**HTTP的前世今生**

[**2019年10月01日**](https://coolshell.cn/articles/19840.html)[**陈皓**](https://coolshell.cn/articles/author/haoel)**评论**[**58 条评论**](https://coolshell.cn/articles/19840.html#comments)**45,457 人阅读**

HTTP (Hypertext transfer protocol) 翻译成中文是超文本传输协议，是互联网上重要的一个协议，由欧洲核子研究委员会CERN的英国工程师 [Tim Berners-Lee](https://en.wikipedia.org/wiki/Tim_Berners-Lee) v发明的，同时，他也是WWW的发明人，最初的主要是用于传递通过HTML封装过的数据。在1991年发布了HTTP 0.9版，在1996年发布1.0版，1997年是1.1版，1.1版也是到今天为止传输最广泛的版本（初始[RFC 2068](https://tools.ietf.org/html/rfc2068) 在1997年发布， 然后在1999年被 [RFC 2616](https://tools.ietf.org/html/rfc2616) 取代，再在2014年被 [RFC 7230](https://tools.ietf.org/html/rfc7230) /[7231](https://tools.ietf.org/html/rfc7231)/[7232](https://tools.ietf.org/html/rfc7232)/[7233](https://tools.ietf.org/html/rfc7233)/[7234](https://tools.ietf.org/html/rfc7234)/[7235](https://tools.ietf.org/html/rfc7235)取代），2015年发布了2.0版，其极大的优化了HTTP/1.1的性能和安全性，而2018年发布的3.0版，继续优化HTTP/2，激进地使用UDP取代TCP协议，目前，HTTP/3 在2019年9月26日 被 Chrome，Firefox，和Cloudflare支持，所以我想写下这篇文章，简单地说一下HTTP的前世今生，让大家学到一些知识，并希望可以在推动一下HTTP标准协议的发展。

目录

* [HTTP 0.9 / 1.0](https://coolshell.cn/articles/19840.html#HTTP_0_9_/_1_0)
* [HTTP/1.1](https://coolshell.cn/articles/19840.html#_HTTP/1_1)
* [HTTP/2](https://coolshell.cn/articles/19840.html#HTTP/2)
* [HTTP/3](https://coolshell.cn/articles/19840.html#HTTP/3)

**HTTP 0.9 / 1.0**

0.9和1.0这两个版本，就是最传统的 request – response的模式了，HTTP 0.9版本的协议简单到极点，请求时，不支持请求头，只支持 GET 方法，没了。HTTP 1.0 扩展了0.9版，其中主要增加了几个变化：

* 在请求中加入了HTTP版本号，如：GET /coolshell/index.html HTTP/1.0
* HTTP 开始有 header了，不管是request还是response 都有header了。
* 增加了HTTP Status Code 标识相关的状态码。
* 还有 Content-Type 可以传输其它的文件了。

我们可以看到，HTTP 1.0 开始让这个协议变得很文明了，一种工程文明。因为：

* 一个协议有没有版本管理，是一个工程化的象征。
* header是协议可以说是把元数据和业务数据解耦，也可以说是控制逻辑和业务逻辑的分离。
* Status Code 的出现可以让请求双方以及第三方的监控或管理程序有了统一的认识。最关键是还是控制错误和业务错误的分离。

（注：国内很多公司HTTP无论对错只返回200，这种把HTTP Status Code 全部抹掉完全是一种工程界的倒退）

但是，HTTP1.0性能上有一个很大的问题，那就是每请求一个资源都要新建一个TCP链接，而且是串行请求，所以，就算网络变快了，打开网页的速度也还是很慢。所以，HTTP 1.0 应该是一个必需要淘汰的协议了。

**HTTP/1.1**

HTTP/1.1 主要解决了HTTP 1.0的网络性能的问题，以及增加了一些新的东西：

* 可以设置 keepalive 来让HTTP重用TCP链接，重用TCP链接可以省了每次请求都要在广域网上进行的TCP的三次握手的巨大开销。这是所谓的“**HTTP 长链接**” 或是 “**请求响应式的HTTP 持久链接**”。英文叫 HTTP Persistent connection.
* 然后支持pipeline网络传输，只要第一个请求发出去了，不必等其回来，就可以发第二个请求出去，可以减少整体的响应时间。（注：非幂等的POST 方法或是有依赖的请求是不能被pipeline化的）
* 支持 Chunked Responses ，也就是说，在Response的时候，不必说明 Content-Length 这样，客户端就不能断连接，直到收到服务端的EOF标识。这种技术又叫 “**服务端Push模型**”，或是 “**服务端Push式的HTTP 持久链接**”
* 还增加了 cache control 机制。
* 协议头注增加了 Language, Encoding, Type 等等头，让客户端可以跟服务器端进行更多的协商。
* 还正式加入了一个很重要的头—— [HOST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Host)这样的话，服务器就知道你要请求哪个网站了。因为可以有多个域名解析到同一个IP上，要区分用户是请求的哪个域名，就需要在HTTP的协议中加入域名的信息，而不是被DNS转换过的IP信息。
* 正式加入了 OPTIONS 方法，其主要用于 [CORS – Cross Origin Resource Sharing](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS) 应用。

HTTP/1.1应该分成两个时代，一个是2014年前，一个是2014年后，因为2014年HTTP/1.1有了一组RFC（[7230](https://tools.ietf.org/html/rfc7230) /[7231](https://tools.ietf.org/html/rfc7231)/[7232](https://tools.ietf.org/html/rfc7232)/[7233](https://tools.ietf.org/html/rfc7233)/[7234](https://tools.ietf.org/html/rfc7234)/[7235](https://tools.ietf.org/html/rfc7235)），这组RFC又叫“HTTP/2 预览版”。其中影响HTTP发展的是两个大的需求：

* 一个需要是加大了HTTP的安全性，这样就可以让HTTP应用得广泛，比如，使用TLS协议。
* 另一个是让HTTP可以支持更多的应用，在HTTP/1.1 下，HTTP已经支持四种网络协议：
  + 传统的短链接。
  + 可重用TCP的的长链接模型。
  + 服务端push的模型。
  + WebSocket模型。

自从2005年以来，整个世界的应用API越来多，这些都造就了整个世界在推动HTTP的前进，我们可以看到，**自2014的HTTP/1.1 以来，这个世界基本的应用协议的标准基本上都是向HTTP看齐了，也许2014年前，还有一些专用的RPC协议，但是2014年以后，HTTP协议的增强，让我们实在找不出什么理由不向标准靠拢，还要重新发明轮子了。**

**HTTP/2**

虽然 HTTP/1.1 已经开始变成应用层通讯协议的一等公民了，但是还是有性能问题，虽然HTTP/1.1 可以重用TCP链接，但是请求还是一个一个串行发的，需要保证其顺序。然而，大量的网页请求中都是些资源类的东西，这些东西占了整个HTTP请求中最多的传输数据量。所以，理论上来说，如果能够并行这些请求，那就会增加更大的网络吞吐和性能。

另外，HTTP/1.1传输数据时，是以文本的方式，借助耗CPU的zip压缩的方式减少网络带宽，但是耗了前端和后端的CPU。这也是为什么很多RPC协议诟病HTTP的一个原因，就是数据传输的成本比较大。

其实，在2010年时，Google 就在搞一个实验型的协议，这个协议叫[SPDY](https://en.wikipedia.org/wiki/SPDY)，这个协议成为了HTTP/2的基础（也可以说成HTTP/2就是SPDY的复刻）。HTTP/2基本上解决了之前的这些性能问题，其和HTTP/1.1最主要的不同是：

* HTTP/2是一个二进制协议，增加了数据传输的效率。
* HTTP/2是可以在一个TCP链接中并发请求多个HTTP请求，移除了HTTP/1.1中的串行请求。
* HTTP/2会压缩头，如果你同时发出多个请求，他们的头是一样的或是相似的，那么，协议会帮你消除重复的部分。这就是所谓的HPACK算法（参看[RFC 7541](https://tools.ietf.org/html/rfc7541) 附录A）
* HTTP/2允许服务端在客户端放cache，又叫服务端push，也就是说，你没有请求的东西，我服务端可以先送给你放在你的本地缓存中。比如，你请求X，我服务端知道X依赖于Y，虽然你没有的请求Y，但我把把Y跟着X的请求一起返回客户端。

对于这些性能上的改善，在Medium上有篇文章你可看一下相关的细节说明和测试“[HTTP/2: the difference between HTTP/1.1, benefits and how to use it](https://medium.com/@factoryhr/http-2-the-difference-between-http-1-1-benefits-and-how-to-use-it-38094fa0e95b)”

当然，还需要注意到的是HTTP/2的协议复杂度比之前所有的HTTP协议的复杂度都上升了许多许多，其内部还有很多看不见的东西，比如其需要维护一个“优先级树”来用于来做一些资源和请求的调度和控制。如此复杂的协议，自然会产生一些不同的声音，或是降低协议的可维护和可扩展性。所以也有一些争议。尽管如此，HTTP/2还是很快地被世界所采用。

HTTP/2 是2015年推出的，其发布后，Google 宣布移除对SPDY的支持，拥抱标准的 HTTP/2。过了一年后，就有8.7%的网站开启了HTTP/2，根据 [这份报告](https://w3techs.com/technologies/details/ce-http2/all/all) ，截止至本文发布时（2019年10月1日 ）， 在全世界范围内已经有41%的网站开启了HTTP/2。

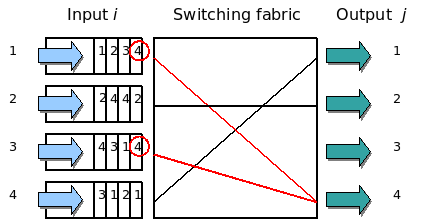
HTTP/2的官方组织在 Github 上维护了一份[各种语言对HTTP/2的实现列表](https://github.com/http2/http2-spec/wiki/Implementations)，大家可以去看看。

我们可以看到，HTTP/2 在性能上对HTTP有质的提高，所以，HTTP/2 被采用的也很快，所以，**如果你在你的公司内负责架构的话，HTTP/2是你一个非常重要的需要推动的一个事，除了因为性能上的问题，推动标准落地也是架构师的主要职责，因为，你企业内部的架构越标准，你可以使用到开源软件，或是开发方式就会越有效率，跟随着工业界的标准的发展，你的企业会非常自然的享受到标准所带来的红利。**

**HTTP/3**

然而，这个世界没有完美的解决方案，HTTP/2也不例外，其主要的问题是：若干个HTTP的请求在复用一个TCP的连接，底层的TCP协议是不知道上层有多少个HTTP的请求的，所以，一旦发生丢包，造成的问题就是所有的HTTP请求都必需等待这个丢了的包被重传回来，哪怕丢的那个包不是我这个HTTP请求的。因为TCP底层是没有这个知识了。

这个问题又叫[Head-of-Line Blocking](https://en.wikipedia.org/wiki/Head-of-line_blocking)问题，这也是一个比较经典的流量调度的问题。这个问题最早主要的发生的交换机上。下图来自Wikipedia。



图中，左边的是输入队列，其中的1，2，3，4表示四个队列，四个队列中的1，2，3，4要去的右边的output的端口号。此时，第一个队列和第三个队列都要写右边的第四个端口，然后，一个时刻只能处理一个包，所以，一个队列只能在那等另一个队列写完后。然后，其此时的3号或1号端口是空闲的，而队列中的要去1和3号端号的数据，被第四号端口给block住了。这就是所谓的HOL blocking问题。

HTTP/1.1中的pipeline中如果有一个请求block了，那么队列后请求也统统被block住了；HTTP/2 多请求复用一个TCP连接，一旦发生丢包，就会block住所有的HTTP请求。这样的问题很讨厌。好像基本无解了。

是的TCP是无解了，但是UDP是有解的 ！**于是HTTP/3破天荒地把HTTP底层的TCP协议改成了UDP！**

然后又是Google 家的协议进入了标准 – QUIC （Quick UDP Internet Connections）。接下来是QUIC协议的几个重要的特性，为了讲清楚这些特性，我需要带着问题来讲（注：下面的网络知识，如果你看不懂的话，你需要学习一下《[TCP/IP详解](https://book.douban.com/subject/1088054/)》一书（在我写blog的这15年里，这本书推荐了无数次了），或是看一下本站的《[TCP的那些事](https://coolshell.cn/articles/11564.html)》。）：

* 首先是上面的Head-of-Line blocking问题，在UDP的世界中，这个就没了。这个应该比较好理解，因为UDP不管顺序，不管丢包（当然，QUIC的一个任务是要像TCP的一个稳定，所以QUIC有自己的丢包重传的机制）
* TCP是一个无私的协议，也就是说，如果网络上出现拥塞，大家都会丢包，于是大家都会进入拥塞控制的算法中，这个算法会让所有人都“冷静”下来，然后进入一个“慢启动”的过程，包括在TCP连接建立时，这个慢启动也在，所以导致TCP性能迸发地比较慢。QUIC基于UDP，使用更为激进的方式。同时，QUIC有一套自己的丢包重传和拥塞控制的协，一开始QUIC是重新实现一TCP 的 CUBIC算法，但是随着BBR算法的成熟（BBR也在借鉴CUBIC算法的数学模型），QUIC也可以使用BBR算法。这里，多说几句，**从模型来说，以前的TCP的拥塞控制算法玩的是数学模型，而新型的TCP拥塞控制算法是以BBR为代表的测量模型**，理论上来说，后者会更好，但QUIC的团队在一开始觉得BBR不如CUBIC的算法好，所以没有用。现在的BBR 2.x借鉴了CUBIC数学模型让拥塞控制更公平。这里有文章大家可以一读“[TCP BBR : Magic dust for network performance.](https://medium.com/google-cloud/tcp-bbr-magic-dust-for-network-performance-57a5f1ccf437)”
* 接下来，现在要建立一个HTTPS的连接，先是TCP的三次握手，然后是TLS的三次握手，要整出六次网络交互，一个链接才建好，虽说HTTP/1.1和HTTP/2的连接复用解决这个问题，但是基于UDP后，UDP也得要实现这个事。于是QUIC直接把TCP的和TLS的合并成了三次握手（对此，在HTTP/2的时候，是否默认开启TLS业内是有争议的，反对派说，TLS在一些情况下是不需要的，比如企业内网的时候，而支持派则说，TLS的那些开销，什么也不算了）。

|  |  |
| --- | --- |
|  |  |

所以，QUIC是一个在UDP之上的伪TCP +TLS +HTTP/2的多路复用的协议。

但是对于UDP还是有一些挑战的，这个挑战主要来自互联网上的各种网络设备，这些设备根本不知道是什么QUIC，他们看QUIC就只能看到的就是UDP，所以，在一些情况下，UDP就是有问题的，

* 比如在NAT的环境下，如果是TCP的话，NAT路由或是代理服务器，可以通过记录TCP的四元组（源地址、源端口，目标地址，目标端口）来做连接映射的，然而，在UDP的情况下不行了。于是，QUIC引入了个叫connection id的不透明的ID来标识一个链接，用这种业务ID很爽的一个事是，如果你从你的3G/4G的网络切到WiFi网络（或是反过来），你的链接不会断，因为我们用的是connection id，而不是四元组。
* 然而就算引用了connection id，也还是会有问题 ，比如一些不够“聪明”的等价路由交换机，这些交换机会通过四元组来做hash把你的请求的IP转到后端的实际的服务器上，然而，他们不懂connection id，只懂四元组，这么导致属于同一个connection id但是四元组不同的网络包就转到了不同的服务器上，这就是导致数据不能传到同一台服务器上，数据不完整，链接只能断了。所以，你需要更聪明的算法（可以参看 Facebook 的 [Katran](https://github.com/facebookincubator/katran) 开源项目 ）

好了，就算搞定上面的东西，还有一些业务层的事没解，这个事就是 HTTP/2的头压缩算法 HPACK，HPACK需要维护一个动态的字典表来分析请求的头中哪些是重复的，HPACK的这个数据结构需要在encoder和decoder端同步这个东西。在TCP上，这种同步是透明的，然而在UDP上这个事不好干了。所以，这个事也必需要重新设计了，基于QUIC的QPACK就出来了，利用两个附加的QUIC steam，一个用来发送这个字典表的更新给对方，另一个用来ack对方发过来的update。

目前看下来，HTTP/3目前看上去没有太多的协议业务逻辑上的东西，更多是HTTP/2 + QUIC协议。但，HTTP/3 因为动到了底层协议，所以，在普及方面上可能会比 HTTP/2要慢的多的多。但是，可以看到QUIC协议的强大，细思及恐，QUIC这个协议真对TCP是个威胁，如果QUIC成熟了，TCP是不是会有可能成为历史呢？

未来十年，让我们看看UDP是否能够逆袭TCP……