网络工作组
请求注释：2616
废弃：2068
类别：标准路线

R. Fielding
加州大学欧文分校
J. Gettys
康柏/W3C
J. C. Mogul
康柏
H. Frystyk
W3C/麻省理工
L. Masinter
施乐
P. Leach
微软
T. Berners-Lee
W3C/麻省理工

1999 年 6 月

超文本传输协议——HTTP/1.1

该备忘录的地位

本文规定因特网社区的因特网标准路线协议，以及讨论的请求和改善的建议。关于该协议的标准化状态及地位请参考当前版的“因特网官方协议标准”（STD 1）。发布该备忘录没有限制。

版权声明

摘要

超文本传输协议（HTTP）是分布式、协作的、超媒体信息系统的应用层协议。它是通用的，无状态的协议，可以用在超文本用途之外的许多任务，如名称服务器和分布式目标管理系统，通过扩展它的请求方法，错误码和头部 [47]。HTTP 的一个特性是数据表示的引入和协商，允许系统建立独立的传输数据。

HTTP 已经首先于 1990 年被 WWW 全球信息使用。该规范定义的协议用“HTTP/1.1”表示，是对 RFC 2068 [33]的更新。

目录

1 序言

1.1 目的

超文本传输协议（HTTP）是分布式、协作的、超媒体信息系统的应用层协议。HTTP 已经首先于 1990 年被 WWW 全球信息使用。HTTP 的第一个版本，引用为 HTTP/0.9，是跨因特网的传输原始数据的简单协议。HTTP/1.0，由 RFC 1945 [6]定义，改善的协议允许消息按 MIME 类似消息的格式，包含所传输数的元信息及对请求/响应的语义上的修改者。然而，HTTP/1.0 没有充分考虑分层代理、缓存、永久连接需求、或虚拟主机的影响。进而，自称为“HTTP/1.0”的非完整实现应用程序的增殖已经需要协议的改版，为了两个通迅应用程序相互判断其真实的能力。

本规范定义的协议作为“HTTP/1.1”引用。该协议包括比 HTTP/1.0 更加来历的要求，为了保证可靠实现其特性。

实践信息系统需要比简单获取，包括查找、前端更新、和注解更多的功能性。HTTP 允许开方的方法集和头部来表示请求的目的 [47]。它基于由统一资源标识符（URI）[3]中提供的参考规则，作为定位（URL）[4]或命名（URN）[20]，为表示方法应用的资源。消息按照与多目的因特网邮件扩展（MIME）[7]中定义的因特网邮件相似的格式传输。

HTTP 还用作用户代理和代理/网关到其它因特网系统，包括由 SMTP [16]、NNTP [13]、FTP [18]、Gopher [2]、和 WAIS [10]支持的系统，间通迅的通用协议。通过该方式，HTTP 允许从不同的应用访问基本超媒体资源。

1.2 要求

在本文中的关键字【必须】、【禁止】、【要求】、【将要】、【不要】、【应该】、【不该】、【推荐】、【可选】按 RFC 2119 [34]的描述来解释。

实现不是一致的，如果它的实现不能满足一个或多个【必须】或【要求】级的协议要求。满足所有【必须】或【要求】级和所有【应该】级协议要求的实现被称为“无条件一致”；满足所有【必须】级协议要求，但不满足全部【应该】协议要求的实现被称为“有条件一致”。

1.3 术语

本规范使用大量的术语来表示参与到 HTTP 通迅中拌演角色和与 HTTP 通迅的对象。

连接　　建立在要通迅的两个程序间的传输层虚电路

消息　　HTTP 通迅的基本单元，由匹配 4 节中所定义语法的结构化八位组序列组成，并通过连接传输。

请求　　HTTP 请求消息，如 5 节中所定义。

响应　　HTTP 响应消息，如 6 节中所定义。

资源　　网络数据对象或服务，可通过 URI 标识，如 3.2 节中所定义。资源可以有多种表示（如，多种语言、数据格式、大小、和精度）或以多种其它方式。

实体　　作为请求或响应负载传输的信息。实体由以实体头部域形式的元信息及实体主体形式的内容组成，如 7 节中描述。

表述　　包含在与内容协商有关的响应中的实体，如 12 节所述。可能存在与特定响应状态相关的多种表述。

内容协商　　当服务请求时，选择适当表述的机制，如 12 节所述。任何响应中的实体表述都可能协商（包括错误响应）。

变量　　在任何给定时刻与某个资源相关的一个或多个表述。每个这种表述称为“变量”。使用术语“变量”并不意味着该资源与内容协商相关。

客户端　　建立连接以发送请求的程序。

用户代理　　发起请求的客户端。经常是浏览器、编辑器、蜘蛛（网游机器）、或其它终端用户工具。

服务器　　接受连接，以便通过回送响应来服务请求的应用程序。任何给定的程序都能够成为客户端和服务器，我们使用该术语只是引用特定连接的程序执行的角色，而不是通常意义上的程序能力。同样，任何服务器可以作为原始服务器、代理、网关、或隧道，基于每个请求的属性来交换行为。

原始服务器　　提供存储或将创建的资源的服务器。

代理　　一种中间程序，既作为服务器，又作为客户端，目的是按其它客户端的行为作出请求。请求在内部服务或传递它们，进行可能的翻译，到其它服务器。代理【必须】实现所有本规范的客户端和服务器要求。“透明代理”是这种代理，除其所需的代理认证和标识以外，它不修改请求或响应。“非透明代理”是另一种代理，即它要修改请求或响应，为了给用户代理提供一些服务，如组注解服务、媒体类型转换、协议变形、或匿名过滤器。除了对透明或非透明行为明确规定外，HTTP 代理需求要应用

到这两种代理上。

网关　一种作为一些其它服务器的媒介服务器。与代理不同，代理接受对所请求资源的请求，如同自己是原始服务器一样；请求客户端可能并不清楚它正与网关通迅。

隧道　一种媒介程序，作为两个连接间的盲目接力者。一旦激活，隧道就不被认为是 HTTP 通迅的一部份，虽然隧道可能已经通过 HTTP 请求被初始化了。隧道将停止工作，当两端接力的连接都关闭时。

缓存　一种在其本地存储响应消息程序和控制它的消息存储、获取及删除行为的子系统。缓存存储可缓存响应，为了减少响应时间和将来相同请求量对网络带宽消费量。任何客户端或服务器都可以包括缓存，尽管缓存不能被作为隧道的服务器使用。

可缓存　如果缓存允许存储响应消息的一份拷贝，以便用来回复后序的请求，则该响应是可缓存的。判断 HTTP 响应缓存性的规则在 13 节中定义。即使如果资源是可缓存的，关于缓存能够为特殊请求使用缓存的拷贝也存在额外的约束。

第一手　响应是第一手的，当它直接从原始服务器来，没有不必要的延迟，可能通过一个或多个代理。响应还也是第一手的，当它已经直接被原始服务所证实。

清楚截止时间　原始服务器想要缓存没有经过进一步证实下不应该返回其实体的时间。

启发截止时间　缓存指定的截止时间，当没有清楚截止时间可用时。

年龄　响应的年龄是原始服务器发送它以来，或它成功被原始服务器所证实的时间。

更新周期　在生成响应和它截止时间之间的时间长度。

更新的　响应是更新的，如果它的年龄没有超过其更新周期。

过期的　响应是过期的，如果它的年龄已经超过其更新周期。

语义透明度　缓存以与特殊响应有关的“语义透明度”方式的行为，当它影响到请求客户端和原始服务器时，除了可改善性能。当缓存是语义透明的时，缓存收到几乎相同的响应（除了跳对跳头部），即它将收到其请求已经直接被原始服务器处理过的响应。

证言　协议元素（如实体标签或最终修改时间），用于发现一个缓存条目是否是实体的等价拷贝。

上游/下游　　上游和下游描述消息流：所有消息从上游流向下游。

入界/出界　　入界和出界代表请求和响应消息的路径：“入界”意思是“移向原始服务器”，“出界”意思是“移向用户代理”。

1.4 全局描述

HTTP 协议是请求/响应协议。客户端发送请求到服务器，以请求方法、URI 和协议版本，接着是 MIME 类的消息包含请求修饰成份、客户端信息、和可能的主体内容，通过与服务器的连接的形式。服务器以状态行响应，包括消息协议版本和成功或错误代码，接着是 MIME 类的消息包含服务器信息、实体元信息、和可能的实体主体内容。HTTP 与 MIME 间的关系在附录 19.4 中描述。

大多 HTTP 通迅由用户代理发起，并由应用到某些原始服务器上的某个资源的请求组成。在最简单的情况下，这可以通过在用户代理（UA）和原始服务器（O）之间的单个连接（v）完成。

```
请求链---------------------------------->
UA-------------------v-------------------O
<----------------------------------响应链
```

更复杂的情况发生在一个或多个媒介存在于请求/响应链中时。存在三种通用形式的媒介：代理、网关和隧道。代理是转发代理，接收含有绝对形式 URI 的请求，重定消息的所有或部分，并转发该重新格式化的请求到由 URI 标识的服务器。网关是接收代理，作为其它某些服务器的上层，且必要时翻译请求到下面的服务器协议。隧道作为接力点，在两个连接间，不改变消息；隧道用于当通迅需要穿过媒介（如防火墙）时，甚至在媒介不能理解消息内容时。

```
请求链------------------------------------------------------>
UA-----v----- A -----v----- B -----v----- C -----v-----O
<------------------------------------------------------响应链
```

上图显示用户代理与原始服务器音的三个媒介（A、B 和 C）。请求或响应消息移过整个链路将穿过 4 个独立的连接。这种区分是重要的，因为一些 HTTP 通迅选项只可应用到最近的、非隧道邻居的连接，只应用到链路的终端，或链路上的所有连接。尽管该图是线性的，每个参与者都可以使用在多重的、同时的通迅中。例如，B 可以收到许多不是 A 的客户端的请求，且/或转发请求到不是 C 的服务器，同时它又处理 A 的请求。

通迅中没有作为隧道的任何部分都可以使用内部缓存来处理请求。缓存影响是缩短请求/响应链，如果链路参与者中的一个已经缓存了可用来该请求上的响应。下面描述结果链，即当 B 有先前从 O（通过 C）来的该请求的响应，且该响应没有被 UA 或 A 缓存过，的一份缓存拷贝时。

请示链---------------------->
UA -----v----- A -----v----- B - - - - - - C - - - - - - O
<----------------------响应链

不是所有响应都可有效缓存，且一些请求可以包含修饰符，其中有对缓存行为的特别要求。HTTP 对缓存行为的要求和可缓存响应在 13 节中定义。

实际上，存在大量缓存和代理的各种结构和配置，当前正通过 WWW 试验或布署。这些系统包括代理缓存的自然层次，以节约越洋带宽、广播或多播缓存实体的系统、发布在 CD-ROM 上缓存数据的子集的组织、等等。HTTP 系统用于企业内连网，通过高带宽连接，和通过 PDA 访问低功率无线连接和间断连接性。HTTP/1.1 的目标是支持已经布署的广范多样化的配置，引入协议构造以满足建立网络应用者的需求，他们都要高可靠，如不可能，至少能够可靠地指示失败。

HTTP 通迅通常使用 TCP/IP 连接。缺省端口号是 TCP 80 [19]，但能够使用其它端口。不排除 HTTP 实现在其它因特网或其它网络协议的上层。HTTP 只假定可靠传输；任何提供如些保证的协议都可使用；承载协议有关的 HTTP/1.1 请求和响应的传输数据单元的结构图超出本规范的范围。

在 HTTP/1.0 中，大多实现为每个请求/响应交换使用新的连接。在 HTTP/1.1 中，一个连接可用于一次或多次请求/响应交换，尽管连接可能由于各种原因被关闭（见 8.1 节）。

2 符号约定和一般语法

2.1 增广 BNF

本文中指定的所有机制都以散文和增广 BNF 相似的用于 RFC 822 [9]中的形式描述。实现者需要熟悉该符号，以便于理解该规范。增广 BNF 包括如下构成：

名称=定义	规则的名称是简单的名称自己（没有用任何“<”和“>”封装），且通过等号“=”与其定义分开。空白符只在继行缩进用于指出规则定义跨越超过一行时有意义。确定的基本规则用大写，如同 SP、LWS、HT、CRLF、DIGIT、ALPHA 等。尖括号用在定义内，使它们的出现将容易判别出是使用的规则名。
“文字”	包围文字文本的饮用记号。除非特别指出，文本是大小写不敏感的。
规则 1|规则 2	由条状符（“|”）分隔的元素是可选的，如，“是|否”将接受是或者否。
（规则 1 规则 2）	封装在括号内的元素作为单个元素。因此，“（elem（foo|bar）elem）”允许的符号序列有“elem foo elem”和“elem bar elem”。

*规则	字符“*”先于元素指出重复。完整形式是“<n>*<m>元素”，指出元素出现至少<n>次和最多<m>次。缺省值为 0 到无限，所以“*(元素)”允许任何次，包括 0 次；“1*元素”要求至少 1 次，且“1*2 元素”允许 1 次或 2 次。
[规则]	方括号封装可选元素；“[foo bar]”与“*1(foo bar)”相同。
N 规则	规定重复：“<n>（元素）”与“<n>*<n>（元素）”相同；即，元素确定出现<n>次。因此，2DIGIT 是 2 位数，且 3ALPHA 是有 3 个字母字符的字符串。
#规则	定义“#”构造，与“*”相似，用于定义元素清单。完整形式是“<n>#<m>元素”，指出最少<n>且最多<m>次元素，它们之间由一个或多个逗号（“,”）和【可选】的线性空白符（LWS）分隔。这使得构造清单的通用形式非常容易，如此（*LWS 元素 *（*LWS “,” *LWS 元素））规则可被指示为 1#元素。当使用该构造时，允许空元素，但不计算元素计数。即，允许“（元素）,,（元素）”，但计数只是 2 个元素。因此，在至少需要 1 个元素时，【必须】至少出现 1 个非空元素。缺省值是 0 到无限，所有“#元素”允许任何次数，包括 0 次；“1#元素”要求至少 1 次；和“1#2 元素”允许 1 次或 2 次。
；注释	分号，放置在规则文本右边一些距离处，引出注释，直到行结束。这是包括与规范并存的有用注释的简单方式。
隐式*LWS	该规范描述的语法是基于单词的。除非特别说明，线性空白符（LWS）可以包含在任何相邻的单词（符号或引用字符串）间，和相邻单词与分隔符间，而不会改变对域的解释。任何两个符号（“符号”定义见下面）间【必须】至少存在 1 个定界符（LWS 和/或分隔符），因为否则它们可能被解释为单个符号。

2.2 基本规则

如下规则用于通篇规范中描述基本分解构造。US-ASCII 编码符号集定义在 ANSI X3.4-1986 [21]中。

```
OCTET = <任何 8 位数据序列>
CHAR = <任何 US-ASCII 字符（8 位组的 0 - 127）>
UPALPHA = <任何 US-ASCII 大写字母“A”..“Z”>
LOALPHA = <任何 US-ASCII 小写字母“a”..“z”>
ALPHA = UPALPHA | LOALPHA
```

DIGIT = <任何 US-ASCII 数字“0”..“9”>
CTL = <任何 US-ASCII 控制字符（8 位组的 0 - 31）和 DEL（127）>
CR = <US-ASCII CR，回车（13）>
LF = <US-ASCII LF，换行（10）>
SP = <US-ASCII SP，空格（32）>
HT = <US-ASCII HT，水平制表符（9）>
<"> = <US-ASCII 双引号符（34）>

HTTP/1.1 定义 CR LF 序列作为除实体主体（容错应用程序见附录 19.3）外的所有协议元素的行结束标志。实体主体中的行结束符由与其相关的媒体类型定义，如 3.7 节中所述。

CRLF = CR LF

HTTP/1.1 头部域值可以拆分为多行，如果续行由空格或水平制表符引导。所有线性空白符，包括拆分的，都有与 SP 相同的语义。接收者【可以】在解释域值或转发消息到下游前替换任何线性空白符为单个 SP。

LWS = [CRLF] 1*(SP | HT)

TEXT 规则只用在描述域内容和值，它们不会被消息解析器解释。*TEXT 词【可以】包含从其它非 ISO-8859-1 [22]中的字符集中的字符，只有当按照 RFC 2047 [14]中的规则来编码时。

TEXT = <任何 OCTET 除去 CTL，但包括 LWS>

TEXT 定义中允许 CRLF 只作为头部域继续的一部分。希望拆分的 LWS 在解释 TEXT 值前被替换为单个 SP。

十六进制数字字符用在几个协议元素中。

HEX = “A” | “B” | “C” | “D” | “E” | “F” | “a” | “b” | “c” | “d” | “e” | “f” | DIGIT

许多 HTTP/1.1 头部域值由 LWS 或特殊字符分隔的单司组成。这此特殊字符【必须】在用于参数值的引用字符串中（如 3.6 节中所述）。

符号 = 1*<任何 CHAR 除去 CTL 或 分隔符>
分隔符 = “(” | “)” | “<” | “>” | “@” | “,” | “;” | “:” | “\”
|<">| “/” | “[” | “]” | “?” | “=” | “{” | “}” | SP | HT

注释可以被包括在一些 HTTP 头部域中，通过使用圆括号包围注释文本。注释只允许在域中的域值定义部分包含“注释”的域中出现。在所有其它域中，圆括号会作为域值的一部分。

注释 = “(” *（注释文本 | 引用对 | 注释）“)”
注释文本 =<任何 TEXT 除去“(”和“)”>

文本字符串解析为单个单词，如果它使用双引号引用。

引用字符串 = （<"> *（引用文本 | 引用对）<"> ）
引用文本 =<任何 TEXT 除去<">>

后斜线符（“\”）【可以】只在引用字符串或注释构造中用作单字符引用机制。

引用对 = “\” CHAR

3 协议参数

3.1 HTTP 版本

HTTP 使用“<主要>.<次要>”编号方案来批示协议版本。协议版本策略用来允许发送者指示消息的格式和它理解将来 HTTP 通迅的能力，比通过通迅来获取特性更正确。增加消息组件面不影响通迅行为或只增加可扩展的域值，将来需更改版本号。<次要>号增加，当协议增加特性的改变不会改变一般消息解析算法，但可以增加消息的语义和暗示发送者的额外能力时。<主要>号增加，当协议中的消息格式修改时。更完整解释见 RFC 2145 [36]。

HTTP 消息的版本通过消息中第一行的 HTTP 版本域来指示。

HTTP 版本 = “HTTP”“/”1*DIGIT “.”1*DIGIT

应注意，主要和次要号【必须】作为独立的整数，且每个【可以】增加到比 1 位数更高。因此，HTTP/2.4 比 HTTP/2.13 版本低，反过来 HTTP/2.13 又比 HTTP/12.3 低。接收者【必须】乎略引导 0，而且【禁止】发送。

应用程序发送请求或响应消息，包括 HTTP 版本是“HTTP/1.1”，则它【必须】至少是有条件与本规范一致。应用程序至少有条件与本规范一致【应该】在其消息中使用“HTTP/1.1”的 HTTP 版本，且任何与 HTTP/1.0 不兼容的消息都【必须】这样做。关于体发发送指定 HTTP 版本的详情见 RFC 2145 [36]。

应用程序的 HTTP 版本比该应用程序至少有条件一致的 HTTP 版本更高。代理和网关应用程序需要小心，当转发消息的协议版本与该应用程序的不同时。由于协议版本指示发送者的协议能力，因此代理/网关【禁止】发送其版本指示比实际版本高的消息。如果由到更高版本的请求，代理/网关【必须】要么降级请求版本，要么响应错误，或者切换到隧道行为。

由于 HTTP/1.0 代理在 RFC 2068 [33]出版前发现存在互操作问题，缓存代理【必须】，网关【可以】，且隧道【禁止】升级请求到超过其支持的版本。对该请求的代理/网关的响应【必须】与请求有相同的主要版本。

注意：在 HTTP 版本间转换可以造成修改由版本引入所需或禁止的头部域。

3.2 统一资源标识符

URI 被表述为许多名称：WWW 地址、统一文档标识符、统一资源标识符 [3]、和统一资源定位符（URL）[4]及名称（URN）[20]间的最终组合。直到被 HTTP 关心，统一资源标识符都是简单的格式化字符串，标识某个资源——通过名称、位置、或其它任何特性。

3.2.1 一般语法

HTTP 中的 URI 能够表述为绝对形式或相对于一些已知的基础 URI [11]，取决于它们使用的内容。这两种形式是不同的，绝对 URI 始终由方案名接着的冒号开始。关于 URL 语法和语义的定义性信息，见“统一资源标识符（URI）：一般语法和语义”,RFC 2396 [42]（替换 RFC 1738 [4]和 RFC 1808 [11]）。本规范适用那个规范中关于“URI-引用”、“绝对 URI”、“相对 URI”、“端口号”、“主机名”、“绝对路径”、“相对路径”、和“权威”的定义。

HTTP 协议没有对 URI 的长度作出任何硬性规定。服务器【必须】能够处理它们服务的任何资源的 URI，且【应该】能够处理长度极大的 URI，如果它们提供基于 GET 的形式可以生成这类 URI。服务器【应该】返回 414（Request-URI Too Long）状态，如果 URI 超过服务器能够处理的长度（见 10.4.15 节）。

注意：服务器应当小心决定 URI 长度超过 255 字节，因为一些老的客户端或代理实现可能不能正确支持这种长度。

3.2.2 超文本传输协议的 URL

“http”方案用来通过 HTTP 协议定位网络资源。本节定义超文本传输协议 URL 方案规定的语法和语义。

超文本传输协议 URL = “http:”“//”主机名[“:”端口号] [绝对路径[“?”查询]]

如果端口号为空或没有给出，则假设端口号是 80。语义是所标识的资源位于服务器在该主机的该端口号上监听的 TCP 连接上，且资源的请求 URI 是绝对路径（5.1.2 节）。【应该】尽量避免在 URL 中使用 IP 地址（见 RFC 1900 [24]）。如果 URL 中不存在绝对路径，则它【必须】作为“/”，在作为资源的请求 URI 时（5.1.2 节）。如果代理收到的主机名不是完整资格域名，则它【可以】增加它的域到其收到的主机名中。如果代理收到完整资源域名，则代理【禁止】修改主机名。

3.2.3 URI 的比较

当比较两个 URI 来判断它们是否匹配时，客户端【应该】对整个 URI 使用大小写敏感的字节对字节的比较，除了下面的例外：

- 空或没有给出的端口号与该 URI 引用的缺省端口号相同；

- 比较主机名【必须】是大小写不敏感的；
- 比较方案名【必须】是大小写不敏感的；
- 空绝对路径与绝对路径“/”相同。

不在“保留”和“不安全”集（见 RFC 2396 [42]）中的字符与其““%”HEX HEX”编码相同。例如，下面三个 URI 是相同的：

http://abc.com:80/~smith/home.html
http://ABC.com/%7Esmith/home.html
http://ABC.com:/%7esmith/home.html

3.3 日期/时间格式

3.3.1 完整日期

HTTP 应用程序因历史原因，允许 3 种不同的日期/时间戳的表述格式：

Sun, 06 Nov 1994 08:49:37 GMT ； RFC 822，被 RFC 1123 更新
Sunday, 06-Nov-94 08:49:37 GMT ； RFC 850，被 RFC 1036 废弃
Sun Nov 6 08:49:37 1994 ； ANSI C 的 asctime()格式

第一种格式被首选的因特网标准，表示 RFC 1123 [8]（对 RFC 822 [9]的更新）中定义的固定长度的子集。第二种格式常用，但基于废弃的 RFC 850 [12]日期格式，且缺少 4 位数的年份。解析日期值的 HTTP/1.1 客户端和服务器【必须】接受全部 3 种格式（以与 HTTP/1.0 兼容），尽管它们【必须】在头部域中只生成 RFC 1123 格式表示的 HTTP 日期值。更多信息见 19.3 节。

注意：应鼓励日期值的接收方尽量接受可能由非 HTTP 应用程序发送的日期值，这在通过代理/网关获取或上传消息到 SMTP 或 NNTP 时常发生。

所有 HTTP 日期/时间戳【必须】以格林尼制标准时间（GMT）表示，没有例外。对于 HTTP 的目的而言，GMT 几乎与 UTC（座标通用时间）相等。在前两种格式中通过包含“” GMT“作为 3 字母时区缩写来指出，且【必须】在读 asctime 格式时推测。HTTP 日期是大小写敏感的，且【禁止】在语法规定包括 SP 之外包括额外的 LWS。

HTTP 日期 = rfc1123 日期 | rfc850 日期 | asctime 日期
rfc1123 日期 = 星期缩写“,” SP 日期 1 SP 时间 SP“GMT”
rfc850 日期 = 星期“,” SP 日期 2 SP 时间 SP“GMT”
asctime 日期 = 星期缩写 SP 日期 3 SP 时间 SP 4DIGIT
日期 1 = 2DIGIT SP 月份 SP 4DIGIT ； 日 月 年（如，02 Jun 1982）
日期 2 = 2DIGIT“-”月份“-”2DIGIT ； 日-月-年（如，02-Jun-82）
日期 3 = 月份 SP (2DIGIT | (SP 1DIGIT)) ； 月 日（如，Jun 2）
时间 =2DIGIT“:”2DIGIT“:”2DIGIT ； 00:00:00 - 23:59:59
星期缩写 =“Mon”|“Tue”|“Wed”|“Thu”|“Fri”|“Sat”|“Sun”

星期 =“Monday”|“Tuesday”|“Wednesday”|“Thursday”
|“Friday”|“Saturday”|“Sunday”
月份 =“Jan”|“Feb”|“Mar”|“Apr”|“May”|“Jun”
|“Jul”|“Aug”|“Sep”|“Oct”|“Nov”|“Dec”

注意：HTTP 要求该日期/时间戳只应用到协议流中它们使用的地址。客户端和服务器不需要使用这些格式作为用户表述、请求日志、等。

3.3.2 变化秒数

一些 HTTP 头部域允许指定时间值为整数秒数，表示为十进制，在收到消息的时刻后。

变化秒数 = 1*DIGIT

3.4 字符集

HTTP 使用与 MIME 中描述的术语“字符集”相同的定义：

术语“字符集”用在本文档中来引用使用一个或多个表格来转换字节序列为字符序列的方法。要注意的是在相反方向上的无条件转换不是必须的，在给定字符集中不是所有的字符可以有效，且字符集可以提供多于一个字节序列来表述特定字符。这种定义目的是允许积压种类型的字符编码，从简单单表映射如 US ASCII 到复杂表交换方法如使用 ISO-2022 技术的这类。然而，与 MIME 字符集名称相关的定义【必须】完全规定从字节到字符所执行的映射。特别是，使用外部轮廓信息来判断确切的映射是不允许的。

注意：术语“字符集”的使用比引用为“字符编码”更普遍。然而，因 HTTP 和 MIME 共享相同的注册表，所以共享术语学是重要的。

HTTP 字符集由大小写不敏感的符号标识。全部符号集由 IANA 字符集注册表 [19]定义。

字符集 = 符号

尽管 HTTP 允许任意符号用作字符集的值，在 IANA 字符集注册表 [19]中预先定义的任何符号【必须】表示由该注册表中定义的字符集。应用程序【应该】限制使用这此由 IANA 注册表定义的字符集。

实现者应该清楚 IETF 字符集要求 [38] [41]。

3.4.1 缺少字符集

一些 HTTP/1.0 软件将 Content-Type 头部没有正确的字符集参数解释为“接收方应该猜测”的意思。发送方希望避免这种行为【可以】包括字符集参数，即使当字符集是 ISO-8859-1 时，且当它知道接收方不会混淆的时候也【应该】这样做。

不幸的是，一些老的 HTTP/1.0 客户端没有正确处理明确的字符集参数。HTTP/1.1 接收方【必须】尊重发送方提供的字符集标签，且这些规定“猜测”字符集的用户代理在最初显示文档时【必须】使用 Content-Type 域中的字符集，如果它们支持该字符集的话，而不是接收方的选项。见 3.7.1 节。

3.5 内容编码

内容编码值指出已经或能够应用到实体上的编码转换。内容编码主要用来允许文档压缩或其它不丢失其底层媒体类型特征和信息的有用的转换。实体常常存储为编码形式，直接转换，且只收接收方解码。

内容编码 = 符号

所有内容编码值都是大小写不敏感的。HTTP/1.1 使用 Accept-Encoding（14.3 节）和 Content-Encoding（14.11 节）头部域中的内容编码值。尽管该值描述内容编码，它是非常重要的，它将指出需要使用哪种解码机制来恢复编码。

因特网分配数字权威（IANA）拌演内容编码值符号的注册表，该注册表包含如下符号：

gzip 一种编码格式，由压缩程序“gzip”（GNU zip）生成，如 RFC 1952 [25] 中所述。该格式是 Lempel-Ziv 编码（LZ77），包含 32 位 CRC。

compress 该编码格式，由通用 UNIX 文件压缩程序“compress”产生。该格式是自适应 Lempel-Ziv-Welch 编码（LZW）。

使用程序名来标识编码格式不是理想的，且不鼓励用于将来的编码。这里使用它们是历史实践的典型，不是好的设计。为了与以前的 HTTP 实现相兼容，应用程序【应该】认为“x-gzip”和“x-compress”分别与“gzip”和“compress”相同。

deflate 定义在 RFC 1950 [31]中的“zlib”格式，并与 RFC 1951 [29]中描述的“deflate”压缩机制组合在一起。

identity 缺省（identity）编码，不使用任何转换。该编码方式只用在 Accept-Encoding 头部中，且【不该】用在 Content-Encoding 头部中。

新的内容编码值符号【应该】注册；以允许客户端与服务器协同工作，内容编码算法的规范需要实现新的值，独立实现【应该】对公众可用和适用，且遵守本节中定义的内容编码目的。

3.6 传输编码

传输编码值用来指示已经、能够、或可能需要应用到实体主体上以确保通过网络“安全传输”的编码转换。这与内容编码是不同的，其中传输编码是消息而非原始实体的属性。

传输编码 =“chunked”| 传输扩展
传输扩展 = 符号 *(“;”参数)

参数是属性/值对的形式。

参数 = 属性“=”值
属性 = 符号
值 = 符号 | 引用字符串

所有传输编码值是大小写不敏感。HTTP/1.1 在 TE 头部域(14.39 节)和 Transfer-Encoding 头部域(14.41 节)中使用传输编码值。不管何时传输编码应用到消息主体，传输编码集【必须】包括“chunked”，除非消息止于连接关闭。当使用“chunked”传输编码时，它【必须】是最后应用到消息主体上的传输编码。“chunked”传输编码【禁止】多次应用到消息主体上。该规则允许接收方判断消息传输的长度(4.4 节)。

传输编码与 MIME [7]中的 Content-Transfer-Encoding 值相似，设计用来允许通过 7 位传输服务安全地传输二进制数据。然而，安全传输对于 8 位清位的传输协议来说有不同的焦点，或希望加密在共享传输上的数据。

因特网分配数字权威(IANA)拌演传输编码值符号的注册表。刚开始，注册表包含下面的符号：“chunked”(3.6.1 节)、“identity”(3.6.2 节)、“gzip”(3.5 节)、“compress”(3.5 节)、和“deflate”(3.5 节)。新传输编码值符号【应该】以同样的方式注册，如新内容编码值符号一样(3.5 节)。

服务器收到的实体主体有它不理解的传输编码，则【应该】返回 501(Unimplemented)，且关闭连接。服务器【禁止】向 HTTP/1.0 客户端发送传输编码。

3.6.1 chunked 传输编码

chunked 编码修饰消息的主体，为了以一系列的大块传输它，每块都有它自己的大小指示符，接在【可选】的包含实体头部域的尾巴后。这允许动态生成的内容与必要信息一起传输，以便接收者检验它已经接收完全部的消息。

大块主体 =*大块 最后大块 尾巴 CRLF
大块 = 大块大小 [大块扩展] CRLF 大块数据 CRLF
大块大小 = 1*HEX
最后大块 = 1*(“0”) [大块扩展] CRLF
大块扩展 = *(“;”大块扩展名 [“=”大块扩展值])
大块扩展名= 符号
大块扩展值 = 符号 | 引用字符串
大块数据 = 大块大小(OCTET)
尾巴 =*(实体头部 CRLF)

大块大小域是十六进制数字串，指出大块的大小。chunked 编码由任何大小为 0 的大块

结束，接着是尾巴，由空行终止。

尾巴允许发送方包括附加的 HTTP 头部域在消息的结尾。Trailer 头部域能够用来指示尾巴中包括哪些头部域（见 14.40 节）。

服务器在响应中使用 chunked 传输编码【禁止】使用任何头部域的尾巴，除非至少满足下面的一条：

a）请求包括 TE 头部域指出“trailers”在响应的传输编码中可接受，如 14.39 节中所述；或者

b）服务器是响应的原始服务器，尾巴域完全由可选的元数据组成，且接收方无需接收该元数据就可以使用该消息（有礼貌地接受原始服务）。换句话说，原始服务器愿意接受这样的可能性，即尾巴域可能在到客户端的路径上被悄悄地放弃。

该要求阻止协同工作失败，当消息由 HTTP/1.1（或更近）的代理接收，并转发到 HTTP/1.0 接收方时。它避免这种状况，代理若遵照协议将必须使用可能是无限的缓冲区。

chunked 主体解码的例程见附录 19.4.6。

所有的 HTTP/1.1 应用程序【必须】能够接收和解码“chunked”传输编码，且如果它们不理解时【必须】乎略大块扩展的扩展项。

3.7 媒体类型

HTTP 在 Content-Type（14.17 节）和 Accept（14.1 节）头部域中使用因特网媒体类型 [17]，为了提供打开和可扩展的数据类型和类型协议。

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

parameter【可以】接在 type/subtype 后面，按 attribute/value 对的形式（如 3.6 节中所定义）。

type、subtype 和 parameter 属性名是大小写非敏感的。parameter 值可以是或不是大小写敏感的，取决于 parameter 名称的语义。【禁止】在 type 和 subtype 间使用线性空白符（LWS），属性与其值间也禁止。存在或缺少 parameter 可以对 media-type 处理有意义，取决于媒体类型注册表中的定义。

要注意，一些老的 HTTP 应用程序不认识媒体类型参数。当发送数据给老 HTTP 应用程序时，实现【应该】只在该 type/subtype 定义需要时使用媒体类型的参数。

media-type 的值由因特网分配数字权威（IANA [19]）注册。媒体类型注册过程在 RFC

1590 [17]中略述。不鼓励使用非注册媒体类型。

3.7.1 规范化和文本缺省值

因特网媒体类型以规范的形式注册。实体主体通过 HTTP 消息传输【必须】在其传输前以适当的规范化形式描绘，除非是“text”类型，如下段所定义。

当按规范化形式时，媒体“text”类型的 subtype 使用 CRLF 作为文本断行符。HTTP 放松该项要求，允许传输文本媒体，以无格式的单独 CR 或 LF 描绘的断行符，当整个实体主体都终止如一的这样做时。HTTP 应用程序【必须】接受 CRLF，光 CR，和光 LF 作为描绘通过 HTTP 接收的文本媒体的断行符。此外，如果文本以不支持字节 13 和 10 作为对应 CR 和 LF 的字符集描绘时，如同对于一些多字节字符集的情况，HTTP 允许使用由该字符集所定义来描绘等价的 CR 和 LF 的任何字节序列作为断行符。这种对待断行符的灵活性只应用于实体主体中的文本类型，光 CR 或 LF【禁止】取代 HTTP 控制结构中的任何 CRLF（如头部域和 multipart 边界）。

如果实体主体按内容编码来编码，则下层数据【必须】在编码前按前面定义的形式。“字符集”参数与一些媒体类型联合使于定义数据的字符集（3.4 节）。当发送方没有提供明确的字符集参数时，通过 HTTP 接收的媒体“text”类型的 subtype 定义为有缺省的“ISO-8859-1”字符集值。不在“ISO-8859-1”或其子集的字符集中的数据【必须】标识适当的字符集值。兼容性问题见 3.4.1 节。

3.7.2 multipart 类型

MIME 提供了一些“multitype”类型——在单个消息主体中封装一个或多个实体。所有的 multitype 类型共享通用语法，如 RFC 2046 [40]中的 5.1.1 节所定义，且【必须】包括 boundary 参数作为媒体类型值的一部分。消息主体自己是协议的元素且因此【必须】在主体各部分间只使用 CRLF 来描绘断行符。与 RFC 2046 不同的是，任何 multipart 消息的结尾【必须】是空的；HTTP 应用【禁止】传输结尾（即使原始 multipart 包含结尾）。存在该限制是为了保持 multipart 消息主体的自定义属性，这里的消息主体“end”由结尾的 multipart 边界来指出。

通常，HTTP 会无区别对待 multipart 消息主体与任何其它媒体类型：严格作为负载。一种例外是“multipart/byteranges”类型（附录 19.2），当其出现在 206（Partial Content）响应中时，一些 HTTP 缓存机制会将其按 13.5.4 节和 14.16 节所描述的来解释。在其它所有情况下，HTTP 用户代理【应该】按与 MIME 用户代理接收 multipart 类型的相同或相似行为。如果应用程序接收不认识的 multipart 的 subtype，则应用程序【必须】将其视为与“multipart/mixed”相同。

注意：“multipart/form-data”类型特别定义来挟带适合 POST 请求方法处理的表单数据，如 RFC 1867 [15]中所述。

3.8 product 符号

product 符号用来允许通迅应用程序通过软件名称和版本号来标识它们自己。大多域使

用 product 的符号还允许 sub-prodrurect 来形成通过空白符分隔列出的应用程序的有意义部分。作为习惯，列出 product，为其标识应用程序的意义。

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

例子：

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

product 符号【应该】简短且切中要害。它们【禁止】用于广告或其它非本质的信息。尽管任何符号集都【可以】出现在 product-version 中，该符号【应该】只用于版本标识（如，相同 product 的后继版本只【应该】在 product 值的 product-version 部分有区别）。

3.9 质量值

HTTP 内容协商（12 节）使用简短的“浮点”数来指出各种协商参数的相对重要性（“权重”）。权重通常是 0 到 1 范围内的实数，这里 0 是最小的，1 是最大的。如果某个参数的质量值为 0，则该参数对于该客户来说“不可接受”。HTTP/1.1 应用程序【禁止】在十进制小数点后生成超过 3 位数。这些值的用户配置【应该】也限制于这种形式。

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

“质量值”是个误导的称呼，因为这些值很少表示期望质量的相关等级。

3.10 语言标签

语言标签标识自然语言说、写、或其它人类间相互传达信息来通迅的方式。计算机语言被明确排除在外。

HTTP 在 Accept-Language 和 Content-Language 域中使用语言标签。HTTP 语言标签的语法和注册表与 RFC 1766 [1]中定义的相同。简而言之，语言标签由 1 个或多个部分组成：主语言标签和可能为空的 subtag 序列：

language-tag = primary-tag *("-" subtag)
primary-tag = 1*8ALPHA
subtag = 1*8ALPHA

在标签间不允许空白符，且所有标签都是大小写非敏感的。语言标签的命名空间由 IANA 管理。标签的例子包括：en, en-US, en-cockney, i-cherokee, x-pig-latin

任何双字母的 primary-tag 是 ISO-639 语言缩写，且任何前 2 个字母大写的 subtag 是

ISO-3166 国家代码。（上面的最后 3 个标签是非注册标签；所有除了最后的标签例子都可能在将来注册）

3.11 实体标签

实体标签用来从相同的请求资源来比较两个或多个实体。HTTP/1.1 在 ETag（14.19 节）、If-Match（14.24 节）、If-None-Match（14.26 节）和 If-Range（14.27 节）头部域中使用实体标签。关于如何使用它们的定义和作为缓存证言的比较在 13.3.3 节中。实体标签由不透明的引用字符串组成，可能在前面放上弱证言。

```
entity-tag = [ weak ] opaque-tag
weak = "W/"
opaque-tag = quoted-string
```

“强实体标签”【可以】在两个资源的实体间共享，仅当它们是字节比较相同的。“弱实体标签”，用“W/”前缀指示，【可以】在两个资源的实体间共享，仅当这些实体是相等的，且可以在不明显改变语义的情况下互相替代。弱实体标签只可用在弱比较。

实体标签【必须】是在与特定资源相关的所有实体的所有版本间统一的。给定的实体标签值【可以】用在从不同的 URI 请求获取的实体上。使用与从不同的 URI 请求获取的实体相关联的相同实体标签值前不意味着这些实体是相同的。

3.12 范围单位

HTTP/1.1 允许客户端只请求响应实体的某部分（范围）包括在响应中。HTTP/1.1 在 Range（14.35 节）和 Content-Range（14.16 节）头部域中使用范围单位。实体可以依据各种结构化单位分割为子范围。

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

HTTP/1.1 定义的唯一范围单位是“bytes”。HTTP/1.1 实现【可以】乎略使用其它单位的范围规定。HTTP/1.1 已经设计以允许应用程序的实现不取决于范围的知识。

4 HTTP 消息

4.1 消息类型

HTTP 消息由从客户端到服务器的请求和从服务器到客户端的响应组成。

```
HTTP-message = Request | Response ; HTTP/1.1 messages
```

Request（5 节）和 Response（6 节）消息使用 RFC 822 [9]的 generic-message 格式来传输实体（消息负载）。两种消息全由 start-line、0 个或多个头部域（还称为“头部”），空行（如，

CRLF 前没有任何东西的行）表示头部域结束，和可能的 message-body。

```
generic-message = start-line *(message-header CRLF) CRLF [ message-body ]
start-line = Request-Line | Status-Line
```

由于精力充沛的原因，服务器【应该】乎略在希望收到 Request-Line 地方的空行。换句话说，如果服务器在读取协议流的初始消息且收到 CRLF 打头，则它应该乎略 CRLF。已知有问题的 HTTP/1.0 客户端实现生成额外的 CRLF 在 POST 请求之后。重新声明，BNF 明确禁止 HTTP/1.1【禁止】客户端以额外的 CRLF 引导或结束请求。

4.2 message-header

HTTP 头部域，包括 general-header（4.5 节）、request-header（5.3 节）、response-header（6.2 节）、和 entity-header（7.1 节）域，按 RFC 822 [9]的 3.1 节中给出的相同的一般格式。敏个头部域由一个跟着冒号（“:”）的名称和域值组成。域名是大小写不敏感的。域值【可以】由任何数量的 LWS 引导，尽管首选是单个 SP。头部域能够扩展到多行，通过用至少一个 SP 或 HT 引导扩展行。应用程序应该按“通用形式”，已知或指出的，当生成 HTTP 构造时，由于可能存在一些实现不能接受非通用形式的任何东西。

```
message-header = field-name ":" [ field-value ]
field-name = token
field-value = *( field-content | LWS )
field-content = <组成 field-value 的 OCTET，
                和*TEXT 的组成或组合 token、separators 和 quoted-string 的组成>
```

field-content 不包括任何前导或结尾的 LWS：线性空白符出现在 field-value 的首个非空白字符前或 field-value 的最后的非空白字符后。【可以】删除这种前导或结尾的 LWS 而一改变 field-value 的语义。在解析 field-value 或转发消息到下游前【可以】将出现在 field-content 间的任意个 LWS 替换为单个 SP。

收到不同 field-name 头部域的顺序没有意义。然而，“好实践”结果是，首先发送 general-header，接着是 request-header 或 response-header 域，最后是 entity-header 域。

多个 field-name 相同的 message-header 域【可以】出现在一个消息中，当且仅当该头部域的整个 field-value 定义为逗号分隔的列表 [如，#(值)] 时。【必须】能够组合多个头部域为单个“field-name: field-value”对，而不改变消息的语义，通过追加每个后序 field-value 到第一个，之间由逗号分隔开。因此收到相同 field-name 头部域的顺序对于组合 field-value 的解析是有意义的，因此代理【禁止】在转发消息时改变这些 field-value 的顺序。

4.3 message-body

HTTP 消息的 message-body（如果存在）用于挟带与请求或响应相关联的 entity-body。message-body 只有在应用了 transfer-coding 时，通过 Transfer-Encoding 头部域指出（14.41 节），与 entity-body 不同。

message-body = entity-body | <按每个 Transfer-Encoding 编码的 entity-body>

Transfer-Encoding【必须】用于指出任何应用程序所应用的 transfer-coding 来确保安全和正确的消息传输。Transfer-Encoding 是消息的属性，不是实体和，因此请示/响应链上的任何应用程序【可以】增加或删除它。（然而，3.6 节作出一些关于使用确知的 transfer-coding 时的限制。）

当允许消息中有 message-body 时的规则对于请求和响应而言是不同的。

请求中存在 message-body 的信号由在请求的 message-header 中引入 Content-Length 或 Transfer-Encoding 来指示。如果请求方法（5.1.1 节）的规定不允许在请求中发送 entity-body 时，【禁止】在请求中包括 message-body。服务器【应该】读取和转发任何请求上的 message-body；如果请求方法没有包括 entity-body 的已定义语义时，在处理请求时【应该】乎略 message-body。

对于响应消息，消息是否包括 message-body 既取决于请求方法，也取决于响应状态码（6.1.1 节）。对 HEAD 请求方法的所有响应【禁止】包括 message-body，即使存在的 entity-header 域可能使接收方确信其中包括 message-body。所有 1xx（信息性）、204（No Content）、和 304（Not Modified）响应【禁止】包括 message-body。所有其它响应确实包括 message-body，即使其长度【可以】是 0。

4.4 消息长度

消息的 transfer-length 是 message-body 的长度，如果它出现在消息中；即，在已经应用任何 transfer-coding 后。当 message-body 包括在消息中时，该 message-body 的 transfer-length 按下面的一条来判断（按优先级顺序）：

1、“【禁止】”包括 message-body 的任何响应（如 1xx、204 和 304 响应、及对 HEAD 请求的任何响应）始终在头部域的首个空行后终止，而不管消息中的 entity-header 域。

2、如果存在 Transfer-Encoding 头部域（14.41 节），且其值不是“identity”，这时 transfer-length 由“chunked”transfer-coding（3.6 节）定义，除非消息由连接关闭终止。

3、如果存在 Content-Length 头部域（14.13 节），则它的 OCTET 十进制值既表示 entity-length 又表示 transfer-length。如果这两个长度不同时（如，当 Transfer-Encoding 头部域存在时）【禁止】发送 Content-Length 头部域。如果收到的消息既有 Transfer-Encoding 头部域，又有 Content-Length 头部域，则【必须】乎略后者。

4、如果消息使用“multipart/byteranges”媒体类型，且没有另外指定 transfer-length，则该自定界的媒体类型定义 transfer-length。【禁止】使用该媒体类型，除非发送方知道接收方能够解析它；1.1 客户端请求中存在有多个 byte-range 规范符的 Range 头部暗示该客户端能够解析 multipart/byteranges 响应。1.0 代理，不理解 multipart/byteranges，可以转发 Range 头部；在这种情况下，服务器【必须】使用本节中的 1、3 或 5 项定义的方法来定界消息。

5、通过服务器关闭连接。（不能使用关闭连接来指出请求主体的结束，由于这将引起服务器不可能回送响应。）

为了与 HTTP/1.0 应用程序兼容，包含 message-body 的 HTTP/1.1 请求【必须】包括有效的 Content-Length 头部域，除非已知服务器与 HTTP/1.1 兼容。如果请求包含 message-body 但没有给出 Content-Length，则服务器【应该】在不能判断消息长度时用 400（Bad Request）响应，或在希望坚持接收有效 Content-Length 时用 411（Length Required）响应。

收到实体的所有 HTTP/1.1 应用程序【必须】接受“chunked”的 transfer-coding（3.6 节），允许在将来不能判断消息长度时将该机制用于消息。【禁止】消息既包括 Content-Length 头部域，又包括非 identity 的 transfer-coding。如果消息确实包括非 identity 的 transfer-coding，则【必须】乎略 Content-Length。当在允许 message-body 的消息给出 Content-Length 时，它的 field-value【必须】精确匹配 message-body 的 OCTET 数量。HTTP/1.1 用户代理【必须】提醒用户接收并检测到无效长度。

4.5 general-header 域

有几个头部域一般可应用于请求和响应消息上，但不应用到传输的实体上。这些头部域只应用到正在传输的消息上。

```
general-header = Cache-Control ; 14.9 节
                | Connection ; 14.10 节
                | Date ; 14.18 节
                | Pragma ; 14.32 节
                | Trailer ; 14.40 节
                | Transfer-Encoding ; 14.41 节
                | Upgrade ; 14.42 节
                | Via ; 14.45 节
                | Warning ; 14.46 节
```

只有与协议版本同时修改时，general-header 域名才能可靠地扩展。然而，新的或试验性头部域可以给出到 general-header 域的语义中，如果通迅中的所有对象识别它们是 general-header 域。不认识的头部域作为 entity-header 域对待。

5 Request

在消息的首先中，从客户端到服务器的请求消息包括应用到资源的方法、资源的标识符和使用的协议版本。

```
Request = Request-Line ; 5.1 节
          *(( general-header ; 4.5 节
          | request-header ; 5.3 节
          | entity-header ) CRLF) ; 7.1 节
```

CRLF
[message-body] ; 节 4.3

5.1 Request-Line

Request-Line 始于方法符号，接收是 Request-URI 和协议版本，由 CRLF 结束。元素间用 SP 字符分隔。除了最终的 CRLF 序列，不允许 CR 或 LF。

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

5.1.1 Method

Method 符号指出在由 Request-URI 标识的资源上所执行的方法。方法是大小写敏感的。

Method = "OPTIONS" ; 9.2 节
| "GET" ; 9.3 节
| "HEAD" ; 9.4 节
| "POST" ; 9.5 节
| "PUT" ; 9.6 节
| "DELETE" ; 9.7 节
| "TRACE" ; 9.8 节
| "CONNECT" ; 9.9 节
| extension-method
extension-method = token

资源允许的方法列表能够在 Allow 头部域（14.7 节）中规定。响应的返回码始终通知客户端资源当前是否允许该方法，由于允许方法集能够动态改变。如果原始服务器知道该方法便所请求资源不允许，则原始服务器【应该】返回状态码 405（Method Not Allowed），如果原始服务器不认识该方法或没有实现，则返回 501（Not Implemented）。所有通用目的服务器【必须】支持 GET 和 HEAD 方法。所有其它方法是【可选的】；然而，如果实现了上面的方法，则【必须】按 9 节中规定的相同语义来实现它们。

5.1.2 Request-URI

Request-URI 是统一资源标识符（3.2 节），标识应用到请求上的资源。

Request-URI = "*" | absoluteURI | abs_path | authority

Request-URI 的 4 个选择取决于请求的特性。星号“*”意思是请求不应用到特定的资源，而是服务器自己，且只在所使用的方法没必要应用到资源时允许。一个例子可能是

OPTIONS * HTTP/1.1

当请求由代理作出时，absoluteURI 形式是【要求的】。请求代理转发请求或从有效缓存

来服务它，并返回响应。要注意，代理【可以】转发请求到其它代理或直接到由 absoluteURI 规定的服务器。为了避免循环请求，代理【必须】能够认识所有它的服务器名称，包括任何别名，本地变称、和数字 IP 地址。Request-Line 的一个例子可能是：

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

为了允许在以后的 HTTP 版本中对所有请求中的 absoluteURI 进行转换，所有 HTTP/1.1 服务器【必须】在请求中接受 absoluteURI 形式，即使 HTTP/1.1 客户端将只在向代理请求时生成它们。

只有 CONNECT 方法（9.9 节）使用 authority 形式。

Request-URI 最通用的形式用于标识原始服务器或网关上的资源。在这种情况下，URI 的绝对路径（见 3.2.1 节，abs_path）【必须】传送为 Request-URI，URI 的网络地址（authority）【必须】在 Host 头部域中传送。例如，客户端希望直接从原始服务器上获取资源，可以创建端口号 80 的 TCP 连接到主机“www.w3.org”并发送消息行：

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

接着是 Request 的余下部分。要注意，绝对路径不能为空；如果原始 URI 中不存在，则它【必须】给出“/”（服务器根）。

Request-URI 以 3.2.1 节中规定的格式传输。如果 Request-URI 使用“% HEX HEX”编码方式 [42]来编码，则原始服务器【必须】解码 Request-URI，以便正确解析请求。服务器【应该】以适当的状态码响应无效的 Request-URI。【禁止】透明代理在转发请求到入界服务器时重写所收到的 Request-URI 的“abs_path”部分，除非上面说明的替换空 abs_path 为“/”。

注意：当原始服务器为保留目的不正确使用非保留 URI 字符时，“不重写”规则阻止代理改变请求的意思。实现者应该清楚，已知一些早期 HTTP/1.1 代理会重写 Request-URI。

5.2 Request 标识的资源

因特网请求标识的确切资源是由检查 Request-URI 和 Host 头部域判断的。原始服务器不允许资源与请求主机不同，【可以】在判断由 HTTP/1.1 请求标识的资源时乎略 Host 头部域值（HTTP/1.1 支持 Host 的其它要求还可见 19.6.1.1 节）。原始服务器要区别对待基于主机请求的资源（有时候用作虚拟主机或空主机名）【必须】例如如下的规则来判断 HTTP/1.1 请求所请求的资源：

1、如果 Request-URI 是 absoluteURI，主机是 Request-URI 的一部分。【必须】乎略请求中的任何 Host 头部域值。

2、如果 Request-URI 不是 absoluteURI，且请求包括 Host 头部域，则主机由 Host 头部域值决定。

3、如果规则 1 或 2 决定的主机在服务器上不是有效的主机，则响应【必须】是 400（Bad Request）错误消息。

缺少 Host 头部域的 HTTP/1.0 请求的接收方【可以】尝试使用启发式（如，为某些统一到特定的主机检查 URI 的路径）判断正请求的确切资源。

5.3 request-header 域

request-header 域允许客户端传输关于请求和关于客户端自己的额外信息给服务器。这些域拌演请求修饰符，与编程语言方法符号上的参数有相同的语义。

```
request-header = Accept ; 14.1 节
                 | Accept-Charset ; 14.2 节
                 | Accept-Encoding ; 14.3 节
                 | Accept-Language ; 14.4 节
                 | Authorization ; 14.8 节
                 | Expect ; 14.20 节
                 | From ; 14.22 节
                 | Host ; 14.23 节
                 | If-Match ; 14.24 节
                 | If-Modified-Since ; 14.25 节
                 | If-None-Match ; 14.26 节
                 | If-Range ; 14.27 节
                 | If-Unmodified-Since ; 14.28 节
                 | Max-Forwards ; 14.31 节
                 | Proxy-Authorization ; 14.34 节
                 | Range ; 14.35 节
                 | Referer ; 14.36 节
                 | TE ; 14.39 节
                 | User-Agent ; 14.43 节
```

request-header 域名能够靠与协议版本修改一起扩展。然而，当通迅中所有成员都认识这些 request-header 域时，【可以】为 request-header 域的语义给出新的或实验性头部域。不认识的头部域将作为 entity-header 域对待。

6 Response

在接收并解析请求消息后，服务器以 HTTP 响应消息响应。

```
Response = Status-Line ; 6.1 节
           *(( general-header ; 4.5 节
           | response-header ; 6.2 节
           | entity-header ) CRLF) ; 7.1 节
```

CRLF
[message-body] ; 7.2 节

6.1 Status-Line

Request 消息的首行是 Status-Line，由协议版本，接着数字状态码和相关的文本短语组成，每个元素间由 SP 字符分隔。除了最后的 CRLF 序列，不允许 CR 或 LF。

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

6.1.1 Status-Code 和 Reason-Phrase

Status-Code 元素是 3 位整数结果码，尝试理解和满足请求。这此代码的完整定义在 10 节中。Reason-Phrase 意图给出简短的 Status-Code 的文本描述。Status-Code 意图用在自动控制，Reason-Phrase 意图给人类用户。不要求客户端检查或显示 Reason-Phrase。

Status-Code 的首位数字定义响应的类别。最后两个数字没有任何分类。首位数字有 5 个值：

- 1xx: 信息性——收到请求，继续处理
- 2xx: 成功性——成功收到、理解并接受行动
- 3xx: 重定向——必须采取进一步行动来完成请求
- 4xx: 客户端错误——请求包含错误语法或不能完成
- 5xx: 服务器错误——服务器没有成功完成显然有效的请求

HTTP/1.1 定义了数字状态码的单个值，下面介绍相应的 Reason-Phase 集的例子。这里列出的原因短语只是推荐的——【可以】不影响协议的情况下用本地化等价的短语替换它们。

Status-Code = "100" ; 10.1.1 节: Continue
| "101" ; 10.1.2 节: Switching Protocols
| "200" ; 10.2.1 节: OK
| "201" ; 10.2.2 节: Created
| "202" ; 10.2.3 节: Accepted
| "203" ; 10.2.4 节: Non-Authoritative Information
| "204" ; 10.2.5 节: No Content
| "205" ; 10.2.6 节: Reset Content
| "206" ; 10.2.7 节: Partial Content
| "300" ; 10.3.1 节: Multiple Choices
| "301" ; 10.3.2 节: Moved Permanently
| "302" ; 10.3.3 节: Found
| "303" ; 10.3.4 节: See Other
| "304" ; 10.3.5 节: Not Modified
| "305" ; 10.3.6 节: Use Proxy
| "307" ; 10.3.8 节: Temporary Redirect

```
                | "400" ; 10.4.1 节: Bad Request
                | "401" ; 10.4.2 节: Unauthorized
                | "402" ; 10.4.3 节: Payment Required
                | "403" ; 10.4.4 节: Forbidden
                | "404" ; 10.4.5 节: Not Found
                | "405" ; 10.4.6 节: Method Not Allowed
                | "406" ; 10.4.7 节: Not Acceptable
                | "407" ; 10.4.8 节: Proxy Authentication Required
                | "408" ; 10.4.9 节: Request Time-out
                | "409" ; 10.4.10 节: Conflict
                | "410" ; 10.4.11 节: Gone
                | "411" ; 10.4.12 节: Length Required
                | "412" ; 10.4.13 节: Precondition Failed
                | "413" ; 10.4.14 节: Request Entity Too Large
                | "414" ; 10.4.15 节: Request-URI Too Large
                | "415" ; 10.4.16 节: Unsupported Media Type
                | "416" ; 10.4.17 节: Requested range not satisfiable
                | "417" ; 10.4.18 节: Expectation Failed
                | "500" ; 10.5.1 节: Internal Server Error
                | "501" ; 10.5.2 节: Not Implemented
                | "502" ; 10.5.3 节: Bad Gateway
                | "503" ; 10.5.4 节: Service Unavailable
                | "504" ; 10.5.5 节: Gateway Time-out
                | "505" ; 10.5.6 节: HTTP Version not supported
                | extension-code
    extension-code = 3DIGIT
    Reason-Phrase = *<TEXT，除 CR, LF 外>
```

HTTP 状态码可扩展。不需要 HTTP 应用程序理解所有注册状态码的含义，尽管这种理解显然是期望的。然而，应用程序【必须】理解任何状态码的类别，由首位数字指出，并对待任何不认识的响应等同为该类的 x00 状态码，除了【禁止】缓存不认识的响应。例如，如果客户端收到不认识的状态码 431，则它可能安全地假设它的请求有一些错误，并对待该响应为如同它已经收到 400 状态码。在这种情况下，用户代理【应该】为用户呈现返回的响应中的实体，由于该实体很可能包含人类可读的信息，其中会解释该非正常状态。

6.2 response-header 域

response-heder 域允许服务器传递不能放在 Status-Line 中的关于响应的额外信息。这些头部域给出关于服务器和关于对 Request-URI 所标识资源的以后访问。

```
    response-header = Accept-Ranges ; 14.5 节
                    | Age ; 14.6 节
                    | ETag ; 14.19 节
                    | Location ; 14.30 节
```

| Proxy-Authenticate ; 14.33 节
| Retry-After ; 14.37 节
| Server ; 14.38 节
| Vary ; 14.44 节
| WWW-Authenticate ; 14.47 节

response-header 域名可能与协议版本修改一起可靠的扩展。然而，当通迅中所有成员都认识这些 response-header 域时，【可以】为 response-header 的语义给出新的或试验性头部域。不认识的头部域按 entity-header 对待。

7 实体

如果请求方法或响应状态码没有其它限制的话，Request 和 Response 消息【可以】传输实体。实体由 entity-header 和 entity-body 组成，尽管一些响应只包括 entity-header。

在本节中，发送方和接收方引用为客户端或服务器，取决于谁发送和谁接收实体。

7.1 entity-header 域

entity-header 域定义关于 entity-body 的元信息，或当主体不存在时关于请求标识的资源。该元信息中的一些是【可选的】；一些可能是【要求的】，由该规范的各部分定义。

entity-header = Allow ; 14.7 节
| Content-Encoding ; 14.11 节
| Content-Language ; 14.12 节
| Content-Length ; 14.13 节
| Content-Location ; 14.14 节
| Content-MD5 ; 14.15 节
| Content-Range ; 14.16 节
| Content-Type ; 14.17 节
| Expires ; 14.21 节
| Last-Modified ; 14.29 节
| extension-header
extension-header = message-header

extension-header 机制允许定义额外的 entity-header 域而不改变协议，但不能假设接收方认识这些域。接收方【应该】乎略不认识的头部域，且透明代理【必须】转发它。

7.2 entity-body

entity-body（如果有）按 entity-header 域定义的格式和编码与 HTTP 请求或响应一起发送。

entity-body = *OCTET

entity-body 只出现在存在 message-body 的消息中，如 4.3 节所述。entity-body 通过解码任何可能应用以确保安全和正确传输消息的 Transfer-Encoding 的 message-body 来获取。

7.2.1 类型

当 entity-body 包括在消息中，则其数据类型由头部域 Content-Type 和 Content-Encoding 决定。这里定义 2 层，按顺序的编码模型：

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

Content-Type 规定下层数据的媒体类型。Content-Encoding 可用于指出应用到数据上的任何额外内容编码，通常为数据压缩用途，它是所请求资源的属性。没有缺省的编码。

包含 entity-body 的任何 HTTP/1.1 消息都【应该】包括 Content-Type 头部域来定义该主体的媒体类型。当且仅当没有 Content-Type 给出媒体类型时，接收方【可以】通过检查用于标识资源的 URI 的内容和/或其扩展名尝试猜测媒体类型。如果媒体类型还是未知，则接收方【应该】将其按“application/octetstream”类型对待。

7.2.2 entity-length

消息的 entity-length 是在应用任何 transfer-coding 前 message-body 的长度。4.4 节定义如何判断 message-body 的 transfer-length。

8 连接

8.1 永久连接

8.1.1 目的

在永久连接前，建立单独的 TCP 连接来获取每个 URL，将增加 HTTP 服务器负荷，且引起因特网的拥塞。使用在线图片和其它相关数据经常需要客户端在大量短时间中从同个服务器请求多次。分析这种性能问题和原型实现的结果可见于[26][30]。实现经验和实际 HTTP/1.1（RFC 2068）实现的测量显示更好的结果[39]。二选一已经发现，例如，T/TCP[27]。

永久 HTTP 连接有许多优势：

• 通过打开和关闭更少的 TCP 连接，将节约在路由器和主机（客户端、服务器、代理、网关、隧道、或缓存）上的 CPU 时间，且能够在主机上节约 TCP 协议控制块所使用的内存。

• HTTP 请求和响应能够如管道一相通过连接。管道允许客户端一需要等街每个响应就发出多个请求，允许单个 TCP 连接更有效地使用，花费更少的时间。

• 减轻网络拥塞，通过减少由 TCP 打开引出的许多包，和通过允许 TCP 有充分时间来

判断网络的拥塞状态。

• 后序请求的延迟养活了，因为没有花费时间在 TCP 连接打开握手上。

• HTTP 能够更有好地进化，由于错误能够被报告，而不需要关闭 TCP 连接的处罚。客户端使用将来的 HTTP 版本可以乐观地尝试新特性，但如果与老服务器通迅，在错误报告时尝试老语义。

HTTP 实现<应该>实现永久连接。

8.1.2 全面操作

HTTP/1.1 与较早版本的 HTTP 的明显区别是永久连接是任何 HTTP 连接的缺省行为。即，除非另外指示，客户端<应该>假设服务器将维护永久连接，即使在从服务器的错误响应后。

永久连接提供这样的机制。客户端和服务器能够使用永久连接来发信号关闭 TCP 连接。

该信号使用 Connection 头部域（14.10 节）来发生。一旦收到关闭信号，客户端<禁止>再发送任何更多的请求在该连接上。

8.1.2.1 协商

HTTP/1.1 服务器<可以>假设 HTTP/1.1 客户端打算维护永久连接，除非所发送请求中的 Connection 头部中包括连接记号“close”。如果服务器选择在发送响应后立即关闭连接，它<应该>发送包括关闭连接记号的 Connection 头部。

HTTP/1.1 客户端<可以>希望连接保持打开，但将靠服务器的响应中是否包含有关闭连接记号的 Connection 头部来决定维持连接打开。在客户端不希望为除该它以外的更多请求维护连接的情况下，它<应该>发送 Connection 头部，包括关闭连接记号。

如果客户端或服务器两者之一在 Connection 头部中发送关闭记号，则该请求成为该连接上的最后一个。

客户端和服务器<不该>假设为 HTTP 版本号小于 1.1 的维护永久连接，除非它明确发信号。关于与 HTTP/1.0 客户端向后兼容的更多信息见 19.6.2 节。

为了保持永久，所有在连接上的消息<必须>有自定义的消息长度（如，不是定义为连接关闭的一种），如 4.4 节中所述的。

8.1.2.2 管道

支持永久连接的客户端<可以>按“管道”方式请求（如，发送多个请求而不需等待每个响应）。服务器<必须>必须以收到请求的相同顺序来响应发送这些请求的响应。

假设永久连接和在连接建立后立即使用管道的客户端<应该>准备好当首次管道尝试失败后重试它们的连接。如果客户端进行这种重试，则它在知道连接是永久的之前<禁止>使用“管道”方式。客户端还<必须>准备好重发它们的请求，如果服务器在发送所有相应的响应前关闭该连接。

客户端<不该>按“管道”方式请求使用非等效方法或非等效方法序列（见 9.1.2 节）。否则，腰折的传输连接终止可能导致不确定的结果。客户端希望发送非等效请求<应该>等到它已经收到前而请求的响应状态时才发送该请求。

8.1.3 代理服务器

代理正确实现 14.10 节中规定的 Connection 头部域的属性是特别重要的。

代理服务器<必须>与它的客户端和它连接的原始服务器（或其它代理服务器）独立地发送永久连接信号。每个永久连接只应用到一个传输链路。

代理服务器<禁止>与 HTTP/1.0 客户端建立 HTTP/1.1 永久连接（而使用许多 HTTP/1.0 客户端实现的 Keep-Alive 头部的信息和问题见 RFC 2068[33]）。

8.1.4 实践考虑

服务器通常设置一些超时值，超过后它们将不再维持非活动连接。

代理服务器可能将该值设置得很高，由于它希望客户端通过相同的服务器建立更多的连接。永久连接的使用就不存客户端或服务器在对该超时的长度（或存在性）需求。

当客户端或服务器希望超时时，它<应该>引出友好关闭该传输连接。客户端和服务器<应该>持续监视另一方的传输关闭和对它的适当响应。

如果客户端或服务器没有迅速地检测出另一方的关闭，这将可能引起网络资源的不必消耗。

客户端、服务器，或代理<可以>在任何时候关闭传输连接。例如，客户端可能已经开发发送新的请求，同时服务器决定关闭该“空闲”的连接。从服务器的观点，连接在空闲时被关闭，但从客户端的观点，请求正在进行。

这就是说，客户端、服务器、和代理<必须>能够从异步关闭事件中恢复过来。客户端<应该>在无用户干预的情况下重开传输连接和重传终止的请求序列，当该请求序列是等效时（见 9.1.2 节）。非等效方法或序列<禁止>自动重传，尽管用户代理<可以>为人类操作者提代重传该请求的选择。由具有语义理解力的用户代理软件来确认<可以>代理用户确认。该自动重传<不该>被重复，如果第二次请求序列与失败了。

服务器<应该>一直为每次连接的一个请求至少响应一次，如果尽可能的话。服务器<不

该>在传输响应的过程中关闭连接，除非怀疑网络或客户端失败。使用永久连接的客户端<应该>限制它们与给定服务器间维护的同时连接数。单用户客户端<不该>维护超过 2 个连接到任何服务器或代理。代理<应该>使用少于 2*N 的连接到其它的服务器或代理，这里的 N 是同时活动用户数。这种方针试图改善 HTTP 响应时间和避免拥塞。

8.2 消息传输要求

8.2.1 永久连接和流控

HTTP/1.1 服务器【应该】维护永久连接和使用 TCP 流控制机制来解决临时性过载，而不是终止连接，并期望客户端将重试。后种技术将恶化网络拥塞。

8.2.2 监视连接的错误状态消息

HTTP/1.1（或更新）客户端发送消息体【应该】在它传输其请求同时监视网络连接的错误状态。如果客户端发现错误状态，则它【应该】立即终止传输消息体。如果消息体正在使用“大块”编码（3.6 节）发送中，则设置大块长为 0，用来提前标志消息结束。如果消息体先于 Content-Length 头部，则客户端【必须】关闭连接。

8.2.3 使用 100（Continue）状态

100（Continue）状态（见 10.1.1 节）的目的是允许客户端发送请示消息来判断原始服务器是否愿意在客户端发送请求体前接受该请求（基于请求头部）。在一些情况下，它可能是不恰当或高度低效地，即客户端发送消息体而服务器将不查看消息体就拒绝该消息。

HTTP/1.1 客户端的要求：

• 如果客户端愿意在发送请求体前等待 100（Continue）响应，它【必须】发送含有“100-continue”期望的 Expect 请求头部域（14.20 节）。

• 客户端【禁止】在它不会发送请求体时发送含有“100-Continue”期望的 Expect 请求头部域（14.20 节）。

由于存在老的实现，协议允许歧义状态，即客户端可以发送“Expect: 100-continue”而不接收 417（Expectation Failed）状态或 100（Continue）状态。

因此，当客户端发送消息头部域到原始服务器（可能通过代理），但没有从它收到任何 100（Continue）状态，则客户端【不该】在地送请求体前无休止地等待。

HTTP/1.1 原始服务器的要求：

•如收到请求中包括含有“100-continue”期望的 Expect 请求头部域，则原始服务器【必须】要么以 100（Continue）响应关继续从输入流中读取，要么以最终止状态响应。原始服务器【禁止】在发送 100（Continue）响应前等待请求体。如果它以最终状态码响应，则它

【可以】关闭该传输连接或它【可以】继续读取并丢弃余下的请求。它【禁止】在返回最终状态码时执行请求方法。

• 原始服务器【不该】发送 100（Continue）响应，如果请求消息不包括含“100-continue”期望的 Expect 请求头部域，且【禁止】发送 100（Continue）响应，如果请求从 HTTP/1.0（或更早）客户端发出。该规则有个例外：为了兼容于 RFC2068，服务器【可以】在响应中发送 100（Continue）状态到 HTTP/1.1 PUT 或 POST 请求，即使不包括含“”期望的 Expect 请求头部域。该例外目的是最小化任何客户端关于没有声明等待 100（Continue）状态的处理延迟，只应用到 HTTP/1.1 请求，而不是有其它 HTTP 版本值的请求。

• 原始服务器【可以】忽略 100（Continue）响应，如果它已经收到相应请求的一些或全部的请求体。

• 原始服务器发送 100（Continue）响应【必须】最后发送最终状态码，一旦请求体收到并处理了，除非它提前终止传输连接。

• 如果原始服务器收到的请求中不包括含有“100-continue”期望的 Expect 请求头部域，而请求包括请求体，且服务器在从传输连接读取完整的请求体前以最终状态码响应，这时服务器【不该】在它已经读完整个请求或在客户端关闭连接前关闭该传输连接。否则，客户端不可能可靠地收到响应消息。然而，该要求不要解释为阻止服务器防卫对它的拒绝服务攻击或严重崩溃的客户端实现。

HTTP/1.1 代理的要求：

• 如果代理收到请求中包括含有“100-continue”希望的 Expect 请求头部域，且代理知道下一跳的服务器用 HTTP/1.1 或更高版编译，或不知道下一跳服务器的 HTTP 版，它都【必须】传发该请求，包括该 Expect 头部域。

• 如果代理知道下一跳服务器的版本是 HTTP/1.0 或更低，则它【禁止】传发该请求，并且【必须】以 417（Expectation Failed）状态响应。

• 代理【应该】维护缓存来记录从最近引用的下一跳服务器收到的 HTTP 版本号。

• 代理【禁止】传发 100（Continue）响应，如果请求消息从 HTTP/1.0（或更早）客户端收到，且不包括含有“100-Continue”期望的 Expect 请求头部域。该要求覆盖转发 1xx 响应（见 10.1 节）的通用规则。

8.2.4 服务器提前关闭连接时的客户端行为

如果 HTTP/1.1 客户端发送请求中含有请求体，但不包括含有“100-continue”期望的 Expect 请求头部域，且如果客户端没有直接连接到 HTTP/1.1 原始服务器，且如果客户端发现在从服务器收到任何状态前连接关闭了，则客户端【应该】重试该请求。如果客户端不重试该请求，则它【可以】使用如下的“二分指数后退”算法来确保获得可靠响应：

1、发起新连接到服务器

2、传输请求头

3、初始化变量 R 为估计到服务器的来回时间（如，基于它花在建立连接上的时间），或为常值 5，如果来回时间不可用。

4、计算 T=R*（2**N），这里的 N 是该请求已经重试的次数。

5、等待服务器的错误响应或 T 秒（无论那个先到）。

6、如果没有收到错误响应，在 T 秒后传输请求体。

7、如果客户端发现连接提前关闭，则重复步骤 1，直到请求被接受、收到错误响应、或用户开始不耐烦并终止重试过程。

如果在任何时候收到错误状态，则客户

- 【不该】继续且
- 【应该】关闭该连接，如果它没有发完请求消息。

9 方法定义

下面定义 HTTP/1.1 的公用方法集。尽管能够扩充该方法集，但是不能假定附加方法与独自扩展的客户端和服务器具有相同的语义。

Host 请求头部域（14.23 节）必须伴随所有的 HTTP/1.1 请求。

9.1 安全和等效方法

9.1.1 安全方法

实现者应该清楚，软件拌演用户与 Internet 交互。软件应该小心地允许用户知道他们将作出的任何行为都可能对他们自己或其它人有非预期的效果。

特别是协议已经确定 GET 和 HEAD 方法不应该有非获取行为的效果。这些方法可认为是“安全”的。允许用户代理通过特殊方式描述其它方法，如 POST、PUT 和 DELETE，由此让用户知道可能正在请求一个不安全行为的情况。

当然，不可能保证服务器在执行 GET 请求时不产生副作用的结果；事实上，一些动态资源将其作为一种特性。这里的重要区别是，用户不是请求副作用，因此不能对控制这些副作用负责。

9.1.2 等效方法

方法还有“等效”的属性（除错误和终止问题以外），即多次同样发出单个请求的副作用 N>0 是相同的。方法 GET、HEAD、PUT 和 DELETE 都有该属性。而且，方法 OPTIONS 和 TRACE 不应该有副作用，因此是固定等效的。

然而，几个请求的序列可能是非等效的。甚至如果所有这些方法按其顺序执行是等效的

也可能是非等效的。（如果单次执行全部序列一直产生的结果与全部或部分重新执行该序列的结果相同，则该序列是等效的。）例如，如果其结果取决于同个序列中最后修改的某个值，则该序列是非等效的。

由定义得知，不产生副作用的序列是等效的（倘若没有在相同的资源集上同时执行操作）。

9.2 OPTIONS

OPTIONS 方法表示在由 Request-URI 标识的请求/响应链上关于有效通迅选项信息的请求。该方法允许客户端判断与某个资源相关的选项和/或需求或者服务器的能力，而不需要采用资源行为或发起资源获取。

该方法的响应不能缓存。

如果 OPTIONS 请求包括实体（如由 Content-Length 或 Transfer-Encoding 的存在表示），这时媒体类型必须通过 Content-Type 域表示。尽管本规范没有定义该实体的用法，将来的 HTTP 扩展可能使用 OPTIONS 消息体来更详细地查询服务器的信息。服务器不支持该扩展可以丢弃该请求消息体。

如果 Request-URI 是星号（“*”），OPTIONS 请求通常试图应用于服务器而不是特定的资源。由于服务器的通迅选项一般由资源决定，所以“*”请求只作为“ping”或“no-op”类型的方法有用；它没有任何作用，除了允许客户端测试服务器的能力。例如，可用来测试 HTTP/1.1 代理的一致性（或缺少因素）。

如果 Request-URI 不是星号，OPTIONS 请求只应用于与该资源通迅时的有效选项。

200 响应应该包括任何头部域来表示服务器实现和可应用到该资源的可选特性（如 Allow），可能包括该规范没有定义的扩展。如果有响应消息体，则应该还包括通迅选项的信息。本规范没有定义该消息体的格式，但可能在将来扩展 HTTP 时定义。内容协商可用于选择适当的响应格式。如果不包括响应消息体，则响应必须包括域值为“0”的 Content-Length 域。

Max-Forwards 请求头部域可能用于请求链中定位特定代理。当代理收到关于允许请求转发的 absoluteURI 的 OPTIONS 请求时，代理必须检查 Max-Forwards 域。如果 Max-Forwards 域值为 0（“0”），则代理不能转发该消息；取而代之，代理应该以它自己的通迅选项来响应。如果 Max-Forwards 域值是大于 0 的整数，代理在转发该请求时必须将域值减一。如果请求中不存在 Max-Forwards 域，则转发的请求中不能包括 Max-Forwards 域。

9.3 GET

GET 方法即获取由 Request-URI 标识的任何信息（以实体的形式）。如果 Request-URI 引用某个数据处理过程，则应该以它产生的数据作为在响应中的实体，而不是该过程的源代码文本，除非该过程碰巧输出该文本。

如果请求消息包括 If-Modified-Since、If-Unmodified-Since、If-Match、If-None-Match 或者 If-Range 头部域，则 GET 方法的语义变为“条件 GET”。条件 GET 方法请求只传输在条件头部域描述情形下的实体。条件 GET 方法试图通过允许刷新缓存的实体而不需要多次请求或传输客户端已经拥有的数据来减少非必要的网络使用。

如果请求消息包括 Range 头部域，则 GET 方法的语义变为“局部 GET”。局部 GET 请求只需传输实体的某部分，如 14.35 节中描述的。局部 GET 方法试图通过允许部分获取实体来完成而不需要传输客户端已经拥有的数据来减少非必要的网络使用。

当且仅当满足 13 节中描述的 HTTP 缓存要求时，GET 请求的响应才是可缓存的。

用于表单时的安全考虑见 15.1.3 节。

9.4 HEAD

除了服务器不能在响应中返回消息体，HEAD 方法与 GET 相同。HEAD 请求的响应中的 HTTP 头部中包含的元信息应该与 GET 请求发送的响应中的信息相同。该方法可用来获取请求暗示实体的元信息，而不需要传输实体本身。该方法常用来测试超文本链接的有效性、可用性和最近的修改。

当响应中包含的信息可用于更新先前从该资源缓存的实体时，HEAD 请求的响应可能是可缓冲的。如果新的域值表明该缓冲的实体与当前实体不同（如可通过 Content-Length、Content-MD5、ETag 或 Last-Modified 的区别来表示），这时缓冲服务器必须将该缓存实体作为过期的。

9.5 POST

POST 方法用来请求原始服务器接受请求中封装的实体作为从属于请求行中的 Request-URI 标识的副属。POST 设计允许完成下列功能的统一方法：

* 注解存在资源；
* 上传消息到论坛、新闻组或相似的讨论组；
* 向数据处理过程提供数据块，如递交表单的结果；
* 通过追加操作来扩展数据库。

POST 方法执行的实际功能由服务器决定，且通常取决于 Request-URI。上传的实体从属于该 URI，通过文件从属于包含它的目录，新的论文从属于它上传的新闻组，或记录从属于数据库的方式。

POST 方法执行的行为可能不导致通过 URI 能够标识的某个资源。在这种情况下，200（OK）或 204（No Content）都是适合的响应状态。这取决于描述结果的响应是否包括实体。

如果原始服务器创建了资源，响应应该是 201（Created），且包含描述请求状态的实体，

和新资源的引用，和 Location 头部（见 14.30 节）。

该方法的响应不能缓存，除非响应包括适当的 Cache-Control 或 Expires 头部域。然而，303（See Other）响应能够用来引导用户代理获取可缓存的资源。

POST 请求必须服从 8.2 节中陈述的消息传输需求。

安全考虑见 15.1.3 节。

9.6 PUT

PUT 方法请求以提供的 Request-URI 存储封装的实体。如果 Request-URI 引用已经存在的资源，该封装实体应该被认作原始服务器存储的修改版本。如果 Request-URI 没有指向已存在的资源，且该 URI 可以被请求的用户代理定义为新的资源，则原始服务器可以用该 URI 创建资源。如果创建了新的资源，则原始服务器必须通过 201（Created）响应提示用户代理。如果修改了已存在的资源，则应该发送 200（OK）或 204（No Content）响应代码来表示成功完成了请求。如果不能按 Request-URI 创建或修改资源，则应该给出适当的错误响应以反映出问题的性质。实体的接受方不能乎略任何不理解或没有实现的 Content-*（如 Content-Range）头部，在这种情况下必须返回 501（Not Implemented）响应。

如果请求通过缓冲服务器且 Request-URI 标识出一个或多个缓冲的实体，则应该认为这些实体过期了。该方法的响应不可缓存。

POST 和 PUT 请求间的基本区别反映在 Request-URI 的不同意义。POST 请求中 URI 标识的资源将处理封装的实体。该资源可能是数据接收过程、其它协议的网关或接受注解的独立实体。与此对应，PUT 请求中的 URI 标识请求封装的实体——用户代理知道该 URI 是目标且服务器不能试图将该请求应用到其它资源上。如果服务器希望该请求应用到不同的 URI 上，则它必须发送 301（Moved Permanently）请求；这时客户代理可以自己决定是否要重定向该请求。

可以用许多不同的 URI 标识同个资源。例如，一篇文章可以有标识为“当前版本”的 URI，它独立于标识每个特别版本的 URI。在这种情况下，使用通用 URI 的 PUT 请求可能造成原始服务器定义的一些不同 URI 的结果。

HTTP/1.1 没有定义 PUT 方法如何影响原始服务器的状态。

PUT 请求必须服从 8.2 节中陈述的消息传输需求。

除了其它特殊实体头部的规定，PUT 请求中的实体头部应该应用到 PUT 创建或修改的资源上。

9.7 DELETE

DELETE 方法请求原始服务器删除 Request-URI 标识的资源。原始服务器可在人为干涉

下（或其它意思）屏闭该方法。客户端不能确保该操作已经提交，即使原始服务器发出的状态码表明动作已经成功完成也如此。然而，在给出响应的时候，服务器不应该表示成功，除非它试图删除该资源或将它移动到不可访问的位置。

如果响应包含描述状态的实体，成功响应应该是 200（OK）。如果动作没有实施，则是 202（Accepted）。如果动作已经实施但响应不包含实体，则是 204（No Content）。

如果请求通过缓冲服务器，且 Request-URI 标识一个或多个当前缓存的实体，则应该认为这些实体已经过期。该方法的响应不可缓存。

9.8 TRACE

TRACE 方法用于引起远程的，该请求消息的应用层回射。请求的最终接收者应该反射 200（OK）响应，并以该消息作为客户端回收消息的实体。最终接收者是原始服务器或第一个收到请求中的 Max-Forwards 值为 0（0）的（见 14.31 节）代理或网关。TRACE 请求不能包括实体。

TRACE 允许客户端看见请求链上的另一端收到了什么，然后使用该数据作为测试或诊断信息。Via 头部域的值（14.45 节）有特殊作用，将它作为请求链路径。使用 Max-Forwards 头部域允许客户端限制请求链的长度，这对于测试无限循环转发消息的代理链非常有用。

如请求有效，则响应应该在实体中包含整个请求消息，设置 Content-Type 为“message/http”。该方法的响应不能缓存。

9.9 CONNECT

本规范保留 CONNECT 方法名。该方法用于代理，使之能够动态切换隧道（例如 SSL 隧道[44]）。

10 状态码定义

下面描述的每个状态码。包括它能够跟随的方法和响应中需要的任何元信息。

10.1 信息性 1xx

本类状态码表示临时的响应，由单独的状态行和可选的头部组成，终止于空行。本类状态码没有必须的头部。由于 HTTP/1.0 没有定义任何 1xx 状态码，因此服务器禁止向 HTTP/1.0 客户端发送 1xx 响应，除非在试验时。

客户端必须准备好接收先于一般响应的一个或多个 1xx 状态响应，即使在客户端不期望 100（Continue）状态消息时。非期望的 1xx 状态响应可能被用户代理乎略。

代理必须转发 1xx 响应，除非代理与它的客户端之间的连接关闭了，或者除非代理自己请求产生 1xx 响应。（例如，当代理转发请求时加入了“Expect: 100-continue”域时，这时它

就不需转发相应的 100（Continue）响应。）

10.1.1 100 Continue

客户端应该继续它的请求。该间歇响应用于提醒客户端服务器已经接收和接受请求的开始部分。客户端应该继续发送请求的剩余部分，或者如果请求已经发送完了，就乎略该响应。服务器在请求完成后必须发送最终响应。该状态码用法和处理的详细讨论见 8.2.3 节。

10.1.2 101 Switching Protocols

服务器理解并愿意答应客户端的请求。通过使用 Upgrade 消息头部域（14.42 节），在该连接上改变应用层协议。服务器将在 101 响应的终止空行后立即切换到响应的 Upgrade 头部域中定义的协议。

只有在有优势时才应该切换协议。例如，切换到 HTTP 的新版本比老版本有优势，切换到实时、同步协议在递交使用这种特性的资源时可能有优势。

10.2 成功 2xx

这类状态码表示客户端的请求成功被接收、理解和接受了。

10.2.1 200 OK

请求已经成功。该响应返回的信息取决于请求中使用的方法，例如：

GET 与所请求资源相对应的实体将在响应中发送；
HEAD 与所请求资源相对应的实体头部将在响应中发送，而没有消息体；
POST 描述或包含行为结果的实体；
TRACE 包含终点服务器收到的请求消息的实体。

10.2.2 201 Created

请求全部成功，且创建了新资源。可通过响应实体中返回的 URI 引用新创建的资源，使用 Location 头部域中给出的最特别的 URI。该响应应该包含资源特性和位置清单的实体，从而用户或用户代理能够从中选择最适当的一个。实体格式通过在 Content-Type 头部域中的媒体类型指定。原始服务器必须在返回 201 状态码之前创建资源。如果该行为不能立即实施，服务器应该代之以 202（Accepted）响应。

201 响应可能包含 ETag 响应头部域，指示刚创建的请求变量的实体标签的当前值，见 14.19 节。

10.2.3 202 Accepted

请求已经接受处理，但是处理还没有完成。当处理实际发生时，请求最后可能会或不会

被执行，如同它可能被拒绝一样。没有设置在这样的异步操作后重发状态码的功能。

202 响应是故意不承担该义务的。它的目的是允许服务器可接收其它过程的请求（可能是每天只运行一次的面向批处理的过程），而不需要用户便一直连接到服务器，直到处理完成。该响应返回的实体应该包括请求当前状态的指示和状态显示器或一些估计的指针，使用户能够预料请求已经完成。

10.2.4 203 Non-Authoritative Information

实体头部中返回的元信息不是在原始服务器有效的确定集合，而是从本地或第三方拷贝收集的。现在的集合可能是原始版本的子集或超集。例如，包括关于资源的本地注释住息可能导致原始处理器知道元信息的超集。使用该响应码没有要求，且只有响应是 200（OK）的其它状态时才是适当的。

10.2.5 204 No Content

服务器已经完成请求，但不需要返回实体，且可能希望返回更新的元信息。响应可能包括新的或更新的元信息，通过实体头部的形式。如果存在这些头部，则应该与所请求变量相关。

如果客户是用户代理，则它不能改变引起发送请求的文档视图。该响应主要用于允许输入行为发生，而不会引起用户代理的活动文档视图的改变，即使任何机新的或更新的元信息应该应用到当前在用户代理活动视图中的文档。

204 响应禁止包括消息体，且始终止于头部域后的首个空行。

10.2.6 205 Reset Content

服务器已经完成请求且用户代理应该复位引起请求发送的文档视图。该响应主要用于允许行为输入通过用户输入发生，接着清除给出输入的表单，以使用户能够轻松地开始另一次输入行为。响应禁止包括实体。

10.2.7 206 Partial Content

服务器已经完成局部资源的 GET 请求。请求必须包括 Range 头部域（14.35 节）来指示期望的范围，且可能包含 If-Range 头部域（14.27 节）来作为请求条件。

响应必须包含如下的头部域：

•Content-Range 头部域（14.16 节）指示包含在响应中的范围，或值为 multipart/byteranges 的 Content-Type 且各部分都包含 Content-Range 域。如果响应中存在 Content-Length 头部域，则它的值必须与在消息体中传输的实际 8 位组数相符。

• Date

• ETag、和/或 Content-Location（当相同请求的 200 响应中已经发送了该头部）。

• Expires、Cache-Control、和/或 Vary（当其域值可能与先前的相同变量的任何响应不同）

如果 206 响应是由 If-Range 请求使用强缓冲验证（见 13.3.3 节）所引起的，则该响应不该包含其它实体头部。如果响应是由 If-Range 请求使用弱缓冲验证引起的，则响应禁止包含其它实体头部；这样就可阻止缓存实体与更新头部间的不一致。换句话说，即响应必须包括相同请求的 200（OK）响应中会返回的所有实体头部。

如果 ETag 或 Last-Modified 头部不能精确匹配，则缓冲服务器禁止将 206 响应与其它以前的缓存内容组合在一起，见 13.5.4。

不支持 Range 和 Content-Range 头部的缓冲服务器禁止缓存 206（Partial）响应。

10.3 重定向 3xx

这类状态码指示，需要用户代理采取更进一步的行为来完成请求。当且仅当第二次请求所使用的方法是 GET 或 HEAD 时，所需要行为可能被用户代理在不通知用户的情况下所提交。客户端应该检测无限循环重定向，因为这类循环每次重定向都会产生网络流量。

注意：本规范的以前版建议最多 5 次重定向。内容开发者应该明白，可能有实施这种固定限制的客户端存在。

10.3.1 300 Multiple Choices

所请求的资源符合表述集合中的任何一个，每个都有它自己的特殊位置。代理驱动的协商信息（12 节）提供给用户（或用户代理）来选择喜欢的表述，并重定向请求到它的位置。

除非是 HEAD 请求，否则响应应该包括含有资源特性和位置清单的实体，以便用户或用户代理能够选择最适当的一个。该实体格式由 Content-Type 头部域中给出的媒体类型规定。取决于格式和用户代理的能力，作出最适当的选择可能会自动执行。然而，本规范没有定义任何这类自动选择算法。

如果服务器对表述有喜欢的选择，它应该在 Location 域中包括该表述的指定 URI；用户代理可能使用 Location 域值来作出自动重定向。该响应可缓存，除非有特别指示。

10.3.2 301 Moved Permanently

所请求的资源已经指定到一个新的永久 URI，且将来任何对该资源的引用都应该使用所返回的 URI 之一。如果可能的话，客户端有修改链接的能力应该自动重链接引用该 Request-URI 到服务器所返回的新引用的一个或多个。该响应可缓存，除非有特别指示。

新的永久 URI 应该在响应中的 Location 域中给出。除非请求方法是 HEAD，否则响应实体应该包含链接到新的 URI 的简短的超文本说明。

如果非 GET 或 HEAD 请求收到了 301 状态码响应，则用户代理禁止自动重定向请求，除非它得到了用户的确认。因为这可能改变请求发生的条件。

当自动重定向收到 301 状态码的 POST 请求时，一些现有的 HTTP/1.0 用户代理将会错误地将其变为 GET 请求。

10.3.3 302 Found

所请求的资源临时存在于不同的 URI。由于重定向随时可会改变，客户端应该继续在将来的请求中使用该 Request-URI。只有通过 Cache-Control 或 Expires 头部域指示下，该响应才是可缓存的。

临时的 URI 应该在响应中的 Location 域中给出。除非请求方法是 HEAD，否则响应实体中应该包含链接到新 URI 的简短超文本说明。

如果非 GET 或 HEAD 请求收到了 302 状态码的响应，则用户代理禁止自动重定向请求，除非它能够获得用户的确认。因为这可能导致发出请求的条件改变。

注意：RFC 1945 和 RFC 2068 指出，不允许客户端在重定向请求时改变方法。然而，大多数现存的用户代理实现都将 302 作为 303 响应，且执行到 Location 域值的 GET，而不管原始请求方法。303 和 307 状态码已经加到服务器，希望客户端作出所期望的有明显区别的不同类的返应。

10.3.4 303 See Other

请求的响应可以在不同的 URI 中发现，且应该使用 GET 方法到该资源来获取它。存在该方法中要是用于允许由 POST 激活的脚本输出重定向用户代理到所选择的资源。新的 URI 不是原始请求资源的代替引用。禁止缓存 303 响应，但到第二个（重定向的）请求的响应可能可缓存。

不同 URI 应该在响应中通过 Location 域给出。除非请求方法是 HEAD，响应实体应该链接到新的 URI 的简短的超文本说明。

注意：许多先于 HTTP/1.1 的用户便不理解 303 状态。当与这种客户端交互时要当心，可使用 302 状态码代替，因为大多数用户代理将这里描述的 303 作为 302 来处理。

10.3.5 304 Not Modified

如果客户端执行条件 GET 请求，且允许访问，但文档没有变化，服务器应该用该响应码响应。304 响应禁止包含消息体，因此它始终止于头部域后的首个空行。

该响应必须包括如下头部域：

• Date，除非如 14.18.1 节中描述的需要省略

如果无时钟的原始服务器服从该规则，且代理和客户端添加它们自己的 Date 到任何所收到的没有 Date 的响应中（如已经在[RFC 2068]，14.19 节中指定），缓存将会正确操作。

• ETag、和/或 Content-Location（如果头部已经在相同请求的 200 响应中发送过）

• Expires、Cache-Control、和/或 Vary（如果域值与任何以前相同变量的响应发送过的不同）

如果条件 GET 使用强制缓冲验证（见 13.3.3 节），则响应不该包括其它实体头部。否则（如，条件使用弱验证），响应禁止包括其它实体头部；这会阻止缓存实体与更新头部之间的不一致。

如果 304 响应指示实体没有缓存，这时缓存必须不理会该响应，并无条件重复该请求。

如果缓冲服务器使用所收到的 304 响应来更新缓存条目，则缓冲服务器必须更新反映响应中给出的任何新域值的条目。

10.3.6 305 Use Proxy

所请求的资源必须通过 Location 域中给出的代理来访问。Location 域给出代理的 URI。期户接收方通过代理重复这次请求。305 响应必须只由原始服务器产生。

注意：RFC 2068 没有说清，305 用于重定向单次请求，且只由原始服务器生成。没注意到这些限制已有明显的安全性后果。

10.3.7 306 (Unused)

306 状态码在前版规范中使用，现在没用了，应保留该代码。

10.3.8 307 Temporary Redirect

所请求资源临时存在于不同的 URI。由于重定向可能随时会改变，所以客户端应该继续在以后的请求中使用 Request-URI。只有通过 Cache-Control 或 Expires 头部域指示，该响应才是可缓存的。

临时 URI 应该通过响应中的 Location 域给出。除非请求方法是 HEAD，否则响应实体应该包含链接到新 URI 的简短超文本说明，由于许多先于 HTTP/1.1 的用户代理不理解 307 状态。因此，该说明应该包含必要的信息来帮助用户重复原始请求到新的 URI。

如果 307 状态码响应不是在 GET 或 HEAD 请求时收到，则用户代理禁止自动重定向请

求，除非它能够得到用户的确认，因为这可能会改变发生请求的条件。

10.4 客户端错误 4xx

4xx 类的状态码用于看起来客户端有错误的情况下。除了相应的 HEAD 请求，服务器应该包括解释错误状况的实体，不管是临时还是永久的条件下。该状态码可应用到所有请求方法。

用户代理应该显示实体给用户。如果客户端正在发送数据，使用 TCP 实现的服务器应该小心确保，在服务器关闭输入连接前，客户已经确认其所收到的包含响应的包。如果客户端在关闭后继续发送数据到服务器，则服务器的 TCP 协议栈将发送复位包到客户端，这将会引起在 HTTP 应用读和解析之前清除客户端没有确认的输入缓冲。

10.4.1 400 Bad Request

服务器不能理解请求，由于畸形的语法。客户端不应该重复没经修改的请求。

10.4.2 401 Unauthorized

请求需要用户认证。响应必须包括 WWW-authenticate 头部域（14.47 节），其中包含对所请求资源的适用的要求。客户端可以使用合适的 Authorization 头部域（14.8 节）来重复该请求。如果请求已经包括 Autoorization 证书，这时 401 响应表示该证书的认证被拒绝了。如果 401 响应包含与先前的响应相同的要求，且用户代理已经尝试认证了至少一次，这时应该将响应中给出的实体提示给用户，因为该实体可能包含有关的诊断信息。HTTP 访问认证的解释见“HTTP Authentication: Basic and Digest Access Authentication”[43]。

10.4.3 402 Payment Required

该代码留作将来使用。

10.4.4 403 Forbidden

服务器理解请求，但拒绝完成它。认证也没用，请求不该重复。如果请求方法不是 HEAD，且服务器希望公开为什么没有完成请求，则它应该在实体上描述拒绝的原因。如果服务器不希望向客户端给出该信息，则可用状态码 404（Not Found）代替。

10.4.5 404 Not Found

服务器不能发现匹配 Request-URI 的任何东西。没有表明这种状况是临时的还是永久的。如果服务器通过一些内部配置机制，知道旧资源永久不可用了，且没有转发的地址，则应该使用 410（Gone）状态码。本状态码通常用于服务器不希望透露为什么拒绝请求，或没有其它响应可用时。

10.4.6 405 Method Not Allowed

Request-Line 中指定的方法不允许用到由 Request-URI 标识的资源。该响应必须包括 Allow 头部，并在其中包含对所请求资源有效的方法清单。

10.4.7 406 Not Acceptable

请求所标识的资源的内容特性不被请求中所发送的接受头部所接受，所以不能生成响应。除非是 HEAD 请求，响应应该包括实体，其中包含有效的实体特性和位置，以便用户或用户代理能够选择最适当的一个。实体格式通过 Content-Type 头部域中给出的媒体类型指定。取决于该格式和用户代理的能力，作出最适当选择可以自动执行。然而，本规范没有定义任何标准的这种自动选择算法。

注意：HTTP/1.1 服务器允许返回不被请示发送的接受头部所接受的响应。在某种条件下，这可能甚至要好于发送 406 响应。

鼓励用户代理检测接收的响应头部，以判断是否可接受。如果响应不可接受，用户代理应该临时停止接收更多数据，并询问用户作出更进一步的选择。

10.4.8 407 Proxy Authentication Required

该代码与 401（Unauthorized）类似，但指示客户端必须首先向代理认证它自己。

代理必须返回 Proxy-Authenticate 头部域（14.33 节），在其中包含为所请求资源对代理有效的需求。客户端可能使用适当的 Proxy-Authoriztion 头部域（14.34 节）来重复请求。HTTP 访问认证解释见“HTTP Authentication: Basic and Digest Access Authentication”[43]。

10.4.9 408 Request Timeout

在服务器准备接收等待的时间内，客户端没有产生任何请求。客户端可能在以后的任何时候重复该没经修改的请求。

10.4.10 409 Conflict

请求没有完成，因为对资源现有状态的冲突。该代码只在如下情况下允许，即希望用户能够解决该冲突并重新提交该请求。响应体应该包括足够的信息为用户识别冲突源。典型例子是，响应实体可能包括足够信息为用户或用户代理解决问题；然而，这可能会不可能且没必要。

冲突大多发生在 PUT 请求的响应中。例如，如果使用版本控制，且 PUT 包括的实体改变资源，这会与早先（第三方）请求作出的相冲突，服务器可能会使用 409 响应表明它不能完成请求。在这种情况下，响应实体将可能以响应 Content-Type 定义的格式包含这两个版本的区别清单。

10.4.11 410 Gone

所请求资源不在服务器上有效，且不知道转发地址。该状态期户是永久的。客户端有链接修改能力应该在用户同意的情况下删除到该 Request-URI 的引用。如果服务器不知道，或没有判断机制，不管状态是否是永久的，都应该使用状态码 404（Not Found）来代替。该响应可缓存，除非另行指明。

410 响应主要用于辅助 WEB 维护任务，通过提示接收者，源故意不可用，且服务器拥有都希望删除到该资源的远程链接。

这种事件通常是限时的，升级服务和属于个人的资源在服务器站点没有使用。没必要标识所有永久不可用资源为“消失”，或保持标志任何时长——这留给服务器拥有都考虑。

10.4.12 411 Length Required

服务器拒绝接受没有定义 Content-Length 的请求。客户端可以通过添加有效的 Content-Length 头部域并包含请求消息中的消息体的长度来重复该请求。

10.4.13 412 Precondition Failed

在一个或多个请求头部域给出的前提在服务器上测试评估失败。该响应代码允许客户端在当前资源元信息（头部域数据）中放置前提，这将阻止所请求方法应用到不需要的资源上。

10.4.14 413 Request Entity Too Large

服务器拒绝处理请求，因为请求实体长于服务器愿意或能够处理的长度。服务器可以关闭连接来阻止客户端继续请求。如果状态是临时的，服务器应该包括 Retry-After 头部域来表示这是临时的，且在该时间后客户端可以重试。

10.4.15 414 Request-URI Too Long

服务器拒绝服务请求，因为 Request-URI 长于服务器愿意解析的长度。该罕见的状态只好像发生在，当客户端使用长请求信息不正确转换 POST 请求为 GET 请示时，当客户端已经降入 URI 重定向“黑洞”时（例如，重定向 URI 前缀指向它自己的后缀），或当服务器被客户端试图利用在某些服务器中存在的安全漏洞（使用固定长度缓冲来读取或控制 Request-URI）发起攻击时。

10.4.16 415 Unsupported Media Type

服务器拒绝服务请求，因为请求的实体是请求方法所请求的资源所不支持的格式。

10.4.17 416 Requested Range Not Satisfiable

服务器应该返回该状态码的响应，如果请求包括 Range 请求头部域（14.35 节），且该域中的范围指定值没有覆盖任何所选择资源的当前长度，且请求没有包括 If-Range 请求头部域

（对于 byte-ranges，这意未着所有 byte-range-spec 值的 first-byte-pos 都大于所选资源的当前长度。）

当 byte-range 请求返回该状态码时，响应应该包括 Content-Range 实体实部域，并指定选择资源的当前长度（见 14.16 节）。该响应禁止使用 multipart/byteranges 内容类型。

10.4.18 417 Expectation Failed

Expect 请求头部域（见 14.20 节）中给出的期望不能满足服务器，或者，如果服务器是代理，服务器已经明确证明请求不能满足下一跳的服务器。

10.5 服务器错误 5xx

由数字“5”打头的响应状态码表示服务器已经明显处于错误的状况下或没有能力执行请求。除了相应的 HEAD 请求，服务器应该包括解释错误状态的实体，且不管理是临时或永久状态。用户代理应该显示实体中包括的任何东西给用户。这些响应码可用于任何请求方法。

10.5.1 500 Internal Server Error

服务器发生非预期状况，阻止它完成请求。

10.5.2 501 Not Implemented

服务器不提供完成请求所需的功能。这是适当的响应，当服务器不认识请求方法或没有能力在任何资源上支持它。

10.5.3 502 Bad Gateway

当作为网关或代理时，服务器从它靠近的上游服务器收到试图完成请求的无效响应。

10.5.4 503 Service Unavailable

服务器当前不能处理请求，因为临时性的负载过重或服务器维护中。

其涵义是临时性状况将会在一些时延后缓和。如果可知，时延长度可以能过 Retry-After 头部指示。如果没有给出 Retry-After，客户端应该如 500 响应一样处理响应。

注意：存在 503 状态码并不意未着服务器在开始负载过重时必须使用它。一些服务器可能希望简单地拒绝连接请求。

10.5.5 504 Gateway Timeout

当作为网关或代理时，服务器试图完成请求时没有从 URI（例如，HTTP、FTP 或 LDAP）

指定我上流服务器或一些其它所需来访问的辅助服务器（如，DNS）收到定时响应。

注意：实现者需注意，一些已布置的代理在 DNS 查询超时后已知会返回 400 或 500。

10.5.6 505 HTTP Version Not Supported

服务器不支持，或拒绝支持，请求消息中使用的 HTTP 协议版本。服务器指示它不能或不愿意使用与客户端相同的主版本来完成请求，如 3.1 节中所述，而是返回该错误消息。响应中应该包括实体，来描述为什么不支持该版本和服务器还支持其它什么协议。

11 访问认证

HTTP 提供几个可选的需求响应认证机制，可被服务器用来要求客户请求且由客户提供认证住处。访问认证的一般框架，及“basic”和“digest”认证规范，在“HTTP Authentication: Basic and Digest Access Authentication”[43]中规定。本规范适应那个规范定义的“challenge”和“credentials”。

12 内容协商

大多数 HTTP 响应包括实体，其中包含由人类用户解释的信息。自然而然，对应与请求提供用户“最有用”的实体是最理想的。不幸的是，对于服务器和缓存服务器而言，不是所有的用户对什么是“最佳的”有相同的取向，且非所有用户代理都相同的能力来描绘所有的实体类型。基于该原因，HTTP 为“内容协商”，——当存在几个有效的表示时为给定的响应选择最佳表述的过程，准备了几套机制。

注意：这不是称为“格式协商”，因为可选的表述可能有相同的媒体格式，但使用不同的该类型的特性，用不同的语言，等。

任何包含实体的响应可能成为协商的主题，包括错误响应。在 HTTP 可能存在两种类型的内容协商。服务器驱动和代理驱动的协商。这两种类型的协商是互不相关的，因此可能单独或组合使用。

作为透明协商，一种使用组合方法的情况发生在，当缓存服务器使用代理驱动协商由原始服务器提供的住处，以便为后序请求提供服务器驱动协商时。

12.1 服务器驱动协商

如果关于响应的最佳表述的选择由位于服务器的算法作出，它就称为服务器驱动协商。选择是基于响应的可用表述（能够改变的范围；如，语言，内容编码，等。）和请求消息中的特定头部域的内容，或者与请求相关的其它信息（如客户端的网络地址）。

在从大量的可用表述中选择的算法对于用户代理来说描述非常困难时，或者当服务器期望发送它的“最佳预测”连同首次响应到客户端时（如果“最佳预测”对用户够好了，希望避免后序请求的来回延迟），服务器驱动协商是有优势的。为了改善服务器的预测，用户代

理可能包括请求头部域（Accept，Accept-Language，Accept-Encoding，等）来描述它对该响应的选择。

服务器驱动协商有如下劣势：

1、对于任何给定用户而言，服务器精确判断什么可能是“最佳的”是不可能的，因为这将需要关于服务代理的能力及打算如何使用响应（例如，用户是希望将它显示在屏幕上呢还是打印在纸上呢？）的完全的知识。

2、要求用户在其每次请求中都包含其能力可能即是非常低效（只占响应小比例的所发信息将多次表述），而且又潜在侵范了用户的隐私。

3、它将原始服务器的实现和生成请求响应的算法复杂化。

4、它将限制公共缓存服务器使用相同的响应为多数用户请求服务的能力。

HTTP/1.1 包括如下的请求头部域来通过用户代理能力的描述和用户选择来允许服务器驱动协商：Accept（14.1 节），Accept-Charset（14.2 节，Accept-Encoding（14.3 节），Accept-Language（14.4 节）和 User-Agent（14.43）节。然而，原始服务器没有局限在这些范围内，且可能会基于请求的任何样子来修改响应，包括请求头部域以外或这里没有定义的扩展头部域内的信息。

Vary 头部域可用来表示服务器用来选择受服务器驱动协商响应的表述的参数。Vary 头部域的在缓存服务器中的用法见 13.6 节。服务器如何使用 Vary 头部域见 14.44 节。

12.2 代理驱动协商

使用代理驱动协商，用户代理在收到原始服务器的初始响应后执行对响应的最佳表述的选择。选择基于初始响应中包括的头部域或实体的可用表述，每个表述由它自己的 URI 标识。从大量的表述中选择可能会自动执行（如果用户有能力这么作）或由用户从生成的菜单（可能是超文本）中手动选择。

当响应在通用范围（例如类型、语言或编码）内可变时，当原始服务器从检验请求中判断服务代理的能力时，和通常法公共缓存服务器用于分散服务器负载和减少网络流量时，代理驱动协商是有优势的。

代理驱动遭受需要第二次请求来获得最的备用表述的劣势。第二次请求只在使用缓存时有效。而且，本规范没有定义任何支持自动选择的机制，尽管它也没有阻止表发任何这类机制来作为扩展和用于 HTTP/1.1 内部。

当服务器使用服务器驱动协商非意愿或不能提供可变响应时，HTTP/1.1 定义 300（Multiple Choices）和 406（Not Accpetable）状态码来允许代理驱动协商。

12.3 透明协商

透明协商组合了服务器驱动和代理驱动协商。当向缓存服务器提供响应的可用表述清单的表格（作为代理驱动协商）且缓存服务器完全了解变更的范围，这时，缓存服务器开始能够执行服务器驱协商该资源上的后序请求到原始服务器的行为。

透明协商的优势是，分布要么需要原始服务器做的协商工作，而且当缓存服务器能够正确推测正确响应时还删除了代理驱动协商的第二次请求的延迟。

本规范没有定义任何关于透明协商的机制，尽管它也没有阻止开发任何这类机制作为用于 HTTP/1.1 内部的扩展。

13 HTTP 缓存

HTTP 一般用在分布式信息系统，其性能可通过使用响应缓存来改善。HTTP 协议包括一些元素用于使缓存尽可能有效。因此为些元素与协议的其它方面有密切关联，而且它们互相交互，所以独立于方法、头部、响应码等的详细描述来描述 HTTP 的基本缓存设计是有用的。缓存会没用，如果它不能显著地改善性能。HTTP/1.1 中的缓存的目标是除去许多情况下对发送请求的需求，和除去在其它许多情况下发送完整响应的需求。前者减少许多操作所需的大量网络来回奔波；我们为了该目的使用“截止”机制（见 13.2 节）。后者减少网络带宽需求；我们为了该目的使用“证实”机制（见 13.3 节）。

性能、可用和离线操作的需求需要我们能够放宽语义透明度的目标。HTTP/1.1 协议允许原始服务器、缓存服务器和客户端按需要显著减少透明度。然而，因为非透明操作可能拒绝非专家用户，且可能与已知的服务器应用（如定单生意）不兼容，因此协议需要放宽透明度：

- 客户端或原始服务器只能通过明确的协议层请求来放宽
- 缓存服务器或客户端只能使用明确的警告给终端用户来放宽

因此，HTTP/1.1 协议提供这些重要的元素：

1、为所有需要的部分提供完整语义透明度的协议特性。

2、允许原始服务器或用户代理明确请求和控制非透明操作的协议特性。

3、允许缓存服务器给没有维护请求语义透明度近似性的响应附上警告的协议特性。

一条基本规则是客户端必须能够检测任何潜在对语义透明度的放宽。

注意：服务器、缓存服务器或客户端实现都可能要面对本规范没有明确讨论的设计决择。如果决择可能影响语义透明度，则实现者需要在维护透明度的一方报错，除非细心和完整的分析显示破坏透明度有明显的好处。

13.1.1 缓存正确性

正确的缓存必须反映缓存拥有的请求的最新响应。该响应对该请求是合适的（见 13.2.5、13.2.6 和 13.12 节），且满足如下条件：

1、通过等效检查，即原始服务器更新的响应与原来已经返回的等效（13.3 节）；

2、它是“足够更新”的（见 13.2 节）。在默认情况下，即它需要满足客户端、原始服务器和缓存服务器最低限度的更新度要求（见 14.9 节）；如果原始服务器如此规定，则它只是原始服务器的更新度需求。

如果所存储的响应不是“足够更新”的，不满足客户端和原始服务器的最高限度的更新度需求，在仔细考虑情况下，缓存服务器可能依然以适当的警告头部来返回响应（见 13.1.5 和 14.46 节），除非该响应被禁止（例如，通过缓存指令“非存储”，或通过“非缓存”缓存请求指令；见 14.9 节）。

3、它是适当的 304（Not Modified）、305（Proxy Redirect）或错误（4xx 或 5xx）响应消息。

如果缓存服务器不能与原始服务器相交流，这时如果响应能够通过缓存正确地服务，则正确的缓存服务器应该如上响应；如果不这么做，则它必须返回错误或警告表明存在通迅失败。

如果缓存服务器收到通常需要转发到作出请求的客户端的响应（是完整响应或 304（Not Modifed）响应），且收到的响应没有更新，则缓存服务器应该将其转发到请求客户端而不添加新 Warning（但也不删除任何已存在的 Warning 头部）。缓存服务器不应该只是因为响应在传输过程中过期了而尝试重新证实响应；这可能导致无限循环。用户代理收到不含 Warning 的过期响应可能显示警告指示给用户。

13.1.2 警告

不管缓存服务器返回的响应不是第一手的或没有“足够更新”（类似于 13.1.1 节中的条件 2），它都必须附加警告来达到效果，通过使用 Warning 通用头部。Warning 头部和当前定义的警告在 14.46 节中描述。该警告允许客户端采取适当的行为。

警告可能用于其它目的，缓存相关或其它。使用警告而不是错误状态码将这些响应与真正的失败相区分开来。

警告指定 3 位警告码。首位数字表示警告是否必须或禁止在成功重新证实后从存储缓存表中删除：

1xx 警告描述响应的更新度或重新证实状态，因此必须在成功重新证实后删除。1xx 警告码可能只由缓存服务器在证实缓存条目时产生。它禁止由客户端产生。

2xx 警告描述实体或实体头部的一些方面，它没有被重新证实所更正（例如，丢失实体原缩算法），且在成功证实后禁止删除。

关于这些代码自身的定义见 14.46 节。

HTTP/1.0 的缓存服务器将缓存响应中的所有 Warning，不删除第一类中的任何一条。响应中的警告通过 HTTP/1.0 缓存服务器并挟带附加的警告日期域，它阻止 HTTP/1.1 接收者认为是错误缓存的 Warning。

警告还捎带警告文字。该文字可能是任何适当的自然语言（可能基于客户端的 Accept 头部），且包括选项表明使用何种字符集。

多个警告可能附加到响应中（通过原始服务器或缓存服务器），包括多个有相同代码的警告。例如，服务器可能提供英语或巴斯克语文字的相同警告。

当多个警告附加到响应中时，显示所有的给用户是不现实和没道理的。这版 HTTP 没有规定严格的优先级规则来判断显示哪个警告和以什么顺序，但有些启发式的建议。

13.1.3 缓存控制机制

HTTP/1.1 中的基础缓存机制是隐式的缓存指令。在一些情况下，服务器或客户端可能需要提供隐式的指令给 HTTP 缓存服务器。我们为此目的使用 Cache-Control 头部。

Cache-Control 头部允许客户端或服务器在请求与响应中传输各类指令。这些指令一般覆盖缺省的缓存算法。作为通用规则，如果头部值间有任何明显的冲突，则应用最严格的解释（即，最像维护语义透明度的一个）。然而，在一些情况下，缓存控制指令的明确规定削弱了语义透明的近似度（例如，“max-stale”或“public”）。

缓存控制指令在 14.9 节中详细描述。

13.1.4 明确的用户代理警告

许多用户代理允许用户覆盖基本缓存机制。例如，用户代理可能允许用户指定从不证实的缓存实体（甚至是确定过期的）。或者用户代理可能习惯于为每个请求添加“Cache-Control: max-stale=3600”。用户代理不应该缺省为任何非透明的行为，或者导致不正常的低效缓存结果的行为，但用户的明确行为可能会明确配置为这样做。

如果用户覆盖基本缓存机制，则用户代理应该在会引起所显示的信息可能不符合服务器的透明度需求时（特别是，当获知所显示的实体是过期的）明确提示用户。既然协议一般允许用户代理决定响应是否过期，这种提示只需要在实际发生时显示。提示不需要是对话框，它可能是一个图标（例如，一条鱼干图片）或其它的指示器。

如果用户以非正常地降低缓存效率的方式来覆盖缓存机制，用户代理应该持续指示这种状态给用户（例如，通过显示冒火的货币），以使用户不至于无意获取过量的资源或遭受过

量的潜在问题。

13.1.5 规则和警告的例外

在一些情况下，缓存服务器的操作者可能选择配置为返回过期响应，即使不是客户所请求的。这种决策可轻易作出，但可能对于可用性和性能的因素而言是必须的，特别当缓存服务器到原始服务器的连接很差时。不管何时缓存服务器返回过期的响应，它必须允许（使用 Warning 头部）客户端软件警告用户可能存在潜在的问题。它还允许客户代理采取进一步操作来获取第一手或更新的响应。基于这种原因，当客户端明确请求第一手或更新的响应时，缓存服务器不应该返回过期的响应，除非由于技术或策略因素不可能答应。

13.1.6 客户控制行为

当原始服务器（少量扩展，立即缓存，通过响应年龄的辅助）是截止信息的主要源时，在一些情况下，客户端可能需要控制缓存服务器关于是否返回没经证实的缓存响应的决策。客户端这样做可使用 Cache-Control 头部的几个指令。

客户端的请求可能指定它愿意接受的未经证实的响应的最大年龄；指定 0 值强制缓存服务器重新证实所的响应。客户端可能还规定响应过期的最小残留时间。这两种选项约束缓存服务器的行为，且不能进一步放宽缓存的语义透明的相似度。

客户端可能还会指定它将接受过期响应，基于一些最大数量的过期值。这放宽对缓存服务器的约束，且可能违反原始服务器规定的旅语义透明度的约束，但可能对于离线操作是必须的，或都面对差的连接性的高可用性。

13.2 截止模型

13.2.1 服务器规定截止

HTTP 缓存效率最佳，当缓存服务器能够完全避免请求原始服务器时。避免请求的主要机制是原始服务器提供明确的在将来截止的时间，表示响应可满足后序请求。换句话说，缓存服务器能够不首先联系服务器而返回更新的响应。

我们希望服务器将为响应指定将来明确截止时间，在截止时间到达之前实体不会改变，通过语法明显的方式。这通常会维护语义透明度，服务器的截止时间尽量要仔细选择。

截止机制只应用到缓存服务器获取的响应，而不是立即转发到请求客户的第一手响应。

如果原始服务器希望强制语义透明的缓存服务器证实每个请求，它可能指定过期的明确截止时间。这就是说，响应始终是过期的，因此缓存服务器应该在为后序请求使用它之前证实它。更严格的强制重新证实的方式见 14.9.4 节。

如果原始服务器希望强制任何 HTTP/1.1 缓存服务器，不管它如何配置，证实每个请求，它应该使用“must-revalidate”缓存控制指令（见 14.9 节）。

服务器既可使用 Expires 头部，也可使用 Cache-Control 头部的 max-age 指令来指定明确的截止时间。截止时间不能使用强制用户代理刷新它的显示或重新加载资源；它的语义只应用到缓存机制，且当对该资源的新的请求发起时这种机制只需要检查资源的截止状态。关于缓存和历史机制间区别的解释见 13.13 节。

13.2.2 启发式截止

既然原始服务器并非始终提供明确的截止时间，HTTP 缓存服务器一般指它启发式截止时间，用使用其它头部值（如同 Last-Modified 时间）的算法来估计好象有理的截止时间。HTTP/1.1 规范没有提供特别的算法，但在它的结上强制最差状况约束。由于启发式截止时间可能损害语义透明度，它们要使用非常谨甚，且我们鼓励原始服务器尽可能提供明确的截止时间。

13.2.3 年龄计算

为了知道缓存项目是否更新，缓存服务器需求知道其年龄是否超出其更新周期。我位在 13.2.4 节中讨论如何计算后者；本节描述如何计算响应或缓存项目的年龄。

在本讨论中，我们使用术语“现在”，即“主机执行计算时时钟的当前值。”使用 HTTP 的主机，但特别是运行原始服务器和缓存服务器的主机，应该使用 NTP[28]或一些相似的协议来同步它们的时钟到全球精确的标准时间。

HTTP/1.1 需要原始服务器尽可能在每个响应中发送 Date 头部，它给响应生成的时间（见节 14.18）。我们使用术语“日期值”表示 Date 头部的值，以适合算法运算的形式。

HTTP/1.1 使用 Age 响应头部来传输从缓存服务器获取时的响应消息的估计年龄。Age 域值是缓存服务器估计从响应产生或被原始服务器重新证实以来的总时间，Age 值是响应已经在从原始服务器到每个缓存服务器中停留的总时间，加上在网络路径上传输的总时间。

我们使用术语“年龄值”表示 Age 头部的值，以适合算法运算的形式。

响应的年龄可以用两种完全独立的方式来计算：

1、“现在”减去“日期值”，如果假设本地时钟与原始服务器的时钟很好地同步。如果结果是负数，则结果用 0 代替。

2、“年龄值”，如果响应路径上的所有缓存服务器都实现 HTTP/1.1。

上面给出我们有两种独立的方式计算所收到响应的年龄，我们可以将它们组合为：正确接收年龄=取最大值（现在-日期值，年龄值），且如果我们有几乎同步的时钟或全为 HTTP/1.1 的路径，就可得到可靠（保守的）的结果。

因为网络引入的延迟，一些明显简隔可能存在服务器生成响应的时间和下一跳的缓存服

务器或客户端收到的时间之间。如果不修正，这种延迟可能导致不正确的小年龄值。

国库请求导致返回的 Age 值必须在 Age 值生成之前被始终化，我位能够修正网络引入的延迟，通过记录请求发起的时间。这时，当收到 Age 值后，它必须被认为是请求发起的相对时间，不是响应收到的时间。该算法导致保守的行为，不管经历了多少延迟，因此我们计算：正确发起年龄=正确收到年龄+（现在-请求时间），这里的“请求时间”是引导该响应发送的请求的时间（通过本地时钟）。

总结当缓存服务器收到响应的年龄的计算算法：

```
/*
* 年龄值
*       是 Age 的值：缓存服务器收到的该响应的头部
* 日期值
*       是原始服务器 Date 头部的值
* 请求时间
*       是缓存服务器作出导致缓存服务器的响应的请求的（本地）时间
* 响应时间
*       缓存服务器收到响应的（本地）时间
* 现在
*       当前（本地）时间
*/
外观年龄=取最大值（0，响应时间-日期值）；
修正接收年龄=取最大值（外观年龄，年龄值）；
响应延迟=响应时间-请求时间；
修正发起年龄=修正接收年龄+响应延迟；
常驻时间=现在-响应时间；
当前年龄=修正发起年龄+常驻时间；
```

缓存条目的“当前年龄”通过增加从缓存条止最后被原始服务器所证实到修正发起时间之间的总时间（以秒为单位）来计算。当响应从缓存条目生成时，缓存服务器必须包括单个 Age 头部域在响应中，使用下缓存条目的“当前年龄”相同的值。

响应中存在 Age 头部意味着该响应不是第一手的。然而，返过来就不成立，因为响应中缺少 Age 头部域并不意味着该响应是第一手的，除非请求路径上的所有缓存服务器都与 HTTP/1.1 一致（例如，老版 HTTP 缓存服务器并不实现 Age 头部域）。

13.2.4 截止计算

为了判断响应是更新的或过期的，我们需要比较它的更新周期和它的年龄。年龄的计算在 13.2.3 中描述；本节描述如何计算更新周期，和如何判断响应已经截止。在如下的讨论中，值可以表示为适合算术运算的任何形式。我们使用术语“截止值”来表示 Expires 头部的值。我们使用术语“最大年龄值”来表示响应中的 Cache-Control 头部的“max-age”指令挟带的秒数的适当值（见 14.9.3 节）。

max-age 指令优先于 Expires，所以如果响应中存在 max-age，则计算是简单的：

更新周期=最大年龄值

否则，如果 Expires 存在于响应中，则计算方法是：

更新周期=截止值-日期值

注意：这两种计算方法都不易受到时钟不同步的攻击，因为所有的信息都来自于原始服务器。

如果 Expires、Cache-Control: max-age 或 Cache-Control: s-maxage（见 14.9.3 节）都没出现在响应中，且响应没有包括其它缓存限制，则缓存服务器可能使用启动式的更新周期计算方法。缓存服务器必须在任何年龄大于 24 小时的响应中附加没有被添加的 Warning 113。而且，如果响应的确有 Last-Modified 时间，则启发式截止值应该不超过从那时起的间隔的一小片。关于这种片段的典型设置可能是 10%。

判断响应是否截止的计算是非常简单的：

响应是更新的=（更新周期>当前年龄）

13.2.5 消除截止值的歧义

因为截止值是乐观分配的，所以两个缓存服务器包含关于同个资源的更新值可能是不同的。如果客户端执行获取收到请求的非第一手响应，该响应已经在其自己的缓存中更新过，且它的已存缓存条目的 Date 头部比新响应中的 Date 更新，这时客户端可以乎略该响应。如果这样，它可能用“Cache-Control: max-age=0”指令（见 14.9 节）来重发请求，强制检查原始服务器。如果缓存服务器拥有不同证实者的相同表述的两种更新响应，它必须使用有更近 Date 头部的一个。这种状况可能发生，因为缓存服务器是从其它缓存服务器共享响应的，或者因为客户端已经要求重载或重新证实显然更新的缓存条目。

13.2.6 消除多次响应的歧义

因为客户端可能通过多种路径接收响应，所以一些响应流通过一群缓存服务器集而其它的响应流通过另一群不同的缓存服务器集，客户端可能以不同于原始服务发送的顺序收到响应。我们将希望客户端使用最近生成的响应，即使如果旧响还应明显更新些。

体实标签和截止值都不能利用响应的顺序，由于晚的响应有意挟带更早的截止时间是不可能的。Date 值以一秒的粒度排序。

当客户端尝试重新证实缓存条目，且收到的响应包含似乎比现存的更老的 Date 头部时，这时客户端应该无条件重复该请求，且包含 Cache-Control: max-age=0 来强制任何中间的缓存服务器直接向原始服务器证实它们的拷贝，或者 Cache-Control: no-cache 来强制任何中间

的缓存服务器从原始服务器获取新的拷贝。

如果 Date 值相同，这时客户可以使用任何一个响应（或者可以请求新的响应，如果它是非常谨慎的话）。服务器禁止决定客户端能够自由在相同时间生成的响应间作出选择，如果截止时间交迭的话。

13.3 证实模型

当缓存服务器有过期的条目，它将会可能用于客户请求的响应时，它首先必需向原始服务器（可能是有更新响应的中间缓存服务器）检查来判断它的缓存条目是否依然可用。我们称其为“证实”缓存条目。既然我们不希望因缓存条目是好的而必需为重传全部响应付出过多代价，且我们不希望因缓存条止是无效的而为额外的来回旅程付出过多代价，因此 HTTP/1.1 协议支持使用条件方法。

支持条件方法的关键协议特性是与“缓存证实”有关的。当原始服务器产生完整响应时，它给它附加某种证实，并保留在缓存条目中。

当客户端（用户代理或代理缓存）为它拥有的缓存条目的资源作出条件请求时，它包括相关的证实在请求中。

服务器这时检查该证实与实体的当前证实，且，如果它们匹配（见 13.3.3 节），它用特殊状态码（通常是 304（Not Modified））响应且包括实体。否则，它返回完整响应（包括实体）。因此，当证实匹配时我们避免传输完整响应，当它不匹配时我们避免额外的来回旅程。

HTTP/1.1 中，条件请求看起来与对相同资源的常规请求极其相似，除了它挟带特殊的头部（其中包含证实）来暗中改变方法（通常是 GET）为条件化的。

协议包括正面和负面的缓存证实条件的理解。即，请求当且仅当证实匹配或当且仅当无证实匹配时执行方法是可能的。

注意：响应不含证实可能依然被缓存，且直到它截止为止被缓存服务器服务，除非这被 cache-control 指令明确地禁止。然而，缓存不能在它没有实体的证实时执行条件获取，这就是说当它截止后将不可更新。

13.3.1 Last-Modified 日期

Last-Modified 实体头部域值经常用来证实缓存。简单说来，如果实体在 Last-Modified 值以来不曾修改过，则可认为缓存条目是有效的。

13.3.2 实体标签缓存证实

ETag 响应头部域值，实体标签，提供“不透明的”缓存证实。这可能允许更可靠的证实，在不便存储修改日期，HTTP 日期值 1 秒精度不够，或原始服务器希望避免可由使用修改日期引起的某种矛盾，的状况下。

实体标签在 3.11 节中描述。实体标签使用的头部在 14.19、14.24、14.26 和 14.44 节中描述。

13.3.3 弱和强证言

既然所有原始服务器和缓存器都会比较两个证言来判断它们表述相同或不同的实体，通常一个希望如果实体（实体或任何实体头部）作任何改变，这时关联的证言同样会改变。如果这是真实的，这时我们称该证言是“强证言”。

然而，可能存在这种情况，当服务器宁愿只在语义上有明显改变时修改证言，并不是当实体修改方面非显著时。当资源改变而证言不一定改变，这称为“弱证言。”

实体标签一般是“强证言，”但协议提供机制来标示实体标签为“弱”的。

可认为强证言就是无论何时实体的位修改时都会改变的证言，而弱值在何时实体的意思修改时改变。也就是，可认为弱证言是特定实体的标识符，而弱证言是语义等同实体集的标识符的一部分。

注意：强证言的一个例子是在稳定存在器上每当实体修改时都加一的一个整数。

实体的修改时间，如果表示为 1 秒精度，将会是弱证言，因为资源可能在一秒内修改两次。

支持弱证言是可选的。然而，弱证言允许对等同对象更有效地缓存；例如，站点的点击计数器可能每隔几天或几周更新一次就足够好了，而在此期间内的相当的任何值似乎是“足够好”。

“使用”证言，要么在客户端生成请求且在证实头部域中包括证言时，要么在服务器比较两个证言时。

强证言可用在任何内容中。弱证言只用在不需要实体完全相同的内容中。例如，使用条件 GET 来获取完整实体可用任何类型。然而，只有强证言可用来获取子范围，不然客户端可能获取内部不一致的实体。客户端可以使用弱证言或强证言发出简单（非子范围）GET 请求。客户端禁止在其它形式的请求中使用弱证言。

HTTP/1.1 协议定义的唯一证言功能是比较。存在两种证言比较功能，取决于所比较的内容是否允许使用弱证言：

- 强比较功能：为了确认相同，两个证言必须完全相同，且都禁止是弱的。
- 弱比较功能：为了确认相同，两个语言必须完全相同，但两者之一或全部可以在不影响结果的情况下标为“弱”的。

实体标签是强的，除非它明确被标为弱的。3.11 节给出实体标签的语法。

Last-Modified 时间，当用于请求中的证言时，是潜在弱的，除非可能使用如下规则推出它是强的：

- 证言由原始服务器与实体的当前证言进行比较，且
- 原始服务器确信在表述证言的秒内相关实体没有修改两次。

或者，

- 客户端要使用的证言在 If-Modified-Since 或 If-Unmodified-Since 头部中，因为客户端有相关实体的缓存条目，且
- 该缓存条目包括 Date 值，由原始服务器发送原始响应时给出该值，且
- 表述的 Last-Modifed 时间至少是 Date 值的 60 秒以后。

或者，

- 证言由中间的缓存与存在缓存条目中的证言来比较，且
- 该缓存条目包含 Date 值，由原始服务器发送原始响应时给出该值，且
- 表述的 Last-Modified 时间至少是 Date 值的 60 秒以后。

该方法基于这样的事实，如果两个不同的响应由原始服务器在同一秒内发送，但都有相同的 Last-Modified 时间，则两个响应中至少有一个有与 Date 值相同的 Last-Modified 时间。武断的 60 秒限制可保护，Date 和 Last-Modified 值可能由不同的时钟生成，或可能在响应表述期间些许不同的时间。实现可以使用大于 60 秒的值，如果它认为 60 秒太短了。

如果客户端希望执行子范围获取，但它只使用 Last-Modified 时间且没有非透明的证言，则仅当 Last-Modified 时间如这时描述一样是强的时，它可以这么做。

缓存或原始服务器收到条件请求，而不是完整消息体的 GET 请求时，必须使用强比较功能来计算条件。

这些规则允许 HTTP/1.1 缓存和客户端安全执行由 HTTP/1.0 服务器获取的值的子范围获取。

13.3.4 使用实体标签和 Last-Modifed 日期时的规则

Last-Modified 日期

关于当应该使用不同证言类型时，我们为原始服务器，客户端和缓存采用规则和建议的集合。

HTTP/1.1 原始服务器：

- 应该发送实体标签证言，除非不可生成。

• 可以发送弱实体标签代替强实体标签，如果性能考虑支持使用弱实体标签，或如果不可发送强实体标签。

• 应该发送 Last-Modified 值，如果可发送的话，除非存在语义透明度崩溃的风险，可能会造成使用 If-Modified-Since 头部将导致严重问题的结果。

换句话说，HTTP/1.1 原始服务器的期望行为是发送强实体标签和 Last-Modified 值。

为了合法，强实体标签必须在不管何时关联实体值用任何方式修改时改变。弱实体标签应该在一旦关联实体通过语义上重要的方式修改时改变。

注意：为了提供语义透明缓存，原始服务器必须避免为两个不同实体重复使用特定的强实体标签值，或为两个语义不同的实体重复使用特定的弱实体标签。缓丰条目可能武断地持续长的周期，而不管截止时间，所以认为缓存将不会再次尝试使用它过去某刻获取的证言来证实某个条目将是种奢望。

HTTP/1.1 客户端：

• 如果原始服务器已经提供了实体标签，则必须在任何缓存条件请求中使用该实体标签（使用 If-Match 或 If-None-Match）。

• 如果原始服务器只提供了 Last-Modified 值，则应该在非子范围缓存条件请求中使用该值（使用 If-Modified-Since）。

• 如果 HTTP/1.0 原始服务器只提供了 Last-Modified 值，则可以在子范围缓存条件请求中使用该值（使用 If-Unmodified-Since）。用户代理应该在困难的情况下提供方式来取消这种行为。

• 如果原始服务器既提供了实体标签，又提供了 Last-Modified 值，则应该在缓存条件请求中使用两个证言。这就允许 HTTP/1.0 和 HTTP/1.1 缓存都正确响应。

HTTP/1.1 原始服务器收到条件请求，它包括 Last-Modified 日期（如，在 If-Modified-Since 或 If-Unmodified-Since 头部域中）和一个或多个实体标签（如，在 If-Match、If-None-Match 或 If-Range 头部域中）作为缓存证言，则禁止返回 304（Not Modified）状态响应，除非请求中的所有条件头部域都相容。

HTTP/1.1 缓存代理收到条件请求，它包括 Last-Modified 日期和一个或多个实体标签来作为缓存证言，则禁止返回本地缓存的响应给客户端，除非缓存的响应与请求中所有的条件头部域都相容。

注意：这些规则后的通用规则是，HTTP/1.1 服务器和客户端应该尽可能在它们的响应或请求中传输尽量多的非冗余信息。接收该信息的 HTTP/1.1 系统将对所收到的证言作出最保守的假设。

HTTP/1.0 客户端和缓存将乎略实体标签。通常，这些系统收到或使用的最后修改值将支持透明和高效的缓存，因此 HTTP/1.1 原始服务器应该提供 Last-Modified 值。在罕见的情况下，当 HTTP/1.0 系统使用 Last-Modified 值作为证言的地方可能造成严重问题时，HTTP/1.1 原始服务器就不该提供了。

13.3.5 非证实条件

实体标签后的规则只由服务器作者了解，资源的语义足够用于选择适当的缓存证实机制，且任何证言比较功能的规定比字节相等更为复杂，将打开蠕虫的罐头。因此，任何其它头部（除了 Last-Modified，为与 HTTP/1.0 兼容）的比较从不用于证实缓存条目。

13.4 响应可缓存

除非通过缓存控制（14.9 节）指令特别强制，缓存系统可以始终保存成功的响应（见 13.8 节）作为缓存条目，可以不经证实它是否更新就返回它，且可以在成功证实后返回它。没有缓存证言或清楚的截止时间与响应相关，我们不认为它会缓存，但确定的缓存可以违反这种期望（例如，当很窄或无网络连接可用时）。客户端可能通常通过比较 Date 头部与当前时间来检测从缓存中取出的这类响应。

注意：一些 HTTP/1.0 缓存明知违反这种期望而不提供任何 Warning。

然而，在一些状况下，缓存保留实体或在后序请求时返回它可能是不恰当。这可能因为服务器作者认为绝对语义透明是必要的，或因安全或隐私考虑。因此提供确定的缓存控制指令，以使服务器能够指示确定人资源实体，或其中部份，将不被缓存而不管其它考虑。

要注意，14.8 节通常阻止从保存和返回响应到之前包括 Authorization 头部的请求间共享缓存。

收到的状态码为 200、203、206、300、301 或 410 的响应可以存在在缓存中，并用于后序请求的响应，与截止机制有关的主题，除非缓存控制指令禁止缓存。然而，若缓存不支持 Range 和 Content-Range 头部，则禁止缓存 206（Partial Content）响应。

收到的任何其它状态码（如，状态码 302 和 307）的响应禁止作为后序请求的响应返回，除非有缓存控制指令或明确允许的其它头部。例如，包括如下：Expires 头部（14.21 节）；“max-age”、“s-maxage”、“mustrevalidate”、“proxy-evalidate”、“public”或“private”缓存控制指令（14.9 节）。

13.5 从缓存构造响应

HTTP 缓存的目的是保存请求对应的响应中收到的信息，以用来响应将来的请求。在许多情况下，缓存只简单驼回响应的适当部分给请求者。然而，如果缓存拥有基于以前响应的缓存条目，则它可能必须组合缓存条目作为新的响应的成份。

13.5.1 端到端和跳对跳头部

为了定义缓存和非缓存代理行为，我们将 HTTP 头部划分为两类：

• 端到端头部，将传输到请求或响应的最终接收者。响应中的端到端头部必须作为缓存条目的成份保存，且必须从缓存条目中经任何响应形式传输。

• 跳对跳头部，只对单传输级的连接有意义，且不会被缓存或代理转发。

下面的 HTTP/1.1 头部是跳对跳头部：
• Connection
• Keep-Alive
• Proxy-Authenticate
• Proxy-Authorization
• TE
• Trailers
• Transfer-Encoding
• Upgrade

HTTP/1.1 中定义的所有其它头部都是端到端头部。

其它的跳对跳头部必须在 Connection 头部中列出（14.10 节），将被引入 HTTP/1.1（或更新）中。

13.5.2 不可更改头部

HTTP/1.1 协议的一些特性，如摘要认证，取决于确定的端到端头部的值。透明的代理不该修改端到端头部，除非该头部的定义需要或特别允许这样做。

透明代理禁止修改请求或响应中的下列的任何域，且禁止在已经存在时添加任何这些域：

• Content-Location
• Content-MD5
• ETag
• Last-Modified

透明代理禁止修改响应中的任何下列域：

• Expires

但如果不存在时，它可以添加任何这些域。如果添加 Expires 头部域，它必须给出与响应中的 Date 头部相同的域值。

代理禁止修改或添加任何下列的域到包含非转化缓存控制指令的消息中，或任何请求中：

- Content-Encoding
- Content-Range
- Content-Type

非透明代理可以修改或添加这些域到不包含非转化的消息中，但如果这样做，它必须添加 Warning 214（Transformation applied）到不存在该警告的消息中（见 14.46 节）。

警告：对端到端头部不必要的修改将导致认证失败，如果将来的 HTTP 版本引入强认证机制。这种认证机制可以依赖这时没有列出的头部的值。

请求或响应的 Content-Length 域添加或删除按 4.4 节中的规则处理。透明代理必须保持实体的实体长度（7.2.2 节），尽管它可以改变传输长度（4.4 节）。

13.5.3 组合头部

当缓存向服务器证实请求，且服务器提供 304（Not Modified）响应或 206（Partial Content）响应时，缓存这时构造响应来发送给请求客户端。

如果状态码是 304（Not Modified），缓存使用缓存条目中保存的实体作为该发送响应的实体。如果状态码是 206（Partial Content）且 ETag 或 Last-Modified 头部精确匹配，缓存可以组合缓存条止中保存的内容和所收到响应的新内容，并使用其结果作为该发出响应的实体（见 13.5.4）。

端到端头部存储在缓存条目中，并用来构造响应，除了

- 任何保存的有警告码 1xx（见 14.46 节）的 Warning 头部必须从缓存条目和转发的响应中删除。

- 任何保存的有警告码 2xx 的 Warning 头部必须保留在缓存条目和转发的响应中。

- 任何 304 或 206 响应中提供的端到端头部必须从缓存条目中用相应的头部取代。

除非缓存决定删除缓存条目，它还必须用到来的响应中的相应头部替换缓存条目中保存的端到端头部，除了上面刚描述的 Warning 头部。如果到来的响应中的头部域名匹配缓存条目中的多个头部，则所以这类的老头部都必须替换。

换句话说，在到来的响应中收到的端到端头部的集体覆盖缓存条目中保存的所有相应的端到端头部（除了保存的警告码 1xx 的 Warning 头部，即使没有覆盖也会删除）。

注意：该规则允许原始服务器使用 304（Not Modified）或 206（Partial Content）响应来为同个实体或其中的子范围更新与以前的响应相关的任何头部，尽管这样做可能不是一直有

意义或正确的。该规则不允许原始服务器使用 304（Not Modified）或 206（Partial Content）响应来完全删除它在以前的响应中提供的头部。

13.5.4 组合字节范围

响应可能只传输实体子范围内的字节，要么因为请求包含一个或多个 Range 规定，要么因为连接过早地断开了。在几次这种传输后，缓存可能收到相同实全的多个范围。

如果缓存了实体的非空子范围集，且到来的响应传输另一个子范围，则缓存可以将新子范围与现在的集合组合在一起，如果满足如下两个条件：

- 到来的响应和缓存条目都有缓存证言。
- 这两个缓存证言使用强比较功能匹配（见 13.3.3 节）。

如果任何一个不满足，则缓存必须只使用最近的部分响应（基于每个响应传输的 Date 值，和使用这些值相等或缺失的到来响应），且必须丢弃其它的部分信息。

13.6 缓存协商响应

使用服务器驱动内容协商（12.1 节），如通过响应中的存在的 Vary 头部域来表示，改变缓存能够为后序请求响应该响应的条件和过程。服务器如何使用 Vary 头部域见 14.44 节。

服务器应该使用 Vary 头部域告诉缓存使用哪个请求头部域来从基于服务器驱动协商的多个可缓存响应的表述中选择。Vary 域值中给出的头部域集作为“选择”请求头部。

当缓存收到后序请求，它的 Request-URI 指定一个或多个缓存实体包括 Vary 头部域，缓存禁止使用这种缓存条目来给新请求构造响应，除非新请求中存在的所有选择请求头部与原始请求中存储的相应请求头部相匹配。

两个请求的选择请求头部定义来匹配当且仅当第一个请求中的选择请求头部能够被传输到第二个请求中的选择请求头部中，可增加或删除线性空白符（LWS）在相应的 NBF 允许的地方，且/或按照在 4.2 节中的消息头部规则组合多个有相同域名的消息头部域。

域值为“*”的 Vary 头部将始终匹配失败，且对该资源的后序请求只能够被原始服务器正确解释。

如果缓存条目的选择请求头部域与新请求的选择请求头部域不匹配，这时缓存禁止使用缓存条目来满足请求，除非它首先用条件请求接力新请求到原始服务器且服务器响应 304（Not Modified），其中包含实体标签或 Content-Location 来指明所使用的实体。

如果实体标签指定到缓存表述，则转发请求<应该>是有条件的，且从所有它缓存的该资源的条目包含实体标签在 If-None-Match 头部域中。这转让缓存当前拥有的实体集给服务器，因此如果这些实体中的任何一个匹配请求实体，服务器能够使用 ETag 头部域在它的 304（Not Modified）响应中，来告诉缓存该条目是适当的。

如果新响应的实体标签匹配存在条目，则新响应<应该>用于更新现在条目的头部域，且结果<必须>返回给客户端。

如果任何现存的缓存条目只包含相关实体的部分内容，则它的实体标签<不该>包括在 If-None-Match 头部域中，除非该条目可以完全满足所请求的范围。

如果缓存收到成功响应，其 Content-Location 域匹配相同 Request-URI 的存在条目，其实体标签与已存条目的不同，且其 Date 比已存条目更近，则<不该>为以后的请求在响应中返回该已存条目，且<应该>从缓存中删除。

13.7 共享和非共享缓存

基于安全和策略的原始，区分“共享”和“非共享”缓存是必要的。

非共享缓存只可由单个用户访问。在这种情况下的可访问性<应该>通过适当的安全机制来强制。所有其它的缓存都被认为是“共享”的。本规范的其它章节中存在操作共享缓存的特定约束，为阻止策略丢失或访问控制失败。

13.8 错误或非完整响应缓存行为

缓存收到非完整响应（例如，有比 Content-Length 头部中指定的字节数更少的数据）<可以>保存该响应。然而，缓存<必须>将其作为部分响应。部分响应<可以>按 13.5.4 节中描述的来组合；结果可能是个完整响应或可能依然是部分的。缓存<禁止>使用 206（Partial Content）状态码来返回部分响应给客户，除非客户明确要求这样。缓存<禁止>使用 200（OK）状态码来返回部分响应。

如果缓存在尝试重新证实条目时收到 5xx 响应，则它<可以>转发该响应给请求客户端，或如服务器响应失败一样做。在后种情况下，它<可以>返回以前收到的响应，除非缓存条目中包括“must-revalidate”缓存控制指令（见 14.9 节）。

13.9 GET 和 HEAD 的副作用

除非原始服务器明确缓存其响应，否则当其响应是从缓存中获取时，在任何资源上的 GET 和 HEAD 方法的应用程序<不该>有导致错误行为的副作用。它们依然<可以>有副作用，但缓存在其缓存决策中不需要考虑。缓存始终希望观在缓存时察原始服务器清楚的约束。

我们说明该规则一个例外：既然一些应用程序已经习惯用查询 URL（在 rel_path 部分中包含“?”的 URL）来使用 GET 和 HEAD，以执行有明显的副作用的操作，则缓存<禁止>将这种 URI 的响应作为更新的，除非服务器提供明确的截止时间。这对于 HTTP/1.0 服务器的这类 URI 的响应<不该>从缓存中获取的行为有明显的意义。相关信息见 9.1.1 节。

13.10 更新或删除后失效

原始服务器招行在资源上的特定操作的效果可能引起一个或多个已存缓存条目变为非透明失效。即，尽管它们可能继续“更新”的，但是它们不能精确反映原始服务器对该资源上新响应的返回响应。

HTTP 协议没有办法保证所有这类缓存条目都标为失效。例如，引起原始服务器上改变的请求可能没有从它存在的代理中消失。然而，有几个规则将帮助减少错误行为的可能性。

在本节中，短语“失效实体”意思是，缓存将从存储器中删除它，或将标为“失效”且需要强制重新证实来在后序请求中返回它们。

一些 HTTP 方法<必须>引起缓存失效实体。这是 Request-URI 或 Location 或 Content-Location 头部（如果存在）引用的实体。这些方法是：

- PUT
- DELETE
- POST

为了阻止拒绝服务器攻击，无效基于 Location 或 Content-Location 头部中的 URI<必须>只在主可部分与 Request-URI 中的相同时执行。

有其不认识的请求方法通过的缓存<应该>无效由 Request-URI 引用的任何实体。

13.11 写通强制

可能希望引起对原始服务器资源的修改的所有方法<必须>写通到原始服务器。当前这包括除了 GET 和 HEAD 外的所有方法。<禁止>缓存在已经传输请求到边界服务器且已经从边界服务器收到了相应的响应前回复客户端的这种请求。这不会阻止代理缓存在边界服务器已经发送其最终回复前发送 100（Continue）响应。

二选一（如“回写”或“回拷”缓存）在 HTTP/1.1 中不允许，因为提供一致更新是困难的，且提高服务器、缓存或网络先于回写前失败的可能性的问题。

13.12 缓存替换

如果新的可缓存（见 14.9.2、13.2.5、13.2.6 和 13.8 节）响应从某个资源收到，然而相同资源的任何存在响应已经缓存，则缓存<应该>使用新和响应来加得当前的响应。它<可以>插入缓存存储器中，且<可以>，如果它满足所有其它要求，使用它来响应任何将来的请求，这些请求以前会引起返回老的响应。如果它将新响应插入缓存存储器中，则要应用 13.5.3 节中的规则。

注意：新的响应的 Date 头部值比存在缓存响应的老，则该响应是不可缓存的。

13.13 历史清单

用户经常使用历史机制，如“后退”按钮和历史清单，它们能够用于重新显示会话中获取的实体。

历史机制和缓存是不同的。特别是，历史机制<不该>显示资源当前状态的语义透明的视图。正确的是，历史机制意思是精确显示资源获取时用户所见。

缺省时，截止时间不应用到历史机制。如果实体还在存在器中，则历史机制<应该>显示它，即使该实体已经截止，除非用户已经特别配置代理更新截止的历史文档。

这不应解释为禁止历史机制告诉用户视图可能过期了。

注意：如果历史清单机制非必要地阻止用户查看过期的资源，这将趋向于强制服务器作者避免使用他们所需的 HTTP 截止控制和缓存控制。服务器作者可能认为，当用户使用导航控制（如后退）来显示以前获取的资源时，用户不会被提供错误消息或警告消息。即使有时这类资源将不会缓存或可能很快截止，用户接口考虑也可能强制服务器作者求助阻止缓存的其它意思（如，“once-only”URL），为了不遭受历史机制的不正确功能的影响。

14 头部域定义

本节定义所有标准 HTTP/1.1 头部域的语法和语义。对于 entity-header 域，发送方和接收方指客户端或服务器，取决于谁发送和谁接收该实体。

14.1 Accept

可能使用 Accept request-header 域来指定响应可接受的确定媒体类型。能够使用 Accept 头部来指出请求明确限制到小范围的希望类型集，如请求在线图片的情况。

```
Accept = "Accept" ":" #( media-range [ accept-params ] )
media-range = ( "*/*" | ( type "/" "*" ) | ( type "/" subtype ) ) *( ";" parameter )
accept-params = ";" "q" "=" qvalue *( accept-extension )
accept-extension = ";" token [ "=" ( token | quoted-string ) ]
```

星号“*”字符用于媒体类型的范围组，使用“*/*”指出所有媒体类型，“type/*”指出该 type 的所有 subtype。media-range【可以】包括可应用到该范围的媒体类型 parameter。

敏个 media-range【可以】接着一个或多个 accept-params，开始的“q”parameter 指出相对的质量因素。首个“q”parameter（如果有）分隔 media-range parameter 与 accpet-params。质量因素允许用户或用户代理指出对该 media-range 的相对选择等级，使用从 0 到 1 的 qvalue 因子（3.9 节）。缺省值是 q=1。

注意：按历史实践来使用“q”参数名来分隔媒体类型 parameter 与 Accept 扩展参数。尽管这样会阻止媒体类型使用任何命名为“q”的媒体类型参数，可认为这种事情不情愿要求 IANA 媒体类型注册表中不存在任何“q”参数，且很少使用在 Accept 中的任何媒体类型参数中。不鼓励未来的媒体类型注册命名为“q”的任何参数。

例子

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

【应该】解释为“我喜欢 audio/basic，但如果最佳有效的质量在 80%以上则给我发送任何 audio 类型。”

如果不存在 Accept 头部域，这时假定客户端接受所有媒体类型。如果存在 Accept 头部域，且如果服务器不能依据 Accept 域值的组合来发送任何可接受的响应，这时服务器【应该】发送 406（Not Acceptable）响应。

更精心准备的例子是

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

口头上应解释为“text/html 和 text/x-c 是喜欢的媒体类型，但如果不存在，这时发送 text/x-dvi 实体，如果它也不存在，发送 text/plain 实体。”

媒体范围可由更多规定媒体范围或规定媒体类型所覆盖。如果多于 1 个媒体范围应用到给出的类型，则最精确的规定参考有优先权。例如，

Accept: text/*, text/html, text/html;level=1, */*

按如下优先级：

1） text/html;level=1
2） text/html
3） text/*
4） */*

与给出类型相关的媒体类型质量因素由按最高优先级发现的匹配该类型的媒体范围决定。例如，

Accept: text/*;q=0.3, text/html;q=0.7, text/html;level=1, text/html;level=2;q=0.4, */*;q=0.5

将引起关联如下的值：

text/html;level=1	= 1
text/html	= 0.7
text/plain	= 0.3
image/jpeg	= 0.5
text/html;level=2	= 0.4
text/html;level=3	= 0.7

注意：用户代理可以为确定媒体范围提供缺省的质量集。然而，除非用户代理关闭系统，使其不能与其它显示代理交互，该缺省集应由用户配置。

14.2 Accept-Charset

Accept-Charset request-header 域能够用来指出响应可接受的字符集。该域允许客户端能够理解更复杂或特殊用途的字符集来发送该能力的信号给服务器，即它有能力描绘使用该字符集的文档。

Accept-Charset = "Accept-Charset" ":" 1#((charset | "*")[";" "q" "=" qvalue])

字符集值在 3.4 节中描述。每个 charset【可以】给出相关的质量值，描绘用户对该 charset 的喜欢程度。缺省值是 q=1。一个例子是

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

特别值“*”，如果出现在 Accept-Charset 域中，匹配在 Accept-Charset 域的其他地方没有提及的每个字符集（包括 ISO-8859-1）。如果 Accept-Charset 域中不存在“*”，这时全部没有明确提及的字符集其质量值都为 0，除 ISO-8859-1 外，如果没有明确提及其质量值为 1。

如果不存在 Accept-Charset，缺省可接受任何字符集。如果存在 Accept-Charset 头部，且如果服务器不能依照 Accept-Charset 头部发送可接受的响应，这时服务器【应该】发送 406（Not Acceptable）状态码的错误响应。尽管允许发送不可接受的响应。

14.3 Accept-Encoding

Accept-Encoding rqeust-header 域与 Accept 相似，但限制响应可接受的 content-coding（3.5 节）。

Accept-Encoding = "Accept-Encoding" ":" 1#(codings [";" "q" "=" qvalue])
codings = (content-coding | "*")

其使用例子有：

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

服务器依照 Accept-Encoding 域测试 content-coding 是否可接受，使用如下规则：

1、如果 content-coding 是 Accept-Encoding 域中所列 content-coding 中的一项，这时它可

接受，除非它陪伴的 qvalue 是 0。（如 3.9 节所述，qvalue 为 0 意味着“不可接受”。）

2、Accept-Encoding 域中的特殊符号“*”匹配没有明确在头部域中列出的任何可用 content-coding。

3、如果有多个 content-coding 可接受，这时选择 qvalue 为最大非 0 的可接受 content-coding。

4、“identity”content-coding 始终可接受，除非特别拒绝，因为 Accept-Encoding 域包括“identity;q=0”，或因为该域包括“*;q=0”且没有明确包括“identity”content-coding。如果 Accept-Encoding 的 field-value 为空，这时只有“identity”编码可接受。

如果请求中存在 Accept-Encoding 域，且如果服务器不能按照 Accept-Encoding 头部发送可接受的响应，这时服务器【应该】发送 406（Not Acceptable）状态码的错误响应。

如果请求中不存在 Accept-Encoding 域，服务器【可以】假设客户端可接受任何内容编码。在该情况下，如果“identity”是可用 content-coding 中的一个，这时服务器【应该】使用“identity”content-coding，除非额外信息表明其它 content-coding 对客户端更有意义。

注意：如果请求中不包括 Accept-Encoding 域，且如果“identity”content-coding 不可用，这时选择 HTTP/1.0 客户端通常理解的 content-coding（如，“gzip”和“compress”）；一些老的客户端不正确显示用其它 content-coding 发送的消息。服务器还可以依据特殊用户代理或客户端的信息做出决策。

注意：大多数 HTTP/1.0 应用程序不认识或服从与 content-coding 关联的 qvalue。意思是 qvalue 将没有作用，且不允许与 x-gzip 或 x-compress 一起使用。

14.4 Accept-Language

Accept-Language request-header 域与 Accept 相似，但限制请求所选择的作为响应的自然语言集。语言标签在 3.10 节中定义。

```
Accept-Language = "Accept-Language" ":" 1#( language-range [ ";" "q" "=" qvalue ] )
language-range = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

每个 language-range【可以】给出相关的质量值，表示该范围指定语言的用户选择的估价。质量值缺省为“q=1”。例如，

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

将意味着“我选择丹麦语，但会接受英国英语和其它类型的英语。”如果与标签精确相等时，或如果与首个标签字符后接前缀“-”的标签的前缀精确相等时，language-range 匹配 language-tag。如果在 Accept-Language 域中存在特殊范围“*”，则匹配与在 Accept-Language 域存在的其它范围不匹配其它全部标签。

注意：前缀匹配规则的使用不意味着语言标签按这种方式分配语言，即始终是真实的，如果用户理解特定标签的语言，这时该用户将也会理解以该标签为前缀的所有语言。前缀规则简单允许在这种情况下使用前缀标签。

由 Accept-Language 域指定给 language-tag 的语言质量因子是域中匹配 language-tag 的最长的 language-range 的质量值。如果域中没有 language-range 匹配标签，则语言质量因子指定为 0。如果请求中不存在 Accept-Language 头部，则服务器【应该】假设所有语言是同等可接受。如果存在 Accept-Language 头部，这时其指定的质量因子大于 0 的所有语言都是可接受的。

用户在每个请求中都发送有完整用户语言选择的 Accept-Language 头部可能与保密希望相违备。关于该问题的讨论，见 15.1.4 节。

由于可理解性高度取决于个人用户，推荐客户端应用程序使用户可选择语言选项。当选不可用时，这时【禁止】在请求中给出 Accept-Language 头部域。

注意：当使语言选项的选择对用户有效时，我们提醒实现者要注意这样的事实，即用户并不熟悉上述的语言匹配细节，且应该提供适当的指导。例如，用户可能假设，选择“en-gb”，他们将获得任何类型的英语文档，若英国英语不可用。在这种情况下，用户代理可能建议增加“en”来获得最佳匹配的行为。

14.5 Accept-Ranges

Accept-Ranges response-header 域允许服务器指出它对某个资源请求的可接受范围：

```
Accept-Ranges = "Accept-Ranges" ":" acceptable-ranges
acceptable-ranges = 1#range-unit | "none"
```

接受 byte-range 请求的原始服务器【可以】发送

```
Accept-Ranges: bytes
```

但不要求这样做。客户端【可以】生成 byte-range 请求，即使没有收到所用资源的该头部。范围单位在 3.12 节中定义。

服务器不接受对某个资源的任何类型的范围请求【可以】发送

```
Accept-Ranges: none
```

来建议客户端不要尝试范围请求。

14.6 Age

Age response-header 域传达发送方对原始服务器生成响应（或它重新生效）以来所逝总时间的估计值。如果年龄没有超过它的更新周期，则缓存响应是“更新的”。年龄值按 13.2.3 节中指定的方法计算。

Age = "Age" ":" age-value
age-value = delta-seconds

Age 值是非负十进制整数，表示按秒为单位的时间。

如果缓存收到大于它能够表示的最大正整数的值，或如果它的年龄计算的任何步骤有溢出，则它【必须】转换 Age 头部值为 2147483648（2^31）。包括缓存的 HTTP/1.1 服务器【必须】在由其自己的缓存生成的每个响应中包括 Age 头部域。缓存【应该】使用至少 31 位范围的算数类型。

14.7 Allow

Allow entity-header 域列出由 Request-URI 标识的资源所支持的方法集。该域的用途是严格提醒接收方与该资源相关的有效方法。任何 Allow 头部域【必须】出现在 405（Method Not Allowed）响应中。

Allow = "Allow" ":" #Method

用法举例：

Allow: GET, HEAD, PUT

该域不能阻止客户端尝试其它方法。然而，【应该】遵守由 Allow 头部域值所给出的指示。允许方法的实际集合由原始服务器在每个请求的时刻定义。

Allow 头部域【可以】提供给 PUT 请求，以建议由新的或修改资源所支持的方法。不需要服务器支持这些方法，且【应该】在响应中包括 Allow 头部域来给出实际所支持的方法。

【禁止】代理修改 Allow 头部域，即使它理解所指定的全部方法，因为用户代理可能有与原始服务器交流的其它用意。

14.8 Authorization

在收到 401 响应后，通常用户代理希望向服务器认证它自己，但非必需。通过在请求中包括 Authorization 请求头部域可以做到。Authorization 域值由包含用户代理为所请求资源的 realm 的认证信息的 credentials 组成。

Authorization = "Authorization" ":" credentials

HTTP 访问认证在“HTTP Authentication: Basic and Digest Access Authentication” [43]

中描述。如果请求被认证用指定了 realm，则相同的 credentials【应该】对在该 realm 内的所有其它请求都是有效的（假设认证方案自己没有其它需要，如 credentials 依 challenge 值变化或使用同步时钟）。

当共享缓存（见 13.7 节）收到包含 Authorization 域的请求时，它【禁止】返回相应的响应作为任何其它请求的回复，除非出现某个下面规定的例外：

1、如果响应包括“s-maxage”缓存控制指令，缓存【可以】使用该响应来回复后序的请求。但（如果所指定的最大年龄值已经超过）代理缓存【必须】首先重新向原始服务器证实它，使用新请求的请求头部来允许原始服务器认证新请求。（这是 s-maxage 所定义的行为。）如果响应包括“maxage=0”，则代理【必须】在重用它前始终重新证实它。

2、如果响应包括“must-revalidate”缓存控制指令，缓存【可以】使用该响应来回复后序的请求。但如果响应是过期的，所有缓存条目【必须】向原始服务器重新证实，使用新请求的请求头部以允许原始服务器认证新请求。

3、如果响应包括“public”缓存控制指令，它【可以】将其返回以回复任何后序请求。

14.9 Cache-Control

Cache-Control general-header 域用于指定在请求/响应链上所有缓存机制【必须】服从的指令。该指令规定的行为意图阻止缓存对请求或响应的逆向妨碍。这些指令通常覆盖缺省缓存算法。缓存指令是在一些情况下是不相同的，即在请求中指令的存在交不意味着在响应中给出相同的指令。

要注意，HTTP/1.0 缓存可能没有实现 Cache-Control，且可能没有实现 Pragma: no-cache（见 14.32 节）。

缓存指令【必须】穿过代理或网关应用程序，不管它们对该应用程序的意义，因为该指令可能应用到请求/响应链上的所有接收方。不可能为特定缓存指定缓存指令。

```
Cache-Control = "Cache-Control" ":" 1#cache-directive
cache-directive = cache-request-directive | cache-response-directive
cache-request-directive = "no-cache"                          ; 14.9.1 节
                        | "no-store"                          ; 14.9.2 节
                        | "max-age" "=" delta-seconds         ; 14.9.3、14.9.4 节
                        | "max-stale" [ "=" delta-seconds ]    ; 14.9.3 节
                        | "min-fresh" "=" delta-seconds       ; 14.9.3 节
                        | "no-transform"                      ; 14.9.5 节
                        | "only-if-cached"                    ; 14.9.4 节
                        | cache-extension                     ; 14.9.6 节
 cache-response-directive = "public"                          ; 14.9.1 节
                          | "private" [ "=" <"> 1#field-name <"> ] ; 14.9.1 节
                          | "no-cache" [ "=" <"> 1#field-name <"> ]; 14.9.1 节
```

```
                    | "no-store"                    ; 14.9.2 节
                    | "no-transform"                ; 14.9.5 节
                    | "must-revalidate"             ; 14.9.4 节
                    | "proxy-revalidate"            ; 14.9.4 节
                    | "max-age" "=" delta-seconds   ; 14.9.3 节
                    | "s-maxage" "=" delta-seconds  ; 14.9.3 节
                    | cache-extension               ; 14.9.6 节
cache-extension = token [ "=" ( token | quoted-string ) ]
```

当指令没有出现任何 1#field-name 参数时，该指令应用到全部请求或响应中。当这类指令出现 1#field-name 参数时，它只应用到该命令域或域，而非其它请求或响应。该机制支持可扩展性，HTTP 协议的将来版本的实现可能应用这些指令到 HTTP/1.1 中没有定义的头部域中。

缓存控制指令可能被分解为这些一般种类：

——限制任何可缓存；这只可以由原始服务器利用。

——限制缓存可存储什么；这可以由原始服务器或用户代理利用。

——修改基本截止机制；这可以由原始服务器或用户代理利用。

——控制缓存重证实或重载；这只可由用户代理利用。

——控制实体转化。

——扩展缓存系统。

14.9.1 什么可缓存

缺省时，若请求方法、请求头部域、和响应状态的需求指出它是可缓存的，则响应是可缓存的。13.4 节总结这些缺省的可缓存性。下面的 Cache-Control 响应指令允许原始服务器覆盖响应的缺省可缓存性：

public
: 指出该响应【可以】被任何缓存器缓存，即使它通常是不可缓存的或只在非共享缓存中可缓存的。（其它细节另见 Authorization、14.8 节。）

private
: 指出响应消息中的所有部分是用于单个用户的且【禁止】被共享缓存器所缓存。这允许原始服务器使只用于一个用户的响应和对其他用户所请求的无效响应的指定部分过期。私有（非共享）缓存【可以】缓存该响应。

  注意：该用法的描述只私有控制响应可以被缓存的地方，且不能确保消息内容的私密性。

no-cache

如果 no-cache 指令没有指定 field-name，这时缓存【禁止】在没有成功向原始服务器重证实前使用该响应来满足后序请求。这允许原始服务器阻止缓存，甚至缓存已经配置为返回过期响应给客户端请求。

如果 no-cache 指令确实规定一个或多个 field-name，这时缓存【可以】使用响应来满足后序请求，到任何其它缓存限制的主题。然而，规定的 field-name【禁止】在响应中发送给后序请求，除非成功向原始服务器重证实。这允许原始服务器阻止重用响应中确定的头部域，同时依然允许缓存响应的其余部分。

注意：大多数 HTTP/1.0 缓存将不认训或服从该指令。

14.9.2 缓存可存储什么

no-store

no-store 指令的用途是阻止不经意间释放或保存敏感信息（例如，在备份磁带上）。no-store 指令应用到整个消息，且【可以】在响应或请求中发送。若在请求中发送，缓存【禁止】保存该请求或任何对其响应的任何部分。若在响应中发送，缓存【禁止】保存该响应或引起的请求的任何部分。该指令应用到非共享和共享缓存。“【禁止】保存”在本处意味着缓存【禁止】在内部保存信息到固定存储器上，且【必须】在转发后尽可能迅速且最有效地尝试从非固定存储器上删除该信息。

即使该指令与响应相关，用户可能在缓存系统外部明确保存这种响应（如，通过“另存为”对话框）。历史缓存【可以】将这种响应作为它们正常操作的一部分保存。

该指令的用途是满足关心通过非可预料地访问缓存数据结构意外释放信息的确定用户和服务作者的规定要求。使用该指令可能在一些情况下改进私密性的同时，我们提醒它“并非”在任何方式下都是可靠或充分的机制来确保私密性。特别是，恶意或妥协缓存可能不认识或服从该指令，且通迅网络可能易受偷听。

14.9.3 修改基本截止机制

实体的截止时间【可以】由原始服务器使用 Expires 头部来指定（见 14.21 节）。另一种选择是，它【可以】在响应中使用 max-age 指令来规定。当 max-age 缓存控制指令出现在缓存响应中时，如果该年龄超过该资源新响应时候给出的年龄值（以秒为单位），则该响应是过期的。响应中的 max-age 指令意味着该响应是可缓存的（如，“public”），除非其它某种、更严厉的限制也存在。

如果响应中包括 Expires 头部和 max-age 指令，则 max-age 指令覆盖 Expires 头部，即使 Expires 头部更严厉。该规则允许原始服务器为给定响应提供比 HTTP/1.0 缓存更长截止时间

给 HTTP/1.1（或更新）缓存。当确定 HTTP/1.0 缓存可能因非同步时钟不正确计算年龄或截止时间时，这可能有用。

许多 HTTP/1.0 缓存实现将小于等于响应的 Date 值的 Expires 值作为与 Cache-Control 响应指令“no-cache”等价。如果 HTTP/1.1 缓存收到这种响应，且响应没有包括 Cache-Control 头部域，它【应该】认为该响应是不可缓存的，以便保持与 HTTP/1.0 服务器的兼容性。

注意：原始服务器可能希望在包括老式不理解该特性的缓存的网络中使用相关新的 HTTP 缓存控制特性，比如“private”指令。原始服务器将需要组合新特性与 Expires 域，后者的值小于等于 Date 值。这将防止对老式缓存的不正确响应。

s-maxage 如果响应包括 s-maxage 指令，这时对于共享缓存（但不对私有缓存），最大年龄由该指令规定，将覆盖由其它 max-age 指令或 Expires 头部规定的最大年龄。s-maxage 指令还实现 proxy-revalidate 指令（见 14.9.4 节）的语义，如，共享缓存在它成为对后序请求过期的响应后且没能首先向原始服务器重证实前禁止使用该条目。私有缓存始终乎略 s-maxage 指令。

要注意，最老的缓存，不与该规定一致，没有实现任何缓存控制指令。原始服务器希望使用缓存控制指令来限制，但非阻止，HTTP/1.1 一致的缓存器缓存【可以】使用 max-age 指令会覆盖 Expires 头部的要求，和这种事实，即前期 HTTP/1.1 一致的缓存没能观察 max-age 指令。

其它指令允许用户代理修改基本截止机制。这些【可以】在请求中指定：

max-age 指出客户端希望接受响应，其年龄不比规定的秒数长。除非 max-stale 指令也包括其中，客户端不希望接受过期响应。

min-fresh 指出客户端希望接受响应，其更新周期不比当前年龄加上指定秒数小。即，客户端希望响应依然是更新的，至少在指定的秒数以内。

max-stale 指出客户端希望接受响应，它已经超过其截止时间。如果 max-stale 指定了值，这时客户端希望接受响应，它没有超过截止时间指定的秒数。如果没有指定值给 max-stale，这时客户端希望接受任何年龄的过期响应。

如果缓存返回过期响应，或者因为请求中的 max-stale 指令，或者因为缓存配置为覆盖响应的截止时间，缓存【必须】附加 Warning 头部到过期响应中，使用 Warning 110（Response is stale）。

缓存【可以】配置为返回未经证实的过期响应，但仅当这没有与缓存证实有关（例如，“must-revalidate”缓存控制指令）的任何“【必须】”级相冲突时。

若新请求和缓存条目包括“max-age”指令，这时这两个值中的小者用来决定该请求的缓存条目的更新度。

14.9.4 缓存重证实和重载控制

某些时候，用户代理可能希望或需要坚持，缓存向原始服务器（而非只向原始服务器路径上的下个缓存）重证实它的缓存条目，或向原始服务器重载它的缓存条目。端对端重证实可能是需要的，如果缓存或原始服务器都对缓存的响应的截止时间评估过高。端对商量重载可能是需要的，如果缓存条目已经因某种原因成为坏的。

端到端重证实可以请求，或者当客户端没有其自己的本地拷贝，在这种情况下我们称之为“非确定端到端重证实”，或者当客户端确有本地拷贝，在这种情况下我们称之为“确定端到端重证实”。

客户端可以使用 Cache-Control 请求指令指定这 3 种行为：

端到端重载	请求包括“no-cache”缓存控制指令，或为与 HTTP/1.0 客户端兼容的“Pragma: no-cache”。【禁止】在请求中包括有域名的 no-cache 指令。服务器【禁止】使用缓存拷贝，当对这种请求响应时。
确定端到端重证实	请求包括“max-age=0”缓存控制指令，它强制到原始服务器路径上的每个缓存重证实它自己的条目，如果有，通过下个缓存或服务器。初始请求包括对客户端当前证言的 cache-validating 条件。
非确定端到端重证实	请求包括“max-age=0”缓存控制指令，它强制到原始服务器路径上的每个缓存重证实其自身的条目，如果有，通过下个缓存或服务器。初始请求不包括缓存证实条件；首个路径上拥有缓存该资源的缓存（如果有）包括对其当前证言的缓存证实条件。
max-age	当强制中间缓存,意思是 max-age=0 指令，来重证实它自己的缓存条目，且客户端已经在请求中提供它自己的证言，所提供的证言可能与缓存条目中保存的当前证言不同。在这种情况下，缓存【可以】使用两个证言之一来创建它自己的请求，而不影响语义透明度。 然而，证言的选择可能影响性能。在创建它的请求时对中间缓存最好的方法是使用它自己的证言。如果服务器回复 304（Not Modified），这时缓存可能返回它已经证实的拷贝给客户端，用 200（OK）响应。如果服务器回复新的实体和缓存证言，然而，中间缓存可以比较返回的证言与客户端请求中提供的，使用强比较功能。如果客户端的证言与原始服务器的相同，这时中间缓存简单返回 304（Not Modified）。否则，它以 200（OK）响应返回新的实体。 若请求包括 no-cache 指令，它【不该】包括 min-fresh、max-stale 或 max-age。
only-if-cached	在某些情况下，如网络连接性极差时，客户端可能希望缓存

只返回该响应，它当前已经存储的，且没有向原始服务器重载或重证实。在这样做，客户端可以包括 only-if-cached 指令在请求中。如果它收到该指令，缓存【应该】或者响应以缓存的条目，它与请求的其它约束一致，或者响应以 504（Gateway Timeout）状态。然而，如果缓存组下操作为有良好内部连接的统一系统，该请求【可以】在该缓存组中转发。

must-revalidate

因为缓存【可以】配置为乎略服务器规定的截止时间，且因为客户端请求【可以】包括 max-stale 指令（有相似效果），协议还包括机制来为原始服务器要求重证实任何后序使用的缓存条目。当 must-revalidate 指令存在于缓存所收到的响应中时，该缓存【禁止】在没有首先向原始服务器重证实前它使用该成为对后序请求过期的响应条目。（如，缓存【必须】每次进行端到端证实，如果缓存响应是过期的，独自基于原始服务器的 Expires 或 max-age 值。）

对于支持确定协议特性的可靠操作来说，must-revalidate 指令是需要的。在所有情况下，HTTP/1.1 缓存【必须】服从 must-revalidate 指令；特别是，如果缓存因任何原因不能访问原始服务器，它【必须】生成 504（Geteway Timeout）响应。

服务器【应该】发送 must-revalidate 指令，当且仅当重证实实体请求失败可能导致不正确地操作，如默默地非执行金融事务。接收方【禁止】作出任何违反该指令的自动行为，且若重证实失败，【禁止】自动提供实体的未证实拷贝。

尽管不是推荐的，用户在严厉连接性约束下操作【可以】违反该指令，但如果这样，【必须】明确警告用户，已经提供未证实响应。该警告【必须】提供给每个未证实访问，且【应该】明确要求用户确认。

proxy-revalidate

proxy-revalidate 指令有与 must-revalidate 指令相同的意思，除了它不应用到非共享用户代理缓存。它可用在对已认证请求的响应中，以允许用户的缓存保存，并且以后返回的响应不需要重证实（因为它已经被该用户认证过一次），同时依然要求服务许多用户的代理每次重证实（为了确保每个用户已经被认证）。要注意，这种认证响应还需要公共缓存控制指令，以便允许它们也被缓存。

14.9.5 no-transform 指令

no-transform

中间缓存的实现者（代理）已经发现转换确定实体主体的媒体类型是有用的，例如，在图像格式间转换，以节省缓存空间或减少低速链路上的通信总量。

然而，当这些转换应用到用于特定类型应用程序的实体主体

时，将发生严重操作性问题。例如，医学图像、科学数据分析和使用端到端认证的应用程序全部取决于接收的实体主体，它是与原始实体主体按位相同的。

因此，如果消息包括 no-transform 指令，中间缓存或代理【禁止】修改 13.5.2 节中所列的与 no-transform 指令对应的头部。这意味着，缓存或代理【禁止】修改指定该头部的实体主体的任何方面，包括实体主体自己的值。

14.9.6 缓存控制扩展

Cache-Control 头部域能够通过使用一个或多个 cache-extension 符号来扩展，每个有可选的指定值。信息性扩展（不需要改变缓存行为）【可以】添加，而不修改其它指令的语义。行为扩展设计来作为对已存基本缓存指令的修饰符。新的指令和标准指令都支持，这样应用程序不理解新指令将使用由标准指令规定的缺省行为，且理解的新指令将识别为对标准指令相关需求的修改。通过这种方式，扩展 cache-control 指令能够完成，而不会要求修改基本协议。

该扩展机制取决于 HTTP 缓存服从其本地 HTTP 版本定义的所有 cache-control 指令，服从明确的扩展，并乎略不认识的所有指令。

例如，考虑一个假设的称为 community 的新响应指令，拌演为对 private 指令的修饰符。我们定义该新指令意思是，特别对任何非共享缓存，任何只以以其值命名的 community 的成员间共享的缓存可以缓存该响应。原始服务器希望允许 URI community 使用其它 private 响应在它们共享的缓存中可包括来做到

Cache-Control: private, community="UCI"

看见该头部域的缓存将正确行为，即使缓存不认识 community cache-extension，因为它还将看见并理解 private 指令和其缺省的安全行为。

不认识的 cache-directive【必须】被乎略，假设任何不被 HTTP/1.1 缓存认识的 cache-directive 似乎将与标准指令组合（或者响应的缺省可缓存性），因此缓存行为将保持最低的正确性，即使缓存不认识该扩展。

14.10 Connection

Connection general-header 域允许发送方指定希望特定连接的选项，且【禁止】由代理在将来的连接中通迅。

Connection 头部有如下语法：

```
Connection = "Connection" ":" 1#(connection-token)
connection-token = token
```

HTTP/1.1 代理【必须】在消息转发着解析 Connection 头部域，对于在该域中的每个 connection-token，删除消息中的与 connection-token 有相同名称的任何头部域。Connection 选项由 Connection 头部域中存在的 connection-token 来指示，而不是由任何相应的其它头部域，因为如果没有参数与该连接选项相关，其它头部域就不可以发送。

在 Connection 头部中所列的消息头部【禁止】包括端到端头部，如 Cache-Control。

HTTP/1.1 为发送者定义“close”连接选项以表示连接将会在完成响应后关闭。例如，

Connection: close

在请求或响应头部域是指出该连接【不该】在当前请求/响应完成后被认为是“永久的”（8.1 节）。

不支持永久连接的 HTTP/1.1 应用程序【必须】在每个消息中包括“close”连接选项。

收到包括 Connection 头部的 HTTP/1.0（或更低版本）消息的系统【必须】把该域中的每个 connection-token 删除且乎略与 connection-token 同名的消息中的任何头部域。该保护防止早期 HTTP/1.1 代理错误转发该类头部域。见 19.6.2 节。

14.11 Content-Encoding

Content-Encoding entity-header 域用作对 media-type 的修饰符。当存在时，它的值指出应用到 entity-body 上的额外内容编码，且必须应用这种解码机制，以便获得由 Content-Type 头部域引用的 media-type。Content-Encoding 主要用来允许文档被压缩，而不丢失其下层媒体类型的标识。

Content-Encoding = "Content-Encoding" ":" 1#content-coding

content-coding 在 3.5 节中定义。其用法的一个例子是

Content-Encoding: gzip

content-coding 由 Request-URI 所标识实体的特征。一般，entity-body 与该编码一起存储，且只在显示或类似的用法时解码。然而，非透明代理【可以】修改其 content-coding，如果新编码已知会被接收方接受，除非“no-transform”缓存控制指令存在于消息中。

如果实体的 content-coding 不是“identity”，这时响应【必须】包括 Content-Encoding entity-header（14.11 节），它列出所使用的非 identity content-coding。

不果请求消息中的实体的 content-coding 不被原始服务器接受，该服务【应该】以状态码 415（Unsupported Media Type）响应。

若多个编码应用到某个实体，则 conent-coding【必须】以它们所应用的顺序列出。关于编码参数的额外信息【可以】由其它本规范没有定义的 entity-header 域提供。

14.12 Content-Language

Content-Language entity-header 域描述目标观众对所封装实体的自然语言。要注意，这可能不与用于 entity-body 内的所有语言相同。

Content-Language = "Content-Language" ":" 1#language-tag

language-tag 在 3.10 节定义。Content-Langauge 的主要用途是允许用户按用户自己所选语言来标识并区别实体。因此，若主体内容只用于丹麦语学者观众，则适当的域是

Content-Language: da

如果没有指定 Content-Language，则该内容的缺省是用于所有语言观众。这可能意味着，发送方不认为它对任何自然语言是特殊的，或者发送方不知道用于哪个语言。

多语言内容【可以】被列出，该内容用于多个观众。例如，“Treaty of Waitanqi”表演，同时表演为原始毛利语和英语版本，将称为

Content-Language: mi, en

然而，只因多语言表示在一个实体中不意味着它用于多种语言的听众。一个例子是初学者的语言初级读本，如“A First Lesson in Latin”，明显用于英语语言观众。在这种情况下，Content-Language 将正确地只包括“en”。

Content-Language【可以】应用到任何媒体类型——它不限制文本文档。

14.13 Content-Length

Content-Length entity-header 域指出 entity-body 的尺寸，按十进制数的字节流，发送给接收方，或在 HEAD 方法的情况下，可能已经发送的 entity-body 的尺寸与 GET 请求相同。

Content-Length = "Content-Length" ":" 1*DIGIT

一个例子是

Content-Length: 3495

应用程序【应该】使用该域来指出 message-body 的 transfer-length，除非这被 4.4 节中的规则禁止。

大于等于 0 的任何 Content-Length 都是有效值。如果没有给出 Content-Length，4.4 节描

述如何决定 message-body 的长度。

要注意，该域的意思是 MIME 中相应的定义明确不同的，MIME 中它是可选域，用在“message-external-body”的 content-type 中。在 HTTP 中，它【应该】被发送，无论何时消息的长度能够先于被传输前决定，除非它被 4.4 节中的规则禁止。

14.14 Content-Location

Content-Location entity-header 域【可以】用于提供消息封装的实体的资源位置，当实体从独立于所请求资源的 URI 的位置可访问时。服务器【应该】提供与响应实体对应的 Content-Location 变量，特别在这种情况下，即资源有与之相关的多个实体的地方，且这些实体实际有独自的位置，通过这些位置，它们可能被独立访问，服务器【应该】为所返回的特别变量提供 Content-Location。

Content-Location = "Content-Location" ":" (absoluteURI | relativeURI)

Content-Location 的值还定义实体的基本 URI。

Content-Location 值不是原始请求 URI 的替代者；它只请求同时与该特别实体对应资源的位置的声明。将来请求【可以】指定 Content-Location 的 URI 作为 Request-URI，如果希望标识该特别实体源。

缓存器不能假设包括与用于获取的 URI 不同的 Content-Location 的实体能够用来响应以后的请求，用该 Content-Location URI。然而，Content-Location 能够用来区分从单个请求资源获取的多个实体，如 13.6 节所述。

如果 Content-Location 是 relativeURI，则该 relativeURI 相对于 Request-URI 相对解释。

PUT 或 POST 请求中的 Content-Location 意义未定；在该情况下，服务器可自由乎略它。

14.15 Content-MD5

Content-MD5 entity-header 域，如 RFC 1864 [23]中定义，是 entity-body 的 MD5 摘要，其用途是提供对 entity-body 的端到端消息完整性检查（MIC）。（注意：MIC 对于检测在传输中对 entity-body 的意外修改很好，但未经证实可防卫恶意攻击。）

Content-MD5 = "Content-MD5" ":" md5-digest
md5-digest = <如同 RFC 1864 的 128 位 MD5 摘要的 base64 编码>

Content-MD5 头部域【可以】由原始服务器或客户端生成，实现对 entity-body 的完整性检查的功能。只有原始服务器或客户端【可以】生成 Content-MD5 头部域；代理和网关【禁止】生成它，因为这将防碍它作为端到端完整性检查的价值。任何 entity-body 的接收方，包括网关和代理，【可以】检查该头部域中的摘要值是否匹配所收到的 entity-body。

MD5 摘要按 entity-body 的内容计算，包括任何已经应用的 content-coding，但不包括任何应用到 message-body 的 transfer-encoding。如果消息与 transfer-encoding 同时收到，该编码【必须】在检查 Content-MD5 值与所接收实体前被删除。

有这样的结果，即摘要按 entity-body 的确切字节流计算，且按它们将发送的顺序，若没有应用 transfer-encoding。

HTTP 扩展 RFC 1864 以允许为 MIME 复合 media-type（如，multipart/*和 message/rfc822）计算摘要，但这不改变如何按前述段落的定义来计算摘要。

有几个与此相关的结果。复合类型的 entity-body【可以】包含许多 body-part，每个有它自己的 MIME 和 HTTP 头部（包括 Content-MD5、Content-Transfer-Encoding 和 Content-Encoding 头部）。如果 body-part 有 Content-Transfer-Encoding 或 Content-Encoding 头中，假设 body-part 的内容已经应用编码，且 body-part 似乎包括在 Content-MD5——如，在应用程序后。Transfer-Encoding 头部域不允许在 body-part 中。

转换所有断行符为 CRLF【禁止】在计算或检查摘要前完成：用在实际传输文本中的断行符转换【必须】在计算摘要是保持未改变。

注意：HTTP 的 Content-MD5 的定义与 MIIE entity-body 的 RFC1864 中的几乎相同，同时在几个方面 HTTP entity-body 的 Content-MD5 的应用程序与 MIME entity-body 的应用程序不同。一方面是，HTTP，不像 MIME，不使用 Content-Transfer-Encoding，而后者不使用 Transfer-Encoding 和 Content-Encoding。另一方面是，HTTP 比 MIME 更常使用二进制内容类型，所以在该情况下，无所谓，用于计算摘要的字节序是该类型定义的传输字节序。最后一点是，HTTP 允许以任何几个断行符鳌来传输 text 类型，而非只使用 CRLF 的正式形式。

14.16 Content-Range

Content-Range entity-header 与部分 entity-body 一起发送，以指定该部分主体应该应用到全部 entity-body 的哪个地方。Range 单位在 3.12 节中描述。

```
Content-Range = "Content-Range" ":" content-range-spec
content-range-spec = byte-content-range-spec
byte-content-range-spec = bytes-unit SP byte-range-resp-spec "/" ( instance-length | "*" )
byte-range-resp-spec = (first-byte-pos "-" last-byte-pos) | "*"
instance-length = 1*DIGIT
```

该头部【应该】指出全部 entity-body 的部长度，除非该长度未知或很难决定。星号“*”符意味着 instance-length 在响应生成时刻是未知的。

不像 byte-ranges-specifier 值（见 14.35.1 节），byte-range-resp-spec【必须】只指定 1 个范围，且【必须】包含关于范围前面和后面的绝对字节位置。

byte-content-range-spec 与 byte-range-resp-spec 一起，其 last-byte-pos 值小于它的

first-byte-pos 值，或其 instance-length 值小于等于它的 last-byte-pos 值，都是有效的。无效 byte-content-range-spec 的接收方【必须】乎略它和与之一起传输的任何内容。

发送状态码 416（Requested range not satisfiable）响应的服务器【应该】包括有“*”的 byte-range-resp-spec 的 Content-Range 域。该 instance-length 指定所选择资源的当前长度。状态码 206（Partial Content）的响应【禁止】包括有“*”的 byte-range-resp-spec 的 Content-Range 域。

byte-content-range-spec 值的例子，假设实体包含总共 1234 字节：

——首个 500 字节：bytes 0-499/1234

——次个 500 字节：bytes 500-999/1234

——除了首个 500 字节的全部：bytes 500-1233/1234

——最后 500 字节：bytes 734-1233/1234

当 HTTP 消息包括单个范围的内容（例如，对单个范围的请求的响应，或对范围集覆盖且没有洞的请求的响应）时，该内容与 Content-Range 头部一起传输，且 Content-Lenght 头部显示实际传输的字节数。例如，

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

当 HTTP 消息包括多个范围的内容（例如，对于多个非覆盖范围的请求的响应）时，这些作为 multipart 消息传输。用于该用途的 multipart 媒体类型是“multipart/byteranges”，如附录 19.2 中所定义。兼容问题见附录 19.6.3。

对单个范围请求的响应【禁止】使用 multipart/byteranges 媒体类型发送。对多范围请求的响应，其结果是单范围，【可以】作为单个的 multipart/byteranges 媒体类型发送。不能解码 multipart/byteranges 消息的客户端【禁止】在单个请求中要求多个 byte-range。

当客户端在单个请求中请求多个 byte-range 时，服务器【应该】按它们出现在请求中的顺序返回它们。

如果服务器乎略 byte-ranges-spec，因它在语句上无效，则服务器【应该】对待该请求如同该无效 Range 头部域不存在。（通常，这意味着返回包含完整实体的 200 响应）。

如果服务器收到有不满意 Range 请求头部域（即，其所有的 byte-range-spec 值的

first-byte-pos 值大于所选择资源的当前长度)的请求(而非包括 If-Range 请求头部域的请求)，它【应该】返回响应码 416（Requested range not satisiable）（10.4.17 节）。

注意：对于不满足的 Range 请求头部，客户端不能决定服务器发送 416（Requested range not satisfiable）响应来代理 200（OK）响应，因为并非所有服务器实现该请求头部。

14.17 Content-Type

Content-Type entity-header 域指出发送给接收方的 entity-body 的媒体类型，在使用 HEAD 方法的情况下，或指出 GET 请求将发送的媒体类型。

Content-Type = "Content-Type" ":" media-type

媒体类型在 3.7 节中定义。该域的一个例子是

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

关于标识实体的媒体类型的方法的进一步讨论在 7.2.1 节中提供。

14.18 Date

Date general-header 域表示消息发生的日期和时间，与 RFC 822 中的 orig-date 有相同的语义。该域值是 HTTP-date，如 3.3.1 节所述；它【必须】以 RFC 1123 [8]的日期格式发送。

Date = "Date" ":" HTTP-date

一个例子是

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

原始服务器【必须】包括 Date 头部域在所有响应中，除这些情况外：

1、如果响应状态码是 100（Continue）或 101（Switching Protocols），则响应【可以】包括 Date 头部域，由服务器选择。

2、如果响应状态码传达服务器错误，如，500（Internel Server Error）或 503（Service Unavailable），且生成有效的 Date 有困难或不可能。

3、如果服务器没有时钟，它不能提供合近似的当前时间，则它的响应【禁止】包括 Date 头部域。在该情况下，【必须】遵守 14.18.1 节中的规则。

所收到的没有 Date 头部域的消息【必须】由接收方指定一个，若该消息将由接收方缓存，或通过要求有 Date 的协议的网关。没有时钟的 HTTP 实现在每次使用时【禁止】在没有重证实它们时缓存响应。HTTP 缓存器，特别是共享缓存器，【应该】使用机制，如 NTP [28]，

来同步其时钟与可靠的外部标准。

客户端【应该】只在包括 entity-body 的消息发送 Date 头部域，如 PUT 和 POST 请求的情况下，且即使它是可选的时候。没有时钟的客户端【禁止】在请求中发送 Date 头部域。

在 Date 头部中发送的 HTTP-date【不该】表示生成消息之后的日期和时间。它【应该】表示消息生成的最可能精确的日期和时间，除非实现没有办法生成相当精确的日期和时间。在理论上，日期应该表示实体刚生成前的时刻。实际上，日期可能在消息生成的任何时候生成，而不影响它的语义值。

14.18.1 无时钟原始服务器的操作

一些原始服务器实现可能没有有用的时钟。无时钟的原始服务器【禁止】为响应指定 Expires 或 Last-Modified 值，除非这些值与系统或有可靠时钟的用户的资源相关。它【可以】分配已知过去的（这允许响应“预截止”，而无需为每个资源存储单独的 Expires 值）Expires，在服务器配置时刻或之前。

14.19 ETag

ETag response-header 域提供所请求变量的实体标签的当前值。与实体标签一起使用的该头部在 14.24、14.26 和 14.44 节中描述。该实体标签【可以】用来比较来自相同资源的其它实体（见 13.3.3 节）。

ETag = "ETag" ":" entity-tag

例子：

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

14.20 Expect

Expect request-header 域用于指出客户端要求的特殊服务器行为。

Expect = "Expect" ":" 1#expectation
expectation = "100-continue" | expectation-extension
expectation-extension = token ["=" (token | quoted-string) *expect-params]
expect-params = ";" token ["=" (token | quoted-string)]

不理解或不能顺从请求的 Expect 域中的任何希望值的服务器【必须】用适当的错误状态响应。该服务器【必须】响应 417（Expectation Failed）状态，若不能满足任何希望，如果请求没有其它问题，则响应一些其它 4xx 状态。

该头部域定义扩展语法以允许将来扩展。如果服务器收到包含 Expect 域的请求，且其中包括不支持的 expectation-extension，则它【必须】以 417（Expectation Failed）状态响应。

希望值的比较对于非引用符号（包括 100-continue 符号）是大小写不敏感的，且对于 quoted-string 的 expectation-extension 是大小写敏感的。

Expect 机制是跳对跳；即，HTTP/1.1 代理【必须】返回 417（Expectation Failed）状态，若它收到有它不能满足的希望的请求。然而，Expect 请求头部自己是端对端；它【必须】被转发，若请求被转发。

许多老的 HTTP/1.0 和 HTTP/1.1 应用程序不理解 Expect 头部。

关于使用 100（Continue）状态见 8.2.3 节。

14.21 Expires

Expires entity-header 域给出响应被认为过期的日期/时间。过期缓存条目可能由缓存器（或者是代理缓存器或用户代理缓存器）非正常返回，除非它首先向原始服务器（或向有实体更新拷贝的中间缓存器）证实。关于截止模型的进一步讨论见 13.2 节。

存在 Expires 域不意味着原始资源将修改或停止存在，在该时刻或之前或之后。

其格式是绝对日期和时间，由 HTTP-date 在 3.3.1 节中定义；它【必须】按 RFC 1123 日期格式。

Expires = "Expires" ":" HTTP-date

其用法的一个例子是

Expires: Thu, 01 Dec 1994 16:00:00 GMT

注意：如果响应包括有 max-age 指令（见 14.9.3 节）的 Cache-Control 域，则该指令覆盖 Expires 域。

HTTP/1.1 客户端和缓存器【必须】对待其它无效日期格式为过时的（如，“已经截止”），特别包括值“0”。

要使响应“已经截止”，原始服务器发送与 Date 头部值相同的 Expires 日期。（关于截止计算见 13.2.4 节中的规则。）

要使响应“从不截止”，原始服务器发送响应发送时大约一年后的 Expires 日期。HTTP/1.1 服务器【不该】发送超过一年以后的 Expires 日期。

存在 Expires 头部域，其日期值是对其它情况下可能缺省是不可缓存的响应的将来的某

个时刻，则指出该响应可缓存，除非由 Cache-Control 头部域另行指出（14.9 节）。

14.22 From

From request-header 域，若给出，【应该】包含因特网邮件地址给正控制请求用户代理的人类用户。该地址【应该】是机器可用的，如 RFC 822 [9]，由 RFC 1123 [8]更新，中的“mailbox”所定义：

From = "From" ":" mailbox

一个例子是：

From: webmaster@w3.org

该头部域【可以】用来登录用途，且对标识无效或非希望请求的源有意义。它【不该】用作访问保护的非安全形式。对该域的解释是，请求执行人给出的行为，他对所执行方法负责。特别是，机器代理【应该】包括该头部，以便对运行机器负责人类能够被联系，如若在接收端发生问题时。

在该域中的因特网邮件地址【可以】与引起请求的因特网主机相分离。例如，请求穿过代理时【应该】使用原始发行者的地址。

客户端在没有用户承认时【不该】发送 From 头部域，因它可能与用户的私密利益或它们的站点安全策略相冲突。强烈推荐用户能够在请求前的任何时刻取消、允许和修改该域的值。

14.23 Host

Host request-header 域规定正被请求的资源的因特网主机和端口号，如从由用户或正引用资源提供的原始 URI 获得（通常是 HTTP URL，如 3.2.2 节所述）。Host 域值【必须】表示原始服务器或网关由原始 URL 给出的权威名称。这允许原始服务器或网关与内部不明确的 URL 相区别，如有单个 IP 地址并有多个主机名称的服务器的根“/”URL。

Host = "Host" ":" host [":" port] ; 3.2.2 节

没有任何后跟 port 信息的“host”意味着使用所请求服务的缺省端口（如，HTTP URL 的“80”）。例如，对原始服务器<http://www.w3.org/pub/WWW/>的请求可正确包括：

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

客户端【必须】包括 Host 头部域于所有 HTTP/1.1 请求消息中。如果所请求 URI 没有包括对所请求服务器因特网主机名，这时 Host 头部域【必须】指定为空值。HTTP/1.1 代理【必须】确保，由代理转发的任何请求消息确实包含适当的 Host 头部域，该域标识正请求

的服务。所有基于因特网的 HTTP/1.1 服务器【必须】响应 400（Bad Request）状态码给任何缺少 Host 头部域的 HTTP/1.1 请求消息。

与 Host 相关的其它需求见 5.2 和 19.6.1.1 节。

14.24 If-Match

If-Match request-header 域与方法一起用于使其条件化。客户端以前从该资源获取的一个或多个实体能够检验这些实体的其中之一是当前的，通过包括它们相关的实体标签的列表于 If-Match 头部域中。裉标签在 3.11 节中定义。该特性的用途是允许有效地更新缓存信息，以最少量的上层事务代价。它还用于更新请求，以阻止对资源的错误版本的意外修改。任何特殊情况，值“*”匹配该资源的任何当前实体。

If-Match = "If-Match" ":" ("*" | 1#entity-tag)

若任何实体标签与对对该资源的相似 GET 请求（没有 If-Match 头部）的响应中返回的实体的实体标签相匹配，或者如果给出“*”，且该资源的任何当前实体存在，这时服务器【可以】执行该请求方法，如同 If-Match 头部域不存在。

服务器【必须】使用强比较功能（见 13.3.3 节）来比较 If-Match 中的实体标签。

如果没有实体标签匹配，或如果“*”被指定，且无当前实体存在，服务器【禁止】执行请求方法，且【必须】返回 412（Precondition Failed）响应。该行为是最有用的，当客户端希望阻止更新方法，如 PUT，去修改客户最后获取它以来已经改变的资源。

若没有 If-Match 头部域，请求导致任何其它非 2xx 或 412 状态，这时 If-Match 头部【必须】被乎略。

“If-Match: *”的意思是方法【应该】被执行，若由原始服务器（或由缓存器，可能使用 Vary 机制，见 14.44 节）选择的表述存在，且【禁止】执行，若表述不存在。

意图更新资源的请求（如，PUT）【可以】包括 If-Match 头部域来发出信号，即该请求方法【禁止】被应用，若与 If-Match 值（单个实体标签）对应的实体不在是该资源的表述。这允许用户指出他们不希望请求成功，若资源已经被修改且不被他们所知。例如：

If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *

本规范未定义包括 If-Match 头部域和 If-None-Match 或 If-Modified-Since 头部域之一的请求的结果。

14.25 If-Modified-Since

If-Modified-Since request-header 域用于方法使其条件化如果所请求的变量在本域指定的时间以来不曾修改过，则实体将不会从服务器返回；取而代之，304（Not Modified）响应将返回，而没有任何 message-body。

If-Modified-Since = "If-Modified-Since" ":" HTTP-date

该域的一个例子是：

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

仅当从 If-Modified-Since 头部给出的日期以来已经修改过，有 If-Modified-Since 头部，且没有 Range 头部的 GET 方法请求所标识的实体才会被传输。判断的算法包括如下情况：

a）如果请求将正常导致非 200（OK）的任何其它状态，或如果传输的 If-Modified-Since 日期是无效的，则响应与正常 GET 几乎相同。比服务器的当前时间晚的日期是无效的。

b）如果该变量自 If-Modified-Since 日期以来已经修改过，则响应同正常 GET 的几乎相同。

c）如果该变量自 If-Modified-Since 日期以以来未曾修改过，则服务器【应该】返回 304（Not Modified）响应。

该特性的用途是允许以最少量的事务代价来有效地更新缓存信息。

注意：Range request-header 域修改 If-Modified-Since 的意义；完整细节见 14.35 节。

注意：If-Modified-Since 时间由服务器解释，其时钟可能与客户端的不同步。

注意：当处理 If-Modified-Since 头部域时，一些服务器将使用精确日期比较功能，而非小于功能，来判断是否发送 304（Not Modified）响应。为取得为缓存证实发送 If-Modified-Since 的最佳结果，建议客户端尽可能使用以前 Last-Modified 头部域所收到的精确日期字符串。

注意：如果客户端在 If-Modified-Since 头部域中使用任意日期，而不是相同请求中获得的 Last-Modified 头部的日期，则客户端应该清楚该事实，即该时期由服务器对时间的理解来解释。客户端应该意识到非同步时钟和因客户端与服务器间不同时间编码造成的舍入问题。这包括空条件的可能性，如果该文在它首次被请求的时间与后序请求的 If-Modified-Since 日期间已经修改过，和时钟不准确问题的可能性，如果 If-Modified-Since 日期由客户端的未经校正的时钟到服务器的时钟。在客户端与服务器间校正不同时间基础因网络延迟只是极近似值。

本规范没有定义这类请求的结果，即其中既有 If-Modified-Since 头部域，又有 If-Match 或 If-Unmodified-Since 头部域之一。

14.26 If-None-Match

If-None-Match request-header 域与方法一起用来使它条件化。客户端有一个或多个以前从资源获取的实体，它能够检验这些实体中没有当前的，通过包括它们相关的实体标签清单在 If-None-Match 头部域中。该特性的用途是允许以最少量的事务代价来高效更新缓存信息。它还用于阻止方法（如，PUT）意外修改现存资源，当客户端认识该资源不存在时。

作为特殊情况，值“*”匹配任何资源的当前实体。

If-None-Match = "If-None-Match" ":" ("*" | 1#entity-tag)

如果任何实体标签匹配对该资源的相似 GET 请求（无 If-None-Match 头部）的响应中返回的实体的实体标签，或如果“*”给出且该资源的任何当前实体存在，这时服务器【禁止】执行所请求的方法，除非因资源的修改日期不能匹配请求中的 If-Modified-Since 头部域中所提供的而要求这样做。取而代之，如果请求方法是 GET 或 HEAD，则服务器【应该】响应以 304（Not Modified）响应，包括匹配实体的一个缓存相关的头部域（特别是 ETag）。对于所有其它请求方法，服务器【必须】响应以状态 412（Precondition Failed）。

见 13.3.3 节的对于如何判断两个实体标签是否匹配的规则。强比较功能只能用于 GET 或 HEAD 请求。

如果没有实体标签匹配，这时服务器【可以】挂靠该请求的方法，如同 If-None-Match 头部域不存在，但【必须】也乎略请求中的任何 If-Modified-Since 头部域。即，如果没有实体标签匹配，这时服务器【禁止】返回 304（Not Modified）响应。

如果无 If-None-Match 头部域的请求导致任何非 2xx 或 304 状态，这时 If-None-Match 头部【必须】被乎略。（见 13.3.4 节关于当在相同请求中同时出现 If-Modified-Since 和 If-None-Match 的服务器行为的讨论。）

“If-None-Match: *”的意思是，该方法【禁止】被执行，如果由原始服务器（或由缓存器，可能使用 Vary 机制，见 14.44 节）选择的表述存在，且【应该】被执行，若表述不存在。该特性可有效阻止 PUT 间的空转。

例子：

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: *

本规范没有定义这样请求的结果，即既有 If-None-Match 头部域，又有 If-Match 或 If-Unmodified-Since 头部域之一。

14.27 If-Range

若客户端在其缓存中在某个实体的部分拷贝，且希望其缓存中有完整实体的最新拷贝，它能够使用 Range request-header 的条件化 GET（使用之一或全部 If-Unmodified-Since 和 If-Match。）然而，如果条件失败，因为实体已经修改，客户端这时将必须进行第二次请求来获取整个当前 entity-body。

If-Range 头部允许客户端“短路”第二次请求。非正式地，其意思是，“若实体没有修改，给我发送我缺少的部分，否则，给我发送完整的新实体”。

If-Range = "If-Range" ":" (entity-tag | HTTP-date)

若客户端没有实体的实体标签，但确有 Last-Modified 日期，则它【可以】在 If-Range 头部中使用该时期。（服务器能够检查不超过 2 个字符就区分出有效的 HTTP-date 和任何形式的 entity-tag。）If-Range 头部【应该】只与 Range 头部一起使用，且【必须】被乎略，若请求中不包括 Range 头部，或若服务器不支持子范围操作。

若 If-Range 头部中给出的实体标签匹配该实体的当前实体标签，这时服务器【应该】使用 206（Partial Content）响应来提供实体的指定子范围。如果实体标签不匹配，这时服务器【应该】使用 200（OK）响应返回整个实体。

14.28 If-Unmodified-Since

If-UnModified-Since request-header 域与方法一起用于使其条件化。如果请求资源自该域中指定的时间以来未曾修改过，则服务器【应该】执行该请求操作，如同 If-Unmodified-Since 头部不存在。

若自指定时间以来所请求变量已经修改过，则服务器【禁止】执行所请求操作，且【必须】返回 412（Precondition Failed）。

If-Unmodified-Since = "If-Unmodified-Since" ":" HTTP-date

该域的一个例子是：

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

若请求正常（如，无 If-Unmodified-Since 头部）导致任何非 2xx 或 412 状态，则 If-Unmodified-Since 头部【应该】被乎略。

如果指定日期是无效的，则该头部被乎略。

本规范没有定义这种请求的结果，即既有 If-Unmodified-Since 头部域，又有 If-None-Match 或 If-Modifiec-Since 头部域之一。

14.29 Last-Modified

Last-Modified entity-header 域指出原始服务器认为该变量最后修改的日期和时间。

Last-Modified = "Last-Modified" ":" HTTP-date

其用法的一个例子是

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

该头部域的确切意思取决于原始服务器的实现和原始资源的属性。对于文件，它可能只是文件系统的最后修改时间。对于有动态包括部分的实体，它可能是其组成部分的最后修改时间的最近设置值。对于数据库网关，它可能是该记录的最后更新时间戳。对于虚拟对象，它可能是内部状态修改的最后时间。

原始服务器【禁止】发送比服务器的消息发源时间更晚的 Last-Modified 日期。在这种情况下，该资源的最后修改地将指出以后的一些时间，服务器【必须】以消息发源日期代替该日期。

原始服务器【应该】获取尽量靠近它生成其响应的 Date 值的时间的实体的 Last-Modified 值。这允许接收方精确指定实体修改时间，特别当实体在响应生成的几乎同时被修改。

HTTP/1.1 服务器【应该】在任何可能时发送 Last-Modified。

14.30 Location

Location response-header 域用来重定向接收方到非 Request-URI 的位置来完成请求或标识新的资源。对于 201（Created）响应，Location 是由请求所创建的新资源的标识。对于 3xx 响应，位置【应该】指出服务器用于自动重定向资源所选择的 URI。该域值由单个绝对 URI 组成。

Location = "Location" ":" absoluteURI

一个例子是：

Location: http://www.w3.org/pub/WWW/People.html

注意：Content-Location 头部域（14.14 节）与 Location 不同，Content-Location 标识请求中封装的实体的原始位置。因此，响应中既包括 Location 头部域又包括 Content-Location 头部域而言是可能的。关于一些方法的缓存需求另见 13.30 节。

14.31 Max-Forwards

Max-Forwards request-header 域提供某种机制，TRACE（9.8 节）和 OPTIONS（9.2 了）方法用来限制可以转发该请求给下个入界服务器的代理或网关的数量。这可能用于当客户端

试图跟踪请求链，似乎将在链中间失败或循环。

Max-Forwards = "Max-Forwards" ":" 1*DIGIT

Max-Forwards 值是十进制整数指出该请求消息可能被转发的余下次数。

包含 Max-Forwards 头部域的 TRACE 或 OPTIONS 请求的每个代理或网关接收方【必须】在转发该请求前检查且更新其值。若所收到的值是 0，则接收方【禁止】转发该请求；妈而代之，它【必须】按最终接收方响应。若收到的 Max-Forwards 值大于 0，这时所转发的消息【必须】包含其值减 1 的更新的 Max-Forwards 域。

Max-Forwards 头部域【可以】被本规范中定义的所有其它方法乎略，且被没有明确引用其作为该方法定义的一部分的任何扩展方法乎略。

14.32 Pragma

Pragma general-header 域用来包括实现特定的指令，它可能应用到请求/响应链上的任何接收方。所有 Pragma 指令指出协议观点上的可选行为；然而，一些系统【可以】要求该行为与该指令一致。

Pragma = "Pragma" ":" 1#pragma-directive
pragma-directive = "no-cache" | extension-pragma
extension-pragma = token ["=" (token | quoted-string)]

当在请求消息中存在 no-cache 指令时，应用程序【应该】直接向原始服务器转发该请求，即使它已经缓存有所请求的一分拷贝。该 Pragma 指令有与 no-cache 缓存指令相同的语义（见 14.9 节），且在这里定义以与 HTTP/1.0 向后兼容。客户端【应该】包括全部两种头部域，当 no-cache 请求发送给不知与 HTTP/1.1 一致的服务器时。

Pragma 指令【必须】穿过代理或网关应用程序，不管它们对该应用程序是否有意义，因为该指令可能被应用到请求/响应链上的所有接收方。不可能为特殊接收方指定 Pragma；然而，不与接收方法相关的任何 Pragma 指令【应该】被该接收方乎略。

HTTP/1.1 缓存【应该】对待“Pragma: no-cache”如同客户端已经发送“Cache-Control: no-cache”。HTTP 中将不会定义新的 Pragma 指令。

注意：因为作为响应头部域的“Pragma: no-cache”意思没有真正指定，不提供可靠替换为响应中的“Cache-Control: no-cache”的操作。

14.33 Proxy-Authenticate

Proxy-authenticate response-header 域【必须】包括作为 407（Proxy Authentication Required）响应的一部分。该域值由 challenge 组成，它指出认证方案和可应用到代理的该 Request-URI 上的参数。

Proxy-Authenticate = "Proxy-Authenticate" ":" 1#challenge

HTTP 访问认证过程在“HTTP Authentication: Basic and Digest Access Authentication”[43]中描述。不像 WWW-Authenticate，Proxy-Authenticate 头部域只应用到当前连接上，且【不该】传递到下游客户端。然而，中间代理可能需要获取它自己的 credentials，通过从下游客户端请求它们，在某些情况下它将出现，如同代理转发该 Proxy-Authenticate 头部域。

14.34 Proxy-Authorization

Proxy-Authorization request-header 域允许客户端向代理标识它自己（或其用户），该代理需要认证。Proxy-Authorization 域值由 credentials 组成，其中包含用户代理对代理的认证信息和/或所请求资源的 realm。

Proxy-Authorization = "Proxy-Authorization" ":" credentials

HTTP 访问认证过程在“HTTP Authentication: Basic and Digest Access Authentication”[43]中描述。与 Authorization 不同，Proxy-Authorization 头部域只应用到下个出界代理，它要求使用 Proxy-Authenticate 域认证。当多个代理用在某个链路中时，Proxy-Authorization 头部域由首个出界代理消费，该代理希望接收 credentials。代理【可以】从客户端到下个代理接力 credentials，若这时代理协作认证给定请求的机制。

14.35 Range

14.35.1 字节范围

因为所有 HTTP 实体在 HTTP 消息中表示为字节序列，字节范围的概念对任何 HTTP 实体都是有意义的。（然而，并非所有客户端和服务器需要支持字节范围操作。）

HTTP 中规定的字节范围应用到 entity-body（不要求与 message-body 相同）的字节序列。

字节范围操作【可以】指定单个实体中的单个字节范围、或范围集。

ranges-specifier = byte-ranges-specifier
byte-ranges-specifier = bytes-unit "=" byte-range-set
byte-range-set = 1#(byte-range-spec | suffix-byte-range-spec)
byte-range-spec = first-byte-pos "-" [last-byte-pos]
first-byte-pos = 1*DIGIT
last-byte-pos = 1*DIGIT

byte-range-spec 中的 first-byte-pos 值给出范围的首字节的字节位移。last-byte-pos 值给出范围的最后字节的字节位移；即，指定所包括的字节位置。字节位移从 0 开始。

如果存在 last-byte-pos 值，它【必须】大于等于该 byte-range-spec 中的 first-byte-pos，

要么该 byte-range-spec 就是句法无效的。包括一个或多个句法幸无效 byte-range-spec 值的 byte-range-set 的接收方法【必须】乎略包括该 byte-range-set 的头部域。

如果缺少 last-byte-pos 值，或如果该值大于或等于 entity-body 的当前长度，则 last-byte-pos 取比 entity-body 的当前长度小 1 的字节数。

通过选择 last-byte-pos，客户端可能限制所接收的字节数，而不需知道实体的长度。

suffix-byte-range-spec = "-" suffix-length
suffix-length = 1*DIGIT

suffix-byte-range-spec 用来指定 entity-body 的前缀，其长度由 suffix-length 值给出。（即，该形式指定 entity-body 的最后 N 个字节。）如果实体比指定的 suffix-length 短，则使用全部 entity-body。

如果句法有效的 byte-range-set 包括至少 1 个 byte-range-spec，其 first-byte-pos 比 entity-body 的当前长度小，或至少 1 个 suffix-byte-range-spec 有非 0 的 suffix-length，这时该 byte-range-set 是可满足的。否则，该 byte-range-set 就是不可满足的。如果 byte-range-set 不可满足，则服务器【应该】返回响应状态 416（Request Range Not Satisfiable）。否则，服务器【应该】返回响应状态 206（Partial Content），其中包含可满足的 entity-body 范围。

byte-ranges-specifier 值的例子（假设 entity-body 长度为 10000）：

——首 500 字节（字节位移 0—499，包含）：bytes=0-499

——次 500 字节（字节位移 500-599，包含）：bytes=500-999

——最后 500 字节（字节位移 9500-9999，包含）：bytes=-500 或 bytes=9500-

——首个和最后一个字节（字节 0 和 9999）：bytes=0-0,-1

——次 500 字节的几个合法但非正式的规范（字节位移 500-999，包含）：bytes=500-600,601-999 或 bytes=500-700,601-999

14.35.2 范围获取请求

HTTP 获取请求使用条件化或非条件 GET 方法，【可以】请求 1 个或多个实全的子范围，而非整个实体，使用 Range 请求头部，将其应用于所返回的实体，以作为该请求的结果：

Range = "Range" ":" ranges-specifier

服务器【可以】乎略 Range 头头部。然而，HTTP/1.1 原始服务器和中间缓存器应该支持字节范围，当可能时，因为 Range 支持高效地从部分传输失败中恢复，且支持高效地巨大实体的部分获取。

如果服务器支持 Range 头部且指定的范围对实体是适当的：

——在无条件 GET 中存在在 Range 头部修改其返回内容，若 GET 会成功。换句话说，响应挟带状态码 206（Partial Content），而非 200（OK）。

——条件 GET（使用 If-Modified-Since 或 If-None-Match 之一或 2 个都使用，使用 If-Unmodified-Since 或 If-Match 之一或 2 个都使用）中存在 Range 头部修改所返回的内容，若 GET 将成功且条件为真。它不影响返回 304（Not Modified）响应，若条件为假。

在某些情况下，使用 If-Range 头部（见 14.27 节）代替 Range 头部可能更恰当。

若支持范围的代理收到以 Range 请求，转发该请求给入界服务器，且在回复中收到整个实体，则它【应该】只返回所请求的范围给其客户端。它【应该】在其缓存中保存整个所收到的实体，若与其缓存分配策略一致的话。

14.36 Referer

Referer[sic] request-header 域允许客户端指定，为服务器的利益，资源的地址（URI），从中获取 Request-URI（“referrer”，尽管头部域拼写错误。）Referer request-header 允许服务器为感兴趣的资源、日志、优化缓存、等生成向后链接清单。它还允许废弃或维护跟踪打错链接。Referer 域【禁止】被发送，若 Request-URI 从没有其自己 URI 的源获取，如用户键盘的输入。

Referer = "Referer" ":" (absoluteURI | relativeURI)

例子：

Referer: http://www.w3.org/hypertext/DataSources/Overview.html

若域值是相对 URI，它【应该】被解释为与 Request-URI 相对。该 URI【禁止】包括分片。安全考虑见 15.1.3 节。

14.37 Retry-After

Retry-After response-hedaer 域能够与 503（Service Unavailable）响应一起用于指出希望该服务对请求客户端可以维持有效多长。该域还【可以】与 3xx（）响应一起用于指出要求用户代理在重定向请求前应等待的最小时间。该域的值可以是响应时刻后的 HTTP-date 或整数秒数（十进制）。

Retry-After = "Retry-After" ":" (HTTP-date | delta-seconds)

其用法的 2 个例子是

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

在后个例子中，延迟是 2 分钟。

14.38 Server

Server response-header 域包含原始服务器处理请求所使用的软件的信息。该域能够包含多个 product 符号（3.8 节）和 comment，以标识服务器和任何重要的子产品。product 符号按标识该应用程序的重要性顺序列出。

Server = "Server" ":" 1*(product | comment)

例子：

Server: CERN/3.0 libwww/2.17

若响应通过代理转发，则代理应用程序【禁止】修改 Server response-header。取而代之，它【应该】包括 Via 域（如 14.45 节所述）。

注意：启示服务器的特定软件版本可能允许服务器主机更易受到攻击，因已知该软件包含安全漏洞。鼓励服务器实现者将该域作为可配置选项。

14.39 TE

TE request-header 域指出愿意在响应中接受任何扩展 transfer-coding，且是否愿意接受 chunked transfer-coding 中的尾巴域。其值可能由关键字“trailers”和/或逗号分隔的有可选接受参数（如 3.6 节所述）的扩展 transfer-coding 名的清单。

TE = "TE" ":" #(t-codings)
t-codings = "trailers" | (transfer-extension [accept-params])

存在关键字“trailers”指出该客户端愿意接受尾巴域于 chunked transfer-coding 中，如 3.6.1 节所述。该关键字保留给 transfer-coding 值使用，即使它自己不表示任何 transfer-coding。

其用法的例子是：

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

TE 头部域只应用到中间连接。因此该关键字【必须】在 Connection 头部域（14.10 节）中提供，无论何时 TE 在 HTTP/1.1 消息中存在时。

服务器按照 TE 域测试某个 transfer-coding 是否可接受，使用如下规则：

1、“chunked” transfer-coding 始终可接受。若关键字“trailers”列出，则客户端指出它愿意接受尾巴域于 chunked 响应中，按其自己和任何下游客户端的行为。这意味着，若给出，客户端声明全部下游客户端中的任何一个都愿意接受尾巴域于转发的响应中，或它愿意尝试以下游接收方的行为缓存响应。

注意：HTTP/1.1 没有定义限制 chunked 响应的大小的任何意思，因此，客户端能够确保缓存整个响应。

2、若正测试的 transfer-coding 是 TE 域中所列出的 transfer-coding 中的一个，这时它是可接受的，除非它伴随的 qvalue 为 0。（如 3.9 节所定义，qvalue 为 0 意味着“不可接受”。）

3、若多个 transfer-coding 都可接受，这时选择有最大非 0 qvalue 的可接受的 transfer-coding。“chunked” transfer-coding 始终有 qvalue 为 1。

若 TE 域值为空，或若 TE 域不存在，则唯一 transfer-coding 是“chunked”。无 transfer-coding 的消息始终是可接受的。

14.40 Trailer

Trailer 通用域值指出给出的头部域集合在以 chunked transfer-coding 编码的消息的尾巴中存在。

Trailer = "Trailer" ":" 1#field-name

HTTP/1.1 消息【应该】包括 Trailer 头部域在使用 chunked transfer-coding 且有非空尾巴的消息中。这样做将允许接收方知道在尾巴中希望有哪些头部域。

若不存在 Trailer 头部域，则该尾巴【不该】包括任何头部域。关于在“chunked” transfer-coding 中使用尾巴域的限制见 3.6.1 节。

在 Trailer 头部域中列出的消息头部域【禁止】包括如下头部域：

- Transfer-Encoding
- Content-Length
- Trailer

14.41 Transfer-Encoding

Transfer-Encoding general-header 域指出为了安全在发送方和接收方间传输，已经应用什么类型的转换（若有的话）到消息主体上。这与 content-coding 是不同的，其中，transfer-coding

是消息的属性，而非实体的。

Transfer-Encoding = "Transfer-Encoding" ":" 1#transfer-coding

transfer-coding 在 3.6 节中定义。一个例子是：

Transfer-Encoding: chunked

如果多个编码应用已经到某个实体，则 transfer-coding【必须】以它们被应用和顺序列出。关于编码参数的额外信息【可以】由本规范没有定义的其它 entity-header 域提供。

许多老的 HTTP/1.0 应用程序不理解 Transfer-Encoding 头部。

14.42 Upgrade

Upgrade general-header 允许客户端指定它所支持的其它通迅协议，且愿意使用之，若服务器发现切换协议是适当的。服务器【必须】使用 Upgrade 头部在 101（Switching Protocols）响应中，以指出将切换的协议。

Upgrade = "Upgrade" ":" 1#product

例如，

Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11

Upgrade 头部域希望为从 HTTP/1.1 转移到其它，非兼容协议提供简单机制。通过允许客户端告知它所希望使用的其它协议来达成目的，如 HTTP 的新版本，有更高的主要版本号，即使已经使用 HTTP/1.1 作出当前请求。这减轻在非兼容协议间转移的困难，通过卵客户端以更普遍支持的协议发起请求，同时指示服务器它愿意使用“更好的”协议，若可用的话（这里“更好的”是由服务器决定，可能依据方法的属性和/或被请求的资源）。

Upgrade 头部域只应用到切换应用层协议，基于已经存在的传输层连接。Upgrade 不能用于持续改变协议；它的可接受和使用对服务器是可选的。协议改变后的应用层通迅的能力属性完全由所选择的新协议决定，尽管修改协议后的首个行为【必须】是响应包含 Upgrade 头部域的初始 HTTP 请求。

Upgrade 头部域只应用到中间连接。因此，upgrade 关键字【必须】在 Connection 头部域中提供（14.10 节），不管何时 HTTP/1.1 消息中存在 Upgrade。

Upgrade 头部域不能用于指出在同连接上切换协议。为该用途，使用 301、302、303 或 305 重定向响应更适当。

本规范只定义协议名“HTTP”，由超文本传输协议簇使用，如 3.1 节的 HTTP 版本规则和将来对本规范的更新所定义。任何符号能够被用作协议名；然而，只有当客户端与服务器

都将该名称与相同的协议相关联时，它才是有用的。

14.43 User-Agent

User-Agent request-header 域包含发起请求的用户代理的信息。这用于统计用途、协议侵害跟踪和自动识别用户代理以裁剪响应避免特定用户代理的限制的好处。用户代理【应该】在请求中包括该域。该域能够包含多个 product 和 comment 符号（3.8 节），以标识该代理和任何子产品，它们形成用户代理的重要部分。一般约定，product 符号按标识该应用程序的重要程度的顺序列出。

User-Agent = "User-Agent" ":" 1*(product | comment)

例子：

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

14.44 Vary

Vary 域值指出 request-header 域的集合，它完全决定缓存器，若响应是更新的话，是否允许使用该响应来回复后序列请求，而不需重证实。对于不可缓存或过期响应，Vary 域值建议用户代理用于选择表述的标准。域值为“*”的 Vary 意思是，缓存器不能从后序请求的请求头部中判断出该响应是否是适当的表述。关于缓存器如何使用 Vary 头部域见 13.6 节。

Vary = "Vary" ":" ("*" | 1#field-name)

HTTP/1.1 服务器【应该】在从属于服务器驱动协商的任何可缓存响应中包括 Vary 头部域。这样做，将允许缓存器正确解释将来对该资源的请求，且提示用户代理存在对该资源的协商。服务器【可以】在从属于服务器驱动协商的某个不可缓存响应中包括 Vary 头部域，因为这可以为用户代理提供关于响应变化时刻的尺度的有用信息。

Vary 域值由 field-name 清单缓成，发信号表示，为响应所选择的表述是基于某个选择算法，在选择最适当表述中，它“只”认识所列出的 reuest-header 域值。缓存器【可以】假设，相同的选择将为以后的有所列域名的相同值的请求中作出，并持续响应还是更新的时间。

给出的 field-name 不限于由本规范所定的标准 request-header 域。域名是大小写不敏感的。

域值为“*”的 Vary 发信号表明，不限制为 request-header（如，客户端的网络地址）的非指定参数在响应表述选择中拌演某个角色。“*”值【禁止】由代理服务器生成，它只可能由原始服务器生成。

14.45 Via

Via general-header 域【必须】由网关或代理用来指出在请求中的用户代理和服务器、及

在响应中的原始服务器和客户端的中间协议和接收方。它与 RFC 822 [9]的“Received”域相似，且希望用于跟踪消息转发、避免请求循环、和在请示/响应链上所有发送方的协议能力。

```
Via = "Via" ":" 1#( received-protocol received-by [ comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
protocol-name = token
protocol-version = token
received-by = ( host [ ":" port ] ) | pseudonym
pseudonym = token
```

received-protocol 指出由请求/响应链上每段的服务器或客户端所收到消息的协议版本。received-protocll 版本附加到 Via 域值中，当该消息被转发时，因此关于上游应用程序的协议能力的信息保持对所有接收方可见。

protocol-name 是可选的，当且仅当它是“HTTP”时。received-by 域通常是后序列所转发的消息的接收方服务器或客户端的 host 和可选 port。然而，若真正的主机被认为是敏感信息的话，它【可以】替换为假名。若没给出 port，它【可以】被假设为 received-protocol 的缺省 port。

多个 Via 域值表示已经转发消息的每个代理或网关。每个接收方【必须】追回其信息，因此最终结果就是按转发应用程序的顺序排列的。

comment【可以】用在 Via 头部域中，以标识接收方代理或网关的软件，与 User-Agent 和 Server 头部域相似。然而，Via 域中的全部 comment 都是可选的，且【可以】由任何接收方在转发消息前删除。

例如，请求消息可能从 HTTP/1.0 用户代理发送给编码为“fred”的内部代理，它使用 HTTP/1.1 转发该请求给在 nowhere.com 的公共代理，该代理通过转发给位于 www.ics.uci.edu 的原始服务器来完成请求。由 www.ics.uci.edu 接收的请求这时将有如下 Via 头部域：

```
Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
```

用作穿过网络防火墙的代理和网关【不该】，缺省时，转发在防火墙区域内部的主机的名称和端口。该信息只【应该】在明确允许时传播。若不允许，在防火墙后的任何主机的 received-by host【应该】以该主机的适当假名替换。

对隐藏内部结构有强烈私密要求的组织，代理【可以】将排序的 Via 头部域项序列与可标识的 received-protocol 值组合为单个这类条目。例如，

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

可以重组为

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

应用程序【不该】组合多个条目，除非它们全部在相同的组织控制下，且主机已经替换为假名。应用程序【禁止】有不同 received-protocol 值的条目。

14.46 Warning

Warning general-header 域用于挟带关于消息的状态和转换的额外信息，它可能不会在消息中反映。该信息一般用于警告应用到消息实体上的缓存操作或转换可能缺少语义透明度。

Warning 头部与响应一起发送，使用：

```
Warning = "Warning" ":" 1#warning-value
warning-value = warn-code SP warn-agent SP warn-text [SP warn-date]
warn-code = 3DIGIT
warn-agent = ( host [ ":" port ] ) | pseudonym ; 增加 Warning 头部的服务器的名称或假名，
                                                ; 用于调试
warn-text = quoted-string
warn-date = <"> HTTP-date <">
```

一个响应【可以】挟带超过 1 个 Warning 头部。

warn-text【应该】是接收响应的人类用户最可能理解的自然语言和字符集。该决定【可以】基于任何可用的知识，如，缓存器或用户的位置、请求中的 Accept-Language 域，响应中的 Content-Langauge 域、等等。缺省语言是英语，且缺省字节集是 ISO-8859-1。

若使用非 ISO-8859-1 字节集，则它【必须】使用 RFC 2047 [14]中描述的方法编码在 warn-text 中。

Warning 头部通常可应用到任何消息，然而，一些特别 wan-code 是缓存特殊的，且只能应用到响应消息中。新 Warning 头部【应该】增加到任何现存 Warning 头部后。缓存器【禁止】删除它所收到消息的任何 Warning 头部。然而，若缓存器成功证实某个缓存条目，则它【应该】删除以前附加到该条目上的任何 Warning 头部，除特殊 Warning 代理规定外。这时它【必须】在证实响应中增加所收到的任何 Warning 头部。换句话说，Warning 头部将附加到最近有关的响应中。

当多个 Warning 头部附加到响应中，用户代理应该尽可能多地提醒用户，按照它们出现在响应中的顺序。若不可能向用户提醒全部的警告，则用户代理【应该】遵守如下启发：

• 出现在响应前面的 Warning 比出现在响应后面的有更高的优先级。

• 按用户所选字符集的 Warning 比按其它字符集但有相同 warn-code 和 warn-agent 的警告有更高的优先级。

生成多个 Warning 头中的系统【应该】注意按用户代理的行为将它们排序。

对遵守 Warning 的缓存器的行为的要求在 13.1.2 节中声明。

这是当前定义的 warn-code 的清单，每个有推荐的英文 warn-text，和对它意思的描述。

110 Response is stale	【必须】被包括，不管何时响应过期。
111 Revalidation failed	【必须】被包括，若缓存器因尝试重证实响应失败而返回过期响应，可能因不能联系服务器。
112 Disconnected operation	【应该】被包括，若缓存器敌意有周期地断开其余的网络连接。
113 Heuristic expiration	【必须】被包括，若缓存器启发地选某个更新的，其周期大于 24 小时，且响应的年龄大于 24 小时。
199 Miscellaneous warning	警告文本【可以】包括提供给人类用户或日志的任意信息。接收该警告的系统【禁止】作出任何自动操作，包括为用户展现该警告。
214 Transformation applied	【必须】由中间缓存器或代理增加，若它应用人造棉修改响应的 content-coding（如 Content-Encoding 头部中所规定的）或 media-type（如 Content-Type 所规定的）或响应的 entity-body 的转换，除非该 Warning 代码已经出现在响应中。
299 Miscellaneous persistent warning	该警告文本【可以】包括任意信息来提示给人类用户或日志。接收该警告的系统【禁止】作出任何自动操作。

若实现发送的消息有 1 个或多个 Warning 头部，而且版本是 HTTP/1.0 或更低，这时发送方【必须】在每个 warning-value 中包括匹配响应日期的 warn-date。

若实现收到消息的 warning-value 包括 warn-date，且该 warn-date 与响应中的 Date 值不同，这时该 warning-value【必须】在存储、转发或使用它之前从消息中删除。（这将阻止天真缓存 Warning 头部域的错误结果。）若所有 warning-value 因这种原因被删除，则 Warning 头部同样【必须】被删除。

14.47 WWW-Authenticate

WWW-Authenticate response-header 域【必须】包括在 401（Unauthorized）响应消息中。域值由至少一个 challenge 组成，它指出认证方案和可应用到 Request-URI 上的参数。

WWW-Authenticate = "WWW-Authenticate" ":" 1#challenge

HTTP 访问认证过程在“HTTP Authentication: Basic and Digest Access Authentication”[43] 中描述。建议用户代理特别小心解析 WWW-Authenticate 域值，因它可能包括多于一个的 challenge，或如果提供超过一个 WWW-Authenticate 头部域，challenge 自己的内容能够包括确定的逗号分隔的认证参数表。

15 安全考虑

本节意在为应用程序开发者、信息提供者、和用户提示在本方所述的 HTTP/1.1 中的安全限制。该讨论不包括对所揭露问题的权威解决方案，尽管有一些关于减少安全风险的建议。

15.1 个人信息

HTTP 客户端经常接触大量个人信息（如，用户名、位置、邮件地址、密码、加密密匙、等），且【应该】当心，不要意外将这些信息通过 HTTP 协议泄漏给其它资源。我们非常强烈建议为用户提供控制这类信息分发的方便接口，且设计者和实现者在这方面应该特别小心。历史显示该方面的错误经常引起严重的安全和/或策略问题，并为实现者的公司造成极度不利的暴光。

15.1.1 滥用服务器日志信息

服务器是存储用户请求的个人数据的地方，它可能标识他们读取方式或关心的主题。该信息是明显的自然机密，在一些国家它的处理能够被法律所约束。使用 HTTP 协议提供数据的人有责任确保这类资源未经任何可识别个人的允许不得通过公布结果来分发。

15.1.2 传输敏感信息

与任何一般数据传输协议一样，HTTP 不能管制所传输数据的内容，也不能提供任何预处理方法来判断任何给出请求的内容中的任何信息片段的敏感性。因此，应用程序【应该】为该信息的提供者提供关于该信息的尽可能多的控制。4 个头部域值得在此处提及：Server、Via、Referer 和 From。

泄漏服务器的特定软件版本可能允许服务器主机上已知包含安全漏洞的软件更容易受到攻击。实现者【应该】使 Server 头部域为可配置选项。

服务为通过网络防火墙入口的代理【应该】特别当心关于标识防火墙后主机的头部信息的传输。特别是，它们【应该】删除任何在防火墙后生成的 Via 域，或替换为干净版本。

Referer 头部允许学习读取方式和作出返向链接。尽管它可能非常有用，它的能力可能被滥用，如果用户细节没有从 Referer 包含的信息中分离。即使当已经删除了个人消息，Referer 头部域可能指出私有文档的 URI，将其公开将是不适当的。

From 域中发送的信息可能与用户私有利益或其它站点安全策略相冲突，因此，【不该】在用户能够取消、允许和修改该域的内容时传输它。用户【必须】能够在用户选项或应用程序缺省配置中设置该域的内容。

我们建议，尽管不要求，为用户提供方便的切换接口来允许或禁止发送 From 和 Referer 信息。

一些时候，能够使用 User-Agent（14.43 节）或 Server（14.38 节）头部域来判断特定客

户端或服务器拥有的可利用的特定安全漏洞。不幸的是，由于 HTTP 当前没有更好的机制，相同的信息还经常用在其它有价值的用途上。

15.1.3 编码 URI 敏感信息

因为源链接可能是私有信息或可能泄漏其它私有信息源，强烈建议用户能够选择是否发送 Referer 域。例如，浏览器客户端可能有切换选项来公开/匿名浏览，将相应地允许/林发送 Referer 和 From 信息。

如果引用页通过安全协议传输，客户端【不该】在（非安全）HTTP 请求中包括 Referer 头部域。

使用 HTTP 协议的服务作者【不该】使用基于 GET 的形式来提交敏感数据，因为这将引起数据在 Request-URI 中编码。许多现存服务器、代理和用户代理将在一些地址记录该请求 URI，它可能对第三者可见。服务器可以使用 POST 形式的提交来代替。

15.1.4 与接受头部相关的隐私问题

接受 request-header 可以泄漏用户信息给所有访问的服务器。特别是 Accept-Language 头部可以泄漏用户将认为是私有属性的信息，因为理解特定的语言经常与特殊人种成员高度相关。强烈鼓励用户代理提供选项来配置所发送的 Accept-Language 头部的内容，并使配置过程包括提示导致用户失去隐私的消息。

防止失去隐私的途径将使用户代理缺省时省去发送 Accept-Language 头部，并询问用户是否要开始发送 Accept-Language 头部给服务器，如果用户代理通过观察服务器生成的任何 Vary response-header 域检测到该发送可能改善服务质量。

精心制作的在每个请求中发送的用户定制接受头部域，特别是如果包括质量值，可被服务器用来作为相关可靠的和长期存在的用户标识。这种用户标识将允许内容提供者跟踪点击痕迹，且将允许合作内容提供者匹配跨服务器点击痕迹或个人用户提交的表单。要注意，对于许多不在代理后的用户而言，运行用户代理的主机的网络地址也可作为长期用户标识。在代理用来增强隐私的环境中，用户代理应该为终端用户保守提供接受头部配置选项。作为极度隐私的尺度，代理将在接力请求中过滤接受头部。提供高等级头部可配置的一般用途的用户代理【应该】警告用户可能导致失去隐私的行为。

15.2 基于文件和路径名的攻击

HTTP 原始服务器实现【应该】小心限制 HTTP 请求返回的文档只是服务器管理员想要的。如果 HTTP 服务器将 HTTP URI 直接翻译为文件系统调用，则服务器【必须】特别小心服务不是期望的文件给 HTTP 客户端。例如，UNIX、微软视窗、和其它操作系统使用“..”作为指出当前目录上层的路径成分。在这类系统中，HTTP 服务器【必须】禁止 Request-URI 中的这类构造，否则它将允许访问通过 HTTP 服务器期望可访问范围之外的资源。与此相似，只希望由服务器在内部引用的文件（如，访问控制文件、配置文件、和脚本代码）【必须】保护不被非正确地获取，因为它们可能包含敏感信息。经验显示，这类 HTTP 服务器实现中

的小问题已经变为安全风险。

15.3 DNS 欺骗

使用 HTTP 的客户端严重依赖于域名服务，因此一般倾向于对故意非相关的 IP 地址和 DNS 名称的安全攻击。客户需要小心假设 IP 号/DNS 名相关的持续有效性。

特别是，HTTP 客户端【应该】依赖它们自己的名称解析器来确认 IP 地址/DNS 名称的相关性，而不是以前主机名称查询缓存的结果。许多平台已经能够在本地缓存正确的主机名称查询，且【应该】配置它们这样做。缓存这些查询是正确的，然而，仅当名称服务器报告的 TTL（存活期）信息似乎表明缓丰信息依然有用时。

如果 HTTP 客户端了为达到改善性能的目的而缓存主机名称查询的结果，它们【必须】观察 DNS 报告的 TTL 信息。

如果 HTTP 客户端不观察该规则，它们可能在以前所访问的服务器的 IP 地址改变时被欺骗。因为网络重复编号预期会日益普遍 [24]，这种攻击形式的可能性将会增长。留意这种要求会缩小该潜在的安全弱点。

该要求还会改善客户端重复使用相同 DNS 名称的服务器的加载平衡行为，并会减少在访问使用该策略的站点时用户经历失败的可能性。

15.4 Location 头部和欺骗

如果单个服务器支持多个组织，它们之间并不互相信任，这时它【必须】检查响应的 Location 和 Content-Location 头部的值以确认它们没人尝试它们没有权力的无效资源，该响应由宣称它的组织控制其生成。

15.5 Content-Disposition 问题

RFC 1806 [35]有大量的非常严重的安全考虑，它是 HTTP 经常实现 Content-Disposition（见 19.5.1 节）头部的根源。Content-Disposition 不是 HTTP 标准的成份，但由于已经非常广泛实现，我们归档其用法和实现者的风险。详见 RFC 2183 [49]（更新 RFC 1806）。

15.6 认证证书和空闲客户端

现存 HTTP 客户端和用户代理一般非确定地保留认证信息。HTTP/1.1 没有为服务器引导客户端丢弃这些缓存证书提供方法。这是明显过失，需要在将来扩展 HTTP。证书缓存下的环境能够通用应用程序的安全模型来干预，包括但不限于：

-客户端已经空闲一长段时间，接着服务器可能希望引起客户端重新提示用户认证。

-应用程序包括会话终止指示（如“注销”或页面中的“提交”按钮），之后应用程序的服务器端“知道”没有更多的原因为客户端保留该证书。

当前该问题正在独立研究。有大量的工作围绕该问题的某部分，且我们鼓励在屏保中使用密码保护、空闲超时和可减轻该问题固有的安全问题的其它方法。特别是，鼓励缓存证书的用户代理为在用户控制下放弃缓存证书提供容易访问的机制。

15.7 代理和缓存

因它们的特别的本性，HTTP 代理是中间者，且存在中间者攻击的机会。作为在代理所运行系统的妥协，可能造成严重的安全和隐私问题。代理有权获得安全相关的信息、关于个人用户和组织的个人信息、和属于用户和内容提供者的私有信息。妥协的代理、或实现或配置的没有注意安全和隐私考虑的代理，可以用于委托广范围潜在攻击。

代理操作者应该保护代理运行的系统，同时它们应该保护包含或传输敏感信息的系统。特别是，代理收集的日志信息经常包含高度敏感的个人信息，和/或组织信息。日志信息应该小心保护，并开发和遵循正确的用法指南。（15.1.1 节）

缓存代理提供额外潜在缺点，因为缓存的内容表述对恶意宣传有吸引力的目标。因为缓存内容在 HTTP 请求完成后会持续，对缓存的攻击能够在用户认为该信息已经从网络上删除后长时间泄漏信息。因此，缓存内容应该作为敏感信息来保护。

代理实现者应该考虑它们设计和编码决策，及它们提供给代理操作者的配置选项（特别是缺省配置）方面的隐私和安全含义。

代理的用户需要清楚，它们不比运行代理的人更值得信任。HTTP 自己不能解决该问题。

明智地使用密码学，在适当时，可足以防卫大范围的安全和隐私攻击。这种密码学超出 HTTP 规范的范围。

15.7.1 对代理的拒绝服务攻击

它们确实存在，且很难防范。还需继续研究。要小心。

16 感谢

本规范大量利用增广 BNF 和 David H. Crocker 的 RFC 822 [9]定义的一般构造。相似地，它重用了由 Nathaniel Borenstein 和 Ned Freed 的 MIME [7]提供的许多定义。我们希望将它们引入本规范中将帮助减少过去对 HTTP 与因特网邮件消息格式之间关系的混淆。

HTTP 协议已经进化相当多年。它已经从大量活动的开发者社团获益——许多人已经参与到 WWW 论坛邮件列表中，且就是这些普通社团造成了 HTTP 和 WWW 的成功。Marc Andreessen、Robert Cailliau、Daniel W. Connolly、Bob Denny、John Franks、Jean-Francois Groff、Phillip M. Hallam-Baker、Hakon W. Lie、Ari Luotonen、Rob McCool、Lou Montulli、Dave Raggett、Tony Sanders 和 Marc VanHeyningen 值得特别认识，因他们在定义早期协议方面的影响。

本文已经从 HTTP-WG 中的所有参与者的注释中极大地获益。补充这些已经提及的，下面的个人对本规范做出过贡献：

Gary Adams
Harald Tveit Alvestrand
Keith Ball
Brian Behlendorf
Paul Burchard
Maurizio Codogno
Mike Cowlishaw
Roman Czyborra
Michael A. Dolan
David J. Fiander
Alan Freier
Marc Hedlund
Greg Herlihy
Koen Holtman
Alex Hopmann
Bob Jernigan
Shel Kaphan
Rohit Khare
John Klensi
Martijn Koster
Alexei Kosut
David M. Kristol
Daniel LaLiberte
Ben Laurie
Paul J. Leach
Daniel DuBois
Ross Patterson
Albert Lunde
John C. Mallery
Jean-Philippe Martin-Flatin
Mitra
David Morris
Gavin Nicol
Bill Perry
Jeffrey Perry
Scott Powers
Owen Rees
Luigi Rizzo
David Robinson
Marc Salomon
Rich Salz
Allan M. Schiffman
Jim Seidman
Chuck Shotton
Eric W. Sink
Simon E. Spero
Richard N. Taylor
Robert S. Thau
Bill (BearHeart) Weinman
Francois Yergeau
Mary Ellen Zurko
Josh Cohen

缓存设计的大多数内容和表述归于个人的建议和注释，包括：Shel Kaphan、Paul Leach、Koen Holtman、David Morris 和 Larry Masinter。

范围的大多数规定基于 Ari Luotonen 和 John Franks 的最初工作，和 Steve Zilles 的额外输入。

感谢 Palo Alto 的“cave men”。你知道你是谁。

Jim Gettys（本文的当前编辑）特别希望感谢 Roy Fielding，本文的前任编辑，与 John Klensin、Jeff Mogul、Paul Leach、Dave Kristol、Koen Holtman、John Franks、Josh Cohen、Alex Hopmann、Scott Lawrence 和 Larry Masinter 的帮助。并且要特别感谢 Jeff Mogul 和 Scott Lawrence 执行“【必须】/【可以】/【应该】”验证。

Apache 项目组、Anselm Baird-Smith、Jigsaw 的作者和 Henrik Frystyk 实现早期的 RFC 2068，我们希望感谢他们发现了许多本文尝试调整的问题。

17 参考资料

[1] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[2] Anklesaria, F., McCahill, M., Lindner, P., Johnson, D., Torrey, D. and B. Alberti, "The Internet Gopher Protocol (a distributed document search and retrieval protocol)", RFC 1436, March 1993.

[3] Berners-Lee, T., "Universal Resource Identifiers in WWW", RFC 1630, June 1994.

[4] Berners-Lee, T., Masinter, L. and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[5] Berners-Lee, T. and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, November 1995.

[6] Berners-Lee, T., Fielding, R. and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[7] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[8] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1123, October 1989.

[9] Crocker, D., "Standard for The Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

[10] Davis, F., Kahle, B., Morris, H., Salem, J., Shen, T., Wang, R., Sui, J., and M. Grinbaum, "WAIS Interface Protocol Prototype Functional Specification," (v1.5), Thinking Machines Corporation, April 1990.

[11] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, June 1995.

[12] Horton, M. and R. Adams, "Standard for Interchange of USENET Messages", RFC 1036, December 1987.

[13] Kantor, B. and P. Lapsley, "Network News Transfer Protocol", RFC 977, February 1986.

[14] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[15] Nebel, E. and L. Masinter, "Form-based File Upload in HTML", RFC 1867, November 1995.

[16] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, August 1982.

[17] Postel, J., "Media Type Registration Procedure", RFC 1590, November 1996.

[18] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[19] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[20] Sollins, K. and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, December 1994.

[21] US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[22] ISO-8859. International Standard -- Information Processing --
8-bit Single-Byte Coded Graphic Character Sets --
Part 1: Latin alphabet No. 1, ISO-8859-1:1987.
Part 2: Latin alphabet No. 2, ISO-8859-2, 1987.
Part 3: Latin alphabet No. 3, ISO-8859-3, 1988.
Part 4: Latin alphabet No. 4, ISO-8859-4, 1988.
Part 5: Latin/Cyrillic alphabet, ISO-8859-5, 1988.
Part 6: Latin/Arabic alphabet, ISO-8859-6, 1987.
Part 7: Latin/Greek alphabet, ISO-8859-7, 1987.
Part 8: Latin/Hebrew alphabet, ISO-8859-8, 1988.
Part 9: Latin alphabet No. 5, ISO-8859-9, 1990.

[23] Meyers, J. and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995.

[24] Carpenter, B. and Y. Rekhter, "Renumbering Needs Work", RFC 1900, February 1996.

[25] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

[26] Venkata N. Padmanabhan, and Jeffrey C. Mogul. "Improving HTTP Latency", Computer Networks and ISDN Systems, v. 28, pp. 25-35, Dec. 1995. Slightly revised version of paper in Proc. 2nd International WWW Conference '94: Mosaic and the Web, Oct. 1994, which is available at http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html.

[27] Joe Touch, John Heidemann, and Katia Obraczka. "Analysis of HTTP Performance", <URL: http://www.isi.edu/touch/pubs/http-perf96/>, ISI Research Report ISI/RR-98-463, (original report dated Aug. 1996), USC/Information Sciences Institute, August 1998.

[28] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992.

[29] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[30] S. Spero, "Analysis of HTTP Performance Problems," http://sunsite.unc.edu/mdma-release/http-prob.html.

[31] Deutsch, P. and J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[32] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP: Digest Access Authentication", RFC 2069, January 1997.

[33] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[34] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[35] Troost, R. and Dorner, S., "Communicating Presentation Information in Internet Messages: The Content-Disposition

Header", RFC 1806, June 1995.

[36] Mogul, J., Fielding, R., Gettys, J. and H. Frystyk, "Use and Interpretation of HTTP Version Numbers", RFC 2145, May 1997. [jg639]

[37] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997. [jg640]

[38] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO-10646", RFC 2279, January 1998. [jg641]

[39] Nielsen, H.F., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H., and C. Lilley. "Network Performance Effects of HTTP/1.1, CSS1, and PNG," Proceedings of ACM SIGCOMM '97, Cannes France, September 1997.[jg642]

[40] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996. [jg643]

[41] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998. [jg644]

[42] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax and Semantics", RFC 2396, August 1998. [jg645]

[43] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999. [jg646]

[44] Luotonen, A., "Tunneling TCP based protocols through Web proxy servers," Work in Progress. [jg647]

[45] Palme, J. and A. Hopmann, "MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2110, March 1997.

[46] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[47] Masinter, L., "Hyper Text Coffee Pot Control Protocol

(HTCPCP/1.0)", RFC 2324, 1 April 1998.

[48] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.

[49] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

18 作者地址

Roy T. Fielding
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA

Fax: +1 (949) 824-1715
EMail: fielding@ics.uci.edu

James Gettys
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail: jg@w3.org

Jeffrey C. Mogul
Western Research Laboratory
Compaq Computer Corporation
250 University Avenue
Palo Alto, California, 94305, USA

EMail: mogul@wrl.dec.com

Henrik Frystyk Nielsen
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square

Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail: frystyk@w3.org

Larry Masinter
Xerox Corporation
3333 Coyote Hill Road
Palo Alto, CA 94034, USA

EMail: masinter@parc.xerox.com

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052, USA

EMail: paulle@microsoft.com

Tim Berners-Lee
Director, World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail: timbl@w3.org

19 附录

19.1 因特网媒体类型 message/http 和 application/http

作为定义 HTTP/1.1 协议的附加，本文用作因特网媒体类型“message/http”和“appliction/http”的规范。可以使用 message/http 来封装单个 HTTP 请求或响应消息，提供它遵守 MIME 对所有“message”类型关于行长度和编码的限制。application/http 可以用来封装一个或多个 HTTP 请求或响应消息的管道（非混合的）。下面由 IANA [17]注册。

媒体 type 名：message
媒体 subtype 名：http
所需 parameter：无
可选 parameter：version，msgtype
version：封装消息的 HTTP-Version 号（如，“1.1”）。如果不存在，版本可从主体

的首行判断出。

msgtype：消息类型——“request”或“response”。如果不存在，类型可由主全的首行判断出。

编码考虑：只允许“7bit”、“8bit”或“binary”。

安全考虑：无

媒体 type 名：application

媒体 subtype 名：http

所需 parameter：无

可选 parameter：version, msgtype

version：封装消息的 HTTP-Version 号（如，“1.1”）。如果不存在，版本可从主体的首行判断出。

msgtype：消息类型——“request”或“response”。如果不存在，类型可由主全的首行判断出。

编码考虑：由该类型封装的 HTTP 消息为“binary”格式；当通过电子邮件传输时需要使用适当的 Content-Transfer-Encoding。

安全考虑：无

19.2 因特网媒体类型 multipart/byteranges

当 HTTP 206（Partial Content）响应消息包括多范围的内容时（请求多个非覆盖范围的响应），作为 multipart message-body 来传输。用于该用途的媒体类型称为“multipart/byteranges”。

multipart/byteranges 媒体类型包括 2 个或多个部分，每个有自己的 Content-Type 和 Content-Range 域。需要使用 boundary parameter 规定的边界字符串来分隔每个 body-part。

媒体 type 名：multipart

媒体 subtype 名：byteranges

所需 parameter：boundary

可选 parameter：none

编码考虑：只允许“7bit”、“8bit”或“binary”。

安全考虑：无

例如：

```
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-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES

--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 500-999/8000
```

...the first range...
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 7000-7999/8000

...the second range
--THIS_STRING_SEPARATES--

注意：

1）要在实体的首个边界字符串前加入额外的 CRLF。

2）尽管 RFC 2046 [40]允许边界字符串是引用的，一些现存的实现不能正确处理引用边界字符串。

3）一些浏览器和服务器按 byteranges 规范的早期草案编码，使用 multipart/x-byteranges 媒体类型，它几乎与，但不完全与 HTTP/1.1 归档的版本兼容。

19.3 容错应用程序

尽管本文档规定 HTTP 消息生成的要求，非所有应用程序分正确实现。因此我们推荐可操作应用程序应是容忍可能解释含糊的背离。

客户端【应该】在解析 Status-Line 时容错，服务器在解析 Request-Line 当容错。特别是，它们【应该】接受域间任何数量的 SP 或 HT 字符，即使只需要单个 SP。

message-header 域的行终止符是 CRLF 序列。然而，我们建议当应用程序解析该头部时，认别单个 LF 作为行终止符并乎略引导 CR。

entity-body 的字符集【应该】标注为用在该主体中的字符编码的最不通用的命名者，除了不标注实体比用 US-ASCII 或 ISO-8859-1 标注实体更好。见 3.7.1 和 3.4.1 节。

解析和编码日期的额外规则要求，及日期编码的其它潜在问题包括：

——HTTP/1.1 客户端和缓存【应该】假设在将来出现超过 50 年的 RFC-850 日期实际上是在过去（这可帮助解决“2000 年”问题）。

——HTTP/1.1 实现【可以】在内部表示比正确值更早的已解析 Expires 日期，但【禁止】在内部表示比正确值更晚的已解析 Expires 日期。

——所有截止相关的计算【必须】按 GMT 进行。本地时区【禁止】影响计算或年龄或截止时间的比较。

——如果 HTTP 头部不正确挟带时区是 GMT 的日期值，【必须】将其转换为 GMT，使用可能的最保守转换。

19.4 HTTP 实体和 RFC 2045 实体间的区别

HTTP/1.1 使用大量因特网邮件（RFC 822 [9]）中定义的构造及多用途因特网邮件扩展（MIME [7]），以允许实体按开放表示类别来传输，且可扩展。然而，RFC 2045 讨论邮件，且 HTTP 有些特性与 RFC 2045 中描述的不同。这些区别通过详细选择，来优化通过二进制连接的性能，以允许更自由地使用新媒体类型，使用日期比较更容易，并承认一些早期 HTTP 服务器和客户端的实践。

本附录描述 HTTP 与 RFC 2045 不同的特定地方。到严格 MIME 环境的代理和网关【应该】明白这些区别，并提供适当的必须的转换。从 MIME 环境到 HTTP 的代理和网关也需要清楚这些区别，因为需要一些转换。

19.4.1 MIME-Version

HTTP 并非 MIME 一致的协议。然而，HTTP/1.1 消息【可以】包括单个 MIME-Version 的 general-header 域来指出所用来构造消息的 MIME 协议的版本。使用 MIME-Version 头部域指出该消息完全与 MIME 协议一致（如 RFC 2045 [7]所定义）。当导出 HTTP 消息到严格的 MIME 环境时，代理/网关有责任确保完全一致（尽可能）。

```
MIME-Version = "MIME-Version" ":" 1*DIGIT "." 1*DIGIT
```

HTTP/1.1 缺省使用 MIME 版本“1.0”。然而，HTTP/1.1 消息的解析和语义由本文定义，而非 MIME 规范。

19.4.2 转换为正式形式

RFC 2045 [7]需要因特网邮件实体在传输前转换为正式形式，如 RFC 2049 [48]的 4 节所述。本文的 3.7.1 描述在通过 HTTP 传输时所允许的“text”媒体类型的 subtype 的形式。RFC 2046 需要“text”type 的内容描绘行终止符为 CRLF，用禁止在行终止序列外使用 CR 或 LF。HTTP 允许当消息通过 HTTP 传输时在文本中用 CRLF、光 CR 和光 LF 来表示行终止。

当可能时，从 HTTP 到严格 MIME 环境的代理或网关【应该】翻译在本文的 3.7.1 节中描述的文本媒体类型中的所有行终止符为 RFC 2049 的正式形式 CRLF。然而，要注意这可因 Content-Encoding 的存在和 HTTP 实际上允许使用非以字节 13 和 10 来表示 CR 和 LF 的字符集，如一些多字节字符集的情况，而复杂化。

实现者应该注意，该转换将打破应用到原始内容上的任何密码样验，除非原始内容已经是正式形式。因此，推荐在 HTTP 中使用该校验的任何内容按正式形式。

19.4.3 日期格式的转换

HTTP/1.1 使用严格限制的日期格式集（3.3.1 节）来简化日期的比较过程。从其它协议来的代理和网关【应该】确保消息中的任何日期头部域表述都转换为 HTTP/1.1 格式中的一个，并按需要重写数据。

19.4.4 引入 Content-Encoding

RFC 2045 没有包括任何与 HTTP/1.1 的 Content-Encoding 头部域相等的概念。因为这作为媒体类型的修饰符，从 HTTP 到 MIME 一致协议的代理和网关【必须】要么修改 Content-Type 头部域的值，要么在转发消息前解码 entity-body。（一些因特网邮件的 Content-Type 的实验性应用程序已经使用 media-type 参数“; conversions=<content-coding>”来执行与 Content-Encoding 相同的功能。然而，该参数不是 RFC 2045 的一部分。）

19.4.5 无 Content-Transfer-Encoding

HTTP 不使用 RFC 2045 的 Content-Transfer-Encoding（CTE）域。从 MIME 一致协议到 HTTP 的代理和网关【必须】先于将响应消息交付给 HTTP 客户端前删除任何非 identity 的 CTE（“quoted-printable”或“base64”）编码。

从 HTTP 到 MIME 一致协议的代理和网关有责任确保消息按正确的格式，用为该协议上的安全传输编码，这时的“安全传输”由正使用的协议的约束定义。这类代理或网关【应该】标注数据为适当的 Content-Transfer-Encoding，如果这样做，将会提高通过目标协议安全传输的可能性。

19.4.6 引入 Transfer-Encoding

HTTP/1.1 引入 Transfer-Encoding 头部域（14.41 节）。代理/网关【必须】在转发消息到 MIME 一致协议前删除任何 transfer-coding。

解码“chunked”transfer-coding（3.6 节）的过程可以用伪代码描述为：

```
length := 0
read chunk-size, chunk-extension (if any) and CRLF
while (chunk-size > 0) {
    read chunk-data and CRLF
    append chunk-data to entity-body
    length := length + chunk-size
    read chunk-size and CRLF
}
read entity-header
while (entity-header not empty) {
    append entity-header to existing header fields
    read entity-header
}
Content-Length := length
```

Remove "chunked" from Transfer-Encoding

19.4.7 MHTML 和行长度限制

与 MHTML [45]实现共享代码的 HTTP 实现需要清楚 MIME 行长度限制。因为 HTTP 没有这种限制，HTTP 不折回长行。由 HTTP 传输的 MHTML 消息遵循 MHTML 的约定，包括行长度限制和折回、正式化、等。由于 HTTP 传输所有 message-body 作为负载（见 3.7.2 节），且不解释内容或可能包含在其中的任何 MIME 头部行。

19.5 附加特性

一些已存 HTTP 实现使用 RFC 1945 和 RFC 2068 文档协议元素，但不是一直和正确地用在大多数 HTTP/1.1 应用程序中。建议实现者清楚这些特性，但不能领先它们的存在，或与其它 HTTP/1.1 应用程序的交互性。其中一些描述提议实验性特性，一些描述特性在实验布署中发现缺少在基本 HTTP 规范中的当前说明。

一些其它头部，如从 SMTP 和 MIME 引入的 Content-Dispostion 和 Title，经常被实现（见 RFC 2076 [37]）。

19.5.1 Content-Disposition

Content-Disposition response-header 域已经被提议作为原始服务器建议的缺省文件名的意思，如果用户请求将内容保存为文件。该用法起源于 RFC 1806 [35]中的 Content-Dispostion 的定义。

```
content-disposition = "Content-Disposition" ":" disposition-type *( ";" disposition-parm )
disposition-type = "attachment" | disp-extension-token
disposition-parm = filename-parm | disp-extension-parm
filename-parm = "filename" "=" quoted-string
disp-extension-token = token
disp-extension-parm = token "=" ( token | quoted-string )
```

一个例子是

```
Content-Disposition: attachment; filename="fname.ext"
```

接收用户代理【不该】遵从 filename-param 参数中出现的任何目录路径信息，这是当前认为应用到 HTTP 实现的唯一参数。文件名【应该】只按终止组件对待。

如果该组件用在有 application/octet-stream content-type 的响应中，潜在建议是用户代理不该显示该响应，而是直接进入“保存响应为...”对话框。

关于 Content-Disposition 的安全问题见 15.5 节。

19.6 与以前版本兼容

要求与以前版本兼容已经超出协议规范的范围。然而，HTTP/1.1 故意设计以使支持以前版本容易。它没有任何价值，在构造该规范的时候（1996），我们将希望商业 HTTP/1.1 服务器：

——认识 HTTP/0.9、1.0 和 1.1 请求的 Request-Line 格式；

——理解 HTTP/0.9、1.0 和 1.1 格式的任何有效请求；

——响应以按客户端所用主要版本相同的适当的消息。

且我们将希望 HTTP/1.1 客户端：

——认识 HTTP/1.0 和 1.1 响应的 Status-Line 的格式；

——理解 HTTP/0.9、1.0 或 1.1 格式的任何有效响应。

对于大多数 HTTP/1.0 实现，每个连接由客户端在请求前建立，并由服务器在发送响应后关闭。一些实现 RFC 2068 [33]的 19.7.1 节中描述的永久连接的 Keep-Alive 版本。

19.6.1 从 HTTP/1.0 以来的修改

本节总结 HTTP/1.0 和 HTTP/1.1 之间的主要区别。

19.6.1.1 改变为简单多主页服务器和保存 IP 地址

客户端和服务器支持 Host request-header 的要求将报告错误，如果 HTTP/1.1 请求缺少 Host request-header（14.23 节），并接受绝对 URI（5.1.2 节）是本规范定义的最重要的修改。

老的 HTTP/1.0 客户端假设 IP 地址和服务器是一对一的关系；除了请求指向的 IP 地址包没有建立其它机制来区别请求目的服务器。一旦老 HTTP 客户端不再通用时，上面提及的要点将允许因特网支持单 IP 地址多网站，极大地简化动作大量的网站服务器。若单个主机分配大量 IP 地址将导致严重的问题。因特网还能够修复为允许特殊用途域名用来作为根层 HTTP URL 的唯一目的所分配的 IP 地址。给出网站增长率，和已经布署的服务器数量，所有的 HTTP 实现（包括更新现存 HTTP/1.0 应用程序）正确实现这种要求是极其重要的：

——客户端和服务器都【必须】支持 Host request-header。

——发送 HTTP/1.1 请求的客户端【必须】发送 Host 头部。

——若 HTTP 请求不包括 Host request-header，服务器【必须】报告 400（Bad Request）错误。

—服务器【必须】接受绝对 URI。

19.6.2 与 HTTP/1.0 永久连接兼容

一些客户端或服务器可能钫与以前的 HTTP/1.0 客户端和服务器的持久连接兼容。HTTP/1.0 的持久连接是明确协商的，因它们不是缺省行为。HTTP/1.0 持久连接的实验性实现是错误的，并设计新的 HTTP/1.1 的工具来矫正这种问题。该问题是，一些现存的 1.0 客户端可能发送 Keep-Alive 给不理解 Connection 的代理服务器，这时它可能被错误地转发给下个入界服务器，这可能建立 Keep-Alive 连接，并导致代理长时间等待响应关闭。其结果是，HTTP/1.0 客户端必须阻止使用 Keep-Alive 来与代理对话。

然而，与代理对话是持久连接最重要的应用之一，所以禁止明显不可接受。因此，我们需要一些其它机制来指出希望持久连接，即使与乎略 Connection 的老代理对话也使用安全。持久连接是 HTTP/1.0 消息的缺省行为；我们引入关键字（Connection: close）来声明非持久连接。见 14.10 节。

老的 HTTP/1.0 格式的持久连接（Connection: Keep-Alive 和 Keep-Alive 头部）在 RFC 2068 [33]中归档。

19.6.3 从 RFC 2068 以来的修改

已经仔细审计本规范，以修正和消除关键字用法的岐义；RFC 2068 在遵守 RFC 2119 [34] 列出的规定方面有许多问题。

阐明应该为入界服务器失败（如，DNS 失败）使用哪个错误代码。（10.5.5 节）。

“创建”有步骤，当资源首次创建时需要发送 ETag。（10.2.2 节）。

Content-Base 从本规范中删除：它没有广泛实现，且没有简单、安全的方法来引入它而不用良好的扩展机制。而且，它用在与 MHTML [45]中相似，但不相同的样式中。

transfer-coding 和消息长度全部按这种方式互相作用，即当使用 chunked 编码时（允许传输非自定界的编码）需要正确修正的方式；正确直接指出如何计算消息长度是重要的。（3.6、4.4、7.2.2、13.5.2、14.13、14.16 节）

引入“identity”content-coding 来解决缓存中发现的问题。（3.5 节）

质量值为 0 指出“我不要某类”，允许客户端拒绝某类描绘。（3.9 节）

HTTP 版本号的用法和解释已经在 RFC 2145 中阐明。要求代理更新请求到它们支持的最高协议版本，以解决在 HTTP/1.0 实现中发现的问题。（3.1 节）

引入通配 charset 来避免 Accept 头部中出现大量字符集名。（14.2 节）

HTTP/1.1 的 Cache-Control 模型中缺少 case；引入 s-maxage 来增加该缺少的 case。（13.4、14.8、14.9、14.9.3 节）

Cache-Control: max-age 指令没有正确为响应定义。（14.9.3 节）

存在这样的情况，服务器（特别是代理）不了解响应的全部长度，但能够服务 byterage 请求。因此，我们需要机制来允许用 content-range 在没有指出消息的全部长度的 byterange 上。（14.16 节）

若元数据始终要返回，则范围请求响应将变得非常冗余；通过允许服务器只发送所需的头部在 206 响应中，可避免该问题。（10.2.7、13.5.3、和 14.27 节）

修正不满足范围请求的问题；有两种情况：语义问题，用范围在文档中不存在。需要 416 状态码来解决该岐义，需要指出 byterange 请求的错误，即超出文档的实际内容。（10.4.17、10.16 节）

重定消息传输要求使实现者更难范错误，因为在这里错误的后果能够对因特网有明显的冲击。实现者将处理如下问题：

1、修改不正确要求 HTTP/1.x 未来版本实现的行为的内容的“HTTP/1.1 或更新”为“HTTP/1.1”。

2、要阐明，通常用户代理应该重试请求，而非“客户端”。

3、转换对客户端乎略非预期 100（Continue）响应的要求，和对代理转发 100 响应的要求为对 1xx 响应的一般要求。

4、修改一些 TCP 特定的语言，以阐明，非 TCP 传输可用于 HTTP。

5、要求原始服务器【禁止】在发送所要求的 100（Continue）响应前等待请求主体。

6、允许而不是要求服务器在已经收到部分请求主体时省略 100（Continue）。

7、允许服务器防卫拒绝服务器攻击和被破坏的客户端。

本修正增加 Expect 头部和 417 状态码。消息传输要求修正在 8.2、10.4.18、8.1.2.2、13.11、和 14.20 节中。

代理应该能够增加适当的 Content-Length。（13.5.2 节）

澄清 403 和 404 响应间的混乱。（10.4.4、10.4.5、和 10.4.11 节）

警告可能被不正确地缓存，或不适当地更新。（13.1.2、13.2.4、13.5.2、13.5.3、14.9.3、和 14.46 节）警告还需要是一般头部，因为 PUT 或其它方法可能在请求中需要它。

transfer-coding 有明显的问题，特别是与 chunked 编码相互作用时。解决方案是 transfer-coding 问题完全与 content-coding 不同步。这导致增加 IANA 的 transfer-coding 注册表（与 content-coding 独立）、增加新的头部域（TE），并增加将来允许尾巴头部。传输编码是主要的性能益处，所以它值得修正 [39]。TE 还解决另一个模糊的、向下交互操作问题。该问题已经因认证尾巴、chunked 编码和 HTTP/1.0 客户端交互而出现。

定义了 PATCH、LINK、UNLINK 方法，但没有在本规范的前个版本中普遍实现。见 RFC 2068 [33]。

Alternates、Content-Version、Derived-From、Link、URI、Public 和 Content-Base 头部域在本规范的首一版中定义，但没有普遍实现。见 RFC 2068 [33]。

20 完整版权声明

20.1 感谢

目前，RFC 编辑的运作资金由因特网社区提供。

21 索引

索引请见本 RFC 的 PostScript 版本。

22 翻译选择

22.1 翻译说明

本文由 Cobras Zhang（cobras@yeah.net）翻译，完全按照原文内容及布局翻译，没有任何删减或添加，当然本节除外。由于水平有限，翻译问题不少。若使用者有任何修改建议，

欢迎发邮件告知，我将及时更正及发布改版。

翻译原则是, 除了 BNF 语法有效元素及其符号名、响应原因短语和其它特殊词汇不翻译，其它的都要按上下文翻译。下面的 22.2 和 22.3 小节列出不翻译的词汇及其所对应的中文意思，22.4 和 22.5 小节列出本译文中对特殊词汇所选择的意思。

当然，原文的作者地址和参考资料也没有翻译。

22.2 BNF 语法符号名

Content-Length	内容长度
Range	范围
Transfer-Encoding	传输编码

22.3 特殊词汇

BNF	巴柯斯范式，一种语法描述规则
FTP	文件传输协议
Gopher	一种因特网资源浏览器
HTTP	超文本传输协议
IANA	因特网分配数字权威
IETF	因特网工程攻坚任务组
IP	网际协议
ISO	国际标准化组织
MIME	多用途因特网邮件扩展
NNTP	网络新闻传输协议
RFC	IETF 的请求评论组织
SMTP	简单邮件传输协议
STD	走标准路线的 RFC 文档
TCP	传输控制协议
URI	统一资源标识符
URL	统一资源定位符
URN	统一资源名称
WAIS	广域信息服务器
WWW	世界范围的网络

22.4 规范要求词汇

MUST	aux.	【必须】
MUST NOT	aux.	【禁止】
MAY	aux.	【可以】
MAY NOT	aux.	【不可】
SHALL	aux.	【将要】
SHALL NOT	aux.	【不要】

SHOULD	aux.	【应该】
SHOULD NOT	aux.	【不该】
CAN	aux.	【能够】
CAN NOT	aux.	【不能】
REQUIRED	adj.	【要求的】
RECOMMENDED	adj.	【推荐的】
OPTIONAL	adj.	【可选的】

22.5 其它词汇

cache	n.	缓存
client	n.	客户端
entity	n.	实体
entry	n.	条目
recipient	n.	接收方
sender	n.	发送方
server	n.	服务器
tag	n.	标签
	v.	标注
validate	v.	证实
validator	n.	证言