Оглавление

Протокол HTTP

1. Особенности изучения темы и исторический обзор

В этой и четырех следующих лекциях всесторонне изучается протокол HTTP (hyper-text transfer protocol) – протокол передачи гипертекста. Начнем с особенностей изучения этой темы.

Разработанный Тимом Бернерсом Ли в конце 80-х – начале 90-х, протокол HTTP за более чем 25-летнюю историю прошел большой пусть развития. Ниже приведены основные вехи.

В 1992 г. появилась версия 0.9, очень простая, был предусмотрен только один метод запроса данных – метод GET.

В мае 1997 г. был опубликован документ RFC 1945, содержавший описание HTTP/1.0. Это был большой шаг вперед по сравнению с версией 0.9.

В январе 1997 г. был опубликован документ RFC 2068, содержавший описание HTTP/1.1. В июне 1999 г. вышло обновление HTTP/1.1 в документе RFC 2616, и этот стандарт (с обновлениями и расширениями) продержался долгие 15 лет. За это время вышли RFC 2817, 5785, 6266, 6585, которые, в частности, определили процедуру переключения на защищенный протокол TLS, дополнительные заголовки и коды состояния.

В июне 2014 г. вышла серия RFC 7230 – 7235 суммарным объемом 305 страниц, посвященная HTTP/1.1, которая отменила как устаревший RFC 2616. Каждый из этих документов рассматривает отдельный аспект протокола, например, синтаксис, кэширование или способы авторизации.

В мае 2015 г. были опубликованы документ RFC 7540, 7541, содержавшие описание HTTP/2.0. В отличие от предыдущих версий, протокол HTTP/2.0 является бинарным. Среди ключевых особенностей – мультиплексирование запросов, расстановка приоритетов для запросов, сжатия заголовков, загрузка нескольких элементов параллельно, посредством одного TCP соединения, поддержка push уведомлений со стороны сервера.

На изучение HTTP отводится на более 5 пар, т. е. у студентов есть не более месяца на освоение протокола, что защищает документацию (объем которой превышает 460 страниц) от прочтения. К счастью многие формальности можно опустить, в частности, расширенные формы Бекуса-Наура на начальном этапе только мешают изучению. Существует немало самоучителей, поверхностно освещающих формат протокольных сообщений, по прочтению которых складывается ложное убеждение о понимании протокола, так как

многие важные аспекты в них даже не упоминаются. Первая особенность нашего изложения в том, что мы постараемся всесторонне рассмотреть протокол и создать «арматуру» понимания. Этих знаний будет достаточно для дальнейшего осмысленного чтения документации, статей и книг.

Сложности в изучении добавляет и тот факт, что для начала подробного и детализированного изучения HTTP необходимо заранее знать, многие протокольные возможности и особенности, которые должны изучаться в лишь конце. Чтение документации напоминает фильмы Квентина Тарантино с нелинейным сюжетом. Вторая особенность изложения состоит в том, что подача материала будет вестись как «по спирали», сначала рассматриваем простейшую схему взаимодействия клиента и сервера, а потом последовательно рассматриваем отдельные темы все более и более подробно.

2. Взаимодействие клиента и сервера по протоколу HTTP/1.1

Рассмотрим простейшую систему, состоящую из веб клиента и веб сервера.

Сервер – компьютер, подключенный к сети и слушающий, не придет ли запрос; а когда запрос приходит, сервер его обрабатывает и отвечает. Термин «сервер» является перегруженным, т. е. имеет несколько смыслов.

1) Сервер – компьютер (или специальное компьютерное оборудование), выделенный и/или специализированный для выполнения определенных сервисных функций.
2) Сервер – программное обеспечение, принимающее запросы от клиентов.

Итак, клиент посылает запрос. Как же выглядит этот запрос в простейшем случае? Этот запрос мог выглядеть так: «Дай мне документ, расположенный по адресу /docs и имеющий имя index.html». Сервер, обработав запрос и поняв, что нужно клиенту, отвечает «OK, вот запрошенный документ, он имеет формат html, его длина 3051 байт».

Рис. 1. Взаимодействие сервера и клиента

Простейшая схема описана, в следующих параграфах рассматривается формат сообщений и, от части, синтаксис протокола. Как сказано выше, изложение будет вестись «по спирали», при этом читатель должен понимать, что протокольные возможности почти полностью определяются его заголовками. Например (без привязки к HTTP), если в заголовках предусмотрено поле но-

мера пакета или временной отметки, то клиент сможет упорядочить пакеты, доставленные в неверном порядке или отбросить более неактуальные пакеты. Таким образом, какую бы тему мы далее не изучали – установление соединений, согласование содержимого, кэширование, авторизацию – мы будем следовать схеме: проблема => эвристические методы ее решения => протокольные методы и заголовки.

Поясним, что такое гипертекст, который передается по протоколу. Гипертекст – текст, сформированный с помощью языка разметки, потенциально содержащий в себе гиперссылки, т. е. возможности перехода между отдельными частями документа. В Интернете наиболее распространенным языком создания гипертекстовых документов является HTML (hypertext markup language) – язык разметки гипертекста. Приведем простейший пример HTML документа – текстового файла с расширением .html.

```
<!DOCTYPE html public "-//w3c//dtd html 4.0//en">
<html>
  <head>
    <title>Первый html документ</title>
  </head>
  <body>
    <p><a href='two.html'>Второй html документ</a></p>
    <div><img src='photo.jpg' alt='tiger'></div>
  </body>
</html>
```

Не смотря на свое название протокол HTTP способен передавать не только текстовые данные, но и произвольные бинарные, например, изображения и видео. HTTP используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP, XML-RPC, WebDAV.

Протокол HTTP/1.1 является *текст ориентированным*. Что это означает? Это означает то, что служебные протокольные данные представлены в виде текстовых команд, заголовков, кодов. Если в захватить HTTP/1.1 пакет и открыть в текстовом редакторе, его содержимое будет понятно человеку. В этом смысле HTTP отличается от DNS или NTP, которые являются бит ориентированными. Еще раз подчеркнем, что текст ориентированность – это свойство протокольных элементов, а не передаваемых данных; данные как раз могут быть не текстовыми, например, картинками или произвольными файлами, типа .zip архивов или .pdf книг. Протокол HTTP/2.0 является бит ориентированным, его рассмотрение отложим до последней лекции.

Большинство HTTP-обменов инициируются пользователем и состоят из запросов ресурсов, имеющихся на определенном сервере. В простейшем случае такой запрос может быть реализован путем соединения пользовательского агента и базового сервера.

Более сложная ситуация возникает, когда присутствует один или более посредников в цепочке обслуживания запроса/отклика. Существует три стандартные формы посредников: прокси, туннель и внешний порт (gateway).

1. Прокси представляет собой агент переадресации, получающий запрос для URI, переписывающий все сообщение или его часть и отправляющий переделанный запрос серверу, указанному URI.
2. Внешний порт (gateway) представляет собой приемник, который работает на уровень выше некоторых других серверов и транслирует, если необходимо, запрос нижележащему протоколу сервера.
3. Туннель действует как соединитель точка-точка и не производит каких-либо видоизменений сообщений. Туннель используется тогда, когда нужно пройти через какую-то систему (например, Firewall) в условиях, когда эта система не понимает (не анализирует) содержимое сообщений.

Любой участник обмена, который не используется в качестве туннеля, может воспользоваться кэшем для запоминания запросов. Кэш может сократить длину цепочки в том случае, если у одного из участников процесса имеется в буфере отклик для конкретного запроса, что может, кроме прочего, заметно снизить требования к пропускной способности сети.

3. Формат сообщений

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

1) стартовая строка (англ. starting line) – определяет тип сообщения;
2) заголовки (англ. headers) – характеризуют тело сообщения, параметры передачи и прочие сведения;
3) тело сообщения (англ. message body) – непосредственно данные сообщения, отделенные от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа. Заголовки отделяются от тела пустой строкой, т. е. строкой содержащей только два символа CRLF.

Строка запроса имеет формат:

```
метод URI HTTP/версия
```

Здесь

- метод (англ. method) – команда, определяющая, что клиент просит у сервера седлать с идентифицируемым ресурсом; одно слово заглавными буквами, например, GET, HEAD, POST, OPTION;
- URI (англ. uniform resource identifier) – унифицированный идентификатор ресурса, обычно представленный в виде адреса сервера и пути к ресурсу на сервере;
- версия (англ. version) – пара разделенных точкой цифр, например, 1.1.

Например, чтобы запросить картинку title.gif с сервера example.org, клиент должен послать запрос:

```
GET / title.gif HTTP/1.1
Host: example.org
```

Рис. 2. Взять рис. 3-4.

Стартовая строка ответа сервера имеет формат:

HTTP/версия код_состояния пояснение,

Здесь

- версия – пара разделенных точкой цифр;
- код состояния (англ. status code) – три цифры, определяющие дальнейшее содержимое сообщения и поведение клиента;
- пояснение (англ. reason phrase) – текстовое короткое пояснение к коду ответа для пользователя, которое никак не влияет на сообщение и является необязательным.

Например, ответ сервера на предыдущий запрос может выглядеть так:

```
HTTP/1.1 200 OK
Content-type: image/gif
Content-length: 5346
```

4. Методы

Для начала приведем список стандартных методов с их кратким описанием.

Метод	Описание
GET	Запрашивает документ с сервера
HEAD	Запрашивает только заголовки документа с сервера
POST	Отправляет данные на сервер для обработки
PUT	Сохраняет данные из тела запроса на сервере
DELETE	Удаляет документ на сервера
CONNECT	Преобразует соединение запроса в прозрачный TCP/IP туннель
TRACE	Трассирует сообщение через прокси до сервера назначения
OPTION	Запрашивает коммуникационные опции сервера или ресурса

Табл. 1. Стандартные HTTP методы и их краткое описание.

Отметим, что веб сервера зачастую поддерживают не все методы[1]. Чтобы быть совместимым с HTTP/1.1, веб сервер должен поддерживать методы GET и HEAD. Даже когда сервер поддерживает все указанные в табл. № 1 методы, использование их ограничено политиками безопасности. Не стоит, к примеру, разрешать использование методов DELETE или PUT неавторизированным пользователям. Обычно политики задаются в конфигурационных файлах.

Стандарт определяет *безопасные методы*. Установлено соглашение, что методы GET и HEAD никогда не должны выполнять какие либо функции помимо доставки информации. Эти методы должны рассматриваться как вполне безопасные. С дугой стороны методы PUT или DELETE приводят к выполнению действий на стороне сервера, связанных с сохранением или удалением ресурса, что делает эти методы небезопасными.

Естественно, невозможно гарантировать, что сервер не будет вызывать побочные эффекты, как следствие выполнения запроса GET; в действительности, некоторые динамические ресурсы предусматривают такую возможность. Например, GET запрос к .php или .pl скриптам приводят к запуску этих серверных скриптов; что буду делать эти скрипты – зависит от программиста, например, они могут удалять аккаунт пользователя, что отнюдь не является «безопасной» операцией.

Стандарт определяет *идемпотентные методы*, т. е. такие, что побочный эффект от $N > 0$ идентичных запросов является таким же, как и от одного запроса (помимо ошибок и таймаутов). Методы GET, HEAD, PUT и DELETE являются таковыми.

[1] Серверам рекомендуется возвращать статусный код 405 (Метод не допустим), если метод известен серверу, но не приемлем для запрашиваемого ресурса, и 501 (Не применим), если метод не узнан или не приемлем для сервера. Список известных серверу методов может быть представлен в поле заголовка отклика Public, а список методов поддерживаемых конкретным ресурсом – в заголовке Allow.

4.1. Метод GET

Метод GET предполагает извлечение любой информации (в форме объекта), заданной Request-URI. Если Request-URI относится к процессу, генерирующему данные, например .php скрипту, то в результате в виде объекта будут присланы эти данные, а не исходный текст самого процесса.

Забегая вперед отметим, что семантика метода меняется на условный, если сообщение-запрос включает в себя поля заголовка If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match или If-Range. Метод условного GET запрашивает, пересылку объекта только при выполнении требований, описанных в соответствующих полях заголовка. Рассмотрим пример.

Утром пользователь запросил с сервера файл prices.html, содержащий список цен; полученный файл браузер отобразил пользователю на экране и сохранил в собственном кэше. Днем пользователю снова потребовался этот же файл. С одной стороны посылать такой же запрос на сервер снова, не самое хорошее решение, так как скорее всего цены не менялись, и сеть будет загружена пересылкой того, что у браузера уже было в кэше. С другой стороны просто взять данные из кэша тоже решение не самое лучшее – цены могли измениться. Решение состоит в том, чтобы послать GET запрос файла prices.html и выставить заголовок If-Modified-Since (англ. Если-Изменен-С) с соответствующим значением, что означает «Если файл изменен с момента X, то вышли файл мне заново, иначе, просто скажи, что не менялся, и я буду пользоваться тем, что ты мне выслал ранее».

Таким образом, метод условного GET имеет целью уменьшить ненужное использование сети путем разрешения актуализации кэшированых объектов без посылки множественных запросов или пересылки данных, которые уже имеются у клиента.

Семантика метода меняется на частичный, если сообщение запроса включает в себя поле заголовка Range. Метод частичного GET ориентирован на уменьшение ненужного сетевого обмена, допуская пересылку лишь части объекта, которая нужна клиенту, например, первых двух страниц большого .pdf документа. Условные и частичные GET запросы будут подробно рассмотрены в дальнейших параграфах.

4.2. Метод HEAD

Метод HEAD идентичен GET за исключением того, что сервер не должен присылать тело сообщения. Метаинформация, содержащаяся в заголовках

отклика на запрос HEAD должна быть идентичной информации посланной в отклик на запрос GET.

При первом знакомстве с методом HEAD может возникнуть когнитивный диссонанс: с одной стороны, тела в отклике нет, и, казалось бы, значение заголовка Content-Length должно быть рано нулю, с другой стороны «метаинформация, содержащаяся в заголовках отклика на запрос HEAD должна быть идентичной информации посланной в отклик на запрос GET». Протокол устанавливает (см. определение метода HEAD), что, не смотря на отсутствие тела сообщения, заголовок Content-Length имеет значение как в ответе на GET запрос. Логика этого правила вытекает из предназначения метода HEAD.

Метод HEAD может использоваться для получения метаинформации об объекте, указанном в запросе, без передачи тела самого объекта. Этот метод часто используется для тестирования гипертекстных связей на корректность, доступность и актуальность.

Отклик на запрос HEAD может кэшироваться в том смысле, что информация, содержащаяся в отклике, может использоваться для актуализации кэшированных ранее объектов данного ресурса. Если новые значения поля указывают на то, что кэшированный объект отличается от текущего объекта (как это индицируется изменением Content-Length, Content-MD5, ETag или Last-Modified), тогда запись в кэше должна рассматриваться как устаревшая.

4.3. Метод POST

Метод POST применяется для передачи пользовательских данных заданному ресурсу, при этом передаваемые данные включаются в тело запроса. Например, посетитель вводит свои комментарии в HTML-форму, после чего они передаются серверу методом POST, и сервер помещает их на страницу. Аналогично с помощью метода POST обычно загружаются файлы на сервер.

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

Операция, выполняемая методом POST, может не иметь последствий для ресурса, который может быть идентифицирован URI. В этом случае приемлемым откликом является 200 (OK) или 204 (No Content – никакого содержимого), в зависимости от того, включает ли в себя отклик объект, описывающий ресурс.

Если ресурс был создан на исходном сервере, отклик должен быть равен 201 (Created – создан) и содержать с указанием URI нового ресурса в заголовке Location.

Замечание. Пользовательские данные на сервер можно передавать не только методом POST в теле запроса, но и методом GET как параметры адресной строки. Например, в запросе

```
GET /site.com/cgi/script.php?user=Vova&age=23 HTTP/1.1
```

Часть после знака вопроса – это передаваемые параметры и их значения. Однако на длину URL существуют ограничения, а потому метод POST лучше подходит для передачи больших данных, например, файлов.

Оставшиеся методы будут рассмотрены в следующей лекции подробно.

5. Заголовки

Заголовки HTTP – это служебные строки в HTTP-сообщении, каждое поле заголовка состоит из имени, за которым следует двоеточие, и поля значения. Заголовки привносят дополнительную метаинформацию в HTTP запросы и ответы. Поля имен безразличны в отношении использования строчных и прописных букв. Рекомендуют начинать поле значения с одного пробела. Поля заголовка могут занимать несколько строк, каждая новая строка должна открываться, по крайней мере, одним символом пробела или табуляции, как показывает пример.

```
content-type: text/html;
    charset=windows-1251
```

Все заголовки разделяются на четыре группы.

Общие заголовки (англ. General Headers) могут включаться как в запросы, так и отклики. Можно сказать, что эти заголовки являются инфраструктурными для протокола HTTP, ибо характеризуют сообщение и процесс его передачи, а не передаваемый в этом сообщении объект. Приведем примеры общих заголовков.

- Cache-Control – управление кэшированием, в том числе на промежуточных прокси-серверах.
- Connection – спецификация опций, которые желательны для конкретного соединения; например, оставить TCP соединение открытым после завершения передачи сообщения или сразу закрыть.

- Upgrade – перечисление дополнительных коммуникационных протоколов, которые клиент или сервер хотели бы использовать.

Ниже показаны примеры использования этих трех заголовков:

```
Cache-Control: max-age=3600
Connection: keep-alive
Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

Заголовки запроса (англ. Request Headers) используются только в запросах клиента. Поля заголовка запроса позволяют клиенту передавать серверу дополнительную информацию о запросе и о самом клиенте. Эти поля действуют как модификаторы запроса. Приведем примеры заголовков запроса.

- User-Agent – информация об агенте пользователя, инициировавшем запрос. Этот заголовок нужен для сбора статистических данных, отслеживания нарушений протокола и автоматического распознавания агентов пользователя. Поле может содержать несколько кодов продуктов, комментарии, идентифицирующие агента и любые субпродукты, которые образуют существенную часть агента пользователя. Согласно договоренности коды программных продуктов перечисляются в порядке их важности для идентифицируемого приложения.
- Accept – спецификация определенных MIME-типов и их приоритетов, которые приемлемы для клиента, запросившего данный ресурс.
- Accept-Language – спецификация набора естественных языков, которые предпочтительны в отклике на запрос.
- Host – спецификация имени сервера в Интернете. Старые HTTP/1.0 клиенты предполагали однозначное соответствие IP адресов и веб сайтов, т. е. на одном компьютере мог работать один веб сервер, который обслуживал один веб сайт. Сегодня на одном сервере может обслуживаться несколько сайтов (так называемый «виртуальный хостинг»). Использование заголовка Host помогает серверу правильно разруливать запросы и направлять их к нужному сайту.

Ниже показаны примеры использования этих четырех заголовков:

```
Accept: text/html, text/plain; q=0.8
Accept-Language: da, en-gb; q=0.8, en; q=0.7
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:2.0.1)
Gecko/20100101 Firefox/4.0.1
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
Host: www.w3.org
```

Заголовки ответа (англ. Response Headers) используются только в ответах сервера. Поля заголовка отклика позволяют серверу передавать дополнительную информацию об отклике, который не может быть помещен в статусную строку. Эти поля заголовка дают информацию о сервере и доступе к ресурсу, идентифицированному Request-URI. Приведем примеры заголовков ответа.

- Server – информация о программном обеспечении, используемым исходным сервером для обработки запросов. Поле может содержать коды многих продуктов, комментарии, идентифицирующие сервер, и некоторые важные субпродукты. Коды программных продуктов перечисляются в порядке важности приложений.
- Retry-After – указание на то, как еще долго данная услуга предполагается быть недоступной для запрашивающего клиента. Значением заголовка может быть либо дата, либо целое число секунд после отправки отклика. Поле заголовка отклика Retry-After может использоваться с кодом статуса 503 (Service Unavailable).
- WWW-Authenticate – спецификация требований, указывающих на схему идентификации и параметры, применимые для запрошенного ресурса. Заголовок WWW-Authenticate должен включаться в сообщения откликов со статусным кодом 401 (Unauthorized).

Ниже показаны примеры использования этих двух заголовков:

```
Retry-After: Fri, 31 Dec 2015 23:59:59 GMT
Server: Apache/2.2.17 (Win32) PHP/5.3.5
WWW-Authenticate: Basic realm="myRealm"
```

Заголовки сущности (англ. Entity Headers) или, что то же самое, заголовки объекта, сопровождают каждую сущность сообщения. Сообщения запрос и отклик могут нести в себе объект, если это не запрещено методом запроса или статусным кодом отклика. Заголовки объекта характеризуют тело объекта, хотя некоторые отклики включают в себя только заголовки объектов. Приведем примеры заголовков сущности.

- Allow – перечисление методов, поддерживаемых ресурсом, идентифицированным Request-URI. Целью этого заголовка является точное информирование клиента о рабочих методах данного ресурса. Заголовок Allow должен быть представлен в отклике 405 (Method Not Allowed).
- Content-Language – описание естественных языков потенциальных читателей вложенного объекта. Заметим, что это может быть не эквива-

лентно всем языкам, использованным в теле объекта. Примером может быть языковый курс для начинающих «A First Lesson in Latin», который предназначен для англо-говорящей аудитории. В этом случае заголовок Content-Language должен включать только "en". Заголовок Content-Language может быть применен к любому типу среды, не ограничен только текстовыми документами.

- Expires – время, начиная с которого отклик может считаться устаревшим.

Ниже показаны примеры использования этих трех заголовков:

```
Allow: OPTIONS, GET, HEAD
Content-Language: en
Expires: Sat, 31 Jan 2015 15:02:53 GMT
```

Рекомендуется посылать заголовки получателю в следующем порядке: общие заголовки, заголовки запроса или отклика, заголовки сущности.

Нет смысла подробно разбирать все заголовки в первой лекции, вышеприведенных достаточно для общего знакомства. В следующих лекциях будут разобраны отдельные темы (например, согласование содержимого или условные запросы) и там мы подробно остановимся на нужных в каждой теме заголовках.

6. Коды состояний

Код состояния (англ. Status-Code) представляет собой 3-значный цифровой результирующий код попытки понять и исполнить запрос. Словесный комментарий (англ. Reason-Phrase), сопровождающий код, предназначен для того, чтобы дать краткое описание статусного кода. Статусный код служит для использования автоматами, а словесный комментарий для пользователей.

Первая цифра статусного кода определяет класс отклика, последние две цифры не имеют четко определенной функции. Существует пять значений первой цифры и, соответственно, пять классов кодов состояний.

1xx: Информационный (Informational). Этот класс статусных кодов соответствует информационным откликам, состоящим только из статусной строки и опционных заголовков с пустой строкой в конце. На сегодня определено лишь два информационных отклика: 100 Continue и 101 Switching Protocols. Кратко обсудим их.

- 100 Continue (продолжение). Сервер сообщает, что клиент может продолжать работу, получив этот отклик. Этот промежуточный отклик ис-

пользуется для информирования клиента о том, что начальная часть запроса получена и пока не отклонена сервером. Клиенту следует продолжить отправлять оставшуюся часть запроса, если же запрос уже отправлен, то игнорировать этот отклик. Сервер должен послать окончательный отклик по завершении реализации запроса.

- 101 Switching Protocols (Переключение протоколов). Если клиент запросил переключение протокола с HTTP/1.1 на другой, например, HTTP/2.0 или TLS, сервер оповещает клиента о том, что он понял и принял к исполнению запрос. С помощью поля заголовка сообщения Upgrade клиент уведомляется об изменении прикладного протокола для данного соединения.

  Протокол следует изменять лишь в случае, если он предоставляет существенные преимущества. Например, переключение на новую версию HTTP предоставляет преимущества по отношению к старой версии, а переключение на синхронный протокол реального времени может иметь преимущество, когда ресурс использует это свойство.

2xx: Успех (Success). Этот класс статусных кодов показывает, что запрос клиента благополучно получен, понят и принят к исполнению. Стандарт определяет 7 кодов успеха. Приведем для примера описание трех, остальные рекомендуем посмотреть в RFC 7231.

- 200 OK. Запрос успешно исполнен. Информация, возвращаемая вместе с откликом, зависит от метода, использованного запросом, например:

  GET – объект, соответствующий запрошенному ресурсу;

  HEAD – поля заголовка объекта (без какого-либо тела), соответствующего запрошенному ресурсу;

  POST – объект, описывающий или содержащий результат операции;
- 201 Created (Создано). Запрос исполнен и в результате создан новый ресурс, который доступен через URL, присланный в заголовке Location.
- 206 Partial Content (Частичное содержимое). Сервер прислал не весь ресурс, а ту его часть, которую запросил клиент. Запрос должен включать заголовок Range, указывающий на желательный диапазон, например, Range: bytes=516-894.

3xx: Переадресация (Redirection). Этот класс статусных кодов указывает, что для выполнения (завершения выполнения) запроса, нужны дальнейшие действия агента пользователя. *Необходимые действия могут быть выполнены агентом пользователя без взаимодействия с пользователем, тогда и только*

тогда, когда используемый метод соответствует GET или HEAD. Стандарт определяет 7кодов переадресации. Приведем для примера описание трех, остальные рекомендуем посмотреть в RFC 7231.

- 300 Multiple Choices (Множественный выбор). Сервер сообщает, что запрошенный ресурс имеет более чем одно представление, каждое со своим собственным идентификатором, отсылает информацию об этих альтернативах; таким образом, пользователь (или пользовательский агент) может выбрать наиболее предпочтительный вариант представления.
- 307 Temporary Redirect (Временно перемещен). Запрошенный ресурс временно располагается по другому URL, и сервер сообщает этот URL в заголовке Location. Браузер может использовать значение этого заголовка для автоматического перехода. Тело ответа (если есть) обычно содержит гиперссылку на перемещенный ресурс и пояснения для пользователя.
- 304 Not Modified (Не модифицировано). Упрощенно говоря, сервер сообщает клиенту, что ресурс не менялся с момента последнего запроса, клиент может пользоваться закэшированной версией.

Говоря о перенаправлении заметим, что агент пользователя не должен автоматически переадресовывать запрос более чем 5 раз, так как такая переадресация обычно свидетельствует о зацикливании запроса.

4xx: Ошибка клиента (Client Error). Запрос содержит синтаксическую ошибку или не может быть выполнен. Наиболее частая ошибка 404 Not Found (Не найдено) говорит о том, что пользователь ошибся в URL.

5xx: Ошибка сервера (Server Error). Сервер не смог выполнить корректный запрос.

- 500 Internal Server Error (Внутренняя ошибка сервера). Сервер столкнулся с непредвиденными условиями, которые мешают ему исполнить запрос. Эта ошибка часто возникает при отладке серверных скриптов.
- 502 Bad Gateway (Плохой шлюз). Сервер при работе в качестве шлюза или прокси получил неверный отклик от вышестоящего сервера, к которому он обратился, выполняя запрос.

7. Тело сообщения

Как было сказано ранее, сообщения состоят из стартовой строки, заголовков и, возможно, тела. Заголовки несут в себе полезную информацию, но при

этом информация, представленная в заголовках, как правило, предназначена для пользовательского агента (браузера или менеджера закачек), а не для пользователя. В свою очередь тело сообщения несет в себе содержимое, предназначенное для пользователя – страницу .html или картинку. В стандарте [RFC 2616] написано «тело HTTP сообщения, если оно присутствует, используется для передачи тела объекта, связанного с запросом или ответом». Таким образом, для дальнейшего изучения нам надо ввести понятия «объект», «тело объекта».

Объект – это те полезные данные (и метаинформация об этих данных), которые передаются в сообщении. И запрос, и отклик могут нести в себе объект, если это не запрещено методом запроса или статусным кодом отклика (например, отклик на HEAD тела не содержит). Объект состоит из *заголовков объекта* и *тела объекта*, хотя некоторые отклики включают в себя только заголовки объектов. Картинка, которую будут передавать по сети, составляет тело объекта, а поля заголовков Content-Length: 3021, Content-Type: img/gif, Content-MD5: Q2hlY2sgSW50ZWdyaXR5IQ== – это примеры заголовков объекта; вместе они образуют объект.

Можно заметить, что объект «размазан» по сообщению: заголовки объекта попадают в заголовки сообщения (и соседствуют там с заголовками клиента или сервера), а тело объекта попадает в тело сообщения.

В самом простом случае тело сообщения совпадает с телом объекта. В случае[2], когда применяется транспортное кодирование, что указывается заголовком Transfer-Encoding, тело сообщения отличается от тела объекта. Сразу поясним, что Transfer-Encoding – это удобный способ передачи динамически формируемого контента (см. п. 7.1), он ничего общего не имеет с Content-Encoding – способом сжатия данных[3].

7.1 Chunked transfer encoding

Легко решается задача передачи статического контента, например, .html или .css фалов, существовавших на сервере еще до поступления запроса. В этом случае значение заголовка объекта Content-Length известно серверу сразу, Content-MD5 тоже можно вычислить заранее. Теперь, когда придет клиентский запрос, сервер сможет выслать заголовки объекта и вслед за тем передать тело объекта.

---

[2] И только в этом случае.

[3] Хранимые на сервере данные можно сжать алгоритмом gzip и передавать по сети файлы меньшего объема. Достоинство такого метода сжатия в том, что он допускает разжатие «на лету», что важно для быстрого парсинга получаемых данных. Клиент может начать разжимать данные, не дожидаясь получения всего файла. Способы сжатия задаются в заголовке Content-Encoding.

Задача усложняется, когда контент формируется динамически; примером тому может служить генерация большой HTML страницы по результатам доступа к базе данных. В этом случае невозможно послать заголовок Content-Length, не имея самого объекта. Ни имея значения заголовка Content-Length, клиент не будет знать, где объект закончился[4].

Первый способ решения проблемы – свести задачу к предыдущей, т. е. сформировать объект полностью, забуферизировав его на диске сервера, а потом, зная его длину и вычислив хэш Content-MD5, отправлять сообщение так же, как раньше в случае со статическим контентом. Минусы этого решения очевидны.

Второй способ – передавать динамически формируемое сообщение частями, с применением специального транспортного кодирования[5]. На сегодняшний день в HTTP/1.1 определен только один способ транспортного кодирования – транспортное кодирование по частям (Chunked transfer encoding).

Транспортное кодирование по частям модифицирует тело сообщения для того, чтобы передать его в виде последовательности пакетов, каждый со своим индикатором размера, тело сообщения завершается пакетом нулевой длины, за которыми следует опционная завершающая запись (footer) и пустая строка. Назначение завершающей записи заключается в том, чтобы дать информацию о динамически сформированном объекте, например, передать заголовки Content-MD5, Expires или заголовки с цифровой подписью, которые, возможно, появятся в будущих расширениях HTTP.

Ниже приведено формальное описание сообщения, к которому применено транспортное кодирование [RFC 2616].

```
Chunked Body = *chunk "0" CRLF footer CRLF
Chunk = chunk-size [ chunk-ext ] CRLF chunk-data CRLF
Hex-no-zero = Шестнадцатеричное число неравное 0
Chunk-size = hex-no-zero *HEX
Chunk-ext = *(";" chunk-ext-name ["=" chunk-ext-val])
Chunk-ext-name   = token
Chunk-ext-val    = token | quoted-string
```

[4] Забегая вперед отметим, что для определения размера тела сообщения заголовок Content-Length, вообще говоря, не требуется. Например, при передаче байтовых диапазонов заголовок Content-Range позволяет определить размер тела сообщения. Детально этот вопрос будет разобран позже.

[5] Название «транспортное кодирование» может сбить с толку. Этот способ ничего не имеет общего с транспортным уровнем, на котором работает протокол TCP.

```
Chunk-data   = chunk-size(OCTET)
footer   = *entity-header
```

Читатель, возможно, удивлен тем, что часть заголовков может передаваться после тела сообщения. Да, такое допустимо при использовании транспортного кодирования. Среди заголовков, которые присутствуют до разделенного тела, выставляется заголовок Trailer; его значениями являются названия тех заголовков, которые будут присутствовать в footer части. Приведем пример.

Ответ сервера с использованием заголовка Content-Length.

```
HTTP/1.1 200 OK
Date: Mon, 22 Mar 2004 11:15:03 GMT
Content-Type: text/html
Content-Length: 129
Expires: Sat, 28 Mar 2015 21:12:00 GMT

<html><body><p>The file you requested is 3,400 bytes
long and was last modified: Sat, 20 Mar 2004 21:12:00
GMT.</p></body></html>
```

Ответ сервера использующего транспортное кодирование по частям.

```
HTTP/1.1 200 OK
Date: Sat, 28 Mar 2015 11:15:03 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Trailer: Expires
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.3.6

29
<html><body><p>The file you requested is
5
3,400
23
bytes long and was last modified:
1d
Sat, 20 Mar 2004 21:12:00 GMT
13
.</p></body></html>
```

```
0
Expires: Sat, 27 Mar 2004 21:12:00 GMT
```

Приведем алгоритм на псевдокоде, описывающий обработку сообщения, к которому было применено транспортное кодирование. Здесь обрабатывается и тело, и возможные заголовки из части footer.

```
length := 0
read chunk-size, chunk-ext (if any), and CRLF
while (chunk-size > 0) {
  read chunk-data and CRLF
  append chunk-data to decoded-body
  length := length + chunk-size
  read chunk-size, chunk-ext (if any), and CRLF}
read trailer field
while (trailer field is not empty) {
  if (trailer field is allowed to be sent in a trailer) {
    append trailer field to existing header fields
  }
  read trailer-field
}
Content-Length := length
Remove "chunked" from Transfer-Encoding
Remove Trailer from existing header fields
```

Контрольные вопросы

1. Протокол HTTP использует понятия:

1) ресурс, хранимый на сервере и идентифицируемый с помощью URL;
2) объект, передаваемый в сообщении;
3) HTTP сообщение, возможно, несущее в себе объект;
4) тело сообщения, тело объекта.

Какая связь между этими понятиями? Разберите все варианты взаимоотношений: что без чего может (не может) существовать, что является составной частью другого, является одним из вариантов представления другого и т. д.

2. В чем кардинальная разница между заголовками и параметрами, которые передаются в URL? То есть, зачем нужны URL-параметры, если можно вводить свои собственные заголовки и в них передавать информацию на сервер?