| Connection ; Section 14.10
| Date ; Section 14.18
| Pragma ; Section 14.32
| Trailer ; Section 14.40
| Transfer-Encoding ; Section 14.41
| Upgrade ; Section 14.42
| Via ; Section 14.45
| Warning ; Section 14.46

常用头域名能被扩展，但这要和协议版本的变化相结合。然而，如果通信里的所有参与者都认同新的或实践性的头域是常用头域，那么它们可能就被赋于常用头域的语意。不被识别的头域会被作为实体头（entity-header）头域来看待。

5 请求（Request）

一个请求消息是从客户端到服务器端的，在消息首行里包含方法，资源指示符，协议版本。

Request = Request-Line ; Section 5.1
*（（ general-header ; Section 4.5
| request-header ; Section 5.3
| entity-header ） CRLF） ; Section 7.1
CRLF
[message-body] ; Section 4.3

5.1 请求行 （Request-Line）

请求行（Request-Line）是以一个方法标记开始，后面跟随 Request-URI 和协议版本（HTTP-Version），最后以 CRLF 结束。元素是以 SP 字符分隔。除了最后的 CRLF，CR 或 LF 是不被允许的。

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

5.1.1 方法 （Method）

方法标记（token）指明了在被 Request-URI 指定的资源上执行的方法。这种方法是大小写敏感的。

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

Extension-method = token

资源所允许的方法由 Allow 头域指定（14.7 节）。响应的返回码总是通知客户某个方法对当前

资源是否是被允许的，因为被允许的方法能被动态的改变。如果服务器能理解某方法但此方法对请求资源不被允许的，那么源服务器应该返回 405 状态码（方法不允许）；如果源服务器不能识别或没有实现某个方法，那么服务器应返回 501 状态码（没有实现）。方法 GET 和 HEAD 必须被所有一般的服务器支持。所有其它的方法是可选的；然而，如果上面的方法都被实现，这些方法遵循的语意必须和第 9 章指定的相同。

5.1.2 请求 URL（Request-URI）

Request-URI 是一种通用资源标识符（ 3 .2 节），并且它用于指定请求的请求资源。

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

Request-URI 的四个选项依赖于请求的性质。星号 “*”意味着请求不能应用于一个特定的资源，只能应用于服务器本身，并且只能在方法不应用于一个资源的时候才被允许。举例如下

OPTIONS * HTTP/1.1

当向代理（proxy）提交请求时，绝对 URI（absoluteRUI）格式是不可缺少的。代理（proxy）可能会被要求再次转发请求或者从一个有效的缓存（cache）里构造响应去响应请求。注意：代理可能转发请求给另一个代理或直接给被 absoluteURI 指定的源服务器。为了避免循环请求，代理（proxy）必须能识别所有的服务器名字，包括任何别名，本地的变化值，数字 IP 地址。一个请求行（Request-Line）的例子如下：

G ET http://www.w3.org/pub/www/TheProject.html HTTP/1.1

为了未来 HTTP 版本的所有请求能迁移到 absoluteURI 地址，所有基于 H TTP/1.1 的服务器必须接受绝对 absoluteURI 形式的请求，即使 HTTP/1.1 客户端只为代理产生绝对 absoluteURI 请求。

authority 格式只被用于 CONNECT 方法（ 9 .9 节）.

Request-URI 大多数情况是被用于指定一个源服务器或网关（gateway）上的资源。这种情况下，URI 的绝对路径（abs_path，见 3.2.1 节）作为 Request-URI 被传输，并且 URI 网络位置（authority）必须在 Host 头域里指出。例如：客户希望直接从源服务器获取资源，这种情况下，它可能会建立一个 TCP 连接，此连接是特定于主机 “www.w3.org”的 80 端口的，这时会发送下面行：

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

接下来是请求的其他部分。注意绝对路径（absolute path）不能是空的；如果在原始 URI 里没有出现绝对路径，必须给出" / "（服务器的根）。

Request-URI 是以 3.2.1 节里指定的格式传输。如果 Request-URI 用"%HEX HEX"[42]编码，源服务器为了解析请求必须能对它进行解码。服务器接收到一个无效的 Request-URI 时必须以一个合适的状态码响应。

透明代理（proxy）不能重写接收到的 Request-URI 里的 “abs_path”部分，当它转发此请求到下一个内向服务器（inbound server，见 1.3 术语）时，但除了上面说的把一个空的 abs_path 用一个 ”/”代替之外。

注：不重写的规则是为了防止代理（proxy）改变请求的原意，因为存在源服务器利用非保留

（non-reserved）的 URI 字符为相反的目的 。实现者应该知道某些低于 HTTP/1.1 版本的代理（proxy）会重写 Request-URI。

5.2 请求资源的识别 （The Resource Identified by a Request）

请求资源的精确定位是由请求里的 Request-URI 和 Host 头域决定的。

如果源服务器不允许资源根据请求的不同主机来区分时，那么它可以会忽略 Host 头域的值，当它决定通过 HTTP/1.1 请求来识别资源的时候。（在 HTTP/1.1 里关于对 Host 的支持在 19.6.1.1 节描述了其他的要求）。

一个源服务器如果必须基于请求主机来区分资源（这是因为存在虚拟主机（virtual hosts）和虚拟主机名（vanity host names）），那么，对 HTTP/1.1 请求，它必须遵循下面的规则去决定请求的资源：

1． 如果 Request-URI 是绝对地址（absoluteURI），那么主机（host）是 Request-URI 的一部分。任何出现在请求里 Host 头域的值应当被忽略。

2． 假如 Request-URI 不是绝对地址（absoluteURI），并且请求包括一个 Host 头域，则主机（host）由该 Host 头域的值决定.

3． 假如由规则 1 或规则 2 定义的主机（host）对服务器来说是一个无效的主机（host），则应当以一个 400（坏请求）错误消息返回。

缺少 Host 头域的 HTTP/1.0 请求的接收者可能会尝试去利用启发式的规则（例如：检查 URI 路径是否是特定某个主机）去决定被请求的真正资源。

5.3 请求头域 （Request Header Fields）

请求头域允许客户端传递请求的附加信息和客户端自己的附加信息给服务器。这些头域作为请求的修饰，这和程序语言方法调用的参数语义是等价的。

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

| User-Agent ;14.43 节

请求头域的名字是能被扩展，但只能随协议版本改变而被扩展。然而新的或实践性的头域可以给定请求头域语义，如果所有通信方都把它看作请求头域。不能识别的头域会被看作实体头域（entity-header）。

6 响应 （Response）

接收和解析一个请求消息后，服务器发出一个 HTTP 响应消息。

```
response   =Status-Line                    ;6.1 节
        *（（ general-header）                                ; 4.5 节
                    | response-header    ;6.2 节
                    | entity-header）CRLF）      ;7.1 节
                   CRLF
                   [ message-body ]       ;7.2 节
```

6.1 状态行 （Status-Line）

响应消息的第一行是状态行（stauts-Line），由协议版本以及数字状态码和相关的文本短语组成，各部分间用空格符隔开，除了最后的 CRLF 序列，中间不允许有 CR 或 LF。

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

6.1.1 状态码与原因短语 （Status Code and Reason Phrase）

Status-Code 元素是一个试图理解和满足请求的三位数字整数码，这些码的完整定义在第十章。原因短语（Reason-Phrase）是为了给出关于状态码的简单的文本描述。状态码用于控制，而原因短语（Reason-Phrase）是让用户便于阅读。客户端不需要检查和显示原因短语。

状态码的第一位数字定义响应类别。后两位数字没有任何分类角色。第一位数字有五种值：

-1xx ： 报告的 -请求被接收到，继续处理
-2xx ： 成功 - 被成功地接收（received），理解（understood），接受（accepted）的动作 。
-3xx ： 重发 - 为了完成请求必须采取进一步的动作。
-4xx ： 客户端出错 - 请求包括错的语法或不能被满足。
-5xx ： 服务器出错 - 服务器无法完成显然有效的请求。

下面列举了为 HTTP/1.1 定义的态码值，和对应的原因短语（Reason-Phrase）的例子。原因短语在这里例举只是建议性的----它们也许被一个局部的等价体代替而不会影响此协议的语义。

```
Status-Code =
        "100" ; 10.1.1 节: 继续
       |"101"  ; 10.1.2 节: 转换协议
       |"200"  ; 10.2.1 节: OK
       |"201"  ; 10.2.2 节:  已创建
        |"202"  ; 10.2.3 节:  接受
       |"203"  ; 10.2.4 节: 非权威信息
```

|"204" ; 10.2.5 节: 无内容
|"205" ; 10.2.6 节: 重置内容
|"206" ; 10.2.7 节: 部分内容
|"300" ; 10.3.1 节: 多个选择
|"301" ; 10.3.2 节: 永久移动
|"302" ; 10.3.3 节: 发现
|"303" ; 10.3.4 节: 见其它
|"304" ; 10.3.5 节: 没有被改变
|"305" ; 10.3.6 节: 使用代理
|"307" ; 10.3.8 节 临时重发
|"400" ; 10.4.1 节: 坏请求
|"401" ; 10.4.2 节: 未授权的
|"402" ; 10.4.3 节: 必要的支付
|"403" ; 10.4.4 节: 禁用
|"404" ; 10.4.5 节: 没有找到
|"405" ; 10.4.6 节: 方式不被允许
|"406" ; 10.4.7 节: 不接受的
|"407" ; 10.4.8 节: 需要代理验证
|"408" ; 10.4.9 节: 请求超时
|"409" ; 10.4.10 节; 冲突
|"410" ; 10.4.11 节: 不存在
|"411" ; 10.4.12 节: 长度必需
|"412" ; 10.4.13 节；先决条件失败
|"413" ; 10.4.14 节: 请求实体太大
|"414" ; 10.4.15 节; 请求 URI 太大
|"415" ; 10.4.16 节: 不被支持的媒体类型
|"416" ; 10.4.17 节: 请求的范围不满足
|"417" ; 10.4.18 节: 期望失败
|"500" ; 10.5.1 节: 服务器内部错误
|"501" ; 10.5.2 节: 不能实现
|"502" ; 10.5.3 节: 坏网关
|"503" ; 10.5.4 节: 服务不能获得
|"504" ; 10.5.5 节: 网关超时
|"505" ; 10.5.6 节: HTTP 版本不支持
|扩展码

extension-code =3DIGIT
Reason-Phrase = *<TEXT,excluding CR,LF>

HTTP 状态码是可扩展的。HTTP 应用程序不需要理解所有已注册状态码的含义，尽管那样的理解是很希望的。但是，应用程序必须了解由第一位数字指定的状态码的类别，任何未被识别的响应应被看作是那个类别的 x00 状态码，未被识别的响应不能被缓存除外。例如，如果客户端收到一个未被识别的状态码 431，则可以安全的认为请求有错，并且它会对待此响应就像它接收了一个状态码是 400 的响应。在这种情况下，用户代理（user agent）应当把响应的实体展现给用户，因为实体有可能包括人类可读的信息，这些信息也许能解释非正常状态的原因。

6 .2 响应头域 （Response Header Fields）

响应头域允许服务器传送响应的附加信息，这些信息不能放在状态行（Status-Line）里.。这些头域给出有关服务器的信息以及请求 URI（Request-URI）指定资源的更进一步访问信息。

```
response-header  =  Accept-Ranges          ; 14.5 节
                   |Age          ; 14.6 节
                   |Etag           ; 14.19 节
                   |Location          ; 14.30 节
                   |Proxy-Autenticate     ; 14.33 节
                   |Retry-After     ; 14.37 节
                   |Server      ; 14.38 节
                   |Vary        ; 14.44 节
                   |WWW-Authenticate  ; 14.47 节
```

响应头域的名字能依赖于协议版本的变化而扩展。然而，新的或者实践性的头域可能会给予响应头域的语义如果通信所有成员都能识别他们并把他们看作响应头域。不被识别的头域被看作实体头域。

7 实体（Entity）

如果不被请求方法或响应状态码所限制，请求和响应消息都可以传输实体。 实体包括实体头域（entity-header）与实体主体（entity-body），而有些响应只包括实体头域（entity-header）。

在本节中的发送者和接收者是否是客户端或服务器，这依赖于谁发送或谁接收此实体。

7.1 实体头域（Entity Header Fields）

实体（entity-header）头域定义了关于实体主体的的元信息，或在无主体的情况下定义了请求的资源的元信息。有些元信息是可选的；一些是必须的。

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

extension-header = message-header
```

扩展头机制允许在不改变协议的前提下定义额外的实体头域，但不保证这些域在接收端能够被识别。未被识别的头域应当被接收者忽略，且必须被透明代理（transparent proxy）转发。

7.2 实体主体（Entity Body）

由 HTTP 请求或响应发送的实体主体（如果存在的话）的格式与编码方式应由实体的头域决定。

```
Entity-body= *OCTET
```

才有用；它干不了任何事情除了允许客户端测试服务器的能力。例如：它能被用来测试代理是否遵循 HTTP/1.1。

如果请求 URI 不是一个星号（"*"）,，OPTIONS 请求只能应用于请求 URI 指定资源的选项。

200 响应应该包含任何指明选项性质的头域，这些选项性质由服务器实现并且只适合那个请求的资源（例如，Allow 头域），但也可能包一些扩展的在此规范里没有定义的头域。如果有响应主体的话也应该包含一些通信选项的信息。这个响应主体的格式并没有在此规范里定义，但是可能会在以后的 HTTP 里定义。内容协商可能被用于选择合适的响应格式。如果没有响应主体包含，响应就应该包含一个值为“0”的 Content-Length 头域。

Max-Forwards 请求头域可能会被用于针对请求链中特定的代理。当代理接收到一个 OPTIONS 请求，且此请求的 URI 为 absoluteURI，并且此请求是可以被转发的，那么代理必须要检测 Max-Forwards 头域。如果 Max-Forwards 头域的值为“0”，那么此代理不能转发此消息；而是代理应该以它自己的通信选项响应。如果 Max-Forwards 头域是比 0 大的整数值，那么代理必须递减此值当它转发此请求时。如果没有 Max-Forwards 头域出现在请求里，那么代理转发此请求时不能包含 Max-Forwards 头域。

9.3 GET

GET 方法意思是获取被请求 URI（Request-URI）指定的信息（以实体的格式）。如果请求 URI 涉及到一个数据生成过程，那么这个过程生成的数据应该被作为实体在响应中返回而不是过程的源文本，除非源文本恰好是过程的输出。

如果请求消息包含 If-Modified-Since,，If-Unmodified-Since，If-Match，If-None-Match 或者 If-Range 头域，GET 的语义将变成“条件（conditionall） GET”。一个条件 GET 方法会请求满足条件头域的实体。条件 GET 方法的目的是为了减少不必要的网络使用，这通过允许利用缓存里仍然保鲜的实体而不用多次请求或传输客户端已经拥有的实体来实现的。.

如果请求方法包含一个 Range 头域，那么 GET 方法就变成“部分 Get”（partial GET）方法。一个部分 GET 会请求实体的一部分，这在 14.35 节里描述了。 部分 GET 方法的目的是为了减少不必要的网络使用，可以允许客户端从服务器获取实体的部分数据，而不需要获取客户端本地已经拥有的部分实体数据。

GET 请求的响应是可缓存的（cacheable）如果此响应满足第 13 节 HTTP 缓存的要求。

看 15.1.3 节关于 GET 请求用于表单时安全考虑。

9.4 HEAD

HEAD 方法和 GET 方法一致，除了服务器不能在响应里返回消息主体。HEAD 请求响应里 HTTP 头域里的元信息（译注：元信息就是头域信息）应该和 GET 请求响应里的元信息一致。此方法被用来获取请求实体的元信息而不需要传输实体主体（entity-body）。此方法经常被用来测试超文本链接的有效性，可访问性，和最近的改变。.

HEAD 请求的响应是可缓存的，因为响应里的信息可能被缓存用于更新以前那个资源对应缓存的实体.。如果出现一个新的域值指明缓存的实体和当前源服务器上的实体有所不同（可能因为 Content-Length，Content-MD5，ETag 或 Last-Modified 值的改变），那么缓存（cache）必须认为缓存项是过时的（stale）。

9.5 POST

POST 方法被用于请求源服务器接受请求中的实体作为请求资源的一个新的从属物。 POST 被设计涵盖下面的功能。

--已存在的资源的注释；

--发布消息给一个布告板，新闻组，邮件列表，或者相似的文章组。

--提供一个数据块，如提交一个表单给一个数据处理过程。

--通过追加操作来扩展数据库。

POST 方法的实际功能是由服务器决定的，并且经常依赖于请求 URI（Request-URI）。POST 提交的实体是请求 URI 的从属物，就好像一个文件从属于一个目录，一篇新闻文章从属于一个新闻组，或者一条记录从属于一个数据库。

POST 方法执行的动作可能不会对请求 URI 所指的资源起作用。在这种情况下，200（成功）或者 204（没有内容）将是适合的响应状态，这依赖于响应是否包含一个描述结果的实体。

如果资源被源服务器创建，响应应该是 201（Created）并且包含一个实体，此实体描述了请求的状态。并且引用了这个新资源和一个 Location 头域（见 14.30 节）。

POST 方法的响应是不可缓存的。除非响应里有合适的 Cache-Control 或者 Expires 头域。然而，303（见其他）响应能被用户代理利用去获得可缓存的响应。

POST 请求必须遵循 8.2 节里指明的消息传送的要求。

参见 15.1.3 节关于安全性的考虑.

9.6 PUT

PUT 方法请求服务器去把请求里的实体存储在请求 URI（Request-URI）标识下。如果请求 URI（Request-URI）指定的的资源已经在源服务器上存在，那么此请求里的实体应该被当作是源服务器关于此 URI 所指定资源实体的最新修改版本。如果请求 URI（Request-URI）指定的资源不存在，并且此 URI 被用户代理定义为一个新资源，那么源服务器就应该根据请求里的实体创建一个此 URI 所标识下的资源。如果一个新的资源被创建了，源服务器必须能向用户代理（user agent） 发送 201（已创建）响应。如果已存在的资源被改变了，那么源服务器应该发送 200（Ok）或者 204（无内容）响应。如果资源不能根据请求 URI 创建或者改变，一个合适的错误响应应该给出以反应问题的性质。实体的接收者不能忽略任何它不理解和不能实现的 Content-*（如：Content-Range）头域，并且必须返回 501（没有被实现）响应。

如果请求穿过一个缓存（cache），并且此请求 URI（Request-URI）指示了一个或多个当前缓存的实体，那么这些实体应该被看作是旧的。PUT 方法的响应是不可缓存的。

POST 方法和 PUT 方法请求最根本的区别是请求 URI（Request-URI）的含义不同。POST 请求里的 URI 指示一个能处理请求实体的资源（译注：此资源可能是一段程序，如 jsp 里的 servlet） 。此资源可能是一个数据接收过程，一个网关（gateway，译注：网关和代理的区别是：网关可以进行协议转换，而代理不能，只是起代理的作用，比如缓存服务器其实就是一个代理），或者一个单独接收注释的实体。对比而言，PUT 方法请求里的 URI 标识请求里封装的

实体一一用户代理知道 URI 意指什么，并且服务器不能把此请求应用于其它资源（resource）。如果服务器期望请求被应用于一个不同的 URI，那么它必须发送 301（永久移动）响应；用户代理可以自己决定是否重定向请求。

一个单独的资源可能会被许多不同的 URI 指定。如：一篇文章可能会有一个 URI 指定当前版本，而这个 URI 区别于这篇文章其它特殊版本的 URI。这种情况下，对一个通用 URI 的 PUT 请求可能会导致其资源的其它 URI 请求被源服务器重定义。

HTTP/1.1 没有定义 PUT 方法对源服务器的状态影响。

PUT 请求必须遵循 8.2 节中的消息传送的要求。

除非特别指出，PUT 方法请求里的实体头域应该被用于资源的创建或修改。

9.7 DELETE（删除）

DELETE 方法请求源服务器删除请求 URI 指定的资源。此方法可能会在源服务器上被人为的干涉（或通过其他方法）。客户端不能保证此操作能被执行，即使源服务器返回成功状态码。然而，服务器不应该指明成功除非它打算删除资源或把此资源移到一个不可访问的位置。

如果响应里包含描述成功的实体，响应应该是 200（OK）；如果 DELETE 动作还没有执行，应该以 202（已接受）响应；如果 DELETE 请求方法已经执行但响应不包含实体，那么应该以 204（无内容）响应。

如果请求穿过缓存，并且请求 URI（Request-URI）指定了一个或多个缓存当前实体，那么这些缓存项应该被认为是旧的。DELETE 方法的响应是不能被缓存的。

9.8 TRACE

TRACE方法被用于激发一个远程的，应用层的请求消息回路（译注：TRACE方法让客户端测试到服务器的网络通路，回路的意思如发送一个请返回一个响应，这就是一个请求响应回路，）。最后的接收者也许是源服务器，也许是接收到包含Max-Forwards头域值为0请求的代理或网关。TRACE请求不能包含一个实体。

TRACE方法允许客户端去了解数据被请求链的另一端接收的情况，并且利用那些数据信息去测试或诊断。Via头域值（见14.45）有特殊的用途，因为它可以作为请求链的跟踪信息。利用Max-Forwards头域允许客户端限制请求链的长度，这是非常有用的，因为可以利用此去测试代理链在无限循环里转发消息。

如果请求是有效的，响应应该在实体主体里包含整个请求消息，并且响应应该包含一个Content-Type头域值为”message/http”的头域。此方法的响应不能被缓存。

9.9 CONNECT（连接）

HTTP1.1 协议规范保留了 CONNECT 方法，此方法是为了能用于能动态切换到隧道的代理（(如 SSL tunneling [44]).）。