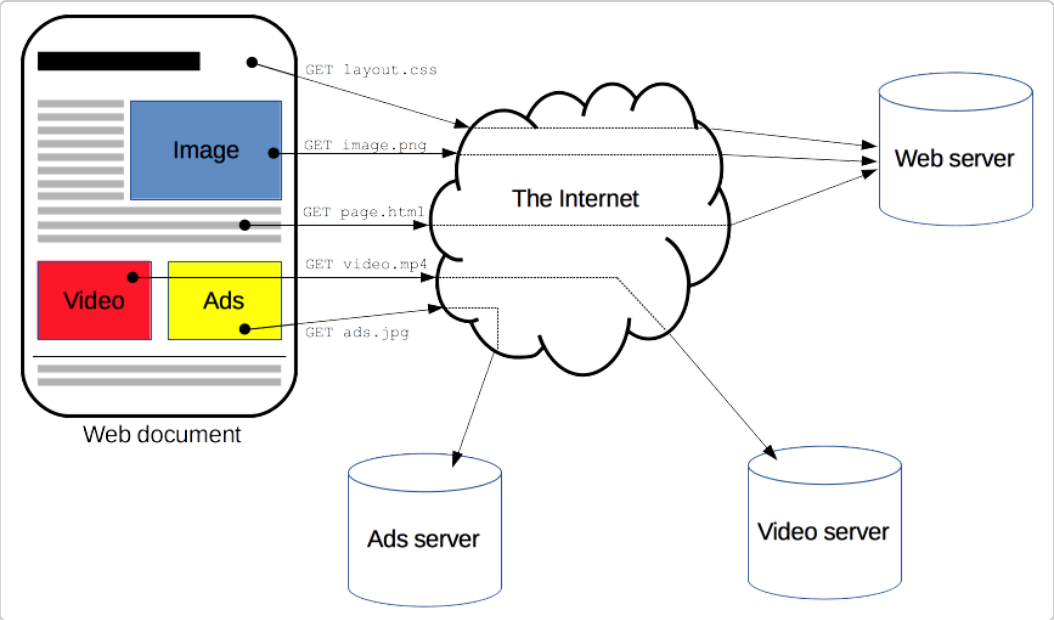
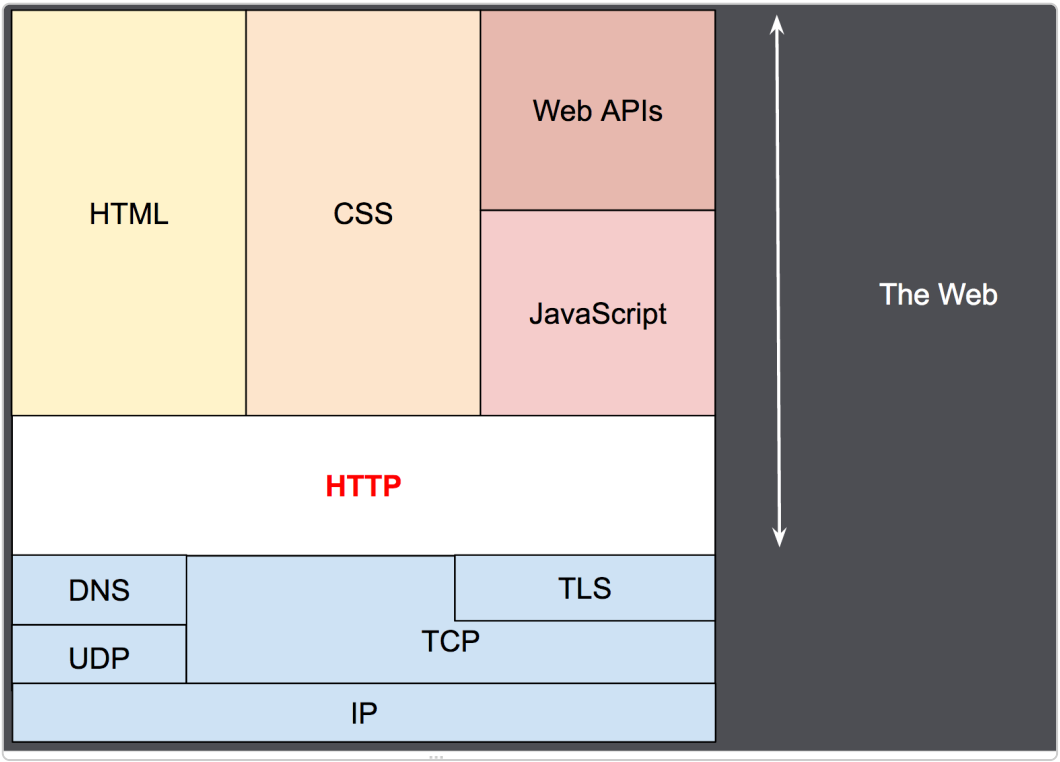
HTTP 概述

HTTP是一种用于获取 HTML 文档等资源的协议。它是 Web 上任何数据交换的基础，它是一种客户端-服务器协议，这意味着请求由接收方（通常是 Web 浏览器）发起。从获取的不同子文档中重建一个完整的文档，例如文本、布局描述、图像、视频、脚本等。



客户端和服务器通过交换单个消息（而不是数据流）进行通信。客户端（通常是 Web 浏览器）发送的消息称为*请求*，服务器发送的作为应答的消息称为*响应*。

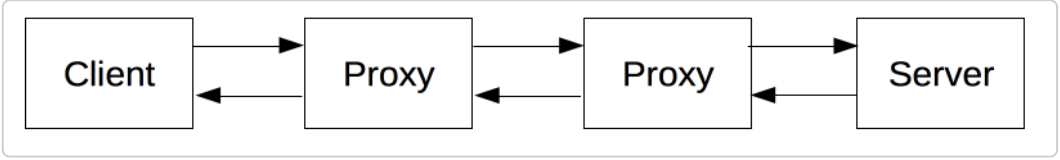


HTTP 设计于 1990 年代初期，是一种随着时间的推移而发展的可扩展协议。它是一种通过TCP或通过TLS加密的 TCP 连接发送的应用层协议，尽管理论上可以使用任何可靠的传输协议。由于其可扩展性，它不仅用于获取超文本文档，还用于获取图像和视频或将内容发布到服务器，例如 HTML 表单结果。HTTP 还可用于获取部分文档以按需更新网页。

基于 HTTP 的系统的组件

HTTP 是一种客户端-服务器协议：请求由一个实体发送，即用户代理（或代表它的代理）。大多数情况下，用户代理是 Web 浏览器，但它可以是任何东西，例如，爬行 Web 以填充和维护搜索引擎索引的机器人。

*每个单独的请求都被发送到服务器，服务器对其进行处理并提供称为响应的*答案。在客户端和服务器之间有许多实体，统称为代理，它们执行不同的操作，例如 充当网关或缓存。



实际上，在浏览器和处理请求的服务器之间有更多的计算机：有路由器、调制解调器等等。由于 Web 的分层设计，这些都隐藏在网络和传输层中。HTTP 在应用层之上。尽管对于诊断网络问题很重要，但底层大多与 HTTP 的描述无关。

客户端：用户代理

用户*代理*是代表用户的任何工具。此角色主要由 Web 浏览器执行，但也可能由工程师和 Web 开发人员用来调试其应用程序的程序执行。

浏览器**始终**是发起请求的实体。它永远不是服务器（尽管多年来已经添加了一些机制来模拟服务器发起的消息）。

要显示网页，浏览器会发送一个原始请求以获取代表该页面的 HTML 文档。然后它解析这个文件，发出与执行脚本、要显示的布局信息 (CSS) 和页面中包含的子资源（通常是图像和视频）相对应的额外请求。然后 Web 浏览器组合这些资源以呈现完整的文档，即网页。浏览器执行的脚本可以在后期获取更多资源，浏览器相应地更新网页。

网页是超文本文档。这意味着显示内容的某些部分是链接，这些链接可以被激活（通常通过单击鼠标）以获取新的网页，从而允许用户指导他们的用户代理并在 Web 中导航。浏览器将这些指示翻译成 HTTP 请求，并进一步解释 HTTP 响应以向用户呈现明确的响应。

网络服务器

在通信通道的对面是服务器，它根据客户端的请求*提供*文档。服务器实际上只是一台机器；但它实际上可能是一组共享负载（负载平衡）的服务器，或者是询问其他计算机（如缓存、数据库服务器或电子商务服务器）的复杂软件，全部或部分按需生成文档。

服务器不一定是一台机器，但可以在同一台机器上托管多个服务器软件实例。使用 HTTP/1.1 和 `Host` 标头，它们甚至可以共享相同的 IP 地址。

代理

在 Web 浏览器和服务器之间，许多计算机和机器中继 HTTP 消息。由于 Web 堆栈的分层结构，其中大部分在传输、网络或物理级别运行，在 HTTP 层变得透明，并可能对性能产生重大影响。运行在应用层的一般称为**代理**。这些可以是透明的，转发他们收到的请求而不以任何方式改变它们，或者是非透明的，在这种情况下，他们将在将请求传递给服务器之前以某种方式更改请求。代理可以执行多种功能：

- 缓存（缓存可以是公共的也可以是私有的，就像浏览器缓存一样）
- 过滤（如防病毒扫描或家长控制）
- 负载均衡（允许多个服务器服务于不同的请求）
- 身份验证（控制对不同资源的访问）
- 日志记录（允许存储历史信息）

HTTP 的基本方面

HTTP很简单

HTTP 通常被设计为简单且易于阅读，即使在 HTTP/2 中通过将 HTTP 消息封装到帧中而增加了复杂性。HTTP 消息可以被人类阅读和理解，为开发人员提供更容易的测试，并降低新手的复杂性。

HTTP 是可扩展的

在 HTTP/1.0 中引入，HTTP 标头使该协议易于扩展和试验。甚至可以通过客户端和服务器之间关于新标头语义的简单协议来引入新功能。

HTTP 是无状态的，但不是无会话的

HTTP 是无状态的：在同一连接上连续执行的两个请求之间没有链接。对于试图连贯地与某些页面进行交互的用户（例如，使用电子商务购物篮），这可能会立即产生问题。但是，虽然 HTTP 的核心本身是无状态的，但 HTTP cookie 允许使用有状态会话。使用标头可扩展性，将 HTTP Cookie 添加到工作流中，允许在每个 HTTP 请求上创建会话以共享相同的上下文或相同的状态。

HTTP 和连接

连接在传输层控制，因此从根本上超出了 HTTP 的范围。HTTP 不要求底层传输协议是基于连接的；它只要求它是*可靠的*，或者不会丢失消息（至少，在这种情况下会出现错误）。在 Internet 上最常见的两种传输协议中，TCP 是可靠的而 UDP 则不可靠。因此，HTTP 依赖于基于连接的 TCP 标准。

在客户端和服务器可以交换 HTTP 请求/响应对之前，它们必须建立 TCP 连接，这个过程需要多次往返。HTTP/1.0 的默认行为是为每个 HTTP 请求/响应对打开一个单独的 TCP 连接。当连续发送多个请求时，这比共享单个 TCP 连接效率低。

为了减轻这个缺陷，HTTP/1.1 引入了*流水线*（事实证明很难实现）和*持久连接*：底层的 TCP 连接可以使用 `Connection` 标头进行部分控制。HTTP/2 更进一步，通过单个连接多路复用消息，帮助保持连接温暖和更高效。

正在进行实验以设计更适合 HTTP 的更好的传输协议。例如，谷歌正在试验QUIC ，它建立在 UDP 之上，以提供更可靠和高效的传输协议。

HTTP可以控制什么

随着时间的推移，HTTP 的这种可扩展特性允许对 Web 进行更多控制和使用。缓存和身份验证方法是 HTTP 历史早期处理的功能。相比之下， 放宽*原点约束的*能力是在 2010 年代才添加的。

以下是可通过 HTTP 控制的常见功能列表：

- *缓存*：如何缓存文档可以由 HTTP 控制。服务器可以指示代理和客户端缓存什么内容以及缓存多长时间。客户端可以指示中间缓存代理忽略存储的文档。

放宽这种严格的分离，允许文档成为来自不同域的信息拼凑而成；甚至可能出于与安全相关的原因而这样做。

- *身份验证*：某些页面可能受到保护，因此只有特定用户才能访问它们。基本身份验证可以由 HTTP 提供，使用和类似的标头，或者通过使用 HTTP cookie `WWW-Authenticate` 设置特定会话。
- *代理和隧道*：服务器或客户端通常位于 Intranet 上，对其他计算机隐藏其真实 IP 地址。然后 HTTP 请求通过代理来跨越这个网络障碍。并非所有代理都是 HTTP 代理。例如，SOCKS 协议在较低级别运行。这些代理可以处理其他协议，如 ftp。
- *会话*：使用 HTTP cookie 允许您将请求与服务器状态相关联。这会创建会话，尽管基本 HTTP 是一种无状态协议。这不仅对电子商务购物篮很有用，而且对任何允许用户配置输出的站点也很有用。

HTTP流程

当客户端想要与服务器通信时，无论是最终服务器还是中间代理，它都会执行以下步骤：

1. 打开 TCP 连接：TCP 连接用于发送一个或多个请求，并接收应答。客户端可以打开一个新的连接，重用一个现有的连接，或者打开几个到服务器的 TCP 连接。
2. 发送 HTTP 消息：HTTP 消息（在 HTTP/2 之前）是人类可读的。在 HTTP/2 中，这些简单的消息被封装在帧中，无法直接读取，但原理是一样的。例如：

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

3. 读取服务器发送的响应，如：

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

```
<!DOCTYPE html>… (here come the 29769 bytes of the requested web page)
```

4. 关闭或重新使用连接以进行进一步的请求。

如果激活了 HTTP 流水线，则可以发送多个请求，而无需等待第一个响应被完全接收。事实证明，HTTP 流水线很难在现有网络中实施，旧软件与现代版本共存。HTTP/2 中的 HTTP 流水线已被取代，在一个框架内具有更强大的多路复用请求。

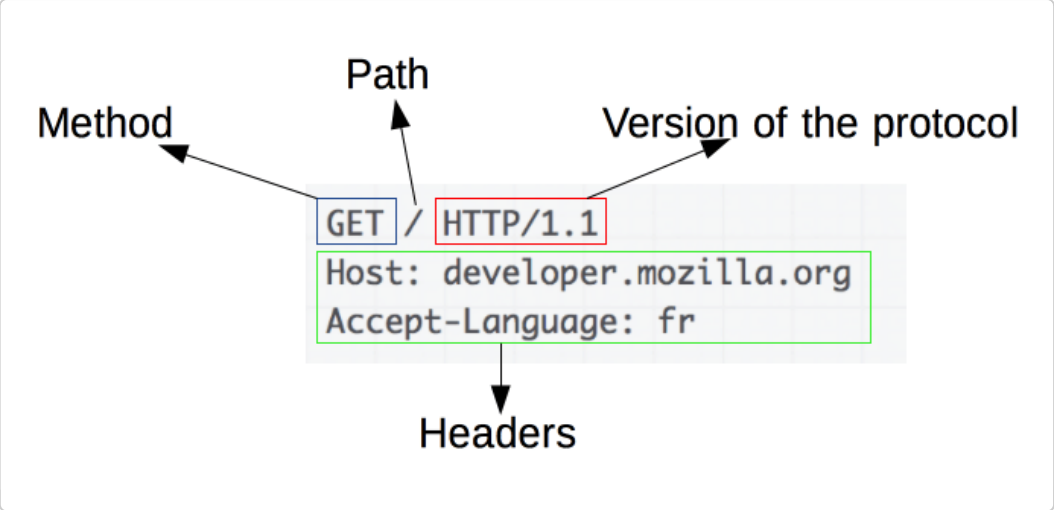
HTTP 消息

HTTP/1.1 及更早版本中定义的 HTTP 消息是人类可读的。在 HTTP/2 中，这些消息被嵌入到一个二进制结构中，一个*frame*，允许像压缩头和多路复用这样的优化。即使在此版本的 HTTP 中只发送部分原始 HTTP 消息，每条消息的语义都不会改变，并且客户端会（虚拟地）重构原始 HTTP/1.1 请求。因此，理解 HTTP/1.1 格式的 HTTP/2 消息非常有用。

有两种类型的 HTTP 消息，请求和响应，每种都有自己的格式。

要求

一个示例 HTTP 请求：



请求由以下元素组成：

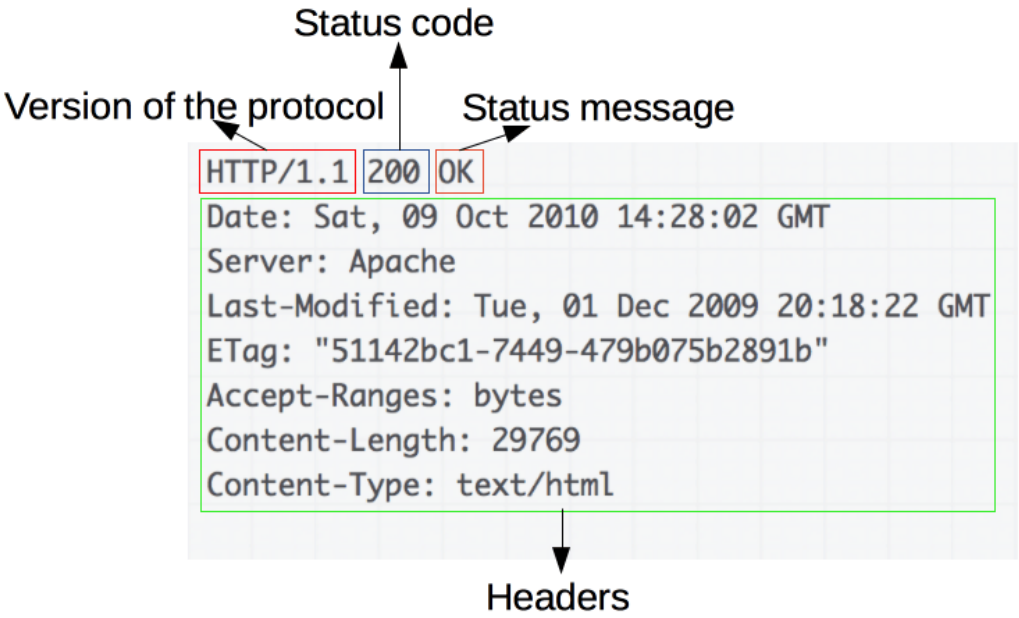
- 一种 HTTP方法，通常是一个动词，如 GET，POST，或名词如 OPTIONS or HEAD，它定义了客户端想要执行的操作。通常，客户端想

要获取资源（使用）或发布HTML 表单 GET 的值（使用），但在其他情况下可能需要更多操作。 POST

- 要获取的资源的路径；从上下文中显而易见的元素中剥离的资源 URL，例如没有协议( ) http:// 、域（此处为 developer.mozilla.org ）或 TCP端口（此处为 80 ）。
- HTTP 协议的版本。
- 为服务器传达附加信息的可选标头。
- 一个正文，对于某些方法，如 POST ，类似于响应中的方法，其中包含发送的资源。

回应

示例响应：



响应由以下元素组成：

- 他们遵循的 HTTP 协议的版本。
- 状态代码，指示请求是否成功，以及原因。
- 状态消息，状态码的非权威性简短描述。
- HTTP标头，如请求的标头。
- 可选地，包含获取的资源的正文。

基于 HTTP 的 API

最常用的基于 HTTP 的 API 是 `XMLHttpRequest` API，它可以用来在用户代理和服务器之间交换数据。现代版 `Fetch API` 提供了相同的功能，但功能更强大、更灵活。

另一个 API server-sent events是一种单向服务，它允许服务器使用 HTTP 作为传输机制向客户端发送事件。使用该 `EventSource` 接口，客户端打开连接并建立事件处理程序。客户端浏览器自动将到达 HTTP 流的消息转换为适当的 `Event` 对象。然后它将它们传递给已经为事件注册的事件处理程序（`type` 如果已知），或者 `onmessage` 如果没有建立特定类型的事件处理程序则传递给事件处理程序。

结论

HTTP 是一种易于使用的可扩展协议。客户端-服务器结构与添加标头的能力相结合，使 HTTP 能够随着 Web 的扩展功能而进步。

尽管 HTTP/2 通过在帧中嵌入 HTTP 消息来提高性能增加了一些复杂性，但消息的基本结构自 HTTP/1.0 以来一直保持不变。会话流仍然很简单，允许使用简单的HTTP 消息监视器 对其进行调查和调试。

此页面最后修改于 2023 年 4 月 10 日由MDN 贡献者提供。