互联网工程任务组（IETF）
注解：7230
废止：2145、2616
更新：2817，2818
类别：标准跟踪
ISSN（国际标准连续出版物编号）：2070-1721

R. Fielding编辑。请求
奥多比
J。Reschke编。
绿色字节（注：此处“greenbytes”可能为特定术语或缩写，具体含义需根据上下文或相关文档确定）

2014年6月

超文本传输协议（HTTP/1.1）：消息语法与路由摘要

超文本传输协议（HTTP）是一种无状态的应用层协议，用于分布式、协作式的超文本信息系统。 本文档概述了HTTP架构及其相关术语，定义了“http”和“https”统一资源标识符（URI）方案，规定了HTTP/1.1消息语法和解析要求，并描述了实现过程中相关的安全问题。

本备忘录的状态

这是一份互联网标准跟踪文档。

本文件是互联网工程任务组（IETF）的成果。 它代表了IETF社区的共识。 本文件已接受公众审查，并已获得互联网工程指导小组（IESG）的批准发布。 有关互联网标准的更多信息，请参阅RFC 5741的第2节。

有关本文档的当前状态、任何勘误以及如何提供反馈的信息，请访问http://www.rfc-editor.org/info/rfc7230。

版权声明

版权所有 (c) 2014 IETF Trust 以及本文件所列作者。保留所有权利。

本文件受BCP 78和IETF信托机构关于IETF文件的法律条款（http://trustee.ietf.org/license-info）的约束，该法律条款自本文件发布之日起生效。 请仔细阅读这些文件，因为它们描述了您对本文件的权利和限制。 从本文件中提取的代码组件必须包含IETF信托机构法律条款第4.e节中所述的简化BSD许可证文本，并且如简化BSD许可证所述，这些代码组件的提供不附带任何担保。

本文档可能包含2008年11月10日之前发布或公开的IETF文档或IETF贡献材料中的内容。 部分材料的版权控制者可能未授予IETF信托在IETF标准流程之外修改此类材料的权利。未经此类材料版权控制者的充分许可，不得在IETF标准流程之外修改本文档，也不得在IETF标准流程之外创建其衍生作品，除非是为了将其格式化以作为RFC发布或翻译成英语以外的语言。

目录

1. 引言

超文本传输协议（HTTP）是一种无状态的应用层请求/响应协议，它使用可扩展的语义和自描述的消息载荷，以实现与基于网络的超文本信息系统的灵活交互。 本文件是一系列共同构成HTTP/1.1规范的文档中的第一份：

1. “消息语法与路由”（本文档）
2. “语义和内容”[RFC7231]
3. “条件请求”[RFC7232]
4. “范围请求” [RFC7233]
5. “缓存”[RFC7234]
6. “认证”[RFC7235]

本HTTP/1.1规范取代了RFC 2616和RFC 2145（关于HTTP版本控制）。 本规范还更新了先前在RFC 2817中定义的用于建立隧道的CONNECT方法的使用，并定义了先前在RFC 2818中非正式描述的“https”URI方案。

HTTP是信息系统的一种通用接口协议。 它旨在通过向客户端呈现一个统一的接口来隐藏服务实现方式的细节，该接口与所提供的资源类型无关。 同样，服务器无需了解每个客户端的目的：HTTP请求可以被视为独立的，而不必与特定类型的客户端或预定的应用步骤序列相关联。 其结果是，该协议可在多种不同情境下有效使用，并且其实现可以随时间独立发展。

HTTP还被设计为一种中介协议，用于将通信转换为非HTTP信息系统，以及将非HTTP信息系统的通信转换为HTTP格式。HTTP代理和网关可以通过将其多种协议转换为超文本格式来提供对其他信息服务的访问，客户端可以像访问HTTP服务一样查看和操作这些超文本格式的信息。

这种灵活性的一个后果是，协议不能根据接口背后的具体情况来定义。 相反，我们仅限于定义通信的语法、接收到的通信的意图以及接收方的预期行为。 如果将通信视为孤立存在，那么成功的操作

这应该体现在服务器提供的可观察接口的相应变化中。 然而，由于多个客户端可能并行操作，甚至可能存在相互冲突的情况，因此我们不能要求这些变化在单个响应范围之外仍然可观察。

本文档描述了HTTP中使用的或引用的架构元素，定义了“http”和“https”URI方案，描述了整体网络操作和连接管理，并定义了HTTP消息的成帧和转发要求。 我们的目标是定义HTTP消息处理所需的所有机制，这些机制独立于消息语义，从而为消息解析器和消息转发中介定义了一整套要求。

1.1. 需求表示法

本协议中的关键词“必须（MUST）”、“不得（MUST NOT）”、“要求（REQUIRED）”、“应（SHALL）”、“不应（SHALL NOT）”、“宜（SHOULD）”、“不宜（SHOULD NOT）”、“建议（RECOMMENDED）”、“可（MAY）”和“可选（OPTIONAL）”文档应按照[RFC2119]中的描述进行解释。

第2.5节定义了关于错误处理的合规性标准和注意事项。

1.2. 语法表示法

本规范采用[RFC5234]中的增强巴科斯-诺尔范式（ABNF）表示法，并带有第7节中定义的列表扩展，该扩展允许使用’#’运算符（类似于’*’运算符表示重复）来简洁地定义逗号分隔的列表。附录B展示了收集的语法，其中所有列表运算符都扩展为标准的ABNF表示法。

以下核心规则通过引用纳入，如[RFC5234]附录B.1中所定义：ALPHA（字母）、CR（回车符）、CRLF（CR LF）、CTL（控制符）、DIGIT（十进制数字0-9）、DQUOTE（双引号）、HEXDIG（十六进制数字0-9/A-F/a-f）、HTAB（水平制表符）、LF（换行符）、OCTET（任意8位数据序列）、SP（空格）和VCHAR（任意可见的[USASCII]字符）。

按照惯例，以“obs-”为前缀的ABNF规则名称表示因历史原因而出现的“过时”语法规则。

2. 建筑

HTTP是为万维网（WWW）架构而创建的，并随着时间的推移不断发展，以支持全球超文本系统的可扩展性需求。 该架构的许多方面都体现在用于定义HTTP的术语和语法结构中。

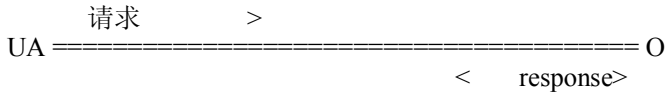
2.1. 客户端/服务器消息传递

HTTP是一种无状态的请求/响应协议，它通过在可靠的传输层或会话层“连接”（第6节）上交换消息（第3节）来运行。 HTTP“客户端”是一个程序，用于建立与服务器的连接，以便发送一个或多个HTTP请求。 HTTP“服务器”是一个程序，用于接受连接，并通过发送HTTP响应来处理HTTP请求。

术语“客户端”和“服务器”仅指这些程序在特定连接中所扮演的角色。 同一程序可能在某些连接上充当客户端，而在其他连接上充当服务器。 “用户代理”一词指的是发起请求的各种客户端程序中的任何一种，包括（但不限于）浏览器、爬虫（基于网络的机器人）、命令行工具、自定义应用程序和移动应用程序。 “源服务器”一词指的是能够为给定目标资源发起权威响应的程序。 “发送方”和“接收方”分别指的是发送或接收给定消息的任何实现。

HTTP依赖于统一资源标识符（URI）标准[RFC3986]来指示目标资源（第5.1节）以及资源之间的关系。 消息的传递格式类似于互联网邮件[RFC5322]和多用途互联网邮件扩展（MIME）[RFC2045]所使用的格式（有关HTTP和MIME消息之间的差异，请参阅[RFC7231]的附录A）。

大多数HTTP通信都包含一个检索请求（GET），用于获取由URI标识的某个资源的表示形式。 在最简单的情况下，这可以通过用户代理（UA）和源站之间的单个双向连接（===）来实现服务器（O）。



客户端以请求消息的形式向服务器发送HTTP请求，消息以请求行开头，包含方法、URI和协议版本（第3.1.1节），后跟包含请求修饰符、客户端信息和表示元数据的头部字段（第3.2节），一个空行表示头部部分的结束，最后是包含有效载荷正文的消息正文（如果有的话，第3.3节）。

服务器通过发送一个或多个HTTP响应消息来响应客户端的请求，每个消息都以一个状态行开头，该状态行包含协议版本、成功或错误代码以及文本原因短语（见第3.1.2节），后面可能跟着包含服务器信息、资源元数据和表示元数据的头部字段（见第3.2节），一个空行表示头部的结束，最后是包含有效载荷正文（如果有的话，见第3.3节）的消息正文。

如第6.3节所述，一个连接可能用于多次请求/响应交换。

以下示例说明了在URI“http://www.example.com/hello.txt”上针对GET请求（[RFC7231]的第4.3.1节）的典型消息交换：

客户端请求：

GET /hello.txt HTTP/1.1
用户代理：curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3 主机：www.example.com
Accept-Language: en, mi

服务器响应：

HTTP/1.1 200 OK
日期：2009年7月27日，星期一，12:28:53（格林尼治标准时间）
服务器：Apache
Last-Modified: 2009年7月22日星期三 19:15:56 GMTETag:
"34aa387-d-1568eb00"
Accept-Ranges: 字节 内容长
度: 51 Vary: Accept-Encoding
Content-Type: text/plain

你好世界！我的载荷包含一个尾部CRLF。【注】CRLF是Carriage Return Line Feed的缩写，表示换行符，在HTTP协议中常用于表示请求或响应的结束。

2.2. 实现多样性

在考虑HTTP的设计时，人们很容易陷入一种误区，认为所有用户代理都是通用浏览器，所有源服务器都是大型公共网站。 但实际情况并非如此。 常见的HTTP用户代理包括家用电器、立体声系统、体重秤、固件更新脚本、命令行程序、移动应用程序以及各种形状和大小的通信设备。 同样，常见的HTTP源服务器包括家庭自动化设备

设备、可配置网络组件、办公设备、自主机器人、新闻源、交通摄像头、广告选择器以及视频传输平台。

“用户代理”一词并不意味着在发出请求时，有一个人类用户直接与软件代理进行交互。 在许多情况下，用户代理被安装或配置为在后台运行，并保存其结果以供日后检查（或仅保存其中可能有趣或出错的部分结果）。 例如，爬虫通常被赋予一个起始URI，并被配置为在爬取作为超文本图形的网络时遵循特定行为。

HTTP的实现多样性意味着，并非所有用户代理都能向其用户提供交互式建议，或针对安全或隐私问题提供足够的警告。 在本规范要求向用户报告错误的少数情况下，此类报告仅在错误控制台或日志文件中可见是可以接受的。 同样，要求自动化操作在继续之前需经用户确认，这一要求可通过预先配置选项、运行时选项或简单避免不安全操作来实现；如果用户已做出该选择，则确认并不意味着任何特定的用户界面或正常处理的打断。

2.3. 中介

HTTP支持通过一系列连接使用中介来满足请求。 HTTP中介有三种常见形式：代理、网关和隧道。 在某些情况下，单个中介可能同时充当源服务器、代理、网关或隧道，根据每个请求的性质切换行为。

```
>             >             >             >
UA ========== A ========== B ========== C ========== O
<             <             <             <
```

上图展示了用户代理和源服务器之间的三个中介（A、B和C）。 在整个链路中传输的请求或响应消息将通过四个独立的连接。一些HTTP通信选项可能仅适用于与最近的非隧道邻居的连接，仅适用于链路的端点，或适用于链路上的所有连接。 尽管该图是线性的，但每个参与者都可能同时参与多个通信。 例如，B可能在处理A的请求的同时，从除A以外的许多客户端接收请求，和/或将请求转发给除C以外的服务器

同样，后续请求可能会通过不同的连接路径发送，这通常基于负载均衡的动态配置。

术语“上游”和“下游”用于描述与消息流相关的方向性要求：所有消息都从上游流向下游。术语“入站”和“出站”用于描述与请求路由相关的方向性要求：“入站”表示朝向源服务器，“出站”表示朝向用户代理。

“代理”是一种消息转发代理，通常由客户端通过本地配置规则选择，用于接收针对某些类型（或多种类型）的绝对URI的请求，并尝试通过HTTP接口进行转换来满足这些请求。有些转换是微小的，例如针对“http”URI的代理请求，而其他请求可能需要转换为完全不同的应用层协议，或者需要进行从完全不同的应用层协议到HTTP协议的转换。为了安全、注释服务或共享缓存，代理通常用于通过一个共同的中间层对组织的HTTP请求进行分组。
如第5.7.2节所述，一些代理被设计为在转发选定的消息或有效负载时对其进行转换。

“网关”（亦称“反向代理”）是一种中介，它作为出站连接的源服务器，但会翻译接收到的请求并将其入站转发到另一台或多台服务器。
网关通常用于封装遗留或不受信任的信息服务，通过“加速器”缓存提高服务器性能，并在多台机器上实现HTTP服务的分区或负载平衡。

所有适用于源服务器的HTTP要求同样适用于网关的出站通信。网关可使用其所需的任何协议与入站服务器进行通信，包括本规范范围之外的HTTP私有扩展。然而，希望与第三方HTTP服务器互操作的HTTP到HTTP网关，应当遵循用户代理对网关入站连接的要求。

“隧道”作为两个连接之间的盲中继，不改变消息内容。一旦激活，隧道即不被视为HTTP通信的一方，尽管隧道可能是由HTTP请求发起的。当被中继连接的两端关闭时，隧道即不复存在。隧道用于通过中介扩展虚拟连接，例如当使用传输层安全（TLS，[RFC5246]）通过共享防火墙代理建立机密通信时。

上述关于中介的分类仅考虑了作为HTTP通信参与者的中介。 还有一些中介可以在网络协议栈的较低层上操作，在消息发送者不知情或未经其许可的情况下过滤或重定向HTTP流量。 网络中介（在协议层面）与中间人攻击难以区分，由于错误地违反了HTTP语义，常常会引入安全漏洞或互操作性问题。

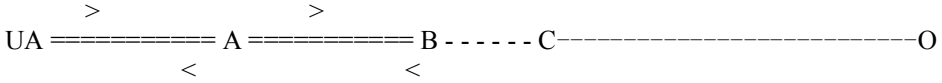
例如，“拦截代理”[RFC3040]（通常也称为“透明代理”[RFC1919]或“强制门户”）与HTTP代理不同，因为它不是由客户端选择的。 相反，拦截代理会过滤或重定向传出TCP端口80的数据包（偶尔也会过滤或重定向其他常见端口流量）。 拦截代理通常存在于公共网络接入点上，作为在允许使用非本地互联网服务之前强制执行账户订阅的一种手段，同时也存在于企业防火墙中，以强制执行网络使用策略。

HTTP被定义为一种无状态协议，这意味着每个请求消息都可以单独理解。 许多实现依赖于HTTP的无状态设计，以便重用代理连接或在多个服务器之间动态地负载平衡请求。 因此，除非连接是安全的且特定于该代理，否则服务器绝不能假定同一连接上的两个请求来自同一用户代理。 已知一些非标准的HTTP扩展（例如，[RFC4559]）违反了这一要求，从而导致安全和互操作性问题。

2.4. 缓存

“缓存”是本地存储先前响应消息的机制，以及控制其消息存储、检索和删除的子系统。缓存存储可缓存的响应，以减少未来等效请求的响应时间和网络带宽消耗。 任何客户端或服务器都可以使用缓存，但当服务器充当隧道时，则不能使用缓存。

缓存的作用在于，如果请求/响应链中的某个参与者拥有适用于该请求的缓存响应，则可以缩短该链。 以下示例说明了当B拥有O（通过C）针对UA或A未缓存的请求的较早响应的缓存副本时，产生的响应链。



如果缓存被允许存储响应消息的副本以用于回答后续请求，则该响应是“可缓存的”。 即使响应是可缓存的，客户端或源服务器也可能对何时可以将缓存的响应用于特定请求施加额外的限制。 [RFC7234]的第2节定义了HTTP对缓存行为和可缓存响应的要求。

在万维网和大型组织内部，部署了多种多样的缓存架构和配置。
这些包括用于节省跨洋带宽的国家代理缓存层级结构、用于广播或多播缓存条目的协作系统、用于离线或高延迟环境下的预取缓存条目存档，等等。

2.5. 一致性及错误处理

本规范根据HTTP通信中参与者的角色设定了符合性标准。 因此，根据需求所约束的行为，HTTP要求适用于发送方、接收方、客户端、服务器、用户代理、中介、源服务器、代理、网关或缓存。
当（社交）要求超出单次通信的范围时，会对实现、资源所有者和协议元素注册施加额外的（社交）要求。

在需求中区分创建协议元素和仅将接收到的元素转发到下游的情况下，使用动词“生成”而不是“发送”。

如果一个实现符合其在HTTP中所扮演角色的所有相关要求，则认为该实现是合规的。

一致性包括协议元素的语法和语义。 发送方不得生成其明知含义错误的协议元素。 发送方不得生成不符合相应ABNF规则所定义语法的协议元素。 在给定的消息中，发送方不得生成仅允许由其他角色（即发送方在该消息中不具有的角色）的参与者生成的协议元素或语法选项。

当解析收到的协议元素时，接收方必须能够解析任何合理长度的值，该值适用于接收方的角色，并且符合相应ABNF规则定义的语法。 但请注意，某些收到的协议元素可能无法解析。 例如，中间人

转发消息时，可能会将一个头部字段解析为通用字段
字段名和字段值组件，但随后转发该头字段，不对字段值内部进行进一步解析。

HTTP协议中的许多元素并没有特定的长度限制，因为合适的长度可能会因部署环境和实现目的的不同而大相径庭。 因此，发送方和接收方之间的互操作性依赖于双方对每个协议元素合理长度的共同预期。 此外，在过去二十年的HTTP使用过程中，对于某些协议元素通常认为的合理长度已经发生了变化，并且预计未来还会继续变化。

至少，接收方必须能够解析和处理协议元素长度，这些长度至少与其在其他消息中为相同协议元素生成的值一样长。 例如，
发布指向自身资源的超长URI引用的源服务器，在接收到这些相同的引用作为请求目标时，需要能够解析和处理它们。

接收方必须根据本规范（包括本规范的扩展）为其定义的语义来解释收到的协议元素，除非接收方（通过经验或配置）确定发送方错误地实现了这些语义所暗示的内容。 例如，源服务器可能会忽略收到的内容
如果检查 User-Agent 头部字段发现特定的实现版本在接收到某些内容编码时会出现故障，则应检查 Accept-Encoding 头部字段。

除非另有说明，否则接收方可以尝试从无效构造中恢复出可用的协议元素。 HTTP并未定义特定的错误处理机制，除非这些机制对安全性有直接影响，因为协议的不同应用需要不同的错误处理策略。 例如，Web浏览器可能希望从Location头字段未按照ABNF解析的响应中透明地恢复，而系统控制客户端则可能认为任何形式的错误恢复都是危险的。（注：ABNF：抽象语法标记法，用于描述文本的语法结构。在此上下文中，它用于描述HTTP响应中的Location头字段的格式。由于原文中未给出具体定义，因此在译文中保留了“ABNF”这一术语。）。

2.6. 协议版本控制

HTTP采用“<major>.<minor>”的编号方案来表示协议的版本。 本规范定义了版本“1.1”。 协议版本整体表明发送方符合该版本对应的HTTP规范中提出的一系列要求。

HTTP消息的版本由消息第一行中的HTTP-version字段表示。 HTTP-version区分大小写。

HTTP版本 = HTTP名称 "/" 数字 "." 数字
HTTP-name = %x48.54.54.50；"HTTP"，区分大小写

HTTP版本号由两个十进制数字组成，中间用“.”（句点或小数点）分隔。 第一个数字（“主版本”）表示HTTP消息传递语法，而第二个数字（“次版本”）表示该主版本中发送方符合并能够理解的最高次版本，以便于未来通信。 即使发送方仅使用协议的向后兼容子集，次版本也会通告发送方的通信能力，从而使接收方知道（服务器）可以在响应中使用或（客户端）可以在未来请求中使用更高级的功能。

当向HTTP/1.0接收方[RFC1945]或版本未知的接收方发送HTTP/1.1消息时，HTTP/1.1消息的构造方式使得在忽略所有较新特性后，该消息仍可被解释为有效的HTTP/1.0消息。 本规范对某些新特性提出了接收方版本要求，以便符合规范的发送方在通过配置或接收消息确定接收方支持HTTP/1.1之前，仅使用兼容特性。

在相同的主要HTTP版本的不同次要版本之间，对头部字段的解释不会发生变化，尽管在没有此类字段的情况下，接收方的默认行为可能会发生变化。 除非另有说明，否则HTTP/1.1中定义的头部字段适用于HTTP/1.x的所有版本。 特别是，无论是否声明符合HTTP/1.1，所有HTTP/1.x实现都应实现Host和Connection这两个头部字段。

如果新引入的头部字段的定义语义允许无法识别它们的接收方安全地忽略它们，则可以在不更改协议版本的情况下引入这些字段。 头部字段的可扩展性在第3.2.1节中进行了讨论。

处理HTTP消息的中介（即除充当隧道的中介之外的所有中介）必须在转发消息时发送自己的HTTP版本。 换言之，它们在转发HTTP消息的第一行时，必须确保该消息中的协议版本与该中介在接收和发送消息时所符合的版本相匹配，否则不得盲目转发。 转发HTTP消息时不进行重写

当下游接收方使用消息发送方的版本信息来判断后续与该发送方通信时可安全使用的功能时，HTTP版本可能会引发通信错误。

客户端应发送一个请求版本，该版本应等于客户端所符合的最高版本，且其主版本号不得高于服务器所支持的最高版本（如果已知）。 客户端不得发送其不符合的版本。（注：请求版本、最高版本、主版本号、服务器支持的最高版本均为HTTP标准协议中的术语，为确保翻译准确性，保留了原词。）。

如果客户端知道服务器错误地实现了HTTP规范，则可以在客户端至少尝试过一次正常请求，并根据响应状态码或头字段（例如Server）确定服务器无法正确处理更高版本的请求后，发送较低版本的请求。

服务器应发送一个响应版本，该版本应等于服务器符合的最高版本，且该版本的主版本号小于或等于请求中接收到的主版本号。 服务器不得发送其不符合的版本。 如果出于任何原因，服务器希望拒绝为客户端的主协议版本提供服务，则可以发送505（HTTP版本不支持）响应。

如果已知或怀疑客户端未正确实现HTTP规范，并且无法正确处理更高版本的响应，例如当客户端无法正确解析版本号，或者当已知中间设备会盲目转发HTTP版本（即使该版本不符合协议的给定次要版本）时，服务器可以向请求发送HTTP/1.0响应。 除非由特定客户端属性触发，否则不应执行此类协议降级，例如当一个或多个请求头字段（如User-Agent）与已知存在错误的客户端发送的值唯一匹配时。

HTTP版本设计的目的在于，仅当引入了不兼容的消息语法时，主版本号才会递增；而仅当对协议所做的更改增加了消息语义或暗示了发送方具备额外能力时，次版本号才会递增。
然而，对于[RFC2068]和[RFC2616]之间所做的更改，次版本号并未随之递增，并且此次修订特别避免了对此协议进行任何此类更改。

当接收方收到的主要版本号是其所实现的版本，但次要版本号高于其所实现的版本时，接收方应将该消息视为符合其所遵循的该主要版本中的最高次要版本进行处理。 接收方可以假设

当发送给尚未表示支持更高版本的接收方时，带有更高次版本号的消息具有足够的向后兼容性，能够被相同主版本号的任何实现安全处理。

2.7. 统一资源标识

统一资源标识符（URI）[RFC3986]在整个HTTP协议中被用作标识资源（[RFC7231]的第2节）的手段。URI引用被用于定位请求、指示重定向以及定义关系。

“URI引用”、“绝对URI”、“相对部分”、“方案”、“权威”、“端口”、“主机”、“路径-绝对”、“段”、“查询”和“片段”的定义均源自URI通用语法。 为可包含非空路径组件的协议元素定义了“绝对路径”规则。 （此规则与RFC 3986中的路径-绝对规则略有不同，后者允许在引用中使用空路径，而路径-绝对规则则不允许路径以“//”开头。 ）为可包含相对URI但不能包含片段组件的协议元素定义了“部分URI”规则。

URI引用 = <URI引用，参见[RFC3986]，第4.1节> 绝对URI = <绝对URI，参见[RFC3986]，第4.3节> 相对部分 = <相对部分，参见[RFC3986]，第4.2节> 方案= <方案，参见[RFC3986]，第3.1节> 权威= <权威，参见[RFC3986]，第3.2节>

uri-host = <主机，参见[RFC3986]，第3.2.2节> port = <端口，参见[RFC3986]，第3.2.3节>
path-abempty = <path-abempty，见[RFC3986]第3.3节> segment = <segment，见[RFC3986]第3.3节> query = <query，见[RFC3986]第3.4节>
fragment = <fragment，见[RFC3986]第3.5节>

绝对路径 = 1*("/" 片段)
部分URI =相对部分 ["?" 查询]

在HTTP中，每个允许URI引用的协议元素都会在其ABNF（抽象语法标记法）产生式中明确指出，该元素是允许任何形式的引用（URI引用），还是仅允许绝对形式的URI（绝对URI），仅包含路径和可选查询组件，或上述部分的组合。 除非另有说明，否则URI引用将相对于有效请求URI（第5.5节）进行解析。

2.7.1. HTTP URI方案

“http”URI方案特此定义，旨在根据标识符与由潜在HTTP源服务器管理的分层命名空间的关联来生成标识符，该服务器在给定端口上侦听TCP（[RFC0793]）连接。

```
http-URI = "http:" "//" 权威路径-绝对路径 [ "?" 查询参数 ] [ "#" 片段 ]
```

“http”URI的源服务器由权威组件标识，该组件包括主机标识符和可选的TCP端口（[RFC3986]，第3.2.2节）。 层次路径组件和可选的查询组件作为源服务器命名空间内潜在目标资源的标识符。 可选的片段组件允许独立于URI方案，间接标识次级资源，如[RFC3986]第3.5节所定义。

发送方绝不能生成主机标识符为空的“http”URI。 处理此类URI引用的接收方必须将其视为无效并拒绝。

如果主机标识符以IP地址的形式提供，则源服务器是该IP地址所指示的TCP端口上的监听器（如果有的话）。 如果主机是一个注册名称，则该注册名称是与名称解析服务（如DNS）一起使用的间接标识符，用于查找该源服务器的地址。 如果端口子组件为空或未给出，则默认使用TCP端口80（WWW服务的保留端口）。

请注意，存在具有给定授权组件的URI并不意味着该主机和端口上始终有一个HTTP服务器在监听连接。 任何人都可以生成一个URI。 授权组件的作用是确定谁有权对针对所标识资源的请求进行权威响应。 注册名称和IP地址的委托性质创建了一个基于对指定主机和端口控制权的联邦命名空间，无论HTTP服务器是否存在。有关建立授权的安全考虑，请参见第9.1节。（注：1.“authority component”在此处翻译为“授权组件”，指的是URI中的授权部分，用于确定谁有权对请求进行响应。2.“federated namespace”翻译为“联邦命名空间”，指的是由多个命名空间组成的系统，这些命名空间由不同的实体或组织控制，以实现资源的有效管理和共享。3.“control over the indicated host and port”翻译为“对指定主机和端口的控制”，指的是对特定主机和端口的访问权限控制。4.“establishing authority”翻译为“建立授权”，指的是在HTTP协议中，通过注册名称和IP地址来建立对特定资源的控制权和响应权的过程。

当在需要访问指定资源的上下文中使用“http”URI时，客户端可以通过将主机解析为IP地址、在指定端口上建立到该地址的TCP连接，并发送HTTP请求消息来尝试访问（第3节）向服务器发送包含URI标识数据（第5节）的请求。如果服务器对该请求的响应不是临时的

根据[RFC7231]第6节所述的HTTP响应消息，该响应被视为对客户端请求的权威性回答。

尽管HTTP独立于传输协议，但“http”方案特定于基于TCP的服务，因为名称委托过程依赖于TCP来建立权威性。 基于其他底层连接协议的HTTP服务可能会使用不同的URI方案进行标识，就像“https”方案（如下）用于需要端到端安全连接的资源一样。 其他协议也可能用于提供对“http”标识资源的访问——只有权威接口特定于TCP。

权威的URI通用语法还包括一个已弃用的userinfo子组件（[RFC3986]，第3.2.1节），用于在URI中包含用户身份验证信息。 一些实现利用userinfo组件进行身份验证信息的内部配置，例如在命令调用选项、配置文件或书签列表中，尽管这种使用可能会暴露用户标识符或密码。 当在消息中生成“http”URI引用作为请求目标或标头字段值时，发送方不得生成userinfo子组件（及其“@”分隔符）。 在使用从不受信任的来源接收到的“http”URI引用之前，接收方应解析是否存在userinfo，并将其存在视为错误；这很可能是为了网络钓鱼攻击而掩盖权威。

2.7.2. https统一资源标识符（URI）方案

“https”URI方案特此定义，旨在根据标识符与由潜在HTTP源服务器管理的分层命名空间的关联来生成标识符，该服务器在给定的TCP端口上侦听TLS安全连接（[RFC5246]）。

上述针对“http”方案列出的所有要求同样适用于“https”方案，但以下情况除外：如果端口子组件为空或未指定，则默认使用TCP端口443；并且用户代理在发送第一个HTTP请求之前，必须确保其与源服务器的连接是通过端到端的强加密来保护的。

```
https-URI = "https:" "//" 权威路径-abempty [ "?" 查询参数 ] [ "#" 片段 ]
```

请注意，“https”URI方案依赖于TLS和TCP来建立权威性。 通过“https”方案提供的资源与“http”方案没有共享的身份，即使它们

资源标识符表示相同的权威机构（即监听相同TCP端口的主机）。 它们是不同的命名空间，被视为不同的源服务器。 然而，HTTP的一种扩展，如Cookie协议[RFC6265]，被定义为适用于整个主机域，可以允许一个服务设置的信息影响与匹配的主机域组中的其他服务之间的通信。

权威访问“https”标识资源的过程在[RFC2818]中进行了定义。

2.7.3. HTTP和HTTPS统一资源标识符（URI）的规范化和比较

由于“http”和“https”方案符合URI通用语法，因此此类URI会根据[RFC3986]第6节中定义的算法进行规范化处理和比较，同时使用上述每种方案的默认设置。

如果端口等于某个方案的默认端口，则通常省略端口子组件。 当不以绝对形式作为OPTIONS请求的请求目标时，空路径组件等同于绝对路径“/”，因此通常提供路径“/”。 方案和主机不区分大小写，通常以小写形式提供；所有其他组件则以区分大小写的方式进行比较。 除“保留”集合中的字符外，其他字符均等同于其
百分号编码的字节：通常的做法是不对其进行编码（参见[RFC3986]的第2.1节和第2.2节）。

例如，以下三个统一资源标识符（URI）是等价的：

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

3. 消息格式

所有HTTP/1.1消息均由一个起始行和一系列八位字节组成，其格式类似于互联网消息格式[RFC5322]：零个或多个头部字段（统称为“头部”或“头部部分”），一个空行表示头部部分的结束，以及一个可选的消息体。

```
HTTP消息          =起始行
                   *( 头部字段 CRLF ) CRLF
                   [消息体]
```

解析HTTP消息的正常程序是读取
将起始行解析为一种结构，按字段名称将每个头部字段读入哈希表，直到遇到空行，然后使用解析后的数据来判断是否需要消息体。 如果已指示有消息体，则将其作为流读取，直到读取的字节数等于消息体长度或连接关闭为止。

接收方必须将HTTP消息解析为采用US-ASCII（USASCII）超集编码的八位字节序列。 将HTTP消息解析为Unicode字符流而不考虑具体编码，会因字符串处理库处理包含八位字节LF（%x0A）的无效多字节字符序列的方式不同，从而产生安全漏洞。
基于字符串的解析器只能在从消息中提取出协议元素后，在协议元素内部安全使用，例如在消息解析已划分出各个字段后，在头字段-值内部使用。

HTTP消息可以解析为流，以便进行增量处理或向下游转发。 然而，接收方不能依赖于部分消息的增量传递，因为出于网络效率、安全检查或有效载荷转换的考虑，某些实现会缓存或延迟消息转发。

发送方不得在起始行和第一个头部字段之间发送空格。 若接收方在起始行和第一个头部字段之间收到空格，则必须将该消息视为无效而拒绝，或者在不进一步处理的情况下消费每行以空格开头的行（即忽略整行以及任何后续以空格开头的行，直至收到格式正确的头部字段或头部部分终止）。

请求中存在此类空白可能是企图欺骗服务器忽略该字段或将该字段后的行作为新请求处理，如果请求链中的其他实现以不同方式解释相同消息，则这两种情况都可能导致安全漏洞。 同样，响应中存在此类空白可能会被某些客户端忽略，或导致其他客户端停止解析。

3.1. 起始行

HTTP消息可以是客户端向服务器发出的请求，也可以是服务器向客户端发出的响应。 从语法上讲，这两种消息类型的区别仅在于起始行，即请求行（针对请求）或状态行（针对响应），以及确定消息体长度的算法（见第3.3节）。

理论上，客户端可以接收请求，服务器可以接收响应，通过它们不同的起始行格式来区分。但在实践中，服务器通常只实现为期望接收请求（响应被解释为未知或无效的请求方法），而客户端通常只实现为期望接收响应。

起始行 = 请求行 / 状态行

3.1.1. 请求行

请求行以方法令牌开头，后跟一个空格（SP）、请求目标、另一个空格（SP）、协议版本，并以CRLF结束。

请求行 = 方法 SP 请求目标 SP HTTP版本 CRLF

方法令牌（method token）表示要对目标资源执行的请求方法。 请求方法是区分大小写的。

method = token

本规范定义的请求方法可在[RFC7231]的第4节中找到，同时该节还提供了有关HTTP方法注册表以及定义新方法时的注意事项的信息。

请求目标标识了要对其应用请求的目标资源，如第5.3节所述。

接收方通常通过空格分割来解析请求行的各个组成部分（见第3.5节），因为这三个组成部分中不允许出现空格。 遗憾的是，一些用户代理未能正确编码或排除超文本引用中的空格，导致这些不允许出现的字符被发送到请求目标中。

无效请求行的接收者应当以以下两种方式之一进行响应：
400（错误请求）错误或301（永久移动）重定向，且请求目标已正确编码。 接收方不应尝试自动更正请求，然后在没有重定向的情况下处理请求，因为无效的请求行可能是故意设计的，以绕过请求链中的安全过滤器。

HTTP对（某个内容）的长度没有预设限制
请求行，如第2.5节所述。 如果服务器接收到的方法比其实现的任何方法都长，则应返回501（未实现）状态码。 如果服务器接收到

如果请求目标（request-target）的长度超过其希望解析的任何URI，则必须返回414（URI过长）状态码（参见[RFC7231]的第6.5.12节）。

实践中存在各种针对请求行长度的临时限制。 建议所有HTTP发送方和接收方至少支持8000个八位字节的请求行长度。

3.1.2. 状态行

响应消息的第一行是状态行，由协议版本、一个空格（SP）、状态码、另一个空格、一个可能为空的描述状态码的文本短语组成，并以CRLF（回车换行）结束。

状态行 = HTTP版本 空格 状态码 空格 原因短语 CRLF

状态码元素是一个3位整数代码，用于描述服务器尝试理解和满足客户端相应请求的结果。 响应消息的其余部分将根据该状态码定义的语义进行解释。

有关状态码语义的信息，包括状态码类别（由第一位数字表示）、本规范定义的状态码、定义新状态码的考虑因素以及IANA注册表，请参见[RFC7231]的第6节。

状态码 = 3位数字

reason-phrase元素存在的唯一目的是提供与数字状态码相关的文本描述，这主要是出于对早期互联网应用协议的尊重，这些协议更常用于交互式文本客户端。 客户端应忽略reason-phrase的内容。

原因短语 = *(HTAB / SP / VCHAR / obs-text)

3.2. 头字段

每个头字段由一个不区分大小写的字段名、一个冒号（“:”）、可选的前导空格、字段值和可选的尾部空格组成。

```
头字段          = 字段名 ":" OWS 字段值 OWS

字段名          =令牌
字段值          =*(字段内容/观察折叠)
字段内容        =字段-vchar [ 1*( 空格 / 制表符 ) 字段-vchar ] 字段-vchar
                =VCHAR / 观察文本

obs-fold        = CRLF 1*( SP / HTAB )
                ; 过时的行折叠
                ; 见第3.2.4节
```

字段名标记用于标示相应的字段值，表明其具有该头字段所定义的语义。 例如，在[RFC7231]的第7.1.1.2节中定义了Date头字段，该字段包含其所出现消息的发起时间戳。

3.2.1. 字段可扩展性

头字段是完全可扩展的：对于引入新的字段名称（每个字段可能定义新的语义）以及在给定消息中使用的头字段数量，均无限制。 现有字段在本规范的各个部分以及本文件集以外的许多其他规范中均有定义。

可以定义新的头部字段，这样当接收方理解这些字段时，它们可能会覆盖或增强对先前定义的头部字段的解释，定义请求评估的先决条件，或细化响应的含义。

代理必须转发未识别的头字段，除非该字段名列在Connection头字段（第6.1节）中，或者代理被特别配置为阻止或以其他方式转换此类字段。 其他接收方应忽略未识别的头字段。这些要求使得HTTP的功能能够得到增强，而无需事先更新已部署的中介设备。

所有已定义的头部字段都应按照[RFC7231]第8.3节的描述，在“消息头部”注册表中向互联网号码分配机构（IANA）进行注册。

3.2.2. 字段顺序

接收具有不同字段名的标头字段的顺序并不重要。 然而，良好的做法是先发送包含控制数据的标头字段，例如请求中的Host和响应中的Date，以便实现可以尽早决定何时不处理消息。 在收到整个请求之前，服务器不得对目标资源应用请求

由于后续的头部字段可能包含条件语句、身份验证凭据或故意误导的重复头部字段，这些字段会影响请求处理，因此需要接收头部部分。

发送方在一条消息中不得生成多个具有相同字段名的头部字段，除非该头部字段的整个字段值被定义为逗号分隔的列表[即，#(values)]，或者该头部字段是一个众所周知的例外（如下所述）。

接收方可以将具有相同字段名的多个头部字段合并为一个“字段名:字段值”对，而不改变消息的语义，方法是将每个后续字段值按顺序附加到合并后的字段值后面，并以逗号分隔。 因此，具有相同字段名的头部字段的接收顺序对于合并字段值的解释具有重要意义；代理在转发消息时，不得改变这些字段值的顺序。

> 注：在实践中，“Set-Cookie”头部字段（[RFC6265]）通常会在响应消息中出现多次，并且不使用列表语法，这违反了上述关于多个同名头部字段的要求。 由于它不能合并为一个字段值，因此接收方在处理头部字段时应将“Set-Cookie”作为特殊情况处理。 （详情请参见[Kri2001]的附录A.2.3。）

3.2.3. 空白

本规范使用三条规则来表示线性空白的使用：OWS（可选空白）、RWS（必需空白）和BWS（“不良”空白）。

OWS规则用于可能出现零个或多个线性空白字节的情况。 对于那些为了提高可读性而首选可选空白的协议元素，发送方应将可选空白生成为一个单一的空格（SP）；否则，除非在就地消息过滤过程中需要用来覆盖无效或不需要的协议元素，否则发送方不应生成可选空白。

当需要至少一个线性空白字节来分隔字段标记时，使用RWS规则。 发送方应将RWS作为单个空格（SP）生成。

BWS规则用于语法中仅出于历史原因允许可选空白字符的情况。 发送方不得在消息中生成BWS。 接收方必须解析此类不良空白字符，并在解释协议元素之前将其删除。（注：BWS：Blank White Space，空白字符）。

```
OWS            = *( SP / HTAB )
                 ; 可选空格
RWS            = 1*( SP / HTAB )
                 ; 必需的空格
BWS            = OWS
                 ; “不良”空白
```

3.2.4. 字段解析

消息解析采用通用算法，与各个头字段名称无关。 给定字段值中的内容在消息解释的后期阶段（通常在处理完消息的整个头部分之后）才会被解析。 因此，与之前的版本不同，本规范不使用ABNF规则来定义每个“字段名称：字段值”对。 相反，本规范使用根据每个已注册字段名称命名的ABNF规则，其中规则定义了该字段对应字段值的有效语法（即在通用字段解析器从头部分提取字段值之后）。

在HTTP标准协议中，头字段名与冒号之间不允许有空格。 过去，对此类空格的不同处理方式曾导致请求路由和响应处理中出现安全漏洞。 服务器必须拒绝接收任何在头字段名与冒号之间包含空格的请求消息，并返回400（错误请求）的响应码。 代理在将响应消息转发给下游之前，必须移除其中的所有此类空格。

字段值前后可添加可选的空白字符（OWS）；为保持人类阅读的一致性，字段值前最好添加一个空格（SP）。 字段值不包括任何前导或尾随的空白字符：OWS出现在字段值的第一个非空白字符之前或最后一个非空白字符之后
在从头字段中提取字段值时，解析器应排除字段值中的非空白八位字节。

从历史上看，HTTP头字段值可以通过在每行额外内容前至少添加一个空格或水平制表符（obs-fold）来扩展到多行。 本规范不推荐使用此类行折叠方式，除非是在message/http媒体类型中
（第8.3.1节）。 发送方不得生成包含行折叠（即任何字段值包含与obs-fold规则匹配的内容）的消息，除非该消息旨在打包在message/http媒体类型中。（注：obs-fold规则是HTTP标准协议中的一种规则，用于检测和消除可能造成混淆的行折叠，具体定义和实现方式在协议文档中有详细说明。在此处，我们保留了原英文术语，并在括号中添加了中文解释，以便读者理解。）。

如果服务器在请求消息中接收到不在message/http容器内的obs-fold，则必须通过发送400（错误请求）来拒绝该消息，最好附带一个表示，说明过时的行折叠是不可接受的；或者在解释字段值或将消息转发到下游之前，用一或多个SP八位字节替换每个接收到的obs-fold。

在响应消息中接收到不在message/http容器内的obs-fold的代理或网关，必须选择以下两种操作之一：一是丢弃该消息，并替换为502（错误网关）响应，最好附带一个表示，说明收到了不可接受的行折叠；二是在解释字段值或将消息转发到下游之前，将每个接收到的obs-fold替换为一个或多个SP八位字节。

如果一个用户代理在响应消息中接收到一个不在message/http容器内的obs-fold，那么它必须替换每个接收到的
在解释字段值之前，先观察并折叠一个或多个SP（空格）字节。

历史上，HTTP协议一直允许字段内容使用ISO-8859-1字符集[ISO-8859-1]中的文本，仅支持其他字符集
通过使用[RFC2047]编码。在实际应用中，大多数HTTP头字段值仅使用US-ASCII字符集[USASCII]的一个子集。新定义的头部字段应将其字段值限制为
US-ASCII 八位字节。接收方应将字段内容（obs-text）中的其他八位字节视为不透明数据。

3.2.5. 字段限制

HTTP并未对每个头部字段的长度或整个头部部分的长度设置预定义的限制，如第2.5节所述。实践中，对单个头部字段长度的各种临时限制随处可见，这通常取决于特定字段的语义。

当服务器收到一个或多个超出其处理能力的请求头字段时，必须以适当的4xx（客户端错误）状态码进行响应。忽略此类头字段会增加服务器遭受请求欺骗攻击（第9.5节）的风险。

如果接收到的标头字段大于客户端希望处理的范围，并且字段语义表明丢弃的值可以在不改变消息格式或响应语义的情况下被安全忽略，则客户端可以丢弃或截断这些字段。

3.2.6. 字段值组件

大多数HTTP头字段值都是使用常见的语法组件（标记、引号字符串和注释）定义的，这些组件由空格或特定的分隔字符分隔。 分隔符是从不允许出现在标记中的US-ASCII可视字符集中选择的（双引号（"DQUOTE"）和"()、/、:、<、=、>、?、@、[、]、{}"）。

```
token          = 1*tchar

tchar          = "!" / "#" / "$" / "%" / "&" / "’ " / "*"
               / "+" / "-" / "." / "^" / "_" / " ‘" / "|" / "~"
               / 数字 / 字母
               ; 任何VCHAR，分隔符除外
```

如果文本字符串用双引号括起来，则会被解析为单个值。

```
quoted-string  = DQUOTE *( qdtext / quoted-pair ) DQUOTE
qdtext         = HTAB / SP /%x21 / %x23-5B / %x5D-7E / obs-text obs-text
               = %x80-FF
```

在某些HTTP头字段中，可以通过在注释文本前后加上括号来包含注释。 仅当字段值定义中包含“comment”时，才允许在该字段中添加注释。

```
注释           = "(" *( 文本 / 双引号对 / 注释 ) ")"
ctext          = HTAB / SP / %x21-27 / %x2A-5B / %x5D-7E / obs-text
```

在引号字符串和注释构造中，反斜杠字节（“\”）可用作单字节引号机制。 处理引号字符串值的接收方必须将引号对视为已被反斜杠后的字节所替换。

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

发送方在引号字符串中不应生成引号对，除非该字符串中出现的双引号（DQUOTE）和反斜杠字节需要被引用。 发送方在注释中也不应生成引号对，除非该注释中出现的括号["("和")"]和反斜杠字节需要被引用。

3.3. 消息主体

HTTP消息的消息体（如果有的话）用于承载该请求或响应的有效载荷体。 除非应用了传输编码（如第3.3.1节所述），否则消息体与有效载荷体相同。

消息体 = *八位字节

在请求和响应中，允许在消息中包含消息体的规则有所不同。

请求中消息体的存在由Content-Length或Transfer-Encoding头字段来指示。 即使方法没有定义任何使用消息体的场景，请求消息的构建也与方法语义无关。

响应中是否存在消息体取决于它所响应的请求方法以及响应状态码（见第3.1.2节）。 对于HEAD请求方法（见[RFC7231]第4.3.2节）的响应，从不包含消息体，因为相关的响应头字段（如Transfer-Encoding、Content-Length等）即使存在，也仅指示如果请求方法是GET（见[RFC7231]第4.3.1节）时其值会是什么。对于CONNECT请求方法（见[RFC7231]第4.3.6节）的2xx（成功）响应，会切换到隧道模式，而不包含消息体。所有1xx（信息性）、204（无内容）和304（未修改）响应都不包含消息体。 所有其他响应都包含消息体，尽管该消息体的长度可能为零。【注】1xx：1xx是HTTP协议中的一类状态码，表示请求已成功接收，但尚未处理。此类状态码通常用于信息性回复，如HTTP 200 OK（成功）和HTTP 204 No Content（无内容）等。204：204是HTTP协议中的一类状态码，表示请求已成功处理，但服务器未返回任何内容。此类状态码通常用于信息性回复，如HTTP 204 No Content（无内容）等。304：304是HTTP协议中的一类状态码，表示请求未修改。此类状态码通常用于信息性回复，如HTTP 304 Not Modified（未修改）等。

3.3.1. 传输编码

Transfer-Encoding头字段列出了与已（或将）应用于有效载荷主体以形成消息主体的传输编码序列相对应的传输编码名称。 传输编码在第4节中定义。

传输编码 = 1#传输编码

传输编码（Transfer-Encoding）类似于MIME中的内容传输编码（Content-Transfer-Encoding）字段，该字段旨在通过7位传输服务（[RFC2045]，第6节）实现二进制数据的安全传输。 然而，对于8位清洁传输协议而言，安全传输的重点有所不同。在HTTP中，传输编码的主要目的是准确界定动态生成的有效载荷，并将仅用于传输效率或安全性的有效载荷编码与作为所选资源特征的编码区分开来。

接收方必须能够解析分块传输编码（第4.1节），因为在有效载荷正文大小未知的情况下，分块传输编码在消息分帧中起着至关重要的作用。 发送方不得对消息正文多次应用分块传输编码（即，不允许对已分块的消息再次进行分块）。如果对请求有效载荷正文应用了除分块传输编码以外的任何传输编码，则发送方必须将分块传输编码作为最终传输编码，以确保消息正确分帧。 如果对响应有效载荷正文应用了除分块传输编码以外的任何传输编码，则发送方必须将分块传输编码作为最终传输编码，或者通过关闭连接来终止消息。

例如，

传输编码：gzip，分块

表示在形成消息体时，有效载荷正文已使用gzip编码进行压缩，然后使用分块编码进行分块处理。

与Content-Encoding（见[RFC7231]第3.1.2.1节）不同，
Transfer-Encoding是消息的属性，而非表示的属性，请求/响应链上的任何接收方都可以对收到的传输编码进行解码，或者对消息体应用额外的传输编码，前提是对Transfer-Encoding字段值进行了相应的更改。 有关编码参数的更多信息，可以通过本规范未定义的其他头字段提供。

Transfer-Encoding 可以作为对 HEAD 请求的响应发送，或者作为...（此处由于原文不完整，无法给出完整译文）
对于GET请求的304（未修改）响应（见[RFC7232]第4.1节），该响应和GET请求均不包含消息体，用以表明如果请求是无条件GET，则源服务器本应对消息体应用传输编码。 然而，此指示并非必需，因为响应链上的任何接收方（包括源服务器）都可以在不需要时移除传输编码。

服务器在任何状态码为1xx（信息性）或204（无内容）的响应中，不得发送Transfer-Encoding头字段。 服务器在任何对CONNECT请求（见[RFC7231]第4.3.6节）的2xx（成功）响应中，也不得发送Transfer-Encoding头字段。

传输编码（Transfer-Encoding）是在HTTP/1.1中新增的。 通常认为，仅声明支持HTTP/1.0的实现将无法理解如何处理经过传输编码的有效载荷。 客户端除非知道，否则不得发送包含传输编码的请求

服务器将处理HTTP/1.1（或更高版本）的请求；此类知识可能以特定用户配置的形式存在，或者通过记忆先前接收到的响应的版本而获得。 除非相应的请求指示为HTTP/1.1（或更高版本），否则服务器不得发送包含Transfer-Encoding的响应。

如果服务器收到带有其不理解的传输编码的请求消息，则应响应501（未实现）。

3.3.2. 内容长度

当消息中没有Transfer-Encoding（传输编码）头字段时，Content-Length（内容长度）头字段可以提供潜在有效载荷体的预期大小，以八位字节的十进制数表示。 对于包含有效载荷体的消息，Content-Length字段值提供了确定有效载荷体（和消息）结束位置所需的分帧信息。 对于不包含有效载荷体的消息，Content-Length指示所选表示的大小（见[RFC7231]第3节）。

Content-Length = 1位数字 一个示

例是

内容长度：3495

发送方在任何包含Transfer-Encoding头字段的消息中，不得发送Content-Length头字段。

当未发送Transfer-Encoding且请求方法定义了封闭载荷体的含义时，用户代理应在请求消息中发送Content-Length。 例如，即使在POST请求中值为0（表示空载荷体），通常也会发送Content-Length头字段。 当请求消息不包含载荷体且方法语义不预期此类载荷体时，用户代理不应发送Content-Length头字段。

服务器可以在对HEAD请求的响应中发送Content-Length头部字段（见[RFC7231]第4.3.2节）；但除非该字段的值等于如果使用GET方法发送相同请求时，响应的有效载荷体中本应发送的八位字节的十进制数，否则服务器不得在此类响应中发送Content-Length。

服务器可以在对条件式GET请求（见[RFC7232]第4.1节）的304（未修改）响应中发送Content-Length头字段；但在这种情况下，服务器不得发送Content-Length

除非其字段值等于对同一请求的200（OK）响应的负载体中本应发送的八位字节的十进制数。

服务器在任何状态码为1xx（信息性）或204（无内容）的响应中，不得发送Content-Length头字段。 服务器在任何对CONNECT请求（见[RFC7231]第4.3.6节）的2xx（成功）响应中，也不得发送Content-Length头字段。

除上述定义的情况外，在无传输编码（Transfer-Encoding）方面，当源服务器在发送完整的头部部分之前已知载荷正文的大小时，应发送Content-Length头字段。 这将使下游接收方能够测量传输进度，了解接收到的消息何时完成，并可能重用该连接以处理更多请求。

任何大于或等于零的Content-Length字段值都是有效的。 由于有效载荷的长度没有预定义的限制，接收方必须预料到可能存在较大的十进制数字，并防止因整数转换溢出而导致的解析错误（第9.3节）。

如果收到的消息中包含多个Content-Length头字段，且这些字段的值由相同的十进制值组成，或者包含一个Content-Length头字段，且该字段的值包含一组相同的十进制值（例如，“Content-Length: 42, 42”），这表明上游消息处理器生成或合并了重复的Content-Length头字段，则接收方必须拒绝该消息作为无效消息，或者在确定消息体长度或转发消息之前，用一个包含该十进制值的单个有效Content-Length字段替换重复的字段值。

> 注意：HTTP使用Content-Length字段进行消息分帧的方式与MIME中该字段的使用方式有很大不同，在MIME中，该字段是可选的，并且仅在“message/external-body”媒体类型中使用。

3.3.3. 消息体长度

消息体的长度由以下因素之一决定（按优先级排序）：

1. 对于HEAD请求的任何响应，以及任何带有1xx（信息性）、204（无内容）或304（未修改）状态码的响应，无论消息中存在哪些头部字段，都始终以头部字段后的第一个空行结束，因此不能包含消息体。

2. 对于CONNECT请求的任何2xx（成功）响应，都意味着在标头字段结束的空行之后，连接将立即成为隧道。 客户端必须忽略在此类消息中收到的任何Content-Length或Transfer-Encoding标头字段。（注：1. 2xx代表的是HTTP状态码中的成功状态码，具体含义请参考相关文档。2. “CONNECT请求”指的是通过HTTP协议建立的连接请求，具体含义请参考相关文档。3. “空行”指的是在HTTP消息中用于分隔标头字段和消息体的换行符，具体含义请参考相关文档。4. “标头字段”指的是HTTP消息中的请求或响应的头部信息，具体含义请参考相关文档。）。

3. 如果存在Transfer-Encoding头字段，并且分块传输编码（第4.1节）是最终编码，则通过读取和解码分块数据来确定消息体长度，直到传输编码指示数据已完成。

   如果响应中存在Transfer-Encoding头字段，且分块传输编码不是最终编码，则通过读取连接直到服务器关闭连接来确定消息体长度。 如果请求中存在Transfer-Encoding头字段，且分块传输编码不是最终编码，则无法可靠地确定消息体长度；服务器必须以400（错误请求）状态码进行响应，然后关闭连接。

   如果接收到的消息同时包含Transfer-Encoding和Content-Length头字段，则Transfer-Encoding将覆盖Content-Length。 此类消息可能表示尝试执行请求欺骗（第9.5节）或响应拆分（第9.4节），应作为错误处理。 发送方在将此类消息转发到下游之前，必须移除接收到的Content-Length字段。

4. 如果接收到的消息没有指定Transfer-Encoding，并且存在多个Content-Length头字段且每个字段的值不同，或者存在单个Content-Length头字段且其值无效，则该消息的帧结构无效，接收方必须将其视为不可恢复的错误。 如果这是一个请求消息，则服务器必须以400（错误请求）状态码进行响应，然后关闭连接。 如果这是一个由代理接收到的响应消息，则代理必须关闭与服务器的连接，丢弃收到的响应，并发送502（错误）状态码

网关）对客户端的响应。 如果这是用户代理收到的响应消息，则用户代理必须关闭与服务器的连接，并丢弃收到的响应。

5. 如果存在有效的Content-Length头部字段但没有Transfer-Encoding，则其十进制值定义了预期的消息体长度（以八位字节为单位）。 如果在收到指示的八位字节数之前，发送方关闭连接或接收方超时，则接收方必须认为该消息不完整，并关闭连接。

6. 如果这是一条请求消息，且上述条件均不满足，则消息体长度为零（即不存在消息体）。

7. 否则，这是一条未声明消息体长度的响应消息，因此消息体长度由服务器关闭连接前接收到的八位字节数决定。

由于无法区分成功完成的情况，
如果部分接收到的消息因网络故障中断，服务器应生成编码或
尽可能使用长度定界消息。 关闭定界功能主要是为了与HTTP/1.0向后兼容。

服务器可能会拒绝包含消息体但没有Content-Length的请求，并返回411（需要长度）的响应。

除非应用了非分块的传输编码，否则，如果预先知道消息体的长度，发送包含消息体的请求的客户端应使用有效的Content-Length头字段，而不是分块传输编码，因为即使某些现有服务理解分块传输编码，它们对分块响应时也会返回411（需要长度）状态码。这通常是因为这些服务是通过网关实现的，网关在调用之前需要内容长度，而服务器无法或不愿意在处理之前缓存整个请求。

如果用户代理不知道服务器是否支持HTTP/1.1（或更高版本）的请求，则发送包含消息体的请求时，必须发送有效的Content-Length头字段；这种知识可以通过特定的用户配置或记住先前接收到的响应的版本获得。

如果已完全接收到对连接上最后一个请求的最终响应，并且仍有其他数据要读取，则用户代理可以丢弃剩余数据，或尝试确定这些数据是否

数据属于先前响应正文的一部分，如果先前消息的Content-Length值不正确，则可能会出现这种情况。 客户端绝不能将此类额外数据作为单独的响应进行处理、缓存或转发，因为这种行为容易受到缓存中毒的攻击。

3.4. 处理未完成消息

当服务器接收到一个不完整的请求消息时，通常是由于请求被取消或触发了超时异常，该服务器可以在关闭连接之前发送一个错误响应。

当客户端接收到一个不完整的响应消息时（这种情况可能发生在连接过早关闭或对所谓的分块传输编码解码失败时），客户端必须将该消息记录为不完整。 关于不完整响应的缓存要求，请参见[RFC7234]的第3节。

如果一个响应在头部部分中间（在收到空行之前）终止，并且状态码可能依赖于头部字段来传达响应的全部含义，那么客户端就不能假定含义已经传达；客户端可能需要重复请求，以确定下一步应采取什么行动。

如果未收到终止编码的零大小块，则使用分块传输编码的消息体被视为不完整。 如果收到的消息体大小（以八位字节为单位）小于Content-Length给出的值，则使用有效Content-Length的消息被视为不完整。 如果响应既没有使用分块传输编码也没有使用Content-Length，则通过关闭连接来终止该响应，因此，只要报头部分完整接收，无论收到多少消息体八位字节，该响应都被视为完整。

3.5. 消息解析鲁棒性

较旧的HTTP/1.0用户代理实现可能会在POST请求后发送额外的CRLF，这是为了解决一些早期服务器应用程序无法读取未以换行符结尾的消息体内容的问题。 HTTP/1.1用户代理不得在请求前或请求后添加额外的CRLF。 如果希望以换行符结束请求消息体，则用户代理必须将结束的CRLF字节计入消息体长度。

为了确保健壮性，预期会接收并解析请求行的服务器应当忽略在请求行之前收到的至少一个空行（CRLF）。

尽管起始行和头部字段的行终止符是CRLF序列，但接收方可以将单个LF识别为行终止符，并忽略任何前面的CR。

尽管请求行和状态行语法规则要求每个组成元素之间用单个空格（SP）八位字节分隔，但接收方也可以选择以空格分隔的单词边界进行解析，并且除CRLF终止符外，将任何形式的空格视为SP分隔符，同时忽略前导或尾随空格；此类空格包括以下一个或多个八位字节：SP、HTAB、VT（%x0B）、FF（%x0C）或纯CR。 然而，如果消息有多个接收方，且每个接收方对健壮性有各自独特的理解（见第9.5节），则宽松解析可能导致安全漏洞。

当服务器仅监听HTTP请求消息，或处理从起始行看起来像是HTTP请求消息的内容时，如果接收到不符合HTTP消息语法的八位字节序列（上述列出的健壮性异常除外），服务器应返回400（错误请求）响应。

4. 传输代码

传输编码名称用于指示已对、可对或可能需要应用于有效载荷体的编码转换，以确保在网络中的“安全传输”。 这与内容编码不同，因为传输编码是消息的属性，而不是正在传输的表示形式的属性。

```
transfer-coding    ="chunked"; 第4.1节
                   “compress” ； 第4.2.1节
                   “deflate” ； 第4.2.2节
                   “gzip”； 第4.2.3节
                   / transfer-extension
transfer-extension = token *( OWS ";" OWS transfer-parameter ) 参数的形式为名称或名称=值对。

传输参数 = 标记 BWS "=" BWS ( 标记 / 引号字符串 )
```

所有传输编码名称均不区分大小写，并且应在HTTP传输编码注册表中注册，如第8.4节所述。 它们在TE（第4.3节）中使用，并且传输编码（第3.3.1节）头部字段。

4.1. 分块传输编码

分块传输编码将有效载荷正文打包，以便将其作为一系列块进行传输，每个块都有自己的大小指示符，后面跟随一个可选的尾部，其中包含头部字段。 分块传输编码使得未知大小的内容流能够作为一系列长度定界的缓冲区进行传输，从而使发送方能够保持连接的持久性，并使接收方能够知道何时已收到整个消息。

```
分块体          = *块
                  最后一个块尾部
                  部分回车换行

chunk           = 分块大小 [ 分块扩展 ] CRLF 分块数据
                  CRLF
chunk-size      = 1*HEXDIG
last-chunk      = 1*("0") [ chunk-ext ]回车换行

chunk-data      = 1*OCTET；一个由块大小个八位字节组成的序列
```

chunk-size字段是一个十六进制数字字符串，表示块数据的大小（以八位字节为单位）。 当接收到一个chunk-size为零的块时，分块传输编码即告完成，该块后可能跟有一个尾部，最后以一个空行结束。

接收方必须能够解析和解码分块传输编码。（注：“chunked transfer coding”是HTTP协议中的一种传输编码方式，用于将数据分割成多个部分进行传输，接收方需要能够解析并解码这些部分。）。

4.1.1. 块扩展

分块编码允许每个块在紧接块大小之后包含零个或多个块扩展，以便提供每个块的元数据（如签名或哈希值），
消息中间控制信息，或消息体大小的随机化。

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

块扩展名 = 标记
chunk-ext-val   = 标记符 / 引号字符串
```

分块编码特定于每个连接，并且很可能在任何更高级别的应用程序有机会检查扩展之前，被每个接收方（包括中间节点）移除或重新编码。 因此，分块扩展的使用通常受到限制

用于专门的HTTP服务，如“长轮询”（其中客户端和服务器可以对块扩展的使用有共同的预期），或用于端到端安全连接中的填充。

接收方必须忽略无法识别的块扩展。 服务器应将请求中接收到的块扩展的总长度限制在所提供服务的合理范围内，就像它对消息的其他部分应用长度限制和超时限制一样，如果超过该限制，则应生成适当的4xx（客户端错误）响应。（注：1）“chunk extensions”在此处翻译为“块扩展”，因为在HTTP协议中，“chunk”通常指的是数据块，而“extensions”则表示扩展，因此“块扩展”是对其含义的准确解释。2）“ought to”在此处翻译为“应该”，表示建议或义务，但考虑到中文表达习惯，将其省略，因为“应该”的含义在中文中通常可以通过其他方式表达。）。

4.1.2. 分块预告片部分

预告片允许发送方在分块消息的末尾包含额外的字段，以便提供在发送消息正文时可能动态生成的元数据，如消息完整性检查、数字签名或后处理状态。 预告片字段与头部字段相同，只是它们是在分块预告片中发送的，而不是在消息的头部部分。

```
trailer-part = *( header-field CRLF )
```

发送方不得生成包含以下必要字段的尾部：消息分帧字段（如Transfer-Encoding和Content-Length）、路由字段（如Host）、请求修饰符字段（如[RFC7231]第5节中的controls和conditionals）、认证字段（如参见[RFC7235]和[RFC6265]）、响应控制数据字段（如参见[RFC7231]第7.1节）或用于确定如何处理有效载荷的字段（如Content-Encoding、Content-Type、Content-Range和Trailer）。

当接收到包含非空trailer的分块消息时，接收方可以处理这些字段（除了上述禁止的字段之外），就像它们被附加到消息的头部部分一样。

接收方必须忽略（或视为错误）任何禁止在trailer中发送的字段，因为如果将这些字段视为存在于头部部分来处理，可能会绕过外部安全过滤器。【注】1. "chunked message"：分块消息，是HTTP协议中的一种消息格式，消息被分割成多个部分，每个部分包含一部分数据和消息头，然后通过消息头中的信息进行重组。2. "trailer"：trailer，在HTTP协议中，是指消息的最后部分，包含一些额外的信息，如内容长度、内容MD5校验码等。3. "forbidden"：禁止的，这里指的是在HTTP协议中，某些字段是不允许出现在trailer中的，因为如果这些字段被当作存在于头部部分来处理，可能会绕过外部安全过滤器，从而带来安全风险。

除非请求中包含一个如第4.3节所述的TE头字段，表明“trailers”是可以接受的，否则服务器不应生成其认为用户代理接收所必需的trailer字段。 如果没有包含“trailers”的TE，服务器应假定trailer字段在传输到用户代理的途中可能会被默默丢弃。 这一要求允许中间设备将解块后的消息转发给HTTP/1.0接收方，而无需缓存整个响应。

4.1.3. 解码分块

分块传输编码的解码过程可以用伪代码表示为：

```
长度 := 0
读取块大小、块扩展（如果有）和CRLF，当（块大小 > 0）
时，执行以下操作：
    读取数据块和CRLF
    将解码后的正文长度加上数据块大小，作为已解
    码正文长度
    读取数据块大小、数据块扩展（如果有的话）以及CRLF（回车换行符）
}
读取预告片字段
while (trailer字段不为空) {
    如果（trailer字段被允许在trailer中发送）{将trailer字段附加到现有的header字段
        中
    }
    读取trailer字段
}
Content-Length（内容长度）: 等于长度
从传输编码中移除“chunked” 从现有头部字段中移除
Trailer
```

4.2. 压缩编码

下面定义的编码可用于压缩消息的有效载荷。

4.2.1. 压缩编码

“compress”编码是一种自适应的Lempel-Ziv-Welch（LZW）编码[Welch]，通常由UNIX文件压缩程序“compress”生成。 接收方应将“x-compress”视为与“compress”等同。

4.2.2. Deflate编码

“deflate”编码是一种“zlib”数据格式[RFC1950]，其中包含一个“deflate”压缩数据流[RFC1951]，该数据流结合使用了Lempel-Ziv（LZ77）压缩算法和霍夫曼编码。

> 注意：一些不符合标准的实现会发送“deflate”压缩数据，而不使用zlib封装。

4.2.3. Gzip编码

“gzip”编码是一种带有32位循环冗余校验（CRC）的LZ77编码，通常由gzip文件压缩程序生成[RFC1952]。 接收方应将“x-gzip”视为与“gzip”等效。

4.3. 技术专家

请求中的“TE”头部字段表示客户端愿意接受的除分块传输编码外的其他传输编码，以及客户端是否愿意接受分块传输编码中的尾部字段。

TE字段值由逗号分隔的传输编码名称列表组成，每个名称都允许有可选参数（如第4节所述），和/或关键字“trailers”。 客户端不得在TE中发送分块传输编码名称；对于HTTP/1.1接收方，分块始终是可接受的。

```
TE        = #t-编码
t-codings = "trailers" / ( transfer-coding [ t-ranking ] ) t-ranking = OWS ";" OWS "q="
rank
rank      = ( "0" [ "." 0*3DIGIT ] )
            / ( "1" [ "." 0*3("0") ] )
```

以下是TE使用的三个示例。

```
TE: 压缩 TE:
TE: 拖车，放气；q=0.5
```

关键字“trailers”的存在表明，客户端愿意接受第4.1.2节中定义的分块传输编码中的trailer字段，这既代表自身，也代表任何下游客户端。 对于来自中介的请求，这意味着：（a）所有下游客户端都愿意接受转发响应中的trailer字段；或者，（b）中介将尝试代表下游接收者缓存响应。请注意，HTTP/1.1并未定义任何限制分块响应大小的方法，因此中介无法确保能够缓存整个响应。

当多个传输编码均可接受时，客户端可以通过使用不区分大小写的“q”参数（类似于内容协商字段中使用的q值）按优先级对编码进行排序，部分

[RFC7231]中的5.3.1）。 排名值是一个介于0到1之间的实数，其中0.001表示最不受欢迎，1表示最受欢迎；值为0表示“不可接受”。

如果TE字段值为空或不存在TE字段，则唯一可接受的传输编码为chunked。 没有传输编码的消息始终是可接受的。（注：TE字段，即Transfer Encoding字段，是HTTP协议中的一个字段，用于指定数据传输的编码方式。chunked是一种传输编码方式，表示数据以块的形式传输，而非一次性全部传输。）。

由于TE报头字段仅适用于直接连接，因此TE的发送方还必须在连接报头字段（第6.1节）中发送一个“TE”连接选项，以防止不支持其语义的中间设备转发TE字段。

4.4. 预告片

当消息包含使用分块传输编码的消息体，并且发送方希望在消息末尾以尾部字段的形式发送元数据时，发送方应在消息体之前生成一个Trailer头字段，以指示哪些字段将出现在尾部中。 这允许接收方在开始处理消息体之前，为接收这些元数据做好准备，这在消息以流式传输且接收方希望即时确认完整性检查时非常有用。（注：1.“chunked transfer coding”翻译为“分块传输编码”，是HTTP协议中的一种编码方式，用于将消息体分割成多个部分进行传输，以提高传输效率和可靠性。2.“trailer fields”翻译为“尾部字段”，是HTTP协议中的一种字段类型，用于在消息尾部添加额外的信息，如元数据等。3.“integrity check”翻译为“完整性检查”，是计算机科学中的一个术语，用于验证数据是否被篡改或损坏。4.“on the fly”翻译为“即时”，在这里表示接收方在处理消息的过程中，可以即时进行完整性检查。

Trailer = 1#字段名

5. 消息路由

HTTP请求消息的路由由每个客户端根据目标资源、客户端的代理配置以及入站连接的建立或重用情况来确定。 相应的响应路由则沿着相同的连接链返回客户端。

5.1. 标识目标资源

HTTP被广泛应用于各种场合，从通用计算机到家用电器。 在某些情况下，通信选项被硬编码在客户端的配置中。然而，大多数HTTP客户端都依赖于与通用Web浏览器相同的资源识别机制和配置技术。

HTTP通信由用户代理出于某种目的而发起。该目的结合了[RFC7231]中定义的请求语义以及应用这些语义的目标资源。 通常使用URI引用（第2.7节）作为...（此处原文信息不完整，无法给出完整译文）

“目标资源”的标识符，用户代理会将其解析为绝对形式，以获取“目标URI”。目标URI不包括引用的片段组件（如果有的话），因为片段标识符是保留给客户端处理的（[RFC3986]，第3.5节）。

5.2. 入站连接

一旦确定了目标URI，客户端需要判断是否需要通过网络请求来实现所需的语义，如果是，则需确定该请求应定向到何处。【注】1. HTTP：超文本传输协议（Hypertext Transfer Protocol），是互联网上用于传输和接收网页的协议。2. URI：统一资源标识符（Uniform Resource Identifier），是用于唯一标识网络资源的字符串。

如果客户端拥有缓存[RFC7234]并且请求可以通过缓存得到满足，那么通常会优先将请求定向到缓存。

如果缓存无法满足请求，则典型的客户端会检查其配置，以确定是否使用代理来满足请求。代理配置取决于具体实现，但通常基于URI前缀匹配、选择性授权匹配或两者兼有，并且代理本身通常由“http”或“https”URI标识。 如果代理适用，则客户端通过建立（或重用）到该代理的连接来进行入站连接。

如果没有可适用的代理，典型的客户端将调用一个处理程序例程（通常特定于目标URI的方案），以直接连接到目标资源的授权机构。具体实现方式取决于目标URI方案，并由其相关规范定义，类似于本规范如何定义“http”（第2.7.1节）和“https”（第2.7.2节）方案的源服务器访问解析。

第6节定义了与连接管理相关的HTTP要求。

5.3. 请求目标

一旦获得入站连接，客户端就会发送一条HTTP请求消息（第3节），其中包含从目标URI派生出的请求目标。 根据所请求的方法以及请求是否针对代理，请求目标有四种不同的格式。

```
请求目标（request-target）= 原始形式（origin-form）
                          / absolute-form
                          / 权限表单
                          / 星号形式
```

5.3.1. origin-form（来源形式）

请求目标最常见的形式是源形式。源形式 =绝对路径[“？”查询]

在直接向源服务器发出请求时，除CONNECT或服务器范围的OPTIONS请求（详见下文）外，客户端必须仅发送目标URI的绝对路径和查询组件作为请求目标。 如果目标URI的路径组件为空，则客户端必须在源形式的请求目标中发送“/”作为路径。 同时，还会发送一个Host头字段，如第5.4节所述。

例如，一个客户端希望检索被标识为的资源的表示形式

http://www.example.org/where?q=now

直接从源服务器向主机“www.example.org”的80端口打开（或重用）一个TCP连接，并发送以下行：

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

紧随其后的是请求消息的其余部分。

5.3.2. 绝对形式

在向代理发出请求时，除CONNECT或服务器范围的OPTIONS请求（详见下文）外，客户端必须以绝对形式发送目标URI作为请求目标。

绝对形式 =绝对URI

代理服务器被要求，如果可能的话，从有效的缓存中为该请求提供服务，或者代表客户端向下一个入站代理服务器或直接向请求目标所指示的源服务器发出相同的请求。 关于此类消息“转发”的要求在第5.7节中定义。

一个请求行的绝对格式示例如下：

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

为了在HTTP的未来版本中允许所有请求都转换为绝对形式，服务器必须接受请求中的绝对形式，即使HTTP/1.1客户端仅在向代理发送请求时使用绝对形式。

5.3.3. 权威表单

请求目标的权威形式仅用于CONNECT请求（见[RFC7231]第4.3.6节）。

权威形式 = 权威

当发出CONNECT请求以通过一个或多个代理建立隧道时，客户端必须仅发送目标URI的授权部分（不包括任何用户信息及其“@”分隔符）作为请求目标。 例如，

CONNECT www.example.com:80 HTTP/1.1

5.3.4. 星号形式

星号形式的请求目标仅用于服务器范围的OPTIONS请求（见[RFC7231]第4.3.7节）。

星号形式 =“*”

当客户端希望对整个服务器（而非该服务器的特定命名资源）请求OPTIONS时，客户端必须仅发送“*”（%x2A）作为请求目标。 例如，

选项* HTTP/1.1

如果代理接收到一个OPTIONS请求，且该请求的request-target为绝对形式，其中URI的路径为空且没有查询部分，则请求链上的最后一个代理在将请求转发到指定的源服务器时，必须发送一个“*”作为request-target。

例如，请求

选项http://www.example.org:8001 HTTP/1.1 将由最终代理转发

选项* HTTP/1.1
主机：www.example.org:8001

连接到主机“www.example.org”的8001端口后。

5.4. 主机

请求中的“Host”头字段提供了目标URI中的主机和端口信息，使源服务器能够在处理单个IP地址上多个主机名的请求时区分不同的资源。

主机 = URI主机 [":" 端口]；第2.7.1节

客户端必须在所有HTTP/1.1请求消息中发送Host头字段。 如果目标URI包含授权组件，则客户端必须发送一个与该授权组件相同的Host字段值，但不包括任何用户信息子组件及其“@”分隔符（见第2.7.1节）。 如果目标URI缺少授权组件或授权组件未定义，则客户端必须发送一个值为空的Host头字段。

由于Host字段的值对于处理请求至关重要，因此用户代理应将Host作为请求行后的第一个头部字段生成。

例如，向源服务器发出的GET请求
<http://www.example.org/pub/WWW/> 的开头部分为：

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

即使请求目标是绝对形式的，客户端也必须在HTTP/1.1请求中发送Host头字段，因为这样可以将Host信息通过可能未实现Host功能的旧版HTTP/1.0代理进行转发。

当代理接收到一个绝对形式的请求时
请求目标（request-target）方面，代理必须忽略接收到的Host头部字段（如果有的话），并替换为请求目标的主机信息。 转发此类请求的代理必须根据接收到的请求目标生成新的Host字段值，而不是转发接收到的Host字段值。（注：1.“request-target”翻译为“请求目标”，这是HTTP协议中的术语，表示请求的目标地址或资源。2.“proxy”翻译为“代理”，在HTTP协议中，代理是指用于转发HTTP请求的服务器或设备。3.“Host header field”翻译为“Host头部字段”，是HTTP协议中的术语，表示服务器或代理设备在HTTP请求中发送的服务器信息。4.“forward”翻译为“转发”，在HTTP协议中，转发是指将一个HTTP请求从一个服务器或代理设备发送到另一个服务器或代理设备。）。

由于Host头字段充当应用层路由机制，因此它经常成为恶意软件的目标，这些恶意软件试图污染共享缓存或将请求重定向到非预期的服务器。 如果拦截代理依赖于Host字段值来将请求重定向到内部服务器，或在共享缓存中用作缓存键，而未首先验证被拦截的连接是否指向该主机的有效IP地址，则该拦截代理尤其容易受到攻击。

对于任何缺少Host头字段的HTTP/1.1请求消息，以及任何包含多个Host头字段或包含无效字段值的Host头字段的请求消息，服务器必须返回400（错误请求）状态码。

5.5. 有效请求URI

由于请求目标通常仅包含用户代理目标URI的一部分，因此服务器会通过重构预期目标，生成一个“有效请求URI”，以便正确处理请求。 这一重构过程涉及服务器的本地配置以及请求目标、主机头字段和连接上下文中传递的信息。

对于用户代理而言，有效请求URI即目标URI。

如果请求目标（request-target）是绝对形式的，则有效请求URI（effective request URI）与请求目标相同。否则，有效请求URI的构造方式如下：

如果服务器的配置（或出站网关）提供了固定的URI方案，则该方案将用于有效请求URI。 否则，如果请求是通过TLS加密的TCP连接接收的，则有效请求URI的方案为“https”；否则，方案为“http”。

如果服务器的配置（或出站网关）提供了固定的URI授权组件，则该授权将用于有效请求URI。 若没有，且请求目标为授权形式，则有效请求URI的授权组件与请求目标相同。 若没有，且提供了带有非空字段值的Host头字段，则授权组件与Host字段值相同。 否则，授权组件将被分配为服务器配置的默认名称，并且如果连接的传入TCP端口号与有效请求URI方案的默认端口不同，则会在授权组件后附加一个冒号（“:”）和传入端口号（以十进制形式表示）。【注】1.“URI authority component”翻译为“URI授权组件”，这是HTTP协议中的一个术语，指的是URI中的授权部分，包括用户名、密码、主机名等。2.“authority-form”翻译为“授权形式”，这是HTTP协议中描述请求目标格式的一种方式，表示请求目标以授权形式存在。3.“Host header field”翻译为“Host头字段”，这是HTTP协议中的一个请求头字段，用于指定服务器的域名或IP地址。4.“incoming TCP port number”翻译为“传入TCP端口号”，这是指客户端向服务器发送请求时使用的TCP端口号。5.“default name configured for the server”翻译为“服务器配置的默认名称”，这是指在服务器配置中设定的默认名称，用于标识服务器。

如果请求目标是权威形式或星号形式，则有效请求URI的组合路径和查询组件为空。 否则，组合的路径和查询组件与请求目标相同。

一旦按照上述方法确定了有效请求URI的各个组成部分，就可以通过将方案、冒号“://”、授权部分以及组合后的路径和查询部分连接起来，将其组合成绝对URI的形式。

示例1：通过不安全的TCP连接接收到的以下消息

```
GET /pub/WWW/TheProject.html HTTP/1.1 Host:
www.example.org:8080
```

其有效请求URI为http://www.example.org:8080/pub/WWW/TheProject.html

示例2：以下是通过TLS加密的TCP连接接收到的消息

```
选项* HTTP/1.1
主机：www.example.org
```

其有效请求URI为https://www.example.org

在HTTP/1.0请求中，如果缺少Host头字段，接收方可能需要使用启发式方法（例如，检查URI路径中特定主机独有的内容）来猜测有效请求URI的权威组件。

一旦构建了有效的请求URI，源服务器需要决定是否通过接收请求的连接为该URI提供服务。 例如，请求可能被有意或无意地误导，导致接收到的请求目标或主机头字段中的信息与建立连接的主机或端口不一致。 如果连接来自可信网关，这种不一致可能是预期的；否则，它可能表明有人试图绕过安全过滤器，诱使服务器提供非公开内容，或污染缓存。 有关消息路由的安全考虑，请参见第9节。

5.6. 将响应与请求相关联

HTTP协议中并未包含用于将特定请求消息与其对应的一个或多个响应消息相关联的请求标识符。因此，它依赖于响应到达的顺序，以准确对应同一连接上发出的请求顺序。 仅当对同一请求的最终响应之前存在一个或多个信息性响应（1xx，见[RFC7231]第6.2节）时，才会出现每个请求对应多个响应消息的情况。

在连接上有多个未完成请求的客户端必须按发送顺序维护一个未完成请求列表，并且必须将该连接上收到的每个响应消息与尚未收到最终响应的最高顺序请求相关联（非1xx）响应。

5.7. 消息转发

如第2.3节所述，在处理HTTP请求和响应的过程中，中介可以扮演多种角色。 其中一些中介用于提高性能或可用性。其他一些则用于访问控制或内容过滤。 由于HTTP流具有类似于管道和过滤器架构的特性，因此中间设备对流的任一方向进行增强（或干扰）的程度没有固有限制。

非隧道中介必须按照第6.1节的规定实现Connection头字段，并排除仅针对传入连接而设计的字段的转发。

除非有防止无限请求循环的保护机制，否则中介不得将消息转发给自己。 通常，中介应能识别自己的服务器名称，包括任何别名、本地变体或字面IP地址，并直接响应此类请求。

5.7.1. 经由

“Via”头字段表示在用户代理与服务器之间（在请求时）或在源服务器与客户端之间（在响应时）存在中间协议和接收方，类似于电子邮件中的“Received”头字段（见[RFC5322]第3.6.7节）。 Via可用于跟踪消息转发、避免请求循环，以及识别请求/响应链中发送方的协议能力。

```
Via = 1#( 接收协议 RWS 接收者 [ RWS 注释 ] ) 接收协议 = [ 协议名称 "/" ] 协议版本
                                    ; 见第6.7节
received-by                 = (uri-host [":" port]) / pseudonym 伪名       pseudonym=
token
```

多个“Via”字段值表示每个已转发消息的代理或网关。 每个中间节点都会添加其自身关于消息接收方式的信息，以便最终结果按照转发接收者的顺序进行排序。

代理必须在它转发的每条消息中发送一个适当的Via头字段，如下所述。 HTTP到HTTP网关必须在每个入站请求消息中发送一个适当的Via头字段，并且可以在转发的响应消息中发送Via头字段。（注：Via头字段是HTTP协议中的一个标准字段，用于描述请求或响应消息的传输路径。在此上下文中，“HTTP到HTTP网关”指的是将HTTP请求从一个HTTP服务器转发到另一个HTTP服务器的中间设备。）。

对于每个中间节点，received-protocol表示消息上游发送者所使用的协议和协议版本。因此，Via字段值记录了请求/响应链所声明的协议能力，以便下游接收方能够看到这些能力；这对于确定在响应中或在后续请求中可以安全使用哪些向后不兼容的特性非常有用，如第2.6节所述。
为简洁起见，当收到的协议为HTTP时，会省略协议名称。

字段值的“received-by”部分通常表示接收方服务器或客户端（该服务器或客户端随后可能进行了消息转发）的主机名和可选端口号。 然而，如果真实主机名被视为敏感信息，发送方可以用假名替换它。 如果未提供端口号，接收方可以将其解释为该消息是在接收协议的默认TCP端口（如果有的话）上接收的。

发送方可以在Via头字段中生成注释，以标识每个接收方的软件，类似于User-Agent和Server头字段。 然而，Via字段中的所有注释都是可选的，接收方在转发消息前可以将其删除。

例如，一个请求消息可以从HTTP/1.0用户代理发送到一个代号为“fred”的内部代理，该代理使用HTTP/1.1将请求转发到p.example.net的公共代理，该公共代理通过将请求转发到www.example.com的源服务器来完成请求。 www.example.com收到的请求将包含以下Via头字段：

通过：1.0 fred，1.1 p.example.net

作为网络防火墙门户的中介，除非明确被允许，否则不应转发防火墙区域内主机的名称和端口。 若未被允许，此类中介应将防火墙后任何主机的接收方名称替换为该主机的适当化名。

如果多个Via头字段条目具有相同的接收协议值，则中介可以将这些有序的子序列合并为一个此类条目。 例如，

通过：1.0 ricky，1.1 ethel，1.1 fred，1.0 lucy 可以被折叠为

通过：1.0 ricky，1.1 mertz，1.0 lucy

发送方不应合并多个条目，除非这些条目都处于同一组织控制之下，并且主机已被替换为伪名。 发送方严禁合并接收协议值不同的条目。

5.7.2. 变换

一些中介体包含用于转换消息及其有效载荷的功能。 例如，代理可能会在图像格式之间进行转换，以节省缓存空间或减少慢速链路上的流量。 然而，当这些转换应用于旨在用于关键应用程序（如医学成像或科学数据分析）的有效载荷时，可能会出现操作问题，尤其是在使用完整性检查或数字签名来确保接收到的有效载荷与原始有效载荷完全相同的情况下。

如果HTTP到HTTP的代理被设计或配置为以语义上有意义的方式修改消息（即，除正常HTTP处理所需的修改之外，以一种对原始发送者具有重要意义或对下游接收者可能具有重要意义的方式更改消息），则该代理被称为“转换代理”。 例如，转换代理可能充当共享注释服务器（修改响应以包含对本地注释数据库的引用）、恶意软件过滤器、格式转码器或隐私过滤器。 选择该代理的任何客户端（或客户端组织）都假定这些转换是其所期望的。

如果代理服务器收到一个包含非完全限定域名主机名的请求目标，它可以在转发请求时将自身的域名添加到所收到主机名中。 如果请求目标包含完全限定域名，则代理服务器不得更改主机名。

在将接收到的请求目标转发给下一个入站服务器时，代理服务器不得修改请求目标中的“绝对路径”和“查询”部分，除非如上所述，用“/”或“*”替换空路径。

代理可以通过应用或移除传输编码（第4节）来修改消息体。

代理服务器不得转换包含“no-transform”缓存控制指令（见[RFC7234]第5.2节）的消息的有效负载（见[RFC7231]第3.3节）。

代理可以转换不包含“no-transform”缓存控制指令的消息的有效负载。 如果消息中尚未包含警告码为214（“已应用转换”）的警告头字段，则转换有效负载的代理必须添加该字段（参见[RFC7234]的第5.5节）。 转换有效负载的代理
200（OK）响应可以通过将响应状态码更改为203（非权威信息）（[RFC7231]的第6.3.4节）来进一步通知下游接收方已应用了转换。

代理服务器不应修改提供有关通信链端点、资源状态或所选表示（有效载荷除外）信息的头字段，除非该字段的定义明确允许此类修改，或者出于隐私或安全考虑认为有必要进行修改。

6. 连接管理

HTTP消息传递独立于底层传输层或会话层连接协议。 HTTP仅假定存在一个可靠的传输层，该层能按顺序传递请求和相应的响应。 将HTTP请求和响应结构映射到底层传输协议的数据单元上，不在本规范的范围之内。

如第5.2节所述，用于HTTP交互的具体连接协议由客户端配置和目标URI决定。 例如，“http”URI方案
（第2.7.1节）指出默认连接为基于IP的TCP连接，默认TCP端口为80，但客户端可能配置为通过其他连接、端口或协议使用代理。

HTTP实现需要进行连接管理，包括维护当前连接的状态、建立新连接或重用现有连接、处理连接上收到的消息、检测连接故障以及关闭每个连接。 大多数客户端并行维护多个连接，包括每个服务器端点多个连接。 大多数服务器设计为可维护数千个并发连接，同时控制请求队列以实现公平使用并检测拒绝服务攻击。

6.1. 连接

“Connection”头部字段允许发送方指示当前连接的期望控制选项。 为了避免给下游接收方造成混淆，代理或网关在转发消息之前，必须删除或替换任何收到的连接选项。

当使用除Connection以外的头部字段来提供当前连接的控制信息或与当前连接相关的信息时，发送方必须在Connection头部字段中列出相应的字段名。 代理或网关在转发消息之前，必须解析收到的Connection头部字段，对于该字段中的每个connection-option，从消息中移除与connection-option同名的所有头部字段，然后移除Connection头部字段本身（或用转发消息的中间设备自己的connection options替换它）。

因此，Connection头字段提供了一种声明性方式，用以区分仅针对直接接收方（“逐跳”）的头字段和针对链上所有接收方（“端到端”）的头字段，从而使消息具有自描述性，并允许部署未来特定于连接的扩展，而不必担心它们会被旧版中介盲目转发。

连接头字段的值遵循以下语法：

```
连接                    =1#连接选项 连接选项=令牌
```

连接选项不区分大小写。

发送方不得发送与旨在用于有效载荷所有接收方的标头字段相对应的连接选项。 例如，缓存控制（Cache-Control）绝不适宜作为连接选项（见[RFC7234]第5.2节）。

连接选项并不总是与消息中存在的头部字段相对应，因为如果没有与连接选项相关的参数，可能就不需要特定于连接的头部字段。 相反，如果接收到特定于连接的头部字段但没有相应的连接选项，通常表明该字段已被中间人不当转发，接收方应忽略该字段。

在定义新的连接选项时，规范作者应调查现有的头字段名称，并确保新的连接选项不会与已部署的头字段重名。 定义新的连接选项实际上就是保留了该潜在字段名，用于承载与连接选项相关的额外信息，因为发送方将该字段名用于其他用途是不明智的。

“close”连接选项是为发送方定义的，用于表示此连接将在响应完成后关闭。 例如，

连接：关闭

在请求或响应的头部字段中，如果包含有某个特定内容，则表示发送方将在当前请求/响应完成后关闭连接（见第6.6节）。

不支持持久连接的客户端必须在每个请求消息中发送“close”连接选项。

不支持持久连接的服务器必须在每个不具有1xx（信息性）状态码的响应消息中发送“close”连接选项。

6.2. 建立

本规范不涉及描述如何通过各种传输层或会话层协议建立连接。 每个连接仅适用于一个传输链路。

6.3. 持久性

HTTP/1.1默认使用“持久连接”，允许通过单个连接承载多个请求和响应。 “关闭”连接选项用于表示当前请求/响应结束后连接将不再保持。

HTTP实现应支持持久连接。

接收方根据最近接收到的消息的协议版本和Connection（连接）头字段（如果有的话）来判断连接是否为持久连接：

- 如果存在“close”连接选项，则连接在当前响应后将不会持续；否则，
- 如果收到的协议是HTTP/1.1（或更高版本），则当前响应结束后连接将保持；否则，
- 如果收到的协议是HTTP/1.0，并且存在“keep-alive”连接选项，接收方不是代理，且接收方希望遵守HTTP/1.0的“keep-alive”机制，则当前响应结束后，连接将保持；否则，
- 当前响应结束后，连接将关闭。

客户端可以在持久连接上发送额外的请求，直到它发送或接收到一个“close”连接选项，或者接收到一个没有“keep-alive”连接选项的HTTP/1.0响应。

为了保持持久性，连接上的所有消息都需要有一个自定义的消息长度（即，不是通过连接关闭来定义的），如第3.3节所述。服务器必须读取整个请求消息体，或在发送其响应后关闭连接，否则持久连接上的剩余数据将被误解为下一个请求。同样，如果客户端打算在后续请求中重用同一连接，则必须读取整个响应消息体。

代理服务器不得与HTTP/1.0客户端保持持久连接（有关许多HTTP/1.0客户端实现的Keep-Alive头字段的问题的信息和讨论，请参阅[RFC2068]的第19.7.1节）。

有关与HTTP/1.0客户端向后兼容性的更多信息，请参阅附录A.1.2。

6.3.1. 重试请求

连接可以在任何时候关闭，无论是有意还是无意。实现时应当预见到需要从异步关闭事件中恢复。

当一个入站连接被提前关闭时，如果所有被中止的请求都采用了幂等方法（见[RFC7231]第4.2.2节），客户端可以打开一个新的连接并自动重新传输这些被中止的请求序列。 代理服务器不得自动重试
非幂等请求。

用户代理绝不能自动重试使用非幂等方法的请求，除非它有某种方法能确定该请求的语义实际上是幂等的（无论使用何种方法），或者有某种方法能检测到原始请求从未被应用过。 例如，如果用户代理（通过设计或配置）知道对给定资源的POST请求是安全的，则可以自动重复该请求。 同样，专门设计用于操作版本控制存储库的用户代理可能能够在连接失败后通过检查目标资源的修订版，撤销或修复部分应用的任何更改，然后自动重试失败的请求，从而从部分失败的情况中恢复。

客户端不应自动重试失败的自动重试。

6.3.2. 流水线

支持持久连接的客户端可以“流水线”发送请求（即，在不等待每个响应的情况下发送多个请求）。 如果流水线发送的请求都使用了安全方法（见[RFC7231]第4.2.1节），则服务器可以并行处理这些请求，但必须按照接收请求的顺序发送相应的响应。

使用流水线发送请求的客户端，如果在收到所有相应响应之前连接关闭，则应重试未应答的请求。 在连接失败（即服务器在最后一个完整响应中未明确关闭连接）后重试流水线请求时，客户端不得在连接建立后立即进行流水线操作，因为如果在一个过早关闭的连接上发送多个请求，先前流水线中的第一个剩余请求可能会导致错误响应，而这些响应可能会再次丢失（参见第6.6节中描述的TCP重置问题）。

幂等方法（见[RFC7231]第4.2.2节）对流水线处理具有重要意义，因为它们可以在连接失败后自动重试。 除非用户代理有检测并从涉及流水线序列的部分失败情况中恢复的方法，否则在非幂等方法之后，用户代理不应流水线处理请求，直至收到该方法的最终响应状态码。

接收流水线请求的中介在将请求转发入站时，可以流水线处理这些请求，因为它可以依赖出站用户代理来确定哪些请求可以安全地流水线处理。 如果在收到响应之前入站连接失败，且所有请求都采用幂等方法，则流水线中介可以尝试重试尚未收到响应的请求序列；否则，流水线中介应转发所有收到的响应，然后关闭相应的出站连接，以便出站用户代理能够相应地进行恢复。

6.4. 并发

客户端应限制其与给定服务器保持的同时打开连接的数量。

HTTP的早期版本设定了一个特定的连接数量上限，但人们发现这对于许多应用来说并不切实际。因此，本规范并未强制规定特定的最大连接数量，而是鼓励客户端在打开多个连接时保持谨慎。

通常使用多个连接来避免“队头阻塞”问题，即一个请求占用大量时间
服务器端处理和/或处理大数据量会阻塞同一连接上的后续请求。 然而，每个连接都会消耗服务器资源。 此外，在拥塞的网络中使用多个连接可能会产生不良的副作用。

请注意，服务器可能会拒绝其认为具有滥用性质或典型拒绝服务攻击特征的流量，例如来自单个客户端的过多开放连接。

6.5. 失败与超时

服务器通常会有一个超时值，超过该值后，服务器将不再维护非活动连接。 由于客户端可能会通过同一代理服务器建立更多连接，因此代理服务器可能会将此超时值设置得更高。
使用持久连接对客户端或服务器的超时时长（或是否存在超时）没有要求。

希望超时的客户端或服务器应在连接上发出优雅关闭（graceful close）的指令。 实现时应持续监控打开的连接，以接收关闭信号并做出适当响应，因为连接双方及时关闭可以回收分配的系统资源。（注：1.“graceful close”翻译为“优雅关闭”，这是一个网络通信领域的术语，指的是在断开连接时，双方都能以一种有礼貌且高效的方式结束连接，从而避免对系统造成不必要的干扰和负担。2.“prompt closure”翻译为“及时关闭”，指的是在连接建立后，如果不再需要使用，应尽快关闭连接，以释放占用的系统资源。3.“reclaimed”翻译为“回收”，指的是系统资源在使用完毕后，可以被重新分配给其他需要使用的进程或线程。）。

客户端、服务器或代理可以在任何时候关闭传输连接。 例如，在服务器决定关闭“空闲”连接的同时，客户端可能已经开始发送新的请求。 从服务器的角度来看，连接是在空闲状态下被关闭的，但从客户端的角度来看，请求正在进行中。

服务器应尽可能维持持久连接，并允许底层传输的流量控制机制解决临时过载问题，而不是终止连接并期望客户端会重试。 后一种技术可能会加剧网络拥塞。（注：1）“persistent connections”翻译为“持久连接”，指的是在服务器和客户端之间建立的能够长时间存在的连接，用于传输数据。2）“flow-control mechanisms”翻译为“流量控制机制”，是一种用于控制网络流量，防止网络拥塞的技术。3）“temporary overloads”翻译为“临时过载”，指的是网络或服务暂时无法处理过多的请求或数据，导致处理速度减慢或服务中断。4）“clients will retry”翻译为“客户端会重试”，指的是当网络连接中断或服务暂时不可用时，客户端会尝试重新连接或重新发送请求，以恢复服务。）。

发送消息体的客户端在传输请求时应监控网络连接是否有错误响应。 如果客户端收到响应，表明服务器不希望接收消息体并正在关闭连接，则客户端应立即停止传输消息体并关闭其连接端。

6.6. 拆卸

连接头字段（第6.1节）提供了一个“关闭”连接选项，当发送方希望在当前请求/响应对之后关闭连接时，应发送此选项。

发送“close”连接选项的客户端（在包含“close”的请求之后）不得在该连接上发送进一步的请求，并且必须在读取与此请求对应的最终响应消息后关闭连接。

接收到“close”连接选项的服务器，在向包含“close”的请求发送最终响应后，必须主动关闭连接（见下文）。 服务器应在该连接的最终响应中发送“close”连接选项。
服务器不得处理在该连接上收到的任何后续请求。

发送“close”连接选项的服务器在发送包含“close”的响应后，必须主动关闭连接（见下文）。 服务器不得处理该连接上收到的任何后续请求。

接收到“close”连接选项的客户端必须停止在该连接上发送请求，并在读取包含“close”的响应消息后关闭连接；如果该连接上已发送了其他流水线请求，则客户端不应假定这些请求会被服务器处理。

如果服务器立即关闭TCP连接，那么客户端将面临无法读取最后一个HTTP响应的重大风险。 如果服务器在完全关闭的连接上从客户端接收到额外数据，例如客户端在收到服务器响应之前发送的另一个请求，则服务器的TCP栈将向客户端发送一个重置包；遗憾的是，在客户端的HTTP解析器能够读取和解释这些数据之前，重置包可能会清除客户端未确认的输入缓冲区。

为了避免TCP重置问题，服务器通常会分阶段关闭连接。 首先，服务器执行半关闭操作，仅关闭读/写连接中的写端。 然后，服务器继续从连接中读取数据，直到接收到客户端的相应关闭操作，或者直到服务器合理确信其TCP栈已收到客户端对包含服务器最后响应的数据包的确认。 最后，服务器完全关闭连接。

目前尚不清楚重置问题是否仅限于TCP，还是也可能出现在其他传输连接协议中。

6.7. 升级

“Upgrade”头字段旨在为在同一连接上从HTTP/1.1过渡到其他协议提供一种简单机制。 客户端可以在请求的Upgrade头字段中发送一个协议列表，以邀请服务器在发送最终响应之前，按照优先级从高到低切换到这些协议中的一个或多个。 如果服务器希望继续在该连接上使用当前协议，则可以忽略收到的Upgrade头字段。 Upgrade不能用于强制更改协议。

```
Upgrade          = 1#protocol

protocol         = 协议名称 ["/" 协议版本] 协议名称         = 标记
协议版本 = 标记
```

发送101（切换协议）响应的服务器必须发送一个Upgrade头部字段，以指示连接将要切换到的新协议；如果正在切换多个协议层，发送方必须按层递增的顺序列出这些协议。
服务器不得切换到客户端在相应请求的Upgrade头部字段中未指明的协议。
A.

服务器可以选择忽略客户端指示的优先级顺序，并根据其他因素（如请求的性质或服务器当前的负载）来选择新的协议。

发送426（需要升级）响应的服务器必须发送一个Upgrade头部字段，以按优先级降序指示可接受的协议。

服务器可以在任何其他响应中发送Upgrade头部字段，以声明其支持升级到所列协议，并按优先级从高到低排序，以便在适用于未来请求时使用。【注】1. Upgrade头部字段：在HTTP协议中，Upgrade头部字段用于指示服务器支持升级到哪些协议。2. 优先级从高到低排序：这里指的是服务器支持的协议列表，优先级从高到低排列，表示如果客户端请求的协议不在列表中，服务器会优先尝试升级到列表中的协议。

以下是一个客户端发送的假设示例：

GET /hello.txt HTTP/1.1 Host:
www.example.com Connection:
upgrade
升级：HTTP/2.0、SHTTP/1.3、IRC/6.9、RTA/x11

协议更改后，应用层通信的能力和性质完全取决于所选的新协议。 然而，在发送101（切换协议）响应后，服务器应立即继续响应原始请求，就像它在新协议中收到了等效请求一样（即，在协议更改后，服务器仍有未处理的请求需要处理，并且预计会在不需要重复请求的情况下完成处理）。

例如，如果在GET请求中接收到Upgrade头字段，并且服务器决定切换协议，那么它首先会以一个（具体的响应内容）进行响应
在HTTP/1.1中，101（切换协议）消息之后，紧接着是目标资源上新协议对GET请求的等效响应。 这允许连接升级到与HTTP具有相同语义的协议，而无需额外往返的延迟成本。 除非新协议能够支持接收到的消息语义，否则服务器不得切换协议；任何协议都可以支持OPTIONS请求。

以下是对上述假设请求的示例响应：

HTTP/1.1 101 切换协议 连接：升级
升级：HTTP/2.0

[... 数据流切换至HTTP/2.0，并对“GET /hello.txt”请求作出相应（如新协议所定义）的响应 ...]

当发送Upgrade时，发送方还必须发送一个包含“upgrade”连接选项的Connection头字段（见第6.1节），以防止Upgrade被可能未实现所列协议的中间设备意外转发。 服务器必须忽略在HTTP/1.0请求中收到的Upgrade头字段。

在客户端完全发送请求消息之前，它不能在连接上开始使用已升级的协议（即，客户端不能在消息发送过程中更改其正在使用的协议）。如果服务器同时接收到带有“100-continue”预期的Upgrade和Expect头字段（见[RFC7231]第5.1.1节），则服务器必须在发送101（切换协议）响应之前，先发送一个100（继续）响应。

Upgrade 头部字段仅适用于在现有连接之上切换协议；它不能用于切换底层连接（传输）协议，也不能用于将现有通信切换到不同的连接。 对于这些目的，使用3xx（重定向）响应（[RFC7231] 的第 6.4 节）更为合适。

本规范仅定义了协议名称“HTTP”，供第2.6节中HTTP版本规则以及本规范未来更新所定义的超文本传输协议系列使用。 其他标记应按照第8.6节中定义的注册程序向互联网号码分配机构（IANA）进行注册。

7. ABNF列表扩展：#规则

[RFC5234]中ABNF规则的一个#rule扩展用于提高某些头字段值定义的可读性。

定义了一个类似于“*”的构造“#”，用于定义
逗号分隔的元素列表。完整形式为“<n>#<m>element”，表示至少有<n>个且至多有<m>个元素，每个元素之间用单个逗号（“,”）分隔，并可包含可选的空白（OWS）。

在使用列表构造的任何生产环境中，发送方严禁生成空列表元素。 换言之，发送方必须生成满足以下语法的列表：

1#元素 => 元素 *(OWS "," OWS 元素) 且：

#元素 => [1#元素]，且对于 n ≥

1 且 m > 1 的情况：

<n>#<m>元素 => 元素 <n-1>*<m-1>(空格 "," 空格 元素)

为了与旧版列表规则兼容，接收方必须解析并忽略合理数量的空列表元素：数量足以处理发送方合并值时常见的错误，但又不至于多到可以被用作拒绝服务机制。 换言之，接收方必须接受满足以下语法的列表：

#元素 => [("," / 元素) *(空白符 "," [空白符 元素])] 1#元素 => *("," 空白符) 元素 *(空白符 "," [空白符 元素])

空元素不计入当前元素的计数。例如，给定以下ABNF（抽象巴科斯范式）生成式：

示例列表 =1#示例列表元素 示例列表元素=标记；见第3.2.6节

以下是示例列表的有效值（不包括双引号，双引号仅用于分隔）：

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

相比之下，以下值将是无效的，因为示例列表生成至少需要一个非空元素：

""
","
",,"
",, "

附录B展示了在列表构造被展开后，为收件人收集的抽象语法标记法（ABNF）。

8. IANA（互联网号码分配机构）考虑因素

8.1. 头字段注册

HTTP头字段在“消息头”注册表中注册，该注册表位于
<http://www.iana.org/assignments/message-headers/>。

本文档定义了以下HTTP头字段，因此“永久消息头字段名称”注册表已相应更新（见[BCP90]）。

头部字段名称	协议	状态	参考
连接	http	标准	第6.1节
Content-Length	http	标准	第3.3.2节
主机	http	标准	第5.4节
TE	http	标准	第4.3节
预告片	http	标准	第4.4节
传输编码	http	标准	第3.3.1节
升级	http	标准	第6.7节
协议	http	标准	第5.7.1节

此外，头部字段名“Close”已被注册为“保留”字段，因为将其作为HTTP头部字段可能会与Connection头部字段的“close”连接选项（第6.1节）发生冲突。

头部字段名称	协议	状态	参考
关闭	http	保留	第8.1节

变更控制者是：“IETF（iesg@ietf.org）- 互联网工程任务组”。

8.2. URI方案注册

IANA负责维护URI方案[BCP115]的注册表，注册表位于
<http://www.iana.org/assignments/uri-schemes/>。

本文档定义了以下URI方案，因此“永久URI方案”注册表已相应更新。

URI方案	描述	参考
http	超文本传输协议	第2.7.1节
https	安全超文本传输协议	第2.7.2节

8.3. 互联网媒体类型注册

IANA（互联网号码分配机构）负责维护互联网媒体类型注册表[BCP13]，该注册表位于
<http://www.iana.org/assignments/media-types>（该链接指向互联网号码分配机构（IANA）的媒体类型分配页面）。

本文档作为互联网媒体类型“message/http”和“application/http”的规范。 以下内容已在IANA注册。（注：1. IANA：Internet Assigned Numbers Authority，即互联网号码分配机构，负责全球互联网的号码分配和管理。2. "message/http"：一种用于传输HTTP消息的媒体类型。3. "application/http"：一种用于传输HTTP应用程序的媒体类型。）。

8.3.1. 互联网媒体类型 message/http

message/http类型可用于封装单个HTTP请求或响应消息，前提是该消息遵循所有“message”类型在行长和编码方面的MIME限制。

类型名称： message 子类型名

称： http 必选参数：

无

可选参数： version（版本）、msgtype（消息类型）

版本： 所附消息的HTTP版本号（例如，“1.1”）。 若未给出，则可从正文的第一行确定版本。

msgtype： 消息类型——“请求”或“响应”。 若未给出，则可从正文的第一行确定类型。

编码注意事项： 仅允许使用“7位”、“8位”或“二进制”编码

安全性考虑： 见第9节 互操作性

考虑： 不适用

已发布规范： 本规范（见第8.3.1节）。使用此媒体类型的应用程序：

不适用

片段标识符注意事项： 无

附加信息：

幻数： 不适用

此类型的已弃用别名： 无 文件扩展名： 无

Macintosh文件类型代码： 不适用

如需获取更多信息，请联系以下人员及电子邮件地址：详见“作者地址”部分。

预期用途： 通用 使用限

制： 无

作者： 见作者地址部分。变更控制者：

IESG

8.3.2. 互联网媒体类型 application/http

application/http类型可用于封装一个或多个HTTP请求或响应消息的管道（不混合）。

类型名称： application 子类型

名称： http 必需参数：

无

可选参数： version（版本）、msgtype（消息类型）

版本： 所包含消息的HTTP版本号（例如，“1.1”）。 若未给出，则可从正文的第一行确定版本。

msgtype： 消息类型 —— “请求”或“响应”。 若未给出，则可从正文的第一行确定类型。

编码注意事项： 此类型包含的HTTP消息采用“二进制”格式；通过电子邮件传输时，需要使用适当的内容传输编码。

安全性考虑： 见第9节 互操作性

考虑： 不适用

已发布规范： 本规范（见第8.3.2节）。使用此媒体类型的应用程序：

不适用

片段标识符注意事项： 无

附加信息：

此类型的已弃用别名： 无 幻数： 无

文件扩展名： 不适用

Macintosh文件类型代码： N/A

如需获取更多信息，请联系以下人员及电子邮件地址：详见“作者地址”部分。

预期用途： 通用 使用限

制： 无

作者： 见作者地址部分。变更控制者：

IESG

8.4. 传输编码注册表

“HTTP传输编码注册表”定义了传输编码名称的命名空间。 该注册表位于 <http://www.iana.org/assignments/http-parameters>。

8.4.1. 程序

注册必须包含以下字段：

1. 名字

2. 描述

3. 规范文本指针

传输编码的名称不得与内容编码的名称重叠（见[RFC7231]第3.1.2.1节），除非编码转换完全相同，如第4.2节中定义的压缩编码。

添加到此命名空间的数值需要经过IETF审核（参见[RFC5226]第4.1节），并且必须符合本规范中定义的传输编码目的。

使用程序名称来识别编码格式的做法并不可取，也不建议在未来用于编码。

8.4.2. 注册

“HTTP传输编码注册表”已更新，包含以下注册信息：

名称	描述	参考
chunked	分块传输	第4.1节
compress	UNIX "compress" 数据格式 [Welch]	第4.2.1节
deflate	“deflate”压缩数据 ([RFC1951]) 在 "zlib" 数据内部格式（[RFC1950]）	第4.2.2节
gzip	GZIP文件格式[RFC1952]	第4.2.3节
x-compress	已弃用（compress的别名）	第4.2.1节
x-gzip	已弃用（gzip的别名）	第4.2.3节

8.5. 内容编码注册

IANA（互联网号码分配机构）负责维护位于（具体位置）的“HTTP内容编码注册表”<http://www.iana.org/assignments/http-parameters>。

“HTTP内容编码注册表”已更新，包含以下注册信息：

名称	描述	参考
compress	UNIX "compress" 数据格式 [Welch]	第4.2.1节
deflate	“deflate”压缩数据 ([RFC1951]) 在 "zlib" 数据内部格式（[RFC1950]）	第4.2.2节
gzip	GZIP文件格式[RFC1952]	第4.2.3节
x-compress	已弃用（compress的别名）	第4.2.1节
x-gzip	已弃用（gzip的别名）	第4.2.3节

8.6. 升级令牌注册表

“超文本传输协议（HTTP）升级令牌注册表”定义了用于在Upgrade头字段中标识协议的协议名称令牌的命名空间。 该注册表位于<http://www.iana.org/assignments/http-upgrade-tokens>（此为超文本传输协议升级令牌的分配信息，由互联网号码分配机构维护）。

8.6.1. 程序

每个已注册的协议名称都关联有联系信息以及一组可选的规范，这些规范详细说明了连接升级后将如何处理。

注册遵循“先到先得”的原则（详见[RFC5226]第4.1节），并受以下规则约束：

1. 协议名称令牌一旦注册，将永久保持注册状态。

2. 注册必须指明负责注册的责任方。

3. 注册时必须指定一个联系人。

4. 注册信息中可以指定与该令牌相关联的一组规范。 这些规范无需公开。（注：此译文对原文进行了直译，保留了原文的语义和结构。其中，“registration”翻译为“注册信息”，“token”翻译为“令牌”，以符合HTTP标准协议中的术语。）。

5. 注册时，应指定一组预期的“协议版本”令牌，这些令牌与注册时的特定令牌相关联。

6. 责任方可以随时更改注册信息。互联网号码分配机构（IANA）将记录所有此类更改，并根据请求提供这些信息。

7. IESG可以重新分配协议令牌的责任。这通常只在无法联系到责任方的情况下使用。

此HTTP升级令牌的注册程序取代了先前在[RFC2817]第7.2节中定义的程序。

8.6.2. 升级令牌注册

升级令牌注册表中的“HTTP”条目已更新为以下注册信息：

数值	描述	预期版本令牌	参考
HTTP	超文本传输协议	任意数字.数字（例如，“2.0”）	第2.6节

责任方为：“IETF（iesg@ietf.org）- 互联网工程任务组”。

9. 安全注意事项

本节旨在向开发者、信息提供者和用户告知与HTTP消息语法、解析和路由相关的已知安全注意事项。有关HTTP语义和有效载荷的安全注意事项，请参阅[RFC7231]。

9.1. 建立权威

HTTP依赖于权威响应的概念：即在响应消息发出时，由目标URI内标识的权威机构（或在其指示下）确定的对给定目标资源状态的请求的最恰当响应。 从...提供响应
非权威性来源（如共享缓存）通常有助于提高性能和可用性，但前提是该来源值得信任，或者可以安全地使用不受信任的响应。

不幸的是，建立权威可能很困难。例如，网络钓鱼攻击就是利用用户对权威的认知，通过展示相似的品牌标识来误导这种认知

超文本，可能通过用户信息（userinfo）来模糊权限组件（见第2.7.1节）。 用户代理可以通过以下方式减少网络钓鱼攻击的影响：使用户在执行操作前能够轻松检查目标URI；当存在用户信息时，通过显著区分（或拒绝）用户信息；以及当引用文档来自未知或不受信任的来源时，不发送存储的凭证和Cookie。

当在授权组件中使用注册名称时，“http”URI方案（第2.7.1节）依赖于用户的本地名称解析服务来确定在何处可以找到权威响应。 这意味着，对用户网络主机表、缓存名称或名称解析库的任何攻击都可能成为攻击建立权威的途径。 同样，用户选择的域名服务（DNS）服务器以及从中获取解析结果的服务器层级结构，都可能影响地址映射的真实性；DNS安全扩展（DNSSEC，[RFC4033]）是提高真实性的方法之一。

此外，在获取IP地址后，为“http”URI建立权威性容易受到互联网协议路由攻击。

“https”方案（第2.7.2节）旨在防止（或至少揭示）许多在建立信任关系时可能发生的潜在攻击，前提是协商的TLS连接是安全的，并且客户端正确验证了通信服务器的身份与目标URI的权威组件（authority component）相匹配（参见[RFC2818]）。
正确实现此类验证可能很困难（见[Georgiev]）。

9.2. 中介机构的风险

就其本质而言，HTTP中间设备就是中间人，因此，它们为中间人攻击提供了机会。
中介机构运行的系统若被攻破，可能会引发严重的安全和隐私问题。 中介机构可能获取与安全相关的信息、个人用户和组织的个人信息，以及属于用户和内容提供商的专有信息。 一个被攻破的中介机构，或者一个在实施或配置时未考虑安全和隐私因素的中介机构，可能会被用于实施一系列潜在的攻击。

如[RFC7234]第8节所述，包含共享缓存的中介尤其容易受到缓存投毒攻击。

实现者需要考虑其设计和编码决策以及向操作员提供的配置选项（尤其是默认配置）对隐私和安全的影响。

用户需要意识到，中介机构并不比运营它们的人更值得信赖；HTTP协议本身无法解决这个问题。

9.3. 通过协议元素长度进行的攻击

由于HTTP主要使用文本格式、字符分隔的字段，解析器往往容易受到基于发送非常长（或非常慢）的数据流的攻击，尤其是在实现预期的协议元素没有预定义长度的情况下。

为了提升互操作性，对请求行（第3.1.1节）和头字段（第3.2节）的最小尺寸限制给出了具体建议。 这些是最低建议值，即使资源有限的实现也能支持；预计大多数实现会选择更高的限制值。

服务器可以拒绝请求目标过长（见[RFC7231]第6.5.12节）或请求载荷过大（见[RFC7231]第6.5.11节）的消息。 与容量限制相关的其他状态码已由HTTP扩展定义[RFC6585]。

接收方应谨慎限制对其他协议元素的处理程度，包括（但不限于）请求方法、响应状态短语、头部字段名、数值和正文块。 若不限制此类处理，可能导致缓冲区溢出、算术溢出，或增加遭受拒绝服务攻击的风险。

9.4. 响应拆分

响应拆分（亦称CRLF注入）是一种常见技术，用于针对Web使用的各种攻击，它利用了HTTP消息分帧的基于行的特性以及持久连接上请求与响应的有序关联[Klein]。 当请求通过共享缓存时，这种技术可能具有特别大的破坏性。

响应拆分利用了服务器（通常在应用服务器内部）中的漏洞，攻击者可以在请求的某个参数中发送编码数据，这些数据稍后会在响应的任何响应头字段中被解码并回显。 如果解码后的数据被精心构造，使其看起来像是响应已经结束，并且

后续响应已开始，该响应已被拆分，且拆分后看似第二个响应的内容由攻击者控制。
攻击者随后可以在同一持久连接上发出任何其他请求，并诱使接收方（包括中间节点）相信拆分后的第二部分是对第二个请求的权威性回答。

例如，请求目标中的参数可能被应用服务器读取并在重定向中重复使用，从而导致相同的参数在响应的Location头字段中回显。 如果应用程序对参数进行解码，但在将其放入响应字段时未正确编码，则攻击者可以发送编码的CRLF字节组和其他内容，使应用程序的单个响应看起来像两个或更多个响应。

针对响应拆分的常见防御措施是过滤掉看起来像编码的CR和LF（例如，“%0D”和“%0A”）的数据请求。
然而，这假设应用服务器仅执行URI解码，而不进行更复杂的数据转换，如字符集转码、XML实体转换、base64解码、sprintf重新格式化等。 一种更有效的缓解方法是阻止除服务器核心协议库外的任何组件在头部部分发送CR或LF，这意味着将头部字段的输出限制为过滤不良字节的API，并且不允许应用服务器直接写入协议流。

9.5. 请求欺骗

请求欺骗（[Linhart]）是一种利用不同接收方在协议解析上的差异，将额外的请求（这些请求可能因策略而被阻止或禁用）隐藏在看似无害的请求中的技术。 与响应拆分类似，请求欺骗也可能导致针对HTTP使用的各种攻击。

本规范引入了关于请求解析的新要求，特别是在消息帧方面
第3.3.3节，旨在降低请求欺骗的有效性。

9.6. 消息完整性

HTTP并未定义特定的机制来确保消息的完整性，而是依赖于底层传输协议的错误检测能力以及长度或（其他相关参数）的使用
使用数据块分隔的帧结构来检测完整性。 可以通过可扩展机制有选择地在消息中添加额外的完整性机制，如对内容应用哈希函数或数字签名

元数据头字段。 历史上，由于大多数HTTP通信的非正式性，缺乏单一的完整性机制被认为是合理的。 然而，随着HTTP作为信息访问机制的普及，在那些消息完整性验证至关重要的环境中，其使用日益增加。

鼓励用户代理实现可配置的方法来检测和报告消息完整性的失败，以便在需要完整性的环境中启用这些方法。例如，当用于查看病史或药物相互作用信息的浏览器检测到协议中的此类信息在传输过程中不完整、已过期或已损坏时，需要向用户提示。 此类机制可通过用户代理扩展或响应中存在的消息完整性元数据来选择性地启用。 至少，当需要此类验证时，用户代理应提供某种指示，使用户能够区分完整和不完整的响应消息（第3.4节）。

9.7. 消息机密性

HTTP依赖于底层传输协议来提供所需的消息保密性。 HTTP经过专门设计，使其独立于传输协议，从而可以在多种不同形式的加密连接上使用，这些传输方式的选择由URI方案的选择或用户代理配置中的设置来确定。

“https”方案可用于标识需要保密连接的资源，如第2.7.2节所述。

9.8. 服务器日志信息的隐私

服务器能够保存用户随时间推移的请求中的个人数据，这些数据可能揭示用户的阅读习惯或感兴趣的主题。 特别是，在中介服务器上收集的日志信息通常包含用户代理在多个网站上的交互历史，这些历史信息可以追溯到单个用户。

HTTP日志信息本质上具有保密性；其处理往往受到法律法规的限制。 日志信息需要安全存储，并遵循适当的分析指南。对单个条目中的个人信息进行匿名化处理有所帮助，但通常不足以防止基于与其他访问特征的相关性重新识别真实的日志痕迹。 因此，即使密钥是伪匿名的，与特定客户端绑定的访问痕迹也不宜公开。

为了最大限度地降低被盗或意外发布的风险，日志信息中应清除个人身份信息，包括用户标识符、IP地址等
用户提供的查询参数，一旦这些信息不再为支持安全、审计或欺诈控制等运营需求所必需时，即应删除。

10. 致谢

本版HTTP/1.1建立在RFC 1945、RFC 2068、RFC 2145和RFC 2616的众多贡献之上，其中包括前几任作者、编辑和工作组主席所做出的重大贡献：蒂姆·伯纳斯-李（Tim Berners-Lee）、阿里·卢奥托宁（Ari Luotonen）、罗伊·T·菲尔丁（Roy T. Fielding）、亨里克·弗雷斯蒂克·尼尔森（Henrik Frystyk Nielsen）、吉姆·盖蒂斯（Jim Gettys）、杰弗里·C·莫古尔（Jeffrey C. Mogul）、拉里·马辛特（Larry Masinter）和保罗·J·利奇（Paul J. Leach）。
马克·诺丁汉（Mark Nottingham）作为工作组主席监督了这项工作。

自1999年以来，以下贡献者通过报告错误、提出有见地的问题、起草或审阅文本以及评估未决问题，为改进HTTP规范做出了贡献：

Adam Barth、Adam Roach、Addison Phillips、Adrian Chadd、Adrian Cole、Adrien W. de Croy、Alan Ford、Alan Ruttenberg、Albert Lunde、Alek Storm、Alex Rousskov、Alexandre Morgaut、Alexey Melnikov、Alisha Smith、Amichai Rothman、Amit Klein、Amos Jeffries、Andreas Maier、Andreas Petersson、Andrei Popov、Anil Sharma、Anne van Kesteren、Anthony Bryan、Asbjorn Ulsberg、Ashok Kumar、Balachander Krishnamurthy、Barry Leiba、Ben Laurie、Benjamin Carlyle、Benjamin Niven-Jenkins、Benoit Claise、Bil Corry、Bill Burke、Bjoern Hoehrmann、Bob Scheifler、Boris Zbarsky、Brett Slatkin、Brian Kell、Brian McBarron、Brian Pane、Brian Raymor、Brian Smith、Bruce Perens、Bryce Nesbitt、Cameron Heavon-Jones、Carl Kugler、Carsten Bormann、Charles Fry、Chris Burdess、Chris Newman、Christian Huitema、Cyrus Daboo、Dale Robert Anderson、Dan Wing、Dan Winship、Daniel Stenberg、Darrel Miller、Dave Cridland、Dave Crocker、Dave Kristol、Dave Thaler、David Booth、David Singer、David W. Morris、Diwakar Shetty、Dmitry Kurochkin、Drummond Reed、Duane Wessels、Edward Lee、Eitan Adler、Eliot Lear、Emile Stephan、Eran Hammer-Lahav、Eric D. Williams、Eric J. Bowman、Eric Lawrence、Eric Rescorla、Erik Aronesty、EungJun Yi、Evan Prodromou、Felix Geisendoerfer、Florian Weimer、Frank Ellermann、Fred Akalin、Fred Bohle、Frederic Kayser、Gabor Molnar、Gabriel Montenegro、Geoffrey Sneddon、Gervase Markham、Gili Tzabari、Grahame Grieve、Greg Slepak、Greg Wilkins、Grzegorz Calkowski、Harald Tveit Alvestrand、Harry Halpin、Helge Hess、Henrik Nordstrom、Henry S. Thompson、Henry Story、Herbert van de Sompel、Herve Ruellan、Howard Melman、Hugo Haas、Ian Fette、Ian Hickson、Ido Safruti、Ilari Liusvaara、Ilya Grigorik、Ingo Struck、J. Ross Nicoll、James Cloos、James H. Manger、James Lacey、James M. Snell、Jamie

Lokier、Jan Algermissen、Jari Arkko、Jeff Hodges（他提出了“有效请求URI”这一术语）、Jeff Pinner、Jeff Walden、Jim Luther、Jitu Padhye、Joe D. Williams、Joe Gregorio、Joe Orton、Joel Jaeggli、John C. Klensin、John C. Mallery、John Cowan、John Kemp、John Panzer、John Schneider、John Stracke、John Sullivan、Jonas Sicking、Jonathan A. Rees、Jonathan Billington、Jonathan Moore、Jonathan Silvera、Jordi Ros、Joris Dobbelsteen、Josh Cohen、Julien Pierre、Jungshik Shin、Justin Chapweske、Justin Erenkrantz、Justin James、Kalvinder Singh、Karl Dubost、Kathleen Moriarty、Keith Hoffman、Keith Moore、Ken Murchison、Koen Holtman、Konstantin Voronkov、Kris Zyp、Leif Hedstrom、Lionel Morand、Lisa Dusseault、Maciej Stachowiak、Manu Sporny、Marc Schneider、Marc Slemko、Mark Baker、Mark Pauley、Mark Watson、Markus Isomaki、Markus Lanthaler、Martin J. Duerst、Martin Musatov、Martin Nilsson、Martin Thomson、Matt Lynch、Matthew Cox、Matthew Kerwin、Max Clark、Menachem Dodge、Meral Shirazipour、Michael Burrows、Michael Hausenblas、Michael Scharf、Michael Sweet、Michael Tuexen、Michael Welzl、Mike Amundsen、Mike Belshe、Mike Bishop、Mike Kelly、Mike Schinkel、Miles Sabin、Murray S. Kucherawy、Mykyta Yevstifeyev、Nathan Rixham、Nicholas Shanks、Nico Williams、Nicolas Alvarez、Nicolas Mailhot、Noah Slater、Osama Mazahir、Pablo Castro、Pat Hayes、Patrick R. McManus、Paul E. Jones、Paul Hoffman、Paul Marquess、Pete Resnick、Peter Lepeska、Peter Occil、Peter Saint-Andre、Peter Watkins、Phil Archer、Phil Hunt、Philippe Mougin、Phillip Hallam-Baker、Piotr Dobrogost、Poul-Henning Kamp、Preethi Natarajan、Rajeev Bector、Ray Polk、Reto Bachmann-Gmuer、Richard Barnes、Richard Cyganiak、Rob Trace、Robby Simpson、Robert Brewer、Robert Collins、Robert Mattson、Robert O’Callahan、Robert Olofsson、Robert Sayre、Robert Siemer、Robert de Wilde、Roberto Javier Godoy、Roberto Peon、Roland Zink、Ronny Widjaja、Ryan Hamilton、S. Mike Dierken、Salvatore Loreto、Sam Johnston、Sam Pullara、Sam Ruby、Saurabh Kulkarni、Scott Lawrence（谁）（保留了原始问题列表），肖恩·B·帕尔默（Sean B. Palmer）、肖恩·特纳（Sean Turner）、塞巴斯蒂安·巴努德（Sebastien Barnoud）、谢恩·麦卡伦（Shane McCarron）、大津茂树（Shigeki Ohtsu）、西蒙·亚德（Simon Yarde）、斯特凡·艾辛（Stefan Eissing）、斯特凡·蒂尔科夫（Stefan Tilkov）、斯特凡诺斯·哈尔哈拉基斯（Stefanos Harhalakis）、斯特凡·博茨梅尔（Stephane Bortzmeyer）、斯蒂芬·法雷尔（Stephen Farrell）、斯蒂芬·肯特（Stephen Kent）、斯蒂芬·卢丁（Stephen Ludin）、斯图尔特·威廉姆斯（Stuart Williams）、苏布·阿拉马拉朱（Subbu Allamaraju）、苏布拉马尼安·穆内萨米（Subramanian Moonesamy）、苏珊·黑尔斯（Susan Hares）、西尔万·赫勒古阿尔克（Sylvain Hellegouarch）、塔潘·迪维卡尔（Tapan Divekar）、辻川达弘（Tatsuhiro Tsujikawa）、林达也（Tatsuya Hayashi）、特德·哈迪（Ted Hardie）、特德·莱蒙（Ted Lemon）、托马斯·布罗耶（Thomas Broyer）、托马斯·福萨蒂（Thomas Fossati）、托马斯·马斯伦（Thomas Maslen）、托马斯·纳多（Thomas Nadeau）、托马斯·诺丁（Thomas Nordin）、托马斯·罗斯勒（Thomas Roessler）、蒂姆·布雷（Tim Bray）、蒂姆·摩根（Tim Morgan）、蒂姆·奥尔森（Tim Olsen）、汤姆·周（Tom Zhou）、特拉维斯·斯诺兹（Travis Snoozy）、泰勒·克洛斯（Tyler Close）、文森特·墨菲（Vincent Murphy）、朱文博（Wenbo Zhu）、维尔纳·鲍曼（Werner Baumann）、威尔伯·斯特里特（Wilbur Streett）、威尔弗雷多·桑切斯·维加（Wilfredo Sanchez Vega）、小威廉·A·罗（William A. Rowe Jr.）、威廉·陈（William Chan）、威利·塔罗（Willy Tarreau）、王小曙（Xiaoshu Wang）、亚龙·戈兰德（Yaron Goland）、英格夫·尼斯埃特·佩特森（Yngve Nysaeter Pettersen）、尤瓦夫·尼尔（Yoav Nir）、约格什·邦（Yogesh Bang）、郑宇春（Yuchung Cheng）、大岩裕（Yutaka Oiwa）、伊夫·拉丰（Yves Lafon，编辑团队的长期成员）、泽德·A·肖（Zed A. Shaw）和钟宇（Zhong Yu）。

有关先前版本的其他致谢内容，请参见[RFC2616]的第16节。

11. 参考文献

11.1. 规范性引用文件

[RFC0793] Postel, J.，“传输控制协议”，STD 7，RFC 793，1981年9月。

[RFC1950] Deutsch, L. 和 J-L. Gailly，《ZLIB压缩数据格式规范3.3版》，RFC 1950，1996年5月。

[RFC1951] Deutsch, P.，“DEFLATE压缩数据格式规范1.3版”，RFC 1951，1996年5月。

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., 和 G. Randers-Pehrson，“GZIP文件格式规范版本4.3”，RFC 1952，1996年5月。

[RFC2119] Bradner, S.，“用于在RFC中指示需求级别的关键词”，BCP 14，RFC 2119，1997年3月。

[RFC3986] Berners-Lee, T., Fielding, R., 和 L. Masinter,“统一资源标识符（URI）：通用语法”，STD 66，RFC 3986，2005年1月。

[RFC5234] Crocker, D.（编辑）和P. Overell，“用于语法规范的增强型巴科斯范式：扩展巴科斯范式（ABNF）”，STD 68，RFC 5234，2008年1月。

[RFC7231] Fielding, R.（编辑）和J. Reschke（编辑），《超文本传输协议（HTTP/1.1）：语义和内容》，RFC 7231，2014年6月。

[RFC7232] Fielding, R., Ed. 和 J. Reschke, Ed.,“超文本传输协议（HTTP/1.1）：条件请求”，RFC 7232，2014年6月。

[RFC7233] Fielding, R.（编辑）、Lafon, Y.（编辑）和J. Reschke（编辑），《超文本传输协议（HTTP/1.1）：范围请求》，RFC 7233，2014年6月。

[RFC7234] Fielding, R.（编辑），Nottingham, M.（编辑），以及J. Reschke（编辑），《超文本传输协议（HTTP/1.1）：缓存》，RFC 7234，2014年6月。

[RFC7235] Fielding, R.（编辑）和J. Reschke（编辑），《超文本传输协议（HTTP/1.1）：认证》，RFC 7235，2014年6月。

[USASCII] 美国国家标准协会，《编码字符集——7位美国信息交换标准代码》，ANSI X3.4，1986年。

[Welch] Welch, T.，“一种高性能数据压缩技术”，《IEEE计算机》17(6)，1984年6月。

11.2. 资料性引用

[BCP115] Hansen, T., Hardie, T., 和 L. Masinter,“新统一资源标识符（URI）方案的指南和注册程序”，BCP 115, RFC 4395, 2006年2月。

[BCP13] Freed, N., Klensin, J., 和 T. Hansen,“媒体类型规范与注册流程”，BCP 13，RFC 6838，2013年1月。

[BCP90] Klyne, G., Nottingham, M., 和 J. Mogul,“消息头字段的注册程序”，BCP 90,，
RFC 3864，2004年9月。

[Georgiev] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., 和 V. Shmatikov,“世界上最危险的代码：在非浏览器软件中验证SSL证书”，发表于2012年ACM计算机与通信安全会议（CCS '12）论文集，第38-49页，2012年10月，<http://doi.acm.org/10.1145/2382196.2382204>。

[ISO-8859-1] 国际标准化组织，“信息技术——8位单字节编码图形字符集——第1部分：拉丁字母No. 1”，ISO/IEC 8859-1:1998，1998年。

[Klein] Klein, A.，“分而治之——HTTP响应拆分、Web缓存投毒攻击及相关主题”，2004年3月，
<http://packetstormsecurity.com/papers/general/whitepaper_httpresponse.pdf>。

[Kri2001] 克里斯托尔（Kristol）D.，《HTTP Cookies：标准、隐私与政治》，
美国计算机协会（ACM）互联网汇刊
《技术1(2)》，2001年11月，
<http://arxiv.org/abs/cs.SE/0105018>。

[Linhart] Linhart, C., Klein, A., Heled, R., 和 S. Orrin，《HTTP请求欺骗》，2005年6月，
<http://www.watchfire.com/news/whitepapers.aspx>。

[RFC1919] Chatel, M.，“经典IP代理与透明IP代理”，RFC 1919，1996年3月。

[RFC1945] Berners-Lee, T., Fielding, R., 和 H. Nielsen, “超文本传输协议——HTTP/1.0”，RFC 1945，1996年5月。

[RFC2045] Freed, N. 和 N. Borenstein，《多用途互联网邮件扩展（MIME）第一部分：互联网邮件正文格式》，RFC 2045，1996年11月。

[RFC2047] Moore, K.，“MIME（多用途互联网邮件扩展）第三部分：非ASCII文本的消息头扩展”，RFC 2047，1996年11月。

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., 和 T. Berners-Lee，《超文本传输协议 --HTTP/1.1》，RFC 2068，1997年1月。

[RFC2145] Mogul, J., Fielding, R., Gettys, J., 和 H. Nielsen, “HTTP版本号的使用与解释”，RFC 2145，1997年5月。

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., 和 T. Berners-Lee, “超文本传输协议 - HTTP/1.1”，RFC 2616，1999年6月。

[RFC2817] Khare, R. 和 S. Lawrence，《在HTTP/1.1中升级到TLS》，RFC 2817，2000年5月。

[RFC2818] Rescorla, E.，“基于TLS的HTTP”，RFC 2818，2000年5月。

[RFC3040] Cooper, I., Melve, I., 和 G. Tomlinson, “互联网Web复制与缓存分类”，RFC 3040，
2001年1月。

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., 和 S. Rose，《DNS安全介绍与要求》，
RFC 4033，2005年3月。

[RFC4559] Jaganathan, K., Zhu, L., 和 J. Brezak, “基于SPNEGO的Kerberos和NTLM HTTP认证在Microsoft Windows中的应用”，RFC 4559，2006年6月。

[RFC5226] Narten, T. 和 H. Alvestrand，“在RFC中编写IANA考虑事项部分的指南”，BCP 26，
RFC 5226，2008年5月。

[RFC5246] Dierks, T. 和 E. Rescorla，《传输层安全（TLS）协议版本1.2》，RFC 5246，2008年8月。

[RFC5322] Resnick, P.，“互联网消息格式”，RFC 5322，2008年10月。

[RFC6265] Barth, A.，“HTTP状态管理机制”，RFC 6265，2011年4月。

[RFC6585] Nottingham, M. 和 R. Fielding，《附加HTTP状态码》，RFC 6585，2012年4月。

附录A. HTTP版本历史

HTTP自1990年起投入使用。 其首个版本（后称为HTTP/0.9）是一个简单的协议，用于通过互联网传输超文本数据，仅使用一种请求方法（GET）且不包含元数据。 根据[RFC1945]的定义，HTTP/1.0增加了一系列请求方法和类似MIME的消息传递方式，允许传输元数据并在请求/响应语义上添加修饰符。然而，HTTP/1.0并未充分考虑分层代理、缓存的影响，也未充分考虑对持久连接的需求或基于名称的虚拟主机。 由于大量未完全实现的应用程序自称“HTTP/1.0”，因此有必要更改协议版本，以便两个通信的应用程序能够确定彼此的真实能力。

HTTP/1.1通过纳入更严格的要求来确保可靠实现，从而保持与HTTP/1.0的兼容性。它只添加了那些HTTP/1.0接收方可以安全忽略的特性，或者只有在与声明符合HTTP/1.1协议的一方通信时才会发送的特性。

HTTP/1.1的设计旨在使支持旧版本变得容易。通用的HTTP/1.1服务器应能理解HTTP/1.0格式的任何有效请求，并使用HTTP/1.0客户端能理解（或可安全忽略）的特性，以HTTP/1.1消息做出适当响应。 同样，HTTP/1.1客户端应能理解任何有效的HTTP/1.0响应。

由于HTTP/0.9不支持在请求中包含头部字段，因此它没有机制来支持基于名称的虚拟主机（通过检查Host头部字段来选择资源）。 任何实现基于名称的虚拟主机的服务器都应该禁用对HTTP/0.9的支持。 大多数看似HTTP/0.9的请求，实际上是客户端未能正确编码请求目标而导致的构造错误的HTTP/1.x请求。

A.1. 与HTTP/1.0相比的变化

本节总结了HTTP/1.0和HTTP/1.1版本之间的主要差异。

A.1.1. 多宿主Web服务器

客户端和服务器支持Host头字段（第5.4节）、在HTTP/1.1请求中缺少该字段时报告错误，以及接受绝对URI（第5.3节）的要求，是HTTP/1.1定义的最重要变更之一。

较旧的HTTP/1.0客户端假定IP地址和服务器之间存在一对一的关系；除了请求所指向的IP地址外，没有其他已建立的机制来区分请求所指向的预期服务器。 在开发HTTP/1.1的过程中引入了Host头字段，尽管大多数HTTP/1.0浏览器很快实现了该字段，但为了确保完全采用，对所有HTTP/1.1请求提出了额外要求。 在撰写本文时，大多数基于HTTP的服务都依赖于Host头字段来定位请求。

A.1.2. 保持活动连接

在HTTP/1.0协议中，每个连接由客户端在发送请求之前建立，并在服务器发送响应后由服务器关闭。
然而，一些实现采用了第节中描述的显式协商（“Keep-Alive”）版本的持久连接[RFC2068]的第19.7.1节。

一些客户端和服务器可能希望通过显式地使用“Connection: keep-alive”请求头字段来协商，以与这些先前的持久连接方法兼容。 然而，HTTP/1.0持久连接的一些实验性实现存在缺陷；例如，如果HTTP/1.0代理服务器不理解Connection，它会错误地将该头字段转发给下一个入站服务器，从而导致连接挂起。

一种尝试的解决方案是引入专门针对代理的“代理连接”头字段。 然而，在实践中，这一方案也未能奏效，因为代理通常部署在多个层级中，从而引发了上述同样的问题。

因此，建议客户端在任何请求中都不要发送Proxy-Connection头字段。

客户端也被鼓励在请求中谨慎考虑使用Connection: keep-alive；虽然它们可以与HTTP/1.0服务器建立持久连接，但使用它们的客户端将需要
监控连接中是否存在“挂起”请求（这表明客户端应停止发送标头字段），并且当使用代理时，客户端根本不应使用此机制。（注：“hung”请求：在HTTP协议中，“hung”请求指的是客户端在发送请求后，由于某种原因（如服务器处理请求时发生故障）而长时间等待响应，导致请求被挂起。在这种情况下，客户端应停止发送标头字段，以避免继续浪费网络资源。）。

A.1.3. 传输编码介绍

HTTP/1.1引入了Transfer-Encoding头字段（第3.3.1节）。 在通过符合MIME的协议转发HTTP消息之前，需要对传输编码进行解码。

A.2. 与RFC 2616的差异

HTTP的错误处理方式已经解释过了 （第2.5节）

HTTP版本的ABNF（抽象语法标记法）生成规则已明确为区分大小写。 此外，由于已知实现无法正确处理多位数字的版本号，因此版本号已被限制为一位数字。 （第2.6节）

由于在线传输安全问题，HTTP和HTTPS统一资源标识符（URI）中现在不允许使用用户信息（即用户名和密码）。 （第2.7.1节）

HTTPS统一资源标识符（URI）方案现由本规范定义；此前，该方案在[RFC2818]的第2.4节中定义。 此外，它还隐含了端到端的安全性。 （第2.7.2节）

HTTP消息可以（并且经常）被实现所缓冲；尽管它有时可以作为流来使用，但HTTP从根本上来说是一种面向消息的协议。 为了提高互操作性，已经为各种协议元素提出了最小支持大小。 （第3节）

现在，必须拒绝字段名前后存在无效空格的情况，因为接受这种情况会带来安全漏洞。 定义头部字段的ABNF（抽象语法标记法）生成式现在仅列出字段值。（第3.2节）

关于某些语法产生式之间隐含的线性空白规则已被删除；现在，只有在ABNF（抽象巴科斯范式）中明确规定的地方才允许使用空白。 （第3.2.3节）

跨越多行的头部字段（“行折叠”）已被弃用。 （第3.2.4节）

在注释和引号字符串文本中，不再允许使用NUL（空）字节，并且已明确其中反斜杠转义的处理方式。 引号对规则不再允许转义除HTAB（水平制表符）以外的控制字符。 标头字段和原因短语中的非US-ASCII内容已被废弃并设为不透明（TEXT规则已移除）。 （第3.2.6节）

现在，接收方必须将伪造的Content-Length头字段视为错误进行处理。 （第3.3.2节）

用于确定消息体长度的算法已得到明确，以指出所有影响它的特殊情况（例如，由方法或状态码驱动），并指出了新的协议

元素无法定义此类特殊情况。CONNECT是确定消息体长度时的一个新的特殊情况。“multipart/byteranges”不再是确定消息体长度的一种检测方式。（第3.3.3节）

“身份”传输编码标记已被移除。（第3.3节和第4节）

块长度不包括块头和块尾中八位字节的数量。 不允许在块扩展中进行换行。（第4.1节）

“deflate”内容编码的含义已得到明确。（第4.2.2节）

RFC 3986中的段+查询组件已被用于定义请求目标，而不是使用RFC 1808中的abs_path。 这请求目标的星号形式仅允许用于OPTIONS方法。 （第5.3节）

引入了“有效请求URI”这一术语。 （第5.5节）

网关不再需要生成Via头字段。（第5.7.1节）

已明确何时必须发送“关闭”连接选项。 此外，连接头字段中必须显示“逐跳”头字段；仅因为它们在本规范中被定义为逐跳，并不意味着可以免除其显示要求。 （第6.1节）

每台服务器两个连接的限制已被取消。 不再要求对幂等请求序列进行重试。当服务器提前关闭连接时，在某些情况下重试请求的要求已被取消。 此外，关于服务器何时可以提前关闭连接的一些额外要求也已被取消。 （第6.3节）

Upgrade标头字段的语义现在已在除101以外的响应中定义（此内容摘自[RFC2817]）。 此外，字段值中的排序现在具有重要意义。 （第6.7节）

列表生成中的空列表元素（例如，包含“,,”的列表头字段）已被弃用。 （第7节）

传输编码的注册现在需要经过IETF审核（第8.4节）

本规范现在定义了升级令牌注册表，该注册表之前已在[RFC2817]的第7.2节中定义。（第8.6节）

已取消支持HTTP/0.9请求的预期。（附录A）

指出了请求中Keep-Alive和Proxy-Connection头字段存在的问题，并建议完全避免使用后者。（附录A.1.2）

附录B. 收集的ABNF BWS

= OWS

连接 = *("," OWS) 连接选项 *(OWS "," [OWS 连接选项])

Content-Length = 1位数字

HTTP消息 = 开始行 *(头字段 CRLF) CRLF [消息体]
HTTP名称 = %x48.54.54.50；HTTP
HTTP版本 = HTTP名称 "/" 数字 "." 数字 主机 = URI主机 [":" 端口]

OWS = *(空格符 / 水平制表符)

RWS = 1*(空格符 / 水平制表符)

TE = [("," / t-codings) *(OWS "," [OWS t-codings])] Trailer = *("," OWS) 字段名 *(OWS "," [OWS 字段名])
传输编码 = *("," OWS) 传输编码 *(OWS "," [OWS 传输编码])

URI引用 = <URI引用，参见[RFC3986]，第4.1节> 升级 = *("," OWS) 协议 *(OWS "," [OWS协议])

Via = *("," OWS) (接收协议 RWS 接收者 [RWS 注释]) *(OWS "," [OWS (接收协议 RWS 接收者 [RWS 注释])])

绝对URI（absolute-URI）= <绝对URI，参见[RFC3986]，第4.3节> 绝对形式（absolute-form）= 绝对URI
绝对路径 = 1*("/" 片段) 星号形式 = "*"
authority = <权威，参见[RFC3986]，第3.2节> authority-form = 权威

chunk = chunk-size [chunk-ext] CRLF chunk-data CRLF chunk-data =
1*OCTET
chunk-ext = *(";" chunk-ext-name ["=" chunk-ext-val]) chunk-ext-name = token这是一个HTTP标准协议中的字段定义。其中，chunk-ext表示一个字段，其值为一个列表，列表中的每个元素都是一个由分号（;）分隔的字段名和字段值对。字段名（chunk-ext-name）是一个标记，可以包含一个等号（=）分隔的字段名和字段值对
chunk-ext-val = 标记符 / 引号字符串 chunk-size =
1*HEXDIG
分块体 = *块 最后一个块 尾部部分 CRLF 注释 = "(" *(文本 / 双引号对 /
注释) ")" 连接选项 = 标记
ctext = HTAB / SP / %x21-27 ; ’!’-’’’
/ %x2A-5B ; ’*’-’[’
/ %x5D-7E ; ’]’-’~’
/ obs-text

字段内容 = 字段变量字符 [1*(空格 / 制表符) 字段变量字符] 字段名称 = 标记
字段值 = *(字段内容 / 观察折叠) 字段值字符 = VCHAR /
观察文本
片段 = <片段，见[RFC3986]第3.5节>

头部字段 = 字段名 ":" OWS 字段值 OWS
http-URI = "http://" 权威路径-绝对路径 ["?" 查询参数] ["#" 片段]
https-URI = "https://" 权威路径-abempty ["?" 查询参数] ["#" 片段]

last-chunk = 1*"0" [chunk-ext]（注：此句为HTTP标准协议中的部分内容，具体含义需根据上下文或相关文档进行解释。在此，我仅对部分生造词进行了注释，以帮助理解。）回车换行

消息体 = *八位字节 方法 = 令
牌

obs-fold = CRLF 1*(SP / HTAB) obs-text
= %x80-FF
origin-form = 绝对路径 ["?" 查询字符串]

部分URI = 相对部分 ["?" 查询]
path-abempty = <path-abempty，参见[RFC3986]第3.3节> port = <port，参见[RFC3986]第3.2.3节>
协议 = 协议名称 ["/" 协议版本] 协议名称 = 标记
协议版本 = 标记 假名 = 标记

qdtext = HTAB / SP / "!" / %x23-5B ; ’#’-’[’
/ %x5D-7E ; ’]’-’~’
/ obs-text
query = <查询，参见[RFC3986]，第3.4节>
quoted-pair = "\" (水平制表符 / 空格符 / 可见字符 / 保留文本)

quoted-string = 双引号 *(引号文本 / 引号对) 双引号

等级 = ("0" ["." *3位数字]) / ("1" ["." *3"0"])
原因短语 = *(HTAB / SP / VCHAR / obs-text) 接收方 = (uri-host [":"
端口]) / 假名
received-protocol = [协议名称 "/"] 协议版本 相对部分 = <相对部分，参见[RFC3986]，
第4.2节> request-line = 方法 SP 请求目标 SP HTTP版本 CRLF 请求目标 = 源形式 / 绝对
形式 / 权威形式 /
星号形式

scheme = <方案，参见[RFC3986]，第3.1节> segment = <段，参见
[RFC3986]，第3.3节> start-line = 请求行 / 状态行
状态码 = 3位数字
状态行 = HTTP版本 SP 状态码 SP 原因短语 CRLF

t-codings = "trailers" / (transfer-coding [t-ranking]) t-ranking = OWS ";" OWS "q="
rank
tchar = "!" / "#" / "$" / "%" / "&" / "’ " / "*" / "+" / "-" / "." / "^" / "_" / " ‘" / "|" / "~" / 数字 / 字母
token = 1*tchar
trailer-part = *(header-field CRLF)
传输编码 = "分块" / "压缩" / "deflate" / "gzip" / 传输扩展
transfer-extension = token *(OWS ";" OWS transfer-parameter) transfer-parameter = token
BWS "=" BWS (token / quoted-string)

uri-host = <主机，参见[RFC3986]，第3.2.2节>

索引

作者地址

Roy T. Fielding（编辑）奥多比系统
公司
公园大道345号
美国加利福尼亚州圣何塞
，邮编
95110

电子邮件：fielding@gbiv.com 统一资
源标识符（URI）：
http://roy.gbiv.com/

Julian F. Reschke（编辑） greenbytes
GmbH
Hafenweg 16（港口路16号）
德国明斯特，NW 48155

电子邮件：julian.reschke@greenbytes.de
URI： http://greenbytes.de/tech/webdav/