互联网工程任务组 (IETF)
征求意见:7231
过时:2616
更新:2817
类别:标准轨道
ISSN:2070-1721

R. 菲尔丁,埃德。
Adobe
J. Reschke,编辑。
greenbytes 2014
年 6 月

超文本传输协议 (HTTP/1.1):语义和内容

抽象的

超文本传输协议 (HTTP) 是用于分布式协作超文本信息系统的无状态应用程序级协议。本文档定义了 HTTP/1.1 消息的语义,如请求方法、请求标头字段、响应状态代码和响应标头字段所表达,以及消息的有效负载 (元数据和正文内容)和内容协商机制。

本备忘录的状态

这是一份 Internet 标准跟踪文档。

本文档是 Internet 工程任务组 (IETF) 的产品。它代表了 IETF 社区的共识。它已接受公众审查,并已被互联网工程指导小组 (IESG) 批准发布。有关 Internet 标准的更多信息,请参阅 RFC 5741 的第 2 节。

有关本文档的当前状态、任何勘误表以及如何提供反馈的信息,请访问 http://www.rfc-editor.org/info/rfc7231。

目录

一、简介

每个超文本传输协议 (HTTP) 消息都是请求或响应。服务器在连接上侦听请求,解析收到的每条消息,解释与已识别请求目标相关的消息语义,并使用一个或多个响应消息响应该请求。客户端构造请求消息来传达特定意图,检查收到的响应以查看意图是否已执行,并确定如何解释结果。本文档根据 [RFC7230] 中定义的体系结构定义了 HTTP/1.1 请求和响应语义。

HTTP 提供了一个统一的接口来与资源交互（第 2 节）,无论其类型、性质或实现如何,通过表示的操作和传输（第 3 节）。

HTTP 语义包括每个请求方法定义的意图（第 4 节）、可能在请求标头字段中描述的那些语义的扩展（第 5 节）、指示机器可读响应的状态代码的含义（第 6 节）,以及响应头字段中可能给出的其他控制数据和资源元数据的含义（第 7 节）。

本文档还定义了表示元数据,这些元数据描述了接收者打算如何解释有效负载、可能影响内容选择的请求标头字段以及统称为“内容协商”的各种选择算法（第 3.4 节）。

1.1.一致性和错误处理

本文档中的关键词“必须”、“不得”、“必需”、“应”、“不应”、“应该”、“不应”、“推荐”、“可以”和“可选”按照 [RFC2119] 中的描述进行解释。

[RFC7230] 的第 2.5 节中定义了有关错误处理的一致性标准和注意事项。

1.2.语法符号

本规范使用 [RFC5234] 的增强巴科斯-诺尔形式 (ABNF) 表示法和 [RFC7230] 第 7 节中定义的列表扩展,允许使用“#”运算符（类似于 * 运算符如何表示重复）。附录 C 描述了从其他文档导入的规则。附录 D 显示了收集到的语法,其中所有列表运算符都扩展为标准 ABNF 表示法。

本规范使用 [RFC6365] 中定义的术语“字符”、“字符编码方案”、“字符集”和“协议元素”。

2.资源

HTTP 请求的目标称为“资源”。 HTTP 不限制资源的性质;它仅仅定义了一个可用于与资源交互的接口。每个资源都由统一资源标识符 (URI) 标识,如 [RFC7230] 的第 2.7 节所述。

当客户端构造 HTTP/1.1 请求消息时,它会以各种形式之一发送目标 URI,如（[RFC7230] 的第 5.3 节)中所定义。收到请求时,服务器会为目标资源重建一个有效的请求 URI（[RFC7230] 的第 5.5 节）。

HTTP 的一个设计目标是将资源标识与请求语义分开,这可以通过将请求语义赋予请求方法（第 4 节)和一些请求修改标头字段（第 5 节)来实现。如果方法语义与 URI 本身隐含的任何语义之间存在冲突,如第 4.2.1 节所述,则方法语义优先。

3. 交涉

考虑到资源可以是任何东西,并且 HTTP 提供的统一接口类似于一个窗口,通过该窗口,人们只能通过与另一端的某个独立参与者的消息通信来观察和操作这样的东西,抽象是需要在我们的通信中表示（“取代”)该事物的当前或期望状态。这种抽象称为表示 [REST]。

就 HTTP 而言，“表示”是旨在反映给定资源的过去、当前或期望状态的信息,其格式可以通过协议很容易地进行通信,并且由一组表示组成元数据和潜在无限的表示数据流。

源服务器可能提供或能够生成多个表示,每个表示都旨在反映目标资源的当前状态。在这种情况下,源服务器使用某种算法来选择最适用于给定请求的表示之一,通常基于内容协商。此“选定表示”用于提供

用于评估条件请求 [RFC7232] 和构建对 GET 的 200（OK）和 304（未修改）响应的有效负载（第 4.3.1 节）的数据和元数据。

3.1.表示元数据

表示标题字段提供有关表示的元数据。当消息包含有效负载主体时，表示标头字段描述如何解释包含在有效负载主体中的表示数据。在对 HEAD 请求的响应中，表示标头字段描述了表示数据，如果同一请求是 GET，则表示数据将包含在有效负载主体中。

以下标头字段传达表示元数据：

标题字段名称	定义在...
内容类型	第 3.1.1.5 节
内容编码	第 3.1.2.2 节
内容语言	第 3.1.3.2 节
内容位置	第 3.1.4.2 节

3.1.1.处理表示数据

3.1.1.1.媒体类型

HTTP 在 Content-Type（第 3.1.1.5 节）和 Accept（第 5.3.2 节）标头字段中使用 Internet 媒体类型 [RFC2046]，以提供开放和可扩展的数据类型化和类型协商。

媒体类型定义了数据格式和各种处理模型：如何根据接收数据的每个上下文处理数据。

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

类型/子类型后面可以跟以名称=值对形式的参数。

```
parameter = token "=" ( token / quoted-string )
```

类型、子类型和参数名称标记不区分大小写。
参数值可能区分大小写,也可能不区分大小写,具体取决于参数名称的语义。参数的存在与否可能对媒体类型的处理很重要,这取决于它在媒体类型注册表中的定义。

与令牌生产相匹配的参数值可以作为令牌或在引号字符串中传输。引用和未引用的值是等效的。例如,以下示例都是等价的,但为了保持一致性,首选第一个:

```
文本/html;字符集=utf-8 文本/html;字符
集=UTF-8
文本/HTML;Charset=  utf-8  文本/html;字
符集 =“utf-8”
```

互联网媒体类型应该根据 [BCP13] 中定义的程序向 IANA 注册。

注意:与其他标头字段中的一些类似构造不同,媒体类型参数不允许在“=”字符周围使用空格(甚至是“坏”空格)。

3.1.1.2。字符集

HTTP 使用字符集名称来指示或协商文本表示的字符编码方案 [RFC6365]。字符集由不区分大小写的标记标识。

```
字符集 = 令牌
```

根据 [RFC2978] 中定义的程序,字符集名称应该在 IANA“字符集”注册表 (<http://www.iana.org/assignments/character-sets>) 中注册。

3.1.1.3.规范化和文本默认值

互联网媒体类型以规范形式注册,以便在具有不同本机编码格式的系统之间进行互操作。

由于多用途 Internet 邮件扩展 (MIME) [RFC2045] 描述的许多相同原因,通过 HTTP 选择或传输的表示应该采用规范形式。然而,电子邮件部署的性能特征(即,存储消息并将消息转发给对等方)与 HTTP 和 Web(基于服务器的信息服务)的常见特征有很大不同。

此外,为了与旧邮件传输协议兼容而对 MIME 进行的限制不适用于 HTTP(请参阅附录 A)。

MIME 的规范形式要求“文本”类型的媒体子类型使用 CRLF 作为文本换行符。 HTTP 允许传输带有纯 CR 或 LF 的文本媒体单独代表一个换行符,当这样的换行符对于整个表示是一致的。 HTTP 发送方可以生成,并且接收方必须能够解析由 CRLF、裸 CR 或裸 LF 组成的文本媒体中的换行符。此外,HTTP 中的文本媒体不限于分别将八位字节 13 和 10 用于 CR 和 LF 的字符集。这种关于换行符的灵活性仅适用于已分配“文本”媒体类型的表示中的文本;它不适用于有效负载主体之外的“多部分”类型或 HTTP 元素(例如,标头字段)。

如果表示使用内容编码进行编码,则基础数据在编码之前应该采用上面定义的形式。

3.1.1.4。多部分类型

MIME 提供了多种“多部分”类型 将一个或多个表示封装在单个消息主体中。所有多部分类型都共享一个通用语法,如 [RFC2046] 的第 5.1.1 节中所定义,并包含一个边界参数作为媒体类型值的一部分。消息体本身就是一个协议元素;发送者必须只生成 CRLF 来表示正文部分之间的换行符。

HTTP 消息框架不使用多部分边界作为消息正文长度的指示符,尽管生成或处理有效负载的实现可能会使用它。例如,“multipart/form-data”类型通常用于在请求中携带表单数据,如 [RFC2388] 中所述,而“multipart/ byteranges”类型由该规范定义用于某些 206(部分内容)响应 [RFC7233]。

3.1.1.5。内容类型

“Content-Type”头字段指示相关表示的媒体类型:消息有效负载中包含的表示或所选表示,由消息语义确定。在 Content-Encoding 指示的任何内容编码被解码后,指示的媒体类型定义了数据格式以及接收者打算如何在接收到的消息语义范围内处理该数据。

内容类型 = 媒体类型

媒体类型在第 3.1.1.1 节中定义。该字段的一个例子是

内容类型:文本/html;字符集=ISO-8859-4

生成包含有效负载主体的消息的发送方应该在该消息中生成 Content-Type 头字段,除非发送方不知道所包含的表示的预期媒体类型。如果 Content-Type 头字段不存在,接收者可以假定媒体类型为“application/octet-stream”

([RFC2046],第 4.5.1 节)或检查数据以确定其类型。

在实践中,资源所有者并不总是正确配置他们的源服务器来为给定的表示提供正确的内容类型,结果是一些客户端将检查有效负载的内容并覆盖指定的类型。这样做的客户端冒着得出错误结论的风险,这可能会暴露额外的安全风险(例如,“权限升级”)。此外,不可能通过检查数据格式来确定发送者的意图:许多数据格式匹配仅在处理语义上不同的多种媒体类型。鼓励实施者提供一种在使用时禁用这种“内容嗅探”的方法。

3.1.2.压缩或完整性编码

3.1.2.1.内容编码

内容编码值指示已经或可以应用于表示的编码转换。内容编码主要用于允许对表示进行压缩或以其他方式进行有用的转换,而不会丢失其底层媒体类型的身份并且不会丢失信息。通常,表示以编码形式存储,直接传输,并且仅由最终接收者解码。

内容编码 = 令牌

所有内容编码值都不区分大小写,并且应该在“HTTP 内容编码注册表”中注册,如第 8.4 节中所定义。它们用于 Accept-Encoding(第 5.3.4 节)和 Content-Encoding(第 3.1.2.2 节)头字段。

本规范定义了以下内容编码值：

compress（和 x-compress）：参见 [RFC7230] 的第 4.2.1 节。

deflate：参见 [RFC7230] 的第 4.2.2 节。

gzip（和 x-gzip）：参见 [RFC7230] 的第 4.2.3 节。

3.1.2.2.内容编码

“Content-Encoding”报头字段指示哪些内容编码已应用于表示，超出了媒体类型固有的编码，因此必须应用哪些解码机制才能获得 Content-引用的媒体类型中的数据类型标题字段。 Content-Encoding 主要用于允许压缩表示的数据而不丢失其底层媒体类型的标识。

内容编码= 1#内容编码

它的一个使用例子是

内容编码：gzip

如果一种或多种编码已应用于表示，则应用编码的发送方必须生成一个 Content-Encoding 标头字段，该字段按应用顺序列出内容编码。关于编码参数的附加信息可以由本规范未定义的其他头字段提供。

与 Transfer-Encoding（[RFC7230] 的第 3.3.1 节）不同，Content-Encoding 中列出的编码是表示的一个特征；表示是根据编码形式定义的，除非元数据定义中另有说明，否则关于表示的所有其他元数据都是关于编码形式的。

通常，表示仅在渲染或类似使用之前才被解码。

如果媒体类型包含固有编码，例如始终压缩的数据格式，则该编码不会在 Content-Encoding 中重述，即使它恰好与其中一种内容编码的算法相同。这样的内容编码只有在出于某种奇怪的原因被第二次应用以形成表示时才会被列出。同样，源服务器可能会选择将相同的数据发布为多种表示形式，不同之处仅在于编码是否定义为 Content-Type 的一部分

或内容编码,因为某些用户代理在处理每个响应时会有不同的行为（例如,打开“另存为...”对话框而不是自动解压缩和呈现内容）。

如果请求消息中的表示具有不可接受的内容编码,则源服务器可以使用状态代码 415（不支持的媒体类型)进行响应。

3.1.3.观众语言

3.1.3.1.语言标签

语言标签,如 [RFC5646] 中所定义,识别人类口头、书面或以其他方式传达的自然语言,用于与其他人交流信息。计算机语言被明确排除在外。

HTTP 在 Accept-Language 和 Content-Language 标头字段中使用语言标签。 Accept-Language 使用第 5.3.5 节中定义的更广泛的语言范围产生式,而 Content-Language 使用下面定义的语言标签产生式。

```
language-tag = <语言标签,参见 [RFC5646],第 2.1 节>
```

语言标签是一个或多个不区分大小写的子标签的序列,每个子标签由连字符（“-”、%x2D)分隔。在大多数情况下,语言标签由一个主要语言子标签组成,该子标签标识一个广泛的相关语言家族（例如，“en”= 英语）,它后面可选地跟有一系列细化或缩小该语言范围的子标签（例如， “en-CA”= 在加拿大交流的各种英语）。语言标记中不允许使用空格。示例标签包括：

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

有关更多信息,请参阅 [RFC5646]。

3.1.3.2.内容语言

“Content-Language”标题字段描述了表示的预期受众的自然语言。请注意,这可能不等同于表示中使用的所有语言。

```
内容语言= 1#language-tag
```

语言标签在第 3.1.3.1 节中定义。 Content-Language 的主要目的是允许用户根据自己的首选语言来识别和区分表示。

因此,如果内容仅供丹麦语读者使用,则相应的字段是

内容语言:da

如果未指定 Content-Language,则默认内容适用于所有语言的受众。这可能意味着发件人不认为它特定于任何自然语言,或者发件人不知道它打算用于哪种语言。

可以为面向多个受众的内容列出多种语言。例如,同时以毛利语原版和英文版呈现的《怀唐伊条约》的翻版要求

内容语言:mi, en

然而,仅仅因为一种表示中存在多种语言并不意味着它适用于多种语言的受众。一个例子是初学者的语言入门,例如“拉丁语第一课”,它显然旨在供英语读者使用。在这种情况下,Content-Language 应该只包含“en”。

Content-Language 可以应用于任何媒体类型 它不限于文本文档。

3.1.4.鉴别

3.1.4.1.识别表示

当在消息有效负载中传输完整或部分表示时,通常希望发送方提供或接收方确定与该表示对应的资源的标识符。

对于请求消息:

o 如果请求有一个 Content-Location 头字段,那么发送者断言有效载荷是由 Content-Location 字段值标识的资源的表示。但是,除非可以通过其他方式(本规范未定义)对其进行验证,否则此类断言是不可信的。该信息对于修订历史链接可能仍然有用。

o 否则,有效载荷是无法识别的。

对于响应消息,将按顺序应用以下规则,直到找到匹配项:

1. 如果请求方法是 GET 或 HEAD 并且响应状态代码是 200（OK）、204（无内容）、206（部分内容)或 304（未修改）,则有效负载表示由有效请求 URI（[RFC7230] 的第 5.5 节）。

2. 如果请求方法是 GET 或 HEAD 并且响应状态代码是 203（非权威信息）,则有效负载是中介提供的目标资源的潜在修改或增强表示。

3. 如果响应有一个 Content-Location 头字段,并且它的字段值是对与有效请求 URI 相同的 URI 的引用,则有效负载是有效请求 URI 标识的资源的表示。

4. 如果响应有一个 Content-Location 头字段及其 field-value 是对与有效请求 URI 不同的 URI 的引用,然后发送方断言有效负载是由 Content-Location field-value 标识的资源的表示。但是,除非可以通过其他方式（本规范未定义)对其进行验证,否则此类断言是不可信的。

5.否则,载荷不明。

3.1.4.2.内容位置

“Content-Location”报头字段引用一个 URI,该 URI 可用作与此消息有效负载中的表示相对应的特定资源的标识符。换句话说,如果在生成此消息时在此 URI 上执行 GET 请求,则 200 (OK) 响应将包含与此消息中作为负载包含的相同表示。

内容位置 = 绝对 URI / 部分 URI

Content-Location 值不能替代有效的 Request URI（[RFC7230] 的第 5.5 节）。它是表示元数据。它与 [RFC2557] 第 4 节中为 MIME 正文部分定义的同名标头字段具有相同的语法和语义。然而,它在 HTTP 消息中的出现对 HTTP 接收者有一些特殊的含义。

如果 Content-Location 包含在 2xx（成功)响应消息中并且其值引用（在转换为绝对形式后)与有效请求 URI 相同的 URI,则接收者可以将有效负载视为当前表示该资源在消息发起日期指示的时间。对于 GET（第 4.3.1 节)或 HEAD（第 4.3.2 节)请求,这与服务器未提供 Content-Location 时的默认语义相同。对于像 PUT（第 4.3.4 节)或 POST（第 4.3.3 节)这样的状态更改请求,它意味着服务器的响应包含该资源的新表示,从而将其与可能仅报告有关操作的表示区分开来（例如,“它成功了！”）。这允许创作应用程序更新其本地副本,而无需后续 GET 请求。

如果 Content-Location 包含在 2xx（成功)响应消息中,并且其字段值引用与有效请求 URI 不同的 URI,则源服务器声称该 URI 是对应于所包含的不同资源的标识符表示。只有当两个标识符共享相同的资源所有者时,这样的声明才能被信任,这不能通过 HTTP 以编程方式确定。

o 对于 GET 或 HEAD 请求的响应,这表明有效请求 URI 指的是受内容协商约束的资源,Content-Location 字段值是所选表示的更具体的标识符。

o 对于状态改变方法的 201（已创建)响应,a
与 Location 字段值相同的 Content-Location 字段值指示此有效负载是新创建资源的当前表示。

o 否则,这样的 Content-Location 表示此有效负载是报告所请求操作状态的表示,并且相同的报告在给定的 URI 处可用（用于将来使用 GET 访问）。例如,通过 POST 请求进行的购买交易可能包含收据文档作为 200 (OK) 响应的有效负载； Content-Location 字段值提供了一个标识符,用于将来检索同一收据的副本。

在请求消息中发送 Content-Location 的用户代理声明其值是指用户代理最初从何处获得所附表示的内容（在该用户代理进行任何修改之前）。换句话说,用户代理正在提供指向原始表示源的反向链接。

接收请求消息中的 Content-Location 字段的源服务器必须将信息视为临时请求上下文,而不是作为表示的一部分逐字保存的元数据。原始服务器可以使用该上下文来指导处理请求或将其保存以供其他用途,例如在源链接或版本控制元数据中。但是,源服务器不得使用此类上下文信息来更改请求语义。

例如,如果客户端对协商资源发出 PUT 请求,并且源服务器接受该 PUT(没有重定向),则该资源的新状态应与该 PUT 中提供的一种表示一致;Content-Location 不能用作反向内容选择标识符的一种形式来仅更新协商表示之一。如果用户代理想要后一种语义,它会直接将 PUT 应用于 Content-Location URI。

3.2.表示数据

与 HTTP 消息关联的表示数据要么作为消息的有效负载主体提供,要么由消息语义和有效请求 URI 引用。表示数据采用由表示元数据报头字段定义的格式和编码。

表示数据的数据类型通过标题字段 Content-Type 和 Content-Encoding 确定。这些定义了一个两层的有序编码模型:

表示数据 :=内容编码(内容类型(位))

3.3.有效载荷语义

一些 HTTP 消息将完整或部分表示作为消息“有效负载”进行传输。在某些情况下,有效载荷可能仅包含相关表示的头部字段(例如,对 HEAD 的响应)或仅包含表示数据的某些部分(例如,206(部分内容)状态代码)。

请求中有效载荷的用途由方法语义定义。例如,如果成功应用请求,PUT 请求(第 4.3.4 节)的有效负载中的表示表示目标资源的期望状态,而 POST 请求(第 4.3.3 节)的有效负载中的表示表示目标资源要处理的信息。

在响应中,负载的目的由请求方法和响应状态代码定义。例如,对 GET（第 4.3.1 节)的 200（OK)响应的有效负载表示目标资源的当前状态,正如在消息发起日期（第 7.1.1.2 节)时观察到的那样,而有效负载对 POST 的响应中的相同状态代码可能表示处理结果或应用处理后目标资源的新状态。带有错误状态代码的响应消息通常包含一个表示错误条件的有效负载,这样它就描述了错误状态以及建议的后续步骤来解决它。

专门描述有效负载而非关联表示的报头字段称为“有效负载报头字段”。有效负载标头字段在本规范的其他部分中定义,因为它们对消息解析有影响。

标题字段名称	定义在...
内容长度	[RFC7230] 第 3.3.2 节
内容范围	[RFC7233] 第 4.2 节
预告片	[RFC7230] 的第 4.4 节
传输编码	[RFC7230] 第 3.3.1 节

3.4.内容协商

当响应传递有效载荷信息时,无论是指示成功还是错误,源服务器通常有不同的方式来表示该信息;例如,以不同的格式、语言或编码。同样,不同的用户或用户代理可能具有不同的能力、特征或偏好,这些可能会影响在可用的表示中最好交付的表示。为此,HTTP 提供了内容协商机制。

本规范定义了两种可以在协议中可见的内容协商模式：“主动”模式,其中服务器根据用户代理声明的偏好选择表示,以及“反应式”协商,其中服务器为以下内容提供表示列表可供选择的用户代理。内容协商的其他模式包括“条件内容”,其中表示由基于用户代理参数选择性呈现的多个部分组成,“活动内容”,其中表示包含一个脚本,该脚本根据用户代理特征和“透明内容协商”（[RFC2295]）,其中内容

选择由中介进行。这些模式并不相互排斥,每个模式都在适用性和实用性方面有所取舍。

请注意,在所有情况下,HTTP 都不知道资源语义。
随着时间的推移和内容协商的不同维度,源服务器响应请求的一致性,以及因此随着时间的推移观察到的资源表示的“相同性”,完全由选择
或生成这些响应的任何实体或算法决定。 HTTP 不关注幕后的人。

3.4.1.主动谈判

当用户代理在请求中发送内容协商首选项以鼓励位于服务器的算法选择首选表示时,这称为主动协商(也称为服务器驱动协商)。选择是基于与请求中提供的各种信息(包括第 5.3 节的显式协商字段和隐式协商字段)相比,响应的可用表示(它可能变化的维度,例如语言、内容编码等)特性,例如客户端的网络地址或 User-Agent 字段的一部分。

当从可用表示中进行选择的算法难以向用户代理描述时,或者当服务器希望将其“最佳猜测”连同第一个响应(希望避免回合)发送
给用户代理时,主动协商是有利的如果“最佳猜测”对用户来说足够好,则后续请求的行程延迟)。为了改进服务器的猜测,用户代理可以发送描述
其偏好的请求标头字段。

主动协商有严重的缺点:

o 服务器不可能准确地确定对于任何给定用户什么可能是“最好的”,因为这需要完全了解用户代理的能力和响应的预期用途(例如,用户是否想要在
屏幕上查看还是打印在纸上?);

o 让用户代理在每个请求中描述它的能力可能是非常低效的(假设只有一小部分响应有多种表示)并且对用户隐私有潜在风险;

o 它使源服务器的实现和生成对请求的响应的算法变得复杂;和,

o 它限制了共享缓存响应的可重用性。

用户代理不能依赖于始终如一地遵守主动协商首选项,因为源服务器可能不会对请求的资源实施主动协商,或者可能决定发送不符合用户代理首选项的响应比发送 406 (不可接受)回应。

Vary 报头字段（第 7.1.4 节)通常在响应中发送以进行主动协商,以指示在选择算法中使用了请求信息的哪些部分。

3.4.2.被动谈判

对于反应式协商（又名,代理驱动的协商）,用户代理在收到来自源服务器的初始响应后执行最佳响应表示的选择（无论状态代码如何）,其中包含用于替代表示的资源列表。如果用户代理对初始响应表示不满意,它可以对根据列表中包含的元数据选择的一个或多个替代资源执行 GET 请求,以获得该响应的不同形式的表示。备选方案的选择可能由用户代理自动执行,或由用户从生成的（可能是超文本)菜单中选择手动执行。

请注意,上面提到的是响应的表示,通常不是资源的表示。仅当提供替代表示的响应具有作为目标资源表示的语义（例如,对 GET 请求的 200（OK)响应)或具有以下语义时,替代表示才被视为目标资源的表示提供指向目标资源替代表示的链接（例如,对 GET 请求的 300（多项选择）响应）。

服务器可能会选择不发送初始表示,而不是备选列表,从而表明用户代理的反应式协商是首选。例如,在具有 300（多项选择)和 406（不可接受）状态代码的响应中列出的备选方案包括有关可用表示的信息,以便用户或用户代理可以通过做出选择来做出反应。

当响应在常用维度（例如类型、语言或编码)上发生变化时,当源服务器无法通过检查请求来确定用户代理的能力时,以及通常当公共缓存用于分发时,反应式协商是有利的服务器负载并减少网络使用。

反应式协商的缺点是向用户代理传输替代列表,如果在标头部分传输,这会降低用户感知的延迟,并且需要第二次请求以获得替代表示。此外,本规范没有定义支持自动选择的机制,尽管它不阻止将这种机制开发为扩展。

4.请求方法

4.1.概述

请求方法令牌是请求语义的主要来源;它表明客户提出此请求的目的以及客户对成功结果的期望。

如果这些额外的语义不与方法冲突,则请求方法的语义可能会通过请求中出现的某些标头字段的语义进一步专门化（第 5 节）。例如,客户端可以发送条件请求标头字段（第 5.2 节)以使请求的操作以目标资源的当前状态为条件（[RFC7232]）。

```
方法=令牌
```

HTTP 最初设计为可用作分布式对象系统的接口。请求方法被设想为将语义应用于目标资源,其方式与在已识别对象上调用定义的方法将应用语义的方式大致相同。方法标记区分大小写,因为它可能被用作使用区分大小写方法的基于对象的系统的网关名字。

与分布式对象不同,HTTP 中的标准化请求方法不是特定于资源的,因为统一接口在基于网络的系统 [REST] 中提供了更好的可见性和重用性。一旦定义,标准化方法在应用于任何资源时应该具有相同的语义,尽管每个资源自行决定是否实现或允许这些语义。

本规范定义了 HTTP 中常用的许多标准化方法,如下表所示。按照惯例,标准化方法以全大写 US-ASCII 字母定义。

```
+--------+----------------------------------- ----------+--------+
|方法 |描述 |秒。 |
+--------+----------------------------------- ----------+--------+
|获取 |传输目标的当前表示 | 4.3.1 | | |资源。 | |头 |与 GET 相同,但只传输状态行 | 4.3.2 | | |和标题部分。 | | |发布|对 | 执行特定于资源的处理4.3.3 | | |请
求负载。 | | |放 |替换 | [illegible]
|目标资源。
|
+--------+----------------------------------- ----------+--------+
```

所有通用服务器必须支持 GET 和 HEAD 方法。
所有其他方法都是可选的。

本规范范围之外的其他方法已标准化用于 HTTP。所有此类方法都应在 IANA 维护的“超文本传输协议 (HTTP) 方法注册表”中注册，如第 8.1 节中所定义。

目标资源允许的方法集可以在 Allow 头字段中列出（第 7.4.1 节）。但是，允许的方法集可以动态更改。当接收到原始服务器无法识别或未实现的请求方法时，原始服务器应该使用 501（未实现)状态代码进行响应。当接收到源服务器已知但目标资源不允许的请求方法时，源服务器应该响应 405（方法不允许)状态代码。

4.2.通用方法属性

4.2.1.安全方法

如果请求方法定义的语义本质上是只读的，则请求方法被认为是“安全的”；即，客户端不请求也不期望由于对目标资源应用安全方法而导致源服务器上的任何状态更改。同样，合理使用安全方法预计不会对源服务器造成任何伤害、财产损失或异常负担。

安全方法的这种定义不会阻止实现包含潜在有害的行为、不完全只读的行为或在调用安全方法时导致副作用的行为。然而,重要的是客户没有要求额外的行为并且不能为此负责。例如,大多数服务器都会在每次响应完成时附加请求信息以访问日志文件,无论使用何种方法,这被认为是安全的,即使日志存储可能已满并导致服务器崩溃。同样,通过选择 Web 上的广告发起的安全请求通常会产生向广告帐户收费的副作用。

在本规范定义的请求方法中,GET、HEAD、OPTIONS 和 TRACE 方法被定义为安全的。

区分安全和不安全方法的目的是让自动检索过程(蜘蛛)和缓存性能优化(预取)工作而不必担心造成伤害。此外,它允许用户代理在处理可能不受信任的内容时对不安全方法的自动使用应用适当的约束。

用户代理在向用户呈现潜在操作时应该区分安全和不安全方法,以便用户可以在请求之前意识到不安全操作。

当构建资源时,有效请求 URI 中的参数具有选择操作的效果,资源所有者有责任确保操作与请求方法语义一致。例如,基于 Web 的内容编辑软件通常在查询参数内使用操作,例如“page?do=delete”。如果此类资源的目的是执行不安全的操作,则资源所有者必须在使用安全请求方法访问时禁用或不允许该操作。如果不这样做,当自动化进程为了链接维护、预取、构建搜索索引等而对每个 URI 引用执行 GET 时,将导致不幸的副作用。

4.2.2.幂等方法

如果使用该方法的多个相同请求对服务器的预期效果与单个此类请求的效果相同,则请求方法被认为是“幂等的”。在本规范定义的请求方法中,PUT、DELETE 和安全请求方法是幂等的。

与安全的定义一样,幂等属性仅适用于用户请求的内容;服务器可以自由地单独记录每个请求,保留修订控制历史记录,或为每个幂等请求实现其他非幂等副作用。

幂等方法之所以与众不同,是因为如果在客户端能够读取服务器响应之前发生通信故障,则可以自动重复请求。例如,如果客户端发送 PUT 请求,并且在收到任何响应之前关闭了底层连接,则客户端可以建立新连接并重试幂等请求。它知道重复请求将产生相同的预期效果,即使原始请求成功,但响应可能不同。

4.2.3.可缓存的方法

请求方法可以定义为“可缓存”的,以表明允许存储对它们的响应以供将来重用;具体要求参见 [RFC7234]。通常,不依赖于当前或权威响应的安全方法被定义为可缓存的;该规范将 GET、HEAD 和 POST 定义为可缓存的,尽管绝大多数缓存实现仅支持 GET 和 HEAD。

4.3.方法定义

4.3.1.得到

GET 方法请求传输目标资源的当前选定表示。 GET 是信息检索的主要机制,也是几乎所有性能优化的重点。

因此,当人们谈到通过 HTTP 检索一些可识别信息时,他们通常指的是发出 GET 请求。

人们很容易将资源标识符视为远程文件系统路径名,并将表示视为此类文件内容的副本。事实上,这就是实现了多少资源(有关安全考虑,请参阅第 9.1 节)。然而,在实践中没有这样的限制。资源的 HTTP 接口很可能被实现为内容对象树、各种数据库记录的编程视图或其他信息系统的网关。即使当 URI 映射机制绑定到文件系统时,源服务器也可能被配置为将请求作为输入来执行文件,并将输出作为表示发送,而不是直接传输文件。无论如何,只有源服务器需要知道它的每个资源如何

标识符对应于一个实现以及每个实现如何在对 GET 的响应中设法选择和发送目标资源的当前表示。

客户端可以将 GET 的语义更改为“范围请求”,通过在请求中发送 Range 标头字段（[RFC7233]）,请求仅传输所选表示的某些部分。

GET 请求消息中的负载没有定义的语义;在 GET 请求上发送有效负载主体可能会导致某些现有实现拒绝该请求。

对 GET 请求的响应是可缓存的;缓存可以使用它来满足后续的 GET 和 HEAD 请求,除非 Cache-Control 头字段（[RFC7234] 的第 5.2 节）另有说明。

4.3.2.头

HEAD 方法与 GET 相同,除了服务器不得在响应中发送消息主体（即,响应在标头部分的末尾终止）。服务器应该发送相同的头字段来响应 HEAD 请求,如果请求是 GET,它会发送相同的头字段,除了有效负载头字段（第 3.3 节)可以省略。此方法可用于在不传输表示数据的情况下获取有关所选表示的元数据,并且通常用于测试超文本链接的有效性、可访问性和最近修改。

HEAD 请求消息中的有效负载没有定义的语义;在 HEAD 请求上发送有效负载主体可能会导致某些现有实现拒绝该请求。

对 HEAD 请求的响应是可缓存的;缓存可以使用它来满足后续的 HEAD 请求,除非 Cache-Control 头字段（[RFC7234] 的第 5.2 节)另有说明。 HEAD 响应也可能对先前缓存的 GET 响应产生影响;请参阅 [RFC7234] 的第 4.3.5 节。

4.3.3.邮政

POST 方法请求目标资源根据资源自己的特定语义处理包含在请求中的表示。例如,POST 用于以下功能（以及其他功能）：

o 提供数据块,例如输入到 HTML 中的字段
形式,到数据处理过程;

o 将消息发布到公告板、新闻组、邮件列表、
博客或类似的文章组；

o 创建一个尚未被源服务器识别的新资源；和

o 将数据附加到资源的现有表示中。

源服务器根据处理POST请求的结果选择合适的状态码来指示响应语义；本规范定义的几乎所有状态代码都可能在对 POST 的响应中收到（例外情况是 206（部分内容）、304（未修改）和 416（范围不可满足））。

如果作为成功处理 POST 请求的结果在源服务器上创建了一个或多个资源，源服务器应该发送一个 201（已创建）响应，其中包含一个 Location 头字段，为创建的主要资源提供标识符（第 7.1 节） .2) 以及在引用新资源时描述请求状态的表示。

对 POST 请求的响应只有在包含明确的新鲜度信息时才可缓存（请参阅 [RFC7234] 的第 4.2.1 节）。

但是，POST 缓存并未得到广泛实施。对于源服务器希望客户端能够缓存 POST 的结果以供稍后 GET 重用的情况，源服务器可以发送包含结果和 Content-Location 的 200（OK）响应与 POST 的有效请求 URI（第 3.1.4.2 节）具有相同值的标头字段。

如果处理 POST 的结果等同于现有资源的表示，源服务器可以通过发送 303（参见其他）响应将用户代理重定向到该资源，并在 Location 字段中包含现有资源的标识符。这具有为用户代理提供资源标识符并通过更适合共享缓存的方法传输表示的好处，但如果用户代理尚未缓存表示，则会以额外请求为代价。

4.3.4.放

PUT 方法请求创建目标资源的状态或将其替换为请求消息负载中包含的表示所定义的状态。给定表示的成功 PUT 表明对同一目标资源的后续 GET 将导致在 200 (OK) 响应中发送等效表示。但是，不能保证

这样的状态变化是可以观察到的,因为在接收到任何后续 GET 之前,目标资源可能会被其他用户代理并行操作,或者可能会受到源服务器的动态处理。成功的响应仅意味着用户代理的意图在原始服务器处理时已实现。

如果目标资源没有当前表示并且 PUT 成功创建了一个,则源服务器必须通过发送 201（已创建)响应通知用户代理。如果目标资源确实有当前表示并且该表示已根据封闭表示的状态成功修改,则源服务器必须发送 200（OK)或 204（无内容)响应以指示成功完成要求。

源服务器应该忽略在 PUT 请求中接收到的无法识别的头字段（即,不要将它们保存为资源状态的一部分）。

源服务器应该验证 PUT 表示是否符合服务器对目标资源的任何约束,这些约束不能或不会被 PUT 更改。当源服务器使用与 URI 相关的内部配置信息以便在 GET 响应上设置表示元数据的值时,这一点尤为重要。当 PUT 表示与目标资源不一致时,源服务器应该通过转换表示或更改资源配置使它们一致,或者以包含足够信息的适当错误消息响应,以解释表示不合适的原因。建议使用 409（冲突)或 415（不支持的媒体类型)状态代码,后者特定于对 Content-Type 值的约束。

例如,如果目标资源配置为始终具有
“文本/html”的内容类型和被 PUT 的表示有一个
“图像/jpeg”的内容类型,源服务器应该执行以下操作之一：

一种。重新配置目标资源以反映新的媒体类型；

b.在将 PUT 表示形式保存为新的资源状态之前,将其转换为与资源一致的格式;或者,

C。使用 415（不受支持的媒体类型)响应拒绝请求,指示目标资源仅限于“text/html”,可能包括指向不同资源的链接,该资源将是新表示的合适目标。

除了用户代理请求的意图和源服务器响应的语义可以表达的内容之外,HTTP 没有准确定义 PUT 方法如何影响源服务器的状态。它没有定义资源可能是什么,在任何意义上,超出了通过 HTTP 提供的接口。它没有定义资源状态是如何“存储”的,也没有定义这种存储如何因资源状态的变化而改变,也没有定义源服务器如何将资源状态转换为表示。一般来说,资源接口背后的所有实现细节都是服务器有意隐藏的。

源服务器不得在对 PUT 的成功响应中发送验证器标头字段（第 7.2 节）,例如 ETag 或 Last-Modified 字段,除非请求
的表示数据在保存时未对正文应用任何转换（即,资源的新的表示数据与 PUT 请求中接收到的表示数据相同)并且验证器字段
值反映了新的表示。此要求允许用户代理知道它在内存中的表示体何时由于 PUT 而保持最新,因此不需要从源服务器再次检索,并且新的验证器在响应中收到可用于未来的条件请求,以防止意外覆盖（第 5.2 节）。

POST 和 PUT 方法之间的根本区别体现在封闭表示的不同意图上。

POST 请求中的目标资源旨在根据资源自身的语义处理封闭表示,而 PUT 请求中的封闭表示被定义为替换目标资源的状态。因此,PUT 的
意图是幂等的并且对中介可见,即使确切的效果只有源服务器知道。

PUT 请求的正确解释假设用户代理知道需要哪个目标资源。代表客户端选择适当 URI 的服务,在收到状态更改请求后,应该使用 POST 方法而不是 PUT 方法来实现。

如果源服务器不会将请求的 PUT 状态更改为目标资源,而是希望将其应用于不同的资源,例如当资源已移动到不同的 URI 时,那么源服务器必须发送适当的 3xx （重定向)响应;然后用户代理可以自己决定是否重定向请求。

应用于目标资源的 PUT 请求可能会对其他资源产生副作用。例如,一篇文章可能有一个用于标识“当前版本”（资源)的 URI,该 URI 与标识每个特定版本（不同资源)的 URI 是分开的

在某一时刻与当前版本资源共享相同的状态）。因此,除了更改目标资源的状态之外,对“当前版本”URI 的成功 PUT 请求可能会创建新版本资源,并且还可能导致在相关资源之间添加链接。

允许在给定目标资源上进行 PUT 的源服务器必须向包含 Content-Range 标头字段（[RFC7233] 的第 4.2 节)的 PUT 请求发送 400（错误请求)响应,因为有效负载可能是部分内容已被错误地 PUT 为完整表示。部分内容更新可以通过将单独标识的资源作为目标,其状态与较大资源的一部分重叠,或者使用专门为部分更新定义的不同方法（例如,[RFC5789] 中定义的 PATCH 方法）。

对 PUT 方法的响应不可缓存。如果一个成功的 PUT 请求通过一个缓存,缓存中存储了一个或多个有效请求 URI 的响应,这些存储的响应将失效（参见 [RFC7234] 的第 4.4 节）。

4.3.5.删除

DELETE 方法请求源服务器删除目标资源与其当前功能之间的关联。实际上,这种方法类似于UNIX中的rm命令:它表达了对源站URI映射的删除操作,而不是期望删除之前关联的信息。

如果目标资源具有一个或多个当前表示,它们可能会或可能不会被源服务器销毁,并且相关存储可能会或可能不会被回收,这完全取决于资源的性质及其源服务器的实现（这超出了本规范的范围）。

同样,作为 DELETE 的结果,资源的其他实现方面可能需要停用或存档,例如数据库或网关连接。通常,假设源服务器只允许对它具有完成删除的规定机制的资源进行 DELETE。

允许使用 DELETE 方法的资源相对较少 它的主要用途是用于远程创作环境,在这种环境中,用户对其效果有一定的指导。例如,先前使用 PUT 请求创建的资源,或在对 POST 请求的 201（已创建)响应后通过 Location 标头字段标识的资源,可能允许相应的 DELETE 请求撤消这些操作。同样,实现的自定义用户代理实现

创作功能,例如使用 HTTP 进行远程操作的版本控制客户端,可能会基于服务器的 URI 空间已被设计为与版本存储库相对应的假设来使用 DELETE。

如果成功应用 DELETE 方法,如果操作可能成功但尚未执行,源服务器应该发送 202（已接受)状态代码,如果操作已执行且没有进一步执行,则发送 204（无内容)状态代码将提供信息,或 200 (OK) 状态代码（如果操作已执行且响应消息包含描述状态的表示）。

DELETE 请求消息中的有效载荷没有定义的语义;在 DELETE 请求上发送有效负载主体可能会导致某些现有实现拒绝该请求。

对 DELETE 方法的响应不可缓存。如果一个 DELETE 请求通过一个缓存,缓存中存储了一个或多个针对有效请求 URI 的响应,这些存储的响应将失效（参见 [RFC7234] 的第 4.4 节）。

4.3.6.连接

CONNECT 方法请求接收者建立到由请求目标标识的目标源服务器的隧道,如果成功,此后将其行为限制为双向盲目转发数据包,直到隧道关闭。隧道通常用于通过一个或多个代理创建端到端的虚拟连接,然后可以使用 TLS（传输层安全性,[RFC5246])对其进行保护。

CONNECT 仅用于对代理的请求。为自己接收 CONNECT 请求的源服务器可以用 2xx（成功)状态代码进行响应,以指示已建立连接。然而,大多数源服务器没有实现 CONNECT。

发送 CONNECT 请求的客户端必须发送请求目标的授权形式（[RFC7230] 的第 5.3 节）;即,请求目标仅包含隧道目的地的主机名和端口号,以冒号分隔。例如,

```
连接 server.example.com:80 HTTP/1.1 主机：
server.example.com:80
```

接收代理可以通过直接连接到请求目标来建立隧道,或者如果配置为使用另一个代理,则可以通过将 CONNECT 请求转发到下一个入站代理来建立隧道。

任何 2xx（成功)响应表明发送者（以及所有

入站代理)将在结束成功响应的标头部分的空行之后立即切换到隧道模式;在该空行之后收到的数据来自请求目标标识的服务器。除成功响应之外的任何响应都表示隧道尚未形成并且连接仍由 HTTP 管理。

当隧道中介检测到任何一方已关闭其连接时,隧道将关闭:中介必须尝试将来自封闭端的任何未完成数据发送到另一端,关闭两个连接,然后丢弃任何未交付的剩余数据。

代理身份验证可用于建立创建隧道的权限。例如,

```
连接 server.example.com:80 HTTP/1.1 主机:
server.example.com:80 代理授权:基本 aGVsbG86d29ybGQ=
```

建立通往任意服务器的隧道存在重大风险,特别是当目标是众所周知或保留的 TCP 端口,而不是用于 Web 流量时。例如,连接到“example.com:25”的请求目标将建议代理连接到 SMTP 流量的保留端口;如果允许,这可能会诱使代理转发垃圾邮件。支持 CONNECT 的代理应该将其使用限制在一组有限的已知端口或可配置的安全请求目标白名单中。

服务器不得在 2xx(成功)响应中向 CONNECT 发送任何 Transfer-Encoding 或 Content-Length 标头字段。客户端必须忽略在对 CONNECT 的成功响应中收到的任何 Content-Length 或 Transfer-Encoding 标头字段。

CONNECT 请求消息中的有效载荷没有定义的语义;在 CONNECT 请求上发送有效负载主体可能会导致某些现有实现拒绝该请求。

对 CONNECT 方法的响应不可缓存。

4.3.7.选项

OPTIONS 方法请求关于目标资源可用的通信选项的信息,在源服务器或中间中介。此方法允许客户端确定与资源关联的选项和/或要求,或服务器的功能,而不暗示资源操作。

带有星号（“*”）作为请求目标（[RFC7230] 的第 5.3 节）的 OPTIONS 请求通常适用于服务器而不是特定资源。由于服务器的通信选项
通常取决于资源，因此“*”请求仅用作“ping”或“no-op”类型的方法；除了允许客户端测试服务器的功能外，它什么都不做。例如，这可用于测试
代理是否符合 HTTP/1.1（或不符合）。

如果请求目标不是星号，则 OPTIONS 请求适用于与目标通信时可用的选项

资源。

生成对 OPTIONS 的成功响应的服务器应该发送任何标头字段，这些字段可能指示服务器实现的可选功能并适用于目标资源（例如，允许），包括本规范未定义的潜在扩展。

响应负载（如果有的话）也可能以机器或人类可读的表示形式描述通信选项。这种表示的标准格式未由本规范定义，但可能由未来的 HTTP 扩展定义。如果没有负载主体要在响应中发送，服务器必须生成一个值为“0”的 Content-Length 字段。

客户端可以在 OPTIONS 请求中发送一个 Max-Forwards 头字段，以将请求链中的特定接收者作为目标（参见第 5.1.2 节）。代理在转发请求时不得生成 Max-Forwards 标头字段，除非该请求是使用 Max-Forwards 字段接收的。

生成包含有效负载主体的 OPTIONS 请求的客户端必须发送描述表示媒体类型的有效 Content-Type 标头字段。尽管本规范未定义此类负载的任何用途，但未来对 HTTP 的扩展可能会使用 OPTIONS 主体来对目标进行更详细的查询

资源。

对 OPTIONS 方法的响应不可缓存。

4.3.8.痕迹

TRACE 方法请求请求消息的远程应用程序级环回。请求的最终接收者应该将接收到的消息（不包括下面描述的一些字段）作为 200（OK）响应的消息正文反映给客户端，其中 Content-Type 为“message/http”（第 8.3.1 节） [RFC7230]）。最终接收者是源服务器或第一个在请求中接收到 Max-Forwards 值为零 (0) 的服务器（第 5.1.2 节）。

客户端不得在包含敏感数据的 TRACE 请求中生成标头字段,这些数据可能会被响应泄露。

例如,用户代理在 TRACE 请求中发送存储的用户凭证 [RFC7235] 或 cookie [RFC6265] 是愚蠢的。请求的最终接收者应该在该接收者生成响应主体时排除任何可能包含敏感数据的请求标头字段。

TRACE 允许客户端查看在请求链的另一端接收到的内容,并将该数据用于测试或诊断信息。 Via 标头字段（[RFC7230] 的第 5.7.1 节)的值特别令人感兴趣,因为它充当请求链的跟踪。使用 Max-Forwards 标头字段允许客户端限制请求链的长度,这对于测试在无限循环中转发消息的代理链很有用。

客户端不得在 TRACE 请求中发送消息体。

对 TRACE 方法的响应不可缓存。

5. 请求头字段

客户端发送请求标头字段以提供有关请求上下文的更多信息、根据目标资源状态使请求有条件、建议响应的首选格式、提供身份验证凭证或修改预期的请求处理。这些字段充当请求修饰符,类似于编程语言方法调用的参数。

5.1.控件

控件是指示请求的特定处理的请求标头字段。

标题字段名称	定义在...
缓存控制	[RFC7234] 的第 5.2 节
预计	第 5.1.1 节
主持人	[RFC7230] 的第 5.4 节
最大前锋	第 5.1.2 节
语用	[RFC7234] 的第 5.4 节
范围	[RFC7233] 的第 3.1 节
TE	[RFC7230] 第 4.3 节

5.1.1.预计

请求中的“Expect”标头字段表示服务器需要支持的一组特定行为（期望)才能正确处理此请求。本规范定义的唯一此类期望是 100-continue。

期望=“100-继续”

Expect 字段值不区分大小写。

接收到除 100-continue 之外的 Expect 字段值的服务器可以使用 417（Expectation Failed)状态代码进行响应,以指示无法满足意外的期望。

100-continue 期望通知收件人客户端将在此请求中发送（可能是大的)消息体,并且如果请求行和标头字段不足以引起立即响应,则希望接收 100（继续)临时响应成功、重定向或错误响应。这允许客户端在实际发送消息体之前等待值得发送消息体的指示,这可以在消息体很大或客户端预计可能会出错时（例如,发送状态时)提高效率- 更改方法,第一次,无需先前验证的身份验证凭据）。

例如,以以下开头的请求

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

允许源服务器在客户端开始用不必要的数据传输填充管道之前立即响应错误消息,例如 401（未授权)或 405（方法不允许）。

对客户的要求：

o 客户端不得在请求中生成 100-continue 期望
不包括消息正文。

o 客户端将等待 100（继续)响应
发送请求消息体必须发送一个包含 100-continue 期望的 Expect 头字段。

o 发送 100-continue 期望的客户端不需要
等待任何特定的时间长度;这样的客户端可以继续发送消息体,即使它还没有收到响应。

此外,由于 100（继续)响应不能通过 HTTP/1.0 中介发送,这样的客户端不应该在发送消息正文之前无限期地等待。

o 收到 417（预期失败)状态代码的客户端
对包含 100-continue 期望的请求的响应应该重复没有 100-continue 期望的请求,因为 417 响应仅表示响应链不支持期望（例如,它通过
HTTP/1.0 服务器）。

服务器要求：

o 在 HTTP/1.0 中接收 100-continue 期望的服务器
请求必须忽略该期望。

o 服务器可以忽略发送 100（继续)响应,如果它有
已经收到相应请求的部分或全部消息体,或者如果帧指示没有消息体。

o 发送 100（继续)响应的服务器必须最终发送最终状态代码,一旦消息正文被接收和处理,除非连接过早关闭。

o 在读取整个消息正文之前以最终状态代码响应的服务器应该在该响应中指示它是打算关闭连接还是继续读取并丢弃请求消息（参见 [RFC7230] 的
第 6.6 节）。

源服务器必须在收到 HTTP/1.1（或更高版本)请求行和包含 100-continue 期望并指示请求消息正文将跟随的完整标头部分后,
发送带有最终状态代码的立即响应,如果可以通过仅检查请求行和标头字段来确定该状态,或者立即发送 100（继续)响应以鼓励
客户端发送请求的消息体。源服务器在发送 100（继续)响应之前不得等待消息体。

代理必须在收到 HTTP/1.1（或更高版本)请求行和包含 100-continue 期望并指示请求消息正文将跟随的完整标头部分后,发送带有最终状态代码
的立即响应,如果该状态可以通过仅检查请求行和标头字段来确定,或者通过发送

相应的请求行和标题部分到下一个入站服务器。如果代理认为（从配置或过去的交互)下一个入站服务器仅支持 HTTP/1.0,则代理可以立即生成 100（继续)响应以鼓励客户端开始发送消息体。

注意:Expect 标头字段是在 HTTP/1.1 [RFC2068] 的原始发布之后添加的,作为请求临时 100（继续)响应的方法和指示必须理解的扩展的一般机制。但是,扩展机制并没有被客户端使用,很多服务端也没有实现必须了解的需求,导致扩展机制无用武之地。本规范去除了扩展机制以简化100-continue的定义和处理。

5.1.2.最大前锋

“Max-Forwards”头字段提供了一种机制,使用 TRACE（第 4.3.8 节)和 OPTIONS（第 4.3.7 节)请求方法来限制代理转发请求的次数。当客户端试图跟踪一个似乎失败或在链中循环的请求时,这可能很有用。

最大转发 = 1*DIGIT

Max-Forwards 值为十进制整数,表示该请求报文可以转发的剩余次数。

每个接收包含 Max-Forwards 头字段的 TRACE 或 OPTIONS 请求的中介必须在转发请求之前检查并更新其值。如果接收到的值为零 (0),中介不得转发请求;相反,中介必须作为最终接收者响应。如果接收到的 Max-Forwards 值大于零,中介必须在转发消息中生成一个更新的 Max-Forwards 字段,其字段值是 a) 接收到的值减一 (1) 或 b)收件人对 Max-Forwards 的最大支持值。

接收者可以忽略通过任何其他请求方法接收到的 Max-Forwards 标头字段。

5.2.条件句

HTTP 条件请求标头字段 [RFC7232] 允许客户端在目标资源的状态上放置一个先决条件,以便如果先决条件的计算结果为假,则不会应用与方法语义对应的操作。每个先决条件由

本规范包含一组从目标资源的先前表示中获得的验证器与所选表示的验证器的当前状态之间的比较（第 7.2 节）。因此，这些先决条件评估目标资源的状态自客户端已知的给定状态以来是否发生了变化。这种评估的效果取决于方法语义和条件的选择，如 [RFC7232] 第 5 节中所定义。

标题字段名称	定义在...
如果匹配	[RFC7232] 的第 3.1 节
如果不匹配	[RFC7232] 的第 3.2 节
If-Modified-Since	[RFC7232] 的第 3.3 节
If-Unmodified-Since	[RFC7232] 第 3.4 节
如果范围	[RFC7233] 的第 3.2

5.3.内容协商

以下请求标头字段由用户代理发送，以参与响应内容的主动协商，如第 3.4.1 节中所定义。在这些字段中发送的首选项适用于响应中的任何内容，包括目标资源的表示、错误或处理状态的表示，甚至可能出现在协议中的杂项文本字符串。

标题字段名称	定义于...
接受	第 5.3.2 节
接受字符集	第 5.3.3 节
接受编码	第 5.3.4 节
接受语言	第 5.3.5 节

5.3.1.品质价值观

许多用于主动协商的请求标头字段使用一个名为“q”（不区分大小写）的公共参数来为相关内容类型的偏好分配相对“权重”。这个权重被称为“质量值”（或“qvalue”），因为在服务器配置中经常使用相同的参数名称来为可以为资源选择的各种表示的相对质量分配权重。

权重被归一化为 0 到 1 范围内的实数,其中 0.001 是最不优选的,1 是最优选的;值为 0 表示“不可接受”。如果不存在“q”参数,则默认权重为 1。

```
权重 = OWS “;” OWS q= qvalue qvalue =
( 0 [ . 0*3DIGIT ] ) / ( 1 [ . 0*3( 0 ) ] )
```

qvalue 的发送者不得在小数点后生成超过三位数字。这些值的用户配置应该以相同的方式进行限制。

5.3.2.接受

用户代理可以使用“Accept”头字段来指定可接受的响应媒体类型。 Accept 标头字段可用于指示请求专门限于一小组所需类型,如请求内联图像的情况。

```
接受 = #( 媒体范围 [ 接受参数 ] )

媒体范围 = ( */* /
( 输入 / * ) / ( 输入
/ 子类型 )
) *( OWS ; OWS 参数 ) accept-params =
weight *( accept-ext ) accept-ext = OWS ; OWS token [ = ( token /
quoted-string ) ]
```

星号“*”字符用于将媒体类型分组到范围内,“*/*”表示所有媒体类型,“type/*”表示该类型的所有子类型。媒体范围可以包括适用于该范围的媒体类型参数。

每个媒体范围后面可能跟着零个或多个适用的媒体类型参数(例如,字符集),一个可选的“q”参数用于指示相对权重(第 5.3.1 节),然后是零个或多个扩展参数。如果存在任何扩展 (accept-ext),则“q”参数是必需的,因为它充当两个参数集之间的分隔符。

注意:使用“q”参数名称将媒体类型参数与接受扩展参数分开是由于历史惯例。虽然这可以防止任何名为“q”的媒体类型参数与媒体范围一起使用,但鉴于 IANA 中缺少任何“q”参数,这种事件被认为是不太可能的

媒体类型注册表和 Accept 中很少使用任何媒体类型参数。不鼓励未来的媒体类型注册任何名为“q”的参数。

这个例子

接受:音频/*； q=0.2,音频/基本

被解释为“我更喜欢音频/基本,但如果它是质量降价 80% 后最好的音频类型,请发送给我”。

没有任何 Accept 标头字段的请求意味着用户代理将接受任何媒体类型作为响应。如果请求中存在标头字段,并且没有可用的响应表示具有列为可接受的媒体类型,则源服务器可以通过发送 406（不可接受)响应来尊重标头字段,或者忽略标头通过将响应视为不受内容协商影响来处理字段。

一个更详细的例子是

接受:文本/纯文本； q=0.5,文本/html,文本/x-dvi； q=0.8,文本/
xc

在口头上,这将被解释为“text/html 和 text/xc 是同样首选的媒体类型,但如果它们不存在,则发送 text/x-dvi 表示,如果不存在,则发送 text/
简单的表示”。

媒体范围可以被更具体的媒体范围或特定的媒体类型覆盖。如果多个媒体范围适用于给定类型,则最具体的参考具有优先权。例如,

接受:text/*, text/plain, text/plain;format=flowed, */*

具有以下优先级:

1. 文本/纯文本;格式=流式

2. 文本/纯文本

3.文字/*

4. */*

与给定类型关联的媒体类型质量因子是通过查找与该类型匹配的具有最高优先级的媒体范围来确定的。例如,

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

将导致关联以下值:

+----------+----------+
|媒体类型 |品质价值 |
+----------+----------+
|文本/html;级别=1 | 1 |文本/html | 0.7 |文本/
纯 | 0.3 |图片/jpeg | 0.5 |文本/html;级别=2|4 |文本/
html;级别=3 | 0.7

+----------+----------+

注意:用户代理可能会为某些媒体范围提供一组默认的质量值。然而,除非用户代理是一个不能与其他渲染代理交互的封闭系统,否则这个默认设置应该可以由用户配置。

5.3.3.接受字符集

“Accept-Charset”标头字段可以由用户代理发送,以指示在文本响应内容中哪些字符集是可接受的。

该字段允许能够理解更全面或专用字符集的用户代理向能够在这些字符集中表示信息的源服务器发出该能力信号。

接受字符集 = 1#((字符集 / *) [重量])

字符集名称在第 3.1.1.2 节中定义。用户代理可以将质量值与每个字符集相关联,以指示用户对该字符集的相对偏好,如第 5.3.1 节中所定义。

一个例子是

接受字符集:iso-8859-5、unicode-1-1;q=0.8

特殊值“*”如果出现在 Accept-Charset 字段中,则匹配 Accept-Charset 字段中其他地方未提及的每个字符集。如果 Accept-Charset 字段中没有“*”,则该字段中未明确提及的任何字符集都被视为客户端“不可接受”。

没有任何 Accept-Charset 标头字段的请求意味着用户代理将接受任何字符集作为响应。大多数通用用户代理不发送 Accept-Charset,除非特别说明

配置为这样做,因为支持的字符集的详细列表使服务器更容易根据用户代理的请求特征（第 9.7 节)识别个人。

如果请求中存在 Accept-Charset 标头字段,并且没有可用的响应表示具有列为可接受的字符集,则源服务器可以通过发送 406（不可接受)响应来尊重标头字段,或者通过将资源视为不受内容协商影响来忽略标头字段。

5.3.4.接受编码

用户代理可以使用“Accept-Encoding”头字段来指示响应中可接受的响应内容编码（第 3.1.2.1 节）。 “身份”令牌用作“无编码”的同义词,以便在首选无编码时进行通信。

```
Accept-Encoding = #( 编码 [ 权重 ] 编码 )
                  = 内容编码/“身份”/“*”
```

如第 5.3.1 节中所定义,每个编码值都可以被赋予一个相关联的质量值,代表该编码的偏好。 Accept-Encoding 字段中的星号“*”符号匹配任何未在标头字段中明确列出的可用内容编码。

例如,

```
接受编码:压缩,gzip 接受编码:接受编码:* 接受编码:压缩;q=0.5,
gzip;q=1.0 接受编码:gzip;q=1.0,身份； q=0.5, *;q=0
```

没有 Accept-Encoding 标头字段的请求意味着用户代理没有关于内容编码的偏好。虽然这允许服务器在响应中使用任何内容编码,但这并不意味着用户代理能够正确处理所有编码。

服务器使用以下规则测试给定表示的内容编码是否可接受：

1.如果请求中没有Accept-Encoding字段,任何content-coding
被用户代理认为是可以接受的。

2.如果表示没有内容编码,那么它是
默认情况下可接受,除非 Accept-Encoding 字段明确排除说明“identity;q=0”或“*;q=0”而没有更具体的“identity”条目。

3.如果表示的内容编码是其中之一
Accept-Encoding 字段中列出的内容编码,那么它是可以接受的,除非它伴随着 qvalue 为 0。(如第 5.3.1 节中所定义,qvalue 为 0 表示“不可接受”。)

4.如果多个内容编码是可接受的,那么具有最高非零qvalue的可接受内容编码是优选的。

具有空组合字段值的 Accept-Encoding 标头字段意味着用户代理不希望任何内容编码作为响应。如果请求中出现 Accept-Encoding 头字段,并且响应的可用表示都没有列为可接受的内容编码,则源服务器应该发送没有任何内容编码的响应。

注意:大多数 HTTP/1.0 应用程序不识别或遵守与内容编码关联的 qvalues。这意味着 qvalues 可能不起作用并且不允许与 x-gzip 或 x-compress 一起使用。

5.3.5.接受语言

用户代理可以使用“Accept-Language”标头字段来指示响应中首选的自然语言集。语言标签在第 3.1.3.1 节中定义。

Accept-Language = 1#(语言范围 [权重] 语言范围 =)

<语言范围,参见 [RFC4647],第 2.1 节>

每个语言范围都可以被赋予一个相关的质量值,表示用户对该范围指定的语言的偏好估计,如第 5.3.1 节中所定义。例如,

接受语言:da, en-gb;q=0.8, en;q=0.7

意思是:“我更喜欢丹麦语,但会接受英式英语和其他类型的英语”。

没有任何 Accept-Language 标头字段的请求意味着用户代理将接受任何语言作为响应。如果请求中存在标头字段,并且响应的可用表示都没有匹配的语言标记,则源服务器可以通过将响应视为好像它一样来忽略标头字段

不受内容协商或通过发送 406（不可接受）响应来遵守标头字段。但是，不鼓励使用后者，因为这样做会阻止用户访问他们可能会使用的内容（例如，使用翻译软件）。

请注意，一些收件人将语言标签列出的顺序视为优先级递减的指示，特别是对于分配了相同质量值的标签（没有值与 q=1 相同）。

但是，不能依赖此行为。为了一致性和最大化互操作性，许多用户代理为每个语言标签分配一个唯一的质量值，同时还按质量递减的顺序列出它们。可以在 [RFC4647] 的第 2.3 节中找到关于语言优先级列表的其他讨论。

对于匹配，[RFC4647] 的第 3 节定义了几种匹配方案。实现可以根据他们的要求提供最合适的匹配方案。 “基本过滤”方案（[RFC4647]，第 3.3.1 节）与先前在 [RFC2616] 的第 14.4 节中为 HTTP 定义的匹配方案相同。

在每个请求中发送包含用户完整语言偏好的 Accept-Language 标头字段可能有悖于用户的隐私期望（第 9.7 节）。

由于可理解性高度依赖于单个用户，因此用户代理需要允许用户控制语言偏好（通过用户代理本身的配置或默认为用户可控的系统设置）。不向用户提供此类控制的用户代理不得发送 Accept-Language 标头字段。

> 注意：用户代理应该在设置首选项时为用户提供指导，因为用户很少熟悉上述语言匹配的细节。例如，用户可能会假设在选择“en-gb”时，如果英式英语不可用，他们将获得任何类型的英文文档。在这种情况下，用户代理可能会建议将“en”添加到列表中以获得更好的匹配行为。

5.4.身份验证凭据

两个标头字段用于携带身份验证凭据,如 [RFC7235] 中所定义。请注意,用于用户身份验证的各种自定义机制为此目的使用 Cookie 标头字段,如 [RFC6265] 中所定义。

标题字段名称	定义在...
授权	[RFC7235] 的第 4.2 节
代理授权	[RFC7235] 第 4.4 节

5.5.请求上下文

以下请求标头字段提供有关请求上下文的其他信息,包括有关用户、用户代理和请求背后的资源的信息。

标题字段名称	定义于...
从	第 5.5.1 节
推荐人	第 5.5.2 节
用户代理	第 5.5.3 节

5.5.1.从

“发件人”标题字段包含控制请求用户代理的人类用户的 Internet 电子邮件地址。地址应该是机器可用的,如 [RFC5322] 第 3.4 节中的“邮箱”所定义:

```
From    = mailbox

mailbox = <mailbox, see [RFC5322], Section 3.4>
```

一个例子是:

```
From: webmaster@example.org
```

From 头字段很少由非机器人用户代理发送。用户代理不应该在没有用户明确配置的情况下发送 From 标头字段,因为这可能与用户的隐私利益或他们站点的安全策略发生冲突。

机器人用户代理应该发送一个有效的 From 头字段,以便在服务器出现问题时可以联系负责运行机器人的人员,例如机器人发送过多的、不需要的或无效的请求。

服务器不应该使用 From 标头字段进行访问控制或身份验证,因为大多数接收者会假定该字段值是公共信息。

5.5.2.推荐人

“Referer”[sic] 标头字段允许用户代理为从中获取目标 URI 的资源指定 URI 引用（即“referrer”,尽管字段名称拼写错误）。在生成 Referer 字段值时,用户代理不得包含 URI 引用 [RFC3986] 的片段和用户信息组件（如果有的话）。

Referer = 绝对 URI / 部分 URI

Referer 标头字段允许服务器生成指向其他资源的反向链接,以进行简单的分析、日志记录、优化缓存等。它还允许找到过时或输入错误的链接以进行维护。一些服务器使用 Referer 标头字段作为拒绝来自其他站点的链接（所谓的“深度链接”）或限制跨站点请求伪造 (CSRF) 的手段,但并非所有请求都包含它。

例子：

推荐人:http://www.example.org/hypertext/Overview.html

如果目标 URI 是从一个没有自己的 URI 的来源获得的（例如,从用户键盘输入,或用户书签/收藏夹中的条目）,用户代理必须排除 Referer 字段或将其与“关于:空白”的价值。

Referer 字段有可能泄露有关用户请求上下文或浏览历史的信息,如果引用资源的标识符泄露个人信息（例如帐户名）或本应保密的资源（例如在防火墙后面或安全服务的内部）。当引用资源是本地“文件”或“数据”URI 时,大多数通用用户代理不会发送 Referer 标头字段。如果引用页面是使用安全协议接收的,则用户代理不得在不安全的 HTTP 请求中发送 Referer 标头字段。有关其他安全注意事项,请参阅第 9.4 节。

众所周知,一些中介会不加区别地从传出请求中删除 Referer 标头字段。这有一个不幸的副作用,即干扰对 CSRF 攻击的保护,这可能对他们的用户造成更大的伤害。

希望在 Referer 中限制信息披露的中介和用户代理扩展应该限制他们对特定编辑的更改,例如用假名替换内部域名或截断查询和/或路径组件。当字段值与请求目标共享相同的方案和主机时,中间人不应该修改或删除 Referer 头字段。

5.5.3.用户代理

“User-Agent”标头字段包含有关发起请求的用户代理的信息,服务器通常使用这些信息来帮助识别报告的互操作性问题的范围,解决或定制响应以避免特定的用户代理限制,以及用于分析关于浏览器或操作系统的使用。用户代理应该在每个请求中发送一个 User-Agent 字段,除非特别配置为不这样做。

User-Agent = product *(RWS (产品 / 评论))

User-Agent 字段值由一个或多个产品标识符组成,每个产品标识符后跟零个或多个注释（[RFC7230] 的第 3.2 节）,它们共同标识用户代理软件及其重要的子产品。按照惯例,产品标识符按其识别用户代理软件的重要性降序排列。每个产品标识符都包含名称和可选版本。

产品 = 令牌 [/ 产品版本] 产品版本 = 令牌

发件人应该将生成的产品标识符限制为识别产品所必需的;发件人不得在产品标识符中生成广告或其他非必要信息。发件人不应在产品版本中生成不是版本标识符的信息（即,相同产品名称的连续版本应该仅在产品标识符的产品版本部分有所不同）。

例子:

用户代理:CERN-LineMode/2.15 libwww/2.17b3

用户代理不应生成包含不必要的细粒度细节的用户代理字段,并且应限制第三方添加子产品。过长和详细的 User-Agent 字段值会增加请求延迟和用户被识别为违背其意愿的风险（“指纹识别”）。

同样,鼓励实现不要使用其他实现的产品令牌来声明与它们的兼容性,因为这会规避该字段的目的。如果用户代理伪装成不同的用户代理,接收者可以假设用户有意希望看到为该识别的用户代理量身定制的响应,即使它们可能不适用于正在使用的实际用户代理。

6.响应状态码

status-code 元素是一个三位整数代码,给出尝试理解和满足请求的结果。

HTTP 状态代码是可扩展的。 HTTP 客户端不需要理解所有已注册状态代码的含义,尽管这种理解显然是可取的。但是,客户端必须理解任何状态代码的类,如第一个数字所示,并将无法识别的状态代码视为等同于该类的 x00 状态代码,但接收者不得缓存响应无法识别的状态代码。

例如,如果客户端收到无法识别的状态代码 471,则客户端可以假设其请求有问题并将响应视为收到 400（错误请求）状态代码。响应消息通常包含解释状态的表示。

状态代码的第一位数字定义了响应的类别。
最后两位数字没有任何分类作用。第一个数字有五个值：

- o 1xx（信息）：收到请求,继续处理
- o 2xx（成功）：请求被成功接收、理解和接受
- o 3xx（重定向）：需要采取进一步的行动以完成请求
- o 4xx（客户端错误）：请求包含错误的语法或无法履行

o 5xx（服务器错误）：服务器未能完成明显有效的请求

6.1.状态码概览

下面列出的状态代码在本规范、[RFC7232] 的第 4 节、[RFC7233] 的第 4 节和 [RFC7235] 的第 3 节中定义。此处列出的短语仅是建议的原因 它们可以在不影响协议的情况下替换为本地等效项。

具有默认定义为可缓存的状态代码的响应（例如，本规范中的 200、203、204、206、300、301、404、405、410、414 和 501）可以由具有启发式过期的缓存重用，除非方法定义或显式缓存控制 [RFC7234] 另有说明；默认情况下，所有其他状态代码都不可缓存。

代码	原因短语	定义在...
100	继续	第 6.2.1 节
101	交换协议	第 6.2.2 节
200	好的	第 6.3.1 节
201	已创建	第 6.3.2 节
202	已接受	第 6.3.3 节
203	非权威信息	第 6.3.4 节
204	没有内容	第 6.3.5 节
205	重置内容	第 6.3.6 节
206	部分内容	[RFC7233] 第 4.1 节
300	多种选择	第 6.4.1 节
301	永久移动	第 6.4.2 节
302	找到	第 6.4.3 节
303	查看其他	第 6.4.4 节
304	未修改	[RFC7232] 第 4.1 节
305	使用代理	第 6.4.5 节
306	（未使用）	第 6.4.6 节
307	临时重定向	第 6.4.7 节
400	错误请求	第 6.5.1 节
401	未经授权	[RFC7235] 第 3.1 节
402	需要付款	第 6.5.2 节
403	禁止	第 6.5.3 节
404	未找到	第 6.5.4 节
405	方法不允许	第 6.5.5 节
406	不可接受	第 6.5.6 节
407	需要代理身份验证	[RFC7235] 第 3.2 节
408	请求超时	第 6.5.7 节
409	冲突	第 6.5.8 节
410	走了	第 6.5.9 节
411	要求长度	第 6.5.10 节
412	前提条件失败	[RFC7232] 第 4.2 节
413	请求实体太大	第 6.5.11 节
414	URI 太长	第 6.5.12 节
415	不支持的媒体类型	第 6.5.13 节
416	范围不满足	[RFC7233] 第 4.4 节
417	期望失败	第 6.5.14 节
426	需要升级	第 6.5.15 节
500	内部服务器错误	第 6.6.1 节
501	未实施	第 6.6.2 节
502	坏网关	第 6.6.3 节
503	服务不可用	第 6.6.4 节
504	网关超时	第 6.6.5 节
505	HTTP 版本不支持	第 6.6.6 节

请注意,此列表并不详尽 它不包括其他规范中定义的扩展状态代码。状态代码的完整列表由 IANA 维护。有关详细信息,请参阅第 8.2 节。

6.2.信息 1xx

1xx（信息）类状态代码表示在完成请求的操作并发送最终响应之前,用于通信连接状态或请求进度的临时响应。 1xx 响应由状态行之后的第一个空行终止（表示标题部分结束的空行）。由于 HTTP/1.0 没有定义任何 1xx 状态代码,服务器不得向 HTTP/1.0 客户端发送 1xx 响应。

客户端必须能够解析在最终响应之前收到的一个或多个 1xx 响应,即使客户端不希望收到一个。用户代理可以忽略意外的 1xx 响应。

代理必须转发 1xx 响应,除非代理本身请求生成 1xx 响应。例如,如果代理在转发请求时添加了“Expect: 100-continue”字段,则它不需要转发相应的 100 (Continue) 响应。

6.2.1. 100 继续

100（Continue）状态码表示请求的初始部分已经收到,还没有被服务器拒绝。服务器打算在完全接收并执行请求后发送最终响应。

当请求包含包含 100-continue 期望的 Expect 标头字段时,100 响应表示服务器希望接收请求有效负载主体,如第 5.1.1 节所述。客户端应该继续发送请求并丢弃 100 响应。

如果请求不包含包含 100-continue 期望的 Expect 标头字段,则客户端可以简单地丢弃此中间值

回复。

6.2.2. 101 交换协议

101（切换协议）状态代码表示服务器理解并愿意通过升级标头字段（[RFC7230] 的第 6.7 节）遵守客户端的请求,以更改此连接上使用的应用程序协议。服务器

必须在响应中生成一个 Upgrade 标头字段,指示将在终止 101 响应的空行之后立即切换到哪个协议。

假设服务器只会在有利时才同意切换协议。例如,切换到较新版本的 HTTP 可能优于旧版本,而切换到实时同步协议可能在交付使用此类功能的资源时更有优势。

6.3.成功 2xx

2xx (Successful) 类的状态码表示客户端的请求被成功接收、理解和接受。

6.3.1. 200 好

200（OK)状态码表示请求成功。
200 响应中发送的负载取决于请求方法。
对于本规范定义的方法,有效载荷的预期含义可以概括为:

GET 目标资源的表示;

HEAD 与 GET 相同的表示,但没有表示
数据;

发布状态的表示或从中获得的结果,
那个行动;

PUT、DELETE 表示动作的状态;

OPTIONS 通信选项的表示;

TRACE 表示最后收到的请求消息
服务器。

除了对 CONNECT 的响应之外,200 响应总是有一个有效负载,尽管源服务器可能会生成一个零长度的有效负载主体。

如果不需要负载,源服务器应该发送 204（无内容）。对于 CONNECT,不允许任何负载,因为成功的结果是一个隧道,它在 200 响应头部分之后立即开始。

默认情况下,200 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.3.2. 201 创建

201（已创建）状态代码表示请求已完成并导致创建一个或多个新资源。请求创建的主要资源由响应中的 Location 标头字段标识，或者如果未收到 Location 字段，则由有效请求 URI 标识。

201 响应负载通常描述并链接到创建的资源。请参阅第 7.2 节，了解 201 响应中验证器标头字段（例如 ETag 和 Last-Modified）的含义和用途的讨论。

6.3.3. 202 已接受

202（已接受）状态码表示请求已被接受处理，但处理尚未完成。

该请求最终可能会或可能不会被执行，因为在实际进行处理时可能会拒绝该请求。 HTTP 中没有用于从异步操作重新发送状态代码的工具。

202 响应是故意不置可否的。它的目的是允许服务器接受对其他进程（可能是每天只运行一次的面向批处理的进程）的请求，而不需要用户代理与服务器的连接持续到进程完成。与此响应一起发送的表示应该描述请求的当前状态并指向（或嵌入）状态监视器，该监视器可以为用户提供请求何时完成的估计。

6.3.4. 203非权威信息

203（非权威信息）状态代码表示请求成功，但包含的负载已被转换代理（[RFC7230] 的第 5.7.2 节）从源服务器的 200（OK）响应中修改。此状态代码允许代理在应用转换时通知收件人，因为该信息可能会影响以后有关内容的决策。例如，未来对内容的缓存验证请求可能仅适用于相同的请求路径（通过相同的代理）。

203 响应类似于 214 Transformation Applied（[RFC7234] 的第 5.5 节）的警告代码，其优点是适用于任何状态代码的响应。

默认情况下,203 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.3.5. 204 无内容

204（无内容）状态代码表示服务器已成功完成请求,并且响应有效负载正文中没有要发送的其他内容。响应标头字段中的元数据指的是目标资源及其在应用请求的操作后选择的表示。

例如,如果收到 204 状态代码以响应 PUT 请求并且响应包含 ETag 标头字段,则 PUT 成功并且 ETag 字段值包含该目标资源的新表示的实体标签。

204 响应允许服务器指示操作已成功应用于目标资源,同时暗示用户代理不需要遍历其当前的“文档视图”（如果有）。服务器假定用户代理将根据其自己的界面向其用户提供一些成功指示,并在对其活动表示的响应中应用任何新的或更新的元数据。

例如,204 状态码通常用于对应于“保存”操作的文档编辑界面,使得正在保存的文档仍然可供用户编辑。它还经常与期望自动数据传输流行的接口一起使用,例如在分布式版本控制系统中。

204 响应由标头字段后的第一个空行终止,因为它不能包含消息正文。

默认情况下,204 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.3.6. 205重置内容

205（重置内容）状态代码表示服务器已完成请求并希望用户代理将导致发送请求的“文档视图”重置为从源服务器接收到的原始状态。

此响应旨在支持常见的数据输入用例,其中用户接收支持数据输入的内容（表单、记事本、画布等）,在该空间中输入或操作数据,

导致在请求中提交输入的数据,然后为下一个条目重置数据输入机制,以便用户可以轻松发起另一个输入操作。

由于 205 状态代码暗示不会提供额外的内容,因此服务器不得在 205 响应中生成有效负载。换句话说,服务器必须为 205 响应执行以下操作之一:
a) 通过包含值为 0 的 Content-Length 标头字段来指示响应的零长度主体; b) 通过包含值为 chunked 的 Transfer-Encoding 标头字段和由单个零长度块组成的消息正文来指示响应的零长度有效载荷;或者,c) 在发送终止标题部分的空行后立即关闭连接。

6.4.重定向 3xx

状态代码的 3xx(重定向)类表示用户代理需要采取进一步的操作才能完成请求。如果提供 Location 头字段(第 7.1.2 节),用户代理可以自动将其请求重定向到 Location 字段值引用的 URI,即使不理解特定的状态代码。自动重定向需要小心处理未知的安全方法,如第 4.2.1 节中所定义,因为用户可能不希望重定向不安全的请求。

有几种类型的重定向:

1. 指示资源可能在不同 URI 可用的重定向,如位置字段提供的,如状态代码 301(永久移动)、302(找到)和 307(临时重定向)。

2. 提供匹配资源选择的重定向,每个
   能够表示原始请求目标,如 300(多项选择)状态代码。

3. 重定向到由 Location 标识的不同资源
   字段,可以表示对请求的间接响应,如 303(参见其他)状态代码。

4. 重定向到先前缓存的结果,如 304 (Not
   修改)状态码。

   注意:在 HTTP/1.0 中,状态代码 301(永久移动)和 302(找到)是为第一种重定向定义的([RFC1945],第 9.3 节)。早期的用户代理在应用于重定向目标的方法是否与

原始请求或将被重写为 GET。尽管 HTTP 最初为 301 和 302 定义了前者语义（以匹配其在 CERN 的原始实现），并定义了 303（参见其他）以匹配后者语义，但流行的做法也逐渐向 301 和 302 的后者语义收敛。 HTTP/1.1 的第一次修订增加了 307（临时重定向）来指示以前的语义，而不受不同实践的影响。 10 多年过去了，大多数用户代理仍然对 301 和 302 进行方法重写；因此，当原始请求是 POST 时，本规范使该行为符合要求。

客户端应该检测并干预循环重定向（即“无限”重定向循环）。

注意：本规范的早期版本推荐最多五个重定向（[RFC2068]，第 10.3 节）。内容开发人员需要注意某些客户端可能会实施此类固定限制。

6.4.1. 300多项选择

300（多项选择）状态代码表示目标资源有多个表示，每个都有自己更具体的标识符，并且正在提供有关备选方案的信息，以便用户（或用户代理）可以通过以下方式选择首选表示将其请求重定向到这些标识符中的一个或多个。换句话说，服务器希望用户代理参与反应性协商以选择最适合其需要的表示（第 3.4 节）。

如果服务器有首选，服务器应该生成一个 Location 头字段，其中包含首选的 URI 引用。

用户代理可以使用 Location 字段值进行自动重定向。

对于 HEAD 以外的请求方法，服务器应该在 300 响应中生成一个负载，其中包含表示元数据和 URI 引用的列表，用户或用户代理可以从中选择最喜欢的一个。如果用户代理理解提供的媒体类型，它可以自动从该列表中进行选择。本规范未定义用于自动选择的特定格式，因为 HTTP 试图与其有效负载的定义保持正交。在实践中，表示以一些易于解析的格式提供，这些格式被认为是用户代理可以接受的，如共享设计或内容协商所确定的，或者以一些普遍接受的超文本格式提供。

默认情况下,300 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

注意:300 状态代码的原始提案将 URI 标头字段定义为提供替代表示列表,以便它可用于 200、300 和 406 响应,并在响应 HEAD 方法时传输。

然而,缺乏部署和对语法的分歧导致 URI 和 Alternates（后续提案）被从该规范中删除。可以使用一组链接头字段 [RFC5988] 来传达列表,每个字段都具有“交替”关系,尽管部署是先有鸡还是先有蛋的问题。

6.4.2. 301 永久移动

301（永久移动）状态代码表示目标资源已分配了一个新的永久 URI,并且将来对该资源的任何引用都应该使用其中一个包含的 URI。

如果可能,具有链接编辑功能的客户端应该自动将对有效请求 URI 的引用重新链接到服务器发送的一个或多个新引用。

服务器应该在响应中生成一个 Location 头字段,其中包含新永久 URI 的首选 URI 引用。用户代理可以使用 Location 字段值进行自动重定向。服务器的响应负载通常包含一个简短的超文本注释,其中包含指向新 URI 的超链接。

注意:由于历史原因,用户代理可以将后续请求的请求方法从 POST 更改为 GET。如果不需要此行为,则可以改用 307（临时重定向）状态代码。

默认情况下,301 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.4.3. 302 找到

302（已找到）状态码表示目标资源暂时驻留在不同的 URI 下。由于有时可能会更改重定向,因此客户端应该继续为以后的请求使用有效的请求 URI。

服务器应该在包含不同 URI 的 URI 引用的响应中生成 Location 头域。用户代理可以使用 Location 字段值进行自动重定向。服务器的响应负载通常包含一个简短的超文本注释,带有指向不同 URI 的超链接。

注意:由于历史原因,用户代理可以将后续请求的请求方法从 POST 更改为 GET。如果不需要此行为,则可以改用 307(临时重定向)状态代码。

6.4.4. 303 看其他

303 (See Other) 状态代码表示服务器正在将用户代理重定向到不同的资源,如 Location 标头字段中的 URI 所示,旨在提供对原始请求的间接响应。用户代理可以执行针对该 URI 的检索请求(如果使用 HTTP,则为 GET 或 HEAD 请求),该请求也可能被重定向,并将最终结果作为对原始请求的答复。请注意,Location 标头字段中的新 URI 不被视为等同于有效请求 URI。

此状态代码适用于任何 HTTP 方法。它主要用于允许 POST 操作的输出将用户代理重定向到选定的资源,因为这样做以一种可以单独识别、添加书签和缓存的形式提供与 POST 响应对应的信息,独立于原始请求。

对 GET 请求的 303 响应表示源服务器没有可以由服务器通过 HTTP 传输的目标资源的表示。但是,Location 字段值指的是描述目标资源的资源,因此对该其他资源发出检索请求可能会导致对接收者有用的表示,而不暗示它代表原始目标资源。请注意,可以表示什么、什么表示是合适的以及什么可能是有用的描述等问题的答案超出了 HTTP 的范围。

除了对 HEAD 请求的响应之外,303 响应的表示应该包含一个简短的超文本注释,其中包含指向 Location 标头字段中提供的相同 URI 引用的超链接。

6.4.5. 305 使用代理

305（使用代理)状态代码在本规范的先前版本中定义,现在已弃用（附录 B）。

6.4.6. 306（未使用）

306 状态码在本规范的前一版本中定义,不再使用,该代码被保留。

6.4.7. 307临时重定向

307（临时重定向)状态代码表示目标资源暂时驻留在不同的 URI 下,如果用户代理执行自动重定向到该 URI,则用户代理不得更改请求方法。由于重定向会随着时间的推移而改变,因此客户端应该继续使用原始有效的请求 URI 来处理未来的请求。

服务器应该在包含不同 URI 的 URI 引用的响应中生成 Location 头域。用户代理可以使用 Location 字段值进行自动重定向。服务器的响应负载通常包含一个简短的超文本注释,带有指向不同 URI 的超链接。

> 注意 :这个状态码类似于302（Found）,只是它不允许将请求方式从POST改为GET。本规范没有定义 301（永久移动)的等效对应物（[RFC7238],但是,为此目的定义了状态代码 308（永久重定向））。

6.5.客户端错误 4xx

4xx (Client Error) 类状态码表示客户端似乎出错了。除了响应 HEAD 请求时,服务器应该发送一个包含对错误情况的解释的表示,以及它是临时的还是永久的条件。这些状态代码适用于任何请求方法。

用户代理应该向用户显示任何包含的表示。

6.5.1. 400 错误请求

400（Bad Request)状态代码表示服务器不能或不会处理请求,因为某些事情被认为是客户端错误（例如,格式错误的请求语法、无效的请求消息框架或欺骗性请求路由）。

6.5.2. 402 需要付款

402（需要付款）状态代码保留供将来使用。

6.5.3. 403禁止访问

403（禁止访问）状态码表示服务器理解请求但拒绝授权。希望公开请求被禁止的原因的服务器可以在响应有效负载（如果有）中描述该原因。

如果请求中提供了身份验证凭据，则服务器认为它们不足以授予访问权限。客户端不应该使用相同的凭据自动重复请求。客户端可以使用新的或不同的凭据重复请求。但是，由于与凭据无关的原因，请求可能会被禁止。

希望“隐藏”当前存在的禁止目标资源的源服务器可以用状态代码 404（未找到）来响应。

6.5.4. 404 未找到

404（未找到）状态代码表示源服务器未找到目标资源的当前表示或不愿意公开存在该表示。 404 状态代码不表示这种缺乏表示是暂时的还是永久的； 410（Gone）状态代码比 404 更受欢迎，如果源服务器知道（可能通过一些可配置的方式）这种情况可能是永久性的。

默认情况下，404 响应是可缓存的；即，除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.5.5. 405 方法不允许

405（Method Not Allowed）状态代码表示请求行中收到的方法为源服务器所知，但目标资源不支持。源服务器必须在 405 响应中生成一个 Allow 头字段，其中包含目标资源当前支持的方法列表。

默认情况下，405 响应是可缓存的；即，除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.5.6. 406 不可接受

406（Not Acceptable）状态码表示根据请求中收到的主动协商标头字段（第 5.3 节），目标资源没有用户代理可以接受的当前表示，并且服务器不愿意提供默认表示。

服务器应该生成一个包含可用表示特征列表和相应资源标识符的有效负载，用户或用户代理可以从中选择最合适的。用户代理可以自动从该列表中选择最合适的选项。然而，本规范并未为此类自动选择定义任何标准，如第 6.4.1 节所述。

6.5.7. 408请求超时

408（Request Timeout）状态码表示服务器在准备等待的时间内没有收到完整的请求消息。服务器应该在响应中发送“关闭”连接选项（[RFC7230] 的第 6.1 节），因为 408 意味着服务器已决定关闭连接而不是继续等待。如果客户端在传输过程中有未完成的请求，则客户端可以在新连接上重复该请求。

6.5.8. 409 冲突

409（冲突）状态码表示由于与目标资源的当前状态发生冲突，请求无法完成。此代码用于用户可能能够解决冲突并重新提交请求的情况。服务器应该生成一个负载，其中包含足够的信息供用户识别冲突的来源。

冲突最有可能在响应 PUT 请求时发生。例如，如果正在使用版本控制并且正在 PUT 的表示包含对与早期（第三方）请求所做的冲突的资源的更改，则源服务器可能会使用 409 响应来指示它无法完成要求。在这种情况下，响应表示可能包含对基于修订历史合并差异有用的信息。

6.5.9. 410 没了

410（Gone）状态代码表示在源服务器上不再提供对目标资源的访问，并且这种情况可能是永久性的。如果源服务器没有

知道或无法确定条件是否是永久性的,应该使用状态代码 404（未找到）。

410 响应主要旨在通过通知接收者资源有意不可用以及服务器所有者希望删除指向该资源的远程链接来协助 Web 维护任务。对于限时促销服务和属于不再与原始服务器站点相关联的个人的资源,此类事件很常见。没有必要将所有永久不可用的资源标记为“消失”或将标记保留任何时间长度 这由服务器所有者自行决定。

默认情况下,410 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.5.10。 411 长度要求

411（需要长度)状态代码表示服务器拒绝接受没有定义内容长度的请求（[RFC7230] 的第 3.3.2 节）。如果客户端在请求消息中添加了包含消息正文长度的有效 Content-Length 标头字段,则客户端可以重复该请求。

6.5.11。 413 负载太大

413（负载太大)状态代码表示服务器拒绝处理请求,因为请求负载大于服务器愿意或能够处理的负载。服务器可以关闭连接以防止客户端继续请求。

如果条件是临时的,服务器应该生成一个 Retry-After 头字段来指示它是临时的,并且在什么时间之后客户端可以重试。

6.5.12。 414 URI 太长

414（URI 太长)状态代码表示服务器拒绝为请求提供服务,因为请求目标（[RFC7230] 的第 5.3 节)比服务器愿意解释的要长。

这种罕见的情况只有当客户端不正确地将 POST 请求转换为具有长查询信息的 GET 请求时,当客户端陷入重定向的“黑洞”时（例如,重定向的 URI 前缀指向本身的后缀)或当服务器受到试图利用潜在安全漏洞的客户端的攻击时。

默认情况下,414 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.5.13。 415 不支持的媒体类型

415（不支持的媒体类型)状态代码表示源服务器拒绝为请求提供服务,因为有效负载的格式不受此方法在目标资源上的支持。

格式问题可能是由于请求指定的 Content-Type 或 Content-Encoding,或者是直接检查数据的结果。

6.5.14。 417 期待失败

417（Expectation Failed)状态码表示请求的 Expect 标头字段（第 5.1.1 节)中给出的期望至少有一个入站无法满足

服务器。

6.5.15。 426 需要升级

426（需要升级)状态码表示服务器拒绝使用当前协议执行请求,但在客户端升级到不同的协议后可能愿意这样做。服务器必须在 426 响应中
发送 Upgrade 标头字段以指示所需的协议（[RFC7230] 的第 6.7 节）。

例子：

```
HTTP/1.1 426 需要升级升级:HTTP/3.0 连接:升级内容长度：
53 内容类型:文本/纯文本

此服务需要使用 HTTP/3.0 协议。
```

6.6.服务器错误 5xx

5xx（服务器错误)类状态代码表示服务器知道它有错误或无法执行请求的方法。除了响应 HEAD 请求时,服务器应该发送一个包含对错误情况的解释
的表示,以及它是临时的还是永久的

健康)状况。用户代理应该向用户显示任何包含的表示。这些响应代码适用于任何请求方法。

6.6.1. 500内部服务器错误

500（内部服务器错误)状态代码表示服务器遇到了阻止它完成请求的意外情况。

6.6.2. 501 未实现

501（未实现)状态代码表示服务器不支持完成请求所需的功能。当服务器无法识别请求方法并且无法支持任何资源时,这是适当的响应。

默认情况下,501 响应是可缓存的;即,除非方法定义或显式缓存控制另有说明（参见 [RFC7234] 的第 4.2.2 节）。

6.6.3. 502错误的网关

502（错误的网关)状态代码表示服务器在充当网关或代理时,在尝试完成请求时从其访问的入站服务器收到了无效响应。

6.6.4. 503服务不可用

503（Service Unavailable)状态码表示服务器当前由于临时过载或定期维护而无法处理请求,延迟一段时间后可能会得到缓解。服务器可以发送一个 Retry-After 头字段（第 7.1.3 节)来建议客户端在重试请求之前等待的适当时间。

注意:503 状态码的存在并不意味着服务器在超载时必须使用它。一些服务器可能只是拒绝连接。

6.6.5. 504网关超时

504（网关超时)状态代码表示服务器在充当网关或代理时,没有从它需要访问以完成请求的上游服务器及时收到响应。

6.6.6. 505 不支持 HTTP 版本

505（不支持 HTTP 版本）状态代码表示服务器不支持或拒绝支持请求消息中使用的 HTTP 主要版本。服务器表明它不能或不愿意使用与客户端相同的主要版本来完成请求，如 [RFC7230] 的第 2.6 节所述，除了此错误消息。服务器应该为 505 响应生成一个表示，描述为什么不支持该版本以及该服务器支持哪些其他协议。

7.响应头字段

响应标头字段允许服务器传递有关响应的附加信息，超出状态行中的信息。这些标头字段提供有关服务器的信息、有关对目标资源的进一步访问或有关的相关信息

资源。

尽管每个响应标头字段都有定义的含义，但一般来说，精确的语义可能会通过请求方法和/或响应状态代码的语义进一步细化。

7.1.控制数据

响应标头字段可以提供补充状态代码、指导缓存或指示客户端下一步去向的控制数据。

标题字段名称	定义在...
年龄	[RFC7234] 的第 5.1 节
缓存控制	[RFC7234] 的第 5.2 节
过期	[RFC7234] 的第 5.3 节
日期	第 7.1.1.2 节
地点	第 7.1.2 节
之后重试	第 7.1.3 节
各不相同	第 7.1.4 节
警告	[RFC7234] 的第 5.5 节

7.1.1.发起日期

7.1.1.1.日期/时间格式

在 1995 年之前,服务器通常使用三种不同的格式来传达时间戳。为了与旧实现兼容,所有三个都在此处定义。首选格式是 Internet 消息格式 [RFC5322] 使用的日期和时间规范的固定长度和单区域子集。

```
HTTP 日期 = IMF-fixdate / 观察日期
```

首选格式的一个例子是

```
1994 年 11 月 6 日,星期日 08:49:37 GMT ;国际货币基金组织固定日期
```

两种过时格式的示例是

```
格林威治标准时间 2094 年 11 月 6 日星期日 08:49:37 ;过时的 RFC 850 格式
1994 年 11 月 6 日星期日 08:49:37 ; ANSI C 的 asctime() 格式
```

解析 HTTP 标头字段中的时间戳值的接收者必须接受所有三种 HTTP 日期格式。当发送方生成包含一个或多个定义为 HTTP-date 的时间戳的标头字段时,发送方必须以 IMF-fixdate 格式生成这些时间戳。

HTTP 日期值将时间表示为协调世界时 (UTC) 的一个实例。前两种格式通过格林威治标准时间的三个字母缩写“GMT”表示 UTC,这是 UTC 名称的前身;假定 asctime 格式的值采用 UTC。从本地时钟生成 HTTP 日期值的发送方应该使用 NTP（[RFC5905])或一些类似的协议将其时钟同步到 UTC。

首选格式:

```
IMF-fixdate = day-name , SP date1 SP time-of-day SP GMT ;格式的固定长度/区域/大写子集;参见 [RFC5322] 的
第 3.3 节

日名 = %x4D.6F.6E ; “周一”,区分大小写 / %x54.75.65 ; “星期
二”,区分大小写 / %x57.65.64 ; “星期三”,区分大小写 /
%x54.68.75 ; “周四”,区分大小写 / %x46.72.69 ; “星期
五”,区分大小写 / %x53.61.74 ; “星期六”,区分大小写 /
%x53.75.6E ; “太阳”,区分大小写
```

日期 1 = 日 SP 月 SP 年 ;例如,1982 年 6 月 2 日

日月 = 2DIGIT =
%x4A.61.6E ; “一月”,区分大小写 / %x46.65.62 ; “二月”,区分大小写 / %x4D.61.72 ; “Mar”,区分大小写 / %x41.70.72 ; “四月”,区分大小写 / %x4D.61.79 ; “可能”,区分大小写 / %x4A.75.6E ; “君”,区分大小写 / %x4A.75.6C ; “七月”,区分大小写 / %x41.75.67 ; “八月”,区分大小写 / %x53.65.70 ; “九月”,区分大小写 / %x4F.63.74 ; “Oct”,区分大小写 / %x4E.6F.76 ; “11 月”,区分大小写 / %x44.65.63 ; “Dec”,区分大小写 = 4DIGIT

年

格林威治标准时间 = %x47.4D.54 ; “GMT”,区分大小写

time-of-day = 小时 : 分钟 : 秒; 00:00:00 - 23:59:60（闰秒）

时分秒 = 2 位数字
= 2 位数字
= 2 位数字

过时的格式：

观察日期 = rfc850-日期/asctime-日期

rfc850-date = day-name-l , SP date2 SP time-of-day SP GMT date2
=日“-”月“-”2DIGIT;例如,02-Jun-82

日名-l = %x4D.6F.6E.64.61.79 ; “星期一”,区分大小写 / %x54.75.65.73.64.61.79 ; “星期二”,区分大小写 / %x57.65.64.6E.65.73.64.61.79 ; “星期三”,区分大小写 / %x54.68.75.72.73.64.61.79 ; “星期四”,区分大小写 / %x46.72.69.64.61.79 ; “星期五”,区分大小写 / %x53.61.74.75.72.64.61.79 ; “星期六”,区分大小写 / %x53.75.6E.64.61.79 ; “星期日”,区分大小写

asctime-date = day-name SP date3 SP time-of-day SP year date3 = month SP (2DIGIT / (SP 1DIGIT)) ;例如，
6 月 2 日

HTTP 日期区分大小写。发送者不得在 HTTP 日期中生成额外的空格,超出语法中作为 SP 明确包含的空格。 day-name、day、month、year 和 time-of-day 的语义与为具有相应名称的 Internet 消息格式构造定义的语义相同（[RFC5322],第 3.3 节）。

使用两位数年份的 rfc850-date 格式的时间戳值的接收者必须将未来 50 年以上的时间戳解释为代表过去具有相同最后两位数的最近一年.

除非字段定义另有限制,否则鼓励时间戳值的接收者在解析时间戳时保持健壮。例如,消息偶尔会从非 HTTP 源通过 HTTP 转发,这可能会生成 Internet 消息格式定义的任何日期和时间规范。

注意:日期/时间戳格式的 HTTP 要求仅适用于它们在协议流中的使用。实现不需要将这些格式用于用户呈现、请求记录等。

7.1.1.2.日期

“Date”头字段表示消息发出的日期和时间,与 [RFC5322] 第 3.6.1 节中定义的发出日期字段 (orig-date) 具有相同的语义。字段值是 HTTP 日期,如第 7.1.1.1 节中所定义。

日期 = HTTP 日期

一个例子是

日期:1994 年 11 月 15 日星期二 08:12:31 GMT

当生成 Date 头字段时,发送方应该生成其字段值作为消息生成日期和时间的最佳可用近似值。理论上,日期应该代表生成有效载荷之前的时刻。实际上,日期可以在消息发起期间的任何时间生成。

如果原始服务器没有能够提供协调世界时当前实例的合理近似值的时钟,则它不得发送 Date 标头字段。如果响应是 1xx（信息)或 5xx（服务器错误)状态代码类,源服务器可以发送日期头字段。在所有其他情况下,源服务器必须发送日期头字段。

接收没有 Date 头字段的响应消息的带有时钟的接收者必须记录它被接收的时间,并且如果它被缓存或转发到下游,则将相应的 Date 头字段附加到消息的头部分。

用户代理可以在请求中发送 Date 标头字段,但通常不会这样做,除非它被认为可以向服务器传达有用的信息。例如,如果期望服务器根据用户代理和服务器时钟之间的差异调整其对用户请求的解释,则 HTTP 的自定义应用程序可能会传达一个日期。

7.1.2.地点

“Location”头字段在某些响应中用于引用与响应相关的特定资源。关系的类型由请求方法和状态代码语义的组合定义。

位置 = URI 参考

字段值由单个 URI 引用组成。当它具有相对引用的形式（[RFC3986],第 4.2 节)时,最终值是通过根据有效请求 URI（[RFC3986],第 5 节)解析它来计算的。

对于 201（已创建)响应,Location 值指的是请求创建的主要资源。对于 3xx（重定向)响应,Location 值是指用于自动重定向请求的首选目标资源。

如果在 3xx（重定向)响应中提供的 Location 值没有片段组件,用户代理必须处理重定向,就好像该值继承了用于生成请求目标的 URI 引用的片段组件（即,重定向继承原始参考的片段,如果有的话）。

例如,为 URI 引用“http://www.example.org/~tim”生成的 GET 请求可能会导致包含标头字段的 303（参见其他)响应：

位置：/People.html#tim

这表明用户代理重定向到“http://www.example.org/People.html#tim”

同样,为 URI 引用“http://www.example.org/index.html#larry”生成的 GET 请求可能会导致包含标头字段的 301
（永久移动)响应：

位置:http://www.example.net/index.html

这表明用户代理重定向到“http://www.example.net/index.html#larry”,保留原始片段标识符。

在某些情况下,位置值中的片段标识符是不合适的。例如,201（已创建)响应中的 Location 标头字段应该提供特定于已创建资源的 URI。

注意:一些收件人试图从不是有效 URI 引用的位置字段中恢复。本规范不强制或定义此类处理,但出于稳健性考虑允许这样做。

注意:Content-Location 头字段（第 3.1.4.2 节)与 Location 的不同之处在于 Content-Location 指的是与所附表示相对应的最具体的资源。

因此,响应可能同时包含 Location 和 Content-Location 标头字段。

7.1.3.之后重试

服务器发送“Retry-After”标头字段以指示用户代理在发出后续请求之前应该等待多长时间。当与 503（服务不可用)响应一起发送时,Retry-After 指示预计服务对客户端不可用的时间。

当与任何 3xx（重定向)响应一起发送时,Retry-After 指示用户代理在发出重定向请求之前被要求等待的最短时间。

此字段的值可以是 HTTP 日期或收到响应后延迟的秒数。

Retry-After = HTTP-date / delay-seconds

delay-seconds 值是一个非负十进制整数,表示以秒为单位的时间。

延迟秒数 = 1*DIGIT

它的两个使用示例是

```
重试之后:1999 年 12 月 31 日星期五 23:59:59 GMT
重试后:120
```

在后一个示例中,延迟为 2 分钟。

7.1.4.各不相同

响应中的“Vary”头字段描述了请求消息的哪些部分,除了方法、主机头字段和请求目标之外,可能会影响源服务器选择和表示此响应的过程。该值由单个星号 (*) 或标题字段名称列表(不区分大小写)组成。

```
变化 = * / 1#字段名
```

“*”的 Vary 字段值表示关于请求的任何内容都可能在选择响应表示中发挥作用,可能包括消息语法之外的元素(例如,客户端的网络地址)。如果不将请求转发到源服务器,接收者将无法确定此响应是否适合以后的请求。代理不得生成具有

“*“价值。

由逗号分隔的名称列表组成的 Vary 字段值指示命名的请求标头字段(称为选择标头字段)可能在选择表示中起作用。潜在的选择报头字段不限于本规范定义的那些。

例如,包含的响应

```
变化:接受编码,接受语言
```

表示源服务器在选择此响应的内容时可能已使用请求的 Accept-Encoding 和 Accept-Language 字段(或缺少)作为决定因素。

源服务器可能会发送带有两个字段列表的 Vary
用途:

1. 通知缓存接收者他们不得使用此响应
   以满足后来的请求,除非后来的请求对列出的字段具有与原始请求相同的值([RFC7234] 的第 4.1 节)。换句话说,Vary 扩展了将新请求与存储的缓存条目匹配所需的缓存键。

2. 通知用户代理接收者此响应受内容协商（第 5.3 节)的约束,并且如果在列出的标头字段中提供附加参数（主动协商）,则可能会在后续请求中发送不同的表示。

当源服务器选择表示的算法根据请求消息的方面而不是方法和请求目标而变化时,源服务器应该发送一个 Vary 头字段,除非不能跨越差异或源服务器已被故意配置为防止缓存透明.例如,无需在 Vary 中发送授权字段名称,因为跨用户的重用受字段定义的限制（[RFC7235] 的第 4.2 节）。同样,源服务器可能会使用 Cache-Control 指令（[RFC7234] 的第 5.2 节)来取代 Vary,如果它认为差异不如 Vary 对缓存的影响的性能成本那么重要的话。

7.2.验证器标头字段

验证器标头字段传达有关所选表示的元数据（第 3 节）。在响应安全请求时,验证器字段描述了源服务器在处理响应时选择的选定表示。请注意,根据状态代码语义,为给定响应选择的表示不一定与作为响应负载包含的表示相同。

在对状态更改请求的成功响应中,验证器字段描述了作为处理请求的结果已替换先前选择的表示的新表示。

例如,201（已创建)响应中的 ETag 标头字段传达新创建资源表示的实体标签,以便它可以在以后的条件请求中使用,以防止“丢失更新”问题 [RFC7232]。

标题字段名称	定义在...
电子标签	[RFC7232] 的第 2.3 节
最后修改	[RFC7232] 的第 2.2 节

7.3.身份验证挑战

身份验证挑战指示客户端可以使用哪些机制在未来的请求中提供身份验证凭据。

标题字段名称	定义在...
WWW-认证	[RFC7235] 第 4.1 节
代理验证	[RFC7235] 第 4.3 节

7.4.响应上下文

其余的响应标头字段提供了有关目标资源的更多信息,以便在以后的请求中使用。

标题字段名称	定义在...
接受范围	[RFC7233] 的第 2.3 节
允许	第 7.4.1 节
服务器	第 7.4.2 节

7.4.1.允许

“允许”标头字段列出了目标资源支持的一组方法。这个字段的目的是严格地通知接收者与资源相关的有效请求方法。

允许=#method

使用示例:

允许:GET、HEAD、PUT

实际允许的方法集由源服务器在每次请求时定义。源服务器必须在 405（方法不允许）响应中生成允许字段,并且可以在任何其他响应中这样做。空的 Allow 字段值表示该资源不允许任何方法,如果该资源已被配置暂时禁用,则可能会在 405 响应中出现这种情况。

代理不得修改 Allow 头字段 它不需要理解所有指定的方法来根据通用消息处理规则处理它们。

7.4.2.服务器

“服务器”头字段包含有关源服务器用于处理请求的软件的信息,客户端通常使用这些信息来帮助确定报告的互操作性问题的范围,解决或定制请求以避免特定的服务器限制,以及用于有关服务器或操作系统使用情况的分析。源服务器可以在其

回应。

服务器 = 产品 *(RWS (产品 / 评论))

服务器字段值由一个或多个产品标识符组成,每个产品标识符后跟零个或多个注释（[RFC7230] 的第 3.2 节）,它们共同标识源服务器软件及其重要的子产品。按照惯例,产品标识符按照其识别原始服务器软件的重要性的降序排列。每个产品标识符都包含一个名称和可选版本,如第 5.5.3 节中所定义。

例子：

服务器:CERN/3.0 libwww/2.17

源服务器不应该生成包含不必要的细粒度细节的服务器字段,并且应该限制第三方添加子产品。过长和详细的服务器字段值会增加响应延迟并可能泄露内部实现细节,这可能使攻击者（稍微)更容易找到和利用已知的安全漏洞。

8. IANA 考虑

8.1.方法注册

“超文本传输协议 (HTTP) 方法注册表”定义了请求方法令牌的名称空间（第 4 节）。方法注册表已创建,现在维护在 <http://www.iana.org/assignments/http-methods>。

8.1.1.程序

HTTP 方法注册必须包含以下字段：

o 方法名称（见第 4 节）

o 安全（“是”或“否”,参见第 4.2.1 节）

o 幂等（“是”或“否”,参见第 4.2.2 节）

o 指向规范文本的指针

要添加到此名称空间的值需要 IETF 审查（参见 [RFC5226],第 4.1 节）。

8.1.2.新方法的注意事项

标准化方法是通用的;也就是说,它们可能适用于任何资源,而不仅仅是一种特定的媒体类型、资源种类或应用程序。因此,最好在不特定于单个应用程序或数据格式的文档中注册新方法,因为正交技术值得正交规范。

由于消息解析（[RFC7230] 的第 3.3 节)需要独立于方法语义（除了对 HEAD 的响应）,新方法的定义不能更改解析算法或禁止请求或响应中出现消息体信息。新方法的定义可以通过要求 Content-Length 标头字段值为“0”来指定仅允许零长度的消息正文。

一个新的方法定义需要指明它是否安全（第 4.2.1 节）、幂等（第 4.2.2 节）、可缓存（第 4.2.3 节）,如果有效载荷主体中存在任何语义,则与有效载荷主体相关联的语义是什么请求以及该方法对标头字段或状态代码语义进行了哪些改进。如果新方法是可缓存的,它的定义应该描述缓存如何以及在什么条件下可以存储响应并使用它来满足后续请求。新方法应该描述它是否可以成为有条件的（第 5.2 节）,如果可以,当条件为假时服务器如何响应。同样,如果新方法可能对部分响应语义有一些用途（[RFC7233]）,它也应该对此进行记录。

注意:避免定义以“M-”开头的方法名称,因为该前缀可能会被误解为具有 [RFC2774] 分配给它的语义。

8.1.3.登记

“超文本传输协议 (HTTP) 方法注册表”已填充以下注册：

方法	安全	幂等	参考
连接	没有	不	第 4.3.6 节
删除	没有	是的	第 4.3.5 节
获取	是	是的	第 4.3.1 节
头	是	是的	第 4.3.2 节
选项	是	是的	第 4.3.7 节
发布	没有	不	第 4.3.3 节
放	没有	是的	第 4.3.4 节
追踪	是	是的	第 4.3.8 节

8.2.状态代码注册表

“超文本传输协议 (HTTP) 状态代码注册表”定义了响应状态代码令牌的命名空间（第 6 节）。状态代码注册表维护在 <http://www.iana.org/assignments/http-status-codes>。

本节取代了先前在 [RFC2817] 的第 7.1 节中定义的 HTTP 状态代码的注册过程。

8.2.1.程序

注册必须包括以下字段：

o 状态代码（3 位数字）

o 简短描述

o 指向规范文本的指针

要添加到 HTTP 状态代码命名空间的值需要 IETF 审查（参见 [RFC5226],第 4.1 节）。

8.2.2.新状态代码的注意事项

当需要表达当前状态代码未定义的响应的语义时,可以注册一个新的状态代码。

状态代码是通用的;它们可能适用于任何资源,而不仅仅是一种特定的媒体类型、资源种类或 HTTP 应用程序。因此,最好在不特定于单个应用程序的文档中注册新的状态代码。

新的状态代码需要属于第 6 节中定义的类别之一。为了允许现有的解析器处理响应消息,新的状态代码不能禁止有效负载,尽管它们可以强制使用零长度的有效负载主体。

尚未广泛部署的新状态代码的提案应避免为代码分配特定编号,直到明确同意将其注册为止;相反,早期的草案可以使用诸如“4NN”或“3N0”..“3N9”之类的符号来指示提议的状态代码的类别,而不会过早地使用数字。

新状态代码的定义应该解释导致包含该状态代码的响应的请求条件(例如,请求标头字段和/或方法的组合)以及对响应标头字段的任何依赖性(例如,什么字段是必需的,哪些字段可以修改语义,以及在与新状态代码一起使用时进一步细化哪些头字段语义)。

新状态码的定义应该指定它是否可缓存。请注意,如果出现的响应具有明确的新鲜度信息,则可以缓存所有状态代码;然而,定义为可缓存的状态代码允许在没有显式新鲜度信息的情况下被缓存。同样,状态代码的定义可以对缓存行为施加约束。有关详细信息,请参阅 [RFC7234]。

最后,新状态码的定义应该表明有效负载是否与已识别资源有任何隐含关联(第 3.1.4.1 节)。

8.2.3.登记

状态代码注册表已更新为以下注册:

价值	描述	参考
100	继续	第 6.2.1 节
101	交换协议	第 6.2.2 节
200	好的	第 6.3.1 节
201	已创建	第 6.3.2 节
202	已接受	第 6.3.3 节
203	非权威信息	第 6.3.4 节
204	无内容	第 6.3.5 节
205	重置内容	第 6.3.6 节
206	部分内容	第 4.1 节
300	多种选择	第 6.4.1 节
301	永久移动	第 6.4.2 节
302	找到	第 6.4.3 节
303	见其他	第 6.4.4 节
304	未修改	第 4.1 节
305	使用代理	第 6.4.5 节
306	（未使用）	第 6.4.6 节
307	临时重定向	第 6.4.7 节
400	错误请求	第 6.5.1 节
401	未经授权	第 3.1 节
402	需要付款	保留
403	禁止	第 6.5.3 节
404	未找到	第 6.5.4 节
405	不允许使用该方法	第 6.5.5 节
406	不可接受	第 6.5.6 节
407	需要代理身份验证	第 3.2 节
408	请求超时	第 6.5.7 节
409	冲突	第 6.5.8 节
410	消失了	第 6.5.9 节
411	长度要求	第 6.5.10 节
412	前提条件失败	第 4.2 节
413	请求实体太大	第 6.5.11 节
414	请求-URI 太长	第 6.5.12 节
415	不支持的媒体类型	第 6.5.13 节
416	请求范围不满足	第 4.4 节
417	期望失败	第 6.5.14 节
426	需要升级	第 6.5.15 节
500	内部服务器错误	第 6.6.1 节
501	未实现	第 6.6.2 节
502	错误的网关	第 6.6.3 节
503	服务不可用	第 6.6.4 节
504	网关超时	第 6.6.5 节
505	不支持 HTTP 版本	第 6.6.6 节

8.3.标头字段注册表

HTTP 标头字段在位于 <http://www.iana.org/assignments/message-headers> 的“消息标头”注册表中注册，如 [BCP90] 所定义。

8.3.1.新标题字段的注意事项

标头字段是键值对,可用于传递有关消息、消息有效负载、目标资源或连接（即控制数据)的数据。有关 HTTP 消息中标头字段语法的一般定义，请参阅 [RFC7230] 的第 3.2 节。

[BCP90] 中定义了头域名称的要求。

建议定义新字段的规范的作者使名称尽可能短,并且不要在名称前加上“X-”,除非标题字段永远不会在 Internet 上使用。 （“X-”前缀习语在实践中被广泛滥用;它的目的只是用作避免专有软件或内部网处理中的名称冲突的机制,因为前缀将确保私有名称永远不会与新注册的名称冲突Internet 名称;有关详细信息,请参阅 [BCP178]）。

新的标头字段值通常使用 ABNF ([RFC5234]) 定义语法,必要时使用 [RFC7230] 第 7 节中定义的扩展,并且通常限制在 US-ASCII 字符范围内。需要更大范围字符的标头字段可以使用 [RFC5987] 中定义的编码。

原始字段值中的前导和尾随空格在字段解析时被删除（[RFC7230] 的第 3.2.4 节）。值中前导或尾随空格很重要的字段定义将必须使用容器语法，例如引号字符串（[RFC7230] 的第 3.2.6 节）。

由于逗号（ , ）用作字段值之间的通用分隔符,因此如果允许在字段值中使用逗号,则需要谨慎对待。通常,可能包含逗号的组件使用带引号的字符串 ABNF 产生式用双引号保护。

例如,文本日期和 URI（其中任何一个都可能包含逗号)可以安全地包含在如下字段值中：

示例 URI 字段：“http://example.com/a.html,foo”、“http://without-a-comma.example.com/”
示例日期字段：“1996 年 5 月 4 日星期六”、“2005 年 9 月 14 日星期三”

请注意,双引号定界符几乎总是用于带引号的字符串生成;在双引号内使用不同的语法可能会造成不必要的混淆。

许多标题字段使用一种格式,包括（不区分大小写)命名参数（例如,Content-Type,在第 3.1.1.5 节中定义）。

允许参数值的不带引号（令牌)和带引号（带引号的字符串)语法使接收者能够使用现有的解析器组件。当允许这两种形式时,参数值的含义应该独立于用于它的语法（例如,参见第 3.1.1.1 节中关于媒体类型的参数处理的注释）。

建议定义新标头字段的规范作者考虑记录：

o 该字段是单个值还是可以是列表（以逗号分隔;参见 [RFC7230] 的第 3.2 节）。

如果它不使用列表语法,请记录如何处理该字段多次出现的消息（明智的默认设置是忽略该字段,但这可能并不总是正确的选择）。

请注意,中介和软件库可能会将多个标头字段实例组合成一个,尽管该字段的定义不允许使用列表语法。稳健的格式使接收者能够发现这些情况（好的例子：“Content-Type”,因为逗号只能出现在带引号的字符串内;不好的例子：“Location”,因为逗号可以出现在 URI 内）。

o 在什么条件下可以使用头字段;例如,仅在响应或请求中,在所有消息中,仅在对特定请求方法的响应中,等等。

o 该字段是否应由原始服务器存储
根据 PUT 请求理解它。

o 字段语义是否通过上下文进一步细化,例如通过现有的请求方法或状态代码。

o 在 Connection 报头字段中列出字段名称是否合适（即,如果报头字段是逐跳的;请参阅 [RFC7230] 的第 6.1 节）。

o 在什么条件下允许中介机构插入,
删除或修改字段的值。

o 在 Vary 中列出 field-name 是否合适
响应头字段（例如,当请求头字段被源服务器的内容选择算法使用时;参见第 7.1.4 节）。

o 标题字段在预告片中是否有用或允许（参见
[RFC7230] 的第 4.1 节）。

o 标题字段是否应该在重定向中保留。

o 它是否引入了任何额外的安全考虑,例如
作为隐私相关数据的披露。

8.3.2.登记

“消息标题”注册表已更新为以下永久注册：

标题字段名称	协议	状态	参考
接受	网址	标准	第 5.3.2 节 | 标准 | 第 5.3.3 节 | 标准 | 第 5.3.4 节 | 标准 | 第 5.3.5 节 | 标准 | 第 7.4.1 节 | 标准 | 第 3.1.2.2 节 | 标准 | 第 3.1.3.2 节 | 标准 | 第 3.1.4.2 节 | 标准 | 第 3.1.1.5 节 | 标准 | 第 7.1.1.2 节 | 标准 | 第 5.1.1 节 | 标准 | 第 5.5.1 节 | 标准 | 第 7.1.2 节 | 标准 | 第 5.1.2 节 | 标准 | 附录 A.1 | 标准 | 第 5.5.2 节 | 标准 | 第 7.1.3 节 | 标准 | 第 7.4.2 节 | 标准 | 第 5.5.3 节 | 标准 | 第 7.1.4 节 |
接受字符集	网址		
接受编码	网址		
接受语言	网址		
允许	网址		
内容编码	网址		
内容语言	网址		
内容位置	网址		
内容类型	网址		
日期	网址		
期待	网址		
来自	网址		
地点	网址		
最大前锋	网址		
MIME 版本	网址		
推荐人	网址		
之后重试	网址		
服务器	网址		
用户代理	网址		
变化	网址		

上述注册的变更控制者是：“IETF (iesg@ietf.org) - Internet Engineering Task Force”。

8.4.内容编码注册表

“HTTP 内容编码注册表”定义了内容编码名称的名称空间（[RFC7230] 的第 4.2 节）。内容编码注册表维护在 <http://www.iana.org/assignments/http-parameters>。

8.4.1.程序

内容编码注册必须包括以下字段：

姓名

o 说明

o 指向规范文本的指针

内容编码的名称不得与传输编码的名称重叠（[RFC7230] 的第 4 节），除非编码转换是相同的（如 [RFC7230] 第 4.2 节中定义的压缩编码的情况）。

添加到此命名空间的值需要 IETF 审查（参见 [RFC5226] 的第 4.1 节)并且必须符合本节中定义的内容编码的目的。

8.4.2.登记

“HTTP 内容编码注册表”已更新为以下注册：

名称	描述	参考
身份	保留（| 第 5.3.4 节中“无编码”的同义词		
	接受编码）	

9. 安全注意事项

本节旨在告知开发人员、信息提供者和用户与 HTTP 语义及其在 Internet 上传输信息的用途相关的已知安全问题。

[RFC7230] 的第 9 节讨论了与消息语法、解析和路由相关的注意事项。

以下注意事项列表并不详尽。大多数与 HTTP 语义相关的安全问题都是关于保护服务器端应用程序（HTTP 接口背后的代码）、保护用户代理

处理通过 HTTP 接收的有效负载,或一般安全使用 Internet,而不是协议的安全性。各种组织维护主题信息和当前 Web 应用程序安全研究的链接（例如,[OWASP]）。

9.1.基于文件名和路径名的攻击

源服务器经常使用它们的本地文件系统来管理从有效请求 URI 到资源表示的映射。大多数文件系统并不是为了防止恶意文件或路径名而设计的。因此,源站在将请求目标映射到文件、文件夹或目录时,需要避免访问对系统有特殊意义的名称。

例如,UNIX、Microsoft Windows 等操作系统使用“..”作为路径组成部分,表示当前目录的上一级目录,它们使用特殊命名的路径或文件名向系统设备发送数据。类似的命名约定可能存在于其他类型的存储系统中。同样,在处理无效或意外字符、分解字符的重组以及不区分大小写的名称的大小写规范化时,本地存储系统有一个令人讨厌的倾向,即更倾向于用户友好而不是安全。

基于此类特殊名称的攻击往往侧重于拒绝服务（例如,告诉服务器从 COM 端口读取）或泄露不应提供的配置和源文件。

9.2.基于命令、代码或查询注入的攻击

源服务器通常使用 URI 中的参数作为识别系统服务、选择数据库条目或选择数据源的手段。但是,请求中收到的数据是不可信的。攻击者可以构造任何请求数据元素（方法、请求目标、标头字段或正文）以包含在通过命令调用、语言解释器或数据库传递时可能被误解为命令、代码或查询的数据界面。

例如,SQL 注入是一种常见的攻击,其中在请求目标或标头字段（例如,Host、Referer 等）的某些部分中插入了额外的查询语言。如果接收到的数据直接在 SELECT 语句中使用,则查询语言可能被解释为数据库命令而不是简单的字符串值。

这种类型的实施漏洞非常普遍,尽管很容易预防。

一般来说,资源实现应该避免在处理或解释为指令的上下文中使用请求数据。

参数应该与固定字符串进行比较,并根据比较的结果采取行动,而不是通过未为不受信任的数据准备的接口传递。接收到的不是基于固定参数的数据应该仔细过滤或编码以避免被误解。

类似的注意事项适用于存储和稍后处理请求数据时,例如在日志文件、监控工具中,或者当包含在允许嵌入脚本的数据格式中时。

9.3.个人信息的披露

客户通常不了解大量个人信息,包括用户提供的与资源交互的信息(例如,用户的姓名、位置、邮件地址、密码、加密密钥等)和有关用户浏览活动的信息时间(例如历史、书签等)。实施需要防止无意中泄露个人信息。

9.4. URI 中敏感信息的泄露

URI 旨在共享,而不是保护,即使它们标识安全资源。 URI 通常显示在显示器上,打印页面时添加到模板,并存储在各种未受保护的书签列表中。因此,在 URI 中包含敏感的、个人可识别的或有泄露风险的信息是不明智的。

服务的作者应该避免使用基于 GET 的表单来提交敏感数据,因为该数据将被放置在请求目标中。许多现有的服务器、代理和用户代理在第三方可能可见的地方记录或显示请求目标。此类服务应该改用基于 POST 的表单提交。

由于 Referer 标头字段告诉目标站点有关导致请求的上下文,它有可能揭示有关用户的即时浏览历史的信息以及可能在引用资源的 URI 中找到的任何个人信息。 Referer 头字段的限制在第 5.5.2 节中描述,以解决它的一些安全问题。

9.5.重定向后片段的披露

虽然在 URI 引用中使用的片段标识符不会在请求中发送,但实施者应该知道它们将对用户代理以及作为响应结果运行的任何扩展或脚本可见。特别是,当发生重定向并且原始请求的片段标识符被 Location 中的新引用继承时（第 7.1.2 节）,这可能会产生将一个站点的片段泄露给另一个站点的效果。如果第一个站点在片段中使用个人信息,它应该确保重定向到其他站点包括一个（可能是空的)片段组件以阻止该继承。

9.6.产品信息披露

User-Agent（第 5.5.3 节）、Via（[RFC7230] 的第 5.7.1 节)和服务器（第 7.4.2 节)标头字段通常会显示有关各个发件人软件系统的信息。从理论上讲,这可以使攻击者更容易利用已知的安全漏洞;实际上,攻击者倾向于尝试所有潜在的漏洞,而不管所使用的明显软件版本如何。

作为通过网络防火墙的门户的代理应该采取特殊的预防措施来传输可能识别防火墙后面主机的标头信息。 Via 标头字段允许中间人用假名替换敏感的机器名称。

9.7.浏览器指纹识别

浏览器指纹识别是一组技术,用于通过其独特的特征集随时间识别特定用户代理。这些特征可能包括与其 TCP 行为、特性功能和脚本环境相关的信息,尽管这里特别令人感兴趣的是可能通过 HTTP 进行通信的一组独特特征。指纹识别被认为是一个隐私问题,因为它可以跟踪用户代理随时间的行为,而无需用户可能对其他形式的数据收集（例如 cookie)进行相应的控制。许多通用用户代理（即 Web 浏览器)已采取措施减少其指纹。

有许多请求标头字段可能会向服务器显示足够唯一以启用指纹识别的信息。 From 报头字段是最明显的,尽管预计只有在用户需要自我识别时才会发送 From。同样,Cookie 标头字段是故意的

旨在启用重新识别,因此指纹问题仅适用于 cookie 被用户代理的配置禁用或限制的情况。

User-Agent 标头字段可能包含足够的信息来唯一标识特定设备,通常与其他特征结合使用时,尤其是当用户代理发送有关用户系统或扩展的过多详细信息时。然而,用户最不期望的唯一信息来源是主动协商（第 5.3 节）,包括 Accept、Accept-Charset、Accept-Encoding 和 Accept-Language 标头字段。

除了指纹问题之外,Accept-Language 标头字段的详细使用可以揭示用户可能认为具有隐私性质的信息。例如,理解给定的语言集可能与特定种族群体的成员身份密切相关。一种限制这种隐私损失的方法是用户代理省略发送 Accept-Language 除了已列入白名单的站点,可能是在检测到指示语言协商可能有用的 Vary 标头字段后通过交互。

在使用代理来增强隐私的环境中,用户代理在发送主动协商标头字段时应该保守。提供高度标头字段可配置性的通用用户代理应该告知用户如果提供过多的细节可能会导致隐私丢失。

作为一种极端的隐私措施,代理可以过滤中继请求中的主动协商标头字段。

10.致谢

请参阅 [RFC7230] 的第 10 节。

11. 参考资料

11.1.规范性参考文献

[RFC2045] Freed, N. 和 N. Borenstein，“多用途互联网邮件
扩展 (MIME) 第一部分:Internet 消息体的格式”,RFC 2045,1996 年 11 月。

[RFC2046] Freed, N. 和 N. Borenstein，“多用途互联网邮件
扩展 (MIME) 第二部分:媒体类型”,RFC 2046,1996 年 11 月。

[RFC2119] Bradner, S.，“RFC 中用于指示的关键词
要求级别”,BCP 14,RFC 2119,1997 年 3 月。

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

[RFC4647] Phillips, A., Ed.和 M. Davis, Ed.，“语言匹配
标签”，BCP 47，RFC 4647，2006 年 9 月。

[RFC5234] Crocker, D., Ed.和 P. Overell，“用于语法规范的增强 BNF：ABNF”，STD 68，RFC 5234，2008 年 1 月。

[RFC5646] Phillips, A., Ed.和 M. Davis, Ed.，“用于识别的标签
语言”，BCP 47，RFC 5646，2009 年 9 月。

[RFC6365] Hoffman, P. 和 J. Klensin，“术语用于
IETF 中的国际化”，BCP 166，RFC 6365，2011 年 9 月。

[RFC7230] 菲尔丁，R.，Ed。和 J. Reschke, Ed.，“超文本传输协议 (HTTP/1.1)：消息语法和路由”，RFC 7230，2014 年 6 月。

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

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
“超文本传输协议 (HTTP/1.1)：范围请求”，RFC 7233，2014 年 6 月。

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed. 和 J. Reschke, Ed.，“超文本传输协议 (HTTP/1.1)：缓存”，RFC 7234，2014 年 6 月。

[RFC7235] 菲尔丁，R.，Ed。和 J. Reschke, Ed.，“超文本传输协议 (HTTP/1.1)：身份验证”，RFC 7235，2014 年 6 月。

11.2.信息参考

[BCP13] Freed, N.、Klensin, J. 和 T. Hansen，“媒体类型规范和注册程序”，BCP 13，RFC 6838，2013 年 1 月。

[BCP178] Saint-Andre, P.、Crocker, D. 和 M. Nottingham，
“弃用应用程序协议中的“X-”前缀和类似结构”，BCP 178，RFC 6648，2012 年 6 月。

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

[OWASP] van der Stock, A., Ed.，“构建安全 Web 应用程序和 Web 服务指南”，开放 Web 应用程序安全项目 (OWASP) 2.0.1，2005 年 7 月，<https://www.owasp.org/ >。

[休息] Fielding, R.，“架构风格和基于网络的软件架构设计”，博士论文，加州大学尔湾分校，2000 年 9 月，<http://roy.gbiv.com/pubs/dissertation/top.htm> .

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

[RFC2049] Freed, N. 和 N. Borenstein，“多用途 Internet 邮件扩展 (MIME) 第五部分：一致性标准和示例”，RFC 2049，1996 年 11 月。

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

[RFC2295] Holtman, K. 和 A. Mutz，“HTTP 中的透明内容协商”，RFC 2295，1998 年 3 月。

[RFC2388] Masinter, L.，“从表单返回值：多部分/表单数据”，RFC 2388，1998 年 8 月。

[RFC2557] Palme, F.、Hopmann, A.、Shelness, N. 和 E. Stefferud，“聚合文档的 MIME 封装，例如 HTML (MHTML)”，RFC 2557，1999 年 3 月。

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

[RFC2774] Frystyk, H.、Leach, P. 和 S. Lawrence，“HTTP 扩展框架”，RFC 2774，2000 年 2 月。

[RFC2817] Khare, R. 和 S. Lawrence，“在内部升级到 TLS HTTP/1.1”，RFC 2817，2000 年 5 月。

[RFC2978] Freed, N. 和 J. Postel，“IANA 字符集注册程序”，BCP 19，RFC 2978，2000 年 10 月。

[RFC5226] Narten, T. 和 H. Alvestrand，“编写
RFC 中的 IANA Considerations Section”，BCP 26，RFC 5226，2008 年 5 月。

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

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

[RFC5789] Dusseault, L. 和 J. Snell，“HTTP 的 PATCH 方法”，RFC 5789，2010 年 3 月。

[RFC5905] Mills, D.、Martin, J.、Ed.、Burbank, J. 和 W. Kasch，
“网络时间协议版本 4：协议和算法规范”，RFC 5905，2010 年 6 月。

[RFC5987] Reschke, J.，“字符集和语言编码
超文本传输协议 (HTTP) 标头字段参数”，RFC 5987，2010 年 8 月。

[RFC5988] Nottingham, M.，“网络链接”，RFC 5988，2010 年 10 月。

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

[RFC6266] Reschke, J.，“在超文本传输协议 (HTTP) 中使用内容配置标头字段”，RFC 6266，2011 年 6 月。

[RFC7238] Reschke, J.，“超文本传输协议 (HTTP)
状态代码 308（永久重定向）”，RFC 7238，2014 年 6 月。

附录 A. HTTP 和 MIME 之间的区别

HTTP/1.1 使用许多为 Internet 消息格式 [RFC5322] 和多用途 Internet 邮件扩展 (MIME) 定义的结构

[RFC2045] 允许消息体以开放的多种表示形式和可扩展的报头字段进行传输。

然而,RFC 2045 只关注电子邮件； HTTP 的应用程序有许多不同于电子邮件的特性;因此,HTTP 具有不同于 MIME 的特性。这些差异经过精心选择，以优化二进制连接的性能,允许更大的自由使用新媒体类型,使日期比较更容易,并承认一些早期 HTTP 服务器和客户端的做法。

本附录描述了 HTTP 与 MIME 不同的特定领域。
进出严格 MIME 环境的代理和网关需要了解这些差异,并在必要时提供适当的转换。

A.1. MIME 版本

HTTP 不是 MIME 兼容协议。但是,消息可以包含单个 MIME-Version 标头字段以指示使用哪个版本的 MIME 协议来构造消息。使用 MIME-Version 标头字段表示消息完全符合 MIME 协议（如 [RFC2045] 中定义）。

将 HTTP 消息导出到严格的 MIME 环境时,发件人有责任确保完全符合（如果可能）。

A2。转换为规范形式

MIME 要求 Internet 邮件正文部分在传输之前转换为规范格式,如 [RFC2049] 第 4 节所述。本文档的第 3.1.1.3 节描述了通过 HTTP 传输时“文本”媒体类型的子类型所允许的形式。 [RFC2046] 要求具有“文本”类型的内容将换行符表示为 CRLF,并禁止在换行符序列之外使用 CR 或 LF。 HTTP 允许 CRLF、裸 CR 和裸 LF 指示文本内容中的换行符。

从 HTTP 到严格 MIME 环境的代理或网关应该将本文档第 3.1.1.3 节中描述的文本媒体类型中的所有换行符转换为 CRLF 的 RFC 2049 规范形式。但是请注意,由于内容编码的存在以及 HTTP 允许使用某些不使用八位字节 13 和 10 分别表示 CR 和 LF 的字符集,这可能会变得复杂。

除非原始内容已经是规范形式,否则转换将破坏应用于原始内容的任何加密校验和。因此,对于在 HTTP 中使用此类校验和的任何内容,建议使用规范形式。

A.3.日期格式转换

HTTP/1.1 使用一组受限制的日期格式（第 7.1.1.1 节)来简化日期比较过程。来自其他协议的代理和网关应该确保消息中出现的任何日期标头字段符合 HTTP/1.1 格式之一,并在必要时重写日期。

A.4.内容编码的转换

MIME 不包含任何等同于 HTTP/1.1 的 Content-Encoding 标头字段的概念。由于这充当媒体类型的修饰符,从 HTTP 到 MIME 兼容协议的代理和网关应该更改 Content-Type 标头字段的值或在转发消息之前解码表示。

（某些用于 Internet 邮件的 Content-Type 实验性应用程序使用媒体类型参数“;conversions=<content-coding>”来执行与 Content-Encoding 等效的功能。但是,此参数不是 MIME 标准的一部分).

A.5. Content-Transfer-Encoding 的转换

HTTP 不使用 MIME 的 Content-Transfer-Encoding 字段。
从 MIME 兼容协议到 HTTP 的代理和网关需要在将响应消息传递给 HTTP 客户端之前删除任何内容传输编码。

从 HTTP 到 MIME 兼容协议的代理和网关负责确保消息采用正确的格式和编码,以便在该协议上安全传输,其中“安全传输”由所使用协议的限制定义。

这样的代理或网关应该使用适当的内容传输编码来转换和标记数据,如果这样做会提高通过目标协议安全传输的可能性。

A.6. MHTML 和行长度限制

与 MHTML [RFC2557] 实现共享代码的 HTTP 实现需要注意 MIME 行长度限制。

由于 HTTP 没有这个限制,所以 HTTP 不会折叠长行。由 HTTP 传输的 MHTML 消息遵循 MHTML 的所有约定,包括行长度限制和折叠、规范化等,因为 HTTP 将消息体传输为

有效载荷,并且除了“multipart/byteranges”类型（[RFC7233] 的附录 A)之外,不解释其中可能包含的内容或任何 MIME 标题行。

附录 B. RFC 2616 的变化

此修订版中的主要更改本质上是编辑性的:提取消息传递语法并将 HTTP 语义划分为核心功能、条件请求、部分请求、缓存和身份验证的单独文档。一致性语言已被修订以明确针对需求,并且术语已得到改进以区分有效载荷与表示和表示与资源。

添加了一项新要求,即当 URI 中嵌入的语义与请求方法不一致时,禁用这些语义,因为这是互操作性失败的常见原因。

（第 2 节）

添加了一种算法,用于确定有效载荷是否与特定标识符相关联。 （第 3.1.4.1 节）

文本媒体类型的默认 ISO-8859-1 字符集已被删除;默认值现在是媒体类型定义所说的任何内容。

同样,ISO-8859-1 的特殊处理已从 Accept-Charset 标头字段中删除。 （第 3.1.1.3 节和第 5.3.3 节）

Content-Location 的定义已更改为不再影响用于解析相对 URI 引用的基本 URI,这是由于实施支持不佳以及可能破坏内容协商资源中的相对链接的不良影响。

（第 3.1.4.2 节）

为了与 [RFC7230] 的方法中立解析算法保持一致,GET 的定义已经放宽,因此请求可以有一个主体,即使主体对 GET 没有意义。

（第 4.3.1 节）

服务器不再需要处理所有 Content-* 标头字段,并且在 PUT 请求中明确禁止使用 Content-Range。

（第 4.3.4 节）

CONNECT 方法的定义已从 [RFC2817] 移至本规范。 （第 4.3.6 节）

OPTIONS 和 TRACE 请求方法已被定义为安全的。 （第 4.3.7 节和第 4.3.8 节）

由于广泛部署的损坏实现,Expect 标头字段的扩展机制已被删除。 （第 5.1.1 节）

Max-Forwards 头域被限制在 OPTIONS 和 TRACE 方法中;以前,扩展方法也可以使用它。 （第 5.1.2 节）

当没有适用的引用 URI 时，“about:blank”URI 已被建议作为 Referer 标头字段的值,这将这种情况与 Referer 字段未发送或已被删除的其他情况区分开来。 （第 5.5.2 节）

以下状态代码现在是可缓存的（也就是说,它们可以在没有显式新鲜度信息的情况下被缓存存储和重用）:204、404、405、414、501。（第 6 节）

201（已创建)状态描述已更改,以允许创建多个资源的可能性。

（第 6.3.2 节）

203（非权威信息)的定义已经扩展到包括负载转换的情况。

（第 6.3.4 节）

安全自动重定向的请求方法集不再关闭;用户代理能够根据请求方法语义做出决定。重定向状态码 301、302 和 307 不再对响应负载和用户交互有规范要求。 （第 6.4 节）

状态代码 301 和 302 已更改为允许用户代理将方法从 POST 重写为 GET。 （第 6.4.2 和 6.4.3 节）

303（查看其他)状态代码的描述已更改为允许在提供明确的新鲜度信息时对其进行缓存,并且已为 GET 的 303 响应添加了特定定义。

（第 6.4.4 节）

由于代理的带内配置方面的安全问题,305（使用代理)状态代码已被弃用。 （第 6.4.5 节）

400（错误请求)状态代码已放宽,因此它不仅限于语法错误。 （第 6.5.1 节）

426（需要升级)状态代码已从 [RFC2817] 中合并。 （第 6.5.15 节）

HTTP-date 和 Date 标头字段的要求目标已减少到那些生成日期的系统,而不是所有发送日期的系统。 （第 7.1.1 节）

Location 标头字段的语法已更改为允许所有 URI 引用,包括相对引用和片段,以及关于何时不适合使用片段的一些说明。 （第 7.1.2 节）

Allow 已被重新归类为响应标头字段,删除了在 PUT 请求中指定它的选项。放宽了对Allow内容的要求;相应地,客户不需要总是相信它的价值。 （第 7.4.1 节）

已定义方法注册表。 （第 8.1 节）

本规范重新定义了 Status Code Registry;以前,它是在 [RFC2817] 的第 7.1 节中定义的。

（第 8.2 节）

内容编码的注册已更改为需要 IETF 审查。 （第 8.4 节）

Content-Disposition 标头字段已被删除,因为它现在由 [RFC6266] 定义。

Content-MD5 标头字段已被删除,因为它在部分响应方面的实现不一致。

附录 C. 导入的 ABNF

以下核心规则通过引用包含在 [RFC5234] 的附录 B.1 中定义:ALPHA（字母）、CR（回车）、CRLF（CR LF）、CTL（控件）、DIGIT（十进制 0-9）、DQUOTE（双引号）、HEXDIG（十六进制 0-9/AF/af）、HTAB（水平制表符）、LF（换行）、OCTET（任意 8 位数据序列）、SP（空格)和 VCHAR (任何可见的 US-ASCII 字符）。

[RFC7230] 中定义了以下规则：

BWS = <BWS,请参阅 [RFC7230],第 3.2.3 节> = <OWS,请参阅 [RFC7230]，第 3.2.3 节> = <RWS,请参阅 [RFC7230],第 3.2.3 节> RWS URI-absolute-URI = <absolute-URI,参见 [RFC7230],Section 2.7> comment = <comment,参见[RFC7230],Section 3.2.6> field-name = <comment,参见[RFC7230],第 3.2 节> partial-URI = <partial-URI，参见 [RFC7230],第 2.7 节>

quoted-string = <quoted-string, see [RFC7230], Section 3.2.6> 标记
= <令牌,参见 [RFC7230],第 3.2.6 节>

附录 D. 收集的 ABNF

在下面收集的 ABNF 中,列表规则根据 [RFC7230] 的第 1.2 节进行了扩展。

接受= [(“，” / (媒体范围[接受参数])) *(OWS “，” [
OWS（媒体范围[接受参数]）]）]
Accept-Charset = *(“，” OWS) ((字符集 / “*”) [重量]) *(
“，” [OWS ((字符集 / “*”) [权重]]))
Accept-Encoding = [(“，” / (编码 [权重] (编码 [权重]))) *(OWS “，” [OWS
)]]
接受语言 = *(“，” OWS) (语言范围 [权重]) *(
“，”[OWS（语言范围[权重]]）（“，”/方法)*（OWS“，”[OWS方法] ）
允许 = [ )]

BWS = <BWS,参见 [RFC7230],第 3.2.3 节>

内容编码 = *(“，” OWS) 内容编码 *(OWS “，” [OWS 内容编码]
)
内容语言 = *(“，” OWS) 语言标签 *(OWS “，” [OWS 语言标签]
)
内容位置 = 绝对 URI / 部分 URI
内容类型 = 媒体类型

日期 = HTTP 日期

期望=“100-继续”

来自=邮箱

GMT = %x47.4D.54 ;格林威治标准时间

HTTP-date = IMF-fixdate / obs-date

IMF-fixdate = day-name “，” SP date1 SP time-of-day SP GMT

位置 = URI 参考

最大转发 = 1*DIGIT

OWS = <OWS,参见 [RFC7230],第 3.2.3 节>

RWS = <RWS,参见 [RFC7230],第 3.2.3 节> Referer = absolute-URI / partial-URI
Retry-After = HTTP-date / delay-seconds

服务器 = 产品 *(RWS (产品 / 评论))

URI-reference = <URI-reference, 参见 [RFC7230], Section 2.7> User-Agent = product *(RWS (product / comment))

变化 =)) * / (*(, OWS) 字段名 *(OWS , [OWS 字段名]

absolute-URI = <absolute-URI, see [RFC7230], Section 2.7> accept-ext = OWS ; OWS token [= (token / quoted-string) accept-params = weight *accept-ext asctime-date = day-name SP date3 SP time-of-day SP year]

charset = token codings = content-coding / “identity” / “*” comment = <comment, see [RFC7230], Section 3.2.6> content-coding = token

date1 = day SP month SP year date2 = day - month - 2DIGIT date3 = month SP (2DIGIT / (SP DIGIT) day = 2DIGIT day-name = %x4D.6F.6E ; Mon)

/ %x54.75.65 ;周二 / %x57.65.64 ;周三 / %x54.68.75 ;周四 / %x46.72.69 ;周五 / %x53.61.74 ;周六 / %x53.75.6E ; 星期日名称-l = %x4D.6F.6E.64.61.79 ; 星期一 / %x54.75.65.73.64.61.79 ;周二 / %x57.65.64.6E.65.73.64.61.79 ; 星期三 / %x54.68.75.72.73.64.61.79 ;星期四 / %x46.72.69.64.61.79 ;星期五 / %x53.61.74.75.72.64.61.79 ;星期六 / %x53.75.6E.64.61.79 ;周日延迟秒数 = 1*DIGIT

field-name = <注释,参见 [RFC7230],第 3.2 节>

小时 = 2DIGIT

language-range = <语言范围,参见 [RFC4647],第 2.1 节> language-tag = <语言标签,参见 [RFC5646],第 2.1 节>

mailbox = <mailbox, see [RFC5322], Section 3.4> media-range = (*/* / (type /*) / (type / 子类型)) *(“;” OWS参数)

media-type = type / subtype *(OWS ; OWS parameter) method = token minute = 2DIGIT month = %x4A.61.6E ;一月 / %x46.65.62 ;二月 / %x4D.61.72 ;三月 / %x41.70.72 ;四月 / %x4D.61.79 ;五月 / %x4A.75.6E ; 六月 / %x4A.75.6C ;七月 / %x41.75.67 ;八月 / %x53.65.70 ;九月 / %x4F.63.74 ;八月 / %x4E.6F.76 ;十一月 / %x44.65.63 ;十二月

obs-date = rfc850-date / asctime-date

parameter = token = (token / quoted-string) partial-URI = <partial-URI, see [RFC7230], Section 2.7> product = token [/ product-version] product-version = token quoted-string = <quoted-string, see [RFC7230], Section 3.2.6> qvalue = (0 [. *3DIGIT]) / (1 [. *3 0]

)

rfc850-date = day-name-l , SP date2 SP time-of-day SP GMT

第二 = 2DIGIT 子类型 = 令牌

time-of-day = hour : minute : second token = <token, see [RFC7230], Section 3.2.6> 类型 = token

权重 = OWS “;” OWS q= qvalue

年份 = 4DIGIT

指数

H

我

大号

米

欧

P

R

小号

作者地址

Roy T. Fielding（编辑）
Adobe Systems Incorporated 345 Park Ave San
Jose, CA 95110 USA

电子邮件：fielding@gbiv.com URI：http://
roy.gbiv.com/

Julian F. Reschke（编辑）greenbytes GmbH
Hafenweg 16 Muenster, NW 48155 德国

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