# HTTP（应用层）

## HTTP和HTTPS有什么区别？

* **端口号** ：HTTP 默认是 80，HTTPS 默认是 443。
* **URL 前缀** ：HTTP 的 URL 前缀是 http://，HTTPS 的 URL 前缀是 https://。
* **安全性和资源消耗** ： HTTP 协议运行在 TCP 之上，所有传输的内容都是明文，客户端和服务器端都无法验证对方的身份。HTTPS 是运行在 SSL/TLS 之上的 HTTP 协议，SSL/TLS 运行在 TCP 之上。所有传输的内容都经过加密，加密采用对称加密，但对称加密的密钥用服务器方的证书进行了非对称加密。所以说，HTTP 安全性没有 HTTPS 高，但是 HTTPS 比 HTTP 耗费更多服务器资源。
  + TLS就是从SSL发展而来的，只是SSL发展到3.0版本后改成了TLS
  + 第一次请求中TLS握手的代价很大
  + 后续的请求会共用第一次请求的协商结果

## HTTPS了解吗？说一下加密过程？

对称加密：简单说就是有一个密钥，它可以加密一段信息，也可以对加密后的信息进行解密，和我们日常生活中用的钥匙作用差不多。

什么是非对称加密：简单说就是有两把密钥，通常一把叫做公钥、一把叫私钥，用公钥加密的内容必须用私钥才能解开，同样，私钥加密的内容只有公钥能解开。

1. 双方都用非对称加密：
   1. 双方各自持有公钥和私钥。传输内容前，将公钥互相明文传输。
   2. 非对称加密算法非常耗时。
2. 用对称加密+非对称加密：
   1. 网站拥有公钥私钥。服务器向浏览器明文传输公钥A，浏览器随机(**用数据哈希**)生成一个用于对称加密的密钥X，用A加密后发给服务器，服务器用密钥A‘解密拿到X。这样双方都拿到了X。用X进行加密解密(对称)
   2. 漏洞：中间人攻击（**根本原因是浏览器无法确认收到的公钥是不是网站自己的，**）
3. 引入了CA数字签名+数字证书验证

HTTPS加密过程：

1. 客户端请求服务器获取证书公钥
2. 客户端(SSL/TLS)解析证书（无效会弹出警告）
3. 生成随机值
4. 用公钥加密随机值生成**密钥**
5. 客户端将密钥发送给服务器
6. 服务端用私钥解密秘钥得到随机值
7. 将信息和随机值混合在一起进行对称加密
8. 将加密的内容发送给客户端
9. 客户端用密钥解密信息

<https://zhuanlan.zhihu.com/p/43789231>

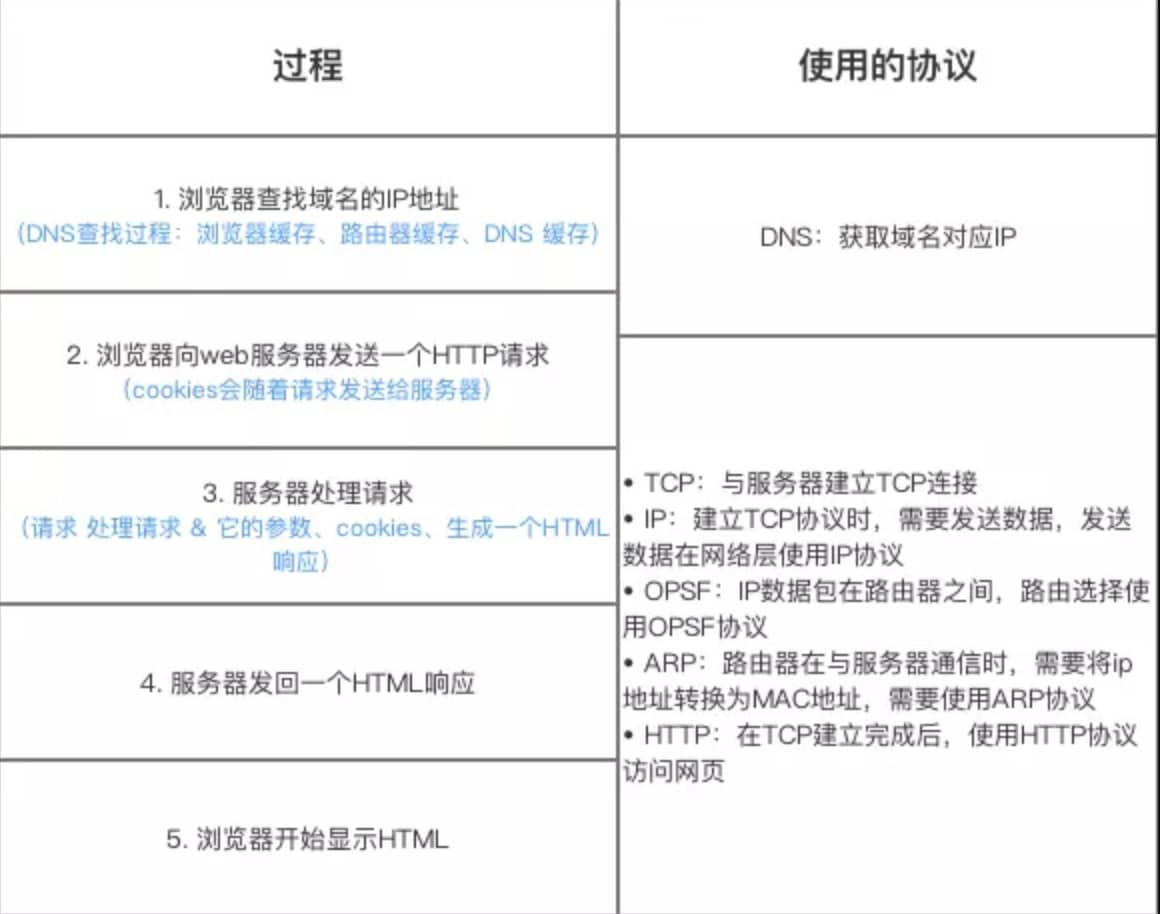
<https://segmentfault.com/a/1190000019976390>

## 从输入URL 到页面展示到底发生了什么？

1. DNS解析

（DNS多级缓存、DNS负载均衡）

1. TCP连接
2. 发送HTTP请求
3. 服务器处理请求并返回HTTP报文
4. 浏览器解析渲染页面
5. 连接结束



<https://segmentfault.com/a/1190000006879700>

## 递归DNS

递归 [DNS](https://www.cloudflare.com/learning/dns/what-is-dns/) 查找是一个 [DNS 服务器](https://www.cloudflare.com/learning/dns/dns-server-types/)与其他几个 DNS 服务器进行通信以搜寻 [IP 地址](https://www.cloudflare.com/learning/dns/glossary/what-is-my-ip-address/)并将其返回给客户端的地方。这与迭代 DNS 查询相反，在迭代 DNS 查询中，客户端直接与查找中涉及的每个 DNS 服务器进行通信。

<https://www.cloudflare.com/zh-cn/learning/dns/what-is-recursive-dns/>

## HTTP状态码



* 301 - Moved **Permanently**
* 401-Unauthorized
* 403-Forbidden
* 404 – Not Found
* 408-Request-Time-out
* 500-Internal-Server-Error

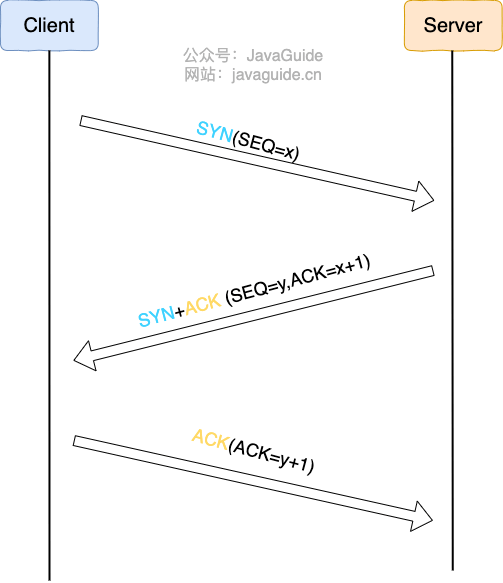
## Post和Get区别

* GET在浏览器回退时是无害的，而POST会再次提交请求。
* GET产生的URL地址可以被Bookmark，而POST不可以。
* GET请求会被浏览器主动cache，而POST不会，除非手动设置。
* GET请求只能进行url编码，而POST支持多种编码方式。
* GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
* GET请求在URL中传送的参数是有长度限制的，而POST没有。
* 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
* GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
* GET参数通过URL传递，POST放在Request body中

注意，POST不是幂等的（产生副作用）

<https://github.com/febobo/web-interview/issues/145>

# TCP三次握手四次挥手（传输层）



## TCP为什么要三次握手？

三次握手的目的是建立可靠的通信信道，说到通讯，简单来说就是数据的发送与接收，而三次握手最主要的目的就是双方确认自己与对方的发送与接收是正常的。

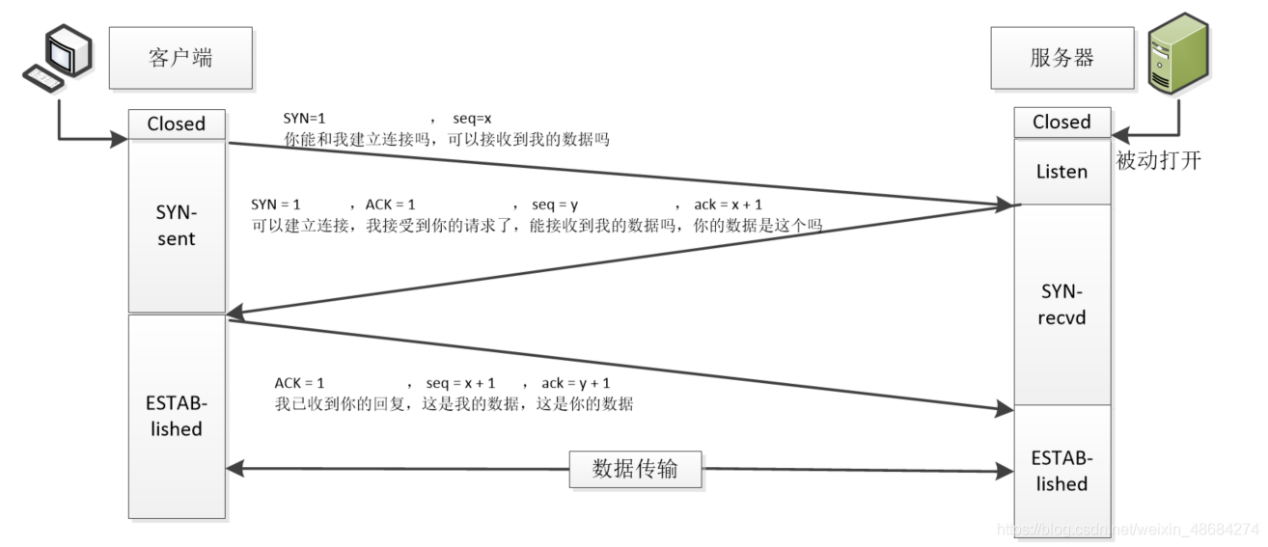
1. **第一次握手** ：Client 什么都不能确认；Server 确认了对方发送正常，自己接收正常
2. **第二次握手** ：Client 确认了：自己发送、接收正常，对方发送、接收正常；Server 确认了：对方发送正常，自己接收正常
3. **第三次握手** ：Client 确认了：自己发送、接收正常，对方发送、接收正常；Server 确认了：自己发送、接收正常，对方发送、接收正常

三次握手就能确认双发收发功能都正常，缺一不可。

## 第2次握手传回了ACK，为什么还要传回SYN？

服务端传回发送端所发送的 ACK 是为了告诉客户端：“我接收到的信息确实就是你所发送的信号了”，这表明从客户端到服务端的通信是正常的。回传 SYN 则是为了建立并确认从服务端到客户端的通信。

## 三次握手中ack和ACK分别表示什么



## 为什么要四次挥手？

TCP是全双工通信，可以双向传输数据。任何一方都可以在数据传送结束后发出连接释放的通知，待对方确认后进入半关闭状态。当另一方也没有数据再发送的时候，则发出连接释放通知，对方确认后就完全关闭了 TCP 连接。

举个例子：A 和 B 打电话，通话即将结束后。

第一次挥手 ： A 说“我没啥要说的了”

第二次挥手 ：B 回答“我知道了”，但是 B 可能还会有要说的话，A 不能要求 B 跟着自己的节奏结束通话

第三次挥手 ：于是 B 可能又巴拉巴拉说了一通，最后 B 说“我说完了”

第四次挥手 ：A 回答“知道了”，这样通话才算结束。

## 为什么不能把服务器发送的 ACK 和 FIN 合并起来，变成三次挥手？

因为服务器收到客户端断开连接的请求时，可能还有一些数据没有发完，这时先回复 ACK，表示接收到了断开连接的请求。等到数据发完之后再发 FIN，断开服务器到客户端的数据传送。

## 如果第二次挥手时服务器的 ACK 没有送达客户端，会怎样？

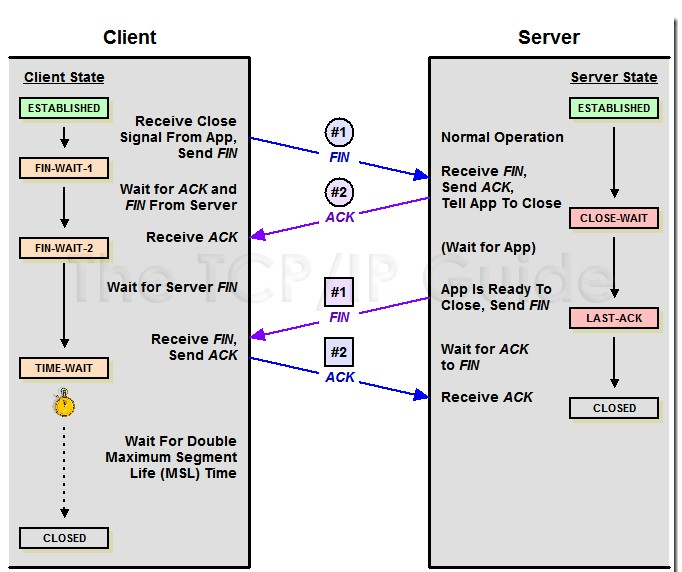
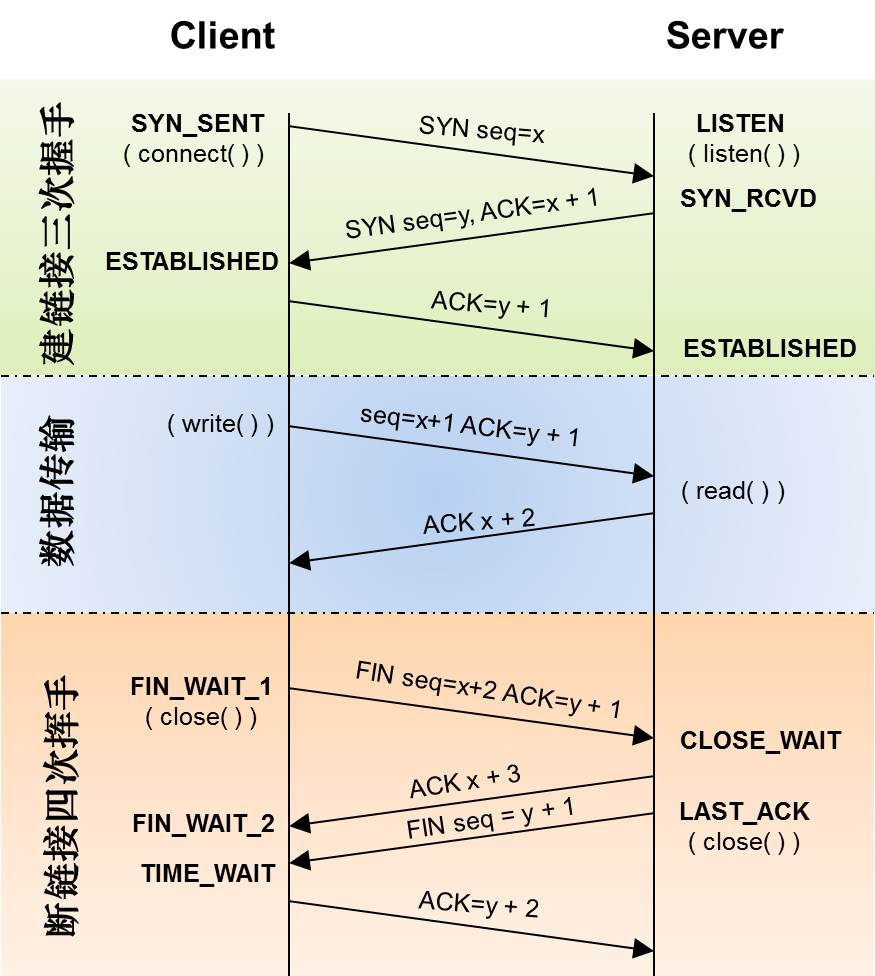
客户端没有收到 ACK 确认，会重新发送 FIN 请求。

## 为什么第四次挥手客户端需要等待 2\*MSL（报文段最长寿命）时间后才进入 CLOSED 状态？

第四次挥手时，客户端发送给服务器的 ACK 有可能丢失，如果服务端没有因为某些原因而没有收到 ACK 的话，服务端就会重发 FIN，如果客户端在 2\*MSL 的时间内收到了 FIN，就会重新发送 ACK 并再次等待 2MSL，防止 Server 没有收到 ACK 而不断重发 FIN。

## 连接一个 IP 不存在的主机时，握手过程是怎样的?

## Tcp建立/断开连接的状态变化



# TCP传输可靠性（传输层）

## TCP 如何保证传输的可靠性？

1. **基于数据快传输** ：应用数据被分割成 TCP 认为最适合发送的数据块，再传输给网络层，数据块被称为报文段或段。
2. **对失序数据包重新排序以及去重**：TCP 为了保证不发生丢包，就给每个包一个序列号，有了序列号能够将接收到的数据根据序列号排序，并且去掉重复序列号的数据就可以实现数据包去重。
3. **校验和** : TCP 将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段。
4. **超时重传** : 当发送方发送数据之后，它启动一个定时器，等待目的端确认收到这个报文段。接收端实体对已成功收到的包发回一个相应的确认信息（ACK）。如果发送端实体在合理的往返时延（RTT）内未收到确认消息，那么对应的数据包就被假设为已丢失并进行重传。
5. **流量控制** : TCP 连接的每一方都有固定大小的缓冲空间，TCP 的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP 使用的流量控制协议是可变大小的滑动窗口协议（TCP 利用滑动窗口实现流量控制）。
6. **拥塞控制** : 当网络拥塞时，减少数据的发送

## TCP 如何实现流量控制？

**TCP 利用滑动窗口实现流量控制。流量控制是为了控制发送方发送速率，保证接收方来得及接收。** 接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为 0，则发送方不能发送数据。

**为什么需要流量控制?** 这是因为双方在通信的时候，发送方的速率与接收方的速率是不一定相等，如果发送方的发送速率太快，会导致接收方处理不过来。如果接收方处理不过来的话，就只能把处理不过来的数据存在 **接收缓冲区(Receiving Buffers)** 里（失序的数据包也会被存放在缓存区里）。如果缓存区满了发送方还在狂发数据的话，接收方只能把收到的数据包丢掉。出现丢包问题的同时又疯狂浪费着珍贵的网络资源。因此，我们需要控制发送方的发送速率，让接收方与发送方处于一种动态平衡才好。

这里需要注意的是（常见误区）：

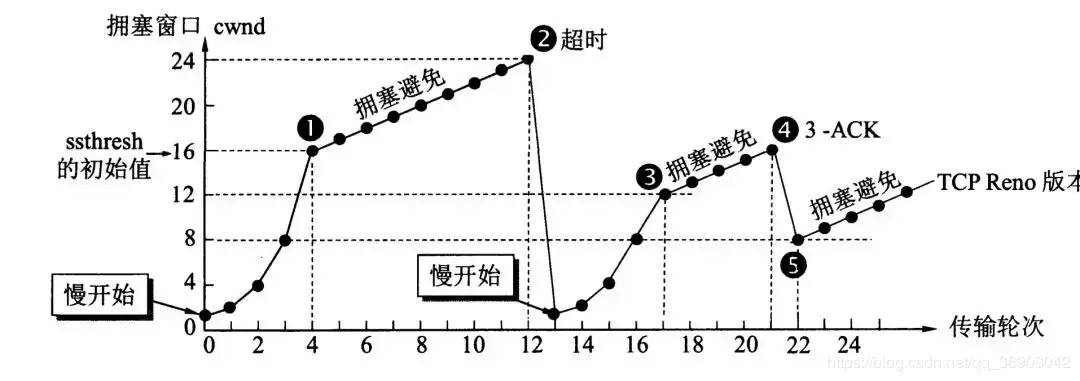
* 发送端不等同于客户端
* 接收端不等同于服务端

TCP 为全双工(Full-Duplex, FDX)通信，双方可以进行双向通信，客户端和服务端既可能是发送端又可能是服务端。因此，两端各有一个发送缓冲区与接收缓冲区，两端都各自维护一个发送窗口和一个接收窗口。接收窗口大小取决于应用、系统、硬件的限制（TCP传输速率不能大于应用的数据处理速率）。通信双方的发送窗口和接收窗口的要求相同

[窗口详见JavaGuide](https://javaguide.cn/cs-basics/network/tcp-reliability-guarantee.html#tcp-%E5%A6%82%E4%BD%95%E5%AE%9E%E7%8E%B0%E6%B5%81%E9%87%8F%E6%8E%A7%E5%88%B6)

## TCP 的拥塞控制是怎么实现的？

在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏。这种情况就叫拥塞。拥塞控制就是为了防止过多的数据注入到网络中，这样就可以使网络中的路由器或链路不致过载。拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。拥塞控制是一个全局性的过程，涉及到所有的主机，所有的路由器，以及与降低网络传输性能有关的所有因素。相反，流量控制往往是点对点通信量的控制，是个端到端的问题。流量控制所要做到的就是抑制发送端发送数据的速率，以便使接收端来得及接收。



为了进行拥塞控制，TCP 发送方要维持一个 **拥塞窗口(cwnd)** 的状态变量。拥塞控制窗口的大小取决于网络的拥塞程度，并且动态变化。发送方让自己的发送窗口取为拥塞窗口和接收方的接受窗口中较小的一个。

TCP 的拥塞控制采用了四种算法，即 **慢开始** 、 **拥塞避免** 、**快重传** 和 **快恢复**。在网络层也可以使路由器采用适当的分组丢弃策略（如主动队列管理 AQM），以减少网络拥塞的发生。

* **慢开始：** 慢开始算法的思路是当主机开始发送数据时，如果立即把大量数据字节注入到网络，那么可能会引起网络阻塞，因为现在还不知道网络的符合情况。经验表明，较好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是由小到大逐渐增大拥塞窗口数值。cwnd 初始值为 1，每经过一个传播轮次，cwnd 加倍。
* **拥塞避免：** 拥塞避免算法的思路是让拥塞窗口 cwnd 缓慢增大，即每经过一个往返时间 RTT 就把发送放的 cwnd 加 1.
* **快重传与快恢复：** 在 TCP/IP 中，快速重传和恢复（fast retransmit and recovery，FRR）是一种拥塞控制算法，它能快速恢复丢失的数据包。没有 FRR，如果数据包丢失了，TCP 将会使用定时器来要求传输暂停。在暂停的这段时间内，没有新的或复制的数据包被发送。有了 FRR，如果接收机接收到一个不按顺序的数据段，它会立即给发送机发送一个重复确认。如果发送机接收到三个重复确认，它会假定确认件指出的数据段丢失了，并立即重传这些丢失的数据段。有了 FRR，就不会因为重传时要求的暂停被耽误。 　当有单独的数据包丢失时，快速重传和恢复（FRR）能最有效地工作。当有多个数据信息包在某一段很短的时间内丢失时，它则不能很有效地工作

## Tcp一定比udp慢吗？

<https://www.51cto.com/article/720681.html>

# 路由转发的流程（网络层）

1. 路由就是一跳一跳问(路由表)路。一跳，就是数据链路层的一个区间。再具体，在以太网中，从源MAC地址到目的MAC地址的帧传输区间。
2. 备注：IP包(==ip分组)在上一层(传输层)被切割成数据包
3. 流程：
   1. 每个主机或路由器接口有路由表
   2. 源主机A向目的主机B发送报文，首先**确定主机B是否在本网络上**（by子网掩码and运算）。If true，直接交付；else，根据路由表转发。
   3. 分组每到达一个路由器，路由器就**根据分组的终点查找路由表**（找到下一跳的IP地址，然后把分组送到数据链路层的接口，接口调用ARP解析出下一跳的MAC地址），把目的MAC地址写入MAC帧的首部，把MAC帧传送给下一跳路由器的数据链路层，该路由器取出MAC帧的数据部分，交给网络层。
   4. 循环步骤3，直到找到目的主机所在的网络，最后就在这个网络上把分组**直接交付给目的主机**。

## ping www.baidu.com 发生了什么

不考虑域名解析、ARP 广播、路由转发、各种校验的**简化版**：  
ping 192.168.0.4　　---->　　构建 ICMP 包　　---->　　构建 IP 分组　　---->　　解析硬件地址封装成帧　　----> 　 　物理层发送

**完整流程**如下：

1. 如果是 ping 域名，首先需要通过 **DNS 服务**拿到对方的 IP。
2. 主机构建一个 **ICMP 格式的数据包**，由 ICMP 协议将这个数据包（有效负荷）连同目的 IP 地址一起交给 IP 协议。
3. IP 协议构建一个**分组（本机 IP + 控制信息 + 目的 IP）**，控制信息至少包含了 01h 的协议字段，这样当此分组到达目的方时，这些内容就会告诉目的主机应该将这个有效负荷交付给哪个协议来处理，本例中就是 ICMP。
4. 如果目的 IP 不跟自己在同一个网段，则需要通过路由转发，先通过路由信息得到**网关 IP**。如果在同一网段则没有此步骤。
5. 如果之前与目的 IP （或网关 IP）通信过，应该在 ARP 缓存表中能查到对应的 **MAC 地址**。如果没有，就需要通过 ARP 广播来获取 MAC 地址了。把 IP 地址解析为 MAC 地址后，此分组就可被传送到数据链路层以组建成帧。
6. 数据链路层将控制信息封装到此分组上，**创建成帧**。在这个帧中，附加有目的 MAC 地址和源 MAC 地址，以及以太网类型字段（用来指明应用于帧数据字段的协议）。在帧的尾部是 FCS（帧校验序列）字段，这个部分装载了 CRC（循环冗余校验）的计算结果。
7. 把帧交付给物理层，**物理层**会以一次一比特的方式将帧**发送**到物理介质。
8. 这时，此冲突域中的每台设备都会接收这些比特，并将它重新组建成帧。每个设备都会对接收到的内容进行 CRC 运算，并与帧中 FCS 字段的内容进行比对。如果值不匹配，接收到的帧将被丢弃。如果匹配，接着将**检查目的 MAC 地址与自己的是否匹配**。如果匹配，则接下来查看以太网类型字段，以获悉完成数据后续处理的网络层协议。
9. 将分组从帧中取出，并将其他部分丢弃。然后，分组被**递交给**以太网类型字段中列出的协议（本例是 **IP 协议**）。
10. IP 协议接收这个分组，并检查它的 IP 的地址。如果这是一个路由器（网关），跳到 11。如果这就是目的主机，跳到下一步 12。
11. 由于分组的目的 IP 地址与此接收路由器上的各个地址均不匹配，路由器会在其**路由表中查找目的 IP 的地址**，（在此路由表中需要包含目的子网的相关表项，否则路由器会立即将收到的分组丢弃，并同时向发送方回送一个携带有目标网络不可达信息的 ICMP 报文。），并按照路由规则被交换到指定的输出接口的缓冲区内。接着重复 5 ~ 11 步骤。
12. 分组的目的 IP 与此主机 IP 地址匹配，检查分组的协议字段，了解分组有效负荷的交付对象。此有效负荷将被递交给 ICMP，后者知道这是一个回应请求数据。ICMP 将负责应答这个请求，它首先立即丢弃这个接收到的分组然后**产生**一个新的有效负荷作为回应**应答数据**。

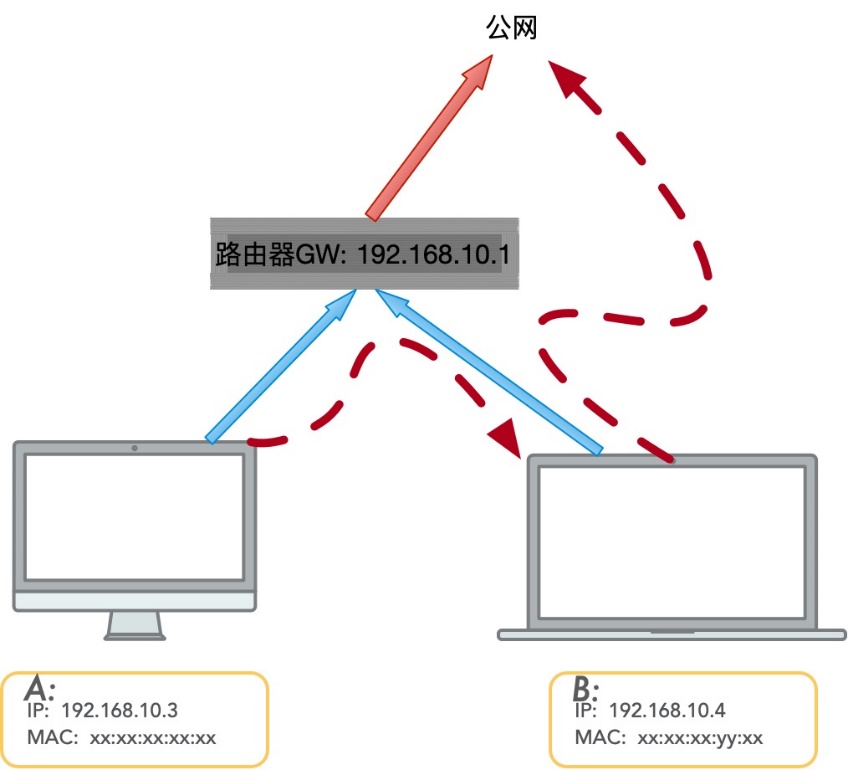
<https://www.jianshu.com/p/52b1c9ba82a1>

## ARP工作原理（网络层/链路层）

1. **ARP 协议在协议栈中的位置？** ARP 协议在协议栈中的位置非常重要，在理解了它的工作原理之后，也很难说它到底是网络层协议，还是链路层协议，因为它恰恰串联起了网络层和链路层。国外的大部分教程通常将 ARP 协议放在网络层。
2. **ARP 协议解决了什么问题，地位如何？** ARP 协议，全称 **地址解析协议（Address Resolution Protocol）**，它解决的是网络层地址和链路层地址之间的转换问题。因为一个 IP 数据报在物理上传输的过程中，总是需要知道下一跳（物理上的下一个目的地）该去往何处，但 IP 地址属于逻辑地址，而 MAC 地址才是物理地址，ARP 协议解决了 IP 地址转 MAC 地址的一些问题。
3. **ARP 工作原理？** 只希望大家记住几个关键词：**ARP 表、广播问询、单播响应**。

Ps. 在[IPv6](https://zh.wikipedia.org/wiki/IPv6)中[邻居发现协议](https://zh.wikipedia.org/wiki/%E9%82%BB%E5%B1%85%E5%8F%91%E7%8E%B0%E5%8D%8F%E8%AE%AE)（NDP）用于代替地址解析协议（ARP）。

## ARP欺骗



* 在A需要往外发送icmp的一个回显请求报文的时候，主机B发起ARP攻击，重复得发送一个ARP回应包，包的内容为mac:xx:xx:xx:yy:xx ip:192.168.10.1，以此来伪装自己就是网关，让A将数据发给自己
* 于此同时，主机B也通过回应报文的形式告诉网关也就是路由器，称自己是主机A，具体的报文内容为max: xx:xx:xx:yy:xx ip: 192.168.10.3，这样主机B就达到了数据包嗅探的能力
* 主机A先将自己的数据包发给B，因为它误以为B是网关，B再将数据转发给路由器，这个时候路由器接收了数据之后将数据转发到公网
* 这里主机B是完全可以侦听A的数据包的，甚至去篡改A的数据包

## MAC地址

1. 与IP地址区别：

Mac地址是身份证号（出生即分配，唯一的，永远不变）

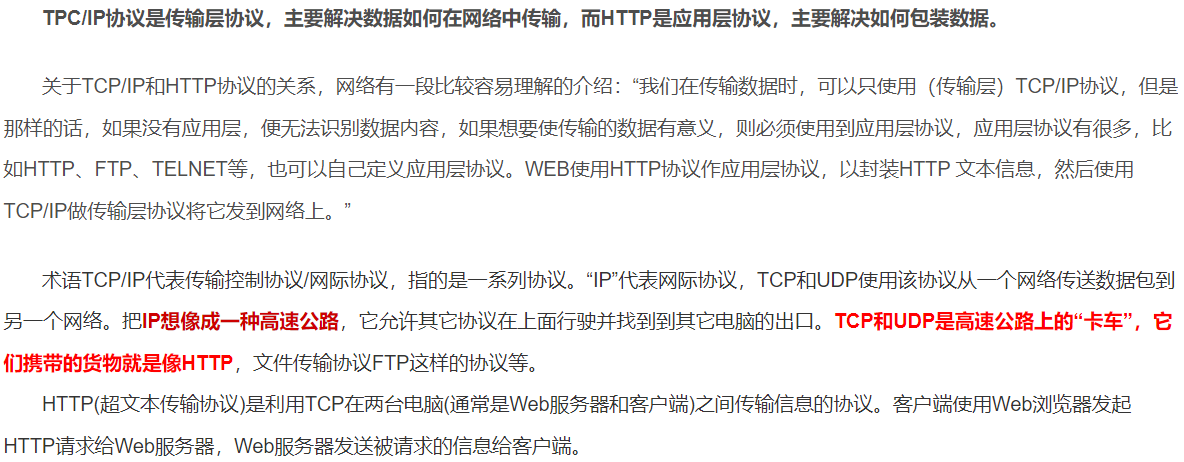
Ip地址是邮政地址（唯一的，随更换网络而改变）

1. MAC 地址有一个特殊地址：FF-FF-FF-FF-FF-FF（全 1 地址），该地址表示广播地址。
2. 注意：一般情况下，我们说网络设备都有一个 IP 地址和一个 MAC 地址，这里说的网络设备，更严谨的说法应该是一个接口。路由器作为互联设备，具有多个接口，每个接口同样也应该具备不重复的 IP 地址和 MAC 地址。因此，在讨论 ARP 表时，路由器的多个接口都个各自维护一个 ARP 表，而非一个路由器只维护一个 ARP 表。

# socket

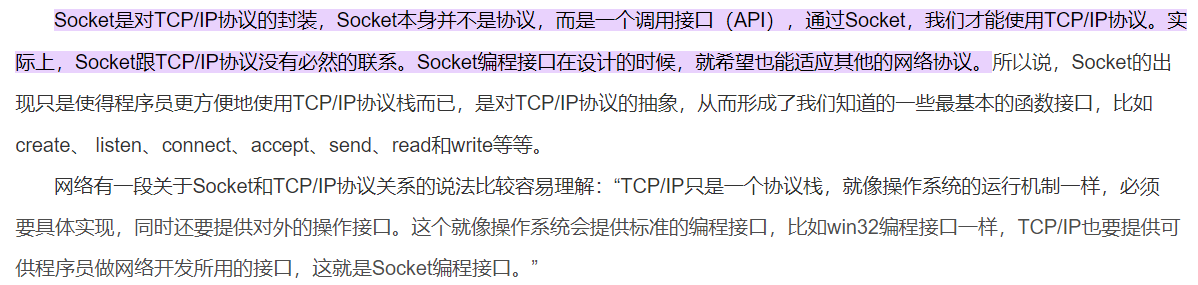
## tcp/ip、http、socket的关系

1. 前两者是协议。传输层与应用层



1. Socket是插头、api、对协议的封装、拿来即用。

是由操作系统提供的进程间通信机制



## 5种IO模型

1. 阻塞与非阻塞



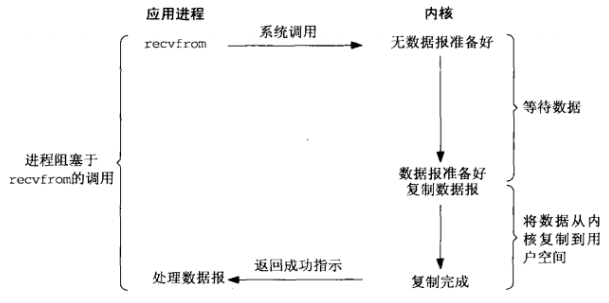
同步与异步



1. IO模型

2.1 阻塞式 IO 模型

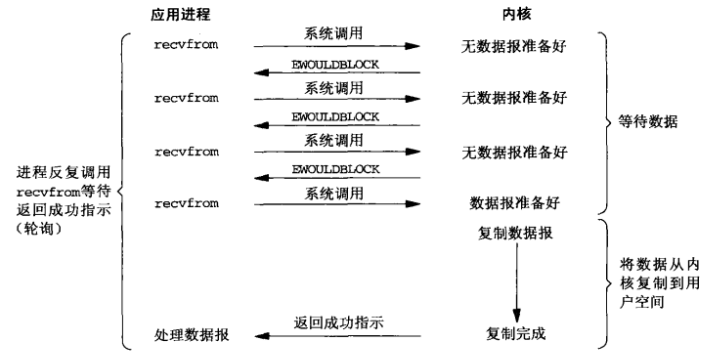
阻塞式 IO （Blocking IO）：应用进程从发起 IO 系统调用，至内核返回成功标识，这整个期间是处于阻塞状态的。



2.2 非阻塞式 IO 模型

非阻塞式IO（Non-Blocking IO）：应用进程可以将 Socket 设置为非阻塞，这样应用进程在发起 IO 系统调用后，会立刻返回。应用进程可以轮询的发起 IO 系统调用，直到内核返回成功标识。

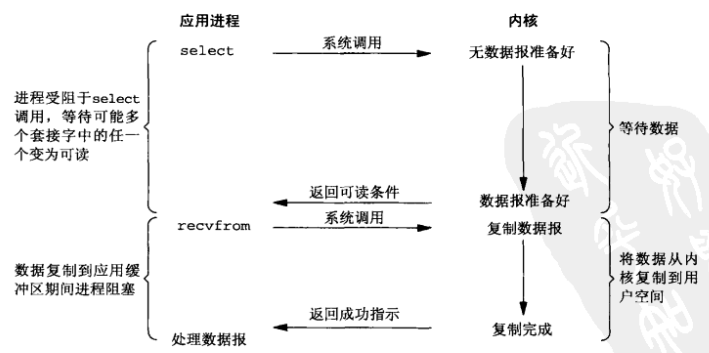
\*io工作任务发起调用后，cpu开始做了，（有可能一会儿才开始~做完了也可以做别的）这个任务时不时回来问“做完了没有？做完了没有？做完了没有？做完了没有？”



2.3 IO 多路复用模型

IO 多路复用（IO Multiplexin）：可以将多个应用进程的 Socket 注册到一个 Select（多路复用器）上，然后使用一个进程来监听该 Select（该操作会阻塞），Select 会监听所有注册进来的 Socket。只要有一个 Socket 的数据准备好，就会返回该Socket。再由应用进程发起 IO 系统调用，来完成数据读取。

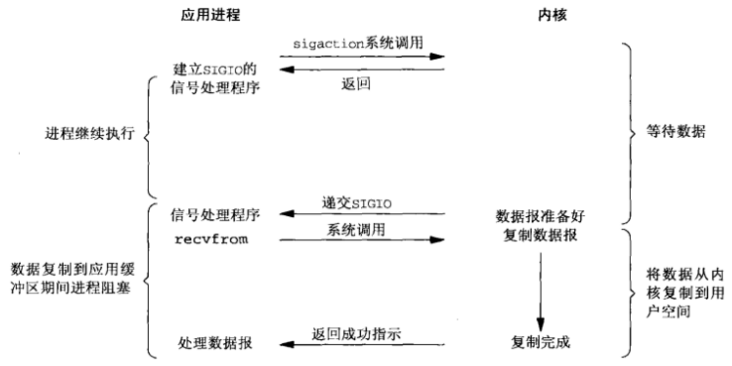
\* Select有点像生产者-消费者里那个blockingqueue，但是size=1，且阻塞



2.4 信号驱动 IO 模型

信号驱动 IO（Signal Driven IO）：可以为 Socket 开启信号驱动 IO 功能，应用进程需向内核注册一个信号处理程序，该操作并立即返回。当内核中有数据准备好，会发送一个信号给应用进程，应用进程便可以在信号处理程序中发起 IO 系统调用，来完成数据读取了。

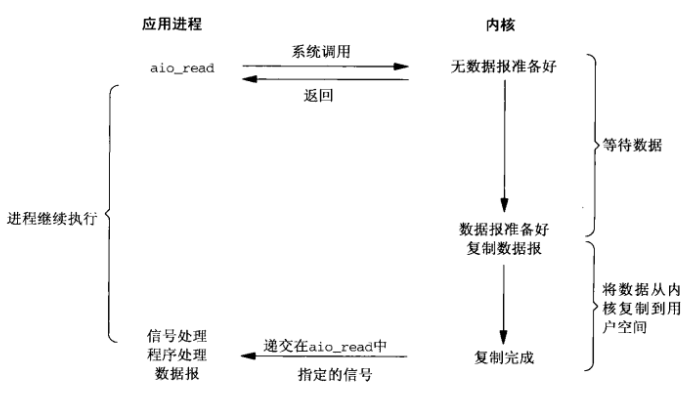
\*内核有一个标识信号“已注册待做任务”，if（flag）



2.5 异步 IO 模型

异步 IO（Asynchronous IO）： 应用进程发起 IO 系统调用后，会立即返回。当内核中数据完全准备后，并且也复制到了用户空间，会产生一个信号来通知应用进程。

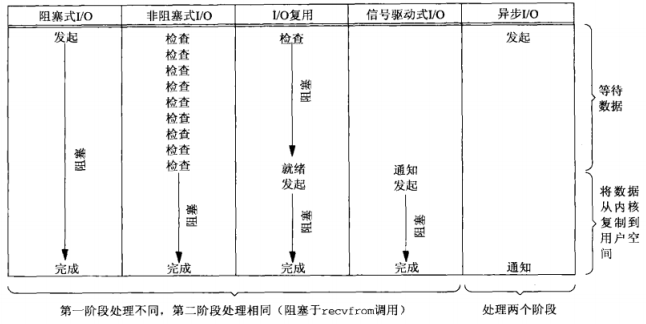
\*完全地异步——任务一到，就交给cpu去做了，需要的时候，才来问问“做完了没有？”



3. 总结

从上述五种 IO 模型可以看出，应用进程对内核发起 IO 系统调用后，内核会经过两个阶段来完成数据的传输：

* 第一阶段：等待数据。即应用进程发起 IO 系统调用后，会一直等待数据；当有数据传入服务器，会将数据放入内核空间，此时数据准备好。
* 第二阶段：将数据从内核空间复制到用户空间，并返回给应用程序成功标识。



前四种模型的第二阶段是相同的，都是处于阻塞状态，其主要区别在第一阶段。而异步 IO 模型则不同，应用进程在这两个阶段是完全不阻塞的。