Определение:

Протокол уровня приложений, используется для доставки ресурсов, наиболее распространен в глобальной сети www

(World Wide Web – [распределённая система](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), предоставляющая доступ к [связанным](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0) между собой документам, расположенным на различных [компьютерах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), подключённых к сети [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), а интернет это сеть сетей, в ней работают сетевые протоколы, в www уровень приложений)

URL – используется для идентификации документов в www, для нее есть стандарт, свляется частью более общего стандарта URI (разница – позволяет идентифицировать ресурс любым возможным способом, может говорить где ресурс взять, может говорить какой это ресурс, какие у него свойства, а URL это все таки локатор, он говорит куда пойти чтобы найти ресурс)



В этой записи:

***схема***

схема обращения к ресурсу; в большинстве случаев имеется в виду [сетевой протокол](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D1%8B%D0%B5_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B)

***логин***

имя пользователя, используемое для доступа к ресурсу

***пароль***

[пароль](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D0%BB%D1%8C) указанного пользователя

***хост***

полностью прописанное [доменное имя](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B8%D0%BC%D1%8F) [хоста](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82) в системе [DNS](https://ru.wikipedia.org/wiki/DNS) или [IP-адрес](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81) хоста в форме четырёх групп [десятичных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) [чисел](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE), разделённых точками; числа — целые в интервале от 0 до 255.

***порт***

[порт](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(TCP/UDP)) хоста для подключения

***URL-путь***

уточняющая [информация](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) о месте нахождения ресурса; зависит от протокола.

***параметры***

строка запроса с передаваемыми на сервер ([методом GET](https://ru.wikipedia.org/wiki/HTTP#GET)) параметрами. Начинается с символа **?**, разделитель параметров — знак **&**. Пример: ?параметр\_1=значение\_1&параметр\_2=значение\_2&параметр3=значение\_3

***якорь***

[идентификатор «якоря»](https://en.wikipedia.org/wiki/fragment_identifier) (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80_%C2%AB%D1%8F%D0%BA%D0%BE%D1%80%D1%8F%C2%BB&action=edit&redlink=1) с предшествующим символом **#**. Якорем может быть указан заголовок внутри документа или [атрибут id](https://en.wikipedia.org/wiki/HTML_attribute) (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=HTML_%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B&action=edit&redlink=1) элемента. По такой ссылке браузер откроет страницу и переместит окно к указанному элементу.

Примеры uri

<mailto:John.Doe@example.com>

схема – http

подключаемся по tcp к указанному порту и хосту (по умолчанию порт 80ый) и используем http поверх tcp

схема - https

открываем соединение на указанный порт и адрес (по ум 443 порт) используем tlc поверх tcp, а уже поверх tlc http, это обеспечивает аутификацию сервера и шифрование передаваемых данных

Логика работы http

Про http 1.1

Запрос-ответный протокол, по умолчанию соединение не закрывается после ответа.

Сервер не хранит какого то состояния для конкретного соединения.

* Если хотим поддерживать какую-то «сессию» в работе с сервером, то необходимо нашу аутификационную информацию отправлять вместе с каждым запросом
* В любой момент можем отключить и подключиться с другого ip

Схема работы http 1.1

- клиент отправляет запрос

- ждет

- после того как ответ прочитан (возможно еще не обработан), может отправлять повторный запрос

Просто но не эффективно при больших задержках. Частичное решение – несколько соединений от одного клиента к одному серверу. Проблемы такого решения : ограничение не более 6 сединений на один хост, конкуренция таких параллельных соединений.

Формат запросов и ответов

Метод запроса //GET

GET path+query HTTP1.1

Почему не указывается полный url? Казалось что это не нужно, так как уже вроде установили соединение с хостом.

Дальше идут заголовки

host:: --- [обязательный заголовок]

user agent: --- (это идентификатор клиентского приложения, которое испол для получения ресурса , например браузер или консольное приложение)

ответ

версия протокола, код ответа, текстовое описание, заголовки, пустая строка, тело ответа

http1.1 200

как понять что закончилось тело и начался новый ответ?

1. Разделитель
2. Знать сколько байт в ответе

2:

Используется заголовок Content-Length – указывают количество байт

Еще есть механизм потоковой передачи, реализуется через заголовок

Transfer-Encoding, могут быть различные значения, например Chunked. Означает что будет передаваться кусочками. В качестве тела передается не поток байт, а кадры, которые формируются с помощью текстовых строк с указанием количества байтов. Например что следующий кусок будет длинной в 250 байтов.

Такое будет тело :

250 [опции]

-------- <250 байт>

Перевод строки

250

--------

0

Когда не указан ни Transfer-Encoding, ни Content-Length может быть указан заголовок conection, в котором пишут свойства соединения (tcp/tlc)

Возможное значение – close – означает, что когда ответ будет завершен, то соединение будет закрыто

Если нет вообще ничего – баг.

Коды возвратов

Начинаются на:

1 – информационные

2 – успех в каком-то виде

3 – перенаправление (ресурса здесь нет, но есть где-то еще)

4 – ошибка пользователя

5 – ошибка сервера

201 – accepted – запрос в очереди на обработку

400 – bad request

404 – not found (у сервера нет такого ресурса)

Методы

GET хотим получить ресурс (сервер не меняет сильно свое состояние) запрос разрешено кешировать

HEAD получить только заголовки для запроса, без тела (content-length, content-type) когда хотим получить мета-информацию о ресурсе

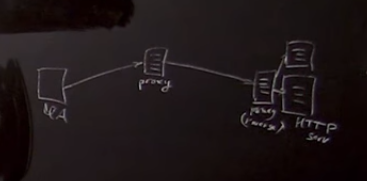
POST модификация чего-либо (ресурса по адресу, перекодирование, совершение операции которая не изменяет видимо ресурс, создание нового ресурса, но при этом мы не помещаем его по конкретному урлу)

PATCH изменение конкретного ресурса по урлу, в теле запроса говорим какие изменения должны произойти

PUT поместить на сервер ресурс, указывается url, по которому будет доступен этот ресурс для сервера, ресурс не будет изменен сервером

DELETE удаляет ресурс , передаем url ресурса

Проксирование



Для использования прокси на стороне клиента используются специальные запросы(в них записывается полный url), так как в обычных недостаточно информации для установления соединения с сервером. Прокси в свою очередь передает обычный запрос.

Контроль трафика, прокси может запретить доступ к какому-то серверу.

Кеширование

Может кешировать UA(USER AGENT) – какой-то ресурс уже запрашивал, второй раз не будет,

Промежуточный клиентский прокси, Промежуточный серверный прокси

Способы кеширования

Подход когда каждый раз запрашиваем свежую версию, но если ресурс не изменился, сервер говорит об этом и тело не отправляет. Случай когда ресурс важен и меняется часто.

Подход когда кешируем на какое-то время (допустимое время указанно в заголовках ресурса).

Подход когда кешируем ресурс навсегда, но если он обновился, то он будет доступен по новому урлу.

Механизмы контроля

Для первого варианта:

В ответ Передаем заголовок date modified – в нем указывается дата изменения

В запросе передаем if modified since - в нем указывается дата пришедшая в ответе

Ресурс не менялся – сервер вернет код 304

Ресурс менялся – вернет весь ресурс

Еще вариант – сервер возвращает заголовок etag – ресурс не имеет даты модификации, но имеет другие свойства (контрольная сумма, модификатор в бд) – в нем указывается произвольная строка. Тогда мы должны ответить <какое-то заголовок>: в нем указать эту строку (из etag)

Для второго варианта:

От сервера. Espaires: date (до этой даты можем использовать ресурс)

От сервера. Cash-control: max age (этот ресурс можно кешировать не более чем на max age)

При кешировании стоит учитывать что запрос может быть на один и тот же ресурс, но чем то отличаться, например :

Есть заголовок > accept: meme-type (или список с такими типами) кот хотим получить по возможности

Есть заголовок > accept-encoding: gzip хотим получить ответ в сжатом виде

< Very : accept , accept-encoding - список имен заголовков, кот влияют на идентичность данного ресурса при кешировании

Тогда прокси должен проверить что новый запрос совпадает не только по урлу но и по всем таким заголовкам

http/2

решается проблема как сделать много запросов с одного клиента одному серверу. (сайты стали тяжелее, требуется 100 запросов чтобы загрузить), при чем чтобы было одно соединение.

Идея: меняем формат запросов. У каждого будет какой-то свой идентификатор, так же как и ответ.

Но ответы еще и делятся на кусочки и отправляются вперемешку. (кусок второго ответа, кусок третьего…)

Как это работает:

Внутри tcp соединения создаются виртуальные каналы(каждая из сторон может их создавать), у него есть идентификатор и его рамках могут передаваться различные кадры(состоят из заголовка с размером (он ограничен) и полезной нагрузки)

Виды кадров

Settings – позволяет договориться о настройках, первый в потоке

Headers Кадр кот содержит заголовки (запроса или ответа)

Push headers отправляется от сервера если был пуш запрос

Continuation если заголовки не входят в один кадр

Data