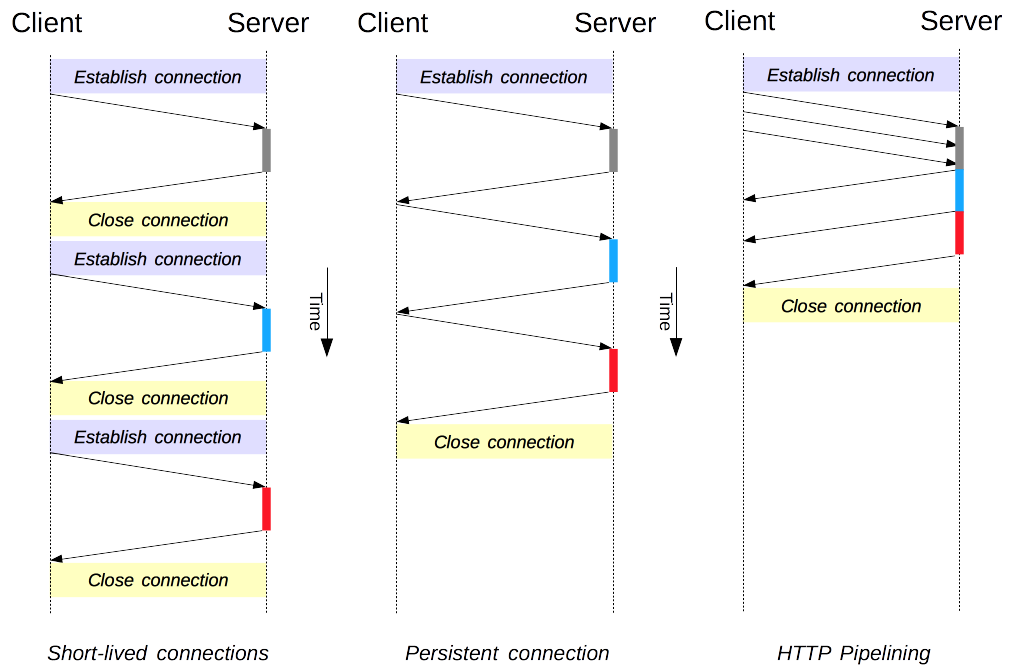
连接管理是一个 HTTP 的关键话题：打开和保持连接在很大程度上影响着网站和 Web 应用程序的性能。在 HTTP/1.x 里有好些个模型：*短连接*, *长连接*, 和 *HTTP 流水线。*

HTTP 的传输协议主要依赖于 TCP 来提供从客户端到服务器端之间的连接。在早期，HTTP 使用一个简单的模型来处理这样的连接。这些连接的生命周期是短暂的：每发起一个请求时都会创建一个新的连接，并在收到应答时立即关闭。

这个简单的模型对性能有先天的限制：打开每一个 TCP 连接都是相当耗费资源的操作。客户端和服务器端之间需要交换好些个消息。当请求发起时，网络延迟和带宽都会对性能造成影响。现代浏览器往往要发起很多次请求(十几个或者更多)才能拿到所需的完整信息，证明了这个早期模型的效率低下。

有两个新的模型在 HTTP/1.1 诞生了。首先是长连接模型，它会保持连接去完成多次连续的请求，减少了不断重新打开连接的时间。然后是 HTTP 流水线模型，它还要更先进一些，多个连续的请求甚至都不用等待立即返回就可以被发送，这样就减少了耗费在网络延迟上的时间。



HTTP/2 新增了其它连接管理模型。

要注意的一个重点是 HTTP 的连接管理适用于两个连续节点之间的连接，如 [hop-by-hop](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers#hbh)，而不是 [end-to-end](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers#e2e)。当模型用于从客户端到第一个代理服务器的连接和从代理服务器到目标服务器之间的连接时(或者任意中间代理)效果可能是不一样的。HTTP 协议头受不同连接模型的影响，比如 [Connection](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Connection) 和 [Keep-Alive](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Keep-Alive)，就是 [hop-by-hop](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers#hbh) 协议头，它们的值是可以被中间节点修改的。

**短连接**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Connection_management_in_HTTP_1.x$edit#%E7%9F%AD%E8%BF%9E%E6%8E%A5)

HTTP 最早期的模型，也是  HTTP/1.0 的默认模型，是短连接。每一个 HTTP 请求都由它自己独立的连接完成；这意味着发起每一个 HTTP 请求之前都会有一次 TCP 握手，而且是连续不断的。

TCP 协议握手本身就是耗费时间的，所以 TCP 可以保持更多的热连接来适应负载。短连接破坏了 TCP 具备的能力，新的冷连接降低了其性能。

这是 HTTP/1.0 的默认模型(如果没有指定 [Connection](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Connection) 协议头，或者是值被设置为 close)。而在 HTTP/1.1 中，只有当 [Connection](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Connection) 被设置为 close 时才会用到这个模型。

除非是要兼容一个非常古老的，不支持长连接的系统，没有一个令人信服的理由继续使用这个模型。

**长连接**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Connection_management_in_HTTP_1.x$edit#%E9%95%BF%E8%BF%9E%E6%8E%A5)

短连接有两个比较大的问题：创建新连接耗费的时间尤为明显，另外 TCP 连接的性能只有在该连接被使用一段时间后(热连接)才能得到改善。为了缓解这些问题，*长连接* 的概念便被设计出来了，甚至在 HTTP/1.1 之前。或者这被称之为一个 *keep-alive*连接。

一个长连接会保持一段时间，重复用于发送一系列请求，节省了新建 TCP 连接握手的时间，还可以利用 TCP 的性能增强能力。当然这个连接也不会一直保留着：连接在空闲一段时间后会被关闭(服务器可以使用 [Keep-Alive](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Keep-Alive) 协议头来指定一个最小的连接保持时间)。

长连接也还是有缺点的；就算是在空闲状态，它还是会消耗服务器资源，而且在重负载时，还有可能遭受 [DoS attacks](https://developer.mozilla.org/en-US/docs/Glossary/DoS_attack) 攻击。这种场景下，可以使用非长连接，即尽快关闭那些空闲的连接，也能对性能有所提升。

HTTP/1.0 里默认并不适用长连接。把 [Connection](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Connection) 设置成 close 以外的其它参数都可以让其保持长连接，通常会设置为 retry-after。

在 HTTP/1.1 里，默认就是长连接的，协议头都不用再去声明它(但我们还是会把它加上，万一某个时候因为某种原因要退回到 HTTP/1.0 呢)。

**HTTP 流水线**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Connection_management_in_HTTP_1.x$edit#HTTP_%E6%B5%81%E6%B0%B4%E7%BA%BF)

HTTP 流水线在现代浏览器中并不是默认被启用的：

* Web 开发者并不能轻易的遇见和判断那些搞怪的[代理服务器](https://en.wikipedia.org/wiki/Proxy_server)的各种莫名其妙的行为。
* 正确的实现流水线式复杂的：传输中的资源大小，多少有效的 [RTT](https://en.wikipedia.org/wiki/Round-trip_delay_time) 会被用到，还有有效带宽，流水线带来的改善有多大的影响范围。不知道这些的话，重要的消息可能被延迟到不重要的消息后面。这个重要性的概念甚至会演变为影响到页面布局！因此 HTTP 流水线在大多数情况下带来的改善并不明显。
* 流水线受制于 [HOL](https://en.wikipedia.org/wiki/Head-of-line_blocking) 问题。

由于这些原因，流水线已经被更好的算法给代替，如 *multiplexing*，已经用在 HTTP/2。

默认情况下，[HTTP](https://developer.mozilla.org/en/HTTP) 请求是按顺序发出的。下一个请求只有在当前请求收到应答过后才会被发出。由于会受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长时间。

流水线是在同一条长连接上发出连续的请求，而不用等待应答返回。这样可以避免连接延迟。理论上讲，性能还会因为两个 HTTP 请求有可能被打包到一个 TCP 消息包中而得到提升。就算 HTTP 请求不断的继续，尺寸会增加，但设置 TCP 的 [MSS](https://en.wikipedia.org/wiki/Maximum_segment_size)(Maximum Segment Size) 选项，任然足够包含一系列简单的请求。

并不是所有类型的 HTTP 请求都能用到流水线：只有 [idempotent](https://developer.mozilla.org/en-US/docs/Glossary/idempotent) 方式，比如 [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET)、[HEAD](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/HEAD)、[PUT](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/PUT) 和 [DELETE](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/DELETE) 能够被安全的重试：如果有故障发生时，流水线的内容要能被轻易的重试。

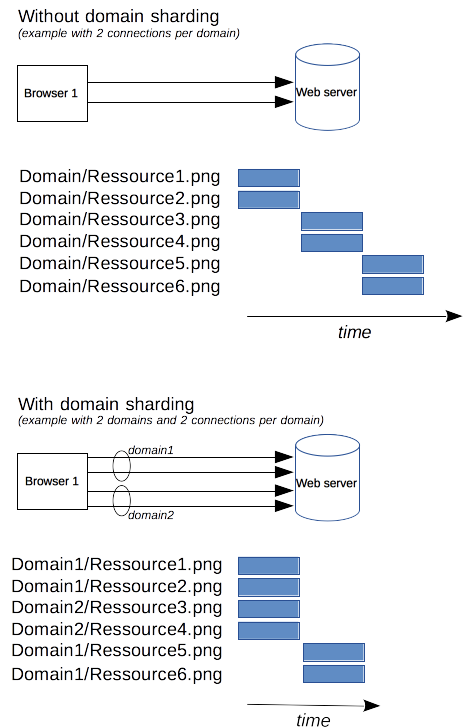
今天，所有遵循 HTTP/1.1 的代理和服务器都应该支持流水线，虽然实际情况中还是有很多限制：一个很重要的原因是，任然没有现代浏览器去默认支持这个功能。

**域名分片**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Connection_management_in_HTTP_1.x$edit#%E5%9F%9F%E5%90%8D%E5%88%86%E7%89%87)

除非你有紧急而迫切的需求，不要使用这一过时的技术，升级到 HTTP/2 就好了。在 HTTP/2 里，做域名分片就没必要了：HTTP/2 的连接可以很好的处理并发的无优先级的请求。域名分片甚至会影响性能。大多数 HTTP/2 的实现还会使用一种称作[连接凝聚](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/I%20wonder%20if%20it's%20related%20to%20the%20nobash/nobreak/nopick%20secret%20exit%20s%20of%20Elrond's%20chambers.)的技术去尝试合并被分片的域名。

作为 HTTP/1.x 的连接，请求时序列化的，哪怕本来是无序的，在没有足够庞大可用的带宽时，也无从优化。一个解决方案是，浏览器为每个域名建立多个连接，以实现并发请求。曾经默认的连接数量为 2 到 3 个，现在比较常用的并发连接数已经增加到 6 条。如果尝试大于这个数字，就有触发服务器 DoS 保护的风险。

如果服务器端想要更快速的响应网站或应用程序的应答，它可以迫使客户端建立更多的连接。例如，不要在同一个域名下获取所有资源，假设有个域名是 www.example.com，我们可以把它拆分成好几个域名：www1.example.com、www2.example.com、www3.example.com。所有这些域名都指向同一台服务器，浏览器会同时为每个域名建立 6 条连接(在我们这个例子中，连接数会达到 18 条)。这一技术被称作域名分片。



**结论**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Connection_management_in_HTTP_1.x$edit#%E7%BB%93%E8%AE%BA)

改进后的连接管理极大的提升了 HTTP 的性能。不管是 HTTP/1.1 还是 HTTP/1.0，使用长连接 – 直到进入空闲状态 – 都能达到最佳的性能。然而，解决流水线故障需要设计更先进的连接管理模型，HTTP/2 已经在尝试了。