**https://learnku.com/articles/59484**

1. **HTTP 常用的请求方式，区别和用途？**

**GET**: 发送请求，获取服务器数据

**POST**：向 URL 指定的资源提交数据

**PUT**：向服务器提交数据，以修改数据

HEAD: 请求页面的首部，获取资源的元信息

**DELETE**：删除服务器上的某些资源。

**CONNECT**：建立连接隧道，用于代理服务器；

OPTIONS：列出可对资源实行的请求方法，常用于跨域

**TRACE**：追踪请求 - 响应的传输路径

1. **HTTP 常用的状态码及含义？**

1xx：接受的请求正在处理 （信息性状态码）

2xx：表示请求正常处理完毕 （成功状态码）

3xx：表示重定向状态，需要重新请求 （重定向状态码）

4xx：服务器无法处理请求 （客户端错误状态码）

5xx：服务器处理请求出错 （服务端错误状态码）

**常用状态码如下：**

101 切换请求协议，从 HTTP 切换到 WebSocket

200 请求成功，表示正常返回信息。

301 永久重定向，会缓存

302 临时重定向，不会缓存

400 请求错误

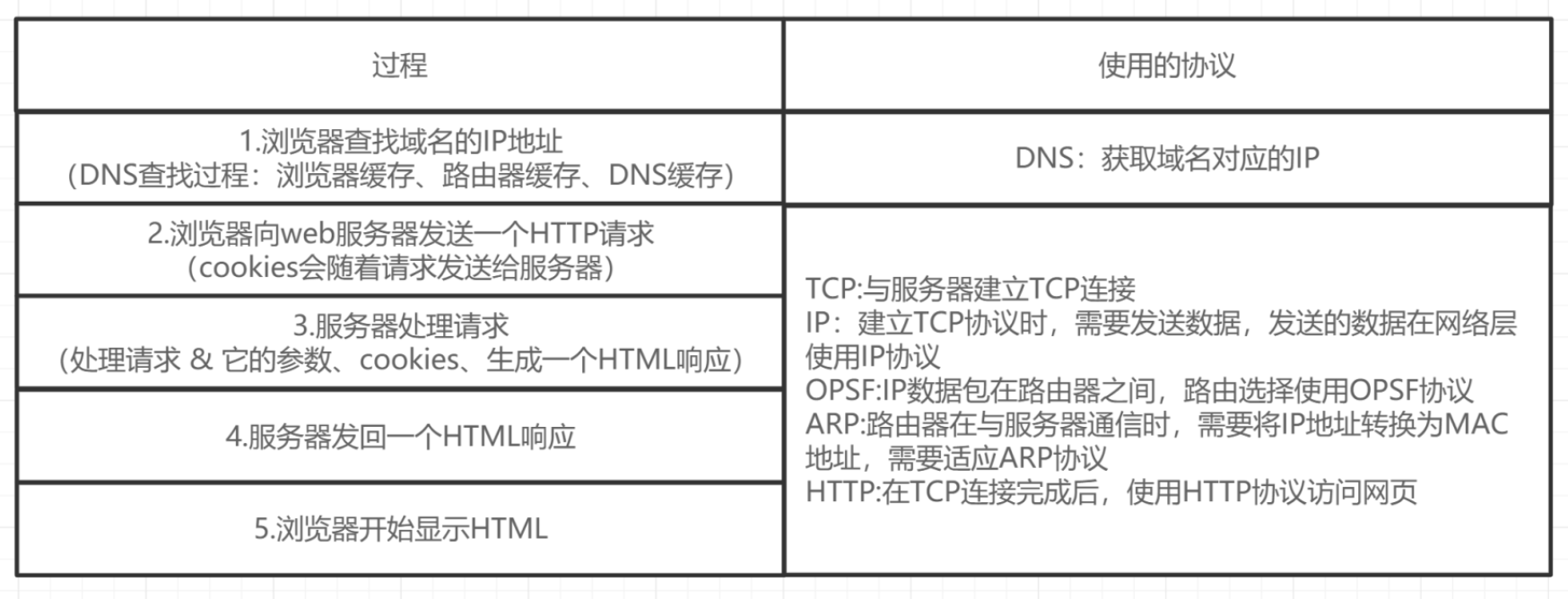
403 服务器禁止访问

404 找不到与 URI 相匹配的资源。

500 常见的服务器端错误

1. **从浏览器地址栏输入 url 到显示主页的过程**

* DNS 解析，查找真正的 ip 地址
* 与服务器建立 TCP 连接
* 发送 HTTP 请求
* 服务器处理请求并返回 HTTP 报文
* 浏览器解析渲染页面
* 连接结束



1. **如何理解 HTTP 协议是无状态的**

* **HTTP 是一种不保存状态，即无状态（stateless）协议**。也就是说 HTTP 协议自身不对请求和响应之间的通信状态进行保存。那么我们保存用户状态呢？Session 机制的存在就是为了解决这个问题**，Session 的主要作用就是通过服务端记录用户的状态**。典型的场景是**购物车**，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。**服务端给特定的用户创建特定的 Session 之后就可以标识这个用户并且跟踪这个用户了**（**一般情况下，服务器会在一定时间内保存这个 Session，过了时间限制，就会销毁这个 Session**）。
* 在服务端保存 Session 的方法很多，最常用的就是**内存和数据库(比如是使用内存数据库 redis 保存)**。既然 Session 存放在服务器端，那么我们如何实现 Session 跟踪呢？大部分情况下，我们都是通过在 **Cookie 中附加一个 Session ID 来方式来跟踪**。

**Cookie 被禁用怎么办?**

**最常用的就是利用 URL 重写把 Session ID 直接附加在 URL 路径的后面。**

每次 HTTP 请求都是独立的，无相关的，默认不需要保存上下文信息的。我们来看个便于理解的例子：

**有状态：**

A：今天吃啥子？

B：罗非鱼！

A：味道怎么样呀？

B：还不错，好香。

**无状态：**

A：今天吃啥子？

B：罗非鱼！

A：味道怎么样呀？

B：？啊？啥？什么鬼？什么味道怎么样？

**加下 cookie 这玩意：**

A：今天吃啥子？

B：罗非鱼

A：你今天吃的罗非鱼味道怎么样呀？

B：还不错，好香。

1. **HTTP 1.0，1.1，2.0 的版本区别**

**HTTP 1.0**

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。它也可以强制开启长链接，例如设置 Connection: keep-alive 这个字段

**HTTP 1.1**

引入了长连接，即 TCP 连接默认不关闭，可以被多个请求复用。

引入了管道机制（pipelining），即在同一个 TCP 连接里面，客户端可以同时发送多个请求。

缓存处理，引入了更多的缓存控制策略，如 Cache-Control、Etag/If-None-Match 等。

错误状态管理，新增了 24 个错误状态响应码，如 409 表示请求的资源与资源的当前状态发生冲突。

**HTTP 2**

采用了多路复用，即在一个连接里，客户端和浏览器都可以同时发送多个请求或回应，而且不用按照顺序一一对应。

服务端推送，HTTP 2 允许服务器未经请求，主动向客户端发送资源

1. **说下计算机网络体系结构**

计算机网路体系结构主要有 **ISO 七层模型、TCP/IP 四层模型、五层体系结构**

**ISO 七层模型**

ISO 七层模型是国际标准化组织（ISO）制定的一个用于计算机或通信系统间互联的标准体系。

应用层：网络服务与最终用户的一个接口，协议有：HTTP FTP TFTP SMTP SNMP DNS TELNET HTTPS POP3 DHCP

表示层：数据的表示、安全、压缩。

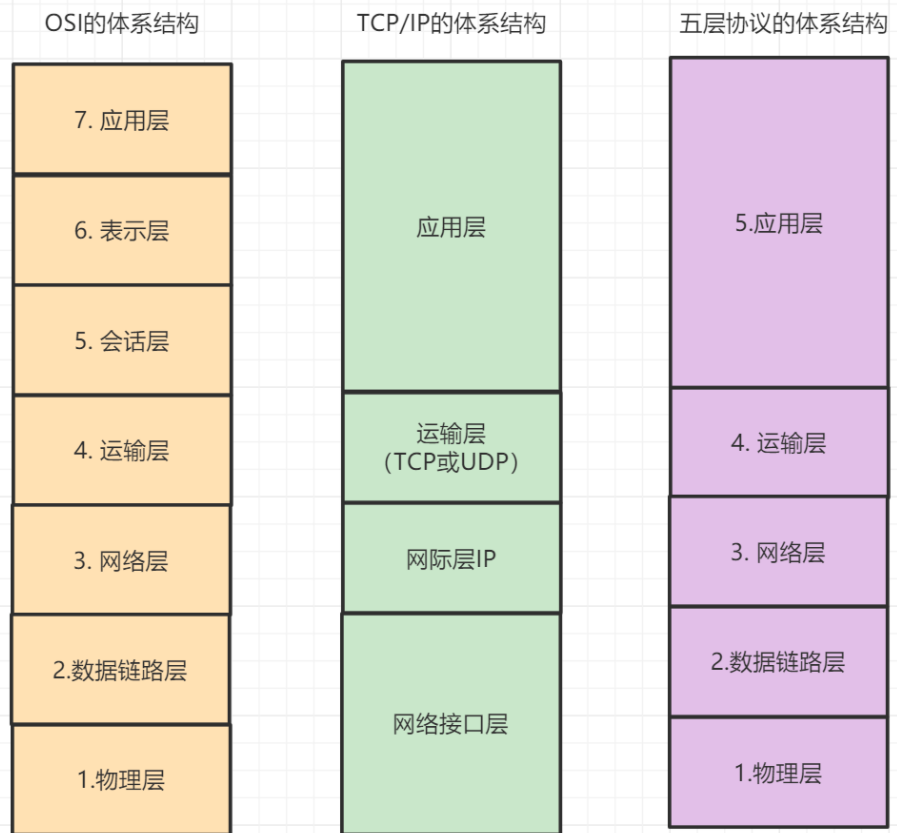
会话层：建立、管理、终止会话。对应主机进程，指本地主机与远程主机正在进行的会话

传输层：定义传输数据的协议端口号，以及流控和差错校验。协议有：TCP UDP，数据包一旦离开网卡即进入网络传输层

网络层：进行逻辑地址寻址，实现不同网络之间的路径选择。协议有：ICMP IGMP IP（IPV4 IPV6）

数据链路层：建立逻辑连接、进行硬件地址寻址、差错校验等功能。

物理层：建立、维护、断开物理连接。



**TCP/IP 四层模型**

应用层：对应于 OSI 参考模型的（应用层、表示层、会话层），为用户提供所需要的各种服务，例如：FTP、Telnet、DNS、SMTP 等

传输层：对应 OSI 的传输层，为应用层实体提供端到端的通信功能，保证了数据包的顺序传送及数据的完整性。定义了 TCP 和 UDP 两层协议。

网际层：对应于 OSI 参考模型的网络层，主要解决主机到主机的通信问题。三个主要协议：网际协议（IP）、互联网组管理协议（IGMP）和互联网控制报文协议（ICMP）

网络接口层：与 OSI 参考模型的数据链路层、物理层对应。它负责监视数据在主机和网络之间的交换。

**TCP/IP五层模型：应用层、传输层、网络层、数据链路层、物理层**。

* **应用层**：为应用程序提供交互服务。在互联网中的应用层协议很多，如域名系统DNS、HTTP协议、SMTP协议等。
* **传输层**：负责向两台主机进程之间的通信提供数据传输服务。传输层的协议主要有传输控制协议TCP和用户数据协议UDP。
* **网络层**：选择合适的路由和交换结点，确保数据及时传送。主要包括IP协议。
* **数据链路层**：在两个相邻节点之间传送数据时，**数据链路层将网络层交下来的 IP 数据报组装成帧**，在两个相邻节点间的链路上传送帧。
* **物理层**：实现相邻节点间比特流的透明传输，尽可能屏蔽传输介质和物理设备的差异。

1. **POST 和 GET 有哪些区别？**

**请求参数**：GET 把参数包含在 URL 中，用 & 连接起来；POST 通过 request body 传递参数。

**请求缓存**：GET 请求会被主动 Cache，而 POST 请求不会，除非手动设置。

**收藏为书签**：GET 请求支持收藏为书签，POST 请求不支持。

**安全性**：POST 比 GET 安全，GET 请求在浏览器回退时是无害的，而 POST 会再次请求。

**历史记录**：GET 请求参数会被完整保留在浏览历史记录里，而 POST 中的参数不会被保留。

**编码方式**：GET 请求只能进行 url 编码，而 POST 支持多种编码方式。

**参数数据类型**：GET 只接受 ASCII 字符，而 POST 没有限制数据类型。

**数据包**: GET 产生一个 TCP 数据包；POST 可能产生两个 TCP 数据包。

1. **在交互过程中如果数据传送完了，还不想断开连接怎么办，怎么维持？**

在 HTTP 中响应体的 Connection 字段指定为 keep-alive

1. **HTTP 如何实现长连接？在什么时候会超时？**

**HTTP 如何实现长连接？**

HTTP 分为长连接和短连接，其实本质上说的是 TCP 的长短连接。TCP 连接是一个双向的通道，它是可以保持一段时间不关闭的，因此 TCP 连接才有真正的长连接和短连接这一个说法。

长连接是指的是 TCP 连接，而不是 HTTP 连接。

**TCP 长连接可以复用一个 TCP 连接来发起多次 HTTP 请求**，这样可以减少资源消耗，比如一次请求 HTML，短连接可能还需要请求后续的 JS/CSS/ 图片等

要实现 HTTP 长连接，在响应头设置 Connection 为 keep-alive，HTTP1.1 默认是长连接，而 HTTP 1.0 协议也支持长连接，但是默认是关闭的。

**在什么时候会超时呢？**

HTTP 一般会有 httpd 守护进程，里面可以设置 keep-alive timeout，当 tcp 链接闲置超过这个时间就会关闭，也可以在 HTTP 的 header 里面设置超时时间

TCP 的 keep-alive 包含三个参数，支持在系统内核的 net.ipv4 里面设置：当 TCP 连接之后，闲置了 tcp\_keepalive\_time，则会发生侦测包，如果没有收到对方的 ACK，那么会每隔 tcp\_keepalive\_intvl 再发一次，直到发送了 tcp\_keepalive\_probes，就会丢弃该连接。

tcp\_keepalive\_intvl = 15

tcp\_keepalive\_probes = 5

tcp\_keepalive\_time = 1800

1. **讲一下 HTTP 与 HTTPS 的区别。**

HTTP，超文本传输协议，英文是 Hyper Text Transfer Protocol，是一个基于 TCP/IP 通信协议来传递数据的协议。

HTTP 存在这几个问题：

* **请求信息明文传输，容易被窃听截取。**
* **数据的完整性未校验，容易被篡改**
* **没有验证对方身份，存在冒充危险**

HTTPS 就是为了解决 HTTP 存在问题的。HTTPS，英文是 HyperText Transfer Protocol over Secure Socket Layer，可以这么理解 Https 是身披 SSL (Secure Socket Layer) 的 HTTP，即 HTTPS 协议 = HTTP+SSL/TLS。**通过 SSL 证书来验证服务器的身份，并为浏览器和服务器之间的传输数据进行加密**。

**它们主要区别：**

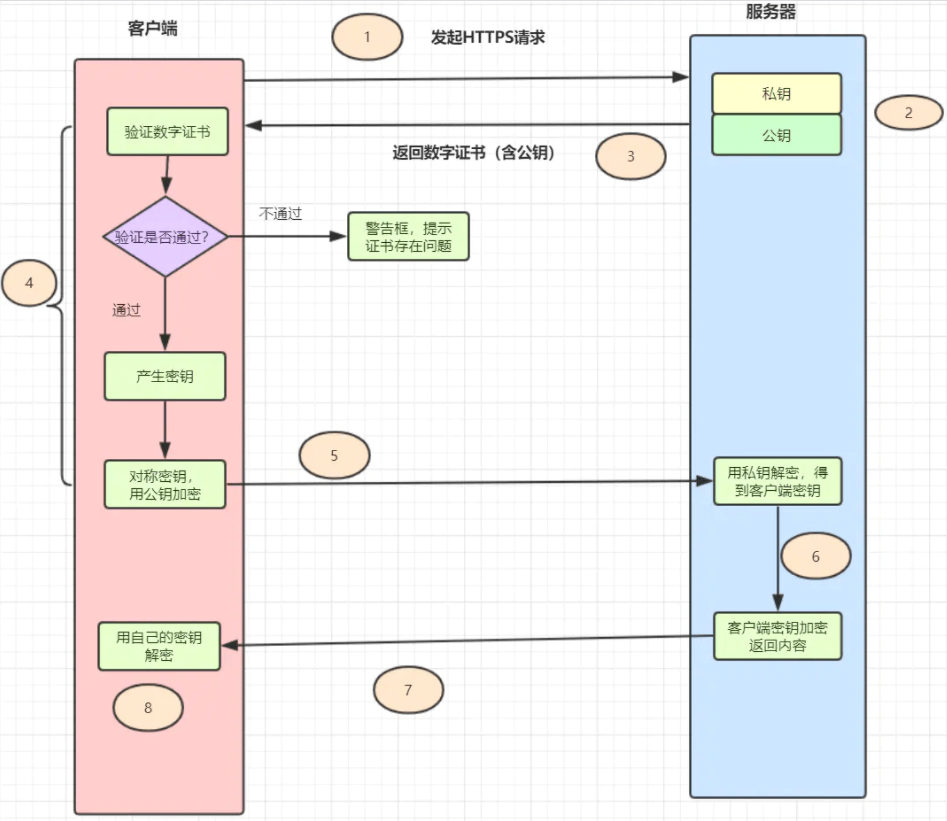
* **数据是否加密**: Http 是明文传输，HTTPS 是密文
* **默认端口:** Http 默认端口是 80，Https 默认端口是 443
* **资源消耗**：和 HTTP 通信相比，Https 通信会消耗更多的 CPU 和内存资源，因为需要加解密处理；
* **安全性**: http 不安全，https 比较安全。

1. **Https 流程是怎样的？**

**HTTPS = HTTP + SSL/TLS**，即用 SSL/TLS 对数据进行加密和解密，Http 进行传输。

**SSL**，即 Secure Sockets Layer（**安全套接层协议**），是网络通信提供安全及数据完整性的一种安全协议。

**TLS**，即 Transport Layer Security (**安全传输层协议**)，它是 SSL 3.0 的后续版本。



1用户在浏览器里输入一个 https 网址，然后连接到 server 的 443 端口。

2服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过。这套证书其实就是一对公钥和私钥。

3服务器将自己的数字证书（含有公钥）发送给客户端。

4客户端收到服务器端的数字证书之后，会对其进行检查，如果不通过，则弹出警告框。如果证书没问题，则生成一个密钥（对称加密），用证书的公钥对它加密。

5客户端会发起 HTTPS 中的第二个 HTTP 请求，将加密之后的客户端密钥发送给服务器。

6服务器接收到客户端发来的密文之后，会用自己的私钥对其进行非对称解密，解密之后得到客户端密钥，然后用客户端密钥对返回数据进行对称加密，这样数据就变成了密文。

7服务器将加密后的密文返回给客户端。

8客户端收到服务器发返回的密文，用自己的密钥（客户端密钥）对其进行对称解密，得到服务器返回的数据。

1. **对称加密与非对称加密有什么区别**

**对称加密**：加密和解密使用相同密钥的加密算法。

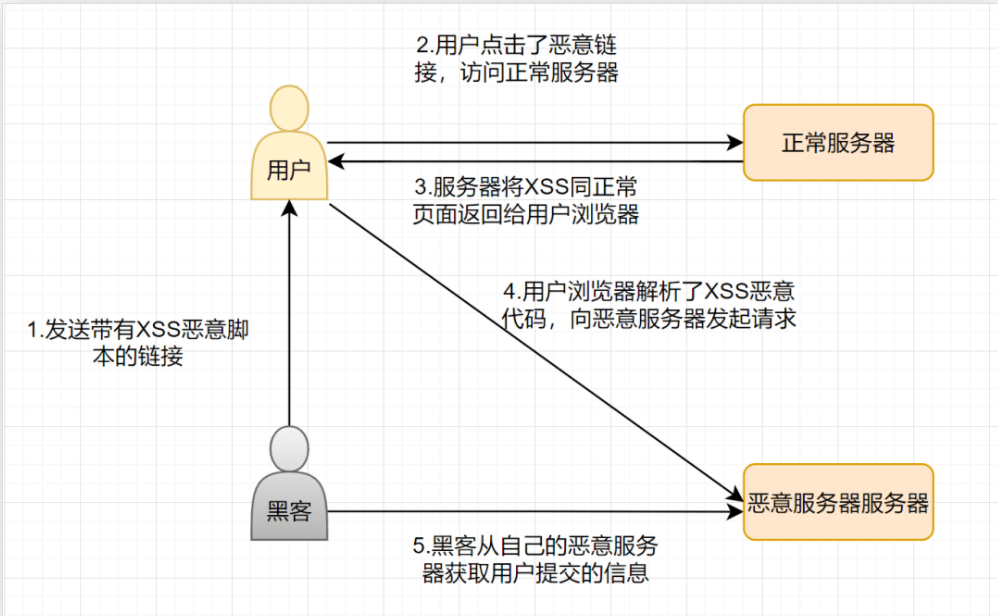
**非对称加密**：非对称加密算法需要两个密钥（公开密钥和私有密钥）。公钥与私钥是成对存在的，如果用公钥对数据进行加密，只有对应的私钥才能解密。

1. **什么是 XSS 攻击，如何避免？**

**XSS 攻击**，全称**跨站脚本攻击（Cross-Site Scripting）**，这会与层叠样式表 (Cascading Style Sheets, CSS) 的缩写混淆，因此有人将跨站脚本攻击缩写为 XSS。它指的是恶意攻击者往 Web 页面里插入恶意 html 代码，当用户浏览该页之时，嵌入其中 Web 里面的 html 代码会被执行，从而达到恶意攻击用户的特殊目的。XSS 攻击一般分三种类型：**存储型 、反射型 、DOM 型 XSS**

**XSS 是如何攻击的？**

拿反射型举个例子吧，流程图如下：



**如何解决 XSS 攻击问题**

1不相信用户的输入，对输入进行过滤，过滤标签等，只允许合法值。

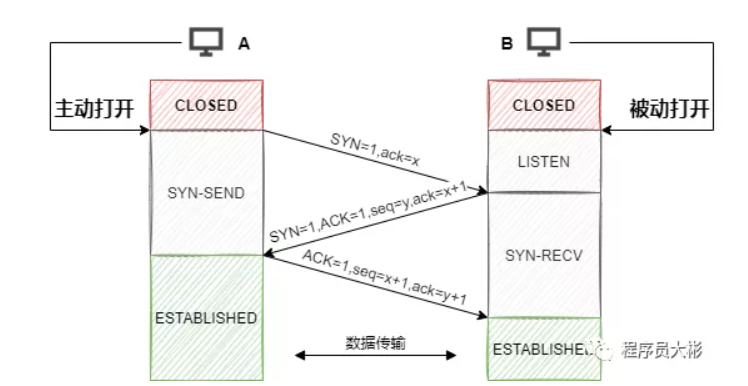
2HTML 转义

3对于链接跳转，如 <a href="xxx" 等，要校验内容，禁止以 script 开头的非法链接。

4限制输入长度等等

1. **请详细介绍一下 TCP 的三次握手机制**

假设发送端为客户端，接收端为服务端。开始时客户端和服务端的状态都是CLOSED。



1. **第一次握手**：客户端向服务端发起建立连接请求，客户端会随机生成一个起始序列号x，客户端向服务端发送的字段中包含标志位SYN=1，序列号seq=x。第一次握手前客户端的状态为CLOSE，第一次握手后客户端的状态为SYN-SENT。此时服务端的状态为LISTEN。
2. **第二次握手**：服务端在收到客户端发来的报文后，会随机生成一个服务端的起始序列号y，然后给客户端回复一段报文，其中包括标志位SYN=1，ACK=1，序列号seq=y，确认号ack=x+1。第二次握手前服务端的状态为LISTEN，第二次握手后服务端的状态为SYN-RCVD，此时客户端的状态为SYN-SENT。（其中SYN=1表示要和客户端建立一个连接，ACK=1表示确认序号有效）
3. **第三次握手**：客户端收到服务端发来的报文后，会再向服务端发送报文，其中包含标志位ACK=1，序列号seq=x+1，确认号ack=y+1。第三次握手前客户端的状态为SYN-SENT，第三次握手后客户端和服务端的状态都为ESTABLISHED。**此时连接建立完成。**

**两次握手可以吗？**

第三次握手主要为了**防止已失效的连接请求报文段**突然又传输到了服务端，导致产生问题。

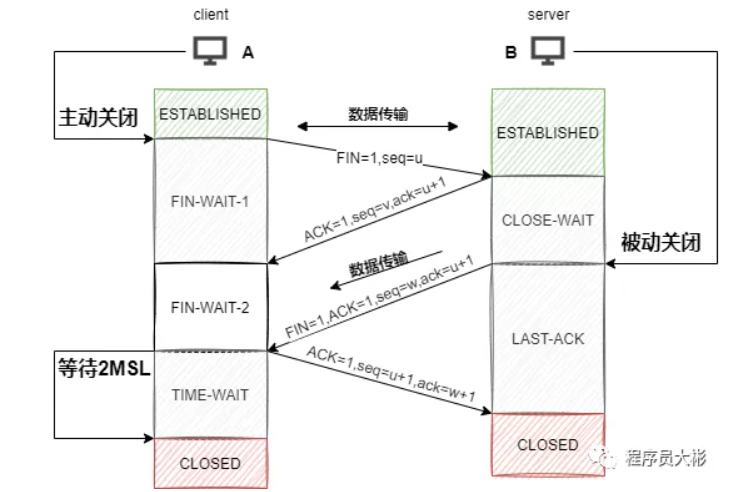
* 比如客户端A发出连接请求，可能因为网络阻塞原因，A没有收到确认报文，于是A再重传一次连接请求。
* 连接成功，等待数据传输完毕后，就释放了连接。
* 然后A发出的第一个连接请求等到连接释放以后的某个时间才到达服务端B，此时B误认为A又发出一次新的连接请求，于是就向A发出确认报文段。
* 如果不采用三次握手，只要B发出确认，就建立新的连接了，**此时A不会响应B的确认且不发送数据，则B一直等待A发送数据，浪费资源。**

**第 2 次握手传回了 ACK，为什么还要传回 SYN？**

接收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。而回传 SYN 则是为了建立并确认从服务端到客户端的通信。”

SYN 同步序列编号(Synchronize Sequence Numbers) 是 TCP/IP 建立连接时使用的握手信号。在客户机和服务器之间建立正常的 TCP 网络连接时，客户机首先发出一个 SYN 消息，服务器使用 SYN-ACK 应答表示接收到了这个消息，最后客户机再以 ACK(Acknowledgement）消息响应。这样在客户机和服务器之间才能建立起可靠的 TCP 连接，数据才可以在客户机和服务器之间传递。

1. **TCP 四次挥手过程？**
2. A的应用进程先向其TCP发出连接释放报文段（FIN=1，seq=u），并停止再发送数据，主动关闭TCP连接，进入FIN-WAIT-1（终止等待1）状态，等待B的确认。
3. B收到连接释放报文段后即发出确认报文段（ACK=1，ack=u+1，seq=v），B进入CLOSE-WAIT（关闭等待）状态，此时的TCP处于半关闭状态，A到B的连接释放。
4. A收到B的确认后，进入FIN-WAIT-2（终止等待2）状态，等待B发出的连接释放报文段。
5. B发送完数据，就会发出连接释放报文段（FIN=1，ACK=1，seq=w，ack=u+1），B进入LAST-ACK（最后确认）状态，等待A的确认。
6. A收到B的连接释放报文段后，对此发出确认报文段（ACK=1，seq=u+1，ack=w+1），A进入TIME-WAIT（时间等待）状态。此时TCP未释放掉，需要经过时间等待计时器设置的时间2MSL（最大报文段生存时间）后，A才进入CLOSED状态。B收到A发出的确认报文段后关闭连接，若没收到A发出的确认报文段，B就会重传连接释放报文段。



1. **TCP 四次挥手过程中，客户端为什么需要等待 2MSL, 才进入 CLOSED 状态**

* **保证A发送的最后一个ACK报文段能够到达B**。这个ACK报文段有可能丢失，B收不到这个确认报文，就会超时重传连接释放报文段，然后A可以在2MSL时间内收到这个重传的连接释放报文段，接着A重传一次确认，重新启动2MSL计时器，最后A和B都进入到CLOSED状态，若A在TIME-WAIT状态不等待一段时间，而是发送完ACK报文段后立即释放连接，则无法收到B重传的连接释放报文段，所以不会再发送一次确认报文段，B就无法正常进入到CLOSED状态。
* **防止已失效的连接请求报文段出现在本连接中**。A在发送完最后一个ACK报文段后，再经过2MSL，就可以使这个连接所产生的所有报文段都从网络中消失，使下一个新的连接中不会出现旧的连接请求报文段。

**2MSL，2 Maximum Segment Lifetime，即两个最大段生命周期**

1 个 MSL 保证四次挥手中主动关闭方最后的 ACK 报文能最终到达对端

1 个 MSL 保证对端没有收到 ACK 那么进行重传的 FIN 报文能够到达

1. **为什么需要四次挥手？**

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。**但是在关闭连接时，当Server端收到Client端发出的连接释放报文时，很可能并不会立即关闭SOCKET**，所以Server端先回复一个ACK报文，告诉Client端我收到你的连接释放报文了。只有等到Server端所有的报文都发送完了，这时Server端才能发送连接释放报文，之后两边才会真正的断开连接。故需要四次挥手。

1. **Session 和 Cookie 的区别。**

**Cookie 是服务器发送到用户浏览器，并保存在浏览器本地的一小块文本串数据**。它会在浏览器下次向同一服务器再发起请求时，被携带发送到服务器。通常，它用于告知服务端两个请求是否来自同一浏览器，一样用于保持用户的登录状态等。Cookie 使基于无状态的 HTTP 协议记录稳定的状态信息成为了可能。

**session 指的就是服务器和客户端一次会话的过程。** Session 利用 Cookie 进行信息处理的，当用户首先进行了请求后，服务端就在用户浏览器上创建了一个 Cookie，当这个 Session 结束时，其实就是意味着这个 Cookie 就过期了。Session 对象存储着特定用户会话所需的属性及配置信息。

**Session 和 Cookie 到底有什么不同呢？**

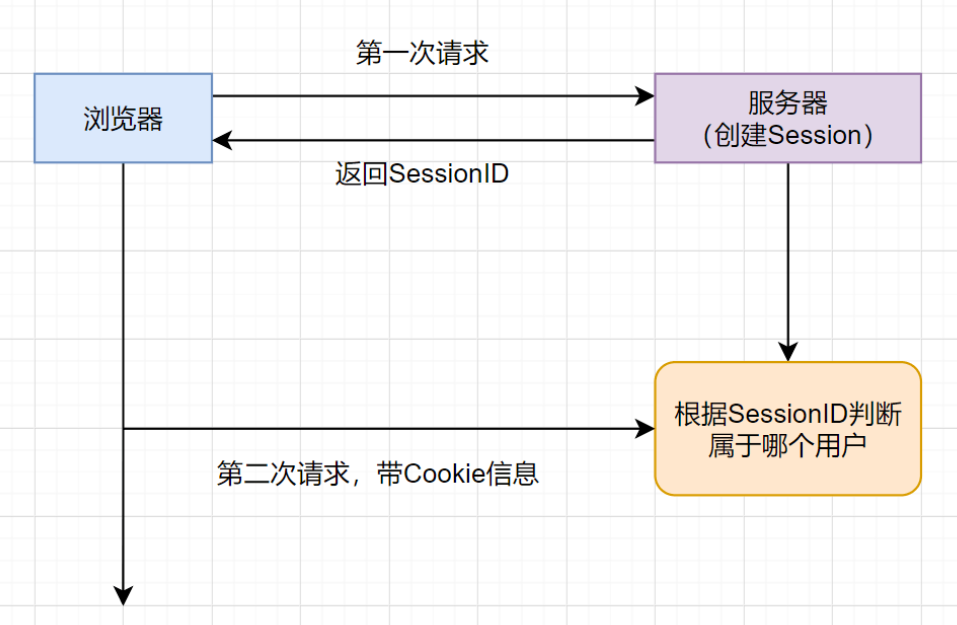
**存储位置不一样**，Cookie 保存在客户端，Session 保存在服务器端。

**存储数据类型不一样**，Cookie 只能保存 ASCII，Session 可以存任意数据类型，一般情况下我们可以在 Session 中保持一些常用变量信息，比如说 UserId 等。

**有效期不同**，Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般有效时间较短，客户端关闭或者 Session 超时都会失效。

**隐私策略不同**，Cookie 存储在客户端，比较容易遭到不法获取，早期有人将用户的登录名和密码存储在 Cookie 中导致信息被窃取；Session 存储在服务端，安全性相对 Cookie 要好一些。

**存储大小不同**， 单个 Cookie 保存的数据不能超过 4K，Session 可存储数据远高于 Cookie。



1、用户第一次请求服务器时，服务器根据用户提交的信息，创建对应的 Session ，请求返回时将此 Session 的唯一标识信息 SessionID 返回给浏览器，浏览器接收到服务器返回的 SessionID 信息后，会将此信息存入 Cookie 中，同时 Cookie 记录此 SessionID 属于哪个域名。

2、当用户第二次访问服务器时，请求会自动判断此域名下是否存在 Cookie 信息，如果存在，则自动将 Cookie 信息也发送给服务端，服务端会从 Cookie 中获取 SessionID，再根据 SessionID 查找对应的 Session 信息，如果没有找到说明用户没有登录或者登录失效，如果找到 Session 证明用户已经登录可执行后面操作。

1. **TCP 是如何保证可靠性的**
2. 应用数据被分割成 TCP 认为最适合发送的数据块。
3. TCP 给发送的每一个包进行**编号**，接收方对数据包进行**排序**，把有序数据传送给应用层。
4. **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。
5. TCP 的接收端会**丢弃重复**的数据。
6. **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）
7. **拥塞控制：** 当网络拥塞时，减少数据的发送。
8. **ARQ 协议：** 也是为了实现可靠传输的，它的基本原理就是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组。
9. **超时重传：** 当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。

**ARQ 协议**

**自动重传请求**（Automatic Repeat-reQuest，ARQ）是 OSI 模型中数据链路层和传输层的错误纠正协议之一。它通过使用确认和超时这两个机制，在不可靠服务的基础上实现可靠的信息传输。如果发送方在发送后一段时间之内没有收到确认帧，它通常会重新发送。ARQ 包括停止等待 ARQ 协议和连续 ARQ 协议。

**停止等待 ARQ 协议**

停止等待协议是为了实现可靠传输的，它的基本原理就是每发完一个分组就停止发送，等待对方确认（回复 ACK）。如果过了一段时间（超时时间后），还是没有收到 ACK 确认，说明没有发送成功，需要重新发送，直到收到确认后再发下一个分组。

在停止等待协议中，若接收方收到重复分组，就丢弃该分组，但同时还要发送确认。

**优缺点：**

* **优点：** 简单
* **缺点：** 信道利用率低，等待时间长

**1) 无差错情况:**

发送方发送分组, 接收方在规定时间内收到, 并且回复确认. 发送方再次发送。

**2) 出现差错情况（超时重传）:**

停止等待协议中超时重传是指只要超过一段时间仍然没有收到确认，就重传前面发送过的分组（认为刚才发送过的分组丢失了）。因此每发送完一个分组需要设置一个超时计时器，其重传时间应比数据在分组传输的平均往返时间更长一些。这种自动重传方式常称为 **自动重传请求 ARQ** 。另外在停止等待协议中若收到重复分组，就丢弃该分组，但同时还要发送确认。**连续 ARQ 协议** 可提高信道利用率。发送维持一个发送窗口，凡位于发送窗口内的分组可连续发送出去，而不需要等待对方确认。接收方一般采用累积确认，对按序到达的最后一个分组发送确认，表明到这个分组位置的所有分组都已经正确收到了。

**3) 确认丢失和确认迟到**

* **确认丢失** ：确认消息在传输过程丢失。当 A 发送 M1 消息，B 收到后，B 向 A 发送了一个 M1 确认消息，但却在传输过程中丢失。而 A 并不知道，在超时计时过后，A 重传 M1 消息，B 再次收到该消息后采取以下两点措施：1. 丢弃这个重复的 M1 消息，不向上层交付。 2. 向 A 发送确认消息。（不会认为已经发送过了，就不再发送。A 能重传，就证明 B 的确认消息丢失）。
* **确认迟到** ：确认消息在传输过程中迟到。A 发送 M1 消息，B 收到并发送确认。在超时时间内没有收到确认消息，A 重传 M1 消息，B 仍然收到并继续发送确认消息（B 收到了 2 份 M1）。此时 A 收到了 B 第二次发送的确认消息。接着发送其他数据。过了一会，A 收到了 B 第一次发送的对 M1 的确认消息（A 也收到了 2 份确认消息）。处理如下：1. A 收到重复的确认后，直接丢弃。2. B 收到重复的 M1 后，也直接丢弃重复的 M1。

**连续 ARQ 协议**

连续 ARQ 协议可提高信道利用率。发送方维持一个发送窗口，凡位于发送窗口内的分组可以连续发送出去，而不需要等待对方确认。接收方一般采用累计确认，对按序到达的最后一个分组发送确认，表明到这个分组为止的所有分组都已经正确收到了。

**优缺点：**

* **优点：** 信道利用率高，容易实现，即使确认丢失，也不必重传。
* **缺点：** 不能向发送方反映出接收方已经正确收到的所有分组的信息。 比如：发送方发送了 5 条 消息，中间第三条丢失（3 号），这时接收方只能对前两个发送确认。发送方无法知道后三个分组的下落，而只好把后三个全部重传一次。这也叫 Go-Back-N（回退 N），表示需要退回来重传已经发送过的 N 个消息。

**滑动窗口和流量控制**

**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。

**拥塞控制**

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。

1. **TCP 和 UDP 的区别**



TCP 面向连接（（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接。

TCP 要求安全性，提供可靠的服务，通过 TCP 连接传送的数据，不丢失、不重复、安全可靠。而 UDP 尽最大努力交付，即不保证可靠交付。

TCP 是点对点连接的，UDP 一对一，一对多，多对多都可以

TCP 传输效率相对较低，而 UDP 传输效率高，它适用于对高速传输和实时性有较高的通信或广播通信。

TCP 适合用于网页，邮件等；UDP 适合用于视频，语音广播等

TCP 面向字节流，UDP 面向报文

UDP 在传送数据之前不需要先建立连接，远地主机在收到 UDP 报文后，不需要给出任何确认。虽然 UDP 不提供可靠交付，但在某些情况下 UDP 却是一种最有效的工作方式（一般用于即时通信），比如： QQ 语音、 QQ 视频 、直播等等

TCP 提供面向连接的服务。在传送数据之前必须先建立连接，数据传送结束后要释放连接。 TCP 不提供广播或多播服务。由于 TCP 要提供可靠的，面向连接的传输服务（TCP 的可靠体现在 TCP 在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源），这一难以避免增加了许多开销，如确认，流量控制，计时器以及连接管理等。这不仅使协议数据单元的首部增大很多，还要占用许多处理机资源。TCP 一般用于文件传输、发送和接收邮件、远程登录等场景。

1. **TCP 报文首部有哪些字段，说说其作用**

**16 位端口号**：源端口号，主机该报文段是来自哪里；目标端口号，要传给哪个上层协议或应用程序

**32 位序号**：一次 TCP 通信（从 TCP 连接建立到断开）过程中某一个传输方向上的字节流的每个字节的编号。

**32 位确认号**：用作对另一方发送的 tcp 报文段的响应。其值是收到的 TCP 报文段的序号值加 1。

**4 位头部长度**：表示 tcp 头部有多少个 32bit 字（4 字节）。因为 4 位最大能标识 15，所以 TCP 头部最长是 60 字节。

**6 位标志位**：URG (紧急指针是否有效)，ACk（表示确认号是否有效），PSH（缓冲区尚未填满），RST（表示要求对方重新建立连接），SYN（建立连接消息标志接），FIN（表示告知对方本端要关闭连接了）

**16 位窗口大小**：是 TCP 流量控制的一个手段。这里说的窗口，指的是接收通告窗口。它告诉对方本端的 TCP 接收缓冲区还能容纳多少字节的数据，这样对方就可以控制发送数据的速度。

**16 位校验和**：由发送端填充，接收端对 TCP 报文段执行 CRC 算法以检验 TCP 报文段在传输过程中是否损坏。注意，这个校验不仅包括 TCP 头部，也包括数据部分。这也是 TCP 可靠传输的一个重要保障。

**16 位紧急指针**：一个正的偏移量。它和序号字段的值相加表示最后一个紧急数据的下一字节的序号。因此，确切地说，这个字段是紧急指针相对当前序号的偏移，不妨称之为紧急偏移。TCP 的紧急指针是发送端向接收端发送紧急数据的方法。

1. **HTTP 状态码 301 和 302 的区别？**

**301（永久移动）**请求的网页已被永久移动到新位置。服务器返回此响应（作为对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。

**302：（临时移动）**服务器目前正从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。此代码与响应 GET 和 HEAD 请求的 301 代码类似，会自动将请求者转到不同的位置。

HTTP 状态码 301 与 302 的区别：

**它们之间关键区别在，资源是否存在有效性；**

301 资源还在只是换了一个位置，返回的是新位置的内容；

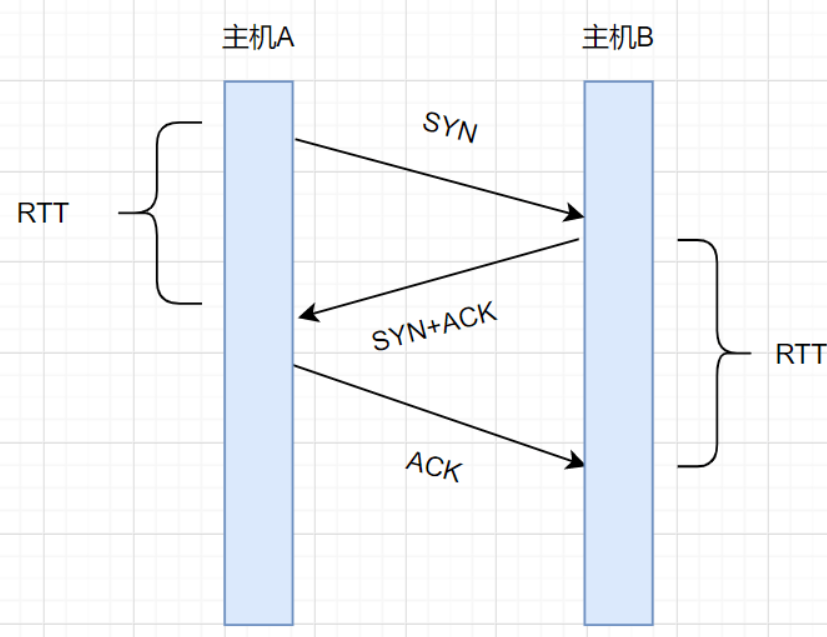
302 资源暂时失效，返回的是一个临时的代替页上。

1. **聊聊 TCP 的重传机制**

超时重传

TCP 为了实现可靠传输，实现了重传机制。最基本的重传机制，就是超时重传，即在发送数据报文时，设定一个定时器，每间隔一段时间，没有收到对方的 ACK 确认应答报文，就会重发该报文。

这个间隔时间，一般设置为多少呢？我们先来看下什么叫 **RTT（Round-Trip Time，往返时间**）。



RTT 就是，一个数据包从发出去到回来的时间，即数据包的一次往返时间。**超时重传时间，就是 Retransmission Timeout ，简称 RTO**。

**RTO 设置多久呢？**

如果 RTO 比较小，那很可能数据都没有丢失，就重发了，这会导致网络阻塞，会导致更多的超时出现。

如果 RTO 比较大，等到花儿都谢了还是没有重发，那效果就不好了。

**一般情况下，RTO 略大于 RTT，效果是最好的**。一些小伙伴会问，超时时间有没有计算公式呢？有的！有个标准方法算 RTO 的公式，也叫 Jacobson / Karels 算法。我们一起来看下计算 RTO 的公式

**1. 先计算 SRTT（计算平滑的 RTT）**

SRTT = (1 - α) \* SRTT + α \* RTT //求 SRTT 的加权平均

**2. 再计算 RTTVAR (round-trip time variation)**

RTTVAR = (1 - β) \* RTTVAR + β \* (|RTT - SRTT|) //计算 SRTT 与真实值的差距

**3. 最终的 RTO**

RTO = µ \* SRTT + ∂ \* RTTVAR = SRTT + 4·RTTVAR

其中，α = 0.125，β = 0.25， μ = 1，∂ = 4，这些参数都是大量结果得出的最优参数。

但是，**超时重传会有这些缺点**：

* 当一个报文段丢失时，会等待一定的超时周期然后才重传分组，增加了端到端的时延。
* 当一个报文段丢失时，在其等待超时的过程中，可能会出现这种情况：其后的报文段已经被接收端接收但却迟迟得不到确认，发送端会认为也丢失了，从而引起不必要的重传，既浪费资源也浪费时间。

并且，TCP 有个策略，就是超时时间间隔会加倍。超时重传需要**等待很长时间**。因此，还可以使用**快速重传机制**。

**快速重传**

快速重传机制，它不以时间驱动，而是以数据驱动。它基于接收端的反馈信息来引发重传。

发送端发送了 1，2，3，4，5,6 份数据:

第一份 Seq=1 先送到了，于是就 Ack 回 2；

第二份 Seq=2 也送到了，假设也正常，于是 ACK 回 3；

第三份 Seq=3 由于网络等其他原因，没送到；

第四份 Seq=4 也送到了，但是因为 Seq3 没收到。所以 ACK 回 3；

后面的 Seq=4,5 的也送到了，但是 ACK 还是回复 3，因为 Seq=3 没收到。

发送端连着收到三个重复冗余 ACK=3 的确认（实际上是 4 个，但是前面一个是正常的 ACK，后面三个才是重复冗余的），便知道哪个报文段在传输过程中丢失了，于是在定时器过期之前，重传该报文段。

最后，接收到收到了 Seq3，此时因为 Seq=4，5，6 都收到了，于是 ACK 回 7.

但快速重传还可能会有个问题：**ACK 只向发送端告知最大的有序报文段，到底是哪个报文丢失了呢？并不确定！那到底该重传多少个包呢？**

是重传 Seq3 呢？还是重传 Seq3、Seq4、Seq5、Seq6 呢？因为发送端并不清楚这三个连续的 ACK3 是谁传回来的。

**带选择确认的重传（SACK）**

为了解决快速重传的问题：应该重传多少个包 ? **TCP 提供了 SACK 方法（带选择确认的重传，Selective Acknowledgment）**。

SACK 机制就是，在快速重传的基础上，**接收端返回最近收到的报文段的序列号范围**，这样发送端就知道接收端哪些数据包没收到，酱紫就很清楚该重传哪些数据包啦。**SACK 标记是加在 TCP 头部选项字段里面的。**

如上图中，发送端收到了三次同样的 ACK=30 的确认报文，于是就会触发快速重发机制，通过 SACK 信息发现只有 30~39 这段数据丢失，于是重发时就只选择了这个 30~39 的 TCP 报文段进行重发。

**D-SACK**

D-SACK，即 Duplicate SACK（重复 SACK），在 SACK 的基础上做了一些扩展，，主要用来告诉发送方，有哪些数据包自己重复接受了。DSACK 的目的是帮助发送方判断，是否发生了包失序、ACK 丢失、包重复或伪重传。让 TCP 可以更好的做网络流控。

1. **IP 地址有哪些分类？**

一句话概括，**IP 地址 = 网络号 + 主机号。**

**网络号**：它标志主机（或路由器）所连接到的网络，网络地址表示属于互联网的哪一个网络

**主机号**：它标志该主机（或路由器），主机地址表示其属于该网络中的哪一台主机

IP 地址 分为 A，B，C，D，E 五大类：

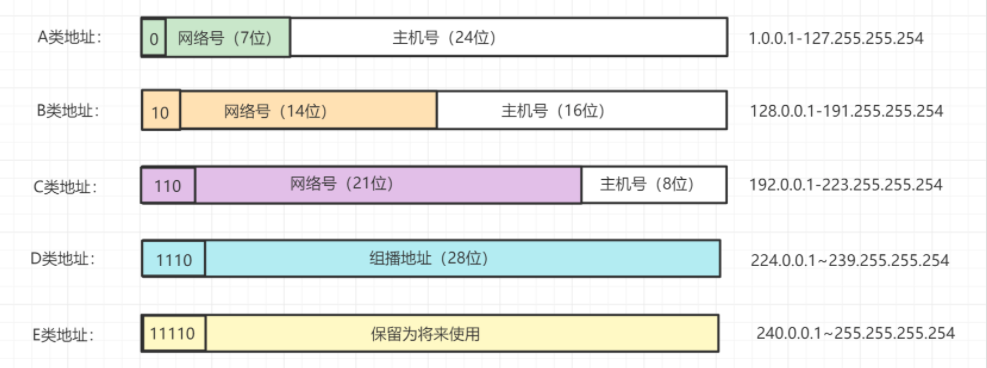
A 类地址 (1~126)：以 0 开头，网络号占前 8 位，主机号占后 24 位。

B 类地址 (128~191)：以 10 开头，网络号占前 16 位，主机号占后 16 位。

C 类地址 (192~223)：以 110 开头，网络号占前 24 位，主机号占后 8 位。

D 类地址 (224~239)：以 1110 开头，保留位多播地址。

E 类地址 (240~255)：以 11110 开头，保留位为将来使用



1. **聊聊 TCP 的滑动窗口**

TCP 发送一个数据，需要收到确认应答，才会发送下一个数据。这样有个缺点，就是效率会比较低。

这就好像我们面对面聊天，你说完一句，我应答后，你才会说下一句。那么，如果我在忙其他事情，没有能够及时回复你。你说完一句后，要等到我忙完回复你，你才说下句，这显然很不现实。

为了解决这个问题，**TCP 引入了窗口**，**它是操作系统开辟的一个缓存空间。窗口大小值表示无需等待确认应答，而可以继续发送数据的最大值**。

**TCP 头部**有个字段叫 **win**，也即那个 **16 位的窗口大小**，它告诉对方本端的 **TCP 接收缓冲区还能容纳多少字节的数据**，这样对方就可以控制发送数据的速度，从而达到流量控制的目的。

通俗点讲，就是接受方每次收到数据包，在发送确认报文的时候，同时告诉发送方，自己的缓存区还有多少空余空间，缓冲区的空余空间，我们就称之为**接受窗口大小**。这就是 win。

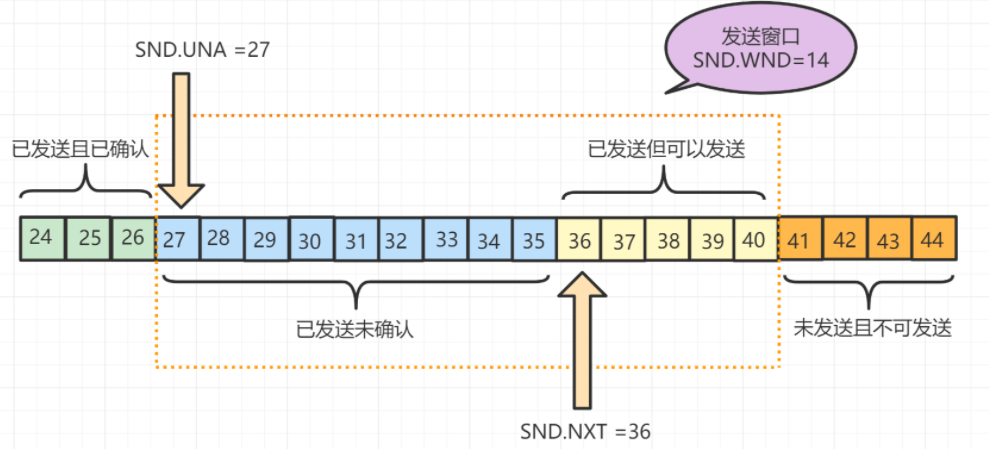
TCP 滑动窗口分为两种：发送窗口和接收窗口。发送端的滑动窗口包含四大部分，如下：

已发送且已收到 ACK 确认

已发送但未收到 ACK 确认

未发送但可以发送

未发送也不可以发送



虚线矩形框，就是**发送窗口**。

**SND.WND**: 表示发送窗口的大小，上图虚线框的格子数就是 14 个。

**SND.UNA:** 一个绝对指针，它指向的是已发送但未确认的第一个字节的序列号。

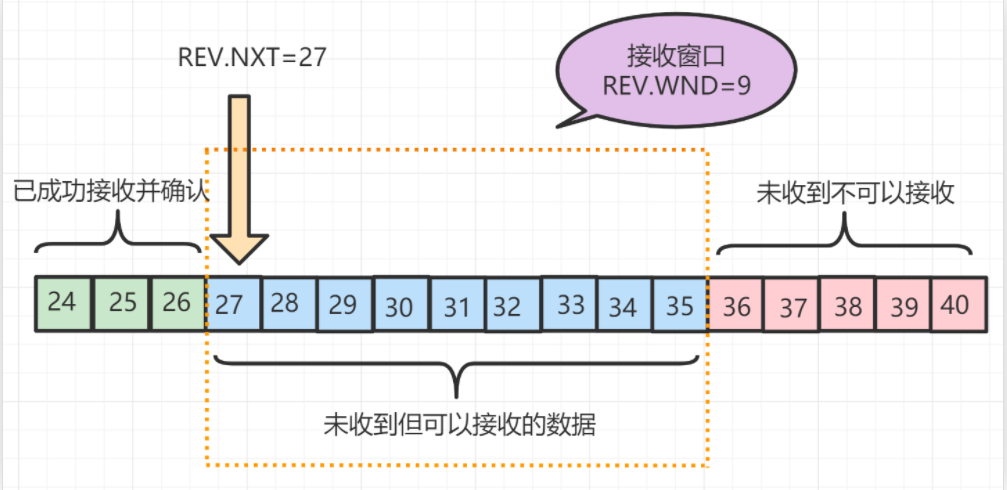
**SND.NXT**：下一个发送的位置，它指向未发送但可以发送的第一个字节的序列号。

接收方的滑动窗口包含三大部分，如下：

**已成功接收并确认**

**未收到数据但可以接收**

**未收到数据并不可以接收的数据**



虚线矩形框，就是**接收窗口**。

**REV.WND**: 表示接收窗口的大小，上图虚线框的格子就是 9 个。

**REV.NXT**: 下一个接收的位置，它指向未收到但可以接收的第一个字节的序列号。

1. **聊聊五层计算机网络体系结构中，每一层对应的网络协议有哪些？**

**应用层**：HTTP超文本传输协议，FTP文本传输协议；SMTP简单邮件传输协议；DNS域名系统；SSH安全外壳协议；DHCP动态主机配置协议；TELNET远程登录协议；

**传输层**：TCP传输控制协议；UDP用户数据报文协议；

**网络层**：IP网际协议；ARP地址转换协议；RARP反向地址转换协议；ICMP(Internet控制报文协议)IGMP(Internet组管理协议)；RIP路由信息协议；OSPF分布式链路状态协议；BGP边界网关协议

**数据链路层**：ARQ自动重传请求协议；CSMA/CD停止等待协议；PPP点对点协议

**物理层**：HUB;网线；中继器

1. **聊聊 TCP 的流量控制**

TCP 三次握手，发送端和接收端进入到 ESTABLISHED 状态，它们即可以愉快地传输数据啦。

但是发送端不能疯狂地向接收端发送数据，因为接收端接收不过来的话，接收方只能把处理不过来的数据存在缓存区里。如果缓存区都满了，发送方还在疯狂发送数据的话，接收方只能把收到的数据包丢掉，这就浪费了网络资源啦。

TCP 提供一种机制可以让发送端根据接收端的实际接收能力控制发送的数据量，这就是**流量控制**。

TCP 通过滑动窗口来控制流量，我们看下流量控制的简要流程吧：

首先双方三次握手，初始化各自的窗口大小，均为 400 个字节。

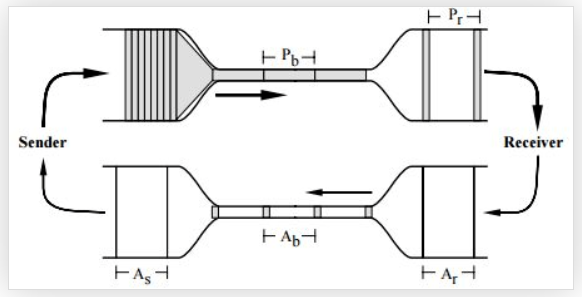
1. 假如当前发送方给接收方发送了 200 个字节，那么，发送方的 SND.NXT 会右移 200 个字节，也就是说当前的可用窗口减少了 200 个字节。
2. 接受方收到后，放到缓冲队列里面，REV.WND =400-200=200 字节，所以 win=200 字节返回给发送方。接收方会在 ACK 的报文首部带上缩小后的滑动窗口 200 字节
3. 发送方又发送 200 字节过来，200 字节到达，继续放到缓冲队列。不过这时候，由于大量负载的原因，接受方处理不了这么多字节，只能处理 100 字节，剩余的 100 字节继续放到缓冲队列。这时候，REV.WND = 400-200-100=100 字节，即 win=100 返回发送方。
4. 发送方继续干活，发送 100 字节过来，这时候，接受窗口 win 变为 0。
5. 发送方停止发送，开启一个定时任务，每隔一段时间，就去询问接受方，直到 win 大于 0，才继续开始发送。
6. **说下 ARP 协议的工作原理？**

**ARP 协议协议，即 Address Resolution Protocol，地址解析协议**，用于实现 IP 地址到 MAC 地址的映射。

1. 首先，每台主机都会在自己的 ARP 缓冲区中建立一个 ARP 列表，以表示 IP 地址和 MAC 地址的对应关系。
2. 当源主机需要将一个数据包要发送到目的主机时，会首先检查自己的 ARP 列表，是否存在该 IP 地址对应的 MAC 地址；如果存在﹐就直接将数据包发送到这个 MAC 地址；如果不存在，就向本地网段发起一个 ARP 请求的广播包，查询此目的主机对应的 MAC 地址。此 ARP 请求的数据包里，包括源主机的 IP 地址、硬件地址、以及目的主机的 IP 地址。
3. 网络中所有的主机收到这个 ARP 请求后，会检查数据包中的目的 IP 是否和自己的 IP 地址一致。如果不相同就忽略此数据包；如果相同，该主机首先将发送端的 MAC 地址和 IP 地址添加到自己的 ARP 列表中，如果 ARP 表中已经存在该 IP 的信息，则将其覆盖，然后给源主机发送一个 ARP 响应数据包，告诉对方自己是它需要查找的 MAC 地址。
4. 源主机收到这个 ARP 响应数据包后，将得到的目的主机的 IP 地址和 MAC 地址添加到自己的 ARP 列表中，并利用此信息开始数据的传输。如果源主机一直没有收到 ARP 响应数据包，表示 ARP 查询失败。
5. **说下 TCP 的拥塞控制**

**拥塞控制是作用于网络的，防止过多的数据包注入到网络中，避免出现网络负载过大的情况**。它的目标主要是最大化利用网络上瓶颈链路的带宽。它跟流量控制又有什么区别呢？**流量控制是作用于接收者的，根据接收端的实际接收能力控制发送速度，防止分组丢失的。**

我们可以把网络链路比喻成一根水管，如果我们想最大化利用网络来传输数据，那就是尽快让水管达到最佳充满状态。



**发送方**维护一个**拥塞窗口 cwnd（congestion window）** 的变量，用来估算在一段时间内这条链路（水管）可以承载和运输的数据（水）的数量。**它大小代表着网络的拥塞程度，并且是动态变化的**，但是为了达到最大的传输效率，我们该如何知道这条水管的运送效率是多少呢？

一个比较简单的方法就是不断增加传输的水量，直到水管快要爆裂为止（对应到网络上就是发生丢包），用 TCP 的描述就是：

**只要网络中没有出现拥塞，拥塞窗口的值就可以再增大一些，以便把更多的数据包发送出去，但只要网络出现拥塞，拥塞窗口的值就应该减小一些，以减少注入到网络中的数据包数。**

实际上，拥塞控制主要有这几种常用算法

1. **慢启动**
2. **拥塞避免**
3. **拥塞发生**
4. **快速恢复**

**慢启动算法**

慢启动算法，表面意思就是，别急慢慢来。它表示 TCP 建立连接完成后，一开始不要发送大量的数据，而是先探测一下网络的拥塞程度。由小到大逐渐增加拥塞窗口的大小，如果没有出现丢包，**每收到一个 ACK，就将拥塞窗口 cwnd 大小就加 1**（单位是 MSS）。**每轮次发送窗口增加一倍，呈指数增长，如果出现丢包，拥塞窗口就减半，进入拥塞避免阶段。**

1. TCP 连接完成，初始化 cwnd = 1，表明可以传一个 MSS 单位大小的数据。
2. 每当收到一个 ACK，cwnd 就加一；
3. 每当过了一个 RTT，cwnd 就增加一倍；呈指数让升

为了防止 cwnd 增长过大引起网络拥塞，还需设置一个**慢启动阀值 ssthresh**（slow start threshold）状态变量。当 cwnd 到达该阀值后，就好像水管被关小了水龙头一样，**减少拥塞状态**。**即当 cwnd >ssthresh 时，进入了拥塞避免算法。**

**拥塞避免算法**

一般来说，慢启动阀值 ssthresh 是 65535 字节，cwnd 到达慢启动阀值后

1. 每收到一个 ACK 时，cwnd = cwnd + 1/cwnd
2. 当每过一个 RTT 时，cwnd = cwnd + 1

显然这是一个**线性上升**的算法，避免过快导致网络拥塞问题。

**拥塞发生**

当网络拥塞发生丢包时，会有两种情况：

1. RTO 超时重传
2. 快速重传

如果是发生了 **RTO 超时重传**，就会使用拥塞发生算法

1. 慢启动阀值 sshthresh = cwnd /2
2. cwnd 重置为 1
3. 进入新的慢启动过程

这真的是辛辛苦苦几十年，一朝回到解放前。其实还有更好的处理方式，就是**快速重传。发送方收到 3 个连续重复的 ACK 时，就会快速地重传，不必等待 RTO 超时再重传。**

慢启动阀值 ssthresh 和 cwnd 变化如下：

1. 拥塞窗口大小 cwnd = cwnd/2
2. 慢启动阀值 ssthresh = cwnd
3. 进入快速恢复算法

**快速恢复**

快速重传和快速恢复算法一般同时使用。**快速恢复算法认为，还有 3 个重复 ACK 收到，说明网络也没那么糟糕，所以没有必要像 RTO 超时那么强烈。**

正如前面所说，进入快速恢复之前，cwnd 和 sshthresh 已被更新：

- cwnd = cwnd /2

- sshthresh = cwnd

然后，真正的快速算法如下：

1. cwnd = sshthresh + 3
2. 重传重复的那几个 ACK（即丢失的那几个数据包）
3. 如果再收到重复的 ACK，那么 cwnd = cwnd +1
4. 如果收到新数据的 ACK 后，cwnd = sshthresh。因为收到新数据的 ACK，表明恢复过程已经结束，可以再次进入了拥塞避免的算法了。
5. **TCP 和 UDP 分别对应的常见应用层协议有哪些？**

**基于 TCP 的应用层协议有：HTTP、FTP、SMTP、TELNET、SSH**

**HTTP**：HyperText Transfer Protocol（**超文本传输协议**），默认端口 **80**

**FTP**: File Transfer Protocol (**文件传输协议**), 默认端口 (**20** 用于**传输数据**，21 用于**传输控制信息**)

**SMTP**: Simple Mail Transfer Protocol (**简单邮件传输协议**) , 默认端口 **25**

**TELNET**: Teletype over the Network (**网络电传**), 默认端口 **23**

**SSH**： Secure Shell（**安全外壳协议**），默认端口 **22**

**基于 UDP 的应用层协议：DNS、TFTP、SNMP**

**DNS** : Domain Name Service (**域名服务**), 默认端口 **53**

**TFTP**: Trivial File Transfer Protocol (**简单文件传输协议**)，默认端口 **69**

**SNMP**：Simple Network Management Protocol（**简单网络管理协议**），通过 UDP 端口 **161** 接收，只有 Trap 信息采用 UDP 端口 **162**。

1. **半连接队列和 SYN Flood 攻击的关系**

TCP 进入三次握手前，**服务端**会从 CLOSED 状态变为 LISTEN 状态，同时在内部创建了两个队列：**半连接队列（SYN 队列）和全连接队列（ACCEPT 队列）**。

什么是半连接队列（SYN 队列） 呢？什么是全连接队列（ACCEPT 队列） 呢？回忆下 TCP 三次握手的图：

1. TCP 三次握手时，客户端发送 SYN 到服务端，服务端收到之后，便回复 ACK 和 SYN，状态由 LISTEN 变为 SYN\_RCVD，此时**这个连接就被推入了 SYN 队列**，即**半连接队列**。
2. 当客户端回复 ACK, 服务端接收后，三次握手就完成了。这时连接会等待被具体的应用取走，在被取走之前，它被**推入 ACCEPT 队列，即全连接队列**。

**SYN Flood 是一种典型的 DoS (Denial of Service，拒绝服务) 攻击**，它在短时间内，伪造不存在的 IP 地址 , 向服务器大量发起 SYN 报文。当服务器回复 SYN+ACK 报文后，不会收到 ACK 回应报文，导致服务器上建立大量的半连接半连接队列满了，这就无法处理正常的 TCP 请求啦。

主要有 **syn cookie 和 SYN Proxy 防火墙**等方案应对。

1. **syn cookie**：在收到 SYN 包后，服务器根据一定的方法，以数据包的源地址、端口等信息为参数计算出一个 cookie 值作为自己的 SYNACK 包的序列号，回复 SYN+ACK 后，服务器并不立即分配资源进行处理，等收到发送方的 ACK 包后，重新根据数据包的源地址、端口计算该包中的确认序列号是否正确，如果正确则建立连接，否则丢弃该包。
2. **SYN Proxy 防火墙**：服务器防火墙会对收到的每一个 SYN 报文进行代理和回应，并保持半连接。等发送方将 ACK 包返回后，再重新构造 SYN 包发到服务器，建立真正的 TCP 连接。
3. **有了 IP 地址，为什么还要用 MAC 地址？**
4. 简而言之，标识网络中的一台计算机，比较常用的就是 **IP 地址和 MAC 地址**，但计算机的 IP 地址可由用户自行更改，管理起来就相对困难，而 **MAC 地址不可更改**，所以**一般会把 IP 地址和 MAC 地址组合起来使用**。
5. 那只使用 MAC 地址不用 IP 地址行不行呢？不行的！因为最早就是 MAC 地址先出现的，并且当时并不用 IP 地址，**只用 MAC 地址，后来随着网络中的设备越来越多，整个路由过程越来越复杂**，便出现了**子网**的概念。**对于目的地址在其他子网的数据包，路由只需要将数据包送到那个子网即可**。
6. 那为什么要用 IP 地址呢？是因为 **IP 地址是和地域相关的，对于同一个子网上的设备，IP 地址的前缀都是一样的**，这样路由器通过 IP 地址的前缀就知道设备在在哪个子网上了，而只用 MAC 地址的话，路由器则需要记住每个 MAC 地址在哪个子网，这需要路由器有极大的存储空间，是无法实现的。
7. **IP 地址可以比作为地址，MAC 地址为收件人**，在一次通信过程中，两者是缺一不可的。
8. **聊聊保活计时器的作用**

除时间等待计时器外，TCP 还有一个**保活计时器（keepalive timer）**。设想这样的场景：客户已主动与服务器建立了 TCP 连接。但后来客户端的主机突然发生故障。显然，服务器以后就不能再收到客户端发来的数据。因此，应当有措施使服务器不要再白白等待下去。这就需要使用保活计时器了。

服务器每收到一次客户的数据，就重新设置保活计时器，时间的设置通常是两个小时。若两个小时都没有收到客户端的数据，服务端就发送一个探测报文段，以后则每隔 75 秒钟发送一次。若连续发送 10 个探测报文段后仍然无客户端的响应，服务端就认为客户端出了故障，接着就关闭这个连接。

1. **聊聊 ARP 协议**

ARP 协议，地址解析协议，是一个由 IP 地址获取 MAC 物理地址的 TCP/IP 协议。

**什么是 IP 地址，什么是 MAC 地址？**

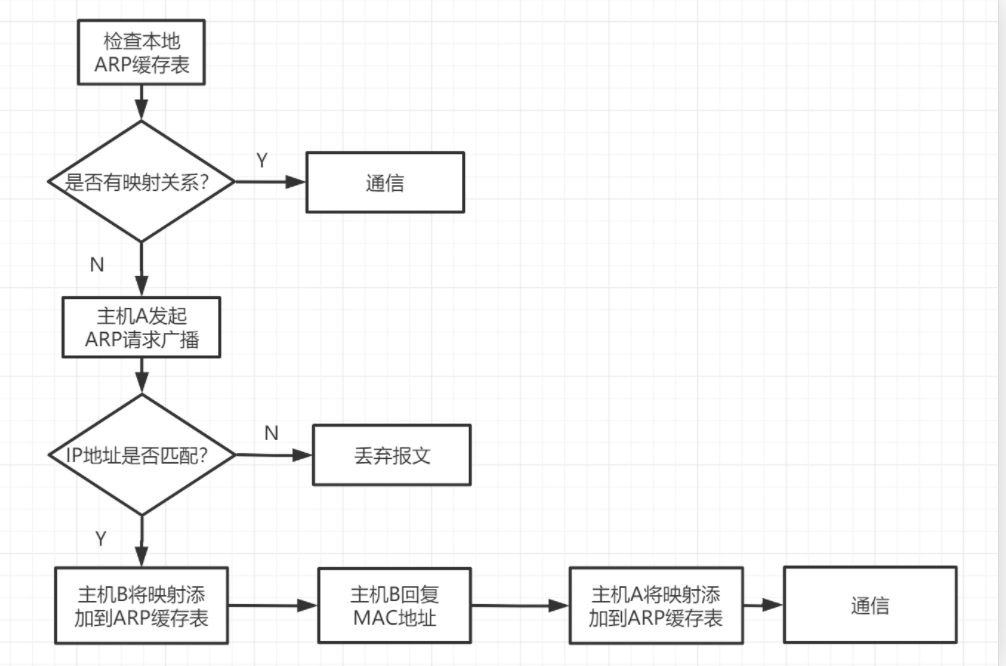
IP 地址：是互联网协议地址，它是 IP 协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。

MAC 地址：以太网地址或物理地址，它是一个用来确认网络设备位置的位址。

**为什么需要 ARP 协议呢？**

1. 在网络访问层中，同一局域网中的一台主机要和另一台主机进行通信，需要通过 MAC 地址进行定位，然后才能进行数据包的发送。
2. 而在网络层和传输层中，计算机之间是通过 IP 地址定位目标主机，对应的数据报文只包含目标主机的 IP 地址，而没有 MAC 地址。
3. 因此，在发送之前需要根据 IP 地址获取 MAC 地址，然后才能将数据包发送到正确的目标主机，而这个获取过程是通过 ARP 协议完成的。

**ARP 的工作流程**



当主机 A 与主机 B 要通信时，工作流程如下：

1. 查询本地 ARP 缓存表，看是否有 IP 地址及其对应的 MAC 地址。
2. 如果没匹配到主机 B 的 MAC 地址，主机 A 会在局域网内广播发送一个 ARP 请求分组，局域网内所有主机都会收到该请求分组。
3. 主机 B 收到请求分组报文，发现报文中的 IP 与自己匹配，就 A 的 IP 和 MAC 地址添加到本地 ARP 缓存表中。
4. 主机 B 向主机 A 响应一个含自身 MAC 地址的报文。
5. 主机 A 收到报文后，将 B 的 IP 和 MAC 地址添加至 ARP 缓存表中。
6. **TCP 的粘包和拆包**

**TCP 是面向流**，**没有界限的一串数据**。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**。

**为什么会产生粘包和拆包呢？**

1. 要发送的数据小于 TCP 发送缓冲区的大小，TCP 将多次写入缓冲区的数据一次发送出去，将会发生粘包；
2. 接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包；
3. 要发送的数据大于 TCP 发送缓冲区剩余空间大小，将会发生拆包；
4. 待发送数据大于 MSS（最大报文长度），TCP 在传输前将进行拆包。即 TCP 报文长度 - TCP 头部长度 > MSS。

**解决方案：**

1. 发送端将每个数据包封装为固定长度
2. 在数据尾部增加特殊字符进行分割
3. 将数据分为两部分，一部分是头部，一部分是内容体；其中头部结构大小固定，且有一个字段声明内容体的大小。
4. **forward 和 redirect 的区别？**
5. **直接转发方式（Forward）** ，客户端和浏览器只发出一次请求，Servlet、HTML、JSP 或其它信息资源，由第二个信息资源响应该请求，在请求对象 request 中，保存的对象对于每个信息资源是共享的。
6. **间接转发方式（Redirect）** 实际是两次 HTTP 请求，服务器端在响应第一次请求的时候，让浏览器再向另外一个 URL 发出请求，从而达到转发的目的。

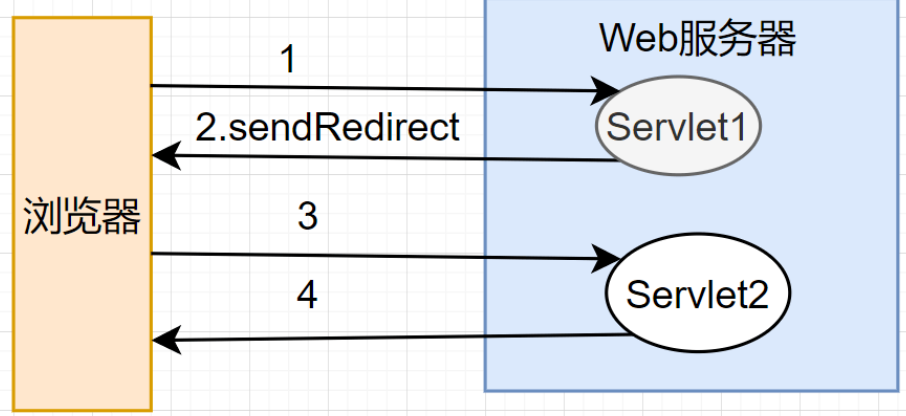
举个通俗的例子：

直接转发就相当于：“A 找 B 借钱，B 说没有，B 去找 C 借，借到借不到都会把消息传递给 A”；

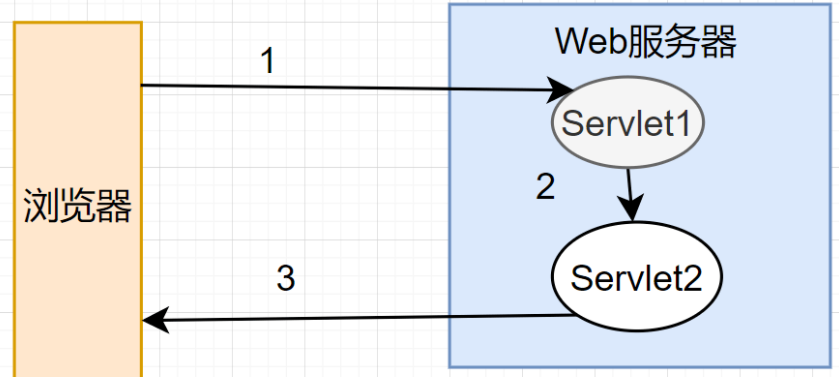
间接转发就相当于：”A 找 B 借钱，B 说没有，让 A 去找 C 借”。\*\*

看这两个图，可以更容易理解一些:

**Redirect 的工作原理：**



**forward 的工作原理**



1. **Nagle 算法与延迟确认**

**Nagle 算法**

如果发送端疯狂地向接收端发送很小的包，比如就 1 个字节，那么亲爱的小伙伴，你们觉得会有什么问题呢？

**TCP/IP 协议中，无论发送多少数据，总是要在数据前面加上协议头，同时，对方接收到数据，也需要发送 ACK 表示确认。为了尽可能的利用网络带宽，TCP 总是希望尽可能的发送足够大的数据。Nagle 算法就是为了尽可能发送大块数据，避免网络中充斥着许多小数据块。**

Nagle 算法的基本定义是：**任意时刻，最多只能有一个未被确认的小段**。 所谓 “小段”，指的是小于 MSS 尺寸的数据块，所谓 “未被确认”，是指一个数据块发送出去后，没有收到对方发送的 ACK 确认该数据已收到。

**Nagle 算法的实现规则：**

1. 如果包长度达到 MSS，则允许发送；
2. 如果该包含有 FIN，则允许发送；
3. 设置了 TCP\_NODELAY 选项，则允许发送；
4. 未设置 TCP\_CORK 选项时，若所有发出去的小数据包（包长度小于 MSS）均被确认，则允许发送；
5. 上述条件都未满足，但发生了超时（一般为 200ms），则立即发送。

**延迟确认**

如果接受方刚接收到发送方的数据包，在很短很短的时间内，又接收到第二个包。那么请问接收方是一个一个地回复好点，还是合并一起回复好呢？

接收方收到数据包后，如果暂时没有数据要发给对端，它可以等一段时再确认（Linux 上默认是 40ms）。如果这段时间刚好有数据要传给对端，ACK 就随着数据传输，而不需要单独发送一次 ACK。如果超过时间还没有数据要发送，也发送 ACK，避免对端以为丢包。

但是有些场景不能延迟确认，比如发现了**乱序包、接收到了大于一个 frame 的报文，且需要调整窗口大小等。**

一般情况下，**Nagle 算法和延迟确认**不能一起使用，**Nagle 算法**意味着延迟发，**延迟确认**意味着延迟接收，酱紫就会造成更大的延迟，会产生性能问题。

1. **URI 和 URL 的区别**

**URI**，全称是 Uniform Resource Identifier)，中文翻译是统一资源标志符，主要作用是**唯一标识一个资源。**

**URL**，全称是 Uniform Resource Location)，中文翻译是**统一资源定位符**，主要作用是**提供资源的路径**。

打个经典比喻吧，URI 像是身份证，可以唯一标识一个人，而 URL 更像一个住址，可以通过 URL 找到这个人。

1. **什么是数字签名？ 什么是数字证书？**

了解过 Https 原理的小伙伴，都知道数字证书这玩意。为了**避免公钥被篡改**，引入了数字证书，如下：

**数字证书构成**

1. 公钥和个人信息，经过 Hash 算法加密，形成消息摘要；将消息摘要拿到拥有公信力的认证中心（CA），用它的私钥对消息摘要加密，形成数字签名.
2. 公钥和个人信息、数字签名共同构成数字证书。
3. **什么是 SQL 注入？举个例子？**

SQL 注入是一种代码注入技术，一般被应用于攻击 web 应用程序。它通过在 web 应用接口传入一些特殊参数字符，来欺骗应用服务器，执行恶意的 SQL 命令，以达到非法获取系统信息的目的。它目前是黑客对数据库进行攻击的最常用手段之一。

**SQL 注入是如何攻击的？**

举个常见的业务场景：在 web 表单搜索框输入员工名字，然后后台查询出对应名字的员工。

这种场景下，一般都是前端页面把一个名字参数 name 传到后台，然后后台通过 SQL 把结果查询出来

name = "田螺"; //前端传过来的

SQL= "select \* from staff where name=" + name; //根据前端传过来的name参数，查询数据库员工表staff

因为 SQL 是直接拼接的，如果我们完全信任前端传的参数的话。假如前端传这么一个参数时'' or '1'='1'，SQL 就变成酱紫的啦。

select \* from staff where name='' or '1'='1';

这个 SQL 会把所有的员工信息全都查出来了，酱紫就请求用户已经越权啦。请求者可以获取所有员工的信息，信息已经暴露了啦。

**如何预防 SQL 注入问题**

1). **使用 #{} 而不是 ${}**

在 MyBatis 中，使用#{} 而不是 ${}，可以很大程度防止 sql 注入。

因为**#{} 是一个参数占位符**，对于**字符串类型，会自动加上””**，其他类型不加。**由于 Mybatis 采用预编译，其后的参数不会再进行 SQL 编译，所以一定程度上防止 SQL 注入**。

**${} 是一个简单的字符串替换**，字符串是什么，就会解析成什么，存在 SQL 注入风险

**2). 不要暴露一些不必要的日志或者安全信息，比如避免直接响应一些 sql 异常信息。**

如果 SQL 发生异常了，不要把这些信息暴露响应给用户，可以自定义异常进行响应

**3). 不相信任何外部输入参数，过滤参数中含有的一些数据库关键词关键词**

可以加个参数校验过滤的方法，过滤 union，or 等数据库关键词

**4). 适当的权限控制**

在你查询信息时，先校验下当前用户是否有这个权限。比如说，实现代码的时候，可以让用户多传一个企业 Id 什么的，或者获取当前用户的 session 信息等，在查询前，先校验一下当前用户是否是这个企业下的等等，是的话才有这个查询员工的权限。

1. **什么是 DoS、DDoS、DRDoS 攻击？**

**DOS: (Denial of Service)**, 中文名称是**拒绝服务**，一切能引起 DOS 行为的攻击都被称为 DOS 攻击。最常见的 DoS 攻击有**计算机网络宽带攻击**和**连通性攻击**。

**DDoS**: (Distributed Denial of Service), 中文名称是**分布式拒绝服务**。是指处于不同位置的多个攻击者同时向一个或数个目标发动攻击，或者一个攻击者控制了位于不同位置的多台机器并利用这些机器对受害者同时实施攻击。常见的 DDos 有 **SYN Flood、Ping of Death、ACK Flood、UDP Flood** 等。

**DRDoS**: (Distributed Reflection Denial of Service)，中文名称是**分布式反射拒绝服务**，该方式靠的是**发送大量带有被害者 IP 地址的数据包给攻击主机，然后攻击主机对 IP 地址源做出大量回应，形成拒绝服务攻击。**

1. **WebSocket 与 socket 的区别**

**Socket = IP 地址 + 端口 + 协议。**

具体来说，Socket 是一套标准，它完成了对 TCP/IP 的高度封装，屏蔽网络细节以方便开发者更好地进行网络编程。

**WebSocket 是一个持久化的协议**，它是伴随 HTTP5 而出的协议，用来解决 http 不支持持久化连接的问题。

**Socket 一个是网编编程的标准接口**，而 WebSocket 是应用层通信协议。

1. **ICMP 协议的功能**

**ICMP**,Internet Control Message Protocol ,Internet **控制消息协议**。

ICMP 协议是一种**面向无连接**的协议，用于**传输出错报告控制信息**。

它是一个非常重要的协议，它对于网络安全具有极其重要的意义。它属于**网络层协议**，主要**用于在主机与路由器之间传递控制信息，包括报告错误、交换受限控制和状态信息等**。

当遇到 IP 数据无法访问目标、IP 路由器无法按当前的传输速率转发数据包等情况时，会自动发送 ICMP 消息。

比如我们日常使用得比较多的 ping，就是基于 ICMP 的。

1. **Http 请求的过程与原理**

HTTP 是一个基于 TCP/IP 协议来传递数据的超文本传输协议，传输的数据类型有 HTML 文件，、图片文件等。以访问百度有例子，看下一次 Http 的请求过程



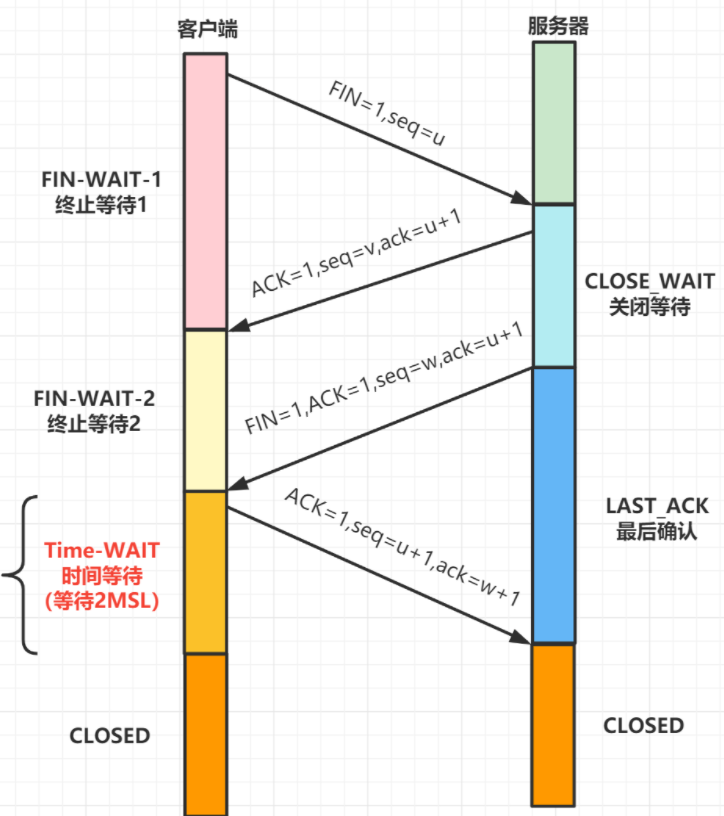
1. 客户端进行 DNS 域名解析，得到对应的 IP 地址
2. 根据这个 IP，找到对应的服务器建立连接（三次握手）
3. 建立 TCP 连接后发起 HTTP 请求（一个完整的 http 请求报文）
4. 服务器响应 HTTP 请求，客户端得到 html 代码
5. 客户端解析 html 代码，用 html 代码中的资源 (如 js,css, 图片等等) 渲染页面。
6. 服务器关闭 TCP 连接（四次挥手）
7. **说下 ping 的原理**

ping，Packet Internet Groper，是一种因特网包探索器，用于测试网络连接量的程序。Ping 是工作在 TCP/IP 网络体系结构中应用层的一个服务命令， 主要是向特定的目的主机发送 ICMP（Internet Control Message Protocol 因特网报文控制协议） 请求报文，测试目的站是否可达及了解其有关状态

一般来说，**ping 可以用来检测网络通不通**。它是基于 ICMP 协议工作的。假设机器 A ping 机器 B，工作过程如下：

1. ping 通知系统，新建一个固定格式的 ICMP 请求数据包
2. ICMP 协议，将该数据包和目标机器 B 的 IP 地址打包，一起转交给 IP 协议层
3. IP 层协议将本机 IP 地址为源地址，机器 B 的 IP 地址为目标地址，加上一些其他的控制信息，构建一个 IP 数据包
4. 先获取目标机器 B 的 MAC 地址。
5. 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
6. 机器 B 收到后，对比目标地址，和自己本机的 MAC 地址是否一致，符合就处理返回，不符合就丢弃。
7. 根据目的主机返回的 ICMP 回送回答报文中的时间戳，从而计算出往返时间
8. 最终显示结果有这几项：发送到目的主机的 IP 地址、发送 & 收到 & 丢失的分组数、往返时间的最小、最大 & 平均值
9. **如果服务器出现了大量 CLOSE\_WAIT 状态如何解决。**

我们先来回忆下 TCP 的四次挥手



服务器端收到客户端发送的 FIN 后，TCP 协议栈就会自动发送 ACK，接着进入 CLOSE\_WAIT 状态。

但是如果服务器端不执行 socket 的 close () 操作，那么就没法进入 LAST\_ACK, 导致大量连接处于 CLOSE\_WAIT 状态

所以，如果服务器出现了大量 CLOSE\_WAIT 状态，一般是程序 Bug，或者关闭 socket 不及时。

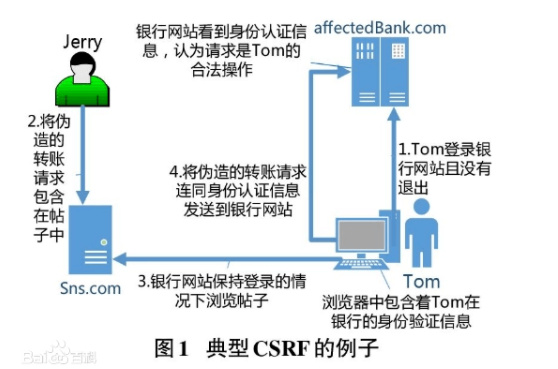
1. **什么是 CSRF 攻击，如何避免**

**什么是 CSRF 攻击？**

CSRF，**跨站请求伪造**（英语：Cross-site request forgery），简单点说就是，攻击者盗用了你的身份，以你的名义发送恶意请求。跟**跨网站脚本**（XSS）相比，**XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户网页浏览器的信任。**

**CSRF 是如何攻击的呢？**

我们来看下这个例子哈（来自百度百科）



1. Tom 登陆银行，没有退出，浏览器包含了 Tom 在银行的身份认证信息。
2. 黑客 Jerry 将伪造的转账请求，包含在在帖子
3. Tom 在银行网站保持登陆的情况下，浏览帖子
4. 将伪造的转账请求连同身份认证信息，发送到银行网站
5. 银行网站看到身份认证信息，以为就是 Tom 的合法操作，最后造成 Tom 资金损失。

**如何解决 CSRF 攻击**

1. 检查 Referer 字段。HTTP 头中有一个 Referer 字段，这个字段用以标明请求来源于哪个地址。
2. 添加校验 token。
3. **RARP 协议的工作原理？**
4. **ARP (地址解析协议)** , 是设备通过自己知道的 IP 地址来获得自己不知道的物理地址的协议。
5. **RARP (反向地址转换协议)** 以与 ARP 相反的方式工作。RARP 发出要反向解析的物理地址并希望返回其对应的 IP 地址，应答包括由能够提供所需信息的 RARP 服务器发出的 IP 地址。（应用于无盘机）

**RARP 工作原理如下：**

1. 发送主机发送一个本地的 RARP 广播，在此广播包中，声明自己的 MAC 地址并且请求任何收到此请求的 RARP 服务器分配一个 IP 地址；
2. 本地网段上的 RARP 服务器收到此请求后，检查其 RARP 列表，查找该 MAC 地址对应的 IP 地址；
3. 如果存在，RARP 服务器就给源主机发送一个响应数据包并将此 IP 地址提供给对方主机使用；
4. 如果不存在，RARP 服务器对此不做任何的响应；
5. 源主机收到从 RARP 服务器的响应信息，就利用得到的 IP 地址进行通讯；如果一直没有收到 RARP 服务器的响应信息，表示初始化失败。
6. **了解下 DNS，解析过程？**

**DNS，domain name system，域名解析系统**，是**因特网上作为域名和 IP 地址相互映射的一个分布式数据库**。它的作用非常简单，就是可以**根据域名查出对应的 IP 地址**。

**解析过程如下：**

1. 首先，检查浏览器缓存中，查找对应的 IP 地址，找到就直接返回；否则下一步。
2. 将请求发送给本地 DNS 服务器，在本地 DNS 服务器缓存中查询，如果查找到就直接返回，否则下一步；
3. 本地 DNS 服务器向根域名服务器发送请求，根域名服务器会告诉本地 DNS 服务器去查询哪个顶级域名服务器。
4. 本地域名服务器向顶级域名服务器发起查询请求，顶级域名服务器会告诉本地 DNS 服务器，去查找哪个权限域名服务器。
5. 本地域名服务器向权限域名服务器发起查询请求，权限域名服务器告诉本地域名服务器请求域名所对应的 IP 地址。
6. 最后，本地域名服务器告诉主机请求域名所对应的 IP 地址。

**比如要查询 www.baidu.com 的 IP 地址:**

1. 首先会在浏览器的缓存中，是否查找到 www.baidu.com 的对应的 IP，找到就直接返回；否则下一步。
2. 将请求发送给本地 DNS 服务器，在**本地 DNS 服务器缓存**中查询，如果查找到就直接返回，否则下一步；
3. 本地 DNS 服务器向**根域名服务器**发送请求，根域名服务器返回负责.com 的顶级域名服务器的 IP 地址的列表。
4. 本地 DNS 服务器再向其中一个负责 .com 的**顶级域名服务器**发送一个请求，返回负责 .baidu 的权威域名服务器的 IP 地址列表。
5. 本地 DNS 服务器再向其中一个**权威域名服务器**发送一个请求，返回 www.baidu.com 所对应的 IP 地址。

# 网络分层结构

计算机网络体系大致分为三种，OSI七层模型、TCP/IP四层模型和五层模型。一般面试的时候考察比较多的是五层模型。

TCP/IP五层模型：应用层、传输层、网络层、数据链路层、物理层。

* **应用层**：为应用程序提供交互服务。在互联网中的应用层协议很多，如域名系统DNS、HTTP协议、SMTP协议等。
* **传输层**：负责向两台主机进程之间的通信提供数据传输服务。传输层的协议主要有传输控制协议TCP和用户数据协议UDP。
* **网络层**：选择合适的路由和交换结点，确保数据及时传送。主要包括IP协议。
* **数据链路层**：在两个相邻节点之间传送数据时，**数据链路层将网络层交下来的 IP 数据报组装成帧**，在两个相邻节点间的链路上传送帧。
* **物理层**：实现相邻节点间比特流的透明传输，尽可能屏蔽传输介质和物理设备的差异。

# TCP有哪些特点？

* TCP是**面向连接**的运输层协议。
* **点对点**，每一条TCP连接只能有两个端点。
* TCP提供**可靠交付**的服务。
* TCP提供**全双工通信**。
* **面向字节流**。

# TCP和UDP的区别？

1. TCP**面向连接**；UDP是无连接的，即发送数据之前不需要建立连接。
2. TCP提供**可靠的服务**；UDP不保证可靠交付。
3. TCP**面向字节流**，把数据看成一连串无结构的字节流；UDP是面向报文的。
4. TCP有**拥塞控制**；UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如实时视频会议等）。
5. 每一条TCP连接只能是**点到点**的；UDP支持一对一、一对多、多对一和多对多的通信方式。
6. TCP首部开销20字节；UDP的首部开销小，只有8个字节。

# HTTP协议的特点？

1. HTTP允许传输**任意类型**的数据。传输的类型由Content-Type加以标记。
2. **无状态**。对于客户端每次发送的请求，服务器都认为是一个新的请求，上一次会话和下一次会话之间没有联系。
3. 支持**客户端/服务器模式**。

# HTTP报文格式

HTTP请求由**请求行、请求头部、空行和请求体**四个部分组成。

* **请求行**：包括请求方法，访问的资源URL，使用的HTTP版本。GET和POST是最常见的HTTP方法，除此以外还包括DELETE、HEAD、OPTIONS、PUT、TRACE。
* **请求头**：格式为“属性名:属性值”，服务端根据请求头获取客户端的信息，主要有cookie、host、connection、accept-language、accept-encoding、user-agent。
* **请求体**：用户的请求数据如用户名，密码等。

**请求报文示例**：

POST /xxx HTTP/1.1 请求行  
Accept:image/gif.image/jpeg, 请求头部  
Accept-Language:zh-cn  
Connection:Keep-Alive  
Host:localhost  
User-Agent:Mozila/4.0(compatible;MSIE5.01;Window NT5.0)  
Accept-Encoding:gzip,deflate  
  
username=dabin 请求体

**HTTP响应**也由四个部分组成，分别是：**状态行、响应头、空行和响应体**。

* **状态行**：协议版本，状态码及状态描述。
* **响应头**：响应头字段主要有connection、content-type、content-encoding、content-length、set-cookie、Last-Modified，、Cache-Control、Expires。
* **响应体**：服务器返回给客户端的内容。

**响应报文示例**：

HTTP/1.1 200 OK  
Server:Apache Tomcat/5.0.12  
Date:Mon,6Oct2003 13:23:42 GMT  
Content-Length:112  
  
<html>  
    <body>响应体</body>  
</html>

# HTTP1.0和HTTP1.1的区别?

* **长连接**：HTTP1.0默认使用短连接，每次请求都需要建立新的TCP连接，连接不能复用。**HTTP1.1支持长连接，复用TCP连接，允许客户端通过同一连接发送多个请求**。不过，这个优化策略也存在问题，当一个队头的请求不能收到响应的资源时，它将会阻塞后面的请求。这就是“**队头阻塞**”问题。
* **断点续传**：HTTP1.0 **不支持断点续传**。HTTP1.1 新增了 **range** 字段，用来指定数据字节位置，**支持断点续传**。
* **错误状态响应码**：在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突、410（Gone）表示服务器上的某个资源被永久性的删除。
* **Host头处理**：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。到了HTTP1.1时代，虚拟主机技术发展迅速，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，故HTTP1.1增加了HOST信息。

# HTTP1.1和 HTTP2.0的区别？

HTTP2.0相比HTTP1.1支持的特性：

* **新的二进制格式**：HTTP1.1 基于文本格式传输数据；HTTP2.0采用二进制格式传输数据，解析更高效。
* **多路复用**：在一个连接里，允许同时发送多个请求或响应，**并且这些请求或响应能够并行的传输而不被阻塞**，避免 HTTP1.1 出现的”队头堵塞”问题。
* **头部压缩**，HTTP1.1的header带有大量信息，而且每次都要重复发送；HTTP2.0 把header从数据中分离，并封装成头帧和数据帧，**使用特定算法压缩头帧**，有效减少头信息大小。并且HTTP2.0**在客户端和服务器端记录了之前发送的键值对，对于相同的数据，不会重复发送。比如请求a发送了所有的头信息字段，请求b则只需要发送差异数据**，这样可以减少冗余数据，降低开销。
* **服务端推送**：HTTP2.0允许服务器向客户端推送资源，无需客户端发送请求到服务器获取。

# HTTPS与HTTP的区别？

1. HTTP是超文本传输协议，信息是**明文传输**；HTTPS则是具有**安全性**的ssl加密传输协议。
2. HTTP和HTTPS用的端口不一样，HTTP端口是80，HTTPS是443。
3. HTTPS协议**需要到CA机构申请证书**，一般需要一定的费用。
4. HTTP运行在TCP协议之上；HTTPS运行在SSL协议之上，SSL运行在TCP协议之上。

# 什么是数字证书？

服务端可以向证书颁发机构CA申请证书，以避免中间人攻击（防止证书被篡改）。证书包含三部分内容：**证书内容、证书签名算法和签名**，签名是为了验证身份。



服务端把证书传输给浏览器，浏览器从证书里取公钥。证书可以证明该公钥对应本网站。

**数字签名的制作过程**：

1. CA使用证书签名算法对证书内容进行**hash运算**。
2. 对hash后的值**用CA的私钥加密**，得到数字签名。

**浏览器验证过程**：

1. 获取证书，得到证书内容、证书签名算法和数字签名。
2. 用CA机构的公钥**对数字签名解密**（由于是浏览器信任的机构，所以浏览器会保存它的公钥）。
3. 用证书里的签名算法**对证书内容进行hash运算**。
4. 比较解密后的数字签名和对证书内容做hash运算后得到的哈希值，相等则表明证书可信。

# HTTPS原理

首先是TCP三次握手，然后客户端发起一个HTTPS连接建立请求，客户端先发一个Client Hello的包，然后服务端响应Server Hello，接着再给客户端发送它的证书，然后双方经过密钥交换，最后使用交换的密钥加解密数据。

1. **协商加密算法** 。在Client Hello里面客户端会告知服务端自己当前的一些信息，包括客户端要使用的TLS版本，支持的加密算法，要访问的域名，给服务端生成的一个随机数（Nonce）等。需要提前告知服务器想要访问的域名以便服务器发送相应的域名的证书过来。
2. 服务端响应Server Hello，告诉客户端服务端**选中的加密算法**。
3. 接着服务端给客户端发来了2个证书。第二个证书是第一个证书的签发机构（CA）的证书。
4. 客户端使用证书的认证机构CA公开发布的RSA公钥**对该证书进行验证**，下图表明证书认证成功。
5. 验证通过之后，浏览器和服务器通过**密钥交换算法**产生共享的**对称密钥**。
6. 开始传输数据，使用同一个对称密钥来加解密。

# DNS 的解析过程？

1. 浏览器搜索**自己的DNS缓存**
2. 若没有，则搜索**操作系统中的DNS缓存和hosts文件**
3. 若没有，则操作系统将域名发送至**本地域名服务器**，本地域名服务器查询自己的DNS缓存，查找成功则返回结果，否则依次向**根域名服务器、顶级域名服务器、权限域名服务器**发起查询请求，最终返回IP地址给本地域名服务器
4. 本地域名服务器将得到的IP地址返回给**操作系统**，同时自己也**将IP地址缓存起来**
5. 操作系统将 IP 地址返回给浏览器，同时自己也将IP地址缓存起来
6. 浏览器得到域名对应的IP地址

# 浏览器中输入URL返回页面过程？

1. **解析域名**，找到主机 IP。
2. 浏览器利用 IP 直接与网站主机通信，**三次握手**，建立 TCP 连接。浏览器会以一个随机端口向服务端的 web 程序 80 端口发起 TCP 的连接。
3. 建立 TCP 连接后，浏览器向主机发起一个HTTP请求。
4. 服务器**响应请求**，返回响应数据。
5. 浏览器**解析响应内容，进行渲染**，呈现给用户。

# Cookie和Session的区别？

* **作用范围不同**，Cookie 保存在客户端，Session 保存在服务器端。
* **有效期不同**，Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般失效时间较短，客户端关闭或者 Session 超时都会失效。
* **隐私策略不同**，Cookie 存储在客户端，容易被窃取；Session 存储在服务端，安全性相对 Cookie 要好一些。
* **存储大小不同**， 单个 Cookie 保存的数据不能超过 4K；对于 Session 来说存储没有上限，但出于对服务器的性能考虑，Session 内不要存放过多的数据，并且需要设置 Session 删除机制。

# 什么是对称加密和非对称加密？

**对称加密**：通信双方使用**相同的密钥**进行加密。特点是加密速度快，但是缺点是密钥泄露会导致密文数据被破解。常见的对称加密有AES和DES算法。

**非对称加密**：它需要生成两个密钥，**公钥和私钥**。公钥是公开的，任何人都可以获得，而私钥是私人保管的。公钥负责加密，私钥负责解密；或者私钥负责加密，公钥负责解密。这种加密算法**安全性更高**，但是**计算量相比对称加密大很多**，加密和解密都很慢。常见的非对称算法有RSA和DSA。

### 1：socket大致介绍

      socket编程是一门技术，它主要是在网络通信中经常用到

      既然是一门技术，由于现在是面向对象的编程，一些计算机行业的大神通过抽象的理念，在现实中通过反复的理论或者实际的推导，提出了抽象的一些通信协议，基于tcp/ip协议，提出大致的构想，一些泛型的程序大牛在这个协议的基础上，将这些抽象化的理念接口化，针对协议提出的每个理念，专门的编写制定的接口，与其协议一一对应，形成了现在的socket标准规范，然后将其接口封装成可以调用的接口，供开发者使用

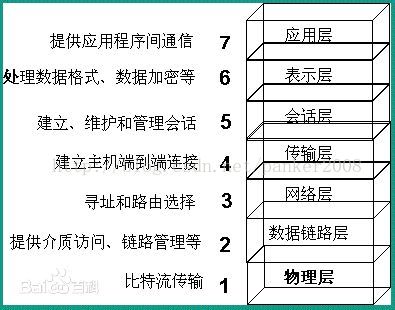
     目前，开发者开发出了很多封装的类来完善socket编程，都是更加方便的实现刚开始socket通信的各个环节，所以我们首先必须了解socket的通信原理，只有从本质上理解socket的通信，才可能快速方便的理解socket的各个环节，才能从底层上真正的把握

### 2：TCP/IP协议

     要理解socket必须的得理解tcp/ip，它们之间好比送信的线路和驿站的作用，比如要建议送信驿站，必须得了解送信的各个细节。

     TCP/IP协议不同于iso的7个分层，它是根据这7个分层，将其重新划分，好比打扫卫生，本来有扫帚，垃圾斗，抹布，涂料，盆栽等就好比OSI的标准几个分层，tcp/ip根据用途和功能，将扫帚，垃圾斗放到粗略整理层，抹布涂料放到中度整理层，盆栽放到最终效果层。这里TCP/IP也对OSI的网络模型层进行了划分：大致如下：

OSI模型：



TCP/IP协议参考模型把所有的TCP/IP系列协议归类到四个抽象层中

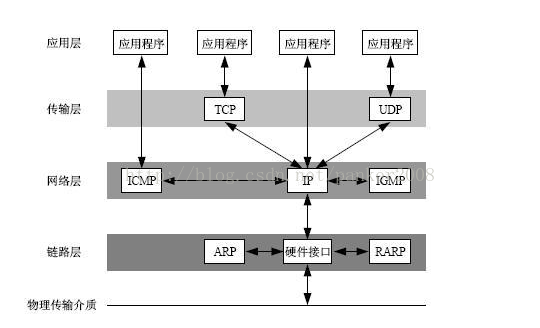
应用层：TFTP，HTTP，SNMP，FTP，SMTP，DNS，Telnet 等等

传输层：TCP，UDP

网络层：IP，ICMP，OSPF，EIGRP，IGMP

数据链路层：SLIP，CSLIP，PPP，MTU

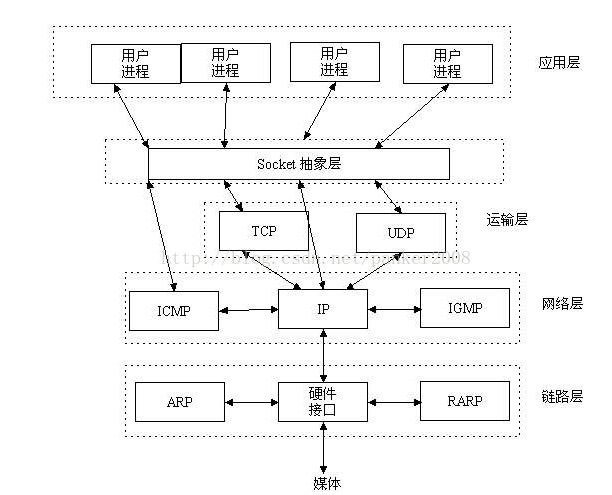
每一抽象层建立在低一层提供的服务上，并且为高一层提供服务，看起来大概是这样子的



通过上面的图形，由于底一层的需要向高一层的提供服务，我们大致的理解应用程序需要传输层的tcp和网络层的ip协议提供服务，但是我们这章要分析的socket它是在tcpip协议的那一部分呢，就好比，我们的通讯线路已经有明确的规定，我们的驿站要设计在哪个地方一样

### 3：回过头再来理解socket

     到目前为止，大致的了解了应用程序和tcpip协议的大致关系，我们只是知道socket编程是在tcp/IP上的网络编程，但是socket在上述的模型的什么位置呢。这个位置被一个天才的理论家或者是抽象的计算机大神提出并且安排出来



我们可以发现socket就在应用程序的传输层和应用层之间，设计了一个socket抽象层，传输层的底一层的服务提供给socket抽象层，socket抽象层再提供给应用层，问题又来了，应用层和socket抽象层之间和传输层，网络层之间如何通讯的呢，了解这个之前，我们还是回到原点

   要想理解socket编程怎么通过socket关键词实现服务器和客户端通讯，必须得实现的了解tcp/ip是怎么通讯的，在这个的基础上在去理解socket的握手通讯

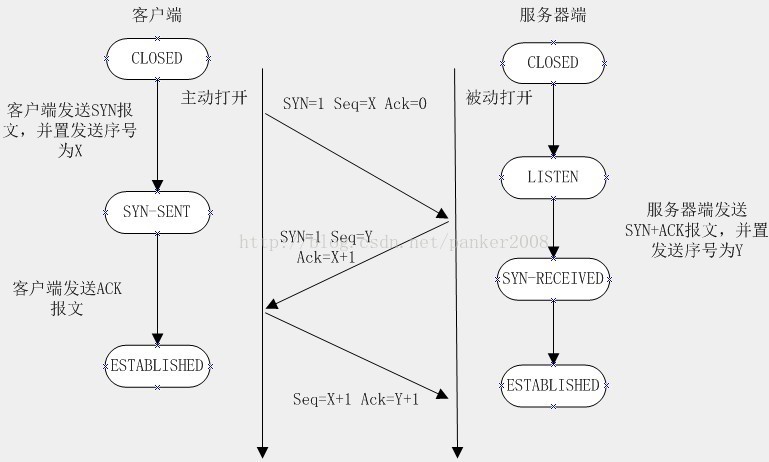
    在tcp/ip协议中，tcp通过三次握手建立起一个tcp的链接，大致如下

     第一次握手：客户端尝试连接服务器，向服务器发送syn包，syn=j，客户端进入SYN\_SEND状态等待服务器确认

    第二次握手：服务器接收客户端syn包并确认（ack=j+1），同时向客户端发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态

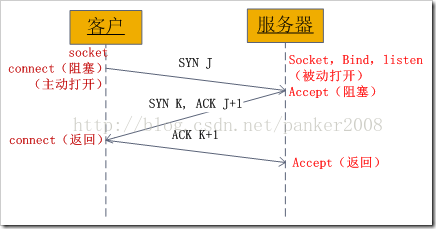
   第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手

    三次握手如下图：



     根据tcp的三次握手，socket也定义了三次握手，也许是参考tcp的三次握手，一些计算机大神们画出了socket的三次握手的模型图

     模型图如下：



     在上面图的基础上，如果我们得到上面的图形，需要我们自己开发一些接口，来满足上面的通讯的三次握手，问题就出来了，我们会需要开发哪些函数

### 4：socket的一些接口函数原理

      通过上面的图，我们清楚，我们好比一些泛型的程序员，一些理论提供者提供给了我们上面的图形的理论，我们需要做的就是讲上面的图形的抽象化的东西具体化

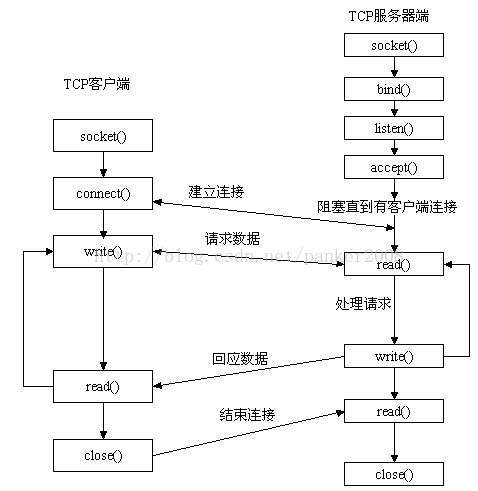
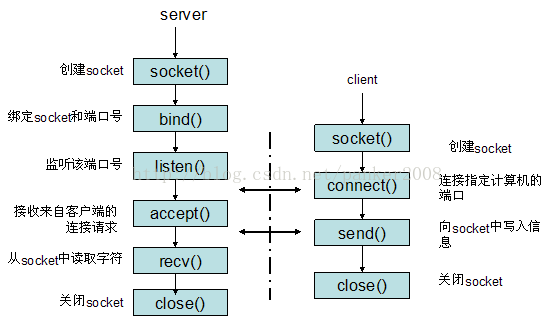
      第一次握手：客户端需要发送一个syn j 包，试着去链接服务器端，于是客户端我们需要提供一个链接函数

     第二次握手：服务器端需要接收客户端发送过来的syn J+1 包，然后在发送ack包，所以我们需要有服务器端接受处理函数

     第三次握手：客户端的处理函数和服务器端的处理函数

     三次握手只是一个数据传输的过程，但是，我们传输前需要一些准备工作，比如将创建一个套接字，收集一些计算机的资源，将一些资源绑定套接字里面，以及接受和发送数据的函数等等，这些功能接口在一起构成了socket的编程

**下面大致的按照客户端和服务端将所需的函数详细的列举出来**



上面的两个图都概述了socket的通讯原理

**服务器端：**

其过程是首先服务器方要先启动，并根据请求提供相应服务：

（1）打开一通信通道并告知本地主机，它愿意在某一公认地址上的某端口（如FTP的端口可能为21）接收客户请求；

（2）等待客户请求到达该端口；

（3）接收到客户端的服务请求时，处理该请求并发送应答信号。接收到并发服务请求，要激活一新进程来处理这个客户请求（如UNIX系统中用fork、exec）。新进程处理此客户请求，并不需要对其它请求作出应答。服务完成后，关闭此新进程与客户的通信链路，并终止。

（4）返回第（2）步，等待另一客户请求。

（5）关闭服务器

**客户端：**

（1）打开一通信通道，并连接到服务器所在主机的特定端口；

（2）向服务器发服务请求报文，等待并接收应答；继续提出请求......

（3）请求结束后关闭通信通道并终止。

**从上面所描述过程可知：**

（1）客户与服务器进程的作用是非对称的，因此代码不同。

（2）服务器进程一般是先启动的。只要系统运行，该服务进程一直存在，直到正常或强迫终止。

### 7：下面就介绍一些API函数：（摘抄carter大神文章）：

**创建套接字──socket()**

应用程序在使用套接字前，首先必须拥有一个套接字，系统调用socket()向应用程序提供创建套接字的手段，其调用格式如下：

1. SOCKET PASCAL FAR socket(**int** af, **int** type, **int** protocol)

该调用要接收三个参数：af、type、protocol。参数af指定通信发生的区域：AF\_UNIX、AF\_INET、AF\_NS等，而DOS、WINDOWS中仅支持AF\_INET，它是网际网区域。因此，地址族与协议族相同。参数type 描述要建立的套接字的类型。这里分三种：

（1）一是TCP流式套接字(SOCK\_STREAM)提供了一个面向连接、可靠的数据传输服务，数据无差错、无重复地发送，且按发送顺序接收。内设流量控制，避免数据流超限；数据被看作是字节流，无长度限制。文件传送协议（FTP）即使用流式套接字。

（2）二是数据报式套接字(SOCK\_DGRAM)提供了一个无连接服务。数据包以独立包形式被发送，不提供无错保证，数据可能丢失或重复，并且接收顺序混乱。网络文件系统（NFS）使用数据报式套接字。

（3）三是原始式套接字(SOCK\_RAW)该接口允许对较低层协议，如IP、ICMP直接访问。常用于检验新的协议实现或访问现有服务中配置的新设备。

参数protocol说明该套接字使用的特定协议，如果调用者不希望特别指定使用的协议，则置为0，使用默认的连接模式。根据这三个参数建立一个套接字，并将相应的资源分配给它，同时返回一个整型套接字号。因此，socket()系统调用实际上指定了相关五元组中的“协议”这一元。

**指定本地地址──bind()**

当一个套接字用socket()创建后，存在一个名字空间(地址族)，但它没有被命名。bind()将套接字地址（包括本地主机地址和本地端口地址）与所创建的套接字号联系起来，即将名字赋予套接字，以指定本地半相关。其调用格式如下：

1. **int** PASCAL FAR bind(SOCKET s, **const** **struct** sockaddr FAR \* name, **int** namelen);

参数s是由socket()调用返回的并且未作连接的套接字描述符(套接字号)。参数name 是赋给套接字s的本地地址（名字），其长度可变，结构随通信域的不同而不同。namelen表明了name的长度。如果没有错误发生，bind()返回0。否则返回SOCKET\_ERROR。

**建立套接字连接──connect()与accept()**

这两个系统调用用于完成一个完整相关的建立，其中connect()用于建立连接。accept()用于使服务器等待来自某客户进程的实际连接。

connect()的调用格式如下：

1. **int** PASCAL FAR connect(SOCKET s, **const** **struct** sockaddr FAR \* name, **int** namelen);

参数s是欲建立连接的本地套接字描述符。参数name指出说明对方套接字地址结构的指针。对方套接字地址长度由namelen说明。

如果没有错误发生，connect()返回0。否则返回值SOCKET\_ERROR。在面向连接的协议中，该调用导致本地系统和外部系统之间连接实际建立。

由于地址族总被包含在套接字地址结构的前两个字节中，并通过socket()调用与某个协议族相关。因此bind()和connect()无须协议作为参数。

accept()的调用格式如下：

1. SOCKET PASCAL FAR accept(SOCKET s, **struct** sockaddr FAR\* addr, **int** FAR\* addrlen);

参数s为本地套接字描述符，在用做accept()调用的参数前应该先调用过listen()。addr 指向客户方套接字地址结构的指针，用来接收连接实体的地址。addr的确切格式由套接字创建时建立的地址族决定。addrlen 为客户方套接字地址的长度（字节数）。如果没有错误发生，accept()返回一个SOCKET类型的值，表示接收到的套接字的描述符。否则返回值INVALID\_SOCKET。

accept()用于面向连接服务器。参数addr和addrlen存放客户方的地址信息。调用前，参数addr 指向一个初始值为空的地址结构，而addrlen 的初始值为0；调用accept()后，服务器等待从编号为s的套接字上接受客户连接请求，而连接请求是由客户方的connect()调用发出的。当有连接请求到达时，accept()调用将请求连接队列上的第一个客户方套接字地址及长度放入addr 和addrlen，并创建一个与s有相同特性的新套接字号。新的套接字可用于处理服务器并发请求。

四个套接字系统调用，socket()、bind()、connect()、accept()，可以完成一个完全五元相关的建立。socket()指定五元组中的协议元，它的用法与是否为客户或服务器、是否面向连接无关。bind()指定五元组中的本地二元，即本地主机地址和端口号，其用法与是否面向连接有关：在服务器方，无论是否面向连接，均要调用bind()，若采用面向连接，则可以不调用bind()，而通过connect()自动完成。若采用无连接，客户方必须使用bind()以获得一个唯一的地址。

**监听连接──listen()**

此调用用于面向连接服务器，表明它愿意接收连接。listen()需在accept()之前调用，其调用格式如下：

1. **int** PASCAL FAR listen(SOCKET s, **int** backlog);

参数s标识一个本地已建立、尚未连接的套接字号，服务器愿意从它上面接收请求。backlog表示请求连接队列的最大长度，用于限制排队请求的个数，目前允许的最大值为5。如果没有错误发生，listen()返回0。否则它返回SOCKET\_ERROR。

listen()在执行调用过程中可为没有调用过bind()的套接字s完成所必须的连接，并建立长度为backlog的请求连接队列。

调用listen()是服务器接收一个连接请求的四个步骤中的第三步。它在调用socket()分配一个流套接字，且调用bind()给s赋于一个名字之后调用，而且一定要在accept()之前调用。

**数据传输──send()与recv()**

当一个连接建立以后，就可以传输数据了。常用的系统调用有send()和recv()。

send()调用用于s指定的已连接的数据报或流套接字上发送输出数据，格式如下：

1. **int** PASCAL FAR send(SOCKET s, **const** **char** FAR \*buf, **int** len, **int** flags);

参数s为已连接的本地套接字描述符。buf 指向存有发送数据的缓冲区的指针，其长度由len 指定。flags 指定传输控制方式，如是否发送带外数据等。如果没有错误发生，send()返回总共发送的字节数。否则它返回SOCKET\_ERROR。

recv()调用用于s指定的已连接的数据报或流套接字上接收输入数据，格式如下：

1. **int** PASCAL FAR recv(SOCKET s, **char** FAR \*buf, **int** len, **int** flags);

参数s 为已连接的套接字描述符。buf指向接收输入数据缓冲区的指针，其长度由len 指定。flags 指定传输控制方式，如是否接收带外数据等。如果没有错误发生，recv()返回总共接收的字节数。如果连接被关闭，返回0。否则它返回SOCKET\_ERROR。

**输入/输出多路复用──select()**

select()调用用来检测一个或多个套接字的状态。对每一个套接字来说，这个调用可以请求读、写或错误状态方面的信息。请求给定状态的套接字集合由一个fd\_set结构指示。在返回时，此结构被更新，以反映那些满足特定条件的套接字的子集，同时， select()调用返回满足条件的套接字的数目，其调用格式如下：

1. **int** PASCAL FAR select(**int** nfds, fd\_set FAR \* readfds, fd\_set FAR \* writefds, fd\_set FAR \* exceptfds, **const** **struct** timeval FAR \* timeout);

参数nfds指明被检查的套接字描述符的值域，此变量一般被忽略。

参数readfds指向要做读检测的套接字描述符集合的指针，调用者希望从中读取数据。参数writefds 指向要做写检测的套接字描述符集合的指针。exceptfds指向要检测是否出错的套接字描述符集合的指针。timeout指向select()函数等待的最大时间，如果设为NULL则为阻塞操作。select()返回包含在fd\_set结构中已准备好的套接字描述符的总数目，或者是发生错误则返回SOCKET\_ERROR。

**关闭套接字──closesocket()**

closesocket()关闭套接字s，并释放分配给该套接字的资源；如果s涉及一个打开的TCP连接，则该连接被释放。closesocket()的调用格式如下：

1. **BOOL** PASCAL FAR closesocket(SOCKET s);

参数s待关闭的套接字描述符。如果没有错误发生，closesocket()返回0。否则返回值SOCKET\_ERROR。