**cookie和session异同**

**1、存储位置不同**

cookie的数据信息存放在客户端浏览器上。

session的数据信息存放在服务器上。

**2、存储容量不同**

单个cookie保存的数据<=4KB，一个站点最多保存20个Cookie。

对于session来说并没有上限，但出于对服务器端的性能考虑，session内不要存放过多的东西，并且设置session删除机制。

**3、存储方式不同**

cookie中只能保管ASCII字符串，并需要通过编码方式存储为Unicode字符或者二进制数据。

session中能够存储任何类型的数据，包括且不限于string，integer，list，map等。

**4、隐私策略不同**

cookie对客户端是可见的，别有用心的人可以分析存放在本地的cookie并进行cookie欺骗，所以它是不安全的。

session存储在服务器上，对客户端是透明对，不存在敏感信息泄漏的风险。

**5、有效期上不同**

开发可以通过设置cookie的属性，达到使cookie长期有效的效果。

session依赖于名为JSESSIONID的cookie，而cookie JSESSIONID的过期时间默认为-1，只需关闭窗口该session就会失效，因而session不能达到长期有效的效果。

Tomcat中Session的默认失效时间为30分钟。

调用Session的invalidate方法。

**6、服务器压力不同**

cookie保管在客户端，不占用服务器资源。对于并发用户十分多的网站，cookie是很好的选择。

session是保管在服务器端的，每个用户都会产生一个session。假如并发访问的用户十分多，会产生十分多的session，耗费大量的内存。

**7、浏览器支持不同**

假如客户端浏览器不支持cookie：

cookie是需要客户端浏览器支持的，假如客户端禁用了cookie，或者不支持cookie，则会话跟踪会失效。关于WAP上的应用，常规的cookie就派不上用场了。

运用session需要使用URL地址重写的方式。一切用到session程序的URL都要进行URL地址重写，否则session会话跟踪还会失效。

假如客户端支持cookie：

cookie既能够设为本浏览器窗口以及子窗口内有效，也能够设为一切窗口内有效。

session只能在本窗口以及子窗口内有效。

**8、跨域支持上不同**

cookie支持跨域名访问。

session不支持跨域名访问。

**HTTPS和HTTP的区别**

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。ssl证书阿里云上可以免费申请一年

2、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。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；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。

以下是具体一些分析

**一、HTTP和HTTPS的基本概念**

**HTTP**：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

**HTTPS**：是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。

**HTTPS协议的主要作用可以分为两种**：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

二 **、**HTTP和HTTPS的主要特点和工作流程

**HTTP特点**

1.支持客户/服务器模式。（C/S模式）

2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5.无状态：HTTP协议是**无状态协议**。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快

**HTTP工作流程**

第一步：建立TCP/IP连接，客户端与服务器通过Socket三次握手进行连接

第二步：客户端向服务端发起HTTP请求（例如：POST/login.html http/1.1）

第三步：客户端发送请求头信息，请求内容，最后会发送一空白行，标示客户端请求完毕

第四步：服务器做出应答，表示对于客户端请求的应答，例如：HTTP/1.1 200 OK

第五步：服务器向客户端发送应答头信息

第六步：服务器向客户端发送请求头信息后，也会发送一空白行，标示应答头信息发送完毕，接着就以Content-type要求的数据格式发送数据给客户端

第七步：服务端关闭TCP连接，如果服务器或者客户端增Connection:keep-alive就表示客户端与服务器端继续保存连接，在下次请求时可以继续使用这次的连接

**HTTPS特点**

HTTPS是HTTP协议的修改，它加密数据并确保其机密性。其配置可保护用户在与网站交互时免于窃取个人信息和计费数据。

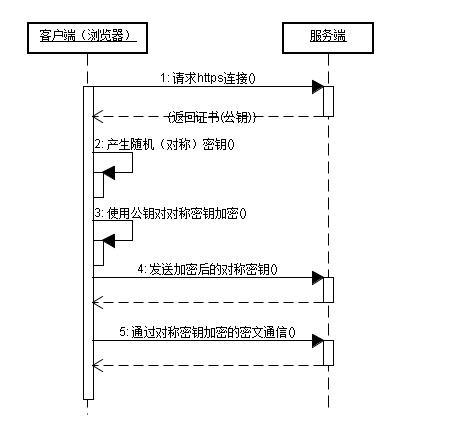
1、优点

相比于http，https可以提供更加优质保密的信息，保证了用户数据的安全性，此外https同时也一定程度上保护了服务端，使用恶意攻击和伪装数据的成本大大提高。

2、缺点

缺点也同样很明显，第一https的技术门槛较高，多数个人或者私人网站难以支撑，CA机构颁发的证书都是需要年费的，此外对接Https协议也需要额外的技术支持；其二，目前来说大多数网站并不关心数据的安全性和保密性，其https最大的优点对它来说并不适用；其三，https加重了服务端的负担，相比于http其需要更多的资源来支撑，同时也降低了用户的访问速度；第四，目前来说Http网站仍然大规模使用，在浏览器侧也没有特别大的差别，很多用户不关心的话根本不感知。

**HTTPS工作流程**



第一步：客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

第二步：Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

第三步：客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

第四步：客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

第五步：Web服务器利用自己的私钥解密出会话密钥。

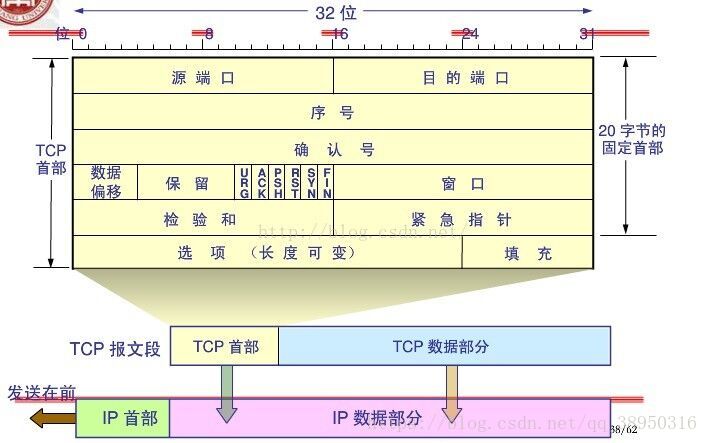
第六步：Web服务器利用会话密钥加密与客户端之间的通信。

ssl证书阿里云上可以免费申请一年

TCP与UDP基本区别  
  1.基于连接与无连接  
  2.TCP要求系统资源较多，UDP较少；   
  3.UDP程序结构较简单   
  4.流模式（TCP）与数据报模式(UDP);   
  5.TCP保证数据正确性，UDP可能丢包   
  6.TCP保证数据顺序，UDP不保证   
　　  
UDP应用场景：  
  1.面向数据报方式  
  2.网络数据大多为短消息   
  3.拥有大量Client  
  4.对数据安全性无特殊要求  
  5.网络负担非常重，但对响应速度要求高  
   
具体编程时的区别  
   1.socket()的参数不同   
　　 2.UDP Server不需要调用listen和accept   
　　 3.UDP收发数据用sendto/recvfrom函数   
　　 4.TCP：地址信息在connect/accept时确定   
　　 5.UDP：在sendto/recvfrom函数中每次均 需指定地址信息   
　　 6.UDP：shutdown函数无效  
   
编程区别  
   通常我们在说到网络编程时默认是指TCP编程，即用前面提到的socket函数创建一个socket用于TCP通讯，函数参数我们通常填为SOCK\_STREAM。即socket(PF\_INET, SOCK\_STREAM, 0)，这表示建立一个socket用于流式网络通讯。   
　  SOCK\_STREAM这种的特点是面向连接的，即每次收发数据之前必须通过connect建立连接，也是双向的，即任何一方都可以收发数据，协议本身提供了一些保障机制保证它是可靠的、有序的，即每个包按照发送的顺序到达接收方。   
  
　　而SOCK\_DGRAM这种是User Datagram Protocol协议的网络通讯，它是无连接的，不可靠的，因为通讯双方发送数据后不知道对方是否已经收到数据，是否正常收到数据。任何一方建立一个socket以后就可以用sendto发送数据，也可以用recvfrom接收数据。根本不关心对方是否存在，是否发送了数据。它的特点是通讯速度比较快。大家都知道TCP是要经过三次握手的，而UDP没有。   
  
基于上述不同，UDP和TCP编程步骤也有些不同，如下：

**TCP:**  
TCP编程的服务器端一般步骤是：   
　　1、创建一个socket，用函数socket()；   
　　2、设置socket属性，用函数setsockopt(); \* 可选   
　　3、绑定IP地址、端口等信息到socket上，用函数bind();   
　　4、开启监听，用函数listen()；   
　　5、接收客户端上来的连接，用函数accept()；   
　　6、收发数据，用函数send()和recv()，或者read()和write();   
　　7、关闭网络连接；   
　　8、关闭监听；   
  
TCP编程的客户端一般步骤是：   
　　1、创建一个socket，用函数socket()；   
　　2、设置socket属性，用函数setsockopt();\* 可选   
　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选   
　　4、设置要连接的对方的IP地址和端口等属性；   
　　5、连接服务器，用函数connect()；   
　　6、收发数据，用函数send()和recv()，或者read()和write();   
　　7、关闭网络连接；  
  
**UDP:**  
与之对应的UDP编程步骤要简单许多，分别如下：   
　　UDP编程的服务器端一般步骤是：   
　　1、创建一个socket，用函数socket()；   
　　2、设置socket属性，用函数setsockopt();\* 可选   
　　3、绑定IP地址、端口等信息到socket上，用函数bind();   
　　4、循环接收数据，用函数recvfrom();   
　　5、关闭网络连接；   
  
UDP编程的客户端一般步骤是：   
　　1、创建一个socket，用函数socket()；   
　　2、设置socket属性，用函数setsockopt();\* 可选   
　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选   
　　4、设置对方的IP地址和端口等属性;   
　　5、发送数据，用函数sendto();   
　　6、关闭网络连接；  
  
TCP和UDP是OSI模型中的运输层中的协议。TCP提供可靠的通信传输，而UDP则常被用于让广播和细节控制交给应用的通信传输。  
  
UDP补充：  
   UDP不提供复杂的控制机制，利用IP提供面向无连接的通信服务。并且它是将应用程序发来的数据在收到的那一刻，立刻按照原样发送到网络上的一种机制。即使是出现网络拥堵的情况下，UDP也无法进行流量控制等避免网络拥塞的行为。此外，传输途中如果出现了丢包，UDO也不负责重发。甚至当出现包的到达顺序乱掉时也没有纠正的功能。如果需要这些细节控制，那么不得不交给由采用UDO的应用程序去处理。换句话说，UDP将部分控制转移到应用程序去处理，自己却只提供作为传输层协议的最基本功能。UDP有点类似于用户说什么听什么的机制，但是需要用户充分考虑好上层协议类型并制作相应的应用程序。  
  
TCP补充：实  
  TCP充分现了数据传输时各种控制功能，可以进行丢包的重发控制，还可以对次序乱掉的分包进行顺序控制。而这些在UDP中都没有。此外，TCP作为一种面向有连接的协议，只有在确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费。TCP通过检验和、序列号、确认应答、重发控制、连接管理以及窗口控制等机制实现可靠性传输。  
  
  
TCP与UDP区别总结：  
1、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  
2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付  
3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的  
  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IP电话，实时视频会议等）  
4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信  
5、TCP首部开销20字节;UDP的首部开销小，只有8个字节  
6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

# TCP的三次握手与四次挥手理解及面试题（很全面）



    序列号seq：占4个字节，用来标记数据段的顺序，TCP把连接中发送的所有数据字节都编上一个序号，第一个字节的编号由本地随机产生；给字节编上序号后，就给每一个报文段指派一个序号；序列号seq就是这个报文段中的第一个字节的数据编号。

    确认号ack：占4个字节，期待收到对方下一个报文段的第一个数据字节的序号；序列号表示报文段携带数据的第一个字节的编号；而确认号指的是期望接收到下一个字节的编号；因此当前报文段最后一个字节的编号+1即为确认号。

    确认ACK：占1位，仅当ACK=1时，确认号字段才有效。ACK=0时，确认号无效

    同步SYN：连接建立时用于同步序号。当SYN=1，ACK=0时表示：这是一个连接请求报文段。若同意连接，则在响应报文段中使得SYN=1，ACK=1。因此，SYN=1表示这是一个连接请求，或连接接受报文。SYN这个标志位只有在TCP建产连接时才会被置1，握手完成后SYN标志位被置0。

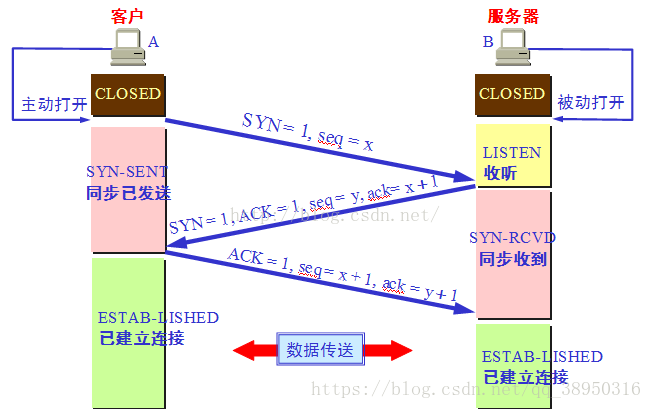
    终止FIN：用来释放一个连接。FIN=1表示：此报文段的发送方的数据已经发送完毕，并要求释放运输连接

    PS：ACK、SYN和FIN这些大写的单词表示标志位，其值要么是1，要么是0；ack、seq小写的单词表示序号。

### 

| **字段** | **含义** |
| --- | --- |
| **URG** | 紧急指针是否有效。为1，表示某一位需要被优先处理 |
| **ACK** | 确认号是否有效，一般置为1。 |
| **PSH** | 提示接收端应用程序立即从TCP缓冲区把数据读走。 |
| **RST** | 对方要求重新建立连接，复位。 |
| **SYN** | 请求建立连接，并在其序列号的字段进行序列号的初始值设定。建立连接，设置为1 |
| **FIN** | 希望断开连接。 |

### 三次握手过程理解

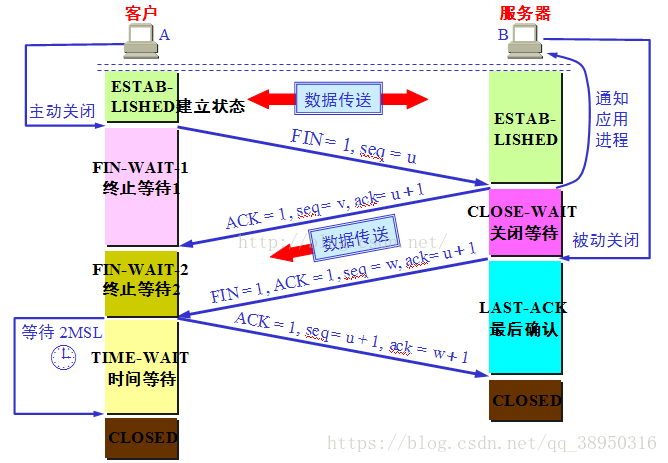


第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入**SYN\_SENT**状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个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**（TCP连接成功）状态，完成三次握手。

### 四次挥手过程理解



1）客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。 TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。  
2）服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。  
3）客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）。  
4）服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认。  
5）客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2∗∗MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。  
6）服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

### 常见面试题

【问题1】为什么连接的时候是三次握手，关闭的时候却是四次握手？

答：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。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

【问题2】为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？

答：虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假象网络是不可靠的，有可以最后一个ACK丢失。所以TIME\_WAIT状态就是用来重发可能丢失的ACK报文。在Client发送出最后的ACK回复，但该ACK可能丢失。Server如果没有收到ACK，将不断重复发送FIN片段。所以Client不能立即关闭，它必须确认Server接收到了该ACK。Client会在发送出ACK之后进入到TIME\_WAIT状态。Client会设置一个计时器，等待2MSL的时间。如果在该时间内再次收到FIN，那么Client会重发ACK并再次等待2MSL。所谓的2MSL是两倍的MSL(Maximum Segment Lifetime)。MSL指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

【问题3】为什么不能用两次握手进行连接？

答：3次握手完成两个重要的功能，既要双方做好发送数据的准备工作(双方都知道彼此已准备好)，也要允许双方就初始序列号进行协商，这个序列号在握手过程中被发送和确认。

       现在把三次握手改成仅需要两次握手，死锁是可能发生的。作为例子，考虑计算机S和C之间的通信，假定C给S发送一个连接请求分组，S收到了这个分组，并发 送了确认应答分组。按照两次握手的协定，S认为连接已经成功地建立了，可以开始发送数据分组。可是，C在S的应答分组在传输中被丢失的情况下，将不知道S 是否已准备好，不知道S建立什么样的序列号，C甚至怀疑S是否收到自己的连接请求分组。在这种情况下，C认为连接还未建立成功，将忽略S发来的任何数据分 组，只等待连接确认应答分组。而S在发出的分组超时后，重复发送同样的分组。这样就形成了死锁。

【问题4】如果已经建立了连接，但是客户端突然出现故障了怎么办？  
  
TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

# 进程间通信的8种方式

2018年11月12日 10:51:34 [Step\_Top](https://me.csdn.net/qq_41333582" \t "_blank) 阅读数 908

版权声明：转载请注明原文地址 <https://blog.csdn.net/qq_41333582/article/details/83988370>

前言：

进程通信：  
每个进程各自有不同的用户地址空间,任何一个进程的全局变量在另一个进程中都看不到，所以进程之间要交换数据必须通过内核,在内核中开辟一块缓冲区,进程A把数据从用户空间拷到内核缓冲区,进程B再从内核缓冲区把数据读走,内核提供的这种机制称为进程间通信。

##### 进程间通信（IPC）介绍

进程间通信（IPC，InterProcess Communication）是指在不同进程之间传播或交换信息。

IPC的方式通常有**管道**（包括无名管道和命名管道）、**消息队列**、**信号量**、**共享存储**、**Socket**、**Streams**等。其中 Socket和Streams支持不同主机上的两个进程IPC。

##### 一. 管道通信

管道，通常指无名管道，是 UNIX 系统IPC最古老的形式。

1、特点：

* 它是半双工的（即数据只能在一个方向上流动），具有固定的读端和写端。
* 它只能用于具有亲缘关系的进程之间的通信（也是父子进程或者兄弟进程之间）。
* 它可以看成是一种特殊的文件，对于它的读写也可以使用普通的read、write 等函数。但是它不是普通的文件，并不属于其他任何文件系统，并且只存在于内存中。

#include <unistd.h>

int pipe(int fd[2]); // 返回值：若成功返回0，失败返回-1

* 1
* 2

2、 类型

###### 1. 匿名管道通信

匿名管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

通过匿名管道实现进程间通信的步骤如下：

* 父进程创建管道，得到两个⽂件描述符指向管道的两端
* 父进程fork出子进程，⼦进程也有两个⽂件描述符指向同⼀管道。
* 父进程关闭fd[0],子进程关闭fd[1]，即⽗进程关闭管道读端,⼦进程关闭管道写端（因为管道只支持单向通信）。⽗进程可以往管道⾥写,⼦进程可以从管道⾥读,管道是⽤环形队列实现的,数据从写端流⼊从读端流出,这样就实现了进程间通信。

###### 2. 高级管道通信

高级管道(popen)：将另一个程序当做一个新的进程在当前程序进程中启动，则它算是当前程序的子进程，这种方式我们成为高级管道方式。

###### 3. 有名管道通信

有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信

##### 二. 信息队列

消息队列，是消息的链接表，存放在内核中。一个消息队列由一个标识符（即队列ID）来标识。  
消息队列是比较高级的一种进程间通信方式，因为它真的是可以在进程间传送message，传送普通字符串也可以。

一个消息队列可以被多个进程所共享（IPC((Inter-Process Communication,进程间通信))就是在这个基础上进行的）;如果一个进程消息太多，一个消息队列放不下，也可以用多于一个的消息队列（不管管理可能会比较复杂）。共享消息队列的进程所发送的消息除了message本身外还有一个标志，这个标志可以指明该消息将由哪个进程或者哪类进程接受。每一个共享消息队列的进程针对这个队列也有自己的标志，可以用来申明自己的身份。

1、特点

* 消息队列是面向记录的，其中的消息具有特定的格式以及特定的优先级。
* 消息队列独立于发送与接收进程。进程终止时，消息队列及其内容并不会被删除。
* 消息队列可以实现消息的随机查询,消息不一定要以先进先出的次序读取,也可以按消息的类型读取。

##### 三. 信息量

信号和信号量是不同的，它们虽然都可以用来同步和互斥，但是信号是使用信号处理器来进行的，信号量是使用P,V操作来实现的。

信号量（semaphore），它是一个计数器。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

###### 1、特点

* 信号量用于进程间同步，若要在进程间传递数据需要结合共享内存。
* 信号量基于操作系统的 PV 操作，程序对信号量的操作都是原子操作。
* 每次对信号量的 PV 操作不仅限于对信号量值加 1 或减 1，而且可以加减任意正整数。
* 支持信号量组。

##### 四. 共享内存通信

共享内存（Shared Memory），指两个或多个进程共享一个给定的存储区。共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

###### 1、特点

* 共享内存是最快的一种 IPC，因为进程是直接对内存进行存取。
* 因为多个进程可以同时操作，所以需要进行同步。
* 信号量+共享内存通常结合在一起使用，信号量用来同步对共享内存的访问。

##### 五. 套接字( socket )

套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信

通信过程如下：

###### 5.1命名socket

SOCK\_STREAM 式本地套接字的通信双方均需要具有本地地址，其中服务器端的本地地址需要明确指定，指定方法是使用 struct sockaddr\_un 类型的变量。

###### 5.2 绑定

SOCK\_STREAM 式本地套接字的通信双方均需要具有本地地址，其中服务器端的本地地址需要明确指定，指定方法是使用 struct sockaddr\_un 类型的变量，将相应字段赋值，再将其绑定在创建的服务器套接字上，绑定要使用 bind 系统调用，其原形如下：

int bind(int socket, const struct sockaddr \*address, size\_t address\_len);

* 1

其中 socket表示服务器端的套接字描述符，address 表示需要绑定的本地地址，是一个 struct sockaddr\_un 类型的变量，address\_len 表示该本地地址的字节长度。

###### 5.3 监听

服务器端套接字创建完毕并赋予本地地址值（名称，本例中为Server Socket）后，需要进行监听，等待客户端连接并处理请求，监听使用 listen 系统调用，接受客户端连接使用accept系统调用，它们的原形如下：

int listen(int socket, int backlog);

int accept(int socket, struct sockaddr \*address, size\_t \*address\_len);

* 1
* 2
* 3

其中 socket 表示服务器端的套接字描述符；backlog 表示排队连接队列的长度（若有多个客户端同时连接，则需要进行排队）；address 表示当前连接客户端的本地地址，该参数为输出参数，是客户端传递过来的关于自身的信息；address\_len 表示当前连接客户端本地地址的字节长度，这个参数既是输入参数，又是输出参数。

###### 5.4 连接服务器

客户端套接字创建完毕并赋予本地地址值后，需要连接到服务器端进行通信，让服务器端为其提供处理服务。

对于SOCK\_STREAM类型的流式套接字，需要客户端与服务器之间进行连接方可使用。连接要使用 connect 系统调用，其原形为

int connect(int socket, const struct sockaddr \*address, size\_t address\_len);

* 1

其中socket为客户端的套接字描述符，address表示当前客户端的本地地址，是一个 struct sockaddr\_un 类型的变量，address\_len 表示本地地址的字节长度。实现连接的代码如下：

connect(client\_sockfd, (struct sockaddr\*)&client\_address, sizeof(client\_address));

* 1

###### 5.5 相互发送接收数据

无论客户端还是服务器，都要和对方进行数据上的交互，这种交互也正是我们进程通信的主题。一个进程扮演客户端的角色，另外一个进程扮演服务器的角色，两个进程之间相互发送接收数据，这就是基于本地套接字的进程通信。发送和接收数据要使用 write 和 read 系统调用，它们的原形为：

int read(int socket, char \*buffer, size\_t len);

int write(int socket, char \*buffer, size\_t len);

* 1
* 2

其中 socket 为套接字描述符；len 为需要发送或需要接收的数据长度；

对于 read 系统调用，buffer 是用来存放接收数据的缓冲区，即接收来的数据存入其中，是一个输出参数；

对于 write 系统调用，buffer 用来存放需要发送出去的数据，即 buffer 内的数据被发送出去，是一个输入参数；返回值为已经发送或接收的数据长度。

###### 5.6 断开连接

交互完成后，需要将连接断开以节省资源，使用close系统调用，其原形为：

int close(int socket);

* 1

##### 五种通讯方式总结：

* 1.管道：速度慢，容量有限，只有父子进程能通讯
* 2.消息队列：容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题
* 3.信号量：不能传递复杂消息，只能用来同步
* 4.共享内存区：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了同一进程内的一块内存
* 5.套接字： 可用于不同机器间的进程通信。