## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "osi%E4%B8%83%E5%B1%82%E6%A8%A1%E5%9E%8B)OSI七层模型

****物理层****

首先解决两台物理机之间的通信需求，具体就是机器A往机器B发送比特流，机器B能收到比特流。

物理层主要定义了物理设备的标准，如网线的类型，光纤的接口类型，各种传输介质的传输速率。

主要作用是传输比特流（0101二进制数据），将比特流转化为电流强弱传输，到达目的后再转化为比特流，即常说的数模转化和模数转换。

这层数据叫做比特。****网卡工作在这层****。

物理层是OSI七层模型的物理基础，没有它就谈不上数据传输了

物理层就是由实物所承载的，所以作比喻的话，公路、汽车和飞机等承载货物（数据）的交通工具，就是物理层的象征

****数据链路层****

在传输比特流的过程中，会产生错传、数据传输不完整的可能。

数据链路层定义了****如何格式化数据进行传输****，以及如何控制对物理介质的访问。通常提供错误检测和纠正，以确保数据传输的准确性。

本层将比特数据组成帧，交换机工作在这层，对帧解码，并根据帧中包含的信息把数据发送到正确的接收方。

该层负责物理层面上互连的节点之间的通信传输。例如与1个以太网相连的两个节点间的通讯。

常见的协议有 HDLC、PPP、SLIP等

数据链路层会将0、1序列划分为具有意义的数据帧传送给对端（****数据帧的生成与接收****）

****网络层****

随着网络节点的不断增加，点对点通讯需要通过多个节点，如何找到目标节点，如何选择最佳路径成为首要需求。

网络层主要功能是将网络地址转化为对应的物理地址，并决定如何将数据从发送方路由到接收方。

网络层通过综合考虑发送优先权、网络拥塞程度、服务质量以及可选路由的花费来决定从一个网络中节点A到另一个网络中节点B的最佳路径。

由于网络层处理并智能指导数据传送，路由器连接网络隔断，所以路由器属于网络层。

此层的数据称之为数据包。本层需要关注的协议TCP/IP协议中的IP协议。

网络层负责将数据传输到目标地址。目标地址可以使多个网络通过路由器连接而成的某一个地址。因此这一层主要负责****寻址和路由选择****。主要由 IP、ICMP 两个协议组成

网络层将数据从发送端的主机发送到接收端的主机，两台主机间可能会存在很多数据链路，但网络层就是负责找出一条相对顺畅的通路将数据传递过去。传输的地址使用的是IP地址。IP地址通过不断转发到更近的IP地址，最终可以到达目标地址

****传输层****

随着网络通信需求的进一步扩大，通信过程中需要发送大量的数据，如海量文件传输，可能需要很长时间，网络在通信的过程中会中断很多次，此时为了保证传输大量文件时的准确性，需要对发送出去的数据进行切分，切割为一个一个的段落（Segement）发送，其中一个段落丢失是否重传，段落是否按顺序到达，是传输层需要考虑的问题。

传输层解决了主机间的数据传输，数据间的传输可以是不同网络，并且传输层解决了****传输质量****的问题。

传输层需要关注的协议有TCP/IP协议中的TCP协议和UDP协议。

****会话层****

自动收发包，自动寻址。

会话层作用是****负责建立和断开通信连接****，何时建立，断开连接以及保持多久的连接。常见的协议有 ADSP、RPC 等

****表示层****

Linux给WIndows发包，不同系统语法不一致，如exe不能在Linux下执行，shell不能在Windows不能直接运行。于是需要表示层。

解决****不同系统之间通信语法问题****，在表示层数据将按照网络能理解的方案进行格式化，格式化因所使用网络的不同而不同。

它主要负责数据格式的转换。具体来说，就是讲设备固有的数据格式转换为网络标准格式。常见的协议有ASCII、SSL/TLS 等

****应用层****

规定发送方和接收方必须使用一个固定长度的消息头，消息头必须使用某种固定的组成，消息头中必须记录消息体的长度等信息，方便接收方正确解析发送方发送的数据。

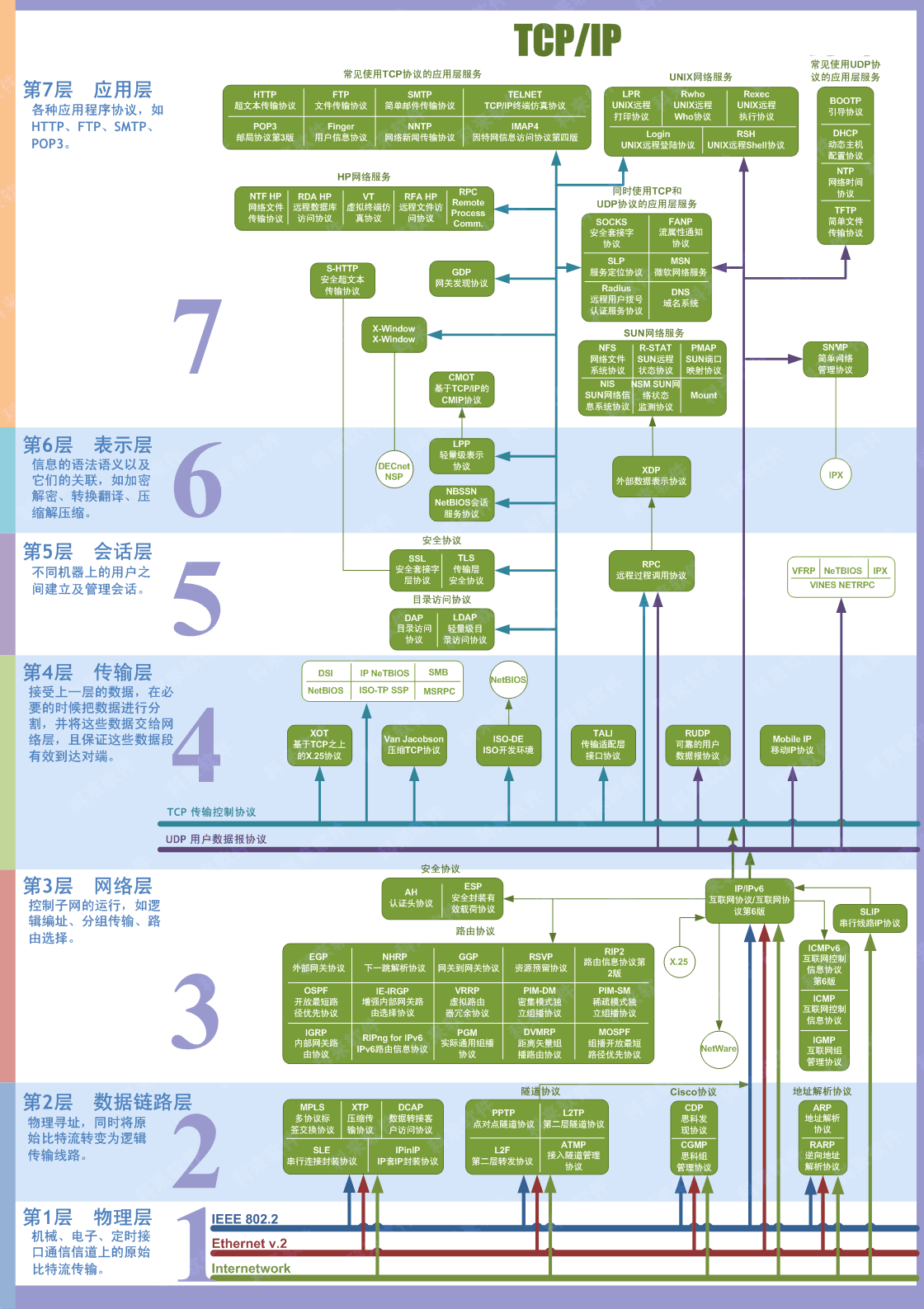
应用层旨在更****方便应用从网络中接收的数据****，重点关注TCP/IP协议中的HTTP协议

四层传输层数据被称作****段****（Segments）；

三层网络层数据被称做****包****（Packages）；

二层数据链路层时数据被称为****帧****（Frames）；

一层物理层时数据被称为****比特流****（Bits）。



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "tcp%E5%92%8Cip%E6%A8%A1%E5%9E%8B)TCP和IP模型

OSI模型注重通信协议必要的功能；TCP/IP更强调在计算机上实现协议应该开发哪种程序

****TCP/IP划分了四层网络模型****

* 第一层：应用层，主要有负责web浏览器的HTTP协议， 文件传输的FTP协议，负责电子邮件的SMTP协议，负责域名系统的DNS等
* 第二层：传输层，主要是有****可靠传输****的TCP协议，特别****高效****的UDP协议。主要负责传输应用层的数据包。
* 第三层：网络层，主要是IP协议。主要负责寻址（找到目标设备的位置）
* 第四层：数据链路层，主要是负责转换数字信号和物理二进制信号。



****四层网络协议的作用****

* 发送端是由上至下，把上层来的数据在头部加上各层协议的数据（部首）再下发给下层。
* 接受端则由下而上，把从下层接受到的数据进行解密和去掉头部的部首后再发送给上层。
* 层层加密和解密后，应用层最终拿到了需要的数据。

****举个例子：****

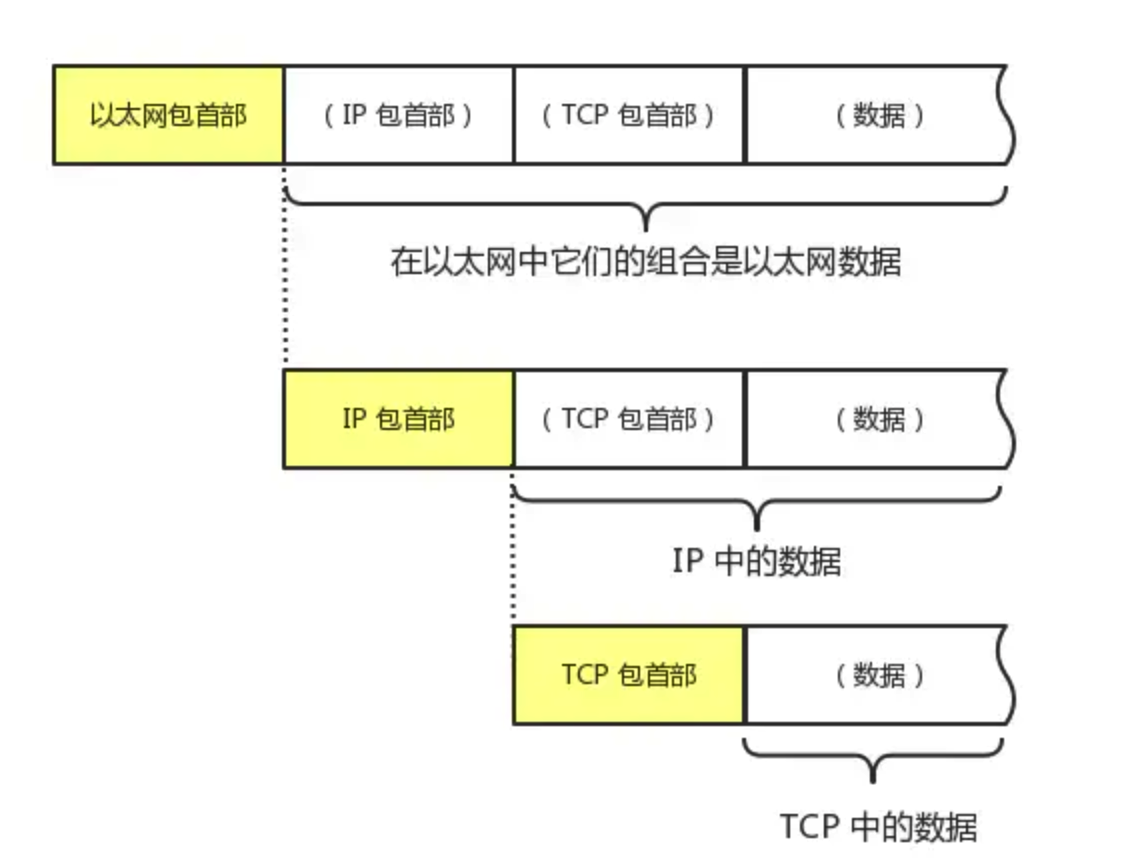
我们需要发送一个****index.html****。

两台电脑在应用层都使用HTTP协议（即都使用浏览器）。

在传输层，TCP协议会将HTTP协议发送的数据看作一个数据包，并在这个数据包前面加上TCP包的一部分信息（部首）

在网络层，IP协议会将TCP协议要发送的数据看作一个数据包，同样的在这个数据包前端加上IP协议的部首

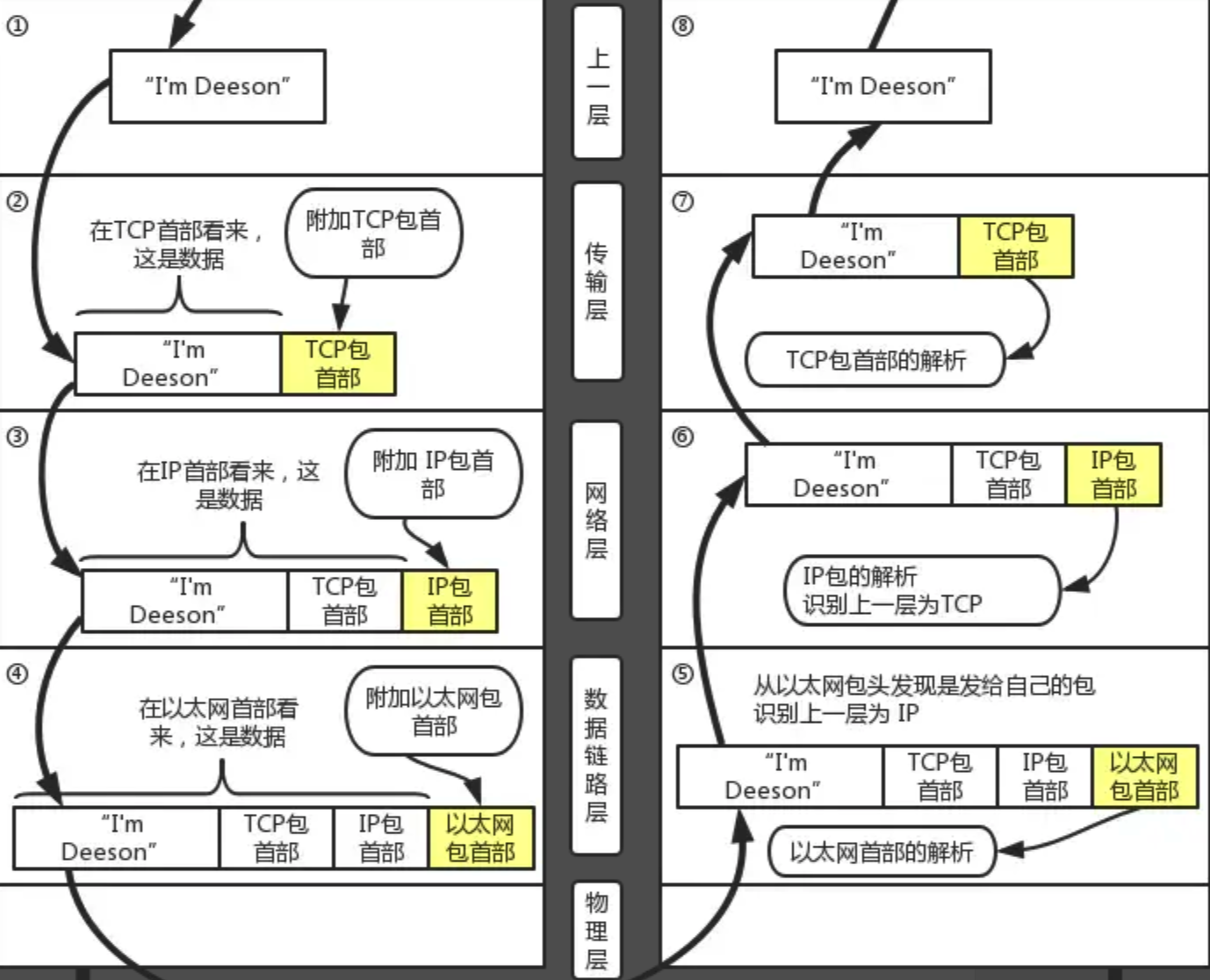
在数据链路层，对应的协议也会在IP数据包前端加上以太网的部首。



源设备和目标设备通过网线连接，就可以通过物理层的二进制传输数据。

数据链路层，会使用对应的协议找到物理层的二进制数据，解码得到以太网的部首信息和对应的IP数据包，再将IP数据包传给上层的网络层。

数据链路层>网络层>传输层>应用层，一层层的解码，最后就可以在浏览器中得到目标设备传送过来的****index.html****。



****TCP/IP协议族****

从字面意义上来讲，TCP/IP是指****传输层****的TCP协议和****网络层****的IP协议。

实际上，TCP/IP只是利用 IP 进行通信时所必须用到的协议群的统称。

具体来说，在网络层是IP/ICMP协议、在传输层是TCP/UDP协议、在应用层是SMTP、FTP、以及 HTTP 等。他们都属于 TCP/IP 协议。



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E7%BD%91%E7%BB%9C%E8%AE%BE%E5%A4%87)网络设备

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E4%BA%A4%E6%8D%A2%E6%9C%BA)交换机

交换机可以接入多台电脑

每个电脑网卡的 ****MAC 地址****都是不一样的，电脑发送数据时，数据头部携带网卡的 MAC 地址，用 MAC 地址标识来不同的电脑

交换机就可以识别数据头部的 MAC 地址来区分不同的电脑

交换机除了能识别不同的电脑，还需要找到电脑连接的****交换机端口****，才能顺利的把数据从相应端口发送出去

交换机通过****自学机制****，把学习到的设备 MAC 地址和交换机端口号添加到 ****MAC 地址表****，并根据 MAC 地址表进行数据****转发****

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E8%B7%AF%E7%94%B1%E5%99%A8)路由器

交换机需要记录的 MAC 地址表也越来越多，需要的交换机也越来越多

但是交换机的****容量和性能有限****，MAC 地址表无法记录全世界电脑的 MAC 地址和对应的端口号，MAC 地址表太大也无法快速查找到对应的 MAC 地址表项

于是就有了三层网络设备****路由器****，路由器可以把全世界的网络连接起来

局域网内的网络连接可以使用****交换机****，例如一个公司内的网络或者一个校园内的网络通过交换机连接

不同区域的局域网互联使用****路由器****

那么如何区分不同的网络区域呢？又是如何跨网络区域进行数据转发的呢？

路由器有多个端口，分别连接不同的网络区域，不同网络区域的 IP 地址****网络号不同****

它通过识别目的 IP 地址的****网络号****，再根据****路由表****进行数据转发



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "http)HTTP

****请求方法****

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

| **序 号** | **方法** | **描述** |
| --- | --- | --- |
| 1 | GET | 请求指定的页面信息，并返回实体主体。 |
| 2 | HEAD | 类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| 3 | POST | 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。 |
| 4 | PUT | 从客户端向服务器传送的数据取代指定的文档的内容。 |
| 5 | DELETE | 请求服务器删除指定的页面。 |
| 6 | CONNECT | HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。 |
| 7 | OPTIONS | 允许客户端查看服务器的性能。 |
| 8 | TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |
| 9 | PATCH | 是对 PUT 方法的补充，用来对已知资源进行局部更新 。 |

****GET请求和POST请求的区别****

1. GET 请求的请求参数是添加到 head 中，可以在 url 中可以看到；POST 请求的请求参数是添加到body中，在url 中不可见。
2. 请求的url有长度限制，这个限制由浏览器和 web 服务器决定和设置的，例如IE浏览器对 URL的最大限制为2083个字符，如果超过这个数字，提交按钮没有任何反应，因为GET请求的参数是添加到URL中，所以GET请求的URL的长度限制需要将请求参数长度也考虑进去。而POST请求不用考虑请求参数的长度。
3. GET请求产生一个数据包; POST请求产生2个数据包，在火狐浏览器中，产生一个数据包，这个区别点在于浏览器的请求机制，先发送请求头，再发送请求体，因为GET没有请求体，所以就发送一个数据包，而POST包含请求体，所以发送两次数据包，但是由于火狐机制不同，所以发送一个数据包。
4. GET 请求会被浏览器主动缓存下来，留下历史记录，而 POST 默认不会。
5. GET是幂等的，而POST不是(幂等表示执行相同的操作，结果也是相同的)
6. GET是获取数据，POST是修改数据

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E7%8A%B6%E6%80%81%E7%A0%81)状态码

****状态码由3位数字组成，第一位定义响应的类别****

1XX：指示信息，表示请求以接收，继续处理

2XX：成功，表示请求已经被成功接收、理解、接受

200 OK 是最常见的成功状态码，表示一切正常。如果是非 HEAD 请求，服务器返回的响应头都会有 body 数据。

204 No Content 也是常见的成功状态码，与 200 OK 基本相同，但响应头没有 body 数据。

206 Partial Content 是应用于 HTTP 分块下载或断电续传，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分，也是服务器处理成功的状态。

3XX：状态码表示客户端请求的资源发送了变动，需要客户端用新的 URL 重新发送请求获取资源，也就是****重定向****。

* 301 Moved Permanently 表示永久重定向，说明请求的资源已经不存在了，需改用新的 URL 再次访问，搜索引擎在抓取新内容的同时也将旧的网址交换为重定向之后的网址。
* 302 Moved Permanently 表示临时重定向，说明请求的资源还在，但暂时需要用另一个 URL 来访问，搜索引擎会抓取新的内容而保存旧的网址。

301 和 302 都会在响应头里使用字段 Location，指明后续要跳转的 URL，浏览器会自动重定向新的 URL。

* 304 Not Modified不具有跳转的含义，表示资源未修改，重定向已存在的缓冲文件，也称缓存重定向，用于缓存控制。

4XX：状态码表示客户端发送的****报文有误****，服务器无法处理，也就是错误码的含义。

400 Bad Request表示客户端请求的报文有错误。

401 Unauthorized：缺失或错误的认证，这个状态代码必须和WWW-Authenticate报头域一起使用。

403 Forbidden表示服务器禁止访问资源，并不是客户端的请求出错。

404 Not Found表示请求的资源在服务器上不存在或未找到，所以无法提供给客户端。

5XX：状态码表示客户端请求报文正确，但是****服务器处理时内部发生了错误****，属于服务器端的错误码。

501 Not Implemented 表示客户端请求的功能还不支持。

502 Bad Gateway 通常是服务器作为网关或代理时返回的错误码，表示服务器自身工作正常，访问后端服务器发生了错误。

503 Service Unavailable 表示服务器当前很忙，暂时无法响应服务器。

504 Gateway Timeout：网关超时，由作为代理或网关的服务器使用，表示不能及时地从远程服务器获得应答。

****301和302的区别****

301重定向，指页面永久性转移，表示为资源或页面永久性地转移到了另一个位置。

301是HTTP协议中的一种状态码，当用户或搜索引擎向服务器发出浏览请求时，服务器返回的HTTP数据流中头信息中包含状态码 301 ，表示该资源已经永久改变了位置。

302重定向是页面暂时性转移，搜索引擎会抓取新的内容而保存旧的网址并认为新的网址只是暂时的。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "http1-1)HTTP1.1

****长连接****

HTTP 1.1支持长连接

HTTP 1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。

HTTP 1.1则支持持久连接Persistent Connection，并且默认使用，在同一个TCP的连接中可以传送多个HTTP请求和响应，多个请求和响应可以重叠，多个请求和响应可以同时进行，更加多的请求头和响应头

HTTP 1.1的持续连接，也需要增加新的请求头来帮助实现，例如，Connection请求头的值为Keep-Alive时，客户端通知服务器返回本次请求结果后保持连接；Connection请求头的值为Close时，客户端通知服务器返回本次请求结果后关闭连接。

****管道网络传输****

HTTP/1.1 采用了长连接的方式，这使得管道网络传输成为了可能。

即可在同一个 TCP 连接里面，客户端可以发起多个请求，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。****

举例来说，客户端需要请求两个资源。以前的做法是，在同一个TCP连接里面，先发送 A 请求，然后等待服务器做出回应，收到后再发出 B 请求，管道机制则是允许浏览器同时发出 A 请求和 B 请求。

但是服务器还是按照****顺序****，先回应 A 请求，完成后再回应 B 请求，要是 前面的回应特别慢，后面就会有许多请求排队等着。

****Host字段****

在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名，但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机，并且它们共享一个IP地址。

HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。

此外，服务器应该接受以绝对路径标记的资源请求。

****100Status****

HTTP/1.1加入了一个新的状态码100。

客户端事先发送一个只带头域的请求，如果服务器因为权限拒绝了请求，就回送响应码401（Unauthorized）；

如果服务器接收此请求就回送响应码100，客户端就可以继续发送带实体的完整请求了。

100状态代码的使用，允许客户端在发request消息body之前先用request header试探一下server，看server要不要接收request body，再决定要不要发request body。

****Chunked Transfer Coding****

HTTP/1.1将发送方将消息分割成若干个任意大小的数据块，每个数据块在发送时都会附上块的长度，最后用一个零长度的块作为消息结束的标志。

这种方法允许发送方只缓冲消息的一个片段，避免缓冲整个消息带来的过载。

****Cache****

HTTP/1.1在1.0的基础上加入了一些Cache的新特性，当缓存对象的Age超过Expire时变为Stable对象，Cache不需要直接抛弃Stable对象，而是与源服务器进行重新激活。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "http2-0)HTTP2.0

****HTTP2.0和HTTP1.X相比的新特性****

新的二进制格式，HTTP1.x的解析是基于文本

多路复用，即连接共享，即每一个request都是是用作连接共享机制的，一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面

header压缩，HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小

服务端推送

HTTP/2 还在一定程度上改善了传统的请求 - 应答工作模式，服务不再是被动地响应，也可以****主动****向客户端发送消息。

举例来说，在浏览器刚请求 HTML 的时候，就提前把可能会用到的 JS、CSS 文件等静态资源主动发给客户端，****减少延时的等待****，也就是服务器推送。

****数据流****

HTTP/2 的数据包不是按顺序发送的，同一个连接里面连续的数据包，可能属于不同的回应。

因此，必须要对数据包做标记，指出它属于哪个回应。

每个请求或回应的所有数据包，称为一个数据流（Stream）。

每个数据流都标记着一个独一无二的编号，其中规定客户端发出的数据流编号为奇数， 服务器发出的数据流编号为偶数

客户端还可以****指定数据流的优先级****。优先级高的请求，服务器就先响应该请求。

****HTTP2.0的多路复用和HTTP1.X中的长连接复用有什么区别****

* HTTP/1.1的Pipeling为若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法；
* HTTP2.0多个请求可同时在一个连接上并行执行，某个请求任务耗时严重，不会影响到其它连接的正常执行

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "http3-0)HTTP3.0

****使用UDP协议****

HTTP/2 主要的问题在于：多个 HTTP 请求在复用一个 TCP 连接，下层的 TCP 协议是不知道有多少个 HTTP 请求的。

所以一旦发生了丢包现象，就会触发 TCP 的重传机制，这样在一个 TCP 连接中的****所有的 HTTP 请求都必须等待这个丢了的包被重传回来****。

* HTTP/1.1 中的管道传输中如果有一个请求阻塞了，那么队列后请求也统统被阻塞住了
* HTTP/2 多请求复用一个TCP连接，一旦发生丢包，就会阻塞住所有的 HTTP 请求。

这都是基于 TCP 传输层的问题，所以 ****HTTP/3 把 HTTP 下层的 TCP 协议改成了 UDP！****

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "https)HTTPS

****HTTP与HTTPS的区别****

HTTP 是明文传输协议，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 HTTP 协议安全

HTTPS比HTTP更加安全，对搜索引擎更友好，利于SEO，谷歌、百度优先索引HTTPS网页

HTTPS需要用到SSL证书，而HTTP不用

HTTPS标准端口443，HTTP标准端口80

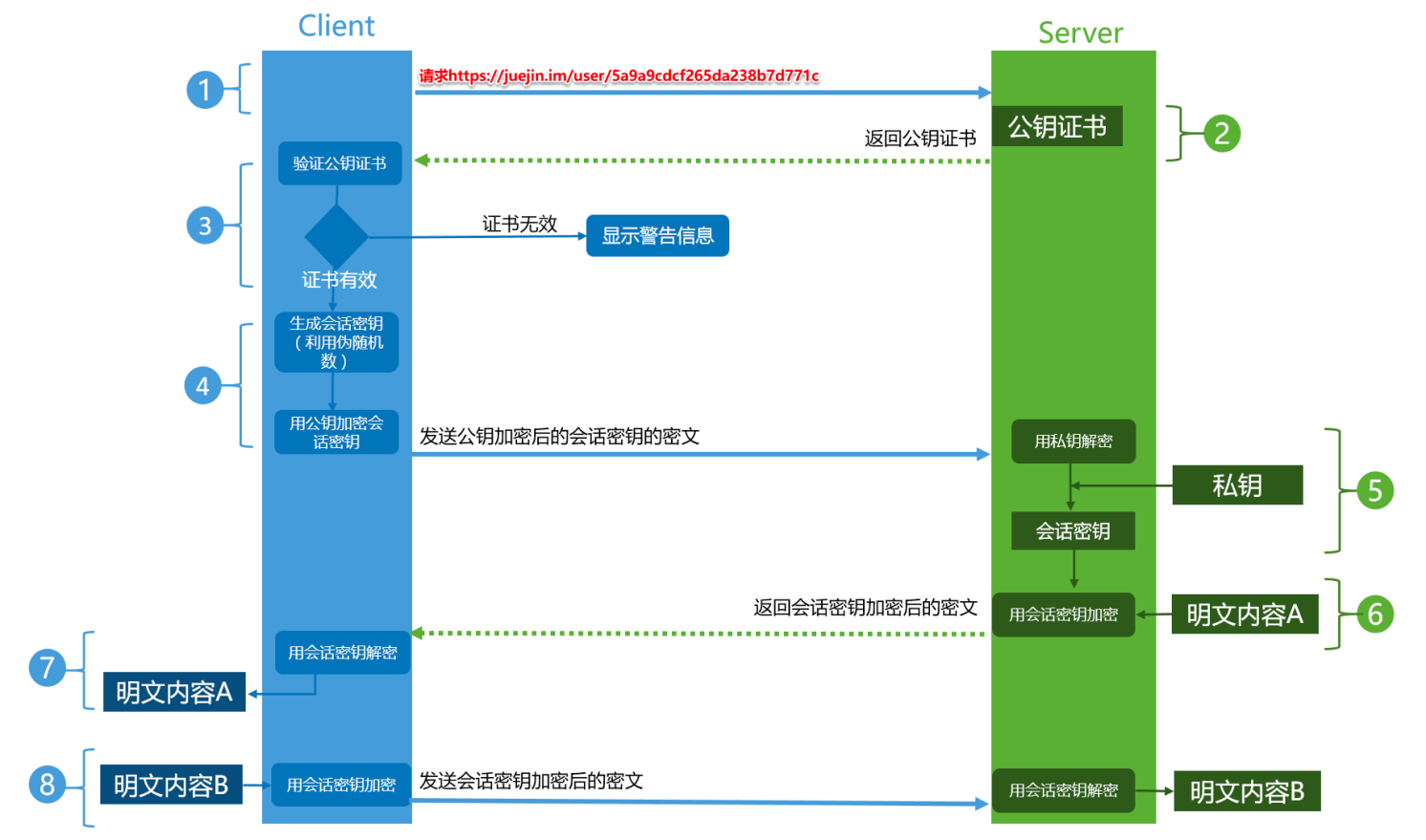
HTTPS基于传输层，HTTP基于应用层

HTTPS在浏览器显示绿色安全锁，HTTP没有显示

****工作原理****

HTTPS 协议会对传输的数据进行加密，而加密过程是使用了非对称加密实现

HTTPS的整体过程分为证书验证和数据传输阶段，具体的交互过程如下：



* Client发起一个HTTPS的请求
* Server把事先配置好的公钥证书返回给客户端。
* Client验证公钥证书：比如是否在有效期内，证书的用途是不是匹配Client请求的站点，是不是在CRL吊销列表里面，它的上一级证书是否有效，这是一个递归的过程，直到验证到根证书（操作系统内置的Root证书或者Client内置的Root证书），如果验证通过则继续，不通过则显示警告信息。
* Client使用伪随机数生成器生成加密所使用的对称密钥，然后用证书的公钥加密这个对称密钥，发给Server。
* Server使用自己的私钥解密这个消息，得到对称密钥。至此，Client和Server双方都持有了相同的对称密钥。
* Server使用对称密钥加密明文内容A，发送给Client。
* Client使用对称密钥解密响应的密文，得到明文内容A。
* Client再次发起HTTPS的请求，使用对称密钥加密请求的明文内容B，然后Server使用对称密钥解密密文，得到明文内容B。

#### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E6%95%B0%E5%AD%97%E8%AF%81%E4%B9%A6)数字证书

客户端先向服务器端索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密。

这就存在些问题，如何保证公钥不被篡改和信任度？

所以这里就需要借助第三方权威机构 CA （数字证书认证机构），将****服务器公钥放在数字证书****（由数字证书认证机构颁发）中，只要证书是可信的，公钥就是可信的。

通过数字证书的方式保证服务器公钥的身份，解决冒充的风险。



### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E8%AF%B7%E6%B1%82%E6%8A%A5%E6%96%87)请求报文

****请求头****

HTTP 请求报文由3部分组成(请求行+请求头+请求体)



****常见的HTTP报文头属性****

* Accpet
  + 告诉服务端，客户端接收什么类型的响应
* Referer
  + 表示这是请求是从哪个URL进来的,比如想在网上购物，但是不知道选择哪家电商平台，你就去问度娘，说哪家电商的东西便宜啊，然后一堆东西弹出在你面前，第一给就是某宝，当你从这里进入某宝的时候，这个请求报文的Referer就是：[www.baidu.com](https://tobebetterjavaer.com/cs/www.baidu.com)
* Cache-Control
  + 对缓存进行控制，如一个请求希望响应的内容在客户端缓存一年，或不被缓可以通过这个报文头设置
* Accept-Encoding
  + 这个属性是用来告诉服务器能接受什么编码格式，包括字符编码,压缩形式(一般都是压缩形式)
    - 例如:Accept-Encoding:gzip, deflate(这两种都是压缩格式)
* Host
  + 指定要请求的资源所在的主机和端口
* User-Agent：告诉服务器，客户端使用的操作系统、浏览器版本和名称
* Connection

决定当前事务（三次握手和四次挥手）完成后，是否关闭网络连接。

* 持久连接，事务完成后不关闭网络连接 ： Connection: keep-alive
* 非持久连接，事务完成后关闭网络连接： Connection: close

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E5%93%8D%E5%BA%94%E6%8A%A5%E6%96%87)响应报文

响应报文与请求报文一样，由三个部分组成(响应行,响应头,响应体)



****HTTP响应报文属性****

* Cache-Control
  + 响应输出到客户端后，服务端通过该属性告诉客户端该怎么控制响应内容的缓存
* ETag
  + 表示你请求资源的版本，如果该资源发生啦变化，那么这个属性也会跟着变
* Location
  + 在重定向中或者创建新资源时使用
* Set-Cookie
  + 服务端可以设置客户端的cookie



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "tcp)TCP

TCP是一个传输层协议，提供可靠传输，支持全双工，是一个连接导向的协议。

****双工/单工****

在任何一个时刻，如果数据只能单向发送，就是单工。

如果在某个时刻数据可以向一个方向传输，也可以向另一个方向反方向传输，而且交替进行，叫作半双工；半双工需要至少 1 条线路。

如果任何时刻数据都可以双向收发，这就是全双工，全双工需要大于 1 条线路。

TCP 是一个双工协议，数据任何时候都可以双向传输。

这就意味着客户端和服务端可以平等地发送、接收信息。

****TCP协议的主要特点****

TCP是面向连接的运输层协议；所谓面向连接就是双方传输数据之前，必须先建立一条通道，例如三次握手就是建议通道的一个过程，而四次挥手则是结束销毁通道的一个其中过程。

每一条TCP连接只能有两个端点（即两个套接字），只能是点对点的；

TCP提供可靠的传输服务。传送的数据无差错、不丢失、不重复、按序到达；

TCP提供全双工通信。允许通信双方的应用进程在任何时候都可以发送数据，因为两端都设有发送缓存和接受缓存；

面向字节流。虽然应用程序与TCP交互是一次一个大小不等的数据块，但TCP把这些数据看成一连串无结构的字节流，它不保证接收方收到的数据块和发送方发送的数据块具有对应大小关系，例如，发送方应用程序交给发送方的TCP10个数据块，接收方的TCP可能只用收到的4个数据块字节流交付给上层的应用程序

****TCP的可靠性原理****

可靠传输有如下两个特点:

1. 传输信道无差错,保证传输数据正确;
2. 不管发送方以多快的速度发送数据,接收方总是来得及处理收到的数据;

首先，采用三次握手来建立TCP连接，四次握手来释放TCP连接，从而保证建立的传输信道是可靠的。

其次，TCP采用了连续ARQ协议（回退N(Go-back-N)；超时自动重传）来保证数据传输的正确性，使用滑动窗口协议来保证接方能够及时处理所接收到的数据，进行流量控制。

最后，TCP使用慢开始、拥塞避免、快重传和快恢复来进行拥塞控制，避免网络拥塞。

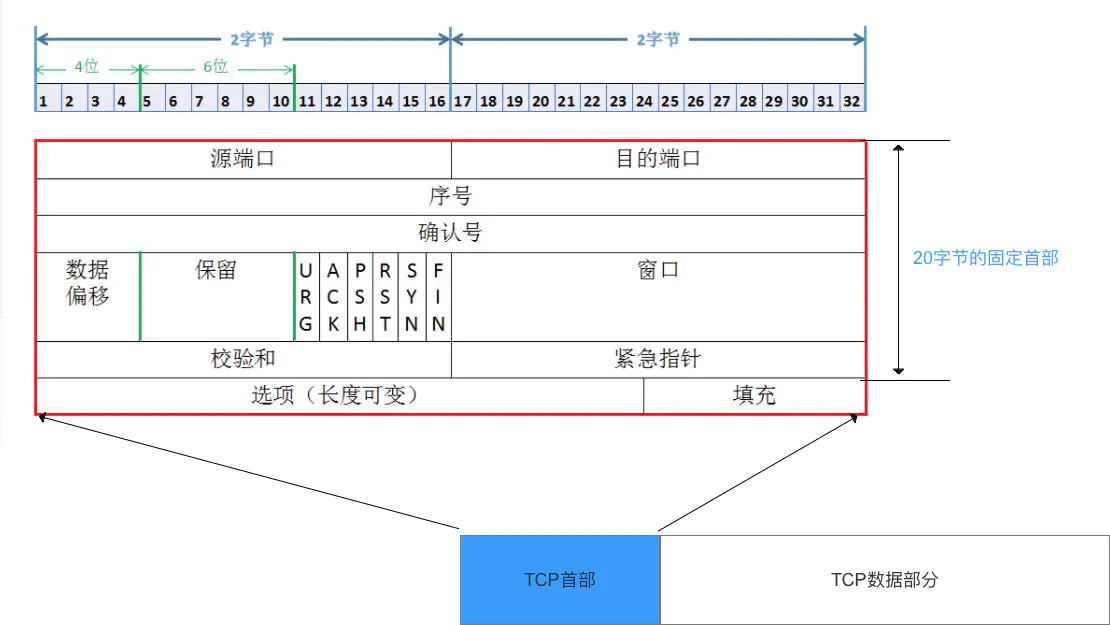
### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E6%8A%A5%E6%96%87%E6%AE%B5)报文段

TCP虽面向字节流，但传送的数据单元为报文段

报文段 = 首部 + 数据2部分

TCP的全部功能体现在它首部中各字段的作用

1. 首部前20个字符固定、后面有4n个字节是根据需而增加的选项
2. 故 TCP首部最小长度 = 20字节



****端口****：

源端口号和目地端口各占16位两个字节，也就是端口的范围是2^16=65535

另外1024以下是系统保留的，从1024-65535是用户使用的端口范围

****seq序号****：占4字节，TCP连接中传送的字节流中的每个字节都按顺序编号。

例如：一段报文的序号字段值是107，携带的数据是100个字段，下一个报文段序号从107+100=207开始。

****ack确认号****：4个字节，是期望收到对方下一个报文段的第一个数据字节的序号。

例如：B收到A发送的报文，其序号字段是301，数据长度是200字节，表明B正确收到A发送的到序号500为止的数据（301+200-1=500），B期望收到A下一个数据序号是501，B发送给A的确认报文段中把ack确认号置为501。

****数据偏移****：头部有可选字段，长度不固定，指出TCP报文段的数据起始处距离报文段的起始处有多远。

****保留****：保留今后使用的，被标为1。

****控制位****：由8个标志位组成。每个标志位表示一个控制功能。

其中主要的6个：

****URG紧急指针标志****，为1表示紧急指针有效，为0忽略紧急指针。

****ACK确认序号标志****，为1表示确认号有效，为0表示报文不含确认信息，忽略确认号字段，上面的确认号是否有效就是通过该标识控制的。

****PSH标志****，为1表示带有push标志的数据，指示接收方在接收到该报文段以后，应尽快将该报文段交给应用程序，而不是在缓冲区排队。

****RST重置连接标志****，重置因为主机崩溃或其他原因而出现错误的连接，或用于拒绝非法的报文段或非法的连接。

****SYN同步序号****，同步序号，用于建立连接过程，在连接请求中，SYN=1和ACK=0表示该数据段没有使用捎带的确认域，而连接应答捎带一个确认，即SYN=1和ACK=1。。

****FIN终止标志****，用于释放连接，为1时表示发送方没有发送了。

****窗口****：滑动窗口大小，用来告知发送端接收端缓存大小，以此控制发送端发送数据的速率，从而达到流量控制。

****校验和****：奇偶校验，此校验和是对整个的TCP报文段（包括TCP头部和TCP数据），以16位进行计算所得，由发送端计算和存储，接收端进行验证。

****紧急指针****：只有控制位中的URG为1时才有效，指出本报文段中的紧急数据的字节数。

****选项****：其长度可变，定义其他的可选参数。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E7%B2%98%E5%8C%85%E4%B8%8E%E6%8B%86%E5%8C%85)粘包与拆包

TCP是面向字节流的协议，把上层应用层的数据看成字节流，所以它发送的不是固定大小的数据包，TCP协议也没有字段说明发送数据包的大小。

而且TCP不保证接受方应用程序收到的数据块和发送应用程序发送的数据块具有对应的大小关系

比如发送方应用程序交给发送方TCP 10个数据块，接受方TCP可能只用了4个数据块就完整的把接受到的字节流交给了上层应用程序。

TCP底层并不了解上层业务数据的具体含义，它会根据TCP缓冲区的实际情况进行包的划分，所以在业务上认为，一个完整的包可能会被TCP拆分成多个包进行发送，也有可能把多个小的包封装成一个大的数据包发送，这就是所谓的TCP粘包和拆包问题

****TCP粘包/拆包解决策略****

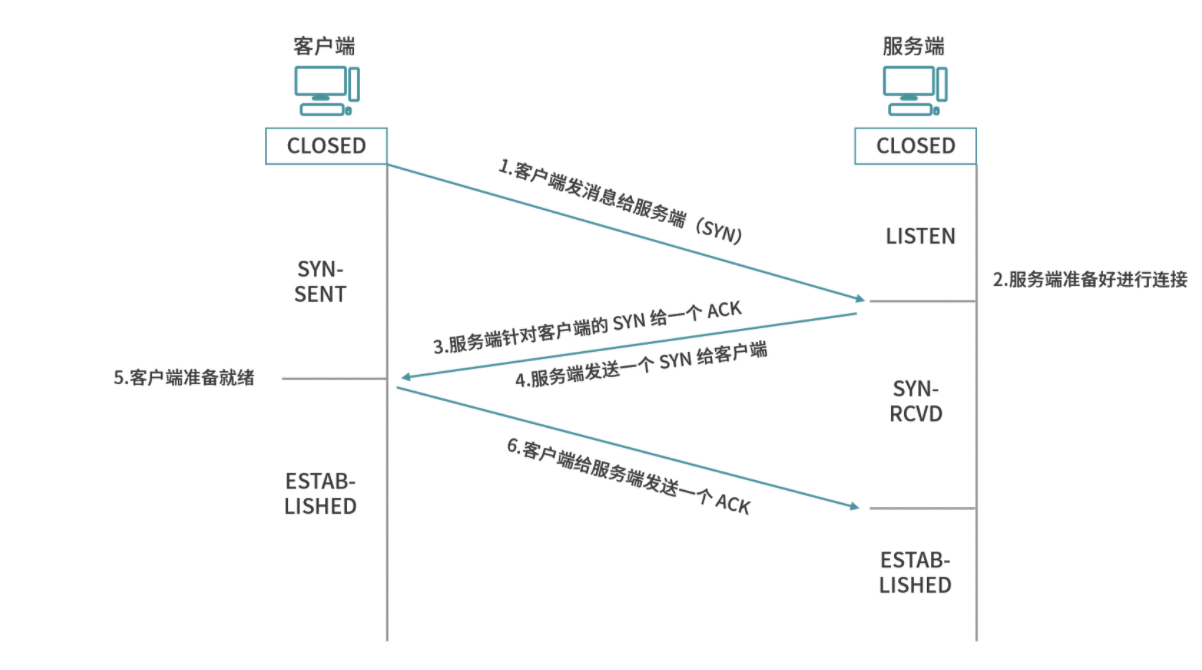
由于TCP无法理解上一层的业务数据特点，所以TCP是无法保证发送的数据包不发生粘包和拆包，这个问题只能通过上层的协议栈设计来解决，解决思路有一下几种：

消息定长：每个发送的数据包大小固定，比如100字节，不足100字节的用空格补充，接受方取数据的时候根据这个长度来读取数据

消息末尾增加换行符来表示一条完整的消息：接收方读取的时候根据换行符来判断是否是一条完整的消息，如果消息的内容也包含换行符，那么这种方式就不合适了。

将消息分为消息头和消息尾两部分，消息头指定数据长度，根据消息长度来读取完整的消息，例如UDP协议是这么设计的，用两个字节来表示消息长度，所以UDP不存在粘包和拆包问题。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B)三次握手



****第一次握手****：

客户端将TCP报文标志位SYN置为1，随机产生一个序号值seq=J，保存在TCP首部的序列号字段里，指明客户端打算连接的服务器的端口，并将该数据包发送给服务器端，发送完毕后，客户端进入SYN\_SENT状态，等待服务器端确认。

****第二次握手****：

服务器端收到数据包后由标志位SYN=1知道客户端请求建立连接，服务器端将TCP报文标志位SYN和ACK都置为1，ack=J+1，随机产生一个序号值seq=K，并将该数据包发送给客户端以确认连接请求，服务器端进入SYN\_RCVD状态。

****第三次握手****：

客户端收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给服务器端，服务器端检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，客户端和服务器端进入ESTABLISHED状态，完成三次握手，随后客户端与服务器端之间可以开始传输数据了。

****上面写的ack和ACK，不是同一个概念：****

小写的ack代表的是头部的确认号Acknowledge number， 缩写ack，是对上一个包的序号进行确认的号，ack=seq+1。

大写的ACK，则是我们上面说的TCP首部的标志位，用于标志的TCP包是否对上一个包进行了确认操作，如果确认了，则把ACK标志位设置成1。

****TCP为什么三次握手而不是两次握手****

为了实现可靠数据传输， TCP 协议的通信双方， 都必须维护一个序列号， 以标识发送出去的数据包中， 哪些是已经被对方收到的。 三次握手的过程即是通信双方相互告知序列号起始值， 并确认对方已经收到了序列号起始值的必经步骤

如果只是两次握手， 至多只有连接发起方的起始序列号能被确认， 另一方选择的序列号则得不到确认

****《计算机网络》中是这样说的：****

为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

在书中同时举了一个例子，如下：

假如client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server，本来这是一个早已失效的报文段，但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。

于是就向client发出确认报文段，同意建立连接，假设不采用****三次握手****，那么只要server发出确认，新的连接就建立了，由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。

但server却以为新的连接已经建立，并一直等待client发来数据，这样，server的很多资源就白白浪费掉了。

采用****三次握手****的办法可以防止上述现象发生，例如刚才那种情况，client不会向server的确认发出确认，server由于收不到确认，就知道client并没有要求建立连接。

****什么是半连接队列****

服务器第一次收到客户端的 SYN 之后，就会处于 SYN\_RCVD状态，此时双方还没有完全建立其连接，服务器会把此种状态下请求连接放在一个队列里，我们把这种队列称之为****半连接队列****。

当然还有一个****全连接队列****，就是已经完成三次握手，建立起连接的就会放在全连接队列中。如果队列满了就有可能会出现丢包现象。

补充一点关于****SYN-ACK 重传次数****的问题：

服务器发送完SYN－ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传，如果重传次数超过系统规定的最大重传次数，系统将该连接信息从半连接队列中删除。

****三次握手过程中可以携带数据吗****

其实第三次握手的时候，是可以携带数据的，也就是说，第一次、第二次握手不可以携带数据，而第三次握手是可以携带数据的。

假如第一次握手可以携带数据的话，如果有人要恶意攻击服务器，那他每次都在第一次握手中的 SYN 报文中放入大量的数据，因为攻击者根本就不理服务器的接收、发送能力是否正常，然后疯狂着重复发 SYN 报文的话，这会让服务器花费很多时间、内存空间来接收这些报文。也就是说，第一次握手可以放数据的话，其中一个简单的原因就是会让服务器更加容易受到攻击了。

而对于第三次的话，此时客户端已经处于 established 状态，也就是说，对于客户端来说，他已经建立起连接了，并且也已经知道服务器的接收、发送能力是正常的了，所以能携带数据没啥毛病。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E5%9B%9B%E6%AC%A1%E6%8C%A5%E6%89%8B)四次挥手



挥手请求可以是Client端，也可以是Server端发起的，我们假设是Client端发起：

第一次挥手： Client端发起挥手请求，向Server端发送标志位是FIN报文段，设置序列号seq，此时，Client端进入FIN\_WAIT\_1状态，这表示Client端没有数据要发送给Server端了。

第二次挥手：Server端收到了Client端发送的FIN报文段，向Client端返回一个标志位是ACK的报文段，ack设为seq加1，Client端进入FIN\_WAIT\_2状态，Server端告诉Client端，我确认并同意你的关闭请求。

第三次挥手： Server端向Client端发送标志位是FIN的报文段，请求关闭连接，同时Client端进入LAST\_ACK状态。

第四次挥手 ： Client端收到Server端发送的FIN报文段，向Server端发送标志位是ACK的报文段，然后Client端进入TIME\_WAIT状态，Server端收到Client端的ACK报文段以后，就关闭连接，此时，Client端等待2MSL的时间后依然没有收到回复，则证明Server端已正常关闭，那好，Client端也可以关闭连接了。

****为什么连接的时候是三次握手，关闭的时候却是四次握手？****

建立连接时因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。所以建立连接只需要三次握手。

由于TCP协议是一种面向连接的、可靠的、基于字节流的运输层通信协议，TCP是全双工模式。

这就意味着，关闭连接时，当Client端发出FIN报文段时，只是表示Client端告诉Server端数据已经发送完毕了。当Server端收到FIN报文并返回ACK报文段，表示它已经知道Client端没有数据发送了，但是Server端还是可以发送数据到Client端的，所以Server很可能并不会立即关闭SOCKET，直到Server端把数据也发送完毕。

当Server端也发送了FIN报文段时，这个时候就表示Server端也没有数据要发送了，就会告诉Client端，我也没有数据要发送了，之后彼此就会愉快的中断这次TCP连接。

****为什么TIME\_WAIT要等待2MSL？****

MSL：报文段最大生存时间，它是任何报文段被丢弃前在网络内的最长时间。

有以下两个原因：

* 第一点：保证TCP协议的全双工连接能够可靠关闭： 由于IP协议的不可靠性或者是其它网络原因，导致了Server端没有收到Client端的ACK报文，那么Server端就会在超时之后重新发送FIN，如果此时Client端的连接已经关闭处于CLOESD状态，那么重发的FIN就找不到对应的连接了，从而导致连接错乱，所以，Client端发送完最后的ACK不能直接进入CLOSED状态，而要保持TIME\_WAIT，当再次收到FIN的时候，能够保证对方收到ACK，最后正确关闭连接。
* 第二点：保证这次连接的重复数据段从网络中消失 如果Client端发送最后的ACK直接进入CLOSED状态，然后又再向Server端发起一个新连接，这时不能保证新连接的与刚关闭的连接的端口号是不同的，也就是新连接和老连接的端口号可能一样了，那么就可能出现问题：如果前一次的连接某些数据滞留在网络中，这些延迟数据在建立新连接后到达Client端，由于新老连接的端口号和IP都一样，TCP协议就认为延迟数据是属于新连接的，新连接就会接收到脏数据，这样就会导致数据包混乱，所以TCP连接需要在TIME\_WAIT状态等待2倍MSL，才能保证本次连接的所有数据在网络中消失。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E6%B5%81%E9%87%8F%E6%8E%A7%E5%88%B6)流量控制

****RTT和RTO****

RTT：发送一个数据包到收到对应的ACK，所花费的时间

RTO：重传时间间隔（TCP在发送一个数据包后会启动一个重传定时器，RTO即定时器的重传时间）

开始预先算一个定时器时间，如果回复ACK，重传定时器就自动失效，即不需要重传；如果没有回复ACK，RTO定时器时间就到了，重传。

RTO是本次发送当前数据包所预估的超时时间，RTO不是固定写死的配置，是经过RTT计算出来的。

****滑动窗口****

TCP的滑动窗口主要有两个作用：

保证TCP的可靠性

保证TCP的流控特性

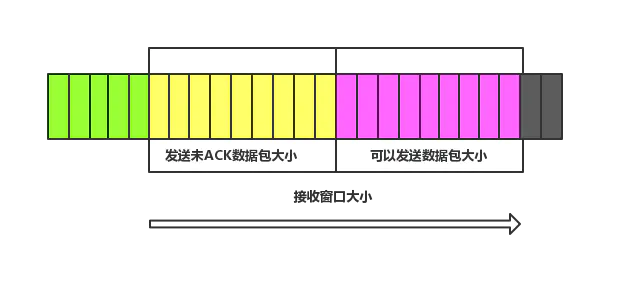
TCP报文头有个字段叫Window，用于接收方通知发送方自己还有多少缓存区可以接收数据，发送方根据接收方的处理能力来发送数据，不会导致接收方处理不过来，这便是流量控制。

发送方都维持了一个连续的允许发送的帧的序号，称为发送窗口；同时，接收方也维持了一个连续的允许接收的帧的序号，称为接收窗口。

发送窗口和接收窗口的序号的上下界不一定要一样，甚至大小也可以不同。

不同的滑动窗口协议窗口大小一般不同。

发送方窗口内的序列号代表了那些已经被发送，但是还没有被确认的帧，或者是那些可以被发送的帧



滑动窗口由四部分组成每个字节的数据都有唯一顺序的编码，随着时间发展，未确认部分与可以发送数据包编码部分向右移动，形式滑动窗口

1. 绿色：发送成功并已经ACK确认的数据
2. 黄色：发送成功等待ACK确认的数据(占用滑动窗口大小)
3. 紫色：滑动窗口剩余大小可以发送的字节数量(滑动窗口可用大小)
4. 灰色：后续数据编码

接收窗口的大小就是滑动窗口的最大值，数据传输过程中滑动窗口的可用大小是动态变化的。

但是还有这么一点，滑动窗口的设计仅仅是考虑到了处理方的处理能力，但是没有考虑到道路的通畅问题

就好像服务端可以处理100M数据，但是传输的数据99M都堵在路上了，这不就是导致道路阻塞了么？这就需要另外一个设计****拥塞避免****

****流量控制的目的****

如果发送者发送数据过快，接收者来不及接收，那么就会有分组丢失。

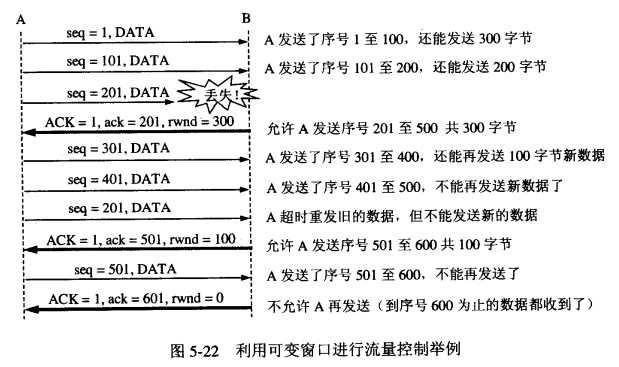
为了避免分组丢失，控制发送者的发送速度，使得接收者来得及接收，这就是流量控制。

流量控制根本目的是防止分组丢失，它是构成TCP可靠性的一方面。

****如何实现流量控制****

由滑动窗口协议（连续ARQ协议）实现。滑动窗口协议既保证了分组无差错、有序接收，也实现了流量控制。

主要的方式就是接收方返回的 ACK 中会包含自己的接收窗口的大小，并且利用大小来控制发送方的数据发送。



****流量控制引发的死锁****

当发送者收到了一个窗口为0的应答，发送者便停止发送，等待接收者的下一个应答。

但是如果这个窗口不为0的应答在传输过程丢失，发送者一直等待下去，而接收者以为发送者已经收到该应答，等待接收新数据，这样双方就相互等待，从而产生死锁。

为了避免流量控制引发的死锁，TCP使用了****持续计时器****。每当发送者收到一个零窗口的应答后就启动该计时器。时间一到便主动发送报文询问接收者的窗口大小。若接收者仍然返回零窗口，则重置该计时器继续等待；若窗口不为0，则表示应答报文丢失了，此时重置发送窗口后开始发送，这样就避免了死锁的产生。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6)拥塞控制

****为什么要进行拥塞控制****

假设网络已经出现拥塞，如果不处理拥塞，那么延时增加，出现更多丢包，触发发送方重传数据，加剧拥塞情况，继续恶性循环直至网络瘫痪。

拥塞控制与流量控制的适应场景和目的均不同。

拥塞发生前，可避免流量过快增长拖垮网络；拥塞发生时，唯一的选择就是降低流量。

主要使用4种算法完成拥塞控制：

1. 慢启动
2. 拥塞避免
3. 快重传算法
4. 快速恢复算法

算法1、2适用于拥塞发生前，算法3适用于拥塞发生时，算法4适用于拥塞解决后（相当于拥塞发生前）。

****rwnd与cwnd****

rwnd（Receiver Window，接收者窗口）与cwnd（Congestion Window，拥塞窗口）：

rwnd是用于流量控制的窗口大小，主要取决于接收方的处理速度，由接收方通知发送方被动调整。

cwnd是用于拥塞处理的窗口大小，取决于网络状况，由发送方探查网络主动调整。

同时考虑流量控制与拥塞处理，则发送方窗口的大小不超过min{rwnd, cwnd}。

****慢启动算法****

慢开始算法的思路就是，不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小。

这里用报文段的个数作为拥塞窗口的大小举例说明慢开始算法，实际的拥塞窗口大小是以字节为单位的。

一个传输轮次所经历的时间其实就是往返时间RTT，而且每经过一个传输轮次，拥塞窗口cwnd就加倍。

为了防止cwnd增长过大引起网络拥塞，还需设置一个慢开始门限ssthresh状态变量。

ssthresh的用法如下：

cwnd<ssthresh时，使用慢开始算法。

当cwnd>ssthresh时，改用拥塞避免算法。

当cwnd=ssthresh时，慢开始与拥塞避免算法任意

注意，这里的慢并不是指cwnd的增长速率慢，而是指在TCP开始发送报文段时先设置cwnd=1，然后逐渐增大，这当然比按照大的cwnd一下子把许多报文段突然注入到网络中要慢得多。

****拥塞避免算法****

让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。

这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多

无论是在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有按时收到确认，虽然没有收到确认可能是其他原因的分组丢失，但是因为无法判定，所以都当做拥塞来处理），就把慢开始门限ssthresh设置为出现拥塞时的发送窗口大小的一半（但不能小于2）。

然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。

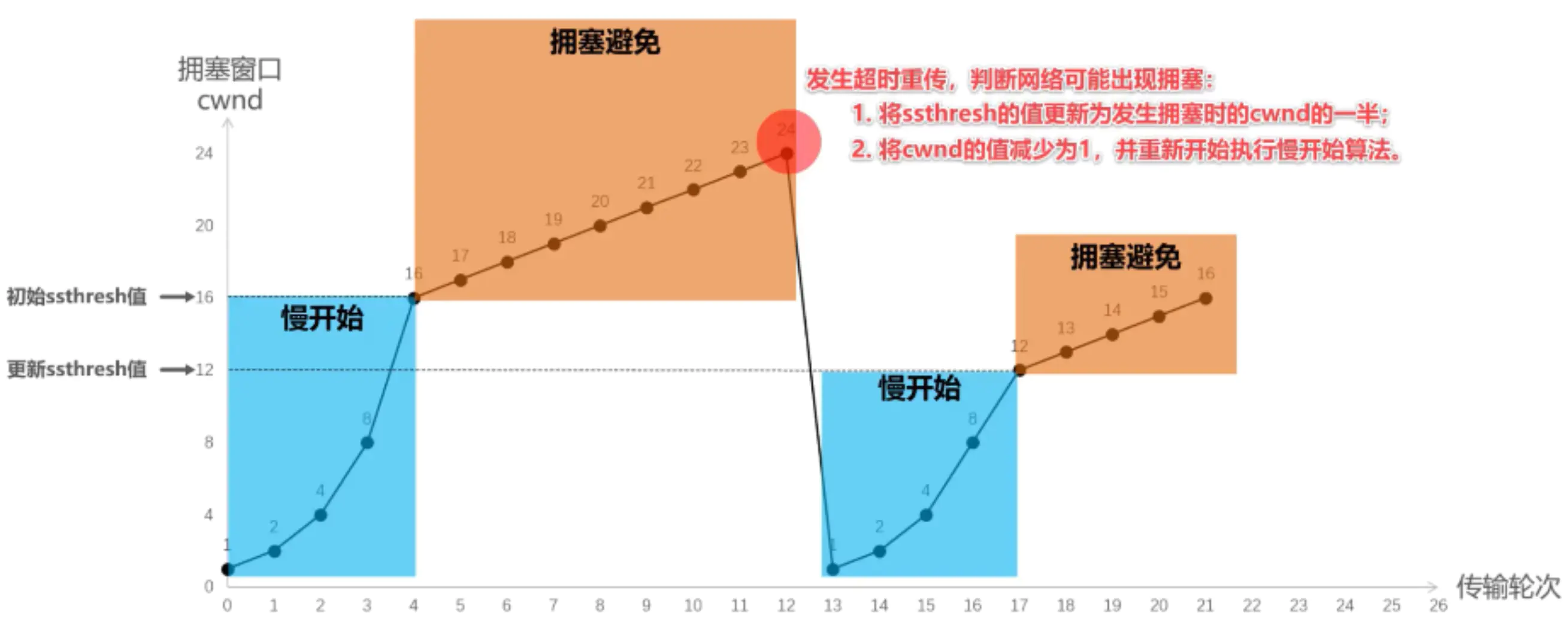
这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

****整个拥塞控制的流程：****

假定cwnd=24时，网络出现超时（拥塞），则更新后的ssthresh=12，cwnd重新设置为1，并执行慢开始算法。

当cwnd=12=ssthresh时，改为执行拥塞避免算法

注意：拥塞避免并非完全能够避免了阻塞，而是使网络比较不容易出现拥塞。



****快重传算法****

快重传要求接收方在收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方，可提高网络吞吐量约20%）而不要等到自己发送数据时捎带确认。

快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期

****快恢复算法****

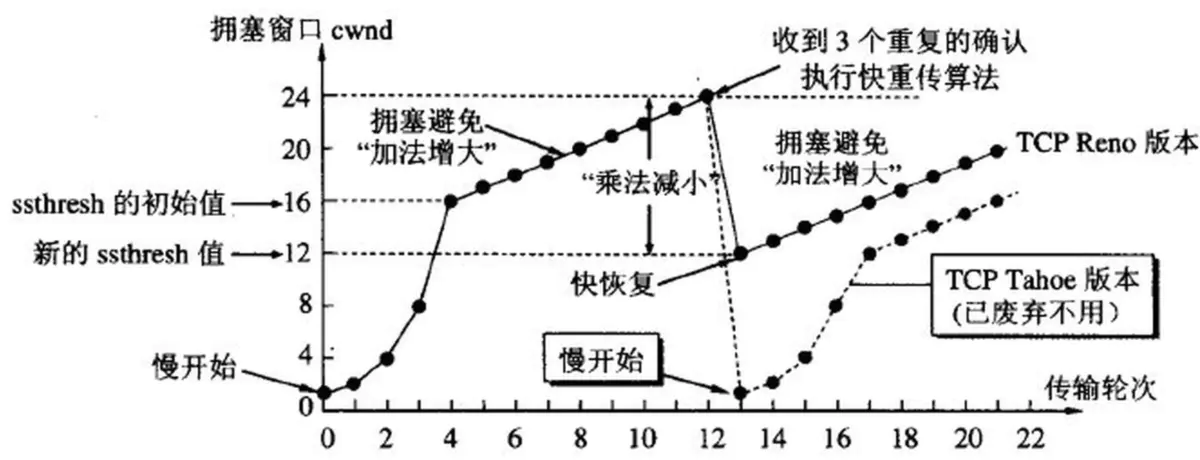
快重传配合使用的还有快恢复算法，有以下两个要点：

当发送方连续收到三个重复确认时，就把ssthresh门限减半（为了预防网络发生拥塞）。

但是接下去并不执行慢开始算法

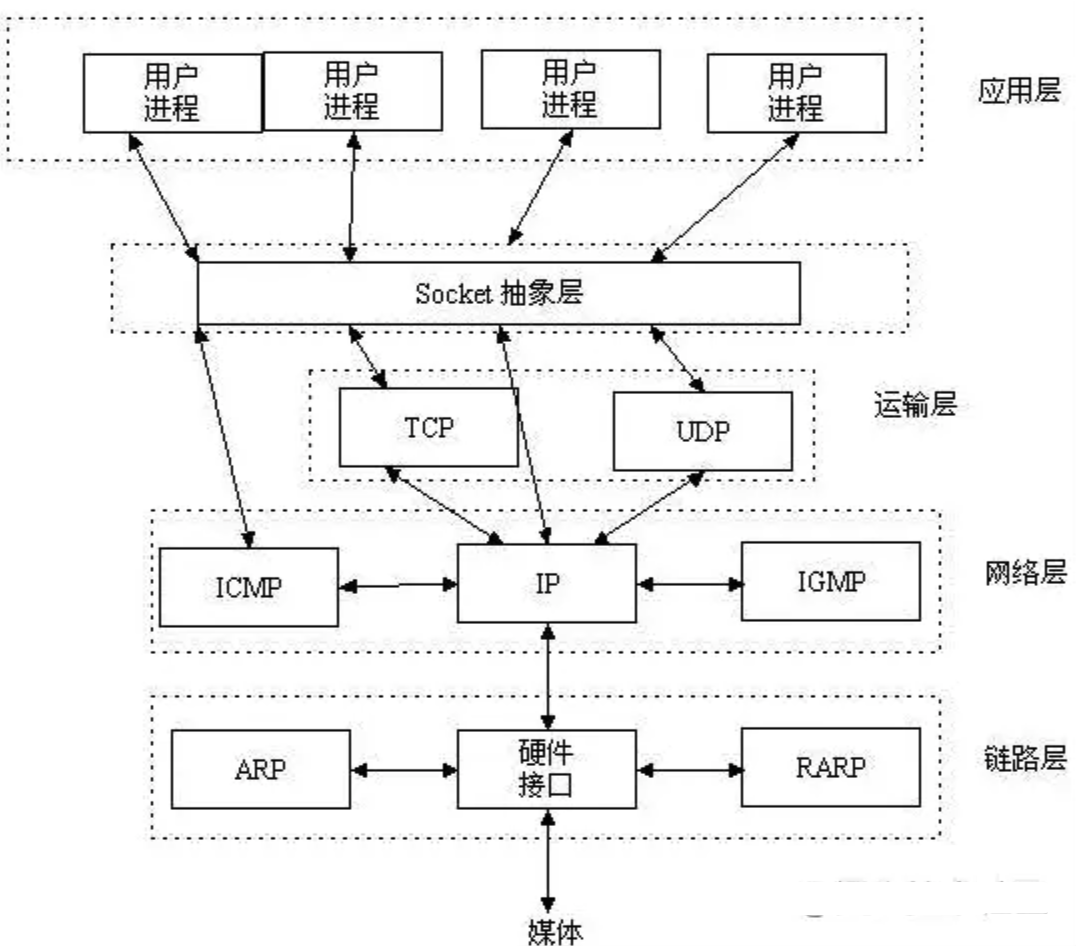
考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。

所以此时不执行慢开始算法，而是将cwnd设置为ssthresh减半后的值，然后执行拥塞避免算法，使cwnd缓慢增大。



### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "socket)Socket

即套接字，是应用层 与 TCP/IP 协议族通信的中间软件抽象层，表现为一个封装了 TCP / IP协议族 的编程接口（API）



Socket不是一种协议，而是一个编程调用接口（API），属于传输层（主要解决数据如何在网络中传输）

对用户来说，只需调用Socket去组织数据，以符合指定的协议，即可通信



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "udp)UDP

****UDP协议特点****

UDP是无连接的传输层协议；

UDP使用尽最大努力交付，不保证可靠交付；

UDP是面向报文的，对应用层交下来的报文，不合并，不拆分，保留原报文的边界；

UDP没有拥塞控制，因此即使网络出现拥塞也不会降低发送速率；

UDP支持一对一　一对多　多对多的交互通信；

UDP的首部开销小，只有８字节

****TCP和UDP的区别****

TCP是可靠传输，UDP是不可靠传输;

TCP面向连接，UDP无连接;

TCP传输数据有序，UDP不保证数据的有序性;

TCP不保存数据边界，UDP保留数据边界;

TCP传输速度相对UDP较慢;

TCP有流量控制和拥塞控制，UDP没有;

TCP是重量级协议，UDP是轻量级协议;

TCP首部较长20字节，UDP首部较短８字节;

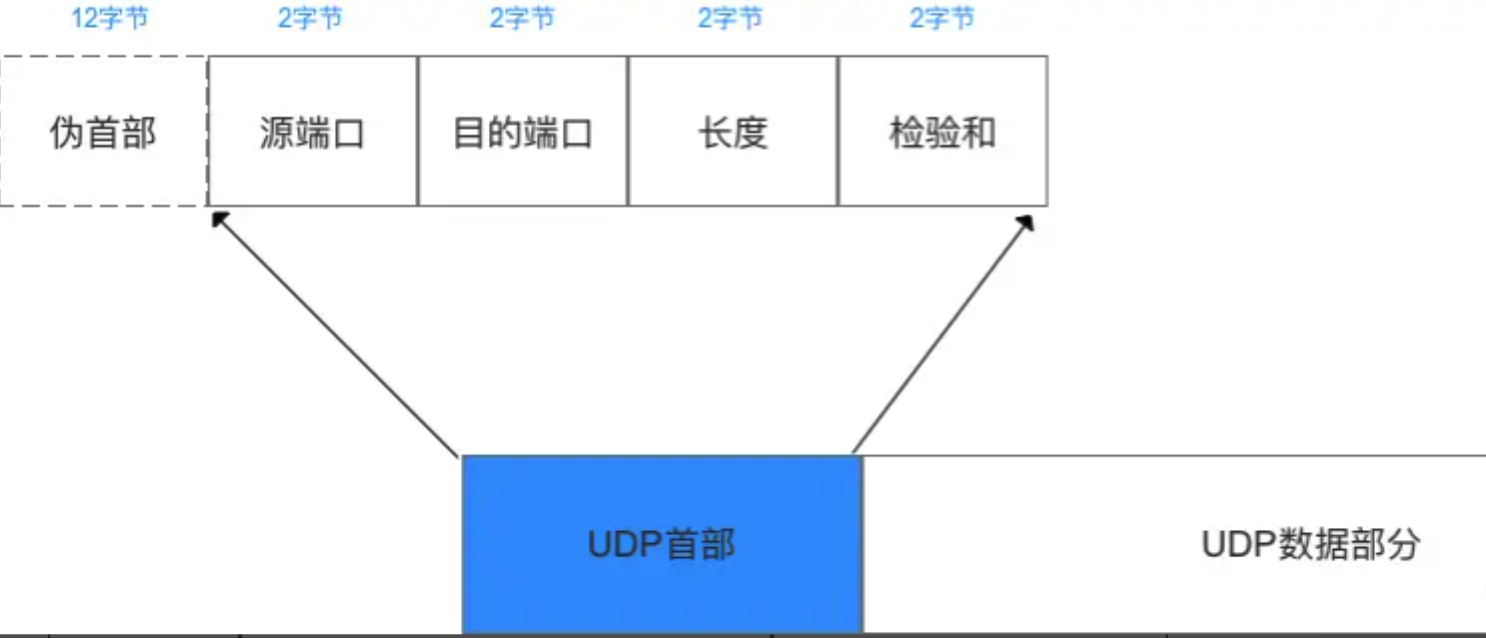
****基于TCP和UDP的常用协议****

HTTP、HTTPS、FTP、TELNET、SMTP(简单邮件传输协议)协议基于可靠的TCP协议。

TFTP、DNS、DHCP、TFTP、SNMP(简单网络管理协议)、RIP基于不可靠的UDP协议

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E6%8A%A5%E6%96%87%E6%AE%B5-1)报文段

UDP的报文段共有2个字段：数据字段 + 首部字段



****UDP报文中每个字段的含义如下：****

* 源端口：这个字段占据 UDP 报文头的前 16 位，通常包含发送数据报的应用程序所使用的 UDP 端口，接收端的应用程序利用这个字段的值作为发送响应的目的地址，这个字段是可选的，所以发送端的应用程序不一定会把自己的端口号写入该字段中，如果不写入端口号，则把这个字段设置为 0，这样，接收端的应用程序就不能发送响应了。
* 目的端口：接收端计算机上 UDP 软件使用的端口，占据 16 位。
* 长度：该字段占据 16 位，表示 UDP 数据报长度，包含 UDP 报文头和 UDP 数据长度，因为 UDP 报文头长度是 8 个字节，所以这个值最小为 8。
* 校验值：该字段占据 16 位，可以检验数据在传输过程中是否被损坏。

## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E7%BD%91%E7%BB%9C%E5%B1%82)网络层

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "mac%E5%9C%B0%E5%9D%80)MAC地址

MAC称为物理地址，也叫硬件地址，用来定义网络设备的位置，MAC地址是网卡出厂时设定的，是固定的（但可以通过在设备管理器中或注册表等方式修改，同一网段内的MAC地址必须唯一）。

MAC地址采用十六进制数表示，长度是6个字节（48位），分为前24位和后24位。

MAC地址对应于OSI参考模型的第二层数据链路层，工作在数据链路层的交换机维护着计算机MAC地址和自身端口的数据库，交换机根据收到的数据帧中的目的MAC地址字段来转发数据帧。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "ip%E5%9C%B0%E5%9D%80)IP地址

常见的IP地址分为IPv4与IPv6两大类，当前广泛应用的是IPv4，目前IPv4几乎耗尽，下一阶段必然会进行版本升级到IPv6；

IP地址是以网络号和主机号来标示网络上的主机的，我们把网络号相同的主机称之为本地网络，网络号不相同的主机称之为远程网络主机

本地网络中的主机可以直接相互通信；远程网络中的主机要相互通信必须通过本地网关（Gateway）来传递转发数据。

IP地址对应于OSI参考模型的第三层网络层，工作在网络层的路由器根据目标IP和源IP来判断是否属于同一网段，如果是不同网段，则转发数据包。

****IP地址格式和表示****

IP地址(IPv4)由32位二进制数组成，分为4段（4个字节），每一段为8位二进制数（1个字节）

每一段8位二进制，中间使用英文的标点符号.隔开

由于二进制数太长，为了便于记忆和识别，把每一段8位二进制数转成十进制，大小为0至255。

IP地址的这种表示法叫做****点分十进制表示法****。

IP地址表示为：xxx.xxx.xxx.xxx

举个栗子：210.21.196.6就是一个IP地址的表示。

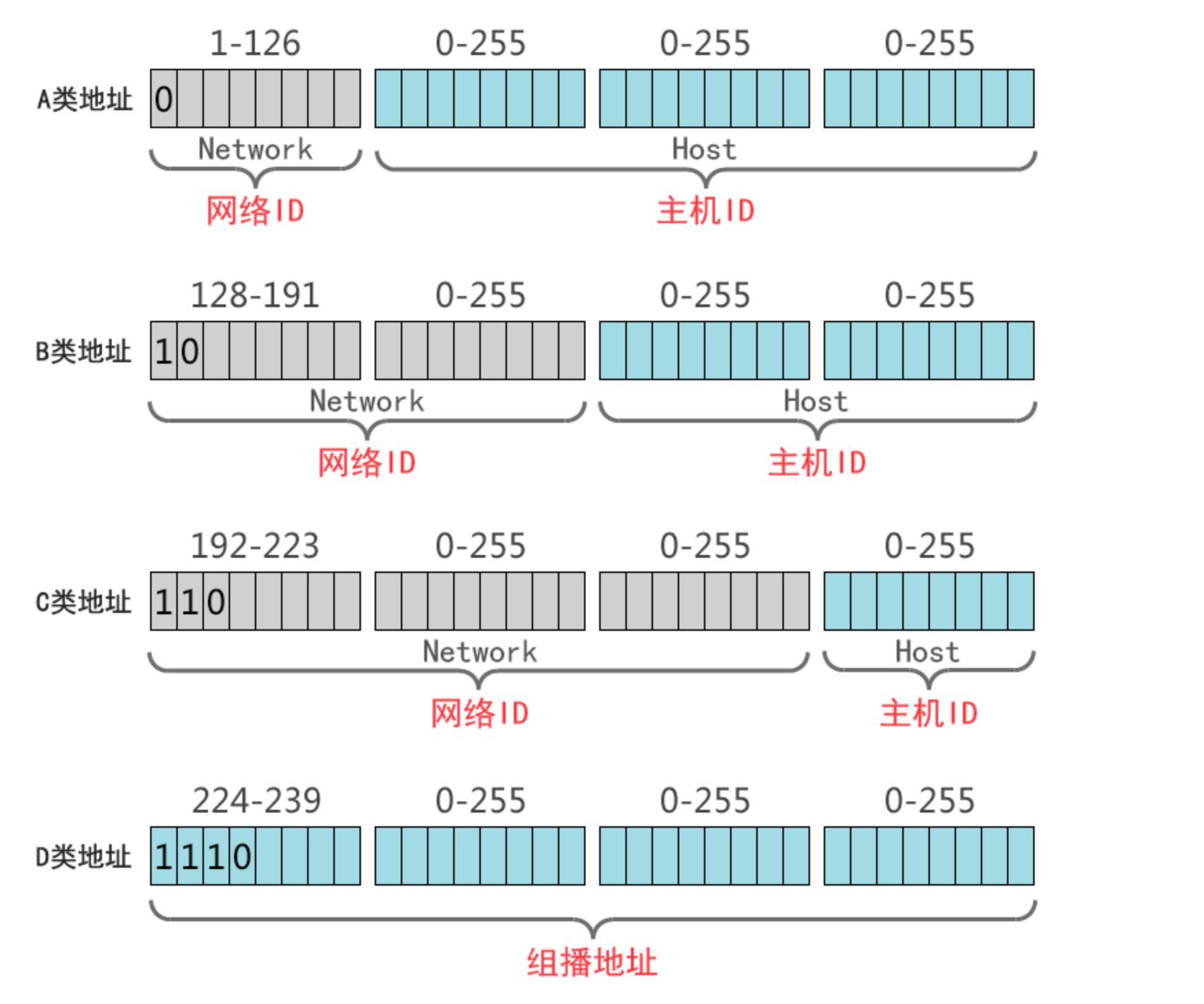
计算机的IP地址由两部分组成，一部分为网络标识，一部分为主机标识，同一网段内的计算机网络部分相同，主机部分不能同时重复出现。

****路由器****连接不同网段，负责不同网段之间的数据转发，****交换机****连接的是同一网段的计算机。

通过设置网络地址和主机地址，在互相连接的整个网络中保证每台主机的IP地址不会互相重叠，即IP地址具有了唯一性。

****IP地址分类详解****

IP地址分A、B、C、D、E五类，其中A、B、C这三类是比较常用的IP地址，D、E类为特殊地址。



### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E5%AD%90%E7%BD%91%E6%8E%A9%E7%A0%81)子网掩码

****子网掩码的概念及作用****

通过子网掩码，才能表明一台主机所在的子网与其他子网的关系，使网络正常工作。

子网掩码和IP地址做与运算，分离出IP地址中的网络地址和主机地址，用于判断该IP地址是在本地网络上，还是在远程网络网上。

子网掩码还用于将网络进一步划分为若干子网，以避免主机过多而拥堵或过少而IP浪费。

****子网掩码的组成****

同IP地址一样，子网掩码是由长度为32位二进制数组成的一个地址。

子网掩码32位与IP地址32位相对应，IP地址如果某位是网络地址，则子网掩码为1，否则为0。

举个栗子：如：11111111.11111111.11111111.00000000

左边连续的1的个数代表网络号的长度，（使用时必须是连续的，理论上也可以不连续），右边连续的0的个数代表主机号的长度。

****为什么要使用子网掩码****

两台主机要通信，首先要判断是否处于同一网段，即网络地址是否相同。

如果相同，那么可以把数据包直接发送到目标主机，否则就需要路由网关将数据包转发送到目的地。

可以这么简单的理解：A主机要与B主机通信，A和B各自的IP地址与A主机的子网掩码进行And与运算，看得出的结果：

1、结果如果相同，则说明这两台主机是处于同一个网段，这样A可以通过ARP广播发现B的MAC地址，B也可以发现A的MAC地址来实现正常通信。

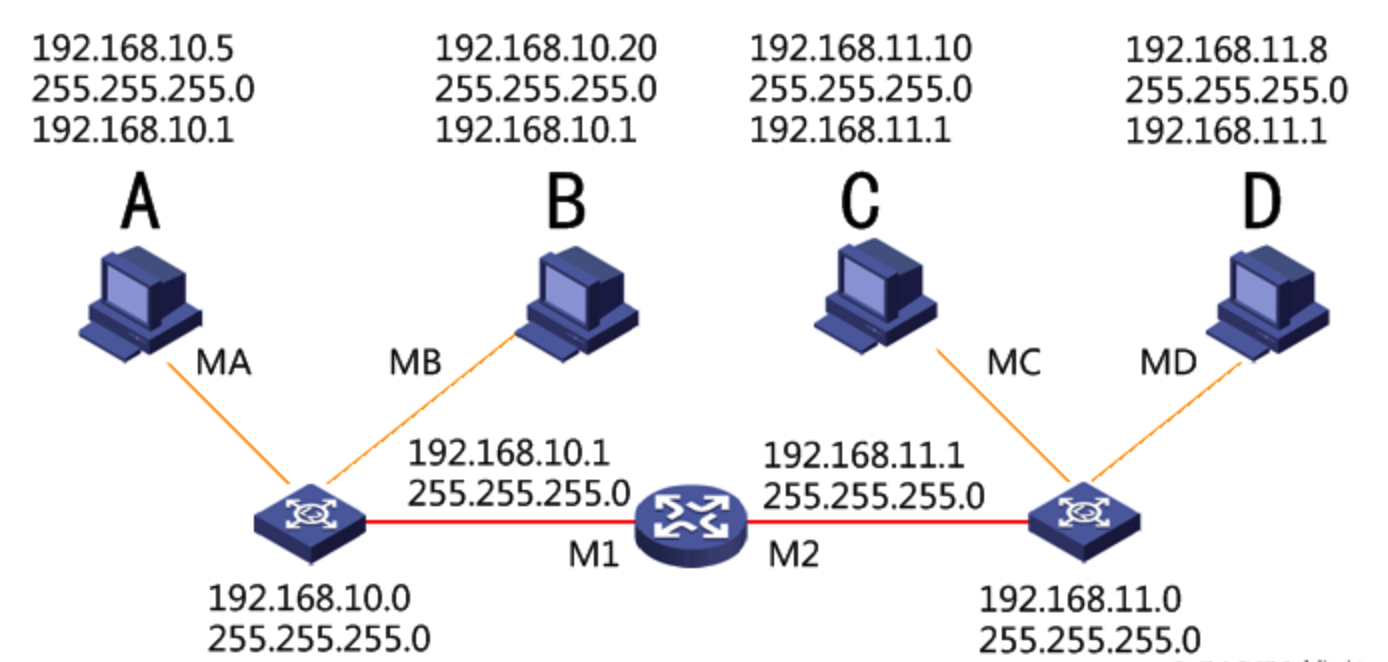
2、如果结果不同，ARP广播会在本地网关终结，这时候A会把发给B的数据包先发给本地网关，网关再根据B主机的IP地址来查询路由表，再将数据包继续传递转发，最终送达到目的地B。

计算机的网关（Gateway）就是到其他网段的出口，也就是路由器接口IP地址。

路由器接口使用的IP地址可以是本网段中任何一个地址，不过通常使用该网段的第一个可用的地址或最后一个可用的地址，这是为了尽可能避免和本网段中的主机地址冲突。

在如下拓扑图示例中，A与B，C与D，都可以直接相互通信（都是属于各自同一网段，不用经过路由器）

但是A与C，A与D，B与C，B与D它们之间不属于同一网段，所以它们通信是要经过本地网关，然后路由器根据对方IP地址，在路由表中查找恰好有匹配到对方IP地址的直连路由，于是从另一边网关接口转发出去实现互连



****子网掩码和IP地址的关系****

子网掩码是用来判断任意两台主机的IP地址是否属于同一网络的依据

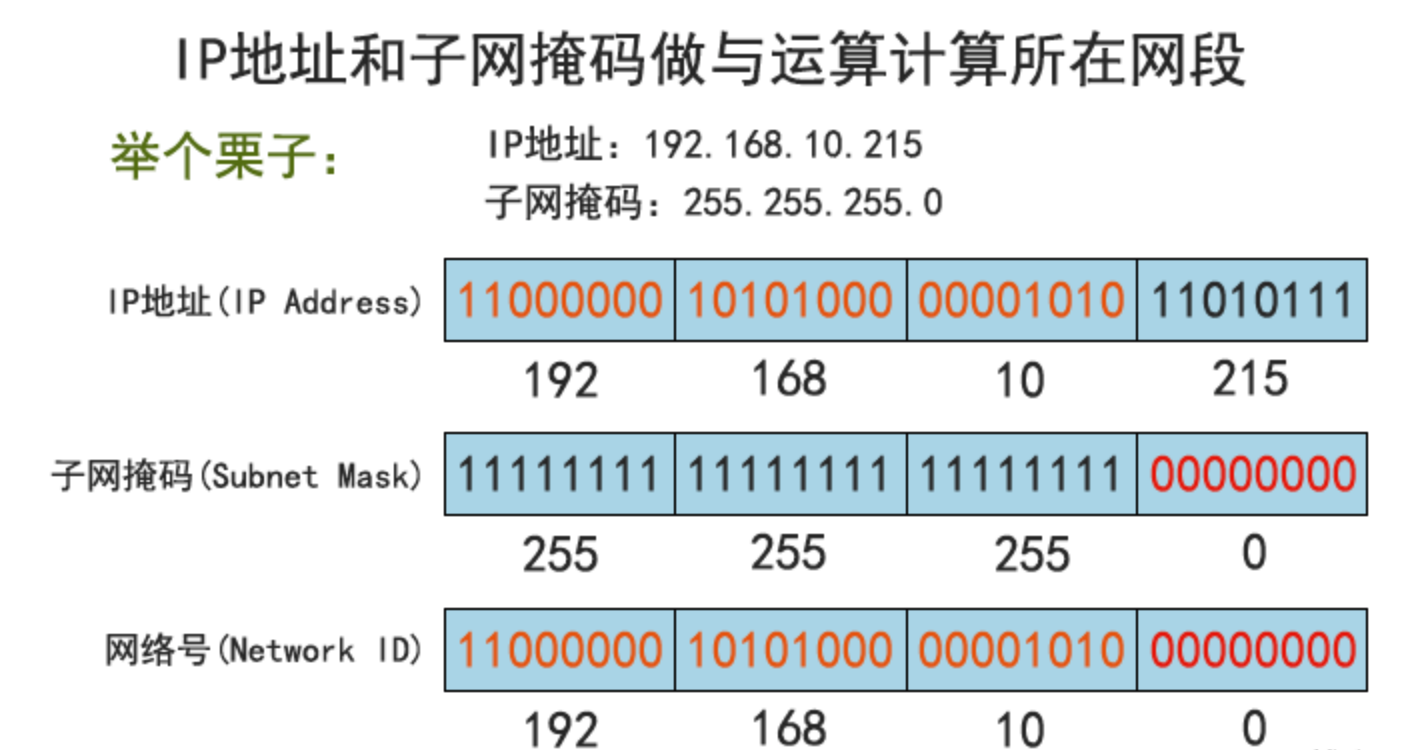
拿双方主机的IP地址和自己主机的子网掩码做与运算，如结果为同一网络，就可以直接通信

****如何根据IP地址和子网掩码，计算网络地址：****

将IP地址与子网掩码转换成二进制数。

将二进制形式的 IP 地址与子网掩码做与运算。

将得出的结果转化为十进制，便得到网络地址。



### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E7%BD%91%E5%85%B3)网关

网关实质上是一个网络通向其他网络的IP地址。

比如有网络A和网络B，网络A的IP地址范围为192.168.1.1~192. 168.1.254，子网掩码为255.255.255.0；

网络B的IP地址范围为192.168.2.1~192.168.2.254，子网掩码为255.255.255.0。

在没有路由器的情况下，两个网络之间是不能进行TCP/IP通信的，即使是两个网络连接在同一台交换机(或集线器)上，TCP/IP协议也会根据子网掩码(255.255.255.0)判定两个网络中的主机处在不同的网络里。

而要实现这两个网络之间的通信，则必须通过网关。

如果网络A中的主机发现数据包的目的主机不在本地网络中，就把数据包转发给它自己的网关，再由网关转发给网络B的网关，网络B的网关再转发给网络B的某个主机。网络B向网络A转发数据包的过程。

****所以说，只有设置好网关的IP地址，TCP/IP协议才能实现不同网络之间的相互通信。****

那么这个IP地址是哪台机器的IP地址呢？

网关的IP地址是具有路由功能的设备的IP地址，具有路由功能的设备有路由器、启用了路由协议的服务器(实质上相当于一台路由器)、代理服务器(也相当于一台路由器)。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "ping)Ping

Ping是我们测试网络连接的常用指令。

它利用ICMP报文检测网络连接。

****假设A ping B****

ping通知系统建立一个固定格式的ICMP请求数据包。

ICMP协议打包这个数据包和B的IP地址转交给IP协议层

IP层协议将机器B的IP地址为目的地址，本机的IP地址为源地址，加上一些头部必要的控制信息，构建一个IP数据包

获取B的MAC地址，做这个操作首先机器A会判断B是否在同一网段内，若IP层协议通过B的IP地址和自己的子网掩码，发现它跟自己属于同一网络，就直接在本网络查找这台机器的MAC，否则则通过路由器进行类似查找。

接下来是ARP协议根据IP地址查找MAC地址的过程:

若两台机器之前有过通信，在机器A的ARP缓存表里应该存有B的IP与其MAC地址的映射关系。

若没有，则通过发送ARP请求广播，得到回应的B机器MAC地址，并交给数据链路层

数据链路层构建一个数据帧，目的地址是IP层传过来的MAC地址，源地址是本机的MAC地址，再附加一些必要的控制信息，依据以太网的介质访问规则将他们传送出去

机器B收到这个数据帧后，先检查目的地址，和本机MAC地址对比：

符合，接受，接收后检查该数据帧，将IP数据包从帧中提取出来，交给本机的的IP地址协议层协议，IP协议层检查之后，将有用的信息提取给ICMP协议，后者处理，马上构建一个ICMP应答包，发送给A，其过程和主机A发送ICMP请求包到B的过程类似，但不用ARP广播收取A的信息，因为请求包中已经有足够的信息用于B回应A。

若不符合，丢弃。

可以知道PING的过程即一段发送报文和接受确认报文的过程，在来回直接可以计算时延。

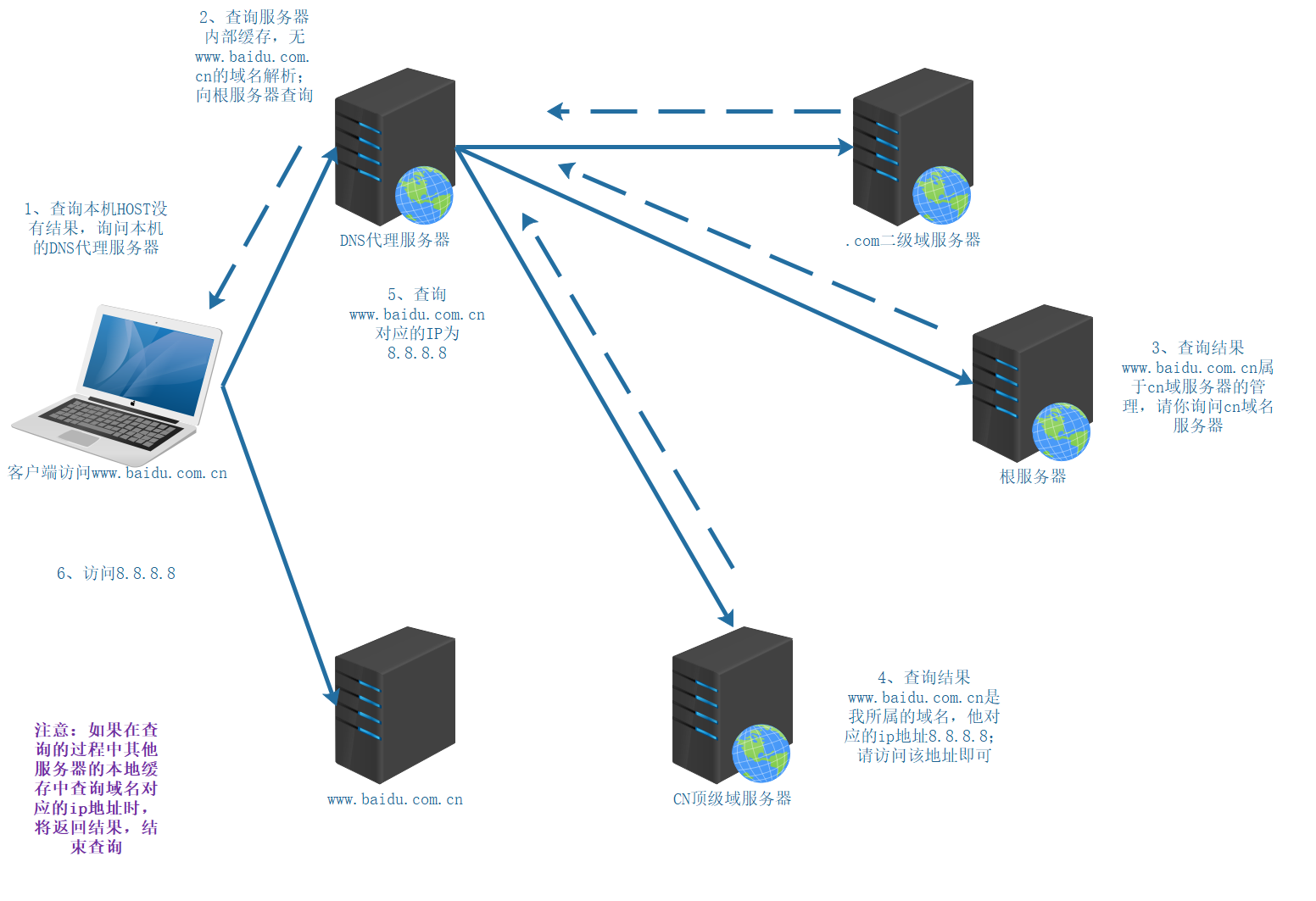
### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "dns)DNS

DNS通过主机名，最终得到该主机名对应的IP地址的过程叫做域名解析（或主机名解析）。

****通俗的讲****，我们更习惯于记住一个网站的名字，[www.baidu.comopen in new window](http://www.baidu.com/" \t "https://tobebetterjavaer.com/cs/_blank)，而不是记住它的ip地址，比如：167.23.10.2

****工作原理****

将主机域名转换为ip地址，属于应用层协议，使用UDP传输。



第一步，客户端向本地DNS服务器发送解析请求

第二步，本地DNS如有相应记录会直接返回结果给客户端，如没有就向DNS根服务器发送请求

第三步,DSN根服务器接收到请求，返回给本地服务器一个所查询域的主域名服务器的地址

第四步，本地dns服务器再向返回的主域名服务器地址发送查询请求

第五步，主域名服务器如有记录就返回结果，没有的话返回相关的下级域名服务器地址

第六步，本地DNS服务器继续向接收到的地址进行查询请求

第七步，下级域名服务器有相应记录，返回结果

第八步，本地dns服务器将收到的返回地址发给客户端，同时写入自己的缓存，以便下次查询

DNS域名查询实际上就是个不断递归查询的过程，直到查找到相应结果，需要注意的时，当找不到相应记录，会返回空结果，而不是超时信息

#### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "dns%E8%AE%B0%E5%BD%95)DNS记录

A记录

定义www.example.com的ip地址

www.example.com. IN A 139.18.28.5;

上面的就是一条 DNS 记录，纯文本即可。

www.example.com 是要解析的域名。

A 是记录的类型，A 记录代表着这是一条用于解析 IPv4 地址的记录。

从这条记录可知，www.example.com的 IP 地址是 139.18.28.5。

CNAME

CNAME用于定义域名的别名，如下面这条 DNS 记录：

定义www.example.com的别名

a.example.com. IN CNAME b.example.com.

这条 DNS 记录定义了 a.example.com 是 b.example.com 的别名。

用户在浏览器中输入 a.example.com 时候，通过 DNS 查询会知道 a.example.com 是 b.example.com 的别名，因此需要实际 IP 的时候，会去拿 b.example.com 的 A 记录。

当你想把一个网站迁移到新域名，旧域名仍然保留的时候；还有当你想将自己的静态资源放到 CDN 上的时候，CNAME 就非常有用。

AAAA 记录

A 记录是域名和 IPv4 地址的映射关系。和 A 记录类似，AAAA 记录则是域名和 IPv6 地址的映射关系。

MX记录

MX 记录是邮件记录，用来描述邮件服务器的域名。

在工作中，我们经常会发邮件到某个同事的邮箱。

比如说，发送一封邮件到 xiaoming@xiaoflyfish.com，那么如何知道哪个 IP 地址是邮件服务器呢？

这个时候就可以用到下面这条 MX 记录：

IN MX mail.xiaoflyfish.com

mail.xiaoflyfish.com 的 IP 地址可以通过查询 mail.xiaoflyfishcom 的 A 记录和 AAAA 记录获得。

NS 记录

NS记录是描述 DNS 服务器网址。从 DNS 的存储结构上说，Name Server 中含有权威 DNS 服务的目录。

也就是说，NS 记录指定哪台 Server 是回答 DNS 查询的权威域名服务器。

当一个 DNS 查询看到 NS 记录的时候，会再去 NS 记录配置的 DNS 服务器查询，得到最终的记录。如下面这个例子：

a.com. IN NS ns1.a.com.

a.com. IN NS ns2.a.com.

当解析 a.com 地址时，我们看到 a.com 有两个 NS 记录，所以确定最终 a.com 的记录在 ns1.a.com 和 ns2.a.com 上。

从设计上看，ns1 和 ns2 是网站 a.com 提供的智能 DNS 服务器，可以提供负载均衡、分布式 Sharding 等服务。

比如当一个北京的用户想要访问 a.com 的时候，ns1 看到这是一个北京的 IP 就返回一个离北京最近的机房 IP。

上面代码中 a.com 配置了两个 NS 记录。

通常 NS 不会只有一个，这是为了保证高可用，一个挂了另一个还能继续服务。

通常数字小的 NS 记录优先级更高，也就是 ns1 会优先于 ns2 响应。

配置了上面的 NS 记录后，如果还配置了 a.com 的 A 记录，那么这个 A 记录会被 NS 记录覆盖。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "arp%E5%8D%8F%E8%AE%AE)ARP协议

ARP即地址解析协议， 用于实现从 IP 地址到 MAC 地址的映射，即询问目标IP对应的MAC地址。

****ARP协议的工作过程****

首先，每个主机都会有自己的ARP缓存区中建立一个ARP列表，以表示IP地址和MAC地址之间的对应关系

当源主机要发送数据时，首先检测ARP列表中是否对应IP地址的目的主机的MAC地址，如果有，则直接发送数据，如果没有，就向本网段的所有主机发送ARP数据包

当本网络的所有主机收到该ARP数据包时，首先检查数据包中的IP地址是否是自己的IP地址，如果不是，则忽略该数据包，如果是，则首先从数据包中取出源主机的IP和MAC地址写入到ARP列表中，如果存在，则覆盖然后将自己的MAC地址写入ARP响应包中，告诉源主机自己是它想要找的MAC地址

源主机收到ARP响应包后，将目的主机的IP和MAC地址写入ARP列表，并利用此信息发送数据，如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E6%95%B0%E5%AD%97%E7%AD%BE%E5%90%8D)数字签名

网络传输过程中需要经过很多中间节点，虽然数据无法被解密，但可能被篡改

数字签名校验数据的完整性

****数字签名有两种功效****：

能确定消息确实是由发送方签名并发出来的，因为别人假冒不了发送方的签名。

数字签名能确定消息的完整性，证明数据是否未被篡改过。

将一段文本先用Hash函数生成消息摘要，然后用发送者的私钥加密生成数字签名，与原文文一起传送给接收者

接收者只有用发送者的公钥才能解密被加密的摘要信息，然后用HASH函数对收到的原文产生一个摘要信息，与上一步得到的摘要信息对比。

如果相同，则说明收到的信息是完整的，在传输过程中没有被修改，否则说明信息被修改过，因此数字签名能够验证信息的完整性。

## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "sql%E6%B3%A8%E5%85%A5)SQL注入

SQL注入的原理是将SQL代码伪装到输入参数中，传递到服务器解析并执行的一种攻击手法。

****SQL注入攻击实例****

比如，在一个登录界面，要求输入用户名和密码，可以这样输入实现免帐号登录：

用户名： ‘or 1 = 1 --

密 码：

用户一旦点击登录，如若没有做特殊处理，那么这个非法用户就很得意的登陆进去了。

下面我们分析一下：从理论上说，后台认证程序中会有如下的SQL语句：

String sql = “select \* from user\_table where username=’ “+userName+” ’ and password=’ “+password+” ‘”;

因此，当输入了上面的用户名和密码，上面的SQL语句变成：

SELECT \* FROM user\_table WHERE username=’’or 1 = 1 –- and password=’’

分析上述SQL语句我们知道，username=‘ or 1=1 这个语句一定会成功；然后后面加两个 -，这意味着注释，它将后面的语句注释，让他们不起作用，这样，上述语句永远都能正确执行，用户轻易骗过系统，获取合法身份。

****应对方法****

预编译

使用预编译手段，绑定参数是最好的防SQL注入的方法。

目前许多的ORM框架及JDBC等都实现了SQL预编译和参数绑定功能，攻击者的恶意SQL会被当做SQL的参数而不是SQL命令被执行。

在mybatis的mapper文件中，对于传递的参数我们一般是使用 ## 和$来获取参数值。

当使用#时，变量是占位符，就是一般我们使用java的jdbc的PrepareStatement时的占位符，所有可以防止sql注入；

当使用$时，变量就是直接追加在sql中，一般会有sql注入问题。

使用正则表达式过滤传入的参数

过滤参数中含有的一些数据库关键词

## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E5%8A%A0%E5%AF%86%E7%AE%97%E6%B3%95)加密算法

加密算法分****对称加密**** 和 ****非对称加密****，其中对称加密算法的加密与解密密钥相同，非对称加密算法的加密密钥与解密密钥不同，此外，还有一类不需要密钥的****散列算法****。

常见的 ****对称加密**** 算法主要有 DES、3DES、AES 等，常见的 ****非对称算法**** 主要有 RSA、DSA 等，****散列算法**** 主要有 SHA-1、MD5 等。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E5%AF%B9%E7%A7%B0%E5%8A%A0%E5%AF%86)对称加密

在 ****对称加密算法**** 中，使用的密钥只有一个，发送和接收双方都使用这个密钥对数据进行 ****加密**** 和 ****解密****。

数据加密过程：在对称加密算法中，数据发送方 将 ****明文**** (原始数据) 和 ****加密密钥**** 一起经过特殊 ****加密处理****，生成复杂的 ****加密密文**** 进行发送。

数据解密过程：****数据接收方**** 收到密文后，若想读取原数据，则需要使用 ****加密使用的密钥**** 及相同算法的 ****逆算法**** 对加密的密文进行解密，才能使其恢复成 ****可读明文****。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E9%9D%9E%E5%AF%B9%E7%A7%B0%E5%8A%A0%E5%AF%86)非对称加密

****非对称加密算法****，它需要两个密钥，一个称为 ****公开密钥**** (public key)，即 ****公钥****，另一个称为 ****私有密钥**** (private key)，即 ****私钥****。

因为 ****加密**** 和 ****解密**** 使用的是两个不同的密钥，所以这种算法称为 ****非对称加密算法****。

如果使用 ****公钥**** 对数据 ****进行加密****，只有用对应的 ****私钥**** 才能 ****进行解密****。

如果使用 ****私钥**** 对数据 ****进行加密****，只有用对应的 ****公钥**** 才能 ****进行解密****。

****例子****：甲方生成 ****一对密钥**** 并将其中的一把作为 ****公钥**** 向其它人公开，得到该公钥的 ****乙方**** 使用该密钥对机密信息 ****进行加密**** 后再发送给甲方，甲方再使用自己保存的另一把 ****专用密钥**** (****私钥****)，对 ****加密**** 后的信息 ****进行解密****。



## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E7%BD%91%E7%BB%9C%E6%94%BB%E5%87%BB)网络攻击

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "csrf%E5%92%8Cxss)CSRF和XSS

****XSS：****

跨站脚本是一种网站应用程序的安全漏洞攻击，是代码注入的一种。

它允许恶意用户将代码注入到网页上，其他用户在观看网页时就会受到影响，这类攻击通常包含了HTML以及用户端脚本语言。

比如通过客户端脚本语言（最常见如：JavaScript）

在一个论坛发帖中发布一段恶意的JavaScript代码就是脚本注入，如果这个代码内容有请求外部服务器，那么就叫做XSS

****XSS攻击分类****

反射性XSS攻击 (非持久性XSS攻击)

例如，正常发送消息：

http://www.test.com/message.php?send=Hello,World！

接收者将会接收信息并显示HelloWorld；但是，非正常发送消息：

http://www.test.com/message.php?send=<script>alert(‘foolish!’)</script>！

接收者接收消息显示的时候将会弹出警告窗口！

持久性XSS攻击 (留言板场景)

一般指XSS攻击代码存储在网站数据库，当一个页面被用户打开的时候执行。

也就是说，每当用户使用浏览器打开指定页面时，脚本便执行。

与非持久性XSS攻击相比，持久性XSS攻击危害性更大。

从名字就可以了解到，持久性XSS攻击就是将攻击代码存入数据库中，然后客户端打开时就执行这些攻击代码。

例如，留言板表单中的表单域：

<input type="text" name="content" value="这里是用户填写的数据">

正常操作流程是：用户是提交相应留言信息 — 将数据存储到数据库 — 其他用户访问留言板，应用去数据并显示；

而非正常操作流程是攻击者在value填写:

<script>alert(‘foolish!’)；</script> <!--或者html其他标签（破坏样式。。。）、一段攻击型代码-->

并将数据提交、存储到数据库中；当其他用户取出数据显示的时候，将会执行这些攻击性代码

****CSRF：****

跨站请求伪造，是一种挟制用户在当前已登录的Web应用程序上执行非本意的操作的攻击方法。

比如冒充用户发起请求（在用户不知情的情况下），完成一些违背用户意愿的请求（如恶意发帖，删帖，改密码，发邮件等）。

### [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "dos%E6%94%BB%E5%87%BB)DOS攻击

DOS：中文名称是拒绝服务，该攻击的效果是使得计算机或网络无法提供正常的服务

****DOS攻击的原理：****

首先攻击者向被攻击的服务器发送大量的虚假IP请求，被攻击者在收到请求后返回确认信息，等待攻击者进行确认，该过程需要TCP的三次握手，由于攻击者发送的请求信息是虚假的，所以服务器接收不到返回的确认信息，在一段时间内服务器会处与等待状态，而分配给这次请求的资源却被有被释放

当被攻击者等待一定的时间后，会因连接超时而断开，这时攻击者在次发送新的虚假信息请求，这样最终服务器资源被耗尽，直到瘫痪

****DDOS：中文名称是分布式拒绝服务攻击****

指的是攻击者控制多台主机同时向同一主机或网络发起DOS攻击

DRDoS分布反射式拒绝服务攻击这是DDoS攻击的变形

****DDOS究竟如何攻击****

目前最流行也是最好用的攻击方法就是使用SYN-Flood进行攻击，SYN-Flood也就是SYN洪水攻击

SYN-Flood不会完成TCP三次握手的第三步，也就是不发送确认连接的信息给服务器，这样，服务器无法完成第三次握手，但服务器不会立即放弃，服务器会不停的重试并等待一定的时间后放弃这个未完成的连接，这段时间叫做SYN timeout，这段时间大约30秒-2分钟左右。

若是一个用户在连接时出现问题导致服务器的一个线程等待1分钟并不是什么大不了的问题，但是若有人用特殊的软件大量模拟这种情况，那后果就可想而知了。一个服务器若是处理这些大量的半连接信息而消耗大量的系统资源和网络带宽，这样服务器就不会再有空余去处理普通用户的正常请求(因为客户的正常请求比率很小)，这样这个服务器就无法工作了，这种攻击就叫做SYN-Flood攻击

到目前为止，进行DDoS攻击的防御还是比较困难的

首先，这种攻击的特点是它利用了TCP/IP协议的漏洞，除非你不用TCP/IP，才有可能完全抵御住DDoS攻击

不过这不等于我们就没有办法阻挡DDoS攻击，我们可以尽力来减少DDoS的攻击

****下面就是一些防御方法:****

关闭不必要的服务

限制同时打开的SYN半连接数目

缩短SYN半连接的time out 时间

正确设置防火墙

禁止对主机的非开放服务的访问

限制特定IP地址的访问

启用防火墙的防DDoS的属性

## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "cookie%E5%92%8Csession)Cookie和Session

Session 是****基于Cookie 实现****的另一种记录服务端和客户端会话状态的机制。

Session 是存储在服务端，而 SessionId 会被存储在客户端的 Cookie 中。

Session 的****认证过程****：

1. 客户端第一次发送请求到服务端，服务端根据信息创建对应的 Session，并在响应头返回 SessionID
2. 客户端接收到服务端返回的 SessionID 后，会将此信息存储在 Cookie 上，同时会记录这个 SessionID 属于哪个域名
3. 当客户端再次访问服务端时，请求会自动判断该域名下是否存在 Cookie 信息，如果有则发送给服务端，服务端会从 Cookie 中拿到 SessionID，再根据 SessionID 找到对应的 Session，如果有对应的 Session 则通过，继续执行请求，否则就中断

****Cookie和Session的区别****

1. 安全性，因为 Cookie 可以通过客户端修改，而 Session 只能在服务端设置，所以安全性比 Cookie 高，一般会用于验证用户登录状态
2. 适用性，Cookie 只能存储字符串数据，而 Session 可以存储任意类型数据
3. 有效期，Cookie 可以设置任意时间有效，而 Session 一般失效时间短

、

## [#](https://tobebetterjavaer.com/cs/wangluo.html" \l "%E5%B8%B8%E8%A7%81%E9%9D%A2%E8%AF%95%E9%A2%98)常见面试题

****在浏览器地址栏键入URL****

1.DNS解析：浏览器会依据URL逐层查询DNS服务器缓存，解析URL中的域名对应的IP地址，DNS缓存从近到远依次是浏览器缓存、系统缓存、路由器缓存、IPS服务器缓存、域名服务器缓存、顶级域名服务器缓存。

从哪个缓存找到对应的IP直接返回，不再查询后面的缓存。

2.TCP连接：结合三次握手

3.发送HTTP请求：浏览器发出读取文件的HTTP请求，该请求发送给服务器

4.服务器处理请求并返回HTTP报文：服务器对浏览器请求做出响应，把对应的带有HTML文本的HTTP响应报文发送给浏览器

5.浏览器解析渲染页面

6.连接结束：浏览器释放TCP连接，该步骤即四次挥手。

第5步和第6步可以认为是同时发生的，哪一步在前没有特别的要求