什么是 HTTP?

HTTP 是基于 TCP/IP 的应用层通信协议，它规范了客户端和服务器之间的通信方式。它定义了如何通过 Internet 请求和传输内容。

免费链接

文章 什么是 HTTP？

文章 HTTP 概述

文章 HTTP/2 之旅

视频 HTTP 速成课程和探索

【文章】什么是 HTTP？

超文本传输协议(HTTP)是万维网的基础，用于使用超文本链接加载网页。HTTP 是一种应用层协议，旨在在联网设备之间传输信息，并在网络协议栈的其他层之上运行。HTTP 上的典型流程涉及客户端向服务器发出请求，然后服务器发送响应消息。

PS:

https://www.cloudflare.com/zh-cn/learning/ddos/glossary/hypertext-transfer-protocol-http/ cloudflare

HTTP 请求中有什么？

HTTP 请求是互联网通信平台（如 web 浏览器）请求加载网站所需信息的方式。通过 Internet 发出的每个 HTTP 请求都带有一系列编码数据，这些数据携带不同类型的信息。一点典型的 HTTP 请求包含：

1. HTTP 版本

2. URL
3. 请求方法
4. 请求头部
5. 请求 BODY 正文

让我们更深入地探讨这些请求是如何工作的，以及如何使用请求的内容来共享信息。

HTTP 请求方法

HTTP 方法，有时称为 HTTP 动词，指示 HTTP 请求期望从查询的服务器获得的操作。例如，两种常见的 HTTP 方法是“GET”和“POST”；“GET”请求期望返回信息（通常以网站的形式），而“POST”请求通常表明客户端正在向 Web 服务器提交信息（例如表单信息，例如提交的用户名和密码）。

什么是 HTTP 请求头

HTTP 标头包含存储在键值对中的文本信息，它们包含在每个 HTTP 请求（和响应，稍后会详细介绍）中。这些标头传达核心信息，例如客户端正在使用什么浏览器，正在请求什么数据。

来自谷歌浏览器网络选项卡的 HTTP 请求标头示例：

HTTP 请求正文中有什么？

请求的主体是包含请求正在传输的信息的“主体”的部分。HTTP 请求的正文包含提交到 Web 服务器的任何信息，例如用户名和密码，或输入到表单的任何其他数据。

HTTP 响应中有什么？（Response）

HTTP 响应是 Web 客户端（通常是浏览器）从 Internet 服务器接收的响应 HTTP 请求的内容。这些响应根据 HTTP 请求中的请求的内容来传递有价值的信息。

一个典型的 HTTP 响应包含：

1. 一个 HTTP 状态码
2. HTTP 响应头
3. HTTP 响应 body 正文

让我们分析一下：

HTTP 状态码是什么？

HTTP 状态码是 3 位的代码，通常用来表示 HTTP 请求是否已成功完成。状态代码分为以下 5 块：

1. 1xx 信息
2. 2xx 成功
3. 3xx 重定向
4. 4xx 客户端错误
5. 5xx 服务器错误

xx 代表 00-99 之间不同的数字

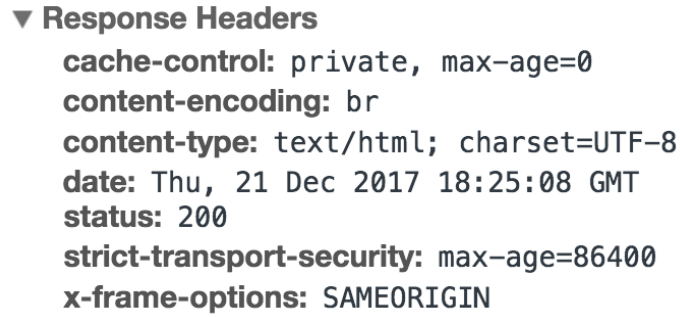
以数字 2 开头的状态码表示成功。例如，在客户机请求一个 web 页面后，最常见的响应的状态码是 200 OK，表明请求已正确完成。

如果响应以 4 或 5 开始，这意味着有一个错误，网页将不会显示。以 4 开头的状态码表示客户端错误（当在 URL 中输入错误时，经常会遇到 404 NOT FOUND 状态码）。以 5 开头的状态码表示服务器端出现了错误。状态码也可以以 1 或 3 开头，分别表示信息响应和重定向。

HTTP 响应头是什么？

与 HTTP 请求非常相似，HTTP 响应带有传达重要信息的标头，例如在响应正文中发送的数据的语言和格式。

谷歌浏览器网络标签的 HTTP 响应头示例：



HTTP 响应正文中有什么？

成功的 GET 请求的 HTTP 响应通常有一个主体，其中包含所请求的信息。在大多数 web 请求中，这是 HTML 数据，web 浏览器会将其转换为网页。

DDoS 攻击是否可以通过 HTTP 进行

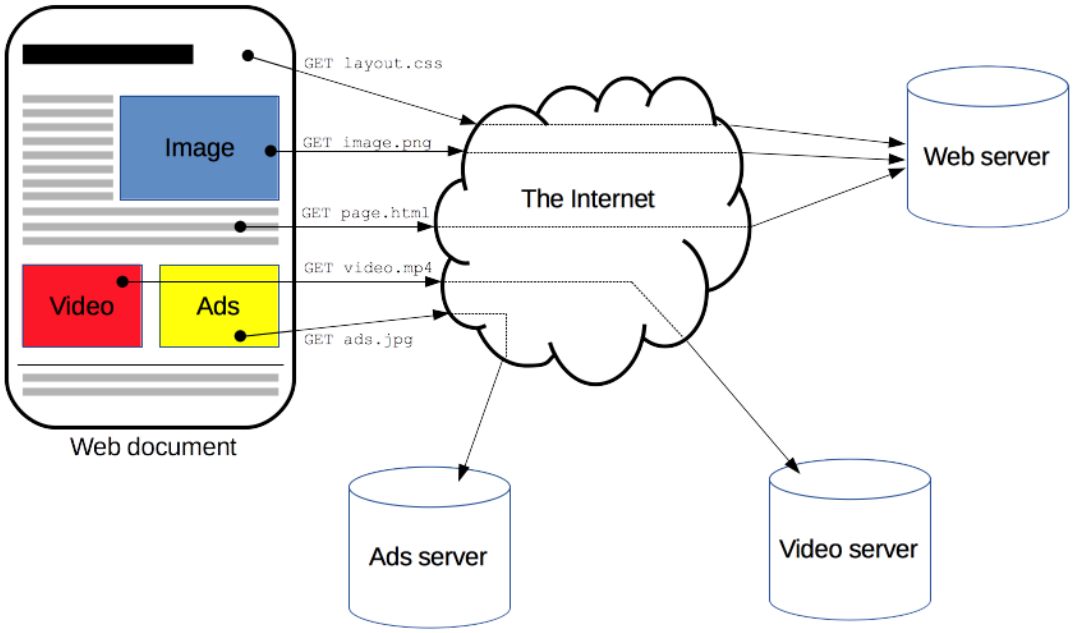
请记住，HTTP 是一种“无状态”协议，这意味着每个命令都独立于任何其他命令运行。在原始规范中，每个 HTTP 请求都创建和关闭一个 TCP 连接。在较新版本的 HTTP 协议（HTTP1.1 及更高版本）中，持久连接允许多个 HTTP 请求通过持久 TCP 连接传递，从而改善资源消耗。在 DoS 或 DDoS 攻击的上下文中，大量的 HTTP 请求可用于对目标设备发起攻击，被视为应用层攻击或第 7 层攻击的一部分。

【文章】HTTP 概述

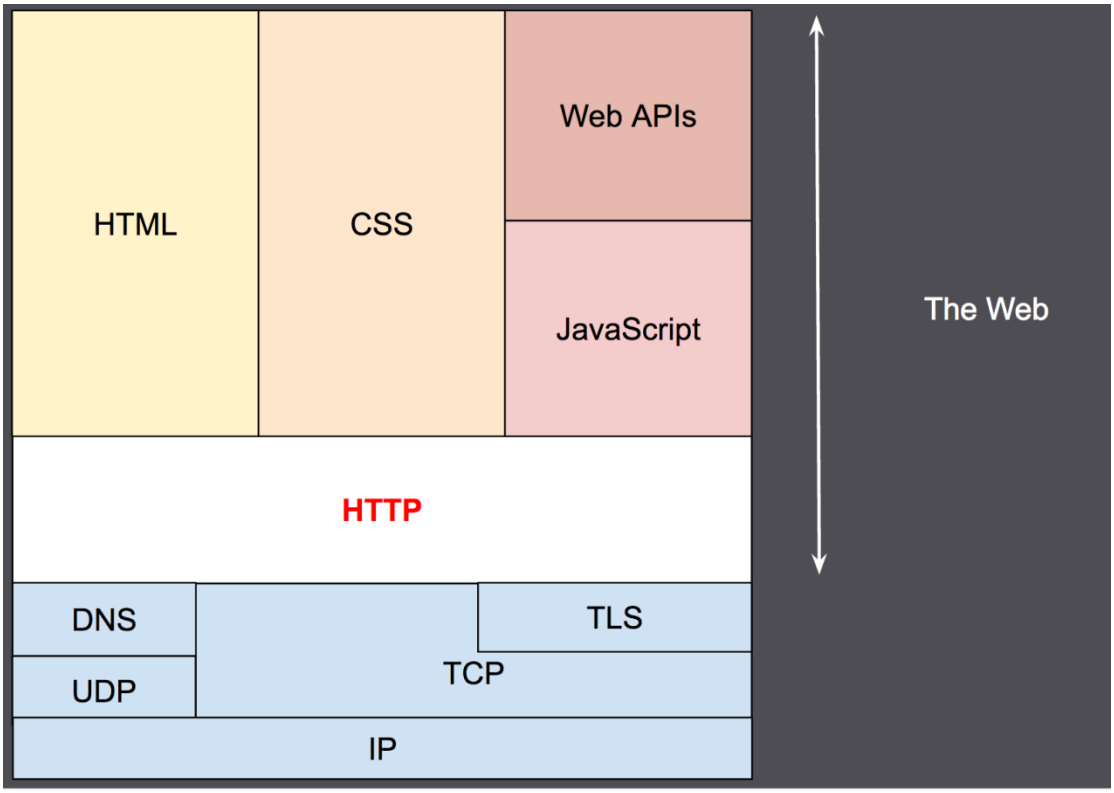
地址：https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

PS: MDN，Mozilla 基金会产品和网络技术开发文档的免费网站，支持中文。特别好的学习资源。

HTTP 是一种能够获取如 HTML 这样的网络资源的 protocol(通讯协议)。它是在 Web 上进行数据交换的基础，是一种 client-server 协议，也就是说，请求通常是由像浏览器这样的接受方发起的。一个完整的 Web 文档通常是由不同的子文档拼接而成的，像是文本、布局描述、图片、视频、脚本等等。



客户端和服务端通过交换各自的消息（与数据流正好相反）进行交互。由像浏览器这样的客户端发出的消息叫做 requests，被服务端响应的消息叫做 responses。



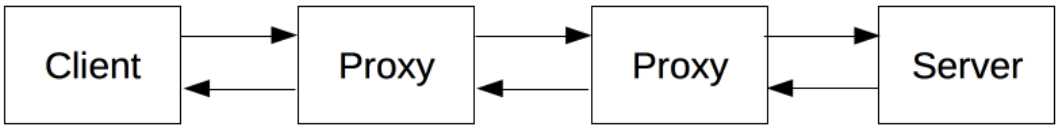
HTTP 被设计于 20 世纪 90 年代初期，是一种可扩展的协议。它是应用层的协议，通过 TCP，或者是 TLS –加密的 TCP 连接来发送，理论上任何可靠的传输协议都可以使用。因为其良好的扩展性，时至今日，它不仅被用来传输超文本文档，

还用来传输图片、视频或者向服务器发送如 HTML 表单这样的信息。HTTP 还可以根据网页需求，仅获取部分 Web 文档内容更新网页。

基于 HTTP 的组件系统

HTTP 是一个 client-server 协议：请求通过一个实体被发出，实体也就是用户代理。大多数情况下，这个用户代理都是指浏览器，当然它也可能是任何东西，比如一个爬取网页生成维护搜索引擎索引的机器爬虫。

每一个发送到服务器的请求，都会被服务器处理并返回一个消息，也就是 response。在这个请求与响应之间，还有许许多多的被称为 proxies 的实体，他们的作用与表现各不相同，比如有些是网关，还有些是 caches 等。



实际上，在一个浏览器和处理请求的服务器之间，还有路由器、调制解调器等许多计算机。由于 Web 的层次设计，那些在网络层和传输层的细节都被隐藏起来了。HTTP 位于最上层的应用层。虽然底层对于分析网络问题非常重要，但是大多都跟对 HTTP 的描述不相干。

客户端：user-agent

user-agent 就是任何能够为用户发起行为的工具。这个角色通常都是由浏览器来扮演。一些例外情况，比如是工程师使用的程序，以及 Web 开发人员调试应用程序。

浏览器总是作为发起一个请求的实体，他永远不是服务器（虽然近几年已经出现一些机制能够模拟由服务器发起的请求消息了）。

要展现一个网页，浏览器首先发送一个请求来获取页面的 HTML 文档，再解析文档中的资源信息发送其他请求，获取可执行脚本或 CSS 样式来进行页面布局渲染，以及一些其它页面资源（如图片和视频等）。然后，浏览器将这些资源整合到一起，展现出一个完整的文档，也就是网页。浏览器执行的脚本可以在之后的阶段获取更多资源，并相应地更新网页。

一个网页就是一个超文本文档。也就是说，有一部分显示的文本可能是链接，启动它（通常是鼠标的点击）就可以获取一个新的网页，使得用户可以控制客户端进行网上冲浪。浏览器来负责发送 HTTP 请求，并进一步解析 HTTP 返回的消息，以向用户提供明确的响应。

Web 服务端

在上述通信过程的另一端，是由 Web Server 来服务并提供客户端所请求的文档。Server 只是虚拟意义上代表一个机器：它可以是共享负载（负载均衡）的一组服务器组成的计算机集群，也可以是一种复杂的软件，通过向其他计算机（如缓存，数据库服务器，电子商务服务器 ...）发起请求来获取部分或全部资源。

Server 不一定是一台机器，但一个机器上可以装载的众多 Servers。在 HTTP/1.1 和 Host 头部中，它们甚至可以共享同一个 IP 地址。

代理（Proxies）

在浏览器和服务器之间，有许多计算机和其他设备转发了 HTTP 消息。由于 Web 栈层次结构的原因，它们大多都出现在传输层、网络层和物理层上，对于 HTTP 应用层而言就是透明的，虽然它们可能会对应用层性能有重要影响。还有一部分是表现在应用层上的，被称为代理（Proxies）。代理（Proxies）既可以表现得透明，又可以不透明（“改变请求”会通过它们）。代理主要有如下几种作用：

- 缓存（可以是公开的也可以是私有的，像浏览器的缓存）
- 过滤（像反病毒扫描，家长控制...）
- 负载均衡（让多个服务器服务不同的请求）
- 认证（对不同资源进行权限管理）
- 日志记录（允许存储历史信息）

HTTP 的基本性质

HTTP 是简单的

虽然下一代 HTTP/2 协议将 HTTP 消息封装到了帧（frames）中，HTTP 大体上

还是被设计得简单易读。HTTP 报文能够被人读懂，还允许简单测试，降低了门槛，对新人很友好。

HTTP 是可扩展的

在 HTTP/1.0 中出现的 HTTP headers 让协议扩展变得非常容易。只要服务端和客户端就新 headers 达成语义一致，新功能就可以被轻松加入进来。

HTTP 是无状态，有会话的

HTTP 是无状态的：在同一个连接中，两个执行成功的请求之间是没有关系的。这就带来了一个问题，用户没有办法在同一个网站中进行连续的交互，比如在一个电商网站里，用户把某个商品加入到购物车，切换一个页面后再次添加了商品，这两次添加商品的请求之间没有关联，浏览器无法知道用户最终选择了哪些商品。而使用 HTTP 的头部扩展，HTTP Cookies 就可以解决这个问题。把 Cookies 添加到头部中，创建一个会话让每次请求都能共享相同的上下文信息，达成相同的状态。

注意，HTTP 本质是无状态的，使用 Cookies 可以创建有状态的会话。

HTTP 和连接

一个连接是由传输层来控制的，这从根本上不属于 HTTP 的范围。HTTP 并不需要其底层的传输层协议是面向连接的，只需要它是可靠的，或不丢失消息的（至少返回错误）。在互联网中，有两个最常用的传输层协议：TCP 是可靠的，而 UDP 不是。因此，HTTP 依赖于面向连接的 TCP 进行消息传递，但连接并不是必须的。

在客户端（通常指浏览器）与服务器能够交互（客户端发起请求，服务器返回响应）之前，必须在这两者间建立一个 TCP 链接，打开一个 TCP 连接需要多次往返交换消息（因此耗时）。HTTP/1.0 默认为每一对 HTTP 请求/响应都打开一个单独的 TCP 连接。当需要连续发起多个请求时，这种模式比多个请求共享同一个 TCP 链接更低效。

为了减轻这些缺陷，HTTP/1.1 引入了流水线（被证明难以实现）和持久连接的概念：底层的 TCP 连接可以通过 Connection 头部来被部分控制。HTTP/2 则发展得更远，通过在一个连接复用消息的方式来让这个连接始终保持为暖连接。

为了更好的适合 HTTP，设计一种更好传输协议的进程一直在进行。Google 就研

发了一种以 UDP 为基础，能提供更可靠更高效的传输协议 QUIC

HTTP 能控制什么

多年以来，HTTP 良好的扩展性使得越来越多的 Web 功能归其控制。缓存和认证很早就可以由 HTTP 来控制了。另一方面，对同源同域的限制到 2010 年才有所改变。

以下是可以被 HTTP 控制的常见特性。

- 缓存

  文档如何缓存能通过 HTTP 来控制。服务端能告诉代理和客户端哪些文档需要被缓存，缓存多久，而客户端也能够命令中间的缓存代理来忽略存储的文档。

- 开放同源限制

  为了防止网络窥听和其它隐私泄漏，浏览器强制对 Web 网站做了分割限制。只有来自于相同来源的网页才能够获取网站的全部信息。这样的限制有时反而成了负担，HTTP 可以通过修改头部来开放这样的限制，因此 Web 文档可以是由不同域下的信息拼接成的（某些情况下，这样做还有安全因素考虑）。

- 认证

  一些页面能够被保护起来，仅让特定的用户进行访问。基本的认证功能可以直接通过 HTTP 提供，使用 Authenticate 相似的头部即可，或用 HTTP Cookies 来设置指定的会话。

- 代理和隧道

  通常情况下，服务器和/或客户端是处于内网的，对外网隐藏真实 IP 地址。因此 HTTP 请求就要通过代理越过这个网络屏障。但并非所有的代理都是 HTTP 代理。例如，SOCKS 协议的代理就运作在更底层，一些像 FTP 这样的协议也能够被它们处理。

- 会话

  使用 HTTP Cookies 允许你用一个服务端的状态发起请求，这就创建了会话。虽然基本的 HTTP 是无状态协议。这很有用，不仅是因为这能应用到像购物车这样的电商业务上，更是因为这使得任何网站都能轻松为用户定制展示内

容了。

HTTP 流

当客户端想要和服务端进行信息交互时（服务端是指最终服务器，或者是一个中间代理），过程表现为下面几步：

1. 打开一个 TCP 连接：TCP 连接被用来发送一条或多条请求，以及接受响应消息。客户端可能打开一条新的连接，或重用一个已经存在的连接，或者也可能开几个新的 TCP 连接连向服务端。
2. 发送一个 HTTP 报文：HTTP 报文（在 HTTP/2 之前）是语义可读的。在 HTTP/2 中，这些简单的消息被封装在了帧中，这使得报文不能被直接读取，但是原理仍是相同的。

```
GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr
```

3. 读取服务端返回的报文信息：

```
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 comes the 29769 bytes of the requested
web page)
```

4. 关闭连接或者为后续请求重用连接

当 HTTP 流水线启动时，后续请求都可以不用等待第一个请求的成功响应就被发送。然而 HTTP 流水线已被证明很难在现有的网络中实现，因为现有网络中有很多老旧的软件与现代版本的软件共存。因此，HTTP 流水线已被在有多请求下表现得更稳健的 HTTP/2 的帧所取代。

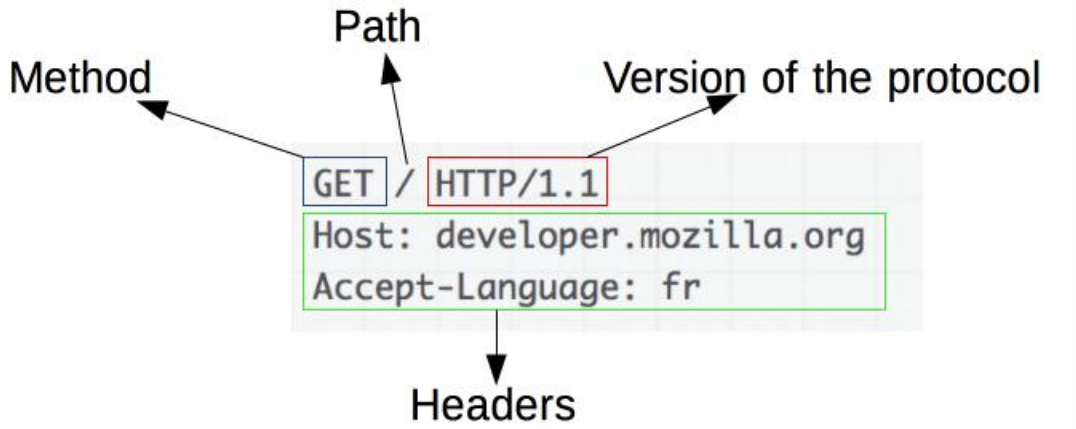
HTTP 报文

HTTP/1.1 以及更早的 HTTP 协议报文都是语义可读的。在 HTTP/2 中，这些报文被嵌入到了一个新的二进制结构，帧。帧允许实现很多优化，比如报文头部的压缩和复用。即使只有原始 HTTP 报文的一部分以 HTTP/2 发送出来，每条报文的语义依旧不变，客户端会重组原始 HTTP/1.1 请求。因此用 HTTP/1.1 格式来理解 HTTP/2 报文仍旧有效。

有两种 HTTP 报文的类型，请求与响应，每种都有其特定的格式。

请求

HTTP 请求的一个例子：



请求由以下元素组成：

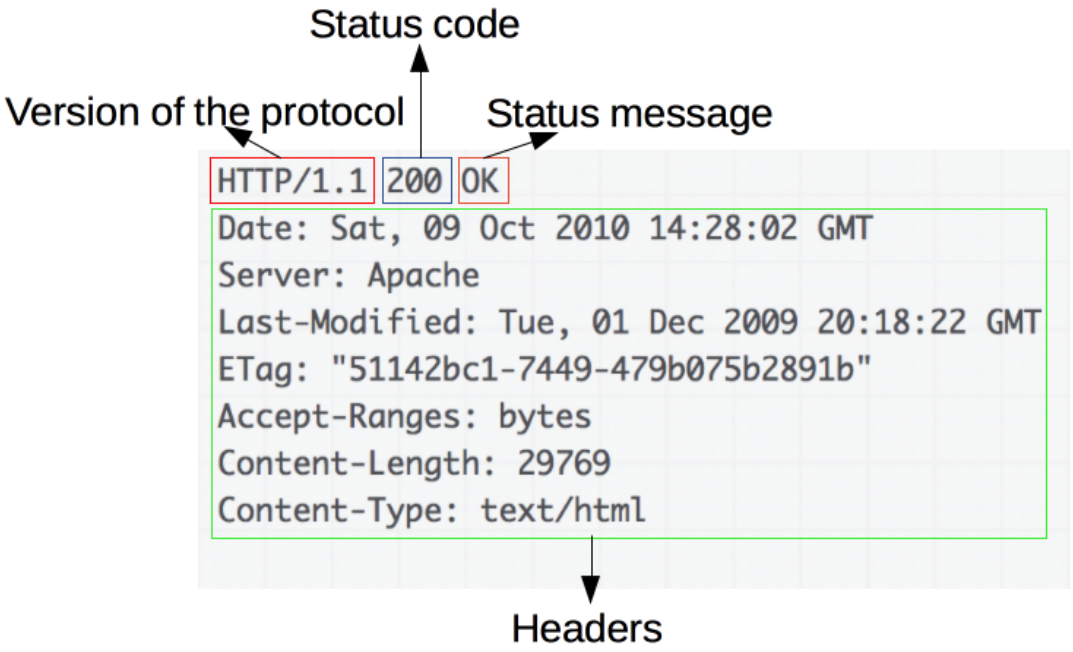
- 一个 HTTP 的 method，经常是由一个动词像 GET，POST 或者一个名词像 OPTIONS，HEAD 来定义客户端的动作行为。通常客户端的操作都是获取资源（GET 方法）或者发送 HTML form 表单值（POST 方法），虽然在一些情况下也会有其他操作。
- 要获取的资源的路径，通常是上下文中就很明显的元素资源的 URL，它没有

protocol（http://），domain（developer.mozilla.org），或是 TCP 的 port (en-US)（HTTP 一般在 80 端口）。

- HTTP 协议版本号。
- 为服务端表达其他信息的可选头部 headers。
- 对于一些像 POST 这样的方法，报文的 body 就包含了发送的资源，这与响应报文的 body 类似。

响应

HTTP 响应的一个例子：



响应报文包含了下面的元素：

- HTTP 协议版本号。
- 一个状态码（status code），来告知对应请求执行成功或失败，以及失败的原因。
- 一个状态信息，这个信息是非权威的状态码描述信息，可以由服务端自行设定。
- HTTP headers，与请求头部类似。
- 可选项，比起请求报文，响应报文中更常见地包含获取的资源 body。

基于 HTTP 的 APIs

基于 HTTP 的最常用 API 是 XMLHttpRequest API，可用于在 user agent 和服务器之间交换数据。现代 Fetch API 提供相同的功能，具有更强大和灵活的功能集。另一种 API，即服务器发送的事件，是一种单向服务，允许服务器使用 HTTP 作为传输机制向客户端发送事件。 使用 EventSource 接口，客户端打开连接并建立事件句柄。 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，并将它们传递给专门处理这类 type 事件的句柄，如果有这么个句柄的话。但如果相应的事件处理句柄根本没有建立，那就交给 onmessage (en-US)事件处理程序处理。

总结

HTTP 是一种简单可扩展的协议，其 Client-Server 的结构以及轻松扩展头部信息的能力使得 HTTP 可以和 Web 共同发展。

即使 HTTP/2 为了提高性能将 HTTP 报文嵌入到帧中这一举措增加了复杂度，但是从 Web 应用的角度看，报文的基本结构没有变化，从 HTTP/1.0 发布起就是这样的结构。会话流依旧简单，通过一个简单的 HTTP message monitor 就可以查看和纠错。

【文章】HTTP/2 之旅

地址：https://kamranahmed.info/blog/2016/08/13/http-in-depth/ 很细节的讲述 HTTP 发展。

【视频】HTTP 速成课程和探索

地址：https://www.youtube.com/watch?v=iYM2zFP3Zn0
这个知识点目前对初学者来说还有早，可以学习一段时间后再来学习

youtobe 视频博主：Traversy Media 前端和后端技术学习很棒的博主