第14讲讲HTTP协议：看个新闻原来这么麻烦

http是应用层的协议，基于TCP。

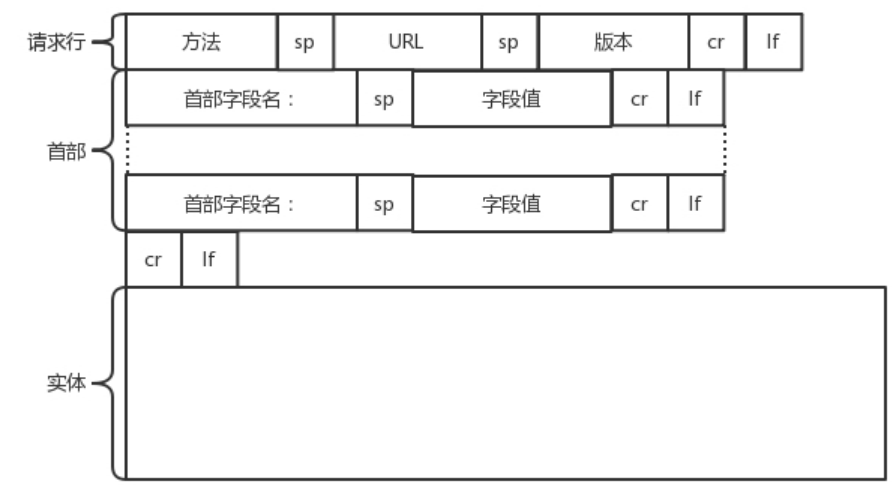
URL，叫作**统一资源定位符**。

HTTP请求的准备

发送域名给DNS。

http协议大部分是1.1版本，默认开启keep-alive，建立的tcp连接可以在多次请求中复用。

HTTP请求的构建



HTTP请求大概分为三部分。第一部分是**请求行**，第二部分是请求的**首部**，第三部分才是请求的**正文实体**。

**第一部分：请求行。**

字段方法有如下几种：

**最常用的类型就是GET。**获取资源。返回例如页面，json字段。到底返回什么，由服务端决定。

**另外一种类型是POST。告诉服务器一些信息。**支付场景中的“我是谁？我要支付多少？我要买啥？ ”告诉服务器。

**PUT类型。**向指定资源位置上传最新内容。但是HTTP服务器往往不允许上传文件，所以PUT和POST都变成了要传给服务器东西的方法。

实际使用中，**POST往往是用来创建一个资源的，而PUT往往是用来修改一个资源的。**

**DELETE类型**。这个顾名思义就是用来删除资源的。例如，我们要删除一个云主机，就会调用DELETE方法。

**第二部分：请求首部**

首部是key value，通过冒号分隔。保存了一些非常重要的字段。

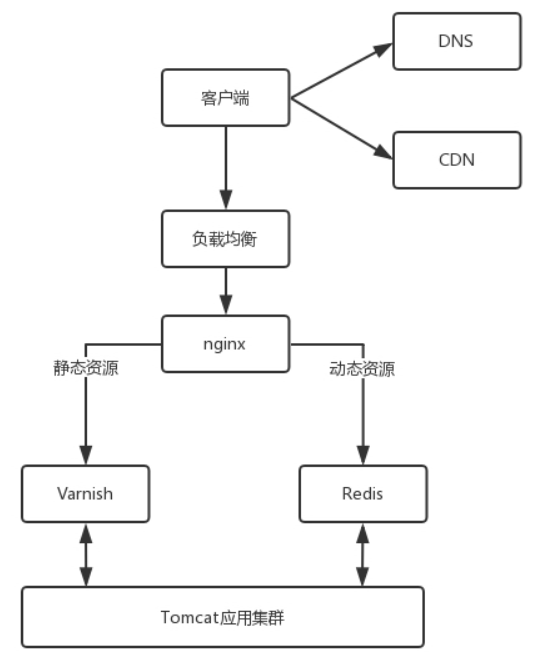
**Accept-Charset**，表示**客户端可以接受的字符集**。防止传过来的是另外的字符集，从而导致出现乱码。

**Content-Type**是指**正文的格式**。例如，我们进行POST的请求，如果正文是JSON，那么我们就应该将这个值设置为JSON。

**缓存：**

使用场景：浏览商品详情，图片价格等经常不变，而库存动态变化。这是火就可以将长时间保持不变的缓存起来。如果图片非常大，而库存数非常小，如果我们每次要更新数据的时候都要刷新整个页面，对于服务器的压力就会很大。

**对于这种高并发场景下的系统，在真正的业务逻辑之前，都需要有个接入层，将这些静态资源的请求拦在最外面。**



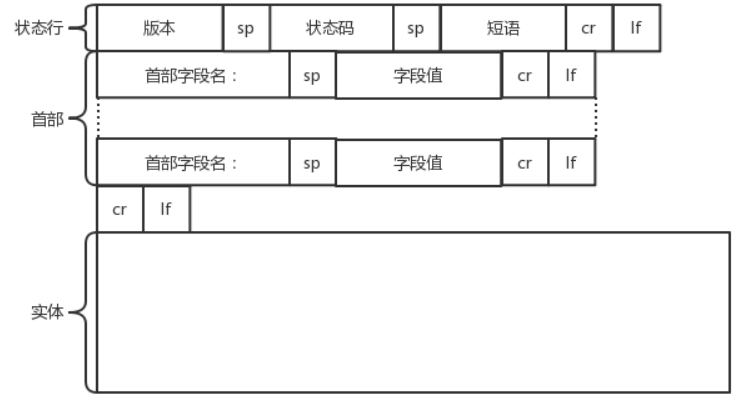
Nginx这一层负责接入层的工作，对于静态资源由Vanish缓存层。当缓存过期的时候，才会访问真正的Tomcat 应用集群。

HTTP请求的发送

由于基于TCP协议，通过二进制流的方式发送给对方，到了目标地址的TCP层，又会把二进制流转换为报文段发送给服务器。

每个报文段，对方都有一个回应ACK，保证报文到达可靠的地方。在网络传输的过程，就是发送ARP请求然后路由接收并转发到下一跳的过程。

HTTP返回的构建



状态码会反馈HTTP请求的结果。 “200”意味着大吉大利；而我们最不想见的，就是“404”，也就是“服务端无法响应这个请求”。

接下来是返回首部的**key value**。

**Retry-After**表示，告诉客户端应该在多长时间以后再次尝试一下。

头部里面也会有**Content-Type**，表示返回的是HTML，还是JSON。

HTTP 2.0

**最显著的特点是可以建立一个连接发送多个请求，有效的提升了效率。**

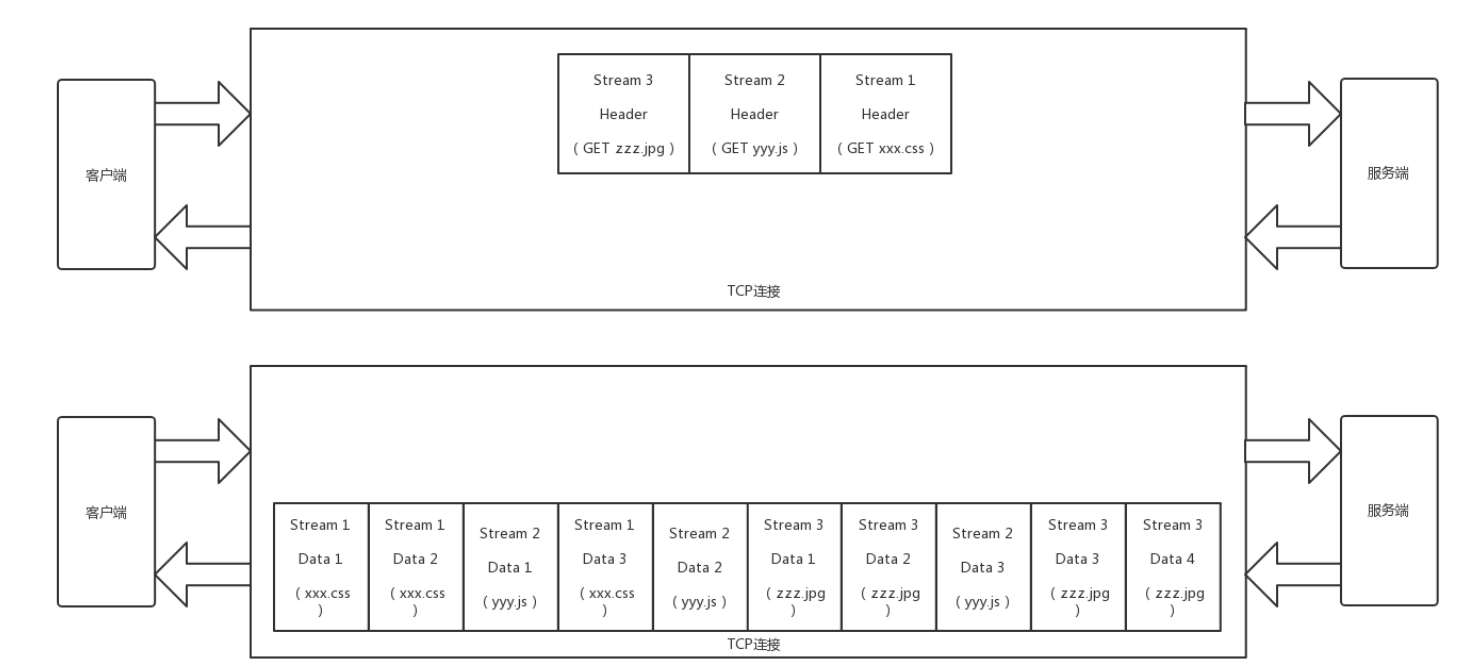
HTTP1.0在实时性，并发性上都存在问题。因为每次发送请求都要传送相同的HTTP首部。HTTP 2.0会对HTTP的头进行一定的压缩，将原来每次都要携带的大量key value在两端建立一个索引表，对相同的头只发送索引表中的索引，而不发送value。

HTTP 2.0协议将一个TCP的连接中**，切分成多个流**，每个流都有自己的ID，流是有优先级的，实现了文本可以乱序发送。

HTTP 2.0还将所有的传输信息分割为更小的消息和帧，并对它们采用二进制格式编码。

HTTP 2.0的客户端可以将多个请求分到不同的流中，然后将请求内容拆成帧，进行二进制传输。这些**帧可以打散乱序发送， 然后根据每个帧首部的流标识符重新组装，**并且可以根据优先级，决定优先处理哪个流的数据。

**一个页面要发送三个独立的请求，一个获取css，一个获取js，一个获取图片jpg。如果使用HTTP 1.1就是串行的，但是如果使用HTTP 2.0。**



小总结：

HTTP2.0的两大特性：将首部压缩传输（某些相同的索引不传输value）。一个数据分流传输。

HTTP 2.0成功解决了HTTP 1.1的队首阻塞问题（因为是串行发送的），同时，也不需要通过HTTP 1.x的pipeline机制用多条TCP连接来实现并行请求与响应；减少了TCP连接数对服务器性能的影响，同时将页面的多个数据css、 js、 jpg等通过一个数据TCP连接进行传输，能够加快页面组件的传输速度。

QUIC协议

HTTP2.0是基于TCP协议的，假如一前一后，前面stream 2的帧没有收到，后面stream 1的帧也会因此阻塞。因为他也是基于TCP协议的，TCP 协议处理包的时候有严格的时序。

Google发明的一种传输机制。同样是面向连接的。全称是quick udp internet connection。

就从TCP切换到UDP。

一条TCP连接是由四元组标识的，分别是源 IP、源端口、目的 IP、目的端口。一旦一个元素发生变化时，就需要断开重连，重新连接。在移动互联情况下，当手机信号不稳定或者在WIFI和 移动网络切换时，都会导致重连，从而进行再次的三次握手，导致一定的时延。

QUIC的机制。

**机制一：自定义连接机制**

使用UDP，结合QUIC自己的逻辑维护连接机制：不再以四元组标识，而是以一个64位的随机数作为ID来标识，而且UDP是无连接的，所以当IP或者端口变化的时候，只要ID不变，就不需要重新建立连接。

**机制二：自定义重传机制**

TCP为了保证可靠性，通过使用**序号**和**应答**机制，来解决顺序问题和丢包问题。

传统TCP连接协议通过自适应重传算法来计算超时的，超时通过采样往返时间RTT不断调整的。RTT的计算是通过计算发送一个包，序号为100，发现没有返回，于是再发送一个100，过一阵返回一个ACK101。这个时候客户端知道这个包肯定收到了，但是往返时间是多少呢？是ACK到达的时间减去后一个100发送的时间，还是减去前一个100发送的时间呢？事实是，第一种算法把时间算短了，第二种算法把时间算长了。

QUIC也有个序列号，是递增的。任何一个序列号的包只发送一次，下次就要加一了。例如，发送一个包，序号是100，发现没有返回；再次发送的时候，序号就是101了；如果返回的ACK 100，就是对第一个包的响应。如果返回ACK 101就是对第二个包的响应， RTT计算相对准确。为了能够区别包100和包101发送的是否是两个包，使用offset来判断是否是发送的是同一个包。

**机制三：无阻塞的多路复用**

核心是通过UDP避免TCP发送不成功的包导致阻塞，但是通过刚才说的序号机制保证重传达到不丢包的效果。

QUIC是基于UDP的，一个连接上的多个stream之间没有依赖。这样，假如stream2丢了一个UDP包，后面跟着stream3的一个UDP包，虽然stream2的那个包需要重传，但是stream3的包无需等待，就可以发给用户。

**机制四：自定义流量控制**

这里没有看，因为TCP的滑动窗口协议也没有看。

总结

* HTTP协议虽然很常用，也很复杂，重点记住GET、 POST、 PUT、 DELETE这几个方法，以及重要的首部字段；
* HTTP 2.0通过头压缩、分帧、二进制编码、多路复用等技术提升性能；
* QUIC协议通过基于UDP自定义的类似TCP的连接、重试、多路复用、流量控制技术，进一步提升性能。