组织：中国互动出版网（http://www.china-pub.com/）
RFC 文档中文翻译计划（http://www.china-pub.com/compters/emook/aboutemook.htm）
E-mail：ouyang@china-pub.com
译者：黄晓东（黄晓东　xdhuang@eyou.com）
译文发布时间：2001-7-14

Network Working Group
Request for Comments: 1945
Category: Informational

T. Berners-Lee
MIT/LCS
R. Fielding
UC Irvine
H. Frystyk
MIT/LCS
May 1996

超文本传输协议 -- HTTP/1.0

（Hyptertext Transfer Protocol - HTTP/1.0）

关于下段备忘（Status of This Memo）

本段文字为 Internet 团体提供信息，并没有以任何方式指定 Internet 标准。本段文字没有分发限制。

IESG 提示（IESG Note）：

IESG 已在关注此协议，并期待该文档能尽快被标准跟踪文档所替代。

摘要（Abstract）

HTTP（Hypertext Transfer Protocol）是应用级协议，它适应了分布式超媒体协作系统对灵活性及速度的要求。它是一个一般的、无状态的、基于对象的协议，通过对其请求方法（request methods）进行扩展，可以被用于多种用途，比如命名服务器（name server）及分布式对象管理系统。HTTP 的一个特性是其数据表现类型允许系统的构建不再依赖于要传输的数据。

HTTP 自从 1990 年就在 WWW 上被广泛使用。该规范反映了“HTTP/1.0”的普通用法。

1. 介绍（Introduction）

1.1 目的（Purpose）

HTTP（Hypertext Transfer Protocol）是应用级协议，它适应了分布式超媒体协作系统对灵活性及速度的要求。它是一个一般的、无状态的、基于对象的协议，通过对其请求方法（request methods）进行扩展，可以被用于多种用途，比如命名服务器（name server）及分布式对象管理系统。HTTP 的一个特性是其数据表现类型允许系统的构建不再依赖于要传输的数据。

HTTP 自从 1990 年就在 WWW 上被广泛使用。该规范反映了“HTTP/1.0”的普通用法。

该规范描述了在大多数 HTTP/1.0 客户机及服务器上看起来已经实现的特性。规范将被分成两个部分：HTTP 特性的实现是本文档的主要内容，而其它不太通行的实现将被列在附录 D 中。

实用的信息系统需要更多的功能，而不仅仅是数据的获取，包括搜索、前端更新及注解。HTTP 允许使用开放的命令集来表示请求的目的，它使用基于 URI[2]（Uniform Resource Identifier），即统一资源标识的规则来定位（URL[4]）或命名（URN[16]）方法所用到的资源。HTTP 使用与邮件（Internet Mail [7]）和 MIME（Multipurpose Internet Mail Extensions [5]）相似的格式来传递消息。

HTTP 也作为用户代理、代理服务器/网关与其它 Internet 协议进行通讯的一般协议，这些协议是 SMTP [12]、NNTP [11]、FTP [14]、Gopher [1]、and WAIS [8]等。HTTP 允许不同的应用可以进行基本的超媒体资源访问，并简化用户代理的实现。

1.2 术语（Terminology）

本规范用了许多与参与方、对象及 HTTP 通讯相关的术语，如下：

连接（connection）

两个应用程序以通讯为目的在传输层建立虚拟电路

消息（message）

HTTP 通讯的基本单元，在连接中传输的结构化的、有顺序的字节（其含义在第四节中定义）

请求（request）

HTTP 的请求消息（在第五节定义）

回应（response）

HTTP 的回应消息（在第六节定义）

资源（resource）

网络上可以用 URI 来标识的数据对象或服务（见 3.2 节）

实体（entity）

可被附在请求或回应消息中的特殊的表示法、数据资源的表示、服务资源的回应等，由实体标题（entity header）或实体主体（entity body）内容形式存在的元信息组成

客户端（client）

指以发出请求为目的而建立连接的应用程序

用户代理（user agent）

指初始化请求的客户端，如浏览器、编辑器、蜘蛛（web 爬行机器人）或其它终端用户工具

服务器（server）

指接受连接，并通过发送回应来响应服务请求的应用程序

原始服务器（origin server）

存放资源或产生资源的服务器

代理（proxy）

同时扮演服务器及客户端角色的中间程序，用来为其它客户产生请求。请求经过变换，被传递到最终的目的服务器，在代理程序内部，请求或被处理，或被传递。代理必须在消息转发前对消息进行解释，而且如有必要还得重写消息。代理通常被用作经过防火墙的客户端出口，用以辅助处理用户代理所没实现的请求

网关（gateway）

服务器之间的服务器。与代理不同，网关接受请求就好象它就是被请求资源所在的原始服务器，发出请求的客户端可能并没有意识到它在与网关进行通讯。网关是网络防火墙服务器端的门户。对非 HTTP 系统资源进行访问时，网关做为中间的协议翻译者

隧道（tunnel）

隧道就好象连接两端看不见的中继器。处于激活状态时，它虽然是由 HTTP 请求来初始化的，但它并不参与 HTTP 通讯。当需要中继连接的两端关闭后，隧道也自然终止。在入口有需求及中间程序无法或不该解释要中继的通讯时，通常要用到隧道技术

缓存（cache）

指程序本地存储的回应消息和用来控制消息存储、重获、删除的子系统。缓存回应的目的是为减少请求回应时间，以及未来一段时间对网络带宽的消耗。任何客户端及服务端都可以包含缓存。服务器在以隧道方式工作时不能使用缓存。

任何指定的程序都有能力同时做为客户端和服务器。我们在使用这个概念时，不是看程序功能上是否能实现客户及服务器，而是看程序在特定连接时段上扮演何种角色（客户或服务器）。同样，任何服务器可以扮演原始服务器、代理、网关、隧道等角色，行为的切换取决于每次请求的内容。

1.3 概述（Overall Operation）

HTTP 协议是基于请求/回应机制的。客户端与服务器端建立连接后，以请求方法、URI、协议版本等方式向服务器端发出请求，该请求可跟随包含请求修饰符、客户信息、及可能的请求体（body）内容的 MIME 类型消息。

服务器端通过状态队列（status line）来回应，内容包括消息的协议版本、成功或错误代码，也跟随着包含服务器信息、实体元信息及实体内容的 MIME 类型消息。

绝大多数 HTTP 通讯由用户代理进行初始化，并通过它来组装请求以获取存储在一些原始服务器上的资源。在最简单的情况下，通过用户代理（UA）与原始服务器（O）之间一个简单的连接（v）就可以完成。

```
          request chain ------------------------>
UA --------------------v---------------------- O
          <------------------------ response chain
```

更复杂的情况是当请求/回应链之间存在一个或更多中间环节。总体看来，有三种中间环节：代理（proxy）、网关（gateway）、隧道（tunnel）。代理（proxy）是向前推送的代理人（agent），它以绝对形式接收 URI 请求，重写全部或部分消息，并将经过改写的请求继续向 URI 中指定的服务器处推送。网关是接收代理，它处于服务器层之上，在必要时候，它用服务器可识别的协议来传递请求。隧道不改变消息，它只是连接两端的中继点。在有中间层（如防火墙）或中间层无法解析消息内容的情况下，需要靠隧道技术来帮助通讯穿越中间层。

```
           request chain ---------------------------------------->
UA -----v----- A -----v----- B -----v----- C -----v----- O
           <---------------------------------------- response chain
```

上面的图形表示在用户代理和原始服务器之间有三个中间层（A，B 和 C）。由图可见，请求或回应消息在整个信息链上运行需要通过四个单独的连接，它与在此之前介绍的简单情况是有区别的，而且此区别是十分重要的。因为 HTTP 通讯选项可以设置成几种情况，如只与最近的非隧道邻居连接、只与信息链末端连接、或者可与链中全部环节连接等等。虽然上面的图是线性的，而实际上每个参与环节都在同时与多方进行通讯活动。例如，B 在接受除 A 之外其它客户端请求的同时，向除 C 之外的其它服务器推送请求，在这个时刻，它可能接受到 A 的请求，并给予处理。

参与通讯的任何一方如果没有以隧道方式进行工作，必须要借助内部缓存机制来处理请求，如果链上某个参与方碰巧缓存了某个请求的回应，那就相应于缩短了请求/回应链。下面的图例演示了当 B 缓存了从 O 经由 C 过来的回应信息，而 UA 和 A 没缓存的情况：

```
          request chain ---------->
UA -----v----- A -----v----- B - - - - - - C - - - - - - - O
          <--------- response chain
```

并非所有的回应都可以被缓存，某些请求所包含的修饰符中可能对缓存行为进行了特别指明。一些基于 HTTP/1.0 的应用使用了启发式的方法来描述哪些回应可被缓存，而哪些则不可以，但遗憾的是，这些规则并没有形成标准。

在 Internet 上，HTTP 通讯往往基于 TCP/IP 的连接方式。缺省的端口是 TCP 80[15]口，但也可以使用其它端口。并不排除基于 Ineternet 上的其它协议或网络协议的 HTTP 实现方式，HTTP 只是假定传输是可靠的，因而任何能提供这种保证的协议都可以被使用。至于 HTTP/1.0 请求和回应在数据传输过程中的数据结构问题，不在本文讨论范围之内。

实验室应用除外，当前的做法是客户端在每次请求之前建立连接，而服务器端在发送回

应后关闭此连接。不管客户端还是服务器端都应注意处理突发的连接中断，因为双方都有可能因为用户操作、自动超时、程序失败等原因关闭与对方的连接。在这种情况下，不管请求处于什么样的状态，如单方或双方同时关闭连接，都会导致当前的请求被终止。

1.4 HTTP and MIME

HTTP/1.0 使用了多种结构来定义 MIME，详见 RFC1521[5]。附录 C 描述了 Internet 承认的 Internet 介质类型与 mail 介质类型的不同工作方式，并给出二者区别的基本解释。

2. 标志转换及通用语法（Notational Conventions and Generic Grammar）

2.1 补充反馈方式（Augmented BNF）

与 RFC822[7]很类似，本文对所有机制的说明都是以散文和补充反馈的方式来描述的。对于实现者来说，要想理解这些约定，必须对这些符号很熟悉。补充反馈方式(augmented BNF)包括下面的结构：

要解释的名词＝名词解释（name = definition）

规则的名字（name）就是它本身（不带任何尖括号，“<”，“>”），后面跟个等号＝，然后就是该规则的定义。如果规则需要用多个行来描述，利用空格进行缩进格式排版。某些基本的规则使用大写，如 SP, LWS, HT, CRLF, DIGIT, ALPHA,等等。定义中还可以使用尖括号来帮助理解规则名的使用。

字面意思（″literal″）

文字的字面意思放在引号中间，除非特别指定，该段文字是大小写敏感的。

规则 1｜规则 2（rule1 ｜ rule2）

“｜”表示其分隔的元素是可选的，比如，“是｜否”要选择‘是’或‘否’。

（规则 1　规则 2）((rule1 rule2))

在圆括号中的元素表明必选其一。如（元素 1（元素 2｜元素 3）元素 4）可表明两种意思，即“元素 1　元素 2　元素 4”和“元素 1　元素 3　元素 4”

*规则（*rule）

在元素前加星号“*”表示循环，其完整形式是“<n>*<m>元素”，表明元素最少产生<n>次，最多<m>次。缺省值是 0 到无限，例如，“1*元素”意思是至少有一个，而“1*2 元素”表明允许有 1 个或 2 个。

[规则]([rule])

方括号内是可选元素。如“[元素 1　元素 2]”与“*1（元素 1　元素 2)”是一回事。

N　规则（N rule）

表明循环的次数：“<n>（元素）”就是“<n>*<n>（元素）”，也就是精确指出<n>取值。因而，2DIGIT 就是 2 位数字，3ALPHA 就是由三个字母组成字符串。

＃规则（#rule）

“#”与“*”类似，用于定义元素列表。

完整形式是“<n>#<m>元素”表示至少有<n>个至多有<m>个元素，中间用“,”或任意数

量的空格(LWS-linear whitespace)来分隔，这将使列表非常方便，如“(*LWS 元素 *(*LWS ″,″ *LWS 元素))”就等同于“1#元素”。

空元素在结构中可被任意使用，但不参与元素个数的计数。也就是说，“(元素 1),,(元素 2)”仅表示 2 个元素。但在结构中，应至少有一个非空的元素存在。缺省值是 0 到无限，即“#(元素)”表示可取任何数值，包括 0；而“1#元素”表示至少有 1 个；而“1#2 元素”表示有 1 个或 2 个。

; 注释 (; comment)

分号后面是注释，仅在单行使用。

隐含*LWS (implied *LWS)

本文的语法描述是基于单词的。除非另有指定，线性空格(LWS)可以两个邻近符号或分隔符(tspecials)之间任意使用，而不会对整句的意思造成影响。在两个符号之间必须有至少一个分隔符，因为它们也要做为单独的符号来解释。实际上，应用程序在产生 HTTP 结构时，应当试图遵照“通常方式”，因为现在的确有些实现方式在通常方式下无法正常工作。

2.2 基本规则(Basic Rules)

下面的规则将用于本文后面对结构基本解析。

US-ASCII 编码字符集定义[17]。

```
OCTET       = <8-bit 的顺序数据，即 bytes>
CHAR        = < US-ASCII 字符 (取值为 0 - 127 的 OCTET) >
UPALPHA     = < US-ASCII 大写字符″A″到″Z″>
LOALPHA     = <US-ASCII 小写字符″a″到″z″>

      ALPHA         = UPALPHA | LOALPHA
      DIGIT         = < US-ASCII 数字″0″到″9″>
      CTL           = < US-ASCII 控制字符 (取值 0 到 31 的 octet ) 和 DEL (127) >
      CR            = <US-ASCII CR, 回车符 carriage return (13) >
      LF            = <US-ASCII LF, 换行符 linefeed (10) >
      SP            = <US-ASCII SP, 空格 space (32) >
      HT            = <US-ASCII HT, 水平制表符 horizontal-tab (9) >
      <″>           = <US-ASCII 双引号标记 double-quote mark (34) >
```

HTTP/1.0 规定，除实体主体(Entity-Body，见附录 B 容错应用)外，所有协议元素的行结束标志都是 CRLF(按字节顺序)。在实体主体内部的行结束标志定义及其对应的介质类型定义，见 3.6 节的描述。

```
      CRLF    = CR LF
```

HTTP/1.0 的头可以折成许多行，只要每个后续行以空格或水平制表符开头。所有的线性空格(LWS)，同空格(SP)有相同的语义。

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

实际上，有些应用并没有考虑处理这样多行的头，所以从兼容性上考虑，HTTP/1.0 应用最好不要产生折成多行的头。

TEXT 规则只是用于描述消息解释器不关心的域的内容及其取值。文本内容可包含不同于US-ASCII 的字符。

TEXT = <除了控制字符（CTLs）之外的任何 OCTET，包括 LWS >

在标题域中的收件人域如包含US-ASCII 字符集以外的字符,这些字符将按照 ISO-8859-1 标准来解释。

十六进制数字型字符在几个协议元素中到。

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

许多 HTTP/1.0 头域的内容由被 LWS 分隔的单词或特殊字符组成,这些特殊字符必须置于引号中间的字符串内，作为参数值使用。

```
Word    = 符号（token）| 被引号引起来的字符串
token   = 1*<除控制字符（CTLs）或 tspecials 之外的任意字符>
tspecials     = "(" | ")" | "<" | ">" | "@"
              | "," | ";" | ":" | "\" | <">
              | "/" | "[" | "]" | "?" | "="
              | "{" | "}" | SP | HT
```

在 HTTP 头域中可用括号表示注释文字。注释只允许写在包含注释的域，做为域值定义的一部分。在除此之外其它域中，括号将被视为域值。

```
comment     = "(" *( ctext | comment ) ")"
ctext       = <除圆括号外的任何 TEXT>
```

被双引号圈中的文本字符串将被视为一个单词。

```
quoted-string= ( <"> *(qdtext) <"> )
qdtext      = <除了双引号及控制字符之外的任何字符，包括 LWS >
```

在 HTTP/1.0 中不允许使用后斜线“\”来引用单字符。

3. 协议参数（Protocol Parameters）

3.1 HTTP 版本（HTTP Version）

HTTP 采用主从（<major>.<minor>）数字形式来表示版本。协议的版本政策倾向于让发送方表明其消息的格式及功能，而不仅仅为了获得通讯的特性，这样做的目的是为了与更高版本的 HTTP 实现通讯。只增加扩展域的值或增加了不影响通讯行为的消息组件都不会导致版本数据的变化。当协议消息的主要解析算法没变，而消息语法及发送方的隐含功能增加了，将会导致从版本号（<minor>）增加；当协议中消息的格式变化了，主版本号（<major>）也将发生改变。

HTTP 消息的版本由消息第一行中的 HTTP 版本域来表示。如果消息中的协议版本没有指定，接收方必须假定它是符合 HTTP/0.9 的简单标准。

HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT

注意，主从版本应当被看作单独的整数，因为它们都有可能增加，从而超过一位整数。因而，HTTP/2.4 比 HTTP/2.13 版本低，而 HTTP/2.13 又比 HTTP/12.3 版本低。版本号前面的 0 将被接收方忽略，而在发送方处也不应产生。

本文档定义了 HTTP 协议的 0.9 及 1.0 版本。发送完整请求（Full-Request）及完整回应（Full-Response）消息的应用必须指明 HTTP 的版本为“HTTP/1.0”。

HTTP/1.0 服务器必须：

- o 识别 HTTP/0.9 及 HTTP/1.0 请求命令中的请求队列（Request-Line）的格式。
- o 理解任何 HTTP/0.9 及 HTTP/1.0 中的合法请求格式。
- o 使用与客户端相同版本的协议进行消息回应。

HTTP/1.0 客户端必须：

- o 识别 HTTP/1.0 回应中状态队列（Status-Line）的格式。
- o 理解 HTTP/0.9 及 HTTP/1.0 中合法回应的格式。

当代理及网关收到与其自身版本不同的 HTTP 请求时，必须小心处理请求的推送，因为协议版本表明发送方的能力，代理或网关不应发出高于自身版本的消息。如果收到高版本的请求，代理或网关必须降低该请求的版本，并回应一个错误。而低版本的请求也应在被推送前升级。代理或网关回应请求时必须遵照前面列出的规定。

3.2 统一资源标识（Uniform Resource Identifiers）

URI 有许多名字，如 WWW 地址、通用文件标识（Universal Document Identifiers）、通用资源标识（Universal Resource Identifiers [2]），以及最终的统一资源定位符（Uniform Resource Locators (URL) [4]）和统一资源名（URN）。在涉及 HTTP 以前，URI 用简单格式的字符串描述－名字、位置、或其它特性，如网络资源。

3.2.1 一般语法（General Syntax）

在 HTTP 中 URI 可以用绝对形式表示，也可用相对于某一基本 URI[9]的形式表示，具体取决于它们的使用方式。这两种形式的不同在于绝对 URI 总是以方法名称后跟“:”开头。

```
URI          = ( absoluteURI | relativeURI ) [ "#" fragment ]
absoluteURI  = scheme ":" *( uchar | reserved )
relativeURI  = net_path | abs_path | rel_path
net_path       = "//" net_loc [ abs_path ]
abs_path       = "/" rel_path
rel_path       = [ path ] [ ";" params ] [ "?" query ]
path           = fsegment *( "/" segment )
fsegment       = 1*pchar
segment        = *pchar
params         = param *( ";" param )
param          = *( pchar | "/" )
scheme         = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc        = *( pchar | ";" | "?" )
query          = *( uchar | reserved )
fragment       = *( uchar | reserved )
pchar          = uchar | ":" | "@" | "&" | "=" | "+"
uchar          = unreserved | escape
unreserved     = ALPHA | DIGIT | safe | extra | national
escape         = "%" HEX HEX
reserved       = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra          = "!" | "*" | "'" | "(" | ")" | ","
safe           = "$" | "-" | "_" | "."
unsafe         = CTL | SP | <"> | "#" | "%" | "<" | ">"
national       = <any OCTET excluding ALPHA, DIGIT,
                  reserved, extra, safe, and unsafe>
```

权威的 URL 语法及语义信息请参见 RFC1738[4]和 RFC1808[9]。上面所提到的 BNF 中包括了合法 URL 中不允许出现的符号（RFC1738），因为 HTTP 服务器并没有限制为只能用非保留字符集中的字符来表示地址路径，而且 HTTP 代理也可能接收到 RFC1738 没有定义的 URI 请求。

3.2.2 http URL

“http”表示要通过 HTTP 协议来定位网络资源。本节定义了 HTTP URL 的语法和语义。

```
http_URL      = "http:" "//" host [ ":" port ] [ abs_path ]
host          = <合法的 Internet 主机域名或 IP 地址（用十进制数及点组成），
见 RFC1123，2.1 节定义>
port          = *DIGIT
```

如是端口为空或没指定，则缺省为 80 端口。对于绝对路径的 URI 来说，拥有被请求的资源的服务器主机通过侦听该端口的 TCP 连接来接收该 URI 请求。如果 URL 中没有给出绝对路

径，要作为请求 URI（参见 5.1.2 节）使用，必须以“/”形式给出。

注意：虽然 HTTP 协议独立于传输层协议，http URL 只是标识资源的 TCP 位置，而对于非 TCP 资源来说，必须用其它的 URI 形式来标识。

规范的 HTTP URL 形式可通过将主机中的大写字符转换成小写（主机名是大小写敏感的）来获得。如果端口是 80，去掉冒号及端口号，并将空路径替换成“/”。

3.3 Date/Time 格式（Date/Time Formats）

出于历史原因，HTTP/1.0 应用允许三种格式来表示时间戳：

```
Sun, 06 Nov 1994 08:49:37 GMT    ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT   ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994         ; ANSI C's asctime() format
```

第一种格式是首选的 Internet 标准格式，表示方法长度固定（RFC1123[6]）。第二种格式在普通情况下使用，但是它是基于已经废弃的 RFC850[10]中的日期格式，而且年不是用四位数字表示的。HTTP/1.0 客户端及服务器端在解析日期时可识别全部三种格式，但是它们不可以产生第三种时间格式(asctime) 。

注意：对于接收到由非 HTTP 应用产生的日期数据时，提倡对接收到的日期值进行填充。这样做是因为，在某些时候，代理或网关可能通过 SMTP 或 NNTP 来获取或发送消息。

所有的 HTTP/1.0 date/timp 时间戳必须用世界时间（Universal Time，UT），即格林威治时间来表示(Greenwich Mean Time,GMT),没有任何修改的余地。前面的两种格式用了“GMT”表示时区，在读 ASC 表示的时间时，也应假定是这个时区。

```
HTTP-date    = rfc1123-date | rfc850-date | asctime-date
rfc1123-date= wkday "," SP date1 SP time SP "GMT"
rfc850-date = weekday "," SP date2 SP time SP "GMT"
asctime-date    = wkday SP date3 SP time SP 4DIGIT
date1        = 2DIGIT SP month SP 4DIGIT
               ; day month year (e.g., 02 Jun 1982)
date2        = 2DIGIT "-" month "-" 2DIGIT
               ; day-month-year (e.g., 02-Jun-82)
date3        = month SP ( 2DIGIT | ( SP 1DIGIT ))
               ; month day (e.g., Jun  2)
time             = 2DIGIT ":" 2DIGIT ":" 2DIGIT
               ; 00:00:00 - 23:59:59
wkday        = "Mon" | "Tue" | "Wed"
               | "Thu" | "Fri" | "Sat" | "Sun"
weekday      = "Monday" | "Tuesday" | "Wednesday"
               | "Thursday" | "Friday" | "Saturday" | "Sunday"
month        = "Jan" | "Feb" | "Mar" | "Apr"
               | "May" | "Jun" | "Jul" | "Aug"
               | "Sep" | "Oct" | "Nov" | "Dec"
```

注意：HTTP 要求只能在协议流中使用 data/time 时间戳格式，不要求客户端及服务器端在用户描述、请求登录等情况下使用这类格式。

3.4 字符集（Character Sets）

HTTP 所使用的字符集定义和描述 MIME 时用的相同：本文档用字符集（character set）来指明利用一个或多个表将顺序字节转换成顺序字符的方法。注意，不需要其它方向的无条件转换，因为不是所有的字符都可以用给定字符集来表示，同时，一个字符集也可能提供一种以上的字节顺序来表示一种特殊的字符。这种定义倾向于允许不同类型的字符编码通过简单的单表映射来实现，如，从表 US-ASCII 切换到复杂表如 ISO2202。实际上，与 MIME 字符集名相关的定义必须完整指定从字节到字符的映射，特别是不允许通过利用外部配置信息来确定精确的映射。

注意：术语字符集（character set）归于字符编码（character encoding)。事实上，由于 HTTP 与 MIME 共同使用同样的注册，所以其术语也应一致。

HTTP 字符集由大小写敏感的符号组成。全部的符号定义参见 IANA 字符集注册[15]。因为注册处不会为每个字符集单独定义一套符号，所以我们在这看到的字符集名大多是与 HTTP 实体使用相关的。这些在 RFC 1521 [5] 中注册的字符集，即 US-ASCII [17] 及 ISO-8859 [18] 字符集，还有一些其它字符集被强烈建议在 MIME 字符集参数内部使用。

```
charset = "US-ASCII"
        | "ISO-8859-1" | "ISO-8859-2" | "ISO-8859-3"
        | "ISO-8859-4" | "ISO-8859-5" | "ISO-8859-6"
        | "ISO-8859-7" | "ISO-8859-8" | "ISO-8859-9"
        | "ISO-2022-JP" | "ISO-2022-JP-2" | "ISO-2022-KR"
        | "UNICODE-1-1" | "UNICODE-1-1-UTF-7" | "UNICODE-1-1-UTF-8"
        | token
```

虽然 HTTP 允许使用专用符号做为字符集值，但是任何在 IANA 字符集注册表[15]中有预定义值的符号都必须表明其所属的字符集。应用应当将其对字符集的使用限制在 IANA 注册表规定的范围之内。

实体主体的字符集如果属于 US-ASCII 或 ISO-8859-1 字符集，则勿需标记，否则，应当用主体字符编码方式中的最基本命名来标记。

3.5 内容译码（Content Codings）

内容译码值用于指示对资源进行的编码转换。内容译码主要用于将经过压缩、加密等操作的文件进行还原，使其保持其原来的介质类型。典型情况下，经过编码保存的资源只能经过解码或类似的操作才能还原。

```
content-coding = "x-gzip" | "x-compress" | token
```

注意：出于对未来兼容性的考虑，HTTP/1.0 应用应将"gzip" 和"compress" 分别与"x-gzip"和"x-compress"对应起来。

所有的内容译码值都是大小写敏感的。HTTP/1.0 在内容编码（10.3 节）头域中使用内容译码值。虽然该值描述的是内容译码，但更为重要的是，它指明了应当用什么机制来进行解码。注意，单独的程序可能有能力实现对多种格式编码的解码。

在这段文字中，提到了两个值：

x-gzip 文件压缩程序"gzip" (GNU zip，由 Jean-loup Gailly 开发)的编码格式。该格式属于典型的带有 32 位 CRC 校验的 Lempel-Ziv 译码 (LZ77)。

x-compress 文件压缩程序"compress"的编码格式，该格式适用于 LZW（Lempel-Ziv-Welch）译码。

注意：用程序名来标识编码格式的做法不是很理想，在将来可能不会继续这样做。现在之所以这样做是出于历史的原因，并非良好的设计。

3.6 介质类型（Media Types）

HTTP 在 Content-Type header 域（10.5 节）中使用 Internet 介质类型[13]，用以提供开放的可扩展的数据类型。

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

参数可参照属性/值对的方式，用类型/子类型的格式来写。

```
Parameter       = attribute "=" value
Attribute       = token
Value           = token | quoted-string
```

其中，类型、子类型、参数属性名是大小写敏感的。而参数值不一定是大小写敏感的，这得看参数名的语法而定。在类型和子类型、属性名和属性值之间不能有 LWS（空格）。当接收到不能识别的介质类型的参数时，用户代理应当忽略它们。

一些老的 HTTP 应用不能识别介质类型参数，所以 HTTP/1.0 的应用程序只能在定义消息内容时使用介质参数。

介质参数（Media-type）值用 Internet 授权分配数字（Internet Assigned Number Authority ，IANA[15]）注册。介质类型注册过程请参见 RFC1590[13]。不鼓励使用未注册的介质类型。

3.6.1 标准及文本缺省（Canonicalization and Text Defaults）

Internet 介质类型是用规范形式注册的。一般来说，在通过 HTTP 协议传输实体主体（Entity-Body）之前，必须先将其表示成适当的规范格式。如果主体用使用了一种 Content-Encoding 进行编码，下面的数据在编码前必须转换成规范形式：

"text"类型的介质子类型在规范形式中使用 CRLF 做为文本行中断。实际上，为和实体主体（Entity body）内的使用方式保持一致，HTTP 允许传输纯以 CR 或 LF 单独表示行中断的文本介质。HTTP 应用程序必须将其通过 HTTP 方式接收到的文本介质中的 CRLF、CR、LF 看做是行中断符。

另外，如果文本介质的字符集没有使用字节 13 和 10 做为 CR 和 LF，象一些多字节字符集，HTTP 允许使用该字符集指定的任何顺序的字节替代 CR 和 LF 做为行中断，这种行中断的灵活运用方式仅可于实体主体（Entity-Body）中。一个纯 CR 或 LF 不应在任何 HTTP 控制结构（如标题域-header field 和多块分界线-multipart boundaries）中替代 CRLF。

参数"charset"在定义数据的字符集（3.4 节）时，与一些介质类型一起使用。当发送方没有显式给出字符参数时，HTTP 在接收时将"text"的介质子类型定义为缺省值"ISO-8859-1"。"ISO-8859-1"字符集或其子集以外的数据必须要标记其相应的字符集值，这

样可以保证接收方能正确地对其进行解析。

注意：许多当前 HTTP 服务器提供的数据使用"ISO-8859-1"以外的其它字符集，而且也没有正确的进行标记，这种工作方式限制了互操作性，建议不要采用。做为一种补救方法，一些 HTTP 用户代理提供了配置选项，使用户可以在字符集参数没指定的情况下，自行更改缺省的介质类型解释方式。

3.6.2 多部分类型（Multipart Types）

MIME 提供了多部分（"multipart"）类型的数字－－在一个单独的消息实体主体（Entity-Body）内可以封装几个实体（entities）。虽然用户代理可能需要了解每种类型，从而可以正确解释每部分主体的意图，但是在 IANA[15]的多部分类型（multipart types）注册中却找不到专为 HTTP/1.0 所指定的内容。HTTP 用户代理只得自己来做接收多部分类型的工作，其过程和行为与 MIME 用户代理是相同或相似的。HTTP 服务器不应假定 HTTP 客户端都有能力处理多部分类型。

所有的多部分类型都使用通用的语法，而且必须在介质类型值部分包括边界参数（boundary parameter）。消息的主体是其自身，做为协议元素，它必须只使用 CRLF 做为段间（body-parts）的行中断符。多段主体（Multipart body-parts）中可能包括对各段有重要意义的 HTTP 标题域。

3.7 产品标识（Product Tokens）

是通讯应用程序用来标识其自身的一个简单符号，常和任意字母及版本描述一起使用。大多数产品标识也将其产品的重要组成部分的版本号一起列出，中间用空格分隔。

按惯例，在标识应用程序时，组件以其重要性顺序排列。

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

例如：

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

产品标识应当短小，因而禁止利用该域填写广告或其它无关信息。虽然任何符号字符都可能出现在产品版本中，该符号应当只用于做版本定义，也就是说，同一个产品的连续版本之间只能通过产品版本值进行区分。

4. HTTP 消息（HTTP Message）

4.1 消息类型（Message Types）

HTTP 消息由客户端到服务器的请求和由服务器到客户端的回应组成。

```
HTTP-message   = Simple-Request           ; HTTP/0.9 messages
               | Simple-Response
               | Full-Request             ; HTTP/1.0 messages
               | Full-Response
```

完整的请求（Full-Request）和完整的回应（Full-Response）都使用 RFC822[7]中实体传输部分规定的消息格式。两者的消息都可能包括标题域(headers，可选)、实体主体(entity body)。实体主体与标题间通过空行来分隔（即 CRLF 前没有内容的行）。

```
Full-Request      = Request-Line          ; Section 5.1
                  *( General-Header       ; Section 4.3
                   | Request-Header       ; Section 5.2
                   | Entity-Header )      ; Section 7.1
                  CRLF
                  [ Entity-Body ]         ; Section 7.2
Full-Response     = Status-Line           ; Section 6.1
                  *( General-Header       ; Section 4.3
                   | Response-Header      ; Section 6.2
                   | Entity-Header )      ; Section 7.1
                  CRLF
                  [ Entity-Body ]         ; Section 7.2
```

简单请求(Simple_Request)与简单回应(Simple-Response)不允许使用任何标题信息，并限制只能使用唯一的请求方法（GET）

```
Simple-Request  = "GET" SP Request-URI CRLF
Simple-Response = [ Entity-Body ]
```

不提倡使用简单方式请求格式，因为它防止了服务器在接到简单请求时对返回实体的介质类型进行验证。

4.2 消息标题（Message Headers）

HTTP 标题域，包括主标题（General-Header,4.3 节)、请求标题（Request-Header ,5.2 节)、回应标题（Response-Header ,6.2 节）及实体标题（Entity-Header,7.1 节），都遵照 RFC822-3.1 节[7]给出的通用格式定义。每个标题域由后紧跟冒号的名字，单空格（SP），字符及域值组成。域名是大小写敏感的。虽然不提倡，标题域还是可以扩展成多行使用，只要这些行以一个以上的 SP 或 HT 开头就行。

```
HTTP-header     = field-name ":" [ field-value ] CRLF
field-name      = token
```

```
field-value     = *( field-content | LWS )
field-content   = <the OCTETs making up the field-value
                  and consisting of either *TEXT or combinations
                  of token, tspecials, and quoted-string>
```

标题域接收的顺序并不重要，但良好的习惯是，先发送主标题，然后是请求标题或回应标题，最后是实体标题。

当且仅当标题域的全部域值都用逗号分隔的列表示时（即，#（值）），多个有相同域名的HTTP 标题域才可以表示在一个消息里。而且必须能在不改变消息语法的前提下，将并发的域值加到第一个值后面，之间用逗号分隔，最终能将多个标题域结合成“域名：域值”对。

4.3 普通标题域（General Header Fields）

有几种标题域是请求与回应都要使用的，但并不用于被传输的实体。这些标题只用于被传输的消息。

```
General-Header = Date        ; Section 10.6
               | Pragma      ; Section 10.12
```

普通标题域名称只有在与协议版本的变化结合起来后，才能进行可靠的扩展。实际上，新的或实验中的标题域只要能被通讯各方识别，其语法就可使用，而无法识别的标题域都将被视为实体域。

5. 请求（Request）

从客户端到服务器端的请求消息包括，消息首行中，对资源的请求方法、资源的标识符及使用的协议。考虑到局限性更大的 HTTP/0.9 的向后兼容问题，有两种合法的 HTTP 请求格式：

```
Request         = Simple-Request | Full-Request
Simple-Request  = "GET" SP Request-URI CRLF
Full-Request    = Request-Line      ; Section 5.1
                *( General-Header   ; Section 4.3
                | Request-Header    ; Section 5.2
                | Entity-Header )   ; Section 7.1
                CRLF
                [ Entity-Body ]     ; Section 7.2
```

如果 HTTP/1.0 服务器收到简单请求，它必须回应一个 HTTP/0.9 格式的简单回应。HTTP/1.0 的客户端有能力接收完整回应，但不能产生简单请求。

5.1 请求队列（Request-Line）

请求队列以一个方法符号开头，跟在请求 URI 及协议版本的后面，以 CRLF 为结尾。该元素用空格 SP 分隔。除了最后的 CRLF，不允许出现单独的 CR 或 LF 符。

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

注意,简单请求与完整请求的请求队列之间的区别在于是否有 HTTP 版本域和是否可以使用除 GET 以外的其它方法。

5.1.1 方法（Method）

方法代号指明了将要以何种方式来访问由请求 URI 指定的资源。方法是大小写敏感的。

```
Method      = "GET"                 ; Section 8.1
            |"HEAD"                 ; Section 8.2
            | "POST"                ; Section 8.3
            | extension-method
extension-method = token
```

可以访问指定资源的方法列表是可以动态变化的；如果用某种方法访问资源被拒绝，客户端可从回应中的返回码得到通知。服务器端在方法无法识别或没有实现时，返回状态代码 501（尚未没实现）。

这些方法被 HTTP/1.0 的应用程序普遍使用，完整定义请参见第 8 节。

5.1.2 请求 URI（Request-URI）

请求 URI 就是统一资源标识符（3.2 节），用来标识要请求的资源。

```
Request-URI      = absoluteURI | abs_path
```

上面两种请求 URI 方式可根据实际的请求方式选择使用。

绝对 URI（absoluteURI）格式只在代理（proxy）在产生请求时使用。代理的责任是将请求向前推送，并将回应返回。如果请求是 GET 或 HEAD 方式，而且之前的回应被缓存，如果代理忽略标题域的过期信息限制，它可能使用缓存中的消息。注意，代理可能将请求推送至另外一个代理，也可将请求直接送至绝对 URI 中所指定的目的服务器。为了避免请求循环，代理必须能够识别它的所有服务器名，包括别名、本地变量及数字形式的 IP 地址。下面是一个请求队列的例子：

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.0
```

最普通的请求 URI 形式就是原始服务器或网关用来标识资源的方式。在这种方式下，只有给出绝对路径的 URI 才能被传输（见 3.2.1 节）。例如，如客户端希望直接从原始服务器上接收资源，它们将产生一个与主机"www.w3.org"80 端口的 TCP 连接，并在完整请求之后发送下面的命令：

```
GET /pub/WWW/TheProject.html HTTP/1.0
```

注意绝对路径不可以为空，如果 URI 中没有内容，也必须加上一个"/"（server root）。

请求 URI 以编码字符串方式传输，有些字符可能在传输过程中被转义（escape），如变成“%HEXHEX”形式。具体这方面内容请参见 RFC1738[4]。原始服务器在正确解释请求之前必须对请求 URI 进行解码。

5.2 请求标题域（Request Header Fields）

请求标题域允许客户端向服务器端传递该请求的附加信息及客户端信息。该域做为请求的修饰部分，遵照编程语言程序调用参数的语法形式。

```
Request-Header = Authorization      ; Section 10.2
               | From               ; Section 10.8
               | If-Modified-Since  ; Section 10.9
               | Referer            ; Section 10.13
               | User-Agent         ; Section 10.15
```

请求标题域名只有在与协议版本的变化结合起来后，才能进行可靠的扩展。实际上，新的或实验中的标题域只要能被通讯各方识别，其语法就可使用，而无法识别的标题域都将被视为实体域。

6. 回应（Response）

在接收、解释请求消息后，服务器端返回 HTTP 回应消息。

```
Response            = Simple-Response | Full-Response
Simple-Response     = [ Entity-Body ]
Full-Response       = Status-Line               ; Section 6.1
                    *( General-Header           ; Section 4.3
                     | Response-Header          ; Section 6.2
                     | Entity-Header )          ; Section 7.1
                    CRLF
                    [ Entity-Body ]            ; Section 7.2
```

当请求是 HTTP/0.9 的或者服务器端只支持 HTTP/0.9 时，只能以 Simple-Response 方式回应。如果客户端发送 HTTP/1.0 完整请求后，接收到的回应不是以状态行（Status-Line）开头的，客户端将其视为简单回应，并相应对其进行分析。注意，简单请求只包括实体主体，它在服务器端关闭连接时终止。

6.1 状态行（Status-Line）

完整回应消息的第一行就是状态行，它依次由协议版本、数字形式的状态代码、及相应的词语文本组成，各元素间以空格（SP）分隔，除了结尾的 CRLF 外，不允许出现单独的 CR 或 LF 符。

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

状态行总是以协议版本及状态代码开头，如：

```
"HTTP/" 1*DIGIT "." 1*DIGIT SP 3DIGIT SP
```

（如，"HTTP/1.0 200"）。

这种表示方式并不足以区分完整请求和简单请求。简单回应可能允许这种表达式出现在实体主体的开始部分，但会引起消息的误解。因为大多数 HTTP/0.9 的服务器都只能回应"text/html"类型，在这种情况下，不可能产生完整的回应。

6.1.1 状态代码和原因分析（Status Code and Reason Phrase）

状态代码（Status-Code）由 3 位数字组成，表示请求是否被理解或被满足。原因分析是用简短的文字来描述状态代码产生的原因。状态代码用来支持自动操作，原因分析是为人类用户准备的。客户端不需要检查或显示原因分析。

状态代码的第一位数字定义了回应的类别，后面两位数字没有具体分类。首位数字有 5 种取值可能：

- o 1xx:：保留，将来使用。
- o 2xx：成功 — 操作被接收、理解、接受（received, understood, accepted）。
- o 3xx：重定向（Redirection）— 要完成请求必须进行进一步操作。
- o 4xx：客户端出错 — 请求有语法错误或无法实现。

o 5xx：服务器端出错 — 服务器无法实现合法的请求。

HTTP/1.0 的状态代码、原因解释在下面给出。下面的原因解释只是建议采用，可任意更改，而不会对协议造成影响。完整的代码定义在第 9 节。

```
Status-Code    = "200"   ; OK
               | "201"   ; Created
               | "202"   ; Accepted
               | "204"   ; No Content
               | "301"   ; Moved Permanently
               | "302"   ; Moved Temporarily
               | "304"   ; Not Modified
               | "400"   ; Bad Request
               | "401"   ; Unauthorized
               | "403"   ; Forbidden
               | "404"   ; Not Found
               | "500"   ; Internal Server Error
               | "501"   ; Not Implemented
               | "502"   ; Bad Gateway
               | "503"   ; Service Unavailable
               | extension-code
extension-code = 3DIGIT
Reason-Phrase  = *<TEXT, excluding CR, LF>
```

HTTP 状态代码是可扩展的，而只有上述代码才可以被当前全部的应用所识别。HTTP 应用不要求了解全部注册的状态代码，当然，如果了解了更好。实际上，应用程序必须理解任何一种状态代码，如果碰到不识别的情况，可根据其首位数字来判断其类型并处理。另外，不要缓存无法识别的回应。

例如，如果客户端收到一个无法识别的状态码 431，可以安全地假定是请求出了问题，可认为回应的状态码就是 400。在这种情况下，用户代理应当在回应消息的实体中通知用户，因为实体中可以包括一些人类可以识别的非正常状态的描述信息。

6.2 回应标题域（Response Header Fields）

回应标题域中包括不能放在状态行中的附加回应信息。该域还可以存放与服务器相关的信息，以及在对请求 URI 所指定资源进行访问的下一步信息。

```
Response-Header = Location             ; Section 10.11
                | Server               ; Section 10.14
                | WWW-Authenticate     ; Section 10.16
```

回应标题域名只有在与协议版本的变化结合起来后，才能进行可靠的扩展。实际上，新的或实验中的标题域只要能被通讯各方识别，其语法就可使用，而无法识别的标题域都将被视为实体域。

7. 实体（Entity）

完整请求和完整回应消息可能会传输请求或回应中的实体。实体由实体标题（Entity-Header）域、（通常还有）实体主体（Entity-Body）组成。从这种角度看，客户端与服务器都将扮演发送方及接收方的角色。某一时刻的角色定位则取决于在这个时刻谁在发送实体，而谁又在接收实体。

7.1 实体标题域（Entity Header Fields）

实体标题域中定义了一些可选的元信息，如有无实体、请求资源。

```
Entity-Header  =    Allow                    ; Section 10.1
                  | Content-Encoding         ; Section 10.3
                  | Content-Length           ; Section 10.4
                  | Content-Type             ; Section 10.5
                  | Expires                  ; Section 10.7
                  | Last-Modified          ; Section 10.10
                  | extension-header

extension-header = HTTP-header
```

扩展标题（extension-header）机制允许在不改变协议的前提下定义附加的实体标题域，但是不能假定接收方可以识别这些域。对于不可识别的标题域，接收方一般是忽略不管，而代理则继续将其向前推送。

7.2 实体主体（Entity Body）

与 HTTP 请求或回应一起发送的实体主体的格式和编码信息都在实体标题域（Entity-Header）中定义。

```
Entity-Body    = *OCTET
```

实体主体只在请求方法有要求时才会被放在请求消息中。请求消息标题域处的内容长度标题域（Content-Length header field）的标志将指明请求中的实体主体是否存在。包含实体主体的 HTTP/1.0 请求必须包含合法的内容长度标题域。

对回应消息来说，消息中是否包含实体主体取决于请求方法和回应代码。所有的 HEAD 请求方法的回应都不应包括主体，即便是实体标题域中指明有主体也一样。在主体中不应包括这些回应信息，全部 1xx（信息）、204（无内容）和 304（未修改）。而其它的回应必须包括实体主体或其内容长度标题（Content-Length header）域的定义值为 0。

7.2.1 类型（Type）

当消息中包括实体主体，主体的数据类型由标题域中的内容类型（Content-Type）和内容编码（Content-Encoding）决定。定义了二层顺序编码模式：

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

内容类型（Content-Type）指定了下列数据的介质类型（media type）。内容编码（Content-Encoding）可用于指示附加内容解码方式，通常用于有数据压缩属性的被请求资源。内容编码的缺省值是 none。

任何包括实体主体的消息必须含有内容类型（Content-Type）标题域，以定义主体的类型。只有当内容类型标题域中没有给出介质类型时，如简单回应消息，接收方可能对该 URL 所指定的资源进行判断，如其内容及扩展名等等，从而猜测出该主体的介质类型。如果还无法确定介质类型，接收方应当将其视为″ application/octet-stream″型。

7.2.2 长度（Length）

当实体主体被包括在消息中，主体长度可以有两种方式确定。如果内容长度（Content-Length）标题域存在，其字节值就是实体主体长度；否则，其主体长度由服务端关闭连接时确定。

由于服务端无法在连接关闭时发送回应信息，所以不能用关闭连接来指示请求主体的结束。因而，HTTP/1.0 请求如果包含主体，就必须在内容长度（Content-Length）标题域中给出合法的值。如果请求包括实体主体，且内容长度没指定，服务端将无法识别或无法从其它域中计算出其主体长度，在这种情况下，服务端将会返回 400（非法请求）回应。

注意：一些老的服务器在发送包含由服务器端动态插入的数据流文件时，支持非法的内容长度使用。要强调的是，未来版本的 HTTP 协议将会很快改变这种情况。除非客户端自己知道正在从一个支持它的服务器端得到回应，否则不应依赖于内容长度域的正确性。

8. 方法定义（Method Definitions）

通用的 HTTP/1.0 的方法集将在下面定义，虽然该方法集可以扩展，但并不保证附加的方法能够被扩展的客户端和服务器端所支持。

8.1 GET

GET 方法就是以实体方式得到由请求 URI 所指定资源的信息。如果请求 URI 只是一个数据产生过程，那么最终要在回应实体中返回的是由该处理过程的结果所指向的资源，而不是返回该处理过程的描述文字，除非那段文字恰好是处理的输出。

如果请求消息包含 If-Modified-Since 标题域，GET 方法的语法就变成“条件 GET”，即“（conditional GET）”。 条件 GET 方法可以对指定资源进行判断，如果它在 If-Modified-Since

标题域（见 10.9 节）中的指定日期后发生了更新，才启动传输，否则不传输。这种条件 GET 允许被缓存的实体在不必经过多次请求或不必要的数据传输就能进行刷新，从而有助于降低网络负载。

8.2 HEAD

HEAD 方法与 GET 几乎一样，区别在于，HEAD 方法不让服务器在回应中返回任何实体。对 HEAD 请求的回应部分来说，它的 HTTP 标题中包含的元信息与通过 GET 请求所得到的是相同的。通过使用这种方法，不必传输整个实体主体，就可以得到请求 URI 所指定资源的元信息。该方法通常用来测试超链接的合法性、可访问性及最近更新。

与条件 GET 不同，不存在所谓的“条件 HEAD”，即"conditional HEAD"。即使在 HEAD 请求中指定 If-Modified-Since 标题域，它也会被忽略。

8.3 POST

POST 方法用来向目的服务器发出请求，要求它接受被附在请求后的实体，并把它当作请求队列（Request-Line）中请求 URI 所指定资源的附加新子项。POST 被设计成用统一的方法实现下列功能：

o 对现有资源的注释（Annotation of existing resources）；

o 向电子公告栏、新闻组，邮件列表或类似讨论组发送消息；

o 提交数据块，如将表格（form [3]）的结果提交给数据处理过程；

o 通过附加操作来扩展数据库。

POST 方法的实际功能由服务器来决定，而且通常依赖于请求 URI。在 POST 过程中，实体是 URI 的从属部分，就好象文件从属于包含它的目录、新闻组文件从属于发出该文件的新闻组、记录从属于其所在的数据库一样。

成功的 POST 不需要在原始服务器创建实体，并将其做为资源；也不需要为未来的访问提供条件。也就是说，POST 方法不一定会指向 URI 指定的资源。在这种情况下，200（成功）或 204（无内容）都是适当的回应状态，取决于实际回应实体中对结果的描述。

如果在原始服务器上创建了资源，回应应是 201（已创建），并包含一个实体（对"text/html"类型最为适合），该实体中记录着对新资源请求的状态描述。

在所有的 HTTP/1.0 的 POST 请求中，必须指定合法的内容长度（Content-Length）。如果 HTTP/1.0 服务器在接收到请求消息内容时无法确定其长度，就会返回 400（非法请求）代码。

应用程序不能缓存对 POST 请求的回应，因为做为应用程序来说，它们没有办法知道服务器在未来的请求中将如何回应。

9. 状态代码定义（Status Code Definitions）

每个状态代码都将在下面描述，包括它们将对应哪个方法、以及回应中需要的全部元信息。

9.1 消息 1xx（Informational 1xx）

该类状态代码用于表示临时回应。临时回应由状态行（Status-Line）及可选标题组成，由空行终止。HTTP/1.0 中没有定义任何 1xx 的状态代码，所以它们不是对 HTTP/1.0 请求的合法回应。实际上，它们主要用于实验用途，这已经超出本文档的范围。

9.2 成功 2xx（Successful 2xx）

表示客户端请求被成功接收、理解、接受。

200 OK

请求成功。回应的信息依赖于请求所使用的方法，如下：

GET　　要请求的资源已经放在回应的实体中了。

HEAD　　没有实体主体，回应中只包括标题信息。

POST　　实体（描述或包含操作的结果）。

201 Created

请求完成，结果是创建了新资源。新创建资源的 URI 可在回应的实体中得到。原始服务器应在发出该状态代码前创建该资源。如果该操作不能立即完成，服务器必须在该资源可用时在回应主体中给出提示，否则，服务器端应回应 202（可被接受）。

在本文定义的方法，只有 POST 可以创建资源。

202 Accepted

请求被接受，但处理尚未完成。请求可能不一定会最终完成，有可能被处理过程随时中断，在这种情况下，没有办法在异步操作中重新发送状态代码。

202 回应是没有义务的，这样做的目的是允许服务器不必等到用户代理和服务器间的连接结束，就可以响应其它过程的请求（象每天运行一次的，基于批处理的过程）。

在某些回应中返回的实体中包括当前请求的状态指示、状态监视器指针或用户对请求能否实现的评估信息。

204 No Content

服务器端已经实现了请求，但是没有返回新的信息。如果客户是用户代理，则勿需为此更新自身的文档视图。该回应主要是为了在不影响用户代理激活文档视图的前提下，进行 script 语句的输入及其它操作。该回应还可能包括新的、以实体标题形式表示的元信息，它可被当前用户代理激活视图中的文档所使用。

9.3 重定向（Redirection 3xx）

该类状态码表示用户代理要想完成请求，还需要发出进一步的操作。这些操作只有当后跟的请求是 GET 或 HEAD 时，才可由用户代理来实现，而不用与用户进行交互。用户代理永远也不要对请求进行 5 次以上的重定向操作，这样可能导致无限循环。

300 Multiple Choices

该状态码不被 HTTP/1.0 的应用程序直接使用，只是做为 3xx 类型回应的缺省解释。存在多个可用的被请求资源。

除非是 HEAD 请求，否则回应的实体中必须包括这些资源的字符列表及位置信息，由用户或用户代理来决定哪个是最适合的。

如果服务器有首选，它应将对应的 URL 信息存放在位置域（Location field）处，用户代理会根据此域的值来实现自动的重定向。

301 Moved Permanently

请求到的资源都会分配一个永久的 URL，这样就可以在将来通过该 URL 来访问此资源。有编辑链接功能的客户端会尽可能地根据服务器端传回的新链接而自动更新请求 URI。

新的 URL 必须由回应中的位置域指定。除非是 HEAD 请求，否则回应的实体主体（Entity-Body）必须包括对新 URL 超链接的简要描述。

如果用 POST 方法发出请求，而接收到 301 回应状态码。在这种情况下，除非用户确认，否则用户代理不必自动重定向请求，因为这将导致改变已发出请求的环境。

注意：当在接收到 301 状态码后而自动重定向 POST 请求时，一些现存的用户代理会错误地将其改为 GET 请求。

302 Moved Temporarily

请求到的资源在一个不同的 URL 处临时保存。因为重定向有时会被更改，客户端应继续用请求 URI 来发出以后的请求。

新的 URL 必须由回应中的位置域指定。除非是 HEAD 请求，否则回应的实体主体（Entity-Body）必须包括对新 URL 超链接的简要描述。

如果用 POST 方法发出请求，而接收到 302 回应状态码。在这种情况下，除非用户确认，否则用户代理不必自动重定向请求，因为这将导致改变已发出请求的环境。

注意：当在接收到 302 状态码后而自动重定向 POST 请求时，一些现存的用户代理会错误地将其改为 GET 请求。

304 Not Modified

如果客户端成功执行了条件 GET 请求，而对应文件自 If-Modified-Since 域所指定的日期以来就没有更新过，服务器应当回应此状态码，而不是将实体主体发送给客户端。回应标题域中只应包括一些相关信息，比如缓存管理器、与实体最近更新（entity's Last-Modified）日期无关的修改。相关标题域的例子有：日期、服务器、过期时间。每当 304 回应中给出的域值发生变化，缓存都应当对缓存的实体进行更新。

9.4 客户端错误（Client Error ）4xx

4xx 类的状态码表示客户端发生错误。如果客户端在收到 4xx 代码时请求还没有完成，它应当立即终止向服务器发送数据。除了回应 HEAD 请求外，不论错误是临时的还是永久的，服务器端都必须在回应的实体中包含错误状态的解释。这些状态码适用于任何请求方法。

注意：如果客户端正在发送数据，服务器端的 TCP 实现应当小心，以确保客户端在关闭输入连接之前收到回应包。如果客户端在关闭后仍旧向服务器发送数据，服务器会给客户端发送一个复位包，清空客户端尚未处理的输入缓冲区，以终止 HTTP 应用程序的读取、解释活动。

400 非法请求（Bad Request）

如果请求的语法不对，服务器将无法理解。客户端在对该请求做出更改之前，不应再次向服务器重复发送该请求。

401 未授权（Unauthorized）

请求需要用户授权。回应中的 WWW-Authenticate 标题域（10.16 节）应提示用户以授权方式请求资源。客户端应使用合适的授权标题域（10.2 节）来重复该请求。如果请求中已经包括了授权信任信息，那回应的 401 表示此授权被拒绝。如果用户代理在多次尝试之后，回应一样还是返回 401 状态代码，用户应当察看一下回应的实体，因为在实体中会包括一些相关的动态信息。HTTP 访问授权会在 11 节中解释。

403 禁止（Forbidden）

服务器理解请求，但是拒绝实现该请求。授权对此没有帮助，客户端应当停止重复发送此请求。如果不是用 HEAD 请求方法，而且服务器端愿意公布请求未被实现原因的前提下，服务器会将拒绝原因写在回应实体中。该状态码一般用于服务器端不想公布请求被拒绝的细节或没有其它的回应可用。

404 没有找到（Not Found）

服务器没有找到与请求 URI 相符的资源。404 状态码并不指明状况是临时性的还是永久性的。如果服务器不希望为客户端提供这方面的信息，还回应 403（禁止）状态码。

9.5 服务器错误（Server Error ）5xx

回应代码以‘5’开头的状态码表示服务器端发现自己出现错误，不能继续执行请求。如果客户端在收到 5xx 状态码时，请求尚未完成，它应当立即停止向服务器发送数据。除了回应 HEAD 请求外，服务器应当在其回应实体中包括对错误情况的解释、并指明是临时性的还永久性的。

这类回应代码没有标题域，可适用于任何请求方法。

500 服务器内部错误（Internal Server Error）

服务器碰到了意外情况，使其无法继续回应请求。

501 未实现（Not Implemented）

服务器无法提供对请求中所要求功能的支持。如果服务器无法识别请求方法就会回应此状态代码，这意味着不能回应请求所要求的任何资源。

502 非法网关（Bad Gateway）

充当网关或代理的服务器从要发送请求的上游（upstream）服务器收到非法的回应。

503 服务不可用（Service Unavailable）

服务器当前无法处理请求。这一般是由于服务器临时性超载或维护引起的。该状态码暗示情况是暂时性的，要产生一些延迟。注意：503 状态码并没有暗示服务器在超载时一定要返回此状态码。一些服务器可能希望在超载时采用简单处理，即断掉连接。

10. 标题域定义（Header Field Definitions）

本节定义了 HTTP/1.0 标题域常用的语法及语义。无论是发送方还是接收方，都有可能做为客户端或服务器端，具体角色取决于在此时刻究竟是谁在接收、谁在发送。

10.1 允许（Allow）

表示由请求 URI 所指定的资源支持在 Allow 实体标题域中列出的方法，目的是让接收方更清楚地知道请求此资源的合法方式。Allow 标题域不允许在 POST 方法中使用，如果非要这么做，将被忽略。

Allow = "Allow" ":" 1#method

Example of use:

Allow: GET, HEAD

该域不能防止客户端尝试其它方法。但实际上由 Allow 标题域中的值所表示的指示信息是有用的，应当被遵守。实际的 Allow 方法集在每次向原始服务器上发出请求时定义。

由于用户代理（user agent）可能出于其它目的与原始服务器通讯，做为代理（proxy）来说，即使无法识别请求所指定的所有方法，也不能修改该请求的 Allow 标题域。Allow 标题域并不表示服务器实现了哪些方法。

10.2 授权（Authorization）

通常，用户代理在收到 401（未授权）回应时，可能（也可能不会）希望服务器对其授权。如果希望授权，用户代理将在请求中加入授权请求标题（Authorization request-header）域。授权域值由信任证书组成，其中有对用户代理所请求资源领域的授权信息。

Authorization = "Authorization" ":" credentials

HTTP 访问授权在 11 节中描述。如果请求中的授权指定了一个范围，那在此范围的其它请求也都具有相同的信任关系。

对包含授权信息域的请求来说，其回应是不能被缓存的。

10.3 内容编码（Content-Encoding）

内容编码的实体标题域（entity-header）用作介质类型（media-type）的修饰符。它指明要对资源采用的附加内容译码方式，因而要获得内容类型（Content-Type）标题域中提及的介质类型，必须采用与译码方式一致的解码机制。内容编码主要用来记录文件的压缩方法。各种压缩方法将在后面列出。

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

内容译码（Content codings）在 3.5 节中定义，例如：

Content-Encoding: x-gzip

内容编码是请求 URI 指定资源的一个特性，一般来说，资源用编码方式存储，只有在通过解码变换以后才能使用。

10.4 内容长度（Content-Length）

内容长度(Content-Length)实体标题域指明了发送到接收方的实体主体(Entity-Body)长度，用字节方式存储的十进制数字表示。对于 HEAD 方法，其尺寸已经在前一次 GET 请求中发出了。

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

例如：

Content-Length: 3495

不论实体是何种介质类型，应用程序都将通过该域来判定将要传输的实体主体尺寸。所有包括实体主体的 HTTP/1.0 的请求消息都必须包括合法的内容长度值。

任何 0 以上（包括 0）的值都是合法的内容长度值。在 7.2.2 节描述了当内容长度值没有给出时，如何决定回应实体主体长度的方法。

注意：该域的含义与在 MIME 中定义的有重要区别。在 MIME 中，它是可选域，只在"message/external-body"内容类型中使用；而在 HTTP 中，在实体被传输前，该域就决定了实体的长度。

10.5 内容类型（Content-Type）

内容类型实体标题域指明了发送给接收方的实体主体长度。对于 HEAD 方法，介质类型已经在前一次 GET 请求中发出了。

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

介质类型在 3.6 节中定义，如下例：

Content-Type: text/html

更多关于介质类型的讨论在 7.2.1 节。

10.6 日期（Date）

日期普通标题（Date general-header）域表示消息产生的时间，其语法与 RFC822 中定义的 orig-date 是一样的。该域值是 HTTP 型日期，在 3.3 节中描述。

Date = "Date" ":" HTTP-date

例如：

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

如果直接通过用户代理（如请求）或原始服务器（如回应）的连接接收到消息，日期可以认为是接收端的当前时间。然而，对于原始服务器来说，时间对其回应缓存的处理非常重要，所以，原始服务器的回应总是包括日期标题。客户端只有在发送带实体的消息时，才可

向服务器发送日期标题域，比如 POST 请求。如果接收到的消息被接收方或网关通过有日期要求的协议缓存起来时，该消息即使没有日期标题域，接收方也会为其分配一个。

理论上，日期应当在实体产生时生成，而实际上，日期可能在消息产生过程的任意时间生成，而不会造成任何不利的影响。

注意：早期版本错误地将此域值定义为实体主体封装时的日期。这已经被实际应用所更正。

10.7 过期（Expires）

过期实体标题域中的日期/时间值指定了实体过期的时间。这为信息提供方提供了使信息失效的手段。当超过此期限时，应用程序不应再对此实体进行缓存了。过期并不意味着原始资源会在此期限后发生改变或停止存在。在实际应用中，信息提供者通过检查过期标题域中所指定的时间，从而获知或预测资源将会发生改变的确切日期。该格式用的是绝对日期时间（3.2 节）。

Expires = "Expires" ":" HTTP-date

例如：

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

如果给定日期比日期标题中的要早（或相同），接收方不应缓存附加的实体。如果资源是动态自然生成的，象有些大量数据的处理，资源的实体应当被加上一个适当的过期时间值。

过期域并不能强制用户代理对其进行刷新或重新载入资源，它只用于缓存机制。当对已初始化过的资源发出新请求时，该机制才检查该资源的过期状态。

用户代理通常都有历史记录机制，如“后退”按钮和历史记录列表。该种机制可以用来重新显示某次对话（session）之前已经获取的实体信息。在缺省状态下，过期域不对历史机制使用。除非用户在配置用户代理时指定了对历史文件进行过期刷新，否则，只要实体还保存着，历史机制就能显示它，不论该实体是否已经过期。

注意：应用程序应兼容对过期标题非法或错误的实现，如碰到 0 值或非法的日期格式，应用程序应将其视为“立即过期（expires immediately）”。虽然这些值不符合 HTTP/1.0，对于一个健壮的应用来说，还是必要的。

10.8 来自（From）

From 请求标题域，如果给出来，则应包括一个使用此用户代理的人类用户的 Internet e-mail 地址。该地址应当能被系统识别，就象 RFC822[7]（已经更新为 RFC1123[6]）中的邮箱定义一样。

From = "From" ":" mailbox

例如：

From: webmaster@w3.org

该标题域可能用来做为登录目的使用，以确定对某资源的请求是否合法。它不应用于不安全的访问保护。该域的解释是，请求已按请求人指定的行为方式完成，而该请求人将为此种方式负责。特殊情况下，机器人（robot）代理也应包括此标题域，域中注明是谁运行它的，这样，当接收端发生任何问题时，都可以同这个人取得联系。

该域中的 Internet e-mail 地址可以与处理该请求的 Internet 主机分离。例如，当请求通过代理（proxy）时，应使用原始的传输地址。

注意：客户端在没有得到用户批准时，不应发送 From 标题域，因为这样做可能会产生用户隐私及网站安全方面的问题。强烈建议在请求之前提供手段以禁止（disable）、允许（enable）、修改（modify）该域的值。

10.9 从何时更改（If-Modified-Since）

If-Modified-Since 请求标题域和 GET 方法一起使用，用于处理下面情况：当在该域指定的日期以来，请求资源没有发生任何变更。这时，服务器将不会下传该资源的拷贝，即，回应不带任何实体主体，只是 304 状态码（未修改）。

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

例如：

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

条件 GET 方法可以请求服务端将在 If-Modified-Since 标题域中的指定日期后发生变更的指定资源下传，也就是说，如果资源没发生变化，就不下传了。其算法如下：

a) 如果请求的回应状态不是 200（成功）码或它传过来的 If-Modified-Since 中的日期不合法，此时按照普通 GET 来回应。如果日期比服务器的当前时间还要晚，则是非法时间。

b) 如果资源在 If-Modified-Since 日期以后有变化，回应也和普通 GET 一样

c) 如果资源在 If-Modified-Since 日期以后没变化，服务器端将回应 304（未修改）。

注：该日期应是合法的。

这样做的目的是为了以最小的代价，对被缓存信息进行有效更新。

10.10 最近更改（Last-Modified）

Last-Modified 实体标题域表示由发送方设定的资源最近修改日期及时间。该域的精确定义在于接收方如何去解释它：如果接收方已有此资源的拷贝，但此拷贝比 Last-Modified 域所指定的要旧，该拷贝就是过期的。

Last-Modified = "Last-Modified" ":" HTTP-date

例如：

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

该标题域的精确含义取决于发送方的执行方式及原始资源的自然状态。对于文件来说，可能是它在文件系统的 last-modified 时间。对于包含多个组成部分的实体来说，可能是组成部分中最新的 last-modify 时间。对数据库网关来说，可能是记录的 last-update 时间戳。对于虚（virtual）对象来说，可能是内部状态的最近改变时间。

原始服务器不应发送比服务器消息产生时间更晚的 Last-Modified 日期，因为该消息会导致服务器在未来的某个时间内，用消息原始的日期对该域值进行再次更新。

10.11 位置（Location）

Location 回应标题域定义了由请求 URI 指定的资源的位置。对于 3xx（重定向）回应，位置域必须能帮助服务器找到相应的 URL，以实现对资源的重定向。只允许用绝对 URL。

Location = "Location" ":" absoluteURI

例如：

Location: http://www.w3.org/hypertext/WWW/NewLocation.html

10.12 注解（Pragma）

Prama 普通标题域包括一些可能对请求/回应链中的任一接收方有用的特殊指示信息。从协议角度看，所有的注解指示了一些特定的可选行为，事实上，一些系统可能会要求行为与指示一致。

Pragma = "Pragma" ":" 1#pragma-directive

pragma-directive = "no-cache" | extension-pragma

extension-pragma = token ["=" word]

当"no-cache"出现在请求消息中时，应用程序应当向原始服务器推送此请求，即使它已经在上次请求时已经缓存了一份拷贝。这样将保证客户端能接收到最权威的回应。它也用来在客户端发现其缓存中拷贝不可用或过期时，对拷贝进行强制刷新。

不管注解（Pragma）指示信息对代理（proxy）及网关（gateway）应用程序如何重要，它必须能穿越这些应用，因为该信息可能对请求/回应链上的其它接收方有用。实际上，如果注解与某个接收方无关，它应当被该接收方忽略。

10.13 提交方（Referer）

提交方请求标题域是出于服务器端利益方面的考虑，允许客户端指明该链接的出处，即该指向资源地址的请求 URI 是从哪里获得的。这样,服务器将产生一个备份链接(back-links)列表，用于维护受欢迎的资源、登录、缓存优化等活动。

Referer = "Referer" ":" (absoluteURI | relativeURI)

例子：

Referer: http://www.w3.org/hypertext/DataSources/Overview.html

如果只给了部分 URI，应当参照请求 URI 来解释它。URI 不能包括段（fragment）。

注意：因为链接的原代码可能暴露一些隐私信息，因此强烈建议由用户来决定是否发送提交人域。例如，浏览器客户端有个选项可以用来进行离线浏览，可以使能或禁止提交人或表单信息的发送。

10.14 服务器（Server）

服务器回应标题域包含由原始服务器用来处理请求的软件信息。该域可包含多个产品标识（3.7 节）及注释以标识服务器及重要的子产品。按照习惯，产品标识将按其应用的重要顺序排列。

Server = "Server" ":" 1*(product | comment)

例如：

Server: CERN/3.0 libwww/2.17

如果回应要通过代理来推送，代理应用程序不应将它自己的数据加入产品列表。

注意：一些指定服务器软件的版本有启示作用，因为这些版本的软件存在一些安全漏洞，将使服务器更易受到攻击。提倡服务器软件在实现时，将此域变成可以进行配置的选项。

注意：有些服务器不遵守服务器域产品标识的语法约束。

10.15 用户代理（User-Agent）

用户代理请求标题域包含用户原始请求的信息，这可用于统计方面的用途。通过跟踪协议冲突、自动识别用户代理以避免特殊用户代理的局限性，从而做到更好的回应。虽然没有规定，用户代理应当在请求中包括此域。该域可包含多个产品标识（3.7 节）及注释以标识该代理及其重要的子产品。按照习惯，产品标识将按子产品对应用的重要性顺序排列。

User-Agent = "User-Agent" ":" 1*(product | comment)

例如：

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

注意：现存有些代理应用将它们的产品信息回到了用户代理域中，这种方法不值得提倡，因为这样做会使机器在解释这些信息时产生混淆。

注意：现在有些客户端不遵守用户代理域中产品标识的语法约束。

10.16 WWW-授权（WWW-Authenticate）

WWW-授权回应标题域必须被包括在 401（未授权）回应消息中。该域值由一个以上的 challenge 组成，这些 challenge 可用于指出请求 URI 的授权方案及参数。

WWW-Authenticate = "WWW-Authenticate" ":" 1#challenge

HTTP 访问授权处理在 11 节中描述。用户代理在解析 WWW-授权域值时要特别注意看看它是否包含一个以上的待解决问题（challenge）或是否提供了一个以上的 WWW-授权标题域，因为待解决问题（challenge）的内容中可能包含用逗号分隔的授权参数列表。

11. 访问鉴别（Access Authentication）

HTTP 提供了一个简单的质询回应（challenge-response）鉴别机制，可用于服务器通过客户端提供的授权信息来对其进行身份鉴别。授权方案用可扩展的、大小写敏感的符号来标识，后跟获取证明所需要的以逗号分隔的‘属性－值’对。

```
auth-scheme    = token
auth-param     = token "=" quoted-string
```

原始服务器用 401（未授权）回应消息来质询用户代理的授权。该回应必须包括 WWW-授权标题域，而该 WWW-授权标题域包括一个以上用于请求资源认证的参数（challenge）。

```
challenge   = auth-scheme 1*SP realm *( "," auth-param )
realm       = "realm" "=" realm-value
realm-value = quoted-string
```

凡涉及到参数（challenge）处理的授权方案都有 realm 属性（大小写敏感）。与标准 URL（相对于被访问服务器 root）联合使用的 realm 值（也是大小写敏感）用来定义保护区域。Realm 使得服务器上的被保护资源被放在特殊的保护分区内，这些区域都有各自的授权方案和（或）授权数据库。Relm 值是个字符串，通常由原始服务器来分配，对于授权方案来说，可能存在些附加的语法处理问题。

通常，用户代理在收到 401（未授权）回应时，可能（也可能不会）希望服务器对其授权。如果希望授权，用户代理将在请求中加入授权请求标题（Authorization request-header）域。授权域值由信任证书组成，其中有对用户代理所请求资源领域的授权信息。

```
credentials    = basic-credentials
                       | ( auth-scheme #auth-param )
```

可由用户代理通过信任方式来访问的区域由保护区域决定。如果早先的请求已经通过认证，在由授权方案，参数和（或）用户选择等所指定的时间间隔内，其它的请求可通过相同的信任来访问该保护区域。

除非由授权另行指定，否则单个保护区域的范围不能扩展到服务器之外。

如果服务器不希望通过请求来接受信任，它应当返回 403（禁止）回应。

HTTP 协议的访问授权不限于这种简单的质询回应（challenge-response）机制，还可以使用其它的方法，比如传输级加密或消息封装及通过附加标题域来指定授权信息等等。但是，这些方法不在本文档的讨论范围。

代理必须完全透明地处理用户授权，也就是说，它们必须在不做任何改动的前提下将 WWW-授权及授权标题向前推送，也不可以对包含授权的回应进行缓存。HTTP/1.0 不为客户端提供通过代理方式授权的方法。

11.1 基本授权方案（Basic Authentication Scheme）

用户代理必须对于每个领域（realm）通过用户标识（user-ID）及口令来对自身进行授权，这是基本授权方案的工作模式。Realm 值应当被看作不透明的字符串，该值将用于同服务器端其它的 realm 值相比较。只有用户标识及口令通过受保护资源的认证，服务器才会给请求授权。授权参数没有可选项。

在接收到对受保护区域的未经认证的资源请求时，服务器应当回应一个质（challenge），

如下：

WWW-Authenticate: Basic realm="WallyWorld"

"WallyWorld"是由服务器分配的字符串，用于对请求 URI 所指定的受保护资源进行标识。

为了接收授权，客户端需要在基于 64 位（base64 [5]）的证书中发送用户标识及口令，中间用冒号':'分隔。

```
basic-credentials = "Basic" SP basic-cookie
basic-cookie      = <base64 [5] encoding of userid-password,
                         except not limited to 76 char/line>
userid-password   = [ token ] ":" *TEXT
```

如果用户代理希望发送用户标识"Aladdin"和口令“open sesame”，应当遵循下面的标题域形式：

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Basic 授权方案是对 HTTP 服务器资源的非授权访问进行过滤的非安全方法。它基于假定客户端与服务器端的连接是安全的，为什么说是假定呢，因为在实际的开放性网络中，使用 basic 授权方案往往存在许多不安全的地方。尽管如此，客户端仍然需要实现此方案，以与采用此种方案的服务器进行通讯。

12. 安全考虑（Security Considerations）

本节的描述对下面各角色有关：信息应用开发者、信息提供者、HTTP/1.0 中受安全性限制的用户。本节仅是讨论安全问题，并对减少隐患提出了建议，但是并不提供对问题的最终解决办法。

12.1 客户授权（Authentication of Clients）

正如 11.1 节中所述，基本授权（Basic authentication）方案不是安全的用户授权方案，也不能用它来防止实体主体源码以文本方式在物理网络中传输。HTTP/1.0 并不反对在目前日益突出的安全问题面前采用其它的授权方式及加密机制。

12.2 安全方法（Safe Methods）

客户端软件开发者应当注意，客户端软件代表用户在 Internet 上与其它方面交互，并应注意避免让用户知道其间发生的具体动作，这些动作可能会暴露对交互各方有重要意义的信息。

特别情况下，按协定来看，GET 和 HEAD 方法应被视作是安全的，同重新获得数据应当没有什么不同。这就允许用户代理采用其它方法，如 POST，在某种情况下，可能存在这样一种情况，即请求中包含不安全的行为。

通常，服务器端在执行过 GET 请求之后，其结果之类的副产品还残留在服务器上；实际上，一些动态资源需要这种特性来实现。这里的重要区别在于用户没有请求这些副产品，因而也不应当对这类请求进行解释。

12.3 服务器日志信息的弊端（Abuse of Server Log Information）

服务器为保存与用户请求相关的个人数据，如阅读方式或感兴趣的主题等提供了空间。这些存储信息显然是受某些国家法律保护的，所以对此类数据的处理应当小心。用 HTTP 协议提供数据的一方应当负责保证这些信息在未得各方许可之前不会散布出去。

12.4 敏感信息传输（Transfer of Sensitive Information）

与其它协议一样，HTTP 协议不能调整传输数据的内容，也不存在未卜先知的方法，通过给定请求的上下文信息片段就能推测出信息的敏感程度。因而，应用程序应当尽可能像信息

提供方一样，为该信息提供更多的控制。在此，有三个标题域值得一提：服务端（Server）、提交方（Referer）和来自（From）。

一些指定服务器软件的版本有启示作用，因为这些版本的软件存在一些安全漏洞，将使服务器更易受到攻击。提倡服务器软件在实现时，将 Server 标题域变成可以进行配置的选项。

提交方（Referer）标题域允许阅读方式（reading patterns）被暴露，并可导出反向链接。虽然该域很有用，但是如果包含在此域的用户信息如果没有被分开，则它的作用很可能被滥用。另外，即使此域中用户信息被清除，从该域的其它信息仍可推测出其私有文件的 URI，这可能是该信息发布者所不想看到的。

来自（From）标题域可能包括一些与用户私人隐私及站点安全相关的信息，因而，在发送数据前，应当允许用户使用一些设定手段，如禁止（disable）、允许（enable）、及修改（modify），对此域信息进行配置。用户应当能够根据他们的选择或使用应用程序提供的缺省配置来设定此域的内容。

我们建议，但不做要求：为用户提供方便的界面来允许（enable）或禁止（disable）发送 From 域或 Referer 域的信息。

12.5 基于文件及路径名的攻击（Attacks Based On File and Path Names）

HTTP 原始服务器的实现应当注意，要对以服务器管理员名义发出的，对某个文件的 HTTP 请求进行限制。如果 HTTP 服务器直接将 HTTP URI 发送给系统调用，服务器要特别注意，当某个请求文件不是发往 HTTP 客户端时，要予以拒绝服务。例如，在 Unix、Microsoft Windows 以及其它的操作系统使用"../"做为上级目录名。在这样的系统下，HTTP 服务器端必须禁止通过使用这种结构的请求 URI 来访问 HTTP 服务器其它范围的资源。同样，服务器端内部使用的一些文件，包括访问控制文件，配置文件、script 代码等，也要受到特别保护，以避免被非法请求获取，导致系统敏感信息暴露。实验证明，哪怕是最小的 bug，也可能导致严重的安全问题。

13. 感谢（Acknowledgments）

本文档着重论述补充反馈方式（augmented BNF）及由 David H. Crocker 在 RFC822[7]中定义的一般结构。同样，它使用了许多由 Nathaniel Borenstein 和 Ned Freed 为 MIME [5]做的许多定义。我们希望通过它们来减少对 HTTP/1.0 及 mail 消息格式之间关系的混淆。

HTTP 协议在过去四年中发展很快，它得益于庞大而活跃的开发团体－－是他们，这些参与 WWW 讨论邮件列表的人们，造就了 HTTP 在全球范围内的成功。Marc Andreessen，Robert Cailliau， Daniel W. Connolly，Bob Denny，Jean-Francois Groff， Phillip M. Hallam-Baker，Hakon W. Lie，Ari Luotonen,Rob McCool，Lou Montulli，Dave Raggett，Tony Sanders,和 Marc VanHeyningen,他们为本文档早期版本投入了巨大精力。Paul Hoffman 提供了关于信息状态方面的信息，以及附录 C、D 的内容。

该文档从 HTTP-WG 成员的评注中得益非浅。以下是对本规范做出贡献的人们：

Gary Adams
Keith Ball
Paul Burchard
Mike Cowlishaw
Michael A. Dolan
Jim Gettys
Koen Holtman
Bob Jernigan
Martijn Koster
Daniel LaLiberte
Albert Lunde
Larry Masinter
Jeffrey Mogul
Bill Perry
Owen Rees
David Robinson
Rich Salz
Chuck Shotton
Simon E. Spero
Francois Yergeau
Jean-Philippe Martin-Flatin
Harald Tveit Alvestrand
Brian Behlendorf
Maurizio Codogno
Roman Czyborra
John Franks
Marc Hedlund
Alex Hopmann
Shel Kaphan
Dave Kristol
Paul Leach
John C. Mallery
Mitra
Gavin Nicol
Jeffrey Perry
Luigi Rizzo
Marc Salomon
Jim Seidman
Eric W. Sink
Robert S. Thau
Mary Ellen Zurko

14. 参考书目（References）

[1] 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, University of Minnesota, March 1993.

[2] Berners-Lee, T., "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, CERN, June 1994.

[3] Berners-Lee, T., and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, MIT/W3C, November 1995.

[4] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994.

[5] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993.

[6] Braden, R., "Requirements for Internet hosts - Application and Support", STD 3, RFC 1123, IETF, October 1989.

[7] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[8] F. Davis, B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification." (v1.5), Thinking Machines Corporation, April 1990.

[9] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, UC Irvine, June 1995.

[10] Horton, M., and R. Adams, "Standard for interchange of USENET Messages", RFC 1036 (Obsoletes RFC 850), AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[11] Kantor, B., and P. Lapsley, "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News", RFC 977, UC San Diego, UC Berkeley, February 1986.

[12] Postel, J., "Simple Mail Transfer Protocol." STD 10, RFC 821, USC/ISI, August 1982.

[13] Postel, J., "Media Type Registration Procedure." RFC 1590, USC/ISI, March 1994.

[14] Postel, J., and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, USC/ISI, October 1985.

[15] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/ISI, October 1994.

[16] Sollins, K., and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, MIT/LCS, Xerox Corporation, December 1994.

[17] US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[18] 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.

15. 作者地址（Authors' Addresses）

Tim Berners-Lee
Director, W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.

Fax: +1 (617) 258 8682
EMail: timbl@w3.org

Roy T. Fielding
Department of Information and Computer Science
University of California
Irvine, CA 92717-3425, U.S.A.

Fax: +1 (714) 824-4056
EMail: fielding@ics.uci.edu

Henrik Frystyk Nielsen
W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.

Fax: +1 (617) 258 8682
EMail: frystyk@w3.org

附录（Appendices）

这些信息出现在附录中仅有一个理由，即他们没有成为 HTTP/1.0 规范的组成部分。

A. Internet 介质类型消息/http（Internet Media Type message/http）

做为 HTTP/1.0 协议的补充，该文档做为 Internet 介质类型"message/http"的规范。下面内容被登记在 IANA[13]中。

介质类型名（Media Type name）：　　message

介质子类型名（Media subtype name）：　　http

请求参数（Required parameters）：　　none

可选参数项（Optional parameters）：　　version, msgtype

版本（version）：附加消息的 HTTP 版本号，比如“1.0”。如果没有给出，版本可以从其主体的第一行中得到。

消息类型（msgtype）：消息类型－－请求（request）或回应（response）。如果没有给出，版本可以从其主体的第一行中得到。

编码考虑（Encoding considerations）：只允许用"7bit", "8bit",或"binary" 。

安全考虑（Security considerations）： none

B. 容错应用（Tolerant Applications）

虽然此文档指明了产生 HTTP/1.0 消息的必要条件,并非所有的应用程序都校正他们的实现。因此，我们建议应用程序增强其容错能力，以便在岐义仍可被明确解释时，还能保证正常运行。

客户端解析状态行（Status-Line）及服务器解析请求行（Request-Line）时，应当做到容错。特别是，即使只需要一个 SP 分隔的情况下，它们也可接受以任何数量的 SP 或 HT 字符分隔的域。

HTTP 标题域的行终止符是顺序字符 CRLF。而我们建议应用程序在解析这类标题时，也应识别单个 LF（没有前面的 CR）做为终止符情况。

C. 与 MIME 的关系（Relationship to MIME）

HTTP/1.0 使用了许多为 Internet Mail（RFC822[7]）及多用途 Internet 邮件扩展（Multipurpose Internet Mail Extensions）MIME[5]定义的结构，以允许实体通过一种开放的可扩展的机制进行传输。实际上，HTTP 中有些特性与 RFC1521 中讨论的邮件不同，这些区别被用来优化二进制传输的性能，给介质类型的使用提供了更大的自由度，使日期比较变

得更加容易，当然，这也是为了兼容早期的一些 HTTP 服务器及客户端的应用。

在写作本文时，据说 RFC1521 将被修订。修订版本将会包括一些出现在 HTTP/1.0 中的已有的应用，但这些应用在目前的 RFC1521 中尚未包括。

该附录描述了 HTTP 与 RFC1521 中的不同之处。代理和网关在限制 MIME 环境时，应当注意到这些区别，并在必须时提供相应的转换支持。从 MIME 到 HTTP 环境的代理和网关也要注意这些区别，因为一些转换可能是必须的。

C.1 转换为规范形式（Conversion to Canonical Form）

RFC1521 要求 Internet 邮件实体在被传输前转换成规范形式，正如 RFC1521[5]附录 C 中所描述的那样。本文档中 3.6.1 节中描述了 HTTP 在传输时允许的“text”介质类型的子类型的具体形式。

RFC1521 要求"text"的内容类型（Content-Type）必须用 CRLF 作为行中断符，禁止单独使用 CR 或 LF。HTTP 允许在 HTTP 传输时使用 CRLF、单独的 CR 或 LF 做为行中断符。

只要有可能，HTTP 环境或 RFC1521 环境下的代理或网关应当将本文档 3.6.1 节中描述的文本介质类型中的所有行中断符都转换成 CRLF。注意，由于存在着内容编码（Content-Encoding）问题，以及 HTTP 允许使用多字符集，而其中的某些字符集不用字节 13 和 10 做为 CR 和 LF，这样就使实际的处理更加复杂。

C.2 日期格式转换（Conversion of Date Formats）

HTTP/1.0 使用受限制的日期格式集（3.3 节）以简化日期比较的处理。其它协议的代理和网关应当保证消息中的任何日期标题域与 HTTP/1.0 格式一致，否则，要对其进行改写。

C.3 内容编码介绍（Introduction of Content-Encoding）

RFC1521 不包括殊如 HTTP/1.0 中内容编码标题域之类的概念。由于内容类型域是介质类型的修饰，因而从 HTTP 到 MIME 兼容协议中的代理和网关必须在将消息向前推送之前，更改内容类型标题域（Content-Type）的值或者对实体主体（Entity-Body）进行解码（有些 Internet mail 内容类型的实验性应用采用介质类型参数为";conversions=<content-coding>"来替代内容解码，而事实上，该参数并非 RFC1521 的组成部分）。

C.4 无内容传输编码（No Content-Transfer-Encoding）

HTTP 不使用 RFC1521 的 CTE（Content-Transfer-Encoding）域。与 MIME 协议兼容的代理和网关在向 HTTP 客户端传递回应消息前都必须清除任何无标识的 CTE 编码（"quoted-printable"或"base64"）。

从HTTP到MIME兼容协议的代理和网关要负责保证协议上消息格式正确及编码传输安全，所谓安全传输是指满足对应协议所规定的限制或约束标准。代理或网关应当用适当的内容传输编码（Content-Transfer-Encoding）来标识数据，以提高在目的协议上实现安全传输的可能性。

C.5 多个主体的 HTTP 标题域（HTTP Header Fields in Multipart Body-Parts）

在 RFC1521 中，大多数多个主体组成的标题域通常会被忽略，除非其域名以"Content-"开头。在 HTTP/1.0 中，多个主体部分（multipart body-parts）所包含的任何 HTTP 标题域，只对对应的部分有意义。

D. 附加特性（Additional Features）

该附录中包括的一些协议元素存在于一些 HTTP 实现中，但并非对所有的 HTTP/1.0 的应用都适用。开发者应注意这些特性，但不能依赖它们来与其它的 HTTP/1.0 应用程序进行交互。

D.1 附加请求方法（Additional Request Methods）

D.1.1 PUT

PUT 方法请求服务器将附件的实体储存在提供的请求 URI 处。如果该请求 URI 指向的资源已经存在，则附件实体应被看做是当前原始服务器上资源的修改版本。如果请求 URI 没有指向现存的资源，该 URI 将被该请求的用户代理定义成为一个新的资源，原始服务器将用该 URI 产生这个资源。

POST 与 PUT 两种请求的基本区别在于对请求 URI 的理解不同。在 POST 请求方法中的 URI 所标识的资源将做为附件实体被服务器处理，该资源可能是数据接收处理过程、某些其它协议的网关、或可被注释的单独实体。与此相反，用户代理很清楚它发出的 PUT 请求中附带 URI 所标识的实体指向何处，而服务器处不应将该请求用到其它资源头上。

D.1.2 DELETE

DELETE 方法请求原始服务器删除由请求 URI 所指定的资源。

D.1.3 LINK

LINK 方法建立与请求 URI 所指定资源或其它已存在资源之间的一个或多个连接关系。

D.1.4 UNLINK

UNLINK 方法删除与请求 URI 所指定资源之间的一个或多个连接关系。

D.2 附加标题域定义（Additional Header Field Definitions）

D.2.1 Accept

Accept 请求标题域用于指示可被接受的请求回应的介质范围列表。星号"*"用于按范围将介质类型分组，用"*/*"指示可接受全部介质类型；用"type/*"指示可接受 type 类型的所有子类型。对于给定请求的上下文，客户端应当表示出哪些类型是它可以接受的。

D.2.2 可接受的字体集（Accept-Charset）

Accept-Charset 请求标题域用来指示除了 US-ASCII 和 ISO-8859-1 外，首选的字符集。该域将使客户端有能力理解更广泛的或有特殊用途的字符集，从而在服务器上可以存放采用此类字符集的文档。

D.2.3 可接受编码（Accept-Encoding）

Accept-Encoding 请求标题域与 Accept 相似，但是限制了回应中可接受的内容编码（content-coding）值。

D.2.4 可接受语言（Accept-Language）

Accept-Language 请求标题域与 Accept 相似，但限制了请求回应中首选的自然语言集。

D.2.5 内容语言（Content-Language）

Content-Language 实体标题域描述了附加实体中为听众指定的自然语言。注意，这可能与在实体内部使用的各种语言不是一码事。

D.2.6 连接（Link）

Link 实体标题域描述了实体和某些其它资源之间的关系。一个实体可能包括多个连接值。处于元信息级的 Link 指明了分层结构和导航路径之间的关系。

D.2.7 MIME 版本（MIME-Version）

HTTP 消息可能包括一个单独的 MIME 版本的普通标题（general-header）域，用以指示用来构造消息的 MIME 协议的版本。MIME 版本标题域的使用，正如 RFC1521[5]中定义的那样，应当用来指示消息是否符合 MIME 规范。然而不幸的是，一些老的 HTTP/1.0 服务器不加选择地发送此域，导致此域已经被废弃。

D.2.8 在….后重试（Retry-After）

Retry-After 回应标题域可与 503（服务不可用）回应一起使用，用于指示服务器停止响应客户请求的时间长短。该域的值可用 HTTP 格式的日期表示，也可以用整数来表示回应时间后的秒数。

D.2.9 标题（Title）

Title 实体标题域用于指示实体的标题。

D.2.10 URI

URI 实体标题域可能包含一些或全部统一资源标识（Uniform Resource Identifiers），

见 3.2 节，通过这些标识来表示请求 URI 所指定的资源。并不担保根据 URI 一定能够找到指定的资源。

1
RFC 文档中文翻译计划