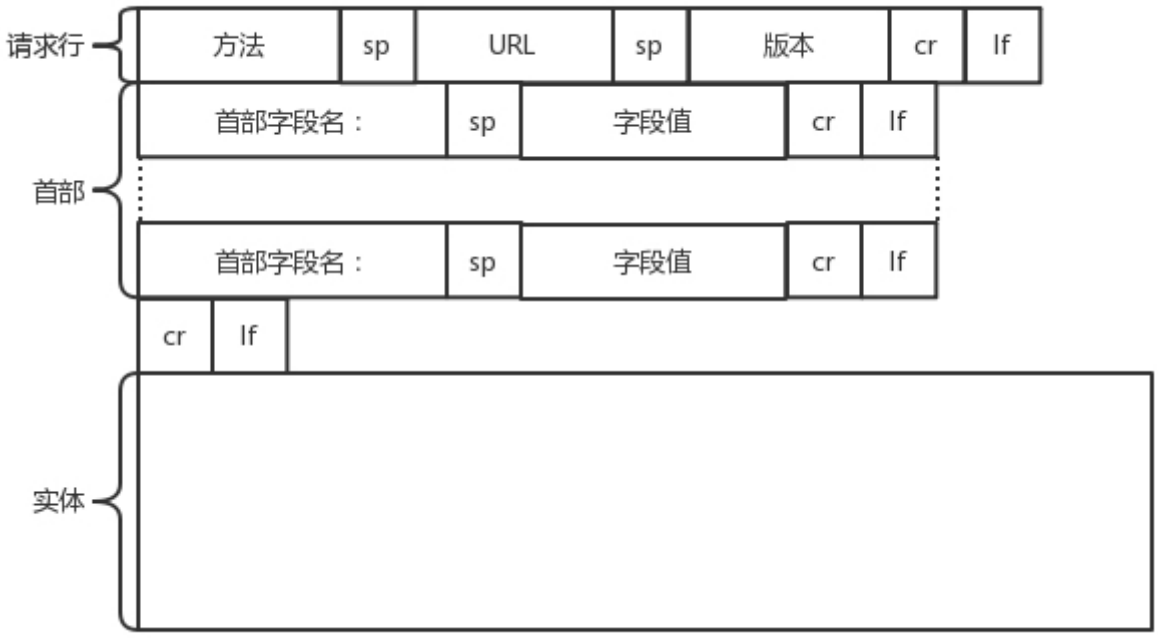
HTTP 协议

访问 www.163.com 这个域名
发送给 DNS 服务器，让它解析为 IP 地址。
HTTP 是基于 TCP 协议的，要先建立 TCP 连接了
目前使用的 HTTP 协议大部分都是 1.1。在 1.1 的协议里面，默认是开启了 **Connection：**Keep-Alive（**Connection** 头决定当前的事务完成后，是否会关闭网络连接。如果该值是“keep-alive”，网络连接就是持久的，不会关闭，使得对同一个服务器的请求可以继续在该连接上完成。） 的，这样建立的 TCP 连接，就可以在多次请求中复用。

HTTP 请求

建立了连接以后，浏览器就要发送 HTTP 的请求。

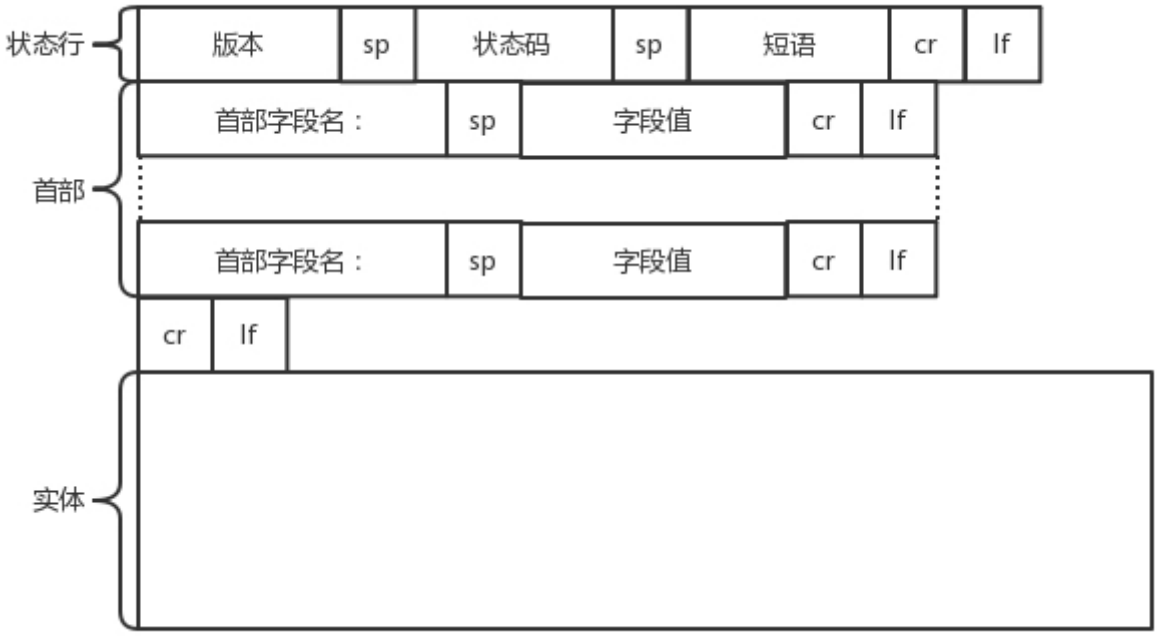


第一部分是请求行，第二部分是请求的首部，第三部分才是请求的正文实体。

首部字段

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

HTTP 返回



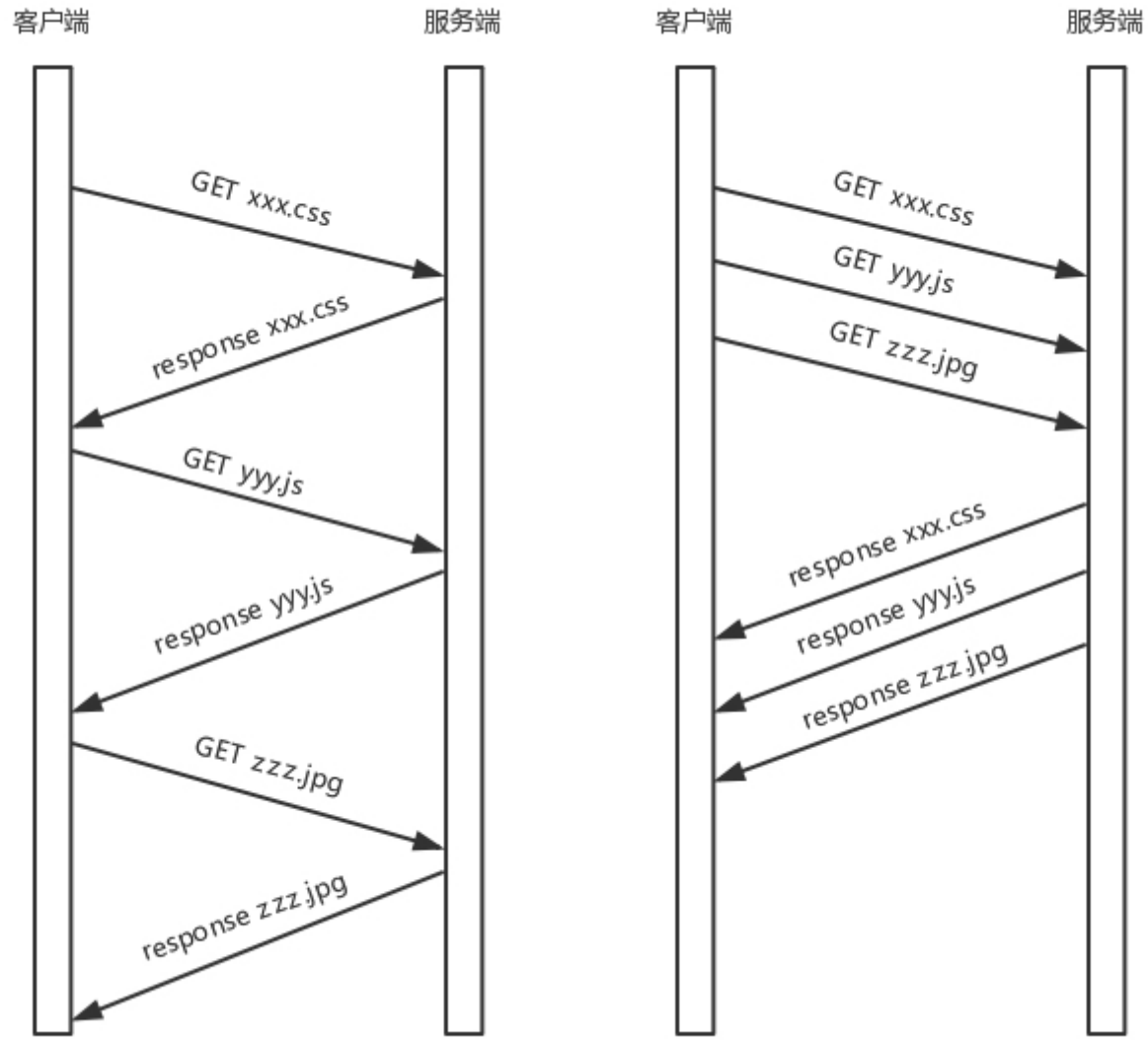
HTTP 2.0

HTTP 1.1 在应用层以纯文本的形式进行通信。每次通信都要带完整的 HTTP 的头，每次的过程总是像上面描述的那样一去一回。这样在实时性、并发性上都存在问题。
头部压缩：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。

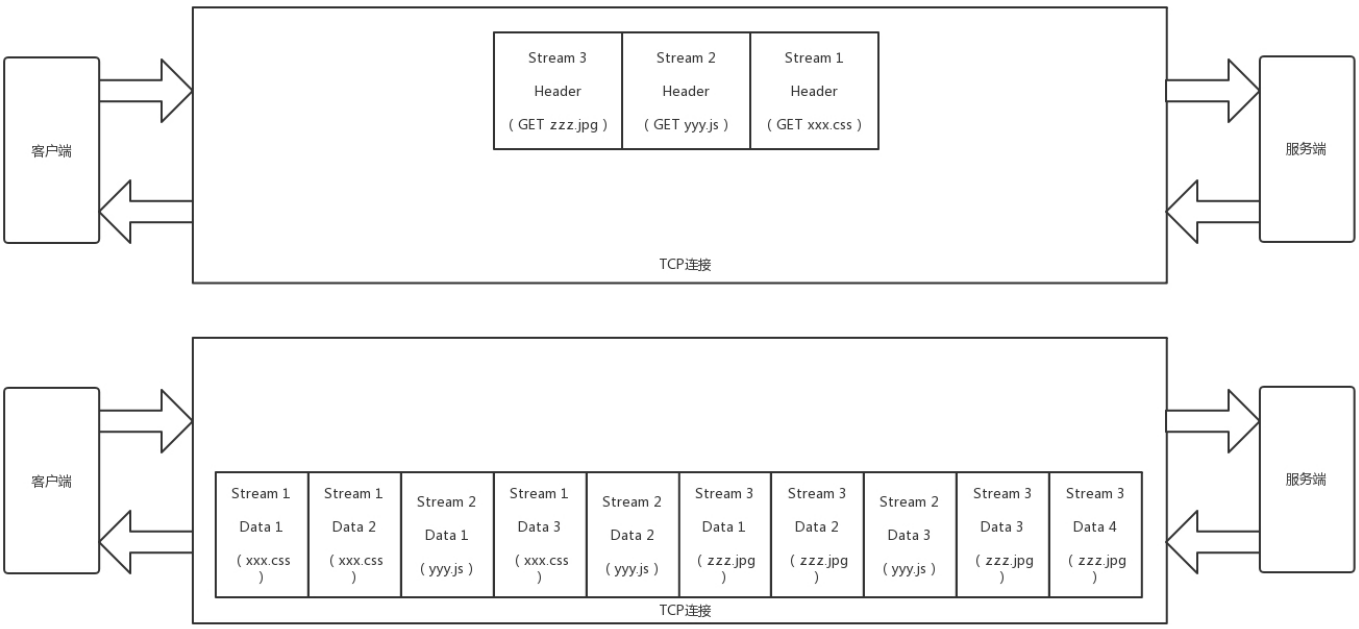
- **多路复用：**HTTP 2.0 协议将一个 TCP 的连接中，切分成多个流，每个流都有自己的 ID，而且**流可以是客户端发往服务端，也可以是服务端发往客户端（服务端推送）**。流是有 **优先级** 的。
- **分帧、二进制编码：**HTTP 2.0 还将所有的传输信息分割为更小的消息和帧，并对它们采用二进制格式编码。常见的帧有 Header 帧，用于传输 Header 内容，并且会开启一个新的流。再就是 Data 帧，用来传输正文实体，多个 Data 帧属于同一个流。

通过这两种机制，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。

我们来举一个例子：假设我们的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，就可以在一个连接里，客户端和服务端都可以同时发送多个请求或回应，而且不用按照顺序一对一对应。



HTTP 2.0 其实是将三个请求变成三个流，将数据分成帧，乱序发送到一个 TCP 连接中。



HTTP 2.0 成功解决了 HTTP 1.1 的 **队首阻塞** 问题（**服务端必须按照请求发送的顺序返回响应，那如果一个响应返回延迟了，那么其后续的响应都会被延迟，直到队头的响应送达**），同时将页面的多个数据 css、js、 jpg 等通过一个数据链接进行传输，能够加快页面组件的传输速度。

http2.0 很完美吗？？

TCP 的 队首阻塞：因为 HTTP 2.0 也是基于 TCP 协议的，TCP 协议在处理包时是有严格顺序的。当其中一个数据包遇到问题，TCP 连接需要等待这个包完成重传之后才能继续进行。

QUIC 协议

基于 UDP

- 自定义连接机制
- 自定义重传机制
- 自定义流量控制