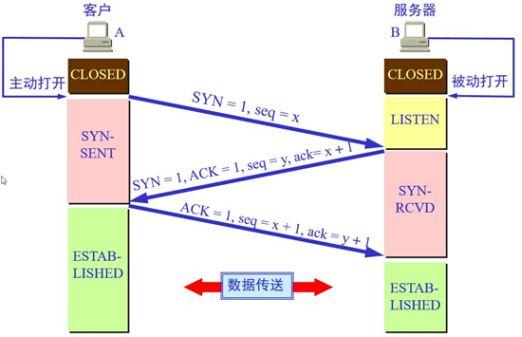
## TCP三次握手（都能必免服务端资源的浪费）



**SYN**表示“请求建立新连接”;

**seq**表示已经发送的序列号，ack表示已经收到的序列号通信双方需要判断自己已经发送的数据包是否都被接收方收到， 如果没收到， 就需要重发。 为了实现这个需求， 很自然地就会引出序号（sequence number） 和 确认号（acknowledgement number） 的使用。

发送方在发送数据包（假设大小为 10 byte）时， 同时送上一个序号( 假设为 500)，那么接收方收到这个数据包以后， 就可以回复一个确认号（510 = 500 + 10） 告诉发送方 “我已经收到了你的数据包， 你可以发送下一个数据包， 序号从 510 开始” 。

这样发送方就可以知道哪些数据被接收到，哪些数据没被接收到， 需要重发。

**ACK**标志位为1时，ack确认序号字段才有效

**为什么要三次握手：**

**1.**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

2.防止已失效（网络延时导致很久才到达服务端，请求端由于未按时收到回复又进行了请求）的连接请求又传送到服务器，浪费服务器资源

### 两次握手和三次握手遇到问题的情况（这就是为什么要三次握手的原因）

**（本质：将问题转移到客户端，不浪费服务端资源）**

1. 第一次握手成功，但第二次握手未接收到：

两次握手情况下：服务端认为连接成功，但客户端未收到第二次握手，此时服务端发送消息，客户端会忽略，而客户端会等待，或者重发第一次握手，但服务端的资源就一直被占用了。

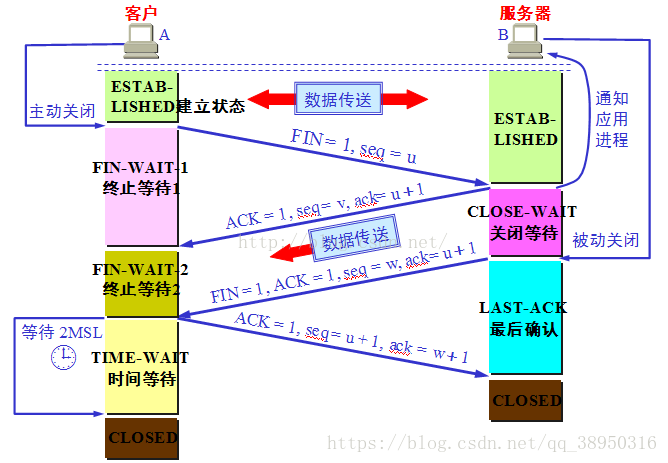
三次握手情况下：服务端就等待第三次握手，不会发送消息，白白浪费资源，客户端久等无望重新发起连接，一切又开始。

1. 第三次握手失败情况下：

三次握手情况下：服务端久等无握手就超时重发第二次握手，最终实现第三次握手。

期间客户端以为连接建立好了会发送一些资源，但一会又会收到第二次握手，让它回到正轨。

## 2.tcp四次挥手



**过程： 1.A主动发起FIN**

**2.B发送完剩下的数据和ack让客户端有底不会再次发送fin包**

**3.B发送FIN**

**4.A发送确认**

**为什么有四次挥手：**

**因为服务端有东西可能没发完，需要把ACK和FIN包分开发。**

首先我们已经知道了TCP的连接是**全双工**的，可以发送也可以接收，主动关闭方发送FIN，表明主动方已经没有数据需要发送了，告诉被动方我要断了，被动方回一个ACK，表明已知晓主动方无数据，准备断开了。

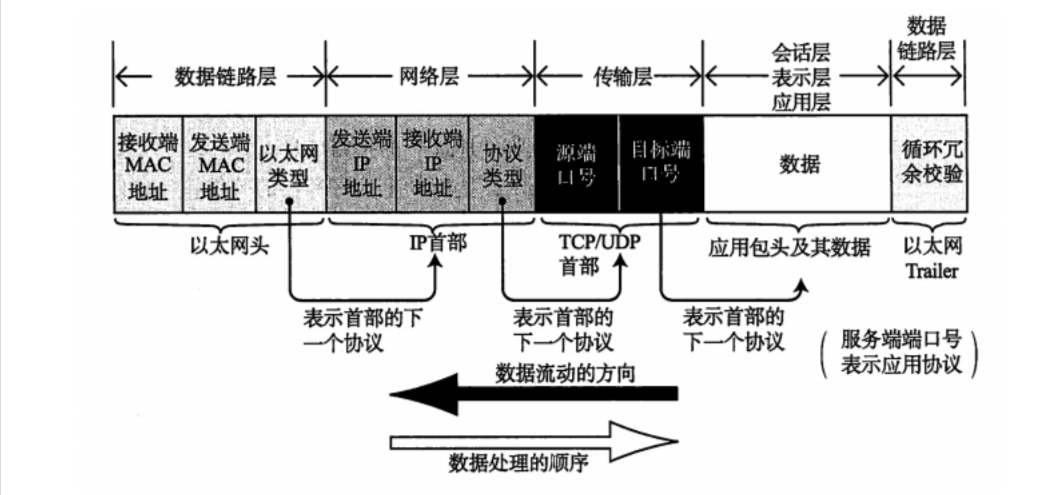
但是被动方也就是服务端可能依旧有数据还没有发送完毕，客户端没有数据发送不代表服务端没有数据发送啊，于是等服务端数据发送完毕。等到服务端这一端数据发完了，就可以彻底断开连接的，所以服务端也发了一个FIN包，告诉客户端我工作做完了，我也要关了，等到服务端收到客户端的ACK或者超出等待时间那么就断开连接，此时的连接才算真正意义上的断开。

## 3.为什么要等待2msl

1.**保证对方一定收到ACK**，因为确保服务器真正的关闭，如果服务端未收到ACK时会再发送FIN，此时就要求客户端未关闭。

2.**避免新的连接被网络中的重复旧数据影响**，2MSL足以让分组最多存活msl秒被丢弃。

## osi七层网络模型：



**每层所增加的数据**

1. 物理层：实现**相邻**计算机间**比特流的传输**，不用考虑介质什么的。（这一层数 据称为比特流）
2. 数据链路层：数据链路层是解决**同一网络**内节点之间的通信（这一层数据称为 帧）

封装成帧（最短64字节：发送方能检测到冲突的最小时间只有检测到冲突才 会重传，最长1518字节：这是规定）

帧的转换（从上到下把帧拆解为位流，从下到上把位流封装成帧），

差错校验判断数据有无错误，

流量控制（**相邻**设备间流量控制）

**交换机：**它维护了一张 MAC 地址表，用来反映各**端口**和 **MAC** 地址的对应关系，以便做好数据转发工作

1. 网络层：解决不同**子网间**的通信（这一层数据称为包package）

不同子网间通信过程：

主机A知道主机B的ip地址了，但是还不知道主机B的MAC地址

主机A通过arp协议找到主机B的MAC地址

arp过程

（

所以这时 A 会发一个 ARP 广播(ARP请求分组），内容是：我的ip地址为：XXXX，MAC 地址为 ：XXXX，询问 IP 地址为 192.168.0.2 的 MAC 地址是多少？在本局域网运行的所有主机上运行的ARP进程都会收到此ARP请求分组，但只有B会响应并向A发送一个ARP响应分组。  
2) 交换机收到该帧后，发现是广播帧，于是转发到其它所有端口，并且会将该帧的源 MAC 地址（即 A 的 MAC 地址）添加到自己的 MAC 地址表中去  
3) B 当然会收到该帧，于是和自己的 IP 进行对比，发现匹配后，再以Ａ为目标Ｍ AC 地址响应Ａ的ＡＲＰ请求帧  
4) 交换机收到Ｂ的回复帧后，将该帧的目标ＭＡＣ地址与自己的ＭＡＣ地址表进行对照，发现该帧对应的的端口是 F0/1 ，于是将帧转发到 F0/1 端口（如果数据帧的目标 MAC 地址在表中不存在，则会转发到除源端口外的其它所有端口），同时将该帧的源 MAC 地址（即 B 的 MAC 地址）加到自己的 MAC 地址表中  
5) A 收到 B 的回复帧后，得知 192.168.0.2 的 MAC 地址是 2c 26.1ae3.2222 ，于是该信息保存到本地的 ARP 高速缓存中，同时以 B 的 MAC 地址为目标地址将要传的数据封装成帧，发送出去  
6) 交换机再次收到 A 的数据帧，发现目的 MAC 地址是 2c 26.1ae3.2222 ，并且该地址对应的端口是 F0/2 ，于是将该数据转发到 F0/2 端口  
7) B 成功收到 A 发来的数据

）

数据发送

（

1) 由于 B 的 IP 地址并没有和 A 在一个网段，所以当 A 向 B 发送数据时， A 并不会直接把数据给 B ，而是交给自己的网关，也就是 192.168.0.254 ，所以 A 首先会 ARP 广播请求 192.168.0.254 的 MAC 地址  
2) A 得到网关的 MAC 地址后，以它为数据帧的目标 MAC 地址进行封装数据，并发送出去  
3) routerA 收到该帧后，检查该帧的目标 IP ，并到自己的路由表查找如何到达该网段（目的IP与各网络的掩码相与：直接较付->特定主机路由->间接交付->默认路由)，找到下一跳地址是 routerB 的 s0 端口，于是将数据重新封装，将源地址改为 s0 端口 MAC 地址，目标 MAC 地址改为 routerB 的 s0 端口 MAC 址址，并发送给 routerB  
4) 中间路由器传递过程同理  
5) 最后一个路由（此例为 routerC ）收到该帧，发现目标 IP 就在自己的直连网段，于是查看 ARP 缓存，如果找到该 IP 的 MAC 地址，则以该 MAC 地址封装数据发送出去，如果在 ARP 缓存没找到，则发出 ARP 广播，请求该 IP 的 MAC 地址，得到对应的 MAC 地址后，再发送给主机 B

）

在以上数据传递过程中，我们发现，数据帧的源 IP 和目标 IP 始终是不变的，而经过每个路由进行重新封装数据时 MAC 地址则在不断的变化，总是以自己的地址作为源 MAC 地址，下一跳的地址作为目标 MAC 地址。

**虽然IP看起来是能进行子网间的通信，但是本质还是在进行节点之间的数据传输，所以要配合MAC一起使用，IP指明目标方向，MAC指向具体哪台主机。**

进行路由（选择最好的路径）

流量控制（**从源节点到目的节点间的流量**）

**路由器：**能知道到达目标的下一ip地址，路由表中**不包含**MAC地址，再查寻ARP表就能知道吓一跳MAC地址，

**每台主机中都有一张ARP表，它记录着主机的IP地址和MAC地址的对应关系**

1. **传输层：**segment （数据段）

传输层的作用是向高层屏蔽下层数据通信的细节，即向用户透明地传送报文，建立**端口到端口**的通信

1. **应用层的主要功能是：利用下层提供的服务实现特定的功能**⑴ 文件抄传输、访问和管理  
   ⑵ 电子邮件  
   ⑶ [虚拟终端](https://www.baidu.com/s?wd=%E8%99%9A%E6%8B%9F%E7%BB%88%E7%AB%AF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)  
   ⑷ 简单[网络管理](https://www.baidu.com/s?wd=%E7%BD%91%E7%BB%9C%E7%AE%A1%E7%90%86&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)  
   ⑸ 查询服务和远程作业登录  
   应用层也称为应用实体（AE），它由若干个特定应用服务zd元素（SASE）和一个或多个公用应用服务元素（CASE）组成。每个SASE提供特定的应用服务，例如文件运输访问和管理（FTAM）、电子文电处理（MHS）、[虚拟终端](https://www.baidu.com/s?wd=%E8%99%9A%E6%8B%9F%E7%BB%88%E7%AB%AF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)协议（VAP）等

## HTTP

### 重定向和转发

**转发是服务器行为，重定向是客户端行为。**

**转发过程：客户浏览器发送http请求——》web服务器接受此请求——》调用内部的一个方法在容器内部完成请求处理和转发动作——》将目标资源发送给客户；**

在这里，转发的路径必须是同一个web容器下的url，其不能转向到其他的web路径上去，中间传递的是自己的容器内的request。在客户浏览器路径栏显示的**仍然是其第一次访问的路径**，也就是说客户是感觉不到服务器做了转发的。转发行为是浏览器只做了一次访问请求。

**重定向过程：客户浏览器发送http请求——》web服务器接受后发送302状态码响应及对应新的location给客户浏览器——》客户浏览器发现是302响应，则自动再发送一个新的http请求，请求url是新的location地址——》服务器根据此请求寻找资源并发送给客户。**

在这里location可以重定向到任意URL，既然是浏览器重新发出了请求，则就没有什么request传递的概念了。在客户浏览器路径栏显示的是其**重定向的路径**，客户可以观察到地址的变化的。重定向行为是浏览器做了至少两次的访问请求的。

**重定向，其实是两次request。**

## TCP拥塞控制

**慢启动：**

发送方每发送一次文段都能正确的**接受确认报文段。**那么就从小到大的增加（**成2倍增加**）拥塞窗口的大小，即增加发送窗口的大小。

注意：需要收到确认报文段，才增加拥塞窗口，发送窗口就增大了

**拥塞算法：**

****慢开始门限****ssthresh

在大于这个门限之后是让cwnd缓慢的增加而不是加倍的增长，每经历过一次往返时间就使cwnd**增加1**，而不是加倍，这样使cwnd缓慢的增长，比慢启动要慢的多

只要判断网络出现拥塞，就要把慢启动**开始门限**(ssthresh)设置为设置为**发送窗口**的一半（>=2），cwnd(拥塞窗口)设置为**1**

**拥塞避免算法：**

当cwnd < ssthresh ，使用慢开始算法；

当cwnd = ssthresh，可以使用慢开始算法，也可以使用拥塞算法；

当cwnd > ssthresh，使用拥塞算法；

**快重传：**

**接收方**一连**收到三个重复**的**ACK**,那么发送方**不必**等待重传**计时器到期**，由发送方尽早重传未被确认的报文段。

**快恢复：**

快重传后没有使用慢启动算法，而是直接进入拥塞避免算法(将慢开始门限值（ssthresh）和cwnd 调整为此时cwnd的一半)