МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ РАДИОТЕХНИКИ, ЭЛЕКТРОНИКИ И АВТОМАТИКИ (ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

ЗАЧЕТНАЯ РАБОТА

ПО

ДИСЦИПЛИНЕ

«Информационно-вычислительные сети»

НА ТЕМУ

«Протокол HTTP 1.1»

|  |  |
| --- | --- |
| *Работу выполнил:* | Студент 5-го курса факультета Кибернетики группы ИО-1-97 Фроловичев Сергей |
| *Преподаватель:* | Дешко И.П. |

МОСКВА 2002 г.

**Оглавление**

1. Введение. 5

1.1. Назначение 5

1.3 Терминология. 5

2. Общее описание. 8

3. Параметры протокола. 11

3.1 Версия HTTP. 11

3.2 Универсальный Идентификатор Ресурса (URI). 12

3.2.1 Общий синтаксис. 12

3.2.2 HTTP URL. 13

3.2.3 Сравнение URI. 13

3.3 Форматы даты/времени. 14

3.3.1 Полная дата. 14

3.3.2 Разность секунд (delta seconds). 15

3.4 Кодовые таблицы (character sets). 15

3.5 Кодирования содержимого (content codings). 15

3.6 Кодирования передачи (Transfer Codings). 16

3.7 Медиатипы (Media Types). 17

3.7.1 Канонизация и предопределенные значения типа text. 18

3.7.2 Типы Multipart. 19

3.8 Маркеры продуктов (Product Tokens). 19

3.9 Величины качества (Quality Values). 20

3.10 Метки языков (Language Tags). 20

3.11 Метки объектов (Entity Tags). 21

3.12 Единицы измерения диапазонов (Range Units). 21

4. HTTP сообщение (HTTP Message). 22

4.1 Типы сообщений. 22

4.2 Заголовки сообщений. 22

4.3 Тело cообщения. 23

4.4 Длина сообщения. 24

4.5 Общие поля заголовка. 25

5. Запрос (Request). 25

5.1 Строка запроса (Request-Line). 25

5.1.1 Метод (Method). 25

5.1.2 URI запроса (Request-URI). 26

5.2 Ресурс, идентифицируемый запросом. 27

5.3 Поля заголовка запроса. 28

6 Ответ (Response). 28

6.1 Строка состояния (Status-Line). 28

6.1.1 Код состояния и поясняющая фраза. 28

6.2 Поля заголовка ответа. 30

7 Объект (Entity). 30

7.1 Поля заголовка объекта. 30

7.2 Тело объекта. 31

7.2.1 Тип. 31

7.2.2 Длина. 32

8 Соединения (Connections). 32

8.1 Постоянные соединения (Persistent Connections). 32

8.1.1 Цель. 32

8.1.2 Общее описание. 32

8.1.2.1 Обсуждение (Negotiation). 33

8.1.2.2 Конвейерная обработка (Pipelining). 33

8.1.3 Прокси-сервера (Proxy Servers). 33

8.1.4 Практические соображения. 34

8.2 Требования к передаче сообщений. 34

9 Определения методов. 36

9.1 Безопасные и идемпотентные методы. 36

9.1.1 Безопасные методы. 36

9.1.2 Идемпотентные методы. 37

9.2 OPTIONS. 37

9.3 GET. 38

9.4 HEAD. 38

9.5 POST. 38

9.6 PUT. 39

9.7 DELETE. 40

9.8 TRACE. 41

10 Определение кодов состояния. 41

10.1 1xx - Информационные коды. 41

10.1.1 100 Продолжать, Continue. 41

10.1.2 101 Переключение протоколов, Switching Protocols 42

10.2 2xx - Успешные коды. 42

10.2.1 200 OK. 42

10.2.2 201 Создан, Created. 42

10.2.3 202 Принято, Accepted. 43

10.2.4 203 Не авторская информация, Non-Authoritative Information. 43

10.2.5 204 Нет содержимого, No Content. 43

10.2.6 205 Сбросить содержимое, Reset Content. 43

10.2.7 206 Частичное содержимое, Partial Content. 44

10.3 3xx - Перенаправление. 44

10.3.1 300 Множественный выбор, Multiple Choices. 44

10.3.2 301 Постоянно перемещен, Moved Permanently. 45

10.3.3 302 Временно перемещен, Moved Temporarily. 45

10.3.4 303 Смотреть другой, See Other. 45

10.3.5 304 Не модифицирован, Not Modified. 46

10.3.6 305 Используйте прокси-сервер, Use Proxy. 46

10.4 4xx - Коды ошибок клиента. 47

10.4.1 400 Испорченный Запрос, Bad Request. 47

10.4.2 401 Несанкционированно, Unauthorized. 47

10.4.3 402 Требуется оплата, Payment Required. 47

10.4.4 403 Запрещено, Forbidden. 47

10.4.5 404 Не найден, Not Found. 48

10.4.6 405 Метод не допустим, Method Not Allowed. 48

10.4.7 406 Не приемлем, Not Acceptable. 48

10.4.8 407 Требуется установление подлинности через прокси-сервер, Proxy Authentication Required. 48

10.4.9 408 Истекло время ожидания запроса, Request Timeout. 49

10.4.10 409 Конфликт, Conflict. 49

10.4.11 410 Удален, Gone. 49

10.4.12 411 Требуется длина, Length Required. 50

10.4.13 412 Предусловие неверно, Precondition Failed. 50

10.4.14 413 Объект запроса слишком большой, Request Entity Too Large. 50

10.4.15 414 URI запроса слишком длинный, Request-URI Too Long. 50

10.4.16 415 Неподдерживаемый медиатип, Unsupported Media Type. 50

10.5 5xx - Коды ошибок сервера. 51

10.5.1 500 Внутренняя ошибка сервера, Internal Server Error. 51

10.5.2 501 Не реализовано, Not Implemented. 51

10.5.3 502 Ошибка шлюза, Bad Gateway. 51

10.5.4 503 Сервис недоступен, Service Unavailable. 51

10.5.5 504 Истекло время ожидания от шлюза, Gateway Timeout. 51

10.5.6 505 Не поддерживаемая версия HTTP, HTTP Version Not Supported. 52

11 Установление подлинности доступа (Access Authentication). 52

11.1 Базовая схема установления подлинности (Basic Authentication Scheme). 53

11.2 Обзорная схема установления подлинности (Digest Authentication Scheme) [1]. 54

13 Кэширование в HTTP. 54

13.1 Общая информация о кэшировании. 55

13.1.1 Правильность кэша. 55

13.1.2 Предупреждения. 56

13.1.3 Механизмы управления кэшем (Cache-control Mechanisms). 57

13.1.4 Явные предупреждения агента пользователя. 57

13.1.5 Исключения из правил и предупреждений. 57

13.1.6 Контролируемое клиентом поведение. 58

13.2 Модель устаревания. 58

13.2.1 Устаревание, указанное сервером. 58

13.2.2 Эвристическое устаревание. 59

13.2.3 Вычисление возраста. 59

13.2.4 Вычисление устаревания. 61

13.2.5 Устранение противоречий в значениях устаревания. 62

13.2.6 Устранение противоречий между несколькими ответами. 62

13.3 Модель проверки достоверности (validation model). 63

Библиографический список 65

# 1. Введение.

Протокол передачи гипертекста (HTTP) - протокол прикладного уровня для распределенных, совместных, многосредных информационных систем. HTTP используется в World Wide Web (WWW) начиная с 1990 года. Первой версией HTTP, известной как HTTP/0.9, был простой протокол для передачи необработанных данных через Интернет. По определению RFC 1945 HTTP/1.0 был улучшением этого протокола, допускал MIME-подобный формат сообщений, содержащий метаинформацию о передаваемых данных и имел модифицированную семантику запросов/ответов. Однако HTTP/1.0 недостаточно учитывал особенности работы с иерархическими прокси-серверами (hierarchical proxies), кэшированием, постоянными соединениями, и виртуальными хостами (virtual hosts). Кроме того, быстрый рост числа не полностью совместимых с протоколом HTTP/1.0 приложений, потребовал введения новой версии протокола, в которой были бы заложены дополнительные возможности, которые помогли бы привести эти приложения к единому стандарту.

Список RFC относящийся к рассмотренным в данной работе вопросам, приведен в разделе «Библиографический список».

## 1.1. Назначение

Протокол HTTP/1.1 содержит более строгие требования, чем HTTP/1.0, гарантирующие более надежную работу.

Большие информационные системы требуют большего количества функциональных возможностей, чем просто загрузку информации, включая поиск и модификацию данных при помощи внешних интерфейсов. HTTP предоставляет открытый (open-ended) набор методов, которые основаны на системе ссылок, которые обеспечиваются URI (Универсальными Идентификаторами Ресурсов). URI могут идентифицировать как расположение (URL), так и имя (URN) ресурса, к которому применяется данный метод. Сообщения передаются в формате, подобному используемому электронной почтой согласно определениям MIME (Многоцелевых Расширений Электронной Почты).

HTTP также используется как обобщенный протокол связи между агентами пользователей (user agents) и прокси-серверами/шлюзами (proxies/gateways) или другими Интернет-сервисами, включая такие как SMTP, NNTP, FTP, Gopher и WAIS. Таким образом, HTTP определяет основы многосредного доступа к ресурсам для разнообразных приложений.

## 1.3 Терминология.

* **Соединение (connection).**

Виртуальный канал транспортого уровня, установленный между двумя программами с целью связи.

* **Сообщение (message).**

Основной модуль HTTP связи, состоящей из структурной последовательности октетов, соответствующих синтаксису протокола и передаваемых по соединению.

* **Запрос (request)**

Любое HTTP сообщение, содержащее запрос.

* **Ответ (response).**

Любое HTTP сообщение, содержащее ответ.

* **Ресурс (resource).**

Сетевой объект данных или сервис, который может быть идентифицирован URI. Ресурсы могут быть доступны в нескольких представлениях (например на нескольких языках, в разных форматах данных, иметь различный размер или различную разрешающую способность) или различаться по другим параметрам.

* **Объект (entity).**

Информация, передаваемая в качестве полезной нагрузки запроса или ответа. Объект состоит из метаинформации в форме полей заголовка объекта и содержания в форме тела объекта.

* **Представление (representation)**.

Объект включенный в ответ, и подчиняющийся обсуждению содержимого (Content Negotiation). Может существовать несколько представлений, связанных со специфическими состояниями ответа.

* **Обсуждение содержимого (content negotiation)**.

Механизм для выбора соответствующего представления во время обслуживания запроса. Представление объектов в любом ответе может быть обсуждено (включая ошибочные ответы).

* **Вариант (variant)**.

Ресурс может иметь одно, или несколько представлений, связанных с ним в данный момент. Каждое из этих представлений называется "вариант". Использование термина "вариант" не обязательно подразумевает, что ресурс подчинен обсуждению содержимого.

* **Клиент (client)**

Программа, которая устанавливает соединения с целью посылки запросов.

* **Агент пользователя (user agent)**.

Клиент, который инициирует запрос. Как правило браузеры, редакторы, роботы (spiders), или другие инструментальные средства пользователя.

* **Сервер (server)**.

Приложение, которое слушает соединения, принимает запросы на обслуживание и посылает ответы. Любая такая программа способна быть как клиентом, так и сервером; наше использование данного термина относится скорее к роли, которую программа выполняет, создавая специфические соединения, нежели к возможностям программы вообще. Аналогично, любой сервер может действовать как первоначальный сервер (origin server), прокси-сервер (proxy), шлюз (gateway) или туннель (tunnel), изменяя поведение, основываясь на характере каждого запроса.

* **Первоначальный сервер (origin server)**.

Сервер, на котором данный ресурс находится постоянно или должен быть создан.

* **Прокси-сервер (proxy)**.

Программа-посредник, которая действует и как сервер, и как клиент с целью создания запросов от имени других клиентов. Запросы обслуживаются прокси-сервером, или пересылаются им, возможно с изменениями. Прокси-сервер, согласно этой спецификации, должен удовлетворять требованиям клиента и сервера.

* **Шлюз (gateway)**.

Сервер, который действует как посредник для некоторого другого сервера. В отличие от прокси-сервера, шлюз получает запросы в качестве первоначального сервера для запрошенного ресурса; клиент запроса может не знать, что он соединяется со шлюзом.

* **Туннель (tunnel)**.

Программа-посредник, которая поддерживает соединение. Один раз созданный, туннель не рассматривается как часть HTTP связи, хотя туннель, возможно, был инициализирован запросом HTTP. Туннель прекращает существовать, когда оба конца соединения закрываются.

* **Кэш (cache)**.

Локальная память, в которой программа хранит сообщения-ответы, и в которой располагается подсистема, управляющая хранением, поиском и удалением сообщений. Кэш сохраняет ответы, которые могут быть сохранены, чтобы уменьшить время ответа и загрузку сети (траффик) при будущих эквивалентных запросах. Любой клиент или сервер может иметь кэш, но кэш не может использоваться сервером, который действует как туннель.

* **Кэшируемый (cachable)**.

Ответ является кэшируемым, если кэшу разрешено сохранить копию ответного сообщения для использования при ответе на последующие запросы. Даже если ресурс кэшируем, могут существовать дополнительные ограничения на использование кэшем сохраненной копии для исходного запроса.

* **Непосредственный (first-hand)**.

Ответ считается непосредственным, если он приходит непосредственно от первоначального сервера без ненужной задержки, возможно через один или несколько прокси-серверов. Ответ также является непосредственным, если его достоверность только что была установлена непосредственно первоначальным сервером.

* **Точное время устаревания (explicit expiration time)**.

Время определенное первоначальным сервером и показывающее кэшу когда объект больше не может быть возвращен клиенту без дополнительной проверки достоверности.

* **Эвристическое время устаревания (heuristic expiration time)**.

Время устаревания, назначенное кэшем, если не указано точное время устаревания.

* **Возраст (age)**.

Возраст ответа - время, прошедшее с момента отсылки, или успешной проверки ответа первоначальным сервером.

* **Время жизни (freshness lifetime)**.

Отрезок времени между порождением ответа и моментом устаревания.

* **Свежий (fresh)**.

Ответ считается свежим, если его возраст еще не превысил время жизни.

* **Просроченнный (stale)**.

Ответ считается просроченным, если его возраст превысил время жизни.

* **Семантически прозрачный (semantically transparent)**.

Говорят, что кэш ведет себя "семантически прозрачным" образом в отношении специфического ответа, когда использование кэша не влияет ни на клиента запроса, ни на первоначальный сервер, но повышает эффективность. Когда кэш семантически прозрачен, клиент получает точно такой же ответ (за исключением промежуточных (hop-by-hop) заголовков), который получил бы, запрашивая непосредственно первоначальный сервер, а не кэш.

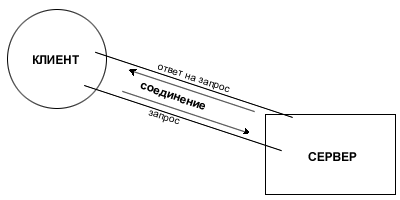
* **Указатель достоверности (validator)**.

Элемент протокола (например, метка объекта или время последней модификации (Last-Modified time)), который используется, чтобы выяснить, является ли находящаяся в кэше копия эквивалентом объекта.

# 2. Общее описание.

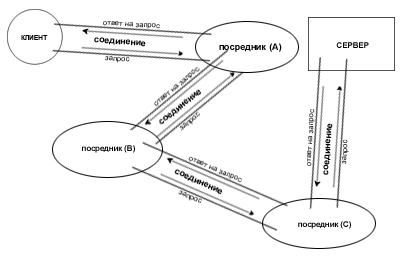
Протокол HTTP - это протокол запросов/ответов. Клиент посылает по соединению запрос серверу, содержащий: метод запроса, URI, версию протокола, MIME-подобное сообщение, включающее модификаторы запроса, клиентскую информацию и, возможно, тело запроса. Сервер отвечает строкой состояния, включающей версию протокола сообщения, кодом успешного выполнения или ошибки, MIME-подобным сообщением, содержащим информацию о сервере, метаинформацию объекта и, возможно, тело объекта.

Большинство HTTP соединений, инициализируется агентом пользователя и состоит из запроса, который нужно применить к ресурсу на некотором первоначальном сервере. В самом простом случае, он может быть выполнен посредством одиночного соединения между агентом пользователя и первоначальным сервером.



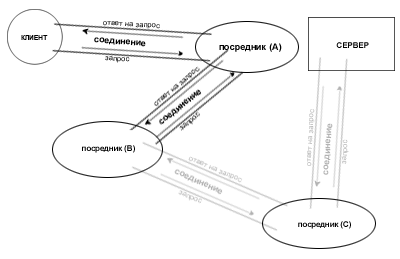
Более сложная ситуация возникает, когда в цепочке запросов/ответов присутствует один или несколько посредников. Существуют три основных разновидности посредников: прокси-сервера, шлюзы, и туннели. Прокси-сервер является агентом-посредником, который получает запросы на некоторый URI в абсолютной форме, изменяет все сообщение или его часть и отсылает измененный запрос серверу, идентифицированному URI. Шлюз - это принимающий агент, действующий как бы на уровень выше некоторого другого сервера(ов) и при необходимости транслирующий запросы в протокол основного сервера. Туннель действует как реле (relay) между двумя соединениями не изменяя сообщений; туннели используются, когда связь нужно производить через посредника (например firewall), который не понимает содержание сообщений.

На рисунке показаны три посредника (A, B и C) между агентом пользователя и первоначальным сервером. Запросы и ответы передаются через четыре отдельных соединения. Это отличие важно, так как некоторые опции HTTP соединения применимы только к соединению с ближайшим не туннельным соседом, некоторые только к конечным точкам цепочки, а некоторые ко всем соединениям в цепочке. Хотя эта диаграмма линейна, каждый участник может быть задействован в нескольких соединениях одновременно. Например, B может получать запросы от других клиентов, а не только от A, и/или пересылать запросы серверам, отличным от C, в то же время, когда он обрабатывает запрос А.



Любая сторона соединения, которая действует не как туннель, может использовать внутренний кэш для обработки запросов. Эффект кэша заключается в том, что цепочка запросов/ответов сокращается, если один из участников в цепочке имеет кэшированный ответ, удовлетворяющий данному запросу. Далее показана цепочка, возникающая в том случае, когда B имеет кэшированую копию раннего ответа O (полеченного через C) на запрос, и который не был кэширован ни UA, ни A.

Не все ответы полезно кэшировать, а некоторые запросы могут содержать модификаторы, которые указывают специальные требования, управляющие поведением кэша.



Фактически, имеется широкое разнообразие архитектур и конфигураций кэшей и прокси-серверов, разрабатываемых в настоящее время или развернутых в World Wide Web; эти системы включают национальные иерархии прокси-кэшей, которые сохраняют пропускную способность межокеанских каналов, системы, которые распространяют по многим адресам содержимое кэша, организации, которые распространяют подмножества кэшируемых данных на CD-ROM, и так далее. HTTP системы используются в корпоративных интранет-сетях с высокоскоростными линиями связи, и для доступа через PDA с маломощными радиолиниями и неустойчивой связью. Цель HTTP/1.1 состоит в поддержании широкого многообразия конфигураций, уже построенных при введении ранних версий протокола, а также в удовлетворении потребностей разработчиков web приложений, требующих все более высокой надежности.

HTTP соединение обычно происходит посредством TCP/IP соединений. Заданный по умолчанию порт TCP - 80, но могут использоваться и другие порты (например: 8080, 8081). HTTP также может быть реализован посредством любого другого протокола Интернет, или других сетей. HTTP необходима только надежная передача данных, следовательно может использоваться любой протокол, который гарантирует надежную передачу данных; отображение структуры запроса и ответа HTTP/1.1 на транспортные модули данных рассматриваемого протокола - вопрос, не решается на уровне самого протокола.

Большинство реализаций HTTP/1.0 использовало новое соединение для каждого обмена запросом/ответом. В HTTP/1.1, установленное соединение может использоваться для одного или нескольких обменов запросом/ответом, хотя соединение может быть закрыто по ряду причин.

# 3. Параметры протокола.

## 3.1 Версия HTTP.

HTTP использует схему нумерации типа "<major>.<minor>", для указания версии протокола. Стратегия версификации протокола предназначена для того, чтобы позволить отправителю указать формат сообщения и свои способности понимания для дальнейшей HTTP связи, прежде чем он получит что-либо посредством этой связи. При добавлении компонентов сообщения, которые не воздействуют на процесс связи, или компонентов, которые добавляются только к расширяемым значениям поля, номер версии не меняется. Когда внесенные в протокол изменения добавляют возможности, которые не изменяют общий алгоритм анализа сообщений, но расширяют семантику сообщения и подразумевают дополнительные возможности отправителя, увеличивается <minor> номер. Когда изменяется формат сообщения протокола увеличивается <major> номер.

Версия HTTP сообщения обозначается полем HTTP-version в первой строке сообщения.

HTTP-Version = "HTTP" "/" 1\*DIGIT "." 1\*DIGIT

Major и minor числа должны обрабатываться как отдельные целые числа и что каждое может состоять более чем из одной цифры. Таким образом, HTTP/2.4 - более низкая версия, чем HTTP/2.13, которая в свою очередь ниже чем HTTP/12.3. Нули должны игнорироваться получателями и не должны посылаться.

Приложения, посылающие сообщения запросов или ответов, которые описывает спецификация HTTP/1.1, должны указывать версию HTTP (HTTP-version) "HTTP/1.1". Использование этого номера версии указывает, что посылающее приложение по крайней мере условно совместимо с этой спецификацией.

HTTP версия приложения - это самая высокая HTTP версия, с которой приложение является по крайней мере условно совместимым ним.

Приложения, реализующие прокси-сервера и шлюзы, должны обрабатывать протокольные сообщения различных версий. Начиная с момента, когда версия протокола указывает возможности отправителя, прокси-сервер/шлюз никогда не должен посылать сообщения, версия которых больше, чем HTTP версия отправителя; если получена более высокая версия запроса, то прокси-сервер/шлюз должен или понизить версию запроса, вернув сообщение об ошибке, или переключиться на туннельное поведение. У запросов, версия которых ниже, чем HTTP версия прокси-сервера/шлюза можно перед пересылкой увеличить версию; ответ прокси-сервера/шлюза на этот запрос должен иметь ту же самую major версию, что и запрос.

Преобразование версий HTTP может включать модификацию полей заголовка, требуемых или запрещенных этими версиями.

## 3.2 Универсальный Идентификатор Ресурса (URI).

URI известны под многими именами: WWW адреса, Универсальные Идентификаторы Документов, Универсальные Идентификаторы Ресурсов (URI), и, в заключение, как комбинация Единообразных Идентификаторов Ресурсов (Uniform Resource Locators, URL) и Единообразных Имен Ресурсов (Uniform Resource Names, URN). HTTP определяет URL просто как строку определенного формата, которая идентифицирует ресурс посредством имени, расположения, или любой другой характеристики.

### 3.2.1 Общий синтаксис.

URI в HTTP могут представляться в абсолютной форме (absolute URI) или относительно некоторого известного основного URI (relative URI), в зависимости от контекста их использования. Эти две формы различаются тем, что абсолютные URI всегда начинаются с имени схемы с двоеточием.

URI = ( absoluteURI | relativeURI ) [ "#" fragment ]

absoluteURI = scheme ":" \*( uchar | reserved )

relativeURI = net\_path | abs\_path | rel\_path

net\_path = "//" net\_loc [ abs\_path ] abs\_path = "/" rel\_path rel\_path = [ path ] [ ";" params ] [ "?" query ]

path = fsegment \*( "/" segment ) fsegment = 1\*pchar segment = \*pchar

params = param \*( ";" param ) param = \*( pchar | "/" )

scheme = 1\*( ALPHA | DIGIT | "+" | "-" | "." ) net\_loc = \*( pchar | ";" | "?" )

query = \*( uchar | reserved ) fragment = \*( uchar | reserved )

pchar = uchar | ":" | "@" | "&" | "=" | "+" uchar = unreserved | escape unreserved = ALPHA | DIGIT | safe | extra | national

escape = "%" HEX HEX reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" extra = "!" | "\*" | "'" | "(" | ")" | "," safe = "$" | "-" | "\_" | "." unsafe = CTL | SP | <"> | "#" | "%" | "<" | ">" national = <любой OCTET за исключением ALPHA, DIGIT, reserved, extra, safe, и unsafe октетов>

Полная информация о синтаксисе и семантике URL содержится в RFC 1738 и RFC 1808. Нормальная запись Бекуса-Наура включает национальные символы, недозволенные в правильных URL, определеных RFC 1738, так как HTTP серверы позволяют использовать для представления части rel\_path адресов набор не зарезервированных символов, и, следовательно, HTTP прокси-сервера могут получать запросы URI, не удовлетворяющие RFC 1738.

Протокол HTTP не накладывает никаких ограничений на длины URI. Серверы должны обрабатывать URI любого ресурса, любой длинны, который они обслуживают, и им надлежит обрабатывать URI неограниченной длины, если они обслуживают сервера, основанные на методе GET, которые могут создавать такой URI. Серверу следует возвращать код состояния 414 (URI запроса слишком длинный, Request-URI Too Long), если URI длиннее, чем сервер в состоянии обработать.

Серверы должны обращать внимание на URI, которые имеют длину более 255 байтов, потому что некоторые старые клиенты или прокси-сервера могут неправильно поддерживать эти длины.

### 3.2.2 HTTP URL.

"Http" схема используется для доступа к сетевым ресурсам при помощи протокола HTTP. Этот раздел определяет схемо-определенный синтаксис и семантику для HTTP URL.

http\_URL = "http:" "//" host [ ":" port ] [ abs\_path ]

host = <допустимое доменное имя машины или IP адрес (в точечно десятичной форме), как определено в разделе 2.1 RFC 1123>

port = \*DIGIT

Если порт пуст или не задан - используется порт 80. Это означает, что идентифицированный ресурс размещен в сервере, ожидающем TCP соединений на специфицированном порте port, компьютера host, и запрашиваемый URI ресурса - abs\_path. Использования IP адресов в URL следует избегать, насколько это возможно (RFC 1900). Если abs\_path не представлен в URL, он должен рассматриваться как "/" при вычислении запрашиваемого URI (Request-URI) ресурса.

### 3.2.3 Сравнение URI.

При сравнении двух URI, чтобы решить соответствуют ли они друг другу или нет, клиенту следует использовать чувствительное к регистру пооктетное (octet-by-octet) сравнение этих URI, со следующими исключениями:

* Порт, который пуст или не указан, эквивалентен заданному по умолчанию порту для этого URI;
* Сравнение имен хостов должно производиться без учета регистра;
* Сравнение имен схем должно производиться без учета регистра;
* Пустой abs\_path эквивалентен "/".
* Символы, отличные от тех, что находятся в "зарезервированных" ("reserved") и "опасных" ("unsafe") наборах эквивалентны их представлению как ""%" HEX HEX ".

Например следующие три URI эквивалентны:

http://abc.com:80/~smith/home.html

http://ABC.com/%7Esmith/home.html h ttp://ABC.com:/%7esmith/home.html

## 3.3 Форматы даты/времени.

### 3.3.1 Полная дата.

HTTP приложения исторически допускали три различных формата для представления даты/времени:

Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, дополненный в ; RFC 1123

Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, переписанный как ; RFC 1036

Sun Nov 6 08:49:37 1994 ; формат asctime() ANSI C

Первый формат выбран в качестве стандарта Интернета и представляет подмножество фиксированной длины, как определено в RFC 1123 (модифицированном RFC 822). Второй формат находится в общем пользовании, но основан на устаревшем и потерявшем статус стандарта RFC 850, описывающем форматы дат, он обладает тем недостатком, что год указывается не в четырехразрядной нотации. Клиенты и серверы HTTP/1.1, которые анализируют значение даты, должны понимать все три формата (для совместимости с HTTP/1.0), но генерировать для представления значений дат в полях заголовка HTTP должны только формат RFC 1123 .

Прис оздании приложений, желательно, чтобы оно умело воспринимать значения дат, которые, возможно, посланы не HTTP приложениями, а например SMTP или NNTP сообщений через прокси-сервера/шлюзы.

Все без исключений форматы даты/времени в HTTP должны быть представлены в Greenwich Mean Time (GMT). В первых двух форматах данный факт указывается включением трехсимвольного сокращения "GMT" в качестве часового пояса. В asctime() формате это ДОЛЖНО подразумеваться при чтении.

HTTP-date = rfc1123-date | rfc850-date | asctime-date

rfc1123-date = wkday "," SP date1 SP time SP "GMT" rfc850-date = weekday "," SP date2 SP time SP "GMT" asctime-date = wkday SP date3 SP time SP 4DIGIT

date1 = 2DIGIT SP month SP 4DIGIT ; день месяц год (например 02 Jun 1982)

date2 = 2DIGIT "-" month "-" 2DIGIT ; день-месяц-год (например 02-Jun-82)

date3 = month SP ( 2DIGIT | ( SP 1DIGIT )) ; месяц день (например, Jun 2)

time = 2DIGIT ":" 2DIGIT ":" 2DIGIT ; 00:00:00 - 23:59:59

wkday = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"

weekday = "Monday" | "Tuesday" | "Wednesday" | "Thursday" | "Friday" | "Saturday" | "Sunday"

month = "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"

Это требования формата даты/времени, которые применяются внутри потока протокола HTTP. Клиентам и серверам не требуется использовать эти форматы для представления пользователю, регистрации запросов и т.д.

### 3.3.2 Разность секунд (delta seconds).

Некоторые поля HTTP заголовка позволяют указывать значения времени в виде целого числа секунд, представленного в десятичной форме, которые должны пройти с того момента, как сообщение было получено.

delta-seconds = 1\*DIGIT

## 3.4 Кодовые таблицы (character sets).

HTTP использует то же самое определение термина "кодовая таблица", которое определено для MIME:

Термин "кодовая таблица" используется, чтобы сослаться на метод, использующий одну или несколько таблиц для преобразования последовательности октетов в последовательность символов. Стоит отметить, что однозначное преобразование в обратном направлении не требуется, и что не все символы могут быть доступны в данной кодовой таблице, и что кодовая таблица может обеспечивать более чем одну последовательность октетов для представления специфических символов. Это определение допускает различные виды кодирования символов, от простых однотабличных отображений типа US-ASCII до сложных методов, переключающих таблицы, наподобие тех, которые используют методики ISO 2022. Однако определение, связанное с именем кодовой таблицы MIME должно полностью определять отображение, которое преобразует октеты в символы. В частности использование внешней информации профилирования для определения точного отображения не разрешается.

Кодовые таблицы HTTP идентифицируются лексемами, не чувствительными к регистру. Полный набор лексем определен реестром кодовых таблиц IANA [19].

charset = token

Хотя HTTP позволяет использовать в качестве значения charset произвольную лексему, любая лексема, которая имеет предопределенное значение в реестре кодовых таблиц IANA, должна представлять набор символов, определенный в данном реестре. Приложениям следует ограничить использование символьных наборов теми, которые определены в реестре IANA.

## 3.5 Кодирования содержимого (content codings).

Значение кодирования содержимого указывает какое преобразование кодирования было или будет применено к объекту. Кодирование содержимого используется прежде всего для сжатия или другого полезного преобразования документа без потери идентификации основного медиатипа и информации. Часто, объект сохраняется в кодированной форме, затем передается, а потом декодируется получателем.

content-coding = token

Все значения кодирования содержимого (content-coding) не чувствительны к регистру. HTTP/1.1 использует значения кодирования содержимого (content-coding) в полях заголовка Accept-Encoding и Content-Encoding. Хотя значение описывает кодирование содержимого, но, что более важно - оно указывает, какой механизм декодирования потребуется для обратного процесса.

Internet Assigned Numbers Authority (IANA) действует как реестр для значений лексем кодирования содержимого (content-coding). Первоначально реестр содержал следующие лексемы:

**gzip** Формат кодирования, производящий сжатие файла программой "gzip" (GNU zip), описанный в RFC 1952. Это формат Lempel-Ziv кодирования (LZ77) с 32 разрядным CRC.

**compress** Формат кодирования, производимый общей программой "compress" для сжатия UNIX файлов. Это формат адаптивного Lempel-Ziv-Welch кодирования (LZW).

Конечно, использовать названия программ для идентификации форматов кодирования не желательно и может пересекаться с форматами, которые возникнут в последствии. Их использование объясняется исторической практикой. Для совместимости с предыдущими реализациями HTTP, приложения должны рассматривать "x-gzip" и "x-compress" как эквиваленты "gzip" и "compress" соответственно.

**deflate** Формат zlib, определенный в 1950, в комбинации с механизмом сжатия "deflate", описанным в RFC 1951.

Новая лексема значения кодирования содержимого (content-coding) должна быть зарегистрирована; чтобы обеспечить взаимодействие между клиентами и серверами, спецификация алгоритма кодирования содержимого, необходимого для определения нового значения, должна быть открыто опубликована и адекватна для независимой реализации, а также соответствовать цели кодирования содержимого определенного в этом разделе.

## 3.6 Кодирования передачи (Transfer Codings).

Значения кодирования передачи используются для указания преобразования кодирования, которое было или должно быть применено к телу объекта (entity-body) в целях гарантирования "безопасной передачи" по сети. Оно отличается от кодирования содержимого тем, что кодирование передачи - это свойство сообщения, а не первоначального объекта.

transfer-coding = "chunked" | transfer-extension

transfer-extension = token

Все значения кодирования передачи (transfer-coding) не чувствительны к регистру. HTTP/1.1 использует значения кодирования передачи (transfer-coding) в поле заголовка Transfer-Encoding.

Кодирования передачи - это аналоги значений Content-Transfer-Encoding MIME, которые были разработаны для обеспечения безопасной передачи двоичных данных при использовании 7-битного обслуживания передачи. Однако безопасный транспорт имеет другое предназначение для чисто 8-битного протокола передачи. В HTTP единственная опасная характеристика тела сообщения вызвана сложностью определения точной длины тела сообщения, или желанием шифровать данные при пользовании общедоступным транспортом.

Кодирование по кускам (chunked encoding) изменяет тело сообщения для передачи его последовательностью кусков, каждый из которых имеет собственный индикатор размера, сопровождаемым опциональным завершителем, содержащим поля заголовка объекта. Это позволяет динамически создаваемому содержимому передаваться вместе с информацией, необходимой получателю для проверки полноты получения сообщения.

Chunked-Body = \*chunk "0" CRLF footer CRLF

chunk = chunk-size [ chunk-ext ] CRLF chunk-data CRLF

hex-no-zero = <HEX за исключением "0">

chunk-size = hex-no-zero \*HEX chunk-ext = \*( ";" chunk-ext-name [ "=" chunk-ext-value ]) chunk-ext-name = token chunk-ext-val = token | quoted-string chunk-data = chunk-size(OCTET)

footer = \*entity-header

Кодирование по кускам (chunked encoding) оканчивается куском нулевого размера, следующим за завершителем, оканчивающимся пустой строкой. Цель завершителя состоит в эффективном методе обеспечения информации об объекте, который сгенерирован динамически; приложения не должны посылать в завершителе поля заголовка, которые явно не предназначены для использования в завершителе, такие как Content-MD5 или будущие расширения HTTP для цифровых подписей и других возможностей.

Все HTTP/1.1 приложения должны быть в состоянии получать и декодировать кодирование передачи "по кускам" ("chunked" transfer coding), и должны игнорировать расширения кодирования передачи, которые они не понимают. Серверу, который получил тело объекта со значением кодирования передачи, которое он не понимает, следует возвратить ответ с кодом 501 (Не реализовано, Not Implemented) и разорвать соединение. Сервер не должен посылать поля кодирования передачи (transfer-coding) HTTP/1.0 клиентам.

## 3.7 Медиатипы (Media Types).

HTTP использует МедиаТипы Интернета (Internet Media Types) в полях заголовка Content-Type и Accept для обеспечения открытой и расширяемой типизации данных и типов.

media-type = type "/" subtype \*( ";" parameter ) type = token subtype = token

Параметры могут следовать за type/subtype в форме пар атрибут/значение (attribute/value).

parameter = attribute "=" value attribute = token value = token | quoted-string

Тип, подтип, и имена атрибутов и параметров не чувствительны к регистру. Значения параметров могут быть чувствительными к регистру, но могут быть и не чувствительны, в зависимости от семантики имени параметра. Линейный пробел (LWS) не должен использоваться между типом и подтипом, между атрибутом и значением. Агенты пользователей, распознающие медиатипы, должны обрабатывать (или подготавливать для обработки любыми внешними приложениями) параметры для тех типов MIME, которые описаны, и сообщать пользователю об обнаруженных проблемах.

Некоторые старые HTTP приложения не распознают параметры медиатипов. При посылке данных к таким HTTP приложениям реализации должны использовать параметры медиатипов только тогда, когда это требуется по определению типа/подтипа.

Значения медиатипов регистрируются Internet Assigned Number Authority (IANA). Процесс регистрации медиатипа определен в RFC 2048. Использование не зарегистрированных медиатипов запрещено.

### 3.7.1 Канонизация и предопределенные значения типа text.

Медиатипы Интернет зарегистрированы в канонической форме. В общем случае тело объекта, передаваемое HTTP сообщением, должно быть представлено в соответствующей каноническиой форме до передачи; исключение составляют типы "text", определяемые в следующем абзаце.

В канонической форме медиаподтипы типа "text" используют CRLF в качестве метки конца строки. HTTP ослабляет это требование и позволяет передавать текст размеченный таким образом, что еденичные CR или LF могут быть метками конца строки, правда это правило должно быть выполнено для всего тела объекта (entity-body). HTTP приложения должны воспринимать CRLF, просто CR и просто LF как представление конца строки в текстовых типах, переданных по HTTP. Кроме того, если текст представляется в кодовой таблице, которая не использует октеты 13 и 10 для CR и LF соответственно, что имеет место в некоторых многобайтовых кодовых таблицах, то HTTP позволяет использовать любые последовательности октетов, определенные этим набором символов для представления эквивалентов CR и LF в качестве кода конца строки. Эта гибкость в отношении концов строк применима только к текстовым типам в теле объекта; просто CR или просто LF не должны заменять CRLF внутри любой управляющей структуры HTTP (например поля заголовка и разделителей типа multipart).

Если тело объекта кодируется при помощи Content-Encoding, то основные данные должны быть в определенной выше форме до кодирования.

Параметр "charset" используется с некоторыми медиатипами для указания кодовой таблицы, используемой для представления данных. Если параметр "charset" не указан отправителем, то при получении по HTTP медиаподтипы типа "text" имеют значение "charset", по умолчанию равное "ISO-8859-1". Данные в кодовых таблицах или их подмножествах, отличных от "ISO-8859-1", должны быть помечены соответствующим значением "charset".

Некоторое программное обеспечение HTTP/1.0 неправильно интерпретировало заголовок Content-Type без параметра "charset", как означающее "должен предположить получатель". Отправители, желающие предусмотреть такое поведение могут включать параметр "charset" даже когда charset равен ISO-8859-1 и должны сделать это, если известно, что это не запутает получателя.

К сожалению, некоторые старые HTTP/1.0 клиенты не работали правильно с определением параметра "charset". HTTP/1.1 получатели должны отдавать приоритет метке "charset", поставленной отправителем; и те агенты пользователей, которые имеют возможность "предположить" charset должны при первоначальном отображении документа использовать charset из поля content-type, если они поддерживают такой charset, а затем использовать собственные установки.

### 3.7.2 Типы Multipart.

MIME предусматривает ряд типов "multipart" - формирующих пакет из одного или нескольких объектов внутри тела одного сообщения. Все типы mulptipart используют общий синтаксис, определеный в MIME, и должны содержать разделительный параметр частью значения медиатипа. Тело сообщения - самостоятельный элемент протокола и, следовательно, должно использовать только СRLF для представления концов строк между частями тела (body-parts). В отличие от MIME, заключение (epilogue) любого multipart сообщения должно быть пустым; HTTP приложения не должны передавать заключение (даже если первоначальный multipart содержит заключение).

В HTTP части тела (body-parts) типа multipart могут содержать поля заголовка, которые являются значащими в примнении к этой части. Поле заголовка Content-Location следует включать в часть тела (body-part) каждого включенного объекта, который может быть идентифицирован URL.

Вообще говоря, HTTP агенту пользователя належит действовать так же как поступил бы MIME агент пользователя после получения типа multipart. Если приложение получает незарегистрированный подтип multipart, оно должно обрабатывать его как подтип "multipart/mixed".

Тип "multipart/form-data" был специально определен для передачи данных формы, подходящих для обработки методом запроса POST, что описано в RFC 1867.

## 3.8 Маркеры продуктов (Product Tokens).

Маркеры продуктов используются, чтобы обеспечить коммуникационным приложениям возможность идентифицировать себя названием и версией программного обеспечения. Большинство полей, использующих маркеры продуктов также допускает перечисление подпрограмм, которые формируют значительную часть приложения, и которые перечисляются через пробел. В соответствии с соглашением, подпрограммы перечисляются в порядке их значения для идентификации приложения.

product = token ["/" product-version] product-version = token

Примеры:

User-Agent: CERN-LineMode/2.15 libwww/2.17b3 Server: Apache/0.8.4

Маркеры продуктов должны быть короткими и по сути - использование их для рекламы или другой несущественной информации однозначно запрещено. Хотя в лексеме product-version может встречаться любой символ, все же ее следует использовать только для идентификатора версии (то есть, последовательным версиям одной и той же программы надлежит иметь отличия только в части product-version лексемы product.

## 3.9 Величины качества (Quality Values).

HTTP использует короткие числа "с плавающей точкой" для указания относительной важности ("веса") различных оговоренных параметров. Вес - это нормализованое вещественное число в диапазоне от 0 до 1, где 0 - минимальное, а 1 - максимальное значение. HTTP/1.1 приложения не должны генерировать более трех цифр после десятичной точки. Пользовательским конфигурациям этих значений следует также ограничиваться этим режимом.

qvalue = ( "0" [ "." 0\*3DIGIT ] ) | ( "1" [ "." 0\*3("0") ] )

"Величины качества" - не корректное название, так как эти значения просто представляют отношение снижения производительности к желательному качеству.

## 3.10 Метки языков (Language Tags).

Метка языка идентифицирует естественный язык: разговорный, письменный, или другой используемый людьми для обмена информацмей с другими людьми. Машинные языки являются исключением. HTTP использует метки языков внутри полей Accept-Language и Content-Language.

Синтаксис и запись меток языка в HTTP те же, что определены в RFC 1766. То есть, метка языка состоит из одной или нескольких частей: метка первичного языка и, возможно пустой, ряд подчиненных меток:

language-tag = primary-tag \*( "-" subtag )

primary-tag = 1\*8ALPHA subtag = 1\*8ALPHA

Внутри метки не допустимы пробелы и все метки не чувствительны к регистру. Пространство имен меток языка управляется IANA. Например метки содержат:

en, en-US, en-cockney, i-cherokee, x-pig-latin

Любая двухсимвольная первичная метка является меткой аббревеатуры языка ISO 639, а любая двухсимвольная подчиненная метка является меткой кода страны ISO 3166. (Последние три метки из вышеперечисленных - не зарегистрированные метки; все, кроме последней - примеры меток, которые скорее всего ьудут зарегистрированы в будущем.)

## 3.11 Метки объектов (Entity Tags).

Метки объектов используются для сравнения двух или более объектов одного и того же запрошенного ресурса. HTTP/1.1 использует метки объектов в полях заголовка ETag, If-Match, If-None-Match, и If-Range. Метка объекта состоит из непрозрачной строки, заключенной в кавычки (opaque quoted string), возможно предваренной индикатором слабости (weakness indicator).

entity-tag = [ weak ] opaque-tag

weak = "W/" opaque-tag = quoted-string

"Сильная метка объекта" ("strong entity tag") может быть распространнена на два объекта ресурса, только тогда, когда они пооктетно эквивалентны.

"Слабая метка объекта" ("weak entity tag"), обозначяемая префиксом "W/", может быть распространена на два объекта ресурса только тогда, когда объекты эквивалентны и могли бы заменять друг друга без значительного изменения в семантике. Слабая метка объекта может использоваться только для слабого сравнения (weak comparison).

Метка объекта должна быть уникальна среди всех версий всех объектов, связанных с конкретным ресурсом. Данное значение метки объекта может использоваться для объектов, полученных запросами различных URI без предположения эквивалентности этих объектов.

## 3.12 Единицы измерения диапазонов (Range Units).

HTTP/1.1 позволяет клиенту запрашивать только часть объекта. HTTP/1.1 использует еденицы измерения диапазонов в полях заголовка Range и Content-Range. Объект может быть разбит на части соответственно различным структурным модулям.

range-unit = bytes-unit | other-range-unit

bytes-unit = "bytes" other-range-unit = token

Единственая еденица измерения диапазонов, определенная в HTTP/1.1 - это "bytes". Реализации HTTP/1.1 могут игнорировать диапазоны, определенные с использованием других едениц измерения. HTTP/1.1 был разработан, чтобы обеспечения возможности реализации приложений, которые не зависят от знания диапазонов.

# 4. HTTP сообщение (HTTP Message).

## 4.1 Типы сообщений.

HTTP сообщения делятся на запросы клиента серверу и ответы сервера клиенту.

HTTP-message = Request | Response ; сообщения HTTP/1.1

Сообщения запроса и ответа используют обобщенный формат сообщения RFC 822 для пересылки объектов (полезной нагрузки сообщения). Оба типа сообщений выглядят следующим образом: сначала идет начальная строка (start-line), затем один или несколько полей заголовка (называемых также просто "заголовки"), затем пустая строка (то есть строка, равная CRLF), указывающая конец полей заголовка, а затем, возможно, тело сообщения.

generic-message = start-line \*message-header CRLF [ message-body ]

start-line = Request-Line | Status-Line

В интересах ошибкоустойчивости, серверам следует игнорировать все пустые строки, полученные перед строкой запроса (Request-Line). Другими словами, если сервер читает поток протокола и в самом начале сообщения получает CRLF, то ему следует этот CRLF игнорировать.

Некоторые ошибочные реализации HTTP/1.0 клиентов генерируют дополнительные CRLF после запроса POST. Стоит вновь повторить, что это явно запрещено нормальной записью Бекуса-Наура. HTTP/1.1 клиент не должен добавлять дополнительные CRLF перед запросом и после него.

## 4.2 Заголовки сообщений.

Поля заголовков HTTP, которые включают поля общих заголовков (general-header), заголовков запроса (request-header), заголовков ответа (response-header), и заголовков объекта (entity-header), имеют такой же обобщенный формат, как тот, что описан в разделе 3.1 RFC 822. Каждое поле заголовка состоит из имени поля, двоеточия (":") и значения поля. Имена полей не чувствительны к регистру. Значению поля может предшествовать любое число LWS, хотя предпочтителен одиночный SP. Поля заголовка могут занимать несколько строк. При этом каждая следующая строка продолжения начинается по крайней мере одним SP или HT. Приложениям следует придерживаться "общей формы" ("common form") при генерации HTTP конструкций, так как могут существовать реализации, которые не в состоянии принимать что-либо кроме общих форм.

message-header = field-name ":" [ field-value ] CRLF

field-name = token field-value = \*( field-content | LWS )

field-content = <октеты, составляющие значение поля и состоящие или из \*TEXT или из комбинаций лексем, tspecials, и quoted-string>

Порядок, в котором получены поля заголовка с различными именами не имеет значения. Однако "хорошей практикой" является то, что сначала посылаются поля общих заголовков, затем поля заголовков запроса или заголовков ответа, и, наконец, поля заголовков объекта.

Несколько полей заголовка с одиннаковыми именами могут присутствовать в сообщении тогда и только тогда, когда все значения полей, входящих в заголовок, определяют разделенный запятыми список [то есть #(value)]. Должно быть возможно объединить несколько таких полей заголовка в одну пару "имя поля: значение поля" (не измененяя этим семантику сообщения) путем присоединения каждого последующего значения поля к первому через запятые. Порядок, в котором получены поля с одинаковыми именами, имеет значение для интерпретации объединенного значения поля, и, следовательно, прокси-сервер не должен изменять порядок значений этого поля при пересылке.

## 4.3 Тело cообщения.

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-Encodingы.

message-body = entity-body | <entity-body закодированно согласно 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, различаются для запросов и ответов.

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

Включать или не включать 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 информационными кодами состояния 1xx, кодом 204 (Нет содержимого, No Content) и кодом 304 (Не модифицирован, Not Modified) не должны содержать тела сообщения (message-body). Все остальные ответы содержат тело сообщения, даже если оно имеет нулевую длину.

## 4.4 Длина сообщения.

Когда тело сообщения (message-body) присутствует в сообщении, длина этого тела определяется одним из следующих методов (в порядке старшинства):

1. Любое сообщение ответа, которое не должно включать тело сообщения (message-body) (например ответы с кодами состояния 1xx, 204, 304 и все ответы на запрос HEAD) всегда завершается пустой строкой после полей заголовка, независимо от полей заголовка объекта (entity-header fields), представленных в сообщении.
2. Если поле заголовка Transfer-Encoding присутствует и указывает на применение кодирования передачи "chunked", то длина определяется кодированием по кускам (chunked encoding).
3. Если поле заголовка Content-Length присутствует, то его значение представляет длину тела сообщения (message-body) в байтах.
4. Если сообщение использует медиатип "multipart/byteranges", который саморазграничен, то он и определяет длину. Этот медиа тип не должен использоваться, если отправитель не знает, способен ли получатель его обработать; присутствие в запросе заголовка Range с несколькими спецификаторами диапазонов байтов (byte-range) подразумевает, что клиент может анализировать multipart/byteranges ответы.
5. Длина определяется закрытием соединения сервером. (Закрытие соединения не может использоваться для указания конца тела запроса, так как в этом случае у сервера не остается никакой возможности послать обратно ответ).

Для совместимости с HTTP/1.0 приложениями HTTP/1.1 запросы, содержащие тело сообщения (message-body) должны включать допустимое поле заголовка Content-Length, пока не известно, что сервер является HTTP/1.1 совместимым. Если запрос содержит тело сообщения (message-body), и Content-Length не указано, серверу следует послать ответ с кодом состояния 400 (Испорченный Запрос, Bad Request), если он не может определить длину сообщения, или с кодом состояния 411 (Требуется длина, Length Required), если он настаивает на получении Content-Length.

Все HTTP/1.1 приложения, которые получают объекты, должны понимать кодирование передачи типа "chunked", таким образом разрешается использование данного механизма для таких сообщений, длина которых не может быть определена заранее.

Сообщения не должны одновременно включать и поле заголовка Content-Length и применять кодирование передачи типа "chunked". Если поступило сообщение с полем Content-Length и закодированное с применением кодирования передачи "chunked", то поле Content-Length должно игнорироваться.

Если поле Content-Length присутствует в сообщении, которое допускает наличие тела сообщения (message-body), то значение поля должно точно соответствовать числу октетов в теле сообщения. HTTP/1.1 агенты пользователя должны информировать пользователя в случае получения и обнаружения недопустимой длины.

## 4.5 Общие поля заголовка.

Имеется несколько полей заголовка, которые применяются как для сообщений запросов, так и для сообщений ответов, но которые не применяются к передаваемому объекту. Эти поля заголовка применяются только к передаваемому сообщению.

general-header = Cache-Control | Connection | Date | Pragma | Transfer-Encoding | Upgrade | Via

Имена общих полей заголовка (general-header fields) могут быть надежно расширены только в сочетании с изменением версии протокола. Однако, новые или экспериментальные поля заголовка могут получить семантику общих полей заголовка (general-header fields), если все стороны соединения распознают их как общие поля заголовка. Нераспознанные поля заголовка обрабатываются как поля заголовка объекта (entity-header).

# 5. Запрос (Request).

Сообщение запроса сервера клиентом содержит в первой строке: метод, который нужно применить к ресурсу, идентификатор ресурса и используемую версию протокола.

Request = Request-Line \*( general-header | request-header | entity-header ) CRLF [ message-body ]

## 5.1 Строка запроса (Request-Line).

Строка запроса (Request-Line) начинается с лексемы метода, затем следует запрашиваемый URI (Request-URI), версия протокола и CRLF. Эти элементы разделяются SP. В строке запроса (Request-Line) не допустимы CR и LF, исключение составляет конечная последовательность CRLF.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

### 5.1.1 Метод (Method).

Лексема метода указывает метод, который нужно применить к ресурсу, идентифицированному запрашиваемым URI (Request-URI). Метод чувствителен к регистру.

Method = "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT" | "DELETE" | "TRACE" | extension-method

extension-method = token

Список методов, применимых к ресурсу, может быть указан в поле заголовка Allow. Возврашаемый код состояния ответа всегда сообщает клиенту, допустим ли метод для ресурса в настоящее время, так как набор допустимых методов может изменяться динамически. Серверам следует возвратить код состояния 405 (Метод не допустим, Method Not Allowed), если метод известен серверу, но не применим для запрошенного ресурса, и 501 (Не реализовано, Not Implemented), если метод не распознан или не реализован сервером. Список методов, известных серверу, может быть указан в поле заголовка ответа Public.

Методы GET и HEAD должны поддерживаться всеми универсальными (general-purpose) серверами. Остальные методы опциональны.

### 5.1.2 URI запроса (Request-URI).

URI запроса (Request-URI) - это Единообразный Идентификатор Ресурса (URL), который идентифицирует ресурс запроса.

Request-URI = "\*" | absoluteURI | abs\_path

Три опции для URI запроса (Request-URI) зависят от характера запроса. Звездочка "\*" означает, что запрашивается не специфический ресурс, а сервер непосредственно, и допустим только в том случае, когда используемый метод не обязательно обращается к ресурсу. В качестве примера:

OPTIONS \* HTTP/1.1

absoluteURI необходим, когда запрос производится через прокси-сервер. Прокси-сервер перенаправляет запрос на сервер или обслуживает его, пользуясь кэшем, и возвращает ответ. Прокси-сервер может переслать запрос другому прокси-серверу или непосредственно серверу, определенному absoluteURI. Чтобы избежать зацикливания запроса прокси-сервер должен быть способен распознавать все имена сервера, включая любые псевдонимы, локальные разновидности, и числовые IP адреса. Request-Line может быть, например, таким:

GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1

Чтобы обеспечить переход к absoluteURI во всех запросах в будущих версиях HTTP, все HTTP/1.1 сервера должны принимать absoluteURI в запросах, хотя HTTP/1.1 клиенты будут генерировать их только в запросах к прокси-серверам.

Наиболее общая форма Request-URI используется для идентификации ресурса на первоначальном сервере или шлюзе. В этом случае абсолютный путь URI должен быть передан как Request-URI, а сетевое расположение URI (net\_loc) должно быть передано в поле заголовка Host. Для последнего примера клиент, желающий получить ресурс непосредственно с первоначального сервера должен создать TCP соединение на 80 порт хоста "www.w3.org" и послать строки:

GET /pub/WWW/TheProject.html HTTP/1.1 Host: www.w3.org

и далее остаток запроса. Абсолютный путь не может быть пустым; если оригинальный URI пуст, то он должен запрашиваться как "/" (корневой каталог сервера).

Если прокси-сервер получает запрос без пути в Request-URI, и метод запроса допускает форму запроса "\*", то последний прокси-сервер в цепочке запросов должен передать запрос, в котором Request-URI равен "\*". Например запрос

OPTIONS http://www.ics.uci.edu:8001 HTTP/1.1

был бы передан прокси-сервером в виде

OPTIONS \* HTTP/1.1 Host: www.ics.uci.edu:8001

после соединения с портом 8001 хоста "www.ics.uci.edu".

Первоначальный сервер должен декодировать Request-URI, чтобы правильно интерпретировать запрос. Серверам сдледует отвечать на недопустимые Request-URI соответствующим кодом состояния.

В запросах, пересылаемых прокси-сервером, часть "abs\_path" URI запроса (Request-URI) никогда не должна перезаписываться, за исключением случая, отмеченного выше, когда пустой abs\_path заменяется на "\*", независимо от внутренней реализации прокси-сервера.

Правило "ничто не перезаписывать" предохраняет прокси-сервера от изменения значения запроса, в котором первоначальный сервер неправильно использует не зарезервированные символы URL для своих целей.

### 5.2 Ресурс, идентифицируемый запросом.

Первоначальные HTTP/1.1 сервера должны учитывать, что точный ресурс, идентифицируемый интернет-запросом определяется путем исследования запрашиваемого URI (Request-URI) и поля заголовка Host.

Первоначальный сервер, который не различает ресурсы по запрошенному хосту (host), может игнорировать значение поля заголовка Host.

Первоначальный сервер, который различает ресурсы на основании запрошенного хоста (host) (иногда называемые виртуальными хостами или vanity hostnames) должен пользоваться следующими правилами для определения ресурса, запрошенного в HTTP/1.1 запросе:

1. Если Request-URI - это absoluteURI, то хост - это часть Request-URI. Любые значения поля заголовка Host в запросе должны игнорироваться.

2. Если Request-URI - не absoluteURI, а запрос содержит поле заголовка Host, то хост определяется значением поля заголовка Host.

3. Если хоста, определенного правилами 1 или 2 не существует на сервере, кодом состояния ответа должен быть 400 (Испорченный Запрос, Bad Request).

Получатели HTTP/1.0 запроса, в котором отсутствует поле заголовка Host, могут попытаться использовать эвристику (например, исследовать путь в URI на предмет уникальности на каком-либо из хостов) для определения какой именно ресурс запрашивается.

### 5.3 Поля заголовка запроса.

Поля заголовка запроса позволяют клиенту передать серверу дополнительную информацию о запросе и о самом клиенте. Эти поля действуют как модификаторы запроса с семантикой, эквивалентной параметрам вызова методов в языках программирования.

request-header = Accept | Accept-Charset | Accept-Encoding | Accept-Language | Authorization | From | Host | If-Modified-Since | If-Match | If-None-Match | If-Range | If-Unmodified-Since | Max-Forwards | Proxy-Authorization | Range | Referer | User-Agent

Множество имен полей заголовка запроса (Request-header) может быть надежно расширено только в сочетании с изменением версии протокола. Однако, новые или экспериментальные поля заголовка могут получить семантику полей заголовка запроса (Request-header), если все стороны соединения распознают их как поля заголовка запроса (Request-header). Нераспознанные поля заголовка обрабатываются как поля заголовка объекта (entity-header).

# 6 Ответ (Response).

После получения и интерпретации сообщения запроса, сервер отвечает сообщением HTTP ответа.

Response = Status-Line \*( general-header | response-header | entity-header ) CRLF [ message-body ]

## 6.1 Строка состояния (Status-Line).

Первая строка ответа - это строка состояния (Status-Line). Она состоит из версии протокола (HTTP-Version), числового кода состояния (Status-Code) и поясняющей фразы (Reason-Phrase) разделенных символами SP. CR и LF не допустимы в Status-Line, за исключением конечной последовательности CRLF.

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

### 6.1.1 Код состояния и поясняющая фраза.

Элемент код состояния (Status-Code) - это целочисленный трехразрядный код результата попытки понять и выполнить запрос. Эти коды полностью определены в разделе 10. Поясняющая фраза (Reason-Phrase) предназначена для короткого текстового описания кода состояния. Код состояния (Status-Code) предназначен для использования автоматами, а поясняющая фраза предназначена для живых пользователей. От клиента не требуется исследовать или отображать поясняющую фразу (Reason-Phrase).

Первая цифра кода состояния определяет класс ответа. Последние две цифры не имеют определенной роли в классификации. Имеется 5 значений первой цифры:

* 1xx: Информационные коды - запрос получен, продолжается обработка.
* 2xx: Успешные коды - действие было успешно получено, понято и обработано.
* 3xx: Коды перенаправления - для выполнения запроса должны быть предприняты дальнейшие действия.
* 4xx: Коды ошибок клиента - запрос имеет плохой синтаксис или не может быть выполнен.
* 5xx: Коды ошибок сервера - сервер не в состоянии выполнить правильный запрос.

Конкретные значения числовых кодов состояния, определенных в HTTP/1.1, и примерный набор соответствующих поясняющих фраз (Reason-Phrase) приводятся ниже. Поясняющие фразы (Reason-Phrase), перечисленные здесь являются рекомендуемыми, но могут быть заменены на эквивалентные не влияя на протокол.

Status-Code = "100" ; Продолжать, Continue |

"101" ; Переключение протоколов, ; Switching Protocols |

"200" ; OK |

"201" ; Создан, Created |

"202" ; Принято, Accepted |

"203" ; Не авторская информация, ; Non-Authoritative Information |

"204" ; Нет содержимого, No Content |

"205" ; Сбросить содержимое, Reset ; Content |

"206" ; Частичное содержимое, Partial ; Content |

"300" ; Множественный выбор, Multiple ; Choices |

"301" ; Постоянно перемещен, Moved ; Permanently |

"302" ; Временно перемещен, Moved ; Temporarily |

"303" ; Смотреть другой, See Other |

"304" ; Не модифицирован, Not Modified |

"305" ; Используйте прокси-сервер, Use ; Proxy |

"400" ; Испорченный запрос, Bad Request |

"401" ; Несанкционированно, Unauthorized |

"402" ; Требуется оплата, Payment ; Required |

"403" ; Запрещено, Forbidden |

"404" ; Не найден, Not Found |

"405" ; Метод не допустим, Method Not ; Allowed |

"406" ; Не приемлем, Not Acceptable |

"407" ; Требуется установление ; подлинности через прокси-сервер, ; Proxy Authentication Required |

"408" ; Истекло время ожидания запроса, ; Request Timeout |

"409" ; Конфликт, Conflict |

"410" ; Удален, Gone |

"411" ; Требуется длина, Length Required |

"412" ; Предусловие неверно, ; Precondition Failed |

"413" ; Объект запроса слишком большой, ; Request Entity Too Large |

"414" ; URI запроса слишком длинный, ; Request-URI Too Long |

"415" ; Неподдерживаемый медиатип, ; Unsupported Media Type |

"500" ; Внутренняя ошибка сервера, ; Internal Server Error |

"501" ; Не реализовано, Not Implemented |

"502" ; Ошибка шлюза, Bad Gateway |

"503" ; Сервис недоступен, Service ; Unavailable |

"504" ; Истекло время ожидания от шлюза, ; Gateway Timeout |

"505" ; Не поддерживаемая версия HTTP, ; HTTP Version Not Supported | extension-code

extension-code = 3DIGIT

Reason-Phrase = \*<TEXT не включающий CR, LF>

Коды состояния HTTP расширяемы. HTTP приложениям не требуется понимать значение всех зарегистрированных кодов состояния, хотя их понимание очень желательно. Приложения должны понимать класс любого кода состояния, который обозначается первой цифрой, и обрабатывать любой нераспознанный ответ как эквивалентный коду состояния x00 этого класса, за исключением тех случаев, когда нераспознанный ответ не должен кэшироваться. Например, если клиентом получен и не был распознан код состояния 431, то он может безопасно считать, что в запросе что-то было неправильно и обрабатывать ответ, как если бы был получен код состояния 400. В таких случаях агентам пользователя следует представить пользователю объект, возвращенный в ответе, так как этот объект, вероятно, включает читабельную для человека информацию, которая поясняет необычное состояние.

### 6.2 Поля заголовка ответа.

Поля заголовка ответа (response-header fields) позволяют серверу передавать дополнительную информацию об ответе, которая не может быть помещена в строку состояния Status-Line. Эти поля заголовка дают информацию о сервере и о дальнейшем доступе к ресурсу, указанному этим Request-URI.

response-header = Age | Location | Proxy-Authenticate | Public | Retry-After | Server | Vary | Warning | WWW-Authenticate

Множество имен полей заголовка ответа (Response-header) может быть надежно расширенО только в сочетании с изменением версии протокола. Однако, новые или экспериментальные поля заголовка могут получить семантику полей заголовка ответа (Response-header), если все стороны соединения распознают их как поля заголовка ответа (Response-header). Нераспознанные поля заголовка обрабатываются как поля заголовка объекта (entity-header).

# 7 Объект (Entity).

Сообщения запросов и ответов могут передать объект, если это не запрещено методом запроса или кодом состояния ответа. Объект состоит из полей заголовка объекта (entity-header) и тела объекта (entity-body), хотя некоторые ответы могут включать только заголовки объекта (entity-headers).

Этот раздел относится как к отправителю, так и к получателю, то есть к клиенту или серверу, в зависимости от того, кто посылает, а кто получает объект.

## 7.1 Поля заголовка объекта.

Поля заголовка объекта (Entity-header fields) определяют опциональную метаинформацию о теле объекта (entity-body) или, если тело отсутствует, о ресурсе, идентифицированном запросом.

entity-header = Allow | Content-Base | Content-Encoding | Content-Language | Content-Length | Content-Location | Content-MD5 | Content-Range | Content-Type | ETag | Expires | Last-Modified | extension-header

extension-header = message-header

Механизм расширения полей заголовка позволяет вводить дополнительные поля заголовка объекта (entity-header fields) не изменяя протокол, но эти поля могут быть и не распознаны получателем. Получатель должен игнорировать нераспознанные поля заголовка, а прокси-сервер должен просто пересылать их без изменений.

## 7.2 Тело объекта.

Тело объекта (если оно присутствует) посылается с HTTP запросом или ответом и имеет формат и кодирование, определяемое полями заголовка объекта (entity-header fields).

entity-body = \*OCTET

Тело объекта (entity-body) представлено в сообщении только тогда, когда присутствует тело сообщения (message-body). Тело объекта (entity-body) получается из тела сообщения (message-body) декодированием любого кодирования передачи, указанного в поле Transfer-Encoding, которое может быть применено для гарантирования безопасной и правильной передачи сообщения.

### 7.2.1 Тип.

Когда в сообщении содержится тело объекта (entity-body), тип данных этого тела определяется полями заголовка Content-Type и Content-Encoding. Они определяют двухуровневую упорядоченную модель кодирования:

entity-body := Content-Encoding( Content-Type( data ) )

Тип содержимого (Content-Type) определяет медиатип лежащих в основе данных. Кодирование содержимого (Content-Encoding) может использоваться для указания любых дополнительных кодирований содержимого, примененных к данным (обычно с целью сжатия). Кодирование содержимого (Content-Encoding) является свойством запрошенного ресурса. По умолчанию никакого кодирования не задано.

В любое HTTP/1.1 сообщение, содержащее тело объекта (entity-body) включает поле заголовка Content-Type, определяющее медиатип этого тела. В том и только в том случае, когда медиатип не указан в поле Content-Type, получатель може попытаться самостоятельно определить медиатип, проверяя содержимое и/или расширение (расширения) в URL, используемого для идентификации ресурса. Если медиатип остался нераспознан, получателю следует обрабатывать его как тип "application/octet-stream".

### 7.2.2 Длина.

Длина тела объекта (entity-body) - это длина тела сообщения (message-body), полученного после декодирования всех кодирований передачи.

# 8 Соединения (Connections).

## 8.1 Постоянные соединения (Persistent Connections).

### 8.1.1 Цель.

До введения в протокол постоянных соединений для запроса каждого URL устанавливалось отдельное TCP соединение, что увеличивало нагрузку на HTTP сервера и вызывало перегрузку сетей. Использование встроенных изображений и других связанных данных часто требует от клиента инициировать несколько запросов к одному серверу за короткий промежуток времени.

Постоянные HTTP соединения имеют ряд преимуществ:

* Открытие и закрытие меньшего количества TCP соединений экономит время центрального процессора и память, используемую для управляющих блоков протокола TCP.
* HTTP запросы и ответы может быть конвейеризованы в соединении. Конвейерная обработка позволяет клиенту делать несколько запросов не ожидая ответа на каждый, позволяет пользоваться единственным TCP соединением более эффективно, с меньшими затратами времени.
* Загрузка сети уменьшается с уменьшением числа пакетов, необходимых для открытия TCP соединений, и, следовательно, предоставляет протоколу TCP достаточно времени для определения состояния перегрузки сети.
* HTTP может развиваться более элегантно; так как ошибки можно сообщать без закрытия TCP соединения в качестве штрафа. Клиенты, использующие будущие версии HTTP могли бы оптимистично пробовать новые возможности, а при связи со старым сервером, повторять запрос, используя старую семантику после сообщения об ошибке.

### 8.1.2 Общее описание.

Значительное отличие HTTP/1.1 от ранних версий HTTP состоит в том, что постоянные соединения являются заданным по умолчанию поведением любого HTTP соединения. То есть если не обозначено иного, клиент может считать, что сервер поддержит постоянное соединение.

Постоянные соединения обеспечивают механизм, согласно которому клиент и сервер могут сообщить о разрыве TCP соединения. Это сигнализируется путем использования поля заголовка Connection.

#### 8.1.2.1 Обсуждение (Negotiation).

HTTP/1.1 сервер в праве считать, что HTTP/1.1 клиент не поддерживает постоянное соединение, если посланный в запросе заголовок Connection содержит лексему соединения (connection-token) "close". Если сервер решает закрыть соединение немедленно после посылки ответа, то ему необходимо послать заголовок Connection, который содержит лексему соединения (connection-token) "close".

HTTP/1.1 клиент долженждать закрытие соединения, но должен держать его открытым на основании того, содержит ли ответ сервера заголовок Connection с лексемой соединения "close". В случае, если клиент не хочет поддерживать соединение для последующих запросов, ему надлежит послать заголовок Connection, содержащий лексему соединения "close".

Если клиент или сервер посылает лексему закрытия соединения "close" в заголовке Connection, то запрос становится последним в соединении.

Чтобы соединение оставалось постоянным, все сообщения, передаваемые по нему должны иметь самоопределенную (self-defined) длину (то есть, не определяемую закрытием соединения).

#### 8.1.2.2 Конвейерная обработка (Pipelining).

Клиент, который поддерживает постоянные соединения умеет производить "конвейерную обработку" запросов (то есть посылать несколько запросов не ожидая ответа на каждый из них). Сервер должен послать ответы на эти запросы в том же самом порядке, в каком были получены запросы.

Клиенты, которые поддерживают постоянные соединения и производят конвейерную обработку немедленно после установления соединения, должны быть готовы повторить соединение, если первая попытка конвейерной обработки дала сбой. Если клиент производит такой повтор, он не должен производить конвейерную обработку прежде чем узнает, что соединение постоянно. Также клиенты должны быть готовы повторить запросы, если сервер закрывает соединение перед посылкой всех соответствующих ответов.

### 8.1.3 Прокси-сервера (Proxy Servers).

Очень важно, чтобы прокси-сервера правильно реализовывали свойства полей заголовка Connection.

Прокси-сервер должен сообщать о постоянных соединениях отдельно своим клиентам и отдельно первоначальным серверам (или другим прокси-серверам), которые с ним соединены. Каждое постоянное соединение применяется только к одной транспортной связи.

Прокси-сервер не должен устанавливать постоянных соединений с HTTP/1.0 клиентом.

### 8.1.4 Практические соображения.

Сервера обычно имеют некоторое значение времени ожидания, после которого они не поддерживают неактивное соединение. Прокси-сервера могут выставлять его значение более высоким, так как, вероятно, клиент сделает больше соединений через этот же сервер. Использование постоянных соединений не вводит никаких ограничений на продолжительность времени ожидания как для клиента, так и для сервера.

Когда у клиента или сервера истекло время ожидания, ему необходимо произвести закрытие транспортного соединения. Как клиентам, так и серверам надлежит постоянно наблюдать за другой стороной на предмет закрытия соединения и отвечать соответственно. Если клиент или сервер не сразу обнаруживает закрытие соединения другой стороной, то это вызывает не оправданную трату ресурсов сети.

Клиент, сервер, или прокси-сервер в праве закрыть транспортное соединение в любое время. Например, клиент может начать посылать новый запрос в то время, когда сервер решает закрыть "бездействующее" соединение. С точки зрения сервера, соединение закрывается, в то время как оно было неактивно, но с точки зрения клиента, запрос произошел.

Это означает, что клиенты, серверы, и прокси-серверы должны быть в состоянии обрабатывать асинхронные события закрытия. Программному обеспечению клиента следует вновь открыть транспортное соединение и повторно передать прерванный запрос без взаимодействия с пользователем, если метод запроса идемпотентен; другие методы не должны быть повторены автоматически, хотя агенты пользователя могут предложить оператору повторить запрос.

Однако это автоматического повтора производить не следует, если сбой происходит уже во втором запросе.

Серверам всегда следует отвечать по крайней мере на один запрос в соединении, если это возможно. Серверам не следует разрывать соединение в середине передачи ответа, если не предполагается сетевого или клиентского отказа.

### 8.2 Требования к передаче сообщений.

Общие требования:

* HTTP/1.1 серверам следует поддерживать постоянные соединения и использовать механизмы управления потоком данных TCP в целях уменьшения временных перегрузок, вместо закрытия соединений, которые, как ожидается, могут быть повторно использованы клиентами. Последняя методика может усиливать сетевую загрузку.
* HTTP/1.1 (или более поздним) клиентам, посылающим тело сообщения (message-body) следует контролировать сетевое соединение на предмет ошибок во время передачи запроса. Если клиент обнаруживает ошибку, ему следует немедленно прекратить передачу тела сообщения. Если тело посылается с использованием кодирования "по кускам" ("chunked"), то кусок нулевой длины, и пустой завершитель могут использоваться для индикации преждевременного конца сообщения. Если телу предшествовал заголовок Content-Length, клиент должен закрыть соединение.
* HTTP/1.1 (или более поздний) клиент должен быть готов принять ответ с кодом состояния 100 (Продолжать, Continue), предшествующий основному ответу.
* HTTP/1.1 (или более поздний) сервер, который получает запрос от HTTP/1.0 (или более раннего) клиента не должен отвечать кодом состояния 100 (Продолжать, Continue); ему следует либо ждать пока запрос будет выполнен обычным образом (то есть без использования прерванного запроса), либо преждевременно закрыть соединение.
* После получения метода, подчиненного этим требованиям, от HTTP/1.1 (или более позднего) клиента, HTTP/1.1 (или более поздний) сервер должен либо ответить кодом состояния 100 (Продолжать, Continue) и продолжать чтение входного потока, либо ответить кодом состояния ошибки. Если сервер ответил кодом состояния ошибки, то он может либо закрыть транспортное соединение (TCP), либо продолжать читать и отбрасывать оставшуюся часть запроса. Он не должен выполнять запрошенный метод, если возвратил код состояния ошибки.

Клиентам следует помнить номер версии HTTP, используемой сервером по крайней мере в последний раз; если HTTP/1.1 клиент встречал HTTP/1.1 или более поздний ответ от сервера, и видит закрытие соединения перед получением какого-либо кода состояния от сервера, клиенту следует повторить запрос без взаимодействия с пользователем, если метод запроса идемпотентен; другие методы не должны быть повторены автоматически, хотя агенты пользователя могут предложить оператору повторить запрос. Если клиент повторяет запрос, то он

* должен сначала послать поля заголовка запроса, а затем
* должен либо ожидать ответа сервера с кодом 100 (Продолжать, Continue) и затем продолжать, либо с кодом состояния ошибки.

Если HTTP/1.1 клиент не встречал ответа сервера версии HTTP/1.1 или более поздней, то ему следует считать, что сервер реализует HTTP/1.0 или более старый протокол и не использовать ответы с кодом состояния 100 (Продолжать, Continue). Если в такой ситуации клиент видит закрытие соединения перед получением какого-либо ответа с кодом состояния от сервера, то ему следует повторить запрос. Если клиент повторяет запрос к этому HTTP/1.0 серверу, то он должен использовать следующий алгоритм "двоичной экспоненциальной задержки" ("binary exponential backoff"), чтобы гарантировать получение надежного ответа:

1. Инициировать новое соединение с сервером.
2. Передать заголовки запроса (request-headers).
3. Инициализировать переменную R примерным временем передачи информации на сервер и обратно (например на основании времени установления соединения), или постоянным значение в 5 секунд, если время передачи не доступно.
4. Вычислить T = R \* (2\*\*N), где N - число предыдущих повторов этого запроса.
5. Либо дождаться от сервера ответа с кодом ошибки, либо просто выждать T секунд (смотря что произойдет раньше).
6. Если ответа с кодом ошибки не получено, после T секунд передать тело запроса.
7. Если клиент обнаруживает, что соединение было закрыто преждевременно, то ему нужно повторять начиная с шага 1, пока запрос не будет принят, либо пока не будет получен ошибочный ответ, либо пока у пользователя не кончится терпение и он не завершит процесс повторения.

Независимо от того, какая версия HTTP реализована сервером, если клиент получает код состояния ошибки, то он

* не должен продолжать и
* должен закрыть соединение, если он не завершил посылку сообщения.

HTTP/1.1 (или более позднему) клиенту, который обнаруживает закрытие соединения после получения ответа с кодом состояния 100 (Продолжать, Continue), но до получения ответа с другим кодом состояния, следует повторить запрос, но уже не ожидать ответа с кодом состояния 100 (Продолжать, Continue).

# 9 Определения методов.

Множество общих для HTTP/1.1 методов приводится ниже. Хотя это множество может быть расширено, нельзя считать, что дополнительные методы имеют одиннаковую семантику, если они являются расширениями разных клиентов и серверов.

Поле заголовка запроса Host должно сопровождать все HTTP/1.1 запросы.

## 9.1 Безопасные и идемпотентные методы.

### 9.1.1 Безопасные методы.

Программистам следует понимать, что программное обеспечение при взаимодействии с Интернетом представляет пользователя, и программе следует информировать пользователя о любых действиях, которые он может произвести, но которые могут иметь непредсказуемое значение для него или других лиц.

В частности было принято соглашение, что методы GET и HEAD никогда не должны иметь иного значения, кроме загрузки (retrieval). Эти методы следует рассматривать как "безопасные". Это позволяет агенту пользователя представлять другие методы, такие как POST, PUT и DELETE, таким образом, чтобы пользователь был проинформирован о том, что он запрашивает выполнение потенциально опасного действия.

Естественно невозможно гарантировать, что сервер не генерирует побочных эффектов в результате выполнения запроса GET; фактически, некоторые динамические ресурсы содержат такую возможность. Важное различие здесь в том, что не пользователь запрашивает побочные эффекты, и, следовательно, пользователь не может нести ответственность за них.

### 9.1.2 Идемпотентные методы.

Методы могут также обладать свойством "идемпотентности" ("idempotence") в том смысле, что побочные эффекты от N > 0 идентичных запросов такие же, как от одиночного запроса (за исключение ошибок и проблем устаревания). Методы GET, HEAD, PUT и DELETE обладают данным свойством.

## 9.2 OPTIONS.

Метод OPTIONS представляет запрос информации об опциях соединения, доступных в цепочке запросов/ответов, идентифицируемой запрашиваемым URI (Request-URI). Этот метод позволяет клиенту определять опции и/или требования, связанные с ресурсом, или возможностями сервера, но не производя никаких действий над ресурсом и не инициируя его загрузку.

Если ответ сервера - это не сообщение об ошибке, то ответ не должен содержать иной информации объекта, кроме той, которую можно рассматривать как опции соединения (например Allow - можно рассматривать как опцию соединения, а Content-Type - нет). Ответы на этот метод не кэшируются.

Если запрашиваемый URI (Request-URI) - звездочка ("\*"), то запрос OPTIONS предназначен для обращения к серверу в целом. Если код состояния ответа - 200, то ответу следует содержать любые поля заголовка, которые указывают опциональные возможности, реализуемые сервером (например, Public), включая любые расширения, не определенные данной спецификацией, в дополнение к соответствующим общим полям или полям заголовка ответа (response-header). Как описано в разделе 5.1.2, запрос "OPTIONS \*" может быть применен через прокси-сервер с определением адресуемого сервера в запрашиваемом URI (Request-URI) с пустым путем.

Если запрашиваемый URI (Request-URI) не звездочка ("\*"), то запрос OPTIONS применяется к опциям, которые доступны при соединении с указанным ресурсом. Если код состояния ответа - 200, то ответу следует содержать любые поля заголовка, которые указывают опциональные возможности, реализуемые сервером и применимые к указанному ресурсу (например Allow), включая любые расширения, не определенные данной спецификацией, в дополнение к соответствующим общим полям или полям заголовка ответа (response-header). Если запрос OPTIONS передается через прокси-сервер, то последний редактирует ответ, исключая те опции, которые не предусмотрены возможностями этого прокси-сервера.

## 9.3 GET.

Метод GET позволяет получать любую информацию (в форме объекта), идентифицированную запрашиваемым URI (Request-URI). Если запрашиваемый URI (Request-URI) обращается к процессу, производящему данные, то в качестве объекта ответа должны быть возвращены произведенные данные, а не исходный текст процесса, если сам процесс не выводит исходный текст.

Различается "условный GET" ("conditional GET"), при котором сообщение запроса включает поля заголовка If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, или If-Range. Условный метод GET запрашивает передачу объекта, только если последний удовлетворяет условиям, описанным в условных полях заголовка. Условный метод GET предназначен для уменьшения неоправданной загрузки сети, и позволяет обновлять кэшированные объекты без использования нескольких запросов или пересылки данных, уже сохраненных клиентом.

Различается также "частичный GET" ("partial GET"), при котором сообщение запроса включает поле заголовка Range. Частичный GET запрашивает передачу только части объекта. Частичный метод GET предназначен для уменьшения ненужной загрузки сети, и позволяет собирать объекты из частей, без передачи частей данных, уже сохраненных клиентом.

Ответ на запрос GET кэшируем тогда и только тогда, когда он отвечает требованиям кэширования в HTTP, описанным ниже.

## 9.4 HEAD.

Метод HEAD идентичен GET, за исключением того, что сервер не должен возвращать в ответе тело сообщения (message-body). Метаинформации, содержащейся в HTTP заголовках ответа на запрос HEAD следует быть идентичной информации, представляемой в ответ на запрос GET. Этот метод может использоваться для получения метаинформации об объекте запроса без непосредственной пересылки тела объекта (entity-body). Этот метод часто используется для тестирования гипертекстовых связей в целях проверки достоверности, достижимости, и времени модификации.

Ответ на запрос HEAD может быть кэшируемым в том смысле, что информация, содержащаяся в ответе может использоваться для модифицикации предварительно кэшированного объекта из этого ресурса. Если новые значения поля указывают, что кэшируемый объект отличается от текущего объекта (по таким параметрам, как Content-Length, Content-MD5, ETag или Last-Modified), то кэш должен обрабатывать содержимое как просроченное.

## 9.5 POST.

Метод POST используется для запроса, при котором адресуемый сервер принимает объект, включенный в запрос, как новое подчинение (subordinate) ресурса, идентифицированного запрашиваемым URI (Request-URI) в строке запроса (Request-Line). POST разработан для того, чтобы общим методом реализовать следующие функции:

* Аннотация существующих ресурсов;
* Регистрация сообщения на электронной доске объявлений (bulletin board), в конференции новостей (newsgroup), списке рассылки (mailing list), или подобной группе статей;
* Передача блока данных, например результат ввода в форме, процессу обработки;
* Расширение базы данных посредством конкатенирующей операции (append operation).

Фактически функция, выполняемая методом POST, определяется сервером и обычно зависит от запрашиваемого URI (Request-URI). Объект, передаваемый методом POST, относится к этому URI так же как файл относится к каталогу, в котором он находится, статья относится к конференции новостей (newsgroup), в которой она зарегистрирована, а запись относится к базе данных.

Действие, выполняемое методом POST может не давать в качестве результата ресурс, который можно было бы идентифицировать URI. В этом случае, в зависимости от того, включает ли ответ объект, описывающий результат, или нет, код состояния в ответе может быть как 200 (OK), так и 204 (Нет содержимого, No Content).

Если ресурс был создан на первоначальном сервере, ответу следует содержать код состояния 201 (Создан, Created) и включать объект, который описывает состояние запроса и ссылается на новый ресурс, а также заголовок Locatio.

Ответы на этот метод не кэшируемы, если ответ не включает соответствующие поля заголовка Cache-Control или Expires. Однако ответ с кодом состояния 303 (Смотреть другой, See Other) может использоваться для перенаправления агента пользователя для загрузки кэшируемого ресурса.

Запросы POST должны отвечать требованиям передачи сообщения.

## 9.6 PUT.

Запросы с методом PUT, которые содержат объект, сохраняются под запрашиваемым URI (Request-URI). Если Request-URI обращается к уже существующему ресурсу, включенный объект следует рассматривать как модифицированную версию объекта, находящегося на первоначальном сервере. Если Request-URI не указывает на существующий ресурс, и может интерпретироваться агентом пользователя как новый ресурс для запросов, первоначальный сервер может создать ресурс с данным URI. Если новый ресурс создан, то первоначальный сервер должен сообщить агенту пользователя об этом посредством ответа с кодом состояния 201 (Создан, Created). Если существующий ресурс модифицирован, то для указания успешного завершения запроса следует послать ответ с кодом состояния либо 200 (OK), либо 204 (Нет содержимого, No Content). Если ресурс не может быть создан или изменен для запрашиваемого URI (Request-URI), то следует послать ответ, отражающий характер проблемы. Получатель объекта не должен игнорировать заголовков Content-\* (например Content-Range), которых не понимает или не реализует, а должен в данном случае возвратить ответ с кодом состояния 501 (Не реализовано, Not Implemented).

Если запрос передается через кэш и запрашиваемый URI (Request-URI) идентифицирует один или несколько кэшированных в настоящее время объектов, то вхождения в кэш этих объектов должны обрабатываться как просроченные. Ответы на этот метод не кэшируемы.

Фундаментальное различие между запросами POST и PUT, отражено в различном значении запрашиваемого URI (Request-URI). URI в запросе POST идентифицирует ресурс, который обрабатывает включенный объект. Этим ресурсом может быть процесс, принимающий данные, шлюз к некоторому другому протоколу, или отдельный объект, который принимает аннотации (accepts annotations). Напротив, URI в запросе PUT идентифицирует объект включенный в запрос - агент пользователя назначает данный URI включенному ресурсу, а сервер не должен пытаться применить запрос к некоторому другому ресурсу. Если сервер желает применить запрос к другому URI, он должен послать ответ с кодом 301 (Перемещен постоянно, Moved Permanently); агент пользователя может затем принять собственное решение относительно переназначения запроса.

Один ресурс может быть идентифицирован несколькими различными URI. Например статья может иметь URI идентифицирующий "текущую версию", который отличен от URI, идентифицирующего каждую специфическую версию. В этом случае, запрос PUT на общий URI может отразиться на некоторых других URI, определенных первоначальным сервером.

HTTP/1.1 не определяет каким образом метод PUT воздействует на состояние первоначального сервера.

Запросы PUT должны подчиняться требованиям передачи сообщений.

## 9.7 DELETE.

Метод DELETE запрашивает первоначальный сервер об удалении ресурса, идентифицируемого запрашиваемым URI (Request-URI). Этот метод может быть отменен человеческим вмешательством (или другими средствами) на первоначальном сервере. Клиенту нельзя гарантировать, что операция была выполнена, даже если код состояния, возвращенный первоначальным сервером указывает на то, что действие было завершено успешно. Однако, серверу не следует отвечать об успешном выполнении, если во время формирования ответа он только собирается удалить ресурс или переместить его в недоступное положение.

Успешному ответу следует иметь код состояния 200 (OK), если он включает объект, описывающий состояние; иметь код состояния 202 (Принято, Accepted), если действие еще не было произведено; либо иметь код состояния 204 (Нет содержимого, No Content), если ответ сообщает об успехе (OK), но не содержит объекта.

Если запрос передается через кэш и запрашиваемый URI (Request-URI) идентифицирует один или несколько кэшированных в настоящее время объектов, то вхождения их должны обрабатываться как просроченные. Ответы на этот метод не кэшируемы.

## 9.8 TRACE.

Метод TRACE используется для вызова удаленного возврата сообщения запроса на уровне приложения. Конечному получателю запроса следует отразить полученное сообщение обратно клиенту как тело объекта ответа с кодом состояния 200 (OK). Конечным получателем является либо первоначальный сервер, либо первый прокси-сервер/шлюз, получивший нулевое значение (0) в поле Max-Forwards в запросе. Запрос TRACE не должен содержать объекта.

TRACE позволяет клиенту увидеть, что получается на другом конце цепочки запросов и использовать эти данные для тестирования или диагностической информации. Значение поля заголовка Via представляет особый интерес, так как оно действует как след цепочки запросов. Использование поля заголовка Max-Forwards позволяет клиенту ограничивать длину цепочки запросов, что полезно при тестировании бесконечных циклов в цепочке прокси-серверов, пересылающих сообщения.

Если запрос выполнен успешно, то ответу следует содержать все сообщение запроса в теле объекта (entity-body), а Content-Type следует быть равным "message/http". Ответы на этот метод не должны кэшироваться.

# 10 Определение кодов состояния.

Каждый код состояния, описанный ниже, включает описание метода (или методов), за которым он может следовать и метаинформацию, требуемую в ответе.

## 10.1 1xx - Информационные коды.

Этот класс кодов состояния указывает предварительный (временный) ответ, состоящий только из строки состояния (Status-Line) и опциональных заголовков, и завершающийся пустой строкой. Так как HTTP/1.0 не определял кодов состояния 1xx, серверы не должны посылать 1xx ответы HTTP/1.0 клиентам, за исключением экспериментальных условий.

## 10.1.1 100 Продолжать, Continue.

Клиент может продолжать запрос. Этот промежуточный ответ используется, для того, чтобы сообщить клиенту, что начальная часть запроса была получена и еще не отвергнута сервером. Клиенту следует продолжить посылку оставшихся данных запроса или, если запрос уже был выполнен, игнорировать этот ответ. Сервер должен послать заключительный ответ после того, как запрос будет выполнен.

## 10.1.2 101 Переключение протоколов, Switching Protocols

Сервер понимает и желает выполнить запрос клиента, если протокол прикладной программы в этом соединении будет изменен на тот, который указан в поле заголовка сообщения Upgrade (раздел 14.41). Сервер переключит протокол на тот, который определен в поле заголовка ответа Upgrade непосредственно после пустой строки, которая завершает ответ с кодом состояния 101.

Протокол должен быть переключен только тогда, когда это принесет выгоду. Например, переключение на более новую версию HTTP выгодно по сравнения с использованием более старых версий, а переключение на синхронный протокол реального времени может быть выгодно при предоставлении ресурсов, которые используют такие возможности.

## 10.2 2xx - Успешные коды.

Этот класс кодов состояния указывает, что запрос клиента был успешно получен, понят, и принят.

### 10.2.1 200 OK.

Запрос был удачно выполнен. Информация, возвращаемая с ответом зависит от метода, используемого в запросе. Например:

GET в ответе представлен объект, соответствующий запрошенному ресурсу;

HEAD в ответе представлены поля заголовка объекта (entity-header), соответствующие запрошенному ресурсу. Тело сообщения (message-body) отсутствует;

POST в ответе представлено описание объекта или содержится результат действия;

TRACE в ответе представлен объект, содержащий сообщение запроса, полученого конечным сервером.

### 10.2.2 201 Создан, Created.

Запрос был выполнен и в результате был создан новый ресурс. Новый созданный ресурс может быть вызван по URI (одному или нескольким), возвращенным в объекте ответа; наиболее специфический URL для ресурса отдается в поле заголовка Location. Первоначальный сервер должен создать ресурс перед возвратом кода состояния 201. Если действие не может быть выполнено немедленно, сервер должен возвратить ответ с кодом состояния 202 (Принято, Accepted) вместо 201.

### 10.2.3 202 Принято, Accepted.

Запрос был принят для обработки, но обработка не была завершена. В конечном счете запрос может быть, а может и не быть выполнен, поскольку он может быть отвергнут при фактической обработке. Не имеется никакой возможности вторичной посылки кода состояния от асинхронной операции типа этой.

Ответ с кодом состояния 202 преднамеренно уклончив. Цель его состоит в том, чтобы позволить серверу принять запрос для некоторого другого процесса (например пакетно-ориентированного процесса, который выполняется только один раз в день) и не требовать при этом, чтобы соединение агента пользователя с сервером сохранялось до завершения процесса. Объекту, возвращенному с этим ответом следует содержать индикатор текущего состояния запроса и либо ссылку на монитор состояния, либо некоторую оценку времени, когда пользователь может ожидать завершения выполнения запроса.

### 10.2.4 203 Не авторская информация, Non-Authoritative Information.

Возвращенная в заголовке объекта (entity-header) метаинформация - это не оригинал, доступный на первоначальном сервере, а документ, собранный из локальных копий или копий третьей стороны. Представленный документ может быть как подмножеством оригинальной версии, так и содержать сведения, которые в ней не были представлены. Например, включение локальной аннотирующей информацию о ресурсе может расширить метаинформацию, известную первоначальному серверу. Использование этого кода состояния в ответе не является необходимым, но может применяться тогда, когда код состояния ответа отличен от 200 (OK).

### 10.2.5 204 Нет содержимого, No Content.

Сервер выполнил запрос, но нет никакой новой информации, которую можно послать обратно. Если клиент - агент пользователя, ему не следует изменять вид документа, который послужил причиной запроса. Этот ответ предназначен прежде всего для того, чтобы позволить вводить данные для действий, не изменяя вид активного документа агента пользователя. Ответ может включать новую метаинформацию в форме заголовков объекта (entity-headers), которые следует добавить к документу, показываемому в настоящее время агентом пользователя.

Ответ с кодом состояния 204 не должен содержать тела сообщения, и, таким образом, всегда завершается первой пустой строкой после полей заголовка.

### 10.2.6 205 Сбросить содержимое, Reset Content.

Сервер выполнил запрос, и агенту пользователя следует отменить просмотр документа, который инициировал запрос. Этот ответ предназначен прежде всего для того, чтобы позволить ввод данных, осуществляемый пользователем, с последующей очисткой формы, в которой сделан ввод, так, чтобы пользователь мог легко инициировать следующее действие ввода. Ответ не должен содержать объект.

### 10.2.7 206 Частичное содержимое, Partial Content.

Сервер выполнил частичный GET запрос ресурса. Запрос должен содержать поле заголовка Range, указывающее желаемый диапазон. Ответ должен содержать либо поле заголовка Content-Range, указывающее диапазон, включенный в ответ, либо тип содержимого (Content-Type) должен быть равным "multipart/byteranges", а поля Content-Range должны содержаться в каждой части. Если "multipart/byteranges" не используется, поле заголовка Content-Length в ответе ДОЛЖНО соответствовать фактическому числу октетов (OCTETs), переданных в теле сообщения (message-body).

Кэш, который не поддерживает заголовки Range и Content-Range не должен кэшировать ответы с кодом состояния 206.

## 10.3 3xx - Перенаправление.

Этот класс кодов состояния указывает, что для выполнения запроса агенту пользователя необходимо придпринять дополнительное действие. Требуемое действие может быть выполнено агентом пользователя без взаимодействия с пользователем, тогда и только тогда, когда во втором запросе используется метод GET или HEAD. Агенту пользователя не следует автоматически перенаправлять запрос более 5 раз, так как такие переадресации обычно указывают бесконечный цикл.

### 10.3.1 300 Множественный выбор, Multiple Choices.

Запрошенный ресурс имеет несколько представлений, и можно использовать любое из перечисленных. Каждое представление имеет свое расположение и информацию для агента по управлению диалогом, представленную таким образом, что пользователь (или агент пользователя) может выбрать наиболее подходящее представление и перенаправить запрос к нему.

Если это не запрос HEAD, то ответу следует содержать объект, включающий список характеристик и адресов, из которого пользователь или агент пользователя может выбрать один наиболее подходящий. Формат объекта определяется медиатипом, указанным в поле заголовка Content-Type. В зависимости от формата и возможностей агента пользователя, выбор наиболее подходящего представления может выполняться автоматически. Однако, эта спецификация не определяет какого-либо стандарта для автоматического выбора.

Если сервер имеет представление по умолчанию (наиболее предпочтительное), то ему следует включить URL этого представления в поле Location; агенты пользователя МОГУТ использовать значение поля Location для автоматической переадресации. Этот ответ является кэшируемым, если не обозначено иного.

### 10.3.2 301 Постоянно перемещен, Moved Permanently.

Запрошенному ресурсу был назначен новый постоянный URI, и все будущие ссылки на этот ресурс следует выполнять, используя один из возвращенных URI. Клиентам с возможностями редактирования связей следует автоматически переопределить ссылки на запрашиваемый URI (Request-URI), используя одну или несколько новых ссылок, возвращенных сервером в тех местах, где это возможно. Этот ответ является кэшируемым, если не обозначено иного.

Если новый URI - это расположение, то ответу следует содержать URL в поле Location. Если метод запроса был не HEAD, то объекту ответа следует содержать короткое гипертекстовое примечание с гиперссылкой на новый (или новые) URI.

Если код состояния 301 был получен в ответ на запрос, отличный от GET или HEAD, агент пользователя не должен автоматически переназначать запрос, пока нет подтверждения пользователя, так как иначе условия запроса изменятся.

При автоматическом переназначении запроса POST после получения кода состояния 301, некоторые существующие HTTP/1.0 агенты пользователя ошибочно меняют метод запроса на GET.

### 10.3.3 302 Временно перемещен, Moved Temporarily.

Запрошенный ресурс временно находится под другим URI. Так как переадресация может быть изменена в любой момент, клиенту следует продолжать использовать запрашиваемый URI (Request-URI) в будущих запросах. Кэшируемость этого ответа зависит только от содержимого полей заголовка Cache-Control или Expires (если этих полей нет, то ответ не кэшируется).

Если новый URI - это расположение, то ответу следует содержать URL в поле Location. Если метод запроса был не HEAD, то объекту ответа следует содержать короткое гипертекстовое примечание с гиперссылкой на новый (или новые) URI.

Если код состояния 302 был получен в ответ на запрос, отличный от GET или HEAD, агент пользователя не должен автоматически переназначать запрос, пока нет подтверждения пользователя, так как иначе условия запроса изменятся.

При автоматическом переназначении запроса POST после получения кода состояния 302, некоторые существующие HTTP/1.0 агенты пользователя ошибочно меняют метод запроса на GET.

### 10.3.4 303 Смотреть другой, See Other.

Ответ на запрос может быть найден под другим URI и его следует запрашивать, используя метод GET для этого ресурса. Этот метод существует прежде всего для того, чтобы производить вывод данных активизированного методом POST сценария, используя перенаправление агента пользователя на указанный ресурс. Новый URI - это не ссылка, заменяющая первоначально запрошенный ресурс. Ответ с кодом состояния 303 не кэшируем, но ответ на второй (переназначенный) запрос может быть кэширован.

Если новый URI - это расположение, то ответу следует содержать URL в поле Location. Если метод запроса был не HEAD, то объекту ответа следует содержать короткое гипертекстовое примечание с гиперссылкой на новый (или новые) URI.

### 10.3.5 304 Не модифицирован, Not Modified.

Если клиент выполнил условный GET запрос, и доступ разрешен, но документ не изменился, то серверу следует ответить, используя этот код состояния. Ответ не должен содержать тела сообщения.

Ответ должен содержать следующие поля заголовка:

* Date
* ETag и/или Content-Location, если заголовок был бы послан в ответе с кодом состояния 200 на этот же запрос
* Expires, Cache-Control, и/или Vary, если значение поля (field-value) может отличаться от посланного в любом предыдущем ответе для такого же варианта

Если условный GET использует строгое сравнение кэша (strong cache validator), то ответу не следует содержать других заголовков объекта (entity-headers). Иначе (то есть, если условный GET использует слабое сравнение (weak validator)) ответ не должен содержать других заголовков объекта; это предотвращает несогласованности между кэшированными телами объектов (entity-bodies) и модифицированными заголовками.

Если ответ с кодом состояния 304 указывает объект, в настоящее время не кэшированный, то кэш должен игнорировать ответ и повторить запрос без условного выражения.

Если кэш использует полученный ответ с кодом состояния 304 для модифицикации вхождения кэша, кэш должен модифицировать вхождение так, чтобы отразить любые новые значения полей, данные в ответе.

Ответ с кодом состояния 304 не должен включать тела сообщения (message-body), и, таким образом, всегда завершается первой пустой строкой после полей заголовка.

### 10.3.6 305 Используйте прокси-сервер, Use Proxy.

Обращение к запрошенному ресурсу ДОЛЖНО производиться через прокси-сервер, указанный в поле Location. В поле Location указан URL прокси-сервера. Ожидается, что получатель повторит запрос через прокси-сервер.

### 10.4 4xx - Коды ошибок клиента.

Класс кодов состояния 4xx предназначен для случаев, когда клиент, возможно, допустил ошибку. За исключением ответа на запрос HEAD, серверу следует включить объект, содержащий объяснение ошибочной ситуации, и объяснение, является ли она временной или постоянной. Эти коды состояния применимы к любому методу запроса. Агентам пользователя следует показывать пользователю любой включенный объект.

Если клиент посылает данные, то реализации сервера, использующей TCP, следует гарантировать, что клиент подтвердил получение пакета(ов), содержащего ответ, прежде чем сервер закроет соединение. Если клиент продолжает посылать данные серверу после закрытия соединения, TCP стек сервера пошлет пакет сброса (RST) клиенту, а TCP стек клиента, в свою очередь, может стереть клиентские неподтвержденные входные буфера прежде, чем они будут прочитаны и интерпретированы приложением HTTP.

### 10.4.1 400 Испорченный Запрос, Bad Request.

Запрос не может быть понят сервером из-за malformed синтаксиса. Клиенту не следует повторять запрос без модификаций.

### 10.4.2 401 Несанкционированно, Unauthorized.

Запрос требует установления подлинности пользователя. Ответ должен включать поле заголовка WWW-Authenticate, содержащее вызов (challenge), применимый к запрошенному ресурсу. Клиент может повторить запрос с подходящим полем заголовка Authorization. Если запрос уже включает рекомендации установления подлинности (Authorization credentials) в поле Authorization, то ответ с кодом состояния 401 указывает, что в установлении подлинности этим рекомендациям отказано. Если ответ с кодом состояния 401 содержит тот же самый вызов, что и предшествующий ответ, а агент пользователя уже делал попытку установления подлинности по крайней мере один раз, то следует показать пользователю объект, который был дан в ответе, так как этот объект может включать актуальную диагностическую информацию..

### 10.4.3 402 Требуется оплата, Payment Required.

Этот код зарезервирован для будущего использования.

### 10.4.4 403 Запрещено, Forbidden.

Сервер понял запрос, но отказывается выполнять его. Установление подлинности (Authorization) не поможет, и запрос не должен быть повторен. Если метод запроса не HEAD и сервер желает указать, почему запрос не был выполнен, ему следует описать причину отказа в объекте. Этот код состояния обычно используется, когда сервер не желает указывать точную причину отказа, или когда никакой другой ответ не подходит.

### 10.4.5 404 Не найден, Not Found.

Сервер не нашел ничего соответствующего данному запрашиваемому URI (Request-URI). Никак не сообщается является ли такое положение временным или постоянным.

Если сервер не желает делать данную информацию доступной клиенту, то вместо этого кода состояния может использоваться код состояния 403 (Запрещено, Forbidden). Код состояния 410 (Удален, Gone) следует использовать, если сервер знает через некоторый внутренний конфигурируемый механизм, что старый ресурс более недоступен, но не знает нового адреса для пересылки.

### 10.4.6 405 Метод не допустим, Method Not Allowed.

Метод, определенный в строке запроса (Request-Line) не допустимо применять для ресурса, идентифицированного запрашиваемым URI (Request-URI). Ответ должен включать заголовок Allow, содержащий список допустимых методов для запрошенного ресурса.

### 10.4.7 406 Не приемлем, Not Acceptable.

Ресурс, идентифицируемый запросом, имеет возможности генерации только таких объектов ответа, которые имеют характеристики содержимого (content characteristics), не согласующиеся с заголовками приема (accept headers), представленными в запросе.

Если это был не запрос HEAD, то в ответ следует включить объект, содержащий список доступных характеристик объекта и адреса (locations), из которых пользователь или агент пользователя может выбрать наиболее подходящий. Формат объекта определеятся медиатипом, представленным в поле заголовка Content-Type. В зависимости от формата и возможностей агента пользователя, выбор наиболее подходящего варианта может выполняться автоматически. Однако эта спецификация не определяет никакого стандарта для автоматического выбора.

HTTP/1.1 серверы возвращают ответы, которые не приемлемы согласно заголовкам приема (accept headers), представленным в запросе. В некоторых случаях, это может быть даже предпочтительно по сравнению с посылкой ответа с кодом состояния 406. Агентам пользователя неплохо бы рассматривать заголовки поступившего ответа, чтобы определить, является ли он приемлемым. Если ответ недопустим, агенту пользователя следует временно остановиться, чтобы получить больше данных и спросить пользователя о дальнейших действиях.

### 10.4.8 407 Требуется установление подлинности через прокси-сервер, Proxy Authentication Required.

Этот код подобен коду 401 (Несанкционированно, Unauthorized), но указывает, что клиент должен сначала установить свою подлинность (authenticate) прокси-серверу. Прокси-сервер должен возвратить поле заголовка Proxy-Authenticate, содержащее вызов (challenge), применяемый прокси-сервером для запрошенного ресурса. Клиент может повторить запрос с подходящим полем заголовка Proxy-Authorization.

### 10.4.9 408 Истекло время ожидания запроса, Request Timeout.

Клиент не произвел запрос в течение времени, которое сервер готов ждать. Клиент может повторить запрос без модификаций позже.

### 10.4.10 409 Конфликт, Conflict.

Запрос не был выполнен из-за конфликта с текущим состоянием ресурса. Этот код допустим только в ситуациях, когда ожидается, что пользователь может решить конфликт и повторно передать запрос. Телу ответа следует содержать достаточное количество информации для пользователя, чтобы он мог распознать источник конфликта. В идеале, объект ответа должен включать достаточно информации для пользователя или агента пользователя для решения проблемы; однако это может быть не возможно, да и не требуется.

Конфликты, наиболее вероятно, будут возникать в ответ на запрос PUT. Если используется версификация, и объект, который должен быть помещен, включает изменения ресурса, которые находятся в противоречии со сделанными раньше каким-либо запросом (третьей стороны), сервер может использовать ответ с кодом состояния 409, чтобы показать, что он не может выполнить запрос. В этом случае, объекту ответа следует содержать список отличий двух версий в формате, определенном полем заголовка ответа Content-Type.

### 10.4.11 410 Удален, Gone.

Запрошенный ресурс больше не доступен на сервере, и нет никакого адреса для перенаправления запроса. Такое состояние следует рассматривать как постоянное. Клиентам с возможностями редактирования гиперсвязей следует удалить ссылки на запрашиваемый URI (Request-URI) после одобрения пользователем. Если сервер не знает, или не может определить, является ли такое положение постоянным или нет, то ему следует вместо этого кода использовать код состояния 404 (Не найден, Not Found). Этот ответ является кэшируемым, если не обозначено иного.

Ответ с кодом состояния 410 предназначен прежде всего для того, чтобы помочь в сопровождении WWW, уведомляя получателя, что ресурс преднамеренно недоступен и что владельцы сервера желают, чтобы удаленные связи, указывающие на этот ресурс были удалены. Такое случается в основном для ограниченных по времени, рекламных сервисов и для ресурсов, принадлежащих личностям, больше не занимающимся сайтом. Не обязательно отмечать все постоянно недоступные ресурсы как "удаленные" ("gone") или хранить запись в течение любого отрезка времени - это предоставляется на усмотрение владельца сервера.

### 10.4.12 411 Требуется длина, Length Required.

Сервер отказывается принимать запрос с неопределенным Content-Length. Клиент может повторить запрос, если добавит допустимое поле заголовка Content-Length, содержащее длину тела сообщения (message-body) в сообщении запроса.

### 10.4.13 412 Предусловие неверно, Precondition Failed.

Предусловие, представленное одним или несколькими полями заголовка запроса (request-header), оказалось ложным при проверке сервером. Этот код ответа позволяет клиенту поместить предусловия на текущую метаинформацию ресурса (данные полей заголовка) и, таким образом, предотвратить применение запрошенного метода к ресурсу, отличному от того, для которого предназначен метод.

### 10.4.14 413 Объект запроса слишком большой, Request Entity Too Large.

Сервер отказывается обрабатывать запрос, потому что объект запроса больше, чем сервер желает или способен обработать. Сервер может закрыть соединение, чтобы не дать клиенту возможность продолжить запрос.

Если это временное состояние, то серверу следует включить поле заголовка Retry-After для указания времени, через которое клиент может снова повторить запрос.

### 10.4.15 414 URI запроса слишком длинный, Request-URI Too Long.

Сервер отказывается обслуживать запрос, потому что запрашиваемый URI (Request-URI) длиннее, чем сервер желает интерпретировать. Это редкое состояние, которое, по всей вероятности, происходит только тогда, когда клиент неправильно преобразовал запрос POST к запросу GET с длинной информацией запроса, либо когда клиент попал в "черную дыру" URL перенаправления (например, перенаправленный URL префикс указывает на свой суффикс), или когда на сервер производится нападение клиентом, пытающимся эксплуатировать лазейки в секретности, имеющиеся в некоторых серверах, использующих буфера фиксированной длины для чтения или манипулирования с запрашиваемым URI (Request-URI).

### 10.4.16 415 Неподдерживаемый медиатип, Unsupported Media Type.

Сервер отказывается обслуживать запрос, потому что объект запроса находится в формате, не поддерживаемом запрошенным ресурсом для запрошенного метода.

### 10.5 5xx - Коды ошибок сервера.

Коды состояния, начинающиеся с цифры "5" указывают случаи, в которых сервер знает, что допустил ошибку или неспособен выполнить запрос. Отвечая на запрос, за исключением запроса HEAD, серверу следует включить объект, содержащий объяснение ошибочной ситуации и информацию, является ли это положение временным или постоянным. Агентам пользователя следует показывать пользователю любой включенный объект. Эти коды состояния применимы к любому методу запроса.

### 10.5.1 500 Внутренняя ошибка сервера, Internal Server Error.

Сервер столкнулся с непредвиденным условием, которое не позволяет ему выполнить запрос.

### 10.5.2 501 Не реализовано, Not Implemented.

Сервер не обладает функциональными возможностями, требуемыми для выполнения запроса. Сервер не распознает метод запроса и не реализует его ни для каких ресурсов.

### 10.5.3 502 Ошибка шлюза, Bad Gateway.

Сервер, действуя в качестве шлюза или прокси-сервера, получил недопустимый ответ от следующего сервера в цепочке запросов, к которому обратился при попытке выполнить запрос.

### 10.5.4 503 Сервис недоступен, Service Unavailable.

Сервер в настоящее время не способен обработать запрос из-за временной перегрузки или обслуживания сервера. Это состояние временное и после некоторой задержки сервер продолжит функционировать. Если известна продолжительность задержки, она может быть указана в заголовке Retry-After. Если Retry-After отсутствует в ответе, клиенту следует обрабатывать этот ответ как ответ с кодом 500.

Существование кода состояния 503 не подразумевает, что сервер должен использовать его, когда перегружен. Некоторые сервера могут просто закрывать соединение.

### 10.5.5 504 Истекло время ожидания от шлюза, Gateway Timeout.

Сервер, действуя в качестве шлюза или прокси-сервера, не получил своевременного ответа от следующего сервера в цепочке запросов, к которому обратился при попытке выполнить запрос.

### 10.5.6 505 Не поддерживаемая версия HTTP, HTTP Version Not Supported.

Сервер не поддерживает или отказывается поддерживать версию протокола HTTP, которая используется в сообщении запроса. Сервер указывает, что не способен или не желает выполнять запрос, используя ту же самую major версию, что и клиент (кроме этого сообщения). Ответу следует включать объект, описывающий, почему эта версия не поддерживается, и какие другие протоколы поддерживаются этим сервером.

# 11 Установление подлинности доступа (Access Authentication).

Для установления подлинности доступа HTTP предоставляет простой механизм вызовов-ответов (challenge-response), который может использоваться сервером для вызова (challenge) клиентского запроса, а клиентом для предоставления опознавательной информации (authentication information). Для этого используется расширяемая, не чувствительная к регистру лексема идентификации схемы установления подлинности (authentication scheme) и список пар атрибут-значение (attribute-value), разделенных запятыми. Пары представляют параметры, необходимые для установления подлинности при использовании этой схемы.

auth-scheme = token

auth-param = token "=" quoted-string

Сообщение ответа с кодом 401 (Несанкционирован, Unauthorized) используется первоначальным сервером для вызова (challenge) установления подлинности (authorization) агентом пользователя. Этот ответ должен содержать поле заголовка WWW-Authenticate, содержащее по крайней мере один вызов (challenge), применимый к запрошенному ресурсу.

challenge = auth-scheme 1\*SP realm \*( "," auth-param )

realm = "realm" "=" realm-value realm-value = quoted-string

Атрибут области (realm) (не чувствительный к регистру) необходим во всех схемах установления подлинности, которые выдают вызов (challenge). Значение аттрибута realm (чувствительное к регистру), в комбинации с каноническим корневым URL (canonical root URL) сервера, к которому обращен запрос, идентифицирует защищенную область (protection space). Эти области позволяют разбивать защищенные ресурсы сервера на множество областей, каждая из которых имеет собственную схему установления подлинности и/или базу данных авторизации (authorization database). Значение realm - это строка, вообще говоря назначенная первоначальным сервером, которая может иметь дополнительную семантику, специфическую для схемы установления подлинности.

Агент пользователя, который хочет доказать свою подлинность серверу, обычно, но не обязательно, может это сделать после получения ответа с кодом состояния 401 или 411, включив поле заголовка Authorization в запрос. Значение поля Authorization состоит из рекомендаций (credentials), которые содержат информацию установления подлинности (authentication information) агента пользователя для области (realm), в которой лежит запрошенный ресурс.

credentials = basic-credentials | auth-scheme #auth-param

Область (domain), над которой рекомендации (credentials) могут автоматически применяться агентом пользователя, определена областью защиты (protection space). Если подлинность была установлена предшествующим запросом, то эти же рекомендации (credentials) могут использоваться многократно во всех других запросах внутри этой области защиты (protection space) в течении времени, определенного схемой установления подлинности, параметрами, и/или установками пользователя. Если схемой установления подлинности не определено иного, то одиночная область защиты (protection space) не может простираться за пределы области сервера (the scope of its server).

Если сервер не желает принимать рекомендации (credentials), посланные в запросе, то ему следует возвратить ответ с кодом 401 (Несанкционирован, Unauthorized). Ответ должен включать поле заголовка WWW-Authenticate, содержащее (возможно новый) вызов (challenge), применимый к запрошенному ресурсу, и объект, объясняющий отказ.

Протокол HTTP не ограничивает возможности приложений по установлению подлинности доступа использованием этого простого механизма вызовов-ответов (challenge-response). можно использовать дополнительные механизмы, такие как шифрование на транспортном уровне или формирование пакета сообщения (message encapsulation) с дополнительными полями заголовка, определяющими информацию установления подлинности. Однако эти дополнительные механизмы не определены данной спецификацией.

Прокси-сервера должны быть полностью прозрачны для установления подлинности агента пользователя. То есть они должны пересылать заголовки WWW-Authenticate и Authorization нетронутыми.

HTTP/1.1 позволяет клиенту передавать информацию установления подлинности для и от прокси-сервера посредством заголовков Proxy-Authenticate и Proxy-Authorization.

## 11.1 Базовая схема установления подлинности (Basic Authentication Scheme).

"Базовая" схема установления подлинности основана на том, что агент пользователя должен доказывать свою подлинность при помощи идентификатора пользователя (user-ID) и пароля (password) для каждой области (realm). Значению области (realm) следует быть непрерывной (opaque) строкой, которую можно проверять только на равенство с другими областями на этом сервере. Сервер обслужит запрос, только если он может проверить правильность идентификатора пользователя (user-ID) и пароля (password) для защищенной области (protection space) запрошенного URI (Request-URI). Никаких опциональных опознавательных параметров нет.

После получения запроса на URI, находящийся в защищаемой области (protection space), сервер может ответить вызовом (challenge), подобным следующему:

WWW-Authenticate: Basic realm="WallyWorld"

где "WallyWorld" - строка, назначенная сервером, которая идентифицирует область защиты запрашиваемого URI (Request-URI).

Чтобы получить права доступа, клиент посылает идентификатор пользователя (userid) и пароль (password), разделенные одним символом двоеточия (":"), в base64-кодированной строке рекомендаций (credentials).

basic-credentials = "Basic" SP basic-cookie

basic-cookie = <base64-кодированный [7] user-pass, за исключением не ограниченных 76 имволами в строке>

user-pass = userid ":" password

userid = \*<TEXT не содержащий ":">

password = \*TEXT

Userid может быть чувствителен к регистру.

Если агент пользователя хочет послать идентификатор пользователя (userid) "Aladdin", и пароль (password) "open sesame", он будет использовать следующее поле заголовка:

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

## 11.2 Обзорная схема установления подлинности (Digest Authentication Scheme) [1].

# 13 Кэширование в HTTP.

HTTP обычно используется для распределенных информационных систем, эффективность которых может быть улучшена при использовании кэшированных ответов. Протокол HTTP/1.1 включает ряд элементов, предлагаемых как возможная реализация кэширования. Так как эти элементы отличаются от других аспектов протокола, и так как они взаимодействуют друг с другом, то полезно будет описать основы кэширования в HTTP отдельно от детализированных описаний методов, заголовков, кодов состояния, и прочего.

Цель кэширования в HTTP/1.1 состоит в том, чтобы устранить потребность посылки запросов во многих случаях, и устранить потребность посылки полных ответов в других случаях. Кэширование уменьшает число пересылок информации по сети, требуемых для многих действий; мы используем для этой цели механизм "устаревания" ("expiration"). Кэширование снижает требования к пропускной способности сети; мы используем для этой цели механизм "проверки достоверности" ("validation").

Требования эффективности, доступности, и раздельного функционирования требуют, чтобы цель семантической прозрачности была отодвинута на второй план. Протокол HTTP/1.1 позволяет первоначальным серверам, кэшам, и клиентам явно ограничивать прозрачность в случае необходимости. Однако, так как непрозрачное функционирование может ввести в заблуждение неопытных (non-expert) пользователей, и может быть несовместимо с некоторыми серверными приложениями (такими как заказ товаров), протокол требует, чтобы прозрачность ослаблялась

* Только явным запросом на уровне протокола если ослабление вызывается клиентом или первоначальным сервером
* Только с явным предупреждением конечного пользователя если ослабление вызывается кэшем или клиентом

Таким образом протокол HTTP/1.1 предоставляет следующие важные элементы:

1. Возможности протокола, которые обеспечивают полную семантическую прозрачность когда это требуется всем сторонам.
2. Возможности протокола, которые позволяют первоначальному серверу или агенту пользователя явно запрашивать непрозрачное функционирование и управлять им.
3. Возможности протокола, которые позволяют кэшу присоединять к ответам предупреждения о том, что запрошенный уровень семантической прозрачности не сохранен.

Базовый принцип состоит в том, что клиенты должны иметь возможность обнаружить любое потенциальное ослабление семантической прозрачности.

Реализатор сервера, кэша или клиента может столкнуться с проблемами, явно не обсужденными в этой спецификации. Если решение может воздействовать на семантическую прозрачность, реализатор должен принимать решения в сторону сохранения прозрачности, если осторожный и полный анализ не показывает значительных выгод раздельного функционирования.

## 13.1 Общая информация о кэшировании.

### 13.1.1 Правильность кэша.

Правильный кэш должен отвечать на запрос наиболее современным ответом, соответствующим запросу, из хранимых кэшем который удовлетворяет одному из следующих условий:

1. Он был проверен на эквивалентность ответу, который возвратил первоначальный сервер, повторно подтверждая достоверность;
2. Он "достаточно свеж" ("fresh enough"). По умолчанию это означает, что он удовлетворяет наименьшему из ограничительных требований свежести клиента, сервера и кэша; если так определено первоначальным сервером, то это - требование свежести единственно первоначального сервера.
3. Он включает предупреждение, если свежесть запрошена клиентом или первоначальный сервер нарушен.
4. Он соответствует сообщению ответа с кодом состояния 304 (не модифицирован, Not Modified), 305 (используйте прокси-сервер, Use Proxy), или ошибочному ответу (4xsx или 5xx).

Если кэш не может связаться с первоначальным сервером, то правильному кэшу следует отвечать как описано выше, если ответ может быть правильно обслужен из кэша; иначе необходимо возвратить ошибку или предупредить о том, что имеется отказ связи.

Если кэш получает ответ (либо весь ответ, либо ответ с кодом состояния 304 (не модифицирован, Not Modified)), который может быть нормально передан запрашивающему клиенту, и полученный ответ устарел, то кэшу следует переслать его запрашивающему клиенту не добавляя нового заголовка Warning (Предупреждение) (но не удаляя существующие заголовки Warning). Кэшу не следует пытаться повторно проверить достоверность ответа просто потому, что тот ответ устарел при передаче; это могло бы привести к бесконечному циклу. Агент пользователя, который получает просроченный ответ без Warning может показать пользователю предупреждение.

### 13.1.2 Предупреждения.

Всякий раз, когда кэш возвращает ответ, который не является ни непосредственным (first-hand), ни "достаточно свежим" (в смысле условия 2 раздела 13.1.1), он должен присоединить предупреждение об этом, используя заголовок ответа Warning. Это предупреждение позволяет клиентам предпринимать соответствующие действия.

Предупреждения могут использоваться для других целей, как связанных с кэшированием, так и не связанных. Использование предупреждений, а не ошибочных кодов состояния, отличает эти ответы от истинных отказов.

Предупреждения кэшируемы всегда, так как не ослабляют прозрачность ответа. Это означает, что предупреждения могут быть переданы HTTP/1.0 кэшам без опасений; такие кэши просто передадут предупреждение дальше как заголовок объекта в ответе.

Предупреждения - это предопределенные числа от 0 до 99. Эта спецификация определяет коды, и значения каждого определенного в настоящее время предупреждения, позволяя клиенту или кэшу предпринимать самостоятельные действия в некоторых (но не во всех) случаях.

Предупреждения также содержат текст предупреждения. Текст может быть на любом соответствующем естественном языке (возможно на основании заголовков Accept клиента), и включать опциональную индикацию используемого набора символов.

К ответу могут быть присоединены несколько предупреждений (как первоначальным сервером, так и кэшем), включая несколько предупреждений с одиннаковыми кодовыми номерами. Например, сервер может добавлять одно и тоже предупреждение как к английским текстам, так и к баскским.

Если к ответу присоединено несколько предупреждений, то может быть практически не целесообразно или не приемлемо показать все из них пользователю. Эта версия HTTP не определяет строгих приоритетных правил для определения, какие из предупреждений отображать и в каком порядке, но предлагает некоторую эвристику.

### 13.1.3 Механизмы управления кэшем (Cache-control Mechanisms).

Основные механизмы кэша в HTTP/1.1 (указанные сервером время устаревания (expiration time) и указатель достоверности (validator)) - неявные директивы кэшу. Возможны случаи, в которых сервер или клиент должен обеспечить явные директивы HTTP кэшу. Мы используем для этой цели заголовок Cache-Control.

Заголовок Cache-Control позволяет клиенту или серверу передавать ряд директив как в запросах, так и в ответах. Эти директивы обычно отменяют испоьзуемые по умолчанию кэширующие алгоритмы. В качестве общего правила: если имеется очевидный конфликт между значениями заголовка, то должна применяться наиболее ограничивающая интерпретация (то есть та, которая, наилучшим образом сохранит семантическую прозрачность). Однако в некоторых случаях директивы управления кэшем (Cache-Control) явно указывают ослабление уровня семантической прозрачности (например, "максимально-просроченный" ("max-stale") или "общий" ("public")).

### 13.1.4 Явные предупреждения агента пользователя.

Многие агенты пользователя делают возможным для пользователей отменить основные механизмы кэширования. Например агент пользователя может позволить пользователю указать такое поведение, при котором кэшированные объекты (даже явно просроченные) никогда не проверяются на достоверность (are never validated). Либо агент пользователя мог бы добавлять "Cache-Control: max-stale=3600" к каждому запросу. Пользователю следует явно запрашивать как непрозрачное поведение, так и поведение, которое неверно приводит к неэффективному кэшированию.

Если пользователь отменил основные механизмы кэширования, агент пользователя должен явно информировать пользователя всякий раз, когда происходит отображение информации, которая может не удовлетворять требованиям прозрачности сервера (в частности если известно, что отображаемый объект просрочен). Так как протокол обычно позволяет агенту пользователя определить, просрочены ответы или нет, то индикация необходима только тогда, когда это фактически происходит. Это может отображаться не только диалоговым окном, но и иконкой (например, изображением гниющей рыбы) или другим визуальным индикатором.

Если пользователь отменил механизмы кэширования таким образом, что неправильно уменьшил эффективность кэшей, агент пользователя должен непрерывно индицировать (например, изображением горящей купюры) то, что пользователь неосторожно потребляет ресурсы или страдает от чрезмерного времени ожидания.

### 13.1.5 Исключения из правил и предупреждений.

В некоторых случаях, оператор кэша может сконфигурировать его таким образом, чтобы он возвращал просроченные ответы, даже если они не запрашиваются клиентами. Это решение не должно быть сделано с легкостью, но может быть необходимо по причинам доступности или эффективности, особенно, когда кэш имеет прохое соединение с первоначальным сервером. Всякий раз, когда кэш возвращает просроченный ответ, он должен пометить его (используя заголовок Warning). Это позволяет программному обеспечению клиента предупреждать пользователя, что возможно имеется потенциальная проблема.

Это также позволяет агенту пользователя предпринимать шаги для получения непосредственного (first-hand) или свежего ответа. По этой причине, кэшу не следует возвращать просроченный ответ, если клиент явно запрашивает непосредственный или свежий, за исключением случаев, когда это невозможно выполнить по техническим или стратегическим причинам.

### 13.1.6 Контролируемое клиентом поведение.

В то время как первоначальный сервер (и меньшей степени промежуточные кэши с их вкладом в возраст ответа) является первичным источником информации об устаревании, в некоторых случаях клиенту может быть необходимо управлять решением кэша о том, возвращать ли кэшированный ответ, не проверяя его достоверность. Клиенты делают это используя несколько директив заголовка управления кэшем (Cache-Control).

Запрос клиента может определять максимальный возраст неутвержденного ответа, который он желает получить; указывая ноль он вынуждает кэш(и) перепроверить достоверность всех ответов. Клиент может также определить минимальное время которое должно пройти до того, как ответ устареет. Обе этих опции увеличивают ограничения на поведение кэшей, и, таким образом, не могут далее ослаблять уровень семантической прозрачности кэша.

Клиент может также указать, что он примет ответы, просроченные до некоторого определенного срока. Это ослабляет ограничения на кэши, и, таким образом, может нарушить ограничения на семантическую прозрачность, определенные первоначальным сервером, но может быть необходимо для поддержки раздельного функционирования, или высокой доступности при плохой связи.

## 13.2 Модель устаревания.

### 13.2.1 Устаревание, указанное сервером.

HTTP кэширование работает лучше всего тогда, когда кэши могут полностью избежать запросов к первоначальному серверу. Первичный механизм избавления от запросов - когда сервер происхождения обеспечивает явное время устаревания в будущем, указывая, что ответ может использоваться для удовлетворения последующих запросов. Другими словами, кэш может возвращать свежий ответ без контакта с сервером.

Мы ожидаем, что серверы будут назначать явное время устаревания ответов будучи убеждены, что объект, вероятно, не будет изменен семантически значимым способом до истечения этого времени. Это обычно сохраняет семантическую прозрачность, если время устаревания тщательно выбрано сервером.

Механизм устаревания применяется только к ответам, полученным из кэша а не к непосредственным ответам, немедленно посланным запрашивающему клиенту.

Если первоначальный сервер хочет вынудить семантически прозрачный кэш проверять достоверность каждого запроса, он может явно указать время устаревания в прошлом. Это означает, что ответ всегда просрочен, и, таким образом, кэш должен проверять его достоверность перед использованием для последующих запросов.

Если первоначальный сервер хочет вынудить любой HTTP/1.1 кэш, независимо от того, как он сконфигурирован, проверять достоверность каждого запроса, он должен использовать директиву Cache-Control "must-revalidate".

Серверы указывают явное время устаревания, используя как заголовок Expires, так и директиву max-age заголовка Cache-Control.

Время устаревания не может использоваться чтобы вынудить агента пользователя обновить отображенную на экране информацию или перезагрузить ресурс; оно имеет значение только в применении к механизмам кэширования, и этим механизмам требуется только проверяет состояние устаревания ресурса, когда происходит новый запрос этого ресурса.

### 13.2.2 Эвристическое устаревание.

Так как первоначальные серверы не всегда указывают явное время устаревания, то HTTP кэши обычно назначают эвристическое время устаревания, используя алгоритмы, которые используют значения других заголовков (таких как время последней модификации (Last-Modified)) для оценки вероятного времени устаревания. Спецификация HTTP/1.1 не обеспечивает специфических алгоритмов, но налагает ограничения в виде оценки наихудшей погрешности их работы. Так как эвристическое временя устаревания может ставить под угрозу семантическую прозрачность, то они должны использоваться осторожно, и мы поощряем первоначальные серверы указывать явное время устаревания насколько возможно часто.

### 13.2.3 Вычисление возраста.

Чтобы знать, является ли содержащийся в кэше объект свежим, кэш должен знать, превышает ли его возраст срок свежести.

Хосты, которые используют HTTP, а в особенности хосты, на которых выполняются первоначальные серверы и кэши, должны использовать NTP или любой подобный протокол для синхронизации их часов с глобальным точным временем.

HTTP/1.1 требует, чтобы первоначальные серверы посылали в каждом ответе заголовок Date, предоставляющий время, когда был сгенерирован ответ.

HTTP/1.1 использует заголовок ответа Age для передачи информации о возрасте между кэшами. Значение заголовка Age является оценкой отправителя количества времени, прошедшего с момента, когда ответ был сгенерирован на первоначальном сервере. В случае кэшированного ответа, который был повторно подтвержден первоначальным сервером, значение Age базируется на времени перепроверки достоверности, а не на времени первоначального ответа.

В сущности, значение Age - сумма времен которые ответ находился в каждом из кэшей по пути от первоначального сервера и времен его передачи по сети.

Возраст ответа может быть вычислен двумя совершенно независимыми способами:

1. "Сейчас" минус date\_value, если локальные часы хорошо синхронизированы с часами первоначального сервера. Если результат отрицателен, результат заменяется нулем.
2. age\_value, если все кэши по пути ответа реализуют HTTP/1.1.

При условии, что мы имеем два независимых способа вычисления возраста ответа при его получении, мы можем объединить их как

Corrected\_received\_age = max("сейчас" - date\_value, age\_value)

и пока часы у нас синхронизированы, а все кэши на пути ответа - HTTP/1.1, получаем надежный (консервативный) результат.

Эта поправка применяется в каждом HTTP/1.1 кэше по пути следования ответа, так что если на пути встретится HTTP/1.0 кэш, то полученный возраст будет вычислен правильно, если часы этого кэша хорошо синхронизированы.

Из-за задержек, обусловленных сетью, некоторое значительное время может пройти между моментом, когда сервер сгенерировал ответ и моментом, когда он был получен следующим внешним кэшем или клиентом. Игнорирование этой задержки, приводить к неправильно низким возрастам.

Если запрос, который привел к возвращенному значению Age, должно быть был инициализирован до порождения значений Age, мы можем исправлять наложенные сетью задержки, делая запись времени, когда был сгенерирован запрос. Таким образом, когда получено значение Age, оно должно быть интерпретировано относительно времени, когда был сгенерирован запрос, а не относительно времени, когда был получен ответ. Этот алгоритм приводит к консервативному поведению независимо от того, какова задержка. Вычисляем:

corrected\_initial\_age = corrected\_received\_age + ("сейчас" - request\_time)

Где "request\_time" - время (согласно локальным часам), когда был послан запрос, вызвавший данный ответ.

Резюме алгоритма вычисления возраста полученного кэшем ответа:

/\* \* age\_value \* - это значение Age: заголовок, полученный кэшем в \* этом ответе. \* date\_value \* - это значение Date первоначального сервера: заголовок \* request\_time \* - это (локальное) времея, когда кэш сделал запрос, \* который вызвал этот кэшируемый ответ \* response\_time \* - это (локальное) время, когда кэш получил ответ \* now \* - текущее (локальное) время \*/ apparent\_age = max(0, response\_time - date\_value); corrected\_received\_age = max(apparent\_age, age\_value); response\_delay = response\_time - request\_time; corrected\_initial\_age = corrected\_received\_age + response\_delay; resident\_time = now - response\_time; current\_age = corrected\_initial\_age + resident\_time;

Когда кэш посылает ответ, он должен добавить к corrected\_initial\_age количество времени, которое ответ содержался в нем. Он должен затем передать этот общий возраст, используя заголовок Age, следующему получающему кэшу.

### 13.2.4 Вычисление устаревания.

Чтобы решить,является ли ответ свежим или просроченным, мы должны сравнить срок его службы с возрастом. Возраст вычисляется по алгоритму, описанному в разделе 13.2.3, а этот раздел описывает, как вычислять срок службы, и определять, не устарел ли ответ. В следующем описании значения могут представляться в любой форме, подходящей для арифметических действий.

Термин "expires\_value" представляет значение заголовка Expires. Термин "max\_age\_value" применяется для указания значения числа секунд, представленного директивой max-age заголовка Cache-Control ответа.

Директива max-age имеет приоритет над Expires. Таким образом если в ответе присутствует max-age, то вычисления просты:

freshness\_lifetime = max\_age\_value

В других случаях, когда в ответе присутствует Expires, вычисления таковы:

freshness\_lifetime = expires\_value - date\_value

Ни одно из этих вычислений не уязвимо от рассинхронизирования часов, так как вся информация поступает от первоначального сервера.

Если ни Expires, ни Cache-Control: max-age не встречаются в ответе, и ответ не содержит других ограничений кэширования, то кэш может вычислить срок службы, используя эвристику. Если это значение больше 24-х часов, то кэш должен присоединять Warning 13 к любому ответу, чей возраст больше 24-х часов, если такое предупреждение еще не было добавлено.

Также если ответ имеет время последнего изменения Last-Modified, то эвристическому значению времени устаревания следует принимать значение не более некоторой части временного интервала, прошедшего этого времени. Типичная значение этой части могло бы быть 10%.

Вычислить, истек ли ответ, совершенно просто:

response\_is\_fresh = (freshness\_lifetime > current\_age)

### 13.2.5 Устранение противоречий в значениях устаревания.

В том случае, когда значения устаревания назначены оптимистично, возможен случай, когда два кэша содержат различные значения свежести одного и того же ресурса.

Если клиент, выполняющий поиск, получает не непосредственный ответ на запрос, который уже был свежим в его собственном кэше, а заголовок Date в существующем вхождении кэша более свежий, чем Date нового ответа, то клиент может игнорировать ответ. Если он игнорирует ответ, то он может повторить запрос с директивой "Cache-Control: max-age=0", вызывая проверку первоначальным сервером.

Если кэш имеет два свежих ответа на запрос одного и того же представления, но с различным указателями достоверности, то он должен использовать тот из них, у которого заголовок Date более свеж. Такая ситуация может вознкнуть либо когда кэш объединяет ответы других кэшей, либо когда клиент запросил перезагрузку или перепроверку достоверности очевидно свежего вхождения кэша.

### 13.2.6 Устранение противоречий между несколькими ответами.

В силу того, что клиент может получать ответы многими путями, возможна ситуация, когда кеш получает один поток ответов через один набор кэшей, а другой поток ответов через другой набор кэшей. В таком случае клиент может получать ответы в порядке, отличном от того, в котором их посылал первоначальный сервер. Мы хотели бы, чтобы клиент использовал наиболее свежий ответ, даже если ранее полученные ответы все еще очевидно свежи.

Ни метка объекта ни значение устаревания не можгут быть использованы для упорядочения ответов, так как, возможно, более поздний ответ преднамеренно несет более раннее время устаревания. Однако, спецификация HTTP/1.1 требует передачи заголовков Date в каждом ответе, и значения Date упорядочивают степень детализации до одной секунды.

Когда клиент пытается повторно проверить достоверность вхождения кэша, и ответ, который он получает, содержит заголовок Date, который старше, чем у существующего вхождения, то клиенту следует повторить запрос без изменений, но включить

Cache-Control: max-age=0

чтобы вынудить промежуточные кэши проверить достоверность их копий непосредственно первоначальным сервером, или

Cache-Control: no-cache

чтобы вынудить промежуточные кэши получить новую копию с первоначального сервера.

Если значения Date равны, то клиент может использовать любой ответ (или может, если он чрезвычайно предусмотрительный, запросить новый ответ). Серверы не должны полагаться на то, что клиенты способны выбрать между ответами, порожденными в течение одной секунды, если их времена устаревания накладываются.

## 13.3 Модель проверки достоверности (validation model).

Если кэш имеет просроченное вхождение, которое он хотел бы использовать в качестве ответа на запрос клиента, он сначала должен свериться с первоначальным сервером (или с промежуточным кэшем, обладающим свежим ответом) чтобы узнать, является ли вхождение кэша все еще пригодным для использования. Мы называем это "проверкой достоверности" вхождения кэша. Так как мы не хотим оплачивать излишнюю передачу полного ответа, когда кэшируемый нас устраивает, и так как мы не хотим оплачивать лишнюю передачу ответа туда и обратно, когда кэшируемое вхождение нас не устраивает, протокол HTTP/1.1 поддерживает использование условных методов.

Ключевые возможности протокола для обеспечения условных методов - те, что имеют отношение к "указателям достоверности (validators) кэша". Когда первоначальный сервер генерирует полный ответ, он присоединяет к нему некоторый указатель достоверности (validator), который сохраняется во вхождении кэша. Когда клиент (агент пользователя или кэш прокси-сервера) делает условный запрос на ресурс, для которого он имеет вхождение кэша, он включает связанный указатель достоверности (validator) в запрос.

Затем сервер проверяет полученный указатель достоверности (validator) на соответствие текущему указателю достоверности (validator) объекта, и, если он соответствует, то сервер посылает ответ со специальным кодом состояния (обычно 304 (не модифицирован), не содержащий тела объекта. В противном случае, он возвращает полный ответ (включая тело объекта). Таким образом, мы избегаем передачи полного ответа в случае соответствия указателя достоверности (validator), и избегаем дополнительной передачи туда-обратно в случае несоответствия.

В HTTP/1.1, условный запрос выглядит точно также, как нормальный запрос того же ресурса, за исключением того, что он содержит специальный заголовок (который включает указатель достоверности), который неявно превращает метод (обычно, GET) в условный.

Протокол содержит как положительные, так и отрицательные условия на указатели достоверности. То есть возможно запросить как метод, который будет выполнен только в случае соответствия указателя достоверности, так и метод, который будет выполнен только в случае несоответствия указателя достоверности.

Ответ, который не содержит указателя достоверности, все же может кэшироваться и обслуживаться из кэша пока не устареет, если это явно не запрещено директивой Cache-Control. Однако, кэш не может производить условный поиск, если он не имеет указателя достоверности объекта, что означает, что ответ не будет регенерирован после того, как устареет.

# Библиографический список

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Номер RFC** | **Название** | **Авторы** | **Дата** | **Статус** | **Примечания** |
| 822 | STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES | David H. Crocker | August 13, 1982 | Proposed Standard | Obsoletes: RFC #733 |
| 850 | Standard for Interchange of USENET Messages | RFC team | June 1983 | Informational |  |
| 1036 | Standard for Interchange of USENET Messages | M. Horton, R. Adams | December 1987 | Proposed Standard | Obsoletes: RFC-850 |
| 1123 | Requirements for Internet Hosts -- Application and Support | Robert Braden | October 1989 | Proposed Standard |  |
| 1738 | Uniform Resource Locators (URL) | Tim Berners-Lee,  Larry Masinter,  Mark McCahill | December 1994 | Proposed Standard |  |
| 1766 | Tags for the Identification of Languages | H. Alvestrand | March 1995 | Standards Track |  |
| 1808 | Relative Uniform Resource Locators | Roy T. Fielding | June 1995 | Proposed Standard |  |
| 1867 | Form-based File Upload in HTML | E. Nebel, L. Masinter | November 1995 | Experimental |  |
| 1900 | Renumbering Needs Work | Brian E. Carpenter, Yakov Rekhter | February 1996 | Informational |  |
| 1945 | Hypertext Transfer Protocol -- HTTP/1.0 | T. Berners-Lee, R. Fielding  , H. Frystyk | May 1996 | Informational |  |
| 1950 | ZLIB Compressed Data Format Specification version 3.3 | P. Deutsch, J-L. Gailly | May 1996 | Informational |  |
| 1952 | GZIP file format specification version 4.3 | P. Deutsch | May 1996 | Informational |  |
| 2048 | Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures | N. Freed, J. Klensin, J. Postel | November 1996 | Best Current Practice | Obsoletes: 1521, 1522, 1590 |
| 2068 | Hypertext Transfer Protocol -- HTTP/1.1 | R. Fielding,  J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee | January 1997 | Standards Track |  |
| 2069 | An Extension to HTTP : Digest Access Authentication | J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart | January 1997 | Standards Track |  |