Содержание

Предисловие

Я рано заинтересовался темой HTTP/2. Появление новой технологии было интригующим. Она обещала почти неограниченный прирост производительности, потенциально устраняя необходимость в некоторых запутанных обходных путях, которые вынуждены были использовать веб-разработчики. Однако на практике все оказалось немного сложнее. Я потратил некоторое время на то, чтобы выяснить, как применить ее в моем сервере Apache. А затем я изо всех сил пытался объяснить увиденные мной результаты влияния на производительность. Ситуацию осложняло отсутствие документации. Я сделал пару публикаций в своем блоге о том, как применять эту технологию, и эти публикации пользовались популярностью. Вместе с этим я начал принимать участие в некоторых проектах, связанных с HTTP/2, на веб-сервисе GitHub, а также просматривать соответствующие темы в системе Stack Overflow и помогать тем, у кого были проблемы, схожие с моими. Когда со мной связалось издательство Manning и предложило начать работу над книгой о HTTP/2, я воспользовался этой возможностью. Я не участвовал в разработке этой технологии, однако я чувствовал, что могу помочь многим веб-разработчикам, столкнувшихся с трудностями. Они, как и я, слышали об этой технологии, но не обладали достаточными знаниями, чтобы применить ее.

За те полтора года, которые я писал эту книгу, технология HTTP/2 стала широко востребованной, и сейчас ее используют в разработке все большего количества веб-сайтов. Некоторые проблемы развертывания решались по мере обновления программного обеспечения. Я надеюсь, что некоторые из проблем, описанных в этой книге, со временем уйдут в прошлое. Я предполагаю, что нам потребуется еще несколько лет, чтобы полностью доработать HTTP/2.

Если вы научитесь использовать протокол HTTP/2, он обеспечит вам мгновенный прирост производительности веб-приложений без необходимости настройки или детального понимания. Однако в этой жизни ничего не дается даром, поэтому владельцам сайтов пойдет на пользу глубокое понимание всех тонкостей и нюансов протокола и его развертывания. Сегодня оптимизация производительности веб-приложений

находится в центре внимания, и HTTP/2 является еще одним инструментом, который даст нам новые интересные методики и возможности как сейчас, так и в будущем.

В сети доступно огромное количество информации для тех, у кого есть время и желание искать, фильтровать и понимать ее. Весьма приятно читать различные мнения и даже общаться непосредственно с разработчиками и исполнителями протоколов. Однако тема HTTP/2 очень обширна, и только объем и глубина книжного формата дают мне возможность полностью объяснить технологию, затрагивая связанные с ней темы, а также дать вам ссылки для дальнейшего изучения, если что-то вызовет у вас интерес. Я надеюсь, что в этой книге достиг своей цели.

Об этой книге

Я написал эту книгу для того, чтобы доходчиво и на реальных примерах объяснить читателю, как протокол работает на практике. Обычно спецификации протоколов изложены довольно сухо и сложны для понимания, поэтому цель данной книги – объяснить все детали на примерах, которые будут понятны каждому пользователю.

Кому следует прочесть эту книгу?

Эта книга создана для веб-разработчиков, администраторов веб-сайтов и тех, кто просто заинтересован в Internet-технологиях. Книга призвана полностью осветить тему HTTP/2 и все тонкости, связанные с протоколом. В блогах можно найти множество публикаций на эту тему, однако большинство из них написано для профессионалов или детально освещает лишь какой-либо отдельный аспект. Данная книга ориентирована на охват сразу всех аспектов функционирования протокола. Она должна подготовить читателя и помочь ему разобраться в спецификации протокола. Также, если после прочтения этой книги читатель захочет изучить тему более детально, ему будет гораздо проще понять некоторые сложные публикации в блогах. HTTP/2 был создан прежде всего для повышения производительности, поэтому любой, кто интересуется оптимизацией веб-производительности, обязательно извлечет из этой книги полезную информацию и сделает для себя некоторые выводы. Кроме того, в книге содержится множество ссылок, которые могут пригодиться для дальнейшего изучения темы.

Как устроена эта книга

Книга состоит из 10 глав и разделена на 4 части.

В части I говорится о предыстории появления протокола, а также о том, почему необходимо перейти на HTTP/2 и как это осуществить:

- в главе 1 дается *предыстория, необходимая для полного понимания книги*. Такой подход дает возможность пользователям, обладающим лишь базовыми знаниями, хорошо разобраться в теме;

- в главе 2 мы рассмотрим *проблемы с HTTP/1.1 и причины, по которым необходим был переход на HTTP/2*;
- в главе 3 мы *обсудим способы переноса вашего веб-сайта на HTTP/2 и некоторые сложности, связанные с этим процессом.* Эта глава дополняется приложением, содержащим инструкции по установке таких популярных веб-сервисов, как Apache, nginx, и IIS.

Часть II содержит еще больше полезной информации. В ней мы рассмотрим непосредственно сам протокол и его значение для практик веб-разработки:

- в главе 4 дается *описание характеристик протокола HTTP/2, порядок установки HTTP/2 соединения, а также рассказывается об основном формате фреймов HTTP/2*;
- глава 5 посвящена *push-серверу HTTP/2*, который является новой частью протокола. Он позволяет владельцам веб-сайтов отправлять ресурсы до того, как их запросят браузеры;
- в главе 6 мы рассмотрим *практическое значение HTTP/2 для веб-разработки.*

Часть III содержит информацию о глубинных компонентах протокола, на которые не могут повлиять ни веб-разработчики, ни даже администраторы веб-серверов:

- в главе 7 говорится *о других моментах, касающихся спецификации HTTP/2, таких как состояние, управление потоком и механизм приоритетности. Кроме того, здесь мы рассмотрим различия между реализациями развертывания протокола HTTP/2*;
- в главе 8 подробно рассматривается *протокол HPACK*, созданный для сжатия заголовков HTTP в HTTP/2.

В части IV мы поговорим о перспективах дальнейшего развития протокола HTTP:

- глава 9 посвящена *TCP, QUIC и HTTP/3.* Технологии постоянно развиваются, и теперь, когда протокол HTTP/2 стал доступен, разработчики уже ищут способы его улучшить. В этой главе мы обсудим недоработки в протоколе HTTP/2 и способы их устранения в его преемнике – HTTP/3;
- в главе 10 *рассмотрим другие способы улучшения HTTP помимо протокола HTTP/3*, а также поразмышляем о проблемах, которые были обнаружены во время разработки и оформления стандарта HTTP/2, и о том, актуальны ли эти проблемы в реальной практике.

После изучения этой книги читатели приобретут четкое представление о том, что представляет из себя протокол HTTP/2 и связанные с ним технологии. Также книга поможет лучше разобраться в оптимизации веб-производительности. Кроме того, когда QUIC и HTTP/3 станут доступны широкой публике, люди, познакомившиеся с этой книгой, будут готовы к работе с ними.

Примеры кода

Эта книга отличается от других книг технической направленности. Здесь вы не увидите большого количества кода, потому что книга посвящена именно протоколу, а не языку программирования. Книга нацелена на то, чтобы познакомить вас с методиками высокого уровня, которые вы сможете применить к любому веб-серверу или языку программирования, используемому для обслуживания страниц в сети. Однако в книге есть несколько примеров на языках NodeJS и Perl, а также фрагменты конфигурации веб-сервера.

Для того чтобы отделить программный код и фрагменты конфигурации от обычного текста, в книге мы используем моноширинный шрифт. В некоторых местах мы выделяем части кода жирным шрифтом, чтобы показать изменения, произошедшие в результате предыдущих шагов, описанных в главе, например когда к строке кода добавляется новая порция информации.

Все программные коды, используемые в примерах, доступны для загрузки с веб-сайтов издательства по адресу https://www.manning.com/books/http2-in-action или GitHub по адресу https://github.com/bazzadp/http2-in-action.

Онлайн-ресурсы

Остались вопросы?

- Официальная страница HTTP/2 доступна по адресу https://http2.github.io/. На этой странице вы сможете найти информацию о спецификации HTTP/2 и HPACK, реализации HTTP/2 и ответы на часто задаваемые вопросы.
- Официальная веб-страница рабочей группы HTTP доступна по адресу: https://httpwg.org/. Большая часть информации о работе группы находится в открытом доступе на веб-странице GitHub https://github.com/httpwg/ и в списке рассылок (https://lists.w3.org/Archives/Public/ietf-http-wg/).
- Вы можете найти дополнительную информацию по тегу HTTP/2 в Stack Overflow: https://stackoverflow.com/questions/tagged/http2. Здесь же вы имеются ответы на вопросы от автора.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Барри Поллард (Barry Pollard) – профессиональный разработчик программного обеспечения. Почти 20 лет он занимается вопросами разработки и поддержки программного обеспечения и IT-инфраструктуры. Он проявляет большой интерес к веб-технологиям, повышению производительности, безопасности и практическому использованию технологий. Вы можете найти блог Барри Полларда на сайте https://www.tunetheweb.com. Кроме того, он ведет Твиттер (@tunetheweb).

Часть I

Переход на HTTP/2

Чтобы понять, почему в области оптимизации производительности веб-приложений существует такой ажиотаж вокруг протокола HTTP/2, сначала вам необходимо узнать, зачем он нужен и какие проблемы он решает. Поэтому первая часть этой книги знакомит с HTTP/1 читателей, которые не знакомы с тем, что это такое и как это работает. Затем мы объясним, почему было необходимо создание версии 2. Сначала поговорим о том, как работает HTTP/2 на более высоком уровне, а разбор работы нижних уровней оставим на потом. Вместо этого в конце первой части мы расскажем о различных методах, которые вы можете использовать для развертывания HTTP/2 на своем сайте.

1 Веб-технологии и HTTP

В этой главе мы рассмотрим:

- как браузер загружает веб-страницу;
- что такое HTTP и как появился HTTP/1.1;
- основы HTTPS;
- основные инструменты HTTP.

В этой главе говорится о том, как устроен веб в современном мире, а также дается объяснение некоторых ключевых концепций, которые помогут вам лучше понять смысл последующих глав книги. Затем мы поговорим о самом понятии HTTP и об истории версий протоколов, предшествующих ему. Мы полагаем, что многие читатели хотя бы немного знакомы с большей частью информации, заключенной в этой главе. Однако мы все же рекомендуем прочесть ее, чтобы освежить свои базовые знания.

ПРИМЕЧАНИЕ РЕДАКТОРА ПЕРЕВОДА Автор книги обращает наше внимание на различие между Всемирной паутиной (World Wide Web), представляющая собой вместилище общедоступного контента – сайтов, файлов, потоковых ресурсов и т. д., и сетью передачи данных Internet, которая представляет собой маршрутизируемые каналы передачи данных и набор протоколов разного уровня. Поскольку в этой книге речь идет именно о втором случае, мы, как и автор, далее используем термин «Internet» в его техническом понимании, а интернетом называем только Всемирную паутину.

1.1 О том, как работает сеть

Internet стал неотъемлемой частью повседневной жизни. В нем мы совершаем покупки, производим банковские операции, а также общаемся и развлекаемся. По мере развития *интернета вещей* (IoT – Internet of Things) к сети подключается все больше и больше устройств, что обеспечивает нам возможность получения удаленного доступа к ним. Реализация такого доступа стала возможной благодаря разработке нескольких технологий, в числе которых протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP). Данный протокол является ключевым методом запроса удаленного доступа к веб-приложениям и ресурсам. Большинство людей умеет выходить в Internet через веб-браузер. Однако не каждый знает, как работает эта технология на самом деле. Многие люди не имеют понятия о том, почему HTTP является основной частью веба и почему следующая версия (HTTP/2) вызывает такой ажиотаж в веб-сообществе.

1.1.1 Internet и Всемирная паутина

Многие люди считают, что понятия Internet и «Всемирная паутина» (или просто «веб») являются синонимами, но на самом деле это два разных термина, которые очень важно различать.

Internet представляет собой глобальную структуру объединенных компьютерных сетей, использующих общий интернет-протокол (Internet Protocol, IP) для маршрутизации сообщений. Он состоит из множества сервисов, таких как веб, электронная почта, система предоставления общего доступа к файлам и интернет-телефония. Таким образом, веб является лишь одной из систем, существующих в сети Internet, хотя и самой заметной. Люди часто пользуются электронной почтой с помощью внешних веб-интерфейсов (например, Gmail, Hotmail и Yahoo!) и поэтому ошибочно отождествляют понятия «веб» и Internet.

HTTP – это протокол, при помощи которого веб-браузер запрашивает и получает веб-страницы с сервера. Это одна из трех основных технологий, созданных Тимом Бернерсом-Ли (Tim Berners-Lee), который также является автором Всемирной паутины (worldwide web, WWW), уникальных идентификаторов ресурсов (ставших основой для унифицированных указателей ресурсов или URL-адресов) и языка гипертекстовой разметки (Hypertext Markup Language, HTML). Другие системы в Internet созданы по своим собственным протоколам и стандартам, определяющим их работу, а также маршрутизацию их сообщений (например, маршрутизация электронной почты происходит посредством протоколов SMTP, IMAP и POP). HTTP связан именно с вебом, однако эта граница постепенно размывается. Появление сервисов без традиционных клиентских веб-интерфейсов, основанных на HTTP, означает, что определить границы Всемирной паутины становится все сложнее и сложнее! Такие сервисы (известные под аббревиатурами REST и SOAP) могут быть применимы

как к веб-страницам, так и к другим ресурсам, размещенным в Internet (например, к мобильным приложениям). Интернет вещей представляет собой совокупность устройств, предоставляющих пользователям определенные функции. С ними могут взаимодействовать другие устройства (компьютеры, мобильные приложения и даже другие устройства интернета вещей). В большинстве случаев связь между ними происходит посредством HTTP-запросов. Таким образом, посредством отправки HTTP-сообщения вы можете включить или выключить лампочку, например, с помощью приложения на своем мобильном телефоне.

Хотя в структуре Internet существует огромное количество сервисов, большая их часть со временем используется все реже и реже, в то время как количество пользователей Всемирной сети продолжает расти. Некоторые читатели вспомнят такие аббревиатуры, как BBS и IRC. Сегодня они считаются устаревшими, а на их место пришли веб-форумы, социальные сети и мессенджеры.

Как уже было сказано, термин «Всемирная паутина» часто ошибочно отождествляется с понятием Internet. Однако непрерывное развитие Всемирной сети (а также специально созданного для нее протокола HTTP) может означать, что очень скоро такое понимание окажется не столь далеким от истины, как раньше.

1.1.2 Что происходит, когда вы просматриваете веб-страницы?

Сейчас мы возвращаемся к первоначальному (и основному) назначению HTTP: передача веб-запросов. Когда вы открываете веб-сайт в своем любимом браузере на стационарном или портативном компьютере, планшете, мобильном телефоне или на любом другом из множества устройств, обеспечивающих доступ в Internet, происходит очень много процессов. Чтобы извлечь максимум пользы из этой книги, вам необходимо понять, какие процессы происходят, когда мы загружаем веб-страницу.

Предположим, что вы запускаете браузер и переходите на сайт www.google.com. В течение нескольких секунд произойдет процесс, показанный на рис. 1.1.

1. Браузер запросит реальный адрес www.google.com с сервера системы доменных имен (Domain Name System, DNS), который переведет понятное человеку имя www.google.com в удобный для машины IP-адрес.

   Если представить, что IP-адрес – это телефонный номер, то DNS – это телефонная книга. Этот IP-адрес будет являться либо адресом более старого формата IPv4, который относительно понятен человеку (например, 216.58.192.4), либо адресом нового формата IPv6, который могут обрабатывать только машины (например, 2607:f8b0:4005:801:0:0:0:2004). Когда в городе заканчиваются свободные телефонные номера, приходится менять телефонный код города. Приблизительно по этой же причине введен протокол IPv6. Он не-

обходим, чтобы справиться с ростом количества устройств, подключающихся к Internet сейчас и в будущем.

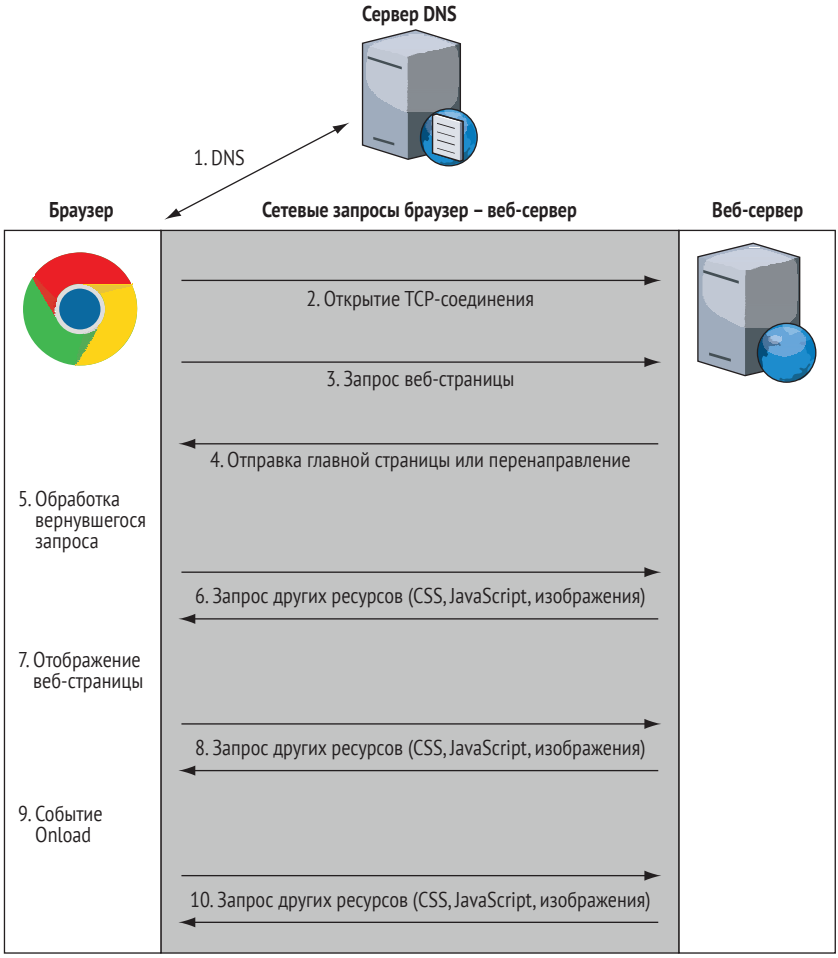

Рис. 1.1 Типичное взаимодействие при просмотре веб-страницы

Имейте в виду, что по причине глобального характера сети Internet крупные компании часто имеют несколько серверов по всему миру. Когда вы запрашиваете IP-адрес у DNS, он обычно предоставляет IP-адрес ближайшего сервера, благодаря чему вы получаете наиболее быстрый доступ к веб-страницам. Например, ответ на запрос IP-адреса для www.google.com в Америке и в Европе будет отличаться. Поэтому не волнуйтесь, если ваши значения IP-адресов для www.google.com будут отличаться от тех, что вы найдете в нашей книге.

Где же версия IPv5?

Если Internet-протокол версии 4 (IPv4) был заменен версией 6 (IPv6), то где же тогда версия 5? И почему вы никогда не слышали об IPv1 или IPv3?

Первые четыре бита в IP-пакете используются для хранения версии протокола. В теории максимально возможная версия – это версия 15. До того как IPv4 начали использовать повсеместно, существовало четыре экспериментальных версии (версия 0 – версия 3). Однако ни для одной из них, кроме версии 4, не было создано официального стандарта[a]. Затем создали версию 5, которая предназначалась для протокола реального времени Internet Stream Protocol, обеспечивающего потоковую передачу аудио и видео в реальном времени, подобно современной IP-телефонии (Voice over IP, VoIP). Однако версия IPv5 так и не стала популярной, так как имела те же ограничения адресного пространства, что и версия 4. Когда появилась версия 6, работа над IPv5 была остановлена, и версия 6 стала преемником IPv4. Судя по всему, изначально IPv6 была названа версией 7, потому что многие ошибочно полагали, что версия 6 уже использована[b]. Номера 7, 8 и 9 также применялись для нумерации версий, но эти версии больше не используются. Если когда-нибудь и появится преемник IPv6, то это, скорее всего, будет IPv10 или более поздняя версия. В связи с этим, несомненно, возникнут вопросы, подобные тем, с которых начинается это примечание!

[a] Стандарт протокола https://tools.ietf.org/html/rfc760 позже был обновлен и заменен на https://tools.ietf.org/html/rfc791.

[b] Обратите внимание на информацию в публикации https://archive.is/QqU73#selection-417.1-417.15.

2 Через веб-браузер ваш компьютер устанавливает TCP-соединение[1] по IP-адресу через стандартный сетевой порт (порт 80)[2] или стандартный защищенный сетевой порт (порт 443).

Передача информационных потоков в Internet осуществляется с помощью протокола IP. Протокол TCP обеспечивает надежность при передаче данных, а также возможность повторной передачи («Здравствуйте, вы все поняли?», «Нет, не могли бы вы повторить последний бит, пожалуйста?»). Эти две технологии часто используются вместе, поэтому их называют TCP/IP. Именно на основе этих протоколов создана большая часть сервисов в Internet.

Один сервер может использоваться несколькими службами (например, электронная почта, FTP, HTTP и HTTPS [HTTP Secure]-серверы). Порт позволяет различным службам размещаться под

[1] Google начал экспериментировать с протоколом QUIC, поэтому, если вы подключаетесь к сайту Google из Chrome, то может быть задействована эта технология. Я расскажу про QUIC в главе 9.

[2] Некоторые веб-сайты, в том числе Google, используют технологию HSTS для автоматической активации защищенного HTTP-соединения (HTTPS) через порт 443, поэтому, даже если вы попытаетесь подключиться через HTTP, соединение автоматически обновится до HTTPS до отправки запроса.

одним IP-адресом, так же как, например, в компании может быть добавочный номер телефона для каждого сотрудника.

3 После того как браузер подключился к веб-серверу, он начинает отправлять запросы веб-сайту. На этом этапе в дело вступает протокол HTTP. Однако мы рассмотрим порядок его работы в следующем разделе. А сейчас сосредоточимся на работе браузера и рассмотрим пример, где с помощью HTTP он запрашивает у сервера Google домашнюю страницу.

ПРИМЕЧАНИЕ На этом этапе ваш браузер автоматически исправляет сокращенный адрес (www.google.com) на синтаксически верный URL-адрес (http://www.google.com). В абсолютном полном URL-адресе указывается номер порта (например, http://www.google.com:80). Если используются стандартные порты (80 для HTTP и 443 для HTTPS), то веб-браузер скроет их номера. Если используются нестандартные порты, их номера будут отображаться в URL. Например, некоторые системы, особенно в среде разработки, используют порт 8080 для HTTP или 8443 для HTTPS.

Если используется протокол HTTPS (более подробно мы рассмотрим его в разделе 1.4), то для настройки шифрования, защищающего соединение, требуются дополнительные шаги.

4 Сервер Google отвечает вне зависимости от типа URL-адреса, который вы запрашиваете. Как правило, ответ приходит в виде текста веб-страницы в формате HTML. HTML – *это стандартизированная, структурированная система разметки текстового содержимого веб-страниц*. Документ на языке HTML обычно представляет собой набор элементов, начало и конец которых определяется HTML-тегами и ссылками на блоки другой информации, необходимой для создания мультимедийных веб-страниц, которые вы привыкли видеть (каскадные таблицы стилей [CSS], код JavaScript, изображения, шрифты и т. д.).

Однако иногда вместо HTML-страницы ответом может быть перенаправление на верный адрес. Например, Google работает только на HTTPS, поэтому, если вы запросите URL http://www.google.com, ответом будет специальная HTTP-инструкция (обычно код такого ответа `301` или `302`), которая перенаправит вас на новый адрес https://www.google.com. Такой ответ запускает процессы некоторых или всех предыдущих шагов снова, в зависимости от того, как выглядит адрес перенаправления. Он может иметь другой порт на другом сервере, другой порт на том же сервере (например, перенаправление на HTTPS) или это может быть другая страница на том же сервере и порте.

При возникновении ошибки вы увидите на экране ответ в виде HTTP-кода, самым известным из которых является `404 Not Found`.

5 Затем браузер обрабатывает возвращенный запрос. Браузер рассчитан на ответы в формате HTML, поэтому он начинает анализи-

ровать HTML-код и строит в памяти объектную модель документа (document object model, DOM), которая отображает внутреннее строение страницы. Скорее всего, в процессе анализа веб-браузер найдет и другие ресурсы, необходимые для правильного отображения страницы (например, CSS, JavaScript и изображения).

6 Веб-браузер запросит необходимые ему дополнительные ресурсы. Веб-страницы Google используют небольшое количество ресурсов. *На момент написания этой книги их требуется всего 16*. Каждый из этих ресурсов запрашивается аналогичным образом, следуя шагам 1–6, так как эти ресурсы в свою очередь могут запрашивать другие ресурсы. Как правило, страницы обычных веб-сайтов более разнообразны, чем страницы Google. Такие веб-сайты нуждаются в 75 ресурсах[1] и зачастую из разных доменов, поэтому шаги 1–6 должны быть выполнены для каждого ресурса. Именно из-за таких ситуаций и замедляется работа браузера. Протокол HTTP/2 помогает оптимизировать запрос дополнительных ресурсов. Об этом мы поговорим в следующих главах.

7 Когда браузер получает достаточно критически необходимых ресурсов, он начинает отображать страницу на экране. Старт рендеринга веб-страницы является не таким простым процессом, как кажется. Браузеру требуется много времени для загрузки ресурсов, необходимых для отображения веб-страницы. Поэтому соединение становится еще более медленным и не дает ожидаемых пользователем результатов. Если веб-браузер начнет отображать страницу слишком рано, она будет «прыгать» по мере загрузки большего количества контента. Особенно раздражает, когда страница начинает «прыгать» после того, как вы прочитали уже половину статьи. Четкое понимание технологий, на которых основан веб, особенно HTTP и HTML/CSS/JavaScript, помогает владельцам веб-сайтов убрать эти раздражающие скачки и оптимизировать загрузку страниц, однако многие из них все еще этого не делают.

8 После отображения страницы веб-браузер продолжает в фоновом режиме загружать другие ресурсы, необходимые странице, и обновляет страницу по мере их обработки. Он загружает второстепенные элементы, такие как изображения и скрипты отслеживания рекламы. Именно поэтому бывает так, что, когда вы загружаете веб-страницу, на ней какое-то время не отображаются изображения (особенно при медленном соединении), а затем, по мере их загрузки, они появляются.

9 Когда страница полностью загружена, браузер останавливает значок загрузки (в большинстве браузеров вращающийся значок в адресной строке или рядом с ней) и запускает событие `OnLoad` JavaScript, которое является маркером для JavaScript и означает, что страница готова к работе.

---

[1] https://httparchive.org/reports/page-weight#reqTotal.

10 Времена, когда веб-страница была статична, уже давно прошли, поэтому браузер продолжает отправлять запросы, даже когда она полностью загружена. Сегодня многие веб-страницы являются многофункциональными приложениями, взаимодействующими с различными серверами в Internet для отправки или загрузки дополнительного контента. Отображение контента может быть инициировано действиями пользователя. Например, вы вводите запросы в строку поиска на домашней странице Google и мгновенно видите предложенные варианты запроса без нажатия кнопки поиска. Также это могут быть действия, управляемые приложением, например автоматическое обновление ленты Facebook или Twitter без нажатия кнопки обновления. Эти действия часто происходят в фоновом режиме. Они скрыты от вас. В качестве примера можно привести рекламные и аналитические скрипты, которые отслеживают ваши действия на сайте, для того чтобы сообщать аналитические данные владельцам веб-сайтов и/или рекламным сетям.

Как видите, после того как вы вводите URL-адрес, происходит огромное количество процессов, и в большинстве случаев они протекают очень быстро. Описание любого из этих шагов могло бы лечь в основу целой книги с различными вариациями при определенных обстоятельствах. В нашей книге мы делаем упор на шаги 3–8 (загрузка веб-сайта с помощью HTTP). Также в некоторых главах (в частности, в главе 9) рассказывается о шаге 2 (базовое сетевое соединение, используемое HTTP).

1.2 Что такое HTTP?

Предыдущий раздел освещает детали протокола HTTP, благодаря чему вы имеете представление о его функционировании в контексте всего Internet. В этом разделе мы кратко опишем, как данный протокол функционирует и где он используется.

Как уже было сказано, *HTTP расшифровывается как «протокол передачи гипертекста»*. Как следует из его названия, HTTP изначально предназначался исключительно для передачи гипертекстовых документов (документов, содержащих ссылки на другие документы). Первая версия не могла передать ничего, кроме таких документов. Название «протокол передачи гипертекста» не совсем актуально, так как разработчики быстро поняли, что протокол может быть использован для передачи других типов файлов (например, изображений). Однако, сейчас HTTP настолько распространен, что переименовывать его уже слишком поздно.

Сетевая модель TCP/IP, построенная на одном из типов физического соединения (Ethernet, Wi-FI и т. д.), обеспечивает надежное сетевое соединение, необходимое для функционирования HTTP. Структура протоколов передачи данных разделена на уровни, каждый из которых отвечает за определенный процесс. HTTP не имеет отношения к установке сетевого соединения. Несмотря на то что HTTP-приложения должны

уметь справляться с сетевыми сбоями или отключениями, сам протокол не предполагает решения этих задач.

Модель взаимодействия открытых систем (Open Systems Interconnection, OSI) – *это многоуровневая модель сетевых протоколов*. Она состоит из 7 уровней, хотя они соотносятся с сетями, и особенно с Internet-трафиком, с некоторой долей условности. TCP (Transmission Control Protocol) является рабочим протоколом как минимум двух уровней, а возможно, и трех (это зависит от того, как вы определяете эти уровни). На рис. 1.2 вы можете увидеть, как данная модель соотносится с Internet-трафиком и как в нее вписывается протокол HTTP.

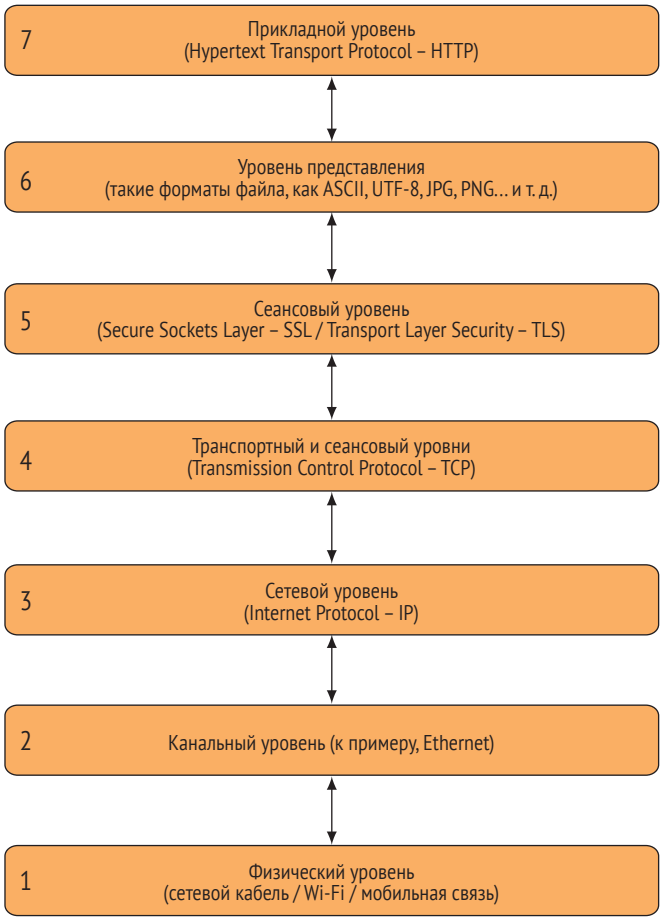

Рис. 1.2 Транспортные уровни Internet-трафика

К слову, сегодня до сих пор ведутся споры о точном определении уровней в модели. В Internet, как и в других сложных системах, не все поддается классификации, и разграничить что-либо бывает не так легко, как этого хотелось бы разработчикам. На самом деле, согласно мнению Ин-

женерного совета интернета, не обязательно делать слишком большой упор на деление модели по уровням[1]. Однако на более высоком уровне такой подход может помочь понять, как в модель вписывается HTTP, а также как она зависит от других протоколов. Многие веб-приложения используют HTTP, поэтому в таких случаях прикладной уровень может больше относиться к сетевому уровню, чем к приложениям JavaScript.

По сути, HTTP является протоколом запроса и ответа. Браузер совершает запрос к веб-серверу, используя синтаксис HTTP. Веб-сервер в свою очередь отвечает сообщением, содержащим запрошенный ресурс. HTTP широко распространен именно благодаря своей простоте. Однако, как вы увидите в последующих главах, HTTP/2 немного сложнее, чем HTTP, но в то же время он и более эффективен.

После того как открывается соединение, базовый синтаксис HTTP-запроса выглядит следующим образом:

```
GET /page.html↵,
```

где символ ↵ означает возврат каретки / начало новой строки (клавиша **Enter** или **Return**). HTTP очень прост в своей базовой форме! Вы указываете один из нескольких HTTP-методов (в данном случае `GET`), за которым следует нужный вам ресурс (`/page.html`). Помните, что на этом этапе вы уже подключились к соответствующему серверу посредством TCP/IP, поэтому просто запрашиваете нужный ресурс с этого сервера. Вас не должно беспокоить то, как работает и чем управляется это соединение.

Первая версия HTTP (0.9) поддерживала только базовый синтаксис и могла использовать только метод `GET`. В данном случае вам может быть интересно, зачем использовали метод `GET` для запроса HTTP 0/9. Здесь он кажется неуместным, но этот и другие методы появились в последующих версиях HTTP, так что спасибо изобретателям HTTP за то, что они предвидели их появление. В следующем разделе мы обсудим различные версии протокола HTTP, однако синтаксис по-прежнему соответствует формату запроса HTTP `GET`.

Рассмотрим пример из реальной жизни. Для получения HTTP-запросов веб-серверу требуется только соединение TCP/IP, и вы можете эмулировать работу браузера с помощью протокола Telnet. *Это простой протокол, который реализует соединение с сервером с помощью TCP/IP и позволяет вводить текстовые команды, а также просматривать текстовые ответы*. Данный протокол необходим при использовании HTTP, но ближе к концу главы мы рассмотрим и другие, более подходящие инструменты для просмотра HTTP-страниц. К сожалению, некоторые технологии теряют свою популярность, и Telnet является одной из них; многие операционные системы больше не включают в себя клиент Telnet по умолчанию. Возможно, вам потребуется установить клиент Telnet, чтобы попробовать реализовать некоторые простые команды HTTP, или же вы можете использовать в качестве эквивалента утилиту nc (netcat), которая уже установлена в большинстве Linux-подобных систем, вклю-

[1] https://tools.ietf.org/html/rfc3439#section-3.

чая macOS. Для рассмотрения простых примеров, которые представлены в нашей книге, она подойдет почти также, как и Telnet.

Для Windows лучше использовать утилиту PuTTY[1] вместо клиента, поставляемого в комплекте с Windows по умолчанию (который обычно не устанавливается и должен добавляться вручную). Клиент, установленный по умолчанию, имеет проблемы с отображением: например, он может не отображать то, что вы печатаете, или записывать новую информацию поверх уже имеющейся. После установки и запуска PuTTY вы увидите окно конфигурации, в котором вы можете ввести хост (www.google.com), порт (80) и тип подключения (Telnet). Убедитесь, что вы выбрали опцию **Never** (Никогда) для закрытия окна после окончания сеанса; в противном случае вы не увидите результатов. Все эти настройки вы можете наблюдать на рис. 1.3. Также обратите внимание на то, что вы можете изменить режим Telnet Negotiation на пассивный (**Connections** > **Telnet** > **Telnet Negotiation**) в случае, если у вас возникли проблемы с вводом любой из следующих команд и вы получаете сообщения о неправильном формате запроса.

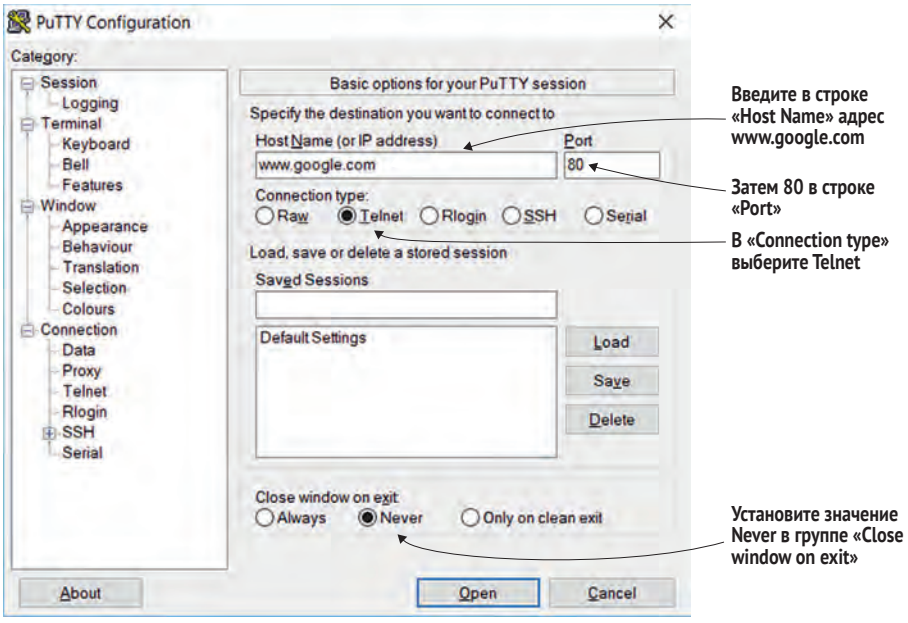

Рис. 1.3 Настройки PuTTY для соединения с Google

Если вы используете компьютер Apple Macintosh или ОС Linux, вы можете выполнить команду Telnet непосредственно из командной строки оболочки, если Telnet уже установлен:

```
$ telnet www.google.com 80
```

[1] https://www.putty.org/.

или, как я уже упоминал ранее, используйте команду `nc` таким же образом:

```
$ nc www.google.com 80
```

Когда вы открываете сеанс Telnet и устанавливаете соединение, вы видите пустой экран или, в зависимости от вашего приложения Telnet, некоторые команды, такие как:

```
Trying 216.58.193.68...
Connected to www.google.com.
Escape character is '^]'.
```

Независимо от того, отображается это сообщение или нет, вы должны иметь возможность вводить свои HTTP-команды, поэтому введите `GET/`, а затем нажмите клавишу ввода, которая сообщает серверу Google, что вы ищете страницу по умолчанию (/) и (поскольку вы не указали версию HTTP) хотите использовать HTTP/0.9. Обратите внимание, что некоторые клиенты Telnet не повторяют то, что вы печатаете по умолчанию (особенно клиент Telnet по умолчанию в комплекте Windows, как я уже упоминал ранее), поэтому может быть трудно точно увидеть, что вы печатаете. Но вы все равно должны отправлять команды.

Использование Telnet через прокси-сервер организации

Если у вашего компьютера нет прямого доступа в Internet, вы не сможете подключиться к Google напрямую с помощью Telnet. Этот сценарий часто встречается в корпоративных сетях, где для ограничения прямого доступа используется прокси (мы поговорим о прокси-серверах в главе 3). В этом случае вы можете использовать в качестве примера один из ваших внутренних веб-серверов (например, ваш сайт во внутренней сети), а не Google. В разделе 1.5.3 мы рассмотрим другие инструменты, которые могут работать с прокси, но пока стоит сосредоточиться на Telnet.

Скорее всего, сервер Google ответит, используя HTTP/1.0, несмотря на то, что вы запросили HTTP/0.9 по умолчанию (так как ни один сервер больше не использует HTTP/0.9). В ответ появится код ответа `200` (если команда была выполнена) или `302` (если сервер хочет перенаправить вас в другое место). Затем соединение закроется. Более подробно я расскажу об этом процессе в следующем разделе, поэтому не стоит сейчас зацикливаться на этих деталях.

Ниже приведен один из некоторых ответов из командной строки на сервере Linux, выделенный жирным шрифтом. Стоит отметить, что возвращаемый HTML-контент не отображается полностью для краткости:

```
$ telnet www.google.com 80
Trying 172.217.3.196...
Connected to www.google.com.
Escape character is '^]'.
GET /
HTTP/1.0 200 OK
```

```
Date: Sun, 10 Sep 2017 16:20:09 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See
     https://www.google.com/support/accounts/answer/151657?hl=en for more info.
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie:
     NID=111=QIMb1TZHhHGXEPjUXqbHChZGCcVLFQOvmqjNcUIejUXqbHChZKtrF4Hf4x4DVjTb01R
     8DWShPlu6_aQ-AnPXgONzEoGOpapm_VOTW0Y8TWVpNap_1234567890-p2g; expires=Mon,
     12-Mar-2018 16:20:09 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"
     lang="en"><head><meta content="Search the world's information, including
     webpages, images, videos and more. Google has many special features to help
     you find exactly what you're looking for." name="description

…и т. д.

</script></div></body></html>Connection closed by foreign host.
```

Если вы находитесь за пределами Соединенных Штатов, вы можете увидеть перенаправление на ваш местный сайт Google вместо указанного здесь адреса. Например, если вы находитесь в Ирландии, Google отправляет ответ `302` и советует браузеру перейти на Google Ireland (http://www.google.ie), как показано здесь:

```
GET /
HTTP/1.0 302 Found
Location: http://www.google.ie/?gws_rd=cr&dcr=0&ei=BWe1WYrf123456qpIbwDg
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See
     https://www.google.com/support/accounts/answer/151657?hl=en for more info."
Date: Sun, 10 Sep 2017 16:23:33 GMT
Server: gws
Content-Length: 268
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: NID=111=ff1KAwIMjt3X4MEg_KzqR_9eAG78CWNGEFlDG0XIf7dLZsQeLerX
     P8uSnXYCWNGEFlDG0dsM-8V8X8ny4nbu2w96GRTZtzXWOHvWS123456dhd0LpD_123456789;
     expires=Mon, 12-Mar-2018 16:23:33 GMT; path=/; domain=.google.com; HttpOnly

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">

     <TITLE>302 Moved</TITLE></HEAD><BODY>  <H1>302 Moved</H1>
          The document has moved
               <A
HREF="http://www.google.ie/?gws_rd=cr&amp;dcr=0&amp;ei=BWe1WYrfIojUgAbqpIbwDg">he
re</A>.
</BODY></HTML> Connection closed by foreign host.
```

Как показано в конце каждого примера, после ответа сервера соединение закрывается. Следовательно, чтобы отправить другую команду HTTP, вам нужно заново открыть соединение. Чтобы избежать этого шага, вы можете использовать протокол HTTP/1.1 (который держит соединение открытым по умолчанию, но об этом я расскажу позже), введя `HTTP/1.1` после запрошенного ресурса:

```
GET / HTTP/1.1↵↵
```

Заметьте, что, если вы используете протокол HTTP/1.0 или HTTP/1.1, вы должны нажать клавишу ввода дважды, чтобы сообщить веб-серверу, что вы закончили отправку HTTP-запроса. В следующем разделе я расскажу, почему для соединений HTTP/1.0 и HTTP/1.1 требуется двойной перевод строки.

После того как сервер ответит, вы можете повторно выполнить команду `GET`, чтобы снова открыть страницу. На практике веб-браузеры обычно используют это открытое соединение для получения других ресурсов, а не того же самого ресурса снова, но концепция одинаковая.

Технически, чтобы соблюдать спецификацию HTTP/1.1, запросы HTTP/1.1 также требуют от вас указания заголовка хоста по причинам, которые мы (опять же) рассмотрим позже. Однако для этих простых примеров не стоит слишком беспокоиться об этом требовании, потому что Google, похоже, не настаивает на нем (хотя, если вы используете другие сайты, а не www.google.com, вы можете увидеть неожиданные результаты).

Как вы можете видеть, основной синтаксис HTTP прост. Это текстовый формат запроса-ответа, хотя он изменился в HTTP/2, где перешел в двоичный формат.

Если вы запрашиваете нетекстовые данные, такие как изображение, программы Telnet будет недостаточно. В терминальном сеансе появится бессмыслица, когда Telnet попытается, но не сможет преобразовать двоичный формат изображения в осмысленный текст, как в этом примере:

```
$ telnet www.google.com 80
Trying 172.217.3.164...
Connected to www.google.com.
Escape character is '^]'.
GET /images/branding/googlelogo/2x/googlelogo_color_120x44dp.png
▒.k▒
I▒¥&▒]▒S▒y▒8▒▒▒▒▒▒▒.к?F▒{I▒iH▒g▒?Sk▒"▒▒▒F>#U▒p▒I▒7E^T▒▒~n▒EG▒I▒^▒▒
     +.`x▒\w ▒▒CR忸▒▒▒U3V▒▒O>6b▒y8▒S▒▒cHj^.▒▒▒F▒▒4=xw
(▒F▒Bc▒]Zu▒ ▒~Hj▒i▒▒▒R▒▒C▒mH▒.|▒▒▒ <▒▒▒xH권
     ▒▒;▒'▒fH▒5▒ת▒▒▒▒|▒%WH▒7's/ÿ▒wʌ▒
▒b▒@▒4▒ـــ▒▒{:$▒.▒(O▒▒▒ÿ▒ Ў▒-▒!i▒▒▒▒\▒DƳM▒▒9
     ▒
.▒$I▒$I▒$I▒$I▒▒▒~▒T▒LC▒
▒IEND▒B`▒Connection closed by foreign host.
```

В настоящее время Telnet уже не является актуальным инструментом для работы с HTTP-запросами, так как сегодня доступны более удобные варианты. Однако на примере Telnet легко объяснить сущность форма-

та HTTP-сообщения, а также показать то, насколько простыми были начальные версии протокола.

Как уже было сказано, HTTP обрел свою популярность благодаря простоте в его реализации и обслуживании. Таким образом, практически любой компьютер при наличии сетевых возможностей (от сложных серверов до лампочек из мира интернета вещей) может работать с HTTP и быстро передавать нужные команды по сети. Более сложной задачей является реализация веб-сервера, работающего исключительно с помощью HTTP. Работа браузеров также чрезвычайно сложна. После того как вы открыли веб-страницу с помощью HTTP, они начинают взаимодействовать со множеством других протоколов (сюда же входят HTML, CSS и JavaScript, благодаря которым страницы отображаются в веб-браузере). Однако создать простую программу, которая будет получать HTTP-запрос GET и давать ответ в виде данных, не так уж сложно. Кроме того, простота использования HTTP привела к буму в области микросервисов, где приложение разбивается на множество независимых веб-сервисов, зачастую основанных на более легких серверах приложений, таких как Node.js (Node).

1.3 Синтаксическая структура HTTP и история его создания

Создателем протокола HTTP является Тим Бернерс-Ли, работавший в то время со своей командой в исследовательской организации ЦЕРН (CERN) в 1989 году. Изначально HTTP был задуман как способ реализации сети взаимосвязанных компьютеров, целью создания которой являлась возможность обеспечить доступ к исследованиям и связать их. Предполагалось, что компьютеры в этой сети смогут легко ссылаться друг на друга в режиме реального времени (достаточно будет кликнуть по ссылке – и откроется связанный документ). Идея создания такой системы зародилась достаточно давно, а термин «гипертекст» появился еще в 1960-х. 1980-е характеризуются бурным ростом и развитием Internet. Неудивительно, что именно в то время и появилась возможность реализовать эту идею. В 1989 и 1990 годах Бернерс-Ли опубликовал предложение[1] о создании такой системы. Кроме того, он создал первый веб-сервер на основе HTTP и первый веб-браузер, который мог запрашивать HTML-документы и отображать их.

1.3.1 HTTP/0.9

В 1991 году была опубликована первая спецификация[2] протокола HTTP версии 0.9. Она изложена в краткой форме и состоит менее чем из 700 слов. Согласно этой спецификации при использовании HTTP 0.9 со-

[1] https://www.w3.org/History/1989/proposal.html.

[2] https://www.w3.org/Protocols/HTTP/AsImplemented.html.

единение с сервером и дополнительным портом (если порт не указан, то по умолчанию происходит соединение с портом 80) осуществляется по протоколу TCP/IP (или с помощью аналогичной службы, ориентированной на установку предварительного соединения). Для этого следует сделать запрос в виде всего одной строки ASCII-текста, состоящей из команды `GET`, адреса документа (без пробелов) и символов возврата каретки и перевода строки (возврат каретки необязателен). Сервер должен ответить сообщением в формате HTML, которое в спецификации описано как «байтовый поток символов ASCII». После каждого запроса соединение закрывается сервером (как это было показано в предыдущих примерах). Также, согласно спецификации, «ответы на ошибки предоставляются в виде понятного человеку текста в формате HTML». Отличить ответ об ошибке от корректного ответа можно только по содержимому текста, иного способа не существует. Ответ заканчивается так: «Запросы идемпотентны. Сервер не сохранит данные о запросе при отключении от сети». HTTP 0.9 не фиксирует данные о запросах, поэтому с одной стороны он является благословлением (так как он очень прост), а с другой – проклятием (так как для создания сложных приложений необходимо использовать другие технологии, например HTTP-cookies). Ниже приведена единственная возможная команда для HTTP/0.9:

```
GET /section/page.html↵
```

Синтаксис носит фиксированный характер, за исключением, конечно, запрашиваемого ресурса (`/section/page.html`). В этой версии еще не существовало концепции полей заголовка (так называемых HTTP-заголовков), а также она не могла работать с медиафайлами, такими как, например, изображения. Сложно представить, что из такого простого протокола формата запрос–ответ, созданного для оптимизации поиска информации в одном исследовательском институте, вскоре возникла та самая Всемирная сеть, со всем ее многообразием информации, без которой уже невозможно представить современный мир. Еще на ранней стадии разработки своего изобретения Бернерс-Ли дал ему название Всемирная паутина (однако она очень отличалась от той, к которой мы привыкли), еще раз продемонстрировав свое предвидение масштабов проекта и планов превращения его в глобальную систему.

1.3.2 HTTP/1.0

Всемирная паутина возымела почти мгновенный успех. По данным NetCraft[1], к сентябрю 1995 года в ней насчитывалось уже 19 705 хостов. Спустя месяц их количество возросло до 31 568 и с тех пор продолжало увеличиваться бешенными темпами. На момент написания нашей книги количество существующих веб-сайтов приближается к 2 млрд. К 1995 году стало ясно, что функционала простейшего HTTP/0.9 уже недостаточно, а большинство веб-серверов реализовало расширения, выходящие

[1] https://news.netcraft.com/archives/category/web-server-survey/.

далеко за рамки его спецификации. Рабочая группа The HTTP Working Group (HTTP WG), возглавляемая Дейвом Раггеттом (Dave Raggett), начала работать над HTTP/1.0 в попытке задокументировать «общее использование протокола». В мае 1996 года рабочей группой IETF был опубликован документ под названием RFC 1945[1] (Request for Comments – «Рабочее предложение»). Некоторые считают его официальным стандартом, однако многие вовсе так не думают[2]. Кроме того, существует еще одна версия RFC уже для HTTP/1.0, которая не является официальной спецификацией. Данный документ представлен нам как «памятка для Internet-сообщества, не задающая какой-либо стандарт».

Независимо от спорного состояния своего статуса RFC для HTTP/1.0 добавил в протокол некоторые ключевые обновления, такие как:

- дополнительные запросы `HEAD` и `POST`, помимо уже существующего `GET`;
- указание номеров HTTP-версии. Изначально предполагалось, что по умолчанию будет использоваться HTTP/0.9 для реализации обратной совместимости;
- HTTP-заголовки, которые использовались как в запросах, так и в ответах и должны были предоставлять больше информации о запрашиваемом ресурсе и отправляемом ответе;
- трехзначный код ответа, который указывал, был ли ответ успешным. Подобные коды свидетельствовали о запросах перенаправления, условных запросах и отображали статус ошибки (например, один из самых известных кодов `404 - Not Found`).

Такие усовершенствования были очень важны, поскольку возникли из реальной необходимости. HTTP/1.0 был создан не столько для внедрения каких-либо новых опций, сколько для того, чтобы задокументировать изменения, уже произошедшие в работе веб-серверов. Такие изменения открыли для Internet множество новых возможностей. Например, пользователи получили возможность добавлять на веб-страницы медиафайлы посредством заголовков HTTP-ответов, позволяющих определить тип содержимого данных в теле страницы.

Методы HTTP/1.0

Метод `GET` остался почти таким же, как и в HTTP/0.9. Однако в HTTP/1.0 появились заголовки, что позволило создавать `GET`-запросы с условием, с помощью которых клиент может уточнить, что хочет получить ресурс, только если он изменился с момента последнего получения – если страница не была изменена, клиент получает соответствующий ответ и продолжает использовать ранее полученную копию ресурса.

С появлением метода `HEAD` клиенты смогли получать все метаданные ресурса (например, HTTP-заголовки), не загружая при этом сам ресурс.

---

[1] https://tools.ietf.org/html/rfc1945.

[2] Замечательную публикацию про RFC можно найти по адресу https://www.mnot.net/blog/2018/07/31/read_rfc.

Этот метод полезен по многим причинам. Например, поисковая система, такая как Google, может проверить, был ли ресурс обновлен, и загрузить его уже в обновленном виде, что сэкономит ресурсы для обеих сторон.

Появление метода `POST` позволило клиенту отправлять данные непосредственно на веб-сервер. Таким образом, пользователи могли работать с файлом непосредственно через HTTP (при условии, что веб-сервер настроен на получение данных), вместо того чтобы выгружать новый HTML-файл на сервер, используя стандартные методы передачи данных. Метод `POST` может использоваться как при работе с целыми файлами, так и с небольшими фрагментами произвольных данных. Обычно этот метод используется в веб-формах для ввода данных на различных веб-сайтах, где содержимое веб-формы отправляется как пара «поле–значение» в виде HTTP-запроса. Таким образом, метод `POST` позволяет клиенту отправлять информацию на сервер в виде HTTP-запроса, имеющего собственное тело, как у HTTP-ответов.

К слову, с помощью `GET` вы можете отправлять данные посредством параметров запроса прямо в URL-адресе, поместив их после знака «?». Например, запрос в виде: https://www.google.com/?q=search+string сообщит поисковой системе Google, что вас интересует `search string`. Параметры запроса были включены уже в самую раннюю спецификацию[1] унифицированного указателя ресурса (URI, Uniform Resource Locator), но они предназначались для уточнения URI путем предоставления дополнительных параметров, а в качестве способа загрузки данных на веб-сервер их не использовали. URL-адреса ограничены в длине и содержании (например, в них не могут быть использованы двоичные данные). URL-адрес не должен содержать конфиденциальные данные (пароли, данные кредитных карт и т. д.), так как в таком случае их можно будет увидеть на экране и в истории браузера. Таким образом, метод `POST` является более безопасным способом отправки данных, так как он обеспечивает конфиденциальность личных данных (тем не менее следует быть осторожным при отправке таких данных по обычному HTTP-соединению, а не по защищенному HTTPS, о чем мы поговорим позже). Еще одно отличие заключается в том, что `GET`-запрос *идемпотентен*, а `POST`-запрос таковым не является. Это означает, что при отправке нескольких `GET`-запросов на один и тот же URL-адрес, ответы на все запросы будут одинаковыми, а при отправке нескольких `POST`-запросов на один и тот же URL-адрес так случается не всегда. Например, когда вы обновляете веб-страницу, после обновления она должна остаться в неизменном виде. А когда вы обновляете страницу с подтверждением оплаты на торговой онлайн-площадке, браузер уточнит: «Вы уверены, что хотите отправить данные повторно? Это может привести к совершению дополнительной покупки» (хотя подобные веб-сайты должны гарантировать, что такого не случится!).

---

[1] https://tools.ietf.org/html/rfc1630.

Заголовки HTTP-запросов

В версии HTTP/0.9 для осуществления `GET`-запроса была отведена единственная строка, а в версии HTTP/1.0 появились заголовки. Они позволили через запрос предоставлять серверу дополнительную информацию, которая помогла бы обработать запрос эффективнее. Для HTTP-заголовков предназначены отдельные строки после начальной строки запроса. Таким образом, HTTP-запрос `GET` будет выглядеть не как

```
GET /page.html↵
```

а как

```
GET /page.html HTTP/1.0↵
Header1: Value1↵
Header2: Value2↵
↵
```

или без заголовков:

```
GET /page.html HTTP/1.0↵
↵
```

То есть к начальной строке был добавлен опциональный раздел *версии* (по умолчанию HTTP/0.9), а за опциональным разделом заголовка HTTP следовали два символа возврата каретки (новой строки, далее для краткости называемые *символами возврата*) в конце вместо одного. Второй символ возврата был необходим для отправки пустой строки, которая указывала на завершение опционального раздела заголовка запроса.

Заголовки HTTP имеют следующий вид: имя заголовка, двоеточие и далее содержимое заголовка. Согласно спецификации имя заголовка (не содержимое) не чувствительно к регистру. Заголовки можно разместить на нескольких строках, но тогда каждую новую строку нужно начинать с пробела или табуляции. Но так делать не рекомендуется, потому что не все клиенты или серверы используют этот формат и поэтому могут обработать заголовки неправильно. Вместо этого можно отправить несколько заголовков одного типа, которые семантически будут идентичны отправке версий, разделенных запятыми. Таким образом,

```
GET/page.html HTTP/1.0↵
Header1: Value1↵
Header1: Value2↵
```

обрабатывается также, как и

```
GET /page.html HTTP/1.0↵
Header1: Value1, Value2↵
```

В спецификации HTTP/1.0 прописано несколько стандартных заголовков, но на вышеприведенном примере мы можем видеть, что использование настраиваемых заголовков (в данном примере `Header1`), не зависит от того, какую версию протокола вы используете. Протокол был разработан так, чтобы его можно было обновлять и улучшать. Однако

в спецификации прямо говорится, что «эти поля не могут считаться распознаваемыми получателем» и могут быть проигнорированы, в то время как стандартные заголовки должны обрабатываться сервером, совместимым с HTTP/1.0.

Типичный GET-запрос HTTP/1.0 выглядит так:

```
GET /page.html HTTP/1.0↵
Accept: text/html,application/xhtml+xml,image/jxr/,*/*↵
Accept-Encoding: gzip, deflate, br↵
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6↵
Connection: keep-alive↵
Host: www.example.com↵
User-Agent: MyAwesomeWebBrowser 1.1↵
↵
```

В данном примере мы можем видеть, что серверу сообщается информация о форматах, в которых вы можете принять ответ (HTML, XHTML, XML и т. д.), о том, что вы можете принять различные кодировки (например, алгоритмы сжатия данных, передаваемых по HTTP, такие как gzip, deflate и Brotli), а также о том, какие языки вы предпочитаете (британский английский, далее по приоритету американский английский и остальные формы английского языка) и о том, какой браузер вы используете. Также серверу сообщается о том, что соединение необходимо сохранять открытым (об этом мы поговорим позже). В конце запроса ставится два символа возврата. С этого момента мы не будем использовать эти символы в тексте нашей книги, чтобы он был более читабельным. Но имейте в виду, что в завершении последней строки запроса они обязательно должны быть.

Коды HTTP-ответов

Типичный ответ от сервера, использующего HTTP/1.0, выглядит следующим образом:

```
HTTP/1.0 200 OK
Date: Sun, 25 Jun 2017 13:30:24 GMT
Content-Type: text/html
Server: Apache

<!doctype html>
<html>
<head>
…и т. д.
```

Остальная часть кода HTML представлена следующим образом. Как видите, первая строка ответа включает информацию о HTTP-версии ответного сообщения (`HTTP/1.0`), трехзначный код состояния HTTP (`200`) и текстовое описание кода состояния (`OK`). Коды состояния и описания появились в HTTP/1.0. В HTTP/0.9 такого понятия, как код ответа, не существовало, а ошибки могли встретиться только в возвращаемом HTML-документе. В табл. 1.1 приведены коды HTTP-ответов согласно спецификации HTTP/1.0.

Таблица 1.1 Коды HTTP-ответов согласно спецификации HTTP/1.0

Категория	Код	Наименование	Описание
1xx (информационные)	нет	нет	HTTP/1.0 не определяет никаких кодов состояния 1xx, но определяет категорию
2xx (успешно)	200	OK	Стандартный код ответа для успешного запроса.
	201	Created	Код должен быть возвращен по POST-запросу.
	202	Accepted	Обработка запроса в процессе.
	204	No content	Запрос успешно принят и обработан, но в ответе нет тела сообщения
3xx (перенаправление)	300	Multiple choices	Данный код используется редко. В нем говорится, что категория 3xx подразумевает, что ресурс доступен в одном (или нескольких) местах, а точный ответ предоставляет более подробную информацию о том, где он находится.
	301	Moved permanently	Заголовок `Location` HTTP-ответа должен предоставить новый URL ресурса.
	302	Moved temporarily	Заголовок `Location` HTTP-ответа должен предоставить новый URL ресурса.
	304	Not modified	Используется для условных ответов, в которых тело не нужно отправлять снова
4xx (ошибка клиента)	400	Bad request	Запрос не может быть обработан, исправьте ошибку и запросите заново.
	401	Unauthorized	Этот код показывает, что вы не авторизированы.
	403	Forbidden	Вы авторизированы, но ваши идентификационные данные не имеют прав доступа.
	404	Not found	Возможно, самый узнаваемый код HTTP статуса, так как часто встречается на странице ошибки
5xx (ошибка сервера)	500	Internal server error	Запрос не удалось выполнить из-за ошибки сервера.
	501	Not implemented	Сервер не распознает запрос (чаще всего из-за метода HTTP, который неизвестен серверу).
	502	Bad gateway	Сервер, выступая в роли шлюза или прокси-сервера, получил сообщение об ошибке от вышестоящего сервера.
	503	Service unavailable	По техническим причинам, например перегрузка, сервер временно не может обрабатывать запросы

Внимательные читатели могут заметить, что некоторые коды (`203`, `303`, `402`) из более ранних версий HTTP / 1.0 RFC здесь отсутствуют. Некоторые дополнительные коды были исключены из окончательного опубликованного RFC. Однако несколько из них вернулось в спецификации в HTTP/1.1, но уже с другими описаниями и значениями. Администрация адресного пространства Internet (Internet Assigned Numbers Authority, IANA) поддерживает полный список кодов состояния HTTP во всех версиях HTTP. Коды состояния, представленные в табл. 1.1, впервые были определены в HTTP/1.0[1] и сейчас используются наиболее часто.

В некоторых случаях ответы могут совпадать. Например, какой код ответа вы получите в случае, если запрос не будет распознан сервером, `400` (неверный запрос) или `501` (запрос не реализован)? Существует большое разнообразие категорий кодов ответа, так что каждое приложение мо-

[1] https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml.

жет использовать наиболее подходящую из них. В спецификации сказано, что список кодов ответов можно пополнить, поэтому по мере необходимости можно добавлять новые коды и для этого не обязательно менять сам протокол. К слову, это еще одна причина, по которой коды ответов делятся на категории. Новый код ответа (например, `504`) может быть не распознан существующим HTTP/1.0 клиентом, однако ему будет понятно, что по какой-то причине на стороне сервера произошла ошибка, и сможет обработать его так же, как он обрабатывает другие коды ответа категории `5xx`.

Заголовки HTTP-ответов

После первой строки возврата идет определенное количество (ноль или более) строк ответа заголовка HTTP/1.0. Заголовки запросов и ответов формируются согласно одному и тому же формату. За ними следуют два символа возврата, а затем содержимое тела (жирным шрифтом):

```
GET /
HTTP/1.0 302 Found
Location: http://www.google.ie/?gws_rd=cr&dcr=0&ei=BWe1WYrf123456qpIbwDg Cache-
Control: private
Content-Type: text/html; charset=UTF-8
Date: Sun, 10 Sep 2017 16:23:33 GMT
Server: gws
Content-Length: 268
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
     <H1>302 Moved</H1>
          The document has moved
<A HREF="http://www.google.ie/?gws_rd=cr&amp;dcr=0&amp;ei=BWe1WYrfIojUgAbqpIbwDg">h
ere</A>.</BODY></HTML> Connection closed by foreign host.
```

Если предыдущая опубликованная версия HTTP/0.9 позволяла лишь извлекать статические документы из репозитория, то с появлением новой версии HTTP/1.0 синтаксис HTTP значительно расширился, и теперь стало возможным создание динамических, многофункциональных приложений. Кроме того, HTTP стал сложнее, а объем спецификации протокола расширился от 700 (как было в HTTP/0.9) до 20 000 слов (в HTTP/1.0 RFC). Однако даже после публикации новой спецификации рабочая группа HTTP Working Group рассматривала ее как временную версию, основной функцией которой было задокументировать текущее использование протокола, и уже работала над версией HTTP/1.1. Как уже было сказано, HTTP/1.0 был опубликован в целях создания стандартов и документации для HTTP, так как до этого протокол не был оформлен должным образом, а его синтаксис, которым могли бы пользоваться клиенты и серверы, не был определен. К стандарту RFC прилагались новые коды ответов, другие методы, такие как `PUT`, `DELETE`, `LINK`, и `UNLINK`, и дополнительные HTTP-заголовки, некоторые из которых позже войдут в стан-

дарт HTTP/1.1. HTTP Working Group с трудом справлялась с реализацией протокола и обнародовала ее только спустя пять лет.

1.3.3 HTTP/1.1

Как мы уже знаем, версия HTTP/0.9 предназначалась в основном для получения текстовых документов. Затем версия была расширена до HTTP/1.0, и ее функциональность вышла далеко за пределы текстового формата. Позже эта версия была дополнительно стандартизирована и уточнена в HTTP/1.1. Согласно системе управления версиями, HTTP/1.1 была по большей части модификацией HTTP/1.0 и не несла в себе радикальных изменений структуры протокола. Переход от 0.9 к 1.0 включал в себя гораздо больше изменений, так как были созданы HTTP-заголовки. HTTP/1.1 внес некоторые дополнительные улучшения и позволил оптимизировать использование протокола HTTP (например, постоянные соединения, обязательные заголовки сервера, улучшенные параметры кеширования и фрагментированное кодирование). Но – что, наверное, важнее всего – на ее основе был создан официальный стандарт, на котором строилось будущее Всемирной паутины. Несмотря на то что HTTP достаточно прост для понимания, существует множество тонкостей, которые могут быть реализованы несколько по-разному, а отсутствие формального стандарта затрудняет работу с ними.

Первая спецификация HTTP/1.1 была опубликована в январе 1997[1] года (всего через девять месяцев после публикации спецификации HTTP/1.0). Спецификации обновлялись два раза: в июне 1999-го[2] и в июне 2014-го[3]. После каждого обновления предыдущие версии считались устаревшими. Спецификация HTTP /1.1 составляла 305 страниц и содержала почти 100 000 слов. Исходя из этого можно увидеть, насколько расширился этот простой протокол и насколько важно было прояснить тонкости использования HTTP. Фактически на момент написания этой книги спецификация снова обновляется[4], и ожидается, что это обновление будет опубликовано в начале 2019 года (к моменту подготовки перевода обещанное большое обновление так и не появилось на сайте w3.org. – *Прим. ред.*). По своей сути, HTTP/1.1 не слишком отличается от HTTP/1.0. В первые два десятилетия своего существования Internet стремительно рос и расширялся, ввиду чего и стало необходимым создание документации, которая включала бы в себя информацию о принципах работы с новыми возможностями.

Описанию всех аспектов HTTP/1.1 может быть посвящена целая книга, а здесь мы попытаемся рассмотреть основные моменты – сформировать фон и контекст для дальнейшего рассмотрения HTTP/2. При переходе от HTTP/1.0 к HTTP/1.1 фундаментальная структура протокола не измени-

---

1 https://tools.ietf.org/html/rfc2068.
2 https://tools.ietf.org/html/rfc2616.
3 https://tools.ietf.org/html/rfc7230 и https://tools.ietf.org/html/rfc7235.
4 https://github.com/httpwg/http-core.

лась, а многие дополнительные функции HTTP/1.1 были созданы с помощью добавления HTTP-заголовков в HTTP/1.0. Однако в синтаксисе протокола произошли и некоторые серьезные изменения, например наличие заголовка хоста стало обязательным, а также были введены постоянные соединения.

Обязательный заголовок HOST

URL-адрес в строках HTTP-запроса (например, команда `GET`), является не абсолютным URL-адресом (например, `http://www.example.com/section/page.html`), а относительным URL (например, `/section/page.html`). При создании HTTP предполагалось, что веб-сервер будет содержать только один веб-сайт, хотя, возможно, на этом сайте будет много разделов и страниц. Прежде чем совершать HTTP-запросы, пользователь должен подключиться к веб-серверу, таким образом задается хост-часть URL-адреса. В настоящее время на многих веб-серверах может быть расположено сразу несколько сайтов (*виртуальный хостинг*), поэтому важно сообщить серверу, *какой именно сайт* вы хотите открыть, а также какой *относительный URL-адрес* вам нужен. Эту функцию можно было бы реализовать, изменив URL-адрес в HTTP-запросах на полный, абсолютный URL-адрес, но считалось, что многие существующие веб-серверы и клиенты не смогут распознать их. Вместо этого проблема была решена путем добавления заголовка `Host`:

```
GET / HTTP/1.1
Host: www.google.com
```

В отличие от HTTP/1.0 в HTTP/1.1 его наличие стало обязательным. Следующий запрос сформирован технически некорректно, так как он указывает на версию (HTTP/1.1), но не содержит заголовок `Host`:

```
GET / HTTP/1.1
```

Согласно спецификации[1] HTTP/1.1 этот запрос должен быть отклонен сервером (с кодом ответа 400), хотя сейчас большинство веб-серверов имеет хост для таких запросов по умолчанию. Введение обязательного заголовка `Host` стало важным шагом в HTTP/1.1. Он позволил серверам более широко использовать виртуальный хостинг и тем самым позволил Internet развиваться, не прибегая к созданию отдельных веб-серверов для каждого сайта. Кроме того, относительно низкие ограничения IP-адресов версии IPv4 были бы достигнуты гораздо раньше, если бы это изменение не вошло в силу. С другой стороны, если бы эти ограничения не было реализованы, возможно, они помогло бы форсировать переход на IPv6, который на момент написания этой книги существует уже более 20 лет и все еще находится в процессе развертывания.

Тот факт, что вместо изменения относительного URL-адреса на абсолютный в спецификации прописывалось наличие обязательного поля

[1] https://tools.ietf.org/html/rfc7230#section-5.4.

заголовка `Host`, привел к появлению разногласий[1]. HTTP-прокси, введенные вместе с HTTP/1.1, позволяли подключаться к HTTP-серверу через посреднический HTTP-сервер. Синтаксис прокси-серверов требовал полных абсолютных URL-адресов для всех запросов, но на действующих веб-серверах (так называемых *серверах-источниках*) использовались заголовки `Host`. Как мы уже знаем, такой принцип работы помогал избежать прекращения работы существующих серверов, однако после того, как он стал обязательным, стало очевидно что после перехода на HTTP/1.1 и в целях лучшей совместимости с его реализациями все клиенты и серверы без исключения должны использовать запросы в стиле виртуального хостинга. В спецификации HTTP/1.1 сказано: «Для дальнейшего перехода к абсолютной форме URL-адресов для всех запросов в последующих версиях HTTP веб-серверы должны принимать абсолютный URL в запросах, даже если клиенты HTTP/1.1 будут отправлять их только прокси-серверам». Тем не менее, как мы увидим позже, в версии HTTP/2 эта проблема не была полностью решена. В ней вместо заголовка `Host` ввели поле для псевдозаголовка `authority` (см. главу 4).

Постоянные соединения (Keep-Alive)

В HTTP/1.0 были введены постоянные соединения. Это изменение было довольно значимым и, кроме того, поддерживалось многими серверами HTTP/1.0, несмотря на то что в спецификации этой версии постоянных соединений еще не было. Изначально HTTP представлял собой примитивный протокол типа «запрос–ответ». Клиент устанавливает соединение, запрашивает ресурс, получает ответ, и соединение закрывается. Сеть росла, и в ней появлялось все больше информации различных типов, в связи с чем закрытие соединений значительно снижало эффективность работы. Отображение одной страницы требовало использования нескольких HTTP-ресурсов, поэтому закрытие соединения (которое потом снова нужно будет открыть) вызывало ненужные задержки. Проблема была решена путем введения нового HTTP-заголовка `Connection`, совместимого с версией HTTP/1.0. Значение `Keep-Alive` в этом заголовке позволяет запросить сервер сохранить соединение открытым, чтобы разрешить отправку дополнительных запросов:

```
GET /page.html HTTP/1.0
Connection: Keep-Alive
```

Если сервер поддерживает постоянные соединения, он будет отвечать как обычно, но уже с использованием заголовка `Connection: Keep-Alive` в ответах:

```
HTTP/1.0 200 OK
Date: Sun, 25 Jun 2017 13:30:24 GMT
Connection: Keep-Alive
Content-Type: text/html
```

[1] С дискуссией по этому поводу можно ознакомиться по адресу https://lists.w3.org/Archives/Public/ietf-http-wg-old/1999SepDec/0014.html.

```
Content-Length: 12345
Server: Apache

<!doctype html>
<html>
<head>
…и т. д.
```

Получив такой ответ, клиент увидит, что он может отправить другой запрос на то же соединение после получения текущего ответа, поэтому серверу не нужно закрывать соединение с клиентом только для того, чтобы потом снова открыть его. При использовании постоянных соединений понять, когда ответ завершен, бывает довольно сложно; закрытие соединения является отчетливым признаком того, что сервер мог отправить ответ несуществующему соединению! HTTP-заголовок `Content-Length` предназначен для определения длины тела ответа. Когда ответ получен полностью, клиент может приступать к отправке следующего запроса.

HTTP-соединение может быть закрыто в любой момент как клиентом, так и сервером. Закрытие может произойти случайно (из-за сбоев сетевого подключения) или намеренно (например, если соединение некоторое время не используется, сервер закроет его, чтобы освободить ресурсы для других соединений). Поэтому даже при использовании постоянных соединений и клиенты, и серверы должны отслеживать их состояние и иметь возможность возобновить их в случае неожиданного закрытия. С некоторыми запросами ситуация усложняется. Например, если вы регистрируетесь на веб-сайте торговой площадки, не следует отправлять повторный запрос, если вы не удостоверились, что сервер обработал предыдущий.

В версии HTTP/1.1 процесс постоянного подключения был добавлен в официальный стандарт и с тех пор выполняется по умолчанию. Любое HTTP/1.1 соединение носит постоянный характер, даже если ответы не содержат заголовка `Connection: Keep-Alive`. Если сервер намерен закрыть соединение, он должен включить в ответ заголовок `Connection: close`:

```
HTTP/1.1 200 OK
Date: Sun, 25 Jun 2017 13:30:24 GMT
Connection: close
Content-Type: text/html; charset=UTF-8
Server: Apache

<!doctype html>
<html>
<head>
…и т. д.
Connection closed by foreign host.
```

Ранее в этой главе мы рассматривали эту тему на примерах работы с Telnet. Теперь попробуйте снова отправить через Telnet следующее:

- запрос HTTP/1.0 без заголовка `Connection: Keep-Alive`. Соединение будет автоматически закрыто сервером после отправки ответа;

- тот же запрос HTTP/1.0, но с заголовком `Connection: Keep-Alive`. Соединение остается открытым;
- запрос HTTP.1.1 с заголовком `Connection: Keep-Alive` или без него. Соединение остается открытым по умолчанию.

Нет ничего необычного в том, что клиенты HTTP/1.1 включают заголовок `Connection: Keep-Alive` в запросы HTTP/1.1 в явном виде, несмотря на то что он используется по умолчанию. Точно так же серверы иногда включают заголовок в HTTP/1.1 ответы, несмотря на то что это не обязательно.

Таким образом, если веб-браузер обрабатывает HTML-документ и видит, что ему нужен файл CSS и файл JavaScript, он должен иметь возможность отправлять запросы на эти файлы одновременно и получать ответы обратно в нужном порядке, а не ждать первого ответа перед отправкой второго запроса. Вот пример:

```
GET /style.css HTTP/1.1
Host: www.example.com

GET /script.js HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Date: Sun, 25 Jun 2017 13:30:24 GMT
Content-Type: text/css; charset=UTF-8
Content-Length: 1234
Server: Apache

.style {
…и т. д.

HTTP/1.1 200 OK
Date: Sun, 25 Jun 2017 13:30:25 GMT
Content-Type: application/x-javascript; charset=UTF-8
Content-Length: 5678
Server: Apache

Function(
…и т. д.
```

К сожалению, существует ряд причин, по которым технология конвейеризации запросов не развивалась, а клиенты и серверы довольно плохо поддерживают ее. Об этом мы поговорим во второй главе. Короче говоря, возможность повторно использовать постоянное TCP-соединение для нескольких запросов сильно увеличивает производительность, однако в большинстве реализаций HTTP/1.1 она по-прежнему не используется. Во время обработки одного запроса для других запросов HTTP-соединение недоступно.

Другие новые функции

В спецификацию HTTP/1.1 было введено множество новых функций:

- к методам `GET`, `POST` и `HEAD` (введенных еще в HTTP/1.0) добавились новые. Среди них `PUT`, `OPTIONS` и менее используемые `CONNECT`, `TRACE`, и `DELETE`;

- методы кеширования. Они позволяли серверу поручить клиенту сохранить ресурс (например, CSS-файл) в кеше браузера, чтобы он мог быть повторно использован позже, если потребуется. HTTP-заголовок `Cache-Control`, представленный в HTTP/1.1, имел больше параметров, чем заголовок `Expires` из HTTP/1.0;
- файлы HTTP-куки, благодаря которым HTTP перестал быть протоколом без сохранения состояния;
- объявление рабочей кодировки (как показано в некоторых примерах в этой главе) и языка HTTP-ответов;
- поддержка прокси;
- аутентификация;
- новые коды состояния;
- завершающие заголовки (которые мы обсудим в главе 4, раздел 4.3.3).

В целях расширения возможностей к HTTP постоянно добавляются новые заголовки, многие из которых обеспечивают лучшую производительность или безопасность. Спецификация HTTP/1.1 не претендует на звание окончательной и поощряет появление новых заголовков. В ней даже есть раздел[1], посвященный тому, как заголовки должны быть определены и задокументированы. Как уже упоминалось, некоторые из этих заголовков добавляются по соображениям безопасности и используются для того, чтобы веб-сайт мог сообщить браузеру о включении определенных дополнительных средств защиты, поэтому они не требуют реализации на стороне сервера (кроме возможности отправки заголовка). Одно время существовало соглашение, согласно которому к нестандартизированным заголовкам должен был добавляться префикс `X-` (`X-Content-Type`, `X-Frame-Options`, `X-XSS-Protection`). Сейчас это соглашение устарело[2], и новые экспериментальные заголовки трудно отличить от заголовков, прописанных в спецификации HTTP/1.1. Нередко такие заголовки закрепляются в локальных RFC (`Content-Security-Policy`[3], `Strict-Transport-Security`[4], и т. д.).

1.4 Введение в HTTPS

Изначально, HTTP был простым текстовым протоколом. HTTP-сообщения передаются через Internet в незашифрованном виде и поэтому они могут быть перехвачены по пути к месту назначения. Как следует из самого названия, Internet представляет собой сеть, состоящую из множества компьютеров, каждый из которых может взаимодействовать друг с другом, а не систему с прямыми связями между двумя машинами. Фиксированной маршрутизации сообщений в Internet не существует.

---

[1] https://tools.ietf.org/html/rfc7231#section-8.3.1.
[2] https://tools.ietf.org/html/rfc6648.
[3] https://tools.ietf.org/html/rfc7762.
[4] https://tools.ietf.org/html/rfc6797.

После того как вы отправляете сообщение, оно совершает длинный путь от вашего Internet-провайдера (ISP) до телекоммуникационных сетей и принимающей стороны, поэтому вы не узнаете, сколько других компьютеров могут увидеть его. Поскольку HTTP-сообщение – это обычный текст, злоумышленникам не составит труда перехватить его, прочитать или даже изменить содержание.

HTTPS является защищенной версией HTTP, которая шифрует передаваемые сообщения с помощью протокола Transport Layer Security (TLS). Предыдущая версия носила название Secure Sockets Layer (SSL), и о ней вы можете прочесть в сноске ниже. HTTPS добавляет к HTTP три важных аспекта:

- *шифрование* (третьи лица не смогут прочитать сообщения во время их передачи);
- *целостность* (сообщение невозможно изменить в процессе передачи, так как зашифрованное сообщение имеет цифровую подпись, и эта подпись криптографически проверяется перед расшифровкой);
- *аутентификация* (обеспечивает передачу сообщения именно на нужный вам сервер).

SSL, TLS, HTTPS и HTTP

HTTPS зашифровывает сообщения по протоколам SSL и TLS. SSL был разработан компанией Netscape. Версия протокола SSLv1 так и не увидела свет, и вместо нее в 1995 году Netscape выпустила версию SSLv2. В 1996 году компания выпустила SSLv3, в которой были исправлены некоторые недостатки предыдущих версий.

SSL не считался официальным Internet-стандартом, поскольку принадлежал компании Netscape, но впоследствии IETF опубликовала его постфактум как официальный документ[a]. Далее на основании SSL был оформлен и стандартизирован новый протокол, получивший название TLS. Протоколы TLSv1.0[b] и SSLv3 были весьма похожи, однако все же несовместимы. Версии TLSv1.1[c] и TLSv1.2[d] были созданы в 2006 и 2008 годах соответственно. Они стали более безопасными. TLSv1.3 был признан стандартом в 2018 году[e]. Он более безопасен и эффективен[f], но потребуется еще много времени, чтобы он получил широкое распространение.

Несмотря на то что были созданы новые и более безопасные стандартизированные версии, многие люди все еще пользовались SSLv3, поэтому он долгое время оставался стандартом де-факто, хотя многие клиенты поддерживали и TLSv1.0. Однако в 2014 году в SSLv3 были обнаружены серьезные уязвимости[g], в связи с чем он перестал поддерживаться браузерами и вышел из употребления[h]. Эта ситуация положила начало развитию TLS. Спустя некоторое время в TLSv1.0 обнаружили аналогичные уязвимости. Совет по стандартам безопасности призывал использоватьTLSv1.1 или более поздние версии[i].

В итоге люди стали путать названия протоколов. Многие до сих пор говорят «SSL», потому что он оставался стандартом очень долгое время, а другие го-

ворят «SSL/TLS» или «TLS». Чтобы избежать недоразумений, иногда говорят просто «HTTPS», хотя этот термин и не совсем корректен. В данной книге мы называем шифрование «HTTPS» (а не SSL или SSL/TLS). Однако если речь идет конкретно о TLS, мы используем соответствующий термин. Следуя тому же принципу, когда мы говорим об основной семантике HTTP, мы говорим просто «HTTP» независимо от того, идет речь о незашифрованном или зашифрованном HTTPS-соединении.

---

[a] https://tools.ietf.org/html/rfc6101.
[b] https://tools.ietf.org/html/rfc2246.
[c] https://tools.ietf.org/html/rfc4346.
[d] https://tools.ietf.org/html/rfc5246.
[e] https://tools.ietf.org/html/rfc8446.
[f] https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/.
[g] https://www.us-cert.gov/ncas/alerts/TA14-290A.
[h] https://tools.ietf.org/html/rfc7568.
[i] https://www.pcisecuritystandards.org/documents/Migrating-from-SSL-Early-TLS-Info-Suppv1_1.pdf.

HTTPS работает с использованием шифрования с открытым ключом, которое позволяет серверам предоставлять пользователям открытые ключи в виде цифровых сертификатов при первом подключении. Ваш браузер шифрует сообщения с помощью этого открытого ключа, расшифровать который может только сервер-получатель, так как только он имеет соответствующий секретный ключ. Такая система обеспечивает вам безопасное взаимодействие с веб-сайтом без необходимости заранее знать общий секретный ключ. Такой подход особенно важен для таких систем, как Internet, где новые веб-сайты и пользователи появляются и исчезают ежедневно каждую секунду.

Цифровые сертификаты выдаются различными центрами сертификации (Certification authority), с которыми работают веб-браузеры, а также подписываются ими цифровой подписью, поэтому проверить подлинность открытого ключа для сервера, к которому вы подключаетесь, очень легко. Проблема заключается в том, что HTTPS показывает, что вы подключаетесь к серверу, но не гарантирует его безопасности. С помощью HTTPS для поддельных фишинговых сайтов можно легко установить другой, но похожий на оригинальный домен (exmplebank.com вместо examplebank.com). HTTPS-сайты обычно отмечены в веб-браузерах значком зеленого навесного замочка. Многие пользователи считают, что это означает, что *сайт безопасен*, однако на самом деле это просто говорит о том, что веб-сайт использует *безопасное шифрование*.

Некоторые центры сертификации проводят дополнительную проверку веб-сайтов при выдаче сертификатов, а также предоставляют расширенный сертификат проверки (известный как сертификат EV – Extended Validation), который обеспечивает шифрование HTTP-трафика так же, как и обычный сертификат, но в большинстве веб-браузеров он отображает название компании, как показано на рис. 1.4.

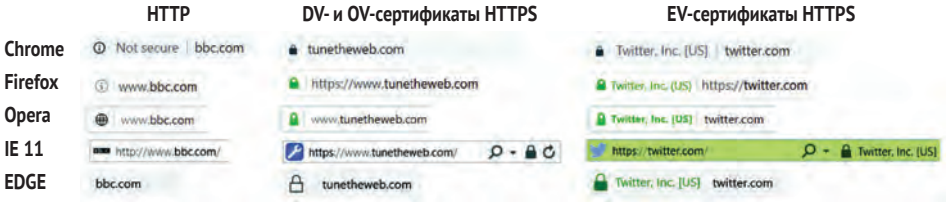

Рис. 1.4 HTTPS-индикаторы веб-браузера

Многие люди оспаривают преимущества сертификатов EV[1], главным образом потому что подавляющее большинство пользователей не замечает названия компании на сайтах и не видит разницы между сертификатами EV и стандартными сертификатами (Domain Validated, DV). Сертификаты проверенных организаций (Organization validation, OV) являются сертификатами среднего уровня безопасности, так как они подразумевают некоторые проверки при оформлении, но не дают дополнительных уведомлений в веб-браузерах, что делает их бессмысленными на техническом уровне (хотя центры сертификации могут предлагать дополнительные поддержку в рамках покупки таких сертификатов).

На момент написания данной книги команда Google Chrome все еще исследует такие индикаторы безопасности и экспериментирует с ними[2]. Они считают схему (`http` и `https`), префикс `www` и, возможно, даже сам значок замочка ненужными элементами и стараются скрыть их (кроме того, они считают, что использование HTTPS является нормой, а сайты, поддерживающие только HTTP, должны иметь пометку «небезопасный»). Команда также рассматривает вопрос о том, не стоит ли отказаться от сертификата EV[3].

Протокол HTTPS разработан на основе HTTP и очень схож с ним. По умолчанию у него другой порт (порт 443, в отличие от стандартного порта 80 для HTTP), а также он имеет другую схему URL-адресов (`https://`, а не `http://`). Однако эти аспекты не столь значительны, так как HTTPS и HTTP имеют одинаковый синтаксис и формат сообщений, а отличаются они лишь в планах шифрования и дешифрования.

Когда клиент подключается к HTTPS-серверу, он проходит через стадию согласования (или TLS-рукопожатия). Во время этого процесса сервер предоставляет открытый ключ, клиент и сервер согласовывают используемые методы шифрования, а затем клиент и сервер согласовывают общий ключ шифрования, чтобы использовать его в будущем. (Криптография с открытым ключом работает медленно, поэтому открытые ключи шифрования применяются только для согласования подключения сервера и клиента, который используется для шифрования

---

[1] https://www.tunetheweb.com/blog/what-does-the-green-padlock-really-mean/.

[2] https://blog.chromium.org/2018/05/evolving-chromes-security-indicators.html.

[3] https://groups.google.com/forum/#!topic/mozilla.dev.security.policy/szD2KBHfwl8%5B1-25%5D.

будущих сообщений, что обеспечивает лучшую производительность.) Подробнее мы поговорим о TLS-рукопожатии в главе 4 (раздел 4.2.1).

После создания сессии HTTPS происходит обмен стандартными HTTP-сообщениями. Клиент и сервер зашифровывают их перед отправкой и расшифровывают при получении, однако для обычного веб-разработчика или менеджера сервера между HTTPS и HTTP нет уже фактически никакой разницы. Все процессы происходят прозрачно, если вы, конечно, не смотрите на необработанные сообщения, отправленные по сети. При переходе на HTTPS мы продолжаем пользоваться стандартными HTTP-запросами и ответами, а не заменяем их каким-то другим протоколом.

HTTPS – это обширная тема, которая выходит далеко за рамки этой книги. Однако мы снова затронем этот вопрос в следующих главах, поскольку с внедрением HTTP/2 ситуация несколько изменилась. Но сейчас нам необходимо знать только то, что HTTPS существует и что он работает на несколько ином уровне, нежели HTTP (между TCP и HTTP). Если вы не смотрите непосредственно на зашифрованные сообщения, вы не почувствуете никакой значительной разницы между HTTP и HTTPS.

Веб-серверам, использующим HTTPS, необходим клиент с поддержкой HTTPS, выполняющий шифрование и дешифрование. Telnet уже не подходит для этой задачи, поэтому для отправки примеров HTTP-запросов на эти серверы следует выбрать другой клиент. Вы можете использовать команду `s_client` в OpenSSL, которая позволит вам отправлять HTTP-команды на HTTPS-сервер аналогично тому, как вы делали это с помощью Telnet:

```
openssl s_client -crlf -connect www.google.com:443 -quiet
GET / HTTP/1.1
Host: www.google.com
HTTP/1.1 200 OK
…и т. д.
```

Между тем мы заканчиваем рассмотрение инструментов командной строки и переходим к изучению более эффективных способов работы с HTTP-запросами. В следующем разделе мы кратко рассмотрим инструменты веб-браузера, с помощью которых работать с HTTP-запросами и ответами становится намного проще.

1.5 Инструменты для просмотра, отправки и получения HTTP-сообщений

Безусловно, такие инструменты командной строки, как Telnet, отлично подходят для изучения основ HTTP, однако они имеют некоторые ограничения и не подходят для работы с веб-страницами больших размеров, каких, к слову, в Internet довольно много. Существуют инструменты, справляющиеся с этими задачами лучше, чем Telnet. Многими из них вы можете пользоваться непосредственно в вашем веб-браузере.

1.5.1 Использование инструментов разработчика в веб-браузерах

Сегодня все веб-браузеры оснащены так называемыми *инструментами разработчика*, с помощью которых вы можете увидеть некоторые детали устройства и работы веб-сайтов, включая HTTP-запросы и ответы.

Инструменты разработчика запускаются сочетанием клавиш (**F12** в большинстве браузеров для ОС Windows или **Option**+**Command**+**I** на компьютерах Apple). Так же вы можете кликнуть на элемент страницы правой кнопкой мыши и во всплывающем контекстном меню выбрать **Посмотреть код**. Панель инструментов разработчика содержит несколько вкладок, в которых вы можете увидеть различные технические детали строения веб-страниц, однако сейчас нас больше всего интересует вкладка **Network** (Сеть). Если вы откроете инструменты разработчика, а затем загрузите веб-страницу, во вкладке **Network** вы увидите все HTTP-запросы, а при нажатии на них получите более подробную информацию, в том числе заголовки запроса и ответа.

На рис. 1.5 вы можете увидеть панель инструментов разработчика в веб-браузере Chrome при загрузке https://www.google.com.

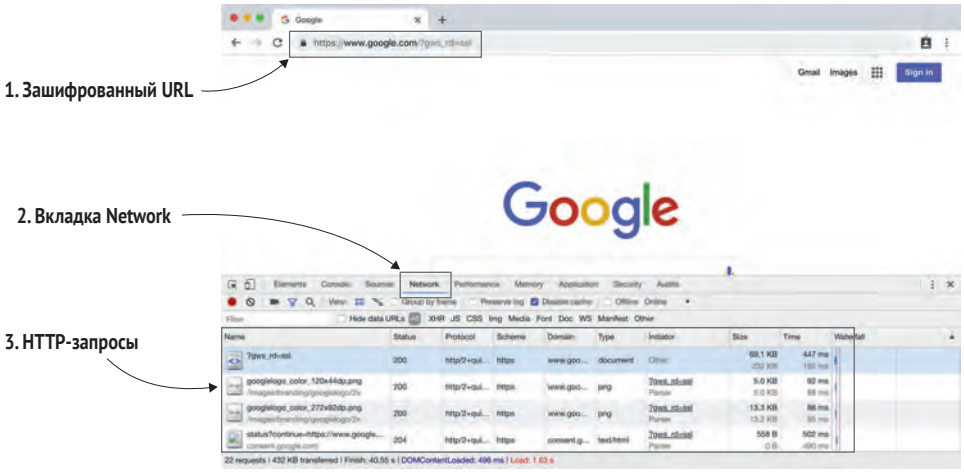

Рис. 1.5 Панель инструментов разработчика в Chrome

URL-адрес вводится в верхней части адресной строки (1). Обратите внимание на значок замочка и `https://`. Это означает, что Google использует HTTPS (хотя, как уже упоминалось, Chrome может перестать использовать его). Веб-страница размещается под адресной строкой. Однако если вы откроете панель инструментов разработчика, то увидите на странице новый раздел, содержащий различные вкладки. При нажатии на вкладку **Сеть** (2) отображаются HTTP-запросы (3), включая такую информацию, как HTTP метод (`GET`), статус ответа (`200`), версия протокола (`http/1.1`) и схема (`https`). Вы можете изменить отображаемые столбцы,

кликнув правой кнопкой мыши на их заголовки. Например, столбцы **Protocol** (Протокол), **Scheme** (Схема) и **Domain** (Домен) не отображаются по умолчанию, и на некоторых веб-сайтах (например, Twitter) в столбце для HTTP/2 вы можете увидеть `h2`, а для более новой версии протокола (которую мы рассмотрим в главе 9), возможно, даже `http/2+quic` (Google).

На рис. 1.6 показано, что происходит при нажатии на первый запрос (1). Справа вы можете увидеть панель вкладок с заголовками ответов (2) и заголовками запросов (3). Со многими из них (но не со всеми) мы уже познакомились в этой главе.

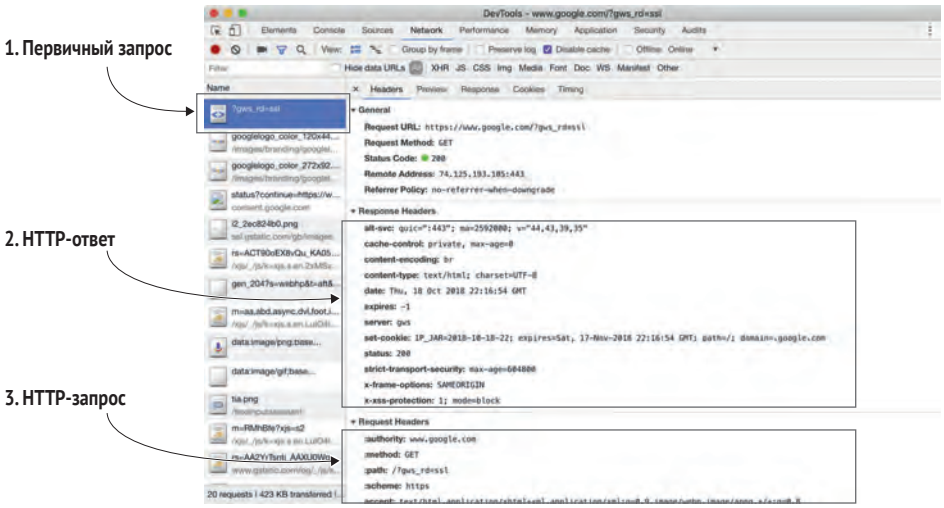

Рис. 1.6 Просмотр HTTP-заголовков в Chrome

Браузер обрабатывает HTTPS-запросы, поэтому инструменты разработчика показывают сообщения запросов до их шифрования и ответные сообщения после их дешифрования. Как правило, если у вас есть правильные инструменты для обработки шифрования и дешифрования сообщений, после настройки HTTPS-соединения дальнейшая работа протокола уже мало чем интересна. Кроме того, инструменты разработчика в большинстве веб-браузеров хорошо справляются с задачей отображения медиафайлов, а код (HTML, CSS, и JavaScript) можно отформатировать, чтобы его было легче читать. Периодически мы будем возвращаться к теме инструментов разработчика. Если вы еще не знакомы с инструментами разработчика вашего браузера для часто посещаемых вами сайтов, то настоятельно рекомендуем изучить их.

1.5.2 Отправка HTTP-запросов

Использование инструментов разработчика веб-браузеров является наилучшим способом просмотра еще не обработанных HTTP-запросов и ответов, однако – что весьма удивительно – отправлять такие запросы эти

инструменты не могут. Инструменты разработчика в браузере редко позволяют отправлять необработанные HTTP-сообщения, но, например, с помощью адресной строки можно отправлять простые `GET`-запросы, а некоторые веб-сайты предоставляют определенные функциональные возможности, например отправку `POST`-запросов с помощью HTML-форм.

Расширение для браузеров Advanced REST Client дает вам возможность отправлять необработанные HTTP-сообщения и просматривать ответы. Отправьте запрос `GET` (1) для URL-адреса https://www.google.com (2) и нажмите **Отправить** (3), затем вы получите ответ (4), как показано на рис. 1.7. Обратите внимание, что это приложение также обрабатывает запросы и ответы с помощью HTTPS.

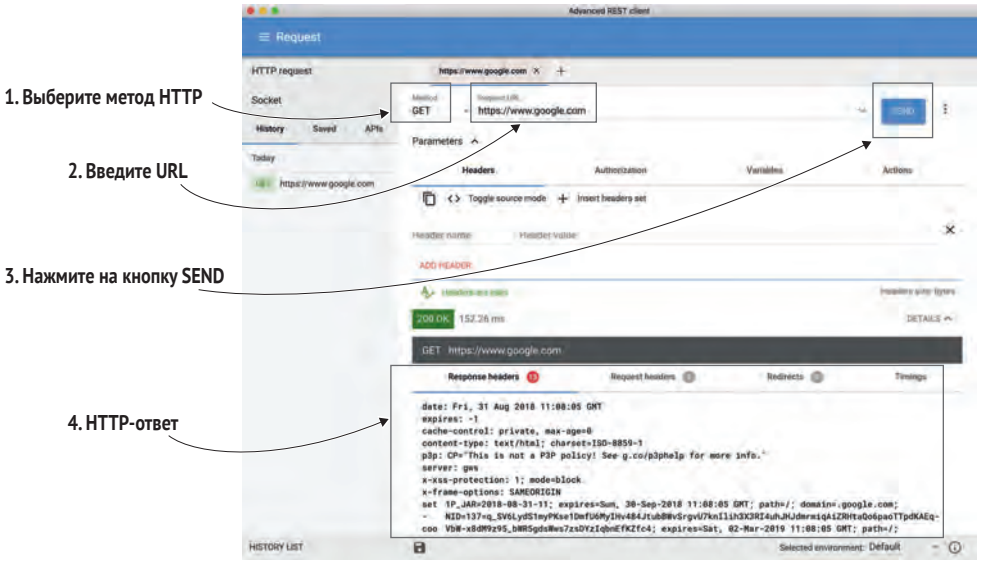

Рис. 1.7 Расширения для браузеров Advanced REST Client

Работа с расширением Advanced Rest Client очень проста и привычна, однако при этом оно позволяет отправлять многие типы HTTP-запросов (например, `POST` и `PUT`), а также создавать заголовок или тело данных, предназначенных для отправки. Изначально Advanced REST Client был расширением для Chrome, но позже его оформили как отдельное приложение. Существуют и другие подобные расширения, имеющие соответствующую функциональность, например Postman (Chrome), Rested, RESTClient (Firefox) и RESTMan (Opera).

1.5.3 Другие инструменты для просмотра и отправки HTTP-запросов

Вне браузера вам также доступно множество других инструментов для отправки или просмотра HTTP-запросов. Среди них и инструменты

командной строки (например, curl, wget и httpie), и настольные клиенты (например, SOAP-UI). Если вы хотите ознакомиться с сетевой информацией, зайдите на страницу net-internals в Chrome или воспользуйтесь анализаторами трафика, такими как Fiddler и Wireshark. О некоторых из таких инструментов мы поговорим в последующих главах, когда будем рассматривать детали HTTP/2, но на данный момент инструментов, упомянутых в этом разделе, должно быть достаточно.

Резюме

- HTTP является одной из основных технологий в Internet.
- Для того чтобы загрузить веб-страницу, веб-браузеры делают несколько HTTP-запросов.
- Изначально протокол HTTP был простым текстовым протоколом.
- За последние 20 лет HTTP стал сложнее, но все еще остался протоколом текстового формата.
- HTTPS шифрует стандартные HTTP-сообщения.
- Для просмотра и отправки HTTP-сообщений существуют различные инструменты.