Протокол HTTP

- **HTTP** (англ. *HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов).

  Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

  HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов.

Спецификация HTTP

- **Название:**Hypertext Transfer Protocol
- **Уровень (по моделиOSI):**Прикладной
- **Семейство:**TCP/IP
- **Создан в:**1990 г.
- **Порт/ID:**80/TCP
- **Назначение протокола:** Доступ к гипертексту, ныне стал универсальным
- **Спецификация:**RFC 1945, RFC 2616
- **Основные реализации (клиенты):**Веб-браузеры, например Internet Explorer, Mozilla Firefox, Opera, Google Chrome и др.
- **Основные реализации (серверы):**Apache, IIS

- **HTTP** используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как **SOAP**, **XML-RPC**, **WebDAV**.

- Основным объектом манипуляции в **HTTP** является ресурс, на который указывает **URI** (англ. Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное.
- 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 данный протокол является текстовым.

- **HTTP** — протокол прикладного уровня, аналогичными ему являются **FTP** и **SMTP**.

  Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов **HTTP** использует глобальные **URI**. В отличие от многих других протоколов, 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. 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.

Достоинства HTTP

- **Простота**

  Протокол настолько прост в реализации, что позволяет с лёгкостью создавать клиентские приложения.
- **Расширяемость**

  Возможности протокола легко расширяются благодаря внедрению своих собственных заголовков, сохраняя совместимость с другими клиентами и серверами. Они будут игнорировать неизвестные им заголовки, но при этом можно получить необходимую функциональность при решении специфической задачи.
- **Распространённость**

  При выборе протокола HTTP для решения конкретных задач немаловажным фактором является его распространённость. Как следствие, это обилие различной документации по протоколу на многих языках мира, включение удобных в использовании средств разработки в популярные IDE, поддержка протокола в качестве клиента многими программами и обширный выбор среди хостинговых компаний с серверами HTTP.

Недостатки и проблемы

- **Большой размер сообщений**

  Использование текстового формата в протоколе порождает соответствующий недостаток: большой размер сообщений по сравнению с передачей двоичных данных. Из-за этого возрастает нагрузка на оборудование при формировании, обработке и передаче сообщений. Для решения данной проблемы в протокол встроены средства для обеспечения кэширования на стороне клиента, а также средства компрессии передаваемого контента.

Недостатки и проблемы

- **Отсутствие «навигации»**

  Хотя протокол 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.

Недостатки и проблемы

- **Нет поддержки распределённости**

  Протокол HTTP разрабатывался для решения типичных бытовых задач, где само по себе время обработки запроса должно занимать незначительное время или вообще не приниматься в расчёт. Но в промышленном использовании с применением распределённых вычислений при высоких нагрузках на сервер протокол HTTP оказывается беспомощен.

  В 1998 году W3C предложил альтернативный протокол HTTP-NG (англ. HTTP Next Generation) для полной замены устаревшего с акцентированием внимания именно на этой области. Идею его необходимости поддержали крупные специалисты по распределённым вычислениям, но данный протокол до сих пор находится на стадии разработки.

Программное обеспечение

Всё программное обеспечение для работы с протоколом HTTP разделяется на три больших категории:

* **Серверы** как основные поставщики услуг хранения и обработки информации (обработка запросов).
* **Клиенты** — конечные потребители услуг сервера (отправка запроса).
* **Прокси** для выполнения транспортных служб.

Для отличия конечных серверов от прокси в официальной документации используется термин origin server (рус. исходный сервер). Разумеется, один и тот же программный продукт может одновременно выполнять функции клиента, сервера или посредника в зависимости от поставленных задач. В спецификациях протокола HTTP подробно описывается поведение для каждой из этих ролей.

История развития

- **HTTP/0.9**

  HTTP был предложен в марте 1991 года **Тимом Бернерсом-Ли**, работавшим тогда в CERN, как механизм для доступа к документам в Интернете и облегчения навигации посредством использования гипертекста. Самая ранняя версия протокола HTTP/0.9 была впервые опубликована в январе 1992 г. (хотя реализация датируется 1990 годом). Спецификация протокола привела к упорядочению правил взаимодействия между клиентами и серверами HTTP, а также чёткому разделению функций между этими двумя компонентами. Были задокументированы основные синтаксические и семантические положения.
- **HTTP/1.0**

  В мае 1996 года для практической реализации HTTP был выпущен информационный документ **RFC 1945**, что послужило основой для реализации большинства компонентов HTTP/1.0.
- **HTTP/1.1**

  «Текущая» версия протокола, принята в июне 1999 года. Новым в этой версии был режим «постоянного соединения»: TCP-соединение может оставаться открытым после отправки ответа на запрос, что позволяет посылать несколько запросов за одно соединение. Клиент теперь обязан посылать информацию об имени хоста, к которому он обращается, что сделало возможной более простую организацию виртуального хостинга.

Структура протокола

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. **Стартовая строка** (англ. Starting line) — определяет тип сообщения;
2. **Заголовки** (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. **Тело сообщения** (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа только тело сообщения.

Стартовая строка

Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так:

GET URI — для версии протокола 0.9.

Метод URI HTTP/Версия — для остальных версий.

Здесь:

* **Метод (англ. Method)** — название запроса, одно слово заглавными буквами. В версии HTTP 0.9 использовался только метод GET, список запросов для версии 1.1 представлен ниже.
* **URI** определяет путь к запрашиваемому документу.
* **Версия (англ. Version)** — пара разделённых точкой арабских цифр. Например: 1.0.

Для запроса страницы, клиент должен передать строку:

GET /net/index.html HTTP/1.0

Стартовая строка

Стартовая строка ответа сервера имеет следующий формат:

HTTP/Версия КодСостояния Пояснение

Здесь:

* **Версия** — пара разделённых точкой арабских цифр как в запросе.
* **КодСостояния (англ. Status Code)** — три арабские цифры. По коду статуса определяется дальнейшее содержимое сообщения и поведение клиента.
* **Пояснение (англ. Reason Phrase)** — текстовое короткое пояснение к коду ответа для пользователя. Никак не влияет на сообщение и является необязательным.

Например, на предыдущий наш запрос клиентом страницы сервер ответил строкой:

HTTP/1.0 200 OK

Методы

Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Обратите внимание, что 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 не применим к конкретному ресурсу, то возвращается сообщение с кодом **405 (Method Not Allowed).** В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

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

Метод **OPTIONS**

Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. В ответ серверу следует включить заголовок Allow со списком поддерживаемых методов. Также в заголовки ответа может включаться информация о поддерживаемых расширениях.

Предполагается, что запрос клиента может содержать тело сообщения для указания интересующих его сведений. Формат тела и порядок работы с ним в настоящий момент не определён. Сервер пока должен его игнорировать. Аналогичная ситуация и с телом в ответе сервера.

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 этого метода не кэшируется.

Метод **GET**

Используется для запроса содержимого указанного ресурса. С помощью **метода GET** можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:

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

Согласно стандарту HTTP, запросы типа GET считаются идемпотентными — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET.

Кроме обычного метода GET, различают ещё **условный GET** и **частичный GET**. Условные запросы GET содержат заголовки **If-Modified-Since, If-Match, If-Range** и подобные. Частичные GET содержат в запросе **Range**. Порядок выполнения подобных запросов определён стандартами отдельно.

Метод **HEAD**

Аналогичен методу 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.

Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.

Метод **POST**

Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в **HTML-форму**, после чего они передаются серверу методом **POST** и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер.

В отличие от метода **GET**, метод **POST** не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария).

При результатах выполнения **200 (Ok)** и **204 (No Content)** в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ **201 (Created)** с указанием URI нового ресурса в заголовке Location.

Сообщение ответа сервера на выполнение метода POST не кэшируется.

Метод **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).** Сервер не должен игнорировать некорректные заголовки Content-* передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки **501 (Not Implemented).**

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.

Сообщения ответов сервера на метод PUT не кэшируются.

Дополнительные методы:

- **PATCH**
  Аналогично PUT, но применяется только к фрагменту ресурса.

- **DELETE**
  Удаляет указанный ресурс.

- **TRACE**
  Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные сервера добавляют или изменяют в запросе.

- **LINK**
  Устанавливает связь указанного ресурса с другими.

- **UNLINK**
  Убирает связь указанного ресурса с другими.

Коды состояния

Код состояния является частью первой строки ответа сервера. Он представляет собой целое число из трех арабских цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.

Примеры:

200 Ok

201 Webpage Created

403 Access allowed only for registered users

507 Insufficient Storage

Классы кодов состояния

- **1xx Informational (русск. Информационный)**

  В этот класс выделены коды, информирующие о процессе передачи. В HTTP/1.0 сообщения с такими кодами должны игнорироваться. В HTTP/1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-сервера подобные сообщения должны отправлять дальше от сервера к клиенту.

Классы кодов состояния

- **2xx Success (русск. Успешно)**

  Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.

Классы кодов состояния

- **3xx Redirection (русск. Перенаправление)**

  Коды класса 3xx сообщают клиенту что для успешного выполнения операции необходимо сделать другой запрос (как правило по другому URI). Из данного класса пять кодов 301, 302, 303, 305 и 307 относятся непосредственно к перенаправлениям (жарг. редирект). Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location. При этом допускается использование фрагментов в целевом URI.

Классы кодов состояния

- **4xx Client Error (русск. Ошибка клиента)**

  Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.

  Для запоминания значений кодов с 400 по 417 существуют приёмы иллюстративной мнемотехники.

Классы кодов состояния

- **5xx Server Error (русск. Ошибка сервера)**

  Коды 5xx выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

Заголовки

Заголовки HTTP (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA (см. RFC 822). Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Примеры заголовков:

Server: Apache/2.2.11 (Win32) PHP/5.3.0

Last-Modified: Sat, 16 Jan 2017 21:16:42 GMT

Content-Type: text/plain; charset=windows-1251

Content-Language: ru

В примере выше каждая строка представляет собой один заголовок. При этом то, что находится до первого двоеточия, называется **именем** (англ. name), а что после неё — **значением** (англ. value).

Основные группы заголовков

1. **General Headers** (русск. Основные заголовки) — должны включаться в любое сообщение клиента и сервера.
2. **Request Headers** (русск. Заголовки запроса) — используются только в запросах клиента.
3. **Response Headers** (русск. Заголовки ответа) — только для ответов от сервера.
4. **Entity Headers** (русск. Заголовки сущности) — сопровождают каждую сущность сообщения.

   Именно в таком порядке рекомендуется посылать заголовки получателю.

Примеры диалогов HTTP

Обычный GET-запрос

Запрос клиента:

GET /wiki/*страница* HTTP/1.1 Host: ru.wikipedia.org User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5 Accept: text/html Connection: close

Ответ сервера:

HTTP/1.0 200 OK
Date: Wed, 11 Feb 2016 11:20:59 GMT
Server: Apache
X-Powered-By: PHP/5.2.4-2ubuntu5wm1
Last-Modified: Wed, 11 Feb 2016 11:20:59 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close

(далее следует запрошенная страница в HTML)

Перенаправления

Запрос:

GET /about.html HTTP/1.1 Host: www.example-corp.com
User-Agent: MyLonelyBrowser/5.0

Ответ:

HTTP/1.x 301 Moved Permanently
Location: http://www.example.com/about.html#contacts
Date: Thu, 19 Feb 2016 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://www.example.com/about.html#contacts">Click
here</body></html>

Докачка и фрагментарное скачивание

- Допустим, вымышленная организация предлагает скачать с сайта видео прошедшей конференции по адресу http://example.org/conf-2016.avi объёмом примерно 160 МБ. Рассмотрим, как происходит докачивание этого файла в случае сбоя и как менеджер закачек организовал бы многопоточную загрузку нескольких фрагментов.
- В обоих случаях клиенты произведут свой первый запрос наподобие этого:

GET /conf-2016.avi HTTP/1.0

Host: example.org

Accept: */*

User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 7)

Referer: http://example.org/

- Заголовок Referer указывает, что файл был запрошен с главной страницы сайта. Менеджеры закачек обычно тоже его указывают, чтобы эмулировать переход со страницы сайта. Без него сервер может ответить 403 (Access Forbidden), если не допускаются запросы с других сайтов. В нашем случае сервер вернул успешный ответ:

HTTP/1.1 200 OK
Date: Thu, 19 Feb 2016 12:27:04 GMT
Server: Apache/2.2.3
Last-Modified: Wed, 18 Jun 2016 16:05:58 GMT
ETag: "56d-9989200-1132c580"
Content-Type: video/x-msvideo
Content-Length: 160993792
Accept-Ranges: bytes
Connection: close
(пустая строка)
(двоичное содержимое всего файла)

Заголовок Accept-Ranges информирует клиента о том, что он может запрашивать у сервера фрагменты, указывая их смещения от начала файла в байтах. Если этот заголовок отсутствует, то клиент может предупредить пользователя, что докачать файл, скорее всего, не удастся. Исходя из значения заголовка Content-Length, менеджер закачек поделит весь объём на равные фрагменты и запросит их по отдельности, организовав несколько потоков. Если сервер не укажет размер, то клиенту параллельное скачивание реализовать не удастся, но при этом он сможет докачивать файл, пока сервер не ответит 416 (Requested Range Not Satisfiable).

Допустим, на 84-ом мегабайте соединение с Интернетом прервалось и процесс загрузки приостановился. Когда соединение с Интернетом было восстановлено, браузер автоматически послал новый запрос на сервер, но с указанием выдать содержимое с 84-ого мегабайта:

```
GET /conf-2016.avi HTTP/1.0
Host: example.org
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 7)
Range: bytes=88080384-
Referer: http://example.org/
```

Сервер не обязан помнить, какие и от кого запросы были до этого, и поэтому клиент снова вставил заголовок Referer, как будто это его самый первый запрос. Указанное значение заголовка Range говорит серверу — «выдай содержимое от 88080384-ого байта до самого конца». В связи с этим сервер вернёт ответ:

HTTP/1.1 206 Partial Content
Date: Thu, 19 Feb 2016 12:27:08 GMT
Server: Apache/2.2.3
Last-Modified: Wed, 18 Jun 2016 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-ого мегабайта)

Особенности протокола

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 в заголовке строчку «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).

Особенности протокола

Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

Перечисленные особенности HTTP позволили создавать поисковые машины (первой из которых стала AltaVista, созданная фирмой DEC), форумы и Internet-магазины. Это превратило Internet из «академической игрушки» в «коммерческий сервис»: появились компании, основным полем деятельности которых стало предоставление доступа в Internet (компании-провайдеры) и создание сайтов.