Network Working Group(网络工作组)
Request for Comments: 2616
Obsoletes(过时弃用）: 2068
Category: Standards Track （类别:标准组 ）

R. Fielding
UC Irvine
J. Gettys
Compaq/W3C
J. Mogul
Compaq
H. Frystyk
W3C/MIT
L. Masinter
Xerox
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT

June 1999

超文本传输协议-HTTP/1.1

本备忘录状况

本文档说明了用于互联网社区的标准化跟踪协议，但还需要讨论和建议以便更加完善。请参考"互联网官方协议标准"（STD1）来了解本协议的标准化状态。分发散布本文是不受限制的。

版权声明

摘要

超文本传输协议（HTTP）是一种应用于分布式、协作式、超媒体信息系统的应用层协议。它是一种通用的，状态无关的协议，可以用于除了超文本以外，还可以通过扩展它的请求方法，错误代码和报头[47]来完成更多任务，比如名称服务和分布对象管理系统。HTTP 的一个特点是数据表示方式的典型性（typing）和可协商性，允许建立独立于被传输数据的系统。

HTTP 在 1990 年 WWW 全球信息刚刚起步的时候就得到了应用。本规范定义了 HTTP/1.1 协议，这是 RFC 2068 的升级版[33]。

目录

[页码 2]

[页码 3]

[页码 4]

[页码 6]

---

1 概述

1.1 目的

超文本传输协议（HTTP）是一种应用于分布式、合作式、多媒体信息系统的应

用层协议。在 1990 年 WWW 全球信息刚刚起步的时候 HTTP 就得到了应用。HTTP 的第一个
版本叫做 HTTP/0.9,是一种为了在互联网传输原始数据的简单协议。由 RFC 1945[6]定义的 HTTP/1.0 进一步完善了这个协议，它允许消息以类似 MIME 的格式传送，包含有关数据传输的维护信息和关于请求/应答的句法修正。但是，HTTP/1.0 没有充分考虑到分层代理、高速缓存的作用以及对持久连接和虚拟主机的需求。并且，随着不完善的应用程序的激增，HTTP/1.0 迫切需要一个新的版本,以便使两个通信应用程序能够确定彼此的真实性能。

本规范定义的协议为"HTTP/1.1"。这个协议与 HTTP/1.0 相比，要求更为严格，以确保各项功能得到可靠实现。

实际的信息系统除了简单的取获外，还要求有更多的功能性，包括查找，前端更新和注解。HTTP 使用开放的方法集和报头集以指示请求[47]的目的。它是建立在统一资源标识符(URI)[3]提供的地址(URL)[4]和名字(URN)上[20]，以指出方法应用于哪个资源的。消息以类似于一种叫做多用途网络邮件扩展（MIME）[7] 的互联网邮件的格式传递。

[页码 7]

HTTP 也是用于用户代理及服务代理（网关）到其他网络系统的通用通信协议，这样的网络系统可能由 SMTP[16],NNTP[13],FTP[18],Gopher[2]和 WAIS[10]协议支持。这样，HTTP 允许不同的应用程序对资源进行基本的超媒体访问。

1.2 要求

本文的关键词"必须", "绝对不可以", "要求", "最好", "最好不", "应该", "不应该", "推荐", "可以", 和 "可选"将由 RFC 2119[34]解释。

一个实现（具体应用）如果不能满足协议提供的一个或多个“必须”或“要求”等级的要求，是不符合要求的。一个实现如果满足所有“必须”或“要求”等级以及所有“应该”等级的要求，则可称为"绝对符合"的；若满足所有“必须”等级的要求但不能满足所有“应该”等级的要求则被称为"部分符合"。

1.3 术语

本规范用到了若干术语，以表示 HTTP 通信中各参与者和对象扮演的不同角色。

“连接”（Connection）
为通信而在两个程序间建立的传输层虚拟电路。

“消息”(Message)
HTTP 通信中的基本单元。它由一个结构化的八比特字节序列组成，与第 4 章定义

的句法相匹配，并通过连接得到传送。

“请求”(Request)
一种 HTTP 请求消息，参看第 5 章的定义。

“应答”(Response)
一种 HTTP 应答消息，参看第 6 章的定义。

[页码 8]

“资源”(Resource)
一种网络数据对象或服务，可以用第 3.2 节定义的 URI 描述。资源可以以多种表现方式（例如多种语言，数据格式，大小和解决方案）或其他不同的途径获得。

“实体”(Entity)
作为请求或应答的有效负荷而传输的信息。一个实体包含报头形式的维护信息和消息体形式的内容,由第 7 节详述.

“表示方法”(Representation)
一个应答包含的实体是由内容协商决定的，如第 12 章所述。一个特定的应答状态所对应的表示方法可能有多个。

“内容协商”(Content Negotiation)
为“请求”服务时选择适当表示方法的机制，如第 12 节所述。任何应答里实体的表示方法都是可协商的(包括出错应答).

“变量”（Variant)
在任何给定时刻，与一个资源对应的表示方法可以有一个或更多。每个表示方法称作一个变量。使用变量这个术语并不一定意味着资源是由内容协商决定的。

“客户”（Client）
为发送“请求”而建立连接的程序.

“用户代理”(User agent)
能初始化“请求”的客户。常见的如浏览器，编辑器，蜘蛛(搜索引擎)，或其他的终端用户工具。

“服务器”(Server)
接受连接以便通过发回应答为请求提供服务的应用程序。任何给定的程序都可以既是客户端又是服务器；我们使用这些术语仅指特定连接中程序扮演的角色，而不是指通常意义上程序的功能，同样，任何服务器都可以基于每个请求的性质转换行为，扮演原服务器，代理，网关，或者隧道等角色。

[页码 9]

“原服务器”（Origin server)
给定的资源驻留或创建的地方.

“服务代理”（ Proxy）
一个既做服务器又做客户端的中介程序，其用途为其他客户发送请求，请求在内部得到服务，或者经过可能的翻译后向前传递请求。一个代理服务器必须能同时实现本规范中对客户和服务器要求。"透明代理"是一种除了代理要求的验证和鉴定外不修改请求或应答的代理。"非透明代理"是一种修改请求或应答以便为用户代理提供附加服务的服务代理，附加服务包括类注释服务，媒体类型转换，协议简化，或者匿名滤除等。除非经明确指出，HTTP 代理的“要求”对两种代理都适用。

“网关”（gateway）
为其他服务器充当中介的服务器。与代理服务器不同，网关接收请求，仿佛它就是被请求资源所在的原服务器；提出请求的客户可能觉察不到它正在同网关通信。

“隧道”（tunnel）
一个在两个连接之间充当仅仅中继的中间程序。从启动时候起，隧道虽然可能已经被 HTTP 请求初始化了，但它并被认为是 HTTP 通信的一方。当中继连接的两端都关闭的时候，隧道就不再存在了。

“高速缓存”（Cache）
一个程序的应答消息的本地存储和控制此信息存储、检索和删除的子系统。一个缓存存储可缓存的应答是为了减少以后对同样请求的应答时间和网络带宽消耗。任一客户或服务器都可能包含一个高速缓存，但缓存不能应用于一个充当隧道的服务器。

“可缓存”（Cacheable）
如果允许存储应答信息的一份拷贝以便运用于应答后继请求，这个应答就是可缓存的。用来确定 HTTP 应答的可缓存能力的规则在 13 节中有定义。即使一个资源是可缓存的，也可能在缓存能否将缓存的拷贝用于某特定请求一问题上存在附加的约束。

[页码 10]

“直接，第一手的“（first-hand）
如果一个应答直接从原服务器而来，而没有经过不必要的服务代理的延迟，那么应答就是直接（第一手的）的。如果它的有效性已经被原服务器直接认证，那么这个应答也同样是第一手的.

”明确终止时间“（explicit expiration time）
原服务器指定的一个实体在无需进一步确认的情况下就不再被缓存返回的时间。

”启发式终止时间“（heuristic expiration time）
当没有”明确终止时间“可用时，由高速缓存所指定的终止时间.

”年龄“（Age）
一个应答的年龄是从它被发送，或被原服务器成功确认到现在的时间.

”保鲜寿命“（Freshness lifetime）
一个应答生成和过期之间的时间长度。

”新鲜的“（Fresh）
如果一个应答的年龄还没有超过保鲜寿命，它就是新鲜的。

”陈旧的“（Stale）
一个应答的年龄已经超过了它的保鲜寿命，就是陈旧的。

”语义透明（semantically transparent）
当缓存的使用除了改善性能外既未影响请求客户机也未影响原服务器时，缓存对于某特定的应答就是工作于语义透明方式了。当缓存语义透明时，客户收到的应答恰好与原服务器直接处理请求后得到的应答相同（除了逐段转接的报头部分）。

“有效性判别器”（Validator）
用来查找一个缓存记录是否是一个实体的等效拷贝的协议元素。(例如,一个实体标签（entity tag）或最终更改时间（Last-Modified time))。

“上游/下游”（upstream/downstream）
上游和下游描述了消息的流动：所有消息都从上游流到下游。

“向内/向外”（inbound/outbound）
向内和向外指的是消息的请求和应答路径："向内"即"移向原服务器"，"向外"即"移向用户代理".

1.4 概述

HTTP 协议是一种请求/应答协议。 与主机建立连接后，客户以请求方法，URI 和协议版本的形式向服务器发送请求，继以类 MIME 信息，其中包括请求修订，客户

信息和可能的正文内容。服务器用状态行进行应答，这包括消息的协议版本和成功或错误代码，后面是包括服务器信息，实体维护信息和可能的实体内容的类 MIME 消息。HTTP 和 MIME 之间的关系在附录 19.4 节中描述。

大部分的 HTTP 通信由用户代理引发，由对某原服务器上一个资源的请求构成。最简单的情形，可以经用户代理（UA）和原服务器（O）之间的单一连接（v）完成。

```
        请求链---------------------->
用户代理(UA)----------单一连接（v)----------原服务器（O）
        <----------------------应答链
```

当一个或多个中介在请求/应答链中出现的时候，会出现更复杂的情形。 常见的中介形式有三种：服务代理，网关和隧道。服务代理是一种向前传送代理，它接收绝对形式的 URI 请求，重写全部或部分消息，然后把重新格式化后的请求发送到 URI 确定的服务器上。网关是一种接收代理，它充当其他服务器的上层，如果必要的话，它将请求翻译为下层服务器的协议。隧道不改变消息而充当两个连接之间的中继点；它应用于通信需要穿过中介（如防火墙），而中介甚至不能理解信息内容的时候。

```
          请求链--------------------------------------->
UA-----v-----A-----v-----B-----v-----C-----v-----O
          <---------------------------------------应答链
```

上图显示了用户代理和原服务器之间的三个中介（A，B 和 C)。请求或应答消息游历整条链的时候需通过四个独立的连接。这个特性很重要，因为某些 HTTP 通信选项只能应用于到最近的非隧道邻居，链的终点的连接，或者沿着链的所有连接。图表尽管是线性的，每部分可能都在忙于多路同时通信。例如，B 可以接收来自不同于 A 的许多客户的请求，并且/或者转送到不同于 C 的服务器，与此同时，它还在处理 A 的请求。

任何非隧道的通信成员都可以使用内部的缓存来处理请求。高速缓存的作用是如果沿着链的一个成员对请求采用了高速缓冲的应答，请求/应答链就会大大缩短。以下图解作为结果产生的链，假定 B 拥有来自 O（通过 C）的一个从前应答的备份，请求尚未被 UA 或 A 缓存。

```
          请求链---------->
UA-----v----------A-----v-----B-----C----O
          <---------应答链
```

并不是所有的应答都能有效地缓存，一些请求可能含有修改量，对缓存动作有特殊的要求。缓存动作和缓存应答的 HTTP 要求将在第 13 节定义。

实际上，目前万维网上有多种结构和配置的高速缓存和代理被实验或使用。这些系统包括节省越洋带宽的国家级代理缓存层，广播或多点通信缓存接口，通过 CD-ROM 分配缓存数据子集的机构，等等。HTTP 系统应用在宽频带连接的企业局域网中，通过 PDA 的低耗无线连接和间歇式连接访问。HTTP1.1 的目标是支持各种各样的应用配置，引进协议结构以满足那些需要较高可靠性，可以排除故障或至少指示故障的网络应用的要求。

HTTP 通信在通常发生在 TCP/IP 连接上。默认端口是 TCP 80，不过其它端口也可以使用。这并不妨碍 HTTP 在其他因特网协议或者其他的网络上实现。http 仅仅期望可靠的传输；任何提供这种保证的协议都可以使用；HTTP/1.1 请求和应答结构在某（传输）协议上的传输数据单元映象问题已经不在本规范的讨论范围之内。

[页码 13]

在 http/1.0 中，大部分的实现为每个请求/应答交换使用了新连接。而 http/1.1 中，一个连接可以用于一个或更多请求/应答交换，虽然连接可能会因为各种原因中断（见第 8.1 节）。

2 符号惯例和一般语法

2．1 扩充 BNF

本文档规定的所有机制都用两种方法描述：散文体（prose)和类似于 RFC 822 的扩充范式(BNF)。要理解本规范，使用者需熟悉符号表示法。扩充 BNF 包括下列结构：

名字 = 定义

一条规则的名字仅仅是名字本身（没有任何"<"和">"），由等号"="把它和后面的定义分离开。只有空格是用来缩排后续行，以表示定义的规则多于一行时，空格才是真正重要的。一些基本规则使用了大写字母，如 SP，LWS，HT，CRLF，DIGIT，ALPHA，等等。无论何时，如果加上尖括号会有利于识别规则名字，就可以在定义的范围内使用。

"文字"

文字原文使用引号。除特殊情况，原文对外界不敏感。

规则 1 | 规则 2

由竖线("|"）分开的元素是可选的，例如，"yes | no"表示 yes 或 no 都是可接受的。

（规则 1 规则 2）

围在括号里的多个元素视作一个元素。这样"(elem (foo | bar) elem)"允许标

记序列"elem foo elem"和 elem bar elem"。

* 规则

"*" 放在元素前表示重复。完整的形式是"<n>*<m>元素"，表示元素至少出现<n>次，至多出现<m>次。默认值是 0 和无穷大，所以"*(元素)"允许任何数值，包括零；"1*元素"至少需要一次；"1*2element"允许一次或两次。

[规则]

方括号里是任选元素；"[foo bar]"相当于"*1(foo bar)".

[页码 14]

N 规则

特殊的重复："<n>(元素)"相当于"<n>*<n>(元素)"； 也就是说，(元素)正好出现了<n>次。这样 2DIGIT 是一个两位数字，3ALPHA 是一个由三个字符组成的字符串。

#规则

类似于"*",结构"#"是用来定义一系列元素的。完整的形式是"<n>#<m>元素,表示至少<n>个元素，至多<m>个元素，元素之间被一个或多个逗号(",")以及可选的线性空白区(LWS)隔开了。这就使得列表的一般形式变得非常容易；像
(*LWS element) *(*LWS ","*LWS element))
就可以表示为
1#element
无论在哪里使用这个结构，空元素都是容许的，但是不计入元素出现的次数。换句话说，"(元素), , (元素) "是允许的，但是仅仅视为两个元素。因此，在至少需要一个元素的地方，必须存在至少一个非空元素。默认值是 0 和无穷大，这样，"#element"允许任何数字，包括零；"1#element"至少需要 1 个元素；"1#2element"允许 1 个或 2 个元素。

；注释

用分号引导的注释，从规则正文的右边一段距离开始直到行尾。这是做注释的简单方法，注释与说明是同样有用的。

隐含 *LWS

范式文法所描述的语法是基于字的。除非特别注明，线性空白可出现在任何两个相邻字之间（标记或引用字符串），以及相邻字和间隔符之间，并不改变一个域的含义。任何两个标记之间（下面会对"token(标记)"进行定义）必须有至少一个分割符，否则将会被理解为单一标记。

2．2 基本规则

下面的规则描述了基本的解析结构，贯穿于本规范的全文。US-ASCII(美国信息

交换标准码)字符规定是由 ANSI X3.4-1986[21]定义的。

[页码 15]

```
OCTET          = <任意八比特的数据序列>
CHAR           = <任意 ASCII 字符（八进制 0-127）>
UPALPHA        = <任意大写字母"A"..."Z">
LOALPHA        = <任意小写字母"a"..."z">
ALPHA          = UPALPHA | LOALPHA
DIGIT          = <任意数字 0，1，...9>
CTL            = <任意控制字符(octets 0 - 31)及删除键 DEL（127）>
CR             = <US-ASCII CR, 回车(13)>
LF             = <US-ASCII LF, 换行符(10)>
SP             = <US-ASCII SP, 空格(32)>
HT             = <US-ASCII HT, 水平制表 (9)>
<">            = <US-ASCII 双引号(34)>
```

HTTP/1.1 将 CR LF 的顺序定义为除了报文以外任何协议元素的行尾标志（宽松应用见附录 19.3）。报文内部的行尾标志是由它的关联媒体类型定义的，如 3.7 节所述。

```
CRLF           = CR LF
```

如果延续行由空格或水平制表开始，HTTP/1.1 的报头字段值可以折叠到到多行上。所有的线性空白，包括折叠，具有同 SP 一样的语义。接收者在解释域值或将消息转送到下游时可以用单个 SP 替代任何线性空白。

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

文本规则仅仅适用于描述字段的内容和不会被消息语法分析程序解析的值。*TEST 的字可以包含 ISO-8859-1[22]里的字符，也可以包含字符规定里的字符[14]。

```
TEXT           = <除 CTLs 以外的任意 OCTET，但包括 LWS>
```

一个 CRLF 仅仅在作为报头字段延续的一部分时才在 TEXT 定义里允许使用。并且最好折叠的 LWS 会在文本分析前被单个的空格所取代。

十六进制数字字符用在数个协议元素里。

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

[页码 16]

许多 HTTP/1.1 的报头字段值是由 LWS 或特殊字符分隔的字构成的。这些特殊字符必须包含在引用字符串里，方可用在参数值（如 3.6 节定义）里。

```
token （标记）          = 1*<除 CTLs 与分割符以外的任意 CHAR >
separators（分割符）    = "(" | ")" | "<" | ">" | "@"
                | "," | ";" | ":" | "\" | <">
                | "/" | "[" | "]" | "?" | "="
                | "{" | "}" | SP | HT
```

用圆括号括起来的注释可以包含在一些 HTTP 报头字段里。只有作为域值定义的一部分时注释才是允许的。在其他域里，圆括号视作字段值的一部分。

```
comment （注释）= "(" *( ctext | quoted-pair | comment ) ")"
ctext           = <除"(" and ")"以外的任意 TEXT >
```

一个文本字符若在双引号里，则当作一个字。

```
quoted-string  = ( <"> *(qdtext | quoted-pair ) <"> )
qdtext         = <除<">以外的任意 TEXT >
```

反斜线("\")可以用作单一字符引用结构，但仅在引号字符串或注释里。

```
quoted-pair    = "\" CHAR
```

3 协议参数

3.1 HTTP 版本

HTTP 使用"<主要>.<次要>"的编号方案表示协议版本。协议的版本方针是希望允许发送者表示消息的格式和性能以便理解更深一层的 HTTP 通信， 而不仅仅是当前通信获得的特征。消息构件的增加不影响通信动作，或仅仅增加了扩展域值，版本号并没有因此变化。协议的改变增加了一些特征，没有改变一般的消息解析规则，但是增加了消息的语义或者暗含了发送者新增的功能，这时<次要>数字便要增大。当协议的消息格式改变时，<主要>数字增大。

[页码 17]

HTTP 消息的版本在消息的第一行 HTTP-版本字段里表示。

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

注意主要和次要数字必须看作是两个分离的整数，二者都可以增加到比单位数还大。这样，HTTP/2.4 的版本比 HTTP/2.13 低，依次 HTTP/2.3 的版本比 HTTP/12.3 低。首位的零必定被接收者忽视，一定不要发送。

一个发送包含 HTTP 版本"HTTP/1.1"的请求或应答消息的应用，必须至少有条件的服从本规范。至少有条件服从本规范的应用应该在消息里使用"HTTP/1.1"的 HTTP -版本，任何与 HTTP/1.0 不兼容的消息则必须这样做。关于何时发送特殊的 HTTP-版本值，更多资料请参看 RFC 2145[36].

一项应用使用了某 HTTP 版本，则这此版本应是它至少有条件服从的最高 HTTP 版本.

服务代理和网关转送的消息的协议版本与应用版本不同时，需要小心。既然协议版本表示发送者的协议性能，代理/网关一定不能发送标示版本高于它本身的实际版本的消息。如果收到更高版本的请求，代理/网关必须降低请求的版本，或者发出出错应答，或者切换到隧道动作。

由于自 RFC 2068[33]发布后发现的 HTTP/1.0 代理协同工作问题，高速缓存代理必须，网关可以，隧道必须不将请求提升到它们支持的最高版本。代理/网关的应答的主要版本号必须同请求相同。

注：HTTP 版本的转换可能会包含相关版本必需或禁止的报头字段修改。

3.2 统一资源标识符（URI）

URI 的许多名字已为人所知：WWW 地址，通用文件标识符，通用资源标识符[3]，以及最后统一资源定位器(URL)[4]和统一资源名称(URN)[20]的结合。只要与 HTTP 相关，统一资源定位器只是格式化的字符串，它通过名称，地址，或任何别的特征确定了资源的位置。

[页码 18]

3.2.1 一般语法

根据使用时的上下文，HTTP 里的 URI 可以表示成绝对形式或基于已知的 URI 的相

对形式。两种形式的区别是根据这样的事实：绝对 URI 总是以一个主题名字作为开头，其后是一个冒号。关于 URL 更详尽的信息请参看"统一资源标识符(URI):一般语法和语义",RFC 2396 [42](代替了 RFCs 1738 [4]和 RFC 1808 [11])。本规范采用那份规范里关于"URI-参考","绝对 URI","相对 URI","端口","主机","绝对路径"和"认证"的定义.

HTTP 协议不对 URI 的长度作事先的限制。服务器必须能够处理它们服务的任何资源的 URI，并且应该能够处理无限长度的 URI。如果它们提供可以产生这种 URI 的基于 GET 的形式。服务器在一个 URI 长度超过它能处理的范围时，应该返回 414（URI 太长）状态值。(见 10.4.15 小节)

注:服务器在依赖长于超过 255 字节的 URI 时应谨慎，因为一些旧的客户或服务代理的实现可能不支持这些长度.

3.2.2 http URL

“http”主题用来通过 HTTP 协议定出网络资源的位置。本节为 HTTP 的 URL 定义了这种主题相关的语法和语义.

http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

如果端口为空或未给出，就假定为 80。语义即：已识别的资源放在服务器上，在那台主机的那个端口上监听 TCP 连接，对资源的请求的 URI 为绝对路径(5.1.2 节)。
无论什么时候，都应该避免在 URL 里使用 IP 地址(参看 RFC 1900 [24])。
如果绝对地址没有出现在 URL 里，它用作对资源的请求的 URI 时必须作为"/"给出。
如果代理收到主机名不是一个资格充分域名的，它可以为收到主机名加上域名。
如果代理收到一个资格充分的域名，一定不能改变主机名。

[页码 19]

3.2.3 URI 比较

当比较两个 URI 是否匹配时，客户应该对整个 URI 进行区分大小写，以八字节为单元的比较。以下情况例外：

1）一个为空或未给定的端口等同于那个 URI 索引里的默认端口;
2）主机名的比较必须是不区分大小写的;
3）主题名（“HTTP”）的比较必须是不区分大小写的;
4）一个空绝对路径等同于绝对路径"/"。

除了"保留"或"危险"集里的字符(参见 RFC 2396 [42])，字符等同于它们的""%" HEX

HEX"编码.

例如,以下三个 URI 是等同的:

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

3.3 日期/时间格式

3.3.1 完整日期

历史上的 HTTP 应用一直允许三种不同的表示日期/时间戳的格式:

```
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
```

第一种格式是作为 Internet 标准提出来的,它表示一个由 RFC 1123 [8](RFC 822[9]的升级版本)定义的固定长度的子集。第二种格式使用比较普遍，但是基于废弃的 RFC 850 [12]，需要(应该)用四位数表示年份。对日期值进行语法分析的 HTTP/1.1 客户和服务器必须接受所有三种格式(为了同 HTTP/1.0 兼容)，虽然它们必须只产生 RFC 1123 格式以在报头字段里表示 HTTP 日期值。（在小节 19.3 中可以看到更多信息）

注:鼓励日期值的接收者在接受可能由非 HTTP 应用发来的日期值时要健壮，因为这种非 HTTP 应用有时是通过服务代理/网关向 SMTP 和 NNTP 检索或发送消息的。

所有的 HTTP 日期/时间戳都必须毫无例外的以格林威治平均时间(GMT)表示。这是为了 HTTP，GMT 完全等同于 UTC(协调世界时间)。这在前两种形式里用三个字母的时区缩写-GMT 的蕴含来表示，并且读取 ASC 时间格式时必须先被假定。HTTP 日期区分大小写，除了在语法中作为 SP 特别包括的 LWS 外，一定不能包括额外的 LWS。

```
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 对日期/时间戳格式的请求仅仅应用在协议流里。客户和服务器不必为了用户表示，请求记录及其他而使用这些格式.

3.3.2 Delta 秒

一些 HTTP 报头字段收到消息后，允许以十进制整数秒表示的时间值。

```
delta-seconds  = 1*DIGIT
```

3.4 字符集

HTTP 使用的关于术语"字符集"的定义和 MIME 中所描述的一样.

[页码 21]

本文档中的术语"字符集"指一种用一个或更多表格将一个八字节序列转换成一个字符序列的方法。注意另一方向的无条件转换是不需要的，在这种转换里，并不是所有的字符都能在一个给定字符集里得到，并且字符集可能提供多个八进制序列表示一个特定字符。这个定义将允许各种字符编码方式，从简单的单表格映射如 US-ASCII 到复杂的表格交换方法如 ISO-2022 的技术里所使用的。然而，与 MIME 字符集名字相关联的定义必须充分说明从八字节变换到字符所实现的映射。特别的，使用外部轮廓信息来决定精确映射是不允许的.。

注:这里使用的术语"字符集"更一般的被称作一种"字符编码"。不过，既然 HTTP 和 MIME 使用同样的注册表，共用术语是很重要的。

HTTP 字符集用不区分大小写的标记表示。完全标记集合由 IANA 字符集注册表[19]

定义。

charset = token

尽管 HTTP 允许用任意标记作为字符集的值，任何在 IANA 字符集注册表里有预先确定值的标记必须表示该注册表定义的字符集。对那些 IANA 定义的字符集，应用应该限制使用字符集。

实现者应该注意 IETF 字符集的要求[38][41].

3.4.1 失踪字符集

一些 HTTP/1.0 软件将没有字符集参数的内容类型报头错误的理解为"接收者应该猜猜。"若发送者希望避免这种情况，可以包含一个字符集参数，即使字符集是 ISO-8859-1；当知道这样不会使接收者混淆以后，应该这样做。

不幸的是，一些旧的 HTTP/1.0 不能适当处理详细的字符集参数。HTTP/1.1 接收者必须重视发送者提供的字符集标注；当最初显示文档时，那些提供"猜"字符集服务的用户代理必须使用内容类型域中的字符集，如果它们支持那个字符集，而不是接收者的首选项。（参看 3.7.1 节）

[页码 22]

3.5 内容编码

内容编码值表示一种已经或可以应用于实体的编码变换。内容编码主要用来允许文档压缩，换句话说，有效的变换而不损失它的基本媒体类型的特性，也不丢失信息。经常地，实体以编码形式储存，直接传送，只能由接收者译码.

content-coding = token

所有内容编码值都是不区分大小写的。HTTP/1.1 在接收译码字段(14.3 节)和内容译码字段(14.11 节)的报头字段里使用内容编码值。尽管该值描述了内容编码，更重要的是它指出需要什么机制来解码。

互联网赋值机构(IANA)充当内容编码值标记的注册处。最初，注册表包含下列标记:

gzip（GUNzip 压缩文件）

一种由文件压缩程序"gzip"(GNU zip)---如 RFC 1952 所描述---生成的编码格式。这种格式是一种 32 位 CRC Lempel-Ziv 编码(LZ77)。

compress（压缩）

由通用 UNIX 文件压缩程序"compress"生成的编码格式。这种格式是一种具有可

适应性的 Lempel-Ziv-Welch 编码。对未来的编码来说，用程序名识别编码格式是不可取。在这里他们的用处是作为历史实践的代表而不是好的方案。为了同以前的 HTTP 实现相兼容，应用应该将"x-gzip"和"x-compress"分别等同于"gzip"和"compress"。

deflate（缩小）
RFC 1950 [31]定义的"zlib"格式与 RFC 1951 [29]描述的"deflate"压缩机制的组合。

[页码 23]

Identity（标识）
缺省(标识)编码；无论如何，不进行转化的应用。这种内容译码仅被用于接受译码报头，并且不能被用在内容编码报头。

新的内容译码值的标记应该注册；为了允许客户和服务器间的互用性，内容译码运算的规范需要实现一个可被公开利用并能独立实现的新值，并且与这节中内容译码定义的目的相一致。

3.6 传输编码

传输编码值被用来表示一个已经能够，或可能需要应用于一个实体的编码转化，为的是能够确保通过网络安全传输。这不同于内容译码，传输译码是消息的特性而不是原始实体的特性。

```
transfer-coding         = "chunked" | transfer-extension
transfer-extension      = token *( ";" parameter )
```

参数采用属性/值对的形式.

```
参数                = 属性 "=" 值
属性                = 标记
值                  = 标记 | 引用-串（quoted-string）
```

所有传输译码值是大小写不敏感的。HTTP/1.1 在 TE 报头字段(14.39 节)和传输译码报头字段(14.41 节)运用传输译码。

无论何时一个传输译码都被应用于一个消息体，传输译码的设置必须包括"大块"，除非消息被结束连接停止。当"大块"传输译码被应用时，它必须是应用于消息体的最后传输译码。并且对消息主体使用“大块”编码方式不能超过一次。这些规则让接收者确定消息的传输长度(4.4 节)。

传输译码与 MINE[7]的内容传输译码值相类似,它被设计为在 7BIT 的传输服务上安全传输二进制数据。不过，安全传输对纯 8 位传输协议有不同的焦点。在 HTTP 中，

消息体唯一不安全的特性是确定准确的主体长度这个难点(7.2.2 节)，或在共享传输上加密的要求。

[页码 24]

网络分配数字权威(IANA)担任了传输译码值标记注册处的角色。起初，注册包含如下标记:大块"chunked"(3.6.1 节)，"identity"身份(3.6.2 节)，"gzip"(3.5 节), "compress"(3.5 节)，和"deflate"(3.5 节).

新的传输编码值标记应和新的内容编码值标记以相同的方式注册(3.5 节)。

服务器接收到一个不能理解的传输译码实体时应返回 501(未实现)，并且关闭连接。服务器不能向 HTTP/1.0 客户发送传输译码。

3.6.1 成块传输编码（Chunked Transfer Coding）

成块编码更改消息主体，为的是将它以一系列大块的形式传送，每一个连同它自己尺寸的指示器，后面跟随一个可选的 trailer，里面包含着 entity-header 字段。这允许动态的生成将和必须的信息一起被传送的内容，必须的信息是为了让接收者确认他已经获得全部消息。

```
Chunked-Body    = *chunk
                  last-chunk
                  trailer
                  CRLF

chunk            = chunk-size [ chunk-extension ] CRLF
                  chunk-data CRLF
chunk-size      = 1*HEX
last-chunk      = 1*("0") [ chunk-extension ] CRLF

chunk-extension= *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
chunk-ext-name = token
chunk-ext-val  = token | quoted-string
chunk-data      = chunk-size(OCTET)
trailer         = *(entity-header CRLF)
```

大块尺寸域是用 16 进制表示大块尺寸的一串数字。成块编码以任一尺寸为 0 的大块结束，后面是 trailer，以一个空行终止.

trailer 允许发送者在消息末尾包含附加的 HTTP 报头字段。trailer 头字段可被应用于表示说明包含于 trailer 的 HTTP 报头字段 (14.40 节)

[页码 25]

一个服务器在应答中使用了”大块“传输译码时不能在任何报头字段使用 trailer，除非以下至少一条为发生:

a)请求包括一个 TE 报头字段，表明"trailer"在应答的转移译码中是可被接受的，就像 14.39 节中描述的那样。

b)服务器是应答的原始服务器。trailer 的域完全由可选的元数据构成，这个接收者可以在不接受这个元数据的情况下使用消息(在一个原始服务器可接受的方式中)。换句话说，原始服务器愿意接受 trailer 域可能会在通往客户的通道上被默默放弃的这种可能性。

当消息被一个 HTTP/1.1 服务代理接收，并且转发给一个 HTTP/1.0 接收者时，这种需要求防止了一个互用性的失误。它避免了一个依据协议将使在服务代理上必须安置一个可能无限大的缓冲器情形发生。

对一个大块主体进行解码处理的例子已在附录 19.4.6 中作过介绍。

所有 HTTP/1.1 应用程序必须能接受和解码"大块"传输译码，并且必须忽略它们不理解的大块扩展扩展名。

3.7 媒体类型

为了提供公开的，可扩展的数据输入和类型协商，HTTP 在 Content-Type(14.17 节)和 Accept(14.1 节)报头字段中运用网络媒体类型.

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

参数可能在属性/值的形式上遵循类型/亚类型。(如 3.6 节定义)

类型，子类型，和参数属性名称是大小写不敏感的。参数值敏感与否，取决于参数名称的意义。线性空白区(LWS)不能被用于类型和子类型之间，也不能用于一种属性和他的值之间。一个参数存在与否对媒体类型的处理可能有重要意义，这取决于它在媒体类型注册中的定义。