Internet Engineering Task Force (IETF)
Request for Comments: 9110
STD: 97
Obsoletes: 2818, 7230, 7231, 7232, 7233, 7235, 7538, 7615, 7694
Updates: 3864
Category: Standards Track
ISSN: 2070-1721

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

June 2022

HTTP Semantics

Семантика HTTP

Аннотация

HTTP[1] - это протокол прикладного уровня без поддержки состояний для распределенных систем гипертекстовой информации совместного использования. Этот документ описывает общую архитектуру HTTP, задаёт базовую терминологию и общие для всех версий аспекты протокола. В определения включены базовые элементы протокола, механизмы расширения и схемы унифицированных идентификаторов ресурсов (Uniform Resource Identifier или URI) http и https.

Этот документ обновляет RFC 3864 и отменяет RFC 2818, 7231, 7232, 7233, 7235, 7538, 7615, 7694, и часть 7230.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[2] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[3]. 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/rfc9110.

Оглавление

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

1. Введение

1.1. Назначение

HTTP - это семейство протоколов прикладного уровня без поддержки состояния, использующих модель «запрос-отклик» и имеющих единый базовый интерфейс, расширяемую семантику и самоописывающиеся сообщения для обеспечения гибкого взаимодействия с сетевыми информационными системами на основе гипертекста.

HTTP скрывает детали реализации сервиса, предоставляя клиентам унифицированный интерфейс, не зависящий от типа представляемых ресурсов. Серверам не нужно знать о целях каждого клиента, запросы могут рассматриваться изолированно без привязки к определённому типу клиента или предопределённой последовательности действий приложения. Это позволяет эффективно применять реализации общего назначения в разном контексте, упрощает взаимодействия и обеспечивает независимое развитие с течением времени.

HTTP также предназначен для использования в качестве промежуточного протокола (посредника), когда прокси и шлюзы могут транслировать отличные от HTTP информационные системы через базовый интерфейс.

Одним из следствий такой гибкости является невозможность определения протокола в терминах происходящего за интерфейсом. Вместо этого определяется синтаксис взаимодействий, намерения и предполагаемое поведение

получателя. Если взаимодействия рассматриваются изолированно, успешные действия должны отражаться соответствующими изменениями наблюдаемого интерфейса сервера. Однако одновременно (параллельно) может действовать множество клиентов, не согласованных между собой, невозможно требовать наблюдаемости таких изменений за рамками одного отклика.

1.2. История развития

HTTP является основным протоколом передачи информации по «всемирной паутине» (World Wide Web или WWW) с момента появления в 1990 г. Сначала это был тривиальный механизм для запросов с малой задержкой и одним методом (GET) для запроса передачи предусмотренного текстового документа, идентифицируемого путём к нему. По мере развития Web протокол HTTP был расширен для размещения запросов и откликов в сообщениях, передачи произвольных форматов данных с использованием типа носителя в стиле MIME и маршрутизации запросов через промежуточные узлы (посредники). В итоге эти протоколы были определены как HTTP/0.9 и HTTP/1.0 (см. [HTTP/1.0]).

Протокол HTTP/1.1 был разработан для совершенствования функций протокола при сохранении имеющегося синтаксиса на основе текстовых сообщений, что позволило улучшить функциональную совместимость, расширяемость и отказоустойчивость в Internet. Были добавлены разделитель данных по размеру для фиксированного и динамического (chunk) содержимого, единая модель для согласования содержимого, неанализируемые (opaque) валидаторы для условных запросов, средства управления кэшированием для лучшей согласованности, запросы диапазонов для частичных обновлений и принятые по умолчанию сохраняющиеся (persistent) соединения. Протокол HTTP/1.1 был представлен в 1995 г. и опубликован как Standards Track в 1997 г. [RFC2068], а затем пересмотрен в 1999 г. [RFC2616] и в 2014 ([RFC7230] - [RFC7235]).

В HTTP/2 ([HTTP/2]) добавлен мультиплексируемый сеансовый уровень поверх имеющихся протоколов TLS и TCP для обмена одновременными (параллельными) сообщениями HTTP с эффективным сжатием полей и выталкиванием (push) с сервера. HTTP/3 ([HTTP/3]) улучшает независимость одновременных соединений за счёт применения QUIC в качестве защищённого мультиплексируемого транспорта по протоколу UDP вместо TCP.

Во всех трёх основных версиях HTTP применяется семантика, описанная в этом документе. Версии протокола не вытесняют одна другую, поскольку у каждой имеются свои преимущества и ограничения в зависимости от контекста применения. Предполагается, что реализации будут выбирать наиболее подходящий транспорт и синтаксис обмена сообщениями для своего конкретного контекста.

В данной редакции HTTP определение семантики (этот документ) и кэширования ([CACHING]) отделено от текущего синтаксиса обмена сообщениями HTTP/1.1 ([HTTP/1.1]) для независимого развития каждой из основных версий протокола с общей базовой семантикой.

1.3. Основная семантика

HTTP обеспечивает унифицированный интерфейс для взаимодействия с ресурсом (3.1. Ресурсы) путем передачи сообщений, которые манипулируют представлением или переносят его (3.2. Представления), независимо от типа, природы и реализации.

Каждое сообщение является запросом или откликом. Клиент создаёт запросные сообщения, указывающие его намерения, и маршрутизирует эти сообщения в направлении идентифицированного сервера-источника. Сервер прослушивает запросы, анализирует каждое принятое сообщение, интерпретирует семантику сообщения применительно к указанному целевому ресурсу и отвечает на запрос одним или несколькими сообщениями с откликами. Клиент проверяет полученные отклики на предмет соответствия его намерениям, определяя дальнейшие действия на основе полученного кода статуса и содержимого.

Семантика HTTP включает намерения, определяемые каждым методом запроса (9. Методы), расширения, которые могут быть описаны в полях заголовка запросов, коды статуса, описывающие отклик (15. Коды статуса), и другие данные управления и метаданные ресурса, которые могут быть указаны в полях отклика. Семантика также включает метаданные представления, описывающие, как содержимое следует интерпретировать получателю, поля заголовков запроса, которые могут влиять на выбор содержимого, и различные алгоритмы выбора, которые совместно называют согласованием содержимого (12. Согласование содержимого).

1.4. Спецификации, отменённые этим документом

Таблица 1.

Название	***Документ***	***Дополнительные сведения***
HTTP Over TLS	[RFC2818]	B.1. Отличия от RFC 2818
HTTP/1.1 Message Syntax and Routing[1]	[RFC7230]	B.2. Отличия от RFC 7230
HTTP/1.1 Semantics and Content	[RFC7231]	B.3. Отличия от RFC 7231
HTTP/1.1 Conditional Requests	[RFC7232]	B.4. Отличия от RFC 7232
HTTP/1.1 Range Requests	[RFC7233]	B.5. Отличия от RFC 7233
HTTP/1.1 Authentication	[RFC7235]	B.6. Отличия от RFC 7235
HTTP Status Code 308 (Permanent Redirect)	[RFC7538]	B.7. Отличия от RFC 7538
HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields	[RFC7615]	B.8. Отличия от RFC 7615
HTTP Client-Initiated Content-Encoding	[RFC7694]	B.9. Отличия от RFC 7694

2. Соответствия

2.1. 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, которое позволяет компактно определять списки, разделяемых запятыми элементов с использованием оператора # (подобно указанию повтора оператором *). В

[1]Этот документ отменяет лишь части RFC 7230, не зависящие от синтаксиса обмена сообщениями и управления соединениями HTTP/1.1, остальные части 7230 отменяет [HTTP/1.1].

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).

В параграфе 5.6 определены некоторые базовые компоненты синтаксиса для значений полей.

В этой спецификации применяются термины character (символ), character encoding scheme (схема кодирования символов), charset (набор символов), protocol element (элемент протокола) в соответствии с [RFC6365].

2.2. Обозначение 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.

Данная спецификация нацеливает критерии соответствия в зависимости от роли участника коммуникаций HTTP. Поэтому требования предъявляются к отправителям, получателям, клиентам, серверам, пользовательским агентам, посредникам, серверам источникам, прокси, шлюзам и системам кэширования в зависимости от того, какое поведение будет ограничиваться требованиями. Дополнительные требования предъявляются к реализациям, владельцам ресурсов и регистрации элементов протокола, если они выходят за рамки отдельного взаимодействия.

Термин generate (генерировать) применяется вместо send (передавать), когда требования предъявляются только к реализации, создающей элемент протокола, а не к реализации, направляющей полученный элемент в нисходящем направлении (downstream).

Реализация считается соответствующей, если она удовлетворяет всем требованиям, связанных с ролями, которые она играет в HTTP.

Отправителю **недопустимо** генерировать элементы протокола, которые не соответствуют грамматике, заданной соответствующими правилами ABNF. В рамках данного сообщения отправителю **недопустимо** генерировать элементы протокола или варианты синтаксиса, которые разрешено генерировать лишь участникам с другой ролью (которой у отправителя данного сообщения нет).

Соответствие HTTP включает соответствие конкретному синтаксису сообщений используемой версии протокола и соответствие семантике передаваемых элементов протокола. Например, клиент может заявить соответствие HTTP/1.1, но не распознавать функций, требуемых от получателей HTTP/1.1, что вызовет отказ при взаимодействии с сервером, который настраивает свои ответы в соответствии с заявлением клиента. Свойства, отражающие выбор пользователя, такие как согласование содержимого и выбранные пользователем расширения, могут влиять на поведение приложений вне протокольного потока и отправка элементов протокола, не отражающих в точности выбор пользователя, будет сбивать того с толку и искажать сделанный выбор.

Если реализация не соответствует семантическим требованиям, получатели сообщений от этой реализации в конце концов найдут обходные пути для соответствующей корректировки поведения. Получатель **может** использовать такие обходные пути, оставаясь соответствующим данному протоколу, если эти обходные пути применяются лишь для некорректной реализации. Например, серверы часто сканируют часть значения поля User-Agent, а пользовательские агенты - значение поля Server для корректировки своего поведения применительно к известным ошибкам или неудачно выбранным значениям по умолчанию.

2.3. Требования к размерам

Получателю протокольного элемента **следует** анализировать его защищённо, предполагая, что он соответствует грамматике ABNF и помещается в буфер разумного размера.

В HTTP нет конкретных ограничений для размера многих элементов протокола, поскольку приемлемый размер может значительно варьироваться в зависимости от контекста внедрения и назначения реализации. Поэтому взаимодействие между отправителями и получателями зависит от общих ожиданий в части разумного размера элементов протокола. Более того, общепринятое представление о разумном размере некоторых элементов изменилось за три десятилетия использования HTTP и в будущем можно ожидать продолжения таких изменений.

Получатель, по меньшей мере, **должен** быть способен проанализировать и обработать элементы протокола, размер которых не меньше размера генерируемых элементов того же типа в других сообщениях. Например, сервер-источник, публикующий очень длинные ссылки URI для своих ресурсов, должен быть способен анализировать и обрабатывать ссылки такого же размера при из получении в качестве целевого URI.

Многие принятые элементы протокола анализируются лишь в той степени, которая требуется для идентификации и пересылки элемента в нисходящем направлении. Например, посредник может найти в принятом поле его имя и компоненты значения, а затем переслать элемент без дальнейшего анализа значения поля.

2.4. Обработка ошибок

Получатель **должен** интерпретировать полученный протокольный элемент в соответствии с семантикой, заданной для него в данной спецификации, включая её расширения, если получатель не определил (на основе опыта или конфигурации), что отправитель некорректно реализует то, что подразумевается этой семантикой. Например, сервер-источник может пренебрегать содержимым принятого поля заголовка Accept-Encoding, если поле заголовка User-Agent указывает конкретную версию реализации, для которой известно об отказах при получении определённых кодировок содержимого.

Если не указано иное, получатель **может** попытаться восстановить пригодный для использования элемент протокола из недействительной конструкции. HTTP не задаёт конкретных механизмов обработки ошибок за исключением случаев непосредственного влияния на безопасность, поскольку в разных применениях протокола нужны различные стратегии обработки ошибок. Например, Web-браузер может пожелать восстанавливать отклики, в которых поле заголовка Location не разбирается в соответствии с ABNF, а системы управления сетью могут считать все формы восстановления ошибок небезопасными.

Некоторые запросы клиенты могут повторять автоматически при возникновении ошибок в базовом соединении, как описано в параграфе 9.2.2. Идемпотентные методы.

2.5. Версия протокола

Номер версии HTTP состоит из 2 десятичных цифр, разделённых точкой (.). Первая цифра (старший номер) указывает синтаксис сообщений, вторая (младший номер) - наибольшую версию в рамках этой старшей версии, которой отправитель соответствует (способен понимать в будущих взаимодействиях).

Хотя базовая семантика HTTP не меняется от версии к версии, её представление «в линии» может меняться, поэтому номер версии HTTP изменяется при внесении несовместимых изменений в формат передачи в линии. Кроме того, HTTP разрешает вносить в протокол нарастающие (incremental) изменения, совместимые с прежними версиями, без смены версии протокола за счёт использования определённых точек расширения (16. Расширения HTTP).

Версия протокола целиком показывает соответствие отправителя набору требований, указанных в соответствующих спецификациях этой версии. Например, версия HTTP/1.1 определяется этим документом, HTTP Caching [CACHING] и HTTP/1.1 [HTTP/1.1].

Старший номер версии HTTP увеличивается при внесении в синтаксис сообщений несовместимых изменений. Младший номер увеличивается при внесении в протокол изменений, расширяющих семантику сообщений или вносящих дополнительные возможности для отправителя.

Младший номер анонсирует коммуникационные возможности отправителя даже при использовании им лишь совместимого с более ранними версиями подмножества протокола, позволяя получателю знать, что в отклике (от сервера) или будущих запросах (от клиента) могут применяться более развитые возможности.

Когда старшая версия HTTP не определяет каких-либо младших версий, предполагается старшая версия 0. Это применяется при ссылках на протокол в элементах, которые требуют идентификатор младшей версии.

3. Термины и базовые концепции

Протокол HTTP был создан для архитектуры WWW и со временем был расширен для поддержки расширяемых потребностей мировых гипертекстовых систем. Большая часть этой архитектуры отражается в терминологии, применяемой для определения HTTP.

3.1. Ресурсы

Цели запросов HTTP именуются ресурсами (resource). HTTP не ограничивает природу ресурсов, а лишь задаёт интерфейс, который можно использовать для взаимодействия с ними. Большинство ресурсов идентифицируется с помощью URI, как описано в разделе 4. Идентификаторы в HTTP.

Одной из целей HTTP является разделение идентификации ресурсов и семантики запросов, возможное благодаря передачи семантики в методе запроса (9. Методы) и нескольких полях заголовка, изменяющих запросы. Например, URI ресурса может предполагать семантику, которая не является безопасной, но клиент может предполагать, что ресурс будет избегать небезопасных действий при обработке запроса безопасным методом (9.2.1. Безопасные методы).

HTTP опирается на стандарт унифицированных идентификаторов ресурсов [URI] для указания целевого ресурса (7.1. Определение целевого ресурса) и связей между ресурсами.

3.2. Представления

Представлением (representation) называется информация, предназначенная для отражения прошлого, текущего или желаемого состояния данного ресурса в формате, который может быть легко передан по протоколу. Представление состоит из набора метаданных и потенциально неограниченного потока данных представления (раздел 8).

HTTP разрешает сокрытие информации (information hiding) за своим унифицированным интерфейсом, определяя взаимодействие по отношению к переносимому представлению состояния ресурса вместо передачи самого ресурса. Это позволяет указанному URI ресурсу быть чем угодно, включая временные функции, такие как текущая погода в Laguna Beach, и в то же время потенциально предоставлять информацию, представляющую ресурс в момент генерации сообщения [REST].

Унифицированный интерфейс похож на окно, через которое можно наблюдать за чем-либо и действовать на этот объект лишь путём обмена сообщениями с независимым актором на другой стороне. Для представления (замещения) текущего или желаемого состояния этого объекта во взаимодействии нужна общая абстракция. Когда представление является гипертекстом, оно может обеспечивать представление состояния ресурса и инструкции по обработке, помогающие направить последующее взаимодействие с получателем.

Целевой ресурс может иметь или генерировать множество представлений, каждое из которых предназначено для отражения текущего состояния ресурса. Для выбора одного из таких представлений, наиболее подходящего для запроса, обычно применяется алгоритм, основанный на согласовании содержимого (12. Согласование содержимого). Выбранное представление обеспечивает данные и метаданные для оценки условных запросов (13. Условные запросы) и создания содержимого для откликов 200 (OK), 206 (Partial Content) и 304 (Not Modified) на запросы GET (9.3.1. GET).

3.3. Соединения, клиенты и серверы

HTTP - это протокол клиент-сервер, работающий через соединение (connection) с гарантией доставки на транспортном или сеансовом уровне.

Клиентами (client) HTTP являются программы, которые организуют соединение с сервером для передачи одного или множества запросов HTTP. Сервер (server) HTTP - это программа, воспринимающая соединения для обслуживания запросов HTTP путём передачи откликов HTTP.

Термины клиент и сервер относятся только к ролям программ в конкретном соединении. Одна и та же программа может быть клиентом в одних соединениях и сервером - в других.

HTTP определяется как протокол без поддержки состояний, поэтому семантика каждого сообщения может быть понята изолированно и отношения между соединениями и сообщениями в них не влияют на интерпретацию сообщений. Например, запрос CONNECT (параграф 9.3.6) или запрос с полем заголовка Upgrade (параграф 7.8) может передаваться в любой момент, а не только в первом сообщении соединения. Многие реализации зависят от отсутствия состояний в HTTP для многократного использования соединений через прокси или динамического распределения загрузки между несколькими серверами.

Поэтому серверу **недопустимо** предполагать, что два запроса в одном соединении исходят от одного пользовательского агента, пока соединение не является защищённым и привязанным к такому агенту. Известны некоторые нестандартные расширения HTTP (например, [RFC4559]), нарушающие это требование, что ведёт к проблемам безопасности и совместимости.

3.4. Сообщения

HTTP - это протокол клиент-сервер без поддержки состояний для обмена сообщениями (message) через соединение. Термины отправитель (sender) и получатель (recipient) относятся к любой реализации, которая передаёт или принимает данное сообщение, соответственно.

Клиент передаёт запросы серверу в форме запросных сообщений (request message), содержащих метод (раздел 9) и цель запроса (параграф 7.1). Запрос может также включать поля заголовка (параграф 6.3) для модификаторов запроса, сведений о клиенте, метаданных представления, и содержимого (параграф 6.4), предназначенные для обработки в соответствии с методом, а также поля трейлера (параграф 6.5) для передачи сведений, собранных при отправке содержимого.

Сервер отвечает на запрос клиента передачей одного или нескольких откликов (response message) содержащих кода статуса (раздел 15). Отклик также может включать поля заголовка для сведений о сервере, метаданных ресурса и представления для интерпретации в соответствии с кодом статуса, а также поля трейлера для передачи сведений, собранных при отправке содержимого.

3.5. Пользовательские агенты

Термин пользовательский агент (user agent) означает любую клиентскую программу, способную инициировать запрос.

Наиболее популярной формой пользовательских агентов являются Web-браузеры общего назначения, но они составляют лишь малую часть реализаций агентов. Другими распространёнными пользовательскими агентами являются спайдеры (роботы, путешествующие по web), консольные инструменты, экраны рекламных щитов, бытовые приборы, весы, лампочки, сценарии обновления микрокода (firmware), мобильные приложения и коммуникационные устройства разных форм и размеров.

Статус пользовательского агента не подразумевает, что при передаче запроса человек напрямую взаимодействует с агентом. Во многих случаях пользовательские агенты устанавливаются и настраиваются для работы в фоновом режиме и сохранения результатов для последующего просмотра (или сохранения лишь части результатов, которые могут быть интересны или сообщают об ошибках). Спайдеры, например, обычно имеют URI для начала работы и настраиваются на определённое поведение при перемещении в Web как по графу гипертекста.

Многие пользовательские агенты не могут или избегают делать интерактивные предложения для своих пользователей или выдавать адекватные предупреждения по соображениям безопасности или приватности. В некоторых случаях, когда данные спецификация требует сообщать пользователю об ошибках, допускается ограничиваться выводом таких отчётов лишь в консоль ошибок или журнальный файл системы (log). Требования подтверждения автоматизированных действий пользователем перед выполнением, тоже могут выполняться через настройки конфигурации, опции рабочего режима или простое исключение небезопасных действий. Подтверждение не предполагает какого-либо конкретного пользовательского интерфейса или прерывания обычной обработки, если пользователь уже сделал свой выбор.

3.6. Сервер-источник

Сервером-источником (origin server) называют программу, которая может создавать полномочные отклики на данный целевой запрос. Наиболее популярной формой таких серверов являются большие публичные web-сайты. Однако, как и в случае приравнивания пользовательских агентов к браузерам, ошибочно считать все серверы-источники похожими. Распространёнными серверами-источниками являются блоки домашней автоматизации, настраиваемые компоненты сетей, офисные машины, автономные роботы, новостные ленты, дорожные камеры, системы с выбором рекламы в реальном масштабе времени, платформы передачи видео по запросам.

Большинство взаимодействий HTTP состоит из запроса на получение (GET) для представления некого ресурса, указанного URI. В простейшем случае это может быть одно двухстороннее соединение (===) между пользовательским агентом (UA) и сервером-источником (O), как показано на рисунке 1.

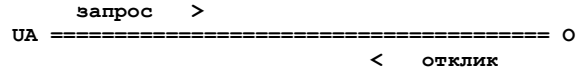

Рисунок 1.

3.7. Посредники

HTTP позволяет использовать посредников для выполнения требования через цепочку соединений. Имеется 3 основных формы посредников (intermediary) HTTP: прокси, шлюзы и туннели. В некоторых случаях один посредник может служить сервером-источником, прокси, шлюзом или туннелем, меняя поведение в зависимости от запроса.

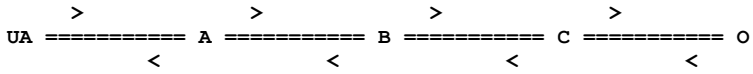

Рисунок 2.

На рисунке 2 показаны три посредника (A, B, C) между агентом пользователя и сервером-источником. Сообщение с запросом или откликом, проходящее через всю цепочку, будет встречать 4 отдельных соединения. Некоторые коммуникационные опции HTTP могут применяться только к соединению с ближайшим нетуннельным соседом, только к конечным точкам в цепочке или ко всем соединениям цепочки. Хотя на рисунке показано линейное соединение, каждый из участников цепочки может быть вовлечён одновременно во множество взаимодействий. Например, B может принимать запросы от множества клиентов, а не только от A, и/или пересылать запросы не только серверу C, в процессе обработки запроса A. Последующие запросы могут пойти по другим путям, зачастую на основе динамической конфигурации для распределения нагрузки.

Термины восходящий (upstream) и нисходящий (downstream) служат для описания направления потока сообщений - все сообщения передаются от восходящего к нисходящему. Термины входной (inbound) и исходящий (outbound) служат для описания направлений применительно к маршруту запроса - входной указывает направление к серверу-источнику, исходящий - к пользовательскому агенту.

Прокси (proxy) - это агент пересылки сообщений, выбираемый клиентом (обычно с помощью правил локальной конфигурации) для приёма запросов к некоторым типам абсолютных URI и попытки выполнить эти запросы путём трансляции через интерфейс HTTP. Некоторые трансляции являются минимальными, например, для запросов http URI, тогда как другие могут требовать трансляции в иные протоколы прикладного уровня или из таких протоколов. Прокси часто применяются для группировки HTTP-запросов организации через общего посредника в целях обеспечения безопасности, услуг аннотирования или общего кэширования. Некоторые прокси разработаны для применения преобразований к выбранным сообщениям или содержимому в процессе их пересылки, как описано в параграфе 7.7.

Шлюз (gateway, он же обратный прокси - reverse proxy) - это посредник, выступающий как сервер-источник для исходящих соединений, но транслирующий входящие запросы и пересылающий их как входящие на другой сервер или серверы. Шлюзы часть применяются для инкапсуляции унаследованных или недоверенных информационных услуг, повышения производительности серверов за счёт «ускоряющего» кэширования и обеспечения разделения или распределения нагрузки служб HTTP между множеством машин.

Все требования HTTP к серверам-источникам применимы и к исходящим коммуникациям шлюзов. Шлюз взаимодействует с серверами входящих сообщений, используя любой желаемый протокол, включая приватные расширения HTTP, выходящие за рамки этой спецификации. Однако шлюз HTTP-HTTP, желающий взаимодействовать со сторонними серверами HTTP, должен соответствовать требованиям к пользовательскому агенту на входящих соединениях шлюза.

Туннель (tunnel) действует как «слепой» ретранслятор между двумя соединениями, не изменяя сообщений. Будучи активным, туннель не считается частью коммуникаций HTTP хотя он может быть инициирован HTTP-запросом. Туннель прекращает своё существование, когда обе стороны ретранслируемого соединения закрываются. Туннели служат для расширения виртуальных соединений через посредника, например, при использовании протокола защиты транспортного уровня (Transport Layer Security или TLS, [TLS13]) для организации конфиденциального взаимодействия через общий межсетевой экран-прокси.

Выше перечислены лишь те категории посредников, которые участвуют в коммуникациях HTTP. Имеются также посредники, работающие на нижележащих уровнях сетевого стека протоколов, фильтруя или перенаправляя трафик HTTP без ведома и разрешения отправителей сообщений. Сетевые посредники неотличимы (на уровне протокола) от злоумышленников на пути и часто препятствуют безопасности и создают проблемы взаимодействия из-за ошибочной трактовки семантики HTTP. Например, прокси-перехватчик (interception proxy) [RFC3040], называемый также прозрачным прокси (transparent proxy [RFC1919]) отличается от HTTP-прокси, поскольку клиент его не выбирает. Такой прокси фильтрует или перенаправляет исходящие пакеты TCP для порта 80 (иногда и трафик других портов). Перехватывающие прокси часто встречаются в точках доступа в публичные сети в качестве средств контроля подписка до того, как будет разрешено использование нелокальных услуг Internet, а также в корпоративных межсетевых экранах для принудительного исполнения правил использования сети.

3.8. Кэширование

Кэш (cache) - это локальное хранилище предыдущих сообщений-откликов, а также подсистема управления хранилищем откликов, их извлечением и удалением. В хранилище записываются кэшированные сообщения для сокращения времени отклика и расхода пропускной способности сети при повторных запросах. Любой клиент или сервер **может** развернуть свой кэш, хотя тот не может использоваться при работе в качестве туннеля.

Влияние кэширования заключается в сокращении цепочки запрос-отклик, если у одного из элементов цепочки имеется кэшированный отклик на такой запрос. На рисунке 3 показана цепочка для случая наличия в узле B кэшированной копии предыдущего отклика от O (через C), который не был кэширован UA или A.

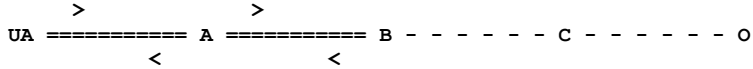

Рисунок 3.

Отклик является кэшируемым (cacheable), если кэш позволяет сохранить сообщение с откликом для использования при ответах на последующие запросы. Даже при возможности кэшировать отклик клиент или сервер-источник могут задать дополнительные ограничения на использование кэшированного отклика для конкретного запроса. Требования HTTP для поведения кэша и кэшируемых откликов заданы в [CACHING].

Имеется множество вариантов архитектуры и конфигурации систем кэширования, реализованных в Web и крупных организациях. Это включает национальные иерархии прокси-кэширования для сокращения расхода пропускной способности и снижения задержки, сети доставки содержимого, использующие кэширование на шлюзах для оптимизации регионального или глобального распределения откликов популярных сайтов, системы совместной работы, передающие кэшированные записи в групповом или широковещательном режиме для автономного (off-line) использования или в средах с высокой задержкой и т. п.

3.9. Пример обмена сообщениями

Ниже приведён пример типичного обмена сообщениями HTTP/1.1 при запросе GET (9.3.1. GET) для URI http://www.example.com/hello.txt.

Запрос клиента

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.64.1
Host: www.example.com
Accept-Language: en, mi
```

Отклик сервера

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My content includes a trailing CRLF.
```

4. Идентификаторы в HTTP

В HTTP применяются унифицированные идентификаторы ресурсов (URI) [URI] для указания ресурсов (параграф 3.1).

4.1. Ссылки URI

Ссылки URI служат для указания целей запросов и перенаправлений, а также для определения взаимосвязей.

Определения URI-reference, absolute-URI, relative-part, authority, port, host, path-abempty, segment, query заимствованы из базового синтаксиса URI. Правило absolute-path определено для элементов протокола, которые могут соедержать непустой компонент пути. Это правили слегка отличается от path-abempty из RFC 3986, разрешающего пустые пути, и path-absolute, не разрешающего пути с префиксом //. Правило partial-URI задано для элементов протокола, которые могут содержать относительный идентификатор URI, но не фрагмент.

```
URI-reference = <URI-reference, см. [URI], параграф 4.1>
absolute-URI  = <absolute-URI, см. [URI], параграф 4.3>
relative-part = <relative-part, см. [URI], параграф 4.2>
authority     = <authority, см. [URI], параграф 3.2>
uri-host      = <host, см. [URI], параграф 3.2.2>
port          = <port, см. [URI], параграф 3.2.3>
path-abempty  = <path-abempty, см. [URI], параграф 3.3>
segment       = <segment, см. [URI], параграф 3.3>
query         = <query, см. [URI], параграф 3.4>

absolute-path = 1*( "/" segment )
partial-URI   = relative-part [ "?" query ]
```

Каждый элемент протокола HTTP, разрешающий ссылки URI, указывает в своей ABNF-форме, разрешает ли он любую форму ссылок (URI-reference), только URI в абсолютной форме (absolute-URI), только путь и необязательные компоненты запроса (partial-URI) или какую-либо комбинацию предыдущих вариантов. Если не указано иное, ссылки URI анализируются относительно URI цели (параграф 7.1).

Всем отправителям и получателям **рекомендуется** поддерживать в элементах протокола URI размером по меньшей мере 8000 октетов. Отметим, что это значения учитывает обязательное увеличение некоторых структур и представления в линии в некоторых случаях (например, в строке запроса HTTP/1.1).

4.2. Связанные с HTTP схемы URI

IANA поддерживает реестр схем URI [BCP35] <https://www.iana.org/assignments/uri-schemes/>. В запросах может применяться любая схема URI, а присущие серверам HTTP схемы указаны в таблице 2.

Таблица 2.

Схема URI	*Описание*	*Параграф*
http	Hypertext Transfer Protocol	4.2.1. Схема http для URI
https	Hypertext Transfer Protocol Secure	4.2.2. Схема https для URI

Отметим, что наличие URI http или https не предполагает, что в указанном месте всегда имеется сервер HTTP, прослушивающий соединения. Кто угодно может ввести URI независимо от существования сервера или сопоставления сервером указанного идентификатора с ресурсом. Делегирование зарегистрированных имён и адресов IP создаёт объединённое пространство имён независимое от присутствия в нем серверов HTTP.

4.2.1. Схема http для URI

Этот документ определяет схему URI http для идентификаторов в иерархическом пространстве имён, управляемым потенциальным сервером-источником HTTP и прослушивающим соединения TCP ([TCP]) на данном порту.

```
http-URI = "http" "://" authority path-abempty [ "?" query ]
```

Сервер-источник для URI http указывается компонентом authority, который включает идентификатор хоста ([URI], параграф 3.2.2) и может включать номер порта ([URI], параграф 3.2.3). Если номер порта пуст или не указан, применяется порт TCP 80 (зарезервирован для служб WWW). Источник определяет, кто имеет право полномочно отвечать на запросы к указанному ресурсу, как описано в параграфе 4.3.2.

Отправителю **недопустимо** генерировать URI http с пустым идентификатором хоста. Получатель такой ссылки URI **должен** отвергать её как недействительную.

Компонент иерархического пути и необязательный компонент запроса указывают целевой ресурс в пространстве имён сервера-источника.

4.2.2. Схема https для URI

Этот документ определяет схему URI https для идентификаторов в иерархическом пространстве имён, управляемым потенциальным сервером-источником HTTP и прослушивающим соединения TCP ([TCP]) на данном порту, а также способном организовать соединение TLS ([TLS13]) для защищённых коммуникаций HTTP. В этом контексте защищенность означает, что сервер аутентифицирован, как выступающий от имени указанного полномочного органа (authority) и все коммуникации HTTP с этим сервером имеют защиту конфиденциальности и целостности, приемлемую как для клиента, так и для сервера.

```
https-URI = "https" "://" authority path-abempty [ "?" query ]
```

Сервер-источник для URI https указывается компонентом authority, который включает идентификатор хоста ([URI], параграф 3.2.2) и может включать номер порта ([URI], параграф 3.2.3). Если номер порта пуст или не указан, применяется порт TCP 443 (зарезервирован для HTTP по протоколу TLS). Источник определяет, кто имеет право полномочно отвечать на запросы к указанному ресурсу, как описано в параграфе 4.3.3.

Отправителю **недопустимо** генерировать URI https с пустым идентификатором хоста. Получатель такой ссылки URI **должен** отвергать её как недействительную.

Компонент иерархического пути и необязательный компонент запроса указывают целевой ресурс в пространстве имён сервера-источника.

Клиент **должен** убедиться, что его запросы HTTP к ресурсу https защищены, до начала взаимодействия и он воспринимает лишь защищённые отклики на свои запросы. Отметим, что выбор криптографических механизмов, приемлемых для клиента и сервера, обычно согласовывается и может меняться с течением времени.

Ресурсы, доступные по схеме https не идентичны ресурсам со схемой http. Это разные источники со своими пространствами имён. Однако расширения HTTP, определённые как применимые ко всем источникам на одном хосте, такие как протокол Cookie [COOKIE], позволяют информации, установленной одной службой, влиять на взаимодействие с другими службами в соответствующей группе доменов хоста. Такие расширения должны разрабатываться с особой тщательностью для предотвращения случайного обмена информацией из защищённого соединения в незащищённом контексте.

4.2.3. Нормализация и сравнение

URI со схемой http или https нормализуются и сравниваются в соответствии с методами, указанными в разделе 6 [URI], с использованием принятых по умолчанию значений, описанных выше для каждой схемы. HTTP не требует использовать конкретный метод определения эквивалентности. Например, ключи кэша могут сравниваться как простые строки после нормализации по синтаксису или схеме.

Нормализация на основе схемы (параграф 6.2.3 в [URI]) для URI http и https включает дополнительные правила.

- Если порт совпадает с принятым по умолчанию для схемы, с нормализованном виде компонент порта исключается.
- При использовании в качестве цели запроса OPTIONS пустой компонент пути (path) эквивалентен абсолютному пути /, поэтому в нормализованной форме указывается путь /.
- В компонентах scheme и host регистр символов не учитывается и обычно применяются строчные буквы. Остальные компоненты сравниваются с учётом регистра символов.
- Символы, не входящие в число зарезервированных, эквивалентны соответствующим октетам %-кодирования и в нормализованной форме они не кодируются (параграфы 2.1 и 2.2 в [URI]).

Например, приведённые ниже идентификаторы URI эквивалентны.

```
http://example.com:80/~smith/home.html
http://EXAMPLE.com/%7Esmith/home.html
http://EXAMPLE.com:/%7esmith/home.html
```

Два HTTP URI эквивалентных после нормализации (любым способом) могут считаться указывающими один ресурс, а нормализацию **может** выполнить любой компонент HTTP. Поэтому разные ресурсы не следует указывать HTTP URI эквивалентными после нормализации (с использованием любого метода из параграфа 6.2 в [URI]).

4.2.4. Запрет userinfo в URI

Базовый синтаксис URI для authority включает субкомпонент userinfo ([URI], параграф 3.2.1) для указания в URI сведений для проверки подлинности пользователя. В этом субкомпоненте использование формата user:password признано устаревшим. Некоторые реализации используют userinfo для внутренней настройки аутентификационных сведений, например, в опциях вызова команд, конфигурационных файлах или списках закладок, несмотря на то, что при этом может раскрываться имя пользователя и пароль.

Отправителям **недопустимо** генерировать субкомпонент userinfo (и его разделитель @) при создании в сообщении ссылки URI http или https в качестве целевого URI или значения поля.

Перед использованием ссылки URI http или https, полученной из недоверенного источника, получателю следует проверить субкомпонент userinfo и считать его наличие ошибкой. Скорее всего этот субкомпонент используется для сокрытия authority в фишинговой атаке.

4.2.5. Ссылки с идентификаторами-фрагментами

Идентификаторы-фрагменты позволяют опосредованно указать ресурс независимо от схемы URI, как указано в параграфе 3.5 [URI]. Одни элементы протокола, ссылающиеся на URI, позволяют включать фрагмент, другие - нет. Они различаются использованием правила ABNF для элементов, где фрагменты разрешены, в ином случае применяется специальное правило, исключающее фрагменты.

> Примечание. Компонент идентификатора-фрагмента для является частью определения схемы URI (см. параграф 4.3 в [URI]), поэтому он не указан в определениях ABNF для схем URI http и https, приведённых выше.

4.3. Полномочный доступ

Полномочным (authoritative) доступом называется разыменование данного идентификатора с целью доступа к указанному ресурсу таким способом, который по мнению клиента является полномочным (контролируется владельцем ресурса). Процесс определения возможности доступа устанавливается схемой URI и зачастую при работе с базовым синтаксисом использует данные из компонентов URI, таких как authority. Однако полномочный доступ не ограничивается идентифицированным механизмом.

В параграфе 4.3.1 определена концепция источника (origin) в качестве вспомогательного средства для такого использования, а в последующих параграфах объясняется, как проверить полномочия партнёра представлять этот источник.

В параграфе 17.1 рассмотрены вопросы безопасности, связанные с установлением полномочий.

4.3.1. URI Origin

Поле источника (origin) для данного URI состоит из схемы (scheme), хоста (host) и порта (port) после приведения схемы и хоста к нижнему регистру и удаления начальных нулей в номере порта. Если порт не указан в URI, используется принятый по умолчанию номер. Например URI https://Example.Com/happy.js будет иметь origin

```
{ "https", "example.com", "443" }
```

что можно представить нормализованным префиксом URI с номером порта

```
https://example.com:443
```

Каждое поле origin задаёт своё пространство имён и управляет сопоставлением идентификаторов из этого пространства с ресурсами. Отклики источника на корректные запросы с течением времени определяют семантику, связываемую пользователями с URI, и полезность этой семантики, что в конечном итоге преобразует эти механизмы в ресурс, на который пользователи могут ссылаться и обращаться к нему в будущем.

Два источника (origin) будут разными, если в них отличаются scheme, host или port. Даже если можно убедиться, что один субъект контролирует два разных источника (origin), пространства имён этих origin будут разными, пока не заданы явные псевдонимы полномочным для данного источника сервером.

Источники также используются внутри HTML и связанных с Web протоколов (не рассматриваются в этом документе), как описано в [RFC6454].

4.3.2. Источники http

Хотя HTTP не зависит от транспортного протокола, схема http (параграф 4.2.1) предназначена для связывания полномочий с тем, кто контролирует сервер-источник, прослушивающий соединения TCP на указанном порту хоста, заданного в компоненте authority. Это очень слабая привязка, поскольку она зависит от механизмов распознавания имён на стороне клиента и коммуникаций, которые могут быть не защищены от злоумышленников на пути. Тем не менее, это достаточный минимум для привязки идентификаторов http к серверу-источнику для согласованного распознавания в доверенной среде.

Если хост указан адресом IP, сервером-источником является прослушивающий (при наличии) указанный порт TCP на этом адресе IP. Если хост указан зарегистрированным именем, это имя является косвенным идентификатором для использования с системой распознавания имён, такой как DNS, для поиска адреса нужного сервера-источника.

Когда URI http используется в контексте, запрашивающем доступ к указанному ресурсу, клиент **может** пытаться получить доступ преобразуя идентификатор хоста в адрес IP, организуя соединение TCP с этим адресом на указанном порту и передавая через это соединение запросного сообщения HTTP с целью запроса, соответствующей клиентскому URI цели (параграф 7.1). Если сервер отвечает на такой запрос непромежуточным (non-interim) сообщением с откликом HTTP, как описано в параграфе 15, этот отклик считается полномочным для запроса клиента.

Отметим, что приведённый выше вариант не является единственным способом получения полномочного отклика и не подразумевает, что полномочный отклик требуется всегда (см. [CACHING]). Например, поле заголовка Alt-Svc [ALTSVC] позволяет серверу-источнику указать другие серверы, которые тоже будут полномочными для этого источника. Доступ к ресурсам, указанным http, может обеспечиваться протоколами, выходящими за рамки документа.

4.3.3. Источники https

Схема https (параграф 4.2.2) связывает полномочия на основе способности сервера использовать закрытый ключ, соответствующий сертификату, который клиент считает доверенным для указанного сервера-источника. Клиент обычно полагается на цепочку доверия, передаваемую от некой заранее согласованной или настроенной привязки доверия, чтобы считать сертификат доверенным (параграф 4.3.4).

В HTTP/1.1 и предшествующих версия клиент принимает полномочия сервера при взаимодействии по защищённому каналу конкретно с хостом-источником из URI. Организация соединения и проверка сертификата служат подтверждают полномочия.

В HTTP/2 и HTTP/3 клиент принимает полномочия сервера при организации защищённого соединения, если хост-источник в URI совпадает с любым из хостов, указанных в сертификате сервера и клиент считает, что он может открыть соединение с этим хостом для URI. На практике клиент делает запрос к DNS, чтобы проверить, что хост источника имеет тот же адрес IP, что и в организованном соединении. Это ограничение может быть снято сервером-источником передачей эквивалентного кадра ORIGIN [RFC8336].

Хост и порт цели запроса передаются в каждом запросе HTTP, указанием origin и отделением от других пространств имён, которые может контролировать тот же сервер (параграф 7.2). Источник должен убедиться, что любые услуги, предоставляемые с использованием закрытого ключа из его сертификата в равной степени ответственны за поддержку соответствующих пространство имён https или хотя бы готовы отклонять запросы, которые кажутся направленными не по назначению (параграф 7.4).

Сервер-источник может не обрабатывать запросы для некоторых целевых URI, даже имея для них полномочия. Например, при работе хоста с разными службами на разных порта (например, 443 и 8000) требуется проверка URI цели на сервере-источнике (даже через защищённое соединение), поскольку злоумышленник в сети может перебросить соединение с одного порта на другой. Отсутствие проверки целевого URI может позволить атакующему заменить отклик для одного целевого URI (скажем, https://example.com/foo) на кажущийся полномочным отклик из другого пора (например, https://example.com:8000/foo).

Отметим, что схема https не полагается на TCP и номер подключённого порта для восприятия полномочий, поскольку оба они являются внешними для защищённых коммуникаций и поэтому не могут считаться неизменными (definitive). Таким образом, взаимодействие HTTP может происходить по любому каналу, который защищён, как описано в параграфе 4.2.2, включая протоколы, не использующие TCP.

При использовании https URI в контексте, запрашивающем доступ к указанному ресурсу, клиент **может** пытаться получить доступ, преобразуя идентификатор хоста в адрес IP, организуя соединение TCP с указанным портом на этом адресе, обеспечивая сквозную защиту организацией TLS по протоколу TCP с защитой конфиденциальности и целостности и передавая через это соединение запрос HTTP с целью, соответствующей целевому URI клиента (параграф 7.1). Если сервер отвечает на такой запрос финальным (non-interim) откликом HTTP, как описано в параграфе 15, отклик считается полномочным для этого запроса клиента.

Отметим, что указанное выше не является способом получения полномочного отклика и не предполагает, что полномочный отклик нужен всегда (см. [CACHING]).

4.3.4. Проверка сертификатов https

Чтобы организовать защищённое соединение для разыменования URI, клиент **должен** убедиться, что отождествление сервера соответствует серверу-источнику из URI. Для предотвращения подмены (impersonation) сервера атакующим на пути или злоумышленником, контролирующим распознавание имён, требуется проверка сертификата. Для этого у клиента должен быть настроен набор привязок доверия.

В общем случае клиент **должен** проверить отождествление сервера, используя процесс, определённый в разделе 6 [RFC6125]. Клиент **должен** создать ссылочное определение из значения host от сервера. Если host содержит адрес IP (параграф 4.3.5), ссылочным отождествлением будет IP-ID, в ином случае host содержит имя, и отождествлением будет DNS-ID.

Клиентам **недопустимо** использовать ссылочное отождествление CN-ID. Как отмечено в параграфе 6.2.1 [RFC6125], тип CN-ID может применяться лишь старыми клиентами.

Клиент может быть специально настроен на восприятие альтернативной формы проверки отождествления сервера. Например, клиент может подключаться к серверу с динамическим адресом и именем хоста и ожидать от сервиса предоставления конкретного сертификата (или сертификата, соответствующего некому внешнему ссылочному отождествлению), а не сертификата, соответствующего источнику в целевом URI.

В особых случаях клиент может просто игнорировать отождествление сервера, но он должен понимать, что это открывает соединение для активных атак.

Если сертификат не действителен для источника целевого URI, агент пользователя **должен** получить перед обработкой подтверждение от пользователя (см. параграф 3.5) или разорвать соединение с ошибкой, указывающей непригодный сертификат. Автоматизированные клиенты **должны** записывать ошибку в подходящий журнал аудита (при наличии), а соединение **следует** разрывать (с ошибкой, указывающей непригодный сертификат). У таких клиентов **может** быть параметр конфигурации, отключающий эту проверку, но **должен** быть параметр, включающий её.

4.3.5. Ссылочное отождествление IP-ID

Сервер, идентифицируемый литералом IP-адреса в поле host внутри https URI, имеет ссылочное отождествление типа IP-ID. В ACK адресе IPv4 используется правило ABNF IPv4address, а в адресе IPv6 используется IP-literal с опцией IPv6address (см. параграф 3.2.2 в [URI]). Ссылочное отождествление IP-ID содержит декодированные байты адреса IP.

Адрес IPv4 состоит из 4 октетов, IPv6 - 16. Использование IP-ID не определено для какой-либо другой версии IP. Выбор IPAddress в расширении сертификата subjectAltName не включает явного указания версии IP, поэтому адреса различаются по размеру (см. параграф 4.2.1.6 в [RFC5280]).

Ссылочное отождествление типа IP-ID соответствует, если адрес идентичен значению iPAddress в расширении сертификата subjectAltName.

5. Поля

В HTTP используются поля (field) для предоставления данных в форме расширяемых пар «имя-значение» из зарегистрированного пространства имён ключей. Поля передаются и принимаются в разделах заголовков и трейлеров сообщений (6. Абстракция сообщения).

5.1. Имена полей

Имя поля помечает соответствующее значение поля, как имеющее семантику, заданную этим именем. Например, поле заголовка Date определено в параграфе 6.6.1 как содержащее временную метку создания сообщения, куда поле включено.

```
field-name     = token
```

Имена полей указываются без учёта регистра символов и должны быть зарегистрированы в реестре Hypertext Transfer Protocol (HTTP) Field Name Registry (см. параграф 16.3.1).

Интерпретация поля не меняется в разных младших версий одной старшей версии HTTP, хотя принятое по умолчанию поведение получателя при отсутствии поля может изменяться. Если не задано иное, поля определяются для всех версий HTTP. В частности, поля Host и Connection должны распознавать все реализации HTTP независимо от того, анонсируют ли они поддержку HTTP/1.1.

Новые поля могут вводиться без смены версии протокола, если заданная для поля семантика позволяет безопасно игнорировать поле не понимающим его получателем (см. параграф 16.3).

Прокси **должны** пересылать непонятные поля заголовка, если только это поле не указано в заголовке Connection (параграф 7.6.1) или прокси не настроен специально на блокировку или преобразование таких полей. Остальным получателям **следует** игнорировать непонятные поля заголовка и трейлера. Соблюдение этих требований позволяет расширять функциональность HTTP без добавления и удаления развёрнутых посредников.

5.2. Строки и совокупное значение поля

Разделы поля могут содержать произвольное число строк (field line), каждая со именем (field name, параграф 5.1), указывающим поле, и значением (field line value), указывающим данные для этого экземпляра поля.

Когда имя поля указано в разделе лишь однократно, совокупным значением поля (field value) будет соответствующее значение строки поля. Если имя встречается в разделе неоднократно, совокупным значением поля будет конкатенация списка значений строк поля в этом разделе (в порядке их следования) через запятую. Например, раздел

```
Example-Field: Foo, Bar
Example-Field: Baz
```

содержит две строки поля Example-Field - Foo, Bar и Baz. Совокупным значением поля будет Foo, Bar, Baz.

5.3. Порядок полей

Получатель **может** объединить несколько строк поля с одним именем в одну строку поля без изменения семантики сообщения, добавляя каждое следующее значение строки поля к прежней строке через запятую (,) и (необязательный) пробельный символ (OWS, параграф 5.6.3). Для согласованности следует использовать запятую и пробел (SP).

Порядок строк с одним именем поля важен для интерпретации значения поля и прокси **недопустимо** менять этот порядок при пересылке сообщения. Это означает, что за описанным ниже общеизвестным исключением отправителю **недопустимо** генерировать сообщения с одним именем в нескольких строках (в заголовке или трейлере) или добавлять строки с уже имеющимся в сообщении именем, если только определение этого поля не позволяет объединять несколько строк поля в список через запятые (т. е. хотя бы один из вариантов определения поля разрешает объединять строки в список через запятые, например, правила ABNF #(values) из параграфа 5.6.1).

> Примечание. На практике поле заголовка Set-Cookie [COOKIE] часто встречается в нескольких строках и не использует синтаксис списков в нарушение приведённых выше требований для строк полей с одним именем. Поскольку эти поля нельзя объединять в одно значение, получатели должны обрабатывать Set-Cookie как особый случай (см. Приложение A.2.3 к [Kri2001]).

Порядок размещения строк полей с разными именами значения не имеет. Однако хорошим тоном является передача сначала полей заголовков, содержащих дополнительные данные управления, такие как Host в запросах и Date в откликах, чтобы реализация могла как можно раньше решить вопрос о возможности обработки сообщения.

Серверу **недопустимо** применять запрос к целевому ресурсу, пока раздел заголовков запроса не принят целиком, поскольку последующие строки полей заголовка могут включать условия, свидетельства для аутентификации или намеренно ложные дубликаты полей, которые могут влиять на обработку запроса.

5.4. Ограничения для полей

HTTP не задаёт предопределённых ограничений для размера каждой строки поля, значения поля, а также размера заголовка или трейлера в целом, как описано в разделе 2. На практике встречаются специальные ограничения отдельных размеров, зачастую связанные с семантикой конкретных полей.

Сервер, получивший строку поля заголовка запроса, значение поля или набор полей, размер которых превышает желаемый для обработки, **должен** ответить подходящим кодом статуса 4xx (ошибка клиента). Игнорирование таких полей заголовка повышает уязвимость сервера к атакам с недопустимыми запросами (параграф 11.2 в [HTTP/1.1]).

Клиент **может** отбросить или отсечь полученные поля, размер которых превышает желаемый для обработки, если семантика поля позволяет без опаски игнорировать отброшенное, не меняя структуры сообщения и семантики отклика.

5.5. Значения полей

Значение поля HTTP состоит из последовательности символов, определяемых грамматикой поля, обычно определяемой с использованием ABNF ([RFC5234]).

```
field-value    = *field-content
field-content  = field-vchar
                 [ 1*( SP / HTAB / field-vchar ) field-vchar ]
field-vchar    = VCHAR / obs-text
obs-text       = %x80-FF
```

Значение поля не пробельных символов в начале и в конце. Конда конкретная версия HTTP разрешает такие символы в сообщении, реализация анализатора **должна** исключать их до оценки значения поля.

В значениях полей обычно используется лишь диапазон символов US-ASCII [USASCII]. Для полей, требующих более широкого диапазона символов можно применять кодирование, вроде того, которае определена в [RFC8187]. Исторически в HTTP допускается содержимое полей в кодировке ISO-8859-1 [ISO-8859-1] и поддержка других наборов символов обеспечивается лишь за счёт применения кодирования [RFC2047]. Спецификациям недавно определённых полей **следует** ограничивать значения печатаемыми (visible) октетами US-ASCII (VCHAR), SP и HTAB. Получателю **следует** считать иные разрешённые октеты в содержимом поля (т. е., obs-text) необрабатываемыми (opaque) данными.

Поля, с символами CR, LF, NUL, недействительны и опасны, поскольку разные реализации могут по-своему анализировать и интерпретировать эти символы. Получатель поля с такими символами **должен** отклонить сообщение или заменить каждый из них символом SP перед обработкой или пересылкой этого сообщения. Поля с другими символами управления (CTL) тоже недействительны, однако получатель **может** сохранять эти символы ради отказоустойчивости, если символы появляются в безопасном контексте (например, в специфической для приложения строке с кавычками, которая не будет обрабатываться ни одним нисходящим синтаксическим анализатором HTTP).

Поля, допускающие в качестве значения поля лишь один элемент, называют одноэлементными (singleton field). Поля, разрешающие несколько элементов в своём значении, называют списочными (list-based field). Расширение оператора списка из параграфа 5.6.1 применяется как общая нотация для определения значений полей, которые могут включать несколько элементов.

Поскольку запятые (,) служат разделителем элементов, к ним нужно относиться с осторожностью, если они разрешены внутри элемента данных. Это верно как для одноэлементных, так и для списочных полей, поскольку одноэлементное поле может быть ошибочно передано с несколькими элементами и обнаружение таких ошибок повышает функциональную совместимость. Поля, в которых ожидаются запятые внутри элемента, такие как HTTP-date и URI-reference, должны определяться с разделителями вокруг такого элемента, чтобы отличить запятую в данных от разделителя списка. Например, текстовая дата и URI (оба могут включать запятую) можно безопасно передавать в значениях списочного поля, как показано ниже.

```
Example-URIs: "http://example.com/a.html,foo",
              "http://without-a-comma.example.com/"
Example-Dates: "Sat, 04 May 1996", "Wed, 14 Sep 2005"
```

Отметим, что разделители в форме двойных кавычек почти всегда применяются при создании строк в кавычках (quoted-string, параграф 5.6.4) и применение иного синтаксиса внутри двойных кавычек приведёт, скорей всего, к ненужной путанице.

Многие поля (такие как Content-Type из параграфа 8.3) используют для параметров базовый синтаксис (параграф 5.6.6), допускающий указывать значения параметра как без кавычек (token), так и в кавычках (quoted-string). Это позволяет получателям использовать имеющиеся компоненты синтаксического анализатора. Если разрешены обе формы, смысл значения параметра должен быть одинаковым при его получении в виде маркера или в кавычках.

> Примечание. Для определения синтаксиса значения полей в этой спецификации применяется правило ABNF, называемое по имени поля, для задания разрешённой грамматики в значении поля (после того, как значение поля извлечено из базового синтаксиса сообщения и экземпляры объединены в список).

5.6. Общие правила определения значений полей

5.6.1. Списки (расширение #rule в ABNF)

Расширение #rule для правил ABNF [RFC5234] служит для улучшения читаемости в некоторых определениях значений полей на основе списков. Определена конструкция #, похожая на *, для задания списков элементов, разделённых запятыми. Полная форма <n>#<m>element указывает наличие от <n> до <m> элементов, разделённых запятой (,) и необязательными пробельными символами (OWS, параграф 5.6.3).

5.6.1.1. Требования к отправителю

В любом результате, использующем список, отправителю **недопустимо** генерировать пустые элементы списка. Иными словами, при генерации списка отправитель должен следовать синтаксису

```
1#element => element *( OWS "," OWS element )
```

и

```
#element => [ 1#element ]
```

а также для n >= 1 и m > 1 должно выполняться условие

```
<n>#<m>element => element <n-1>*<m-1>( OWS "," OWS element )
```

В приложении A дана сводка ABNF для отправителей после преобразования конструкций со списком.

5.6.1.2. Требования к получателю

Пустые элементы не учитываются при подсчёте. Получатель **должен** анализировать и игнорировать разумное число пустых элементов списка, достаточное, чтобы справиться с обычными ошибками отправителей, но не так много, чтобы это можно было использовать для Dos-атак. Иными словами, получатель **должен** воспринимать списки, удовлетворяющий приведённому ниже синтаксису.

```
#element => [ element ] *( OWS "," OWS [ element ] )
```

Отметим, что из-за возможного присутствия пустых элементов RFC 5234 ABNF не может потребовать определённое число элементов списка и поэтому все варианты отображаются, как будто это число не указано. Например,

```
example-list      = 1#example-list-elmt
example-list-elmt = token ; см. параграф 5.6.2
```

Ниже приведены действительные значения списка (без двойных кавычек, указанных лишь в качестве разделителей.

```
"foo,bar"
```

```
"foo ,bar,"
"foo , ,bar,charlie"
```

Ниже приведены значения, которые недействительны, поскольку в example-list нужен хотя бы один непустой элемент.

```
""
","
",   ,"
```

5.6.2. Маркеры

Маркер (token) - короткий текстовый идентификатор без пробелов и разграничителей.

```
token          = 1*tchar

tchar          = "!" / "#" / "$" / "%" / "&" / "'" / "*"
               / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
               / DIGIT / ALPHA
               ; any VCHAR, except delimiters
```

Многие значения полей HTTP определяются с использованием компонентов базового синтаксиса, разделённых пробельными символами или конкретными ограничителями. Ограничители выбираются из числа видимых символов US-ASCII, не разрешённых в маркерах (DQUOTE и (),/:;<=>?@[\]{}).

5.6.3. Пробельные символы

В этой спецификации применяется 3 правила для обозначения пробельных символов - OWS (необязательные), RWS (обязательные), BWS («плохие»).

Правило OWS используется в случаях, когда может присутствовать последовательность пробельных октетов. Для элементов протокола, где необязательные пробелы улучшают читаемость, отправителю **следует** генерировать необязательные пробельные символы в форме одного символа SP, в иных случаях отправителю **не следует** генерировать такие символы за исключением случаев переопределения непригодных или нежелательных элементов протокола при фильтрации сообщения на месте (in-place).

Правило RWS применяется, когда нужен хотя бы один пробельный октет для разделения маркеров поля. Отправителю **следует** генерировать RWS в форме одного символа SP.

OWS и RWS имеют такую же семантику, как один символ SP. Любое содержимое, заданное как OWS или RWS **можно** заменить одним символом SP перед интерпретацией или пересылкой сообщения в нисходящем направлении.

Правило BWS применяется там, где грамматика допускает дополнительные пробельные символы лишь по историческим причинам. Отправителю **недопустимо** генерировать BWS в сообщениях. Получатель **должен** анализировать такие пробельные символы и удалять их перед интерпретацией протокольного элемента. BWS не имеет семантики и любое содержимое, заданное как BWS **можно** удалять перед интерпретацией или пересылкой сообщения в нисходящем направлении.

```
OWS            = *( SP / HTAB )
               ; необязательные пробельные символы
RWS            = 1*( SP / HTAB )
               ; обязательные пробельные символы
BWS            = OWS
               ; «плохие» пробельные символы
```

5.6.4. Строки в кавычках

Строка текста считается одним значением, если она заключена в двойные кавычки.

```
quoted-string  = DQUOTE *( qdtext / quoted-pair ) DQUOTE
qdtext         = HTAB / SP / %x21 / %x23-5B / %x5D-7E / obs-text
```

Октет обратной дробной черты (backslash octet, \) может служить служить в качестве одноктетного символа «кавычек» внутри конструкции с кавычками и комментариев. Получатель, обрабатывающий значение строки в кавычках, **должен** обрабатывать пару кавычек, как будто она заменена октетом, следующим за символом \.

```
quoted-pair    = "\" ( HTAB / SP / VCHAR / obs-text )
```

Отправителям **не следует** генерировать пары кавычек в строке с кавычками за исключением случаев необходимости поместить в кавычки DQUOTE и октеты \, встречающиеся в строке. Отправителю **не следует** генерировать пары кавычек в комментариях за исключением случаев необходимости поместить в кавычки символы круглых скобок () и октеты \ внутри комментария.

5.6.5. Комментарии

Комментарии можно помещать в некоторые поля HTTP, заключая текст комментария в круглые скобки. Комментарии разрешены лишь в полях, включающих comment как часть определения поля.

```
comment        = "(" *( ctext / quoted-pair / comment ) ")"
ctext          = HTAB / SP / %x21-27 / %x2A-5B / %x5D-7E / obs-text
```

5.6.6. Параметры

Параметры - это экземпляры пар «имя-значение». Параметры часто используются в значениях полей как базовый синтаксис добавления вспомогательной информации к элементу. Каждый параметр обычно ограничивается непосредственно предшествующим символом точки с запятой (;).

```
parameters      = *( OWS ";" OWS [ parameter ] )
parameter       = parameter-name "=" parameter-value
parameter-name  = token
parameter-value = ( token / quoted-string )
```

Регистр символов в именах параметров не учитывается, а в значениях регистр может играть роль в зависимости от семантики имени параметра. Примеры параметров и некоторые эквивалентные формы представлены в описании

типов носителей (параграф 8.3.1) и поля заголовка Accept (параграф 12.5.1). Значение параметра, соответствующее маркеру, может быть представлено маркером или строкой в кавычках (эти представления эквивалентны).

Примечание. В параметрах не разрешены пробельные символы (даже BWS) вокруг символа равенства (=).

5.6.7. Форматы дат и времени

До 1995 г. серверы обычно использовали три разных формата для передачи меток времени. Для совместимости со старыми реализациями эти форматы определены ниже. Предпочтительным форматом является подмножество спецификации даны и времени с фиксированным размером и одним часовым поясом, используемый в сообщениях Internet (Internet Message Format) [RFC5322].

```
HTTP-date    = IMF-fixdate / obs-date
```

Пример этого формата приведён ниже.

```
Sun, 06 Nov 1994 08:49:37 GMT    ; IMF-fixdate
```

Далее указаны примеры двух устаревших форматов.

```
Sunday, 06-Nov-94 08:49:37 GMT   ; формат RFC 850
Sun Nov  6 08:49:37 1994         ; формат asctime() из ANSI C
```

Получатель, анализирующий значение метки времени в поле HTTP, **должен** воспринимать все 3 формата HTTP-date. Когда отправитель генерирует поле с одной или несколькими метками времени, определёнными как HTTP-date, он **должен** использовать формат IMF-fixdate.

Значение HTTP-date указывает время как экземпляр всемирного координированного времени (Coordinated Universal Time или UTC). В двух первых форматах UTC обозначается трехсимвольной аббревиатурой GMT (Greenwich Mean Time), которая является предшественницей UTC. Значения в формате asctime предполагают время в UTC.

Часы (clock) - это реализация, способная обеспечить разумное приближение текущего момента в UTC. Реализация часов должна использовать NTP ([RFC5905]) или похожий протокол для синхронизации с UTC.

Ниже представлен предпочтительный формат.

```
IMF-fixdate  = day-name "," SP date1 SP time-of-day SP GMT
; fixed length/zone/capitalization subset of the format
; see Section 3.3 of [RFC5322]

day-name     = %s"Mon" / %s"Tue" / %s"Wed"
             / %s"Thu" / %s"Fri" / %s"Sat" / %s"Sun"

date1        = day SP month SP year
             ; например, 02 Jun 1982

day          = 2DIGIT
month        = %s"Jan" / %s"Feb" / %s"Mar" / %s"Apr"
             / %s"May" / %s"Jun" / %s"Jul" / %s"Aug"
             / %s"Sep" / %s"Oct" / %s"Nov" / %s"Dec"
year         = 4DIGIT

GMT          = %s"GMT"

time-of-day  = hour ":" minute ":" second
             ; 00:00:00 - 23:59:60 (високосная секунда)

hour         = 2DIGIT
minute       = 2DIGIT
second       = 2DIGIT
```

Далее показаны устаревшие форматы.

```
obs-date     = rfc850-date / asctime-date

rfc850-date  = day-name-l "," SP date2 SP time-of-day SP GMT
date2        = day "-" month "-" 2DIGIT
             ; например, 02-Jun-82

day-name-l   = %s"Monday" / %s"Tuesday" / %s"Wednesday"
             / %s"Thursday" / %s"Friday" / %s"Saturday"
             / %s"Sunday"

asctime-date = day-name SP date3 SP time-of-day SP year
date3        = month SP ( 2DIGIT / ( SP 1DIGIT ))
             ; например, Jun  2
```

Регистр символов в HTTP-date имеет значение. Отметим, что в параграфе 4.2 [CACHING] это требование смягчено для получателей кэшированных данных.

Отправителю **недопустимо** включать в HTTP-date пробельные символы сверх указанных в грамматике как SP. Семантика day-name, day, month, year, time-of-day может быть такой же, как определено для конструкций Internet Message Format с соответствующими именами ([RFC5322], параграф 3.3).

Получатели меток в формате rfc850-date, где используется 2 цифры года, **должны** интерпретировать метки, указывающие более чем на 50 лет в будущее, как представляющие последний год в прошлом, имеющий те же 2 последних цифры. Получателям меток времени рекомендуется быть отказоустойчивыми при анализе меток, если иное не задано определением поля. Например, сообщения иногда передаются по HTTP от иного (не HTTP) источника, который может генерировать дату и время в соответствии с любой спецификацией, заданной Internet Message Format.

Примечание. Требования HTTP к форматам меток времени применяются лишь при их использовании в потоке протокола. Реализации не обязаны использовать их в представлении пользователю, при записи в журнал и т. п.

6. Абстракция сообщения

В каждой старшей версии HTTP задан свой формат обмена сообщениями. В этом разделе определяется абстрактный тип данных для сообщений HTTP на основе обобщения характеристик сообщений, базовой структуры и возможностей передачи семантики. Эта абстракция служит для задания требований к получателям и отправителям, не зависящие от версии HTTP, чтобы сообщение одной версии можно было передать через другие версии без искажения смысла.

Сообщение (message) включает:

- данные управления для описания и маршрутизации сообщения;
- таблицу заголовков с парами «имя-значение» для расширения данных управления и передачи дополнительных сведений об отправителе, сообщении, содержимом и контексте;
- потенциально неограниченный поток содержимого;
- таблицу трейлеров с парами «имя-значение» для передачи сведений, полученных при отправке содержимого.

Данные обрамления (framing) и управления передаются первыми, затем следует раздел заголовков с полями для таблицы заголовков. Если в сообщении имеется содержимое (content), оно передаётся после раздела заголовков, а за ним может следовать раздел трейлеров с полями для таблицы трейлеров.

Предполагается, что сообщения будут обрабатываться как поток, причём назначение этого потока и его дальнейшая обработка раскрываются в процессе чтения. Данные управления описывают, что получателю нужно узнать незамедлительно, поля заголовка, - что нужно знать до получения содержимого, а оно (при наличии) предположительно состоит из того, что получатель хочет или что нужно для выполнения семантики сообщения. Поля трейлера могут включать необязательные метаданные, которые не были известны до отправки содержимого.

Сообщения должны быть «самоописывающими» - всё, что получателю нужно знать о сообщении, можно определить просмотром самого сообщения после декодирования и восстановления частей, которые были сжаты или пропущены при передаче, без необходимости понимать текущее состояние приложения-отправителя (установленное через предшествующие сообщения). Однако клиент **должен** сохранять сведения о запросе при анализе, интерпретации или кэшировании соответствующего отклика. Например, отклики для метода HEAD выглядят как начало ответа на GET, но анализируются иначе.

Отметим, что эта абстракция сообщений является обобщением для многих версий HTTP, включая свойства, которые могут отсутствовать в некоторых версиях. Например, трейлеры были введены в версии HTTP/1.1 в форме раздела после содержимого. Эквивалент ACK присутствует в HTTP/2 и HTTP/3 в блоке заголовка, завершающего поток.

6.1. Обрамление и полнота

Обрамление показывает начало и конец сообщения, чтобы можно было отличить одно сообщение от других и шума в том же соединении. В каждой старшей версии HTTP определён свой механизм обрамления. В HTTP/0.9 и ранних внедрениях HTTP/1.0 использовался разрыв соединения в качестве завершения отклика. Для совместимости с прежними версиями такое неявное обрамление разрешено и в HTTP/1.1. Однако при неявном обрамлении может не различаться неполный отклик, если соединение разорвано рано. По этой причине в большинстве современных реализаций применяется явное обрамление в форме указания размера последовательности данных сообщения.

Сообщение считается полным (complete), когда доступны все октеты, указанные его обрамлением. Отметим, что при отсутствии явного обрамления сообщение, завершённое разрывом базового соединения считается полным, даже если его невозможно отличить от неполного отклика, пока ошибка транспортного уровня не указывает эту неполноту.

6.2. Данные управления

Сообщения начинаются с данных управления, описывающих основное назначение сообщения. Данные управления в запросе включают метод (раздел 9), и цель (параграф 7.1) запроса, версию протокола (параграф 2.5). Данные управления в отклике включают код статуса (раздел 15), необязательное описание причины и версию протокола.

В HTTP/1.1 ([HTTP/1.1]) и ранее данные управления передавались как первая строка сообщения, HTTP/2 ([HTTP/2]) и HTTP/3 ([HTTP/3]) - в полях псевдозаголовка, с зарезервированными префиксами имён (например, :authority).

В каждом сообщении HTTP указывается версия протокола. В зависимости от используемой версии она может явно указываться в сообщении или определяется из соединения, передавшего сообщение. Получатель использует сведения о версии для выяснения ограничений и возможностей для последующего взаимодействия с отправителем. Когда сообщение пересылается посредником, версия протокола обновляется в соответствии с версией, применяемой этим посредником. Для передачи сведений о протоколе восходящего направления в пересылаемом сообщении используется поле заголовка Via (параграф 7.6.3).

Клиенту **следует** передавать запрос с номером версии, совпадающим с наибольшей поддерживаемой им версией, но не выше наибольшей версии, поддерживаемой сервером, если та известна. Клиенту **недопустимо** указывать версию, которой он не соответствует. Клиент **может** указать меньшую версию, если он знает, что сервер некорректно реализует спецификацию HTTP, но только после хотя бы однократной попытки обычного запроса и определения по коду отклика и полям заголовка (например, Server), что сервер не обрабатывает корректно запросы с большей версией.

Серверу **следует** передавать отклик с наибольшей поддерживаемой им версией, где старший номер не превышает номер версии в запросе. Серверу **недопустимо** указывать версию, которой он не соответствует. Сервер может передать отклик с кодом 505 (HTTP Version Not Supported), если он хочет отказать в обслуживании указанного клиентом старшего номера версии протокола.

Получателю сообщения со старшим номером версии, который он реализует, и младшим номером выше реализуемого им **следует** обработать сообщение, как будто оно соответствует наибольшему младшему номеру версии, реализуемому клиентом для данного старшего номера. Получатель может считать, что сообщение с большим младшим номером версии, отправленное получателю, ещё не указавшему поддержку этой версии, достаточно совместимо с прежними версиями и может быть безопасно обработано любой реализацией той же старшей версии.

6.3. Поля заголовков

Поля (раздел 5), передаваемые или принимаемые перед содержимым, называют полями заголовка (header field) или просто заголовками. Раздел заголовков (header section) сообщения состоит из последовательности строк полей заголовков. Каждое поле заголовка может изменять или расширять семантику сообщения, описывать отправителя, задавать содержимое или обеспечивать дополнительный контекст.

Примечание. Именуемые поля называются здесь header field, когда из разрешено передавать только с заголовке.

6.4. Содержимое

В сообщениях HTTP часто передаётся полное или частичное представление в форме содержимого (content) - потока октетов, передаваемого после заголовка, как это обозначено обрамлением сообщения. Это абстрактное определение содержимого отражает данные после их извлечения из рамок сообщения. Например, тело сообщения HTTP/1.1 (раздел 6 в [HTTP/1.1]) может состоять из потока данных, закодированного в форме блоков (chunk) - последовательности блоков данных, одного блока нулевого размера и раздела трейлеров, тогда как содержимое того же сообщения включает только поток данных после декодирования транспортного формата и не содержит размеров блоков, синтаксиса обрамления, поля трейлера (параграф 6.5).

Примечание. Имена некоторых полей имеют префикс Content-. Это неформальное соглашение - часть таких полей относится к содержимому сообщения, определённому выше, а другие - к выбранному представлению (параграф 3.2). Неоднозначность устраняется определениями конкретных полей.

6.4.1. Семантика содержимого

Назначение содержимого запроса задаёт семантика метода (раздел 9). Например, представление в содержимом запроса PUT (параграф 9.3.4) показывает желаемое состояние целевого ресурса после успешного применения запроса, а представление в содержимом POST (параграф 9.3.3) - информацию для обработки целевым ресурсом.

Назначение содержимого отклика определяется методом запроса, кодом статуса (раздел 15) и полями отклика, описывающими это содержимое. Например, содержимое отклика 200 (OK) на запрос GET (параграф 9.3.1) представляет текущее состояние целевого ресурса на момент создания сообщения (параграф 6.6.1), а содержимое отклика на POST с тем же кодом может представлять результат или новое состояние целевого ресурса после применения обработки. Содержимым отклика 206 (Partial Content) на запрос GET будут одна часть выбранного представления или многокомпонентное сообщение, содержащее части этого представления (параграф 15.3.7). Содержимым откликов с кодом ошибки обычно является представление условий ошибки, такое как состояние ошибки и предлагаемые шаги по её устранению.

Отклики на запрос HEAD (параграф 9.3.2) не включают содержимого и поля заголовка указывают лишь значения, которые были бы при использовании для запроса метода GET (параграф 9.3.1).

Отклики 2xx (Successful) на запрос CONNECT (параграф 9.3.6) переключают соединение в туннельный режим и не включают содержимого. Отклики с кодами 1xx (Informational), 204 (No Content), 304 (Not Modified) не имеют содержимого. Все прочие отклики включают содержимое, но оно может быть пустым (размера 0).

6.4.2. Идентификация содержимого

При передаче в качестве содержимого полного или частичного представления часто бывает желательно, чтобы отправитель предоставил, а получатель определил идентификатор для ресурса, соответствующий этому представлению. Например, клиент передающий запрос GET для ресурса с текущей сводкой погоды, может захотеть получить идентификатор для возвращаемого содержимого (например, сводка погоды в Laguna Beach в момент 20210720T1711). Это может быть полезно при совместном использовании или создании закладок для ресурсов, представление которых будет меняться с течением времени.

Для сообщений с запросами применяются указанные ниже правила.

- Запрос с полем заголовка Content-Location означает утверждение отправителя, что содержимое является представлением ресурса, указанного значением поля Content-Location. Однако такому утверждению нельзя доверять, пока оно не может быть проверено иными средствами (не заданы в этой спецификации). Информация все равно может быть полезной для просмотра истории.
- В ином случае содержимое не идентифицируется HTTP, но более конкретный идентификатор может быть представлен внутри самого содержимого.

Для откликов применяются приведённые ниже правила в указанном порядке, пока не будет найдено совпадение.

1. Для запроса HEAD или отклик с кодом 204 (No Content) или 304 (Not Modified), в отклике нет содержимого.
2. Для запроса GET и отклика с 200 (OK) содержимым является представление целевого ресурса (параграф 7.1).
3. Для запроса GET и отклика с 203 (Non-Authoritative Information) содержимым является возможно изменённое или расширенное представление целевого ресурса от посредника.
4. Для запроса GET и отклика с 206 (Partial Content) содержимое включает одну или несколько частей представления целевого ресурса.
5. Если в отклике есть поле заголовка Content-Location, значением которого является та же ссылка URI, что и URI цели, содержимым является представление целевого ресурса.
6. Если в отклике есть поле заголовка Content-Location, значением которого является ссылка URI, отличающаяся от URI цели, это означает утверждение отправителя, что содержимым является представление ресурса, указанного значением поля Content-Location. Однако такому утверждению нельзя доверять, пока оно не может быть проверено иными средствами (не заданы в этой спецификации).
7. В ином случае содержимое не идентифицируется HTTP, но более конкретный идентификатор может быть представлен внутри самого содержимого.

6.5. Поля трейлера

Поля (раздел 5), размещаемые в разделе трейлера (trailer section), называют полями трейлера (trailer field) или просто трейлерами. Эти поля могут быть полезны для проверки целостности сообщения, цифровых подписей, параметров доставки и сведение о статусе пост-обработки. Поля трейлера должны обрабатываться и сохраняться отдельно от полей заголовка, чтобы не противоречить семантике сообщения, заданной на момент завершения заголовочной части. Наличие или отсутствие некоторых полей заголовка может влиять на выбор маршрутизации или обработки сообщения в целом до приёма трейлера. Этот выбор не может быть отменен последующими полями трейлера.

6.5.1. Ограничения на использование трейлеров

Раздел трейлера возможен лишь в случае поддержки применяемой версией HTTP и разрешения явным механизмом обрамления. Например блочное (chunk) транспортное кодирование в HTTP/1.1 разрешает передавать трейлерный раздел после содержимого (параграф 7.1.2 в [HTTP/1.1]).

Многие поля не могут быть обработаны за пределами заголовка, поскольку нужна их оценка до получения содержимого, например, поля, описывающие обрамление и маршрутизацию сообщения, проверку подлинности, модификаторы запроса, элементы управления откликом или форма содержимого. Отправителю **недопустимо** генерировать поля трейлера, пока он не знает, что определение соответствующего имени поля заголовка разрешает передавать поле в трейлере.

Поля трейлеров могут быть сложны для обработки посредниками при пересылке между разными версиями протокола. Если все сообщение можно буферизовать при транзите, некоторые реализации могут объединять поля трейлера с заголовком (если это приемлемо) до пересылки. Однако в большинстве случаев трейлеры просто отбрасываются. Получателю **недопустимо** объединять поля трейлера с заголовком, если он не понимает определения соответствующих полей заголовка и это определение явно не разрешает такое слияние, задавая безопасный способ.

Наличие ключевого слова trailers в поле заголовка TE (параграф 10.1.4) в запросе указывает, что клиент желает воспринимать поля трейлеров от своего имени и от имени всех нисходящих клиентов. Для запросов от посредника это означает, что все клиенты нисходящего направления желают воспринимать поля трейлеров в пересылаемых сообщениях. Отметим, что наличие trailers не означает, что клиенты будут обрабатывать какое-либо конкретное поле трейлера и говорит лишь о том, что разделы трейлеров не будут отбрасываться каким-либо из клиентов.

Поскольку поля трейлеров могут отбрасываться при транзите, серверу **не следует** генерировать поля, которые по его мнению необходимо получить пользовательскому агенту.

6.5.2. Обработка полей трейлера

Поле заголовка Trailer (параграф 6.6.2) может передаваться для указания полей, которые, вероятно, будут передаваться в разделе трейлера, что позволит получателям подготовиться к их приёму до обработки содержимого. Например, это может быть полезно, если имя поля указывает, что следует рассчитывать динамическую контрольную сумму по мере получения содержимого, а затем сразу же проверять её после получения значения в поле трейлера.

Как и поля заголовков, трейлерные поля с одинаковыми именами обрабатываются в порядке приёма. Семантика полей с одним именем в нескольких строках эквивалентна добавлению в конец множества значения в форме списка. Поля трейлера, которые могут генерироваться неоднократно в сообщении, **должны** определяться как списочные даже если значение каждого элемента обрабатывается лишь один раз на принятую строку поля.

В конце сообщения получатель **может** рассматривать набор принятых полей трейлера как структуру данных «имя-значение», аналогичную (но отдельную) полям заголовка. Дополнительные ожидания в части обработки могут задаваться в спецификации поля, предназначенного для использования в трейлерах.

6.6. Метаданные сообщения

Поля, описывающие само сообщение, например, момент и способ генерации, разрешены в запросах и откликах.

6.6.1. Date

Поле заголовка Date указывает дату и время создания сообщения, используя семантику поля Origination Date (orig-date), заданную в параграфе 3.6.1 [RFC5322]. Значение поля имеет тип HTTP-date, как указано в параграфе 5.6.7.

```
Date = HTTP-date
```

Ниже приведён пример строки поля.

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
```

Отправителю, генерирующему поле заголовка Date, **следует** создавать значение поля максимально близкое к моменту генерации сообщения. В теории это должен быть момент непосредственно перед генерированием содержимого сообщения, а на практике отправитель может генерировать значение в любой момент в процессе создания сообщения.

Сервер-источник с часами (как указано в параграфе 5.6.7) **должен** генерировать поле Date во всех откликах с кодом 2xx (Successful), 3xx (Redirection), 4xx (Client Error) и **может** генерировать его в откликах 1xx (Informational) и 5xx (Server Error). Серверу-источнику без часов **недопустимо** генерировать поле заголовка Date.

Получатель с часами, принявший отклик без поля заголовка Date, **должен** записать время получения и добавить соответствующее поле заголовка Date в конец раздела заголовков сообщения, если он кэширует или пересылает сообщения в нисходящем направлении. Получатель с часами, принявший отклик с полем заголовка Date, **может** заменить значение этого поля моментом, соответствующим приёму отклика.

Пользовательский агент **может** передавать поле заголовка Date в запросе, но обычно это не делается, если он не считает, что сведения будут полезны серверу. Например, пользовательские приложения HTTP могут передавать Date, если предполагается, что сервер будет подстраивать интерпретаций пользовательского запроса с учётом разницы часов пользовательского агента и сервера.

6.6.2. Trailer

Поле заголовка Trailer обеспечивает список имён полей, которые отправитель предполагает передать в трейлере данного сообщения. Это позволяет получателю приготовиться в приёму указанных метаданных до начала обработки содержимого.

```
Trailer = #field-name
```

Отправитель может указать, например, что подпись рассчитывается в процессе передачи потока и предоставить окончательную подпись в поле трейлера. Это позволит получателю выполнить ту же проверку «на лету» по мере приёма содержимого.

Отправителю, намеренному передать 1 или несколько полей трейлера, **следует** создать поле Trailer в разделе заголовков сообщения для указания полей, которые могут быть помещены в трейлер.

Если посредник отбрасывает раздел трейлеров при транзите, поле Trailer может служить подсказкой о том, какие метаданные потеряны, хотя нет никакой гарантии включения отправителем Trailer указанных им полей.

7. Маршрутизация сообщений HTTP

Маршрутизация запросных сообщений HTTP определяется каждым клиентом в соответствии с целевым ресурсом, конфигурацией прокси для клиента, а также организацией соединения или использованием имеющегося. Маршрутизация соответствующего отклика выполняется через ту же цепочку соединений в направлении клиента.

7.1. Определение целевого ресурса

Хотя HTTP применяется в самых разных приложениях, большинство клиентов полагается на тот же механизм и методы настройки конфигурации, что и Web-браузеры общего назначения. Даже при жёстком задании опций настройки в конфигурации клиента их можно рассматривать в совокупности как ссылку URI (параграф 4.1). Ссылка URI для получения URI цели преобразуется в абсолютную форму. В URI исключён компонент фрагмента (если он был), поскольку идентификаторы фрагменты зарезервированы для обработки на стороне клиента ([URI], параграф 3.5).

Для выполнения действий над целевым ресурсом клиент передаёт сообщение с запросом, содержащее достаточное число компонентов разобранного URI цели, чтобы получатели могли идентифицировать этот ресурс. По историческим причинам разобранные компоненты URI цели, совместно называемые целью запроса (request target), передаются в управляющих данных сообщения и поле заголовка Host (параграф 7.2).

Имеется 2 необычных случая, когда компоненты цели запроса имеют зависящую от метода форму:

- для CONNECT (параграф 9.3.6) целью запроса является имя хоста и номер порта у адресата туннеля, разделённые двоеточием;
- для OPTIONS (параграф 9.3.7) целью запроса может быть один символ звёздочки (*).

Детали приведены в описаниях указанных методов. В других методах эти формы **недопустимы**.

При получении запроса от клиента сервер восстанавливает URI цели из принятых компонентов в соответствии со своей локальной конфигурацией и контекстом входящего соединения. Эта реконструкция зависит от старшей версии протокола. Например, в параграфе 3.3 [HTTP/1.1] сказано, как сервер определяет URI цели в запросе HTTP/1.1.

> Примечание. Предыдущие спецификации определяли восстановленный URI цели как отдельное понятие эффективного URI запроса (effective request URI).

7.2. Host и :authority

Поле заголовка Host в запросе предоставляет сведения о хосте и номере порта из URI цели, позволяя серверу-источнику различать ресурсы при обслуживании запросов для разных имён хостов. В HTTP/2 [HTTP/2] и HTTP/3 [HTTP/3] поле Host в некоторых случаях подменяется полем псевдозаголовка :authority в данных управления запроса.

```
Host = uri-host [ ":" port ] ; Section 4
```

Сведения о полномочиях URI цели критически важны для обработки запроса. Агент пользователя **должен** генерировать поле Host в запросах, если он не передаёт эти сведения в поле псевдозаголовка :authority. Пользовательскому агенту, передающему Host **следует** размещать его первым в разделе заголовков запроса. Например, запрос GET к серверу-источнику <http://www.example.org/pub/WWW/> следует начинать с

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

Поскольку сведения о хосте и номере порта служат для маршрутизации на прикладном уровне, они часто являются мишенью для вредоносных программ, стремящихся отравить общий кэш или перенаправить запрос на другой сервер. Перехватывающие прокси особенно уязвимы, если они полагаются на сведения о хосте и номере порта для перенаправления запросов на внутренние серверы или использования в качестве ключа в общем кэше без предварительной проверки того, что перехваченное соединение нацелено на действительный IP-адрес этого хоста.

7.3. Маршрутизация входящих запросов

После определения URI цели и источника клиент решает, нужен ли сетевой запрос для реализации желаемой семантики и куда его следует направлять (если запрос нужен).

7.3.1. В кэш

Если у клиента имеется кэш [CACHING], способный выполнить запрос, тогда запрос обычно сначала передаётся в кэш.

7.3.2. В прокси

Если запрос не выполнен кэшем, клиент обычно определяет из своей конфигурации наличие прокси, куда можно отправить запрос. Настройка прокси зависит от реализации, но зачастую базируется на сопоставлении префиксов URI,

выбранных полномочных органов или того и другого, а сам прокси обычно указывается URI http или https. Если прокси http или https применим, клиент организует (или использует имеющееся) входящее с этим прокси и передаёт ему запросное сообщение HTTP с целью запроса, соответствующей клиентскому URI цели.

7.3.3. Источнику
Если прокси не применим, типичный клиент вызывает для доступа к указанному ресурсу процедуру-обработчик (в зависимости от схемы URI цели), заданную соответствующей спецификацией.

В параграфе 4.3.2 указано, как получить доступ к ресурсу http путём организации (или использования имеющегося) входящего соединения с указанным сервером-источником и последующей отправки ему запросного сообщения HTTP с целью запроса, соответствующей клиентскому URI цели.

В параграфе 4.3.3 указано, как получить доступ к ресурсу https путём организации (или использования имеющегося) защищённого входящего соединения с сервером-источником, полномочным для указанной цели, и последующей отправки ему запросного сообщения HTTP с целью запроса, соответствующей клиентскому URI цели.

7.4. Отклонение ошибочно направленных запросов
Когда запрос получен сервером и проанализирован достаточно для определения URI цели, сервер решает, обработать ли запрос самостоятельно, перенаправить клиента на другой ресурс, ответить кодом ошибки или сбросить соединение. На это решение могут влиять любые сведения о запросе и контексте соединения, но в первую очередь влияет настроенность сервера на обработку запросов для URI цели и пригодность контекста соединения для этого запроса. Например, запрос может быть направлен по ошибке (намеренной или случайной), такой как несоответствие сведений в полученном поле заголовка Host порту или хосту в соединении. Если соединение осуществляется с доверенного шлюза, такое несоответствие может быть ожидаемым, в ином случае оно может говорить о попытке обойти защитные фильтры, обмануть сервер для получения непубличного содержимого или отравить кэш. Вопросы безопасности, связанные с маршрутизацией сообщений, рассматриваются в разделе 17.

Когда соединение установлено не с доверенным шлюзом, сервер-источник **должен** отклонять запрос, если тот не соответствует какому-либо требованию к URI цели, определяемому схемой. В частности, запрос https **должен** отвергаться, если он не получен через соединение, защищённое с сертификатом, который действителен для источника URI цели, как указано в параграфе 4.2.2.

Код 421 (Misdirected Request) в отклике указывает, что сервер-источник отклонил запрос, по-видимому, сочтя его направленным ошибочно (15.5.20. 421 Misdirected Request).

7.5. Сопоставление запросов и откликов
Соединение может быть использовано для множества обменов «запрос-отклик». Механизм сопоставления сообщений с запросами и откликами зависит от версии HTTP и может использовать явное или неявное упорядочение.

Все отклики, независимо от кода статуса (включая промежуточные), могут передаваться в любой момент после приёма запроса, даже если тот ещё не завершён. Отклик может быть завершён до завершения соответствующего запроса (параграф 6.1) и не предполагается, что клиент ждёт отклика в течение какого-то определённого времени. Клиенты (включая посредников) могут отказаться от запроса, если отклик не получен с течение разумного интервала.

Клиенту, получившему отклик во время передачи связанного с ним запроса, **следует** продолжать передачу запроса, если он не получил явного указания на обратное (см., например, параграф 9.5 в [HTTP/1.1] и параграф 6.4 в [HTTP/2]).

7.6. Пересылка сообщений
Как описано в параграфе 3.7, посредники могут играть разные роли в обработке запросов и откликов HTTP. Некоторые посредники служат для повышения производительности и доступности, другие - для контроля доступа или фильтрации содержимого. Поскольку характеристики потока HTTP похожи на архитектуру «труба и фильтр» (pipe-and-filter), не возникает каких-либо ограничений степени улучшения (или помех) от посредников для любого направления потока. Предполагается, что посредники пересылают сообщения даже с непонятными элементами протокола (например, с новыми методами, кодами статуса или именами полей), поскольку это сохранит расширяемость для нисходящих получателей.

Посредник, не являющийся туннелем, **должен** реализовать поле заголовка Connection, как описано в параграфе 7.6.1, и исключать из пересылки поля, предназначенные лишь для входящего соединения. Посреднику **недопустимо** пересылать сообщение самому себе, если он не защищён от бесконечных циклов запросов. В общем случае посредник должен распознавать свои имена серверов, включая псевдонимы, локальные варианты и адреса IP, отвечая на такие запросы напрямую.

Сообщение HTTP можно разбирать как поток для постепенной обработки или пересылки в нисходящем направлении. Однако отправители и получатели не могут полагаться на постепенную доставку частичных сообщений, поскольку некоторые реализации будут буферизовать или задерживать пересылку сообщений для повышения эффективности сети, проверки безопасности или преобразования содержимого.

7.6.1. Connection
Поле заголовка Connection позволяет отправителю задать список желаемых опций управления для текущего соединения. Регистр символов в опциях не принимается во внимание.

```
Connection        = #connection-option
connection-option = token
```

Если отличное от Connection поле применяется для передачи управляющих сведений для текущего соединения или о нем, отправитель **должен** указать имя соответствующего поля в поле заголовка Connection. Отметим, что некоторые версии HTTP запрещают использование полей для таких сведений и, следовательно, не разрешают Connection.

Посредники **должны** анализировать поле заголовка Connection до пересылки сообщения и для каждой опции соединения (connection-option) в этом поле удалять из сообщения любые поля заголовка или трейлера с тем же

именем как у опции соединения, а затем удалять само поле заголовка Connection или заменять его своими опциями управления для пересылаемого сообщения).

Поле заголовка Connection обеспечивает декларативный способ отличать поля, предназначенные лишь для непосредственного получателя (hop-by-hop), от полей для всех получателей в цепочке (end-to-end), позволяя сообщению быть самоописывающим и обеспечивая возможность внедрения будущих расширений, связанных с соединением, без опасений их пересылки вслепую старыми посредниками. Кроме того, посредникам **следует** удалять или заменять поля, для которых это требуется перед пересылкой, независимо от их присутствия в качестве опций соединения, после применения семантики этих полей. Это включает перечисленные ниже поля (возможно, и другие):

- Proxy-Connection (Приложение C.2.2 к [HTTP/1.1]);
- Keep-Alive (параграф 19.7.1 в [RFC2068]);
- TE (параграф 10.1.4);
- Transfer-Encoding (параграф 6.1 в [HTTP/1.1]);
- Upgrade (параграф 7.8).

Отправителю **недопустимо** передавать опцию соединения, соответствующую полю, которое предназначено для всех получателей содержимого. Например, Cache-Control не подходит как опция соединения (параграф 5.2 в [CACHING]).

Опции соединения не всегда соответствуют полю, имеющемуся в сообщении, поскольку специфичное для соединения поле может не требоваться, если нет параметров, связанных с опцией соединения. Специфичное для соединения поле, полученное без соответствующей опции соединения, обычно указывает его некорректную пересылку посредником и должно игнорироваться получателем.

При определении новой опции соединения, не соответствующей полю, авторы спецификации должны зарезервировать соответствующее имя поля, чтобы впредь не возникало конфликтов. Такие имена полей регистрируются в реестре Hypertext Transfer Protocol (HTTP) Field Name Registry (16.3.1. Реестр имён полей).

7.6.2. Max-Forwards

Поле заголовка Max-Forwards позволяет ограничить число пересылок запроса между прокси с помощью методов TRACE (параграф 9.3.8) и OPTIONS (параграф 9.3.7). Это может быть полезно, когда клиент пытается отследить запрос, который, вероятно, вызвал отказ или попал в петлю пересылки.

```
Max-Forwards = 1*DIGIT
```

Значение Max-Forwards задаётся десятичным целым числом, указывающим оставшееся число возможных пересылок.

Каждый посредник, получивший запрос TRACE или OPTIONS с полем заголовка Max-Forwards, **должен** проверить и обновить значение до пересылки запроса. Если получено значение 0, посреднику **недопустимо** пересылать сообщение и он **должен** ответить как конечный получатель. Если Max-Forwards больше 0, посредник **должен** обновить поле Max-Forwards в пересылаемом сообщении значением a) меньше принятого на 1 или b) максимальным значением Max-Forwards, установленным у посредника.

Получатель **может** игнорировать полученное в заголовке поле Max-Forwards для любого другого метода запроса.

7.6.3. Via

Поле заголовка Via указывает наличие промежуточных протоколов и получателей между агентом пользователя и сервером (для запросов) или между сервером-источником и клиентом (для откликов), подобно полю Received в заголовках электронной почты (параграф 3.6.7 в [RFC5322]). Поле Via может служить для отслеживания пересылок сообщения, предотвращения петель для запросов и указания протокольных возможностей отправителей по цепочке запросов и откликов.

```
Via = #( received-protocol RWS received-by [ RWS comment ] )

received-protocol = [ protocol-name "/" ] protocol-version
                  ; см. параграф 7.8
received-by       = pseudonym [ ":" port ]
pseudonym         = token
```

Каждый элемент поля Via представляет прокси или шлюз, пересылающий сообщение. Каждый посредник добавляет в конец свои сведения о приёме сообщения и конечный результат упорядочен в соответствии с последовательностью получателей при пересылке. Прокси **должны** передавать соответствующее поле заголовка Via, как описано ниже, в каждом пересылаемом сообщении. Шлюз HTTP-HTTP **должен** передавать соответствующее поле заголовка Via в каждом входящем запросе и может передавать Via в пересылаемых откликах.

Для каждого посредника received-protocol указывает протокол и его версию, используемые восходящим отправителем сообщения. В результате значение поля Via фиксирует анонсированные возможности протокола в цепочке запросов и откликов так, что они остаются видимыми для нисходящих получателей. Это может быть полезно для обнаружения возможности безопасного использования несовместимых с прежними версиями свойств в отклике или последующем запросе, как описано в параграфе 2.5. Для краткости protocol-name не указывается, если received-protocol - это HTTP.

Элемент received-by обычно указывает хост и необязательный номер порта сервера-получателя или клиента, который переслал сообщение. Однако если сведения о реальных хостах считаются конфиденциальными, отправитель **может** заменить их псевдонимом. Если порт не указан, получатель **может** считать его принятым по умолчанию, если такой порт задан для received-protocol.

Отправитель **может** вносить комментарии для идентификации программ каждого получателя (как в полях User-Agent и Server). Однако комментарии в Via не обязательны и получатель **может** удалить их до пересылки сообщения.

Например, запрос может передать агент пользователя HTTP/1.0 внутреннему прокси с именем fred, где применяется HTTP/1.1, для пересылки публичному прокси p.example.net, который завершает запрос пересылкой серверу-источнику www.example.com. В полученном сервером запросе будет поле заголовка

```
Via: 1.0 fred, 1.1 p.example.net
```

Посреднику, служащему порталом через межсетевой экран, **не следует** пересылать имена и порты из защищённой области, если это не разрешено явно. При отсутствии такого разрешения посреднику **следует** заменять каждый в received-by хоста из защищённой области подходящим псевдонимом.

Посредник **может** объединять элементы упорядоченного списка поля заголовка Via в один элемент, если эти записи имеют идентичные значения received-protocol. Например,

```
Via: 1.0 ricky, 1.1 ethel, 1.1 fred, 1.0 lucy
```

можно сжать до

```
Via: 1.0 ricky, 1.1 mertz, 1.0 lucy
```

Отправителю **недопустимо** объединять элементы списка, если они не находятся под управлением одной организации, а хосты уже заменены псевдонимами. Отправителю **недопустимо** объединять записи с разными received-protocol.

7.7. Преобразование сообщения

У некоторых посредников имеются средства преобразования сообщений и их содержимого. Например, прокси может преобразовывать формат изображений для экономии пространства в кэше и сокращения трафика в медленных каналах. Однако применение таких преобразований к содержимому, предназначенному для критически важных приложений (например, медицинские изображения или анализ научных данных), могут вызывать проблемы при работе, особенно в случаях применения контроля целостности или цифровых подписей для обеспечения доставки содержимого без изменений.

Прокси HTTP-HTTP называется преобразующим (transforming proxy) если он предназначен и настроен на семантически значимое изменение сообщений (т. е. изменения сверх требуемых для обычной обработки HTTP, которые меняют сообщение так, что это будет значимо для исходного отправителя или может быть значимо для нисходящих получателей). Например, преобразующий прокси может выступать как общий сервер аннотирования (меняя запросы для включения ссылок на локальную базу аннотаций), фильтра вредоносных программ или сохранения приватности. Предполагается, что такие преобразования желательны для выбравшего прокси клиента или организации.

Если прокси получает URI с именем хоста, не являющимся полным доменным именем, он **может** добавить имя своего домена к имени хоста при пересылке запроса. Прокси **недопустимо** менять имя хоста, если в URI цели указано полное доменное имя.

Прокси **недопустимо** менять absolute-path и query в полученном URI цели при пересылке следующему принимающему серверу, если этого не требует протокол пересылки. Например, прокси, пересылающий запрос серверу-источнику по протоколу HTTP/1.1, будет заменять пустой путь символом / (параграф 3.2.1 в [HTTP/1.1]) или * (параграф 3.2.4 в [HTTP/1.1]) в зависимости от метода запроса.

Прокси **недопустимо** преобразовывать содержимое (параграф 6.4) отклика с директивой кэширования no-transform (параграф 5.2.2.6 в [CACHING]). Отметим, что это не относится к преобразованию сообщений, не меняющему содержимого, такому как добавление или удаление транспортного кодирования (раздел 7 в [HTTP/1.1]). Прокси **может** преобразовывать содержимое сообщения без директивы кэширования no-transform. Прокси, преобразующий содержимое отклика 200 (OK), может информировать нисходящих получателей о применении преобразования заменой пода отклика на 203 (Non-Authoritative Information) (параграф 15.3.4).

Прокси **не следует** изменять поля заголовка со сведениями о конечных точках коммуникационной цепочки, статусе ресурса или выбранном представлении (отличном от содержимого), если определение поля специально не разрешает такие изменения или изменения не представляются обязательными для обеспечения приватности или безопасности.

7.8. Upgrade

Поле заголовка Upgrade предназначено для обеспечения простого механизма перехода от HTTP/1.1 к другому протоколу в том же соединении. Клиент **может** передать список имён протоколов в поле заголовка Upgrade в запросе, чтобы предложить серверу переключиться на один или несколько из указанных протоколов до отправки окончательного отклика. Сервер **может** игнорировать полученное поле заголовка Upgrade, если он хочет продолжить использование в соединении текущего протокола. Поле Upgrade не может применяться для настаивания на смене протокола.

```
Upgrade          = #protocol

protocol         = protocol-name ["/" protocol-version]
protocol-name    = token
protocol-version = token
```

Хотя имена протоколов регистрируются в предпочтительном регистре, получателям **следует** использовать сравнение без учёта регистра при сопоставлении protocol-name с поддерживаемыми протоколами.

Сервер, передающий отклик 101 (Switching Protocols), **должен** включить поле заголовка Upgrade для указания новых протоколов, на которые переключено соединение. Если переключаются протоколы нескольких уровней, отправитель **должен** указать список протоколов в порядке роста уровней. Серверу **недопустимо** переключаться на протокол, не указанный клиентом в поле заголовка Upgrade соответствующего запроса. Сервер **может** игнорировать указанный клиентом порядок предпочтения и выбирать новые протоколы на основе других факторов, например, характера запроса или текущей загрузки сервера. Сервер, передающий отклик 426 (Upgrade Required), **должен** включить в него поле заголовка Upgrade для указания приемлемых протоколов в порядке снижения предпочтений.

Сервер **может** передавать поле заголовка Upgrade в любом другом отклике для анонсирования поддержки перехода на указанные в списке протоколы (в порядке снижения предпочтений) в будущих запросах. Ниже приведён пример гипотетического запроса от клиента.

```
GET /hello HTTP/1.1
Host: www.example.com
Connection: upgrade
Upgrade: websocket, IRC/6.9, RTA/x11
```

Возможности и характер взаимодействия на прикладном уровне после смены протокола полностью зависят от выбранных протоколов. Однако предполагается, что сразу после передачи отклика 101 (Switching Protocols) сервер продолжит отвечать на исходный запрос, как будто принят его аналог по новому протоколу (после смены протокола сервер все равно должен выполнить запрос и предполагается, что запрос не потребуется повторять). Например, при получении поля Upgrade в запросе GET сервер, решивший поменять протокол, сначала передаёт сообщение 101 (Switching Protocols) в HTTP/1.1, а сразу после этого отвечает по новым протоколам на запрос GET для целевого ресурса. Это позволяет перевести соединение на протоколы с идентичной HTTP семантикой без издержек в виде дополнительного кругового обхода. Серверу **недопустимо** переключать протоколы, если семантика полученного сообщения не реализуется с новыми протоколами. Запрос OPTIONS может быть выполнен любым протоколом.

Ниже представлен пример отклика, на приведённый выше запрос.

```
HTTP/1.1 101 Switching Protocols
Connection: upgrade
Upgrade: websocket

[...поток данных переключён на websocket с соответствующим откликом
 (определяется новым протоколом) на запрос GET /hello...]
```

Отправитель Upgrade **должен** также передать опцию соединения Upgrade в поле заголовка Connection (параграф 7.6.1) для информирования посредников о том, что это поле не следует пересылать. Сервер, получивший поле заголовка Upgrade в запросе HTTP/1.0, **должен** игнорировать это поле.

Клиент не может начать использование в соединении обновлённого протокола, пока он не передаст полностью запросное сообщение (т. е. клиент не может поменять протокол, если передача сообщения ещё не завершена). Если сервер получает сразу поля заголовка Upgrade и Expect с ожиданием 100-continue (параграф 10.1.1), он **должен** передать отклик 100 (Continue) до отправки отклика 101 (Switching Protocols).

Поле заголовка Upgrade применяется только для переключения протоколов, работающий через имеющееся соединение и не позволяет переключить базовый (транспортный) протокол или поменять имеющееся соединение на другое. Для этих целей можно использовать отклики 3xx (Redirection) (параграф 15.4).

Данная спецификация задаёт лишь имя протокола HTTP только для семейства протоколов передачи гипертекста (Hypertext Transfer Protocols), как задано правилами для версии HTTP в параграфе 2.5 и будущими обновлениями этой спецификации. Другие имена протоколов должны регистрироваться по процедуре, заданной с параграфе 16.7.

8. Представление данных и метаданных

8.1. Представление данных

Данные представления, связанного с сообщением HTTP, обеспечиваются содержимым сообщения или ссылкой семантики сообщения и URI цели. Формат и кодировку данных представления задают поля метаданных в заголовке. Тип данных представления определяется в полях заголовка Content-Type и Content-Encoding. Это задаёт двухуровневую упорядоченную модель кодирования.

```
representation-data := Content-Encoding( Content-Type( data ) )
```

8.2. Представление метаданных

Метаданные представления содержатся в полях заголовка этого представления. Если в сообщении имеется содержимое, поля заголовка представления описывают способы интерпретации данных. В отклике на запрос HEAD поля заголовка представления описывают данные представления, которые мыли бы вложен в отклик на такой же запрос GET.

8.3. Content-Type

Поле заголовка Content-Type указывает тип носителя в связанном представлении - это представление, вложенное в содержимое сообщения или выбранное представление, заданное семантикой сообщения. Указанный тип носителя определяет формат данных и способ их обработки получателем в рамках семантики принятого сообщения после декодирования содержимого в соответствии с Content-Encoding.

```
Content-Type = media-type
```

Типы носителей определены в параграфе 8.3.1. Пример поля приведён ниже.

```
Content-Type: text/html; charset=ISO-8859-4
```

Отправителю, генерирующему сообщение с содержимым, **следует** включать в сообщение поле заголовка Content-Type, если отправитель не знает тип содержимого для вложенного представления. Если поле Content-Type не включено, получатель **может** предположить тип application/octet-stream ([RFC2046], Section 4.5.1) или проверить данные для определения типа.

На практике владельцы ресурсов не всегда настраивают свой сервер-источник на указание корректного Content-Type для данного представления. Некоторые пользовательские агенты проверяют содержимое и, в некоторых случаях, переопределяют полученный тип (см., например, [Sniffing]). В результате такого «вынюхивания» (MIME sniffing) могут быть сделаны ложные выводы о данных, которые могут вызывать для пользователя дополнительный риск в части защиты (например, повышение привилегий). Кроме того, в различных типах носителей часто применяется один формат данных и типы различаются лишь способом обработки данных, которые не определить простым изучением данных. При реализации «обнюхивания» разработчикам рекомендуется давать пользователю возможность отключить его.

Хотя поле Content-Type определено как одноэлементное, иногда его некорректно указывают несколько раз, что ведёт к возникновению списка значений. Получатели часто пытаются обработать такую ошибку используя из списка последний синтаксически верный элемент, что может вести к проблемам совместимости и рискам безопасности, если разные реализации по-своему обрабатывают ошибку.

8.3.1. Тип носителя

HTTP использует тип носителя [RFC2046] в полях заголовка Content-Type (параграф 8.3) и Accept (параграф 12.5.1) для обеспечения открытой, расширяемой типизации данных и согласования типов. Тип носителя задаёт формат данных и способ их обработки в соответствии с контекстом сообщения.

```
media-type = type "/" subtype parameters
type       = token
subtype    = token
```

Регистр символов в маркерах типов и субтипов не принимается во внимание.

За типом (субтипом) **могут** через точку с запятой следовать параметры (параграф 5.6.6) в виде пар «имя-значение». Наличие или отсутствие параметра может быть значимым для обработки типа носителя в зависимости от его определения в реестре типов. Регистр символов в параметрах может иметь значение в зависимости от семантики имени параметра. Ниже в качестве примера приведены эквивалентные записи, описывающие текстовые данные HTML в кодировке UTF-8, среди которых первая строка является предпочтительной с плане согласованности (значение параметра charset определено как независимое от регистра символов в параграфе 4.1.2 [RFC2046]).

```
text/html;charset=utf-8
Text/HTML;Charset="utf-8"
text/html; charset="utf-8"
text/html;charset=UTF-8
```

Типы носителей должны регистрироваться в реестре IANA по процедурам, заданным в [BCP13].

8.3.2. Charset

В HTTP применяются имена charset для указания кодировки символов ([RFC6365], раздел 2) в текстовом представлении. В полях, определяемых этим документом, имена charset указываются в параметрах (Content-Type) или (для Accept-Charset[1]) в форме обычного маркера (token). В обоих случаях имена сопоставляются без учёта регистра.

Имена charset должны регистрироваться в реестре IANA Character Sets (https://www.iana.org/assignments/character-sets) по процедурам, заданным в разделе 2 [RFC2978].

> Примечание. В теории имена charset определяет правило ABNF mime-charset, определённое в параграфе 2.3 [RFC2978] (с учётом [Err1912]). Это правило допускает использование двух символов, не входящих в token ({ и }), однако ни в одном из зарегистрированных на момент написания документа имён этих символов нет (см. [Err5433]).

8.3.3. Составные типы

MIME применяется для большого числа многокомпонентных (multipart) типов, где инкапсулируется одно или несколько представлений в теле одного сообщения. Для всех составных типов применяется общий синтаксис, заданный в параграфе 5.1.1 [RFC2046], и параметр границы как часть значения media-type. Само тело сообщения является элементом протокола. Отправитель **должен** лишь генерировать символы CRLF для разделения частей тела.

В обрамлении сообщения HTTP не применяются границы между частями для указания размера тела сообщения, хотя их могут использовать реализации, генерирующие и обрабатывающие содержимое. Например, тип multipart/form-data часто применяется для переноса данных форм в запросе, как задано в [RFC7578], а тип multipart/byteranges задан этой спецификацией (параграф 15.3.7) для применения в некоторых откликах 206 (Partial Content).

8.4. Content-Encoding

Поле Content-Encoding указывает, какие кодировки содержимого применялись к представлению, помимо присущих типу носителя, м, соответственно, какое нужно использовать декодирование для получения данных с типом носителя, указанным полем Content-Type. Поле Content-Encoding применяется в основном для обеспечения возможности сжатия данных представления без потери идентичности базового типа носителя.

```
Content-Encoding = #content-coding
```

Ниже приведён пример использования поля.

```
Content-Encoding: gzip
```

Если для представления использовано одно или несколько кодирований, применивший их отправитель **должен** создать поле заголовка Content-Encoding, в котором указаны все использованные варианты кодирования в порядке их применения. Отметим, что имя кодирования identity зарезервировано для особой роли в Accept-Encoding и включать его **не следует**. Дополнительные сведения о кодировании могут указываться в других полях заголовка, не заданных этой спецификацией.

В отличие от Transfer-Encoding (параграф 6.1 в [HTTP/1.1]), кодирование в Content-Encoding является характеристикой представления, которое задаётся в терминах кодированной формы и все прочие метаданные о представлении относятся к кодированной форме, если иное не указано в определении метаданных. Обычно представление декодируется только перед визуализацией (rendering) или аналогичным применением.

Если тип носителя имеет встроенное кодирование, такое как сжатие, это кодирование не указывается в Content-Encoding, даже если в нем применяется тот же алгоритм, что и в одном из кодирований содержимого. Такое кодирование указывается лишь в странных и редких случаях повторного применения для формирования представления. Сервер-источник может публиковать одни и те же данные в нескольких представлениях, отличающихся лишь кодированием, указанным в Content-Type или Content-Encoding, поскольку некоторые пользовательские агенты могут по-разному вести себя при обработке каждого отклика (например, открывать диалог Save as … вместо автоматической декомпрессии и отображения содержимого). Сервер-источник **может** отвечать кодом (Unsupported Media Type), если представление в содержимом запроса имеет неприемлемое кодирование.

[1] В оригинале ошибочно сказано Content-Encoding, см. https://www.rfc-editor.org/errata/eid7419. *Прим. перев.*

8.4.1. Кодирование содержимого

Значения кодирования содержимого указывают кодирующие преобразования, которые могут применяться к представлению. Кодирование содержимого применяется в основном для обеспечения возможности сжать представление или выполнить иное полезное преобразование без потери идентичности базового типа носителя и данных. Представление зачастую сохраняется в кодированной форме, передаётся в ней и декодируется только конечным получателем.

```
content-coding   = token
```

Регистр символов в content-coding не принимается во внимание, а имена должны регистрироваться в реестре HTTP Content Coding Registry, как описано в параграфе 16.6. Значения content-coding применяются в полях заголовка Accept-Encoding (параграф 12.5.3) и Content-Encoding (параграф 8.4).

8.4.1.1. Кодирование Compress

Кодирование compress - это адаптивное кодирование Lempel-Ziv-Welch (LZW) [Welch], которое обычно создаёт UNIX-программа compress для сжатия файлов. Получателю **следует** считать x-compress эквивалентом compress.

8.4.1.2. Кодирование Deflate

Кодирование deflate - это формат данных zlib [RFC1950], содержащий сжатый с помощью deflate поток данных [RFC1951], где применяется комбинация алгоритма сжатия Lempel-Ziv (LZ77) и кодирования Huffman.

> Примечание. Некоторые реализации, не соответствующие спецификации, передают сжатые данные deflate без оборачивания в формат zlib.

8.4.1.3. Кодирование gzip

Кодирование gzip - это LZ77 с 32-битовой контрольной суммой (Cyclic Redundancy Check или CRC), которое обычно обеспечивается программой сжатия [RFC1952]. Получателю **следует** считать x-gzip эквивалентом gzip.

8.5. Content-Language

Поле заголовка Content-Language описывает естественные языки предполагаемой аудитории представления. Отметим, что поле может не указывать всех языков, используемых в представлении.

```
Content-Language = #language-tag
```

Теги языков определены в параграфе 8.5.1. Основное назначение Content-Language состоит в том, чтобы позволить пользователю указывать и различать представления в соответствии с предпочитаемым им языком. Например, содержимое, предназначенное для владеющих датским языком может быть указано как

```
Content-Language: da
```

При отсутствии Content-Language предполагается, что содержимое предназначено для аудитории с любым языком. Это может означать, что отправитель не считает представление специфичным для какого-либо языка или не знает, для какого языка оно предназначено.

Для содержимого **можно** указать несколько языков, если оно предназначено для разноязычных аудиторий. Например, текст Договора Вайтанги (Treaty of Waitangi), представленный одновременно на английском и языке Maori, может включать

```
Content-Language: mi, en
```

Однако наличие в представлении нескольких языков не означает, что оно предназначено для разноязычной аудитории. Примером может служить учебник языка для начинающих, такой как A First Lesson in Latin, который явно предназначен для англоязычной аудитории, и в этом случае разумно указывать для Content-Language только значение en.

Поле Content-Language **может** применяться для носителей любого типа, а не только для текста.

8.5.1. Теги языков

Тег языка, как определено в [RFC5646], указывает естественный язык, на котором говорят, пишут или иным способом люди передают информацию между собой. Компьютерные языки здесь явно исключены.

HTTP использует теги языка в полях заголовка Accept-Language и Content-Language. В Accept-Language применяется более широкий диапазон языков, как указано в параграфе 12.5.4, а в Content-Language language-tag имеет форму

```
language-tag = <Language-Tag, see [RFC5646], Section 2.1>
```

Тег языка включает один или несколько субтегов (без учёта регистра символов), разделённых символом дефиса (-, %x2D). В большинстве случаев тег состоит из основного субтега языка, указывающего семью языков (например, en = English), за которым могут следовать субтеги, уточняющие или сужающие языковой диапазон (например, en-CA для канадской разновидности английского языка). Пробелы в тегах языка не разрешаются. Примеры тегов приведены ниже.

```
fr, en-US, es-419, az-Arab, x-pig-latin, man-Nkoo-GN
```

Дополнительные сведения можно найти в [RFC5646].

8.6. Content-Length

Поле заголовка Content-Length указывает размер данных соответствующего представления десятичным неотрицательным целым числом октетов. При передаче представления как содержимого Content-Length относится именно к объёму данных, заключённых в нем, поэтому может служить для указания границы обрамления (см. например, параграф 6.2 в [HTTP/1.1]). В иных случаях Content-Length указывает текущий размер выбранного представления, который получатель могут использовать для оценки времени передачи или сравнения с сохраненным ранее представлением.

```
Content-Length = 1*DIGIT
```

Пример приведён ниже.

```
Content-Length: 3495
```

Агенту пользователя следует передавать Content-Length в запросе, если метод задаёт значение для вложенного содержимого и не передаётся Transfer-Encoding. Например, агент пользователя обычно передаёт Content-Length в запросе POST даже при значении 0 (пустое содержимое). Агенту пользователя **не следует** передавать поле Content-Length, если в запросе нет содержимого и семантика метода не предполагает таких данных.

Сервер **может** передавать поле Content-Length в отклике на запрос HEAD (параграф 9.3.2), но это **недопустимо**, если значение поля не указывает десятичное значение числа октетов, которые будут переданы в содержимом отклика на такой же запрос с методом GET.

Сервер **может** передавать поле Content-Length в отклике 304 (Not Modified) на условный запрос GET (параграф 15.4.5), но это **недопустимо**, если значение поля не указывает десятичное значение числа октетов, которые будут переданы в содержимом отклика 200 (OK) на тот же запрос.

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

Помимо указанных выше случаев, серверу-источнику **следует** передавать поле Content-Length при отсутствии Transfer-Encoding, когда размер содержимого известен до завершения передачи раздела заголовков. Это позволит нисходящим получателям оценивать ход передачи, знать о полноте принятого сообщения и, возможно, использовать то же соединение для дополнительных запросов.

В поле Content-Length действительны любые положительные значения. Поскольку нет предопределённого ограничения на размер содержимого, получатель должен предвидеть возможность больших десятичных чисел и предотвращать ошибки разбора в результате переполнения или потери точности при преобразовании в целое число (параграф 17.5).

Поскольку Content-Length служит для определения границ сообщений в HTTP/1.1, значение поля может влиять на разбор сообщения нисходящими получателями даже при отсутствии на промежуточных соединениях HTTP/1.1. Если сообщение пересылается посредником нисходящего направления, значение поля Content-Length, не соответствующее обрамлению полученного им сообщения, может вызывать проблемы безопасности из-за нелегитимных запросов или разделения откликов. Поэтому отправителю **недопустимо** передавать сообщения с заведомо некорректным значением поля Content-Length. Точно так же отправителю **недопустимо** пересылать сообщения с полем Content-Length, не соответствующим ABNF (см. выше) с единственным исключением - получатель поля Content-Length с одним и тем же десятичным значением, повторяющимся в списке через запятую (например, Content-Length: 42, 42), **может** отвергнуть сообщение как непригодное или заменить недействительное значение поля одним десятичным значением, поскольку дубликат, вероятно, был создан при обработке восходящим процессором сообщений.

8.7. Content-Location

Поле заголовка Content-Location указывает URI для возможного применения в качестве идентификатора конкретного ресурса, соответствующего представлению в содержимом этого сообщения. Иными словами, если выполнить запрос GET с этим URI в момент генерации данного сообщения, отклик 200 (OK) будет содержать то же представление, которое включено в содержимое этого сообщения.

```
Content-Location = absolute-URI / partial-URI
```

Значением поля является absolute-URI или partial-URI. В последнем случае (раздел 4) URI указывается относительно URI цели ([URI], раздел 5).

Значение Content-Location не является заменой URI цели (параграф 7.1) и служит представлением метаданных. В нем применяется тот же синтаксис и семантика, что и в одноимённом поле заголовка, определённом для частей тела MIME в разделе 4 [RFC2557]. Однако наличие этого поля в сообщении HTTP оказывает некоторое особое влияние на получателей HTTP.

Если поле Content-Location включено в сообщение с откликом 2xx (Successful) и значение поля указывает (после преобразования в абсолютную форму) URI, соответствующий URI цели, получатель **может** считать содержимое текущим представлением этого ресурса в момент, указанный датой создания сообщения. Для запросов GET (параграф 9.3.1) и HEAD (параграф 9.3.2) это соответствует принятой по умолчанию семантике, когда сервер не предоставляет Content-Location. Для запросов со сменой состояния, таких как PUT (параграф 9.3.4) или POST (параграф 9.3.3) это означает, что отклик сервера содержит новое представление данного ресурса, что отличает его от представлений, которые могут лишь сообщать о действии (например, It worked!). Это позволяет разработчикам приложений обновлять локальные копии без необходимости дополнительного запроса GET.

Поле Content-Location в сообщении с откликом 2xx (Successful) и значением поля, указывающим URI, отличающийся от URI цели, указывает заявление сервера-источника о том, что этот URI является идентификатором другого ресурса, соответствующего вложенному представлению. Такому утверждению можно доверять лишь в том случае, когда у обоих идентификаторов один владелец, что невозможно проверить программным путем через HTTP.

- Для откликов на запрос GET или HEAD это говорит, что URI цели указывает ресурс, для которого требуется согласование содержимого, а значение поля Content-Location является более конкретным идентификатором для выбранного представления.
- Для отклика 201 (Created) на метод, меняющий состояние, значение поля Content-Location, совпадающее со значением Location, указывает, что содержимое является текущим представлением вновь созданного ресурса.
- В иных случаях Content-Location указывает, что содержимое является представлением, указывающим статус запрошенного действия и доступность (для будущего запроса GET) отчёта о статусе по данному URI. Например, операция покупки через запрос POST может включать чек в форме содержимого отклика 200 (OK) и поле Content-Location будет указывать идентификатор для извлечения копии этого чека в будущем.

Передавая Content-Location в запросном сообщении, агент пользователя указывает, что значение поля соответствует местоположению, откуда этот агент изначально получил содержимое (до того, как он внёс какие-либо изменения). Иными словами, агент пользователя указывает обратную ссылку на источник исходного представления.

Сервер-источник, передающий поле Content-Location в отклике, **должен** считать эти сведения временным контекстом запроса, а не метаданными, сохраняемыми без изменений как часть представления. Сервер-источник **может**

использовать этот контекст как руководство по обработке запроса или сохранить его для иных целей, например, в ссылках на источник или метаданных о версии. Однако ему **недопустимо** использовать данные контекста для смены семантики запроса. Например, если клиент подаёт запрос PUT для согласованного ресурса и сервер-источник воспринимает PUT (без перенаправления), предполагается, что новое состояние этого ресурса будет соответствовать одному из представлений в этом PUT. Поле Content-Location нельзя применять как форму обратного идентификатора выбора содержимого для обновления лишь одного из согласованных представлений. Если пользовательскому агенту нужна такая семантика, он будет применять PUT непосредственно для Content-Location URI.

8.8. Поля валидатора

Метаданные ресурса называют валидатором, если их можно использовать в предварительных условиях (параграф 13.1) для запроса (раздел 13). Поля передают текущий валидатор для выбранного представления (параграф 3.2).

В откликах на безопасные запросы поля валидатора описывают представление, выбранное сервером-источником при обработке отклика. Отметим, что в зависимости от метода и семантики кода статуса, выбранное для данного отклика представление не обязательно совпадает с представлением в содержимом отклика.

В отклике об успехе меняющего состояние запроса поля валидатора описывают новое представление, которое в результате обработки заменило выбранное ранее представление. Например, поле ETag в отклике 201 (Created) передаёт тег сущности недавно созданного представления ресурса так, что этот тег можно использовать в качестве валидатора в последующих условных запросах для предотвращения проблемы потери обновления (lost update).

Эта спецификация задаёт две формы метаданных, которые обычно применяются для наблюдения за состоянием ресурса и проверки предварительных условий - даты изменений (параграф 8.8.2) и необрабатываемые (opaque) теги сущностей (параграф 8.8.3). Дополнительные метаданные, определяющие состояние ресурса, могут определяться расширениями HTTP, такими как Web Distributed Authoring and Versioning [WEBDAV], выходящими за рамки документа.

8.8.1. Строгие и мягкие валидаторы

Валидаторы могут быть строгими (strong) или мягкими (weak). Мягкие валидаторы проще создавать, но они менее полезны при сравнении. Строгие валидаторы идеальны для сравнения, но их генерация может быть очень сложной (иногда невозможно). Вместо навязывания одной формы валидаторов для всех ресурсов HTTP раскрывает тип применяемых валидаторов и вносит ограничения на использование мягких валидаторов при сравнении.

Строгий валидатор - это метаданные представления, меняющие значение при любом изменении данных представления, которое можно видеть в содержимом отклика 200 (OK) на запрос GET. Строгий валидатор может менять значение не только в результате изменения данных представления, такого как изменение семантически значимой части метаданных (например, Content-Type), но сервер-источник заинтересован в смене значения лишь при необходимости аннулировать сохранённые отклики, размещённые в удалённых кэшах и средствах разработки.

Записи в кэше могут сохраняться сколь угодно долго, независимо от завершения срока действия, и кэш может пытаться подтвердить запись с помощью валидатора, полученного в далёком прошлом. Строгий валидатор уникален для всех версий всех представлений конкретного ресурса в течение времени. Однако не предполагается уникальность для представлений разных ресурсов (т. е. один строгий валидатор может служить для представления нескольких ресурсов одновременно, не указывая эквивалентность этих представлений).

На практике применяются различные строгие валидаторы. Лучшие из них основаны на строгом контроле выпусков (ревизий), когда при каждой смене представления ему назначается уникальное имя узла и идентификатор выпуска до того, как представление станет доступным для GET. Устойчивая к конфликтам хэш-функция, применяемая к представлению данных, подходит и для случаев, когда данные доступны до отправки полей заголовка и дайджест не нужно обновлять при каждом получении запроса на проверку. Однако при наличии у ресурса представлений, отличающихся лишь метаданными, например, как при согласовании содержимого по типам носителей с одинаковым форматом данных, серверу-источнику нужно включать в валидатор дополнительные сведения, различающие эти представления.

Мягкий валидатор - это метаданные представления, которые могут не меняться при каждом изменении данных представления. Такая мягкость может быть следствием ограничений (например, точности часов), невозможности обеспечить уникальность всех возможных представлений ресурса или желанием владельца сгруппировать представления по некоторому самоопределяемому набору значений эквивалентности, а не по уникальной последовательности данных.

Серверу-источнику **следует** менять мягкий тег сущности всякий раз, когда он считает прежние представления неприемлемыми в качестве замены текущего представления. Иными словами, мягкий тег сущности должен меняться всякий раз, когда сервер-источник хочет аннулировать старые отклики в кэшах. Например, представления прогноза погоды, содержимое которого меняется каждую секунду на основе динамических измерений, можно сгруппировать в наборы эквивалентных (с точки зрения сервера-источника) представлений с одним мягким валидатором, чтобы кэшированные представления были действительны с течение разумного интервала времени (возможно, динамически корректируемого в зависимости от нагрузки на сервер или качества измерений). Аналогично, время смены представления, указанное с разрешением в 1 секунду, может быть мягким валидатором, если существует возможность неоднократной смены представления в течение 1 секунды и извлечения между такими сменами представлений.

Валидатор будет мягким, если он применяется одновременно для нескольких представлений данного ресурса, если у них не совпадают данные представления. Например, если сервер-источник передаёт один валидатор для представления с кодированием содержимого gzip и представления без кодирования, этот валидатор будет мягким. Однако два одновременных представления могут иметь один строгий валидатор, если они отличаются лишь метаданными, например, при использовании для одних данных представления разных типов носителя.

Строгие валидаторы подходят для всех условных запросов, включая проверку кэша, частичные диапазоны содержимого и предотвращение потери обновлений. Мягкие валидаторы пригодны лишь в случаях, когда клиент не требует точного совпадения с полученными ранее данными представления, например, при проверке записи в кэше или выборе лишь последних изменений при просмотре web.

8.8.2. Last-Modified

Поле Last-Modified в отклике содержит временную метку, указывающую дату и время последнего изменения выбранного представления с точки зрения сервера-источника, которые были определены при завершении обработки запроса.

```
Last-Modified = HTTP-date
```

Пример поля представлен ниже.

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

8.8.2.1. Генерация

Серверу-источнику **следует** передавать Last-Modified для любого выбранного представления, когда можно обоснованно и согласованно определить дату последнего изменения, поскольку использование поля в условных запросах и при оценке свежести кэша ([CACHING]) может существенно сократить ненужные передачи, а также значительно улучшить доступность и расширяемость сервиса.

Представление состоит обычно из множества частей, находящихся за интерфейсом ресурса. Временем последнего изменения обычно является моментом последнего изменения любой из частей. Определение этого момента для ресурса зависит от деталей реализации и выходит за рамки этой спецификации.

Серверу-источнику **следует** получать значение Last-Modified для представления как можно ближе к моменту генерации им значения Date для своего отклика. Это позволит получателю точно оценить время изменения представления, особенно вблизи времени генерации отклика.

Серверу-источнику с часами (см. параграф 5.6.7) **недопустимо** генерировать время Last-Modified позднее момента создания сервером сообщения с откликом (Date, параграф 6.6.1). Если время последнего изменения выводится из зависящих от реализации метаданных, которые относятся к будущему моменту по часам сервера-источника, сервер **должен** заменить это значением временем создания сообщения. Это предотвратит негативное влияние дат из будущего при проверке кэша. Серверу-источнику без часов **недопустимо** генерировать время Last-Modified, если только это значение не было присвоено ресурсу какой-либо другой системой (предположительно имеющей часы).

8.8.2.2. Сравнение

Поле Last-Modified в качестве валидатора неявно является мягким, если иное нельзя принять на основании любого приведённых ниже правил.

- Валидатор сравнивается сервером-источником с фактическим текущим валидатором для представления и сервер-источник точно знает, что связанное представление не изменялось дважды в течение секунды, охватываемой представленным валидатором;
- Валидатор будет применяться клиентом в поле заголовка If-Modified-Since, If-Unmodified-Since или If-Range, поскольку клиент имеет кэшированную запись для соответствующего представления и эта запись включает значение Date, которое по меньшей мере на 1 секунду позже времени Last-Modified, а у клиента есть основания полагать, что метки исходят от одних часом или разница между Last-Modified и Date достаточно велика, чтобы считать проблемы синхронизации часов малозначимыми.
- Валидатор сравнивается промежуточным кэшем с валидатором, сохраненным в записи этого кэша для данного представления, и запись в кэше включает значение Date, которое по меньшей мере на 1 секунду позже времени Last-Modified, а у клиента есть основания полагать, что метки исходят от одних часом или разница между Last-Modified и Date достаточно велика, чтобы считать проблемы синхронизации часов малозначимыми.

Этот метод основан на том, что в случае отправки сервером-источником двух разных откликов в течение 1 секунды с одинаковым значением Last-Modified, хотя бы в одном из них значения Date и Last-Modified будут совпадать.

8.8.3. ETag

Поле Etag в отклике представляет текущий тег сущности для выбранного представления, определённый по завершении обработки запроса. Тег сущности - это неанализируемый (opaque) валидатор для различения разных представлений одного ресурса, независимо смены состояния ресурса со временем, согласования содержимого, ведущего к действительности сразу нескольких представлений или обоих причин. Тег сущности состоит из неанализируемой строки, которая может включать префикс индикатора «слабости» (weakness).

```
ETag       = entity-tag

entity-tag = [ weak ] opaque-tag
weak       = %s"W/"
opaque-tag = DQUOTE *etagc DQUOTE
etagc      = %x21 / %x23-7E / obs-text
           ; VCHAR, за исключением двойных кавычек, плюс obs-text
```

Примечание. Ранее элемент opaque-tag был определён как строка в (quoted-string) кавычках (параграф 3.11 в [RFC2616]) и некоторые получатели могли снимать экранирование (unescape) символа \. Поэтому серверам следует избегать включения данного символа в теги сущности.

Тег сущности может быть надёжней для проверки, чем дата изменения, когда даты изменений хранить неудобно, односекундной точности значений HTTP-date недостаточно или даты изменения не поддерживаются согласованно.

Ниже показаны примеры тегов сущности.

```
ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""
```

Тег сущности может быть мягким или строгим валидатором, по умолчанию он является строгим. Если сервер-источник указывает тег сущности для представления, а генерация этого тега не соответствует всем требованиям к строгому валидатору (параграф 8.8.1), сервер-источник **должен** пометить тег как слабый, добавив к нему префикс W/ (с учётом регистра).

Отправитель **может** передавать поле ETag в разделе трейлеров (параграф 6.5). Однако трейлеры часто игнорируются, поэтому предпочтительно передавать ETag как поле заголовка, если он не создаётся в процесс отправки содержимого.

8.8.3.1. Генерация

Принцип работы тегов сущностей состоит в том, что лишь автор сервиса знает реализацию ресурса достаточно хорошо для выбора наиболее точного и эффективного механизма проверки ресурса, а с любым механизмом можно сопоставить простую последовательность октетов для удобства сравнение. Поскольку значение не анализируется (opaque), клиенту не требуется понимать, как строится каждый тег сущности. Например, ресурс, где применяются зависящие от реализации версии при каждом изменении, может использовать внутренний номер выпуска, возможно, в сочетании с идентификатором различий для согласования содержимого, чтобы точно различать представления. В других реализациях может применяться устойчивый к конфликтам хэш, комбинации атрибутов файла или метка времени изменения с точностью лучше 1 секунды.

Серверу-источнику **следует** передавать ETag для любого выбранного представления, когда можно обоснованно и согласованно определить дату последнего изменения, поскольку использование поля в условных запросах и при оценке свежести кэша ([CACHING]) может существенно сократить ненужные передачи, а также значительно улучшить доступность, надёжность и расширяемость сервиса.

8.8.3.2. Сравнение

Имеется две функции сравнения тегов сущностей в зависимости от возможности применять мягкое сравнение.

Строгое сравнение

Теги эквивалентны, если оба не являются мягкими и необрабатываемые значения совпадают посимвольно.

Мягкое сравнение

Теги эквивалентны, если необрабатываемые значения совпадают посимвольно независимо от наличия маркера мягкого тега.

В таблице 3 приведены примеры результатов сравнения пар тегов сущностей для строго и мягкого сопоставления.

Таблица 3.

ETag1	***ETag2***	***Строгое сравнение***	***Мягкое сравнение***
W/"1"	W/"1"	не соответствует	соответствует
W/"1"	W/"2"	не соответствует	не соответствует
W/"1"	"1"	не соответствует	соответствует
"1"	"1"	соответствует	соответствует

8.8.3.3. Пример со сменой ETag при согласовании содержимого

Рассмотрим ресурс, к которому применяется согласование содержимого (раздел 12), а представления, передаваемые в откликах на GET, отличаются полем заголовка Accept-Encoding (параграф 12.5.3).

>> Запрос

```
GET /index HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip
```

В этом случае отклик может (но не обязан) использовать кодирование содержимого gzip. Несжатый отклик имеет вид

>> Отклик

```
HTTP/1.1 200 OK
Date: Fri, 26 Mar 2010 00:05:00 GMT
ETag: "123-a"
Content-Length: 70
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Представление с кодированием gzip будет иметь вид

>> Отклик

```
HTTP/1.1 200 OK
Date: Fri, 26 Mar 2010 00:05:00 GMT
ETag: "123-b"
Content-Length: 43
Vary: Accept-Encoding
Content-Type: text/plain
Content-Encoding: gzip

...двоичные данные...
```

Примечание. Кодирование содержимого является свойством данных представления, поэтому строгий тег сущности для представления с кодированием содержимого отличается от тега сущности некодированного представления, чтобы предотвратить возможные конфликты при обновлениях кэша и запросах диапазона. Транспортное кодирование (раздел 7 в [HTTP/1.1]), напротив, применяется лишь при передаче сообщения и не ведёт к разным тегам сущности.

9. Методы

9.1. Обзор

Маркер метода запроса является основным индикатором семантики запроса, указывая цель, с которой клиент обратился с данным запросом и что он ожидает в качестве успешного результата. Семантика метода запроса может быть уточнена семантикой полей заголовков, если та не противоречит методу. Например, клиент может включить в запрос поля условий (параграф 13.1), чтобы действия определялись текущим состоянием целевого ресурса.

Протокол HTTP предназначен служить интерфейсом к системам распределенных объектов. Метод запроса вызывает действие, применяемое к целевому ресурсу так же, как удалённый вызов метода можно передать указанному объекту.

```
method = token
```

Регистр символов в маркере метода принимается во внимание, поскольку маркер может служить шлюзом в объектно-ориентированные системы с учётом регистра в именах методов. По соглашению в именах стандартизованных методов применяются только заглавные буквы US-ASCII.

В отличие от распределенных объектов, стандартизованные методы запросов HTTP не привязаны к ресурсам, поскольку унифицированные интерфейсы обеспечивают лучшую наглядность и многократное использованием в сетевых системах [REST]. После определения стандартизованного метода он должен сохранять свою семантику применительно к любому ресурсу, хотя каждый из них сам определяет возможность применения этой семантики. Данная спецификация задаёт ряд стандартизованных методов, широко применяемых в HTTP (таблица 4).

Таблица 4.

Имя метода	*Описание*	*Параграф*
GET	Запрос передачи текущего представления целевого ресурса.	9.3.1
HEAD	Аналогично GET, но без передачи содержимого отклика.	9.3.2
POST	Выполнение специфичной для ресурса обработки содержимого запроса.	9.3.3
PUT	Замена всех текущих представлений целевого ресурса содержимым запроса.	9.3.4
DELETE	Удаление всех текущих представлений целевого ресурса.	9.3.5
CONNECT	Организация туннеля с сервером, указанным целевым ресурсом.	9.3.6
OPTIONS	Описание коммуникационных опций для целевого ресурса.	9.3.7
TRACE	Отслеживание прохождения сообщения по пути к целевому ресурсу.	9.3.8

Все серверы общего назначения **должны** поддерживать методы GET и HEAD, поддержка остальных **необязательна**.

Набор разрешаемых целевым ресурсом методов может быть указан в поле заголовка Allow (параграф 10.2.1), однако этот набор может меняться динамически. Серверу-источнику, получившему запрос с непонятным или не реализованным методом, **следует** отвечать кодом статуса 501 (Not Implemented). При получении сервером-источником запроса с понятным и реализованным, но не разрешённым методом **следует** отвечать кодом 405 (Method Not Allowed).

Для применения в HTTP могут определяться методы, не включённые в эту спецификацию. Такие методы должны регистрироваться в реестре Hypertext Transfer Protocol (HTTP) Method Registry, как описано в параграфе 16.1.

9.2. Общие свойства методов

9.2.1. Безопасные методы

Метод запроса считается безопасным (safe), если заданная для него семантика по существу предусматривает лишь чтение (read-only), т. е. клиент не запрашивает и не ожидает какой-либо смены состояния сервера-источника в результате применения безопасного метода к целевому ресурсу. Предполагается, что разумное применение безопасного метода не будет наносить серверу-источнику какого-либо ущерба, потерь или необычной нагрузки.

Это определение безопасных методов не препятствует реализации поведения, которое может быть опасным, предусматривать не только чтение или приводить к побочным эффектам при вызове безопасного метода. Важно отметить, что клиент не запрашивает такого поведения и не может нести за него ответственность. Например, большинство серверов добавляет сведения о запросах в системный журнал доступа по завершении каждого запроса, независимо от метода, и это считается безопасным даже с учётом возможного переполнения хранилища системных журналов и отказа сервера. Аналогично, безопасный запрос, инициированный выбором рекламы на Web-странице, может приводить к побочным эффектам, таким как списание средств с рекламного счета..

Определённые в этой спецификации методы GET, HEAD, OPTIONS и TRACE считаются безопасными.

Методы делятся на безопасные и небезопасные для того, чтобы автоматизированные процессы поиска (spider) и оптимизации производительности кэширования (pre-fetching) могли работать без опасения причинить ущерб. Кроме того, это позволяет применить подходящие ограничения на автоматизированное применение небезопасных методов при обработке потенциально недоверенного содержимого.

Пользовательским агентам **следует** различать безопасные и небезопасные методы при разрешению пользователю действий, чтобы тот мог узнать о небезопасных действиях до их запроса.

Если ресурс устроен так, что параметры в URI цели влияют на выбор действия, владелец ресурса должен убедиться, что действие соответствует семантике метода запроса. Программы редактирования содержимого Web часто указывают действия в параметрах запроса, например, page?do=delete. Если ресурс предполагает выполнение небезопасных действий, его владелец **должен** отключать или запрещать такие действия при использовании безопасных методов запроса. Отказ от этого может приводить к неприятным побочным эффектам, когда автоматизированные процессы используют GET в каждой ссылке URI для обслуживания ссылок (link), предварительной выборки, создания поисковых индексов и т. п.

9.2.2. Идемпотентные методы

Метод запроса считается идемпотентным, если повторное или многократное применение этого метода на сервере не даёт дополнительного эффекта по сравнению с первым использованием. Определённые в этой спецификации методы PUT, DELETE, а также безопасные методы запросов являются идемпотентными.

Как и определение безопасности, свойство идемпотентности относится лишь к тому, что запрошено пользователем, сервер волен записывать каждый запрос отдельно, сохранять историю контроля выпусков или реализовать неидемпотентные побочные эффекты для каждого идемпотентного запроса.

Идемпотентные методы отличаются тем, что запрос можно автоматически повторить при коммуникационном отказе, возникшем до того, как клиент смог прочитать отклик сервера. Например, если клиент передал запрос PUT, а базовое соединение было разорвано до получения какого-либо отклика, клиент может организовать новое соединение и повторить идемпотентный запрос. Клиент знает, что повторный запрос будет давать тот же предусмотренный эффект, даже если исходный запрос уже был выполнен, хотя отклики могут различаться. Клиенту **не следует** автоматически повторять запрос с неидемпотентным методом, пока у него нет возможности знать, что семантика запроса на деле идемпотентна независимо от метода или исходный запрос не был применён. Например, агент пользователя может автоматически повторить запрос POST, если ему известно (из устройства или конфигурации), что запрос безопасен для этого ресурса. Аналогично, пользовательский агент, созданный специально для работы с репозиторием контроля версий, может восстанавливаться после частичных отказов, проверяя целевые ресурсы после отказа соединения, восстанавливая или исправляя частично внесённые изменения, а затем повторяя неудавшийся ранее запрос. Некоторые клиенты используют более рискованный подход и пытаются угадать, когда возможно автоматически повторить запрос. Например, клиент может автоматически повторить запрос POST, если базовое транспортное соединение было закрыто до приёма какой-либо части отклика, особенно если использовалось бездействовавшее сохраняемое соединение.

Прокси **недопустимо** автоматически повторять неидемпотентные запросы. Клиентам **не следует** автоматически повторять запрос после отказа автоматического повтора.

9.2.3. Методы и кэширование

Чтобы кэш мог сохранить и использовать отклик, связанный метод должен явно разрешать кэширование и подробно описывать, при каких условиях отклик можно использовать для выполнения последующих запросов. Если определение метода не включает этого, кэширование невозможно. Дополнительные требования приведены в [CACHING].

Эта спецификация задаёт семантику кэширования для методов GET, HEAD и POST, но большинство реализаций поддерживают лишь GET и HEAD.

9.3. Определения методов

9.3.1. GET

Метод GET запрашивает передачу текущего выбранного представления целевого ресурса. Успешный отклик отражает «одинаковость» URI цели (параграф 1.2.2 of [URI]). Таким образом, извлечение идентифицируемой цели по протоколу HTTP обычно выполняется с помощью запроса GET по идентификатору, связанному с потенциальной возможностью предоставить информацию в отклике 200 (OK).

GET является основным механизмом извлечения информации и объектом почти всех оптимизаций производительности. Приложения, создающие URI для каждого важного ресурса, могут пользоваться результатами такой оптимизации, обеспечивая возможность многократного её использования, что создаёт в сети эффект, способствующий дальнейшему расширению Web.

Заманчиво рассматривать идентификаторы ресурсов как имена путей в удалённой файловой системе, а представления - как копии содержимого этих файлов. В действительности многие ресурсы реализованы именно так (см. вопросы безопасности, связанные с этим, в параграфе 17.3), однако это не всегда так.

Интерфейс HTTP для ресурса может быть реализован как дерево объектов содержимого, программное представление записей баз данных или шлюз в другие информационные системы. Даже в случае привязки механизма сопоставления URI к файловой системе сервер-источник можно настроить для исполнения файлов с запросами на входе и передачи вывода в качестве представления взамен передачи файлов напрямую. В любом случае только серверу-источнику нужно знать, как каждый идентификатор ресурса соответствует реализации и как это реализация выбирает и передаёт текущее представление целевого ресурса.

Клиент может изменить семантику GET, чтобы это стало запросом диапазона (range request) для передачи лишь некоторых частей выбранного представления, передав в запросе поле заголовка Range (параграф 14.2).

Хотя структура запросного сообщения не зависит от применяемого метода, содержимое принятого запроса GET не имеет общепринятой семантики, не может менять смысл (значение) или цель запроса, а некоторые реализации могут отвергать запрос и закрывать соединение из-за возможности атак с контрабандой (smuggling) запросов (параграф 11.2 в [HTTP/1.1]). Клиенту **не следует** создавать содержимое в запросе GET, если запрос не направлен непосредственно серверу-источнику, который ранее указал (по основному каналу или иначе), что такой запрос имеет смысл и будет адекватно поддерживаться. Серверу-источнику **не следует** полагаться на частные соглашения о получении содержимого, поскольку при взаимодействиях HTTP часто неизвестны посредники в цепочке запросов.

Отклик на запрос GET может кэшироваться и кэш **может** служить для выполнения последующих запросов GET и HEAD, если поле заголовка Cache-Control (параграф 5.2 в [CACHING]) не задаёт иное.

При извлечении информации с помощью механизма, создающего URI цели из предоставленных пользователем сведений, таких как поля запроса формы с применением GET, возможно предоставление конфиденциальных (sensitive) данных, которые не следует раскрывать в URI (см. параграф 17.9). В некоторых случаях данные можно отфильтровать или преобразовать так, чтобы сведения не раскрывались. В иных случаях, особенно, когда кэширование откликов не приносит пользы, применение метода POST (параграф 9.3.3) вместо GET позволяет передать такие данные в содержимом запроса, а не в URI цели.

9.3.2. HEAD

Метод HEAD аналогичен GET, но серверу **недопустимо** передавать содержимое в отклике. HEAD применяется для получения метаданных о выбранном представлении без передачи данных представления (часто для проверки гипертекстовых ссылок или обнаружения недавних изменений).

Серверу **следует** передавать в отклике на запрос HEAD те же поля заголовков, которые он передал бы в отклике на GET. Однако сервер **может** не создавать поля заголовков, которые имеют смысл только при генерации содержимого. Например, некоторые серверы буферизуют динамический отклик на запрос GET, пока не создан минимальный объем данных, чтобы более эффективно разграничивать небольшие отклики или принимать поздние решения по выбору содержимого. Такой отклик для GET может включать, например, поля Content-Length и Vary, которые не нужны в отклике HEAD. Такие незначительные несоответствия считаются более предпочтительными, нежели генерация и отбрасывание содержимого для HEAD, поскольку запросы HEAD обычно служат для повышения эффективноси.

Хотя структура запросных сообщений не зависит от применяемого метода, содержимое принятого запроса HEAD не имеет общепринятой семантики, не может менять смысл (значение) или цель запроса, а некоторые реализации могут отвергать запрос и закрывать соединение из-за возможности атак с контрабандой (smuggling) запросов (параграф 11.2 в [HTTP/1.1]). Клиенту **не следует** создавать содержимое в запросе HEAD, если запрос не направлен непосредственно серверу-источнику, который ранее указал (по основному каналу или иначе), что такой запрос имеет смысл и будет адекватно поддерживаться. Серверу-источнику **не следует** полагаться на частные соглашения о получении содержимого, поскольку при взаимодействиях HTTP часто неизвестны посредники в цепочке запросов.

Отклик на запрос HEAD может кэшироваться и кэш **может** служить для выполнения последующих запросов HEAD, если поле заголовка Cache-Control (параграф 5.2 в [CACHING]) не задаёт иное. На отклик HEAD могут влиять кэшированные ранее отклики на GET (см. параграф 4.3.5 в [CACHING]).

9.3.3. POST

Метод POST запрашивает у целевого ресурса обработку вложенного в запрос представления в соответствии с семантикой этого ресурса. Например, POST можно использовать для

- предоставления блока данных, таких как значения полей HTML-формы, для процесса обработки данных;
- отправки сообщения на «доску объявлений», в группу новостей, список рассылки, блог или иную группу статей;
- создания нового ресурса, который ещё не идентифицирован сервером-источником;
- добавления данных в имеющиеся представления ресурса.

Сервер-источник указывает семантику отклика подходящим кодом статуса в зависимости от результата обработки POST. Для этого подходят почти все коды, заданные в этой спецификации, за исключением 206 (Partial Content), 304 (Not Modified), 416 (Range Not Satisfiable).

Если на сервере-источнике создаётся один или несколько ресурсов в результате обработки запроса POST, серверу **следует** передавать отклик 201 (Created) с полем заголовка Location, указывающим идентификатор созданного первичного ресурса (параграф 10.2.2), и представлением, описывающим статус запроса со ссылкой на новые ресурсы.

Отклики на запросы POST можно кэшировать лишь при наличии в них явных сведений о свежести (см. параграф 4.2.1 в [CACHING]) и совпадении поля заголовка Content-Location со значением, указанным в URI цели запроса POST (параграф 8.7). Кэшированный отклик POST может применяться для выполнения последующих запросов GET или HEAD. Запрос POST не может быть выполнен с использованием кэшированного отклика POST, поскольку метод POST может быть небезопасным (см. раздел 4 в [CACHING]).

Если результат обработки POST эквивалентен представлению имеющегося ресурса, сервер-источник **может** перенаправить агент пользователя, передавая отклик 303 (See Other) с идентификатором имеющегося ресурса в поле Location. Преимуществом такого подхода является предоставление пользовательскому агенту идентификатора ресурса и передача представления с использованием метода, более подходящего для общего кэша, хотя это и связано с дополнительным запросом, если у агента пользователя ещё нет кэшированного представления.

9.3.4. PUT

Метод PUT запрашивает создание или замену состояния целевого ресурса состоянием, заданным представлением из содержимого сообщения с запросом. Успешное выполнение запроса PUT с данным представлением предполагает, что последующий запрос GET к тому же целевому ресурсу приведёт к отправке эквивалентного представления в отклике 200 (OK). Однако нет гарантий наблюдаемости такой смены состояния, поскольку целевой ресурс может параллельно обрабатываться другими пользовательскими агентами или подвергаться динамической обработке сервером-источником до получения последующего запроса GET. Отклик об успехе означает лишь, что намерения пользовательского агента были реализованы в процессе обработки запроса сервером-источником.

Если у целевого ресурса нет текущего представления и PUT создаёт его, сервер-источник **должен** передать агенту пользователя отклик 201 (Created). Если у целевого ресурса есть текущее представления и оно было изменено св соответствии с состоянием вложенного в запрос представления, сервер-источник **должен** передать отклик 200 (OK) или 204 (No Content) для индикации успешного выполнения запроса.

Серверу-источнику **следует** проверить, что представление PUT соответствует заданным для целевого ресурса ограничениям. Например, если сервер-источник определяет метаданные представления ресурса на основе URI, он должен убедиться, что содержимое полученного запроса PUT соответствует этим метаданным. Если представление PUT не соответствует целевому ресурсу, серверу-источнику **следует** привести их в соответствие, путём преобразования представления или ответить подходящим сообщением об ошибке с достаточными сведениями для разъяснения причины непригодности преставления. Предлагается использовать код 409 (Conflict) или 415 (Unsupported Media Type), причём последний относится к ограничениям для значений Content-Type. Например, если целевой ресурс настроен всегда иметь Content-Type со значением text/html, а в представлении PUT задано Content-Type image/jpeg, сервер-источник должен выполнить одно из указанных ниже действий.

a. Перенастроить целевой ресурс в соответствии с новым типом носителя.
b. преобразовать представление PUT в формат, соответствующий ресурсу до сохранения нового состояния.
c. отклонить запрос с кодом 415 (Unsupported Media Type), показывающим, что для целевого ресурса подходит лишь text/html, возможно включив ссылку на другой ресурс, который может подойти для нового представления.

HTTP не задаёт точного влияния метода PUT на состояние сервера-источника помимо того, что может быть выражено намерениями запроса от агента пользователя и семантикой отклика сервера-источника. Протокол не указывает, чем должен быть ресурс (в любом смысле этого слова) за интерфейсом, обеспечиваемым HTTP. Не задаётся способ «хранения» состояния ресурса, изменения этого хранилища в результате смены состояния ресурса, трансляции сервером-источником состояния ресурса в представления. Детали реализации, находящиеся за интерфейсом ресурса, намеренно скрываются сервером. Это относится и к хранению полей заголовков и трейлеров - хотя базовые поля заголовков, такие как Content-Type, обычно сохраняются при последующих запросах GET, обработка полей заголовка и трейлера зависит от принявшего запрос ресурса. Поэтому серверу-источнику **следует** игнорировать непонятные поля заголовка и трейлера в запросе PUT (т. е. не сохранять их как часть состояния ресурса).

Серверу-источнику **недопустимо** передавать поля валидатора (параграф 8.8), такие как ETag или Last-Modified, в отклике на успешно выполненный запрос PUT, пока данные представления из запроса не были сохранены без преобразований содержимого (т. е. новые данные представления ресурса идентичны содержимому запроса PUT), а значение поля валидатора не отражает новое представление. Это требование позволяет агенту пользователя знать, когда переданное им (и сохранённое в памяти) представление является результатом PUT и его не нужно снова извлекать с сервера-источника. Новые валидаторы, полученные в отклике, можно применять в будущих условных запросах, чтобы предотвратить случайную перезапись (параграф 13.1).

Принципиальным различием методов POST и PUT является назначение вложенного представления. Для метода POST целевой ресурс должен обрабатывать вложенное представление в соответствии с семантикой самого ресурса, а вложенное представление запроса PUT определено как замена состояния целевого ресурса. Поэтому назначение PUT является идемпотентным и видимым для посредников, хотя его точное влияние известно лишь серверу-источнику.

Корректная реализация PUT предполагает, что агент пользователя знает, какой целевой ресурс нужен ему. Службу, выбирающую URI от имени клиента после получения меняющего состояние запроса, **следует** реализовать на основе метода POST, а не PUT. Если сервер-источник не выполняет запрошенную PUT смену состояния целевого ресурса и вместо этого хочет применить запрос к другому ресурсу, например, перенесённому на другой URI, этот сервер **должен** передать подходящий отклик 3xx (Redirection), а пользовательский агент может сам принять решение о перенаправлении запроса.

Применение PUT к целевому ресурсу может оказывать побочное влияние на другие ресурсы. Например, статья может иметь URI для идентификации «текущей версии» (ресурса), отдельный от URI, указывающих каждую отдельную версию (другие ресурсы, которые в какой-то момент имели такое же состояние, что и текущая версия ресурса. Успешное применение PUT по URI «текущей версии» может создавать новый ресурс в дополнение к смене состояния целевого ресурса, а также приводить к созданию новых ссылок между связанными ресурсами.

Некоторые серверы-источники могут применять поле заголовка Content-Range (параграф 14.4) в качестве модификатора запроса для частичного выполнения PUT, как описано в параграфе 14.5.

Отклики на PUT не являются кэшируемыми. Если успешный запрос PUT проходит через кэш, где имеется один или несколько сохранённых откликов для URI цели, эти отклики аннулируются (см. параграф 4.4 в [CACHING]).

9.3.5. DELETE

Метод DELETE просит сервер-источник удалить связь между целевым ресурсом и его текущей функциональностью. Фактически это эквивалент команды rm в UNIX, он выражает операцию удаления сопоставления URI сервера-источника, а не ожидание удаления ранее ассоциированной информации.

Если у целевого ресурса есть одно или несколько текущих представлений, они могут быть уничтожены сервером-источником, а связанное с ними хранилище может быть использовано снова, что полностью зависит от природы ресурса и его реализации сервером-источником (это выходит за рамки данной спецификации). В результате запроса DELETE может потребоваться деактивация или архивирование других аспектов реализации ресурса, таких как соединения с базой данных или шлюзом. В общем случае предполагается, что сервер-источник разрешает применять DELETE лишь к ресурсам, для которых он имеет предписанный механизм удаления. Использование метода DELETE разрешается для сравнительно небольшого числа ресурсов, он применяется в основном в средах удалённой разработки (authoring), где пользователь имеет некоторое представление о результате применения запроса. Например, для ресурса, созданного ранее с помощью запроса PUT, или идентифицированного по полю заголовка Location после отклика 201 (Created) на запрос POST, может быть разрешён соответствующий запрос DELETE для отмены упомянутого действия. Аналогично, пользовательские агенты реализующие функции публикации (authoring), такие как клиенты контроля версий, использующие HTTP для удалённых операций, могут использовать DELETE на основе допущения, что создано пространство URI сервера для соответствия репозиторию версий.

Если метод DELETE применён, серверу-источнику **следует** передать

- отклик с кодом 202 (Accepted), если действие вероятно будет успешно, но ещё не было выполнено;
- отклик с кодом 204 (No Content), если действие выполнено и дополнительные сведения не требуются;
- отклик с кодом 200 (OK), если действие выполнено и сообщение с откликом включает представление с описанием статуса.

Хотя обрамление запросного сообщения зависит от метода, содержимое в запросе DELETE не имеет общепринятой семантики, не может изменить смысл или цель запроса и может заставить некоторые реализации отвергнуть запрос и закрыть соединение из-за возможности атаки с контрабандой запросов (smuggling, параграф 11.2 в [HTTP/1.1]). Клиенту **не следует** генерировать содержимое для запроса DELETE, если тот не адресован напрямую серверу-источнику, которые ранее не указал (по основному каналу или вне его), что такой запрос имеет смысл и будет адекватно поддержан. Серверу-источнику **не следует** полагаться на частные соглашения о получении содержимого, поскольку участники взаимодействия HTTP зачастую не знают о посредниках в цепочке запроса.

Отклики на DELETE не являются кэшируемыми. Если успешный запрос DELETE проходит через кэш, где имеется один или несколько сохранённых откликов для URI цели, эти отклики аннулируются (см. параграф 4.4 в [CACHING]).

9.3.6. CONNECT

Метод CONNECT запрашивает у получателя организацию туннеля с сервером-источником, указанным целью запроса и при успешной организации ограничивает поведение туннеля пересылкой данных «вслепую» в обоих направлениях пока туннель не будет закрыт. Туннели обычно применяются для создания сквозного виртуального соединения через 1 или несколько прокси, которое можно дополнительно защитить с помощью TLS (Transport Layer Security, [TLS13]).

В CONNECT применяется особая (уникальная для него) форма цели запроса, включающая только хост и номер порта на целевой стороне туннеля, разделённые двоеточием (:). Принятый по умолчанию порт не задан и клиент **должен** указать номер порта, даже если запрос CONNECT основан на ссылке URI с компонентом authority с известным портом (параграф 4.1). Например,

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

Сервер **должен** отвергать запросы CONNECT с отсутствующим или недействительным номером порта, возвращая обычно код статуса 400 (Bad Request).

Поскольку CONNECT меняет природу (запрос-отклик) соединения HTTP, в конкретных версиях HTTP могут применяться разные способы отображения семантики запроса на формат передачи протокола.

Метод CONNECT предназначен для запросов к прокси. Получатель может организовать туннель, соединившись напрямую с сервером, указанным как цель запроса, или пересылая CONNECT следующему входящему прокси при настройке на работу через другого посредника. Сервер-источник **может** воспринять запрос CONNECT, но на большинстве серверов-источников метод CONNECT не реализован.

Любой отклик 2xx (Successful) указывает, что отправитель (и все входящие прокси) переключились в туннельный режим сразу после раздела заголовков отклика и переданные затем данные будут поступать от сервера, указанного целью запроса. Все прочие отклики указывают, что туннель ещё не создан. Туннель разрывается при обнаружении его посредником закрытия соединения любой из сторон и посредник **должен** попытаться передать все остающиеся данные от закрывшей соединение стороны на другую сторону, закрыть оба своих соединения и затем отбросить все недоставленные данные.

Для установки полномочий на создание туннелей может применяться аутентификация прокси, например,

```
CONNECT server.example.com:443 HTTP/1.1
Host: server.example.com:443
Proxy-Authorization: basic aGVsbG86d29ybGQ=
```

Создание туннелей с произвольными серверами сопряжено со значительным риском, особенно при соединении с общеизвестным или зарезервированным портом TCP, не предназначенным для трафика Web. Например, CONNECT с example.com:25 предлагает прокси соединиться с портом, зарезервированным для трафика SMTP и если его разрешить, прокси можно использовать для ретрансляции спама. Прокси, поддерживающим CONNECT, **следует** разрешать использование метода лишь с ограниченным набором известных портов или настраиваемым списком безопасных целей запросов.

Серверу **недопустимо** передавать поля заголовка Transfer-Encoding или Content-Length в откликах 2xx (Successful) на CONNECT. Клиент **должен** игнорировать поля Content-Length и Transfer-Encoding в откликах об успехе CONNECT.

Сообщения с запросом CONNECT не имеют содержимого. Интерпретация данных после раздела заголовков сообщения с запросом CONNECT зависит от применяемой версии HTTP.

Отклики на CONNECT не кэшируются.

9.3.7. OPTIONS

Метод OPTIONS запрашивает сведения о коммуникационных опциях, доступных для целевого ресурса на сервере-источнике или посреднике. Это позволяет клиенту определить опции и/или требования, связанные с ресурсом, или возможности сервера, не предполагая действий с ресурсом. Запрос OPTIONS с символом * в качестве цели запроса (параграф 7.1) относится к серверу в целом, а не к конкретному его ресурсу. Поскольку коммуникационные опции сервера обычно зависят от ресурса, запрос с * полезен лишь в качестве ping или no-op и не даёт клиенту ничего, кроме проверки возможностей сервера. Например, этом можно использовать для проверки прокси на соответствие HTTP/1.1. Если указана иная цель, запрос OPTIONS применяется к опциям, доступным при взаимодействии с целевым ресурсом.

Серверу, генерирующему отклик об успехе OPTIONS, **следует** передавать любой заголовок, который может указать реализуемые сервером дополнительные возможности, которые применимы к целевому ресурсу (например, Allow), включая расширения, не заданные этой спецификацией. При наличии в отклике содержимого оно также может описывать коммуникационные опции в представлении для машины или человека. Спецификация не задаёт стандартный формат для такого представления, но он может быть определён в будущих расширениях HTTP.

Клиент запросе может передать поле заголовка Max-Forwards в запросе OPTIONS, чтобы указать конкретного получателя в цепочке запроса (см. параграф 7.6.2). Прокси **недопустимо** генерировать поле Max-Forwards при пересылке запроса, если запрос не содержал поля Max-Forwards. Клиент, создающий запрос OPTIONS с содержимым, **должен** включить в него действительное поле заголовка Content-Type с описанием типа носителя для представления. Отметим, что данная спецификация не задаёт такого содержимого.

Отклики на OPTIONS не кэшируются.

9.3.8. TRACE

Метод TRACE запрашивает удалённый возврат запроса на прикладном уровне. Конечному получателю запроса **следует** «отражать» клиенту принятое сообщение, исключая описанные ниже поля, как содержимое отклика 200 (OK). Одним из вариантов является применение формата message/http (параграф 10.1 в [HTTP/1.1]). Конечным получателем является сервер-источник или первый сервер, получивший запрос с Max-Forwards 0 (параграф 7.6.2).

Клиенту **недопустимо** включать в запрос TRACE поля с конфиденциальными сведениями, которые могут раскрываться в отклике. Например, агенту пользователя неразумно передавать в запросе сохранённые свидетельства

пользователя (раздел 11) или cookie [COOKIE]. Конечному получателю запроса **следует** при формировании содержимого отклика исключать поля, которые могут содержать конфиденциальные данные.

Метод TRACE позволяет клиенту увидеть, что было получено на другой стороне, и использовать эти сведения для тестирования и диагностики. Особый интерес представляют значения поля заголовка Via (параграф 7.6.3), позволяющие отследить цепочку запросов. Использование поля заголовка Max-Forwards позволяет клиенту ограничить длину цепочки запросов, что полезно при тестировании цепочки серверов, пересылающих сообщения по петле.

Клиенту **недопустимо** включать содержимое в запрос TRACE.

Отклики на TRACE не кэшируются.

10. Содержимое сообщения

10.1. Поля контекста запроса

Описанные ниже поля заголовка предоставляют дополнительные сведения о контексте запроса, включая информацию о пользователе, его агенте и ресурсе, на который нацелен запрос.

10.1.1. Expect

Поле заголовка Expect в запросе указывает некий набор моделей поведения (ожидания), который должен поддерживаться сервером для корректной обработки этого запроса.

```
Expect =       #expectation
expectation = token [ "=" ( token / quoted-string ) parameters ]
```

Регистр символов в значении поля Expect не принимается во внимание.

Эта спецификация определяет единственное ожидание 100-continue, не задавая для него параметров. Сервер, получивший поле Expect с иным значением, **может** вернуть отклик 417 (Expectation Failed) для указания невозможности выполнить ожидание.

Ожидание 100-continue информирует получателей, что клиент собирается отправить (предположительно крупное) содержимое в данном запросе и хочет получить промежуточный отклик 100 (Continue), если метода, URI цели и полей заголовка недостаточно для немедленного ответа об успехе, перенаправлении или ошибке. Это позволяет клиенту дождаться сигнала о целесообразности отправки содержимого до его реальной передачи, что может повысить эффективность в случае большого объёма данных или ожидания ошибки клиентом (например, при передаче метода, изменяющего состояние в первый раз без уже проверенных свидетельств для аутентификации). Например, запрос, начинающийся с

```
PUT /somewhere/fun HTTP/1.1
Host: origin.example.com
Content-Type: video/h264
Content-Length: 1234567890987
Expect: 100-continue
```

позволяет серверу-источнику ответить сообщением об ошибке, таким как 401 (Unauthorized) или 405 (Method Not Allowed) до того, как клиент начнёт загружать канал ненужной передачей данных.

Ниже приведены требования к клиенту.

- **Недопустимо** помещать ожидание 100-continue в запрос без содержимого.
- Клиент, ожидающий отклика 100 (Continue) перед отправкой следующего запроса, **должен** передать поле заголовка Expect с ожиданием 100-continue.
- Клиент, передавший ожидание 100-continue, не обязан ждать в течение какого-либо определённого времени и **может** продолжить отправку содержимого до получения отклика. Поскольку отклики 100 (Continue) не могут передаваться через посредников HTTP/1.0, такому клиенту **не следует** выжидать неопределённое время перед отправкой содержимого.
- Клиенту, получившему код статуса 417 (Expectation Failed) в отклике на запрос с ожиданием 100-continue, **следует** повторить этот запрос без 100-continue, поскольку код 417 указывает лишь отсутствие поддержки ожиданий в цепочке запросов (например, прохождение через сервер HTTP/1.0).

Ниже приведены требования к серверу.

- Сервер, получивший ожидание 100-continue в запросе HTTP/1.0, **должен** игнорировать это ожидание.
- Сервер **может** не передавать отклик 100 (Continue), если он уже получил часть или все содержимое для соответствующего запроса или обрамление говорит об отсутствии содержимого.
- Сервер, передавший отклик 100 (Continue), **должен** в конечном итоге отправить финальный код статуса после получения и обработки содержимого запроса, если соединение не было закрыто преждевременно.
- Серверу, ответившему финальным кодом статуса до прочтения всего содержимого запроса, **следует** указать, намерен ли он закрыть соединение (см., например, параграф 9.6 в [HTTP/1.1]) или продолжить чтение содержимого запроса.

Получив запрос HTTP/1.1 (или выше) с методом, URI цели и полным разделом заголовков, содержащим ожидание 100-continue и указание наличия содержимого, сервер-источник **должен** передать один из откликов:

- незамедлительный отклик с финальным кодом статуса, если этот статус можно определить по номеру, URI цели и полям заголовка;
- незамедлительный отклик 100 (Continue), чтобы побудить клиента передать содержимое запроса.

Серверу-источнику недопустимо **дожидаться** содержимого перед отправкой отклика 100 (Continue).

Получив запрос HTTP/1.1 (или выше) с методом, URI цели и полным разделом заголовков, содержащим ожидание 100-continue и указание наличия содержимого, прокси **должен** выполнить одно из действий:

- незамедлительно передать отклик с финальным кодом статуса, если этот статус можно определить по номеру, URI цели и полям заголовка;
- переслать запрос в направлении сервера-источника, отправив соответствующую строку запроса (request-line) и раздел заголовков следующему входящему серверу.

Если сервер полагает (из конфигурации или прошлого взаимодействия), что следующий входящий сервер поддерживает лишь HTTP/1.0, он **может** незамедлительно передать отклик 100 (Continue), чтобы побудить клиента передать содержимое запроса.

10.1.2. From

В поле заголовка From указывается адрес электронной почты Internet человека, контролирующего пользовательский агент, передающий запрос. Адрес должен быть пригодным для машины в соответствии с определением mailbox в " in Section 3.4 of [RFC5322]:

```
From    = mailbox

mailbox = <mailbox, см. [RFC5322], параграф 3.4>
```

Пример поля приведён ниже.

```
From: spider-admin@example.org
```

Поле From редко передают нероботизированные пользовательские агенты. Агенту пользователя **не следует** отправлять поле From без явного указания (настройки) пользователем, поскольку это может противоречить интересам приватности пользователя или политики безопасности сайта. Роботизированным пользовательским агентам **следует** передавать действительное поле From, чтобы можно было связаться с человеком, управляющим роботом, при возникновении проблем на сервере, например, при отправке роботом излишних, нежелательных или непригодных запросов

Серверам **не следует** использовать поле From для контроля доступа или аутентификации, поскольку это значение, по-видимому, будет доступно всем, кто получает или отслеживает запрос, и часто записывается в системные журналы (log) и отчёты об ошибках без какой-либо надежды на сохранение приватности.

10.1.3. Referer

Поле заголовка Referer [sic] позволяет агенту пользователя указать для ресурса ссылку URI, из которой был выведен URI цели (т. е. referrer - ссылающийся, если игнорировать ошибку в имени поля). Агенту пользователя **недопустимо** включать компоненты fragment и userinfo в ссылку URI[URI], если таковые имеются при создании поля Referer.

```
Referer = absolute-URI / partial-URI
```

Значением поля является absolute-URI или partial-URI. В последнем случае (раздел 4), указанный URI задаётся относительно URI цели ([URI], раздел 5).

Поле заголовка Referer позволяет серверам генерировать обратные ссылки на другие ресурсы для простой аналитики, протоколирования, оптимизации кэширования и т. п. Оно также позволяет находить устаревшие и ошибочные ссылки для поддержки. Некоторые серверы используют поле Referer как средство отклонения ссылок с других сайтов (deep linking) или ограничения поддельных кросс-сайтовые запросов (cross-site request forgery или CSRF), хотя поле содержится не во всех запросах.

Пример поля представлен ниже.

```
Referer: http://www.example.org/hypertext/Overview.html
```

Если URI цели получен из источника, не имеющего своего URI (например, с клавиатуры пользователя или из записи в закладках), агент пользователя **должен** исключить значение поля Referer или указать в нем значение about:blank. В значении поля Referer не обязательно передавать полный URI ссылающегося ресурса и агент пользователя **может** отсекать части, отличные от ссылающегося источника.

Поле заголовка Referer может раскрывать сведения о контексте запроса и истории просмотров, что создаёт проблему приватности, если идентификатор ссылающегося ресурса раскрывает персональные данные (такие как имя учётной записи) или ресурс, который должен быть конфиденциальным (например, находится за межсетевым экраном или внутри защищённой службы). Большинство пользовательских агентов общего назначения не передаёт поле Referer, когда ссылающимся ресурсом является локальный URI типа file или data. Агенту пользователя **не следует** передавать поле Referer, если доступ к ссылающемуся ресурсу происходил по защищённому протоколу и цель запроса имеет иной источник, нежели ссылающийся ресурс, если этот ресурс не разрешает явно отправку Referer. Агенту пользователя **недопустимо** передавать поле Referer в незащищённом запросе HTTP, если доступ к ссылающемуся ресурсу происходил по защищённому протоколу (соображения безопасности рассмотрены в параграфе 17.9).

Известно, что некоторые посредники без разбора удаляют поля Referer в исходящих запросах. Это имеет неприятный побочный эффект снижения защиты от атак CSRF, которые могут быть гораздо более опасными для пользователей. Посредники и расширения пользовательских агентов, желающие ограничить раскрытие информации в полях Referer, должны ограничиваться конкретными изменениями вроде замены внутренних доменных имён псевдонимами или отсечки компонентов запроса и/или пути. Посредникам АСК **не следует** изменять или удалять поля заголовка Referer, использующие ту же схему и хост, что и URI цели.

10.1.4. TE

Поле заголовка TE описывает возможности клиента в части транспортного кодирования и трейлера. Как указано в параграфе 6.5, поле TE с элементом trailers в запросе указывает, что клиент не будет отбрасывать поля трейлера. TE также применяется в HTTP/1.1 для информирования серверов о транспортном кодировании, которое клиент способен воспринять в отклике. На момент публикации документа транспортные кодировки применялись только в HTTP/1.1 (см. раздел 7 в [HTTP/1.1]).

Значением поля TE служит список элементов, каждый из которых (кроме trailers) включает маркер имени транспортного кодирования с необязательным весом, указывающим относительную предпочтительность этого кодирования для клиента (параграф 12.4.2), и необязательные параметры транспортного кодирования.

```
TE                = #t-codings
t-codings         = "trailers" / ( transfer-coding [ weight ] )
transfer-coding   = token *( OWS ";" OWS transfer-parameter )
transfer-parameter = token BWS "=" BWS ( token / quoted-string )
```

Отправитель TE **должен** также передать опцию соединения TE в поле заголовка Connection (параграф 7.6.1), чтобы посредники не пересылали это поле.

10.1.5. User-Agent

Поле заголовка User-Agent содержит сведения о создавшем запрос агенте пользователя, которые часто используются серверами для определения масштаба проблем совместимости, обхода или адаптации к ограничениям конкретных пользовательских агентов в откликах, а также для аналитики, связанной с используемыми браузерами и операционными системами. Агенту пользователя **следует** включать поле заголовка User-Agent в каждый запрос, если явно не задано иное поведение.

```
User-Agent = product *( RWS ( product / comment ) )
```

Значением поля User-Agent служит один или несколько идентификаторов продукции, за каждым из которых могут следовать комментарии (параграф 5.6.5), что в совокупности указывает программу пользовательского агента и её значимые варианты. По традиции идентификаторы продукции указываются в порядке снижения их значимости для идентификации программы пользовательского агента. Каждый идентификатор включает имя и может включать версию.

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

Отправителю **следует** ограничиваться сведениями, требуемыми для идентификации продукции и **недопустимо** включать в идентификатор рекламу и иные несущественные сведения. Отправителю **не следует** указывать в версии продукции сведения, не являющиеся идентификатором версии (т. е. разные версии продукции с одним именем должны различаться лишь элементами product-version). Например,

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

Агенту пользователя **не следует** создавать поле заголовка User-Agent с излишними подробностями и **следует** ограничивать добавление вариантов другими сторонами. Слишком длинные и подробные значения User-Agent увеличивают задержку для запросов и создают риск нежелательной идентификации пользователя (fingerprinting).

Разработчикам рекомендуется не применять маркеры продукции от других реализаций для декларирования совместимости с теми, поскольку это противоречит назначению поля. Если агент пользователя маскируется под другой агент, получатели могут счесть, что пользователь намеренно хочет видеть отклики, предназначенные для указанного агента, даже если он работает не так, как фактический агент пользователя.

10.2. Поля контекста отклика

Ниже рассмотрены поля заголовка отклика, содержащие сведения об отклике в дополнение к коду статуса, включая информацию о сервере, целевом ресурсе и связанных ресурсах.

10.2.1. Allow

Поле заголовка Allow содержит список методов, анонсируемых как поддерживаемые целевым ресурсом. Назначением этого поля является исключительно информирование получателя о доступных для целевого ресурса методах запроса.

```
Allow = #method
```

Пример поля приведён ниже.

```
Allow: GET, HEAD, PUT
```

Фактический набор разрешённых методов определяется сервером-источником в момент каждого запроса. Сервер-источник **должен** генерировать поле заголовка Allow в откликах 405 (Method Not Allowed) и **может** включать поле в любой другой отклик. Пустое значение поля Allow указывает, что ресурс не поддерживает никаких методов, и может передаваться в откликах 405, если ресурс временно отключён его конфигурацией.

Прокси **недопустимо** менять поле заголовка Allow и ему не нужно понимать все указанные в поле методы для обработки сообщения в соответствии с базовыми правилами.

10.2.2. Location

Поле заголовка Location применяется в некоторых откликах для указания определённого ресурса по отношению к отклику. Тип взаимоотношений определяется сочетанием семантики метода запроса и кода статуса.

```
Location = URI-reference
```

Значением поля является одна ссылка URI. Когда ссылка имеет относительную форму ([URI], параграф 4.2), окончательное значение определяется её распознаванием относительно URI цели ([URI], раздел 5).

Для откликов 201 (Created) значение Location указывает на основной ресурс, созданный запросом, для откликов 3xx (Redirection) - на предпочтительный ресурс для автоматического перенаправления запроса.

Если значение Location, приведённое в отклике 3xx (Redirection), не имеет фрагментного компонента, агент пользователя **должен** обработать перенаправление, как будто значение наследует фрагментный компонент ссылки URI, использованной для генерации URI цели (перенаправление наследует фрагмент исходной ссылки, если он есть). Например, запрос GET, созданный для ссылки URI http://www.example.org/~tim, может приводить к отклику 303 (See Other) с полем заголовка

```
Location: /People.html#tim
```

которое предлагает пользовательскому агенту перенаправление на http://www.example.org/People.html#tim.

Запрос GET для ссылки URI http://www.example.org/index.html#larry может возвращать отклик 301 (Moved Permanently) с полем заголовка

```
Location: http://www.example.net/index.html
```

которое предлагает агенту пользователя перенаправление на http://www.example.net/index.html#larry с сохранением исходного идентификатора фрагмента.

В некоторых случаях идентификатор фрагмента в поле Location будет неуместен. Например, поле заголовка Location в отклике 201 (Created) предполагается содержащим URI для созданного ресурса.

> Примечание. Некоторые получатели пытаются выполнять восстановление по полю Location, которое не является действительной ссылкой URI. Данная спецификация не требует и не задаёт такой обработки, но разрешает её в целях отказоустойчивости. Значение поля заголовка Location не может содержать список элементов, поскольку запятая в качестве разделителя элементов списка является допустимым символом данных с ссылке URI. При отправке недействительного сообщения с несколькими строками Location его получатель на пути доставки может объединить такие строки в одно значение. Восстановление действительного значения поля Location в таких ситуациях затруднено и может не быть совместимым в разных реализациях.
>
> Поле заголовка Content-Location (параграф 8.7) отличается от Location тем, что Content-Location указывает наиболее конкретный ресурс, соответствующий вложенному представлению. Поэтому в откликах могут присутствовать оба поля (Location и Content-Location).

10.2.3. Retry-After

Серверы передают поле заголовка Retry-After для указания пользовательскому агенту времени, которое он должен выждать перед отправкой следующего (follow-up) запроса. В отклике 503 (Service Unavailable) поле Retry-After указывает ожидаемый срок недоступности услуг для клиента, в откликах 3xx (Redirection) - минимальное время, которое клиенту следует выждать перед отправкой перенаправленного запроса. Значением поля Retry-After может быть HTTP-date или величина задержки (в секундах) после приёма сообщения.

```
Retry-After = HTTP-date / delay-seconds
```

Значение delay-seconds является неотрицательным целым числом, представляющим время в секундах.

```
delay-seconds  = 1*DIGIT
```

Ниже представлены два примера использования.

```
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120
```

Во втором примере задержка составляет 2 минуты.

10.2.4. Server

Поле заголовка Server содержит сведения о программах, используемых сервером-источником для обработки запроса, которые часто применяются клиентами для идентификации масштаба указанных проблем совместимости, обхода или адаптации к ограничениям конкретного сервера, а также для анализа распространённости серверов и операционных систем. Сервер-источник **может** включать поле Server в свои отклики.

```
Server = product *( RWS ( product / comment ) )
```

Поле заголовка Server содержит один или несколько идентификаторов продукции, каждый из которых может сопровождаться комментариями (параграф 5.6.5), что в совокупности указывает программы сервера-источника и их значимые варианты. По традиции идентификаторы продукции указываются в порядке снижения их значимости для идентификации программы пользовательского агента. Каждый идентификатор включает имя и может включать версию, как описано в параграфе 10.1.5. Пример поля представлен ниже.

```
Server: CERN/3.0 libwww/2.17
```

Серверу-источнику **не следует** создавать поля заголовка с излишними подробностями и следует ограничивать добавление вариантов другими сторонами. Слишком длинные и подробные значения Server увеличивают задержку для откликов и создают риск утечки деталей реализации, что может (слегка) упростить злоумышленникам поиск и использование известных брешей в защите.

11. Аутентификация HTTP

11.1. Схема проверки подлинности

HTTP обеспечивает общую модель контроля доступа и проверки подлинности на основе расширяемого набора схем аутентификации «вызов-отклик» (challenge-response), которые клиент и сервер могут применять для запроса аутентификационных данных у другой стороны. Для указания схемы служит не зависящий от регистра маркер (token)

```
auth-scheme    = token
```

Этот документ не задаёт схем аутентификации помимо общей модели. Имеющиеся и новые схемы задаются независимо и должны регистрироваться в реестре Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry. Например, схемы аутентификации basic и digest определены в [RFC7617] и [RFC7616], соответственно.

11.2. Параметры аутентификации

За схемой аутентификации следует дополнительная информация, требуемая для проверки подлинности по этой схеме, в форме разделённых запятыми параметров или одной последовательности символов, способной содержать сведения в представлении base64.

```
token68        = 1*( ALPHA / DIGIT /
                     "-" / "." / "_" / "~" / "+" / "/" ) *"="
```

Синтаксис token68 разрешает 66 незарезервированных символов URI ([URI]) и несколько других, что позволяет помещать представление base64, base64url (безопасный алфавит для URL и имён файлов), base32, base16 (hex) с заполнением или без него, но без включения пробельных символов ([RFC4648]).

Параметры аутентификации задаются парами имя-значение и маркеры имён сравниваются без учёта регистра символов, а каждое имя параметра **должно** присутствовать в вызове лишь один раз.

```
auth-param     = token BWS "=" BWS ( token / quoted-string )
```

Значение параметра указывается маркером (token) или строкой в кавычках (quoted-string, параграф 5.6). Определениям схем аутентификации нужно разрешать оба варианта, чтобы получатели могли применять компоненты разбора независимо от схемы аутентификации. Для совместимости с прежними версиями определения схем аутентификации могут ограничиваться поддержкой у отправителя лишь одного варианта. Это может быть важно, если известно, что в развёрнутых реализациях могут возникать отказы для одного из двух форматов.

11.3. Вызов и отклик

Отклик 401 (Unauthorized) может применяться серверами-источниками для запросов (challenge) аутентификации пользовательских агентов, включая поле заголовка WWW-Authenticate, содержащее хотя бы один вызов, применимый к запрашиваемому ресурсу. Отклик 407 (Proxy Authentication Required) применяется прокси для запросов аутентификации клиента, включая поле заголовка Proxy-Authenticate, содержащее хотя бы один вызов, применимый к прокси для запрошенного ресурса.

```
challenge   = auth-scheme [ 1*SP ( token68 / #auth-param ) ]
```

Примечание. Многие клиенты сталкиваются с отказами при разборе вызовов с неизвестной схемой. Для обхода этой проблемы следует сначала указывать наиболее широко поддерживаемые схемы (например, basic).

Агент пользователя, желающий аутентифицироваться на сервере-источнике после получения отклика 401 (Unauthorized) обычно (но не всегда) может сделать это, включив в запрос поле заголовка Authorization. Агент пользователя, желающий аутентифицироваться на прокси после получения отклика 407 (Proxy Authentication Required) обычно обычно (но не всегда) может сделать это, включив в запрос поле заголовка Proxy-Authorization.

11.4. Свидетельства

Значения полей Authorization и Proxy-Authorization содержат свидетельства (credentials) клиента для области действия запрашиваемого ресурса, основанные на вызове (challenge), полученном в отклике (возможно, когда-то раньше). При формировании значений полей агент пользователя должен выбрать вызов с понятной ему и наиболее безопасной по его мнению схемой аутентификации (auth-scheme) для получения свидетельств от пользователя. Передача свидетельств в полях заголовка предполагает существенные требования к конфиденциальности базового соединения, как описано в параграфе 17.16.1.

```
credentials = auth-scheme [ 1*SP ( token68 / #auth-param ) ]
```

При получении запроса для защищённого ресурса без свидетельств, с непригодными (например, неверный пароль) или неполными (например, когда схема аутентификации требует более одного кругового обхода) свидетельствами серверу-источнику **следует** передать отклик 401 (Unauthorized) с полем заголовка WWW-Authenticate, включающим хотя бы один (возможно, новый) вызов, пригодный для запрошенного ресурса. При получении запроса без свидетельств для прокси, с непригодными или неполными свидетельствами прокси **следует** передавать отклик 407 (Proxy Authentication Required) с полем Proxy-Authenticate, включающим хотя бы один (возможно, новый) вызов, пригодный для запрошенного ресурса. Сервер, получивший действительные свидетельства, которых недостаточно для получения доступа, должен отвечать кодом 403 (Forbidden, параграф 15.5.4).

HTTP не ограничивает приложения использованием простой схемы «вызов-отклик» для проверки подлинности. Можно применять дополнительные механизмы, такие как аутентификация на транспортном уровне или инкапсуляция сообщений, а также дополнительные поля заголовка со сведениями для аутентификации. Однако спецификация не задаёт таких механизмов. Отметим, что различные механизмы аутентификации пользователей применяют для передачи связанных с аутентификацией маркеров поля заголовка Set-Cookie и Cookie, определённые в [COOKIE].

11.5. Организация защищённого пространства

Параметр аутентификации realm зарезервирован для использования схемами аутентификации, которые хотят указать область действия защиты.

Пространство защиты определяется источником (origin, параграф 4.3.1) на сервере, к которому происходит доступ с использованием realm. Это позволяет разделить защищаемые ресурсы на сервере по нескольким пространствам защиты, каждое из которых использует свою схему аутентификации и/или базу данных о полномочиях. Значением realm является строка, обычно назначаемая сервером-источником, которая может иметь дополнительную семантику в зависимости от схемы проверки подлинности. Отметим, что отзыв может включать несколько вызовов с одинаковым значением auth-scheme, но разными realm.

Пространство защиты определяет область, к которой могут быть автоматически применены свидетельства. Если предыдущему запросу были предоставлены полномочий, агент пользователя **может** воспользоваться теми же свидетельствами для всех прочих запросов в том же пространстве защиты в течение периода, определяемого схемой аутентификации, параметров и/или пользовательских предпочтений (например, настраиваемым периодом бездействия). Область защиты и, следовательно, запросы, к которым автоматически применяются свидетельства, может быть неведома клиентам без дополнительной информации. Схема аутентификации может задавать параметры, описывающие область защиты. Если схема аутентификации не задаёт это специально, область действия пространства защиты не может выходить за пределы сервера.

По историческим причинам отправитель **должен** генерировать только строки в кавычках. Получателям, возможно, придётся поддерживать маркеры и строки в кавычках для максимальной совместимости с имеющимися клиентами, которые уже давно воспринимают обе формы.

11.6. Аутентификация пользователей на сервере-источнике

11.6.1. WWW-Authenticate

Поле WWW-Authenticate в заголовке отклика указывает схемы и параметры аутентификации, применимые для целевого ресурса.

```
WWW-Authenticate = #challenge
```

Сервер, генерирующий отклик 401 (Unauthorized), **должен** передавать поле заголовка WWW-Authenticate, содержащее хотя бы один вызов. Сервер **может** генерировать поле заголовка WWW-Authenticate в других откликах для указания того, что предоставление этих (или иных) свидетельств может влиять на отклик.

Пересылающим отклик прокси **недопустимо** изменять в откликах поля заголовка WWW-Authenticate.

Агентам пользователя рекомендуется с осторожностью подходить к анализу значений поля, поскольку в нем может быть несколько вызовов, каждый из которых может содержать список параметров, разделённых запятыми. Кроме того, само поле заголовка может указываться неоднократно. Например,

```
WWW-Authenticate: Basic realm="simple", Newauth realm="apps",
                  type=1, title="Login to \"apps\""
```

В этом заголовке указано два вызова - схема Basic с realm = simple и схема Newauth с realm = apps, а также два дополнительных параметра type и title. Некоторые пользовательские агенты могут не понимать такие формы, поэтому отправка полей WWW-Authenticate с несколькими элементами в одной строке может препятствовать совместимости.

> Примечание. Грамматика вызовов также использует списки, поэтому последовательность из разделённых пробелами запятых может считаться относящейся к предыдущему вызову или пустым элементом списка вызовов. На практике эта неоднозначность не меняет семантики значения поля заголовка и не представляет опасности.

11.6.2. Проверка и предоставление полномочий

Поле заголовка Authorization позволяет агенту пользователя подтвердить свою подлинность серверу-источнику, обычно (но необязательно) это происходит после получения отклика 401 (Unauthorized). Значение поля состоит из свидетельств с аутентификационными данными пользовательского агента для realm запрашиваемого ресурса.

```
Authorization = credentials
```

Если запрос аутентифицирован и область действия (realm) указана, те же свидетельства считаются действительными для всех запросов в этой области realm (при условии, что схема аутентификации не задаёт иное, например, смену свидетельств в зависимости от вызова или использование синхронизированных часов).

Прокси, пересылающему запрос, **недопустимо** менять поле Authorization в этом запросе. Детали и требования к обработке поля заголовка Authorization в кэшах HTTP приведены в параграфе 3.5 [CACHING].

11.6.3. Authentication-Info

Схемы аутентификации HTTP могут использовать поле отклика Authentication-Info для передачи сведений после восприятия свидетельств аутентификации клиента. Эти сведения могут включать завершающее сообщение от сервера (например, аутентификацию сервера). Значением поля является список параметров (пар имя-значение) с синтаксисом auth-param, заданным в параграфе 11.3. Эта спецификация описывает лишь базовый формат и применяющие Authentication-Info схемы аутентификации будут задавать отдельные параметры. Например, схема аутентификации Digest задаёт множество параметров в параграфе 3.5 [RFC7616].

```
Authentication-Info = #auth-param
```

Поле Authentication-Info может применяться в любом отклике HTTP, независимо от метода запроса и кода отклика. Семантика поля определяется схемой аутентификации, указанной полем заголовка Authorization (параграф 11.6.2) в соответствующем запросе.

Пересылающим прокси не разрешается вносить в это поле какие-либо изменения.

Authentication-Info можно передавать в трейлере (параграф 6.5), если схема аутентификации явно разрешает это.

11.7. Аутентификация клиентов на прокси

11.7.1. Proxy-Authenticate

Поле заголовка Proxy-Authenticate включает хотя бы один вызов (challenge) для указания схем аутентификации и параметров, применяемых к прокси для этого запроса. Прокси должен передавать хотя бы одно поле заголовка Proxy-Authenticate в каждом создаваемом отклике 407 (Proxy Authentication Required).

```
Proxy-Authenticate = #challenge
```

В отличие от WWW-Authenticate, поле заголовка Proxy-Authenticate применяется только к следующему исходящему клиенту в цепочке запроса. Это обусловлено тем, что свидетельства для аутентификации имеются, скорей всего, лишь у клиента, выбравшего данный прокси. При использовании нескольких прокси в одном административном домене, например, офисных или региональных кэширующих прокси, свидетельства обычно генерируются агентом пользователя и передаются по иерархии, пока не будут восприняты. Поэтому в такой конфигурации будет создаваться впечатление пересылки Proxy-Authenticate, поскольку каждый прокси будет передавать тот же набор вызовов.

При разборе поля используются такие же правила, как для WWW-Authenticate (параграф 11.6.1).

11.7.2. Proxy-Authorization

Поле заголовка Proxy-Authorization позволяет клиенту идентифицировать себя (или своего пользователя) на прокси, требующем аутентификации. Значением поля служат свидетельства с данными аутентификации клиента на прокси и/или область действия (realm) для запрашиваемого ресурса.

```
Proxy-Authorization = credentials
```

В отличие от Authorization, поле заголовка Proxy-Authorization применяется только к следующему входящему прокси, потребовавшему аутентификации с использованием поля заголовка Proxy-Authenticate. При наличии в цепочке нескольких прокси, поле Proxy-Authorization воспринимает (consume) первый входной прокси, ожидающий получения свидетельств. Прокси **может** ретранслировать свидетелства из запроса клиента следующему прокси, если это является механизмом совместной аутентификации запроса несколькими прокси.

11.7.3. Proxy-Authentication-Info

Поле заголовка отклика Proxy-Authentication-Info аналогично Authentication-Info, но применяется к аутентификации на прокси (параграф 11.3) и его семантика определяется схемой аутентификации, указанной полем заголовка Proxy-Authorization (параграф 11.7.2) в соответствующем запросе.

```
Proxy-Authentication-Info = #auth-param
```

Однако, в отличие от Authentication-Info, поле заголовка Proxy-Authentication-Info применяется только к следующему исходящему клиенту в цепочке отклика. Это обусловлено тем, что, скорее всего, требуемые для аутентификации свидетельства имеются лишь у клиента, выбравшего данный прокси. Однако при использовании в административном домене (офисные или региональные кэширующие прокси большой корпоративной сети) свидетельства обычно генерируются агентом пользователя и передаются по иерархии, пока не будут восприняты. Поэтому создаётся впечатление о пересылке Proxy-Authentication-Info, поскольку каждый прокси передаёт одно значение.

Proxy-Authentication-Info можно передавать как поле трейлера (параграф 6.5), когда схема аутентификации явно разрешает это.

12. Согласование содержимого

Когда отклик имеет содержимое (в результате успеха или ошибки), у сервера-источника часто есть несколько вариантов представления такой информации, например, в разных форматах, языках, кодировках. У пользователей или их агентов также могут быть разные возможности, характеристики и предпочтения, способные влиять на выбор для доставки лучшего из имеющихся представлений.

Эта спецификация задаёт три модели согласования содержимого, которые могут быть в протоколе, - упреждающее (proactive) согласование, где сервер выбирает представление на основе заявленных агентом пользователя предпочтений, реактивное согласование, где сервер предоставляет агенту пользователя список представлений для выбора, и согласование содержимого запроса (request content), где агент пользователя выбирает представление для будущих запросов на основе предпочтений сервера, заявленных в прошлых откликах.

Другие модели согласования содержимого включают условное содержимое (conditional content), где представление состоит из нескольких частей, которые выборочно отображаются на основе предпочтений пользовательского агента, активное содержимое (active content), где представление включает сценарий, выдающий дополнительные (более конкретные) запросы на основе характеристик агента пользователя, и прозрачное согласование содержимого (Transparent Content Negotiation [RFC2295]), где представление выбирается посредниками. Эти схемы не являются взаимоисключающими и у каждой имеются свои плюсы и минусы в плане применимости и практичности.

Отметим, что во всех случаях HTTP не знает о семантике ресурса. Согласованность откликов сервера-источника на запросы с учётом времени и вариантов согласования содержимого, а значит и «одинаковость» наблюдаемых представлений ресурса, полностью определяется тем, какая сущность или алгоритм выбирает или генерирует отклики.

12.1. Упреждающее согласование

Передача агентом пользователя предпочтений по согласованию содержимого в запросе называется упреждающим (proactive) согласованием (согласование, управляемое сервером). Выбор основывается на доступных представлениях отклика (например, язык, кодировка и т. п.) с учётом сведений из запроса, включая явное согласование указанных ниже полей заголовка и неявные характеристики, такие как сетевой адрес клиента или части поля User-Agent.

Предварительное согласование полезно в тех случаях, когда агенту пользователя сложно описать алгоритм выбора из числа доступных представлений или сервер желает отправить своё «лучшее представление» в первом отклике агенту пользователя (если «лучшее представление» подойдёт пользователю, это позволит избежать одного кругового обхода для последующего запроса). Для лучшего «угадывания» агент пользователя **может** передать в запросе поля заголовка, описывающие его предпочтения.

Ниже указаны серьёзные недостатки упреждающего согласования.

- Серверу невозможно точно определить «лучший» отклик для пользователя, поскольку для этого нужно знать все возможности агента пользователя и предполагаемое использование отклика (например, просмотр на экране или вывод на печать).
- Требование к пользовательскому агенту описывать свои возможности в каждом запросе может быть неэффективным (с учётом наличия разных представлений лишь для малой части откликов) и вносить риск утраты приватности пользователя.
- Усложняется реализация сервера-источника и алгоритмов генерации откликов на запросы.
- Ограничиваются возможности неоднократного использования откликов из общего кэша.

Агент пользователя не может полагаться на постоянный учёт предпочтений упреждающего согласования, поскольку сервер-источник может не поддерживать такое согласование для запрошенного ресурса или решить, что отправка отклика, не соответствующего предпочтениям пользовательского агента, лучше, чем передача 406 (Not Acceptable).

Для указания частей, использованных в алгоритме выбора, в откликах, являющихся предметом упреждающего согласования, часто передаётся поле заголовка Vary (параграф 12.5.5). Описанные ниже поля заголовка Accept, Accept-Charset, Accept-Encoding, Accept-Language позволяют агенту пользователя участвовать в упреждающем согласовании содержимого. Переданные в этих полях предпочтения применяются к любому содержимому отклика, включая представления целевого ресурса, ошибок или статуса обработки, а также могут применяться к текстовым строкам, появляющимся в протоколе.

12.2. Реактивное согласование

При реактивном согласовании (управляется агентом) выбор содержимого (независимо от кода статуса) выполняется пользовательским агентом после приёма исходного отклика. Механизм реактивного согласования может быть простым как список ссылок на варианты представления.

Если агент пользователя не удовлетворён содержимым исходного отклика, он может передать запрос GET на один или несколько альтернативных ресурсов для получения других представлений. Выбор таких альтернатив может выполняться автоматически (агентом пользователя) или вручную (например, выбором пользователя из гипертекстового меню).

Сервер может не передавать исходное представление, отличное от списка вариантов и тем самым указать предпочтительность реактивного согласования с агентом пользователя. Например, варианты, указанные в откликах с кодом статуса 300 (Multiple Choices) и 406 (Not Acceptable), включают сведения о доступных представлениях, чтобы пользователь или его агент мог реагировать сделанным выбором.

Реактивное согласование подходит в случаях, если отклики различаются по часто используемым параметрам (например, тип, язык, кодировка), когда сервер не способен определить возможности пользовательского агента по запросу, а при использовании общего кэша для распределения нагрузки на сервер и снижения загрузки сети.

Недостатком реактивного согласования является необходимость передачи агенту пользователя списка вариантов, ведущей к росту воспринимаемой пользователем задержки при передаче списка в полях заголовка, и необходимость передачи повторного запроса для получения выбранного представления. Кроме того, эта спецификация не задаёт механизм для поддержки автоматического выбора, хотя и не препятствует разработке таких механизмов.

12.3. Согласование содержимого запроса

Когда предпочтения по согласованию содержимого передаются в отклике сервера, указанные в списке предпочтения называют согласованием содержимого запроса (request content negotiation), поскольку они влияют на выбор содержимого последующих запросов к данному ресурсу. Например, в отклике могут передаваться поля заголовка Accept (параграф 12.5.1) и Accept-Encoding (параграф 12.5.3) для указания предпочтительных типов носителя в последующих запросах к этому ресурсу. Поле Accept-Patch, определённое в параграфе 3.1 [RFC5789], позволяет обнаружить, какие типы содержимого воспринимаются в запросах PATCH.

12.4. Свойства полей согласования содержимого

12.4.1. Отсутствие полей

Для каждого из полей согласования содержимого запрос без такого поля означает отсутствие у отправителя предпочтений по данному аспекту согласования. Если поле согласования содержимого присутствует в запросе, но ни одно из доступных для отклика представлений не может считаться приемлемым для него, сервер-источник может принять поле запроса, возвращая отклик 406 (Not Acceptable), или игнорировать его, как будто это поле не является предметом согласования содержимого. Однако это не означает, что клиент сможет воспользоваться представлением.

> Примечание. Агент пользователя, передающий такие поля заголовка, упрощает серверу идентификацию человека по характеристикам запроса от его агента (параграф 17.13).

12.4.2. Значения качества

Поля согласования содержимого, заданные этой спецификацией, используют общий параметр с именем q (без учёта регистра) для назначения относительного веса соответствующего вида содержимого. Этот вес называют значением качества (quality value или qvalue), поскольку параметр с таким же именем часто применяется в конфигурации серверов для указания веса относительному качеству различных представлений, которые могут быть выбраны для ресурса.

Вес нормализуется к действительному числу от 0 до 1, где 0,001 является наименее, 1 - наиболее предпочтительным, а 0 указывает неприемлемое качество. При отсутствии параметра q применяется вес 1.

```
weight = OWS ";" OWS "q=" qvalue
qvalue = ( "0" [ "." 0*3DIGIT ] )
       / ( "1" [ "." 0*3("0") ] )
```

Отправителю qvalue **недопустимо** генерировать больше 3 цифр после запятой. Пользовательские настройки также должны соблюдать это ограничение.

12.4.3. Шаблонные значения

Для большинства этих полей заголовка задано использование шаблона * для выбора неуказанных значений. При отсутствии такого шаблона не указанные явно значения считаются неприемлемыми. В поле Vary шаблон означает неограниченный разброс.

> Примечание. На практике использование шаблонов в полях согласования содержимого имеет ограниченное применение, поскольку редко бывает полезно сказать: «Я предпочитаю image/* больше или меньше некого конкретного значения». Передавая Accept: */*;q=0, клиент может явно запросить отклик 406 (Not Acceptable), если более предпочтительного формата не существует, но при этом он должен быть готов воспринять иной отклик, поскольку сервер волен игнорировать это предпочтение.

12.5. Поля согласования содержимого

12.5.1. Accept

Поле заголовка Accept пользовательские агенты могут использовать для указания своих предпочтений в части типов носителей. Например, можно использовать это поле для указания того, что запрос ограничен восприятием небольшого набора желаемых типов, как в случае запроса встроенного изображения. При передаче отклика сервером в поле Accept содержатся сведения о предпочтительных типах содержимого в последующих запросах к тому же ресурсу.

```
Accept = #( media-range [ weight ] )

media-range    = ( "*/*"
                 / ( type "/" "*" )
                 / ( type "/" subtype )
                ) parameters
```

Символ * служит для группировки типов носителя в диапазоны, */* указывает все типы, а type/* - все субтипы указанного типа (type). Элемент media-range может включать параметры типа носителя, применимые к диапазону. За каждым media-range могут следовать применимые параметры типа носителя (например, charset), а затем - необязательный параметр q, указывающий относительный вес (параграф 12.4.2).

Предыдущие спецификации разрешали включать дополнительные параметры расширения после параметра weight. Грамматика расширения accept (accept-params, accept-ext) была исключена про причине сложности определения, отсутствия реального использования и большей простоты развёртывания с новыми полями заголовка. Отправителям, использующим weight, **следует** передавать параметр q последним (после всех параметров media-range). Получателям **следует** обрабатывать любые параметры с именем q как вес, независимо от местоположения параметра.

> Примечание. Использование параметра q для управления согласованием содержимого будет конфликтовать с одноимённым параметром типа носителя, поэтому регистр типов носителя запретил параметров с именем q.

Пример расширения представлен ниже.

```
Accept: audio/*; q=0.2, audio/basic
```

Это интерпретируеся как: «Я предпочитаю audio/basic, но можно передать иной тип звука, являющийся лучшим после снижения качества на 80%».

Ниже представлен более сложный пример.

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
```

Это можно описать как одинаковую предпочтительность text/html и text/x-c с передачей при их отсутствии представления text/x-dvi, а при отсутствии и его - text/plain.

Диапазоны носителей могут переопределяться более узкими диапазонами или конкретными типами. Если к данному типу применимо более одного диапазона носителей, предпочтение отдаётся более конкретному. Например,

```
Accept: text/*, text/plain, text/plain;format=flowed, */*
```

задаёт показанный ниже порядок предпочтений.

1. `text/plain;format=flowed`
2. `text/plain`
3. `text/*`
4. `*/*`

Фактор качества, связанный с данным типом носителя, определяется путём нахождения наиболее предпочтительного диапазона, соответствующего типу. Например,

```
Accept: text/*;q=0.3, text/plain;q=0.7, text/plain;format=flowed,
        text/plain;format=fixed;q=0.4, */*;q=0.5
```

приведёт к привязке указанных в таблице 5 значений.

Таблица 5.

Тип носителя	***Значение Quality***
text/plain;format=flowed	1
text/plain	0,7
text/html	0,3
image/jpeg	0,5
text/plain;format=fixed	0,4
text/html;level=3	0,3[1]

> Примечание. Агенту пользователя может быть предоставлен набор принятых по умолчанию значения качества для некоторых диапазонов носителей. Однако, если пользовательский агент не является закрытой системой, не способной взаимодействовать с другими агентами представление (rendering), принятый по умолчанию набор должен быть настраиваемым со стороны пользователя.

12.5.2. Accept-Charset

Агент пользователя может передать поле заголовка Accept-Charset для указания своих предпочтений в части кодировки символов в текстовом содержимом отклика. Например, это поле позволяет пользовательским агентам, способным понимать более полные и специальные наборы символов, сообщить об этом серверу-источнику, чтобы тот мог представлять информацию с таких кодировках.

```
Accept-Charset = #( ( token / "*" ) [ weight ] )
```

Имена кодировок определены в параграфе 8.3.2. Агент пользователя **может** связать с каждым набором символов параметр качества для ранжирования предпочтений пользователя, как описано в параграфе 12.4.2. Например,

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

Специальный символ * в поле Accept-Charset соответствует каждой кодировке, не указанной в других местах поля.

> Примечание. Поле Accept-Charset признано устаревшим, поскольку кодировка UTF-8 стала практически повсеместно и передача подробного списка предпочитаемых пользователем кодировок ведёт к ненужному расходу пропускной способности и способствует упрощению пассивного снятия отпечатков (параграф 17.13). Большинство пользовательских агентов общего назначения не передаёт Accept-Charset без специальной настройки.

[1] В оригинале ошибочно указано значение 0,7. См. https://www.rfc-editor.org/errata/eid7138. *Прим. перев.*

12.5.3. Accept-Encoding

Поле заголовка Accept-Encoding может служить для указания предпочтений при кодировании содержимого (параграф 8.4.1). При отправке агентом пользователя в запросе Accept-Encoding указывает приемлемое в отклике кодирование содержимого, при отправке сервером в отклике - сведения о предпочтительном кодировании в последующих запросах к тому же ресурсу. Маркер identity применяется как псевдоним отсутствия кодирования (no encoding) для взаимодействия при отсутствии предпочтений для кодировки.

```
Accept-Encoding  = #( codings [ weight ] )
codings          = content-coding / "identity" / "*"
```

Каждому кодированию **может** быть назначено значение качества (вес - weight), представляющее уровень предпочтительности для этого кодирования (параграф 12.4.2). Символ * в поле Accept-Encoding соответствует любому доступному типу кодирования, не указанному явно в этом поле. Примеры значений поля приведены ниже.

```
Accept-Encoding: compress, gzip
Accept-Encoding:
Accept-Encoding: *
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

Сервер проверяет пригодность кодирования для данного кодирования с помощью указанных ниже правил.

1. При отсутствии в запросе поля заголовка Accept-Encoding любое кодирование считается приемлемым для пользовательского агента.
2. Представление без кодирования содержимого, оно считается приемлемым, если явно не исключено полем заголовка Accept-Encoding, содержащим identity;q=0 или *;q=0 без указания более конкретного identity.
3. Кодирование представления, совпадающее с одним из указанных в поле Accept-Encoding, считается приемлемым, если оно не сопровождается qvalue = 0 (в параграфе 12.4.2 указано, что qvalue = 0 означает неприемлемость - not acceptable.)

В представлении может использоваться несколько вариантов кодирования, однако большинство способов кодирования просто являются разными способами достижения одной цели (например, сжатия данных). При выборе между вариантами кодирования, имеющими общую цель, предпочтение отдаётся варианту с большим значением qvalue.

Пустое поле Accept-Encoding указывает, что не хочет получать в отклике какое-либо кодирование содержимого. Если в запросе имеется непустое поле Accept-Encoding и ни один из вариантов кодирования представления не указан в числе приемлемых, серверу-источнику **следует** передавать отклик без какого-либо кодирования содержимого, если только кодирование сущности (identity) не указано как неприемлемое.

Поле Accept-Encoding в отклике указывает кодирование содержимого, которое ресурс готов воспринять в связанном запросе. Значение поля оценивается так же, как в запросе.

Отметим, что эта информация относится только к конкретному запросу и набор поддерживаемых вариантов кодирования может меняться с течением времени или зависеть от других аспектов запроса (например, от метода).

Сервер, отклоняющий запрос из-за неподдерживаемого кодирования, должен передавать отклик с кодом статуса 415 (Unsupported Media Type) и полем заголовка Accept-Encoding, позволяющим клиенту различать проблемы, связанные с кодированием и типами носителей. Во избежание путаницы с проблемами из-за типа носителя серверам, отклоняющим запрос с кодом 415 по причинам, не связанным с кодированием содержимого, **недопустимо** включать в отклик поле заголовка Accept-Encoding.

Наиболее часто поле Accept-Encoding применяется в откликах с кодом статуса 415 (Unsupported Media Type) в ответ на оптимистическое использование клиентами вариантов кодирования содержимого. Однако это поле может служить для указания клиента поддерживаемых вариантов кодирования содержимого с целью оптимизации будущих взаимодействий. Например, ресурс может включить это поле в отклик 2xx (Successful), если содержимое запроса было достаточно велико, чтобы использовать сжатие, но клиент не сделал этого.

12.5.4. Accept-Language

Поле заголовка Accept-Language агент пользователя может применять для указания набора естественных языков, предпочитаемых в отклике. Теги языков определены в параграфе 8.5.1.

```
Accept-Language = #( language-range [ weight ] )
language-range  = <language-range, см. [RFC4647], параграф 2.1>
```

Для каждого language-range можно задать значение качества, указывающее оценку предпочтений пользователя для языков диапазона, как указано в параграфе 12.4.2. Например,

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

будет означать: «я предпочитаю датский язык, но буду воспринимать британский английский и другие варианты английского языка».

Отметим, что некоторые получатели трактуют порядок тегов языка, как признак снижения приоритета, особенно для тегов с одинаковым значением качества (отсутствие значения эквивалентно q=1). Однако полагаться на такое поведение нельзя. Для согласованности и максимальной совместимости многие пользовательские агенты назначают каждому тегу языка уникальное значение качества, размещая теги в порядке снижения качества. Дополнительное обсуждение приоритетов в списках языков приведено в параграфе 2.3 [RFC4647].

Раздел 3 в [RFC4647] задаёт различные схемы сопоставления. Реализации могут предложить наиболее подходящую для них схему. Схема Basic Filtering ([RFC4647], параграф 3.3.1) идентична схеме сопоставления, заданной ранее для HTTP в параграфе 14.4 [RFC2616].

Передача поля заголовка Accept-Language с полными языковыми предпочтениями в каждом запросе может противоречить ожиданиям пользователя в части приватности (параграф 17.13).

Поскольку разборчивость в значительной степени зависит от конкретного пользователя, пользовательским агентам нужно предоставлять пользователю возможность контроля языковых предпочтений (через настройку агента пользователя или по контролируемым пользователем настройкам по умолчанию). Пользовательским агентам, не предоставляющим пользователю таких возможностей, **недопустимо** передавать поле заголовка Accept-Language.

> Примечание. Пользовательские агенты должны предоставлять пользователю рекомендации по установке предпочтений, поскольку пользователи редко знакомы с описанными выше деталями сопоставления языков. Например, пользователь может предполагать, что при выборе en-gb ему будут предоставляться любой английский документ, если British English не доступен. В таком случае агент пользователя может предложить добавить в список тег en для более подходящего поведения.

12.5.5. Vary

Поле заголовка Vary в отклике описывает, какие части запросного сообщения, помимо метода и URI цели, могли повлиять на процесс выбора содержимого этого отклика сервером-источником.

```
Vary = #( "*" / field-name )
```

Поле Vary представляет собой шаблон * или список имён полей запроса, известных как выбирающие поля заголовка, которые могли играть роль при выборе представления для данного отклика. Выбирающими полями могут быть не только указанные в этой спецификации поля.

Список, содержащий шаблон *, указывает, что при выборе представления могли сыграть роль другие аспекты запроса, возможно не относящиеся к синтаксису (например, сетевой адрес клиента). Получатель не сможет определить, подходит ли этот отклик для последующего запроса, не переслав запрос серверу-источнику. Прокси недопустимо генерировать * в значении поля Vary. Например, отклик с полем заголовка

```
Vary: accept-encoding, accept-language
```

показывает, что сервер-источник мог использовать поля заголовка Accept-Encoding и Accept-Language (или их отсутствие) в качестве факторов, определивших содержимое отклика.

Поле Vary со списком имён полей служит двум целям.

1. Информировать получателей кэша, что им **недопустимо** использовать этот отклик для выполнения последующих запросов, пока в тех не содержатся те же значения указанных в списке полей заголовка, что и в исходном запросе (параграф 4.1 в [CACHING]), или повторное не было подтверждено сервером-источником. Иными словами, поле Vary раскрывает ключ кэша для сопоставления нового запроса с кэшированной записью.
2. Информировать пользовательские агенты-получатели о том, что этот отклик был предметом согласования содержимого (раздел 12) и при последующем запросе может быть передано другое представление, если в перечисленных полях его заголовка будут указаны другие значения (упреждающее согласование).

Серверу-источнику **следует** генерировать поле Vary в кэшируемом отклике, если он хочет, чтобы этот отклик можно было селективно использовать для последующих запросов. Обычно это происходит в случаях, когда содержимое отклика было адаптировано для лучшего соответствия предпочтениям, указанным в выбирающих полях заголовка, например, при выборе сервером-источником языка отклика по полю Accept-Language в заголовке отклика.

Поле Vary может не применяться, если сервер-источник считает варианты выбора содержимого менее значимыми, нежели влияние Vary на работу кэширования, особенно в случаях, когда повторное использование уже ограничено директивами кэширования откликов (параграф 5.2 в [CACHING]).

В Vary не требуется передавать поле Authorization, поскольку использование этого отклика другими пользователями запрещено определением поля (параграф 11.6.2). Если содержимое отклика выбирается с учётом области сети, а сервер-источник хочет, чтобы кэшированный отклик можно было повторно использовать даже при перемещении получателя в другую область сети, серверу-источнику не требуется указывать это в поле Vary.

13. Условные запросы

Условными называют запросы HTTP с одним или несколькими полями заголовка, указывающими предварительные условия, выполнение которых можно проверить до применения метода запроса к целевому ресурсу. В параграфе 13.2 описана проверка выполнения предварительных условий и порядок применения при наличии нескольких условий.

Условные запросы GET являются наиболее эффективным механизмом обновления кэша HTTP [CACHING]. Условия могут также применяться к методам, меняющим состояние, таким как PUT и DELETE, для предотвращения потери обновлений, когда один клиент случайно переписывает работу другого клиента, действующего параллельно.

13.1. Предварительные условия

Предварительные условия обычно задаются применительно к состоянию всего целевого ресурса (его текущего набора значений) или его состоянию в полученном ранее представлении (одно из значений в наборе). Если у ресурса несколько текущих представлений, каждое из которых имеет своё наблюдаемое состояние, предварительное условие будет предполагать, что сопоставление каждого запроса с выбранным представлением (параграф 3.2) согласовано во времени. Независимо от этого при несогласованности сопоставления или невозможности сервера выбрать подходящее представление не будет никакого вреда в результате невыполнения (false) предварительных условий.

Каждое из определённых ниже предварительных условий включает сравнение набора валидаторов из предыдущих представлений целевого ресурса с текущим состоянием набора валидаторов для выбранного представления (параграф 8.8). Таким образом, предварительные условия позволяют понять, изменилось ли состояние целевого ресурса относительно данного состояния, известного клиенту. Влияние такой оценки зависит от семантики метода и выбора условий, как описано в параграфе 13.2.

Предварительные условия, задаваемые в других спецификациях как поля расширения, могут ставить условия для всех получателей, состояния всего целевого ресурса или группы ресурсов. Например, поле заголовка If в WebDAV может делать запрос зависимым от различных аспектов нескольких ресурсов, таких как блокировки, если получатель понимает и реализует поле ([WEBDAV], параграф 10.4).

Расширяемость предварительных условий возможна лишь в тех случаях, когда неизвестное условие можно безопасно игнорировать (как If-Modified-Since), внедрение можно предположить в данной среде или его реализацию можно указать каким-либо свойством целевого ресурса. Это позволяет сосредоточиться на согласовании общих стандартов.

13.1.1. If-Match
Поле заголовка If-Match делает метод запроса условным в зависимости от наличия у принявшего запрос сервера-источника хотя бы одного текущего представления целевого ресурса. Условие считается выполненным, если в поле указано значение * или имеется текущее представление целевого ресурса, у которого тег сущности совпадает с одним из элементов списка тегов в значении поля.

Сервер-источник **должен** применять строгое сопоставление для тегов If-Match (параграф 8.8.3.2), поскольку намерение клиента состоит в предотвращении применения метода, если данные представления были изменены.

```
If-Match = "*" / #entity-tag
```
Примеры поля представлены ниже.

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```
If-Match чаще всего применяется с методами, меняющими состояние (например, POST, PUT, DELETE), для предотвращения случайной перезаписи при параллельной работе нескольких пользовательских агентов с одним ресурсом (проблема потери обновлений). В общем случае поле может применяться с любым методом, предполагающим выбор или изменение представления, для прерывания запроса при отсутствии текущего тега сущности выбранного представления в списке поля If-Match.

При получении сервером-источником выбирающего представление запроса с полем заголовка If-Match сервер **должен** до применения метода проверить выполнение условия If-Match в соответствии с параграфом 13.2.

1. При значении поля * условие будет выполняться при наличии у сервера-источника текущего представления для целевого ресурса.
2. Если поле содержит список тегов сущностей, условие считается выполненным при соответствии хотя бы одного тега из списка тегу сущности выбранного представления.
3. В иных случаях условия считаются не выполненными.

Серверу-источнику, оценивающему условие If-Match, **недопустимо** выполнять запрошенный метод, если условие не выполняется. Вместо этого сервер **может** указать невыполнение условия откликом с кодом 412 (Precondition Failed). Как вариант, сервер **может** передать отклик 2xx (Successful), если запрос был меняющей статус операцией, которая, по-видимому, уже применена (запрошенное пользовательским агентом изменение, уже внесено, но агент может не знать об этом, возможно, по причине потери предыдущего отклика или внесения эквивалентных изменений другим пользовательским агентом). Разрешение серверу-источнику передавать отклик об успехе, когда представляется, что запрос на изменение уже был применён, является более эффективным для многих случаев использования инструментов публикации, но связано с некоторым риском, если несколько пользовательских агентов делают очень похожие, но не согласованные запросы. Например, несколько пользовательских агентов пишущих в общий ресурс (например, неатомарное добавление) могут вступить в конфликт, влекущий возможность потери важных изменений состояния. Для таких ресурсов серверу-источнику лучше быть строгим и передавать отклик 412 для каждого запроса с небезопасным методом, где условие не было выполнено. В иных случаях исключение поля ETag из отклика об успехе может побудить пользовательский агент к отправке GET в качестве следующего запроса для устранения путаницы с текущим состоянием ресурса.

Клиент **может** передать поле заголовка If-Match в запросе GET для указания того, что он предпочитает получить отклик 412 (Precondition Failed), если выбранное представление не соответствует. Однако это полезно лишь в запросах с диапазоном (раздел 14) для завершения полученного ранее частичного представления, когда новое представление нежелательно. Поле If-Range (параграф 13.1.5) лучше подходит для запросов диапазона, если клиент предпочитает получит новое представление.

Кэш или посредник **может** игнорировать If-Match, поскольку его функциональная совместимость нужна лишь серверу-источнику.

Отметим, что поле заголовка If-Match со списком, содержащим * и другие значения (включая другие экземпляры *), синтаксически некорректно (и не разрешено для генерации) и вряд ли будет функционально совместимым.

13.1.2. If-None-Match
Поле заголовка If-None-Match делает метод запроса условным по отсутствию в кэше получателя или на сервере-источнике соответствующего значению поля текущего представления целевого ресурса. Для поля со значением * условие считается выполненным, если нет никакого текущего представления целевого ресурса. При указании в поле списка тегов сущностей условие считается выполненным, когда имеется тег сущности, не указанный в списке поля.

Получатель **должен** применять функцию мягкого сравнения тегов сущностей (параграф 8.8.3.2) при сравнении с полем If-None-Match, поскольку мягкие теги сущностей можно применять для проверки кэша даже при внесении изменений в данные представления.

```
If-None-Match = "*" / #entity-tag
```
Примеры заголовков приведены ниже.

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```
If-None-Match применяется в основном в условных запросах GET для эффективного обновления кэшированных данных с минимальными издержками на транзакции. Когда клиент хочет обновить один или несколько сохранённых откликов,

имеющих теги сущности, ему **следует** генерировать поле заголовка If-None-Match со списком этих тегов при создании запроса GET. Это позволяет получившему запрос серверу отправлять отклик 304 (Not Modified) в случаях, когда один из сохранённых откликов совпадает с выбранным представлением.

If-None-Match можно использовать со значением * для предотвращения непреднамеренного изменения небезопасным методом (например, PUT) имеющегося представления целевого ресурса, когда клиент считает, что у этого ресурса нет текущего представления (параграф 9.2.1). Этот один из вариантов потери обновлений, возможный при попытке нескольких клиентов создать исходное представление целевого ресурса.

При получении сервером-источником запроса, выбирающего представление, с полем заголовка If-None-Match этот сервер **должен** оценить выполнение условий If-None-Match в соответствии с параграфом 13.2 до применения метода.

1. При значении поля * условие не будет выполняться при наличии у сервера-источника текущего представления для целевого ресурса.
2. Если поле содержит список тегов сущностей, условие считается не выполненным при соответствии хотя бы одного тега из списка тегу сущности выбранного представления.
3. В иных случаях условия считаются выполненными.

Серверу-источнику, оценивающему условие If-None-Match, **недопустимо** выполнять запрошенный метод, если условие не выполняется. Вместо этого сервер **должен** передать отклик с кодом a) 304 (Not Modified) для запросов с методом GET или HEAD или b) 412 (Precondition Failed) в иных случаях.

Требования к обработке кэшем полученного поля заголовка If-None-Match заданы в параграфе 4.3.2 [CACHING].

Отметим, что поле заголовка If-None-Match со списком, содержащим * и другие значения (включая другие экземпляры *), синтаксически некорректно (и не разрешено для генерации) и вряд ли будет функционально совместимым.

13.1.3. If-Modified-Since

Поле заголовка If-Modified-Since делает метод запроса GET или HEAD условным по дате изменения выбранного представления Передача данных выбранного представления не происходит, если они не изменились с момента, указанного значением поля.

```
If-Modified-Since = HTTP-date
```

Пример поля приведён ниже.

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

Получатель **должен** игнорировать If-Modified-Since, если запрос включает поле If-None-Match, поскольку условие If-None-Match считается более точной заменой If-Modified-Since, а комбинация условий применяется лишь для совместимости со старыми посредниками, не реализующими If-None-Match. Получатель **должен** игнорировать If-Modified-Since, если поле не содержит действительного значения HTTP-date, имеет несколько элементов или метод запроса отличается от GET и HEAD. Получатель **должен** игнорировать If-Modified-Since, если ресурс не имеет доступной даты изменения. Получатель **должен** интерпретировать значение временной мети в поле If-Modified-Since по часам сервера-источника.

If-Modified-Since обычно применяется для двух целей: 1) эффективное обновление кэшированного представления без тега сущности и 2) ограничение области обхода web ресурсами, которые недавно были изменены.

При использовании для обновления кэша для генерации If-Modified-Since обычно берётся значение поля заголовка Last-Modified. Это обеспечивает функциональную совместимость в случаях когда часы не синхронизированы точно или сервер предпочитает учитывать лишь точное совпадение меток времени (из-за проблемы дат в Last-Modified, представляющихся прошлым, когда часы сервера корректируются или представление восстанавливается из резервной копии). Однако кэш иногда создаёт значение поля на основе других данных, таких как поле заголовка Date в кэшированном сообщении или момент приёма сообщения, особенно при отсутствии в кэшируемом сообщении поля Last-Modified.

При использовании для ограничения области извлечения данных недавним интервалом времён пользовательский агент создаёт значение поля If-Modified-Since на основе своих часов или поля Date в предыдущем отклике от сервера. Серверы-источники, проверяющие точное совпадение меток времени выбранного сопоставления с полем заголовка Last-Modified, не смогут помочь пользовательскому агенту ограничиться получением лишь данных, изменившихся в заданном интервале времени.

При получении сервером-источником запроса, выбирающего представление, с полем заголовка If-Modified-Since, но без If-None-Match этому серверу **следует** проверить выполнение условия If-Modified-Since в соответствии с параграфом 13.2 до применения метода.

1. Если выбранное представление изменено не позже указанного полем момента, условие считается не выполненным.
2. В иных случаях условие считается выполненным.

Серверу-источнику, проверяющему условие If-Modified-Since, **не следует** применять запрошенный метод, если условие не выполняется. Вместо этого **следует** создать отклик 304 (Not Modified), включающий лишь метаданные, которые полезны для идентификации или обновления кэшированного ранее отклика.

Требования к обработке кэшем полученного поля заголовка If-Modified-Since заданы в параграфе 4.3.2 [CACHING].

13.1.4. If-Unmodified-Since

Поле заголовка If-Unmodified-Since делает запрос условным по дате последнего изменения выбранного представления - условие считается выполненным, если эта дата не позже указанной в значении поля. Поле служит для той же задачи, что и If-Match, в случае отсутствия у агента пользователя тега сущности для представления.

```
If-Unmodified-Since = HTTP-date
```

Пример поля приведён ниже.

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

Получатель **должен** игнорировать поле If-Unmodified-Since, если в запросе имеется поле If-Match - условие If-Match является более точным, чем If-Unmodified-Since, и они применяются вместе лишь для совместимости со старыми посредниками, которые могут не реализовать If-Match. Получатель **должен** игнорировать поле If-Unmodified-Since, если в нем содержится значение, не являющееся действительным HTTP-date (включая значения в списке). Получатель **должен** игнорировать If-Unmodified-Since, если ресурс не имеет доступной даты изменения. Получатель **должен** интерпретировать значение временной мети в поле If-Unmodified-Since по часам сервера-источника.

If-Unmodified-Since чаще всего применяется в методах, меняющих состояние (например, POST, PUT, DELETE), для предотвращения случайной перезаписи при параллельной работе нескольких пользовательских агентов с одним ресурсом без тега сущности в представлениях (проблема потери обновлений). В общем случае поле может применяться с любым методом, предполагающим выбор или изменение представления, для прерывания запроса, если дата последнего изменения выбранного представления позже даты в поле If-Unmodified-Since.

При получении сервером-источником запроса, выбирающего представление, с полем заголовка If-Unmodified-Since, но без If-Match, сервер **должен** проверить выполнение условия If-Unmodified-Since в соответствии с параграфом 13.2.

1. Если последнее изменение выбранного представления произошло не позже указанного в поле времени, условие считается выполненным.
2. В иных случаях условие считается не выполненным.

Серверу-источнику, проверяющему условие If-Unmodified-Since, **недопустимо** применять запрошенный метод, если условие не выполняется. Вместо этого сервер **может** указать невыполнение условия для запроса откликом с кодом статуса 412 (Precondition Failed). Как вариант, сервер **может** передать отклик 2xx (Successful), если запрос был меняющей статус операцией, которая, по-видимому, уже применена к выбранному представлению (запрошенное пользовательским агентом изменение, уже внесено, но агент может не знать об этом, возможно, по причине потери предыдущего отклика или внесения эквивалентных изменений другим пользовательским агентом). Разрешение серверу-источнику передавать отклик об успехе, когда представляется, что запрос уже был применён, является более эффективным для многих случаев использования инструментов публикации, но связано с некоторым риском, если несколько пользовательских агентов делают очень похожие, но не согласованные запросы. В таких случаях серверу-источнику лучше быть строгим и передавать отклик 412 для каждого запроса с небезопасным методом, где условие не было выполнено.

Клиент **может** передать поле заголовка If-Unmodified-Since в запросе GET для указания того, что он предпочитает получить отклик 412 (Precondition Failed), если выбранное представление было изменено. Однако это полезно лишь в запросах с диапазоном (раздел 14) для завершения полученного ранее частичного представления, когда новое представление нежелательно. Поле If-Range (параграф 13.1.5) лучше подходит для запросов диапазона, если клиент предпочитает получит новое представление.

Кэш или посредник **может** игнорировать If-Unmodified-Since, поскольку его функциональная совместимость нужна лишь серверу-источнику.

13.1.5. If-Range

Поле заголовка If-Range обеспечивает специальный механизм условных запросов, подобный If-Match и If-Unmodified-Since, но предписывающий получателю игнорировать поле заголовка Range при несоответствии валидатора, что ведёт к отправке нового выбранного представления вместо отклика 412 (Precondition Failed).

Если клиент имеет частичную копию представления и хочет получить актуальную копию всего представления, он может использовать поле заголовка Range в условном запросе GET (с If-Unmodified-Since и/или If-Match). Однако, если условие не выполняется из-за изменения представления, клиенту придётся сделать второй запрос для получения текущего представления целиком.

Поле заголовка If-Range позволяет клиенту «сократить» второй запрос. Неформально оно говорит: «Если представление не изменилось, отправьте мне части, указанные в Range, в ином случае - представление целиком».

```
If-Range = entity-tag / HTTP-date
```

Действительный тег entity-tag можно отличить от действительной даты HTTP-date по DQUOTE в первых 3 символах.

Клиенту **недопустимо** генерировать поле заголовка If-Range в запросе без поля Range в заголовке. Сервер **должен** игнорировать поле If-Range в запросах без поля заголовка Range. Сервер-источник **должен** игнорировать поле If-Range в запросе к ресурсу, не поддерживающему запросы Range. Клиенту **недопустимо** генерировать поле заголовка If-Range, содержащее тег сущности, который помечен как слабый. Клиенту **недопустимо** генерировать поле заголовка If-Range, содержащее HTTP-date, если у него нет тега сущности для соответствующего представления и дата не является строгим валидатором в смысле, указанном в параграфе 8.8.2.2.

При получении сервером-источником поля заголовка If-Range header в запросе Range он **должен** проверить выполнение условий в соответствии с параграфом 13.2 до применения метода. Проверка поля If-Range с HTTP-date описана ниже.

1. Если предоставленный валидатор HTTP-date не является строгим в смысле, указанном в параграфе 8.8.2.2, условия считаются не выполненными.
2. Если предоставленный валидатор HTTP-date точно совпадает со значением поля Last-Modified для выбранного представления, условия считаются выполненными.
3. В иных случаях условия считаются не выполненными.

Проверка поля If-Range с тегом сущности описана ниже.

1. Если предоставленный валидатор entity-tag точно совпадает со значением поля Etag для выбранного представления при строгом сравнении (параграф 8.8.3.2), условия считаются выполненными.

2. В иных случаях условия считаются не выполненными.

Получатель If-Range **должен** игнорировать поле заголовка Range, если условие If-Range не выполняется. В ином случае получателю **следует** обрабатывать поле Range в соответствии с запросом. Отметим, что для If-Range проверяется точное совпадение, включая случай с указанием HTTP-date, что отличается от условия «не позже» в If-Unmodified-Since.

13.2. Оценка предварительных условий

13.2.1. Когда выполнять оценку

За исключением указанных ниже случаев кэш-получатель или сервер-источник **должен** оценивать предварительные условия в полученном запросе после успешного завершения обычных проверок запроса и непосредственно перед обработкой содержимого запроса (при его наличии) или выполнением действий, связанных с методом запроса. Сервер **должен** игнорировать все полученные предварительные условия, если его отклик на тот же запрос без этих условий имел бы код статуса, отличный от 2xx (Successful) и 412 (Precondition Failed), до обработки содержимого запроса. Иными словами, перенаправления и отказы, которые можно обнаружить до выполнения значительной обработки имеют приоритет перед оценкой предварительных условий.

Серверу, который не является источником для целевого ресурса и не может выступать в роли кэша для запросов к целевому ресурсу, **недопустимо** оценивать поля условий в заголовке запроса, заданные этой спецификацией, и он **должен** переслать их, если запрос пересылается, поскольку создающий запрос клиент предполагает, что условия оцениваются сервером, который может обеспечить текущее представление. Сервер **должен** игнорировать поля заголовка с условиями, определённые этой спецификацией, при получении запроса с методом, не предполагающим выбор или изменение текущего представления (например, CONNECT, OPTIONS, TRACE).

Отметим, что расширения протокола могут изменять обстоятельства оценки предварительных условий или последствия такой оценки. Например, директива кэширования immutable ([RFC8246]) предписывает кэшам не пересылать условные запросы при наличии в кэше свежего отклика.

Хотя поля заголовка запроса с условиями определены как пригодные для использования с методом HEAD (для сохранения в HEAD семантики метода GET), нет смысла передавать условные запросы HEAD, поскольку отклик об успехе имеет примерно такой же размер, как 304 (Not Modified), и более полезен, чем 412 (Precondition Failed).

13.2.2. Порядок применения предварительных условий

Когда в запросе имеется более одного поля заголовка с запросом, важен порядок оценки этих полей. На практике поля, заданные в этом документе, применяются последовательно в логическом порядке, поскольку условия, связанные с потерей обновлений, задают более строгие требования, нежели проверка кэша, проверенный кэш важнее частичного отклика, а теги сущности важнее валидаторов дат. Кэш-получатель или сервер-источник **должен** оценивать в запросе заданные этой спецификацией условия в приведённом ниже порядке.

1. Сервер-источник, получивший запрос с условием If-Match:
   - переходит к п. 3, если условие выполнено;
   - при невыполненном условии сервер передаёт отклик 412 (Precondition Failed), если не установлено, что меняющий состояние запрос уже был выполнен (см. параграф 13.1.1).
2. Сервер-источник, получивший запрос без If-Match, но с условием If-Unmodified-Since:
   - переходит к п. 3, если условие выполнено;
   - при невыполненном условии сервер передаёт отклик 412 (Precondition Failed), если не установлено, что меняющий состояние запрос уже был выполнен (см. параграф 13.1.4).
3. Для запросов с условием If-None-Match получатель:
   - переходит к п. 5, если условие выполнено;
   - при невыполненном условии передаёт отклик 304 (Not Modified) для методов GET и HEAD, 412 (Precondition Failed) - для других методов.
4. Для запросов GET или HEAD без If-None-Match, но с условием If-Modified-Since получатель:
   - переходит к п. 5, если условие выполнено;
   - при невыполненном условии передаёт отклик 304 (Not Modified).
5. Для запросов GET с полем Range и условием If-Range получатель:
   - передаёт отклик 206 (Partial Content), если условие выполнено значение Range применимо в выбранному представлению;
   - в иных случаях получатель игнорирует Range и передаёт отклик 200 (OK).
6. В остальных случаях получатель выполняет запрошенный метод и передаётся соответствующий отклик.

Любое расширение HTTP, задающее дополнительные поля заголовка с условием, должно определять порядок оценки этих полей по отношению к определённому здесь порядку и другим условиям, которые могут возникать на практике.

14. Запросы диапазонов

Клиенты часто сталкиваются с прерыванием передачи данных в результате отмены запросов или разрыва соединений. Если клиент сохранил частичное представление, желательно запросить оставшуюся часть вместо передачи всего представления. Устройства с ограниченным локальным хранилищем могут выиграть за счёт возможности запрашивать лишь часть крупного содержимого, например, одну страницу очень большого документа или фрагмент изображения.

Диапазоны запросов являются **необязательной** функцией HTTP, созданной так, что получатели, не поддерживающие её (совсем или для целевого ресурса) могут ответить как на обычный запрос GET без потери функциональной совместимости. Частичные отклики обозначаются отдельным кодом статуса, чтобы кэш без поддержки этой функции не принимал их за полные отклики.

14.1. Единицы диапазона

Данные представления могут быть поделены на поддиапазоны при наличии адресуемых структурных блоков, присущих кодированию содержимого или типу носителя. Например, границы октетов (байтов) задают структурные блоки для всех представлений данных, что позволяет указывать разделы данных диапазонами байтов с некоторым смещением от начала или конца данных.

Такое базовое обозначение единицы диапазона (range unit) применяется в полях заголовка Accept-Ranges (параграф 14.3) для анонсирования поддержки запросов диапазона, Range (параграф 14.2) для разграничения запрашиваемых частей представления и Content-Range (параграф 14.4) для описания передаваемой части.

```
range-unit       = token
```

Регистр символов в именах единиц диапазона не принимается во внимание, а имена должны регистрироваться в реестре HTTP Range Unit Registry, как указано в параграфе 16.5.1. Предполагается, что единицы диапазона являются расширяемыми, как описано в параграфе 16.5.

14.1.1. Задание диапазона

Диапазоны указываются в единицах, соответствующих спецификатору диапазона. Имя единицы диапазона указывает тип range-spec, разрешённый для спецификаторов. Базовая грамматика приведена ниже и предполагается, что каждая единица диапазона указывает, когда разрешено применять int-range, suffix-range и other-range. Запрос может указывать один или несколько диапазонов в рамках одного представления.

```
ranges-specifier = range-unit "=" OWS[1] range-set
range-set        = 1#range-spec
range-spec       = int-range
                 / suffix-range
                 / other-range
```

Форма int-range задаёт диапазон одним (начало диапазона) или двумя неотрицательными целыми числами. Единица диапазона определяет смысл спецификаторов (например, они могут указывать смещение от начала, пронумерованные части и т. п.). Форма int-range недействительна, если значение last-pos меньше first-pos.

```
int-range     = first-pos "-" [ last-pos ]
first-pos     = 1*DIGIT
last-pos      = 1*DIGIT
```

Форма suffix-range указывает диапазон суффиксом данных представления с неотрицательным значением максимального размера (в единицах диапазона). Иными словами, указывается N последних блоков представления.

```
suffix-range  = "-" suffix-length
suffix-length = 1*DIGIT
```

Для обеспечения расширяемости правило other-range предоставляет почти не ограниченную грамматику, позволяющую задавать спецификаторы для конкретных приложений или будущих единиц диапазонов.

```
other-range   = 1*( %x21-2B / %x2D-7E )
              ; 1*(VCHAR за исключением запятой)
```

Спецификатор ranges-specifier является недействительным при наличии в нем какого-либо недействительного или непонятного range-unit. Действительный ranges-specifier является «удовлетворительным» при наличии хотя бы одного удовлетворительного range-spec, как это определено указанным элементом range-unit. В остальных случаях ranges-specifier будет «неудовлетворительным».

14.1.2. Диапазоны байтов

Единица диапазона bytes служит для указания поддиапазонов последовательностей октетов содержимого представления. Каждый диапазон байтов указывается целыми числами смещения от начала (int-range) или конца (suffix-range) данных представления. В диапазонах байтов не применяется спецификатор other-range.

Значение first-pos в байтовом int-range указывает смещение первого байта диапазона, last-pos - смещение последнего (оба байта включаются в диапазон). Отсчёт смещений начинается с 0.

Если к данным представления применено кодирование содержимого, каждый диапазон байтов определяется для кодированной последовательности байтов, а не для последовательности, получаемой после декодирования.

Примеры указания диапазонов приведены ниже.

- Первые 500 байтов (смещение от 0 до 499, включительно)

```
bytes=0-499
```

- вторые 500 байтов (смещение от 500 до 999, включительно)

```
bytes=500-999
```

Клиент может ограничивать число байтов, запрашиваемых при неизвестном размере выбранного представления. Если значение last-pos не задано или выходит за размер данных представления, диапазон байтов рассматривается как остаток представления (т. е. сервер использует в качестве last-pos значение на 1 меньше текущего размера выбранного представления). Клиент может указать последние N байтов (N > 0) выбранного представления, используя suffix-range. Если выбранное представление меньше указанного suffix-length, используется все представление.

Ниже приведены другие примеры для представления размером 10000.

- последние 500 байтов (смещение от 9500 до 9999, включительно)

[1] В оригинале элемент OWS пропущен, см. https://www.rfc-editor.org/errata/eid7306. *Прим. перев.*

```
bytes=-500
```
или
```
bytes=9500-
```
- Только первый и последний байт (байты 0 и 9999)

```
bytes=0-0,-1
```
- По 1000 байтов из начала, середины и конца

```
bytes= 0-999, 4500-5499, -1000
```
- Другие действительные (но не канонические) спецификации вторых 500 байтов (500 - 999, включительно)

```
bytes=500-600,601-999
bytes=500-700,601-999
```
Для запроса GET байтовая спецификация range-spec удовлетворительна при выполнении одного из условий:

- указан int-range со значением first-pos меньше текущего размера выбранного представления;
- указан suffix-range с ненулевым значением suffix-length.

Для выбранного представления с нулевым размером единственной удовлетворительной формой range-spec в запросе GET является suffix-range с ненулевым значением suffix-length.

В синтаксисе byte-range значения first-pos, last-pos, suffix-length указываются десятичными числами (в октетах). Поскольку нет предопределённых ограничений на размер содержимого, получатели **должны** быть готовы принимать большие целые числа и предотвращать ошибки разбора из-за переполнения при целочисленных преобразованиях.

14.2. Range

Поле заголовка Range в запросе GET меняет семантику метода для запроса передачи лишь одного или нескольких субдиапазонов выбранного представления данных (параграф 8.1) вместо отправки всего представления.

```
Range = ranges-specifier
```
Сервер **может** игнорировать поле заголовка Range, однако серверы-источники и посредники должны поддерживать диапазоны байтов, когда это возможно, поскольку это позволяет поддерживать эффективное восстановление при частичных сбоях передачи и частичном извлечении больших представлений.

Сервер **должен** игнорировать поле заголовка Range, полученное с методом запроса, который непонятен или не предусматривает обработку диапазонов. Эта спецификация задаёт обработку диапазонов лишь для метода GET. Сервер **должен** игнорировать поле заголовка Range, содержащее непонятные единицы диапазона. Прокси **может** отбрасывать поле заголовка Range с непонятными единицами диапазона.

Сервер, поддерживающий запросы диапазонов, **может** игнорировать или отклонять поле заголовка Range с недействительным спецификатором диапазона (параграф 14.1.1), спецификатором, с двумя или более перекрывающимися диапазонами или набором из множества мелких диапазонов, указанных не в порядке возрастания, поскольку это указывает на взломанного (broken) клиента или преднамеренную DoS-атаку (параграф 17.15). Клиенту **не следует** запрашивать несколько диапазонов, поскольку это по сути менее эффективно, чем обработка и передача одного диапазона, охватывающего те же данные. Сервер, поддерживающий запросы диапазонов, **может** игнорировать поле заголовка Range, если в выбранном представлении нет содержимого (т. е. оно имеет размер 0).

Клиенту, запрашивающему несколько диапазонов, **следует** указывать их в порядке роста (как они были бы получены в полном представлении), если нет необходимости специально изменить порядок. Например, агенту пользователя, обрабатывающему большое представление с внутренним каталогом частей может потребоваться сначала запросить более поздние части, особенно если представление состоит из размещённых в обратном порядке страниц, которые агент хочет получать по одной.

Поле заголовка Range оценивается после проверки полей предварительных условий, определённых в параграфе 13.1, и лишь в том случае, когда при отсутствии поля заголовка Range был бы передан отклик 200 (OK). Иными словами, Range игнорируется, если условный запрос GET будет приводить к отклику 304 (Not Modified). Поле заголовка If-Range (параграф 13.1.5) может применяться как предварительное условие для применения поля Range.

Если все предварительные условия выполнены, сервер поддерживает поле заголовка Range для этого ресурса, полученное в поле Range значение имеет действительный спецификатор ranges-specifier с элементом range-unit, поддерживаемым для целевого ресурса, и ranges-specifier подходит для выбранного представления, серверу следует передать отклик 206 (Partial Content) с одним или несколькими частичными представлениями, соответствующими range-spec из запроса. Это не означает, что сервер передаст все запрошенные диапазоны. В некоторых случаях может быть возможна (или эффективна) передача начала лишь части запрошенных диапазонов с ожиданием от клиента повторного запроса для оставшихся частей, если они ещё нужны (см. параграф 15.3.7).

Если все предварительные условия выполнены, сервер поддерживает поле заголовка Range для этого ресурса, полученное в поле Range значение имеет действительный спецификатор ranges-specifier, но range-unit не поддерживается для целевого ресурса или ranges-specifier не подходит для выбранного представления, серверу следует передать отклик 416 (Range Not Satisfiable).

14.3. Accept-Ranges

Поле Accept-Ranges в отклике указывает, поддерживает и сервер восходящего направления запросы диапазонов для целевого ресурса.

```
Accept-Ranges     = acceptable-ranges
acceptable-ranges = 1#range-unit
```
Например, сервер, поддерживающий диапазоны байтов (параграф 14.1.2) может передать

```
Accept-Ranges: bytes
```
для индикации поддержки запросов с диапазонами байтов для целевого ресурса, чтобы клиент могут указывать их в будущих частичных запросах с тем же путём к ресурсу. Единицы диапазонов определены в параграфе 14.1.

Клиент **может** генерировать запросы диапазонов независимо от получения поля Accept-Ranges, которое служит лишь рекомендацией для повышения производительности и сокращения ненужных передач через сеть. Клиенту **недопустимо** предполагать, что полученное поле Accept-Ranges означает, что будущие запросы диапазона будут возвращать частичные отклики. Может измениться содержимое, сервер может поддерживать диапазоны лишь в течение некоторого времени или следующий запрос может быть обработан другим посредником.

Сервер, не поддерживающий запросы диапазонов для целевого ресурса, **может** передать

```
Accept-Ranges: none
```

для информирования клиента о том, что не следует пытаться запрашивать диапазон на том же пути к ресурсу. Единица диапазона none зарезервирована для этой цели.

Поле Accept-Ranges **можно** передавать в разделе трейлеров, но лучше помещать его в раздел заголовков, поскольку сведения из этого поля наиболее полезны для перезапуска передачи больших объёмов информации, для которых возникла ошибка где-то среди содержимого (до получения раздела трейлеров).

14.4. Content-Range

Поле заголовка Content-Range передаётся в отклике 206 (Partial Content) с одной частью для указания частичного диапазона, вложенного в сообщение содержимого представления, в каждой части многокомпонентного отклика 206 для указания частичного диапазона, вложенного в каждую часть (параграф 14.6), и в отклике 416 (Range Not Satisfiable) для предоставления сведений о выбранном представлении.

```
Content-Range       = range-unit SP
                      ( range-resp / unsatisfied-range )

range-resp          = incl-range "/" ( complete-length / "*" )
incl-range          = first-pos "-" last-pos
unsatisfied-range   = "*/" complete-length

complete-length     = 1*DIGIT
```

Если отклик 206 (Partial Content) содержит поле заголовка Content-Range с единицей диапазона (параграф 14.1), которую получатель не понимает, получателю **недопустимо** пытаться рекомбинировать отклик с сохраненным представлением. Прокси, получающему такое сообщение, **следует** переслать его в нисходящем направлении.

Поле Content-Range может передаваться как модификатор запроса для частичного PUT (см. параграф 14.5) на основе частных соглашений между клиентом и сервером-источником. Сервер **должен** игнорировать поле заголовка Content-Range, полученное в запросе с методом, для которого поддержка Content-Range не задана.

Для байтовых диапазонов отправителю **следует** указывать полный размер (complete-length) представления, из которого извлечён диапазон, за исключением случаев, когда определить полный размер трудно или невозможно. Символ * вместо complete-length указывает, что размер представления не был известен в момент создания поля. Ниже приведён пример пример, где полный размер представления известен отправителю и составляет 1234 байта.

```
Content-Range: bytes 42-1233/1234
```

В следующем примере отправитель не знает полного размера представления.

```
Content-Range: bytes 42-1233/*
```

Значение поля Content-Range считается недействительным, если оно содержит range-resp с last-pos меньше first-pos или complete-length меньше или совпадает с last-pos. Получателю недействительного Content-Range **недопустимо** пытаться рекомбинировать полученное содержимое с сохраненным представлением. .

Серверу, генерирующему отклик 416 (Range Not Satisfiable) для запроса byte-range, **следует** передавать поле заголовка Content-Range со значением unsatisfied-range, как показано ниже.

```
Content-Range: bytes */1234
```

Элемент complete-length в отклике 416 указывает текущий размер выбранного представления.

Поле заголовка Content-Range не имеет смысла для кодов статуса, не описывающих его семантику явно. В данной спецификации Content-Range имеет смысл только в откликах 206 (Partial Content) и 416 (Range Not Satisfiable).

Ниже приведены примеры значений Content-Range при полном размере выбранного содержимого 1234 байта.

- Первые 500 байтов

  ```
  Content-Range: bytes 0-499/1234
  ```

- Вторые 500 байтов

  ```
  Content-Range: bytes 500-999/1234
  ```

- Все, кроме первых 500 байтов

  ```
  Content-Range: bytes 500-1233/1234
  ```

- Последние 500 байтов

  ```
  Content-Range: bytes 734-1233/1234
  ```

14.5. Частичный запрос PUT

Некоторые серверы-источники поддерживают PUT для частичного представления, когда пользователь передаёт в запросе поле заголовка Content-Range (параграф 14.4), хотя такая поддержка противоречива и зависит от частного соглашения с агентом пользователя. В общем случае запрашивается замена части целевого ресурса содержимым запроса по смещению и размеру, указанным в поле Content-Range, где смещение указано относительно текущего выбранного представления.

Серверу-источнику **следует** отвечать кодом 400 (Bad Request) при получении Content-Range в запросе PUT для целевого ресурса, который не поддерживает частичные запросы PUT. Частичные запросы PUT не совместимы с исходным определением PUT и могут приводить к полной замене текущего представления содержимым запроса.

Частичные обновления ресурса возможны также при нацеливании на отдельно идентифицированный ресурс с состоянием, которое имеет пересечение или расширяет часть большего ресурса, или при использовании другого метода, который специально определён для частичных обновлений (например, метод PATCH из [RFC5789]).

14.6. Тип носителя multipart/byteranges

Когда сообщение с откликом 206 (Partial Content) включает содержимое из нескольких диапазонов, оно передаётся как части тела многокомпонентного сообщения ([RFC2046], параграф 5.1) с типом носителя multipart/byteranges, который включает одну или несколько частей тела сообщения, каждая из которых имеет свои поля Content-Type и Content-Range. Обязательный параметр границы задаёт строку, отделяющую одну часть от другой.

1. Перед первой граничной строкой в теле сообщения могут указываться дополнительные символы CRLF.
2. [RFC2046] разрешает заключать граничную строку в кавычки, однако некоторые реализации некорректно обрабатывают такие строки.
3. Многие клиенты и серверы разработаны на основе предварительной спецификации byteranges, использовавшей тип носителя multipart/x-byteranges, который почти (но не совсем) совместим с byteranges.

Несмотря на имя, тип носителя multipart/byteranges предназначен не только для байтовых диапазонов. Ниже приведён пример с использованием единицы диапазона exampleunit.

```
HTTP/1.1 206 Partial Content
Date: Tue, 14 Nov 1995 06:25:24 GMT
Last-Modified: Tue, 14 July 04:58:08 GMT
Content-Length: 2331785
Content-Type: multipart/byteranges; boundary=THIS_STRING_SEPARATES

--THIS_STRING_SEPARATES
Content-Type: video/example
Content-Range: exampleunit 1.2-4.3/25

...первый диапазон...
--THIS_STRING_SEPARATES
Content-Type: video/example
Content-Range: exampleunit 11.2-14.3/25

...второй диапазон
--THIS_STRING_SEPARATES--
```

Приведённые ниже сведения служат регистрационной формой типа носителя multipart/byteranges.

```
Type name:  multipart
Subtype name:  byteranges
Required parameters:  boundary
Optional parameters:  N/A
Encoding considerations:  only "7bit", "8bit", or "binary" are permitted
Security considerations:  see Section 17
Interoperability considerations:  N/A
Published specification:  RFC 9110 (see Section 14.6)
Applications that use this media type:  HTTP components supporting
   multiple ranges in a single request
Fragment identifier considerations:  N/A
Additional information:  Deprecated alias names for this type:  N/A
                         Magic number(s):  N/A
                         File extension(s):  N/A
                         Macintosh file type code(s):  N/A
Person and email address to contact for further information:  See Aut
   hors' Addresses section.
Intended usage:  COMMON
Restrictions on usage:  N/A
Author:  See Authors' Addresses section.
Change controller:  IESG
```

15. Коды статуса

Код статуса в отклике является 3-значным десятичным числом, описывающим результат запроса и семантику отклика, включая сведения об успешности запроса и характере вложенного содержимого (при наличии). Действительные коды статуса имеют значения от 100 до 599, включительно.

Первая цифра кода статуса указывает класс отклика, две оставшиеся не задают категории.

- 1xx (Informational) - запрос получен, процесс продолжается.
- 2xx (Successful) - запрос получен, понят и воспринят.
- 3xx (Redirection) - нужны дополнительные действия для завершения запроса.
- 4xx (Client Error) - запрос имеет некорректный синтаксис и не может быть выполнен.
- 5xx (Server Error) - отказ сервера при выполнении запроса, представляющегося действительным.

Коды статуса HTTP могут расширяться. Клиент не обязан понимать смысл всех зарегистрированных кодов статуса, хотя такое понимание желательно. Однако клиент **должен** понимать класс любого кода, указанный первой цифрой и считать нераспознанные коды эквивалентом кода x00 для этого класса. Например, если клиент получает непонятный код 471, по первой цифре он может понять, что в запросе была какая-то ошибка и считать этот код эквивалентом кода 400 (Bad Request). Отклик обычно включает представление, разъясняющее статус.

Значения за пределами диапазона 100 - 599 недействительны. Реализации часто используют такие значения (600- 999) для внутренней передачи статуса, не связанного с HTTP (например, ошибки библиотеки). Клиенту, получившему отклик с таким кодом **следует** обрабатывать его как отклик с кодом 5xx (Server Error).

На запрос может возвращаться несколько связанных откликов - промежуточные (interim, не окончательный) отклики с кодами 1xx, за которыми следует один окончательный (final) отклик с кодом статуса из другого диапазона.

15.1. Обзор кодов статуса

Ниже описаны коды статуса, определённые этой спецификацией. Приведённые здесь фразы причин в кодах являются лишь рекомендациями и могут быть заменены локальными эквивалентами и даже опущены без ущерба для протокола.

Отклики с кодами, которые определены как эвристически кэшируемые (например, 200, 203, 204, 206, 300, 301, 308, 404, 405, 410, 414, 501 в данной спецификации) могут повторно применяться кэшем с эвристическим сроком действия, если иное не задано в определении метода или в элементах явного управления кэшем [CACHING]. Остальные коды статуса не являются эвристически кэшируемыми.

Для использования в HTTP могут быть заданы дополнительные коды статуса, выходящие за рамки этой спецификации. Все коды статуса должны регистрироваться в реестре Hypertext Transfer Protocol (HTTP) Status Code Registry, как описано в параграфе 16.2.

15.2. Информационные коды (1xx)

Коды статуса 1xx (Informational) указывают промежуточный отклик для передачи сведений о состоянии соединения или обработки запроса до завершения запрошенного действия и передачи окончательного отклика. Поскольку в HTTP/1.0 коды статуса 1xx не определены, серверу **недопустимо** передавать отклики 1xx клиенту HTTP/1.0.

Отклик 1xx завершается концом раздела заголовков и не может включать содержимого или трейлеров.

Клиент **должен** быть способен обработать один или несколько откликов 1xx до получения финального отклика. Агент пользователя **может** игнорировать неожиданные отклики 1xx.

Прокси **должен** пересылать отклики 1xx, если только он сам не запросил его генерацию. Например, если прокси добавляет поле заголовка Expect: 100-continue при пересылке запроса, ему не следует пересылать соответствующие отклики 100 (Continue).

15.2.1. 100 Continue

Код статуса 100 (Continue) указывает, что начальная часть запроса получена и ещё не отвергнута сервером. Сервер намерен передать финальный отклик после приёма и обработки запроса полностью.

Если запрос включает поле заголовка Expect с ожиданием 100-continue, код отклика 100 указывает, что сервер желает получить содержимое запроса, как указано в параграфе 10.1.1. Клиент должен продолжить отправку запроса и отбросить отклик 100. Если в запросе нет поля Expect с ожиданием 100-continue, клиент может просто отбросить этот промежуточный отклик.

15.2.2. 101 Switching Protocols

Код 101 (Switching Protocols) указывает, что сервер понимает и готов выполнить запрос клиента через поле заголовка Upgrade (параграф 7.8) для смены прикладного протокола, используемого в этом соединении. Сервер **должен** генерировать в отклике поле заголовка Upgrade, указывающие протокол(ы), который будет действовать после этого отклика. Предполагается, что сервер будет соглашаться на смену протокола лишь в том случае получения выигрыша. Например, переход на более новую версию HTTP может давать выигрыш, а переход на синхронный протокол, работающий в реальном масштабе времени, может быть выгоден при доставке ресурсов, имеющих такие свойства.

15.3. Успешное выполнение (2xx)

Класс кодов 2xx (Successful) показывает, что запрос клиента успешно получен, понят и воспринят.

15.3.1. 200 OK

Код статуса 200 (OK) указывает, что запрос был успешно выполнен. Содержимое отклика 200 зависит от метода запроса. Для заданных этой спецификацией методов предполагаемый смысл содержимого указан в таблице 6.

Таблица 6.

Метод запроса	***Представление содержимого отклика***
GET	Целевой ресурс
HEAD	Целевой ресурс, как для GET, но без передачи данных представления
POST	Статус или результаты действия
PUT, DELETE	Статус действия
OPTIONS	Коммуникационные опции для целевого ресурса
TRACE	Запросное сообщение, полученное сервером, возвращающим трассировку

Кроме случая CONNECT, предполагается, что отклик 200 будет включать содержимое сообщения, если в обрамлении того не указан явно нулевой размер содержимого. Если тот или иной аспект запроса указывает предпочтительность отсутствия содержимого при успехе, сервер-источник должен передавать взамен отклик 204 (No Content). Для CONNECT отклик не имеет содержимого, поскольку успешным результатом является туннель, который начинается сразу после раздела заголовков отклика 200.

Отклик 200 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

В откликах 200 на GET или HEAD серверу-источнику **следует** передавать любые доступные поля валидаторов (параграф 8.8) для выбранного представления, но предпочтительны строгий тег сущности и дата Last-Modified. В откликах 200 для методов, меняющих состояние, любые поля валидаторов (параграф 8.8) в отклике передают текущие

валидаторы для нового представления, созданного в результате успешного применения семантики запроса. С методом PUT (параграф 9.3.4) связаны дополнительные требования, которые могут исключать отправку валидаторов.

15.3.2. 201 Created

Код 201 (Created) указывает, что запрос был выполнен и в результате создан один или несколько новых ресурсов. Основной ресурс, созданный запросом, указывается полем Location в отклике или URI цели, если Location не получено.

Отклик с кодом 201 обычно содержит описание и ссылки на созданные ресурсы. Любые поля валидаторов (параграф 8.8) в отклике содержат текущие валидаторы нового представления, созданного запросом. Отметим, что с методом PUT (параграф 9.3.4) связаны дополнительные требования, которые могут исключать отправку таких валидаторов.

15.3.3. 202 Accepted

Код 202 (Accepted) указывает, что запрос был принят для обработки, но та не была завершена. Запрос может быть выполнен или не выполнен, поскольку при обработке в любой момент возможен запрет. В HTTP нет возможности повторно передать код статуса при асинхронной операции.

Отклик 202 намеренно неполон и его цель заключается в том, чтобы позволить серверу воспринять запрос на выполнение какого-либо иного процесса (возможно, пакетной обработки, выполняемой 1 раз в день) не требуя сохранять соединение с агентом пользователя до момента завершения этого процесса. Передаваемое с этим откликом представление должно описывать текущий статус запроса и указывать (или включать в себя) средство отслеживания состояния, которое может предоставить пользователю оценку времени завершения выполнения запроса.

15.3.4. 203 Non-Authoritative Information

Код 203 (Non-Authoritative Information) указывает, что запрос был успешным, но вложенное содержимое изменено по сравнению с исходным откликом 200 (OK) от сервера-источника преобразующим прокси (параграф 7.7). Этот код позволяет прокси уведомить получателей при выполнении преобразований, поскольку это может влиять на последующие решения, связанные с содержимым. Например, будущие запросы проверки кэша для содержимого могут быть применимы лишь для того же пути запроса (через те же прокси).

Отклик 203 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.3.5. 204 No Content

Код 204 (No Content) указывает, что сервер выполнил запрос и для передачи в отклике нет дополнительного содержимого. Метаданные в полях заголовка отклика относятся к целевому ресурсу и выбранному представлению после применения запрошенного действия. Например, код статуса 204 в отклике с полем ETag на запрос PUT говорит о выполнении метода PUT и указании в поле ETag тега сущности для нового представления этого целевого ресурса.

Отклик 204 позволяет серверу сообщить об успешном применении действия к целевому ресурсу. При этом подразумевается, что агенту пользователя не требуется выходить из текущего просмотра документа (если он есть). Сервер предполагает, что пользовательский агент обеспечит пользователю индикацию успеха в соответствии со своим интерфейсом и применит новые или обновлённые метаданные из отклика к активному представлению. Например, код 204 широко применяется с действием сохранения (save), в результате чего сохранённый документ остаётся доступным пользователю для редактирования. Этот код также часто применяется в интерфейсах, предполагающих преобладание автоматической передачи данных, таких как распределенные системы контроля версий.

Отклик 204 завершается в конце раздела заголовков и не включает содержимого и трейлеров.

Отклик 204 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.3.6. 205 Reset Content

Код 205 (Reset Content) указывает, что сервер выполнил запрос и хочет, чтобы агент пользователя сбросил «представление документа», вызвавшее передачу запроса, в исходное состояние, полученное от сервера-источника. Этот код предназначен для поддержки распространённого случая ввода данных, где пользователь получает содержимое, поддерживающее ввод данных (форма, блокнот, холст и т. п.), вводит или изменяет данные в этом пространстве, отправляет введённые данные в запросе, а затем механизм ввода данных сбрасывается для ввода следующей записи, чтобы пользователь мог продолжить работу по вводу. Поскольку отклик 205 подразумевает отсутствие дополнительного содержимого, серверу **недопустимо** генерировать содержимое для откликов 205.

15.3.7. 206 Partial Content

Код статуса 206 (Partial Content) указывает, что сервер успешно выполнил запрос диапазона для целевого ресурса, передавая одну или несколько частей выбранного представления. Сервер, поддерживающий запросы диапазона (раздел 14), обычно будет пытаться удовлетворить всем запрошенным диапазонам, поскольку отправка меньшего объёма данных, скорей всего, приведёт к другому запросу клиента на оставшуюся часть. Однако сервер может отправить лишь часть запрошенных данных по своим внутренним причинам, таким как временная недоступность, эффективность кэша, распределение нагрузки и т. п. Поскольку отклик 206 описывает себя, клиент все равно способен понять, что выполнена лишь часть запроса диапазона.

Клиент **должен** проверить в отклике 206 поля Content-Type и Content-Range для определения вложенных частей и необходимости дополнительных запросов.

Сервер, генерирующий отклик 206, **должен** в дополнение к полям, указанным в последующих параграфах, создавать поля Date, Cache-Control, ETag, Expires, Content-Location, Vary, если бы они были в отклике 200 (OK) на тот же запрос.

Поле заголовка Content-Length в отклике 206 указывает число октетов содержимого в сообщении, которое обычно отличается от полного размера выбранного представления. Каждое поле Content-Range включает сведения о полном размере выбранного представления.

Отправителю, создающему отклик 206 на запрос с полем заголовка If-Range, **не следует** генерировать поля заголовка сверх требуемых, поскольку у клиента уже есть предыдущий отклик с этими полями. В иных случаях отправитель **должен** генерировать все поля заголовка представления, которые были бы в отклике 200 (OK) на тот же запрос.

Отклик 206 является эвристически кэшируемым, т. е. кэшируется, если иное не указано явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.3.7.1. Одна часть

Если передаётся одна часть, сервер, создающий отклик 206, **должен** генерировать поле заголовка Content-Range, описывающее, в каком диапазоне заключено выбранное представление, и содержимое этого диапазона. Например,

```
HTTP/1.1 206 Partial Content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Range: bytes 21010-47021/47022
Content-Length: 26012
Content-Type: image/gif

... 26012 байтов данных части изображения ...
```

15.3.7.2. Несколько частей

При передаче нескольких частей сервер, создающий отклик 206, **должен** генерировать содержимое типа multipart/byteranges, как указано в параграфе 14.6, и поле заголовка Content-Type, указывающее тип носителя multipart/byteranges и требуемый параметр границы. Для предотвращения путаницы с откликами с одной частью серверу **недопустимо** генерировать поле заголовка Content-Range в разделе заголовков HTTP отклика с несколькими частями (это поле передаётся в каждой части).

В области заголовка каждой части тела сервер **должен** создать поле Content-Range, соответствующее диапазону, включённому в эту часть тела. Если выбранное представление имело бы поле заголовка Content-Type в отклике 200 (OK), серверу **следует** использовать то же поле заголовка Content-Type в области заголовка каждой части. Например,

```
HTTP/1.1 206 Partial Content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Length: 1741
Content-Type: multipart/byteranges; boundary=THIS_STRING_SEPARATES

--THIS_STRING_SEPARATES
Content-Type: application/pdf
Content-Range: bytes 500-999/8000

... первый диапазон ...
--THIS_STRING_SEPARATES
Content-Type: application/pdf
Content-Range: bytes 7000-7999/8000

... второй диапазон
--THIS_STRING_SEPARATES--
```

При запросе нескольких диапазонов сервер **может** объединить диапазоны, которые перекрываются или разделены зазором меньшим, чем издержки на передачу нескольких частей, независимо от порядка указания соответствующих range-spec в поле заголовка Range. Поскольку типичные издержки на разделение частей в multipart/byteranges составляют около 80 байтов, в зависимости от типа носителя для выбранного представления и выбранного граничного параметра передача множества мелких разделённых частей может оказаться менее эффективной, нежели передача всего выбранного представления.

Серверу **недопустимо** создавать многокомпонентный отклик на запрос одного диапазона, поскольку клиент, не запрашивающий несколько частей, может не поддерживать многокомпонентные отклики. Однако сервер **может** генерировать отклик multipart/byteranges с единственной частью в теле, если запрошено несколько диапазонов и лишь один был сочтён удовлетворительным или лишь один остался после слияния. Клиенту, не способному обрабатывать отклик multipart/byteranges, **недопустимо** генерировать запросы с несколькими диапазонами.

Серверу, создающему многокомпонентный отклик, **следует** передавать его части в том же порядке, в каком соответствующие range-spec указаны в принятом поле заголовка Range, исключая диапазоны, сочтённые неудовлетворительными или включённые в другие диапазоны. Получивший многокомпонентный отклик клиент **должен** проверить поле заголовка Content-Range в каждой части тела сообщения для определения содержащейся в нем части представления, не полагаясь на то, что они переданы для тех же диапазонов и в том же порядке, как было в запросе.

15.3.7.3. Объединение частей

Отклик может передавать лишь поддиапазон представления, если соединение закрылось преждевременно или в запросе была спецификация Range. После нескольких таких передач у клиента может быть несколько принятых диапазонов одного представления. Эти диапазоны можно безопасно объединить лишь в случае наличия у каждого из них одного и того же строгого валидатора (параграф 8.8.1). Клиент, получивший несколько частичных откликов на запросы GET для целевого ресурса, **может** объединить отклики в один непрерывный диапазон, если они имеют общий строгий валидатор. Если последним получен неполный отклик 200 (OK), поля заголовка из него применяются для объединённого отклика, заменяя соответствующие значения сохранённых частичных откликов. Если последним получен отклик 206 (Partial Content) и хотя бы один из соответствующих сохранённых откликов имеет код 200 (OK), поля заголовка комбинированного отклика являются полями последнего отклика 200. Если все соответствующие сохранённые отклики имеют код 206, в комбинированном отклике используются последние поля заголовка, но клиент **должен** использовать другие поля заголовка (помимо взятых из Content-Range), предоставленные в новом отклике, для замены всех экземпляров соответствующих полей в сохранённом отклике.

Содержимым комбинированного отклика является объединение частичных диапазонов содержимого из нового отклика и всех соответствующих сохранённых откликов. Если объединение образует весь диапазон представления, клиент **должен** обработать комбинированный отклик, как будто он является полным откликом 200 (OK), включая поле заголовка Content-Length, отражающее полный размер. В ином случае клиент **должен** обработать набор непрерывных диапазонов как неполный отклик 200 (OK), если комбинированный отклик является префиксом представления, один отклик 206 (Partial Content) с содержимым multipart/byteranges или несколько откликов 206 (Partial Content), каждый из которых содержит непрерывный диапазон, указанный поле заголовка Content-Range.

15.4. Перенаправление (3xx)

Класс кодов статуса 3xx (Redirection) указывает необходимость дополнительных действий со стороны пользовательского агента для выполнения запроса. Имеется несколько типов перенаправления.

1. Указание на доступность ресурса по другому URI, указанному в поле заголовка Location с кодом статуса 301 (Moved Permanently), 302 (Found), 307 (Temporary Redirect) или 308 (Permanent Redirect).
2. Предложение выбрать из числа подходящих ресурсов, обеспечивающих представление данного ресурса в отклике с кодом 300 (Multiple Choices).
3. Перенаправление на ресурс, указанный полем заголовка Location, который может предоставить непрямой отклик на запрос, в отклике с кодом 303 (See Other).
4. Перенаправление на сохранённый ранее результат с отклике с кодом 304 (Not Modified).

Примечание. В HTTP/1.0 коды статуса 301 (Moved Permanently) и 302 (Found) исходно были определены как сохраняющие метод ([HTTP/1.0], параграф 9.3) для соответствия их реализации в CERN. Отклик 303 (See Other) был определён для перенаправления со сменой метода на GET. Однако ранние пользовательские агенты разошлись в части перенаправления запросов POST как POST (соответствует данной спецификации) или GET (более безопасный вариант при перенаправлении на другой сайт). В конце концов практика свелась к смене метода на GET. Отклики 307 (Temporary Redirect) и 308 (Permanent Redirect) [RFC7538] были добавлены позднее для однозначного указания перенаправлений с сохранением метода, а коды 301 и 302 были скорректированы, чтобы разрешить перенаправление запросов POST на GET.

Если представлено поле заголовка Location (параграф 10.2.2), агент пользователя **может** автоматически перенаправить запрос на URI из значения поля Location даже в случаях, когда конкретный код статуса не понятен. Автоматическое перенаправление должно выполняться осторожно для методов, о безопасности которых не известно, как указано в параграфе 9.2.1, поскольку пользователь может не захотеть перенаправлять небезопасный запрос.

При автоматическом следовании перенаправленному запросу агенту пользователя **следует** повторно передать исходный запрос, внеся в него указанные ниже изменения.

1. URI цели заменяется на URI из поля Location в отклике о перенаправлении после его преобразования по отношению к URI цели в исходном запросе.
2. Удаляются поля заголовка, автоматически созданные реализацией, с заменой обновлёнными значениями, подходящими для нового запроса:
    1. специфичные для соединения поля заголовка (см. параграф 7.6.1);
    2. поля заголовка, специфичные для настройки прокси у клиента, включая Proxy-Authorization и др.;
    3. специфичные для источника поля заголовка (при наличии), включая Host и др.;
    4. проверка полей заголовка, добавленных кэшем реализации (например, If-None-Match, If-Modified-Since);
    5. специфичные для ресурса поля заголовка, включая Referer, Origin, Authorization, Cookie и др.
3. Рассматривается удаление полей заголовка, которые не были автоматически созданы реализацией (т. е добавлены в запрос вызывающим контекстом), если это связано с безопасностью. Это включает поля Authorization, Cookie и др.
4. Изменяется метод запроса в соответствии с семантикой кода перенаправления, если это уместно.
5. При замене метода на GET или HEAD удаляются специфичные для содержимого поля заголовка, включая Content-Encoding, Content-Language, Content-Location, Content-Type, Content-Length, Digest, Last-Modified и др.

Клиенту **следует** детектировать циклическое перенаправления (петля) и вмешиваться при их обнаружении.

Примечание. Ранняя версия этой спецификации рекомендовала не более 5 перенаправлений ([RFC2068], параграф 10.3). Разработчики содержимого должны учитывать, что некоторые клиенты могут реализовать такое ограничение.

15.4.1. 300 Multiple Choices

Код 300 (Multiple Choices) говорит, что целевой ресурс имеет более одного представления и у каждого из них есть свой более конкретный идентификатор, а сведения о вариантах предоставляются так, чтобы пользователь (или его агент) мог выбрать предпочтительное представление, перенаправляя свой запрос на один или несколько таких идентификаторов. Иными словами, сервер желает, чтобы агент пользователя начал реактивное согласование для выбора наиболее подходящих представлений с учётом своих потребностей (раздел 12).

Если у сервера есть предпочтительный выбор, ему **следует** генерировать поле Location с URI предпочтительного выбора. Агент пользователя **может** использовать значение поля Location для автоматического перенаправления.

Для запросов, отличных от HEAD, серверу **следует** создавать отклик 300 со списком метаданных представления и ссылками URI, из которых пользователь или его агент выбирает наиболее предпочтительный. Пользовательский агент **может** сделать выбор из этого списка автоматически, если он понимает представленный тип носителя. Конкретный формат для автоматического выбора эта спецификация не задаёт, поскольку HTTP пытается сохранить ортогональность к определению содержимого. На практике представление обеспечивается в каком-либо легко

разбираемом формате, который считается приемлемым для агента пользователя, как определено общим решением, согласованием содержимого или неком общепринятым форматом гипертекста.

Отклик 300 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

> Примечание. В исходом предложении для кода 300 поле заголовка URI определялось как предоставляющее список вариантов представления, что делало его применимым для откликов 200, 300, 406 и передавать в откликах для метода HEAD. Однако малый объем внедрения и разногласия в части синтаксиса привели к тому, что URI и Alternates (последующее предложение) были исключены из спецификации. Можно передать список как значение поля заголовка Link [RFC8288], элементы которого имеют отношение alternate, однако внедрение этого списка порождает проблему «курицы и яйца» (chicken-and-egg).

15.4.2. 301 Moved Permanently

Код 301 (Moved Permanently) показывает, что целевому ресурсу назначено новое постоянное значение URI и во всех будущих ссылках на этот ресурс должен применяться один из вложенных URI. Сервер предполагает, что агент пользователя с возможностью редактирования ссылок может на постоянной основе заменить ссылки на URI цели одной из новых ссылок, переданных сервером. Однако это предложение обычно игнорируется, за исключением случая, когда агент пользователя активно занимается редактированием ссылок (например, при создании содержимого), соединение защищено и сервер-источник является доверенным для редактируемого содержимого.

Серверу **следует** создавать в отклике поле заголовка Location с предпочтительной ссылкой URI для нового постоянного URI. Агент пользователя **может** применять значение поля Location для автоматического перенаправления. Содержимое отклика сервера обычно включает короткое гипертекстовое примечание с гиперссылкой на новые URI.

> Примечание. По историческим причинам агент пользователя **может** поменять метод POST на GET для последующего запроса. Если это нежелательно, можно использовать отклик с кодом 308 (Permanent Redirect).

Отклик 301 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.4.3. 302 Found

Код 302 (Found) указывает, что целевой ресурс временно размещён по другому URI. Поскольку перенаправление может быть изменено, клиент должен продолжать использование URI в будущих запросах.

Серверу **следует** создавать в отклике поле заголовка Location с другой ссылкой URI. Агент пользователя **может** применять значение поля Location для автоматического перенаправления. Содержимое отклика сервера обычно включает короткое гипертекстовое примечание с гиперссылкой на новые URI.

> Примечание. По историческим причинам агент пользователя **может** поменять метод POST на GET для последующего запроса. Если это нежелательно, можно использовать отклик с кодом 307 (Temporary Redirect).

15.4.4. 303 See Other

Код 303 (See Other) говорит, что сервер перенаправляет агент пользователя на другой ресурс, указанный URI в поле заголовка Location, который должен дать непрямой отклик на исходный запрос. Агент пользователя может выполнить запрос на извлечение по этому URI (GET или HEAD для HTTP), который тоже может быть перенаправлен, и представить конечный результат как ответ на исходный запрос. Отметим, что URI в поле Location не считается эквивалентом URI цели.

Этот код статуса применим для любого метода HTTP. В основном он служит для того, чтобы вывод действия POST позволял перенаправить агент пользователя на другой ресурс, поскольку при этом сведения, соответствующие отклику на POST, представляются как ресурс, который можно идентифицировать, поместить с закладки и кэшировать.

Отклик 303 на запрос GET указывает, что у сервера-источника нет представления, которое он мог бы передать по протоколу HTTP. Однако значение поля Location указывает ресурс, описывающий целевой ресурс, и запрос извлечения для этого ресурса может дать представление, которое будет полезно получателям, не подразумевая, что оно является представлением исходного целевого ресурса. Отметим, что ответы на вопросы, что может быть представлено, какие представления будут адекватными и что может быть полезным описанием, выходят за рамки HTTP.

За исключением откликов на запросы HEAD, представление отклика 303 должно включать краткое гипертекстовое замечание с гиперссылкой на URI, представленный в поле заголовка Location.

15.4.5. 304 Not Modified

Код статуса 304 (нет изменений) указывает, что был получен условный запрос GET или HEAD, который привёл бы к отклику 200 (OK), если бы проверка условия запроса не давала значение false. Иными словами, серверу нет необходимости передавать представление целевого ресурса, поскольку запрос указывает, что у клиента, сделавшего запрос, уже есть действительное представление. Поэтому сервер перенаправляет клиента на использование сохранённого представления, как будто это содержимое отклика с кодом 200 (OK).

Сервер, создающий отклик 304, **должен** генерировать какие-либо из указанных ниже полей, как при отправке отклика 200 (OK) на тот же запрос.

- Content-Location, Date, ETag, Vary.
- Cache-Control и Expires (см. [CACHING])

Поскольку целью отклика 304 является минимизация передачи информации при наличии у клиента одного или нескольких кэшированных представлений, отправителю **не следует** генерировать метаданные представления сверх указанных выше, если только эти метаданные не служат для управления обновлением кэша (например, может быть полезно поле Last-Modified, если в отклике нет поля ETag).

Требования к кэшу, получившему отклик 304, определены в параграфе 4.3.4 [CACHING]. Если условный запрос был создан исходящим клиентом, таким как агент пользователя со своим кэшем, передающий условный запрос GET общему прокси, этому прокси **следует** переслать отклик 304 клиенту.

Отклик 304 завершается концом раздела заголовков и не может включать содержимое или трейлеры.

15.4.6. 305 Use Proxy

Код 305 (Use Proxy) был задан в предыдущей версии спецификации и сейчас отменен (Приложение B к [RFC7231]).

15.4.7. 306 (Unused)

Код 306 был задан в предыдущей версии спецификации, больше не используется и сейчас переведён в резерв.

15.4.8. 307 Temporary Redirect

Код 307 (Temporary Redirect) указывает, что целевой ресурс временно находится на другом URI и агенту пользователя **недопустимо** менять метод запроса при автоматическом перенаправлении на этот URI. Поскольку перенаправление может измениться со временем, клиент должен продолжать использовать в будущих запросах исходный URI цели.

Серверу **следует** генерировать в отклике поле заголовка Location со ссылкой для другого URI. Агент пользователя **может** применять значение поля Location для автоматического перенаправления. Содержимое отклика сервера обычно включает краткое гипертекстовое замечание с гиперссылкой на URI.

15.4.9. 308 Permanent Redirect

Код 308 (Permanent Redirect) говорит, что целевому ресурсу назначен новый постоянный URI и в будущих ссылках на этот ресурс должен применяться один из вложенных URI. Агенту пользователя **недопустимо** менять метод запроса при автоматическом перенаправлении на этот URI[1]. Сервер предполагает, что агент пользователя с возможностью редактирования ссылок может на постоянной основе заменить URI цели новыми ссылками, переданными сервером. Однако это предложение обычно игнорируется, за исключением случая, когда агент пользователя активно занимается редактированием ссылок (например, при создании содержимого), соединение защищено и сервер-источник является доверенным для редактируемого содержимого.

Серверу **следует** создавать в отклике поле заголовка Location с предпочтительной ссылкой URI для нового постоянного URI. Агент пользователя **может** применять значение поля Location для автоматического перенаправления. Содержимое отклика сервера обычно включает короткое гипертекстовое примечание с гиперссылкой на новые URI.

Отклик 308 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

> Примечание. Этот код значительно моложе (июнь 2014 г.) своих собратьев и его могут распознавать не все. Вопросы внедрения кода рассмотрены в разделе 4 [RFC7538].

15.5. Ошибки клиента (4xx)

Класс кодов статуса 4xx (Client Error) указывает на то, что клиент, по-видимому, допустил ошибку. За исключением откликов на запрос HEAD серверу **следует** передавать представление, содержащее объяснение ошибочной ситуации и указание её временного или постоянного характера. Эти коды применимы к любому методу запроса. Пользовательским агентам **следует** выводить содержимое отклика для пользователя.

15.5.1. 400 Bad Request

Код 400 (Bad Request) говорит, что сервер не может или не будет обрабатывать запрос из-за чего-то, представляющегося ошибкой клиента (например, синтаксическая ошибка в запросе, непригодное обрамление сообщения, обманная маршрутизация запроса).

15.5.2. 401 Unauthorized

Код 401 (Unauthorized) указывает, что запрос не был применён из-за отсутствия действительных аутентификационных свидетельств для целевого ресурса. Сервер, генерирующий отклик 401, **должен** передавать поле заголовка WWW-Authenticate (параграф 11.6.1), содержащее хотя бы один вызов (challenge), применимый к целевому ресурсу.

Если запрос включал аутентификационные свидетельства, отклик 401 говорит, что предоставление полномочий на основании представленных свидетельств отклонено. Агент пользователя **может** повторить запрос с новым или измененным полем заголовка Authorization (параграф 11.6.2). Если отклик 401 содержит тот же вызов, что и предыдущий отклик, а агент пользователя уже предпринял хотя бы одну попытку аутентификации, агенту **следует** вывести вложенное представление для пользователя, поскольку обычно оно содержит диагностические сведения.

15.5.3. 402 Payment Required

Код 402 (Payment Required) зарезервирован на будущее.

15.5.4. 403 Forbidden

Код 403 (Forbidden) показывает, что сервер понял запрос, но отказался выполнять его. Сервер, желающий сообщить причину отказа, может указать её в содержимом отклика (если оно имеется).

Если в запросе были указаны свидетельства для аутентификации, это означает, что сервер счёл их недостаточными для предоставления доступа. Клиенту **не следует** автоматически повторять запрос с теми же свидетельствами, но он **может** повторить запрос с другими или новыми свидетельствами. Однако запрос может быть отклонён по причинам, не связанным со свидетельствами. Сервер-источник, желающий «скрыть» наличие данного защищённого целевого ресурса, **может** передать отклик с кодом 404 (Not Found).

[1] В оригинале это предложение отсутствует, см. https://www.rfc-editor.org/errata/eid7109. *Прим. перев.*

15.5.5. 404 Not Found

Код 404 (Not Found) указывает, что сервер-источник не нашёл текущего представления целевого ресурса или не желает раскрывать его наличие. Код 404 не говорит, является отсутствие постоянным или временным. Если у сервера-источника есть основания предполагать постоянное отсутствие (например, с помощью настраиваемых средств) следует передавать код 410 (Gone) вместо 404.

Отклик 404 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.5.6. 405 Method Not Allowed

Код 405 (Method Not Allowed) указывает, что полученный в строке запроса метод известен серверу-источнику, но не поддерживается целевым ресурсом. Сервер-источник **должен** помещать в отклик 405 поле заголовка Allow со списком методов, поддерживаемых в настоящее время целевым ресурсом.

Отклик 405 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.5.7. 406 Not Acceptable

Код 406 (Not Acceptable) показывает, что у целевого ресурса нет текущего представления, приемлемого для агента пользователя в соответствии с полями упреждающего согласования в заголовке запроса (параграф 12.1), а сервер не желает возвращать принятое по умолчанию представление.

Серверу **следует** генерировать содержимое, включающее список доступных характеристик представления и соответствующие идентификаторы ресурсов, из которых пользователь или его агент могут наиболее подходящее представление. Агент пользователя **может** автоматически выбрать наиболее подходящий вариант из списка, однако эта спецификация не задаёт стандарта для такого автоматического выбора, как указано в параграфе 15.4.1.

15.5.8. 407 Proxy Authentication Required

Код 407 (Proxy Authentication Required) похож на 401 (Unauthorized), но указывает, что клиенту необходимо аутентифицироваться для использования прокси с этим запросом. Прокси **должен** передавать поле заголовка Proxy-Authenticate (параграф 11.7.1) с вызовом (challenge) применимым к этому прокси для запроса. Клиент **может** повторить запрос с новым или измененным полем заголовка Proxy-Authorization (параграф 11.7.2).

15.5.9. 408 Request Timeout

Код 408 (Request Timeout) указывает, что сервер не получил полного сообщения с запросом в течение времени ожидания. Если у клиента имеется незавершённый запрос, он **может** повторить его. Если текущее соединение непригодно (например, из-за потери разграничения запросов в HTTP/1.1), будет использовано новое соединение.

15.5.10. 409 Conflict

Код 409 (Conflict) указывает, что запрос не может быть выполнен из-за конфликта с текущим состоянием целевого ресурса. Этот код применяется в ситуациях, когда пользователь может устранить конфликт и подать запрос снова. Серверу **следует** генерировать содержимое, включающее сведение, достаточные для определения причин конфликта.

Конфликты наиболее вероятны для запросов PUT. Например, при поддержке версий и включении в PUT представления, содержащего изменения, которые конфликтуют с внесёнными ранее (запрос от другой стороны), сервер-источник может использовать отклик 409 для индикации невозможности выполнить запрос. В этом случае представление в отклике будет, вероятно, включать сведения, полезные для слияния разных версий на основе истории выпусков.

15.5.11. 410 Gone

Код 410 (Gone) говорит, что доступ к целевому ресурсу на сервере-источнике больше не предоставляется и это состояние, скорей всего, является постоянным. Если сервер-источник не знает или не имеет возможности определить, является ли это состояние постоянным, должен использоваться код 404 (Not Found).

Код 410 предназначен, прежде всего, для помощи в поддержке web уведомляя получателя о том, что ресурс осознанно сделан недоступным и владельцы сервера хотят избавиться от удалённых ссылок на этот ресурс. Такие события обычны для рекламных услуг с ограниченным сроком действия и ресурсов, принадлежащих лицам, больше не связанным с сайтом сервера-источника. Не требуется помечать все постоянно недоступные ресурсы как «ушедшие» или сохранять такую маркировку определённое время - это отдаётся на откуп владельцу сервера.

Отклик 410 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.5.12. 411 Length Required

Код 411 (Length Required) указывает, что сервер отклонил запрос без Content-Length (параграф 8.6). Клиент **может** повторить запрос, добавив поле заголовка Content-Length, указывающее размер содержимого запроса.

15.5.13. 412 Precondition Failed

Код 412 (Precondition Failed) говорит о невыполнении одного или нескольких условий, заданных в полях заголовка запроса, при проверке на сервере (раздел 13). Этот код в отклике позволяет клиенту задать предварительные условия с учётом текущего состояния ресурса (текущее представление и метаданные) и предотвратить применение метода, когда целевой ресурс находится в неожиданном состоянии.

15.5.14. 413 Content Too Large

Код 413 (Content Too Large) указывает, что сервер отклонил запрос из-за того, что размер его содержимого больше, чем сервер хочет или может обработать. Сервер **может** прервать запрос, если версия протокола позволяет это. В ином случае сервер **может** разорвать соединение.

Если условие является временным, серверу **следует** создать поле заголовка Retry-After, указывающее клиенту, когда **можно** повторить запрос.

15.5.15. 414 URI Too Long

Код 414 (URI Too Long) говорит об отказе сервера обрабатывать запрос из-за размера URI цели, превышающего поддерживаемый. Это редкое состояние может возникать лишь в случае, когда клиент неправильно преобразовал запрос POST в GET со слишком длинной информацией, клиент попал в петлю перенаправления (например, префикс URI указывает свой суффикс) или сервер подвергается атаке со стороны клиента, пытающегося воспользоваться уязвимостями.

Отклик 414 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.5.16. 415 Unsupported Media Type

Код 415 (Unsupported Media Type) указывает, что сервер-источник отказывается обслуживать запрос из-за того, что формат его содержимого не поддерживается на целевом ресурсе для этого метода. Проблема с форматом может быть связана с указанным в запросе Content-Type или Content-Encoding, либо возникать при прямом просмотре данных. Если проблема вызвана неподдерживаемым кодированием содержимого, в заголовке отклика должно использоваться поле Accept-Encoding (параграф 12.5.3) для указания воспринятых в запросе вариантов кодирования содержимого (при наличии). Если причиной является неподдерживаемый тип носителя, можно использовать в отклике поле заголовка Accept (параграф 12.5.1) для указания воспринятых типов носителя в запросе.

15.5.17. 416 Range Not Satisfiable

Код 416 (Range Not Satisfiable) указывает, что набор диапазонов в поле заголовка Range (параграф 14.2) был отвергнут из-за того, что ни один из них не является удовлетворительным или клиент указал слишком много мелких или перекрывающихся диапазонов (возможная атака на отказ а обслуживании).

Каждая единица диапазона указывает, что требуется для того, чтобы её диапазоны были удовлетворительны. Например, в параграфе 14.1.2 указано, что делает диапазоны байтов удовлетворительными.

Серверу, генерирующему отклик 416 на запрос byte-range, **следует** создавать поле Content-Range, задающее текущий размер выбранного представления (параграф 14.4). Например,

```
HTTP/1.1 416 Range Not Satisfiable
Date: Fri, 20 Jan 2012 15:41:54 GMT
Content-Range: bytes */47022
```

Примечание. Поскольку серверы могут игнорировать Range, многие реализации будут отвечать полным выбранным представлением в отклике 200 (OK). Отчасти это связано с тем, что большинство клиентов готовы получить отклик 200 (OK) для завершения задачи (менее эффективно), отчасти с тем, что клиенты не могут остановить создание запроса с недопустимым диапазоном, пока не получат полное представление. Таким образом, клиенты не могут рассчитывать на получение отклика 416 (Range Not Satisfiable), даже ели это наиболее целесообразно.

15.5.18. 417 Expectation Failed

Код 417 (Expectation Failed) указывает, что ожиданию, заданному полем Expect в заголовке запроса (параграф 10.1.1), не соответствует хотя бы один из входных (inbound) серверов.

15.5.19. 418 (Unused)

В первоапрельском [RFC2324] высмеиваются различные способы злоупотребления HTTP, одним из которых является определение специфичного для приложения кода статуса 418, который применялся в качестве шутки достаточно часто, чтобы стать непригодным для использования в будущем. Поэтому код 418 указан как резервный в реестре IANA HTTP Status Code Registry. Это означает, что в настоящее время код не может быть присвоен какому-либо приложению. Если в будущем возникнут обстоятельства, требующие выделить этот код (например, исчерпание кодов 4NN), он может быть назначен снова.

15.5.20. 421 Misdirected Request

Код 421 (ошибочно направленный запрос) указывает, что запрос был направлен серверу, который не может или не хочет создавать полномочный отклик для URI цели. Сервер-источник (или действующий от его имени шлюз) передаёт код 421 для отклонения целевого URI, не соответствующего источнику, для которого сервер настроен (параграф 4.3.1), или контексту соединения, через которое был получен запрос (параграф 7.4).

Клиент, получивший отклик 421 (Misdirected Request), **может** повторить запрос, независимо от идемпотентости метода, через другое соединение, например, через свежее соединение с источником целевого ресурса или через альтернативную службу [ALTSVC].

Прокси **недопустимо** генерировать отклик 421.

15.5.21. 422 Unprocessable Content

Код 422 (Unprocessable Content) указывает, что сервер понимает тип содержимого в запросе (поэтому код 415 (Unsupported Media Type) неуместен) и синтаксис содержимого, но не способен обработать инструкции из запроса. Например, этот код может передаваться, если содержимое XML в запросе содержит корректно сформированные (синтаксически верные), но семантически ошибочные инструкции XML.

15.5.22. 426 Upgrade Required

Код 426 (Upgrade Required) указывает, что сервер отказывается выполнять запрос с использованием текущего протокола, но может сделать это после обновления протокола клиентом. Сервер **должен** передавать поле Upgrade в отклике 426 для указания требуемого протокола или протоколов (параграф 7.8). Например,

```
HTTP/1.1 426 Upgrade Required
Upgrade: HTTP/3.0
Connection: Upgrade
Content-Length: 53
Content-Type: text/plain
```

Эта служба требует использования протокола HTTP/3.0.

15.6. Ошибки сервера (5xx)

Класс кодов 5xx (Server Error) указывает, что сервер знает о своей ошибке или неспособности выполнить запрошенный метод. За исключением откликов на запрос HEAD, серверу **следует** передавать представление с разъяснением ошибки и временный или постоянный характер связанной с ней ситуации. Агенту пользователя **следует** выводить пользователю любое включённое в отклик содержимое. Коды этого класса применимы к любому методу запроса.

15.6.1. 500 Internal Server Error

Код 500 (Internal Server Error) указывает, что сервер столкнулся с неожиданными условиями, помешавшими выполнить запрос.

15.6.2. 501 Not Implemented

Код 501 (Not Implemented) указывает, что сервер не поддерживает функциональность, требуемую для выполнения запроса. Такой отклик подходит для случаев, когда сервер не распознал метод запроса и не способен поддерживать его для какого-либо ресурса.

Отклик 501 является эвристически кэшируемым, т. е. кэшируется, если иное не указано определением метода или явным элементом управления кэшем (см. параграф 4.2.2 в [CACHING]).

15.6.3. 502 Bad Gateway

Код 502 (Bad Gateway) указывает, что сервер, выступая в качестве шлюза или прокси, получил непригодный отклик от входного (inbound) сервера, к которому он обратился для выполнения запроса.

15.6.4. 503 Service Unavailable

Код 503 (Service Unavailable) говорит, что в данный момент сервер не способен обработать запрос из-за временной перегрузки или планового обслуживания, что будет, вероятно, устранено по истечении некоторого времени. Сервер **может** передать поле Retry-After (параграф 10.2.3), указывающее клиенту время ожидания до повтора запроса.

> Примечание. Наличие кода 503 не означает, что сервер обязан использовать его при перегрузке. Некоторые серверы могут просто отвергать соединение.

15.6.5. 504 Gateway Timeout

Код 504 (Gateway Timeout) указывает, что сервер, выступая в качестве шлюза или прокси, не получил своевременного отклика от сервера восходящего направления, к которому должен был обратиться для выполнения запроса.

15.6.6. 505 HTTP Version Not Supported

Код 505 (HTTP Version Not Supported) говорит, что сервер не поддерживает или отвергает старшую (major) версию HTTP, использованную в запросном сообщении. Сервер указывает невозможность или нежелание завершить запрос, используя ту же старшую версию, которую указал клиент, как описано в параграфе 2.5, иначе, нежели отправкой этого сообщения об ошибке. Серверу **следует** генерировать для отклика 505 представление, описывающее причину отклика и протоколы, поддерживаемые этим сервером.

16. Расширения HTTP

В HTTP определено множество базовых точек расширения, которые могут применяться для добавления в протокол возможностей без введения новой версии, включая методы, коды статуса, имена полей и дополнительные точки расширения в имеющихся полях, такие как схемы аутентификации и директивы кэширования (см. расширения Cache-Control в параграфе 5.2.3 [CACHING]). Поскольку семантика HTTP не имеет версий, эти точки расширения являются постоянными и версия протокола не меняет их семантики.

Для независимых от версии расширений рекомендуется избегать зависимости от конкретной версии протокола или интеграции с таковой. При неизбежности этого нужно тщательно продумать функциональную совместимость расширения с разными версиями. Кроме того, в конкретных версиях HTTP могут быть свои точки расширения, такие как транспортное кодирование в HTTP/1.1 (параграф 6.1 в [HTTP/1.1]) и HTTP/2 SETTINGS или типы кадров ([HTTP/2]). Эти точки расширения специфичны для соответствующей версии протокола. Зависящие от версии расширения не могут переопределять или изменять семантику независимых от версии механизмов и точек расширения (например, метода или поля заголовка) без явного разрешения со стороны элемента протокола. Например, такие изменения разрешает метод CONNECT (параграф 9.3.6).

Эти рекомендации обеспечивают корректную и предсказуемую работу протокола даже при использовании на участках пути разных версий HTTP.

16.1. Расширяемость методов

16.1.1. Реестр методов

В реестр IANA Hypertext Transfer Protocol (HTTP) Method Registry (https://www.iana.org/assignments/http-methods) вносятся имена методов. Регистрация метода HTTP **должна** включать указанные ниже поля.

```
Имя метода (раздел 9)
Безопасный (yes или no, параграф 9.2.1)
Идемпотентный (yes или no, параграф 9.2.2)
Указатель на текст спецификации.
```

Значения выделяются по процедуре IETF Review (параграф 4.8 в [RFC8126]).

16.1.2. Рассмотрение новых методов

Стандартизованные методы являются базовыми, т. е. потенциально применимыми для любого ресурса, а не просто для отдельного типа носителя, вида ресурса или приложения. Поэтому предпочтительно регистрировать новые методы в документах, не привязанных к конкретному приложению или формату данных, поскольку ортогональные технологии заслуживают ортогональных спецификаций.

Синтаксический анализ сообщения (раздел 6) должен быть независимым от семантики методов (за исключением откликов на HEAD), поэтому определения новых методов не могут менять алгоритм синтаксического анализа или запрещать наличие содержимого в запросе или отклике. Определения новых методов могут указывать, что разрешено лишь пустое (размером 0) содержимое, требуя наличия поля заголовка Content-Length со значением 0.

Новые методы не могут использовать специальные формы цели запроса host:port и *, которые разрешены для CONNECT и OPTIONS, соответственно (параграф 7.1). Для URI цели требуется полная форма абсолютного URI, т. е. цель запроса должна передаваться в абсолютной форме или URI будет восстанавливаться из контекста запроса как в других методах.

Определение нового метода должно указывать, является ли метод безопасным (параграф 9.2.1), идемпотентным (параграф 9.2.2), кэшируемым (параграф 9.2.3), какая семантика связана с содержимым запроса (при его наличии) и какие уточнения метод вносит в семантику поля заголовка или кода статуса. Если новый метод является кэшируемым, определение должно указывать, как и при каких условиях в кэше может сохраняться отклик и как такой отклик может использоваться для выполнения последующих запросов. Новый метод должен указывать, является ли он условным (параграф 13.1) и задавать поведение сервера при невыполнении условий (для условных методов). Если новый метод может применяться с семантикой частичных откликов (параграф 14.2), документ должен описывать это.

> Примечание. Следует избегать определения методов, имена которых начинаются с M-, поскольку такой префикс может быть неверно истолкован, как имеющий семантику, заданную в [RFC2774].

16.2. Расширяемость кодов статуса

16.2.1. Реестр кодов статуса

В реестр IANA Hypertext Transfer Protocol (HTTP) Status Code Registry (https://www.iana.org/assignments/http-status-codes) вносятся коды статуса для откликов. При регистрации **должны** указываться приведённые ниже поля.

```
Код статуса (3 цифры)
Краткое описание
Указатель на текст спецификации.
```

Значения выделяются по процедуре IETF Review (параграф 4.8 в [RFC8126]).

16.2.2. Рассмотрение новых кодов статуса

Когда требуется выразить семантику отклика, не определяемую имеющимися кодами статуса, могут регистрироваться новые коды. Коды статуса являются базовыми, т. е. потенциально применимыми для любого ресурса, а не просто для отдельного типа носителя, вида ресурса или приложения HTTP. Поэтому предпочтительно регистрировать новые методы в документах, не привязанных к отдельному приложению.

Новые коды статуса должны относиться к одной из категорий, указанных в разделе 15. Чтобы синтаксические анализаторы могли обрабатывать сообщения с откликами, новые коды статуса не могут запрещать содержимое, но могут требовать пустое содержимое (размером 0).

В предложениях по новым кодам статуса, ещё не получившим широкого распространения, следует избегать указания конкретных числовых значений кодов, пока не будет чёткого согласия о возможности регистрации кода. Вместо этого в ранних версиях можно использовать обозначение вида 4NN, 3N0 .. 3N9, указывающие класс предлагаемых кодов статуса без преждевременного назначения номера.

Определение нового кода статуса должно разъяснять условия, при которых запрос будет вызвать отклик с таким кодом статуса (например, комбинация полей заголовка и/или методов, а также любые зависимости полей заголовка отклика, такие как обязательные или меняющие семантику поля, уточнение семантики при использовании нового кода).

По умолчанию код статуса относится лишь к запросу, вызвавшему отклик с этим кодом. Если код статуса относится к более широкой области (например, ко всем запросам к соответствующему ресурсу), это должно указываться явно. При этом следует учитывать, что не все клиенты будут согласованно применять такие коды, поскольку могут не понимать новый код.

В определении кода финального статуса должно быть указано, является ли отклик эвристически кэшируемым. Следует отметить, что любой отклик с кодом финального статуса может кэшироваться, если он содержит явные сведения о свежести информации. Код статуса, заданный как эвристически кэшируемый, можно кэшировать без явно указанной свежести информации. Определение кода статуса может вносить ограничения для поведения кэша при использовании директивы must-understand (см. [CACHING]).

Определение нового кода статуса должно указывать, имеет ли содержимое какую-либо подразумеваемую связь с идентифицированным ресурсом (параграф 6.4.2).

16.3. Расширяемость полей

Наиболее частым расширениям подвергается набор определений полей заголовков и трейлеров. Новые поля могут определяться так, что, будучи понятыми получателем, они переопределяют или расширяют интерпретацию определённых ранее полей, задают предварительные условия для оценки запроса или уточняют значение откликов.

Однако определение поля не гарантирует его внедрения и распознавания получателями. Большинство полей создаётся с расчётом на возможность безопасного игнорирования (но с пересылкой в нисходящем направлении) не понимающим поле получателем. В иных случаях способность отправителя понять данное поле может указываться предшествующим взаимодействие, возможно, версией протокола или полями, переданными в предшествующих сообщениях, а также использованием конкретного типа носителя. Непосредственная проверка поддержки может быть выполнена с помощью запроса OPTIONS или путём взаимодействия с определенным общеизвестным URI [RFC8615], если такая проверка определена вместе с новым полем.

16.3.1. Реестр имён полей

Реестр Hypertext Transfer Protocol (HTTP) Field Name Registry определяет пространство имён для полей HTTP. Регистрацию поля HTTP может запросить любая сторона. Аспекты добавления полей HTTP рассматриваются в параграфе 16.3.2. Реестр Hypertext Transfer Protocol (HTTP) Field Name Registry доступен по ссылке <https://www.iana.org/assignments/http-fields/>. Запросы на регистрацию выполняются в соответствии с приведёнными ниже инструкциями или путём отправки сообщения в список рассылки ietf-http-wg@w3.org.

Имена полей регистрируются по рекомендации эксперта, назначенного IESG или уполномоченным IESG органом. Поля с постоянным статусом (permanent) регистрируются по процедуре Specification Required ([RFC8126], параграф 4.6).

Запрос на регистрацию должен включать указанные ниже поля.

Field name - имя поля
: Имя регистрируемого поля. Имя должно соответствовать синтаксису, заданному в параграфе 5.1, и следует использовать имена, начинающиеся с буквы и включающие только буквы, цифры и символ дефиса (-).

Status - статус
: Постоянное (permanent), временное (provisional), отменённое (deprecated), устаревшее (obsoleted).

Specification document(s) - документы со спецификацией
: Ссылка на документы, определяющие данное поле, желательно с указанием URI для получения копии. Это поле не требуется, но приветствуется для временных регистраций. Можно также включать указание на раздел документа.

Кроме того, можно включать дополнительное поле.

Comments - комментарии
: Дополнительные сведения, например, о зарезервированных записях.

Эксперты, проконсультировавшись с сообществом, могут добавить поля для включения в реестр.

Определённые в стандартах имена полей имеют статус permanent. Другие имена также получают такой статус, если эксперт установит, что они применяются, проконсультировавшись с сообществом. Для других имён следует задавать статус provisional. Эксперты могут удалять временные (provisional) записи после консультации с сообществом, если установлено, что они не применяются. Эксперт может в любой момент сменить временный статус на permanent.

Отметим, что имена могут регистрировать третьи лица (включая экспертов), если эксперт сочтёт, что незарегистрированное имя широко применяется и вряд ли будет зарегистрировано иным путём.

16.3.2. Рассмотрение новых полей

Поля заголовка и трейлера HTTP широко применяются для расширения протокола. Несмотря на возможность специализированного использования, поля предназначенные для более широкого применения, должны тщательно документироваться для обеспечения функциональной совместимости. В частности, авторам спецификаций, задающих новые поля, рекомендуется учитывать и, по возможности, документировать указанные ниже аспекты.

- Условия, при которых может применяться поле, например, только в запросах или откликах, во всех сообщениях, в откликах для определённого метода запроса и т. п.
- Зависимость семантики поля от контекста, например, использование с определёнными методами запроса или кодами статуса.
- Область применимости передаваемых сведений. По умолчанию поля применяются только к сообщению, с которым они связаны, но некоторые поля откликов предназначены для применения ко всем представлениям ресурса, самому ресурсу или более широкой области. В спецификациях, расширяющих область действия поля, требуется тщательно учитывать такие аспекты, как согласование содержимого, временной интервал применимости и (в некоторых случаях) работу серверов для нескольких арендаторов (multi-tenant).
- Условия, при которых посредникам разрешено вставлять, удалять или изменять значение поля.
- Возможность размещения поля в трейлере (по умолчанию запрещено, см. параграф 6.5.1).
- Уместность или необходимость указывать имя поля в поле заголовка Connection (параграф 7.6.1).
- Связанные с полем вопросы безопасности, например, раскрытие связанных с приватностью сведений.

С полями заголовков запросов связаны дополнительные аспекты, которые нужно документировать, если принятое по умолчанию поведение не подходит.

- Уместность включения имени поля в поле заголовка отклика Vary (например, при использовании поля заголовка запроса сервером-источником в алгоритме выбора содержимого, см. параграф 12.5.5).

- Сохранение поля при его получении в запросе PUT (параграф 9.3.4).
- Удаление поля при автоматическом перенаправлении запроса в целях безопасности (параграф 15.4).

16.3.2.1. Рассмотрение новых имён полей

Авторам спецификаций, определяющих новые поля, рекомендуется выбирать для полей короткие и описательные имена. Краткость позволяет сократить расход пропускной способности, описательность избавляет от путаницы и захвата (squatting) имён, которые могут иметь широкое применение. С учётом этого рекомендуется выбирать для полей ограниченного применения (например, для одного приложения или варианта использования) имена, включающие такое применение (или сокращение для него) в качестве префикса. Например, если для приложения Foo требуется поле Description, можно использовать имя Foo-Desc. Имя Description было бы слишком общим, а Foo-Description - избыточно длинным.

Хотя синтаксис имён полей определён с возможностью включения любых символов (token), на практике некоторые реализации вносят ограничения на символы в именах полей. Для совместимости в именах новых полей **следует** ограничиваться буквами, цифрами, а также символами «-» и «.», а начинать имя **следует** с буквы. Например, символ подчёркивания (_) может вызывать проблемы при передаче через интерфейсы шлюзов, не относящихся к HTTP (параграф 17.10). Имена полей должны избегать префикса X- (см. [BCP178]). Другие префиксы иногда применяются в именах полей HTTP. Например, префикс Accept- используется во многих полях согласования содержимого, а применение префикса Content- описано в параграфе 6.4. Такие префиксы помогают распознать назначение поля и не вызывают автоматической обработки.

16.3.2.2. Рассмотрение новых значений полей

При определении нового поля HTTP важной задачей является задание синтаксиса значений - что следует генерировать отправителю и как получателям следует понимать семантику полученного значения. Авторам рекомендуется (но не требуется) применять для задания синтаксиса значений новых полей правила ABNF из данной спецификации или правила [RFC8941]. Авторам рекомендуется тщательно продумать влияние присутствия нескольких строк поля (параграф 5.3). Поскольку отправители могут по ошибке передавать несколько значений, а посредники и библиотеки HTTP могут автоматически объединять их, этот относится ко всем (даже одиночным) значениям поля. Поэтому авторам рекомендуется разграничивать или кодировать значения с запятыми (например, с помощью правила quoted-string из параграфа 5.6.4, типа данных String из [RFC8941] или специального кодирования поля). Это гарантирует, что запятые в значении поля не будут спутаны с запятыми, разделяющими значения в списке. Например, для поля Content-Type запятые можно размещать лишь внутри кавычек, что обеспечивает надёжный разбор даже при наличии нескольких значений. Значения поля Location являются контр-примером, которому лучше не следовать - поскольку URI могут содержать запятые, невозможно надёжно отличить одно значение с запятой от двух значений через запятую. Авторам полей с единственным (singleton) значением (параграф 5.5) дополнительно рекомендуется документировать поведение при получении сообщения с несколькими элементами (разумным поведением по умолчанию является игнорирование поля, но в некоторых случаях это может оказаться неверным решением).

16.4. Расширяемость схем аутентификации

16.4.1. Реестр схем аутентификации

Реестр Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry (https://www.iana.org/assignments/http-authschemes) указывает пространство имён для схем аутентификации в вызовах и свидетельствах. Регистрация **должна** включать указанные ниже поля.

```
Имя схемы аутентификации
Указатель на текст спецификации
Примечания (необязательно)
```

Значения добавляются в реестр по процедуре IETF Review ([RFC8126], параграф 4.8).

16.4.2. Рассмотрение новых схем аутентификации

Ниже указаны аспекты модели аутентификации HTTP, ограничивающие работу новых схем проверки подлинности.

- Предполагается, что при аутентификации HTTP не создаются состояния и все сведения, нужные для проверки подлинности запроса, **должны** предоставляться в запросе, а не зависеть от запоминания сервером предыдущих запросов. Аутентификация на основе привязки к базовому соединению выходит за рамки этой спецификации и по своей природе несовершенна, пока не приняты меры, предотвращающие использование соединения любой другой стороной, кроме аутентифицированного пользователя (см. параграф 3.3).
- Параметр аутентификации realm зарезервирован для задания пространств защиты, как описано в параграфе 11.5. В новых схемах **недопустимо** использовать параметр не совместимым с этим определением способом.
- Нотация token68 была введена для совместимости с имеющимися схемами аутентификации и может применяться в вызове или свидетельстве лишь однократно. Поэтому в новых схемах вместо этого должен применяться синтаксис auth-param, поскольку иначе будущие расширения станут невозможными.
- Синтаксический анализ вызовов и свидетельств задаётся этой спецификацией и не может быть изменён новыми схемами аутентификации. При использовании синтаксиса auth-param все параметры должны поддерживать маркеры (token) и строки в кавычках, а синтаксические ограничения должны задаваться для значений полей после разбора (обработки строк с кавычками). Это требуется для того, чтобы получатель могли использовать базовый синтаксический анализатор, применимый ко всем схемам аутентификации.

  Примечание. Ограничение синтаксиса параметра realm строкой в кавычках было неудачным выбором, который не следует повторять для новых параметров.
- В определениях новых схем должно обращение с неизвестными параметрами расширения. В общем случае правило must-ignore (игнорировать) предпочтительней правила must-understand (нужно понимать), поскольку иначе будут сложно вводить новые параметры при наличии унаследованных получателей. Кроме того,

полезно описать правила определения новых параметров (такие как обновление спецификации или использование реестра).

- В документации к схемам аутентификации должно быть указано, пригодны ли они для аутентификации на сервере-источнике (с использованием WWW-Authenticate) и/или прокси (с использованием Proxy-Authenticate).
- Свидетельства, передаваемые в поле заголовка Authorization, специфичны для пользовательского агента, поэтому оказывают на кэши HTTP такое же влияние, как директива private для кэш-откликов (параграф 5.2.2.7 в [CACHING]), в рамках запроса, где они появляются. Таким образом, новые схемы, отказавшиеся от передачи свидетельство в поле Authorization (например, за счёт передачи в новом поле заголовка), должны явно запретить кэширование, требуя использовать директивы для откликов из кэша (например, private).
- Схемы, использующие Authentication-Info, Proxy-Authentication-Info или какое-либо иное поле заголовка, связанное с аутентификацией, должны рассматривать и документировать связанные с этим вопросы безопасности (см. параграф 17.16.4).

16.5. Расширяемость единиц диапазонов

16.5.1. Реестр единиц диапазонов

Реестр IANA HTTP Range Unit Registry (https://www.iana.org/assignments/http-parameters) содержит пространства имён для блоков диапазонов с указанием соответствующих спецификаций. Каждая запись HTTP Range Unit **должна** включать указанные ниже поля.

Имя
Описание
Указатель на текст спецификации.

Значения добавляются в реестр по процедуре IETF Review ([RFC8126], параграф 4.8).

16.5.2. Рассмотрение новых единиц диапазонов

Другие единицы диапазонов, вроде связанных с форматом границ, как для страниц, разделов, записей, строк, или время могут быть применимы в HTTP для специфичных целей приложений, но не применяться на практике широко. Разработчики таких единиц диапазонов должны рассматривать их работу с кодировками содержимого и посредниками общего назначения.

16.6. Расширяемость кодирования содержимого

16.6.1. Реестр кодирования содержимого

Реестр IANA HTTP Content Coding Registry (https://www.iana.org/assignments/http-parameters/) содержит имена кодирования содержимого. Каждая запись реестра **должна** включать указанные ниже поля.

Имя
Описание
Указатель на текст спецификации.

Именам кодирования содержимого **недопустимо** пересекаться с именами транспортного кодирования (HTTP Transfer Coding Registry, https://www.iana.org/assignments/http-parameters/), если только эти преобразования не являются идентичными (как в случае кодирования со сжатием, рассмотренного в параграфе 8.4.1).

Значения добавляются в реестр по процедуре IETF Review ([RFC8126], параграф 4.8) и **должны** соответствовать цели кодирования содержимого, заданной в параграфе 8.4.1.

16.6.2. Рассмотрение новых вариантов кодирования содержимого

Новые варианты кодирования содержимого должны, по возможности, описывать себя с обнаружением необязательных параметров в самом формате кодирования без внешних метаданных, которые могут теряться при передаче.

16.7. Обновление реестра маркеров

Реест Hypertext Transfer Protocol (HTTP) Upgrade Token Registry (https://www.iana.org/assignments/http-upgrade-tokens) содержит пространства имён для маркеров имён протоколов, служащих для идентификации протокола а поле заголовка Upgrade. С каждым зарегистрированным именем протокола связаны контактные данные и может быть связан набор спецификаций, указывающих, как будет обрабатываться соединение после его обновления. Регистрация выполняется по процедуре First Come First Served (параграф 4.4 в [RFC8126]) и указанным ниже правилам.

1. Однажды зарегистрированный маркер protocol-name сохраняется навсегда.
2. Регистр символов в protocol-name не учитывается, а имя включается в реестр в соответствии с предпочтениями отправителей.
3. При регистрации **должна** быть указана ответственная за регистрацию сторона.
4. При регистрации **должны** быть указаны контактные данные.
5. Регистрация **может** включать связанные с маркером спецификаций, которые должны быть общедоступными.
6. В регистрацию **следует** включать набор ожидаемых пар «протокол-версия», связанных с маркером на момент регистрации.
7. Ответственная сторона **может** изменить регистрацию в любой момент. IANA будет вести реестр таких изменений и предоставлять его по запросу.
8. IESG **может** поменять ответственную за регистрацию маркера сторону (обычно это происходит лишь при отсутствии возможности связаться с текущим ответственным).

17. Вопросы безопасности

Этот раздел содержит сведения для разработчиков, поставщиков информации и пользователей об известных проблемах безопасности, связанных с семантикой HTTP и использованием протокола для передачи информации через Internet. Связанные с кэшированием вопросы рассмотрены в разделе 7 [CACHING], а вопросы, относящиеся к синтаксису и анализу сообщений HTTP/1.1 - в разделе 11 [HTTP/1.1].

Набор рассмотренных вопросов не является исчерпывающим. Большинство вопросов безопасности, связанных с семантикой HTTP, касается проблем безопасности приложений на стороне сервера (код за интерфейсом HTTP), пользовательских агентов, обрабатывающих полученное по HTTP содержимое или безопасной работы в Internet в целом, а не безопасности самого протокола. Соображения безопасности для URI, являющихся основой работы HTTP, рассмотрены в разделе 7 [URI]. Различные организации поддерживают тематическую информацию и ссылки на современные исследования в сфере безопасности приложений Web (например, [OWASP]).

17.1. Установление полномочий

HTTP полагается на полномочный отклик (authoritative response), который определяется сервером источником (или от его имени), указанным в URI цели, как наиболее приемлемый для запроса с учётом состояния целевого ресурса в момент создания отклика.

При использовании зарегистрированного имени в компоненте authority схема URI http (параграф 4.2.1) полагается на локальную службу распознавания имён, чтобы определить, где можно получить полномочный отклик. Это означает, что атака на таблицу хостов в сети пользователя, кэшированные имена или библиотеки распознавания имён превращается в атаку на установление полномочий URI http. Аналогично, выбор пользователем сервера системы доменных имён (Domain Name Service или DNS) и иерархии серверов, от которых приходят результаты распознавания, может влиять на аутентичность сопоставления с адресами. Защитные расширения DNS (DNS Security Extensions или DNSSEC, [RFC4033]) являются одним из способом повышения достоверности, как и другие механизмы передачи запросов DNS через более защищённые транспортные протоколы.

После получения адреса IP установление полномочности для URI http уязвимо к атакам на маршрутизацию IP.

Схема https (параграф 4.2.2) предназначена для предотвращения (или хотя бы раскрытия) множества возможных атак на установление полномочности, если согласованное соединение защищено, а клиент должным образом проверяет соответствие отождествления сервера компоненту authority в URI цели (параграф 4.3.4). Корректная реализация такой проверки может быть достаточно сложной (см. [Georgiev]).

Полномочия для данного сервера-источника могут быть переданы с помощью расширений протокола, например, [ALTSVC]. Набор серверов, для которых соединение считается полномочным, также может быть изменён с помощью расширений, таких как [RFC8336].

Предоставление отклика из неполномочного источника, такого как кэш общего прокси, зачастую позволяет повысить производительность и доступность, но лишь в той мере, в которой такому источнику можно доверять или безопасно пользоваться недоверенным откликом.

К сожалению довести до пользователей сведения о полномочности может быть непросто. Например, фишинг (phishing) представляет собой атаку на восприятие пользователем полномочий, искажённых путём представления похожего имени (brand) в гипертексте, возможно с помощью userinfo, маскирующего компонент authority (см. параграф 4.2.1). Пользовательские агенты могут смягчить влияние фишинговых атак, позволяя пользователю легко проверить URI цели до выполнения действий, заметно выделяя (или отвергая) userinfo (при наличии) и передавая сохранённые свидетельства и cookie, когда ссылающийся документ получен из неизвестного или недоверенного источника.

17.2. Риск, связанный с посредниками

С посредниками HTTP непосредственно связаны атаки в пути (on-path). Компрометация систем, где работают посредники, может вызывать серьёзные проблемы безопасности и приватности. У посредников может быть доступ к связанным с безопасностью сведениям, персональным данным пользователей и организаций, а также к служебной (proprietary) информации, принадлежащей пользователям или поставщикам содержимого. Скомпрометированный посредник, а также посредник, настроенный без учёта соображений безопасности и приватности, может служить для широкого спектра возможных атак. Посредники с общим кэшем особенно уязвимы для атак с отравлением кэша, как описано в разделе 7 [CACHING].

Разработчики должны учитывать последствия для приватности и безопасности при выборе решений по проектированию и кодированию, а также параметров конфигурации, предоставляемых операторам (особенно принятой по умолчанию конфигурации).

Посредники не заслуживают доверия больше, чем люди и правила, в рамках которых они работают. Протокол HTTP не может решить эту проблему.

17.3. Атаки на основе имён путей и файлов

Серверы-источники часто используют свою локальную файловую систему для управления сопоставлениями целевых URI с представлениями ресурсов. Большинство файловых систем не защищены от вредоносных имён путей и файлов, поэтому серверу нужно избегать обращения по именам, имеющим специальное значение в системе, при сопоставлении целевого ресурса с файлами и каталогами. Например, в UNIX, Microsoft Windows и других операционных системах используются две точки (..) в качестве компонента пути, указывающего каталог на один уровень выше текущего, а также применяются имена путей и файлов для передачи данных системным устройствам. Похожие соглашения об именовании могут применяться и в других типах систем хранения. В локальных системах хранения имеется неприятная тенденция отдавать предпочтение удобству пользователя в ущерб безопасности при обработке недопустимых и неожиданных символов, перекомпоновке разложенных символов и нормализации регистра в именах, не принимающих регистр во внимание. Атаки, основанные на специальных именах, обычно нацелены на отказ в обслуживании (например, запросить у сервера чтение из порта COM) или раскрытие файлов конфигурации или исходных кодов, доступ к которым не предусмотрен.

17.4. Атаки путём внедрения команд, кода или запросов

Серверы-источники часто используют URI для идентификации системных служб, выбора записей базы данных или источника данных. Однако данным, полученным в запросе, нельзя доверять. Атакующий может создать любой из элементов данных запроса (метод, URI цели, поля заголовков, содержимое), поместив в него сведения, которые могут быть ошибочно истолкованы как команда, код или запрос при передаче через вызов команды, интерпретатор языка или интерфейс базы данных. Например, внедрение SQL является распространённой атакой, где часть URI цели или полей заголовка (например, Host, Referer и т. п) содержит язык запросов. Если полученные данные напрямую используются в операторе SELECT, язык запроса может быть интерпретирован как команда базы данных, а не просто строка. Такие уязвимости реализаций встречаются очень часто, несмотря на простоту их предотвращения.

В общем случае реализации ресурсов должны избегать использования данных запроса в контексте, где они обрабатываются или интерпретируются как инструкции. Параметры должны сравниваться с фиксированными строками и обрабатываться по результатам сравнения, а не передаваться через интерфейс, не готовый к недостоверным данным. Принятые данные, которые не основаны на фиксированных параметрах, должны тщательно фильтроваться или декодироваться для предотвращения ошибочной интерпретации.

Аналогичные соображения применимы и к данным запросов, сохраняемых для последующей обработки, таким как файлы системных журналов, результаты мониторинга или сведения, включаемые в форматы данных, допускающие вложенные сценарии.

17.5. Атаки через размер элементов протокола

Поскольку в HTTP используются в основном текстовые поля, ограниченные символами, синтаксические анализаторы часто подвержены атакам, основанным на передаче очень длинных (или очень медленных) потоков данных, особенно в случаях, когда реализация ожидает элемент протокола без предопределённого размера (параграф 2.3).

Для обеспечения функциональной совместимости даны конкретные рекомендации по минимальным размерам полей (параграф 5.4). Значения выбраны так, чтобы их могли поддерживать даже реализации с ограниченными ресурсами. Предполагается, что большинство реализаций будет задавать существенно большие значения.

Сервер может отвергнуть сообщение со слишком длинным URI цели (параграф 15.5.15) или слишком большим содержимым запроса (параграф 15.5.14). Для связанных с размерами ограничений расширениями HTTP [RFC6585] были заданы дополнительные коды статуса.

Получатель должны чётко ограничивать процессы обработки других элементов протокола, включая методы запросов, текст откликов со статусом, имена полей, численные значения, размер блоков (chunk) и пр. Отказ от ограничений на обработку может приводить к исполнению произвольного кода в результате переполнения буферов или разрядности и росту уязвимости к DoS-атакам.

17.6. Атаки со сжатием по общему словарю

В некоторых атаках на защищённые протоколы используется различие в размерах при динамическом сжатии для раскрытия конфиденциальных сведений (см., например, [BREACH]). Такие атаки основаны на создании избыточности контролируемого злоумышленником содержимого по сравнению с конфиденциальными данными, при этом алгоритм динамического сжатия, использующий один словарь в обоих случаях, будет давать более эффективное сжатие при совпадении контролируемого атакующим содержимого с частью конфиденциальных данных.

Сообщения HTTP сжимаются разными способами, включая компрессию TLS, кодирование содержимого, транспортное кодирование и другие механизмы, зависящие от расширений или версии.

Наиболее эффективным способом снижения риска является запрет сжатия конфиденциальных данных или строгая изоляция контролируемых атакующим данных от конфиденциальных сведений, чтобы злоумышленники не могли воспользоваться для сжатия тем же словарём. При осторожном проектировании схема сжатия может быть устроена так, что она не будет уязвимой в некоторых вариантах применения, таких как HPACK ([HPACK]).

17.7. Раскрытие персональных сведений

Клиентам часто доступны большие объёмы персональных данных, включая сведения, предоставляемые им для взаимодействия с ресурсами (например, имя пользователя, местоположение, почтовый адрес, пароль, ключи шифрования и т. п.), и данные о просмотре за определённое время (например, история, закладки и т. п.). Реализации должны предотвращать непреднамеренное раскрытие персональных данных.

17.8. Приватность сведений из системного журнала сервера

Сервер может сохранять персональные данные из запросов пользователей в течение некоторого времени, что позволяет выявить характер запросов или интересующие пользователя темы. В частности, сведения из системного журнала посредника часто содержит историю взаимодействий пользовательских агентов с множеством сайтов и из неё можно извлечь сведения об отдельных пользователях.

Журнальная информация HTTP конфиденциальна по своей природе и её обработка зачастую регулируется законами и нормативными актами. Данные журналов нужно хранить в защищённом месте, а при их анализе нужно соблюдать соответствующие правила. Анонимизация персональных данных в отдельных записях помогает, но обычно её недостаточно для предотвращения трассировки файлов для повторной идентификации на основе сопоставления с другими характеристиками доступа. Поэтому трассировки доступа с привязкой к конкретным клиентам небезопасно публиковать даже при использовании псевдонимов.

Для минимизации рисков кражи или случайной публикации журнальные сведения должны очищаться от персональных данных, включая идентификаторы пользователей, адреса IP и представленные пользователями параметры запросов, как только эти сведения перестанут требоваться для обеспечения безопасности, аудита и борьбы с мошенничеством.

17.9. Раскрытие данных в URI

URI предназначены для совместного использования без защиты, даже если они указывают защищённые ресурсы. URI часто отображаются на экранах, добавляются в шаблоны при печати страниц, сохраняются в закладках без защиты. Многие серверы, прокси и пользовательские агенты записывают в системный журнал или отображают URI без защиты от доступа посторонних. Поэтому не следует включать в URI сведения, которые могут быть конфиденциальными, приватными или опасными для раскрытия.

При использовании приложением на стороне клиента механизмов создания URI цели по предоставленным пользователем данным, таким как поля запроса из формы с методом GET, возможно включение в URI сведений, раскрывать которые не следует. В таких случаях часто предпочтительней использовать POST, поскольку для него URI обычно не создаётся, а данные передаются в содержимом запроса. Однако это мешает кэшированию и использует небезопасный метод для запроса, который иначе был бы безопасным. Другой вариант включает преобразование данных от пользователя до создания URI или фильтрация таких данных с сохранением лишь сведений, не являющихся конфиденциальными. Аналогичным образом перенаправление результата запроса на другой URI (созданный сервером) может удалить конфиденциальные сведения из последующих передач и обеспечить кэшируемость данных.

Поскольку поле Referer говорит целевому сайту о контексте, в котором сделан запрос, оно может раскрывать сведения о ближайшей истории просмотров пользователя, а также персональные данные из URI ссылающегося ресурса. Ограничения для поля заголовка Referer, указанные в параграфе 10.1.3, учитывают некоторые аспекты безопасности.

17.10. Обработка имён полей приложениями

Для обработки полученных запросов и формирования содержимого откликов серверы часто используют интерфейсы и схемы шлюзов, не относящихся к HTTP. По историческим причинам такие интерфейсы часто перелают полученные имена полей как имена внешних переменных, используя подходящее для среды сопоставление имён. Например, отображение общего интерфейса шлюзов (Common Gateway Interface или CGI) для связанных с протоколом метапеременных, описанное в параграфе 4.1.18 [RFC3875], применяется к полученным именам полей, которые не соответствуют ни одной из стандартных переменных CGI - к каждому такому имени поля добавляется префикс HTTP_, а все символы дефиса (-) заменяются символами подчёркивания (_). Такое сопоставление было унаследовано многими другими прикладными схемами для упрощения переноса приложений с одной платформы на другую. В CGI полученное поле Content-Length передаётся как метапеременная CONTENT_LENGTH со строковым значением, соответствующим значению принятого поля. Полученное поле Content_Length будет передаваться как связанная с протоколом метапеременная HTTP_CONTENT_LENGTH, что может приводить к путанице, если приложение ошибочно прочитает связанную с протоколом метапеременную вместо стандартной переменной (именно поэтому в параграфе 16.3.2.1 рекомендуется не создавать новых полей с символом подчёркивания).

К сожалению сопоставление имён полей с именами другого интерфейса может приводить к серьёзным уязвимостям, если это сопоставление неполно или неоднозначно. Например, если злоумышленник передаёт поле с именем Transfer_Encoding, наивный интерфейс может отобразить его в ту же переменную, что и поле с именем Transfer-Encoding, что приведёт к возможности контрабанды запросов (параграф 11.2 в [HTTP/1.1]). Для снижения связанных с этим рисков реализациям, выполняющим отображения рекомендуется делать их однозначными и полными для всего диапазона октетов, возможных в имени (включая нерекомендуемые и запрещённые в грамматике HTTP). Например, поле с необычным символом в имени может приводить к блокированию запроса, удалению соответствующего поля или передаче имени с другим суффиксом, чтобы отличить его от остальных полей.

17.11. Раскрытие фрагмента после перенаправления

Хотя идентификаторы фрагментов в ссылках URI не передаются в запросах, разработчики должны понимать, что их будет видеть агент пользователя и любые расширения и сценарии, запускаемыми в результате отклика. В частности, при перенаправлении с наследованием идентификатора фрагмента из исходного запроса в новой ссылке в Location (параграф 10.2.2) фрагмент одного сайта может раскрываться другому сайту. Если первый сайт использует во фрагментах персональные данные, он должен гарантировать включение в перенаправления на другие сайты идентификатора фрагмента (возможно, пустого) для блокировки такого наследования.

17.12. Раскрытие сведений о продукции

Поля заголовка User-Agent (параграф 10.1.5), Via (параграф 7.6.3) и Server (параграф 10.2.4) часто раскрывают сведения о соответствующих программных системах отправителя. В теории это может упростить злоумышленникам использование известных уязвимостей, но на практике атакующие обычно пытаются воспользоваться всеми уязвимостями независимо от фактически используемых версий.

Прокси, служащие порталом с межсетевым экраном, должны принимать особые меры предосторожности при передаче сведений в заголовках, способных указать хосты, расположенные за межсетевым экраном. Поле Via позволяет посредникам заменить скрываемые имена машин псевдонимами.

17.13. Отпечатки браузеров

Отпечатки (fingerprint) браузера - это набор методов для идентификации с течением времени конкретного пользовательского агента по его уникальному набору характеристик. Эти характеристики могут включать сведения об использовании базового транспортного протокола, функциональных возможностях и среде сценариев, но особый интерес представляют уникальные характеристики, которые могут передаваться по протоколу HTTP. Отпечатки связаны с приватностью, поскольку они позволяют отслеживать поведение агента пользователя в течение определённого времени ([Bujlow]) без соответствующего контроля, который может быть у пользователя для других форм сбора данных (например, cookie). Многие пользовательские агенты общего назначения (Web-браузеры) имеют средства сокращения своих отпечатков.

Имеется ряд полей заголовков запроса, которые могут передавать серверам достаточно уникальные сведения, позволяющие отслеживать отпечатки. Поле заголовка From наиболее очевидно, хотя его передача предполагается лишь при желании пользователя указать себя. Поля заголовка Cookie намеренно предназначены для возможности повторной идентификации, поэтому проблемы с отпечатками возникают лишь в ситуациях, когда cookie отключены или

ограничены конфигурацией агента пользователя. Поле User-Agent может включать сведения, достаточные для однозначной идентификации конкретного устройства, обычно в сочетании с другими характеристиками, особенно при передаче пользовательским агентом избыточных сведений о системе пользователя или расширениях. Однако упреждающее согласование (параграф 12.1) является источником уникальных сведений, которые меньше всего ожидает пользователь, включая поля заголовка Accept, Accept-Charset, Accept-Encoding, Accept-Language.

Помимо отпечатка, детали поля Accept-Language могут раскрывать сведения, которые пользователь считает приватными. Например, понимание данного языка может быть строго сопоставлено с принадлежностью к определённой этнической группе. Подход к ограничению такого раскрытия состоит в отказе агента пользователя от передачи поля Accept-Language за исключением сайтов, для которых это явно разрешено, возможно, через взаимодействие после обнаружения поля Vary, указывающего, что согласование языка может быть полезным.

В средах с использованием прокси для обеспечения приватности, пользовательские агенты должны быть консервативны при отправке полей заголовка для упреждающего согласования. Пользовательские агенты общего назначения, обеспечивающие широкие возможности настройки полей заголовка, должны информировать пользователя о возможности утраты приватности при указании избыточных деталей. В качестве крайней меры по сохранению приватности прокси может фильтровать поля упреждающего согласования в ретранслируемых запросах.

17.14. Сохранение валидатора

Определённые в этой спецификации валидаторы не предназначены для обеспечения пригодности представления, защиты от вредоносных изменений или обнаружения атак в пути. В лучшем случае они позволяют более эффективно обновлять кэш и оптимизировать одновременную запись при корректно поведении всех участников. В худшем случае условия не выполняются и клиент будет получать отклики, которые не вреднее обмена HTTP без условных запросов.

Тег сущности может злонамеренно применяться для создания рисков приватности. Например, сайт может намеренно создать семантически непригодный тег сущности, уникальный для пользователя или пользовательского агента, передать его в кэшируемом отклике с продолжительным сроком пригодности (freshness), а затем считывать этот тег в последующих условных запросах как средство повторной идентификации пользователя или пользовательского агента. Такой тег идентификации станет постоянным на время, пока агент пользователя сохраняет исходное кэшированное представление. Пользовательские агенты, кэширующие представления, должны обеспечивать очистку или замену кэша при действиях пользователя по сохранению приватности, таких как очистка сохранённых cookie или переход в приватный режим просмотра.

17.15. DoS-атаки с использованием диапазона

Неограниченные запросы с множеством диапазонов создают риск DoS-атак, поскольку усилия для запроса множества перекрывающихся диапазонов одних данных очень малы по сравнению с затратами времени, памяти и пропускной способности при попытке обслуживания запрошенных данных по частям. Серверы должны игнорировать, объединять или отвергать вопиющие запросы диапазонов, такие как запрос больше двух перекрывающихся диапазонов или множества мелких диапазонов в одном наборе, особенно при запросе диапазонов с нарушением порядка без видимых причин. Запросы с несколькими диапазонами не предназначены для предоставления произвольного доступа.

17.16. Вопросы проверки подлинности

Все, что связано с аутентификацией HTTP, относится к вопросам безопасности, поэтому приведённый ниже список не является исчерпывающим. Кроме того, в него включены лишь вопросы безопасности, связанные с аутентификацией в целом без рассмотрения всех возможных соображений для конкретных схем (они должны быть рассмотрены в спецификациях таких схем). Различные организации поддерживают тематические сведения и ссылки на текущие исследования безопасности Web-приложений (например, [OWASP]), включая основные ошибки при реализации и применении схем проверки подлинности, найденные на практике.

17.16.1. Конфиденциальность свидетельств

Схема аутентификации HTTP не задаёт единого механизма для поддержки конфиденциальности свидетельств и в каждой схеме аутентификации задаётся свой способ кодирования свидетельств перед их отправкой. Это обеспечивает гибкость при разработке схем аутентификации, но недостаточно для защиты имеющихся схем, которые сами не имеют защиты конфиденциальности или недостаточно защищены от атак с воспроизведением (replay). Кроме того при ожидании сервером специфичных для каждого пользователя свидетельств обмен такими свидетельствами приведёт к идентификации пользователей даже без раскрытия содержимого самих свидетельств.

В HTTP конфиденциальность значений передаваемых полей зависит от защитных свойств базового транспортного или сеансового соединения. Для служб, зависящих от аутентификации отдельных пользователей требуется организация защищённого соединения до обмена свидетельствами (параграф 4.2.2).

17.16.2. Свидетельства и бездействующие клиенты

Имеющиеся клиенты HTTP и пользовательские агенты обычно сохраняют данные для аутентификации неограниченно долго. В HTTP нет механизма, позволяющего серверу-источнику направлять клиентов на отбрасывание кэшированных свидетельств, поскольку протокол не знает, ка свидетельства получаются и поддерживаются агентом пользователя. Механизм завершения срока действия или отзыва свидетельств можно задать в определении схемы аутентификации. Ниже приведены некоторые ситуации, когда кэширование свидетельств может нарушать модель безопасности приложения.

- После длительного бездействия клиента сервер может снова запросить свидетельства для аутентификации.
- Приложения с индикацией прерывания сессии (например, кнопка logout или commit на странице), после которого серверная сторона узнаёт, что у клиента больше нет причин сохранять свидетельства.

Агентам пользователя, кэширующим свидетельства, настоятельно рекомендуется предоставлять легкодоступный механизм для отбрасывания кэшированных свидетельств под управлением пользователя.

17.16.3. Пространства защиты

Схемы аутентификации, основанные исключительно на механизме realm для создания защищённого пространства, будут раскрывать свидетельства всем ресурсам на сервере-источнике. Клиенты, успешно выполнившие аутентификационные запросы к ресурсу, могут использовать те же аутентификационные свидетельства для других ресурсов сервера-источника. Это позволяет одному ресурсу воспринимать аутентификационные свидетельства других ресурсов. Это особенно важно при размещении на сервере-источнике ресурсов разных сторон (организаций) с одним источником (origin, параграф 11.5). Возможные схемы смягчения проблемы включают ограничение прямого доступа к аутентификационным свидетельствам (недоступность содержимого поля заголовка Authorization в запросе) и разделение пространства защиты путём использования разных имён хостов (или номеров портоа) для каждой стороны.

17.16.4. Дополнительные поля отклика

Добавление сведений в отклики, передаваемые по незащищённому каналу, может влиять на приватность и безопасность. Наличие полей заголовка Authentication-Info и Proxy-Authentication-Info само по себе указывает на применение аутентификации HTTP. Дополнительные сведения могут раскрывать параметры конкретной схемы аутентификации и это должно учитываться при создании таких схем.

18. Взаимодействие с IANA

Контролёр изменений для описанных ниже регистраций является IETF (iesg@ietf.org) - Internet Engineering Task Force.

18.1. Регистрация схем URI

Агентство IANA обновило реестр Uniform Resource Identifier (URI) Schemes [BCP35] (https://www.iana.org/assignments/uri-schemes/) включив в него постоянные (permanent) схемы из таблицы 2 в параграфе 4.2.

18.2. Регистрация методов

Агентство IANA обновило реестр Hypertext Transfer Protocol (HTTP) Method Registry (https://www.iana.org/assignments/http-methods) с процедурой регистрации, указанной в параграфе 16.1.1, и именами методов, указанными в таблице 7.

Таблица 7.

Метод	*Безопасный*	*Идемпотентный*	*Параграф*
CONNECT	нет	нет	9.3.6
DELETE	нет	да	9.3.5
GET	да	да	9.3.1
HEAD	да	да	9.3.2
OPTIONS	да	да	9.3.7
POST	нет	нет	9.3.3
PUT	нет	да	9.3.4
TRACE	да	да	9.3.8
*	нет	нет	18.2

Имя метода * является зарезервированным, поскольку применение метода с именем * будет конфликтовать с использованием этого символа как шаблона для некоторых методов (например, Access-Control-Request-Method).

18.3. Регистрация кодов статуса

Агентство IANA обновило реестр Hypertext Transfer Protocol (HTTP) Status Code Registry (https://www.iana.org/assignments/http-status-codes) с процедурой регистрации, указанной в параграфе 16.2.1, и именами методов, указанными в таблице 8.

Таблица 8.

Значение	*Описание*	*Параграф*
100	Continue	15.2.1
101	Switching Protocols	15.2.2
200	OK	15.3.1
201	Created	15.3.2
202	Accepted	15.3.3
203	Non-Authoritative Information	15.3.4
204	No Content	15.3.5
205	Reset Content	15.3.6
206	Partial Content	15.3.7
300	Multiple Choices	15.4.1
301	Moved Permanently	15.4.2
302	Found	15.4.3
303	See Other	15.4.4
304	Not Modified	15.4.5
305	Use Proxy	15.4.6
306	(Unused)	15.4.7
307	Temporary Redirect	15.4.8
308	Permanent Redirect	15.4.9
400	Bad Request	15.5.1
401	Unauthorized	15.5.2
402	Payment Required	15.5.3
403	Forbidden	15.5.4
404	Not Found	15.5.5
405	Method Not Allowed	15.5.6
406	Not Acceptable	15.5.7

407	Proxy Authentication Required	15.5.8
408	Request Timeout	15.5.9
409	Conflict	15.5.10
410	Gone	15.5.11
411	Length Required	15.5.12
412	Precondition Failed	15.5.13
413	Content Too Large	15.5.14
414	URI Too Long	15.5.15
415	Unsupported Media Type	15.5.16
416	Range Not Satisfiable	15.5.17
417	Expectation Failed	15.5.18
418	(Unused)	15.5.19
421	Misdirected Request	15.5.20
422	Unprocessable Content	15.5.21
426	Upgrade Required	15.5.22
500	Internal Server Error	15.6.1
501	Not Implemented	15.6.2
502	Bad Gateway	15.6.3
503	Service Unavailable	15.6.4
504	Gateway Timeout	15.6.5
505	HTTP Version Not Supported	15.6.6

18.4. Регистрация имён полей

Эта спецификация обновляет связанные с HTTP аспекты имеющихся процедур регистрации для полей заголовков сообщения, определённых в [RFC3864]. Старые процедуры заменены новыми и определения полей HTTP перенесены в отдельный реестр IANA Hypertext Transfer Protocol (HTTP) Field Name Registry, как указано в параграфе 16.3.1. Агентство IANA перенесло все записи из реестров Permanent Message Header Field Names и Provisional Message Header Field Names (https://www.iana.org/assignments/message-headers/) с протоколом http в новый реестр с внесением указанных ниже изменений.

1. Поле Applicable Protocol исключено.
2. Записи со статусом standard, experimental, reserved, informational получили статус permanent.
3. Временные записи без назначенного статуса получили статус provisional.
4. Постоянные записи без назначенного статуса (после подтверждения его отсутствия в регистрационном документе) получили статус provisional. Эксперты могут принять решение об обновлении таких записей, если есть основания для установки иного статуса.

Агентство IANA анонсировало реестры Permanent Message Header Field Names и Provisional Message Header Field Names с указанным ниже примечанием для указания переноса регистрации имён полей HTTP.

Примечание. Регистрации имён полей HTTP перенесены в https://www.iana.org/assignments/http-fields в соответствии с [RFC9110].

Агентство IANA обновило Hypertext Transfer Protocol (HTTP) Field Name Registry, включив имена полей из таблицы 9.

Таблица 9.

Имя поля	***Статус***	***Параграф***	***Комментарии***
Accept	permanent	12.5.1	
Accept-Charset	deprecated	12.5.2	
Accept-Encoding	permanent	12.5.3	
Accept-Language	permanent	12.5.4	
Accept-Ranges	permanent	14.3	
Allow	permanent	10.2.1	
Authentication-Info	permanent	11.6.3	
Authorization	permanent	11.6.2	
Connection	permanent	7.6.1	
Content-Encoding	permanent	8.4	
Content-Language	permanent	8.5	
Content-Length	permanent	8.6	
Content-Location	permanent	8.7	
Content-Range	permanent	14.4	
Content-Type	permanent	8.3	
Date	permanent	6.6.1	
ETag	permanent	8.8.3	
Expect	permanent	10.1.1	
From	permanent	10.1.2	
Host	permanent	7.2	
If-Match	permanent	13.1.1	
If-Modified-Since	permanent	13.1.3	
If-None-Match	permanent	13.1.2	
If-Range	permanent	13.1.5	
If-Unmodified-Since	permanent	13.1.4	
Last-Modified	permanent	8.8.2	
Location	permanent	10.2.2	
Max-Forwards	permanent	7.6.2	
Proxy-Authenticate	permanent	11.7.1	
Proxy-Authentication-Info	permanent	11.7.3	

Proxy-Authorization	permanent	11.7.2
Range	permanent	14.2
Referer	permanent	10.1.3
Retry-After	permanent	10.2.3
Server	permanent	10.2.4
TE	permanent	10.1.4
Trailer	permanent	6.6.2
Upgrade	permanent	7.8
User-Agent	permanent	10.1.5
Vary	permanent	12.5.5
Via	permanent	7.6.3
WWW-Authenticate	permanent	11.6.1
*	permanent	12.5.5 (резерв)

Имя поля * зарезервировано, поскольку использование такого имени для полей заголовков HTTP может вызывать конфликты с особой семантикой поля заголовка Vary (параграф 12.5.5).

Агентство IANA указало запись Content-MD5 в новом реестре как устаревшую (obsoleted) со ссылкой на параграф 14.15 в [RFC2616] (определение поля заголовка) и Приложение B к [RFC7231] (исключение поля из новой спецификации).

18.5. Регистрация схем аутентификации

Агентство IANA обновило реестр Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry (https://www.iana.org/assignments/http-authschemes), указав процедуру регистрации из параграфа 16.4.1. Этот документ не определяет схем аутентификации.

18.6. Регистрация кодирования содержимого

Агентство IANA обновило реестр HTTP Content Coding Registry (https://www.iana.org/assignments/http-parameters/) с указанием процедуры регистрации из параграфа 16.6.1 и именами кодировок, указанными в таблице 10.

Таблица 10.

Имя	***Описание***	***Параграф***
compress	Формат данных UNIX compress [Welch]	8.4.1.1
deflate	Сжатые данные deflate ([RFC1951]) внутри формата zlib ([RFC1950])	8.4.1.2
gzip	Формат файлов GZIP [RFC1952]	8.4.1.3
identity	Резерв	12.5.3
x-compress	Отменено (псевдоним для compress)	8.4.1.1
x-gzip	Отменено (псевдоним для gzip)	8.4.1.3

18.7. Регистрация Range Unit

Агентство IANA обновило реестр HTTP Range Unit Registry (https://www.iana.org/assignments/http-parameters/) с указанием процедуры регистрации из параграфа 16.5.1 и именами единиц диапазона, указанными в таблице 11.

Таблица 11.

Имя	***Описание***	***Параграф***
bytes	Диапазон октетов	14.1.2
none	Зарезервировано как ключевое слово для указания отсутствия поддержки запросов диапазона	14.3

18.8. Регистрация типов носителей

Агентство IANA обновило реестр Media Types (https://www.iana.org/assignments/media-types) с указанием сведений о регистрации из параграфа 14.6 для типа носителя multipart/byteranges. Обновлено также примечание о параметрах q с включением ссылки на параграф 12.5.1 этого документа.

18.9. Регистрация портов

Агентство IANA обновило реестр Service Name and Transport Protocol Port Number Registry (https://www.iana.org/assignments/service-names-port-numbers/) для служб на портах 80 и 443 протоколов UDP или TCP

1. этот документ указан в качестве ссылки (Reference);
2. для не ещё заданного поля Assignee установлено значение IESG, для Contact - IETF_Chair.

18.10. Регистрация маркера обновления

Агентство IANA обновило реестр Hypertext Transfer Protocol (HTTP) Upgrade Token Registry (https://www.iana.org/assignments/http-upgrade-tokens) с указанием процедуры регистрации из параграфа 16.7 и имени маркера из таблицы 12.

Таблица 12.

Имя	***Описание***	***Ожидаемые маркеры версии***	***Параграф***
HTTP	Hypertext Transfer Protocol	Любое значение DIGIT.DIGIT (например, 2.0)	2.5

19. Литература

19.1. 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>.

[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>.

[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>.

[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>.

[RFC4647] Phillips, A., Ed. and M. Davis, Ed., "Matching of Language Tags", BCP 47, RFC 4647, DOI 10.17487/RFC4647, September 2006, <https://www.rfc-editor.org/info/rfc4647>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

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

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[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>.

[TCP] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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/>.

19.2. Дополнительная литература

[ALTSVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[BCP13] Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, December 2005.
Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013. <https://www.rfc-editor.org/info/bcp13>

[BCP178] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, June 2012. <https://www.rfc-editor.org/info/bcp178>

[BCP35] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, June 2015. <https://www.rfc-editor.org/info/bcp35>

[BREACH] Gluck, Y., Harris, N., and A. Prado, "BREACH: Reviving the CRIME Attack", July 2013, <http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf>.

[Bujlow] Bujlow, T., Carela-Español, V., Solé-Pareta, J., and P. Barlet-Ros, "A Survey on Web Tracking: Mechanisms, Implications, and Defenses", In Proceedings of the IEEE 105(8), DOI 10.1109/JPROC.2016.2637878, August 2017, <https://doi.org/10.1109/JPROC.2016.2637878>.

[COOKIE] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[Err1912] RFC Errata, Erratum ID 1912, RFC 2978, <https://www.rfc-editor.org/errata/eid1912>.

[Err5433] RFC Errata, Erratum ID 5433, RFC 2978, <https://www.rfc-editor.org/errata/eid5433>.

[Georgiev] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software", In Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12), pp. 38-49, DOI 10.1145/2382196.2382204, October 2012, <https://doi.org/10.1145/2382196.2382204>.

[HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/info/rfc7541>.

[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>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.

[Kri2001] Kristol, D., "HTTP Cookies: Standards, Privacy, and Politics", ACM Transactions on Internet Technology 1(2), November 2001, <http://arxiv.org/abs/cs.SE/0105018>.

[OWASP] The Open Web Application Security Project, <https://www.owasp.org/>.

[REST] Fielding, R.T., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Dissertation, University of California, Irvine, September 2000, <https://roy.gbiv.com/pubs/dissertation/top.htm>.

[RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, DOI 10.17487/RFC1919, March 1996, <https://www.rfc-editor.org/info/rfc1919>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <https://www.rfc-editor.org/info/rfc2047>.

[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>.

[RFC2145] Mogul, J. C., Fielding, R., Gettys, J., and H. Frystyk, "Use and Interpretation of HTTP Version Numbers", RFC 2145, DOI 10.17487/RFC2145, May 1997, <https://www.rfc-editor.org/info/rfc2145>.

[RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, DOI 10.17487/RFC2295, March 1998, <https://www.rfc-editor.org/info/rfc2295>.

[RFC2324] Masinter, L., "Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)", RFC 2324, DOI 10.17487/RFC2324, 1 April 1998, <https://www.rfc-editor.org/info/rfc2324>.

[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>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <https://www.rfc-editor.org/info/rfc2617>.

[RFC2774] Nielsen, H., Leach, P., and S. Lawrence, "An HTTP Extension Framework", RFC 2774, DOI 10.17487/RFC2774, February 2000, <https://www.rfc-editor.org/info/rfc2774>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978, October 2000, <https://www.rfc-editor.org/info/rfc2978>.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, DOI 10.17487/RFC3040, January 2001, <https://www.rfc-editor.org/info/rfc3040>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

[RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, DOI 10.17487/RFC3875, October 2004, <https://www.rfc-editor.org/info/rfc3875>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, DOI 10.17487/RFC4559, June 2006, <https://www.rfc-editor.org/info/rfc4559>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/info/rfc5789>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/info/rfc6585>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <https://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[RFC7538] Reschke, J., "The Hypertext Transfer Protocol Status Code 308 (Permanent Redirect)", RFC 7538, DOI 10.17487/RFC7538, April 2015, <https://www.rfc-editor.org/info/rfc7538>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7578] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 7578, DOI 10.17487/RFC7578, July 2015, <https://www.rfc-editor.org/info/rfc7578>.

[RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <https://www.rfc-editor.org/info/rfc7615>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <https://www.rfc-editor.org/info/rfc7616>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <https://www.rfc-editor.org/info/rfc7617>.

[RFC7694] Reschke, J., "Hypertext Transfer Protocol (HTTP) Client-Initiated Content-Encoding", RFC 7694, DOI 10.17487/RFC7694, November 2015, <https://www.rfc-editor.org/info/rfc7694>.

[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>.

[RFC8187] Reschke, J., "Indicating Character Encoding and Language for HTTP Header Field Parameters", RFC 8187, DOI 10.17487/RFC8187, September 2017, <https://www.rfc-editor.org/info/rfc8187>.

[RFC8246] McManus, P., "HTTP Immutable Responses", RFC 8246, DOI 10.17487/RFC8246, September 2017, <https://www.rfc-editor.org/info/rfc8246>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[RFC8336] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", RFC 8336, DOI 10.17487/RFC8336, March 2018, <https://www.rfc-editor.org/info/rfc8336>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8941] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

[Sniffing] WHATWG, "MIME Sniffing", <https://mimesniff.spec.whatwg.org>.

[WEBDAV] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <https://www.rfc-editor.org/info/rfc4918>.

Приложение A. Сводка ABNF

Приведённая ниже сводка ABNF содержит правила, преобразованные в соответствии с параграфом 5.6.1.

```
Accept = [ ( media-range [ weight ] ) *( OWS "," OWS ( media-range [
 weight ] ) ) ]
Accept-Charset = [ ( ( token / "*" ) [ weight ] ) *( OWS "," OWS ( (
 token / "*" ) [ weight ] ) ) ]
Accept-Encoding = [ ( codings [ weight ] ) *( OWS "," OWS ( codings [
 weight ] ) ) ]
Accept-Language = [ ( language-range [ weight ] ) *( OWS "," OWS (
 language-range [ weight ] ) ) ]
Accept-Ranges = acceptable-ranges
Allow = [ method *( OWS "," OWS method ) ]
Authentication-Info = [ auth-param *( OWS "," OWS auth-param ) ]
Authorization = credentials

BWS = OWS
```

```
Connection = [ connection-option *( OWS "," OWS connection-option ) ]
Content-Encoding = [ content-coding *( OWS "," OWS content-coding ) ]
Content-Language = [ language-tag *( OWS "," OWS language-tag ) ]
Content-Length = 1*DIGIT
Content-Location = absolute-URI / partial-URI
Content-Range = range-unit SP ( range-resp / unsatisfied-range )
Content-Type = media-type

Date = HTTP-date

ETag = entity-tag
Expect = [ expectation *( OWS "," OWS expectation ) ]

From = mailbox

GMT = %x47.4D.54 ; GMT

HTTP-date = IMF-fixdate / obs-date
Host = uri-host [ ":" port ]

IMF-fixdate = day-name "," SP date1 SP time-of-day SP GMT
If-Match = "*" / [ entity-tag *( OWS "," OWS entity-tag ) ]
If-Modified-Since = HTTP-date
If-None-Match = "*" / [ entity-tag *( OWS "," OWS entity-tag ) ]
If-Range = entity-tag / HTTP-date
If-Unmodified-Since = HTTP-date

Last-Modified = HTTP-date
Location = URI-reference

Max-Forwards = 1*DIGIT

OWS = *( SP / HTAB )

Proxy-Authenticate = [ challenge *( OWS "," OWS challenge ) ]
Proxy-Authentication-Info = [ auth-param *( OWS "," OWS auth-param )
 ]
Proxy-Authorization = credentials

RWS = 1*( SP / HTAB )
Range = ranges-specifier
Referer = absolute-URI / partial-URI
Retry-After = HTTP-date / delay-seconds

Server = product *( RWS ( product / comment ) )

TE = [ t-codings *( OWS "," OWS t-codings ) ]
Trailer = [ field-name *( OWS "," OWS field-name ) ]

URI-reference = <URI-reference, см. [URI], параграф 4.1>
Upgrade = [ protocol *( OWS "," OWS protocol ) ]
User-Agent = product *( RWS ( product / comment ) )

Vary = [ ( "*" / field-name ) *( OWS "," OWS ( "*" / field-name ) ) ]
Via = [ ( received-protocol RWS received-by [ RWS comment ] ) *( OWS
 "," OWS ( received-protocol RWS received-by [ RWS comment ] ) ) ]

WWW-Authenticate = [ challenge *( OWS "," OWS challenge ) ]

absolute-URI = <absolute-URI, см. [URI], параграф 4.3>
absolute-path = 1*( "/" segment )
acceptable-ranges = range-unit *( OWS "," OWS range-unit )
asctime-date = day-name SP date3 SP time-of-day SP year
auth-param = token BWS "=" BWS ( token / quoted-string )
auth-scheme = token
authority = <authority, см. [URI], параграф 3.2>

challenge = auth-scheme [ 1*SP ( token68 / [ auth-param *( OWS ","
 OWS auth-param ) ] ) ]
codings = content-coding / "identity" / "*"
comment = "(" *( ctext / quoted-pair / comment ) ")"
complete-length = 1*DIGIT
connection-option = token
content-coding = token
credentials = auth-scheme [ 1*SP ( token68 / [ auth-param *( OWS ","
 OWS auth-param ) ] ) ]
ctext = HTAB / SP / %x21-27 ; '!'-'''
 / %x2A-5B ; '*'-'['
 / %x5D-7E ; ']'-'~'
 / obs-text

date1 = day SP month SP year
date2 = day "-" month "-" 2DIGIT
date3 = month SP ( 2DIGIT / ( SP DIGIT ) )
day = 2DIGIT
```

```
day-name = %x4D.6F.6E ; Пнд
 / %x54.75.65 ; Втр
 / %x57.65.64 ; Срд
 / %x54.68.75 ; Чтв
 / %x46.72.69 ; Птн
 / %x53.61.74 ; Сбт
 / %x53.75.6E ; Вск
day-name-l = %x4D.6F.6E.64.61.79 ; Понедельник
 / %x54.75.65.73.64.61.79 ; Вторник
 / %x57.65.64.6E.65.73.64.61.79 ; Среда
 / %x54.68.75.72.73.64.61.79 ; Четверг
 / %x46.72.69.64.61.79 ; Пятница
 / %x53.61.74.75.72.64.61.79 ; Суббота
 / %x53.75.6E.64.61.79 ; Воскресенье
delay-seconds = 1*DIGIT

entity-tag = [ weak ] opaque-tag
etagc = "!" / %x23-7E ; '#'-'~'
 / obs-text
expectation = token [ "=" ( token / quoted-string ) parameters ]

field-content = field-vchar [ 1*( SP / HTAB / field-vchar )
 field-vchar ]
field-name = token
field-value = *field-content
field-vchar = VCHAR / obs-text
first-pos = 1*DIGIT

hour = 2DIGIT
http-URI = "http://" authority path-abempty [ "?" query ]
https-URI = "https://" authority path-abempty [ "?" query ]

incl-range = first-pos "-" last-pos
int-range = first-pos "-" [ last-pos ]

language-range = <language-range, см. [RFC4647], параграф 2.1>
language-tag = <Language-Tag, см. [RFC5646], параграф 2.1>
last-pos = 1*DIGIT

mailbox = <mailbox, см. [RFC5322], Section 3.4>
media-range = ( "*/*" / ( type "/*" ) / ( type "/" subtype ) )
 parameters
media-type = type "/" subtype parameters
method = token
minute = 2DIGIT
month = %x4A.61.6E ; Янв
 / %x46.65.62 ; Фев
 / %x4D.61.72 ; Мар
 / %x41.70.72 ; Апр
 / %x4D.61.79 ; Май
 / %x4A.75.6E ; Июн
 / %x4A.75.6C ; Июл
 / %x41.75.67 ; Авг
 / %x53.65.70 ; Сен
 / %x4F.63.74 ; Окт
 / %x4E.6F.76 ; Ноя
 / %x44.65.63 ; Дек

obs-date = rfc850-date / asctime-date
obs-text = %x80-FF
opaque-tag = DQUOTE *etagc DQUOTE
other-range = 1*( %x21-2B ; '!'-'+'
 / %x2D-7E ; '-'-'~' )

parameter = parameter-name "=" parameter-value
parameter-name = token
parameter-value = ( token / quoted-string )
parameters = *( OWS ";" OWS [ parameter ] )
partial-URI = relative-part [ "?" query ]
path-abempty = <path-abempty, см. [URI], параграф 3.3>
port = <port, см. [URI], параграф 3.2.3>
product = token [ "/" product-version ]
product-version = token
protocol = protocol-name [ "/" protocol-version ]
protocol-name = token
protocol-version = token
pseudonym = token

qdtext = HTAB / SP / "!" / %x23-5B ; '#'-'['
 / %x5D-7E ; ']'-'~'
 / obs-text
query = <query, см. [URI], параграф 3.4>
quoted-pair = "\" ( HTAB / SP / VCHAR / obs-text )
quoted-string = DQUOTE *( qdtext / quoted-pair ) DQUOTE
qvalue = ( "0" [ "." *3DIGIT ] ) / ( "1" [ "." *3"0" ] )
```

```
range-resp = incl-range "/" ( complete-length / "*" )
range-set = range-spec *( OWS "," OWS range-spec )
range-spec = int-range / suffix-range / other-range
range-unit = token
ranges-specifier = range-unit "=" range-set
received-by = pseudonym [ ":" port ]
received-protocol = [ protocol-name "/" ] protocol-version
relative-part = <relative-part, см. [URI], параграф 4.2>
rfc850-date = day-name-l "," SP date2 SP time-of-day SP GMT

second = 2DIGIT
segment = <segment, см. [URI], параграф 3.3>
subtype = token
suffix-length = 1*DIGIT
suffix-range = "-" suffix-length

t-codings = "trailers" / ( transfer-coding [ weight ] )
tchar = "!" / "#" / "$" / "%" / "&" / "'" / "*" / "+" / "-" / "." /
 "^" / "_" / "`" / "|" / "~" / DIGIT / ALPHA
time-of-day = hour ":" minute ":" second
token = 1*tchar
token68 = 1*( ALPHA / DIGIT / "-" / "." / "_" / "~" / "+" / "/" )
 *"="
transfer-coding = token *( OWS ";" OWS transfer-parameter )
transfer-parameter = token BWS "=" BWS ( token / quoted-string )
type = token

unsatisfied-range = "*/" complete-length
uri-host = <host, см. [URI], параграф 3.2.2>

weak = %x57.2F ; W/
weight = OWS ";" OWS "q=" qvalue

year = 4DIGIT
```

Приложение B. Отличия от предшествующих RFC

B.1. Отличия от RFC 2818

Отменено использование CN-ID[1].

B.2. Отличия от RFC 7230

Сюда перенесены разделы, описывающие цели разработки HTTP, историю, архитектуру, критерии соответствия, версии протокола, URI, маршрутизацию сообщений и поля заголовков.

Требование семантического соответствия заменено разрешением игнорировать или обходить характерные для реализаций отказы(параграф 2.2).

Описание источника и полномочного доступа к серверам-источникам расширено для URI http и https с учётом дополнительных служб и защищённых соединений, не обязательно основанных на TCP (параграфы 4.2.1, 4.2.2, 4.3.1, 7.3.3)

Добавлены явные требования для проверки семантики схемы URI цели и отклонения запросов, не соответствующих требованиям (параграф 7.4).

Параметры для типа и диапазона носителей, а также ожиданий могут быть пустыми с одним или несколькими символами точки с запятой в конце (параграф 5.6.6).

Под значением поля понимается значение после объединения нескольких строк поля через запятые (это наиболее распространённое использование). Для одной строки заголовка применяется термин «значение строки поля» (field line value, параграф 6.3).

Семантика полей трейлера выходит за рамки блочного (chunk) транспортного кодирования. Использование трейлерных полей ограничено дополнительно, чтобы разрешить генерацию в качестве таких полей лишь случаями, когда отправитель знает, что для поля определено такое использование, и слияние в раздел заголовка лишь случаями, когда получатель знает, что определение соответствующего поля описывает такое слияние. В остальных случаях реализации рекомендуется сохранять поля трейлеров отдельно или отбрасывать их вместо слияния (параграф 6.5.1).

Приоритет абсолютной формы URI запроса перед полем заголовка Host на сервере-источнике сделан явным для согласования с обработкой на прокси (параграф 7.2).

Определение грамматики поля Via received-by было расширено в RFC 7230 в связи с изменениями грамматики URI для хостов [URI], которые нежелательны для Via. Для простоты из received-by удалён элемент uri-host, поскольку он может быть охвачен имеющейся грамматикой для псевдонимов. В частности, это изменение удаляет запятую из разрешённых символов имени хоста в received-by (параграф 7.6.3).

B.3. Отличия от RFC 7231

Рекомендованы минимальные размеры URI для поддержки реализациями (параграф 4.1).

Добавлено разъяснение, что CR и NUL в значениях полей отвергаются или заменяются SP, а начальные или завершающие пробельные символы должны исключаться из значений полей перед их восприятием (параграф 5.5).

[1] В оригинале указано отсутствие изменений, см. https://www.rfc-editor.org/errata/eid7105. *Прим. перев.*

Параметры и ожидания типа или диапазона типов носителя могут быть пустыми с одним или несколькими символами точки с запятой в конце (параграф 5.6.6).

Введён абстрактный тип данных для сообщений HTTP, определяющий компоненты сообщения и их семантику как абстракцию для разных версий HTTP вместо конкретных форм HTTP/1.1 изи [HTTP/1.1] и отражающий содержимое после синтаксического анализа сообщения. Это упрощает отделение требований к содержимому (что передаётся) от требований к синтаксису (как передавать) и позволяет избежать ограничений ранних версий протокола в будущих HTTP (раздел 6).

Термины payload и payload body заменены на content, для согласования с их использованием в других местах (например, в именах полей) и предотвращения путаницы с содержимым кадров HTTP/2 и HTTP/3 (параграф 6.4).

Термин effective request URI заменён на target URI (параграф 7.1).

Смягчены ограничения для повторных попыток клиента в соответствии с поведением реализаций (параграф 9.2.2).

Уточнено, что тела запросов GET, HEAD, DELETE не являются совместимыми (параграфы 9.3.1, 9.3.2, 9.3.5).

Разрешено использовать поле Content-Range (параграф 14.4) как модификатор запроса PUT (параграф 9.3.4).

Из описания метода OPTIONS исключено лишнее требование об установке Content-Length (параграф 9.3.7).

Исключено нормативное требование об использовании носителя типа message/http в откликах TRACE (параграф 9.3.8).

Для совместимости RFC 2616 восстановлена списочная грамматика Expect (параграф 10.1.1).

Разрешены Accept и Accept-Encoding (добавлен [RFC7694]) в сообщениях с откликами (параграф 12.3).

Исключены параметры восприятия (ABNF accept-params и accept-ext) из определения поля Accept (параграф 12.5.1).

Поле Accept-Charset признано устаревшим (параграф 12.5.2).

Уточнена семантика * в поле заголовка Vary при наличии других значений (параграф 12.5.5).

При сравнении единиц диапазона регистр символов не учитывается (параграф 14.1).

Использование поля Accept-Ranges разрешено не только серверам-источникам (параграф 14.3).

Уточнён процесс создания перенаправленного запроса (параграф 15.4).

Добавлен код статуса 308 (определён в [RFC7538]), чтобы он был ближе к кодам 301, 302, 307 (параграф 15.4.9).

Добавлен код статуса 421 (определён в параграфе 9.1.2 [RFC7540]) по причине распространённости его применения. Этот код больше не считается эвристически кэшируемым, поскольку отклик специфичен для соединения, а не целевого ресурса (параграф 15.5.20).

Добавлен код статуса 422 (определён в параграфе 11.2 [WEBDAV]) по причине распространённости его применения (параграф 15.5.21).

B.4. Отличия от RFC 7232

Предыдущие версии HTTP вносили произвольное ограничение (60 секунд) на проверку, является ли Last-Modified строгим валидатором для обнаружения того, что значения Date и Last-Modified получены от разных часов или в разные моменты при подготовке отклика. Эта спецификация смягчает требование для обеспечения разумной свободы действий (параграф 8.8.2.2).

Исключено требование не передавать валидаторы If-Match и If-Unmodified-Since в откликах 2xx при отрицательном результате проверки, поскольку запрос уже был применён (параграфы 13.1.1 и 13.1.4)

Уточнено, что If-Unmodified-Since не применяется к ресурсу, не понимающему концепции времени изменения (параграф 13.1.4).

Предварительные условия разрешено проверять до обработки содержимого запроса без ожидания его успеха (параграф 13.2).

B.5. Отличия от RFC 7233

Переработана грамматика range-unit и ranges-specifier для упрощения и сокращения искусственных различий между батовыми и другими (расширения) единицами диапазонов. Удалено дублирование грамматики other-range-unit путём определения единиц диапазона как маркеров и размещения расширений в области действия range-spec (other-range). Это устраняет неоднозначность роли синтаксиса списков (запятые) во всех наборах диапазонов, включая расширенные единицы, для индикации range-set с множеством диапазонов. Перенос грамматики расширения в спецификатор диапазона позволяет также отдельно задавать протокол, специфичный для диапазона байтов.

Стало возможным определение обработки Range в методах расширения (параграф 14.2).

Описано применение поля заголовка Content-Range (параграф 14.4) как модификатора запроса для выполнения частичного PUT (параграф 14.5).

B.6. Отличия от RFC 7235

Нет.

B.7. Отличия от RFC 7538

Нет.

B.8. Отличия от RFC 7615

Нет.

B.9. Отличия от RFC 7694

Эта спецификация включает расширения из [RFC7694], но без примеров и рекомендаций по внедрению.

Благодарности

Помимо текущих редакторов, следует отметить вклад в ранние работы по HTTP и базовые спецификации протокола Marc Andreessen, Tim Berners-Lee, Robert Cailliau, Daniel W. Connolly, Bob Denny, John Franks, Jim Gettys, Jean-François Groff, Phillip M. Hallam-Baker, Koen Holtman, Jeffery L. Hostetler, Shel Kaphan, Dave Kristol, Yves Lafon, Scott D. Lawrence, Paul J. Leach, Håkon W. Lie, Ari Luotonen, Larry Masinter, Rob McCool, Jeffrey C. Mogul, Lou Montulli, David Morris, Henrik Frystyk Nielsen, Dave Raggett, Eric Rescorla, Tony Sanders, Lawrence C. Stewart, Marc VanHeyningen, Steve Zilles.

Этот документ основан на прошлых спецификациях HTTP, включая [HTTP/1.0], [RFC2068], [RFC2145], [RFC2616], [RFC2617], [RFC2818], [RFC7230], [RFC7231], [RFC7232], [RFC7233], [RFC7234], [RFC7235]. Включённые в эти документы благодарности сохраняют силу.

После 2014 г помогли улучшить эту спецификацию своими отчётами об ошибках, правильными вопросами, написанием или рецензированием текста и решением проблем Alan Egerton, Alex Rousskov, Amichai Rothman, Amos Jeffries, Anders Kaseorg, Andreas Gebhardt, Anne van Kesteren, Armin Abfalterer, Aron Duby, Asanka Herath, Asbjørn Ulsberg, Asta Olofsson, Attila Gulyas, Austin Wright, Barry Pollard, Ben Burkert, Benjamin Kaduk, Björn Höhrmann, Brad Fitzpatrick, Chris Pacejo, Colin Bendell, Cory Benfield, Cory Nelson, Daisuke Miyakawa, Dale Worley, Daniel Stenberg, Danil Suits, David Benjamin, David Matson, David Schinazi, Дилян Палаузов (Dilyan Palauzov), Eric Anderson, Eric Rescorla, Éric Vyncke, Erik Kline, Erwin Pe, Etan Kissling, Evert Pot, Evgeny Vrublevsky, Florian Best, Francesca Palombini, Igor Lubashev, James Callahan, James Peach, Jeffrey Yasskin, Kalin Gyokov, Kannan Goundan, 奥 一穂 (Kazuho Oku), Ken Murchison, Krzysztof Maczyński, Lars Eggert, Lucas Pardue, Martin Duke, Martin Dürst, Martin Thomson, Martynas Jusevičius, Matt Menke, Matthias Pigulla, Mattias Grenfeldt, Michael Osipov, Mike Bishop, Mike Pennisi, Mike Taylor, Mike West, Mohit Sethi, Murray Kucherawy, Nathaniel J. Smith, Nicholas Hurley, Nikita Prokhorov, Patrick McManus, Piotr Sikora, Poul-Henning Kamp, Rick van Rein, Robert Wilton, Roberto Polli, Roman Danyliw, Samuel Williams, Semyon Kholodnov, Simon Pieters, Simon Schüppel, Stefan Eissing, Taylor Hunt, Todd Greer, Tommy Pauly, Vasiliy Faronov, Vladimir Lashchev, Wenbo Zhu, William A. Rowe Jr., Willy Tarreau, Xingwei Liu, Yishuai Li, Zaheduzzaman Sarker.

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