# **2.1**

2.1. Client/Server Messaging

HTTP is a stateless request/response protocol that operates by

exchanging messages (Section 3) across a reliable transport- or

session-layer "connection" (Section 6). An HTTP "client" is a

program that establishes a connection to a server for the purpose of

sending one or more HTTP requests. An HTTP "server" is a program

that accepts connections in order to service HTTP requests by sending

HTTP responses.

The terms "client" and "server" refer only to the roles that these

programs perform for a particular connection. The same program might

act as a client on some connections and a server on others. The term

"user agent" refers to any of the various client programs that

initiate a request, including (but not limited to) browsers, spiders

(web-based robots), command-line tools, custom applications, and

mobile apps. The term "origin server" refers to the program that can

originate authoritative responses for a given target resource. The

terms "sender" and "recipient" refer to any implementation that sends

or receives a given message, respectively.

HTTP relies upon the Uniform Resource Identifier (URI) standard

[RFC3986] to indicate the target resource (Section 5.1) and

relationships between resources. Messages are passed in a format

similar to that used by Internet mail [RFC5322] and the Multipurpose

Internet Mail Extensions (MIME) [RFC2045] (see Appendix A of

[RFC7231] for the differences between HTTP and MIME messages).

Most HTTP communication consists of a retrieval request (GET) for a

representation of some resource identified by a URI. In the simplest

case, this might be accomplished via a single bidirectional

connection (===) between the user agent (UA) and the origin

server (O).

request >

UA ======================================= O

< response

A client sends an HTTP request to a server in the form of a request

message, beginning with a request-line that includes a method, URI,

and protocol version (Section 3.1.1), followed by header fields

containing request modifiers, client information, and representation

metadata (Section 3.2), an empty line to indicate the end of the

header section, and finally a message body containing the payload

body (if any, Section 3.3).

A server responds to a client's request by sending one or more HTTP

response messages, each beginning with a status line that includes

the protocol version, a success or error code, and textual reason

phrase (Section 3.1.2), possibly followed by header fields containing

server information, resource metadata, and representation metadata

(Section 3.2), an empty line to indicate the end of the header

section, and finally a message body containing the payload body (if

any, Section 3.3).

A connection might be used for multiple request/response exchanges,

as defined in Section 6.3.

The following example illustrates a typical message exchange for a

GET request (Section 4.3.1 of [RFC7231]) on the URI

"http://www.example.com/hello.txt":

Client request:

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

Server response:

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.

2.1. 客户端/服务器消息传递

HTTP 是一个无状态的请求/响应协议，通过可靠的传输层或会话层连接（第 6 节）交换消息。一个 HTTP “客户端”是一个用于发送一个或多个 HTTP 请求的程序，它与服务器建立连接。一个 HTTP “服务器”是一个接受连接以处理 HTTP 请求并发送 HTTP 响应的程序。

“客户端”和“服务器”这两个术语只是指这些程序在特定连接中执行的角色。同一个程序可能在某些连接中充当客户端，在其他连接中充当服务器。“用户代理”一词是指发起请求的各种客户端程序，包括（但不限于）浏览器、网络爬虫（基于网络的机器人）、命令行工具、定制应用程序和移动应用程序。“源服务器”一词是指可以为给定目标资源生成权威响应的程序。“发送方”和“接收方”这两个术语分别指发送或接收给定消息的任何实现。

HTTP 依赖于统一资源标识符（URI）标准 [RFC3986] 来指示目标资源（第 5.1 节）和资源之间的关系。消息以类似于 Internet 邮件 [RFC5322] 和多用途 Internet 邮件扩展（MIME）[RFC2045] 的格式传递（请参阅 [RFC7231] 的附录 A 以了解 HTTP 和 MIME 消息之间的区别）。

大多数 HTTP 通信都是对某个 URI 标识的资源的检索请求（GET）。在最简单的情况下，这可能通过用户代理（UA）和源服务器（O）之间的单个双向连接（===）完成。

request >

UA ======================================= O

< response

客户端以请求消息的形式向服务器发送 HTTP 请求，请求消息以包含方法、URI 和协议版本的请求行（第 3.1.1 节）开始，然后是包含请求修饰符、客户端信息和表示元数据的头字段（第 3.2 节），一个空行表示头部部分的结束，最后是包含负载体的消息体（如果有的话，第 3.3 节）。

服务器通过发送一个或多个 HTTP 响应消息来响应客户端的请求，每个响应消息以包含协议版本、成功或错误代码和文本原因短语的状态行（第 3.1.2 节）开始，可能后跟包含服务器信息、资源元数据和表示元数据的头字段（第 3.2 节），一个空行表示头部部分的结束，最后是包含负载体的消息体（如果有的话，第 3.3 节）。

一个连接可以用于多个请求/响应交换，如第 6.3 节所定义。

以下示例说明了对 URI "http://www.example.com/hello.txt" 进行 GET 请求（[RFC7231] 的第 4.3.1 节）的典型消息交换：

客户端请求：

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

服务器响应：

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.

# **2.2**

2.2. Implementation Diversity

When considering the design of HTTP, it is easy to fall into a trap

of thinking that all user agents are general-purpose browsers and all

origin servers are large public websites. That is not the case in

practice. Common HTTP user agents include household appliances,

stereos, scales, firmware update scripts, command-line programs,

mobile apps, and communication devices in a multitude of shapes and

sizes. Likewise, common HTTP origin servers include home automation

units, configurable networking components, office machines,

autonomous robots, news feeds, traffic cameras, ad selectors, and

video-delivery platforms.

The term "user agent" does not imply that there is a human user

directly interacting with the software agent at the time of a

request. In many cases, a user agent is installed or configured to

run in the background and save its results for later inspection (or

save only a subset of those results that might be interesting or

erroneous). Spiders, for example, are typically given a start URI

and configured to follow certain behavior while crawling the Web as a

hypertext graph.

The implementation diversity of HTTP means that not all user agents

can make interactive suggestions to their user or provide adequate

warning for security or privacy concerns. In the few cases where

this specification requires reporting of errors to the user, it is

acceptable for such reporting to only be observable in an error

console or log file. Likewise, requirements that an automated action

be confirmed by the user before proceeding might be met via advance

configuration choices, run-time options, or simple avoidance of the

unsafe action; confirmation does not imply any specific user

interface or interruption of normal processing if the user has

already made that choice.

2.2. 实现多样性

在考虑设计 HTTP 时，很容易陷入一个思维误区，即所有用户代理都是通用的浏览器，所有源服务器都是大型的公共网站。但实际情况并非如此。常见的 HTTP 用户代理包括家用电器、音响、体重秤、固件更新脚本、命令行程序、移动应用程序和各种形状和大小的通信设备。同样，常见的 HTTP 源服务器包括家庭自动化设备、可配置的网络组件、办公设备、自主机器人、新闻源、交通摄像头、广告选择器和视频传输平台。

“用户代理”一词并不意味着在请求时有一个人类用户直接与软件代理进行交互。在许多情况下，用户代理被安装或配置为在后台运行并保存其结果以供以后检查（或仅保存可能有趣或错误的结果的子集）。例如，网络爬虫通常会被给定一个起始 URI 并配置为在爬行 Web 时遵循某些行为，形成一个超文本图。

HTTP 的实现多样性意味着并非所有用户代理都能向其用户提供交互式建议或提供足够的安全或隐私方面的警告。在少数情况下，本规范要求向用户报告错误，可接受的报告方式可以只在错误控制台或日志文件中可见。同样，要求在进行自动操作之前由用户确认的情况可能通过提前配置选择、运行时选项或简单地避免不安全操作来满足；确认并不意味着任何特定的用户界面或正常处理的中断，如果用户已经做出了选择。

# **2.7**

2.7. Uniform Resource Identifiers

Uniform Resource Identifiers (URIs) [RFC3986] are used throughout

HTTP as the means for identifying resources (Section 2 of [RFC7231]).

URI references are used to target requests, indicate redirects, and

define relationships.

The definitions of "URI-reference", "absolute-URI", "relative-part",

"scheme", "authority", "port", "host", "path-abempty", "segment",

"query", and "fragment" are adopted from the URI generic syntax. An

"absolute-path" rule is defined for protocol elements that can

contain a non-empty path component. (This rule differs slightly from

the path-abempty rule of RFC 3986, which allows for an empty path to

be used in references, and path-absolute rule, which does not allow

paths that begin with "//".) A "partial-URI" rule is defined for

protocol elements that can contain a relative URI but not a fragment

component.

URI-reference = <URI-reference, see [RFC3986], Section 4.1>

absolute-URI = <absolute-URI, see [RFC3986], Section 4.3>

relative-part = <relative-part, see [RFC3986], Section 4.2>

scheme = <scheme, see [RFC3986], Section 3.1>

authority = <authority, see [RFC3986], Section 3.2>

uri-host = <host, see [RFC3986], Section 3.2.2>

port = <port, see [RFC3986], Section 3.2.3>

path-abempty = <path-abempty, see [RFC3986], Section 3.3>

segment = <segment, see [RFC3986], Section 3.3>

query = <query, see [RFC3986], Section 3.4>

fragment = <fragment, see [RFC3986], Section 3.5>

absolute-path = 1\*( "/" segment )

partial-URI = relative-part [ "?" query ]

Each protocol element in HTTP that allows a URI reference will

indicate in its ABNF production whether the element allows any form

of reference (URI-reference), only a URI in absolute form

(absolute-URI), only the path and optional query components, or some

combination of the above. Unless otherwise indicated, URI references

are parsed relative to the effective request URI (Section 5.5).

2.7. 统一资源标识符

统一资源标识符（URIs）[RFC3986] 在整个 HTTP 中被用作标识资源的手段（见[RFC7231]的第 2 节）。URI 引用用于定位请求目标、指示重定向并定义关系。

"URI-reference"、“absolute-URI"、“relative-part"、“scheme"、“authority"、“port"、“host"、“path-abempty"、“segment"、“query"和“fragment" 的定义均采用了 URI 通用语法。为可以包含非空路径组件的协议元素定义了“absolute-path”规则。（此规则与 RFC 3986 中的 path-abempty 规则略有不同，后者允许在引用中使用空路径，而 path-absolute 规则则不允许以“//”开头的路径。）为可以包含相对 URI 但不包含片段组件的协议元素定义了“partial-URI”规则。

URI 引用 = <URI 引用，参见[RFC3986]第 4.1 节>

绝对 URI = <绝对 URI，参见[RFC3986]第 4.3 节>

相对部分 = <相对部分，参见[RFC3986]第 4.2 节>

方案 = <方案，参见[RFC3986]第 3.1 节>

权威 = <权威，参见[RFC3986]第 3.2 节>

URI 主机 = <主机，参见[RFC3986]第 3.2.2 节>

端口 = <端口，参见[RFC3986]第 3.2.3 节>

path-abempty = <path-abempty，参见[RFC3986]第 3.3 节>

段 = <段，参见[RFC3986]第 3.3 节>

查询 = <查询，参见[RFC3986]第 3.4 节>

片段 = <片段，参见[RFC3986]第 3.5 节>

绝对路径 = 1\*( "/" 段 )

部分 URI = 相对部分 [ "?" 查询 ]

HTTP 中的每个允许 URI 引用的协议元素将在其 ABNF 产生中指示该元素是否允许任何形式的引用（URI 引用）、仅绝对形式的 URI（绝对 URI）、仅路径和可选查询组件，或以上述任何形式的组合。除非另有说明，否则 URI 引用将相对于有效的请求 URI（第 5.5 节）进行解析。

## **2.7.1**

2.7.1. http URI Scheme

The "http" URI scheme is hereby defined for the purpose of minting

identifiers according to their association with the hierarchical

namespace governed by a potential HTTP origin server listening for

TCP ([RFC0793]) connections on a given port.

http-URI = "http:" "//" authority path-abempty [ "?" query ]

[ "#" fragment ]

The origin server for an "http" URI is identified by the authority

component, which includes a host identifier and optional TCP port

([RFC3986], Section 3.2.2). The hierarchical path component and

optional query component serve as an identifier for a potential

target resource within that origin server's name space. The optional

fragment component allows for indirect identification of a secondary

resource, independent of the URI scheme, as defined in Section 3.5 of

[RFC3986].

A sender MUST NOT generate an "http" URI with an empty host

identifier. A recipient that processes such a URI reference MUST

reject it as invalid.

If the host identifier is provided as an IP address, the origin

server is the listener (if any) on the indicated TCP port at that IP

address. If host is a registered name, the registered name is an

indirect identifier for use with a name resolution service, such as

DNS, to find an address for that origin server. If the port

subcomponent is empty or not given, TCP port 80 (the reserved port

for WWW services) is the default.

Note that the presence of a URI with a given authority component does

not imply that there is always an HTTP server listening for

connections on that host and port. Anyone can mint a URI. What the

authority component determines is who has the right to respond

authoritatively to requests that target the identified resource. The

delegated nature of registered names and IP addresses creates a

federated namespace, based on control over the indicated host and

port, whether or not an HTTP server is present. See Section 9.1 for

security considerations related to establishing authority.

When an "http" URI is used within a context that calls for access to

the indicated resource, a client MAY attempt access by resolving the

host to an IP address, establishing a TCP connection to that address

on the indicated port, and sending an HTTP request message

(Section 3) containing the URI's identifying data (Section 5) to the

server. If the server responds to that request with a non-interim

HTTP response message, as described in Section 6 of [RFC7231], then

that response is considered an authoritative answer to the client's

request.

Although HTTP is independent of the transport protocol, the "http"

scheme is specific to TCP-based services because the name delegation

process depends on TCP for establishing authority. An HTTP service

based on some other underlying connection protocol would presumably

be identified using a different URI scheme, just as the "https"

scheme (below) is used for resources that require an end-to-end

secured connection. Other protocols might also be used to provide

access to "http" identified resources -- it is only the authoritative

interface that is specific to TCP.

The URI generic syntax for authority also includes a deprecated

userinfo subcomponent ([RFC3986], Section 3.2.1) for including user

authentication information in the URI. Some implementations make use

of the userinfo component for internal configuration of

authentication information, such as within command invocation

options, configuration files, or bookmark lists, even though such

usage might expose a user identifier or password. A sender MUST NOT

generate the userinfo subcomponent (and its "@" delimiter) when an

"http" URI reference is generated within a message as a request

target or header field value. Before making use of an "http" URI

reference received from an untrusted source, a recipient SHOULD parse

for userinfo and treat its presence as an error; it is likely being

used to obscure the authority for the sake of phishing attacks.

2.7.1. http URI Scheme

“http” URI方案特此定义，用于根据潜在的HTTP源服务器与TCP（[RFC0793]）在特定端口上监听连接的关联来生成标识符。

http-URI = "http:" "//" authority path-abempty [ "?" query ] [ "#" fragment ]

“http” URI的原始服务器由权限组件标识，其中包括主机标识符和可选的TCP端口（[RFC3986]，第3.2.2节）。层次化路径组件和可选的查询组件作为标识符，用于标识该原始服务器名称空间内的潜在目标资源。可选的片段组件允许间接标识辅助资源，与URI方案无关，如[RFC3986]的第3.5节中定义。

发送者不得生成带有空主机标识符的“http” URI。处理此类URI引用的接收者必须将其拒绝为无效。

如果主机标识符以IP地址形式提供，则原始服务器是指示的IP地址上的侦听器（如果有）。如果主机是注册名称，则注册名称是用于使用名称解析服务（例如DNS）查找该原始服务器地址的间接标识符。如果端口子组件为空或未给出，则TCP端口80（WWW服务的保留端口）是默认端口。

请注意，具有给定权限组件的URI的存在并不意味着始终有HTTP服务器侦听该主机和端口上的连接。任何人都可以创建URI。权限组件确定的是谁有权对目标资源作出权威回应。注册名称和IP地址的委托性质创建了一个基于对指定主机和端口的控制的联合名称空间，无论是否存在HTTP服务器。有关建立权限的安全性考虑，请参见第9.1节。

在需要访问所指示资源的上下文中使用“http” URI时，客户端可以尝试通过将主机解析为IP地址，在指定端口上建立与该地址的TCP连接，并发送包含URI的标识数据（第5节）的HTTP请求消息（第3节）到服务器。如果服务器以HTTP响应消息回应该请求（如[RFC7231]的第6节所述），则该响应被视为对客户端请求的权威答复。

尽管HTTP独立于传输协议，但“http”方案特定于基于TCP的服务，因为名称委托过程依赖于TCP以建立权限。基于其他底层连接协议的HTTP服务可能使用不同的URI方案来标识，就像“https”方案（下文）用于需要端到端安全连接的资源一样。其他协议也可能用于提供对“http”标识的资源的访问——只有授权接口特定于TCP。

权限的URI通用语法还包括一个已弃用的用户信息子组件（[RFC3986]，第3.2.1节），用于在URI中包含用户认证信息。一些实现利用用户信息组件来内部配置认证信息，例如在命令调用选项、配置文件或书签列表中，尽管这种用法可能会暴露用户标识符或密码。当在消息中生成请求目标或标头字段值的“http” URI引用时，发送者不得生成用户信息子组件（及其“@”分隔符）。在使用从不受信任的源接收的“http” URI引用之前，接收者应该解析用户信息并将其存在视为错误；很可能正在为了钓鱼攻击而隐藏权限。

## **2.7.2**

2.7.2. https URI Scheme

The "https" URI scheme is hereby defined for the purpose of minting

identifiers according to their association with the hierarchical

namespace governed by a potential HTTP origin server listening to a

given TCP port for TLS-secured connections ([RFC5246]).

All of the requirements listed above for the "http" scheme are also

requirements for the "https" scheme, except that TCP port 443 is the

default if the port subcomponent is empty or not given, and the user

agent MUST ensure that its connection to the origin server is secured

through the use of strong encryption, end-to-end, prior to sending

the first HTTP request.

https-URI = "https:" "//" authority path-abempty [ "?" query ]

[ "#" fragment ]

Note that the "https" URI scheme depends on both TLS and TCP for

establishing authority. Resources made available via the "https"

scheme have no shared identity with the "http" scheme even if their

resource identifiers indicate the same authority (the same host

listening to the same TCP port). They are distinct namespaces and

are considered to be distinct origin servers. However, an extension

to HTTP that is defined to apply to entire host domains, such as the

Cookie protocol [RFC6265], can allow information set by one service

to impact communication with other services within a matching group

of host domains.

The process for authoritative access to an "https" identified

resource is defined in [RFC2818].

2.7.2. https URI Scheme

"https" URI方案是为了标识与TLS安全连接的潜在HTTP源服务器关联的层次命名空间而定义的。

所有上述针对"http"方案列出的要求也适用于"https"方案，唯一的区别在于，如果端口子组件为空或未给出，则TCP端口443是默认端口，而且用户代理必须确保与源服务器的连接在发送第一个HTTP请求之前通过端到端的强加密进行安全保护。

https-URI = "https:" "//" authority path-abempty [ "?" query ]

[ "#" fragment ]

请注意，"https" URI方案依赖于TLS和TCP来建立权限。通过"https"方案提供的资源与"http"方案没有共享的身份，即使它们的资源标识符指示相同的权限（相同的主机监听相同的TCP端口）。它们是不同的命名空间，并被认为是不同的源服务器。然而，一个定义应用于整个主机域的HTTP扩展，例如Cookie协议[RFC6265]，可以允许由一个服务设置的信息影响到与匹配组中其他服务之间的通信。

对于"https"标识的资源的权威访问过程在[RFC2818]中定义。

## **2.7.3**

2.7.3. http and https URI Normalization and Comparison

Since the "http" and "https" schemes conform to the URI generic

syntax, such URIs are normalized and compared according to the

algorithm defined in Section 6 of [RFC3986], using the defaults

described above for each scheme.

If the port is equal to the default port for a scheme, the normal

form is to omit the port subcomponent. When not being used in

absolute form as the request target of an OPTIONS request, an empty

path component is equivalent to an absolute path of "/", so the

normal form is to provide a path of "/" instead. The scheme and host

are case-insensitive and normally provided in lowercase; all other

components are compared in a case-sensitive manner. Characters other

than those in the "reserved" set are equivalent to their

percent-encoded octets: the normal form is to not encode them (see

Sections 2.1 and 2.2 of [RFC3986]).

For example, the following three URIs are equivalent:

http://example.com:80/~smith/home.html

http://EXAMPLE.com/%7Esmith/home.html

<http://EXAMPLE.com:/%7esmith/home.html>

2.7.3. http和https URI规范化与比较

由于"http"和"https"方案符合URI通用语法，因此这些URI根据[RFC3986]第6节中定义的算法进行规范化和比较，使用上述每个方案的默认设置。

如果端口等于方案的默认端口，则正常形式是省略端口子组件。当不以绝对形式作为OPTIONS请求的请求目标时，空路径组件等效于绝对路径“/”，因此正常形式是提供路径“/”。方案和主机是不区分大小写的，通常以小写形式提供；所有其他组件都以区分大小写的方式进行比较。除了“保留”集合中的字符外，其他字符都等同于它们的百分比编码八位字节：正常形式是不对它们进行编码（参见[RFC3986]的2.1和2.2节）。

例如，以下三个URI是等效的：

http://example.com:80/~smith/home.html

http://EXAMPLE.com/%7Esmith/home.html

<http://EXAMPLE.com:/%7esmith/home.html>

# **3.**

3. Message Format

All HTTP/1.1 messages consist of a start-line followed by a sequence

of octets in a format similar to the Internet Message Format

[RFC5322]: zero or more header fields (collectively referred to as

the "headers" or the "header section"), an empty line indicating the

end of the header section, and an optional message body.

HTTP-message = start-line

\*( header-field CRLF )

CRLF

[ message-body ]

The normal procedure for parsing an HTTP message is to read the

start-line into a structure, read each header field into a hash table

by field name until the empty line, and then use the parsed data to

determine if a message body is expected. If a message body has been

indicated, then it is read as a stream until an amount of octets

equal to the message body length is read or the connection is closed.

A recipient MUST parse an HTTP message as a sequence of octets in an

encoding that is a superset of US-ASCII [USASCII]. Parsing an HTTP

message as a stream of Unicode characters, without regard for the

specific encoding, creates security vulnerabilities due to the

varying ways that string processing libraries handle invalid

multibyte character sequences that contain the octet LF (%x0A).

String-based parsers can only be safely used within protocol elements

after the element has been extracted from the message, such as within

a header field-value after message parsing has delineated the

individual fields.

An HTTP message can be parsed as a stream for incremental processing

or forwarding downstream. However, recipients cannot rely on

incremental delivery of partial messages, since some implementations

will buffer or delay message forwarding for the sake of network

efficiency, security checks, or payload transformations.

A sender MUST NOT send whitespace between the start-line and the

first header field. A recipient that receives whitespace between the

start-line and the first header field MUST either reject the message

as invalid or consume each whitespace-preceded line without further

processing of it (i.e., ignore the entire line, along with any

subsequent lines preceded by whitespace, until a properly formed

header field is received or the header section is terminated).

The presence of such whitespace in a request might be an attempt to

trick a server into ignoring that field or processing the line after

it as a new request, either of which might result in a security

vulnerability if other implementations within the request chain

interpret the same message differently. Likewise, the presence of

such whitespace in a response might be ignored by some clients or

cause others to cease parsing.

3. 消息格式

所有的HTTP/1.1消息都由一个起始行（start-line）后跟一系列的八位组（octets）组成，格式类似于Internet消息格式[RFC5322]：零个或多个头字段（统称为“头部”或“头部部分”），一个空行表示头部部分的结束，以及一个可选的消息体。

HTTP-消息 = 起始行

\*( 头字段 CRLF )

CRLF

[ 消息体 ]

解析HTTP消息的正常过程是将起始行读入一个结构中，按字段名将每个头字段读入散列表，直到空行，然后使用解析的数据确定是否需要消息体。如果已指示消息体，则将其读取为一个流，直到读取了等于消息体长度的八位组或关闭了连接。

接收者必须将HTTP消息解析为一个八位组序列，该序列的编码是US-ASCII [USASCII]的超集。将HTTP消息解析为Unicode字符流，而不考虑特定的编码，会因字符串处理库处理包含八位组LF（%x0A）的无效多字节字符序列的方式而导致安全漏洞。基于字符串的解析器只能在从消息中提取元素后，例如在消息解析界定了各个字段后，在标头字段值中安全使用。

HTTP消息可以解析为一个流以进行增量处理或向下游转发。但是，接收者不能依赖于部分消息的增量传递，因为一些实现会为了网络效率、安全检查或负载转换而缓冲或延迟消息转发。

发送者不能在起始行和第一个头字段之间发送空白。接收者接收到起始行和第一个头字段之间的空白时，必须将该消息作为无效消息拒绝，或者消耗每个空白前置行而不进一步处理它（即，忽略整行以及任何以空白前导的后续行，直到接收到一个正确形成的头字段或头部部分结束）。

请求中这种空白的存在可能是企图欺骗服务器忽略该字段或处理其后的行作为新的请求，如果请求链中的其他实现以不同方式解释相同的消息，可能会导致安全漏洞。同样，在响应中这种空白的存在可能会被一些客户端忽略，或者导致其他客户端停止解析。

**这段话描述了HTTP/1.1消息的格式和解析过程。HTTP消息由起始行、零个或多个头字段、一个空行和一个可选的消息体组成。解析HTTP消息的一般过程是将起始行和头字段读入数据结构中，然后根据需要读取消息体。接收者必须以US-ASCII的超集编码来解析HTTP消息，以避免安全漏洞。尽管HTTP消息可以逐步解析或转发，但接收者不能依赖于部分消息的逐步传递。发送者不应在起始行和第一个头字段之间发送空白，因为这可能导致安全漏洞。因此，接收者必须在接收到空白行之前或之后立即处理它们，而不是将其视为有效数据。**