Лабораторная работа № 1

Изучение протокола HTTP

Задание

Данная работа является вводной в курсе и призвана ознакомить студента с базовым протоколом веба – протоколом HTTP.

1. Базовая часть работы – 5 баллов
   - 1.1. Выбрать три любых веб-сервиса по собственному усмотрению (социальная сеть, почта, облако, новостной сайт и т.д.)
   - 1.2. Убедиться, что протокол хотя бы одного веб-сервиса реализует подход REST (т.е. работает с GET, POST, PUT и DELETE запросами), и у хотя бы одного есть открытое API с документацией (например, любая социальная сеть). Проверить реализацию REST можно либо изучив API соответствующего сервиса (если оно открыто и есть спецификация), либо изучив трафик запросов в инструментах разработчика в браузере (подробнее - ниже), либо вручную отправляя запросы, либо с помощью запроса OPTIONS (вариант не очень хороший, так как этот метод бывает реализован редко). Если это не так, вернуться в предыдущий пункт.
   - 1.3. С помощью браузера (или другого инструмента, см. ниже) вручную отправить (как это сделать см. ниже) на каждый сервис:
     - 1.3.1. OPTIONS, чтобы получить список доступных методов
     - 1.3.2. Запрос HEAD
     - 1.3.3. Запросы GET и POST не менее чем с двумя параметрами (для сервиса с API отправить 3 GET и 2 POST-запроса)
     - 1.3.4. Запросы PUT и DELETE
   - 1.4. Полученный ответ зафиксировать (либо в браузере, либо в текстовом файле)
   - 1.5. Полностью разобраться в полученном ответе от сервера. Уметь ответить на любой вопрос. Для каждого сервиса проанализировать реализуемый им протокол и сделать вывод. Реализуют ли они стандарт HTTP? Реализуют ли подход REST?
   - 1.6. На самом деле, не очень важно, что вам ответит сервер: что-то осмысленное или сообщение об ошибке. Цель работы – дать представление о работе HTTP и структуре HTTP-запросов и ответов. Ну и посмотреть реализацию протокола на практике. К слову, она редко бывает полной. Совет: обратите внимание на сервисы, которые имеют API с документацией. С такими

сервисами работать проще и, в некотором плане, интереснее. Хотя, разобраться в работе любимого сайта и взломать парочку запросов может быть очень занимательным.

2. Достижение #1 (+ 1 балл): С помощью таких ручных запросов добиться реальных изменений на сервере – например, отправить комментарий, переключить язык, добавить товар в корзину и т.д.
3. Дополнительная часть работы – 3 балла
   На любом языке программирования написать программу (скрипт), выполняющую действия 1.3 – 1.5 автоматически

Теоретический материал

1. Инструменты

Основным инструментом в этой работе будет являться браузер. По умолчанию будем иметь в виду, что это Google Chrome (если у вас другие предпочтения, то пользуйтесь другим бразуером, но там все может быть немного по-другому).

Для просмотра HTTP-траффика:

- Откройте браузер
- Зайдите на интересующий веб-сайт
- Нажмите F12
- В открывшейся панели (или отдельном окне) Инструментов разработчика (Developer Tools) выберите вкладку Сеть (Network).
- Откроется список запросов. Он отражает абсолютно все HTTP-запросы, которые выполняет открытый сайт, будь то запросы ресурсов, реакция на действия пользователя, или работа скриптов.

Конечно, при желании вы можете воспользоваться специализированным ПО для сниффинга сетевого траффика, например, WireShark, Fiddler.

Для ручной отправки запросов также можно использовать различные инструменты. Можно отправлять запросы из консоли разработчика бразуера, с помощью скриптов на чистом JavaScript или с использованием библиотеки jQuery. Можно использовать различные консольные утилиты, например telnet или httpie (https://github.com/jkbrzt/httpie). Можно воспользоваться каким-нибудь GUI-приложением для вашей ОС. Например, для Windows - I'm Only Resting или rest-client. Мы же рассмотрим самый простой и для данной работы вполне достаточный вариант – расширением для Google Chrome под названием DHC. Его можно установить из маркета Google для браузера Chrome. На главной панели этого приложения находится конструктор HTTP-

запросов. Можно выбрать протокол (HTTP/HTTPS), набрать URL, указать параметры запроса, выставить необходимые заголовки и нажать Send. Ответ от сервера отобразится ниже. Удачные запросы можно сохранять, что и рекомендуется делать.

2. Протокол HTTP

Протокол передачи данных — набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения разнесённой в пространствеаппаратуры, соединённой тем или иным интерфейсом.

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). **Основой HTTP является технология «клиент-сервер». Протокол является текстовым протоколом, не сохраняющим состояния.**

HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов. HTTP используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP, XML-RPC,WebDAV.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. (В частности для этого используется HTTP-заголовок.) Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

HTTP — протокол прикладного уровня, аналогичными ему являются FTP и SMTP. Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии,

связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера). Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

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

- **Серверы** как основные поставщики услуг хранения и обработки информации (обработка запросов). Apache, Internet Information Services (IIS), nginx, Google Web Server, lighttpd.
- **Клиенты** — конечные потребители услуг сервера (отправка запроса). Google Chrome, Mozilla Firefox, Internet Explorer (Edge), Opera.
- **Прокси** для выполнения транспортных служб. Squid, UserGate, Multiproxy, Naviscope, nginx.

В данной работе вам предстоит имитировать работу клиента, отправляющего запросы на сервер.

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

История развития.

- HTTP/0.9
  HTTP был предложен в марте 1991 года Тимом Бернерсом-Ли, работавшим тогда в CERN, как механизм для доступа к документам в Интернете и облегчения навигации посредством использования гипертекста. Самая ранняя версия протокола HTTP/0.9 была впервые опубликована в январе 1992 г. Спецификация протокола привела к упорядочению правил взаимодействия между клиентами и серверами HTTP, а также чёткому разделению функций между этими двумя компонентами. Были задокументированы основные синтаксические и семантические положения.
- HTTP/1.0
  В мае 1996 года для практической реализации HTTP был выпущен информационный документ RFC 1945, что послужило основой для реализации большинства компонентов HTTP/1.0.

- HTTP/1.1
  Текущая версия протокола, принята в июне 1999 года. Новым в этой версии был режим «постоянного соединения»: TCP-соединение может оставаться открытым после отправки ответа на запрос, что позволяет посылать несколько запросов за одно соединение. Клиент теперь обязан посылать информацию об имени хоста, к которому он обращается, что сделало возможной более простую организацию виртуального хостинга.
- HTTP/2
  11 февраля 2015 года опубликованы финальные версии черновика следующей версии протокола. В отличие от предыдущих версий, протокол HTTP/2 является бинарным. Среди ключевых особенностей мультиплексирование запросов, расстановка приоритетов для запросов, сжатия заголовков, загрузка нескольких элементов параллельно, посредством одного TCP соединения, поддержка проактивных push-уведомлений со стороны сервера.

Структура протокола

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

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

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

Стартовая строка

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

- GET URI — для версии протокола 0.9.
- Метод URI HTTP/Версия — для остальных версий.

Здесь:

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

Чтобы запросить страницу, например из википедии, клиент должен передать строку (задан всего один заголовок):

```
GET /wiki/HTTP HTTP/1.0
Host: ru.wikipedia.org
```

Стартовая строка ответа сервера имеет следующий формат:

HTTP/Версия Код Состояния Пояснение

где:

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

Например, стартовая строка ответа сервера на предыдущий запрос может выглядеть так:

```
HTTP/1.0 200 OK
```

Методы

Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру. Каждый сервер обязан поддерживать как минимум методы **GET** и **HEAD.** Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Кроме методов GET и HEAD, часто применяется метод POST. Рассмотрим методы протокола HTTP.

- **OPTIONS**

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

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

Для того, чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «*». Запросы «OPTIONS * HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1.

Результат выполнения этого метода не кэшируется.

- **GET**

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

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:
GET /path/resource?param1=value1&param2=value2 HTTP/1.1

Согласно стандарту HTTP, запросы типа GET считаются идемпотентными.

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

- **HEAD**

Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

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

- **POST**

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

При результате выполнения 200 (Ok) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location.

Сообщение ответа сервера на выполнение метода POST не кэшируется.

- **PUT**

Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content). Сервер не должен игнорировать некорректные заголовки, передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки 501 (Not Implemented).

Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу. Обычно, POST используется для создания ресурса, а PUT – для его редактирования.

Сообщения ответов сервера на метод PUT не кэшируются.

- **PATCH**

Аналогично PUT, но применяется только к фрагменту ресурса.

- **DELETE**

Удаляет указанный ресурс.

- **TRACE**

Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе.

- **CONNECT**

Преобразует соединение запроса в прозрачный TCP/IP-туннель, обычно чтобы содействовать установлению защищённого SSL-соединения через нешифрованный прокси.

Коды состояния

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

- 201 Webpage Created
- 403 Access allowed only for registered users
- 507 Insufficient Storage

Клиент узнаёт по коду ответа о результатах его запроса и определяет, какие действия ему предпринимать дальше. Набор кодов состояния является стандартом, и они описаны в соответствующих документах RFC. Введение новых кодов должно производиться только после согласования с IETF. Клиент может не знать все коды состояния, но он обязан отреагировать в соответствии с классом кода.

В настоящее время выделено пять классов кодов состояния.

- **1xx Informational («Информационный»)**

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

- **2xx Success («Успех»)**

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

- **3xx Redirection («Перенаправление»)**

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

- **4xx Client Error («Ошибка клиента»)**

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

- **5xx Server Error («Ошибка сервера»)**

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

Заголовки

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

Примеры заголовков:

- Server: Apache/2.2.11 (Win32) PHP/5.3.0
- Last-Modified: Sat, 16 Jan 2010 21:16:42 GMT
- Content-Type: text/plain; charset=windows-1251
- Content-Language: ru

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

Все заголовки разделяются на четыре основных группы:

- General Headers («Основные заголовки») — должны включаться в любое сообщение клиента и сервера.
- Request Headers («Заголовки запроса») — используются только в запросах клиента.

- Response Headers («Заголовки ответа») — только для ответов от сервера.
- Entity Headers («Заголовки сущности») — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю.

Все необходимые для функционирования HTTP заголовки описаны в основных RFC. Если не хватает существующих, то можно вводить свои. Традиционно к именам таких дополнительных заголовков добавляют префикс «X-» для избежания конфликта имён с возможно существующими.

Тело сообщения

Тело HTTP сообщения (message-body), если оно присутствует, используется для передачи тела объекта, связанного с запросом или ответом. Тело сообщения (message-body) отличается от тела объекта (entity-body) только в том случае, когда применяется кодирование передачи, что указывается полем заголовка Transfer-Encoding.

Поле Transfer-Encoding должно использоваться для указания любого кодирования передачи, примененного приложением в целях гарантирования безопасной и правильной передачи сообщения. Поле Transfer-Encoding — это свойство сообщения, а не объекта, и, таким образом, может быть добавлено или удалено любым приложением в цепочке запросов/ответов.

Правила, устанавливающие допустимость тела сообщения в сообщении, отличны для запросов и ответов.

Присутствие тела сообщения в запросе отмечается добавлением к заголовкам запроса поля заголовка Content-Length или Transfer-Encoding. Тело сообщения (message-body) МОЖЕТ быть добавлено в запрос только когда метод запроса допускает тело объекта (entity-body).

Включается или не включается тело сообщения (message-body) в сообщение ответа зависит как от метода запроса, так и от кода состояния ответа. Все ответы на запрос с методом HEAD не должны включать тело сообщения (message-body), даже если присутствуют поля заголовка объекта (entity-header), заставляющие поверить в присутствие объекта. Никакие ответы с кодами состояния 1xx (Информационные), 204 (Нет содержимого, No Content), и 304 (Не модифицирован, Not Modified) не должны содержать тела сообщения (message-body). Все другие ответы содержат тело сообщения, даже если оно имеет нулевую длину.

Примеры диалогов HTTP

Обычный GET-запрос

Запрос клиента:

```
GET /wiki/страница HTTP/1.1

Host: ru.wikipedia.org

User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5

Accept: text/html

Connection: close

(пустая строка)
```

Ответ сервера:

```
HTTP/1.1 200 OK

Date: Wed, 11 Feb 2009 11:20:59 GMT

Server: Apache

X-Powered-By: PHP/5.2.4-2ubuntu5wm1

Last-Modified: Wed, 11 Feb 2009 11:20:59 GMT

Content-Language: ru

Content-Type: text/html; charset=utf-8

Content-Length: 1234

Connection: close

(пустая строка)

(далее следует запрошенная страница в HTML)
```

Перенаправления

Сервер посылает код 301 и новый URL в Location.

```
HTTP/1.x 301 Moved Permanently

Location: http://example.com/about.html#contacts
```

В заголовке Location можно указывать фрагменты как в данном примере. Браузер автоматически прокрутит страницу до фрагмента «contacts», как только загрузит её. В тело ответа также был помещён короткий HTML-документ со ссылкой, с помощью которой посетитель попадёт на целевую страницу, если браузер не перейдёт на неё автоматически.

Для перенаправления также используются коды ответа 303 (See Other) и 307 (Temporary Redirect).

Accept-Language

Надо отметить, что сервер принимает во внимание заголовок Accept-Language и может сформировать ответ с временным перенаправлением на соответсвующий сервер.

Cache-Control

Значение «private» сообщает остальным серверам (в первую очередь прокси) что ответ может кэшироваться только на стороне клиента.

Referer

Заголовок Referer с указанием URL указывает, что файл был запрошен с главной страницы сайта. Без него сервер может ответить 403 (Access Forbidden), если не допускаются запросы с других сайтов.

Accept-Ranges

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

Основные механизмы протокола

Частичные GET

HTTP позволяет запросить не сразу всё содержимое ресурса, а только указанный фрагмент. Такие запросы называются частичные GET, возможность их выполнения необязательна (но желательна) для серверов. Частичные GET в основном используются для докачки файлов и быстрого параллельного скачивания в нескольких потоках. Некоторые программы скачивают заголовок архива, выводят пользователю внутреннюю структуру, а потом уже запрашивают фрагменты с указанными элементами архива.

Для получения фрагмента клиент посылает серверу запрос с заголовком Range, указывая в нём необходимые байтовые диапазоны. Если сервер не понимает частичные запросы (игнорирует заголовок Range), то он вернёт всё содержимое со статусом 200, как и при обычном GET. В случае

успешного выполнения сервер возвращает вместо кода 200 ответ со статусом 206 (Partial Content), включая в ответ заголовок Content-Range. Сами фрагменты могут быть переданы двумя способами:

В ответе помещается заголовок Content-Range с указанием байтовых диапазонов. В соответствии с ними фрагменты последовательно помещаются в основное тело. При этом Content-Length должен соответствовать суммарному объёму всего тела.

Сервер указывает медиатип multipart/byteranges для основного содержимого и передаёт фрагменты, указывая соответствующий Content-Range для каждого элемента.

Условные GET

Метод GET изменяется на «условный GET», если сообщение запроса включает в себя поле заголовка «If-Modified-Since». В ответ на условный GET, тело запрашиваемого ресурса передается только, если он изменялся после даты, указанной в заголовке «If-Modified-Since». Алгоритм определения этого включает в себя следующие случаи:

- Если код статуса ответа на запрос будет отличаться от «200 OK», или дата, указанная в поле заголовка «If-Modified-Since» некорректна, ответ будет идентичен ответу на обычный запрос GET.
- Если после указанной даты ресурс изменялся, ответ будет также идентичен ответу на обычный запрос GET.
- Если ресурс не изменялся после указанной даты, сервер вернет код статуса «304 Not Modified».

Использование метода условный GET направлено на разгрузку сети, так как он позволяет не передавать по сети избыточную информацию.

Согласование содержимого

Согласование содержимого (англ. Content Negotiation) — механизм автоматического определения необходимого ресурса при наличии нескольких разнотипных версий документа. Субъектами согласования могут быть не только ресурсы сервера, но и возвращаемые страницы с сообщениями об ошибках (403, 404 и т. п.).

Различают два основных типа согласований:

- Управляемое сервером (англ. Server-Driven).
- Управляемое клиентом (англ. Agent-Driven).

Одновременно могут быть использованы оба типа или каждый из них по отдельности.

В основной спецификации по протоколу (RFC 2616) также выделяется так называемое прозрачное согласование (англ. Transparent Negotiation) как предпочтительный вариант комбинирования обоих типов. В спецификации по HTTP под прозрачностью подразумевается, что процесс не заметен для клиента и сервера.

Управляемое сервером

При наличии нескольких версий ресурса сервер может анализировать заголовки запроса клиента, чтобы выдать, по его мнению, наиболее подходящую. В основном анализируются заголовки Accept, Accept-Charset, Accept-Encoding, Accept-Languages и User-Agent. Серверу желательно включать в ответ заголовок Vary с указанием параметров, по которым различается содержимое по запрашиваемому URI.

Географическое положение клиента можно определить по удалённому IP-адресу. Это возможно за счёт того что IP-адреса, как и доменные имена, регистрируются на конкретного человека или организацию. При регистрации указывается регион, в котором будет использоваться желаемое адресное пространство. Эти данные общедоступны, и в Интернете можно найти соответствующие свободно распространяемые базы данных и готовые программные модули для работы с ними.

Управляемое сервером согласование имеет несколько недостатков:

- Сервер только предполагает, какой вариант наиболее предпочтителен для конечного пользователя, но не может знать точно, что именно нужно в данный момент (например, версия на русском языке или английском).
- Заголовков группы Accept передаётся много, а ресурсов с несколькими вариантами — мало. Из-за этого оборудование испытывает избыточную нагрузку.
- Общему кэшу создаётся ограничение возможности выдавать один и тот же ответ на идентичные запросы от разных пользователей.
- Передача заголовков Accept также может раскрывать некоторые сведения о его предпочтениях, таких как используемые языки, браузер, кодировка.

Управляемое клиентом

В данном случае тип содержимого определяется только на стороне клиента. Для этого сервер возвращает ответ с кодом состояния 300 (Multiple Choices) или 406 (Not Acceptable) список вариантов, среди которых пользователь выбирает подходящий. Управляемое клиентом согласование хорошо, когда содержимое различается по самым частым параметрам (например, по языку и кодировке) и используется публичный кэш.

Основной недостаток — лишняя нагрузка, так как приходится делать дополнительный запрос, чтобы получить нужное содержимое.

Прозрачное согласование

Данное согласование полностью прозрачно для клиента и сервера. В данном случае используется общий кэш, в котором содержится список вариантов, как для управляемого клиентом согласования. Если кэш понимает все эти варианты, то он сам делает выбор, как при управляемом сервером согласовании. Это снижает нагрузки с исходного сервера и исключает дополнительный запрос со стороны клиента. В основной спецификации по протоколу HTTP механизм прозрачного согласования подробно не описан.

Множественное содержимое

Протокол HTTP поддерживает передачу нескольких сущностей в пределах одного сообщения. Причём сущности могут передаваться не только в виде одноуровневой последовательности, но в виде иерархии с вложением элементов друг в друга. Для обозначения множественного содержимого используются медиатипы multipart/*. Работа с такими типами осуществляется по общим правилам, описанным в RFC 2046 (если иное не определено конкретным медиатипом). Если получателю не известно как работать с типом, то он обрабатывает его так же, как multipart/mixed.

Параметр boundary означает разделитель между различными типами передаваемых сообщений. Например передаваемый из формы параметр DestAddress передает значение адреса e-mail, а следующий за ним элемент AttachedFile1 отправляет двоичное содержимое изображения формата .jpg

Со стороны сервера сообщения со множественным содержимым могут посылаться в ответ на частичные GET при запросе нескольких фрагментов ресурса. В этом случае используется медиатип multipart/byteranges.

Со стороны клиента при отправке HTML-формы чаще всего пользуются методом POST. Типичный пример: страницы отправки электронных писем со вложенными файлами. При отправке такого письма браузер формирует сообщение типа multipart/form-data, интегрируя в него как отдельные части, введённые пользователем, тему письма, адрес получателя, сам текст и вложенные файлы.

Особенности протокола

Большинство протоколов предусматривают установление TCP-сессии, в ходе которой один раз происходит авторизация, и дальнейшие действия выполняются в контексте этой авторизации. HTTP же устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP

было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт заголовок «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).

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