Протоколы, сервисы и услуги
в Интернет и IP-сетях

Тема № 14
Протоколы WWW — HTTP и HTTPS

доц. каф. СС и ПД, к.т.н. С. С. Владимиров

2017 г.

Протокол HTTP

HyperText Transfer Protocol (HTTP)

Протокол передачи гипертекста. Используется службой WWW для передачи Web-страниц.

- RFC 1945 Hypertext Transfer Protocol - HTTP/1.0. May 1996.
- RFC 2616 Hypertext Transfer Protocol - HTTP/1.1. June 1999.
- RFC 7540 Hypertext Transfer Protocol Version 2 (HTTP/2). May 2015

Транспортным протоколом для HTTP является протокол TCP, причем сервер HTTP (сервер Web) находится в состоянии ожидания соединения со стороны клиента стандартно по порту 80 TCP, а клиент HTTP (браузер Web) является инициатором соединения.
Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. В самом общем случае URI выглядит следующим образом:

```
protocol://user:password@host:port/path/file?parameters#fragment
```

- `protocol` — прикладной протокол, посредством которого получают доступ к ресурсу;
- `user` — пользователь, от имени которого получают доступ к ресурсу;
- `password` — пароль пользователя для аутентификации при доступе к ресурсу;
- `host` — IP-адрес или имя сервера, на котором расположен ресурс;
- `port` — номер порта, на котором работает сервер, предоставляющий доступ к ресурсу;
- `path` — путь к файлу, содержащему ресурс;
- `file` — файл, содержащий ресурс;
- `parameters` — параметры для обработки ресурсом-программой;
- `fragment` — точка в файле, начиная с которой следует отображать ресурс.

Сообщения HTTP

Взаимодействие между клиентом и сервером Web осуществляется путем обмена сообщениями. Сообщения HTTP делятся на *запросы* клиента серверу и *ответы* сервера клиенту. HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос–ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера). Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

Общий формат сообщений HTTP

```
начальная строка
заголовок 1: значение
заголовок 2: значение
...
заголовок N: значение
CR LF (пустая строка)
тело сообщения (может
отсутствовать)
```

Формат начальной строки (start-line) клиента и сервера различаются.
Каждый заголовок состоит из названия, символа двоеточия ":"и значения.

Виды заголовков HTTP-сообщения

- **общие заголовки (general-headers)**, которые могут присутствовать как в запросе, так и в ответе;
- **заголовки запросов (request-headers)**, которые могут присутствовать только в запросе;
- **заголовки ответов (response-headers)**, которые могут присутствовать только в ответе;
- **заголовки объекта (entity-headers)**, которые относятся к телу сообщения и описывают его содержимое.

Сообщения HTTP. HTTP-запрос

Тело сообщения

В теле сообщения содержится собственно передаваемая информация. Тело сообщения представляет собой последовательность октетов (байтов). Тело сообщения может быть закодировано, например, для уменьшения объема передаваемой информации, при этом способ кодирования указывается в заголовке объекта `Content-Encoding`.

HTTP-запрос

Запрос от клиента к серверу состоит из строки запроса (request-line), заголовков (общих, запросов, объекта) и, возможно, тела сообщения.

Строка запроса:

```
<Метод HTTP> <Идентификатор запрашиваемого ресурса> <Версия HTTP>
```

Метод HTTP

Метод HTTP (или команда) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру.

Каждый сервер обязан поддерживать как минимум методы `GET` и `HEAD`. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях в сообщение ответа включается заголовок `Allow` со списком поддерживаемых методов.

Кроме методов `GET` и `HEAD`, часто применяется метод `POST`.

Методы HTTP

OPTIONS

Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. В ответ — заголовок Allow со списком поддерживаемых методов. Также может включаться информация о поддерживаемых расширениях.
Для того, чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «*». Запросы

```
OPTIONS * HTTP/1.1
```

могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1.
Результат выполнения метода не кэшируется.

GET

Используется для запроса содержимого указанного ресурса. С помощью метода `GET` можно также начать какой-либо процесс. Можно передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:

```
GET /path/resource?param1=value1&param2=value2 HTTP/1.1
```

Метод является идемпотентным: на одинаковый `GET` — одинаковый результат.

HEAD

Аналогичен `GET`, за исключением того, что в ответе сервера отсутствует тело. Запрос `HEAD` обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

DELETE

Удаляет указанный ресурс.

Методы HTTP

POST

Применяется для передачи пользовательских данных заданному ресурсу. Аналогично с помощью метода POST обычно загружаются файлы на сервер. В отличие от метода GET, метод POST не считается идемпотентным — многократное повторение одних и тех же запросов POST может возвращать разные результаты.

PUT

Применяется для загрузки содержимого запроса на указанный в запросе URI.
Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу.

PATCH

Аналогично PUT, но применяется только к фрагменту ресурса.

TRACE

Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе.

CONNECT

Преобразует соединение запроса в прозрачный TCP/IP туннель, обычно чтобы содействовать установлению защищенного SSL соединения через нешифрованный прокси.

Сообщения HTTP. HTTP-ответ

Ответ HTTP-сервера клиенту

```
<Версия HTTP> <Код состояния> <Поясняющая фраза>
```

Код состояния (Status-Code) – это целочисленный трехразрядный код результата понимания и удовлетворения запроса.
Поясняющая фраза (Reason-Phrase) – короткое текстовое описание кода состояния.
Код состояния предназначен для обработки программным обеспечением, а поясняющая фраза предназначена для пользователей.

Код состояния

Первая цифра кода состояния определяет класс ответа. Последние две цифры не имеют определенной роли в классификации. Имеется 5 значений первой цифры:

`1xx` Информационные коды – запрос получен, продолжается обработка.

`2xx` Успешные коды – действие было успешно получено, понято и обработано.

`3xx` Коды перенаправления – для выполнения запроса должны быть предприняты дальнейшие действия.

`4xx` Коды ошибок клиента – запрос имеет ошибку синтаксиса или не может быть выполнен.

`5xx` Коды ошибок сервера – сервер не в состоянии выполнить допустимый запрос.

Особенности протокола HTTP

Хотя протокол HTTP разрабатывался как средство работы с ресурсами сервера, у него отсутствуют в явном виде средства навигации среди этих ресурсов. Например, клиент не может явным образом запросить список доступных файлов, как в протоколе FTP. Предполагалось, что конечный пользователь уже знает URI необходимого ему документа, закачав который, он будет производить навигацию благодаря гиперссылкам. Это вполне нормально и удобно для человека, но затруднительно, когда стоят задачи автоматической обработки и анализа всех ресурсов сервера без участия человека. Решение этой проблемы лежит полностью на плечах разработчиков приложений, использующих данный протокол.

Большинство протоколов предусматривают установление TCP-сессии, в ходе которой один раз происходит авторизация, и дальнейшие действия выполняются в контексте этой авторизации. HTTP же устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт заголовок «`Content-Type: тип/подтип`», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).
Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

Примеры диалогов HTTP. Перенаправление на другой домен

Предположим, что у вымышленной компании Example Corp. есть основной сайт по адресу `http://example.com` и домен-псевдоним `example.org`. Клиент посылает запрос страницы «О компании» на вторичный домен (часть заголовков опущена):

```
GET /about.html HTTP/1.1
Host: example.org
User-Agent: MyLonelyBrowser/5.0
```

Так как домен `example.org` не является основным и компания не собирается в будущем его использовать в других целях, их сервер вернёт код для постоянного перенаправления, указав в заголовке Location целевой URL:

```
HTTP/1.x 301 Moved Permanently
Location: http://example.com/about.html#contacts
Date: Thu, 19 Feb 2009 11:08:01 GMT
Server: Apache/2.2.4
Content-Type: text/html; charset=windows-1251
Content-Length: 110
(пустая строка)
<html><body><a href=”http://example.com/about.html#contacts”>Click
here</a></body></html>
```

В заголовке Location можно указывать фрагменты как в данном примере. Браузер не указал фрагмент в запросе, так как его интересует весь документ. Но он автоматически прокрутит страницу до фрагмента «`contacts`», как только загрузит её. В тело ответа также был помещён короткий HTML-документ со ссылкой, с помощью которой посетитель попадёт на целевую страницу, если браузер не перейдёт на неё автоматически. Заголовок Content-Type содержит характеристики именно этого HTML-пояснения, а не документа, который находится по целевому URI.

Примеры диалогов HTTP. Перенаправление на региональный домен

Допустим, эта же компания Example Corp. имеет несколько региональных представительств по всему миру. И для каждого представительства у них есть сайт с соответствующим ccTLD. Запрос главной страницы основного сайта `example.com` может выглядеть так:

```
GET / HTTP/1.1
Host: example.com
User-Agent: MyLonelyBrowser/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru,en-us;q=0.7,en;q=0.3
Accept-Charset: windows-1251,utf-8;q=0.7,*;q=0.7
```

Сервер принял во внимание заголовок Accept-Language и сформировал ответ с временным перенаправлением на российский сервер example.ru, указав его адрес в заголовке Location:

```
HTTP/1.x 302 Found
Location: http://example.ru/
Cache-Control: private
Date: Thu, 19 Feb 2009 11:08:01 GMT
Server: Apache/2.2.6
Content-Type: text/html; charset=windows-1251
Content-Length: 82
(пустая строка)
<html><body><a href="http://example.ru»Example Corp.</a></body></html>
```

Обратите внимание на заголовок Cache-Control. Значение «private» сообщает остальным серверам (в первую очередь прокси) что ответ может кэшироваться только на стороне клиента. В противном случае не исключено, что следующие посетители из других стран будут переходить всё время не в своё представительство.

HTTP cookie

HTTP cookie

Куки — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

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

HTTP cookie. Установка куки

Установка куки

Запрашивая страницу, браузер отправляет веб-серверу короткий текст с HTTP-запросом. Например, для доступа к странице `http://www.example.org/index.html`, браузер отправляет на сервер `www.example.org` следующий запрос:

```
GET /index.html HTTP/1.1
Host: www.example.org
```

Сервер отвечает, отправляя запрашиваемую страницу вместе с текстом, содержащим HTTP-ответ. Там может содержаться указание браузеру сохранить куки:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
```

Строка `Set-cookie` отправляется лишь тогда, когда сервер желает, чтобы браузер сохранил куки. В этом случае, если куки поддерживаются браузером и их приём включён, браузер запоминает строку `name=value` (имя = значение) и отправляет её обратно серверу с каждым последующим запросом. Например, при запросе следующей страницы `http://www.example.org/spec.html` браузер пошлёт серверу `www.example.org` следующий запрос:

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: name=value
Accept: */*
```

Этот запрос отличается от первого запроса тем, что содержит строку, которую сервер отправил браузеру ранее. Таким образом, сервер узнает, что этот запрос связан с предыдущим. Сервер отвечает, отправляя запрашиваемую страницу и, возможно, добавив новые куки.

Значение куки может быть изменено сервером путём отправления новых строк `Set-Cookie: name=newvalue`. После этого браузер заменяет старое куки с тем же name на новую строку.

HTTP cookie. Атрибуты куки

Атрибуты куки

Кроме пары имя/значение, куки может содержать срок действия, путь и доменное имя. RFC 2965 также предусматривает, что куки должны обязательно иметь номер версии, но это используется редко. Эти атрибуты должны идти после пары `name=newvalue` и разделяться точкой с запятой. Например:

```
Set-Cookie: name=newvalue; expires=date; path=/; domain=.example.org.
```

Домен и путь говорят браузеру, что куки должна быть отправлена обратно на сервер при запросах URL для указанного домена и пути. Если они не указаны, используются домен и путь запрошенной страницы.
Фактически, куки определяются тройкой параметров имя-домен-путь. Иными словами, куки с разными путями или доменами являются разными куки, даже если имеют одинаковые имена. Соответственно, куки меняется на новое, только если новое куки имеет те же имя, путь и домен.
Дата истечения указывает браузеру, когда удалить куки. Если срок истечения не указан, куки удаляется по окончании пользовательского сеанса, то есть с закрытием браузера. Если же указана дата истечения срока хранения, куки становится постоянной до указанной даты.
Срок хранения куки истекает в следующих случаях:

- В конце сеанса (например, когда браузер закрывается), если куки не являются постоянными.
- Дата истечения была указана, и срок хранения вышел.
- Браузер удалил куки по запросу пользователя.

Сервер может узнать, когда истекают сроки хранения куки, только когда браузер отправляет на сервер эту информацию.

HTTP cookie. Аутентификация

Аутентификация с использованием куки

Куки могут использоваться сервером для опознания ранее аутентифицированных пользователей. Это происходит следующим образом:
Пользователь вводит имя пользователя и пароль в текстовых полях страницы входа и отправляет их на сервер.
Сервер получает имя пользователя и пароль, проверяет их и, при их правильности, отправляет страницу успешного входа, прикрепив куки с неким идентификатором сессии. Эта куки может быть действительна не только для текущей сессии браузера, но может быть настроена и на длительное хранение.
Каждый раз, когда пользователь запрашивает страницу с сервера, браузер автоматически отправляет куки с идентификатором сессии серверу. Сервер проверяет идентификатор по своей базе идентификаторов и, при наличии в базе такого идентификатора, «узнаёт» пользователя.

Протокол HTTPS

HTTPS (HyperText Transfer Protocol Secure)

Это расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол SSL или TLS. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443. Фактически, HTTPS не является отдельным протоколом. Это обычный HTTP, работающий через шифрованные транспортные механизмы SSL и TLS.

Протокол был разработан компанией Netscape Communications для браузера Netscape Navigator в 1994 году. HTTPS широко используется в мире веб и поддерживается всеми популярными браузерами.

Чтобы подготовить веб-сервер для обработки https-соединений, администратор должен получить и установить в систему сертификат для этого веб-сервера. Сертификат состоит из 2 частей (2 ключей) — public и private. Public-часть сертификата используется для зашифрования трафика от клиента к серверу в защищённом соединении, private-часть — для расшифрования полученного от клиента зашифрованного трафика на сервере. После того как пара ключей приватный/публичный сгенерированы, на основе публичного ключа формируется запрос на сертификат в Центр сертификации, в ответ на который ЦС высылает подписанный сертификат. ЦС, при подписывании проверяет клиента, что позволяет ему гарантировать что держатель сертификата является тем, за кого себя выдаёт (обычно это платная услуга).

Существует возможность создать такой сертификат, не обращаясь в ЦС. Такие сертификаты могут быть созданы для серверов, работающих под Unix. Подписываются такие сертификаты этим же сертификатом и называются самоподписанными (self-signed).

Эта система также может использоваться для аутентификации клиента, чтобы обеспечить доступ к серверу только авторизованным пользователям. Для этого администратор обычно создаёт сертификаты для каждого пользователя и загружает их в браузер каждого пользователя. Также будут приниматься все сертификаты, подписанные организациями, которым доверяет сервер. Такой сертификат обычно содержит имя и адрес электронной почты авторизованного пользователя, которые проверяются при каждом соединении, чтобы проверить личность пользователя без ввода пароля.

В HTTPS для шифрования используются ключи длиной 40, 56, 128 или 256 бит. Многие современные сайты требуют использования новых версий браузеров, поддерживающих шифрование с длиной ключа 128 бит, с целью обеспечить достаточный уровень безопасности. Такое шифрование значительно затрудняет злоумышленнику поиск паролей и другой личной информации.

Источники

- Материалы с сайта `https://wikipedia.org/`
- Telecommunication technologies — телекоммуникационные технологии / Ю. А. Семенов.
  URL: `http://book.itep.ru/`
- Что такое cookies и как с ними работать / А. А. Аликберов.
  URL: `http://citforum.ru/internet/html/cookie.shtml`