本文来说一说HTTP中的连接管理，本文核心点如下：

一、HTTP时延损耗分析和短连接

当我们在浏览器输入一个地址后，将主要经历以下几个阶段：

- DNS域名解析；
- 客户端和服务端进行TCP三次握手建立连接；
- 请求报文进行路由，服务端收到请求报文后，解析、处理，并给出响应报文；
- 响应报文经过路由返回到客户端，客户端进行解析、渲染以及展示页面；
- 由于不需要再请求，由客户端发起四次挥手，断开TCP连接；

整个过程如下：

我们可以清晰看到时间损耗的组成，我们学习过HTTP的前世今生，知道早期版本HTTP/0.9的一个特性是：服务器发送完毕，就关闭TCP连接（一个HTTP请求在一个TCP连接中完成）。

那时候是互联网初期，计算机的处理能力包括网速等等都很弱，所以 HTTP 也逃脱不了那个时代的约束，因此设计的非常简单，而且也是纯文本格式。

李老当时的想法是文档存在服务器里面，我们只需要从服务器获取文档，因此只有“GET”，也不需要啥请求头，并且拿完了就结束了，因此请求响应之后连接就断了。

这就是为什么 HTTP 设计为文本协议，并且一开始只有“GET”、响应之后连接就断了的原因了。

我们将这种模式称为短连接，这是相对于下面要学习的长连接而言的，我们来看看短连接有什么问题。

二、串行短连接

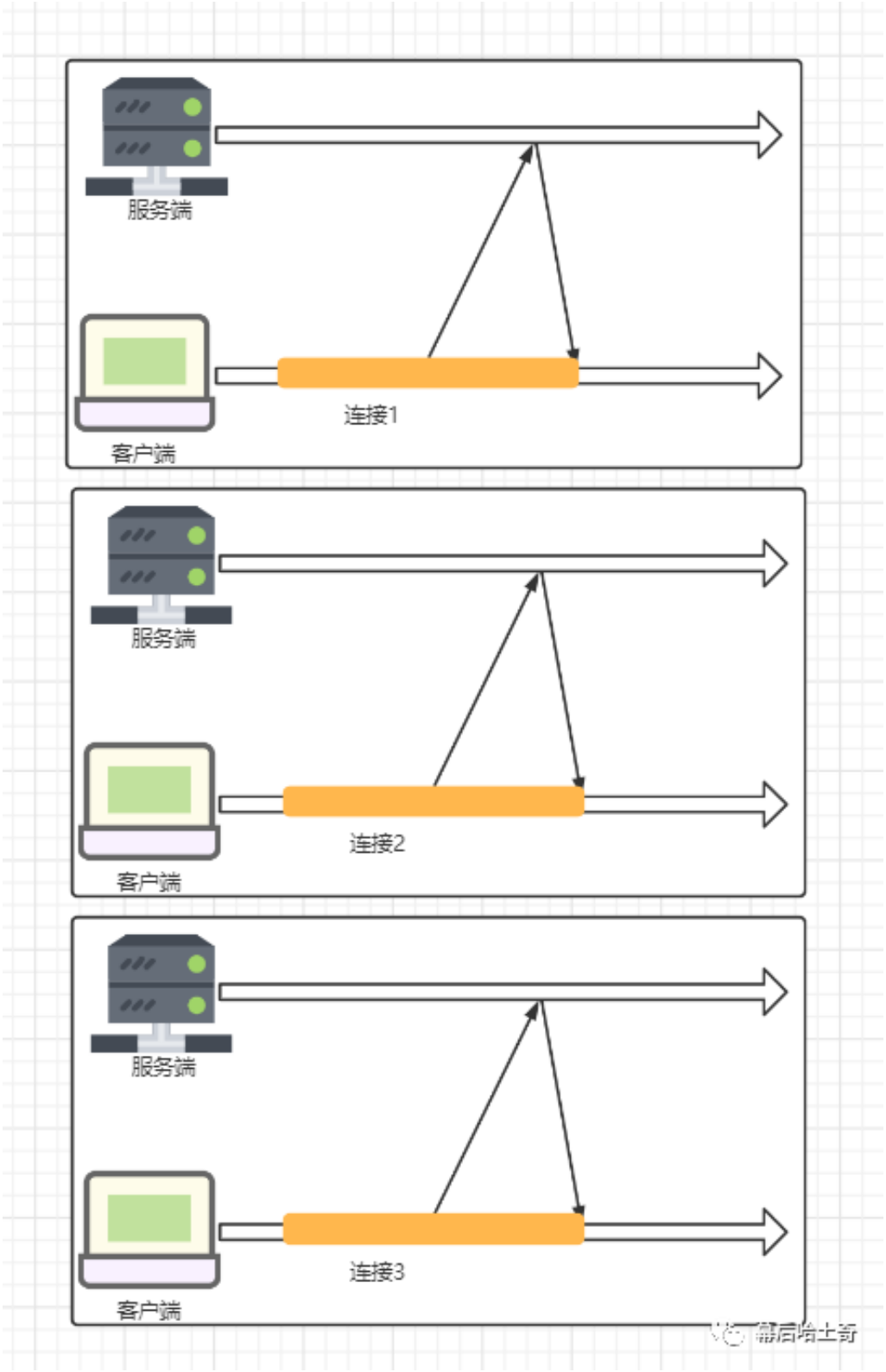
试想一个问题，假设一个页面有五个资源（元素），每个资源都需要客户端打开一个 TCP 连接、获取资源、断开连接，而且每个连接都是串行打开的，如下图所示：

假设页面所需要请求的资源达到上百甚至上千个呢？这个”排队“的队伍是不是有点太长了？万一前面”排队“的哥们迟迟不结束，后面的哥们将排到什么时候？

还有一个问题是，有的时候浏览器需要加载足够多的资源才会开始显示内容，这可能会导致页面渲染很慢，造成用户以为卡住的疑惑。

容易想到，我是不是可以改为并行请求？

三、并行短连接



采用并行连接的方式，可以解决”排队“慢的重大问题，在带宽足够大的情况下会显著提高请求的速率，为什么这么说呢?

虽然是并行，但是每次也都是建立新的连接，每个连接都会去竞争使用有效的带宽，如果带宽不够，可能还不如一个个串行去请求来的快。

此外，打开大量连接会消耗很多内存资源，从而出现性能问题，比如一个客户端可以打开数百个连接，当有无数个这样的客户端来向服务端请求呢？若是同时请求，服务端的内存、带宽都将立刻被打满，服务器将由于太忙碌而进入宕机状态，这样的话，还不如串行连接模式了，毕竟有总比没有强嘛。

可见，并行连接并不一定"快"，使用不当甚至还会带来很大的副作用。

四、长连接

我们能不能有一个折中的办法？大牛们想到了长连接。

我们目前采取的是短连接模式，即每次HTTP的一个请求-响应结束后，即关闭此TCP连接，当有下一个HTTP请求时，重新建立连接和断开连接。

短连接是在HTTP/1.0及以前版本中的默认模型，其过程为：

> 建立连接—数据传输—关闭连接...建立连接—数据传输—关闭连接

假设一个页面有很多的资源要加载，频繁地建立连接-断开连接，极其消耗时间，也没有必要。

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，从HTTP/1.1起，默认使用长连接。

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的 TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。

> 建立连接—数据传输...（保持连接） ...数据传输—关闭连接

这个长连接的持续时间是可以在服务端进行设置的，避免连接长期占用而无活动，白白浪费了服务器资源。

串行短连接VS长连接：

长连接可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用长连接。

长连接还有一个优势，由于TCP协议拥有“慢启动”、“拥塞窗口”等特性，通常新建立的“冷连接”会比打开了一段时间的“热连接”要慢一些。

不过长连接同样也存在问题，在长连接的应用场景下，client端一般不会主动关闭它们之间的连接，Client与server之间的连接如果一直不关闭的话，会存在一个问题，随着客户端连接越来越多，server早晚有扛不住的时候。

这时候server端需要采取一些策略，如关闭一些长时间没有读写事件发生的连接，这样可以避免一些恶意连接导致server端服务受损；如果条件再允许就可以以客户端机器为颗粒度，限制每个客户端的最大长连接数，这样可以完全避免某个蛋疼的客户端连累后端服务。

五、长连接相关头部字段

由于长连接对性能的改善效果非常显著，所以在 HTTP/1.1 中的连接都会默认启用长连接，只要向服务器发送了第一次请求，后续的请求都会重复利用第一次打开的 TCP 连接，也就是长连接，在这个连接上收发数据。

对应到请求头的字段，使用的字段是Connection，值是“keep-alive”。

不过不管客户端是否显式要求长连接，如果服务器支持长连接，它总会在响应报文里放一个“Connection: keep-alive”字段，告诉客户端：“我是支持长连接的，接下来就用这个 TCP 一直收发数据吧”。

如果不想使用长连接，则需要在报文中显式地添加 Connection: close 首部。

六、管道化连接(HTTP pipelining)

默认情况下，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。

HTTP/1.1 允许在持久连接上使用管道技术，这是相对于 Keep-Alive 连接的又一个性能优化。

管道就是一个承载 HTTP 请求的载体，我们可以将多个 HTTP 请求放入管道连续发送出去，而不用等待应答返回。

下图是使用串行短连接、长连接、管道化连接的示意图：

可以明显看出来，管道化连接明显又快了一点，不过使用管道化连接是有条件的。

- 如果 HTTP 客户端无法确认连接是持久的，就不应该使用管道。
- Pipelining is only supported in HTTP/1.1, not in 1.0。HTTP管线化是对HTTP1.1协议下，服务器不能很好处理并行请求的一个改进。
- 必须按照与请求的相同顺序回送 HTTP 响应，因为 HTTP 没有序号这个概念，所以一旦响应失序，就没办法将其与请求匹配起来了。
- （考验下英文）Non-idempotent(非幂等) methods like POST should not be pipelined. Sequences of GET and HEAD requests can be always pipelined. A sequence of other idempotent requests like GET, HEAD, PUT and DELETE can be pipelined or not depending on whether requests in the sequence depend on the effect of others.

HTTP 客户端必须做好连接会在任何时刻关闭的准备，还要准备好重发所有未完成的管道化请求，一条船上的蚂蚱，生死与共！

不过实际上管道化这个技术并不常用，原因是：**HTTP pipelining is disabled in most browsers**。

七、队头阻塞(Head-of-line blocking)

看完了短连接和长连接，接下来就要说到著名的“队头阻塞”（Head-of-line blocking，也叫“队首阻塞”）了。

“队头阻塞”与短连接和长连接无关，而是由 HTTP 基本的“请求 - 应答”模型所导致的。

因为 HTTP 规定报文必须是“一发一收”，这就形成了一个先进先出的“串行”队列。队列里的请求没有轻重缓急的优先级，只有入队的先后顺序，排在最前面的请求被最优先处理。

如果队首的请求因为处理的太慢耽误了时间，那么队列里后面的所有请求也不得不跟着一起等待，结果就是其他的请求承担了不应有的时间成本。

因为“请求 - 应答”模型不能变，所以“队头阻塞”问题在 HTTP/1.1 里无法解决，只能缓解，比如以上提到的浏览器发起多个并发连接请求。

但是我们也明白不能滥用并发，RFC2616 里明确限制每个客户端最多并发 2 个连接。不过实践证明这个数字实在是太小了，众多浏览器都“无视”标准，把这个上限提高到了 6~8。后来修订的 RFC7230 也就“顺水推舟”，取消了这个“2”的限制。

还有一种技术叫做域名分片。

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)。

不过这一技术并不推荐，已经“过时”了。

HTTP/1.1中队头阻塞问题实际上无法很好解决，最多是缓解一点，如何“完美”解决队头阻塞问题就是HTTP/2以及HTTP/3优化的重要方向。