

Лекция 2. Взаимодействие клиент-сервер в WWW

Клиент-серверные технологии интернет

- Основой протокола *HTTP* является взаимодействие «*клиент-сервер*», то есть предполагается, что:
 - потребитель-клиент инициировав соединение с поставщиком-сервером посылает ему запрос;
 - Поставщик-сервер, получив запрос, производит необходимые действия и возвращает обратно клиенту ответ с результатом.
- *Тонкий клиент* — это компьютер-клиент, который переносит все задачи по обработке информации на сервер. Примером тонкого клиента может служить компьютер с *браузером*, использующийся для работы с *веб-приложениями*.
- *Толстый клиент*, напротив, производит обработку информации независимо от сервера, использует последний в основном лишь для хранения данных.



Протокол http

- *HTTP* (*HyperText Transfer Protocol* - [RFC 1945](#)[RFC 1945](#), [RFC 2616](#)) — протокол прикладного уровня для передачи гипертекста.
- Центральным объектом в *HTTP* является ресурс, на который указывает *URI* в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы. Особенностью протокола *HTTP* является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя изначально данный протокол предназначен для передачи символьной информации.

Протокол http

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



Протокол http

- Всё программное обеспечение для работы с протоколом HTTP разделяется на три основные категории:
 - Серверы - поставщики услуг хранения и обработки информации (обработка запросов).
 - Клиенты — конечные потребители услуг сервера (отправка запросов).
 - Прокси-серверы для поддержки работы транспортных служб.



Протокол http

- Основными *клиентами* являются *браузеры* например: *Internet Explorer, Opera, Mozilla Firefox, Netscape Navigator* и др.
- Наиболее известными реализациями *веб-серверов* являются: *Internet Information Services (IIS), Apache, lighttpd, nginx*.
- Наиболее известные реализации *прокси-серверов*: *Squid, UserGate, Multiproxy, Naviscope*.



"Классическая" схема НТТР-сеанса

1. Установление TCP-соединения.
 2. Запрос клиента.
 3. Ответ сервера.
 4. Разрыв TCP-соединения.
- Таким образом, клиент посылает серверу запрос, получает от него ответ, после чего взаимодействие прекращается.
 - Обычно запрос клиента представляет собой требование передать HTML-документ или какой-нибудь другой ресурс, а ответ сервера содержит код этого ресурса.

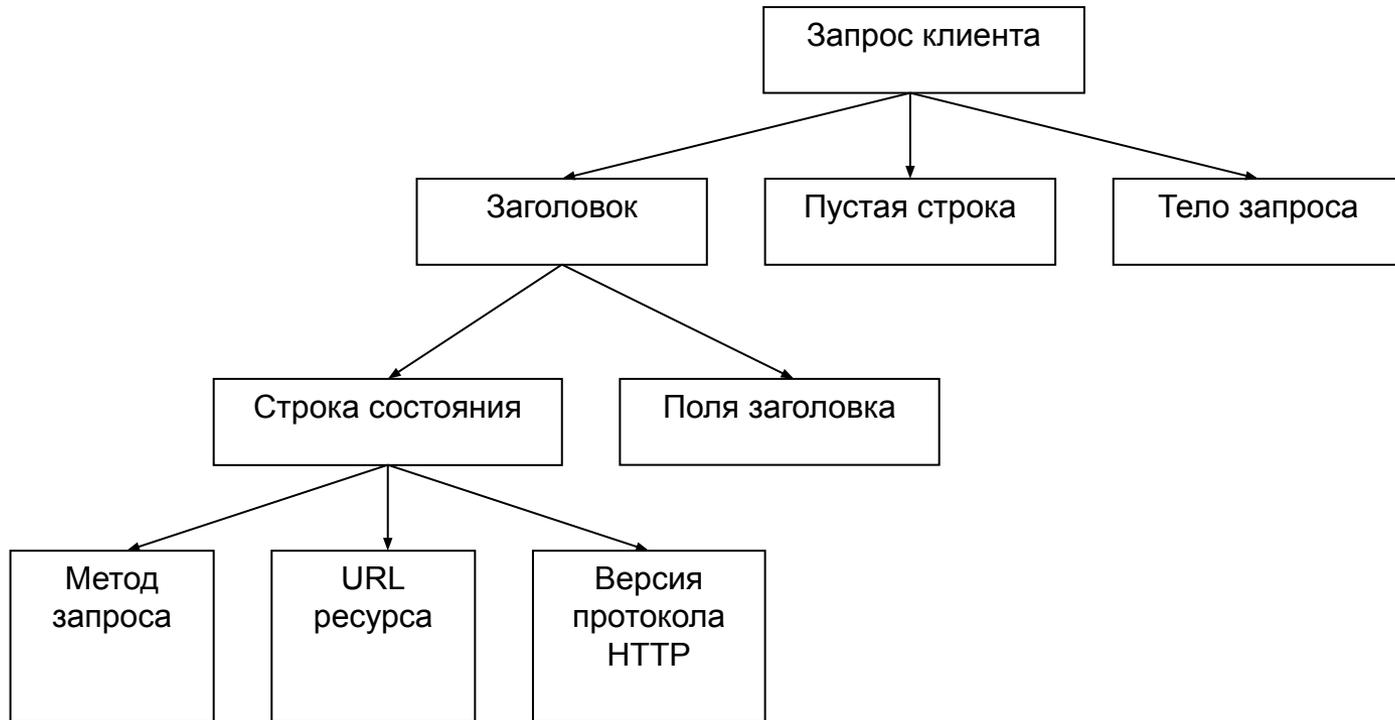


Структура протокола http

- Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:
 - *Заголовок сообщения*, который начинается со *строки состояния*, определяющей тип сообщения, и *полей заголовка*, характеризующих тело сообщения, описывающих параметры передачи и прочие сведения;
 - *Пустая строка*;
 - *Тело сообщения* — непосредственно данные сообщения.
- *Поля заголовка* и *тело* сообщения могут отсутствовать, но *строка состояния* является обязательным элементом, так как указывает на тип запроса/ответа.



Структура запроса клиента http



Методы запроса клиента

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



Методы запроса клиента

- **GET**. Согласно формальному определению, метод GET предназначается для получения ресурса с указанным URL. Получив запрос GET, сервер должен прочитать указанный ресурс и включить код ресурса в состав ответа клиенту. Ресурс, Несмотря на то что, по определению, метод GET предназначен для получения информации, он вполне подходит для передачи небольших фрагментов данных на сервер.
- **POST**. Согласно тому же формальному определению, основное назначение метода POST - передача данных на сервер. Однако, подобно методу GET, метод POST может применяться по-разному и нередко используется для получения информации с сервера. Как и в случае с методом GET, URL, заданный в строке состояния, указывает на конкретный ресурс.
- Методы **HEAD** и **PUT** являются модификациями методов GET и POST.



Поля заголовка запроса клиента

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

Имя_поля: значение

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



Поля заголовка запроса клиента

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

Пример запроса

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

Connection: Keep-Alive

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

Host: oak.oakland.edu

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

Accept-Language: en

Accept-Charset: iso-8859-1,*,utf-8



Структура ответа сервера http

Знание структуры ответа сервера необходимо разработчику веб-приложений, так как программы, которые выполняются на сервере, должны самостоятельно формировать ответ клиенту.

- Получив от клиента запрос, сервер должен ответить ему.
- Подобно запросу клиента, ответ сервера также состоит из четырех перечисленных ниже компонентов.
 - Строка состояния.
 - Поля заголовка.
 - Пустая строка.
 - Тело ответа.



Структура ответа сервера http

- Ответ сервера клиенту начинается со строки состояния, которая имеет следующий формат:
Версия_протокола Код_ответа Пояснительное_сообщение
- **Версия_протокола** задается в том же формате, что и в запросе клиента, и имеет тот же смысл.
- **Код_ответа** - это трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером.
- **Пояснительное_сообщение** дублирует код ответа в символьном виде. Это строка символов, которая не обрабатывается клиентом. Она предназначена для системного администратора или оператора, занимающегося обслуживанием системы, и является расшифровкой кода ответа.



Тело ответа веб-сервера

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

HTTP/1.0 200 OK

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

Тело ответа веб-сервера

- В используемых в настоящее время реализациях протокола HTTP первая цифра не может быть больше 5 и определяет следующие классы ответов.
 - 1 - специальный класс сообщений, называемых *информационными*. Код ответа, начинающийся с 1, означает, что сервер продолжает обработку запроса. При обмене данными между HTTP-клиентом и HTTP-сервером сообщения этого класса используются достаточно редко.
 - 2 - успешная обработка запроса клиента.
 - 3 - перенаправление запроса. Чтобы запрос был обслужен, необходимо пред-принять дополнительные действия.
 - 4 - ошибка клиента. Как правило, код ответа, начинающийся с цифры 4, возвращается в том случае, если в запросе клиента встретилась синтаксическая ошибка.
 - 5 - ошибка сервера. По тем или иным причинам сервер не в состоянии выполнить запрос.



Классы кодов ответа сервера

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

Поля заголовка ответа веб-сервера

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



Тело ответа веб-сервера

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



Пример ответа веб-сервера

HTTP/1.1 200 OK

Date: Thu, 06 Apr 2000 23:39:01 GMT

Server: Apache/1.3.11 (Unix)

Last-Modified: Fri, 03 Mar 2000 22:17:57 GMT

Content-Length: 4685

Connection: close

Content-Type: text/html

<HTML>

<head>

<meta name="GENERATOR" content="Mozilla/4.7 (Macintosh; I; PPC) [Netscape]">

<title>OAK Software Repository</title>

<link REV="made" HREF="mailto:archives@oakland.edu">

</head>

<body text="#000000" bgcolor="#D8CA87" link="#0000C0" vlink="#E00000" alink="#0000FF" background="/images/oak.jpg">

<CENTER>

</body>

</HTML>



Спецификация MIME

- Поле с именем **Content-type** может встречаться как в запросе клиента, так и в ответе сервера. В качестве значения этого поля указывается **MIME-тип** содержимого запроса или ответа.
- *MIME-тип* также передается в поле заголовка *Accept*, присутствующего в запросе.
- Спецификация *MIME* (*Multipurpose Internet Mail Extension*) первоначально была разработана для того, чтобы обеспечить передачу различных форматов данных в составе электронных писем.
- Однако применение MIME не исчерпывается электронной почтой. Средства MIME успешно используются в WWW и, по сути, стали неотъемлемой частью этой системы.



Спецификация MIME

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

тип/подтип

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



MIME типы данных

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

URI, URL, URN

- *URI (Uniform Resource Identifier)* — единообразный идентификатор ресурса, представляющий собой короткую последовательность символов, идентифицирующую абстрактный или физический ресурс.
- Самые известные примеры *URI* — это *URL* и *URN*.
- *URL (Uniform Resource Locator)* - это *URI*, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса.
- *URN (Uniform Resource Name)* — это *URI*, который идентифицирует ресурс в определённом пространстве имён, но, в отличие от *URL*, *URN* не указывает на местонахождение этого ресурса.
- *URI* не указывает на то, как получить ресурс, а только идентифицирует его. Что даёт возможность описывать с помощью *RDF (Resource Description Framework)* ресурсы, которые не могут быть получены через Интернет (имена, названия и т. д.)

Структура URL

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

Где:

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



Структура URL

- Общепринятые схемы (протоколы) URL включают протоколы: *ftp*, *http*, *https*, *telnet*, а также:
 - *gopher* — протокол *Gopher*;
 - *mailto* — адрес электронной почты;
 - *news* — новости *Usenet*;
 - *nntp* — новости *Usenet* через протокол *NNTP*;
 - *irc* — протокол *IRC*;
 - *prospero* — служба каталогов *Prospero Directory Service*;
 - *wais* — база данных системы *WAIS*;
 - *xmpp* — протокол *XMPP* (часть *Jabber*);
 - *file* — имя локального файла;
 - *data* — непосредственные данные (*Data: URL*);
-



Порт TCP/IP

- TCP/IP *порт* — целое число от 1 до 65535, позволяющие различным программам, выполняемым на одном хосте, получать данные независимо друг от друга. Каждая программа обрабатывает данные, поступающие на определённый порт («слушает» этот порт).
 - Самые распространённые сетевые протоколы имеют стандартные номера портов, хотя в большинстве случаев программа может использовать любой порт.
 - Для наиболее распространённых протоколов стандартные номера портов следующие:
 - HTTP: 80
 - FTP: 21 (для команд), 20 (для данных)
 - telnet: 23
 - POP3: 110
 - IMAP: 143
 - SMTP: 25
 - SSH: 22
-

HTTPS

- **HTTPS** — расширение протокола *HTTP*, поддерживающее шифрование. Данные, передаваемые по протоколу *HTTP*, «упаковываются» в криптографический протокол *SSL* или *TLS*, тем самым обеспечивается защита этих данных. В отличие от *HTTP*, для *HTTPS* по умолчанию используется TCP-порт 443.
 - Чтобы подготовить веб-сервер для обработки *HTTPS* соединений, администратор должен получить и установить в систему сертификат для этого веб-сервера.
-



SSL И TLS

- *SSL* (Secure Sockets Layer) — криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет.
 - При его использовании создаётся защищённое соединение между клиентом и сервером. *SSL* изначально разработан компанией *Netscape Communications*. Впоследствии на основании протокола *SSL 3.0* был разработан и принят стандарт *RFC*, получивший название [TLS](#).
 - Протокол использует шифрование с открытым ключом для подтверждения подлинности передатчика и получателя. Поддерживает надёжность передачи данных за счёт использования корректирующих кодов и безопасных хэш-функций.
-



SSL И TLS

- На нижнем уровне многоуровневого транспортного протокола (например, TCP) он является протоколом записи и используется для инкапсуляции различных протоколов (например POP3, IMAP, SMTP или HTTP).
 - Для каждого инкапсулированного протокола он обеспечивает условия, при которых сервер и клиент могут подтвердить друг другу свою подлинность, выполнять алгоритмы шифрования и производить обмен криптографическими ключами, прежде чем протокол прикладной программы начнет передавать и получать данные.
 - Для доступа к веб-страницам, защищённым протоколом SSL, в URL вместо схемы http, как правило, подставляется схема https, указывающая на то, что будет использоваться SSL-соединение. Стандартный TCP-порт для соединения по протоколу https — 443.
 - Для работы SSL требуется, чтобы на сервере имелся SSL-сертификат.
-



Методы аутентификации в WWW

- *Basic* — базовая аутентификация, при которой имя пользователя и пароль передаются в заголовках *http-пакетов*. Пароль при этом не шифруется и присутствует в чистом виде в кодировке *base64*. Для данного типа аутентификации использование *SSL* является обязательным.
- *Digest* — дайджест-аутентификация, при которой пароль пользователя передается в хешированном виде. По уровню конфиденциальности паролей этот тип мало чем отличается от предыдущего, так как атакующему все равно, действительно ли это настоящий пароль или только *хеш* от него: перехватив удостоверение, он все равно получает доступ к конечной точке. Для данного типа аутентификации использование *SSL* является обязательным.

Методы аутентификации в WWW

▣ *Integrated* — интегрированная аутентификация, при которой клиент и сервер обмениваются сообщениями для выяснения подлинности друг друга с помощью протоколов *NTLM* или *Kerberos*. Этот тип аутентификации защищен от перехвата удостоверений пользователей, поэтому для него не требуется протокол *SSL*. Только при использовании данного типа аутентификации можно работать по схеме *http*, во всех остальных случаях необходимо использовать схему *https*.



Cookie

- HTTP-сервер не помнит предыстории запросов клиентов и каждый запрос обрабатывается независимо от других
- Поэтому у сервера нет возможности определить, исходят ли запросы от одного клиента или разных клиентов
- Тем не менее механизм *cookie* позволяет серверу хранить информацию на компьютере клиента и извлекать ее оттуда.



Cookie

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



Пример использования cookie

1. Передача запроса серверу **A**.
2. Получение ответа от сервера **A**.
3. Передача запроса серверу **B**.
4. Получение ответа от сервера **B**. В состав ответа входит поле заголовка *Set-cookie*. Получив его, клиент записывает *cookie* на диск.
5. Передача запроса серверу **C**. Несмотря на то что на диске хранится запись *cookie*, клиент не предпринимает никаких специальных действий, так как значение *cookie* было записано по инициативе другого сервера.



Пример использования cookie

6. Получение ответа от сервера **C**.
7. Передача запроса серверу **A**. В этом случае клиент также никак не реагирует на тот факт, что на диске хранится *cookie*.
8. Получение ответа от сервера **A**.
9. Передача запроса серверу **B**. Перед тем как сформировать запрос, клиент определяет, что на диске хранится запись *cookie*, созданная после получения ответа от сервера **B**. Клиент проверяет, удовлетворяет ли данный запрос некоторым требованиям, и, если проверка дает положительный результат, включает в заголовок запроса поле *Cookie*.



Формат поля Set-Cookie

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

где

- Пара *имя = значение* – именованные данные, сохраняемые с помощью механизм *cookie*. Эти данные должны храниться на клиент-машине и передаваться серверу в составе очередного запроса клиента.
- *Дата*, являющаяся значением параметра *expires*, определяет время, по истечении которого информация *cookie* теряет свою актуальность. Если ключевое слово *expires* отсутствует, данные *cookie* удаляются по окончании текущего сеанса работы браузера.



Формат поля Set-Cookie

- Значение параметра *domain* определяет домен, с которым связываются данные cookie.
- Чтобы узнать, следует ли передавать в составе запроса данные *cookie*, браузер сравнивает доменное имя сервера, к которому он собирается обратиться, с доменами, которые связаны с записями *cookie*, хранящимися на клиент-машине.
- Результат проверки будет считаться положительным, если сервер, которому направляется запрос, принадлежит домену, связанному с *cookie*.
- Если соответствие не обнаружено, данные *cookie* не передаются.



Формат поля Set-Cookie

- Путь, указанный в качестве значения параметра *path*, позволяет выполнить дальнейшую проверку и принять окончательное решение о том, следует ли передавать данные *cookie* в составе запроса.
- Помимо домена с записью *cookie* связывается путь.
- Если браузер обнаружил соответствие *имени домена* значению параметра *domain*, он проверяет, соответствует ли путь к ресурсу пути, связанному с *cookie*.



Формат поля Set-Cookie

- Сравнение считается успешным, если ресурс содержится в каталоге, указанном посредством ключевого слова *path*, или в одном из его подкаталогов.
- Если и эта проверка дает положительный результат, данные *cookie* передаются серверу. Если параметр *path* в поле *Set-cookie* отсутствует, то считается, что запись *cookie* связана с URL конкретного ресурса, передаваемого сервером клиенту.
- Последний параметр, *secure*, указывает на то, что данные *cookie* должны передаваться по защищенному каналу.

Формат поля *Cookie*

- Для передачи данных *cookie* серверу используется поле заголовка *Cookie*.
- Формат этого поля:
Cookie: имя=значение; имя=значение; ...
- С помощью поля *Cookie* передается одна или несколько пар *имя = значение*. Каждая из этих пар принадлежит записи *cookie*, для которой URL запрашиваемого ресурса соответствуют имени домена и пути, указанным ранее в поле *Set-cookie*.

