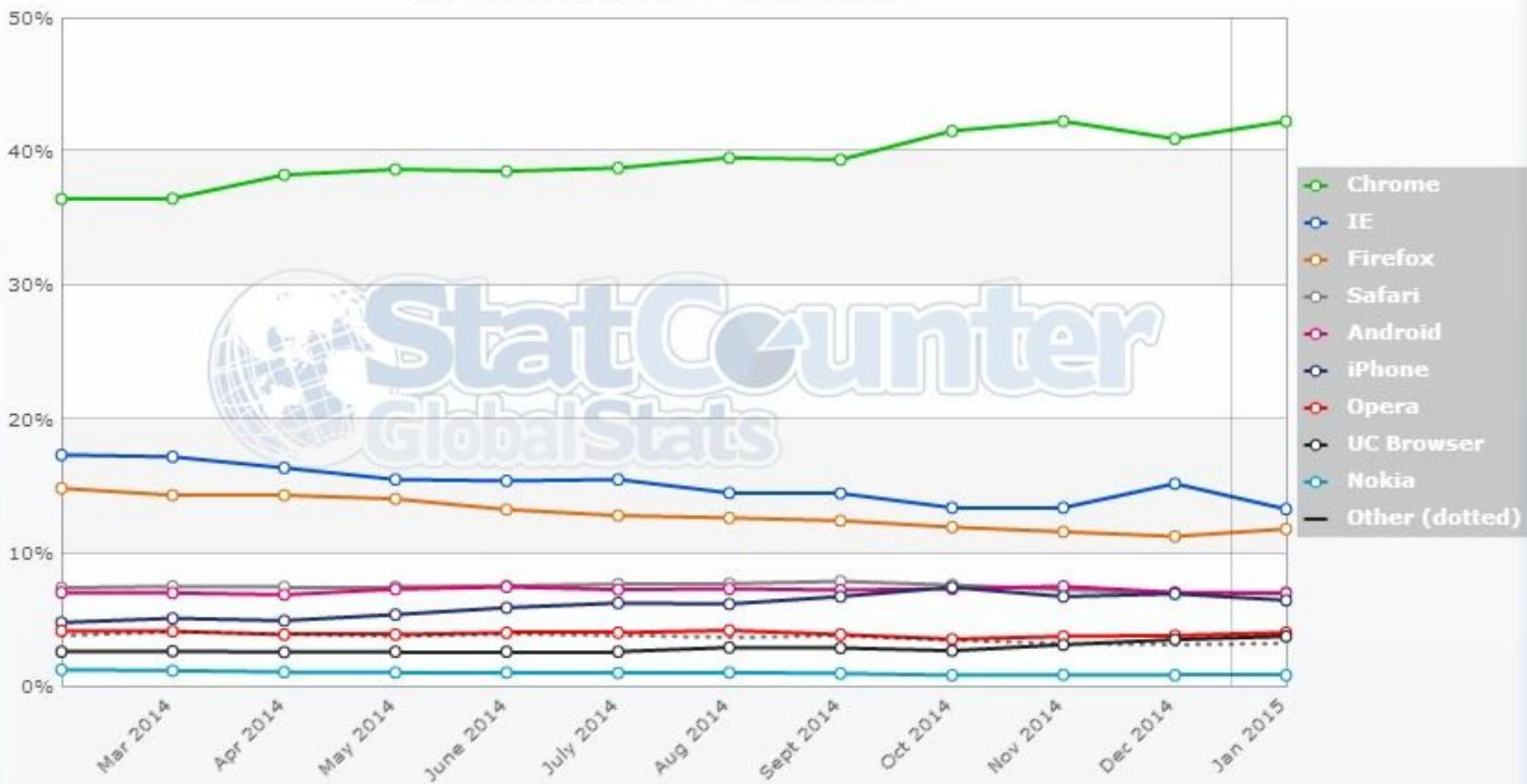
**WebKit** — движок для браузера Apple Safari, включенного в операционную систему Mac OS X, и браузера Google Chrome. Встроен в библиотеку Qt.

**Presto** — проприетарный движок, разработанный Opera Software; являлся базой для браузера Opera до перехода на Blink, а также лицензирован для использования рядом сторонних компаний.

**Blink** - движок браузера Google Chrome с 28 версии и Opera 15. Является переработанным WebKit.

## StatCounter Global Stats

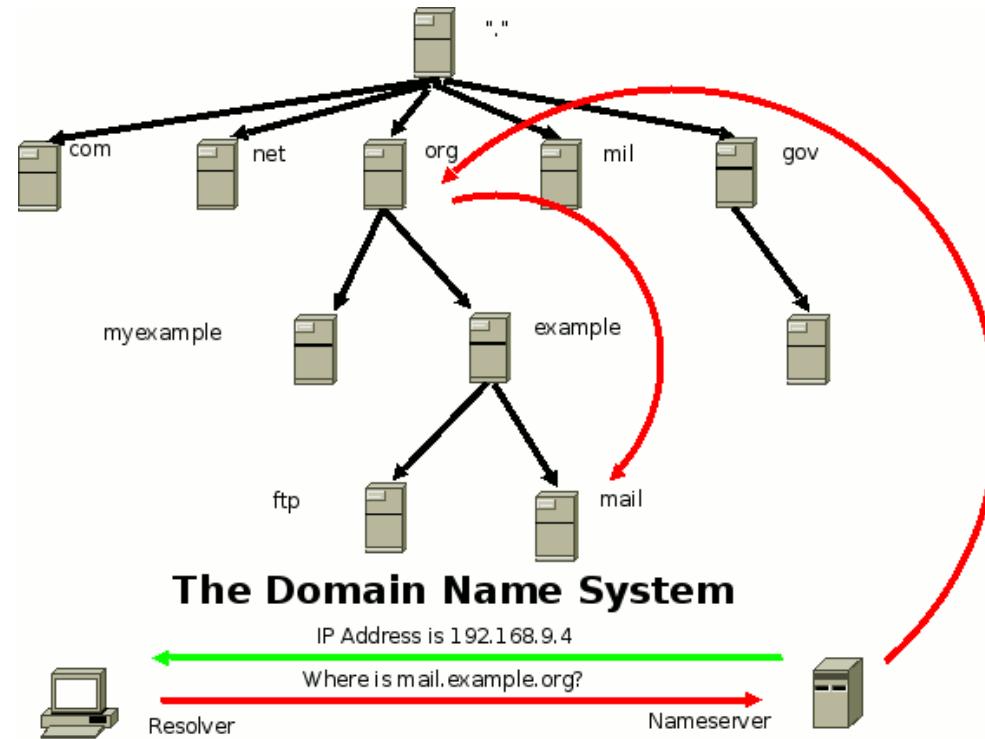
Top 9 Browsers from Feb 2014 to Jan 2015



# Служба DNS:

DNS (англ. Domain Name System — система доменных имен) — распределённая система для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почтовых сообщений, обслуживающих узлах для различных протоколов в домене.

Основой DNS является представление об иерархической структуре **доменного имени** и **зонах**. Каждый сервер, отвечающий за имя, может делегировать ответственность за дальнейшую часть домена другому серверу.



# IP-адрес:

IP-адрес - уникальный сетевой адрес узла в компьютерной сети, построенной по протоколу IP. В сети Интернет требуется глобальная уникальность адреса; в случае работы в локальной сети требуется уникальность адреса в пределах сети.

В версии протокола IPv4 IP-адрес имеет длину 4 байта, в IPv6 – 16 байт.

В 4-й версии IP-адрес представляет собой 32-битовое число. Удобной формой записи IP-адреса (IPv4) является запись в виде четырёх десятичных чисел значением от 0 до 255, разделённых точками, например, 192.0.2.60.

Адреса, используемые в локальных сетях, относят к частным. К частным относятся IP-адреса из следующих подсетей:

10.0.0.0/8

172.16.0.0/12

192.168.0.0/16

В 6-й версии IP-адрес (IPv6) является 128-битовым.

Внутри адреса разделителем является двоеточие  
2001:0db8:85a3:0000:0000:8a2e:0370:7334

# Веб сервер:

**Веб-сервер** – сервер/программное обеспечение, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. (Веб-сервером может называться как программное обеспечение, так и непосредственно компьютер (аппаратное обеспечение), на котором это программное обеспечение работает).

Наиболее распространённые веб-серверы:

- **Apache** — разрабатывается и поддерживается открытым сообществом разработчиков под эгидой Apache Software Foundation, является кроссплатформенным ПО, поддерживает операционные системы Linux, BSD, Mac OS, Microsoft Windows, Novell NetWare, BeOS;
- **IIS** от компании Microsoft, распространяемый с ОС семейства Windows
- **nginx** — веб-сервер и почтовый прокси-сервер, работающий на Unix-подобных операционных системах. Часто используется для отдачи статического содержимого веб-сайтов.
- **lighttpd** — использует так называемую асинхронную обработку сетевых соединений. Благодаря этому загруженность сервера (в отличие от Apache) при доступе к файлам на диске не зависит от количества текущих соединений.

# Веб сервер:

**Веб-служба, веб-сервис** (англ. web service) — идентифицируемая веб-адресом программная система со стандартизованными интерфейсами.

Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на различных **протоколах** (SOAP(протокол обмена сообщениями на базе XML), XML-RPC(стандарт/протокол вызова удалённых процедур), JSON-RPC, REST и т. д.).

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

Сéрвис-ориентíрованная архитектúра (SOA, англ. service-oriented architecture) — модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных (англ. loose coupling) заменяемых компонентов, оснащённых стандартизованными интерфейсами для взаимодействия по стандартизованным протоколам.

# URL:

- **METHOD://SERVERNAME:PORT/PATHNAME#ANCHOR**
- **METHOD** определяет тип операцию, которая будет выполняться при интерпретации данного URL. Наиболее часто используемые методы:
  - **file** - чтение файла с локального диска. Например:  
file://home/alex/index.html
  - **http** - доступ к WEB-странице в сети с использованием HTTP-протокола.
  - **ftp** - запрос файла с анонимного FTP-сервера. Например:  
ftp://hostname/directory/filename;
  - **mailto** - активизирует почтовую сессию с указанным пользователем и хостом. Например: mailto: webmaster@is.sevntu.sebastopol.ua
- **SERVERNAME** - необязательный параметр, описывающий полное сетевое имя машины.
- **PORT** - номер порта TCP, на котором функционирует WEB-сервер.
- **PATHNAME** - частичный или полный путь к документу, который должен вызваться в результате интерпретации URL.
- **#ANCHOR** - данный элемент является ссылкой на строку (точку) внутри HTML-документа.

# **Протокол HTTP (HyperText Transfer Protocol)**

**Все данные в рамках Web-технологии передаются по протоколу HTTP.** Исключение может составлять обмен с использованием программирования на Java(с использованием специализированных протоколов) или обмен из Plugin-приложений.

**При более подробном рассмотрении протокола HTTP остановимся на таких вопросах, как:**

- общая структура сообщений;
- методы доступа;
- оптимизация обменов.

## **Общая структура сообщений**

**HTTP** — это протокол прикладного уровня. Он ориентирован на модель обмена "клиент-сервер". **Клиент и сервер обмениваются фрагментами данных**, которые называются **HTTP-сообщениями**.

Сообщения, отправляемые клиентом серверу, называют **запросами**, а сообщения, отправляемые сервером клиенту — **откликами**.

Сообщение может состоять из **двух частей: заголовка и тела**. Тело от заголовка отделяется пустой строкой.

## Структура HTTP-запроса.

HTTP-запрос состоит из **заголовка запроса** и **тела запроса**, разделенных пустой строкой.

**Тело запроса может отсутствовать.**

**Заголовок запроса состоит из главной (первой) строки запроса и последующих строк, уточняющих запрос в главной строке.**

**Последующие строки также могут отсутствовать.**

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

1. *Метод;*
2. *Ресурс;*
3. *Версия протокола.*

1. *Метод* (иначе говоря, команда HTTP) – GET, HEAD, POST, PUT, DELETE:

- **GET** - запрос документа. Наиболее часто употребляемый метод; в ранних версиях HTTP он был единственным. Метод GET применяется клиентом при запросе к серверу по умолчанию. В этом случае **клиент сообщает адрес** ресурса (URL), который он хочет получить, **версию протокола HTTP**, поддерживаемые им MIME-типы документов, версию и название клиентского программного обеспечения. Все эти параметры указываются в заголовке HTTP-запроса. **Тело в запросе не передается**. В ответ сервер сообщает версию HTTP-протокола, код возврата, тип содержания тела сообщения, размер тела сообщения и ряд других необязательных директив HTTP-заголовка. **В теле отклика обычно передается сам запрашиваемый ресурс**(HTML-страница).

- **HEAD** - Метод HEAD **используется для уменьшения обменов** при работе по протоколу HTTP. Он аналогичен методу GET за исключением того, что **в отклике тело сообщения не передается**. Данный метод используется для проверки времени последней модификации ресурса и срока годности кэшированных ресурсов, а также при использовании программ сканирования ресурсов World Wide Web. Одним словом, метод HEAD предназначен для уменьшения объема передаваемой по сети информации в рамках HTTP-обмена..
- **POST** - это альтернатива методу GET. При обмене данными по методу POST **в запросе клиента присутствует тело HTTP-сообщения**. Это **тело может формироваться из данных, которые вводятся в HTML-форме**, или из присоединенного внешнего файла. **В отклике**, как правило, **присутствует и заголовок, и тело HTTP-сообщения**. Т.о. этот метод **применяется для передачи данных CGI-скриптам**. Сами данные следуют в последующих строках запроса в виде параметров.
- **PUT** - Применяется для **загрузки содержимого запроса** на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content).  
При использовании REST API - POST предполагает создание объекта, а PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу и осуществляется модификация.
- **DELETE** – предполагает удаление указанного в URI ресурса.

- 2. Ресурс** - это путь к определенному файлу на сервере, который клиент хочет получить (или создать, обновить или удалить). 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.
- 3. Версия протокола** - версия протокола HTTP, с которой работает клиентская программа.

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

Некоторые наиболее часто используемые 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: *User-Agent:Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.76 Safari/537.36*
- **Accept** - **список поддерживаемых браузером типов содержимого** в порядке их предпочтения данным браузером, например: *Accept:text/html, application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8*. 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.

## 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.
- **Код ошибки** - кодовое обозначение "успешности" выполнения запроса. Код 200 означает "все нормально" (OK).
- **Словесное описание ошибки** - "расшифровка" предыдущего кода. Например, для 200 это OK, для 500 - Internal Server Error.

## Коды ошибок HTTP

1xx: Informational (информационные):

100 Continue («продолжить»).

...

2xx: Success (успешно):

200 OK («хорошо»).

201 Created («создано»).

202 Accepted («принято»).

...

3xx: Redirection (перенаправление):

300 Multiple Choices («множество выборов»).

301 Moved Permanently («перемещено навсегда»).

302 Moved Temporarily («перемещено временно»).

4xx: Client Error (ошибка клиента):

400 Bad Request («плохой, неверный запрос»).

401 Unauthorized («неавторизован»).

402 Payment Required («необходима оплата»).

403 Forbidden («запрещено»).

404 Not Found («не найдено»).

...

5xx: Server Error (ошибка сервера):

500 Internal Server Error («внутренняя ошибка сервера»).

501 Not Implemented («не реализовано»).

502 Bad Gateway («плохой, ошибочный шлюз»).

503 Service Unavailable («сервис недоступен») [1][3].

504 Gateway Timeout («шлюз не отвечает») [1][3].

Наиболее часто используемые параметры http-ответа:

**Connection** - аналогичен соответствующему параметру запроса. Если сервер не поддерживает Keep-Alive (есть и такие), то значение Connection в ответе всегда close.

**Content-Type** ("тип содержимого") - содержит обозначение типа содержимого ответа.

В зависимости от значения Content-Type браузер воспринимает ответ как HTML-страницу, картинку gif или jpeg, как файл, который надо сохранить на диске. 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;
- application/octet-stream - поток "октетов" (т.е. просто байт) для записи на диск.

**Content-Length** ("длина содержимого") - длина содержимого ответа в байтах.

**Last-Modified** ("Модифицирован в последний раз") - дата последнего изменения документа.

**Пример HTTP-запроса:**

GET / HTTP/1.1  
Host: ya.ru  
Connection: keep-alive  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,  
image/webp,\*/\*;q=0.8  
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/40.0.2214.111 Safari/537.36  
Accept-Encoding: gzip, deflate, sdch  
Accept-Language: ru,en-US;q=0.8,en;q=0.6  
Cookie: yandexuid=3758108361330076137; yp=1362079898.ygu.1

**Пример HTTP-ответа:**

HTTP/1.1 200 Ok  
Server: nginx  
Date: Thu, 12 Feb 2015 20:36:56 GMT  
Content-Type: text/html; charset=UTF-8  
Transfer-Encoding: chunked  
Connection: close  
Cache-Control: no-cache,no-store,max-age=0,must-revalidate  
Expires: Thu, 12 Feb 2015 20:36:56 GMT Last-Modified: Thu, 12 Feb 2015 20:36:56  
GMT  
Content-Encoding: gzip  
*пустая строка*  
<HTML>  
...  
</HTML>

## Оптимизация обменов

Протокол HTTP изначально не был ориентирован на постоянное соединение. Для оптимизации числа открытых TCP-соединений в HTTP-протоколе версий 1.0 и 1.1 предусмотрен режим keep-alive. В этом режиме соединение инициализируется только один раз, и по нему последовательно можно реализовать несколько HTTP-обменов.

```
GET /conf-2009.avi HTTP/1.0
Host: example.org
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
Range: bytes=88080384-
Referer: http://example.org/
```

```
HTTP/1.1 206 Partial Content
Date: Thu, 19 Feb 2009 12:27:08 GMT
Server: Apache/2.2.3
Last-Modified: Wed, 18 Jun 2003 16:05:58 GMT
ETag: "56d-9989200-1132c580"
Accept-Ranges: bytes
Content-Range: bytes 88080384-160993791/160993792
Content-Length: 72913408
Connection: close
Content-Type: video/x-msvideo
(пустая строка)
(двоичное содержимое от 84-го мегабайта)
```

Куки (от англ. cookie — печенье) — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) при попытке открыть страницу соответствующего сайта пересыпает этот фрагмент данных веб-серверу в составе HTTP-запроса.

Используются для:

- аутентификации пользователя;
- хранения персональных данных и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

GET /index.html HTTP/1.1

Host: [www.example.org](http://www.example.org)

HTTP/1.1 200 OK

Content-type: text/html

Set-Cookie: name=value

(содержимое страницы)

GET /spec.html HTTP/1.1

Host: www.example.org

Cookie: name=value

Последний запрос отличается от первого тем, что содержит строку, которую сервер отправил браузеру ранее. Таким образом, сервер узнаёт, что этот запрос связан с предыдущим.

# CGI

В 1991 году специалисты NCSA разработали и реализовали две взаимосвязанные спецификации: HTML-формы и Common Gateway Interface(CGI).

Common Gateway Interface — это спецификация обмена данными между прикладной программой, выполняемой по запросу пользователя, и HTTP-сервером, который данную программу запускает.

Обмен данными в Web-технологии подразделяется в соответствии с типами методов доступа протокола HTTP и видами запросов в спецификации CGI.

В CGI имеет смысл выделить следующие основные моменты:

- понятие CGI-скрипта;
- типы запросов;
- механизмы приема данных скриптом;
- механизм генерации отклика скриптом.

Основное назначение CGI — обработка данных из HTML-форм. В настоящее время область применения CGI гораздо шире.

## **Понятие CGI-скрипта**

CGI-скриптом называют программу, написанную на любом языке программирования или командном языке, которая осуществляет обмен данными с HTTP-сервером в соответствии со спецификацией Common Gateway Interface.

Скрипты могут быть разработаны на различных языках программирования, например, Perl и C.

## CGI. Типы запросов

В основном используются два типа HTTP-запросов к CGI-скриптам: используя метод **GET** и используя метод **POST**. В свою очередь, запросы, использующие метод **GET**, подразделяются на запросы по типам кодирования: **isindex** и **form-urlencoded**, а запросы, использующие метод **POST** – **multipart/form-data** и **form-urlencoded**.

В запросах, использующих метод **GET** данные от клиента передаются скрипту в переменной окружения **QUERY\_STRING**.

Запрос типа **ISINDEX** – это запрос вида:

`http://webserver.com/something-cgi/cgi-script?слово1+слово2+слово3`

Запрос типа **FORM-URLENCODED** – это запрос вида:

`http://webserver.com/something-cgi/cgi-script?field=word1&field2=word2`

При обращении к скрипту по методу **POST** данные после символа "?" не будут размещаться в **QUERY\_STRING**, а будут направлены в поток стандартного ввода скрипта (и в зависимости от установок Web-сервера в переменные окружения). В этом случае количество символов в потоке стандартного ввода скрипта будет указано в переменной окружения **CONTENT\_LENGTH**.

При запросе типа **multipart/form-data** применяется составное тело HTTP-сообщения, которое представляет собой данные, введенные в форме, и данные присоединенного внешнего файла. Это тело помещается в поток стандартного ввода скрипта (и в зависимости от установок Web-сервера в переменные окружения). При этом к данным формы применяется кодирование как в **form-urlencoded**, а данные внешнего файла передаются как есть.

Метод		Клиент --> Сервер	Клиент <-- Сервер
GET	По умолчанию	Только HTTP-заголовок	HTTP-заголовок и страница, как тело HTTP-сообщения
	isindex	Только HTTP-заголовок (список ключевых слов включен в URL. Слова разделены символом "+". Кодирование кириллицы не производится)	HTTP-заголовок и страница, как тело HTTP-сообщения
	form-urlencoded	Только HTTP-заголовок (данные из формы включены в URL страницы. Производится кодирование специальных символов и кириллицы) HTTP-сообщения	HTTP-заголовок и страница, как тело HTTP-сообщения
POST	form-urlencoded	Только HTTP-заголовок (данные из формы включены в URL страницы. Производится кодирование специальных символов и кириллицы) HTTP-сообщения	HTTP-заголовок и страница, как тело HTTP-сообщения
	form-data	HTTP-заголовок и составное тело HTTP-сообщения. Первая часть тела — данные из формы, для которых производится кодирование, вторая часть тела — присоединенный файл как он есть	HTTP-заголовок и страница, как тело HTTP-сообщения

## CGI. Механизмы приема данных скриптом

Скрипт может принять данные от сервера тремя способами:

- через переменные окружения;
- через аргументы командной строки;
- через поток стандартного ввода.

### Переменные окружения

При вызове скрипта сервер выполняет системные вызовы *fork* и *exec*. При этом он создает среду выполнения скрипта, определяя ее переменные. В спецификации CGI определены 22 переменные окружения. При обращении к скрипту разными методами и из различных контекстов реальные значения принимают разные совокупности этих переменных.

Следующие переменные окружения не являются специфичными по типу запросов и устанавливаются для всех запросов:

**SERVER\_SOFTWARE** - название и версия информационного сервера, который отвечает на запрос (и запускает скрипт). Формат: имя/версия.

**SERVER\_NAME** - имя хоста, на котором запущен сервер, DNS имя, или IP адрес в том виде, в котором он представлен в URL.

**GATEWAY\_INTERFACE** - версия CGI спецификации на тот момент, когда компилировался сервер. Формат: CGI/версия .

Следующие переменные окружения являются специфичными для разных запросов, и заполняются перед вызовом скрипта:

**SERVER\_PROTOCOL** - имя и версия информационного протокола, в котором пришел запрос. Формат: протокол/версия.

**SERVER\_PORT** - номер порта, на который был послан запрос.

**REQUEST\_METHOD** - Метод, который был использован для запроса. Для HTTP, это "GET", "HEAD", "POST", и т.д.

**PATH\_INFO** - дополнительная информация о пути, которую передал клиент.

**PATH\_TRANSLATED** - сервер передает преобразованную версию PATH\_INFO, которая включает в себя путь, преобразованный из виртуального в физический.

**SCRIPT\_NAME** - виртуальный путь к скрипту, который должен выполняться.

**QUERY\_STRING** - Информация, следующая за ? в URL, к которому относится данный скрипт. Это информация представляет собой строку запроса.

**REMOTE\_HOST** - имя хоста, производящего запрос. Если сервер не имеет такой информации, он должен установить REMOTE\_ADDR.

**REMOTE\_ADDR** - IP адрес хоста, производящего запрос.

**AUTH\_TYPE** - если сервер поддерживает идентификацию пользователя, и скрипт является защищенным, этот метод идентификации используется для проверки пользователя.

**REMOTE\_USER** - используется в ситуациях, аналогичных предыдущему случаю, для хранения имени пользователя.

**REMOTE\_IDENT** - если HTTP сервер поддерживает идентификацию пользователя согласно RFC 931, то эта переменная будет содержать имя пользователя, полученное от сервера.

**CONTENT\_TYPE** - для запросов, которые содержат дополнительную добавочную информацию, такие как HTTP POST и PUT, здесь содержится тип данных этой информации.

**CONTENT\_LENGTH** - Длина данных, которую передает клиент.

В дополнение к этим, если запрос содержит дополнительные поля заголовка запроса, они помещаются в переменные окружения с префиксом `HTTP_`, за которым следует имя заголовка. Примером такой переменной может служить переменная `HTTP_ACCEPT`, которая была определена в спецификации CGI/1.0. Другим примером может служить заголовок `User-Agent`:

**`HTTP_ACCEPT`** - список MIME типов, которые клиент может обработать, как задано в HTTP заголовках. Другие протоколы должны получить эту информацию из других мест (если она им необходима). Каждый тип в этом списке должен быть отделен запятой согласно HTTP спецификации. Формат: тип/подтип, тип/подтип

**`HTTP_USER_AGENT`** - браузер, который использует клиент для посылки запроса. Общий формат: программа/версия библиотека/версия.

Следующие переменные окружения не являются специфичными по типу запросов и устанавливаются для всех запросов:

**`SERVER_SOFTWARE`** - название и версия информационного сервера, который отвечает на запрос (и запускает скрипт). Формат: имя/версия.

**`SERVER_NAME`** - имя хоста, на котором запущен сервер, DNS имя, или IP адрес в том виде, в котором он представлен в URL.

**`GATEWAY_INTERFACE`** - версия CGI спецификации на тот момент, когда компилировался сервер. Формат: CGI/версия .

## **Аргументы командной строки**

У CGI-скрипта есть такой элемент операционного окружения как командная строка. Это не обязательно означает, что скрипт реально можно вызвать из командной строки через сервер. Тем не менее получить доступ к содержанию командной строки скрипта можно с помощью тех же функций, что и при вызове его из-под интерактивной оболочки

```
// C
void main(argc,argv)
int argc;
char *argv[];
{
int i;
for(i=0;i<argc;i++){
    printf("%s\n",argv[i]);
}
}
```

## **Поток стандартного ввода**

Ввод данных в скрипт через поток стандартного ввода осуществляется только при использовании метода доступа к ресурсу (скрипту) POST. При этом в переменную окружения CONTENT\_LENGTH помещается число символов(байт), которое необходимо считать из потока стандартного ввода скрипта, а в переменную окружения CONTENT\_TYPE помещается тип кодирования данных, которые считаются из потока стандартного ввода.

При посимвольном считывании в C можно применить, например, такой фрагмент кода:

```
int n;
char *buf;
n= atoi(getenv("CONTENT_LENGTH"));
buf = (char *) malloc(n+1);
memset(buf, '\000', n+1);
for(i=0;i<n;i++){
    buf[i]=getchar()
}
free(buf);
```

Следующие переменные окружения не являются специфичными по типу запросов и устанавливаются для всех запросов:

**SERVER\_SOFTWARE** - название и версия информационного сервера, который отвечает на запрос (и запускает скрипт). Формат: имя/версия.

**SERVER\_NAME** - имя хоста, на котором запущен сервер, DNS имя, или IP адрес в том виде, в котором он представлен в URL.

**GATEWAY\_INTERFACE** - версия CGI спецификации на тот момент, когда компилировался сервер. Формат: CGI/версия .

## **Механизм генерации отклика скриптом**

**Существует только один способ вернуть данные серверу и, соответственно, браузеру пользователя — писать в поток стандартного вывода (**STDOUT**).** При этом скрипт должен формировать HTTP-сообщение.

Сначала выводятся директивы HTTP-заголовка. В минимальном варианте это либо

*Content-type: text/html,*  
либо

*Location: http://server.com/*

(В первом случае определяется тип тела HTTP-сообщения, а во втором осуществляется перенаправление запроса).

После заголовка генерируется отклик в виде тела HTTP-сообщения, которое должно быть отделено от заголовка пустой строкой:

```
#!/bin/sh
echo Content-type: text/plain
echo
echo Hello
```

(В данном случае используется командный интерпретатор *sh*)

Если скрипт начинает формирование заголовка с директивы версии HTTP-протокола, то сервер не анализирует отклик и передает его как есть. Если в заголовке, сгенерированном скриптом, эта директива отсутствует, то сервер считает, что заголовок неполный, и вставляет в него дополнительные директивы.

## ***Fast CGI***

Дальнейшим развитием **CGI** стал интерфейс **FastCGI**. По сравнению с CGI является более производительным и безопасным.

FastCGI-процессы используют Unix Domain Sockets или TCP/IP для связи с сервером. Это даёт следующее преимущество над обычными CGI-программами: FastCGI-программы могут быть запущены не только на этом же сервере, но и на других компьютерах в сети. Также возможна обработка запросов несколькими FastCGI-процессами, работающими параллельно.

FastCGI-поддерживается:

- Apache(частично) - используются сторонние модули mod\_fastcgi или mod\_fcgid
- Lighttpd
- Microsoft IIS
- Nginx
- ...