**Others**

1. 计算机网络

**OSI，TCP/IP，五层协议的体系结构，以及各层协议**

**OSI分层 （7层）**：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

**TCP/IP分层（4层）**：网络接口层、 网际层、运输层、 应用层。

**五层协议     （5层）**：物理层、数据链路层、网络层、运输层、 应用层。

**每一层的协议如下：**

物理层：RJ45、CLOCK、IEEE802.3    （中继器，集线器，网关）

数据链路：PPP、FR、HDLC、VLAN、MAC  （网桥，交换机）

网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP、 （路由器）

传输层：TCP、UDP、SPX

会话层：NFS、SQL、NETBIOS、RPC

表示层：JPEG、MPEG、ASII

应用层：FTP、DNS、Telnet、SMTP、HTTP、WWW、NFS

**每一层的作用如下：**

**物理层：**通过媒介传输比特,确定机械及电气规范（比特Bit）

**数据链路层**：将比特组装成帧和点到点的传递（帧Frame）

**网络层**：负责数据包从源到宿的传递和网际互连（包PackeT）

**传输层**：提供端到端的可靠报文传递和错误恢复（段Segment）

**会话层**：建立、管理和终止会话（会话协议数据单元SPDU）

**表示层**：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）

**应用层**：允许访问OSI环境的手段（应用协议数据单元APDU）

**IP地址的分类**

A类地址：以0开头， 第一个字节范围：0~127（1.0.0.0 - 126.255.255.255）；

B类地址：以10开头，    第一个字节范围：128~191（128.0.0.0 - 191.255.255.255）；

C类地址：以110开头，  第一个字节范围：192~223（192.0.0.0 - 223.255.255.255）；

10.0.0.0—10.255.255.255， 172.16.0.0—172.31.255.255， 192.168.0.0—192.168.255.255。（Internet上保留地址用于内部）

**IP地址与子网掩码相与得到主机号**

**ARP是地址解析协议，简单语言解释一下工作原理。**

1：首先，每个主机都会在自己的ARP缓冲区中建立一个ARP列表，以表示IP地址和MAC地址之间的对应关系。

2：当源主机要发送数据时，首先检查ARP列表中是否有对应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，该数据包包括的内容有：**源主机** **IP地址，源主机MAC地址，目的主机的IP 地址**。

3：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，如果已经存在，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。

4：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。

广播发送ARP请求，单播发送ARP响应。

**各种协议**

**ICMP协议：**因特网控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。

**TFTP协议：**是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。

**HTTP协议：**超文本传输协议，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

**DHCP协议：**动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

**NAT协议**：网络地址转换属接入广域网(WAN)技术，是一种将私有（保留）地址转化为合法IP地址的转换技术，

**DHCP协议：**一个局域网的网络协议，使用UDP协议工作，用途：给内部网络或网络服务供应商自动分配IP地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段。

**描述：RARP**

RARP是逆地址解析协议，作用是完成硬件地址到IP地址的映射，主要用于无盘工作站，因为给无盘工作站配置的IP地址不能保存。工作流程：在网络中配置一台RARP服务器，里面保存着IP地址和MAC地址的映射关系，当无盘工作站启动后，就封装一个RARP数据包，里面有其MAC地址，然后广播到网络上去，当服务器收到请求包后，就查找对应的MAC地址的IP地址装入响应报文中发回给请求者。因为需要广播请求报文，因此RARP只能用于具有广播能力的网络。

**TCP三次握手和四次挥手的全过程**

**三次握手：**

第一次握手：客户端发送syn包(syn=x)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），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=y+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。

握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。

**四次握手**

与建立连接的“三次握手”类似，断开一个TCP连接则需要“四次握手”。

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。



**在浏览器中输入www.baidu.com后执行的全部过程**

1、客户端浏览器通过DNS解析到www.baidu.com的IP地址220.181.27.48，通过这个IP地址找到客户端到服务器的路径。客户端浏览器发起一个HTTP会话到220.161.27.48，然后通过TCP进行封装数据包，输入到网络层。

2、在客户端的传输层，把HTTP会话请求分成报文段，添加源和目的端口，如服务器使用80端口监听客户端的请求，客户端由系统随机选择一个端口如5000，与服务器进行交换，服务器把相应的请求返回给客户端的5000端口。然后使用IP层的IP地址查找目的端。

3、客户端的网络层不用关系应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，我不作过多的描述，无非就是通过查找路由表决定通过那个路径到达服务器。

4、客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包现在就可以传输了，然后发送IP数据包到达服务器的地址。

**TCP和UDP的区别？**

TCP提供面向连接的、可靠的数据流传输，而UDP提供的是非面向连接的、不可靠的数据流传输。

TCP传输单位称为TCP报文段，UDP传输单位称为用户数据报。

TCP注重数据安全性，UDP数据传输快，因为不需要连接等待，少了许多操作，但是其安全性却一般。

**TCP对应的协议和UDP对应的协议**

**TCP对应的协议：**

（1） **FTP**：定义了文件传输协议，使用**21**端口。

（2） **Telnet**：一种用于远程登陆的端口，使用**23**端口，用户可以以自己的身份远程连接到计算机上，可提供基于DOS模式下的通信服务。

（3） **SMTP**：邮件传送协议，用于发送邮件。服务器开放的是**25**号端口。

（4） **POP3**：它是和SMTP对应，POP3用于接收邮件。POP3协议所用的是**110**端口。

（5）**HTTP**：是从Web服务器传输超文本到本地浏览器的传送协议。

**UDP对应的协议：**

（1） **DNS**：用于域名解析服务，将域名地址转换为IP地址。DNS用的是**53**号端口。

（2） **SNMP**：简单网络管理协议，使用**161**号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。

（3）**TFTP**(Trival File Tran敏感词er Protocal)，简单文件传输协议，该协议在熟知端口**69**上使用UDP服务。

**DNS域名系统，简单描述其工作原理。**

当DNS客户机需要在程序中使用名称时，它会查询DNS服务器来解析该名称。客户机发送的每条查询信息包括三条信息：包括：指定的DNS域名，指定的查询类型，DNS域名的指定类别。基于UDP服务，端口53. 该应用一般不直接为用户使用，而是为其他应用服务，如HTTP，SMTP等在其中需要完成主机名到IP地址的转换。

**面向连接和非面向连接的服务的特点是什么？**

面向连接的服务，通信双方在进行通信之前，要先在双方建立起一个完整的可以彼此沟通的通道，在通信过程中，整个连接的情况一直可以被实时地监控和管理。

非面向连接的服务，不需要预先建立一个联络两个通信节点的连接，需要通信的时候，发送节点就可以往网络上发送信息，让信息自主地在网络上去传，一般在传输的过程中不再加以监控。

**TCP的三次握手过程？为什么会采用三次握手，若采用二次握手可以吗？**

答：建立连接的过程是利用客户服务器模式，假设主机A为客户端，主机B为服务器端。

（1）TCP的三次握手过程：主机A向B发送连接请求；主机B对收到的主机A的报文段进行确认；主机A再次对主机B的确认进行确认。

（2）采用三次握手是为了防止失效的连接请求报文段突然又传送到主机B，因而产生错误。失效的连接请求报文段是指：主机A发出的连接请求没有收到主机B的确认，于是经过一段时间后，主机A又重新向主机B发送连接请求，且建立成功，顺序完成数据传输。考虑这样一种特殊情况，主机A第一次发送的连接请求并没有丢失，而是因为网络节点导致延迟达到主机B，主机B以为是主机A又发起的新连接，于是主机B同意连接，并向主机A发回确认，但是此时主机A根本不会理会，主机B就一直在等待主机A发送数据，导致主机B的资源浪费。

（3）采用两次握手不行，原因就是上面说的实效的连接请求的特殊情况。

**端口及对应的服务？**

|  |  |  |  |
| --- | --- | --- | --- |
| 服务 | 端口号 | 服务 | 端口号 |
| FTP | 21 | SSH | 22 |
| telnet | 23 | SMTP | 25 |
| Domain(域名服务器) | 53 | HTTP | 80 |
| POP3 | 110 | NTP（网络时间协议） | 123 |
| MySQL数据库服务 | 3306 | Shell或 cmd | 514 |
| POP-2 | 109 | SQL Server | 1433 |

**了解交换机、路由器、网关的概念，并知道各自的用途**

1）交换机

在计算机网络系统中，交换机是针对共享工作模式的弱点而推出的。交换机拥有一条高带宽的背部总线和内部交换矩阵。交换机的所有的端口都挂接在这条背部总线上，当控制电路收到数据包以后，处理端口会查找内存中的地址对照表以确定目的MAC（网卡的硬件地址）的NIC（网卡）挂接在哪个端口上，通过内部交换矩阵迅速将数据包传送到目的端口。目的MAC若不存在，交换机才广播到所有的端口，接收端口回应后交换机会“学习”新的地址，并把它添加入内部地址表中。

交换机工作于OSI参考模型的第二层，即数据链路层。交换机内部的CPU会在每个端口成功连接时，通过ARP协议学习它的MAC地址，保存成一张 ARP表。在今后的通讯中，发往该MAC地址的数据包将仅送往其对应的端口，而不是所有的端口。因此，交换机可用于划分数据链路层广播，即冲突域；但它不能划分网络层广播，即广播域。

交换机被广泛应用于二层网络交换，俗称“二层交换机”。

交换机的种类有：二层交换机、三层交换机、四层交换机、七层交换机分别工作在OSI七层模型中的第二层、第三层、第四层盒第七层，并因此而得名。

2）路由器

**路由器**（**Router**）是一种计算机网络设备，提供了路由与转送两种重要机制，可以决定数据包从来源端到目的端所经过的路由路径（host到host之间的传输路径），这个过程称为路由；将路由器输入端的数据包移送至适当的路由器输出端(在路由器内部进行)，这称为转送。路由工作在OSI模型的第三层——即网络层，例如网际协议。

**路由器的一个作用是连通不同的网络，另一个作用是选择信息传送的线路。** 路由器与交换器的差别，路由器是属于OSI第三层的产品，交换器是OSI第二层的产品(这里特指二层交换机)。

3）网关

**网关**（Gateway），**网关**顾名思义就是连接两个网络的设备，区别于路由器（由于历史的原因，许多有关TCP/IP 的文献曾经把网络层使用的路由器（Router）称为网关，在今天很多局域网采用都是路由来接入网络，因此现在通常指的网关就是路由器的IP），经常在家庭中或者小型企业网络中使用，用于连接局域网和Internet。 网关也经常指把一种协议转成另一种协议的设备，比如语音网关。

在传统TCP/IP术语中，网络设备只分成两种，一种为网关（gateway），另一种为主机（host）。网关能在网络间转递数据包，但主机不能转送数据包。在主机（又称终端系统，end system）中，数据包需经过TCP/IP四层协议处理，但是在网关（又称中介系 统，intermediate system）只需要到达网际层（Internet layer），决定路径之后就可以转送。在当时，网关 （gateway）与路由器（router）还没有区别。

在现代网络术语中，网关（gateway）与路由器（router）的定义不同。网关（gateway）能在不同协议间移动数据，而路由器（router）是在不同网络间移动数据，相当于传统所说的IP网关（IP gateway）。

**网关**是连接两个网络的设备，对于语音网关来说，他可以连接PSTN网络和以太网，这就相当于VOIP，把不同电话中的模拟信号通过网关而转换成数字信号，而且加入协议再去传输。在到了接收端的时候再通过**网关**还原成模拟的电话信号，最后才能在电话机上听到。

对于以太网中的**网关**只能转发三层以上数据包，这一点和路由是一样的。而不同的是**网关**中并没有路由表，他只能按照预先设定的不同网段来进行转发。网关最重要的一点就是端口映射，子网内用户在外网看来只是外网的IP地址对应着不同的端口，这样看来就会保护子网内的用户。

1. **操作系统**

**操作系统基本特性**

1：并发、2：共享、3：虚拟、4：异步

**描述多道批处理、分时、实时操作系统的特点各是什么?**

**批处理操作系统：**多道性、无序性、调度性，系统利**用率高、吞吐量大**、平均周转时间长、但无交互能力。

**分时操作系统：**有多路性、独立性、及时性和**交互性**。 有较好的人机交互的特性，并且可以实现共享主机

**实时系统**：有多路性、独立性、及时性、交互性和**可靠性**。实际上是指操作系统工作时，其各种资源可以根据需要随时进行动态分配。由于各种资源可以进行动态分配，因此，其处理事务的能力较强、速度较快。

**总结：从可靠性：实时系统更强，从交互性：分时系统更强**

**二、进程的描述与控制**

**程序并发执行的特征**

**1：间断性**：程序在并发执行的时候，因为是共享资源，以及完成同一项任务而相互合作，致使在这些并发执行的程序之间形成了相互制约的关系，导致程序执行呈现：执行--暂停--执行

**2：失去封闭性**：当系统中有多个并发执行的程序时，各个资源是他们所共享的，这些资源的状态也由这些程序所改变，所以摸一个程序的运行环境会受到其他程序的影响。

**3：不可再生性**

**进程的特征与三种基本状态**

**1、特征：**

    1：动态性，2：并发性，3：独立性，4：异步性。

**2、状态：**

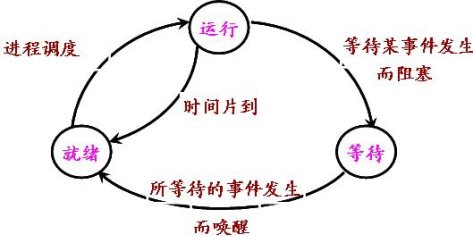
1：就绪状态，2：执行状态，3：阻塞状态

**3、三种基本状态转换：**

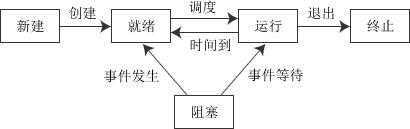
    处于就绪状态的进程，在调度程序为之分配了处理机之后便开始执行， 就绪 -> 执行

    正在执行的进程如果因为分配他的时间片已经用完，而被剥夺处理剂， 执行 -> 就绪

    如果因为某种原因致使当前的进程执行受阻，使之不能执行。               执行 -> 阻塞



4、创建状态和终止状态图



**进程控制块PCB的作用**

　　1：作为独立运行基本单位的标志

　　2：能实现间断性运行方式

　　3：提供进程通信管理所需要的信息

　　4：提供进程调度所需要的信息

**线程与进程的区别联系**

**1、定义：**

**进程：**进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的一个独立单位。（包括程序段，相关数据段，和进程控制块PCB）

**线程：**线程是进程的一个实体，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

**2、关系：**一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.相对进程而言，线程是一个更加接近于执行体的概念，它可以与同进程中的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。

**3、区别：**主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。**但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。**

**4.优缺点**：线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源的管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。

**进程间的通信是如何实现的？**

**早期的属于低级通信：**原因：1：效率低，生产者每次只能向缓冲池投放一个信息  2：通信对用户不透明，隐藏了通信的具体细节。**现在发展为高级通信：**用户可以利用操作系统所提供的一组通信命令传送大量数据。操作系统隐藏了进程通信的实现细节。或者说，通信过程对用户是透明的。

高级通信机制：

**1：共享存储器系统（存储器中划分的共享存储区）**

　　实际操作中对应的是“剪贴板”（剪贴板实际上是系统维护管理的一块内存区域）的通信方式。

**2：消息传递系统（进程间的数据交换以消息（message）为单位）**

　　当今最流行的微内核操作系统中，微内核与服务器之间的通信，都采用了消息传递机制

**3：管道通信系统（连接读写进程实现他们之间通信的共享文件（pipe文件，类似先进先出的队列，由一个进程写，另一进程读））**

　　管道分为匿名管道、命名管道。匿名管道是未命名的、单向管道，通过父进程和一个子进程之间传输数据。匿名管道只能实现本地机器上两个进程之间的通信，不能实现跨网络的通信。命名管道不仅可以在本机上实现两个进程间的通信，还可以跨网络实现进程间的通信。

**4：客户机-服务器系统**

  包括：套接字（socket），远程过程调用和远程方法调用

**什么是临界区？如何解决冲突？**

　　每个进程中访问临界资源的那段程序称为临界区，每次只准许一个进程进入临界区，进入后不允许其他进程进入。

　　1：如果有若干进程要求进入空闲的临界区，一次仅允许一个进程进入；

　　2：任何时候，处于临界区内的进程不可多于一个。如已有进程进入自己的临界区，则其它所有试图进入临界区的进程必须等待；

　　3：进入临界区的进程要在有限时间内退出，以便其它进程能及时进入自己的临界区；

　　4：如果进程不能进入自己的临界区，则应让出CPU，避免进程出现“忙等”现象。

**进程同步原则**

　　进程同步的主要任务：是对多个相关进程在执行次序上进行协调，以使并发执行的诸进程之间能有效地共享资源和相互合作，从而使程序的执行具有可再现性。同步机制遵循的原则：

　　1：空闲让进；

　　2：忙则等待（保证对临界区的互斥访问）；

　　3：有限等待（有限代表有限的时间，避免死等）；

　　4：让权等待，（当进程不能进入自己的临界区时，应该释放处理机，以免陷入忙等状态）。

**进程同步**

　　由于进程同步产生了一系列经典的同步问题“生产者-消费者”问题，“哲学家进餐”问题，“读者-写者”问题。

**程序和进程的区别**

**程序：**计算机指令的集合，它以文件的形式存储在磁盘上。程序是静态实体（passive Entity），在多道程序系统中，它是不能独立运行的，更不能与其他程序并发执行。

使用系统资源情况：不使用（程序不能申请系统资源，不能被系统调度，也不能作为独立运行的单位，它不占用系统的运行资源）。

**进程：**进程是进程实体（包括：程序段、相关的数据段、进程控制块PCB）的运行过程，是一个程序在其自身的地址空间中的一次执行活动。是系统进行资源分配和调度的一个独立单位。

使用系统资源情况：使用（进程是资源申请、调度和独立运行的单位，因此，它使用系统中的运行资源）

**三、处理机调度与死锁**

**处理机调度的层次：**

1：高级调度

  主要用于多道批处理系统中，又称长作业调度，调度队像是作业，根据某种算法决定将后备队列中的哪几个作业调入内存。

2：低级调度

  操作系统中最基本的一种调度方式（频率最高），在多道批处理、分时和时实三中类型的OS中都存在，又称为短作业调度。

3：中级调度

 又称为内存调度，目的是为了提高内存的利用率和系统的吞吐率，

**作业调度的算法：**

**1：先来先服务算法（FSFS）**

  最简单的调度算法，既可用于作业调度也可用于进程调度，系统按照作业到达的先后顺序进行调度，或者是优先考虑在系统中等待时间最长的作业

**2：短作业优先调度算法（SJF）**

  实际情况短作业占有比例很大，为了使他们比长作业优先执行，而产生了短作业优先的调度算法 ，作业越短优先级越高，

  缺点：是必须知道作业的运行时间，对长作业不利，人机无法实现交互，未完全考虑作业的紧迫程度

**3：优先级调度算法（PSA）**

　　优先级：对于先来先服务算法，作业的等待时间就是他的优先级，等待时间越长优先级越高，对于短作业优先级作业的长短就是他的优先级。在优先级算法中，基于作业的紧迫程度。

**4：高响应比优先调度算法（HRRN）**

　　在FSFS中只是考虑作业的等待时间而忽略作业的运行时间，SJF算法正好相反，高响应比算法既考虑作业的等待时间有考虑作业的运行时间，

　　优先权 = （等待时间+要求服务时间）/要求服务时间

　　由于等待时间与服务时间之和就是作业的相应时间，顾优先级相当于响应比：Rp

　　Rp =（等待时间+要求服务时间）/要求服务时间 = 响应时间/要求服务时间

**什么是死锁，死锁产生的4个条件**

**死锁定义：**

　　在两个或多个并发进程中，如果每个进程持有某种资源而又都等待别的进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗地讲，就是两个或多个进程被无限期地阻塞、相互等待的一种状态。

**产生条件：**

　1：互斥条件   一个资源一次只能被一个进程使用

　2：请求保持条件 一个进程因请求资源而阻塞时，对已经获得资源保持不放

　3：不可抢占条件 进程已获得的资源在未使用完之前不能强行剥夺

　4：循环等待条件 若干进程之间形成一种头尾相接的循环等待资源的关系

**预防避免死锁的方法**

　　1：**破坏“请求和保持”条件**：规定所有进程在开始运行之前，都必须一次性的申请其在整个运行过程所需要的全部资源。

    优点：简单，安全。  缺点：资源严重浪费，恶化了系统的利用率；

　　2：**破坏“不剥夺”条件**：进程逐个的提出资源请求，当一个已经保持了某些资源的进程，再提出新的资源请求而不能立即得到满足时，必须释放它已经保持了的所有资源，待以后需要时再重新申请。

　　缺点：实现复杂，代价大，反复地申请和释放资源，而使进程的执行无限的推迟、延长了进程的周转时间增加系统开销、降低系统吞吐量。

　　3：**破坏“环路等待”条件**：将所有的资源按类型进行线性排队，并赋予不同的序号。所有进程请求资源必须按照资源递增的次序提出，防止出现环路。

　　缺点：1、序号必须相对稳定，限制了新设备类型的增加2、作业（进程）使用资源顺序和系统规定的顺序不同而造成资源的浪费3、限制了用户编程

**注意：由于互斥条件是非共享设备所必需的，不能改变**

**四、存储器管理**

**连续分配存储管理方式**

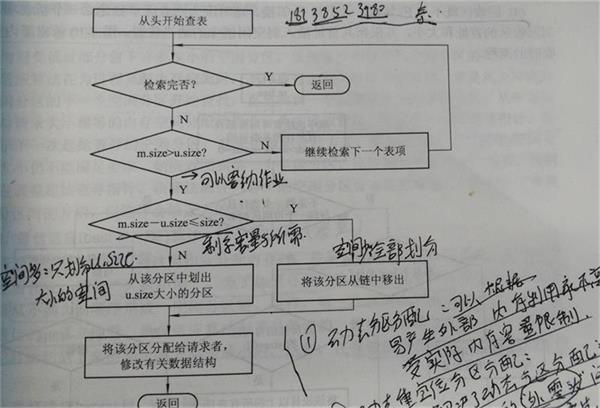
　　1：单一连续分配

　　2：固定分区分配

　　3：动态分区分配

  其中动态分区分配将涉及到分区分配中实际需要的数据结构，分区分配算法和分区的分配与回收操作

　　内存分配的流程：



**动态分区分配算法**

**1：首次适应算法（FF）：**

   要求地址空间递增的顺序链接，再分配内存时从链首开始查找，知道有一个满足的空间为止。该算法优先利用内存中低址空间，保留了高址空间，**缺点是低址部分不断被划分，留下许多内存碎片**，

**2：循环首次适应算法(NF)：**

   为了防止留下碎片，减少低址空间开销，NF算法每次从上一次分配的地方继续分配，该算法需要一个起始查询的指针用于指示下一次查询的空间地址。**缺点是：缺乏大的空间分区**

**3：最加适应算法（BF）：**

 每次作业分配时，总是把满足要求，又是最小的空间分配给作业，该算法把空间分区按其容量大小从小到大排列成空闲区链，**缺点是：留下许多内存碎片**，

**4：最坏适应算法（WF）：**

总是挑选最大的空闲区域分配给作业使用，优点是不至于使空闲区间太小，产生碎片的可能性小，**缺点是：缺乏大的空间分区**

**分页存储管理方式**

分页存储的基本方法：

  1：页面和物理快  页面：分页存储管理将进程的逻辑地址空间分成若干页，并从0开始编号，把内存的物理地址分成若干块（物理快）

  2：地址结构：  页号P ----- 偏移量W（包括页号P和偏移量W），对于特定的机器其地址结构一定，给定逻辑地址A，页面的大小为L，则页号P和页内地址D有一下关系：P = int[A/L]; d=[A] MOD L  例如：页面大小1kb A=2170B 得：p=2 d=122

  3：页表：记录相应页在内存中对应的物理块号

  4：地址转换机构：将用户逻辑空间的地址，转变为空间中的物理地址

**分段存储管理方式**

  分段是管理不会产生内存碎片（短号 ， 段内地址）

  分段管理方式的引入原因：

  1：一般程序分为若干段，如：主程序段、数据段、栈段等，每个段大多是一个相对独立的单位

  2：实现满足信息共享、信息保护、动态链接、以及信息动态增长等需要

**分页和分段的区别：**

共同点：两者都采用离散分配方式，且都地址映射机构来实现地址的转换

不同点： 1：页是信息的物理单位采用分页存储管理方式是为了实现离散分配方法。提高内存的利用率，采用分段目的主要在于能更好的满足用户的需求

2：页的大小固定且有系统决定，在采用分页存储管理方式中直接由硬件实现。而段的大小不固定，决定于用户所编写的程序

3：分页的地址空间是一维的，分页完全是系统完全是行为，分段系统中是二维的。

**段页式存储管理方式**

　基本原理是分段和分页相结合，其地址结构由：段号、段内页号、页内地址三部分组成。在段页式系统中获得一条指令需要三次访问内存，第一次访问内存中的段表，第二次访问内存中的页表，第三次访问内存中的数据。

**五、虚拟存储器**

**操作系统的内容分为几块？什么叫做虚拟内存？他和主存的关系如何？内存管理属于操作系统的内容吗？**

    操作系统的主要组成部分：进程和线程的管理，存储管理，设备管理，文件管理。虚拟内存是一些系统页文件，存放在磁盘上，每个系统页文件大小为4K，物理内存也被分页，每个页大小也为4K，这样虚拟页文件和物理内存页就可以对应，实际上虚拟内存就是用于物理内存的临时存放的磁盘空间。页文件就是内存页，物理内存中每页叫物理页，磁盘上的页文件叫虚拟页，物理页+虚拟页就是系统所有使用的页文件的总和。

**请求分页存储管理方式**

请求页表机制：作用是吧用户的逻辑地址映射为内存空间中的物理地址。

结构

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 页号 | 物理块号 | 状态位P | 访问字段A | 修改位M | 外存地址 |

1：状态位P：指示页面是否调入内存，供程序访问时参考

2：访问字段A：用于记录本也在一段时间内被访问的次数，供换出页面时参考

3：修改位M：标识页面调入内存是否被修改过，供置换页面时参考

4：外存地址：用于指示该页在外存上指示地址

内存分配：

  最小物理块数：若采用单地址指令，且采用直接寻址，需要物理块数是2，一块用于存放指令页面，另一块用于存放数据页面

  若采用间接寻址至少需要3块

**虚拟存储器页面置换算法**

**1：最佳置换算法（Optimal）：**一种理论的算法，选着淘汰的页面是以后一定不再使用的页面（理想化的），该算法无法实现，只能作为其他算法好坏的一个评价对比。

**2：先进先出（FIFO）算法：**总是最先淘汰最先进去的页面，该算法容易实现。缺点：通常程序调入内存的先后顺序和程序执行的先后顺序不一致，导致缺页率高。

**3：最近最久未使用（LEU）：**FIFO算法性能差，LRU算法根据页面调入内存的先后顺序决定，因为违法预测未来的使用情况，就是用过去的使用情况作为将来的使用情况的近似。

**4：最少使用算法（LFU）：**在每个页面设置一个移位寄存器记录该页面的访问频率，最近时期最少使用的页面被淘汰

1. Others
2. 深度优先搜索和广度优先搜索
   1. 深度优先搜索

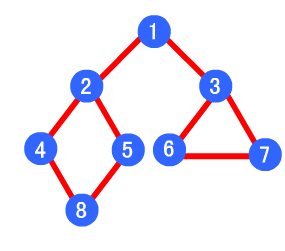
深度优先遍历，从初始访问结点出发，我们知道初始访问结点可能有多个邻接结点，深度优先遍历的策略就是首先访问第一个邻接结点，然后再以这个被访问的邻接结点作为初始结点，访问它的第一个邻接结点。总结起来可以这样说：每次都在访问完当前结点后首先访问当前结点的第一个邻接结点。

我们从这里可以看到，这样的访问策略是优先往纵向挖掘深入，而不是对一个结点的所有邻接结点进行横向访问。

具体算法表述如下：

1. 访问初始结点v，并标记结点v为已访问。
2. 查找结点v的第一个邻接结点w。
3. 若w存在，则继续执行4，否则算法结束。
4. 若w未被访问，对w进行深度优先遍历递归（即把w当做另一个v，然后进行步骤123）。
5. 查找结点v的w邻接结点的下一个邻接结点，转到步骤3。

例如下图，其深度优先遍历顺序为 1->2->4->8->5->3->6->7



public class AMWGraph {

private ArrayList vertexList;//存储点的链表

private int[][] edges;//邻接矩阵，用来存储边

private int numOfEdges;//边的数目

}

//私有函数，深度优先遍历

private void depthFirstSearch(boolean[] isVisited,int i) {

//首先访问该结点，在控制台打印出来

System.out.print(getValueByIndex(i)+" ");

//置该结点为已访问

isVisited[i]=true;

int w=getFirstNeighbor(i);//

while (w!=-1) {

if (!isVisited[w]) {

depthFirstSearch(isVisited,w);

}

w=getNextNeighbor(i, w);

}

}

//对外公开函数，深度优先遍历，与其同名私有函数属于方法重载

public void depthFirstSearch() {

for(int i=0;i<getNumOfVertex();i++) {

//因为对于非连通图来说，并不是通过一个结点就一定可以遍历所有结点的。

if (!isVisited[i]) {

depthFirstSearch(isVisited,i);

}

}

}

* 1. 广度优先搜索

类似于一个分层搜索的过程，广度优先遍历需要使用一个队列以保持访问过的结点的顺序，以便按这个顺序来访问这些结点的邻接结点。

具体算法表述如下：

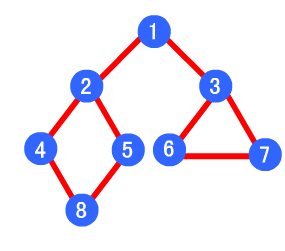
1. 访问初始结点v并标记结点v为已访问。
2. 结点v入队列
3. 当队列非空时，继续执行，否则算法结束。
4. 出队列，取得队头结点u。
5. 查找结点u的第一个邻接结点w。
6. 若结点u的邻接结点w不存在，则转到步骤3；否则循环执行以下三个步骤：

1). 若结点w尚未被访问，则访问结点w并标记为已访问。

2). 结点w入队列

3). 查找结点u的继w邻接结点后的下一个邻接结点w，转到步骤6。

如下图，其广度优先算法的遍历顺序为：1->2->3->4->5->6->7->8



//私有函数，广度优先遍历

private void broadFirstSearch(boolean[] isVisited,int i) {

int u,w;

LinkedList queue=new LinkedList();

//访问结点i

System.out.print(getValueByIndex(i)+" ");

isVisited[i]=true;

//结点入队列

queue.addlast(i);

while (!queue.isEmpty()) {

u=((Integer)queue.removeFirst()).intValue();

w=getFirstNeighbor(u);

while(w!=-1) {

if(!isVisited[w]) {

//访问该结点

System.out.print(getValueByIndex(w)+" ");

//标记已被访问

isVisited[w]=true;

//入队列

queue.addLast(w);

}

//寻找下一个邻接结点

w=getNextNeighbor(u, w);

}

}

}

//对外公开函数，广度优先遍历

public void broadFirstSearch() {

for(int i=0;i<getNumOfVertex();i++) {

if(!isVisited[i]) {

broadFirstSearch(isVisited, i);

}

}

}

1. HTTP请求和响应的全过程

**（1）首先，在浏览器里输入网址。www.baidu.com**

**（2）浏览器根据域名解析IP地址。**

**（3）浏览器与web服务器建立一个TCP连接。**

**（4）浏览器给web服务器发送一个http请求。**

（5）服务器的永久重定向响应。

为什么服务器一定要重定向而不是直接发送用户想看的网页内容呢？其中一个原因跟搜索引擎排名有关。如果一个页面有两个地址，就像http://www.igoro.com/和http://igoro.com/，搜索引擎会认为它们是两个网站，结果造成每个搜索链接都减少从而降低排名。而搜索引擎知道301永久重定向是什么意思，这样就会把访问带www的和不带www的地址归到同一个网站排名下。还有就是用不同的地址会造成缓存友好性变差，当一个页面有好几个名字时，它可能会在缓存里出现好几次。

（6）浏览器跟踪重定向地址

（7）服务器“处理”请求

（8）服务器发回一个HTML响应

（9）释放TCP连接

（10）客户端浏览器解析HTML内容

（11）浏览器获取嵌入在HTML中的对象

1xx：信息性状态码，表示服务器已接收了客户端请求，客户端可继续发送请求。

  100 Continue

  101 Switching Protocols

2xx：成功状态码，表示服务器已成功接收到请求并进行处理。

  200 OK 表示客户端请求成功

  204 No Content 成功，但不返回任何实体的主体部分

  206 Partial Content 成功执行了一个范围（Range）请求

3xx：重定向状态码，表示服务器要求客户端重定向。

  301 Moved Permanently 永久性重定向，响应报文的Location首部应该有该资源的新URL

  302 Found 临时性重定向，响应报文的Location首部给出的URL用来临时定位资源

  303 See Other 请求的资源存在着另一个URI，客户端应使用GET方法定向获取请求的资源

  304 Not Modified 客户端发送附带条件的请求（请求首部中包含如If-Modified-Since等指定首部）时，服务端有可能返回304，此时，响应报文中不包含任何报文主体。

  307 Temporary Redirect 临时重定向。与302 Found含义一样。302禁止POST变换为GET，但实际使用时并不一定，307则更多浏览器可能会遵循这一标准，但也依赖于浏览器具体实现

4xx：客户端错误状态码，表示客户端的请求有非法内容。

  400 Bad Request 表示客户端请求有语法错误，不能被服务器所理解

  401 Unauthonzed 表示请求未经授权，该状态代码必须与 WWW-Authenticate 报头域一起使用

  403 Forbidden 表示服务器收到请求，但是拒绝提供服务，通常会在响应正文中给出不提供服务的原因

  404 Not Found 请求的资源不存在，例如，输入了错误的URL

5xx：服务器错误状态码，表示服务器未能正常处理客户端的请求而出现意外错误。

  500 Internel Server Error 表示服务器发生不可预期的错误，导致无法完成客户端的请求

  503 Service Unavailable 表示服务器当前不能够处理客户端的请求，在一段时间之后，服务器可能会恢复正常

1. 丢包、拆包、流量控制、拥塞控制

3.1 丢包、拆包

在进行Java NIO学习时，发现，如果客户端连续不断的向服务端发送数据包时，服务端接收的数据会出现两个数据包粘在一起的情况，这就是TCP协议中经常会遇到的粘包以及拆包的问题。

我们都知道TCP属于传输层的协议，传输层除了有TCP协议外还有UDP协议。那么UDP是否会发生粘包或拆包的现象呢？答案是不会。UDP是基于报文发送的，从UDP的帧结构可以看出，在UDP首部采用了16bit来指示UDP数据报文的长度，因此在应用层能很好的将不同的数据报文区分开，从而避免粘包和拆包的问题。而TCP是基于字节流的，虽然应用层和TCP传输层之间的数据交互是大小不等的数据块，但是TCP把这些数据块仅仅看成一连串无结构的字节流，没有边界；另外从TCP的帧结构也可以看出，在TCP的首部没有表示数据长度的字段，基于上面两点，在使用TCP传输数据时，才有粘包或者拆包现象发生的可能。**粘包、拆包表现形式**

现在假设客户端向服务端连续发送了两个数据包，用packet1和packet2来表示，那么服务端收到的数据可以分为三种，现列举如下：

第一种情况，接收端正常收到两个数据包，即没有发生拆包和粘包的现象，此种情况不在本文的讨论范围内。

第二种情况，接收端只收到一个数据包，由于TCP是不会出现丢包的，所以这一个数据包中包含了发送端发送的两个数据包的信息，这种现象即为粘包。这种情况由于接收端不知道这两个数据包的界限，所以对于接收端来说很难处理。

第三种情况，这种情况有两种表现形式，如下图。接收端收到了两个数据包，但是这两个数据包要么是不完整的，要么就是多出来一块，这种情况即发生了拆包和粘包。这两种情况如果不加特殊处理，对于接收端同样是不好处理的。**粘包、拆包发生原因**

发生TCP粘包或拆包有很多原因，现列出常见的几点：

1、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。

2、待发送数据大于MSS（最大报文长度），TCP在传输前将进行拆包。

3、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。

4、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。

**粘包、拆包解决办法**

通过以上分析，我们清楚了粘包或拆包发生的原因，那么如何解决这个问题呢？解决问题的关键在于如何给每个数据包添加边界信息，常用的方法有如下几个：

1、发送端给每个数据包添加包首部，首部中应该至少包含数据包的长度，这样接收端在接收到数据后，通过读取包首部的长度字段，便知道每一个数据包的实际长度了。

2、发送端将每个数据包封装为固定长度（不够的可以通过补0填充），这样接收端每次从接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。

3、可以在数据包之间设置边界，如添加特殊符号，这样，接收端通过这个边界就可以将不同的数据包拆分开。

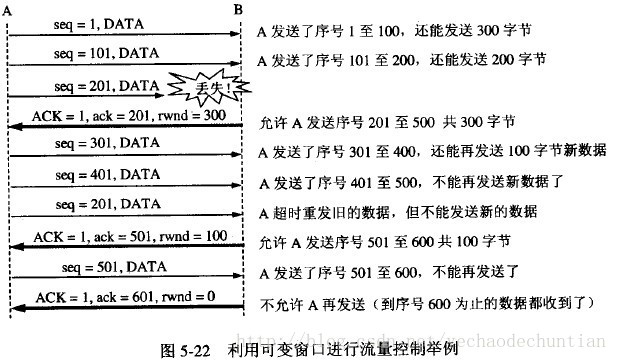
**TCP的流量控制**

1. 利用滑动窗口实现流量控制

    如果发送方把数据发送得过快，接收方可能会来不及接收，这就会造成数据的丢失。所谓**流量控制**就是让发送方的发送速率不要太快，要让接收方来得及接收。

    利用滑动窗口机制可以很方便地在TCP连接上实现对发送方的流量控制。

    设A向B发送数据。在连接建立时，B告诉了A：“我的接收窗口是 rwnd = 400 ”(这里的 rwnd 表示 receiver window) 。因此，发送方的发送窗口不能超过接收方给出的接收窗口的数值。请注意，TCP的窗口单位是字节，不是报文段。TCP连接建立时的窗口协商过程在图中没有显示出来。再设每一个报文段为100字节长，而数据报文段序号的初始值设为1。大写ACK表示首部中的确认位ACK，小写ack表示确认字段的值ack。



    从图中可以看出，B进行了三次流量控制。第一次把窗口减少到 rwnd = 300 ，第二次又减到了 rwnd = 100 ，最后减到 rwnd = 0 ，即不允许发送方再发送数据了。这种使发送方暂停发送的状态将持续到主机B重新发出一个新的窗口值为止。B向A发送的三个报文段都设置了 ACK = 1 ，只有在ACK=1时确认号字段才有意义。

    TCP为每一个连接设有一个持续计时器(persistence timer)。只要TCP连接的一方收到对方的零窗口通知，就启动持续计时器。若持续计时器设置的时间到期，就发送一个零窗口控测报文段（携1字节的数据），那么收到这个报文段的一方就重新设置持续计时器。

2. 必须考虑传输速率

    可以用不同的机制来控制TCP报文段的发送时机。如： <1>. TCP维持一个变量，它等于最大报文段长度MSS。只要缓存中存放的数据达到MSS字节时，就组装成一个TCP报文段发送出去。<2>. 由发送方的应用进程指明要求发送报文段，即TCP支持的推送( push )操作。<3>. 发送方的一个计时器期限到了，这时就把已有的缓存数据装入报文段(但长度不能超过MSS)发送出去。

    Nagle算法：若发送应用进程把要发送的数据逐个字节地送到TCP的发送缓存，则发送方就把第一个数据字节先发送出去，把后面到达的数据字节都缓存起来。当发送方接收对第一个数据字符的确认后，再把发送缓存中的所有数据组装成一个报文段再发送出去，同时继续对随后到达的数据进行缓存。只有在收到对前一个报文段的确认后才继续发送下一个报文段。当数据到达较快而网络速率较慢时，用这样的方法可明显地减少所用的网络带宽。Nagle算法还规定：当到达的数据已达到 发送窗口大小的一半或已达到报文段的最大长度时，就立即发送一个报文段。

    另，**糊涂窗口综合证：**TCP接收方的缓存已满，而交互式的应用进程一次只从接收缓存中读取1字节（这样就使接收缓存空间仅腾出1字节），然后向发送方发送确认，并把窗口设置为1个字节（但发送的数据报为40字节的的话）。接收，发送方又发来1个字节的数据（发送方的IP数据报是41字节）。接收方发回确认，仍然将窗口设置为1个字节。这样，网络的效率很低。要解决这个问题，可让接收方等待一段时间，使得或者接收缓存已有足够空间容纳一个最长的报文段，或者等到接收方缓存已有一半空闲的空间。只要出现这两种情况，接收方就发回确认报文，并向发送方通知当前的窗口大小。此外，发送方也不要发送太小的报文段，而是把数据报积累成足够大的报文段，或达到接收方缓存的空间的一半大小。

**TCP的拥塞控制**

1.  拥塞：即对资源的需求超过了可用的资源。若网络中许多资源同时供应不足，网络的性能就要明显变坏，整个网络的吞吐量随之负荷的增大而下降。

    拥塞控制：**防止过多的数据注入到网络中，这样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：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、路由器，以及与降低网络传输性能有关的所有因素。

    流量控制：指点对点通信量的控制，是端到端正的问题。流量控制所要做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。

    拥塞控制代价：需要获得网络内部流量分布的信息。在实施拥塞控制之前，还需要在结点之间交换信息和各种命令，以便选择控制的策略和实施控制。这样就产生了额外的开销。拥塞控制还需要将一些资源分配给各个用户单独使用，使得网络资源不能更好地实现共享。

2. 几种拥塞控制方法

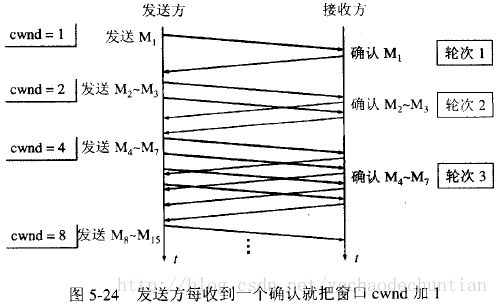
    慢开始( slow-start )、拥塞避免( congestion avoidance )、快重传( fast retransmit )和快恢复( fast recovery )。

2.1 慢开始和拥塞避免

    发送方维持一个拥塞窗口 cwnd ( congestion window )的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞。

    发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

    慢开始算法：当主机开始发送数据时，如果立即所大量数据字节注入到网络，那么就有可能引起网络拥塞，因为现在并不清楚网络的负荷情况。因此，较好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值。通常在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。



每经过一个传输轮次，拥塞窗口 cwnd 就加倍。一个传输轮次所经历的时间其实就是往返时间RTT。不过“传输轮次”更加强调：把拥塞窗口cwnd所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。

另，慢开始的“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，使得发送方在开始时只发送一个报文段（目的是试探一下网络的拥塞情况），然后再逐渐增大cwnd。

    为了防止拥塞窗口cwnd增长过大引起网络拥塞，还需要设置一个慢开始门限ssthresh状态变量（如何设置ssthresh）。慢开始门限ssthresh的用法如下：

    当 cwnd < ssthresh 时，使用上述的慢开始算法。

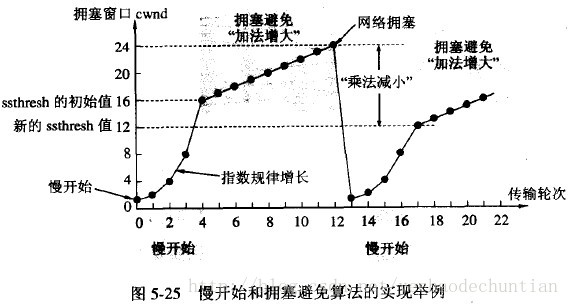
    当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。

    当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞控制避免算法。

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（其根据就是没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送方窗口值的一半（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。

    如下图，用具体数值说明了上述拥塞控制的过程。现在发送窗口的大小和拥塞窗口一样大。



<1>. 当TCP连接进行初始化时，把拥塞窗口cwnd置为1。前面已说过，为了便于理解，图中的窗口单位不使用字节而使用报文段的个数。慢开始门限的初始值设置为16个报文段，即 cwnd = 16 。

<2>. 在执行慢开始算法时，拥塞窗口 cwnd 的初始值为1。以后发送方每收到一个对新报文段的确认ACK，就把拥塞窗口值另1，然后开始下一轮的传输（图中横坐标为传输轮次）。因此拥塞窗口cwnd随着传输轮次按指数规律增长。当拥塞窗口cwnd增长到慢开始门限值ssthresh时（即当cwnd=16时），就改为执行拥塞控制算法，拥塞窗口按线性规律增长。

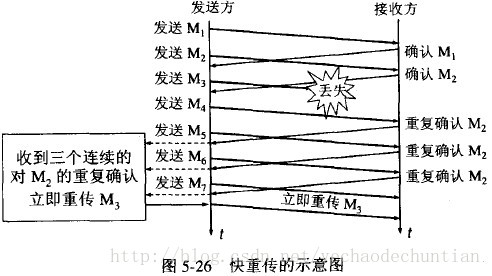
<3>. 假定拥塞窗口的数值增长到24时，网络出现超时（这很可能就是网络发生拥塞了）。更新后的ssthresh值变为12（即变为出现超时时的拥塞窗口数值24的一半），拥塞窗口再重新设置为1，并执行慢开始算法。当cwnd=ssthresh=12时改为执行拥塞避免算法，拥塞窗口按线性规律增长，每经过一个往返时间增加一个MSS的大小。

强调：“拥塞避免”并非指完全能够避免了拥塞。利用以上的措施要完全避免网络拥塞还是不可能的。“拥塞避免”是说在拥塞避免阶段将拥塞窗口控制为按线性规律增长，**使网络比较不容易出现拥塞。**

2.2 快重传和快恢复

如果发送方设置的超时计时器时限已到但还没有收到确认，那么很可能是网络出现了拥塞，致使报文段在网络中的某处被丢弃。这时，TCP马上把拥塞窗口 cwnd 减小到1，并执行慢开始算法，同时把慢开始门限值ssthresh减半。这是不使用快重传的情况。

    快重传算法首先要求接收方每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）而不要等到自己发送数据时才进行捎带确认。



接收方收到了M1和M2后都分别发出了确认。现在假定接收方没有收到M3但接着收到了M4。显然，接收方不能确认M4，因为M4是收到的失序报文段。根据可靠传输原理，接收方可以什么都不做，也可以在适当时机发送一次对M2的确认。但按照快重传算法的规定，接收方应及时发送对M2的重复确认，这样做可以让发送方及早知道报文段M3没有到达接收方。发送方接着发送了M5和M6。接收方收到这两个报文后，也还要再次发出对M2的重复确认。这样，发送方共收到了接收方的四个对M2的确认，其中后三个都是重复确认。快重传算法还规定，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M3，而不必继续等待M3设置的重传计时器到期。由于发送方尽早重传未被确认的报文段，因此采用快重传后可以使整个网络吞吐量提高约20%。

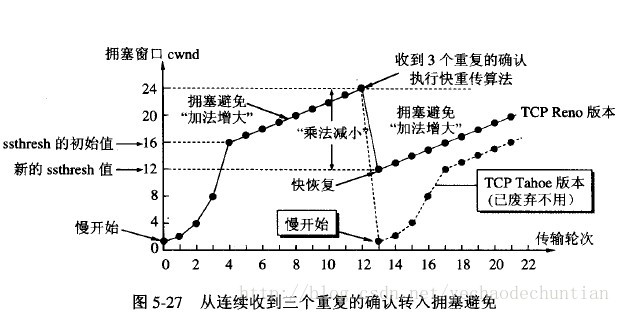
    与快重传配合使用的还有快恢复算法，其过程有以下两个要点：

    <1>. 当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半。这是为了预防网络发生拥塞。请注意：接下去不执行慢开始算法。

    <2>. 由于发送方现在认为网络很可能没有发生拥塞，因此与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

    下图给出了快重传和快恢复的示意图，并标明了“TCP Reno版本”。

区别：新的 TCP Reno 版本在快重传之后采用快恢复算法而不是采用慢开始算法。



 也有的快重传实现是把开始时的拥塞窗口cwnd值再增大一点，即等于 ssthresh + 3 X MSS 。这样做的理由是：既然发送方收到三个重复的确认，就表明有三个分组已经离开了网络。这三个分组不再消耗网络 的资源而是停留在接收方的缓存中。可见现在网络中并不是堆积了分组而是减少了三个分组。因此可以适当把拥塞窗口扩大了些。

    在采用快恢复算法时，慢开始算法只是在TCP连接建立时和网络出现超时时才使用。

    采用这样的拥塞控制方法使得TCP的性能有明显的改进。

    接收方根据自己的接收能力设定了接收窗口rwnd，并把这个窗口值写入TCP首部中的窗口字段，传送给发送方。因此，接收窗口又称为通知窗口。因此，从接收方对发送方的流量控制的角度考虑，发送方的发送窗口一定不能超过对方给出的接收窗口rwnd 。

    发送方窗口的上限值 = Min [ rwnd, cwnd ]

    当rwnd < cwnd 时，是接收方的接收能力限制发送方窗口的最大值。

    当cwnd < rwnd 时，则是网络的拥塞限制发送方窗口的最大值。

1. 数据库范式

关系数据库设计范式介绍

1.1 第一范式（1NF）无重复的列

所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。简而言之，第一范式就是无重复的列。  
说明：在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。

1.2 第二范式（2NF）属性完全依赖于主键[消除部分子函数依赖]  
      第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。例如员工信息表中加上了员工编号（emp\_id）列，因为每个员工的员工编号是惟一的，因此每个员工可以被惟一区分。这个惟一属性列被称为主关键字或主键、主码。

      第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是属性完全依赖于主键。

1.3 第三范式（3NF）属性不依赖于其它非主属性[消除传递依赖]

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在的员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性。

范式应用实例剖析

   下面以一个学校的学生系统为例分析说明，这几个范式的应用。首先第一范式（1NF）：数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。在当前的任何关系数据库管理系统（DBMS）中，傻瓜也不可能做出不符合第一范式的数据库，因为这些DBMS不允许你把数据库表的一列再分成二列或多列。因此，你想在现有的DBMS中设计出不符合第一范式的数据库都是不可能的。   
首先我们确定一下要设计的内容包括那些。学号、学生姓名、年龄、性别、课程、课程学分、系别、学科成绩，系办地址、系办电话等信息。为了简单我们暂时只考虑这些字段信息。我们对于这些信息，说关心的问题有如下几个方面。

学生有那些基本信息

学生选了那些课，成绩是什么

每个课的学分是多少

学生属于那个系，系的基本信息是什么。

2.1 第二范式（2NF）实例分析  
      首先我们考虑，把所有这些信息放到一个表中(学号，学生姓名、年龄、性别、课程、课程学分、系别、学科成绩，系办地址、系办电话)下面存在如下的依赖关系。   
        （学号）→ (姓名, 年龄，性别，系别，系办地址、系办电话)   
         (课程名称) → (学分)   
        （学号，课程）→ (学科成绩)

2.1.1 问题分析  
      因此不满足第二范式的要求，会产生如下问题

      数据冗余： 同一门课程由n个学生选修，"学分"就重复n-1次；同一个学生选修了m门课程，姓名和年龄就重复了m-1次。

      更新异常：

1)若调整了某门课程的学分，数据表中所有行的"学分"值都要更新，否则会出现同一门课程学分不同的情况。

 2)假设要开设一门新的课程，暂时还没有人选修。这样，由于还没有"学号"关键字，课程名称和学分也无法记录入数据库。

  删除异常 ： 假设一批学生已经完成课程的选修，这些选修记录就应该从数据库表中删除。但是，与此同时，课程名称和学分信息也被删除了。很显然，这也会导致插入异常。

2.1.2 解决方案  
      把选课关系表SelectCourse改为如下三个表：

学生：Student(学号，姓名, 年龄，性别，系别，系办地址、系办电话)；

课程：Course(课程名称, 学分)；

选课关系：SelectCourse(学号, 课程名称, 成绩)。

2.2 第三范式（3NF）实例分析

        接着看上面的学生表Student(学号，姓名, 年龄，性别，系别，系办地址、系办电话)，关键字为单一关键字"学号"，因为存在如下决定关系：

       （学号）→ (姓名, 年龄，性别，系别，系办地址、系办电话)

        但是还存在下面的决定关系

       (学号) → (所在学院)→(学院地点, 学院电话)

        即存在非关键字段"学院地点"、"学院电话"对关键字段"学号"的传递函数依赖。

        它也会存在数据冗余、更新异常、插入异常和删除异常的情况。 (數據的更新，刪除異常這里就不分析了，可以參照2.1.1進行分析)

        根据第三范式把学生关系表分为如下两个表就可以滿足第三范式了：

        学生：(学号, 姓名, 年龄, 性别，系别)；

        系别：(系别, 系办地址、系办电话)。

总结

       上面的数据库表就是符合I,II,III范式的，消除了数据冗余、更新异常、插入异常和删除异常。

1. cookie 和session 的区别详解

二者的定义：

当你在浏览网站的时候，WEB 服务器会先送一小小资料放在你的计算机上，Cookie 会帮你在网站上所打的文字或是一些选择，都纪录下来。当下次你再光临同一个网站，WEB 服务器会先看看有没有它上次留下的 Cookie 资料，有的话，就会依据 Cookie里的内容来判断使用者，送出特定的网页内容给你。 Cookie 的使用很普遍，许多有提供个人化服务的网站，都是利用 Cookie来辨认使用者，以方便送出使用者量身定做的内容，像是 Web 接口的免费 email 网站，都要用到 Cookie。

具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

同时我们也看到，由于采用服务器端保持状态的方案在客户端也需要保存一个标识，所以session机制可能需要借助于cookie机制来达到保存标识的目的，但实际上它还有其他选择。

cookie机制。正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以生成cookie。而cookie的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所在的位置，则把该cookie附在请求资源的HTTP请求头上发送给服务器。

cookie的内容主要包括：名字，值，过期时间，路径和域。路径与域一起构成cookie的作用范围。若不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，关闭浏览器窗口，cookie就消失。这种生命期为浏览器会话期的cookie被称为会话cookie。

会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。若设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。存储在硬盘上的cookie可以在不同的浏览器进程间共享，比如两个IE窗口。而对于保存在内存里的cookie，不同的浏览器有不同的处理方式session机制。session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。

  当程序需要为某个客户端的请求创建一个session时，服务器首先检查这个客户端的请求里是否已包含了一个session标识（称为session id），如果已包含则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（检索不到，会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。一般这个cookie的名字都是类似于SEEESIONID。但cookie可以被人为的禁止，则必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。

经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。还有一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。比如：

<form name="testform" action="/xxx">

<input type="hidden" name="jsessionid" value="ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764">

<input type="text">

</form>

实际上这种技术可以简单的用对action应用URL重写来代替。

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗  
   考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能  
   考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：

   将登陆信息等重要信息存放为SESSION

   其他信息如果需要保留，可以放在COOKIE中