# Ультимативный гайд по HTTP. Структура запроса и ответа

Весь современный веб построен на протоколе HTTP. Каждый сайт использует его для общения клиента с сервером. Между собой сервера тоже часто общаются по этому протоколу. На данный момент существует четыре его версии и все они до сих пор используются. Поэтому статьи будут полезны инженерам любых уровней и специализаций, и помогут систематизировать знания об этой важной технологии.

### Что такое HTTP

**HTTP** — это гипертекстовый протокол передачи данных прикладного уровня в сетевой модели OSI. Его представил миру Тим Бернерс-Ли в марте 1991 года. Главная особенность HTTP — представление всех данных в нём в виде простого текста. Через HTTP разные узлы в сети общаются между собой. Модель клиент-серверного взаимодействия классическая: клиент посылает запрос серверу, сервер обрабатывает запрос и возвращает ответ клиенту. Полученный ответ клиент обрабатывает и решает: прекратить взаимодействие или продолжить отправлять запросы.

Ещё одна особенность: протокол не сохраняет состояние между запросами. Каждый запрос от клиента для сервера — отдельная транзакция. Когда поступают два соседних запроса, сервер не понимает, от одного и того же клиента они поступили, или от разных. Такой подход значительно упрощает построение архитектуры веб-серверов.

Как правило, передача данных по HTTP осуществляется через открытое TCP/IP-соединение1. Серверное программное обеспечение по умолчанию обычно использует TCP-порт 80 для работы веб-сервера, а порт 443 — для HTTPS-соединений.

**HTTPS (HTTP *Secure***) — это надстройка над протоколом HTTP, которая поддерживает шифрование посредством криптографических протоколов SSL и TLS. Они шифруют отправляемые данные на клиенте и дешифруют их на сервере. Это защищает данные от чтения злоумышленниками, даже если им удастся их перехватить.

### HTTP/0.9

В 1991 году была опубликована первая версия протокола с названием HTTP/0.9. Эта реализация была проста, как топор. От интернет-ресурса того времени требовалось только загружать запрашиваемую HTML-страницу и HTTP/0.9 справлялся с этой задачей. Обычный запрос к серверу выглядел так:

GET /http-spec.html

В протоколе был определен единственный метод GET и и указывался путь к ресурсу. Так пользователи получали страничку. После этого открытое соединение сразу закрывалось.

### HTTP/1.0

Годы шли и интернет менялся. Стало понятно, что нужно не только получать странички от сервера, но и отправлять ему данные. В 1996 году вышла версия протокола 1.0.

**Что изменилось:**

1. В запросе теперь надо было указывать версию протокола. Так сервер мог понимать, как обрабатывать полученные данные.
2. В ответе от сервера появился статус завершения обработки запроса.
3. К запросу и ответу добавился новый блок с заголовками.
4. Добавили поддержку новых методов:

* HEAD запрашивает ресурс так же, как и метод GET, но без тела ответа. Так можно получить только заголовки, без самого ресурса.
* POST добавляет сущность к определённому ресурсу. Часто вызывает изменение состояния или побочные эффекты на сервере. Например, так можно отправить запрос на добавление нового поста в блог.

### Структура запроса

Простой пример запроса:

GET /path HTTP/1.0  
Content-Type: text/html; charset=utf-8  
Content-Length: 4  
X-Custom-Header: value  
  
test

В первой строчке указаны метод запроса — GET, путь к ресурсу — /path и версия протокола — HTTP/1.0.

Далее идёт блок заголовков. Заголовки — это пары ключ: значение, каждая из которых записывается с новой строки и разделяется двоеточием. Они передают дополнительные данные и настройки от клиента к серверу и обратно.

HTTP — это текстовый протокол, поэтому и все данные передаются в виде текста. Заголовки можно отделить друг от друга только переносом строки. Нельзя использовать запятые, точку с запятой, или другие разделители. Всё, что идет после имени заголовка с двоеточием и до переноса строки, будет считаться значением заголовка2.

**В примере серверу передали три заголовка:**

1. Content-Type — стандартный заголовок. Показывает, в каком формате будут передаваться данные в теле запроса или ответа.
2. Content-Length — сообщает длину контента в теле запроса в байтах.
3. X-Custom-Header — пользовательские заголовки, начинающиеся с X- с произвольными именем. Через них реализуется специфическая логика обработки для конкретного сервера. Если веб-сервер не поддерживает такие заголовки, то он проигнорирует их.

После блока заголовков идёт тело запроса, в котором передается текст test*.*

**А так может выглядеть ответ от сервера:**

HTTP/1.1 200 OK  
Date: Thu, 29 Jul 2021 19:20:01 GMT  
Content-Type: text/html; charset=utf-8  
Content-Length: 2  
Connection: close  
Server: gunicorn/19.9.0  
Access-Control-Allow-Origin: \*  
Access-Control-Allow-Credentials: true  
  
OK

В первой строке — версия протокола и статус ответа, например, 200 ОК. Далее идут заголовки ответа. После блока заголовков — тело ответа, в котором записан текст OK.

### Статусы ответов

Клиенту зачастую недостаточно просто отправить запрос на сервер. Во многих случаях надо дождаться ответа и понять, как сервер обработал запрос. Для этого были придуманы статусы ответов. Это трёхзначные числовые коды с небольшими текстовыми обозначениями. Их можно увидеть в терминале или браузере. Сами коды делятся на 5 классов:

* Информационные ответы: коды 100–199
* Успешные ответы: коды 200–299
* Редиректы: коды 300–399
* Клиентские ошибки: коды 400–499
* Серверные ошибки: коды 500–599

Мы рассмотрим основные коды, которые чаще всего встречаются в реальных задачах. С остальными более подробно можно ознакомиться в реестре кодов состояния HTTP.

#### Информационные ответы

**100 Continue** — промежуточный ответ. Он указывает, что запрос успешно принят. Клиент может продолжать присылать запросы или проигнорировать этот ответ, если запрос был завершён.

Примечание

**101 Switching Protocol** присылается в ответ на запрос, в котором есть заголовок *Upgrade*. Это означает, что сервер переключился на протокол, который был указан в заголовке. Такая методика используется, например, для переключения на протокол *Websocket*.

**102 Processing** — запрос получен сервером, но его обработка ещё не завершена.

#### Успешные ответы

**200 OK** — запрос принят и корректно обработан веб-сервером.

**201 Created** — запрос корректно обработан и в результате был создан новый ресурс. Обычно он присылается в ответ на POST запрос.

#### Редиректы

**301 Moved Permanently** — запрашиваемый ресурс на постоянной основе переехал на новый адрес. Тогда новый путь к ресурсу указывается сервером в заголовке Location ответа.

Примечание

**302 Found** — указывает, что целевой ресурс временно доступен по другому URl. Адрес перенаправления может быть изменен в любое время, а клиент должен продолжать использовать действующий URI для будущих запросов. Тогда временный путь к ресурсу указывается сервером в заголовке Location ответа.

Примечание

**307 Temporary Redirect** — имеет то же значение, что и код 302, за исключением того, что клиент не может менять метод последующего запроса.

**308 Permanent Redirect** — имеет то же значение, что и код 301, за исключением того, что клиент не может менять метод последующего запроса.

#### Клиентские ошибки

**400 Bad Request** — запрос от клиента к веб-серверу составлен некорректно. Обычно это происходит, если клиент не передаёт необходимые заголовки или параметры.

**401 Unauthorized** — получение запрашиваемого ресурса доступно только аутентифицированным пользователям.

**403 Forbidden** — у клиента не хватает прав для получения запрашиваемого ресурса. Например, когда обычный пользователь сайта пытается получить доступ к панели администратора.

**404 Not Found** — сервер не смог найти запрашиваемый ресурс.

**405 Method Not Allowed** — сервер знает о существовании HTTP-метода, который был указан в запросе, но не поддерживает его. В таком случае сервер должен вернуть список поддерживаемых методов в заголовке Allow ответа.

#### Серверные ошибки

**500 Internal Server Error** — на сервере произошла непредвиденная ошибка.

**501 Not Implemented** — метод запроса не поддерживается сервером и не может быть обработан.

**502 Bad Gateway** — сервер, действуя как шлюз или прокси, получил недопустимый ответ от входящего сервера, к которому он обращался при попытке выполнить запрос.

**503 Service Unavailable** — сервер не готов обработать запрос (например, из-за технического обслуживания или перегрузки). Обратите внимание, что вместе с этим ответом должна быть отправлена удобная страница с объяснением проблемы. Этот ответ следует использовать для временных условий, а HTTP-заголовок Retry-After по возможности должен содержать расчётное время до восстановления службы.

**504 Gateway Timeout** — этот ответ об ошибке выдается, когда сервер действует как шлюз и не может получить ответ за отведенное время.

**505 HTTP Version Not Supported** — версия HTTP, используемая в запросе, не поддерживается сервером.

В HTTP из всего диапазона кодов используется совсем немного. Те коды, которые не используются для задания определенной логики в спецификации, являются неназначенными и могут использоваться веб-серверами для определения своей специфической логики. Это значит, что вы можете, например, придать коду 513 значение «Произошла ошибка обработки видео», или любое другое. Неназначенные коды вы можете посмотреть в реестре кодов состояния HTTP.3

### Тело запроса и ответа

Тело запроса опционально и всегда отделяется от заголовков пустой строкой. А как понять, где оно заканчивается? Всё кажется очевидным: где кончается строка, там и заканчивается тело. Однако, два символа переноса строки в HTTP означают конец запроса и отправляют его на сервер. Как быть, если мы хотим передать в теле текст, в котором есть несколько абзацев с разрывами в две строки?

POST /path HTTP/1.1  
Host: localhost  
  
Первая строка  
  
  
Вторая строка после разрыва

По логике работы HTTP соединение отправится сразу после второй пустой строки и сервер получит в качестве данных только строку Первая строка. Описанную проблему решает специальный заголовок Content-Length. Он указывает на длину контента в байтах. Обычно клиенты (например, браузеры) автоматически считают длину передаваемых данных и добавляют к запросу заголовок с этим значением. Когда сервер получит запрос, он будет ожидать в качестве контента ровно столько байт, сколько указано в заголовке.

Однако, этого недостаточно для того, чтобы передать данные на сервер. Поведение зависит от реализации сервера, но для большинства из них необходимо также передать заголовок *Content-Type*. Он указывает на тип передаваемых данных. В качестве значения для этого заголовка используют MIME-типы.4

**MIME (Multipurpose Internet Mail Extensions, многоцелевые расширения интернет-почты)** — стандарт, который является частью протокола HTTP. Задача MIME — идентифицировать тип содержимого документа по его заголовку. К примеру, текстовый файл имеет тип text/plain, а HTML-файл — text/html.

Для передачи данных в формате обычного текста надо указать заголовок Content-Type: text/plain, а для JSON — Content-Type: application/json.

Можно ли передать тело с GET-запросом?

Популярный вопрос на некоторых собеседованиях: «Можно ли передать тело с GET-запросом?». Правильный ответ — да. Тело запроса не зависит от метода. В стандарте не описана возможность принимать тело запроса у GET-метода, но это и не запрещено. Технически вы можете отправить данные в теле, но скорее всего веб-сервер будет их игнорировать.

Представим, что на абстрактном сайте есть форма аутентификации пользователя, в которой есть всего два поля: email и пароль.

Если пользователь ввёл данные и нажал на кнопку «Войти», то данные из полей формы должны попасть на сервер. Самым простым и распространенным форматом передачи таких данных будет MIME application/x-www-form-urlencoded. В нем все поля передаются в одной строке в формате ключ=значение и разделяются знаком &.

**Запрос на отправку данных будет выглядеть так:**

POST /login HTTP/1.0  
Host: example.com  
Content-Type: application/x-www-form-urlencoded; charset=utf-8  
Content-Length: 26  
login=user&password=qwerty

Тут есть небольшая особенность. Как понять, где заканчивается ключ и начинается его значение, если в пароле будет присутствовать знак «=» ?

POST /login HTTP/1.0  
Host: example.com  
Content-Type: application/x-www-form-urlencoded; charset=utf-8  
Content-Length: 26  
login=user&password=123=45

В этом случае сервер не сможет понять, как разбить строку на параметры и их значения. На самом деле значения кодируются при помощи механизма url encoding.5 При использовании этого механизма знак «=» будет преобразован в код %3D .

**Тогда наш запрос приобретёт такой вид:**

POST /login HTTP/1.1  
Host: example.com  
Content-Type: application/x-www-form-urlencoded; charset=utf-8  
Content-Length: 28  
  
login=user&password=123%3D45

### Query string

Данные на сервер можно передавать через тело запроса и через так называемую строку запроса Query String. Это параметры формата *ключ=значение*, которые располагаются в пути к ресурсу:

GET /files?key=value&key2=value2 HTTP/1.0

При этом параметры можно передавать прямо от корня домена:

GET /?key=value&key2=value2 HTTP/1.0

Query String имеет такой же формат, как и тело запроса с MIME application/x-www-form-urlencoded*,* только первая пара значений отделяется от адреса вопросительным знаком.

Некоторые инженеры ошибочно полагают, что Query String являются параметрами GET-запроса и даже называют их GET-параметрами, но на самом деле это не так. Как и тело запроса, Query String не имеет привязки к HTTP-методам и может передаваться с любым типом запросов.

Обычно параметры Query String используются в GET-запросах, чтобы конкретизировать получаемый ресурс. Например, можно получить на сервере список файлов, имена которых будут начинаться с переданного значения.

GET-запросы по своей идеологии должны быть идемпотентными. Это значит, что множественный вызов метода с одними и теми же параметрами должен приводить к одному и тому же результату. Например, поисковые боты перемещаются по сайту только по ссылкам и делают только GET-запросы, потому что исходя из семантики они не смогут таким образом изменить данные на сайте и повлиять на его работу.