Internet Engineering Task Force (IETF)
Request for Comments: 9112
STD: 99
Obsoletes: 7230
Category: Standards Track
ISSN: 2070-1721

R. Fielding, Ed.
Adobe
M. Nottingham, Ed.
Fastly
J. Reschke, Ed.
greenbytes

June 2022

HTTP/1.1

Аннотация

Протокол передачи гипертекста (Hypertext Transfer Protocol или HTTP) - это протокол прикладного уровня без поддержки состояний, предназначенный для распределенных систем групповой работы с гипертекстовой информацией. Этот документ задаёт синтаксис сообщений HTTP/1.1, разбор сообщений, управление соединениями и связанные с этим вопросы безопасности.

Этот документ отменяет некоторые части RFC 7230.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9112.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.
[3]В оригинале - IETF Contributions. *Прим. перев.*
[4]В оригинале - IETF Standards Process. *Прим. перев.*

1. Введение

HTTP - это протокол прикладного уровня «запрос-отклик» без поддержки состояний, использующий расширяемую семантику и самоописательные сообщения для гибкого взаимодействия с сетевыми гипертекстовыми информационными системами. HTTP/1.1 определён в:

- этом документе;
- HTTP Semantics [HTTP];
- HTTP Caching [CACHING].

Этот документ описывает способы передачи семантики HTTP с применением синтаксиса сообщений HTTP/1.1, обрамление (framing) и механизмы управления соединениями. Цель состоит в задании полного набора требваний к синтаксическим анализаторам (parser) HTTP/1.1 и посредникам при пересылке сообщений.

Документ отменяет части RFC 7230, относящиеся к сообщениям HTTP/1.1 и управлению соединениями. Перечени отличий приведён в Приложении C.3. Остальные части RFC 7230 отменены документом HTTP Semantics [HTTP].

1.1. Уровни требований

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

Критерии соответствия и обработка ошибок рассмотрены в разделе 2 [HTTP].

1.2. Синтаксические обозначения

В этом документе применяется расширенная нотация Бэкуса-Наура (Augmented Backus-Naur Form или ABNF) [RFC5234] , дополненная обозначениями для строк с учётом регистра символов, определённых в [RFC7405]. Используется также расширение списков, определённое в параграфе 5.6.1 [HTTP], которое позволяет компактно определять списки, разделяемых запятыми элементов с использованием оператора # (подобно указанию повтора оператором *). В Приложении A приведена сводная грамматика со всеми расширениями операторов списка, добавленными в ABNF. По традиции имена правил ABNF с префиксом obs- обозначают устаревшие правила, указанные по историческим причинам.

Указанные далее базовые правила включены по ссылкам, как задано в Приложении B.1 к [RFC5234]: ALPHA (буквы), CR (возврат каретки), CRLF (CR LF), CTL (управление), DIGIT (десятичные цифры 0-9), DQUOTE (двойные кавычки), HEXDIG (шестнадцатеричные цифры 0-9/A-F/a-f), HTAB (горизонтальная табуляция), LF (перевод строки), OCTET (любая 8-битовая последовательность данных), SP (пробел), VCHAR (любой печатный символ US-ASCII).

Ниже указаны правила, заданные в [HTTP].

```
BWS           = <BWS, см. [HTTP], параграф 5.6.3>
OWS           = <OWS, см. [HTTP], параграф 5.6.3>
RWS           = <RWS, см. [HTTP], параграф 5.6.3>
absolute-path = <absolute-path, см. [HTTP], параграф 4.1>
field-name    = <field-name, см. [HTTP], параграф 5.1>
field-value   = <field-value, см. [HTTP], параграф 5.5>
obs-text      = <obs-text, см. [HTTP], параграф 5.6.4>
quoted-string = <quoted-string, см. [HTTP], параграф 5.6.4>
token         = <token, см. [HTTP], параграф 5.6.2>
transfer-coding = <transfer-coding, см. [HTTP], параграф 10.1.4>
```

Ниже указаны правила, заданные в [URI].

```
absolute-URI  = <absolute-URI, см. [URI], параграф 4.3>
authority     = <authority, см. [URI], параграф 3.2>
uri-host      = <host, см. [URI], параграф 3.2.2>
port          = <port, см. [URI], параграф 3.2.3>
query         = <query, см. [URI], параграф 3.4>
```

2. Сообщение

Клиенты и серверы HTTP/1.1 взаимодействуют путём обмена сообщениями. Базовая терминология и основные концепции HTTP представлены в разделе 3 [HTTP].

2.1. Формат сообщения

Сообщение HTTP/1.1 состоит из стартовой строки (start-line), за которой следуют символы CRLF и последовательность октетов в формате, похожем на формат сообщений Internet (Internet Message Format) [RFC5322] - строки полей заголовков (необязательно), называемые далее просто заголовками или разделом заголовков, пустая строка, завершающая раздел заголовков и (необязательное) тело сообщения.

```
HTTP-message   = start-line CRLF
                 *( field-line CRLF )
                 CRLF
                 [ message-body ]
```

Сообщение может быть запросом клиента к северу или откликом сервера. Синтаксически эти типы сообщения различаются лишь стартовой строкой (request-line для запросов, status-line для откликов) и алгоритмом определения размера тела сообщения (раздел 6).

```
start-line     = request-line / status-line
```

Теоретически клиент может получать запросы, а сервер - отклики, различая их по формату start-line. На практике серверы ожидают только запросы (отклик считается запросом с недействительным методом), а клиенты - отклики.

В HTTP применяются некоторые элементы протокола, похожие на многоцелевые расширения электронной почты (Multipurpose Internet Mail Extension или MIME) [RFC2045]. Различия между HTTP и MIME рассмотрены в Приложении B.

2.2. Разбор сообщения

Обычная процедура разбора сообщения HTTP состоит из чтения start-line в структуру, чтения каждой строки полей заголовка в хэш-таблицу по именам полей до пустой строки и затем использование разобранных данных для определения наличия тела в сообщении. Если указано тело сообщения, оно считывается как поток, пока не будет получено число октетов, равное размеру тела, или сообщение не будет закрыто.

Получатель **должен** разбирать сообщение HTTP как последовательность октетов с кодировкой, являющейся надмножеством US-ASCII [USASCII]. Разбор сообщения HTTP как потока символов Unicode без учёта конкретной кодировки создаёт уязвимости защиты из-за различий в способах обработки строк библиотеками недействительных многобайтовых последовательностей, содержащих октет LF (%x0A). Строковые синтаксические анализаторы можно безопасно применять в элементах протокола только после извлечения элемента из сообщения, например, для поля заголовка после того, как разбор сообщения разделит строки полей.

Хотя завершением start-line и строк полей служит последовательность CRLF, получатель **может** считать символ LF завершением строки, игнорируя предшествующий символ CR. Отправителю **недопустимо** генерировать «голый» символ CR (CR без следующего сразу за ним LF) в элементах протокола, отличных от содержимого. Получатель такого голого CR **должен** считать элемент недействительным или заменять одиночный символ CR пробелом (SP) перед обработкой элемента или пересылкой сообщения.

Старые реализации пользовательских агентов HTTP/1.0 могут передавать лишние символы CRLF после запроса POST в качестве обхода для некоторых ранних серверных приложений, которые не могут читать содержимое тела сообщения без завершения строки в конце. Пользовательскому агенту HTTP/1.1 **недопустимо** добавлять лишние CRLF в начале или в конце запроса. Если желательно завершать тело сообщения символами завершения строки, агент пользователя **должен** учитывать завершающие октеты CRLF в размере тела сообщения.

Для отказоустойчивости серверу, ожидающему получения и разбора request-line, **следует** игнорировать по меньшей мере одну пустую строку (CRLF), полученную до request-line.

Отправителю **недопустимо** включать пробельные символы между start-line и первым полем заголовка. При получении пробелов между start-line и первым полем заголовка сообщение **должно** отбрасываться как недействительное или строки с пробелами в начале **должны** восприниматься без дальнейшей обработки (т. е. игнорироваться целиком и последующие строки с пробелами в начале, пока не будет получено корректное поле заголовка или не завершится раздел заголовков). Отклонение или исключение обработки недействительных строк с пробелами в начале нужно для предотвращения их ошибочной интерпретации нисходящими получателями, которые могут быть уязвимы для атак с контрабандой запросов (параграф 11.2) или расщеплением откликов (параграф 11.1).

Когда сервер, прослушивающий лишь запросные сообщения HTTP или обрабатывающий то, что представляется запросом по строке start-line, получает последовательность октетов, не соответствующую грамматике сообщений HTTP, за исключением указанных выше случаев, ему **следует** генерировать отклик с кодом 400 (Bad Request) и закрывать соединение.

2.3. Версия HTTP

В HTTP используется схема нумерации <major>.<minor> для указания версии протокола. Эта спецификация определяет версию 1.1. Семантика номеров версий HTTP описана в параграфе 2.5 [HTTP].

Версия сообщения HTTP/1.x указывается полем HTTP-version в строке start-line (регистр символов учитывается).

```
HTTP-version  = HTTP-name "/" DIGIT "." DIGIT
HTTP-name     = %s"HTTP"
```

При передаче сообщения HTTP/1.1 получателю HTTP/1.0 [HTTP/1.0] или получателю с неизвестной версией сообщение HTTP/1.1 строится так, чтобы его можно было интерпретировать как действительное сообщение HTTP/1.0, если игнорировать все более новые поля. Данная спецификация вносит требования к версии получателя для некоторых новых функций и соответствующий спецификации отправитель будет использовать только совместимые функции, пока не определит, что получатель поддерживает HTTP/1.1 (по конфигурации или принятому сообщению).

Посредники, обрабатывающие сообщения HTTP (все посредники, кроме выступающих в качестве туннелей), **должны** передавать своё значение HTTP-version в пересылаемых пакетах, если только версия не снижена намеренно для обхода проблем в восходящем направлении. Иными словами, посредникам не разрешается вслепую пересылать start-line, не убедившись, что версия протокола в данном сообщении соответствует версии, которую поддерживает посредник как при получении, так и при отправке сообщений. Пересылка сообщения HTTP без переписывания HTTP-version может приводить к коммуникационным ошибкам, если получатели в нисходящем направлении используют версию отправителя сообщения для определения функций, которые можно безопасно использовать для последующего взаимодействия с отправителем.

Отправитель **может** передавать отклик HTTP/1.0 на запрос HTTP/1.1, если он знает или предполагает, что клиент некорректно реализует спецификацию HTTP и не способен корректно обрабатывать отклики более поздних версий, например, клиент не может корректно разобрать номер версии или известно, что посредник вслепую пересылает HTTP-version даже при несоответствии данной младшей (minor) версии протокола. Такое понижение версии протокола **не следует** применять, пока оно не вызвано конкретными атрибутами клиента или поля заголовка запроса (например, User-Agent) не совпадают однозначно со значениями, переданные клиентом, об ошибках которого известно.

3. Строка запроса

Строка request-line начинается с маркера метода, за которым следует 1 пробел (SP), цель запроса (request-target), ещё 1 пробел и версия протокола.

```
request-line   = method SP request-target SP HTTP-version
```

Хотя правила грамматики request-line требуют разделять элементы строки одиночным октетом SP, получатели **могут** выполнять разбор по границам слов, обозначенным пробельными символами, считая любую форму пробельных символов, кроме завершения строки CRLF, разделителем SP и игнорируя предшествующий или последующий пробельный символ. К пробельным символам относятся SP, HTAB, VT (%x0B), AH (%x0C), отдельный (bare) символ CR. Однако небрежный синтаксический анализ может приводить к уязвимостям защиты (контрабанда запросов), если имеется несколько получателей сообщения со своим уникальным пониманием отказоустойчивости (см. параграф 11.2).

HTTP не задаёт предопределённого ограничения на размер request-line, как указано в параграфе 2.3 [HTTP]. Серверу, получившему элемент method, размер которого превышает размер любого реализуемого им метода, **следует** отвечать кодом статуса 501 (Not Implemented). Сервер, получивший элемент request-target с размером больше размера URI, который он готов анализировать, **должен** отвечать кодом 414 (URI Too Long) (см. параграф 15.5.15 в [HTTP]).

На практике встречаются специальные ограничения на размер request-line. Всем отправителям и получателям HTTP **рекомендуется** поддерживать размер request-line хотя бы в 8000 октетов.

3.1. Метод

Маркер метода указывает метод запроса, применяемого к целевому ресурсу. Регистр символов не учитывается.

```
method         = token
```

Методы запроса, определённые этой спецификацией, приведены в разделе 9 [HTTP], вместе со сведениями о реестре методов HTTP и регистрации новых методов.

3.2. Цель запроса

Элемент request-target указывает целевой ресурс для выполнения запроса. Клиент выводит request-target из URI желаемой цели. Применяется четыре различных формата request-target в зависимости от метода запроса и предназначенности для прокси.

```
request-target = origin-form
               / absolute-form
               / authority-form
               / asterisk-form
```

Пробельные символы в request-target не допускаются. К сожалению некоторые пользовательские агенты не могут корректно представить или исключить пробельные символы, имеющиеся в гипертекстовых ссылках, что ведёт к передаче запрещённых символов в request-target некорректно сформированной request-line. Получателям недействительной request-line **следует** отвечать кодом ошибки 400 (Bad Request) или кодом перенаправления 301 (Moved Permanently) с корректным кодированием request-target. Получателю **не следует** пытаться автоматически исправить строку и обработать запрос без перенаправления, поскольку недействительная строка request-line может быть создана намеренно для обхода фильтров защиты в цепочке запросов.

Клиент **должен** включать поле (параграф 7.2 в [HTTP]) во все запросные сообщения HTTP/1.1. Если URI цели включает компонент authority, клиент **должен** передать в поле Host значение, идентичное authority, за исключением субкомпонента userinfo и разделителя @ (параграф 4.2 в [HTTP]). Если компонент authority отсутствует или не определён для URI цели, клиент **должен** передать поле Host с пустым значением.

Сервер должен отвечать кодом 400 (Bad Request) на любое запросное сообщение HTTP/1.1 без поля заголовка Host, с полем Host в несколько строк или недействительным значением в этом поле.

3.2.1. origin-form

Наиболее распространённой формой request-target является origin-form.

```
origin-form    = absolute-path [ "?" query ]
```

При подаче запроса (кроме CONNECT и OPTIONS для всего сервера, см. ниже) непосредственно серверу-источнику клиент **должен** передавать в качестве request-target только абсолютный путь и компоненты запроса URI цели. Если в URI цели компонент пути пуст, клиент **должен** передать в качестве пути символ / в origin-form строки request-target. Передаётся также поле заголовка Host, как указано в параграфе 7.2 [HTTP]. Например, клиент, желающий извлечь содержимое ресурса, указанного http://www.example.org/where?q=now, непосредственно с сервера-источника, будет создавать (или использовать имеющееся) соединение TCP с портом 80 хоста www.example.org и передавать строки

```
GET /where?q=now HTTP/1.1
Host: www.example.org
```

за которыми следует остальная часть запросного сообщения.

3.2.2. absolute-form

При запросе (кроме CONNECT и OPTIONS для всего сервера, см. ниже) к прокси клиент **должен** передать URI цели в absolute-form как request-target.

```
absolute-form  = absolute-URI
```

Прокси предлагается обслужить этот запрос из действительного кэша (при возможности) или отправить такой же запрос от имени клиента на следующий приёмный прокси-сервер или сервер-источник, указанный request-target. Требования к такой «пересылке» сообщений приведены в параграфе 7.6 [HTTP].

Пример absolute-form для request-line имеет вид

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

Клиент **должен** передавать поле запроса Host в щапросе HTTP/1.1, даже если request-target является absolute-form, поскольку это позволяет передавать Host через древние прокси HTTP/1.0, которые могут не поддерживать Host.

Когда прокси получает запрос с absolute-form для request-target, он **должен** игнорировать полученное поле заголовка Host (при наличии) и заменять его сведениями о хосте из request-target. Прокси, пересылающий такой запрос, **должен** генерировать новое поле заголовка Host на основе полученного request-target вместо пересылки исходного поля Host.

Когда сервер-источник получает запрос с absolute-form для request-target, он **должен** игнорировать полученное поле Host (при наличии) и вместо этого использовать сведения о хосте из request-target. Если в request-target нет компонента authority, будет передаваться пустое поле Host.

Сервер **должен** воспринимать absolute-form в запросах, хотя большинство клиентов HTTP/1.1 передаёт absolute-form только для прокси.

3.2.3. authority-form

Форма authority-form для request-target применяется лишь в запросах CONNECT (параграф 9.3.6 в [HTTP]) и включает лишь uri-host и номер порта целевой точки туннеля через двоеточие (:).

```
authority-form = uri-host ":" port
```

При подаче запроса CONNECT для организации туннеля через один или несколько прокси клиент **должен** передавать в качестве request-target только хост и порт для целевой точки туннеля. Клиент получает их из компонента authority в URI цели, указывая принятый по умолчанию порт, если тот не задан в URI цели. Например, запрос CONNECT к http://www.example.com будет иметь вид

```
CONNECT www.example.com:80 HTTP/1.1
Host: www.example.com
```

3.2.4. asterisk-form

Форма asterisk-form для request-target применяется лишь в запросах OPTIONS на уровне сервера (параграф 9.3.7 в [HTTP]).

```
asterisk-form  = "*"
```

Когда клиент хочет передать запрос OPTIONS для сервера в целом, а не конкретного именованного ресурса на этом сервере, он **должен** указать в request-target только символ * (%x2A). Например,

```
OPTIONS * HTTP/1.1
```

Если прокси получает запрос OPTIONS с absolute-form для request-target, где URI имеет пустой путь и нет компонента запроса, последний прокси в цепочке запроса **должен** передать request-target * при пересылке запроса указанному серверу-источнику. Например, запрос

```
OPTIONS http://www.example.org:8001 HTTP/1.1
```

будет пересылаться финальным прокси в виде

```
OPTIONS * HTTP/1.1
Host: www.example.org:8001
```

после соединения с поротом 8001 хоста www.example.org.

3.3. Восстановление Target URI

URI цели, это request-target, если request-target имеет форму absolute-form. В этом случае сервер будет разбирать URI по базовым компонентам для дальнейшей оценки. В иных случаях сервер по контексту соединения и различным частям запросного сообщения восстанавливает URI для идентификации целевого ресурса (параграф 7.1 в [HTTP]):

- Если конфигурация сервера предусматривает фиксированную схему URI или схема предоставляется доверенным выходным (outbound) шлюзом, эта схема применяется для URI цели. Это часто встречается в больших системах, поскольку шлюзовой сервер получает контекст соединения от клиента и заменяет его своим соединением с принимающим (inbound) сервером. В ином случае при получении запроса по защищённому соединению для URI цели используется схема https, по незащищённому - http.
- Если request-target имеет форму authority-form, компонентом authority в URI цели является request-target, в иных случаях - значение поля заголовка Host. Если поля Host нет в заголовке, он пусто или недействительно, authority в URI цели остаётся пустым.
- Если request-target имеет форму authority-form или asterisk-form, комбинация компонентов path и query в URI цели будет пустой, в ином случае - request-target.
- Компоненты восстановленного URI цели после описанного выше их определения могут быть объединены в форму absolute-URI путём конкатенации схемы, ://, authority и комбинации компонентов path и query.

Пример 1. Сообщение, полученное через защищённое соединение

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

имеет URI цели https://www.example.org/pub/WWW/TheProject.html.

Пример 2. Сообщение, полученное через незащищённое соединение

```
OPTIONS * HTTP/1.1
Host: www.example.org:8080
```

имеет URI цели http://www.example.org:8080

Если компонент authority в URI цели пуст, а схема URI требует непустого значения (как в случае http и https), сервер может отклонить запрос или определить, применимо ли заданное по умолчанию значение, соответствующее контексту входящего сообщения. Контекст может включать такие детали, как адрес и порт, применяемая защита, заданные локально сведения, относящиеся к конфигурации этого сервера. Пустой компонент authority заменяется принятым по умолчанию перед дальнейшей обработкой запроса.

Подстановка принятого по умолчанию значения authority в контексте защищённого соединения по своей сути небезопасна, если есть какой-либо шанс, что предполагаемые полномочия агента пользователя отличаются от принятых по умолчанию. Сервер, способный однозначно определить authority из контекста запроса, **может** без риска использовать своё отождествление в качестве принятого по умолчанию. Как вариант, может оказаться лучше перенаправить запрос на безопасный ресурс, который объясняет, как обрести нового клиента.

Отметим, что восстановление URI цели от клиента - это лишь половина процесса идентификации целевого ресурса. Другой половиной является определение того, указывает ли URI цели ресурс, для которого сервер готов и способен передать отклик, как указано в параграфе 7.4 [HTTP].

4. Строка статуса

Первой строкой сообщения-отклика является status-line, содержащая версию протокола, пробел(SP), код статуса, ещё один пробел и **необязательный** текст с описанием кода статуса.

```
status-line = HTTP-version SP status-code SP [ reason-phrase ]
```

Хотя правило грамматики status-line требует отделять каждый элемент одним октетом SP, получатели **могут** выполнять разбор про пробельным символам границ слов и считать любой пробельный символ (кроме завершения строки) разделителем SP, игнорирую предшествующие и следующие за ним пробельные октеты SP, HTAB, VT (%x0B), AH (%x0C), отдельный символ CR. Однако такой снисходительный разбор может приводить к уязвимостям защиты при наличии нескольких получателей, каждый из которых по своему понимает отказоустойчивость (см. параграф 11.1).

Элемент status-code являет трехзначным целым числом, описывающим результат попытки сервера понять и выполнить соответствующий запрос клиента. Остальную часть запросного сообщения получатель разбирает и

интерпретирует в соответствии с семантикой, указанной кодом статуса, если получатель его понимает, или в соответствии с классом кода, если конкретный код статуса не понятен.

```
status-code    = 3DIGIT
```

Коды статуса HTTP описаны в разделе 15 [HTTP] с разбивкой их на классы. Приведены также соображения по определению новых кодов и описан реестр IANA для сбора таких определений.

Элемент reason-phrase служит лишь для текстового описания, связанного с числовым кодом статуса, в основном из уважения к ранним протоколам приложений Internet, которые более часто применялись с интерактивными текстовыми клиентами.

```
reason-phrase  = 1*( HTAB / SP / VCHAR / obs-text )
```

Клиенту **следует** игнорировать содержимое reason-phrase, поскольку оно не является надёжным источником информации (оно может транслироваться в локальный язык, переопределяться посредниками или отбрасываться при пересылке через другие версии HTTP). Сервер **должен** включать пробел после status-code, даже если reason-phrase не указывается (строка status-line завершается пробелом).

5. Синтаксис полей

Каждая строка поля состоит из имени поля (без учёта регистра символов), двоеточия (:),необязательных начальных пробельных символов, значения поля и необязательных пробельных символов в конце.

```
field-line   = field-name ":" OWS field-value OWS
```

Правила разбора значений полей заданы в параграфе 5.5 [HTTP]. В этом разделе описывается базовый синтаксис включения полей заголовков в сообщения HTTP/1.1 и извлечения полей из заголовков.

5.1. Разбор строки поля

Сообщения разбираются по общему алгоритму, независимо от имён отдельных полей. Содержимое значения данной строки поля не разбирается до более позднего этапа интерпретации сообщения (обычно после обработки всего раздела полей в сообщении). Не разрешается включать пробельные символы между именем поля и двоеточием (:). В прошлом различия при обработке таких пробельных символов приводили к уязвимостям защиты в процессе маршрутизации запроса и обработки отклика. Сервер **должен** отвергать с кодом статуса 400 (Bad Request) любые полученные запросные сообщения с пробельными символами между именем поля заголовка и двоеточием. Прокси **должны** удалять такие символы из сообщений с откликами до пересылки сообщения в нисходящем направлении.

Перед значением поля и после него могут помещаться пробельные символы (optional whitespace ил OWS). Предпочтительно включать один символ SP перед значением для удобства чтения человеком. Значение не включает пробельных символов в начале или в конце и OWS до первого непробельного октета или после последнего значения поля исключаются синтаксическим анализатором при извлечении значения из строки поля.

5.2. Устаревшая фальцовка строк

Исторически сложилось так, что значения полей HTTP/1.x можно было растягивать на несколько строк, предваряя каждую дополнительную строку символом пробела или горизонтальной табуляции (obs-fold). Эта спецификация отказывается от такой фальцовки строк везде, кроме типа носителя message/http (параграф 10.1).

```
obs-fold     = OWS CRLF RWS ; устаревшая фальцовка строки
```

Отправителю **недопустимо** создавать сообщения с фальцовкой строк (т. е. значениями строк полей, соответствующими правилу obs-fold), если они не предназначены для упаковки в носитель типа message/http.

Сервер, получивший obs-fold в запросном сообщении вне контейнера message/http, **должен** отвергнуть это сообщение, возвращая код 400 (Bad Request) предпочтительно с пояснением недопустимости устаревшей фальцовки, или заменить все obs-fold одним или несколькими октетами SP до интерпретации значения поля или пересылки сообщения в нисходящем направлении.

Прокси или шлюз, получивший obs-fold в сообщении-отклике вне контейнера message/http, **должен** отбросить сообщение и заменить его откликом (Bad Gateway) предпочтительно с пояснением недопустимости устаревшей фальцовки, или заменить все obs-fold одним или несколькими октетами SP до интерпретации значения поля или пересылки сообщения в нисходящем направлении.

Агент пользователя, получивший obs-fold в сообщении-отклике вне контейнера message/http, **должен** заменить все obs-fold одним или несколькими октетами SP до интерпретации значения поля.

6. Тело сообщения

Тело сообщения HTTP/1.1 (при наличии) служит для передачи содержимого (параграф 6.4 в [HTTP]) запроса или отклика. Тело сообщения идентично содержимому, если не применяется транспортное кодирование (параграф 6.1).

```
message-body = *OCTET
```

Правила определения наличия содержимого в сообщении HTTP/1.1 различаются для запросов и откликов.

Наличие тела сообщения в запросе указывается полем Content-Length или Transfer-Encoding. Обрамление запросных сообщений зависит от семантики метода.

Наличие тела сообщения в отклике (см. параграф 6.3) зависит от метода запроса, породившего отклик, и кода статуса в отклике в соответствии с семантикой HTTP (параграф 6.4.1 в [HTTP]).

6.1. Transfer-Encoding

В поле заголовка Transfer-Encoding содержится список имён транспортного кодирования, соответствующий последовательности имён транспортного кодирования которые применены (или будут применены) к содержимому для формирования тела сообщения. Определения транспортного кодирования даны в разделе 7.

```
Transfer-Encoding = #transfer-coding ; см. параграф 10.1.4 в [HTTP]
```

Поле Transfer-Encoding аналогично Content-Transfer-Encoding в MIME, разработанному для обеспечения безопасной доставки двоичных данных через 7-битовый транспортный сервис ([RFC2045], раздел 6). Однако безопасная доставка для чистого 8-битового транспортного протокола отличается. В случае HTTP назначение Transfer-Encoding состоит в основном в обеспечении точного разграничения создаваемого динамически содержимого, а также для различения кодирования, применяемого лишь при передаче, от кодирования, являющегося характеристикой выбранного представления.

Получатель **должен** уметь разобрать блочное (chunk) транспортное кодирование (параграф 7.1), поскольку оно играет важную роль в обрамлении сообщений, когда размер содержимого не известен заранее. Отправителю **недопустимо** применять блочное транспортное кодирование для тела сообщения (т. е. создавать блоки из блочных сообщений не разрешается). Если применяется транспортное кодирование, отличное от блочного, отправитель **должен** применять блочное кодирование в качестве окончательного транспортного кодирования для обеспечения подобающего обрамления сообщения. Если отличное от блочного транспортное кодирование применяется для содержимого отклика, отправитель **должен** использовать блочное транспортное кодирование в качестве финального или прервать сообщение, разрывая соединение. Например,

```
Transfer-Encoding: gzip, chunked
```

указывает, что содержимое сжато с помощью gzip и организовано в блоки с применением блочного кодирования при формировании тела сообщения.

В отличие от Content-Encoding (параграф 8.4.1 в [HTTP]), Transfer-Encoding является свойством сообщения, а не представления. Любой получатель в цепочке запрос-отклик **может** декодировать транспортное кодирование или применить дополнительное транспортное кодирование к телу сообщения при условии внесения соответствующих изменений в значение поля Transfer-Encoding. Дополнительные сведения о параметрах кодирования могут предоставляться другими полями заголовка, не заданными этой спецификацией.

Поле Transfer-Encoding **может** передаваться в отклике на запрос HEAD или в отклике 304 (Not Modified) (параграф 15.4.5 в [HTTP]) на запрос GET, не включающих тела сообщения, для указания того, что сервер-источник применил бы кодирование сообщения, если бы запрос был безусловным GET. Такая индикация не требуется, поскольку любой получатель в цепочке отклика (включая сервер-источник) может удалять транспортное кодирование, если оно не нужно.

Серверу **недопустимо** передавать поле заголовка в откликах с кодом 1xx (Informational) или 204 (No Content), а также в откликах с кодами 2xx (Successful) на запрос CONNECT (параграф 9.3.6 в [HTTP]).

Серверу, получившему запросное сообщение с непонятным транспортным кодированием, **следует** возвращать отклик 501 (Not Implemented).

Поле Transfer-Encoding было добавлено в HTTP/1.1. Обычно предполагается, что реализации, анонсирующие поддержку только HTTP/1.0, не понимают, как обрабатывать содержимое с транспортным кодированием, а сообщение HTTP/1.0 с Transfer-Encoding говорит, скорей всего, об отсутствии подобающей обработки при доставке.

Клиенту **недопустимо** передавать запросы с полем Transfer-Encoding, пока он не знает, что сервер обслуживает запросы HTTP/1.1 (или более поздних младших версий). Такие сведения могут предоставляться в виде конкретной пользовательской конфигурации или путём сохранения версии полученного ранее отклика. Серверу **недопустимо** передавать отклик с полем Transfer-Encoding, если соответствующий запрос не указывает версию HTTP/1.1 (или более позднюю младшую версию).

Ранние реализации Transfer-Encoding иногда отправляли блочное транспортное кодирование для обрамления сообщений и оценочное поле заголовка Content-Length для использования в индикаторах выполнения. Именно поэтому поле Transfer-Encoding задано как переопределяющее поле Content-Length, а не взаимно несовместимое с ним. К сожалению, пересылка такого сообщения может вызывать уязвимости, связанные с контрабандой (параграф 11.2) или расщеплением запросов (параграф 11.1), если какой-либо из последующих получателей не может разобрать сообщение в соответствии с этой спецификацией, особенно если получатель нисходящего направления поддерживает только HTTP/1.0.

Сервер **может** отвергать запрос, содержащий оба поля Content-Length и Transfer-Encoding или обрабатывать его с соответствии с полем Transfer-Encoding. В любом случае сервер **должен** закрыть соединение после ответа на такой запрос во избежание возможных атак.

Сервер или клиент, получивший сообщение HTTP/1.0 с полем заголовка Transfer-Encoding, **должен** трактовать его как сообщение с ошибочным обрамлением даже при наличии Content-Length и закрывать соединение после обработки сообщения. Отправитель сообщения может сохранить в буфере часть сообщения, которую можно ошибочно интерпретировать при последующем использовании сообщения.

6.2. Content-Length

Если в сообщении нет поля Transfer-Encoding, поле заголовка Content-Length (параграф 8.6 в [HTTP]) может указать предполагаемый размер содержимого десятичным значением числа октетов. Для сообщений с содержимым поле Content-Length предоставляет требуемые сведение об обрамлении для определения конца данных (и сообщения). Для сообщений без содержимого Content-Length указывает размер выбранного представления (параграф 8.6 в [HTTP]).

Отправителю **недопустимо** передавать поле заголовка Content-Length в сообщениях с полем Transfer-Encoding.

> Примечание. HTTP использует поле Content-Length для обрамления сообщений не так, как это принято в MIME, где это необязательное поле применяется только для типа носителя message/external-body.

6.3. Размер тела сообщения

Размер тела определяется одним из показанных ниже способов (в порядке предпочтения.

1. Любой отклик на запрос HEAD и любой отклик с кодом 1xx (Informational), 204 (No Content), 304 (Not Modified) всегда завершается первой пустой строкой после полей заголовков, независимо от присутствующих в сообщении полей заголовков, поэтому не может включать тело сообщения или раздел трейлеров.

2. Любой отклик 2xx (Successful) на запрос CONNECT предполагает, что соединение становится туннелем сразу после пустой строки, завершающей поля заголовка. Клиент **должен** игнорировать заголвока Content-Length и Transfer-Encoding в таком сообщении.

3. В сообщении с полями Transfer-Encoding и Content-Length поле Transfer-Encoding переопределяет Content-Length. Такое сообщение может указывать попытку контрабанды (параграф 11.2) или расщепления запроса (параграф 11.1) и должно обрабатываться как ошибка. Посредник ACK, решивший переслать такое сообщение, **должен** удалить из него поле Content-Length и обработать Transfer-Encoding (см. ниже) до пересылки сообщения в нисходящем направлении.

4. Если имеется поле заголовка Transfer-Encoding и финальным транспортным кодированием является блочное (параграф 7.1), размер тела сообщения определяется считыванием и декодированием блочных данных, пока транспортное кодирование не укажет завершение данных.

   Если в отклике имеется Transfer-Encoding и финальным транспортным кодированием не является блочное, размер тела сообщения определяется считыванием из соединения до его закрытия сервером.

   Если в запросе имеется Transfer-Encoding и финальным транспортным кодированием не является блочное, размер тела сообщения невозможно определить надёжно и сервер **должен** отвечать кодом 400 (Bad Request), а затем закрывать соединение.

5. Если сообщение получено без Transfer-Encoding и с недействительным полем Content-Length, кадрирование сообщения недействительно и получатель **должен** считать это неисправимой ошибкой, если только значение поля не было успешно разобрано как список значений через запятые (параграф 5.6.1 в [HTTP]), в котором все значения действительны и одинаковы (в этом случае сообщение обрабатывается с этим единственным значением поля Content-Length). Если неисправимая ошибка встречена в запросном сообщении, сервер **должен** ответить кодом 400 (Bad Request) и закрыть соединение. Если это сообщение-отклик, полученное прокси, тот **должен** закрыть соединение, отбросить полученное сообщение и передать клиенту отклик 502 (Bad Gateway). Если это сообщение-отклик, полученное агентом пользователя, тот **должен** закрыть соединение с сервером и отбросить полученное сообщение.

6. Если имеется действительное поле заголовка Content-Length без Transfer-Encoding, предполагается, что десятичное значение поля указывает размер тела сообщения в октетах. Если отправитель закрывает соединение или у получателя возникает тайм-аут до получения указанного числа октетов, получатель **должен** считать сообщение неполным и закрыть соединение.

7. Если это запросное сообщение и ни одно из приведённых выше условий не выполняется, тело сообщения имеет размер 0 (в сообщении нет тела).

8. В иных случаях сообщение является откликом без указания размера его тела, поэтому размер определяется числом октетов, полученных до закрытия соединения сервером.

Поскольку нет возможности отличить успешно завершённое сообщение от ограниченного закрытием соединения отклика, принятого частично из-за отказа в сети, серверу **следует** генерировать сообщение, граница которого задана кодированием или размером. Функция ограничения сообщения закрытием соединения предназначена в основном для совместимости с HTTP/1.0.

> Примечание. Запросные сообщения никогда не завершаются закрытием соединения, поскольку они всегда явно кадрируются размером или транспортным кодированием и отсутствие их подразумевает, что запрос завершается сразу после раздела заголовков.

Сервер **может** отклонить запрос, содержащий тело сообщения без Content-Length, откликом 411 (Length Required).

Если не применяется транспортное кодирование, отличное от блочного, клиенту, передающему запрос с телом сообщения, **следует** использовать при известном заранее размере тела сообщения действительное поле Content-Length, а не блочное транспортное кодирование, поскольку некоторые из существующих служб отвечают на блочное кодирование кодом 411 (Length Required) даже если они понимают блочное транспортное кодирование. Обычно это обусловлено тем, что такие службы реализуются через шлюз, которому нужно заранее знать размер, а сервер не может или не хочет буферизовать запрос целиком до его обработки.

Агент пользователя, передающий запрос с телом сообщения, **должен** передать действительное поле заголовка Content-Length или использовать блочное транспортное кодирование. Клиенту **недопустимо** использовать блочное транспортное кодирование, если он не знает, что сервер будет обрабатывать запросы HTTP/1.1 (или более поздние). Такие сведения могут быть получены из конкретной пользовательской конфигурации или в результате запоминания полученного ранее отклика.

Если финальный отклик на последний запрос в соединении был получен полностью и остались дополнительные данные для чтения, агент пользователя **может** отбросить оставшиеся данные или попытаться определить являются ли они частью предшествующего тела сообщения, что может произойти из-за некорректного значения Content-Length. Клиенту **недопустимо** обрабатывать, кэшировать или пересылать такие дополнительные данные как отдельный отклик, поскольку такое поведение может привести к отравлению кэша.

7. Транспортное кодирование

Имена транспортного кодирования служат для индикации кодировочного преобразования, которое было, могло быть или может потребоваться применить к сообщению для обеспечения безопасной транспортировки через сеть. Это отличается от кодирования содержимого тем, что транспортное кодирование является свойством сообщения, а не передаваемого представления.

В именах транспортного кодирования не учитывается регистр символов, а имена должны регистрироваться в реестре HTTP Transfer Coding, как указано в параграфе 7.3. Они указываются в полях заголовка Transfer-Encoding (параграф 6.1) и TE (параграф 10.1.4 в [HTTP]) (последнее также задаёт грамматику transfer-coding).

7.1. Блочное транспортное кодирование

Блочное (chunked) транспортное кодирование оборачивает содержимое для передачи в форме последовательности блоков (chunk), каждый из которых содержит индикатор размера, на которыми может следовать необязательный раздел с полями трейлера. Это позволяет передавать потоки содержимого неизвестного размера в виде последовательности буферов ограниченного размера, давая отправителю возможность сохранять соединение, а получателю - знать, когда сообщение получено целиком.

```
chunked-body   = *chunk
                 last-chunk
                 trailer-section
                 CRLF

chunk          = chunk-size [ chunk-ext ] CRLF
                 chunk-data CRLF
chunk-size     = 1*HEXDIG
last-chunk     = 1*("0") [ chunk-ext ] CRLF

chunk-data     = 1*OCTET ; последовательность октетов размером chunk-size
```

Поле chunk-size - это строка шестнадцатеричных цифр, указывающая размер chunk-data в октетах. Блочное транспортное кодирование завершается блоком с chunk-size = 0, после чего может следовать раздел трейлеров, а затем пустая строка.

Получатель **должен** быть способен разбирать и декодировать блочное транспортное кодирование.

HTTP/1.1 не задаёт средств ограничения на размер блочного отклика, чтобы посредник мог быть уверен в буферизации всего отклика. Кроме того, очень большой размер блоков может приводить к переполнению или потере точности, если их значения не будут точно представлены в принимающей реализации. Поэтому получатели **должны** предусматривать большие шестнадцатеричные числа и предотвращать ошибки разбора в результате переполнения при преобразовании чисел или потери точности при представлении целых чисел.

Блочное кодирование не предусматривает каких-либо параметров и их наличие **следует** считать ошибкой.

7.1.1. Расширения блоков

Блочное кодирование разрешает включать расширения в каждый блок сразу после chunk-size для предоставления метаданных блока (например, подписи или хэша), сведений об управлении или изменения размера тела сообщения.

```
chunk-ext      = *( BWS ";" BWS chunk-ext-name
                    [ BWS "=" BWS chunk-ext-val ] )

chunk-ext-name = token
chunk-ext-val  = token / quoted-string
```

Блочное кодирование специфично для каждого соединения и, скорей всего, будет удаляться или перекодироваться каждым получателей (включая посредников) до того, как приложение вышележащего уровня получит возможность проверить расширения. Поэтому расширения блоков обычно применяются специализированными службами HTTP, такими как долгий опрос (где у клиента и сервера могут быть общие ожидания в части применения расширений блока), или для заполнения в соединениях со сквозной защитой.

Получатель **должен** игнорировать нераспознанные расширения. Сервер должен ограничивать суммарный размер расширений блока в запросе разумным значением, как это делается для размера и тайм-аутов в других частях сообщения, и генерировать подходящий отклик 4xx (Client Error) в случае превышения размера.

7.1.2. Раздел трейлеров

Раздел трейлеров позволяет отправителю включать дополнительные поля в конце блочного сообщения для представления метаданных, которые могут создаваться динамически при отправке сообщения, такие как проверка целостности сообщения, цифровая подпись, статус постобработки. Корректное использование и ограничения для полей трейлера описаны в параграфе 6.5 [HTTP].

```
trailer-section   = *( field-line CRLF )
```

Получатель, удаляющий блочное кодирование из сообщения, **может** селективно сохранять или отбрасывать полученные поля трейлеров. Получатель, сохраняющий полученные поля, **должен** сохранять/пересылать поля трейлера отдельно от полей заголовка или сливать их с разделом заголовка. Получателю **недопустимо** сливать полученные поля трейлера с заголовком, если только определение соответствующего поля заголовка явно не разрешает это, указывая способ безопасного слияния.

7.1.3. Декодирование

Ниже представлен псевдокод декодирования блочного транспортного кодирования.

```
length := 0
считывание chunk-size, chunk-ext (при наличии) и CRLF
пока (chunk-size > 0) {
   считывание chunk-data и CRLF
   добавление chunk-data в конец содержимого
   length := length + chunk-size
   считывание chunk-size, chunk-ext (if any) и CRLF
}
считывание трейлера
пока (трейлер не пуст) {
   если (поля трейлера сохраняются/пересылаются forwarded separately) {
       поле трейлера добавляется в конец имеющихся трейлерных полей
   }
   иначе если (поле трейлера понятно и задано как сливаемое) {
       слияние поля трейлера с имеющимися
```

```
        }
        иначе {
            отбрасывание поля трейлера
        }
        считывание поля трейлера
    }
    Content-Length := length
    Удаление chunked из Transfer-Encoding
```

7.2. Транспортное кодирование со сжатием

Указанные ниже имена транспортного кодирования со сжатием совпадают с именами соответствующих алгоритмов.

```
compress (и x-compress), см. параграф 8.4.1.1 в [HTTP].
deflate, см. параграф 8.4.1.2 в [HTTP].
gzip (и x-gzip), см. параграф 8.4.1.3 в [HTTP].
```

Кодирование со сжатием не использует параметров и их наличие **следует** считать ошибкой.

7.3. Реестр транспортного кодирования

Реестр HTTP Transfer Coding Registry (https://www.iana.org/assignments/http-parameters) определяет пространство имён транспортного кодирования. Каждая запись реестра **должна** включать поля:

- имя;
- описание;
- указатель на текст спецификации.

Именам транспортного кодирования **недопустимо** совпадать с именами кодирования содержимого (параграф 8.4.1 в [HTTP]), если только в них не применяется идентичное кодирующее преобразование, как для сжатия (параграф 7.2).

Поле заголовка TE (параграф 10.1.4 в [HTTP]) использует псевдопараметр q как значение ранга при наличии нескольких приемлемых вариантов транспортного кодирования. В будущих регистрациях транспортного кодирования **не следует** задавать параметров с именем q (в любом регистре), чтобы не возникало неоднозначности.

Значения добавляются в пространство имён по процедуре IETF Review (параграф 4.8 в [RFC8126]) и **должны** соответствовать назначению транспортного кодирования, заданному в этой спецификации.

Использовать имена программ для идентификации форматов кодирования нежелательно и не рекомендуется для будущих вариантов кодирования.

7.4. Согласование транспортного кодирования

Поле TE (параграф 10.1.4 в [HTTP]) служит в HTTP/1.1 для указания методов транспортного кодирования в откликах, которые клиент готов воспринимать в дополнение к блочному кодированию (chunked), и готовности сохранять поля трейлера в блочном транспортном кодировании. Клиенту **недопустимо** передавать имя блочного транспортного кодирования в TE, это кодирование получатели HTTP/1.1 воспринимают всегда. Ниже приведены три примера TE.

```
TE: deflate
TE:
TE: trailers, deflate;q=0.5
```

Если приемлемо несколько вариантов транспортного кодирования, клиент **может** ранжировать их, используя параметр q (в любом регистре, подобно использованию с полях согласования содержимого, см. параграф 12.4.2 в [HTTP]). Ранг указывается действительным числом из диапазона 0 - 1 и 0,001 соответствует низшему рангу, а 1 - наиболее предпочтительному. Значение 0 указывает неприемлемый вариант (not acceptable).

Если значение поля TE пусто или TE нет совсем, единственным вариантом транспортного кодирования является блочное (chunked). Сообщения без транспортного кодирования воспринимаются всегда. Ключевое слово trailers указывает, что отправитель не будет отбрасывать поля трейлера, как описано в параграфе 6.5 [HTTP].

Поскольку поле заголовка TE относится только к непосредственному соединению, отправитель TE **должен** также передавать опцию соединения TE в поле заголовка Connection (параграф 7.6.1 в [HTTP]), чтобы предотвратить пересылку поля заголовка TE посредниками, которые не поддерживают семантику этого поля.

8. Обработка неполных сообщений

Сервер, получивший неполное сообщение с запросом (обычно из-за его отмены или возникновения тайм-аута) **может** передать сообщение об ошибке перед закрытием соединения. Клиент, получивший неполное сообщение с откликом, что может быть вызвано преждевременным закрытием соединения или отказом при декодировании предположительно блочного кодирования, **должен** записать сообщение как неполное. Требования к кэшированию неполных откликов приведены в параграфе 3.3 [CACHING].

Если отклик прерван посреди раздела заголовков (до пустой строки) а код статуса может быть основан на полях заголовка для передачи полного смысла отклика, клиент не может предполагать, что смысл передан и ему может потребоваться повторять запрос, чтобы выбрать дальнейшие действия.

Тело сообщения с блочным транспортным кодированием неполно, если не был получен блок нулевого размера, завершающий кодирование. Сообщение с действительным Content-Length неполно, если размер принятого сообщения в октетах меньше значения Content-Length. Отклик без блочного транспортного кодирования и Content-Length прерывается закрытием соединения и (при получении раздела заголовков в целости и сохранности) считается полным, если базовое соединение не указало ошибку (например, неполное закрытие в TLS оставит отклик неполным, как описано в параграфе 9.8).

9. Управление соединениями

Обмен сообщениями HTTP не зависит от базовых протоколов соединений транспортного и сеансового уровня. HTTP лишь предполагает надёжный транспорт с упорядоченной доставкой запросов и соответствующих откликов. Отображение запросов и откликов HTTP на блоки данных базового транспортного протокола выходит за рамки этой спецификации.

Как указано в параграфе 7.3 [HTTP], конкретные транспортные протоколы, используемые для взаимодействий HTTP, определяются конфигурацией клиента и URI цели. Например, схема URI http (параграф 4.2.1 в [HTTP]) указывает принятое по умолчанию соединение TCP по протоколу IP через принятый по умолчанию порт TCP 80, но клиент может быть настроен на работу через прокси с другим соединением, портом или протоколом.

Предполагается, что реализации HTTP будут управлять соединениями, включая поддержку состояния имеющегося соединения, организацию нового соединения или использование имеющегося, обработку сообщений, полученных в соединении, и завершение каждого соединения. Большинство клиентов поддерживает множество параллельных соединений, включая несколько соединений на сервер. Большинство серверов спроектировано для поддержки тысяч одновременных соединений с контролем очередей запросов для беспристрастного использования и предотвращения DoS[1]-атак.

9.1. Организация соединения

Описание организации соединений по различным протоколам транспортного и сеансового уровня. Каждое соединение HTTP отображается на одно базовое транспортное соединение.

9.2. Связывание отклика с запросом

HTTP/1.1 не включает идентификаторов запроса для связывания данного запросного сообщения с соответствующими сообщениями откликов. В результате протокол полагается на точное соответствие порядка откликов порядку отправки запросов в том же соединении. Более одного отклика на запрос приходит лишь в том случае, когда финальному отклику на запрос предшествует один или несколько информационных откликов (1xx, см. параграф 15.2 в [HTTP]) на тот же запрос.

Клиент у которого остаётся более одного запроса в соединении, **должен** поддерживать список остающихся запросов в порядке их передачи, а также **должен** связывать каждое полученное через это соединение сообщение с первым оставшимся запросом, для которого ещё не получен финальный (не 1xx) отклик. Если клиент получает данные, для которых нет оставшегося запроса, ему **недопустимо** считать эти данные действительным откликом и **следует** закрыть соединение, поскольку разграничение сообщений становится неоднозначным, если только данные не состоят или из одной или нескольких последовательностей CRLF (они могут отбрасываться в соответствии с параграфом 2.2).

9.3. Сохраняемое соединение

В HTTP/1.1 по умолчанию используются сохраняемые (persistent) соединения, что позволяет передавать через одно соединение множество запросов и откликов. Реализациям HTTP **следует** поддерживать сохраняемые соединения.

Получатель определяет является ли соединение постоянным по версии протокола и полю заголовка Connection (параграф 7.6.1 в [HTTP]) в полученном последним сообщении как указано ниже (до первого совпадения).

- При наличии опции соединения close (параграф 9.6), соединение не сохраняется после текущего отклика.
- Для протокола HTTP/1.1 (или выше) соединение сохраняется после текущего отклика.
- Если указан протокол HTTP/1.0 и имеется опция соединения keep-alive получатель не является прокси или сообщение является откликом и получатель хочет соблюдать механизм HTTP/1.0 keep-alive, соединение сохраняется после текущего отклика.
- Соединение закрывается после текущего отклика.

Не поддерживающий сохраняемые соединения клиент **должен** передавать опцию close в каждом запросном сообщении. Сервер, поддерживающий сохраняемые соединения, **должен** передавать опцию close в каждом сообщении-отклике, не имеющем кода статуса 1xx (Informational). Клиент **может** передавать дополнительные запросы на сохранения соединений, пока им не отправлена или не получена опция соединения close или не получен отклик HTTP/1.0 без опции соединения keep-alive.

Для сохранения соединения все сообщения в нем должны иметь определённый размер сообщения (т. е. размер не определяется закрытием соединения), как описано в разделе 6. Сервер **должен** прочитать тело запросного сообщения целиком или закрыть соединение после отправки отклика, в противном случае оставшиеся данные для сохраняемого сообщения будут ошибочно считаться следующим запросом. Клиент **должен** целиком прочитать тело сообщения-отклика, если он намерен использовать это же сообщение для последующего запроса.

Прокси-серверу **недопустимо** поддерживать постоянные соединения с клиентом HTTP/1.0 (см. Приложение C.2.2, где рассматриваются проблемы с полем заголовка Keep-Alive, используемым многими клиентами HTTP/1.0). Совместимость с клиентами HTTP/1.0 рассматривается в Приложении C.2.2.

9.3.1. Повторение запросов

Соединения могут закрываться в любой момент (намеренно или нет). Реализации должны предвидеть необходимость восстановления после асинхронных событий закрытия. Условия, при которых клиент может автоматически повторять остающиеся запросы, указаны в параграфе 9.2.2 [HTTP].

9.3.2. Конвейеры

Клиент, поддерживающий сохранение соединений, **может** организовать конвейер (pipeline) своих запросов (т. е. передать несколько запросов, не ожидая каждого отклика). Сервер **может** обрабатывать конвейерные запросы

[1]Denial-of-service - отказ в обслуживании.

параллельно, если все они используют безопасный метод (параграф 9.2.1 в [HTTP]), но он **должен** передавать соответствующие отклики в порядке получения запросов.

Клиенту, применяющему конвейер запросов, **следует** повторять безответные запросы, если соединение было закрыто до получения всех соответствующих откликов. При повторе конвейерных запросов после неудачного соединения (соединения, не закрытого явно сервером в его последнем полном отклике), клиенту **недопустимо** создавать конвейер сразу после организации соединения, поскольку первый запрос, оставшийся от прежнего конвейера, мог вызвать отклик об ошибке, который снова будет потерян, если передать несколько запросов по преждевременно закрытому соединению (см. проблему сброса TCP в параграфе 9.6).

Идемпотентные методы (параграф 9.2.2 в [HTTP]) важны для конвейеров, поскольку они могут автоматически повторяться при отказе соединения. Агенту пользователя **не следует** применять конвейер запросов после неидемпотентного метода, пока не получен код статуса финального отклика, если только у пользовательского агента нет средств обнаружения и восстановления после частичных отказов, связанных с последовательностью конвейера.

Посредник, получающий конвейерные запросы, **может** создать конвейер из них при пересылке в направлении приёма (inbound), поскольку он может полагаться на выходные (outbound) агенты пользователя для определения запросов, которые можно безопасно включить в конвейер. При отказе входящего (inbound) соединения до получения отклика, конвейеризующий посредник **может** попытаться повторить последовательность запросов, на которые ещё не получен отклик, если все запросы использовали идемпотентные методы, иначе этому посреднику **следует** пересылать любые полученные отклики, а затем создавать соответствующие исходящие соединения, чтобы исходящие (outbound) агенты могли выполнить нужно восстановление.

9.4. Одновременные соединения

Клиент должен ограничивать число одновременно открытых соединений, поддерживаемых с данным сервером.

Прежние версии HTTP задавали конкретное число одновременно открытых соединений, но это не подошло для многих приложений. Поэтому данные спецификация не задаёт конкретного максимума для числа соединений, взамен предлагая клиентам быть консервативными при создании множества соединений. Несколько соединений обычно используется для предотвращения блокировки head-of-line, когда требующий значительной обработки и/или передачи очень большого объёма информации запрос может препятствовать обработке последующих запросов в том же соединении. Однако для каждого соединения нужно выделять ресурсы. Кроме того, использование множества соединений может приводить к нежелательным побочным эффектам в загруженных сетях, а также в иных ситуациях, поскольку совокупная, изначально синхронизированная передача может вызывать перегрузку, которая не возникла бы при меньшем числе соединений.

Отметим, что сервер может отвергать трафик, который он считает неправомерным или воспринимает как DoS-атаку, например, чрезмерное число соединений, организованных одним клиентом.

9.5. Отказы и тайм-ауты

У серверов обычно имеется тот или иной тайм-аут, по достижении которого они прекращают поддержку неактивных соединений. Прокси-серверы могут использовать большее значение, поскольку вероятно, что клиент может организовывать больше соединений через один прокси-сервер. Использование сохраняемых соединений не задаёт требований к наличию и величине такого тайм-аута как для клиентов, так и для серверов.

Клиенту или серверу, желающему закрыть соединение по тайм-ауту, **следует** выполнить аккуратное завершение соединения. Реализациям **следует** постоянно отслеживать открытые соединения на предмет получения сигналов о закрытии и подобающей реакции на них, поскольку оперативное закрытие соединений обеими сторонами позволяет освободить выделенные для соединений ресурсы.

Клиент, сервер или прокси **может** закрыть транспортное соединение в любой момент. Клиент может начать передачу нового запроса в тот момент, когда сервер решит закрыть бездействующее (idle) соединение. С точки зрения сервера соединение закрывается во время бездействия, а с точки зрения клиент - при запросе.

Серверу **следует** поддерживать сохраняемые соединения, пока это возможно, и разрешать механизмам управления потоком данных базового транспорта обрабатывать временные перегрузки вместо прерывания соединений, ожидая повтора попытки клиентом. Прерывание соединения может усугублять перегрузку сети или загрузку сервера.

Клиенту, передающему тело сообщения, **следует** отслеживать сетевое соединение на предмет сообщений об ошибках в процессе передачи запроса. Если клиент видит отклик, который указывает, что сервер не хочет получать тело сообщения и закрывает соединение, ему **следует** незамедлительно прекратить отправку тела сообщения и закрыть соединение.

9.6. Разрыв соединения

Опция соединения close задана в качестве сигнала о том, что отправитель будет закрывать соединение после выполнения запроса. Отправителю **следует** передавать поле заголовка Connection (параграф 7.6.1 в [HTTP]) с опцией соединения close, когда он намерен закрыть соединение. Например,

```
Connection: close
```

в заголовке запроса указывает, что это последний запрос, передаваемый клиентом в соединении, а в отклике это поле означает, что сервер закроет соединение по завершении отклика. Отметим, что имя поля Close зарезервировано, поскольку его использование в заголовке может конфликтовать с опцией соединения close.

Клиенту, передавшему опцию соединения close, **недопустимо** передавать последующие запросы в это соединение и он **должен** закрыть соединение после приёма финального сообщения с откликом, соответствующего этому запросу. Сервер, получивший опцию соединения close, **должен** инициировать закрытие соединения (см. ниже) после отправки финального отклика на запрос с опцией close. Серверу **следует** передавать опцию соединения close в финальном отклике в данном соединении и **недопустимо** обрабатывать какие-либо запросы, полученные через это соединение. Сервер, передавший опцию соединения close, **должен** инициировать закрытие соединения (см. ниже) после отправки отклика с опцией close и **недопустимо** обрабатывать какие-либо запросы, полученные через это соединение.

Клиент, получивший опцию соединения close, **должен** прекратить отправку запросов по этому соединению и закрыть его после прочтения отклика с опцией close. Если в это соединение были переданы дополнительные конвейерные запросы, клиенту **не следует** предполагать их обработку сервером.

При незамедлительном закрытии соединения сервером возникает риск того, что клиент не сможет прочитать последний отклик HTTP. Если сервер получит от клиента дополнительные данные через полностью закрытое соединение (например, другой запрос, переданный клиентом до получения отклика от сервера), серверный стек TCP будет передавать клиенту пакет сброса, который, к сожалению, может удалить у клиента неподтвержденные входные буферы до их прочтения и интерпретации клиентским анализатором HTTP. Для предотвращения проблемы сброса TCP серверы обычно закрывают соединение поэтапно. Сначала сервер закрывает только строну записи (half-close) с соединении с чтением и записью, продолжая читать данные из соединения, пока не получит от клиента соответствующее закрытие или не будет уверен, что его стек TCP получил от клиента подтверждения пакетов с последним откликом сервера. После этого сервер закрывает соединение полностью. Сведений о возникновении проблемы сброса а других протоколах (не TCP) нет.

Отметим, что закрытие клиентом соединения TCP наполовину, не означает, что этот клиент больше не заинтересован в отклике. В общем случае на транспортные сигналы полагаться нельзя, поскольку HTTP/1.1 не зависит от транспорта.

9.7. Инициирование соединения TLS

Концептуально HTTP/TLS - это просто передача сообщений HTTP через соединения, защищённые TLS [TLS13].

Клиент HTTP является также клиентом TLS. Он инициирует соединение с сервером через подходящий порт и передаёт сообщение TLS ClientHello для начала согласования TLS. По завершении согласования TLS клиент может инициировать первый запрос HTTP. Все данные HTTP **должны** передаваться как данные приложения TLS, а в остальном это рассматривается как обычное соединение HTTP (включая возможность многократного использования сохраняемых соединений).

9.8. Закрытие соединения TLS

TLS использует обмен сигналами (не ошибка) о закрытии соединения, что это было безопасно (параграф 6.1 в [TLS13]). При получении действительного сигнала о закрытии, реализация может быть уверена, что не получит данных через это соединение. Когда реализация знает, что она больше не будет передавать или принимать данные сообщений, в которых она заинтересована (обычно путём обнаружения границ сообщений HTTP), она может генерировать»неполное закрытие» (incomplete close), передавая сигнал о закрытии и закрывая соединение без получения соответствующего сигнала о закрытии от своего партнёра.

Неполное закрытие не ставит под сомнение безопасность уже полученных данных, но может говорить о возможности отсечки последующих данных. Поскольку TLS не знает напрямую о кадрировании сообщений HTTP, нужно проверять сами данные HTTP для определения полноты сообщений. Обработка неполных сообщений описана в раздел 8.

Клиенту, столкнувшемуся с неполным завершением, **следует** считать завершёнными все отклики, для которых получено что-либо из указанного:

1. объем данных, указанный полем Content-Length;
2. завершающий блок нулевого размера (при использовании блочного транспортного кодирования).

Отклик, для которого не используется блочного транспортного кодирования и Content-Length, считается полным лишь при получении действительного сигнала о закрытии. Признание неполного сообщения полным может открывать реализацию для атак.

Клиенту, обнаружившему неполное закрытие, **следует** выполнить аккуратное восстановление.

Клиент **должен** передать сигнал о закрытии перед закрытием соединения. Клиент, не ожидающий приёма дополнительных данных **может** не ждать от сервера сигнала закрытия и просто закрыть соединение, вызывая неполное закрытие на стороне сервера.

Серверу **следует** быть готовым к получению от клиента неполного закрытия, поскольку клиент зачастую может обнаружить окончание данных от сервера.

Сервер **должен** пытаться инициировать обмен сигналами о закрытии с клиентом до закрытия соединения. Сервер **может** закрыть соединение после передачи сигнала о закрытии, вызывая неполное закрытие на стороне клиента.

10. Встраивание сообщений как данных

10.1. Тип носителя message/http

Тип носителя message/http может служить для передачи одного сообщения с запросом или откликом HTTP при условии соблюдения ограничений MIME для типов message в части размера и кодирования. Из-за ограничений на размер строк для типа message/http разрешается применять фальцовку строк (obs-fold), как описано в параграфе 5.2, для передачи значения поля в нескольких строках. Получатель данных message/http при восприятии сообщения **должен** заменять устаревшую фальцовку строк одним или несколькими символами SP.

Имя типа
message
Имя субтипа
http
Требуемые параметры
нет
Необязательные параметры
version, msgtype

version
Номер версии HTTP во вложенном сообщении (например, 1.1). Если номер не указан, версию можно определить из первой строки тела.
msgtype
Тип сообщения - request или response. Если тип не указан, его можно определить из первой строки тела.

Кодировки
Только 7bit, 8bit или binary.

Вопросы безопасности
См. раздел 11.

Вопросы совместимости
Нет

Опубликованная спецификация
RFC 9112 (параграф 10.1).

Приложения, использующие этот тип носителя
Не заданы.

Вопросы идентификации фрагментов
Не применимо.

Дополнительные сведения
Magic number(s)
Не применимо.
Запрещённые имена псевдонимов
Не применимо.
Расширения имён файлов
Не применимо.
Коды типа файлов Macintosh
Не применимо.

Контактные данные для получения дополнительных сведений
См. раздел «Адреса авторов».

Предусмотренное использование
Общее назначение

Ограничения на использование
Нет

Автор
См. раздел «Адреса авторов».

Контролёр изменений
IESG

10.2. Тип носителя application/http

Тип носителя application/http может служить для передачи вложенного конвейера с одним или несколькими запросами или откликами HTTP (без смешивания).

Имя типа
message

Имя субтипа
http

Требуемые параметры
нет

Необязательные параметры
version, msgtype
version
Номер версии HTTP во вложенном сообщении (например, 1.1). Если номер не указан, версию можно определить из первой строки тела.
msgtype
Тип сообщения - request или response. Если тип не указан, его можно определить из первой строки тела.

Кодировки
HTTP messages enclosed by this type are in "binary" format; use of an appropriate Content-Transfer-Encoding is required when transmitted via email.

Вопросы безопасности
См. раздел 11.

Вопросы совместимости
Нет

Опубликованная спецификация
RFC 9112 (параграф 10.2).

Приложения, использующие этот тип носителя
Не заданы.

Вопросы идентификации фрагментов
Не применимо.

Дополнительные сведения
Запрещённые имена псевдонимов
Не применимо.
Magic number(s)
Не применимо.
Расширения имён файлов
Не применимо.
Коды типа файлов Macintosh
Не применимо.

Контактные данные для получения дополнительных сведений
См. раздел «Адреса авторов».

Предусмотренное использование
Общее назначение
Ограничения на использование
Нет
Автор
См. раздел «Адреса авторов».
Контролёр изменений
IESG

11. Вопросы безопасности

Этот раздел информирует разработчиков, поставщиков информации и пользователей о вопросах безопасности, связанных с синтаксисом и обработкой сообщений HTTP. Вопросы безопасности, связанные семантиков, содержимым и маршрутизацией HTTP, рассмотрены [HTTP].

11.1. Расщепление откликов

Расщепление откликов (внедрение CRLF) - это распространённый метод, используемый в разных атаках на Web и основанный на построчной природе сообщений HTTP и упорядоченной связи упорядоченных запросов с откликами в сохраняемых соединениях [Klein]. Этот метод может быть особо опасным при прохождении запросов через общий кэш.

Расщепление откликов использует уязвимость серверов (обычно серверов приложений), где злоумышленник может передать закодированные данные внутри того или иного параметра запроса, который будет потом декодирован и возвращён (echo) в любом из полей заголовка отклика. Если декодированные данные выглядят как завершение отклика и начало следующего, значит отклик расщеплён и содержимое и содержимое того, что представляется вторым откликом, контролируется атакующим. Затем злоумышленник может выполнить любой другой запрос по тому же сохраняемому соединению и обмануть получателей (включая посредников), заставив их поверить, что вторая половина разделённого отклика является полномочным откликом на второй запрос. Например, параметр в request-target может быть прочитан сервером приложения и снова использован в перенаправлении, в результате чего параметр будет повторён в поле заголовка отклика Location. Если параметр декодируется приложением и не закодирован должным образом при включении в поле отклика, атакующий может внедрить закодированные октеты CRLF и другое содержимое, что сделает один отклик приложения выглядящим как два или несколько откликов.

Обычная защита от расщепления откликов заключается в фильтрации запросов на предмет данных, представляющихся закодированными CR и LF (например, %0D и %0A). Однако это предполагает, что сервер приложений декодирует лишь URI, а не менее понятные преобразования данных, такие как смена кодировки, трансляция сущностей XML, декодирование base64, переформатирование sprintf и т. п. Более эффективная защита обеспечивается предотвращением отправки CR и LF в разделе заголовков чем-либо, кроме библиотек ядра протокола сервера, что означает возможность вывода полей заголовков только для API, которые фильтруют непригодные октеты и не позволяют серверам приложений напрямую писать в поток протокола.

11.2. Контрабанда запросов

Контрабандой (smuggling) запросов [Linhart] называют метод, использующий различия при разборе у разных получателей для сокрытия дополнительных запросов (которые иначе могут блокироваться или запрещаться правилами) в безобидном на первый взгляд запросе. Как и расщепление откликов, контрабанда запросов может приводить к различным атакам на HTTP. В этой спецификации заданы новые требования к разбору запросов, в частности, к кадрированию сообщений (параграф 6.3) для снижения эффективности контрабанды запросов.

11.3. Целостность сообщений

HTTP не задаёт конкретных механизмов обеспечения целостности сообщений, полагаясь на возможности обнаружения ошибок в базовых транспортных протоколах и использование размера или блочного кодирования для определения полноты. Исторически отсутствие одного механизма контроля целостности оправдывалось неформальной природой большинства взаимодействий HTTP. Однако широкое распространение HTTP как механизма доступа к информации привело к расширению использования в средах, где проверка целостности сообщений имеет решающее значение.

Механизмы, обеспечиваемые схемой https, такие как аутентифицированное шифрование, обеспечивают защиту от изменения сообщений. Однако требуется следить за тем, чтобы закрытие соединения не могло использоваться для отсечки сообщений (см. параграф 9.8). Пользовательские агенты могут отказываться воспринимать неполные сообщения или обрабатывать их особо. Например, браузер, служащий для просмотр истории болезни или сведений о взаимодействии лекарств, должен информировать пользователя, когда протокол обнаруживает, что сведения неполны, устарели или повреждены при передаче. Такие механизмы можно включать селективно через расширения пользовательского агента или включение в отклик метаданных для контроля целостности.

Схема http не обеспечивает защиты от случайного или преднамеренного изменения сообщений.

Расширения протокола могут служить для снижения риска нежелательного изменения сообщений посредниками даже при использовании схемы https. Целостность можно обеспечить с помощью кодов аутентификации, добавляемых в сообщения через расширяемые поля метаданных.

11.4. Конфиденциальность сообщений

HTTP полагается на базовый транспортный протокол для обеспечения конфиденциальности, когда это желательно. Протокол HTTP специально создан независимым от транспортного протокола, чтобы его можно было использовать со многими формами шифрованных соединений, выбор которых определяет схема URI или конфигурация пользовательского агента. Схему https можно применять для указания ресурсов, которые требуют конфиденциального соединения, как описано в параграфе 4.2.2 [HTTP].

12. Взаимодействие с IANA

Контролёром изменений для указанных ниже регистраций является IETF (iesg@ietf.org).

12.1. Регистрация имён полей

Агентство IANA добавило имена полей, приведённые в таблице 1, в реестр Hypertext Transfer Protocol (HTTP) Field Name Registry (https://www.iana.org/assignments/http-fields), как указано в параграфе 18.4 [HTTP].

Таблица 1.

Имя поля	*Статус*	*Параграф*	*Комментарии*
Close	permanent	9.6	(резерв)
MIME-Version	permanent	B.1	
Transfer-Encoding	permanent	6.1	

12.2. Регистрация типов носителей

Агентство IANA обновило реестр Media Types (https://www.iana.org/assignments/media-types) с добавлением записей, указанных в параграфах 10.1 и 10.2 для типов носителя message/http и application/http.

12.3. Регистрация транспортного кодирования

Агентство IANA обновило реестр HTTP Transfer Coding Registry (https://www.iana.org/assignments/http-parameters/) с процедурой регистрации, указанной в параграфе 7.3 и именами кодирования содержимого из таблицы 2.

Таблица 2.

Имя	*Описание*	*Параграф*
chunked	Передача в последовательности блоков (chunk)	7.1
compress	Формат данных UNIX compress [Welch]	7.2
deflate	Формат сжатия данных deflate [RFC1951] внутри формата данных zlib [RFC1950]	7.2
gzip	Формат файлов GZIP [RFC1952]	7.2
trailers	Резерв	12.3
x-compress	Устарело (псевдоним для compress)	7.2
x-gzip	Устарело (псевдоним для gzip)	7.2

Примечание. Имя кодирования trailers зарезервировано для предотвращения конфликтов с ключевым словом trailers в поле заголовка TE (параграф 10.1.4 в [HTTP]).

12.4. Регистрация ALPN Protocol ID

Агентство IANA обновило реестр TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs (https://www.iana.org/assignments/tls-extensiontype-values/) приведённым в таблице 3 значением.

Таблица 3.

Протокол	*Идентификационная сигнатура*	*Документ*
HTTP/1.1	0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31 (http/1.1)	RFC 9112

13. Литература

13.1. Нормативные документы

[CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/info/rfc1950>.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <https://www.rfc-editor.org/info/rfc1951>.

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, DOI 10.17487/RFC1952, May 1996, <https://www.rfc-editor.org/info/rfc1952>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/info/rfc7405>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[Welch] Welch, T., "A Technique for High-Performance Data Compression", IEEE Computer 17(6), DOI 10.1109/MC.1984.1659158, June 1984, <https://ieeexplore.ieee.org/document/1659158/>.

13.2. Дополнительная литература

[HTTP/1.0] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, <https://www.rfc-editor.org/info/rfc1945>.

[Klein] Klein, A., "Divide and Conquer - HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics", March 2004, <https://packetstormsecurity.com/papers/general/whitepaper_httpresponse.pdf>.

[Linhart] Linhart, C., Klein, A., Heled, R., and S. Orrin, "HTTP Request Smuggling", June 2005, <https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, DOI 10.17487/RFC2049, November 1996, <https://www.rfc-editor.org/info/rfc2049>.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, "Hypertext Transfer Protocol — HTTP/1.1", RFC 2068, DOI 10.17487/RFC2068, January 1997, <https://www.rfc-editor.org/info/rfc2068>.

[RFC2557] Palme, J., Hopmann, A., and N. Shelness, "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, DOI 10.17487/RFC2557, March 1999, <https://www.rfc-editor.org/info/rfc2557>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

Приложение A. Сводка ABNF

В приведённой ниже сводке ABNF указаны правила below, расширяемые в соответствии с параграфом 5.6.1 в [HTTP].

```
BWS = <BWS, см. [HTTP], параграф 5.6.3>
HTTP-message = start-line CRLF *( field-line CRLF ) CRLF [
 message-body ]
HTTP-name = %x48.54.54.50 ; HTTP
HTTP-version = HTTP-name "/" DIGIT "." DIGIT

OWS = <OWS, см. [HTTP], параграф 5.6.3>

RWS = <RWS, см. [HTTP], параграф 5.6.3>

Transfer-Encoding = [ transfer-coding *( OWS "," OWS transfer-coding
 ) ]

absolute-URI = <absolute-URI, см. [URI], параграф 4.3>
absolute-form = absolute-URI
absolute-path = <absolute-path, см. [HTTP], параграф 4.1>
asterisk-form = "*"
authority = <authority, см. [URI], параграф 3.2>
authority-form = uri-host ":" port

chunk = chunk-size [ chunk-ext ] CRLF chunk-data CRLF
chunk-data = 1*OCTET
chunk-ext = *( BWS ";" BWS chunk-ext-name [ BWS "=" BWS chunk-ext-val
 ] )
chunk-ext-name = token
chunk-ext-val = token / quoted-string
chunk-size = 1*HEXDIG
chunked-body = *chunk last-chunk trailer-section CRLF

field-line = field-name ":" OWS field-value OWS
field-name = <field-name, см. [HTTP], параграф 5.1>
field-value = <field-value, см. [HTTP], параграф 5.5>

last-chunk = 1*"0" [ chunk-ext ] CRLF

message-body = *OCTET
method = token

obs-fold = OWS CRLF RWS
obs-text = <obs-text, см. [HTTP], параграф 5.6.4>
origin-form = absolute-path [ "?" query ]

port = <port, см. [URI], параграф 3.2.3>

query = <query, см. [URI], параграф 3.4>
quoted-string = <quoted-string, см. [HTTP], параграф 5.6.4>

reason-phrase = 1*( HTAB / SP / VCHAR / obs-text )
```

```
request-line = method SP request-target SP HTTP-version
request-target = origin-form / absolute-form / authority-form /
 asterisk-form

start-line = request-line / status-line
status-code = 3DIGIT
status-line = HTTP-version SP status-code SP [ reason-phrase ]

token = <token, см. [HTTP], параграф 5.6.2>
trailer-section = *( field-line CRLF )
transfer-coding = <transfer-coding, см. [HTTP], параграф 10.1.4>

uri-host = <host, см. [URI], параграф 3.2.2>
```

Приложение B. Различия между HTTP и MIME

В HTTP/1.1 используются многие конструкции, определённые для формата сообщений Internet [RFC5322] и многоцелевых почтовых расширений (Multipurpose Internet Mail Extensions или MIME) [RFC2045], чтобы тело сообщения можно было передавать в разных открытых представлениях с расширяемыми полями. Однако некоторые из таких конструкций были переосмыслены для лучшего соответствия потребностям интерактивных коммуникаций, что привело к некоторым отличиям при обработке конструкций MIME в HTTP. Эти отличия были внимательно рассмотрены для оптимизации производительности по сравнению с двоичными соединениями, предоставления большей свободы использования новых типов носителей, упрощения сравнения данных и адаптации к базовым реализациям.

В этом приложении описаны конкретные аспекты отличий HTTP от MIME. Прокси и шлюзы в среде со строгими требованиями MIME должны учитывать эти различия и при необходимости выполнять соответствующие преобразования.

B.1. MIME-Version

Протокол HTTP не соответствует MIME, однако сообщения могут включать одно поле заголовка MIME-Version для указания версии протокола MIME, использованной для создания сообщения. Поле MIME-Version говорит, что сообщение полностью совместимо с протоколом MIME (как задано в [RFC2045]). Отправитель отвечает за обеспечения полного соответствия (когда это возможно) при экспорте сообщений HTTP в строгие среды MIME.

B.2. Преобразование в каноническую форму

MIME требует преобразования части тела сообщения почты Internet преобразовывалась в каноническую форму до передачи, как указано в разделе 4 [RFC2049], а в содержимом типа text требуется представлять перевод строк символами CRLF с запретом использования CR и LF вне переноса строк [RFC2046]. HTTP не запрещает применять CRLF и отдельные символы CR или LF для указания завершения строки в содержимом.

Прокси и шлюзы HTTP, передающие в строгую среду MIME должны транслировать все переводы строк внутри носителя типа text в каноническую форму RFC 2049 (CRLF). Однако это может осложняться наличием поля Content-Encoding и тем, что HTTP разрешает использовать некоторые кодировки, где не применяются октеты 13 и 10 для представления CR и LF, соответственно.

Преобразование будет приводить к некорректности криптографических контрольных сумм исходного содержимого, если оно не имело канонической формы. Поэтому рекомендуется применять каноническую форму для любого содержимого, использующего такие контрольные суммы в HTTP.

B.3. Преобразование форматов дат

В HTTP/1.1 применяется ограниченный набор форматов дат (параграф 5.6.7 в [HTTP]) для упрощения сравнения дат. Прокси и шлюзы, принимающие от других протоколов, должны поверять соответствие полей заголовка Date в сообщениях одному из форматов HTTP/1.1 и при необходимости преобразовывать поля.

B.4. Преобразование Content-Encoding

В MIME нет эквивалента поля заголовка HTTP Content-Encoding. Поскольку это поле служит модификатором типа носителя, прокси и шлюзы из HTTP в совместимые с MIME протоколы должны менять значение поля заголовка Content-Type или декодировать представление до пересылки сообщения. Некоторые экспериментальные варианты применения поля Content-Type для почты Internet используют параметр media-type ;conversions=<content-coding> для выполнения функции, эквивалентной Content-Encoding, однако этот параметр не является стандартным для MIME.

B.5. Преобразование Content-Transfer-Encoding

В HTTP не применяется поле Content-Transfer-Encoding из MIME. Прокси и шлюзы из совместимого с MIME протокола в HTTP должны удалять Content-Transfer-Encoding перед доставкой сообщения с откликом клиенту HTTP. Прокси и шлюзы из HTTP в совместимые с MIME протоколы отвечают за корректность формата сообщения и кодирование для безопасной доставки по соответствующему протоколу (безопасность транспорта определяется ограничениями применяемого протокола). Такие прокси и шлюзы должны преобразовывать и помечать данные соответствующим значением Content-Transfer-Encoding, если это повышает вероятность безопасной доставки по соответствующему протоколу.

B.6. MHTML и ограничения размера строк

Реализации HTTP, имеющие общий код с реализациями MHTML [RFC2557], должны принимать во внимание ограничения MIME на размер строк. Поскольку в HTTP таких ограничений нет, фальцовка строк в HTTP не применяется. Сообщения MHTML, передаваемые через HTTP, соответствуют всем соглашениям MHTML, включая ограничение размера строк, фальцовку и т. п., поскольку HTTP доставляет тело сообщений без изменения и (за исключением типа multipart/byteranges, см. параграф 14.6 в [HTTP]) не интерпретирует содержимое и строки заголовков MIME, которые могут содержаться в нём.

Приложение C. Отличия от предшествующих RFC

C.1. Отличия от HTTP/0.9

Поскольку в HTTP/0.9 не поддерживаются поля заголовков запроса, здесь нет возможности поддерживать виртуальные хосты по именам (выбор ресурса по значению поля Host). Серверы, реализующие поддержку виртуальных хостов по именам, должны отключить использование HTTP/0.9. Большинство запросов, выглядящих как HTTP/0.9, на деле являются некорректно построенными запросами HTTP/1.x, обусловленными неверным кодированием клиентом цели запроса (request-target).

C.2. Отличия от HTTP/1.0

C.2.1. Многодомные Web-серверы

Требование к клиентам и серверам поддерживать поле заголовка Host (параграф 7.2 в [HTTP]), сообщать об ошибке при его отсутствии в запросе HTTP/1.1 и воспринимать абсолютные URI (параграф 3.2), является наиболее важным изменением, внесённым HTTP/1.1.

Старые клиенты HTTP/1.0 предполагали взаимно-однозначное соответствие адресов IP и серверов и не было механизма определить нужный сервер кроме адреса IP, по которому передан запрос. Поле заголовка Host было добавлено при разработке HTTP/1.1 и, хотя оно было быстро реализовано в большинстве браузеров HTTP/1.0, для полной адаптации к HTTP/1.1 были внесены дополнительные требования ко всем запросам. На момент создания этого документа большинство служб на основе HTTP уже зависело от поля Host для нацеливания запросов.

C.2.2. Соединения Keep-Alive

В HTTP/1.0 каждое соединение организуется клиентом перед запросом и закрывается сервером после отправки отклика. Однако некоторые реализации применяли явно согласованный (Keep-Alive) вариант сохраняемых соединений, описанный в параграфе 19.7.1 [RFC2068]. Некоторые клиенты и серверы могут захотеть совместимости с этими подходами к постоянным соединениям, явно согласовывая их с помощью поля заголовка запроса Connection: keep-alive. Однако часть экспериментальных реализаций сохраняемых соединений HTTP/1.0 оказалась ошибочной, например, при непонимании прокси-сервером HTTP/1.0 поля Connection он мог ошибочно пересылать поле заголовка следующему приёмному (inbound) серверу, что вело к зависанию (hung) соединения.

Одной из попыток решения было введение поля заголовка Proxy-Connection, предназначенного специально для прокси. На деле это оказалось неэффективным, поскольку прокси зачастую работают на нескольких уровнях, что снова ведёт к возникновению описанной выше проблемы. В результате клиентам рекомендуется не применять поле Proxy-Connection в каких-либо запросах.

Клиентам также рекомендуется аккуратно относиться к использованию поля Connection: keep-alive в запросах. Хотя оно и позволяет организовать постоянные соединения с серверами HTTP/1.0, использующие их клиенты должны следить за «повисшими» (hung) соединениями (они указывают, что клиент должен прекратить отправку этого поля) и этот механизм вообще не должен использоваться клиентами при работе через прокси.

C.2.3. Введение поля Transfer-Encoding

В HTTP/1.1 добавлено поле заголовка Transfer-Encoding (параграф 6.1). Транспортное кодирование должно декодироваться до пересылки сообщения HTTP по протоколу, соответствующему MIME.

C.3. Отличия от RFC 7230

Большинство разделов, посвящённых целям разработки HTTP, истории, архитектуре, критериям соответствия, версиям протокола, URI, маршрутизации сообщений и полям заголовков, перенесены в [HTTP]. Этот документ был сведён к синтаксису сообщений и требованиям к поддержке соединений, характерным для HTTP/1.1.

Использование «голых» символов CR вне содержимого запрещено (параграф 2.2).

Изменено определение ABNF для authority-form с заменой общей формы компонента authority в URI (с необязательным указанием порта) на конкретную форму host:port, требуемую для CONNECT (параграф 3.2.3).

Получатели должны избегать атак с расщеплением и контрабандой запросов при обработке неоднозначного кадрирования сообщений (параграф 6.1).

В ABNF для блочных (chunk) расширений снова добавлены «плохие» пробелы вокруг символов ; и =, исключённые в [RFC7230], поскольку это исключение стало препятствием для имеющихся реализаций (параграф 7.1.1).

Семантика полей трейлера выходит за рамки блочного транспортного кодирования. Был обновлён алгоритм декодирования для chunked (параграф 7.1.3), чтобы поощрять сохранение/пересылку полей трейлера отдельно от полей заголовка, разрешать слияние заголовка и трейлера, когда получатель знает, что определения соответствующих полей позволяют это и отбрасывать поля трейлера в ином случае. Трейлерная часть сейчас называется разделом трейлеров, чтобы быть более согласованной с разделом заголовков и отличаться от тела сообщения (параграф 7.1.2).

Запрещены параметры транспортного кодирования с именем q во избежание путаницы с использованием рангов в поле заголовка TE (параграф 7.3).

Благодарности

См. одноимённый раздел в [HTTP].

Предметный указатель

Адреса авторов

Roy T. Fielding (editor)
Adobe
345 Park Ave
San Jose, CA 95110
United States of America
Email: fielding@gbiv.com
URI: https://roy.gbiv.com/

Mark Nottingham (editor)
Fastly
Prahran
Australia
Email: mnot@mnot.net
URI: https://www.mnot.net/

Julian Reschke (editor)
greenbytes GmbH
Hafenweg 16
48155 Münster
Germany
Email: julian.reschke@greenbytes.de
URI: https://greenbytes.de/tech/webdav/

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru