2017.4.4 计算机基础

2017年4月4日

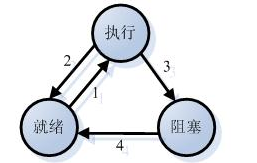
20:40

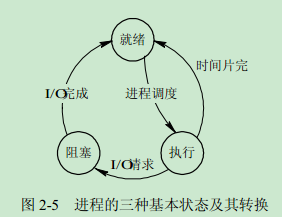
**基础篇：操作系统、计算机网络、设计模式**

**一：操作系统**

* 1. 进程的有哪几种状态，状态转换图，及导致转换的事件。

状态：绪（Ready）状态 进程已获得除处理机外的所需资源，等待分配处理机资源，只要分配到CPU就可执行。在某一时刻，可能有若干个进程处于该状态。 2）运行状态 占用处理机资源运行，处于此状态的进程的数目小于等于CPU的数目。 3）阻塞状态 由于进程等待某种条件（如I/O操作或进程同步），在条件满足之前无法继续执行。该事件发生前即使把处理机分配给该进程，也无法运行。





转换解释：从状态转换图中，存在四种状态转换。当进程调度程序从就绪队列中选取一个进程投入运行时引起转换1；正在执行的进程如因时间片用完而被暂停执行就会引起转换2；正在执行的进程因等待的事件尚未发生而无法执行（如进程请求完成I/O）则会引去转换3；当进程等待的事件发生时（如I/O完成）则会引起转换4。

事件：就绪队列非空，则一个进程的转换3会立即引去另一个进程的转换1。这是因为一个进程发生转换3意味着正在执行的进程由执行状态变为阻塞状态，这时处理机空闲，进程调度程序必然会从就绪队列中选取一个进程并将它投入运行，因此只要就绪队列非空，一个进程的转换3能立即引起一个进程的转换1。

* 1. 进程与线程的区别。

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,进程是系统进行资源分配和调度的一个独立单位.

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.

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

* 1. 进程通信的几种方式。

管道( pipe )：管道是一种半双工（不能同时接收发送数据，只能进行其中一种）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。

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。

信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。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。

消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存( shared memory ) ：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（进程间通信） 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。

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，它可用于不同及其间的进程通信。

* 1. 线程同步几种方式。(一定要会写生产者、消费者问题，完全消化理解)

临界区（CCriticalSection）：通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。

事件（CEvent）：为协调共同对一个共享资源的单独访问而设计的。

互斥量（CMutex）：为控制一个具有有限数量用户资源而设计。

信号量（CSemaphore）：用来通知线程有一些事件已发生，从而启动后继任务的开始。

<http://c.biancheng.net/cpp/html/2600.html>

* 1. 线程的实现方式. (也就是用户线程与内核线程的区别)

根据操作系统内核是否对线程可感知，可以把线程分为内核线程和用户线程。

内核线程建立和销毁都是由操作系统负责、通过系统调用完成的，操作系统在调度时，参考各进程内的线程运行情况做出调度决定，如果一个进程中没有就绪态的线程，那么这个进程也不会被调度占用CPU。

和内核线程相对应的是用户线程，用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，用户进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。用户线程多见于一些历史悠久的操作系统，例如Unix操作系统，不需要用户态/核心态切换，速度快，操作系统内核不知道多线程的存在，因此一个线程阻塞将使得整个进程（包括它的所有线程）阻塞。由于这里的处理器时间片分配是以进程为基本单位，所以每个线程执行的时间相对减少为了在操作系统中加入线程支持，采用了在用户空间增加运行库来实现线程，这些运行库被称为“线程包”，用户线程是不能被操作系统所感知的。

引入用户线程，具体而言，有以下四个方面的优势：

（1）可以在不支持线程的操作系统中实现。

（2）创建和销毁线程、线程切换代价等线程管理的代价比内核线程少得多。

（3）允许每个进程定制自己的调度算法，线程管理比较灵活。

（4）线程能够利用的表空间和堆栈空间比内核级线程多。

用户线程的缺点主要有以下两点：

（1）同一进程中只能同时有一个线程在运行，如果有一个线程使用了系统调用而阻塞，那么整个进程都会被挂起。

（2）页面失效也会产生类似的问题。

内核线程的优缺点刚好跟用户线程相反。实际上，操作系统可以使用混合的方式来实现线程。

* 1. 用户态和核心态的区别、7.用户栈和内核栈的区别。

当一个任务（进程）执行系统调用而陷入内核代码中执行时，我们就称进程处于内核运行态（或简称为内核态）。此时处理器处于特权级最高的（0级）内核代码中执行。当进程处于内核态时，执行的内核代码会使用当前进程的内核栈。每个进程都有自己的内核栈。当进程在执行用户自己的代码时，则称其处于用户运行态（用户态）。即此时处理器在特权级最低的（3级）用户代码中运行。当正在执行用户程序而突然被中断程序中断时，此时用户程序也可以象征性地称为处于进程的内核态。因为中断处理程序将使用当前进程的内核栈。这与处于内核态的进程的状态有些类似。

用户态切换到内核态的3种方式：系统调用、异常、外围设备中断。

<http://blog.csdn.net/xieyutian1990/article/details/38413413>

8.内存池、进程池、线程池。(c++程序员必须掌握)

自定义内存池的思想通过这个"池"字表露无疑，应用程序可以通过系统的内存分配调用预先一次性申请适当大小的内存作为一个内存池，之后应用程序自己对内存的分配和释放则可以通过这个内存池来完成。只有当内存池大小需要动态扩展时，才需要再调用系统的内存分配函数，其他时间对内存的一切操作都在应用程序的掌控之中。

应用程序自定义的内存池根据不同的适用场景又有不同的类型。

从线程安全的角度来分，内存池可以分为单线程内存池和多线程内存池。单线程内存池整个生命周期只被一个线程使用，因而不需要考虑互斥访问的问题；多线程内存池有可能被多个线程共享，因此则需要在每次分配和释放内存时加锁。相对而言，单线程内存池性能更高，而多线程内存池适用范围更广。

从内存池可分配内存单元大小来分，可以分为固定内存池和可变内存池。所谓固定内存池是指应用程序每次从内存池中分配出来的内存单元大小事先已经确定，是固定不变的；而可变内存池则每次分配的内存单元大小可以按需变化，应用范围更广，而性能比固定内存池要低。

9. 死锁的概念，导致死锁的原因.

死锁<DeadLock>: 是指两个或两个以上的进程在执行过程中,因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去.此时称系统处于死锁状态或系统产生了死锁,这些永远在互相等竺的进程称为死锁进程.

原因：系统资源不足、进程运行推进的顺序不合适、资源分配不当等。

10. 导致死锁的四个必要条件。

（1） 互斥条件：一个资源每次只能被一个进程使用。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3） 不剥夺条件:进程已获得的资源，在末使用完之前，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关系。

11. 处理死锁的四个方式。

1）忽略该问题。例如鸵鸟算法，该算法可以应用在极少发生死锁的的情况下。为什么叫鸵鸟算法呢，**（鸵鸟策略）**

2）检测死锁并且恢复。**（检测与解除策略）**

3）仔细地对资源进行动态分配，以避免死锁。**（避免策略）**

4）通过破除死锁四个必要条件之一，来防止死锁产生。**（预防策略）**

<http://blog.csdn.net/bxyill/article/details/8237339>

12. 预防死锁的方法、避免死锁的方法。

通过破除死锁四个必要条件之一，来预防死锁产生，有两种方法，一种是当其申请的资源得不到满足时，也必须放弃其原先占有的资源；另一种方法是只适用于申请资源的进程优先级比占有该资源的进程优先级高时，如果一个进程申请的资源被其它进程占用，而申请进程的优先级较高，那么它可以强迫占有资源的进程放弃。

仔细地对资源进行动态分配，以避免死锁。

13. 进程调度算法。(周转时间 =  程序结束时间 -- 开始服务时间、带权周转时间=  周转时间 /  要求服务时间)？？？

<http://blog.csdn.net/luyafei_89430/article/details/12971171>

14. Windows内存管理的方式(块式、页式、段式、段页式).

内存管理是操作系统中的重要部分，两三句话恐怕谁也说不清楚吧～～我先说个大概，希望能够抛砖引玉吧 当程序运行时需要从内存中读出这段程序的代码。代码的位置必须在物理内存中才能被运行，由于现在的操作系统中有非常多的程序运行着，内存中不能够完全放下，所以引出了虚拟内存的概念。把哪些不常用的程序片断就放入虚拟内存，当需要用到它的时候在load入主存（物理内存）中。这个就是内存管理所要做的事。内存管理还有另外一件事需要做：计算程序片段在主存中的物理位置，以便CPU调度。 内存管理有块式管理，页式管理，段式和段页式管理。现在常用段页式管理。

**块式管理：**把主存分为一大块、一大块的，当所需的程序片断不在主存时就分配一块主存空间，把程序片断load入主存，就算所需的程序片度只有几个字节也只能把这一块分配给它。这样会造成很大的浪费，平均浪费了50％的内存空间，但是易于管理。

**页式管理：**把主存分为一页一页的，每一页的空间要比一块一块的空间小很多，显然这种方法的空间利用率要比块式管理高很多。

**段式管理：**把主存分为一段一段的，每一段的空间又要比一页一页的空间小很多，这种方法在空间利用率上又比页式管理高很多，但是也有另外一个缺点。一个程序片断可能会被分为几十段，这样很多时间就会被浪费在计算每一段的物理地址上（计算机最耗时间的大家都知道是I/O吧）。

**段页式管理：**结合了段式管理和页式管理的优点。把主存分为若干页，每一页又分为若干段。

<http://blog.csdn.net/wang379275614/article/details/13765599>

15. 内存连续分配方式采用的几种算法及各自优劣。

**1) 单一连续分配**是一种最简单的存储管理方式，其优点是软件处理简单，最大缺点是存储器不能充分利用。多用于单用户微机操作系统中。

**2) 动态分区分配**是多道程序环境下各种存储管理方式中最简单的一种。它将内存划分成若干个分区，在每个分区中按照连续分配方式分配给一个作业。分区形式: **a) 固定分区分配：**指内存在处理作业前已被划分成若干个大小不等的分区，存储管理程序根据每个作业步的最大存储量分配一个足够大的分区给它。当找不到一个足够大的分区时，则通知作业调度挑选另一作业，这种方式分配和回收内存简单，但内存利用不充分，会产生“碎片”空间。**b) 可变分区分配：**指内存事先并未被分区，只有当作业进入内存时，才根据作业的大小建立分区。其特点是分区的个数和大小都是可变的，但需要建立分配区表和空白区表，且表的长度是不固定的。

**3) 可重定位分区分配** 为使各分区中的用户程序能移到内存的一端，使碎片集中于另一端成为一个大分区，在程序执行过程中，需对作业移动过程中的与地址有关项进行调整。这种分配方法的优点是清除碎片，更大程度地利用内存空间，但必须硬件的支持，且要花费时间。

16. 动态链接及静态链接.

静态链接库与动态链接库都是共享代码的方式，如果采用静态链接库，则无论你愿不愿意，lib 中的指令都全部被直接包含在最终生成的 EXE 文件中了。但是若使用 DLL，该 DLL 不必被包含在最终 EXE 文件中，EXE 文件执行时可以“动态”地引用和卸载这个与 EXE 独立的 DLL 文件。静态链接库和动态链接库的另外一个区别在于静态链接库中不能再包含其他的动态链接库或者静态库，而在动态链接库中还可以再包含其他的动态或静态链接 库。静态链接库与静态链接库调用规则总体比较如下。

17. 基本分页、请求分页储存管理方式。

<http://blog.csdn.net/xieyutian1990/article/details/38414479>

18. 分段分页方式的比较各自优缺点。

a) 页是信息的物理单位，分页是为实现离散分配方式，以消减内存的外零头，提高内存的利用率；或者说，分页仅仅是由于系统管理的需要，而不是用户的需要（也是对用户透明的）。段是信息的逻辑单位，它含有一组其意义相对完整的信息（比如数据段、代码段和堆栈段等）。分段的目的是为了能更好的满足用户的需要（用户也是可以使用的）。

<http://blog.csdn.net/xieyutian1990/article/details/38414673>

19. 几种页面置换算法，会算所需换页数。(LRU用程序如何实现？)

<http://www.cnblogs.com/fkissx/p/4712959.html>

20. 虚拟内存的定义及实现方式。

虚拟内存，它的作用与物理内存基本相似，但它是作为物理内存的“后备力量”而存在的，也就是说，只有在物理内存已经不够使用的时候，它才会发挥作用。

改变页面文件位置的方法是：用鼠标右键点击“我的电脑”，选择“属性→高级→性能设置→高级→更改虚拟内存”，在驱动器栏里选择想要改变到的位置

<http://c.biancheng.net/cpp/html/2612.html>

21. 操作系统的四个特性。

**并发性（concurrency）：**指在计算机系统中存在着许多并发执行的活动。对计算机系统 而言，并发是指宏观上看系统内有多道程序同时运行，微观上看是串行运行。因为在 大多数计算机系统中一般只有一个CPU，在任意时刻只能有一道程序占用CPU。

**共享性（sharing）：**系统中各个并发活动要共享计算机系统中的各种软、硬件资源，因此操作系统必须解决在多道程序间合理地分配和使用资源问题。

**虚拟性（virtual）：**虚拟是操作系统中的重要特征，所谓虚拟是指把物理上的一台设备 变成逻辑上的多台设备。例如，在操作系统中采用了spooling技术，可以利用快速、 大容量可共享的磁盘作为中介，模拟多个非共享的低速的输入输出设备，这样的设备 称为虚拟设备。

**异步性：**在多道程序环境下允许多个进程并发执行，但只有进程在获得所需的资源后方能执行。在单处理机环境下，由于系统中只有一台处理机，因而每次只允许一个进程执行，其余进程只能等待。

22. DMA。

DMA方式，Direct Memory Access，也称为成组[数据传送](http://baike.baidu.com/view/5593592.htm)方式，有时也称为直接内存操

<http://baike.baidu.com/link?url=vOZTQwKqOfhlaOg9n_OvLHBTbAWWDq7i9cU72ijHYXCpQ3HbwJ8zDmUSCrBVl7OpP2xCxOl7YLf6MK3Z1wNB3q>

23. Spooling。

  脱机输入和脱机输出

在多道环境下，可以用OS的一道管理程序实现从I/O设备输入数据并存放到磁盘上，模拟脱机输入；用OS的另一道管理程序将磁盘上的数据输出到I/O设备上，模拟脱机输出；这种假脱机I/O操作称为Spooling技术。

Spooling是一种虚拟设备技术、一种资源转换技术。

24. 外存分配的几种方式，及各种优劣。

**连续分配：**为每一个文件分配一组相邻接的盘块；物理上形成了顺序文件结构；外存上会出现“ 碎片” ，用“ 紧凑” 的方法解决。**优缺点：**顺序（批量）访问容易、速度快；要求有连续的存储空间（有时需要作紧凑处理）、必须事先知道文件的长度。

**链接分配：**离散分配方式。**优缺点**：消除了“碎片”，有利于文件的增/删/改。隐式链接

在文件的每个目录项中，都含有指向链接文件第一盘块和最后一个盘块的指针，只适合于顺序访；显式链接，把用于链接文件各物理块的指针，显式地存放在内存的一张“链接表”中。

**索引分配：**单级索引分配每个文件一个索引块（表）；多级索引分配当文件较大，需要很多个索引块时，可以为各索引块建立一个索引表（块）；混合索引分配方式。

**二：计算机网络**

1. 电路交换与分组交换的区别？优劣对比。

电路交换

优点：1.信息传输时延小2.信息以数字信号的形式在数据信道上进行“透明”传输，交换机对用户的数据信息不存储、处理，交换机在处理方面的开销比较小，对用户的数据信息不用附加控制信息，使信息的传送效率较高3.信息的编译吗和代码格式由通信双方决定，与交换网络无关。

缺点:1.网络的利用率低2.线路的利用率低3.限不同速率、不同代码格式、不同控制方式的相互直通4.无呼损。

报文交换：

优点：1.不同的终端接口之间可以相互直通2.无呼损3.利用动态的复用技术，线路的利用率较高。

缺点：传输时延大，而且变化的范围比较大2.利用“存储-转发”，所以要求交换系统有较高的处理速度和大的存储能力3.实时性较差。

分组交换

优点：1.可以对不同的接口终端进行匹配2.网络轻载情况下，传输时延较小，且比较稳定3.线路利用率高4.可靠性高5.经济效益好

缺点：1.网络系统附加了大量的控制信息，对于报文较长的信息传输率低2.技术实现复杂

<http://blog.csdn.net/njchenyi/article/details/1540657>

2. OSI有哪几层，会画出来，知道主要几层的各自作用。

应用层 为用户的应用程序提供网络接口

表示层 数据格式转换

会话层 数据同步过程中的会话管理。

传输层 提供端到端的透明传输

网络层 提供逻辑编址，数据寻址和路由

数据链路层 建立相邻节点间的数据链路，在有差错的物理介质上无差错的传输数据帧

物理层 比特流的传输

<http://blog.csdn.net/yaopeng_2005/article/details/7064869>

3. TCP/IP有哪几层，会画出来，知道所有层数的作用，会列举各层主要的协议名称。

TCP/IP由四个层次组成：网络接口层、网络层、传输层、应用层。

网络接口层：

物理层是定义物理介质的各种特性：机械特性、电子特性、功能特性、规程特性。

数据链路层是负责接收IP数据报并通过网络发送之，或者从网络上接收物理帧，抽出IP数据报，交给IP层。

常见的接口层协议有：Ethernet 802.3、Token Ring 802.5、X.25、Frame relay、HDLC、PPP ATM。

网络层

负责相邻计算机之间的通信。其功能包括三方面。

一、处理来自传输层的分组发送请求，收到请求后，将分组装入IP数据报，填充报头，选择去往信宿机的路径，然后将数据报发往适当的网络接口。

二、处理输入数据报：首先检查其合法性，然后进行寻径--假如该数据报已到达信宿机，则去掉报头，将剩下部分交给适当的传输协议；假如该数据报尚未到达信宿，则转发该数据报。

三、处理路径、流控、拥塞等问题。

网络层包括：IP(Internet Protocol)协议、ICMP(Internet Control Message Protocol) 控制报文协议、ARP(Address Resolution Protocol)地址转换协议、RARP(Reverse ARP)反向地址转换协议。

IP是网络层的核心，通过路由选择将下一跳IP封装后交给接口层。IP数据报是无连接服务。 ICMP是网络层的补充，可以回送报文。用来检测网络是否通畅。

Ping命令就是发送ICMP的echo包，通过回送的echo relay进行网络测试。

ARP是正向地址解析协议，通过已知的IP，寻找对应主机的MAC地址。

RARP是反向地址解析协议，通过MAC地址确定IP地址。比如无盘工作站和DHCP服务。

传输层

提供应用程序间的通信。其功能包括：

一、格式化信息流；

二、提供可靠传输。为实现后者，传输层协议规定接收端必须发回确认，并且假如分组丢失，必须重新发送。

传输层协议主要是：传输控制协议TCP(Transmission Control Protocol)和用户数据报协议UDP(User Datagram protocol)。

应用层

向用户提供一组常用的应用程序，比如电子邮件、文件传输访问、远程登录等。远程登录TELNET使用TELNET协议提供在网络其它主机上注册的接口。TELNET会话提供了基于字符的虚拟终端。文件传输访问FTP使用FTP协议来提供网络内机器间的文件拷贝功能。

应用层一般是面向用户的服务。如FTP、TELNET、DNS、SMTP、POP3。

FTP(File Transfer Protocol)是文件传输协议，一般上传下载用FTP服务，数据端口是20H，控制端口是21H。

Telnet服务是用户远程登录服务，使用23H端口，使用明码传送，保密性差、简单方便。

DNS(Domain Name Service)是域名解析服务，提供域名到IP地址之间的转换。

SMTP(Simple Mail Transfer Protocol)是简单邮件传输协议，用来控制信件的发送、中转。

POP3(Post Office Protocol 3)是邮局协议第3版本，用于接收邮件。

4. 硬件(MAC)地址的概念及作用。

概念：MAC地址就是在媒体接入层上使用的地址，也叫物理地址、硬件地址或链路地址，由网络设备制造商生产时写在硬件内部。MAC地址与网络无关，也即无论将带有这个地址的硬件（如网卡、集线器、路由器等）接入到网络的何处，都有相同的MAC地址，它由厂商写在网卡的BIOS里。MAC地址可采用6字节（48比特）或2字节（16比特）这两种中的任意一种。但随着局域网规模越来越大，一般都采用6字节的MAC地址。这个48比特都有其规定的意义，前24位是由生产网卡的厂商向IEEE申请的厂商地址，目前的价格是1000美元买一个地址块，后24位由厂商自行分配，这样的分配使得世界上任意一个拥有48位MAC地址的网卡都有唯一的标识。另外，2字节的MAC地址不用网卡厂商申请。

作用：MAC地址是标志一台终端的象徵，每台能上网的终端都有属於自己的唯一标识。MAC地址是在局域网里面传输的方式。局域网里，交换机通过识别MAC地址来进行数据包的传输。这个和网络层上根据IP地址来传输道理差不多。

5. ARP协议的用途及算法、在哪一层上会使用arp ？

用途：ARP协议其主要用作将IP地址翻译为以太网的MAC地址

算法：

应用于网络层/数据链路层

<http://baike.baidu.com/link?url=sHxuGxK8wIVXVYzBZiTzslwEQlvxkSX1ZoxbPc5cxBzzKXXeqfWq8dU6uYXdxzzoMt8wr2nn-3tbKJQO2yjUxK>

6. CRC冗余校验算法，反码和检验算法。

<http://www.cnblogs.com/esestt/archive/2007/08/09/848856.html>

7. 如何实现透明传输。

所谓透明传输，意思是物理层不对传输的比特流采取任何处理，只是单纯的将比特流从一个节点传到下一个节点，实现就是根据地址把比特流往不同的链路上转发就可以了。

透明传输是指不管所传数据是什么样的比特组合，都应当能够在链路上传送。当所传数据中的比特组合恰巧与某一个控制信息完全一样时，就必须采取适当的措施，使接收方不会将这样的数据误认为是某种控制信息。这样才能保证数据链路层的传输是透明的。

发送方和接收方数据的长度和内容完全一致，相当于一条无形的传输线。

关于透明传输的问题

透明传输就是在传输过程中，对外界透明，就是说你看不见他是传送网络，不管传输的业务如何，我只负责将需要传送的业务传送到目的节点，同时保证传输的质量即可，而不对传输的业务进行处理。

透明传输是指数据直接通过系统中的互连功能模式而不进行RLP纠错，如果进行了RLP纠错即为非透明传输。

就是所谓的透明传输，不管传的是什么，所采用的设备只是起一个通道作用，把要传输的内容完好的传到对方！

透传的设备是个黑箱子，进来是什么出去也是什么。

透明传输不用关心下层协议的传输，比如你要寄信，只需要写地址交给邮局就行了，然后对方就能收到你的信，但是中途经过多少车站，火车，邮递员，你根本不知道，所以对于你来说邮递的过程是透明的。

与面向连接和电话系统的工作模式相似，其特点是：

1、面向连接的数据传输过程必须经过建立连接、连接维护和释放连接的3个过程；

2、数据传输过程中，各分组不需要携带目的地址。

面向连接数据传输的收发数据顺序不变，因此传输的可靠性好，但协议复杂，通信效率不高。

说明：RLP(Radio Link Protocol): 无线连接协议，是一种严格的纠错方式。当数据在接收端被 “查出”有错后，接收端可要求电话重新进行数据传输，如有必要，反复进行，直至接收数据完全正确为止。

8. 知道各个层使用的是哪个数据交换设备。（交换机、路由器、网关）

物理层（即常说的第一层、层L1）中继系统，即转发器（repeater）。

数据链路层（即第二层，层L2），即交换机或网桥（bridge）。

网络层（第三层，层L3）中继系统，即路由器（router）。

网桥和路由器的混合物桥路器（brouter）兼有网桥和路由器的功能。

在网络层以上的中继系统，即网关（gateway）。

9. 路由表的内容。

对每组网络接口(interface)，路由表至少会存有下面的资讯：

网络ID（Network ID, Network number）：就是目标地址的网络ID。

子网掩码 (用来判断 IP 所属网络)

下一跳地址/接口（Next hop / interface）：就是数据在发送到目标地址的旅途中下一站的地址。

其中 interface 指向 next hop(即为下一个 route)。一个自治系统(Autonomous System)中的 route 应该包含区域内所有的子网络，而默认网关 (Network id: 0.0.0.0, Netmask: 0.0.0.0) 指向自治系统的出口。

根据应用和执行的不同，路由表可能含有如下附加信息：

花费（Cost）：就是数据发送过程中通过路径所需要的花费。

路由的服务质量

路由中需要过滤的出/入连接列表

路由表也是一个网络安全的关键，像单播可逆路径传输(uRPF)就是一个安全路由表的例子。在这种具有多种变体的技术中，路由器也在路由表中查找数据包的源地址，如果源地址不正确，数据包就会显示出错或受到攻击。

10. 分组转发算法。

分组转发算法::

1、从数据报的首部提取目的主机的IP地址D，计算出目的主机的网络地址N。

(将IP数据报中目的主机的IP地址和路由表上的子网掩码进行&运算，就可以得出网络地址N)

2、若N就是与此路由器直接相连的某个网络的网络地址。则直接进行交付，不需要经过其他路由器，而是直接将IP数据报交付给目的主机。

(注意，直接交付时，路由器需要将目的主机地址D转换为具体的硬件地址，把数据报封装在MAC帧，在发送此帧。)

若N不是与此路由器直接相连的网络，就进行间接交付。执行3或执行4

3、若路由表中有目的地址为D的特定主机路由，则把数据报传送给路由表中所指明的下一跳路由器；否则，执行4。

(这是特殊情况)

4、若路由表中有到达网络N的路由，则把数据报传送给路由表中所指明的下一跳路由器；否则，执行5。

5、如果3和4都没能将IP数据报转发出去，若路由表中有一个默认路由，则把数据报传送给路由表中所指明的默认

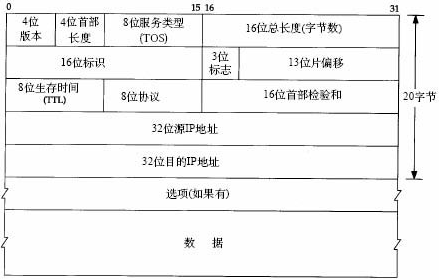
路由器；负责，执行6

6、报告转发分组出错。

11. IP报文的格式，格式的各个字段的含义要理解。

IP协议是TCP/IP协议族中最为核心的协议。它提供不可靠、无连接的服务，也即依赖其他层的协议进行差错控制。在局域网环境，IP协议往往被封装 在以太网帧中传送。而所有的TCP、UDP、ICMP、IGMP数据都被封装在IP数据报中传送。TCP/IP报文封装如图2-3所示：

C:\4C695045\0E2AB6F3-F9DA-47DC-9631-5A563F3DE12A.files\image003.jpg



其中：

●版本（Version）字段：占4比特。用来表明IP协议实现的版本号，当前一般为IPv4，即0100。

●报头长度（Internet Header Length，IHL）字段：占4比特。是头部占32比特的数字，包括可选项。普通IP数据报（没有任何选项），该字段的值是5，即160比特=20字节。此字段最大值为60字节。

●服务类型（Type of Service ，TOS）字段：占8比特。其中前3比特为优先权子字段（Precedence，现已被忽略）。第8比特保留未用。第4至第7比特分别代表延迟、吞吐量、 可靠性和花费。当它们取值为1时分别代表要求最小时延、最大吞吐量、最高可靠性和最小费用。这4比特的服务类型中只能置其中1比特为1。可以全为0，若全 为0则表示一般服务。服务类型字段声明了数据报被网络系统传输时可以被怎样处理。例如：TELNET协议可能要求有最小的延迟，FTP协议（数据）可能要 求有最大吞吐量，SNMP协议可能要求有最高可靠性，NNTP（Network News Transfer Protocol，网络新闻传输协议）可能要求最小费用，而ICMP协议可能无特殊要求（4比特全为0）。实际上，大部分主机会忽略这个字段，但一些动态 路由协议如OSPF（Open Shortest Path First Protocol）、IS-IS（Intermediate System to Intermediate System Protocol）可以根据这些字段的值进行路由决策。

●总长度字段：占16比特。指明整个数据报的长度（以字节为单位）。最大长度为65535字节。

●标志字段：占16比特。用来唯一地标识主机发送的每一份数据报。通常每发一份报文，它的值会加1。

●标志位字段：占3比特。标志一份数据报是否要求分段。

●段偏移字段：占13比特。如果一份数据报要求分段的话，此字段指明该段偏移距原始数据报开始的位置。

●生存期（TTL：Time to Live）字段：占8比特。用来设置数据报最多可以经过的路由器数。由发送数据的源主机设置，通常为32、64、128等。每经过一个路由器，其值减1，直到0时该数据报被丢弃。

●协议字段：占8比特。指明IP层所封装的上层协议类型，如ICMP（1）、IGMP（2） 、TCP（6）、UDP（17）等。

●头部校验和字段：占16比特。内容是根据IP头部计算得到的校验和码。计算方法是：对头部中每个16比特进行二进制反码求和。（和ICMP、IGMP、TCP、UDP不同，IP不对头部后的数据进行校验）。

●源IP地址、目标IP地址字段：各占32比特。用来标明发送IP数据报文的源主机地址和接收IP报文的目标主机地址。

可选项字段：占32比特。用来定义一些任选项：如记录路径、时间戳等。这些选项很少被使用，同时并不是所有主机和路由器都支持这些选项。可选项字段的长度必须是32比特的整数倍，如果不足，必须填充0以达到此长度要求。

12.MTU的概念，啥叫路径MTU？ MTU发现机制，TraceRoute(了解)。

MTU概念：最大传输单元（Maximum Transmission Unit，MTU）是指一种通信协议的某一层上面所能通过的最大数据包大小（以字节为单位）。最大传输单元这个参数通常与通信接口有关（网络接口卡、串口等）。

路径MTU：当一台IP主机有大量的数据要发送给另一台主机的时候，数据是作为一系列的IP数据报传输。数据报最好具有在从源点到目的点的路径上不需要分片的最大尺寸。（避免分片的情况）这种数据报的尺寸称作为路径MTU（PMTU）,它等于路径上每一跳的MTU之中的最小值。

路径MTU的发现机制：当路由器收到一份需要分片的数据报，而在IP首部又设置了不可分片（DF）的标志位时，则会产生ICMP不可达差错。（注：产生ICMP不可达差错可能有3种情况，TTL为0，端口不可达，以及IP需要分片但又设置了不允许标志。）这种情况下的ICMP不可达差错报文可以提供下一站的MTU。如果路由器没有提供这种新的ICMP差错报文格式，那么下一站的MTU就设为0。尽管大多数的系统不支持路径MTU发现功能，但可以很容易地修改traceroute程序，用它来确定路径MTU。要做的是发送分组，并设置“不可分片”标志比特。发送的第一个分组的长度正好与出口MTU相等，每次收到ICMP“不能分片”差错时就减小分组的长度。如果路由器发送的ICMP差错报文是新格式，包含出口的MTU，那么就用该MTU值来发送，否则就用下一个最小的MTU值来发送。正如RFC1191[Mogul and Deering 1990]声明的那样，MTU值的个数是有限的，因此在我们的程序中有一些由近似值构成的表，取下一个最小MTU值来发送。

13.RIP协议的概念及算法。

RIP协议的全称是一种内部网关协议（IGP），是一种动态路由选择，用于一个自治系统（AS）内的路由信息的传递。RIP协议是基于距离矢量算法（DistanceVectorAlgorithms）的，它使用“跳数”，即metric来衡量到达目标地址的路由距离。这种协议的路由器只关心自己周围的世界，只与自己相邻的路由器交换信息，范围限制在15跳(15度)之内，再远，它就不关心了。RIP应用于OSI网络七层模型的网络层。管理距离（AD）为110。

14.ICMP协议的主要功能。

ICMP是“Internet Control Message Protocol”（Internet控制消息协议）的缩写。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。控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。

ICMP的重要性

ICMP协议对于网络安全具有极其重要的意义。ICMP协议本身的特点决定了它非常容易被用于攻击网络上的路由器和主机。

15.组播和广播的概念，IGMP的用途。(环回地址、广播地址)

组播（multicasting）是一种多点投递的形式，它使用硬件技术，通过使用大量组播地址来通信。当某一组机器需要通信时，选择一个组播地址，并配置好相应的网络接口硬件，识别组播地址，从而收到该组播地址上分组的拷贝。

广播（broadcasting）是多点投递的最普遍的形式，它向每一个目的站投递一个分组的拷贝。它可以通过多个单次分组的投递完成，也可以通过单独的连接传递分组的拷贝，直到每个接收方均收到一个拷贝为止。

ARP地址解析协议：a.作用：用于建立IP地址到物理地址转换关系的一种协议，以支持物理地址的恰当确定。b.协议主要内容：协议为每一个网络主机建立一个IP地址到物理地址的“映射表”，并将其保存在所谓的“ARP高速缓存”中，地址的转换通过查表实现。

RARP逆地址解析协议：使只知道自己硬件地址的主机能够知道其IP地址。实现MAC地址到IP地址的解析。用于无盘工作站环境。

IP协议：实现网络互连，使参与互连的性能各异的网络从用户看起来像一个统一的网络。

ICMP因特网控制报文协议：允许主机或路由器报告差错情况和提供有关异常情况的报告。从而提高IP数据报交付成功的机会。

IGMP数据组控制协议：用来支持使单个的主机能够把自己的组成员关系及时通告给本网络上路由器的一种机制。

16.Ping协议的实现原理，ping 命令格式。

Ping协议的实现原理：PING (Packet Internet Grope)，因特网包探索器，用于测试网络连接量的程序。Ping发送一个ICMP回声请求消息给目的地并报告是否收到所希望的ICMP回声应答。 PING因特网包探索器，用于测试网络连接量的程序。Ping发送一个ICMP回声请求消息给目的地并报告是否收到所希望的ICMP回声应答。

ping 命令格式：ping IP地址 –t

使用Ping检查连通性有六个步骤：

  1. 使用ipconfig /all观察本地网络设置是否正确；

  2. Ping 127.0.0.1，127.0.0.1 回送地址Ping回送地址是为了检查本地的TCP/IP协议有没有设置好；

  3. Ping本机IP地址，这样是为了检查本机的IP地址是否设置有误；

  4. Ping本网网关或本网IP地址，这样的是为了检查硬件设备是否有问题，也可以检查本机与本地网络连接是否正常；（在非局域网中这一步骤可以忽略）    5.Ping本地DNS地址，这样做是为了检查DNS是否能够将IP。   6.Ping远程IP地址，这主要是检查本网或本机与外部的连接是否正常。

17. 子网划分的概念，子网掩码。

子网划分：Internet组织机构定义了五种IP地址，有A、B、C三类地址。A类网络有126个，每个A类网络可能有16777214台主机，它们处于同一广播域。而在同一广播域中有这么多结点是不可能的，网络会因为广播通信而饱和，结果造成16777214个地址大部分没有分配出去。可以把基于类的IP网络进一步分成更小的网络，每个子网由路由器界定并分配一个新的子网网络地址,子网地址是借用基于类的网络地址的主机部分创建的。划分子网后，通过使用掩码，把子网隐藏起来，使得从外部看网络没有变化，这就是子网掩码。

子网掩码：它是一种用来指明一个IP地址的哪些位标识的是主机所在的子网以及哪些位标识的是主机的位掩码。子网掩码不能单独存在，它必须结合IP地址一起使用。子网掩码只有一个作用，就是将某个IP地址划分成网络地址和主机地址两部分。

18. IP地址的分类，如何划分的，及会计算各类地址支持的主机数。

IP地址分类：

A：首位为0；1.0.0.0~126.0.0.0；主机号24位

B：前两位为10；128.1.0.0~191.255.0.0；主机号16位

C：前三位为110；192.0.1.0~223.255.255.0；主机号8位

D：前四位为1110；224.0.0.0~239.255.255.255

E：前四位为1111；240.0.0.0~255.255.255.254

主机号为0的网络地址，表示网络本身。例如202.120.95.0表示一个C类网络。

主机号全为1的地址保留做为定向广播。例如202.120.95.255表示一个广播地址。

127.0.0.0保留用于环回用来测试TCP/IP以及本机进程间的通信。故网络号为127的分组永远不会出现在网络上，而且主机或者路由器永远不能为127的地址传播选路或者可达性信息。他不是一个网络地址。

在IP地址3种主要类型里，各保留了3个区域作为私有地址，其地址范围如下：

A类地址：10.0.0.0～10.255.255.255

B类地址：172.16.0.0～172.31.255.255

C类地址：192.168.0.0～192.168.255.255

还有一个全1的网络地址，即255.255.255.255是一个受限的网络地址，用于广播！！！

19.DNS的概念，用途，DNS查询的实现算法。

概念：DNS 是计算机域名系统或域名解析服务器的缩写，它是由解析器以及域名服务器组成的。域名服务器是指保存有该网络中所有主机的域名和对应IP地址，并具有将域名转换为IP地址功能的服务器。DNS使用TCP与UDP端口号都是53，主要使用UDP，服务器之间备份使用TCP。

用途：把域名转换成为网络可以识别的ip地址。

DNS查询实现算法：

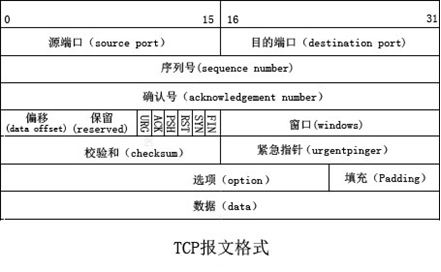
20. TCP与UDP的概念，相互的区别及优劣。

<http://blog.csdn.net/xieyutian1990/article/details/24603341>

21.UDP报文的格式，字段的意义。



22. TCP 报文的格式，字段的意义。



首部固定部分各段意义：

(1) 源端口 source port 2个字节

(2) 目的端口 destination port 2个字节

(3) 序号：sequence number,TCP传送的是面向连接的连续的数据流，所传送的数据每一个字节都编上一个序号，首部的这个序号指的是本报文段所发送的数据的第一个字节的序号，占4个字节

(4) 确认号：Ack number,是期望受到对方的下一个报文段的数据的第一个字节的序号，也就是期望收到的下一个报文段首部的序号字段的值，占4个字节

(5)数据偏移：占4位，（单位是4字节），类似于IP包头的首部长度，他是指TCP报文段首部的长度，由于存在长度不确定的选项字段，所以此值最小为20字节，最大为60字节

(6)保留：6bit,目前没用，设为0

(7)这里的6位是说明本报文段性质的，下面再详细说

(8)窗口：window，占2个字节，用来控制对方发送的数量

(9)检验和：checksum，2个字节，检验范围为首部加数据两部分

下面看看那6位的控制部分各位的含义：

SYN:该标志位用来建立连接,让连接双方同步序列号.

FIN:表示发送端已经没有数据要求传输了，希望释放连接.

RST:用来复位一个连接.RST标志置位的数据包称为复位包.一般情况下,如果TCP收到的一个分段明显不是属于该主机上的任何一个连接,则向远程发送一个复位包.

URG:为紧急数据标志.如果他为1,表示本数据包中包含紧急数据.此时紧急数据指针有效.

ACK:为确认标志位.如果为1,表示包中的确认号时有效的.否则,包中的确认号无效.

PSH:如果置位,接收端应尽快把数据传诵给应用层.

l

23.TCP通过哪些措施，保证传输可靠？

采用数据重传和数据确认应答机制，三次握手

<http://www.2cto.com/net/201310/251896.html>

24. 三次握手，四次断开过程。

三次握手：

①主机A通过向主机B 发送一个含有同步序列号的标志位的数据段给主机B ,向主机B 请求建立连接,通过这个数据段,主机A告诉主机B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.

②主机B 收到主机A的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机A,也告诉主机A两件事:我已经收到你的请求了,你可以传输数据了;你要用哪佧序列号作为起始数据段来回应我。

③主机A收到这个数据段后,再发送一个确认应答,确认已收到主机B 的数据段:"我已收到回复,我现在要开始传输实际数据了。

四次挥手：

①当主机A完成数据传输后,将控制位FIN置1,提出停止TCP连接的请求

②主机B收到FIN后对其作出响应,确认这一方向上的TCP连接将关闭,将ACK置1

③由B 端再提出反方向的关闭请求,将FIN置1

④主机A对主机B的请求进行确认,将ACK置1,双方向的关闭结束.

25. TIME\_WAIT状态的概念及意义。

客户端与服务器端建立TCP/IP连接后关闭SOCKET后，服务器端连接的端口状态为TIME\_WAIT。

是不是所有执行主动关闭的socket都会进入TIME\_WAIT状态呢？

有没有什么情况使主动关闭的socket直接进入CLOSED状态呢？

主动关闭的一方在发送最后一个ack 后就会进入TIME\_WAIT 状态停留2MSL（max segment lifetime）时间，这个是TCP/IP必不可少的，也就是“解决”不了的。

也就是TCP/IP设计者本来是这么设计的

主要有两个原因

1。防止上一次连接中的包，迷路后重新出现，影响新连接

（经过2MSL，上一次连接中所有的重复包都会消失）

2。可靠的关闭TCP连接

在主动关闭方发送的最后一个ack(fin)，有可能丢失，这时被动方会重新发fin, 如果这时主动方处于CLOSED 状态，就会响应rst 而不是ack。所以主动方要处于TIME\_WAIT 状态，而不能是CLOSED 。

TIME\_WAIT 并不会占用很大资源的，除非受到攻击。

还有，如果一方send 或recv 超时，就会直接进入CLOSED 状态

26.滑动窗口协议 与停止等待协议的区别。

**滑动窗口协议：**滑动窗口协议，是TCP使用的一种[流量控制](http://baike.baidu.com/view/190232.htm)方法。该协议允许发送方在停止并等待确认前可以连续发送多个分组。由于发送方不必每发一个分组就停下来等待确认，因此该协议可以加速数据的传输。可以实现流量的控制，让发送方不要发送的太快以便接收方来得及接收。

**停止等待协议：**就是发送方每发送完一次分组就停止发送，等待对方的确认，在收到确认后再发送下一个分组。

27. TCP的流量控制和拥塞控制实现原理(会画拥塞控制的典型图)。

拥塞控制是防止过多的数据注入网络，这样可以使网络中的路由器或链路不致于过载，拥塞控制要做的一个前提，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。

几种拥塞控制的方法：慢开始、拥塞避免、快重传、快恢复。

流量控制是指点对点的通信量的控制，是个端到端的问题（接收端控制发送端）。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。

28.TCP的快速重传与快速恢复算法。

<http://blog.chinaunix.net/uid-26548237-id-3966297.html>

29.TFTP 与 FTP的区别。

TFTP是一个传输文件的简单协议，它其于UDP协议而实现，但是我们也不能确定有些TFTP协议是基于其它传输协议完成的。此协议设计的时候是进行小文件传输的。因此它不具备通常的FTP的许多功能，它只能从文件服务器上获得或写入文件，不能列出目录，不进行认证，它传输8位数据。传输中有三种模式：netascii，这是8位的ASCII码形式，另一种是octet，这是8位源数据类型；最后一种mail已经不再支持，它将返回的数据直接返回给用户而不是保存为文件。

FTP 是完整、面向会话、常规用途文件传输协议。TFTP用作bones bare-特殊目的文件传输协议。

交互使用 FTP。 TFTP允许仅单向传输的文件。

FTP 依赖于 TCP， 是面向, 连接并提供可靠的控件。 TFTP 依赖 UDP， 需要减少开销, 并提供几乎没有控件。

FTP 提供身份验证。 TFTP 不。

FTP 使用已知 TCP 端口号： 20 的数据和 21 用于连接对话框。 TFTP 用于 UDP 端口号 69 其文件传输活动。

因为 TFTP 不支持验证 WindowsNT FTP 服务器服务不支持 TFTP。

30.阻塞方式和非阻塞方式，阻塞connect与非阻塞connect。(比较难，有兴趣可以了解)

<http://blog.csdn.net/bobkentblog/article/details/50850724>

31. HTTP基本格式。（java程序员必须掌握）

HTTP协议（HypertextTransferProtocol――超文本传输协议），浏览器端（客户端）向WEB服务器端访问页面的过程和HTTP协议报文的格式。

    基于HTTP协议的客户机访问包括4个过程，分别是建立TCP套接字连接、发送HTTP请求报文、接收HTTP应答报文和关闭TCP套接字连接：

1. 创建TCP套接字连接

    客户端与WEB服务器创建TCP套接字连接，其中WEB端服务器的地址可以通过域名解析确定，WEB端的套接字侦听端口一般是80。

2. 发送HTTP请求报文

    客户端向WEB服务端发送请求报文，HTTP协议的请求报文格式为：

|  |
| --- |
| 请求消息 = 请求行（实体头信息）CRLF[实体内容]  请求行 = 方法URLHTTP版本号CRLF  方法 = GET|HEAD|POST|扩展方法  URL = 协议名称 ＋ 宿主名 ＋ 目录与文件名 |

    其中"CRLF"表示回车换行。

    "请求行"中的"方法"描述了对指定资源执行的动作，常用的方法"GET"、"HEAD"和"POST"等3种，它们的含义如表15-8所示：

表15-8 HTTP请求方法

|  |  |
| --- | --- |
| 取值 | 描述 |
| GET | 从WEB服务器中获取对象，不同类型的对  象将获取不同的信息，比如：  · 文件类型对象，获取该文件的内容。  · 程序类型对象，获取该程序执行的结果。  · 数据库查询类型对象，获取该查询的结果。 |
| HEAD | 要求服务器查找对象的元信息。 |
| POST | 从客户端向WEB服务器发送数据。 |

    "实体头信息"中记载了报文的属性，利用这些信息可以实现客户端与WEB服务器之间的请求或应答，它包括报文的数据类型、压缩方法、语言、长度、压缩方法、最后一次修改时间、数据有效期等信息。

    实体内容是报文传送的附加信息，一般供POST请求填写。

GET请求报文例子

    一个URL为"<http://www.zhiliaowang.com/toupiaoceshi.asp?name>=朱云翔"的GET请求报文例子如下：

|  |
| --- |
| GET <http://www.zhiliaowang.com/toupiaoceshi.asp?name>=朱云翔 HTTP/1.1  Accept: \*/\*  Accept-Language: zh-cn  Accept-Encoding: gzip, deflate  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0;  Windows NT 5.1; SV1; .NET CLR 2.0.50727)  Host: [www.zhiliaowang.com](http://www.zhiliaowang.com/)  Connection: Keep-Alive |

    其中"/toupiaoceshi.asp?name=朱云翔"是URL信息，语句"ACCEPT：\*/\*"及其后的语句是"实体头信息"。

    注意：在语句"Connection：Keep-Alive"下的空行是必须的。

POST请求报文例子

    下面是HTTP协议中关于POST请求报文例子，节自《中国联合通信公司短消息网关系统接口协议(SGIP)》V1.2版：

|  |
| --- |
| POST /sp.cgi HTTP/1.0     /\* 请求行，服务程序为sp.cgi \*/  Host: [www.spserver.com](http://www.spserver.com/)     /\* 以下为实体头信息 \*/  Authorization: Basic <Base64 串>  Content-Length: <Content-Length>  <CommandId>=<4>       /\* 以下为POST请求的实体信息 \*/  <SequenceNumber>=<205502327125025327> /\* 实体部分输送报文 \*/  <UserNumber>=<8613001125453>  <SPNumber>=<168>  <MessageContent>=<ABCD 1234> |

    一般情况下，采用POST报文传送信息的数据存储在"实体"部分中。

请求报文附注：

    HTTP请求包括三部分：请求行(Request Line)，头部(Headers)和数据体(Body)。其中，请求行由请求方法(method)，请求网址Request-URI和协议 (Protocol)构成，而请求头包括多个属性，数据体则可以被认为是附加在请求之后的文本或二进制文件。

    下面这个例子显示了一个HTTP请求的Header内容，这些数据是真正以网络HTTP协议从IE浏览器传递到Apache服务器上的。

GET /qingdao.html HTTP/1.1

Accept:text/html, \*/\*

Accept-Language:zh-cn

Accept-Encoding:gzip,deflate

User-Agent:Mozilla/4.0(compatible;MSIE 5.01;Windows NT 5.0;DigExt)

Host: [www.6book.net](http://www.6book.net/)

Referer: <http://www.6book.net/beijing.html>

Connection:Keep-Alive

    这段程序使用了6个Header，还有一些Header没有出现。我们参考这个例子具体解释HTTP请求格式。

1.HTTP请求行：请求行格式为Method Request-URI Protocol。在上面这个例子里，"GET / HTTP/1.1"是请求行。

2.Accept:指浏览器或其他客户可以接爱的MIME文件格式。可以根据它判断并返回适当的文件格式。

3.Accept-Charset：指出浏览器可以接受的字符编码。英文浏览器的默认值是ISO-8859-1.

4.Accept-Language：指出浏览器可以接受的语言种类，如en或en-us，指英语。

5.Accept-Encoding：指出浏览器可以接受的编码方式。编码方式不同于文件格式，它是为了压缩文件并加速文件传递速度。浏览器在接收到Web响应之后先解码，然后再检查文件格式。

6.Authorization：当使用密码机制时用来标识浏览器。

7.Cache-Control：设置关于请求被代理服务器存储的相关选项。一般用不到。

8.Connection：用来告诉服务器是否可以维持固定的HTTP连接。HTTP/1.1使用Keep-Alive为默认值，这样，当浏览器需要多个文件时(比如一个HTML文件和相关的图形文件)，不需要每次都建立连接。

9.Content-Type：用来表名request的内容类型。可以用HttpServletRequest的getContentType()方法取得。

10.Cookie：浏览器用这个属性向服务器发送Cookie。Cookie是在浏览器中寄存的小型数据体，它可以记载和服务器相关的用户信息，也可以用来实现会话功能。

11.Expect：表时客户预期的响应状态。

12.From：给出客户端HTTP请求负责人的email地址。

13.Host：对应网址URL中的Web名称和端口号。

14.If-Match：供PUT方法使用。

15.If-Modified-Since：客户使用这个属性表明它只需要在指定日期之后更改过的网页。因为浏览器可以使用其存储的文件而不必从服务器请求，这样节省了Web资源。由于Servlet是动态生成的网页，一般不需要使用这个属性。

16.If-None-Match：和If-Match相反的操作，供PUT方法使用。

17.If-Unmodified-Since：和If-Match-Since相反。

18.Pragma：这个属性只有一种值，即Pragma：no-cache,表明如果servlet充当代理服务器，即使其有已经存储的网页，也要将请求传递给目的服务器。

19.Proxy-Authorization：代理服务器使用这个属性，一般用不到。

20.Range：如果客户有部分网页，这个属性可以请求剩余部分。

21.Referer：表明产生请求的网页URL。

比如从网页/beijing.html中点击一个链接到网页/qingdao.html,在向服务器发送的GET /beijing.html中的请求中，Referer是<http://www.6book.net/qingdao.html> 。这个属性可以用来跟踪Web请求是从什么网站来的。

22.Upgrage：客户通过这个属性设定可以使用与HTTP/1.1不同的协议。

23.User-Agent：是客户浏览器名称。

24.Via：用来记录Web请求经过的代理服务器或Web通道。

25.Warning：用来由客户声明传递或存储(cache)错误。

3. 接收HTTP应答报文

    WEB服务器处理客户请求，并向客户机发送应答报文，HTTP协议的应答报文格式为：

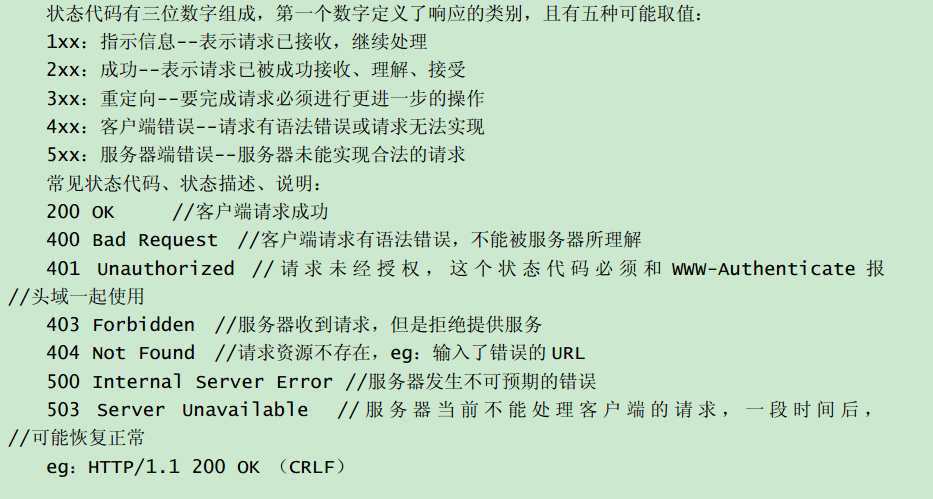
    应答报文 = 状态行（实体头信息）CRLF [实体内容]

    状态行   = HTTP版本号　状态码　原因叙述

    状态码描述了WEB服务器执行客户机请求的状态信息，其取值含义如表15-8所示：

表15-8 HTTP应答响应码含义

|  |  |
| --- | --- |
| 取值 | 描述 |
| 1×× | 保留。 |
| 2×× | 成功接收，比如“200”表示处理成功。 |
| 3×× | 客户需进一步细化请求。 |
| 4×× | 客户错误，比如“404”表示访问的  指定资源不存在。 |
| 5×× | 服务器错误。 |



GET应答报文例子

    一个请求URL为"<http://www.zhiliaowang.com/toupiaoceshi.asp?name>=朱云翔"的应答报文例子如下：

|  |
| --- |
| HTTP/1.1 200 OK  Connection: keep-alive  Date: Thu, 26 Jul 2007 14:00:02 GMT  Server: Microsoft-IIS/6.0  X-Powered-By: ASP.NET  Content-Length: 190  Content-Type: text/html  Set-Cookie: ASPSESSIONIDSAATTCSQ=JOPPKDCAMHHBEOICJPGPBJOB; path=/  Cache-control: private  <html>  <head>  <title>精通Unix下C语言编程</title>  </head>  <body>  <b>精通Unix下C语言编程与项目实战<br></b>  <b>投票测试<br></b>  感谢你为选手  朱云翔  投票!  </body>  </html> |

    语句"HTTP/1.1 200 OK"中，"200"是响应码，"OK"是原因描述。

POST应答报文例子

    下面是HTTP协议中关于POST应答报文例子，节自《中国联合通信公司短消息网关系统接口协议(SGIP)》V1.2版：

|  |
| --- |
| HTTP/1.0 200 OK       /\* 状态行，应答成功 \*/  Date: Tue, 13 Mar 2001 02:45:12 GMT /\* 以下为实体头信息 \*/  Server: Apache/1.3.12 (Unix)  Content-Type: text/html  Connection: close  <CommandId>=<80000004>     /\* 以下为POST应答的实体信息 \*/  <SequenceNumber>=<205502327125025327>  <UserNumber>=<8613001125453>  <SPNumber>=<168>  <MessageContent>=< EFGH 5678> <Result>=<0> |

    一般情况下，POST应答报文的"实体"部分存储实际传输的信息。

4、 关闭TCP套接字连接

 客户机与服务器双方关闭套接字连接，结束TCP/IP对话。

**三：Linux基本命令**

1. 绝对路径用什么符号表示？当前目录、上层目录用什么表示？主目录用什么表示? 切换目录用什么命令？

/../…/ ./ ../ / cd

2. 怎么查看当前进程？怎么执行退出？怎么查看当前路径？

ps –aux ctrl+c pwd

3. 怎么清屏？怎么退出当前命令？怎么执行睡眠？怎么查看当前用户id？查看指定帮助用什么命令？

clear quit ? id man

4. Ls 命令执行什么功能？ 可以带哪些参数，有什么区别？

查看当前路径下的文件。 -a显示隐藏文件 -l显示详细信息

5. 建立软链接(快捷方式)，以及硬链接的命令。

ln –s 源 目的 ln 源 目的

6. 目录创建用什么命令？创建文件用什么命令？复制文件用什么命令？

Mkdir touch vi cp （-r 递归） 源 目的

7. 文件权限修改用什么命令？格式是怎么样的？

chmod 777 file （4可读2可写1可执行）

8. 查看文件内容有哪些命令可以使用？

cat more less …

9. 随意写文件命令？怎么向屏幕输出带空格的字符串，比如”hello  world”?

vi

echo “hello word”;

10. 终端是哪个文件夹下的哪个文件？黑洞文件是哪个文件夹下的哪个命令？

/etc/inittab /dev/null

11. 移动文件用哪个命令？改名用哪个命令？

都是用mv

12. 复制文件用哪个命令？如果需要连同文件夹一块复制呢？如何需要有提示功能呢？

cp，如果要连同文件夹一块加一个 –r参数

13. 删除文件用哪个命令？如果需要连目录及目录下文件一块删除呢？删除空文件夹用什么命令？

rm （加 –r）rm

14. Linux下命令有哪几种可使用的通配符？分别代表什么含义?

？ \*

15. 用什么命令对一个文件的内容进行统计？(行号、单词数、字节数)

grep | wc -l（统计行数） -w （单词数） -c（字节数）-m（字符数）

16. Grep命令有什么用？ 如何忽略大小写？ 如何查找不含该串的行?

查找给定的字符串， grep –I(忽略大小写) -v （不含该串的行）

17. Linux中进程有哪几种状态？在ps显示出来的信息中，分别用什么符号表示的？

（R执行）、可中断的睡眠(S)、僵尸(Z)、不可中断的睡眠（D）、暂停或跟踪（T）

18. 怎么使一个命令在后台运行?

&

19. 利用ps怎么显示所有的进程? 怎么利用ps查看指定进程的信息？

ps –aux ps –aux|grep 进程名

20. 哪个命令专门用来查看后台任务?

jobs -l

21. 把后台任务调到前台执行使用什么命令（fg %number（非PID））?把停下的后台任务在后台执行起来用什么命令（bg %number）?

fg %number（非PID）bg %number

详解见http://blog.csdn.net/zhucanxiang/article/details/9151589

22. 终止进程用什么命令? 带什么参数?、

kill pid -9 killall

23. 怎么查看系统支持的所有信号？

uname -a

24. 搜索文件用什么命令? 格式是怎么样的?

find 路径 –name file

25. 查看当前谁在使用该主机用什么命令? 查找自己所在的终端信息用什么命令?

who who am i

26. 使用什么命令查看用过的命令列表?

history

27. 使用什么命令查看磁盘使用空间？ 空闲空间呢?

df free

28. 使用什么命令查看网络是否连通?

ping

29. 使用什么命令查看ip地址及接口信息？

ifconfig

30. 查看各类环境变量用什么命令?

env

31. 通过什么命令指定命令提示符?

table

32. 查找命令的可执行文件是去哪查找的? 怎么对其进行设置及添加?

bin常用命令sbin（有权限的使用命令）33. 通过什么命令查找执行命令?34. 怎么对命令进行取别名？

history alias（例如 alias llh = ‘ls –l /home’;）

34. 简述raid0、raid1、raid5三种工作模式的工作原理以及特点？

RAID 0：连续以位或字节为单位分割数据，并行读/写于多个磁盘上，因此具有很高的数据传输率，但它没有数据冗余，因此并不能算是真正的RAID结构。RAID 0只是单纯地提高性能，并没有为数据的可靠性提供保证，而且其中的一个磁盘失效将影响到所有数据。因此，RAID 0不能应用于数据安全性要求高的场合。

RAID 1：它是通过磁盘数据镜像实现数据冗余，在成对的独立磁盘上产生互为备份的数据。当原始数据繁忙时，可直接从镜像拷贝中读取数据，因此RAID 1可以提高读取性能。RAID 1是磁盘阵列中单位成本最高的，但提供了很高的数据安全性和可用性。当一个磁盘失效时，系统可以自动切换到镜像磁盘上读写 ，而不需要重组失效的数据。简单来说就是：镜象结构，类似于备份模式，一个数据被复制到两块硬盘上。

RAID10:高可靠性与高效磁盘结构

一个带区结构加一个镜象结构，因为两种结构各有优缺点，因此可以相互补充。

主要用于容量不大，但要求速度和差错控制的数据库中。

RAID5：分布式奇偶校验的独立磁盘结构，它的奇偶校验码存在于所有磁盘上，任何一个硬盘损坏，都可以根据其它硬盘上的校验位来重建损坏的数据。支持一块盘掉线后仍然正常运行。

35. 如何查看file1文件的第300-500行的内容？

sed –n ‘300,500p’ file1

**四：Linux编程**

1. 列举Linux内存管理相关的几个函数，各自有什么功能？

2. GCC只编译的选项是什么？只进行预处理的选项是什么？在命令行定义宏的选项是什么？

3. Linux静态库的使用，怎么创建一个静态库？ 怎么使用一个静态库？静态库文件的后缀名是什么？静态库的命名规范。

4. nm 工具的作用，ldd工具的作用。

5. Linux动态库的使用，怎么创建一个动态库？ 动态库文件的后缀名是什么？怎么使用一个动态库？动态库的命名规范？ 系统默认的动态库的查找路径？动态库显示连接所使用的系统库是什么？

6. Linux下make 与makefile。用什么参数指定makefile文件？ 什么是默认的makefile文件。

7. 在哪个文件夹下存有进程运行时的所有信息？

8. 每个程序默认打开哪三个文件设备？

9. 操作文件描述符的读写函数是什么？打开文件描述符以及关闭文件描述符的函数？

10. Fcntl函数主要的几个作用。

11. 创建进程的几个函数，及各自区别。

12. 父进程先结束，子进程会变为？ 子进程先结束，子进程会变为？

13. 怎么使父进程回收子进程？ 相关的几个函数。

14. 子进程在创建时与父进程的内存资源的复制问题。

15. 中断信号、终止信号、定时器信号的宏是什么？

16. 在程序中注册信号和发送信号用什么函数?

17. 信号的可靠与不可靠的含义。哪些信号是可靠的？哪些信号是不可靠的？

18. 信号屏蔽用什么函数？ SigSuspend的作用？ Sigpending的作用？

19. 信号处理函数处理信号时是否会被信号中断？

20. 最新版本的信号发送与处理函数？与老版本的发送与数据函数有什么区别？

21. Linux下进程有哪些通信方式(IPC)？

22. TCP服务器编程模型。

23. UDP服务器编程模型。

24. TCP编程特征，由于每次不定长数据到达，则一般使用什么选项？

25. Selected模型是什么？ 相比而言多进程实现方式有什么区别？Selected、epoll模型、poll模型的区别。

26. 常用的socket选项有哪些？利用哪个函数来设置socket选项？

27. 怎么发送和设置带外(OOB)数据？

28. Linux 下多线程的库是什么？ 利用哪个函数创建线程？利用哪个函数强制结束线程？主线程等待子线程结束时使用的函数?

29. 线程怎样在被迫退出时能做一些善后处理？以及怎么用linux特有的方式处理？

**五：数据库知识**

* 1. 能说说 数据库事务 ACID 的特性吗 ?   
       
     2、你用过 Prepared statement( 预处理语句 )? 它的作用是什么 ?   
       
     3、什么叫做数据库事务，请举例说明用法.  
       
     4、Java中访问数据库的步骤，Statement和PreparedStatement之间的区别  
       
     5、存储过程和函数的区别.  
       
     6、游标的作用？如何知道游标已经到了最后？  
       
     7、触发器分为事前触发和事后触发，这两种触发有和区别。语句级触发和行级触发有何区别  
       
     8、给你一个:驱动程序A,数据源名称为B,用户名称为C,密码为D,数据库表为T，请用JDBC检索出表T的所有数据。  
       
     9、Class.forName的作用?为什么要用?  
       
     10、内连接与外联结的区别  
       
     11、Oracle中查询前几条记录  
       
     12、某一表经常死锁，分析其原因以及解决方案  
       
     13、如果网易通行证现在有1亿用户，怎么样解决登录缓慢问题.  
       
     14、分页怎么实现的，你在项目中怎么用的，都有什么参数  
     **算法篇：算法与数据结构  
     一：算法**1. 算法的几个特征是什么。  
     2. 算法复杂性的定义。大O、θ、、小o分别表示的含义。  
     3. 递归算法的定义、递归算法的两要素。  
     4. 分治算法的思想，经典的分治算法(全排列、二分搜索、归并排序、快速排序、线性时间选择、最接近点对问题)。  
     5. 动态规划算法解题框架，动态规划算法的两个要素是什么？备忘录方法是什么？  
     6. 经典的动态规划问题(矩阵连乘问题、最长公共子序列问题、0-1背包问题)。  
     7. 贪心算法的思想，贪心算法的两个要素。  
     8. 经典的贪心问题(活动安排问题、背包问题、装载问题、哈夫曼编码、单源最短路径、最小生成树问题)。9. 回溯法的思想，回溯法中有哪两种典型的模型。  
     10. 经典的回溯算法(n后问题、0-1背包问题、旅行售货商问题)。  
     11. 分支限界法思想，有哪两种分支限界法。  
     12. 经典的分支限界算法(0-1背包问题、旅行售货商问题)。

**四：数据结构**

* 1. 数据结构的定义。

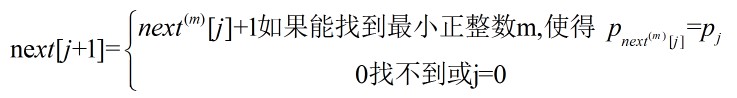
数据结构是指相互之间存在着一种或多种关系的[数据](http://baike.baidu.com/view/38752.htm)元素的集合和该集合中数据[元素](http://baike.baidu.com/view/19993.htm)之间的关系组成。记为：Data-Structure=(D,R)

其中D是数据元素的[集合](http://baike.baidu.com/view/15216.htm)，R是该集合中所有元素之间的关系的[有限集合](http://baike.baidu.com/view/514042.htm)。

* 1. 栈的两个应用：括号匹配和表达式的计算。是怎么应用的？表达式计算用的是哪种表达方式？有什么好处?

* 1. 字符串匹配算法：朴素的匹配算法、KMP算法。

KMP算法的关键是在匹配失败时，确定下一次匹配的位置，设next[j]=k，表示当模式串P中第j个字符与母串T相应字符不匹配时，模式串P中应当由第K个字符与目标串中刚不匹配的字符对齐继续进行比较。



* 1. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
  2. created:    2006/07/02
  3. filename:     KMP.cpp
  4. author:        李创
  5. <http://www.cppblog.com/converse/>
  7. 参考资料: 严蔚敏<<数据结构>>
  9. purpose:    KMP字符串匹配算法的演示
  10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
  12. #include  < stdio.h >
  13. #include  < stdlib.h >
  14. #include  < assert.h >
  15. #include  < string .h >
  17. #define  MAX\_LEN\_OF\_STR    30             //  字符串的最大长度
  19. **typedef**  **struct**  String                 //  这里需要的字符串数组,存放字符串及其长度
  20. {
  21. **char**     str[MAX\_LEN\_OF\_STR];     //  字符数组
  22. **int**         length;                     //  字符串的实际长度
  23. } String,  \* PString;
  25. //  得到字符串的next数组
  26. **void**  GetNextArray(PString pstr,  **int**  next[])
  27. {
  28. assert(NULL  !=  pstr);
  29. assert(NULL  !=  next);
  30. assert(pstr -> length  >   0 );
  32. //  第一个字符的next值是-1,因为C中的数组是从0开始的
  33. next[ 0 ]  =   - 1 ;
  34. **for**  ( **int**  i  =   0 , j  =   - 1 ; i  <  pstr -> length  -   1 ; )
  35. {
  36. //  i是主串的游标,j是模式串的游标
  37. //  这里的主串和模式串都是同一个字符串
  38. **if**  ( - 1   ==  j  ||                          //  如果模式串游标已经回退到第一个字符
  39. pstr -> str[i]  ==  pstr -> str[j])     //  如果匹配成功
  40. {
  41. //  两个游标都向前走一步
  42. ++ i;
  43. ++ j;
  44. //  存放当前的next值为此时模式串的游标值
  45. next[i]  =  j;
  46. }
  47. **else**                                  //  匹配不成功j就回退到上一个next值
  48. {
  49. j  =  next[j];
  50. }
  51. }
  52. }
  54. //  KMP字符串模式匹配算法
  55. //  输入: S是主串,T是模式串,pos是S中的起始位置
  56. //  输出: 如果匹配成功返回起始位置,否则返回-1
  57. **int**  KMP(PString S, PString T,  **int**  pos)
  58. {
  59. assert(NULL  !=  S);
  60. assert(NULL  !=  T);
  61. assert(pos  >=   0 );
  62. assert(pos  <  S -> length);
  64. **if**  (S -> length  <  T -> length)
  65. **return**   - 1 ;
  67. printf( " 主串\t = %s\n " , S -> str);
  68. printf( " 模式串\t = %s\n " , T -> str);
  70. **int**   \* next  =  ( **int**   \* )malloc(T -> length  \*   **sizeof** ( **int** ));
  71. //  得到模式串的next数组
  72. GetNextArray(T, next);
  74. **int**  i, j;
  75. **for**  (i  =  pos, j  =   0 ; i  <  S -> length  &&  j  <  T -> length; )
  76. {
  77. //  i是主串游标,j是模式串游标
  78. **if**  ( - 1   ==  j  ||                  //  模式串游标已经回退到第一个位置
  79. S -> str[i]  ==  T -> str[j])  //  当前字符匹配成功
  80. {
  81. //  满足以上两种情况时两个游标都要向前进一步
  82. ++ i;
  83. ++ j;
  84. }
  85. **else**                          //   匹配不成功,模式串游标回退到当前字符的next值
  86. {
  87. j  =  next[j];
  88. }
  89. }
  91. free(next);
  93. **if**  (j  >=  T -> length)
  94. {
  95. //  匹配成功
  96. **return**  i  -  T -> length;
  97. }
  98. **else**
  99. {
  100. //  匹配不成功
  101. **return**   - 1 ;
  102. }
  103. }

4. 二叉树前序、中序、后序递归遍历算法。二叉树前序非递归遍历算法。

二叉树节点结构如下：

typedef struct node\_t

{

char data;//数据域

struct node\_t \*lchild; //左孩子

struct node\_t \*rchild;//右孩子

}node, \*tree;

1、前序遍历

a、前序遍历递归实现：

void preorder\_traverse(tree root, void(\*visit)(node))

{

if(NULL != root)

{

visit(\*root); //先访问根节点

preorder\_traverse(root->lchild, visit);//先序遍历左子树

preorder\_traverse(root->rchild, visit);//前序遍历右子树

}

}

b、前序变量的非递归实现：

在非递归实现里面用栈来模拟递归的过程，这里用了一个线性的数组来模拟栈的功能。

void preorder\_traverse\_norec(tree root, void(\*visit(node)))

{

tree stack[32]; //这里用一个数组来模拟栈,假设节点不超过32个

int top = 0;

tree p = root;

for(; top<32; top++)

{

stack[top++] = NULL; //初始化栈

}

top = 0;

while(NULL != p || top > 0)

{

while(p != NULL)

{

visit(\*p); //先访问根节点

stack[top++] = p;//把节点压入栈中

p = p->lchild;//将p指向其左孩子

}

p = stack[—top]; //弹出p，需要将NULL弹出

p = p->rchild; //p指向其右孩子

}

}

2、二叉树的中序遍历

a、二叉树中序遍历的递归实现：

中序遍历的递归实现也比较直观，代码如下：

void inorder\_traverse(tree root, void(\*visit)(node))

{

if(NULL != root)

{

inorder\_traverse(root->lchild, visit);//先遍历其左子树

visit(\*root);//遍历该节点

inorder\_traverse(root->rchild, visit);//最后遍历右子树

}

}

b、中序遍历的非递归实现：

非递归实现同样用栈来模拟，这里还是使用线性数组来模拟栈的功能。

void inorder\_traverse\_norec(tree root, void(\*visit)(node))

{

tree stack[32];

int top = 0;

tree p = root;

for(; top<32; top++)

{

stack[top] = NULL;

}

top = 0;

while(p != NULL || top > 0)

{

while(p != NULL)

{

stack[top++] = p; //将所有左子树压入栈中

p = p->lchild;

}

p = stack[—top]; //弹出空指针

visit(\*p); //访问节点

p = p->rchild; //处理其右子树

}

}

3、后序遍历二叉树

a、后序遍历二叉树递归实现：

void postorder\_traverse(tree root, void(\*visit)(node))

{

if(NULL != root)

{

postorder\_traverse(root->lchild, visit); //遍历其左子树

postorder\_traverse(root->rchild, visit);//遍历其右子树

visit(\*root);//访问节点

}

}

b、后序遍历二叉树的非递归实现：

void postorder\_travese\_norec(tree root, void(\*visit)(node))

{

tree stack[32];

int top = 0;

tree p = root;

tree lastvisit = NULL; //用于标记上次访问的节点

for(; top<32; top++)

{

stack[top] = NULL;

}

top = 0;

while(p != NULL || top > 0)

{

while(p != NULL)

{

stack[top++] = p;  //将所有左子树压入栈中

p = p->lchild;

}

p = stack[top - 1];  //注意这里，不是p = stack[—top],因为我们还不知道是否要访问p

if(p->rchild == NULL || lastvisit == p->rchild) //判断p的右子树是否访问过或者是否为空

{

visit(\*p);//如果p的右子树为空或者已经访问过，则访问p

lastvisit = p;//标记上次访问的是节点p

top--; //这里p已经访问过，则将栈中的p弹出即可

p = NULL;

}

else

{

p = p->rchild;  //如果p的右子树还没有访问过，那么访问其右子树

}

}

}

* 1. 堆，建堆算法，堆的插入和删除算法，堆排序。

堆是个完全二叉树，而且每个节点都比左右子节点大（或小），因为堆分为max堆和min堆。

完全二叉树有个非常高效的存储方法，就是数组，一般的树都要用链表去存储。

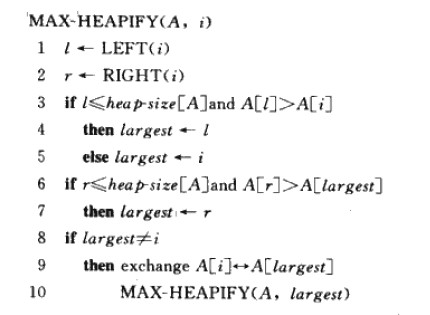
对于heap sort的输入数组，如A[16,14,10,8,7,9,3,2,4,1]，要进行堆排序，首先要建堆，建堆可以分为两步：

将输入数组抽象成完全二叉树

建堆BUILD-MAX-HEAP

开始建堆，

先来学习一个重要的堆操作MAX-HEAPIFY



这个函数就是对数组A中的第i个节点进行heapify操作

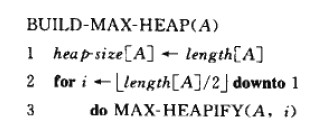
其实比较简单，1～7就是比较找出，i节点和左右子节点中，哪个最大

8～10，如果最大的不是i，那就把最大节点的和i节点交换，然后递归对从最大节点位置开始继续进行heapify

显而易见，对于n个节点的完全二叉树，高为lgn，对每个节点的heapify操作是常数级的，所以这个操作的时间复杂度就是lgn

**6.3 建堆**

那么有了heapify操作，建堆的算法很简单的，

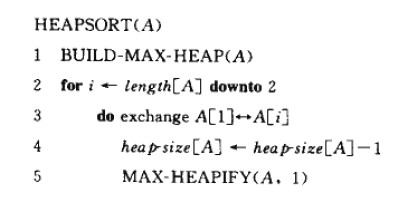


说白了，就是对i从length[A]/2到1的节点进行heapify操作。所以这个操作的时间复杂度上限咋一看应该是nlgn，其实比这个小的多，约等于2n，就是说建堆的时间复杂度是O(n)，能够在线性时间内完成，这个是很高效的。

所以我们只需要对所有非叶节点进行heapify操作就ok了

**6.4 堆排序算法**

折腾半天堆建好了，怎么堆排序了，光从堆是得不到一个有序序列的。



原理很简单，从堆我们只能知道最大的那个，那么就把最大的那个去掉，然后heapify找到第二大的，依次下去。

实现也很巧妙，没有用到额外的存储空间，把堆顶放到堆尾，然后堆size-1

这个算法的时间复杂度也是nlgn。

堆排序算法最重要的三步就是：保持堆的性质，建堆，堆排序算法

* 1. 哈希。哈希函数的有哪些种？余数的取法？ 处理冲突的方法？ 闭散列方法有哪些？

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

哈希表的做法其实很简单，就是把Key通过一个固定的算法函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将value存储在以该数字为下标的数组空间里。

构造哈希函数的方法很多，这里只介绍一些常用的，计算简便的方法。

**1.平方取中法**

算出关键字值的平方，再取其中若干位作为哈希函数值 ( 散列地址 ) 。

**2.除留余数法**

    这种方法是用模运算 (%) 得到的。设给出的关键字值为 K ，存储区单元数为 m ，则用一个小于 m 的质数 P 去除 K ，得到的余数为 R ，即： R ＝ K % P 。如果 R 落在存储区地址范围内，则 R 就取为哈希函数值 ( 散列地址 ) ；否则，再用一个线性数求出哈希函数值。

**3.数字分析法**

对各个关键字内部代码的各个码位进行分析。假设有 n 个 d 位的关键字，使用 s 个不同的符号 ( 如，对于十进制数，每一位可能出现的符号有 10 个，即 0 、 1 、 2 、…、 9) ，这 s 个不同的符号在各位上出现的频率不一定相同，它们可能在某些位上分布比较均匀，即每一个符号出现的次数都接近 n/s 次；而在另一些位上分布不均匀。这时，选取其中分布比较均匀的某些位作为哈希函数值 ( 散列地址 ) ，所选取的位数应视存储区地址范围而定，这就是数字分析法。注意：这种方法适合于关键字值中各位字符分布为已知的情况。

构造哈希函数除了上面介绍的几种常用方法外，还有截段法，即截取关键字中的某一段数码作为哈希函数；分段迭加法，即把关键字的机内代码分成几段，再进行迭加 ( 可以是算术加，也可以是按位加 ) 得到哈希函数值。解决哈希（HASH）冲突的主要方法

* 1. *开放地址法*

这个方法的基本思想是：当发生地址冲突时，按照某种方法继续探测哈希表中的其他存储单元，直到找到空位置为止。这个过程可用下式描述：

H i ( key ) = ( H ( key )+ d i ) mod m ( i = 1,2,…… ， k ( k ≤ m – 1))

其中： H ( key ) 为关键字 key 的直接哈希地址， m 为哈希表的长度， di 为每次再探测时的地址增量。

采用这种方法时，首先计算出元素的直接哈希地址 H ( key ) ，如果该存储单元已被其他元素占用，则继续查看地址为 H ( key ) + d 2 的存储单元，如此重复直至找到某个存储单元为空时，将关键字为 key 的数据元素存放到该单元。

增量 d 可以有不同的取法，并根据其取法有不同的称呼：

（ 1 ） d i ＝ 1 ， 2 ， 3 ， …… 线性探测再散列；

（ 2 ） d i ＝ 1^2 ，－ 1^2 ， 2^2 ，－ 2^2 ， k^2， -k^2…… 二次探测再散列；

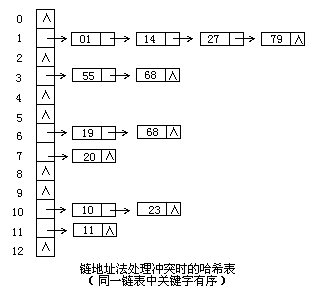
（ 3 ） d i ＝ 伪随机序列 伪随机再散列；

*b)再哈希法*

当发生冲突时，使用第二个、第三个、哈希函数计算地址，直到无冲突时。缺点：计算时间增加。

*c)链地址法*

将所有关键字为同义词的记录存储在同一线性链表中。如下：



因此这种方法，可以近似的认为是筒子里面套筒子

**与开放定址法相比，拉链法有如下几个优点：**

①拉链法处理冲突简单，且无堆积现象，即非同义词决不会发生冲突，因此平均查找长度较短；

②由于拉链法中各链表上的结点空间是动态申请的，故它更适合于造表前无法确定表长的情况；

③开放定址法为减少冲突，要求装填因子α较小，故当结点规模较大时会浪费很多空间。而拉链法中可取α≥1，且结点较大时，拉链法中增加的指针域可忽略不计，因此节省空间；

④在用拉链法构造的散列表中，删除结点的操作易于实现。只要简单地删去链表上相应的结点即可。而对开放地址法构造的散列表，删除结点不能简单地将被删结 点的空间置为空，否则将截断在它之后填人散列表的同义词结点的查找路径。这是因为各种开放地址法中，空地址单元(即开放地址)都是查找失败的条件。因此在 用开放地址法处理冲突的散列表上执行删除操作，只能在被删结点上做删除标记，而不能真正删除结点。

（3）拉链法的缺点

    拉链法的缺点是：指针需要额外的空间，故当结点规模较小时，开放定址法较为节省空间，而若将节省的指针空间用来扩大散列表的规模，可使装填因子变小，这又减少了开放定址法中的冲突，从而提高平均查找速度。

**冲突解决策略/闭散列方法**

闭散列方法把所有记录直接存储在散列表中。每个记录关键码key有一个由散列函数计算出来的基位置，即h(key)。如果要插入一个关键码，而另一个记录已经占据了R的基位置(发生碰撞)，那么就把R存储在表中的其它地址内，由冲突解决策略确定是哪个地址。

    闭散列表解决冲突的基本思想是：当冲突发生时，使用某种方法为关键码K生成一个散列地址序列d0，d1，d2，... di ，...dm-1。其中d0=h（K）称为K的基地址地置( home position )；所有di(0< i< m)是后继散列地址。当插入K时，若基地址上的结点已被别的数据元素占用，则按上述地址序列依次探查，将找到的第一个开放的空闲位置di作为K的存储位置；若所有后继散列地址都不空闲，说明该闭散列表已满，报告溢出。相应地，检索K时，将按同值的后继地址序列依次查找，检索成功时返回该位置di ；如果沿着探查序列检索时，遇到了开放的空闲地址，则说明表中没有待查的关键码。删除K时，也按同值的后继地址序列依次查找，查找到某个位置di具有该K值，则删除该位置di上的数据元素（删除操作实际上只是对该结点加以删除标记）；如果遇到了开放的空闲地址，则说明表中没有待删除的关键码。因此，对于闭散列表来说，构造后继散列地址序列的方法，也就是处理冲突的方法。

    形成探查的方法不同，所得到的解决冲突的方法也不同。下面是几种常见的构造方法。

    1、线性探查法

    将散列表看成是一个环形表，若在基地址d（即h(K)=d）发生冲突，则依次探查下述地址单元：d+1，d+2，......，M-1，0，1，......，d-1直到找到一个空闲地址或查找到关键码为key的结点为止。当然，若沿着该探查序列检索一遍之后，又回到了地址d，则无论是做插入操作还是做检索操作，都意味着失败。

    用于简单线性探查的探查函数是： p(K，i) = i

    例9.7 已知一组关键码为（26，36，41，38，44，15，68，12，06，51，25），散列表长度M= 15，用线性探查法解决冲突构造这组关键码的散列表。

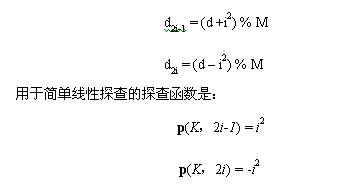
    因为n=11，利用除余法构造散列函数，选取小于Ｍ的最大质数P=13，则散列函数为：h(key) = key%13。按顺序插入各个结点： 26: h(26) = 0，36: h(36) = 10， 41: h(41) = 2，38: h(38) = 12， 44: h(44) = 5。

    插入15时，其散列地址为2，由于2已被关键码为41的元素占用，故需进行探查。按顺序探查法，显然3为开放的空闲地址，故可将其放在3单元。类似地，68和12可分别放在4和13单元中，下图显示了插入15和68时的过程。

C:\4C695045\0E2AB6F3-F9DA-47DC-9631-5A563F3DE12A.files\image013.gif

    2、二次探查法

    二次探查法的基本思想是：生成的后继散列地址不是连续的，而是跳跃式的，以便为后续数据元素留下空间从而减少聚集。二次探查法的探查序列依次为：12，-12，22 ，-22，...等，也就是说，发生冲突时，将同义词来回散列在第一个地址的两端。求下一个开放地址的公式为：



    3、随机探查法

    理想的探查函数应当在探查序列中随机地从未访问过的槽中选择下一个位置，即探查序列应当是散列表位置的一个随机排列。但是，我们实际上不能随机地从探查序列中选择一个位置，因为在检索关键码的时候不能建立起同样的探查序列。然而，我们可以做一些类似于伪随机探查( pseudo-random probing )的事情。在伪随机探查中，探查序列中的第i个槽是(h(K) + ri) mod M，其中ri是1到M - 1之间数的“随机”数序列。所有插入和检索都使用相同的“随机”数。探查函数将是 p(K，i) = perm[i - 1]， 这里perm是一个长度为M - 1的数组，它包含值从1到M – 1的随机序列。

    4、双散列探查法

    伪随机探查和二次探查都能消除基本聚集——即基地址不同的关键码，其探查序列的某些段重叠在一起——的问题。然而，如果两个关键码散列到同一个基地址，那么采用这两种方法还是得到同样的探查序列，仍然会产生聚集。这是因为伪随机探查和二次探查产生的探查序列只是基地址的函数，而不是原来关键码值的函数。这个问题称为二级聚集( secondary clustering )。

    为了避免二级聚集，我们需要使得探查序列是原来关键码值的函数，而不是基位置的函数。双散列探查法利用第二个散列函数作为常数，每次跳过常数项，做线性探查。

* 1. 二叉搜索树的搜索、插入、删除。时间复杂度。

**二叉查找树**（*Binary Search Tree*），或者是一棵空树，或者是具有下列性质的[二叉树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%A0%91)：

* 1. 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；
  2. 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；
  3. 它的左、右子树也分别为二叉排序树。

二叉排序树的查找过程和[次优二叉树](http://zh.wikipedia.org/w/index.php?title=%E6%AC%A1%E4%BC%98%E4%BA%8C%E5%8F%89%E6%A0%91&action=edit&redlink=1)类似，通常采取二叉[链表](http://zh.wikipedia.org/wiki/%E9%93%BE%E8%A1%A8)作为二叉排序树的[存储结构](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)。中序遍历二叉排序树可得到一个关键字的有序序列，一个无序序列可以通过构造一棵二叉排序树变成一个有序序列，构造树的过程即为对无序序列进行排序的过程。每次插入的新的结点都是二叉排序树上新的叶子结点，在进行插入操作时，不必移动其它结点，只需改动某个结点的指针，由空变为非空即可。搜索,插入,删除的复杂度等于树高，期望O(logn),最坏O(n)(数列有序,树退化成线性表).

虽然二叉排序树的最坏效率是O(n),但它支持动态查询,且有很多改进版的二叉排序树可以使树高为O(logn),如[SBT](http://zh.wikipedia.org/wiki/SBT),[AVL](http://zh.wikipedia.org/wiki/AVL),[红黑树](http://zh.wikipedia.org/wiki/%E7%BA%A2%E9%BB%91%E6%A0%91)等.

在二叉查找树b中查找x的过程为：

* 1. 若b是空树，则搜索失败，否则：
  2. 若x等于b的根节点的数据域之值，则查找成功；否则：
  3. 若x小于b的根节点的数据域之值，则搜索左子树；否则：
  4. 查找右子树。

*/\* 以下代码为C++写成, 下同 \*/*

Status SearchBST(BiTree T, KeyType key, BiTree f, BiTree &p){

//在根指针T所指二元排序樹中递归地查找其關键字等於key的數據元素，若查找成功，

//則指针p指向該數據元素節點，并返回TRUE，否則指针指向查找路徑上訪問的最後

//一個節點并返回FALSE，指针f指向T的雙親，其初始调用值為NULL

if(!T) { //查找不成功

p=f;

return false;

}

else if (key == T->data.key) { //查找成功

p=T;

return true;

}

else if (key < T->data.key) //在左子樹中繼續查找

return SearchBST(T->lchild, key, T, p);

else //在右子樹中繼續查找

return SearchBST(T->rchild, key, T, p);

}

向一个二叉查找树b中插入一个节点s的算法，过程为：

* 1. 若b是空树，则将s所指结点作为根节点插入，否则：
  2. 若s->data等于b的根节点的数据域之值，则返回，否则：
  3. 若s->data小于b的根节点的数据域之值，则把s所指节点插入到左子树中，否则：
  4. 把s所指节点插入到右子树中。

*/\*当二元搜尋樹T中不存在关键字等于e.key的数据元素时，插入e并返回TRUE，否则返回FALSE\*/*

Status InsertBST(BiTree &T, ElemType e){

if(!SearchBST(T, e.key, NULL,p){

s = new BiTNode;

s->data = e; s->lchild = s->rchild = NULL;

if(!p)

T=s; //被插節点\*s为新的根结点

else if (e.key < p->data.key)

p->lchild = s; //被插節点\*s为左孩子

else

p->rchild = s; //被插節点\*s为右孩子

return true;

}

else

return false; //树中已有关键字相同的節点，不再插入

}

在二叉排序树删除结点的算法[[编辑](http://zh.wikipedia.org/w/index.php?title=%E4%BA%8C%E5%85%83%E6%90%9C%E5%B0%8B%E6%A8%B9&action=edit&section=3)]

在二叉排序树删去一个结点，分三种情况讨论：

* 1. 若\*p结点为叶子结点，即PL(左子树)和PR(右子树)均为空树。由于删去叶子结点不破坏整棵树的结构，则只需修改其双亲结点的指针即可。
  2. 若\*p结点只有左子树PL或右子树PR，此时只要令PL或PR直接成为其双亲结点\*f的左子树（当\*p是左子树）或右子树（当\*p是右子树）即可，作此修改也不破坏二叉排序树的特性。
  3. 若\*p结点的左子树和右子树均不空。在删去\*p之后，为保持其它元素之间的相对位置不变，可按中序遍历保持有序进行调整，可以有两种做法：其一是令\*p的左子树为\*f的左/右(依\*p是\*f的左子树还是右子树而定)子树，\*s为\*p左子树的最右下的结点，而\*p的右子树为\*s的右子树；其二是令\*p的直接前驱（或直接后继）替代\*p，然后再从二叉排序树中删去它的直接前驱（或直接后继）。在二叉排序树上删除一个结点的算法如下：

Status DeleteBST(BiTree &T, KeyType key){

//若二叉排序树T中存在关键字等于key的数据元素时，则删除该数据元素，并返回

//TRUE；否则返回FALSE

if(!T)

return false; //不存在关键字等于key的数据元素

else{

if(key == T->data.key) { // 找到关键字等于key的数据元素

return Delete(T);

}

else if(key < T->data.key)

return DeleteBST(T->lchild, key);

else

return DeleteBST(T->rchild, key);

}

}

Status Delete(BiTree &p){

//该节点为叶子节点，直接删除

if (!p->rchild && !p->lchild)

{

delete p;

}

else if(!p->rchild){ //右子树空则只需重接它的左子树

q=p->lchild;

p->data = p->lchild->data;

p->lchild=p->lchild->lchild;

p->rchild=p->lchild->rchild;'

delete q;

}

else if(!p->lchild){ //左子树空只需重接它的右子树

q=p->lchild;

p->data = p->rchild->data;

p->lchild=p->rchild->lchild;

p->rchild=p->rchild->rchild;'

delete q; }

else{ //左右子树均不空

q=p;

s=p->lchild;

while(s->rchild){

q=s;

s=s->rchild;

} //转左，然后向右到尽头

p->data = s->data; //s指向被删结点的“前驱”

if(q!=p)

q->rchild = s->lchild; //重接\*q的右子树

else

q->lchild = s->lchild; //重接\*q的左子树

delete s;

}

return true;

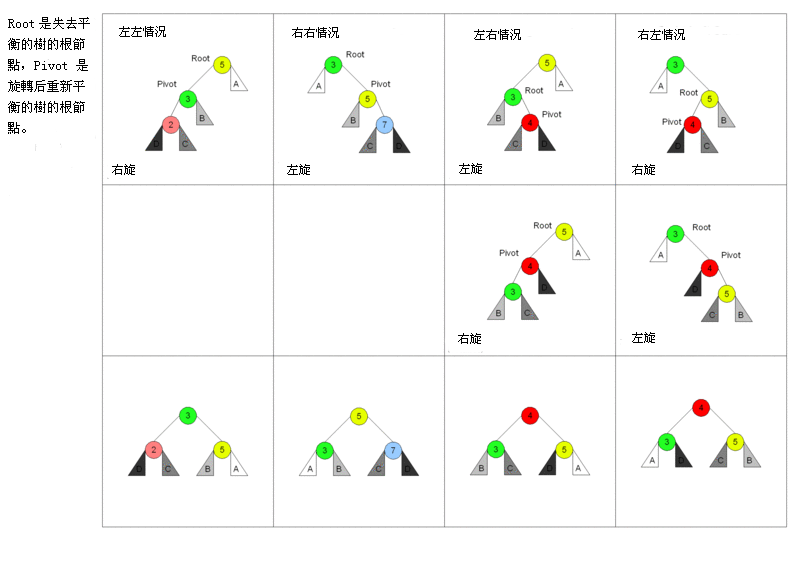
}

8. 二叉平衡树的插入结点的原理，有哪几种旋转方式？分别适用于哪种情况。分析二叉平衡树的时间复杂度。

平衡二叉树关于树的深度是平衡的，具有较高的检索效率。平衡二叉树或是一棵空树，或是具有下列性质的二叉排序树：其左子树和右子树都是平衡二叉树，而且左右子树深度之差绝对值不超过1. 由此引出了平衡因子（balance factor）的概念，bf定义为该结点的左子树的深度减去右子树的深度（有些书是右子树深度减去左子树深度，我是按照左子树减去右子树来计算的，下面的代码也是这样定义的），所以平衡二叉树的结点的平衡因子只可能是 -1，0，1 ，某个结点的平衡因子绝对值大于1，该二叉树就不平衡。

平衡二叉树在出现不平衡状态的时候，要进行平衡旋转处理，有四种平衡旋转处理（单向右旋处理，单向左旋处理，双向旋转（先左后右）处理，双向旋转（先右后左）处理），归根到底是两种（单向左旋处理和单向右旋处理）。

AVL树的基本操作一般涉及运作同在不平衡的[二叉查找树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)所运作的同样的算法。但是要进行预先或随后做一次或多次所谓的"AVL旋转"。

[](http://zh.wikipedia.org/wiki/File:Tree_Rebalancing.png)

**平衡二叉搜索树**(**Balanced Binary Tree**)是一种结构平衡的[二叉搜索树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)，即叶节点深度差不超过1，它能在[O](http://zh.wikipedia.org/wiki/%E5%A4%A7O%E7%AC%A6%E5%8F%B7)(log *n*)内完成插入、查找和删除操作。

9. 红黑树的定义，红黑树的性能分析和与二叉平衡树的比较。

**红黑树**是一种[自平衡二叉查找树](http://zh.wikipedia.org/wiki/%E8%87%AA%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)，是在[计算机科学](http://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6)中用到的一种[数据结构](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)，典型的用途是实现[关联数组](http://zh.wikipedia.org/wiki/%E5%85%B3%E8%81%94%E6%95%B0%E7%BB%84)。它是在[1972年](http://zh.wikipedia.org/wiki/1972%E5%B9%B4)由[鲁道夫·贝尔](http://zh.wikipedia.org/wiki/%E9%B2%81%E9%81%93%E5%A4%AB%C2%B7%E8%B4%9D%E5%B0%94)发明的，他称之为"对称二叉B树"，它是复杂的，但它的操作有着良好的最坏情况[运行时间](http://zh.wikipedia.org/wiki/%E7%AE%97%E6%B3%95%E5%88%86%E6%9E%90)，并且在实践中是高效的: 它可以在[O](http://zh.wikipedia.org/wiki/%E5%A4%A7O%E7%AC%A6%E5%8F%B7)(log *n*)时间内做查找，插入和删除，这里的*n*是树中元素的数目。

红黑树是每个节点都带有*颜色*属性的[二叉查找树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%85%83%E6%90%9C%E5%B0%8B%E6%A8%B9)，颜色为*红色*或*黑色*。在二叉查找树强制一般要求以外，对于任何有效的红黑树我们增加了如下的额外要求:

性质1. 节点是红色或黑色。

性质2. 根是黑色。

性质3. 所有叶子都是黑色（叶子是NIL节点）。

性质4. 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)

性质5. 从任一节点到其每个叶子的所有[简单路径](http://zh.wikipedia.org/wiki/%E9%81%93%E8%B7%AF_(%E5%9B%BE%E8%AE%BA))都包含相同数目的黑色节点。

这些约束强制了红黑树的关键性质: 从根到叶子的最长的可能路径不多于最短的可能路径的两倍长。结果是这个树大致上是平衡的。因为操作比如插入、删除和查找某个值的最坏情况时间都要求与树的高度成比例，这个在高度上的理论上限允许红黑树在最坏情况下都是高效的，而不同于普通的[二叉查找树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)。

10. 图有哪些储存表示。

图的存储表示[[编辑](http://zh.wikipedia.org/w/index.php?title=%E5%9B%BE&action=edit&section=9)]

* 1. 数组（[邻接矩阵](http://zh.wikipedia.org/wiki/%E9%82%BB%E6%8E%A5%E7%9F%A9%E9%98%B5)）存储表示（有向或无向）
  2. [邻接表](http://zh.wikipedia.org/wiki/%E9%82%BB%E6%8E%A5%E8%A1%A8)存储表示
  3. [前向星](http://zh.wikipedia.org/w/index.php?title=%E5%89%8D%E5%90%91%E6%98%9F&action=edit&redlink=1)存储表示
  4. [有向图](http://zh.wikipedia.org/wiki/%E6%9C%89%E5%90%91%E5%9B%BE)的[十字链表](http://zh.wikipedia.org/w/index.php?title=%E5%8D%81%E5%AD%97%E9%93%BE%E8%A1%A8&action=edit&redlink=1)存储表示
  5. [无向图](http://zh.wikipedia.org/wiki/%E7%84%A1%E5%90%91%E5%9C%96)的[邻接多重表](http://zh.wikipedia.org/w/index.php?title=%E9%82%BB%E6%8E%A5%E5%A4%9A%E9%87%8D%E8%A1%A8&action=edit&redlink=1)存储表示

11. 链表插入排序、链表归并排序。

12. 常见的有哪几种排序算法，试比较其时间复杂度，以及是否稳定，及各自使用的情形。

常用的排序算法的时间复杂度和空间复杂度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 排序法 | 最差时间分析 | 平均时间复杂度 | 稳定度 | 空间复杂度 |
| 冒泡排序 | O(n2) | O(n2) | 稳定 | O(1) |
| 快速排序 | O(n2) | O(n\*log2n) | 不稳定 | O(log2n)~O(n) |
| 选择排序 | O(n2) | O(n2) | 稳定 | O(1) |
| 二叉树排序 | O(n2) | O(n\*log2n) | 不一顶 | O(n) |
| 插入排序 | O(n2) | O(n2) | 稳定 | O(1) |
| 堆排序 | O(n\*log2n) | O(n\*log2n) | 不稳定 | O(1) |
| 希尔排序 | O | O | 不稳定 | O(1) |

13. 常用分配排序有哪几种？ 基数排序的定义，分类及原理。

所谓排序，就是要将一堆记录，使之按关键字递增(或递减)次序排列起来。根据排序所采用的策略，可以分为如下五种：

1、[插入排序（直接插入排序、希尔排序）](http://www.cppblog.com/kesalin/archive/2011/03/03/insert_sort.html)

2、[交换排序（冒泡排序、快速排序）](http://www.cppblog.com/kesalin/archive/2011/03/04/exchange_sort.html)

3、[选择排序（直接选择排序、堆排序）](http://www.cppblog.com/kesalin/archive/2011/03/09/select_sort.html)

    4、[归并排序](http://www.cppblog.com/kesalin/archive/2011/03/13/merge_sort.html)

5、桶排序（桶排序，基数排序）；

**基数排序**（[英语](http://zh.wikipedia.org/wiki/%E8%8B%B1%E8%AF%AD)：Radix sort）是一种非比较型[整数](http://zh.wikipedia.org/wiki/%E6%95%B4%E6%95%B0)[排序算法](http://zh.wikipedia.org/wiki/%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95)，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。由于整数也可以表达字符串（比如名字或日期）和特定格式的浮点数，所以基数排序也不是只能使用于整数。

它是这样实现的：将所有待比较数值（正整数）统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后, 数列就变成一个有序序列。

基数排序的方式可以采用LSD（Least significant digital）或MSD（Most significant digital），LSD的排序方式由键值的最右边开始，而MSD则相反，由键值的最左边开始。

14. 外部排序的过程。

**外排序**（External sorting）是指能够处理极大量[数据](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE)的[排序算法](http://zh.wikipedia.org/wiki/%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95)。通常来说，外排序处理的数据不能一次装入[内存](http://zh.wikipedia.org/wiki/%E5%86%85%E5%AD%98)，只能放在读写较慢的[外存储器](http://zh.wikipedia.org/w/index.php?title=%E5%A4%96%E5%AD%98%E5%82%A8%E5%99%A8&action=edit&redlink=1)（通常是[硬盘](http://zh.wikipedia.org/wiki/%E7%A1%AC%E7%9B%98)）上。外排序通常采用的是一种“排序-[归并](http://zh.wikipedia.org/w/index.php?title=%E5%BD%92%E5%B9%B6&action=edit&redlink=1)”的策略。在排序阶段，先读入能放在内存中的数据量，将其排序输出到一个临时文件，依此进行，将待排序数据组织为多个有序的临时文件。尔后在归并阶段将这些临时文件组合为一个大的有序文件，也即排序结果。

外排序的一个例子是外[归并排序](http://zh.wikipedia.org/wiki/%E5%BD%92%E5%B9%B6%E6%8E%92%E5%BA%8F)（External merge sort），它读入一些能放在内存内的数据量，在内存中排序后输出为一个顺串（即是内部数据有序的临时文件），处理完所有的数据后再进行归并。[[1]](http://zh.wikipedia.org/wiki/%E5%A4%96%E6%8E%92%E5%BA%8F#cite_note-1)[[2]](http://zh.wikipedia.org/wiki/%E5%A4%96%E6%8E%92%E5%BA%8F#cite_note-2)比如，要对 900 [MB](http://zh.wikipedia.org/wiki/%E5%85%86%E5%AD%97%E8%8A%82) 的数据进行排序，但机器上只有 100 MB 的可用内存时，外归并排序按如下方法操作：

* 1. 读入 100 MB 的数据至内存中，用某种常规方式（如[快速排序](http://zh.wikipedia.org/wiki/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F)、[堆排序](http://zh.wikipedia.org/wiki/%E5%A0%86%E6%8E%92%E5%BA%8F)、[归并排序](http://zh.wikipedia.org/wiki/%E5%BD%92%E5%B9%B6%E6%8E%92%E5%BA%8F)等方法）在内存中完成排序。
  2. 将排序完成的数据写入磁盘。
  3. 重复步骤 1 和 2 直到所有的数据都存入了不同的 100 MB 的块（临时文件）中。在这个例子中，有 900 MB 数据，单个临时文件大小为 100 MB，所以会产生 9 个临时文件。
  4. 读入每个临时文件（顺串）的前 10 MB （ = 100 MB / (9 块 + 1)）的数据放入内存中的输入[缓冲区](http://zh.wikipedia.org/wiki/%E7%BC%93%E5%86%B2%E5%8C%BA)，最后的 10 MB 作为输出缓冲区。（实践中，将输入缓冲适当调小，而适当增大输出缓冲区能获得更好的效果。）
  5. 执行九路[归并](http://zh.wikipedia.org/w/index.php?title=%E5%BD%92%E5%B9%B6&action=edit&redlink=1)算法，将结果输出到输出缓冲区。一旦输出缓冲区满，将缓冲区中的数据写出至目标文件，清空缓冲区。一旦9个输入缓冲区中的一个变空，就从这个缓冲区关联的文件，读入下一个10M数据，除非这个文件已读完。这是“外归并排序”能在主存外完成排序的关键步骤 -- 因为“归并算法”(merge algorithm)对每一个大块只是顺序地做一轮访问(进行归并)，每个大块不用完全载入主存。

15. B树、B+树、Trie的概念及用途，添加删除结点的原理。

**B树**（B-tree）是，存储排序数据并允许以O(log n)的运行时间进行查找，顺序读取，插入和删除的数据结构。B树，概括来说是一个节点可以拥有多于2个子节点的[二叉查找树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)。与[自平衡二叉查找树](http://zh.wikipedia.org/wiki/%E8%87%AA%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)不同，B-树为系统最优化大块数据的读和写操作。B-tree算法减少定位记录时所经历的中间过程，从而加快存取速度。普遍运用在[数据库](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93)和[文件系统](http://zh.wikipedia.org/wiki/%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F)。

**B+ 树**是一种[树数据结构](http://zh.wikipedia.org/wiki/%E6%A0%91_(%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84))，通常用于[数据库](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93)和[操作系统](http://zh.wikipedia.org/wiki/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)的[文件系统](http://zh.wikipedia.org/wiki/%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F)中。B+ 树的特点是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度。B+ 树元素自底向上插入，这与[二叉树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%A0%91)恰好相反。

B+ 树在节点访问时间远远超过节点内部访问时间的时候，比可作为替代的实现有着实在的优势。这通常在多数节点在[次级存储](http://zh.wikipedia.org/w/index.php?title=%E6%AC%A1%E7%BA%A7%E5%AD%98%E5%82%A8&action=edit&redlink=1)比如[硬盘](http://zh.wikipedia.org/wiki/%E7%A1%AC%E7%9B%98)中的时候出现。通过最大化在每个[内部节点](http://zh.wikipedia.org/w/index.php?title=%E5%86%85%E9%83%A8%E8%8A%82%E7%82%B9&action=edit&redlink=1)内的[子节点](http://zh.wikipedia.org/w/index.php?title=%E5%AD%90%E8%8A%82%E7%82%B9&action=edit&redlink=1)的数目减少树的高度，平衡操作不经常发生，而且效率增加了。这种价值得以确立通常需要每个节点在次级存储中占据完整的[磁盘块](http://zh.wikipedia.org/w/index.php?title=%E7%A3%81%E7%9B%98%E5%9D%97&action=edit&redlink=1)或近似的大小。

B+ 背后的想法是内部节点可以有在预定范围内的可变数目的子节点。因此，B+ 树不需要象其他[自平衡二叉查找树](http://zh.wikipedia.org/wiki/%E8%87%AA%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)那样经常的重新平衡。对于特定的实现在子节点数目上的低和高边界是固定的。例如，在 2-3 B 树（常简称为**2-3 树**）中，每个内部节点只可能有 2 或 3 个子节点。如果节点有无效数目的子节点则被当作处于违规状态。

**trie**，又称**前缀树**，是一种有序[树](http://zh.wikipedia.org/wiki/%E6%A0%91_(%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84))，用于保存[关联数组](http://zh.wikipedia.org/wiki/%E5%85%B3%E8%81%94%E6%95%B0%E7%BB%84)，其中的键通常是[字符串](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E4%B8%B2)。与[二叉查找树](http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)不同，键不是直接保存在节点中，而是由节点在树中的位置决定。一个节点的所有子孙都有相同的[前缀](http://zh.wikipedia.org/wiki/%E5%89%8D%E7%BC%80)，也就是这个节点对应的字符串，而根节点对应[空字符串](http://zh.wikipedia.org/wiki/%E7%A9%BA%E5%AD%97%E5%85%83%E4%B8%B2)。一般情况下，不是所有的节点都有对应的值，只有叶子节点和部分内部节点所对应的键才有相关的值。

**五：数据库知识**

* 1. 能说说数据库事务ACID的特性吗 ?

事务(Transaction)是并发控制的基本单位，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。例如，银行转帐工作：从一个帐号扣款并使另一个帐号增款，这两个操作要么都执行，要么都不执行。

数据库事务必须具备ACID特性，ACID是Atomic（原子性）、Consistency（一致性）、Isolation（隔离性）和Durability（持久性）的英文缩写。

原子性：指整个数据库事务是不可分割的工作单位。只有使据库中所有的操作执行成功，才算整个事务成功；事务中任何一个SQL语句执行失败，那么已经执行成功的SQL语句也必须撤销，数据库状态应该退回到执行事务前的状态。

一致性：指数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。例如对银行转帐事务，不管事务成功还是失败，应该保证事务结束后ACCOUNTS表中Tom和Jack的存款总额为2000元。

隔离性：指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看到中间状态的数据。

持久性：指的是只要事务成功结束，它对数据库所做的更新就必须永久保存下来。即使发生系统崩溃，重新启动数据库系统后，数据库还能恢复到事务成功结束时的状态。

事务的（ACID）特性是由关系数据库管理系统（RDBMS，数据库系统）来实现的。数据库管理系统采用日志来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。

数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。

2、你用过 PreparedStatement( 预处理语句 )? 它的作用是什么 ?

①用PreparedStatement替换Statement的作用主要是为了防止数据库注入式攻击（直接使用String作为sql的参数导致的潜在危险）。原理是用占位符（？）作为实参定义sql语句。②PreparedStatement是预编译的,支持批处理，对于批量处理可以大大提高效率. 也叫JDBC存储过程。

⑤PreparedStatement对象不仅包含了SQL语句，而且大多数情况下这个语句已经被预编译过，因而当其执行时，只需DBMS运行SQL语句，而不必先编译。当你需要执行Statement对象多次的时候，PreparedStatement对象将会大大降低运行时间，当然也加快了访问数据库的速度。这种转换也给你带来很大的便利，不必重复SQL语句的句法，而只需更改其中变量的值，便可重新执行SQL语句。

3、什么叫做数据库事务，请举例说明用法.

事务(Transaction)是并发控制的基本单位。所谓事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。例如，银行转帐工作：从一个帐号扣款并使另一个帐号增款，这两个操作要么都执行，要么都不执行。

4、Java中访问数据库的步骤，Statement和PreparedStatement之间的区别

①PreparedStatement支持动态设置参数，Statement不支持。

②PreparedStatement可避免如类似单引号的编码麻烦，Statement不可以。

③PreparedStatement支持预编译，Statement不支持。

④在sql语句出错时PreparedStatement不易检查，而Statement则更便于查错。

⑤PreparedStatement可防止Sql助于，更加安全，而Statement不行。

5、存储过程和函数的区别.

①存储过程是用户定义的一系列SQL语句的集合，涉及特定的表或其他对象的任务，用户可以调用存储过程。而函数通常是数据库已定义的方法，它接收参数并返回某种类型的值，而且不涉及特定用户表。

②存储过程功能强大，限制少。可以在单个存储过程中执行一系列 SQL 语句，可以从自己的存储过程内引用其它存储过程，这可以简化一系列复杂语句。

③存储过程在创建时进行编译，所以执行起来比单个 SQL 语句快。可以有多个返回值，即多个输出参数，并且可以使用SELECT返回结果集。

④而函数有限制，只能返回一个标量。自定义函数诸多限制，有许多语句不能使用，许多功能不能实现。函数可以直接引用返回值，用表变量返回记录集。但是，用户定义函数不能用于执行一组修改全局数据库状态的操作。

6、游标的作用？如何知道游标已经到了最后？

游标的作用用于定位结果集的行。通过判断全局变量@@FETCH\_STATUS，可以判断游标是否到了最后，通常此变量不等于0表示出错或到了最后。

7、触发器分为事前触发和事后触发，这两种触发有和区别。语句级触发和行级触发有何区别？

事前触发器运行于触发事件发生之前，而事后触发器运行于触发事件发生之后。通常，事前触发器可以获取事件之前和新的字段值。语句级触发可以在语句执行之前或执行之后执行，而行级触发在触发器所影响的每一行触发一次。

8、给你一个:驱动程序A,数据源名称为B,用户名称为C,密码为D,数据库表为T，请用JDBC检索出表T的所有数据。

public class TestPrepStmt {

public static void main(String[] args) {

String sql="select \* from T";

Statement stmt = null;

Connection conn = null;

ResultSet rs=null;

try {

Class.forName("A");

conn=DriverManager.getConnection("B", "C", "D");

stmt = conn.createStatement();

rs=statement.executeQuery(sql);

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(stmt != null) {

stmt.close();

stmt = null;

}

if(conn != null) {

conn.close();

conn = null;

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

9、Class.forName的作用?为什么要用?

（1）按参数中指定的字符串形式的类名去搜索并加载相应的类，如果该类字节码已经被加载过，则返回代表该字节码的Class实例对象，否则，按类加载器的委托机制去搜索和加载该类，如果所有的类加载器都无法加载到该类，则抛出ClassNotFoundException。加载完这个Class字节码后，接着就可以使用Class字节码的newInstance方法去创建该类的实例对象了。

（2）有时候，我们程序中所有使用的具体类名在设计时（即开发时）无法确定，只有程序运行时才能确定，这时候就需要使用Class.forName去动态加载该类，这个类名通常是在配置文件中配置的，例如，spring的ioc中每次依赖注入的具体类就是这样配置的，jdbc的驱动类名通常也是通过配置文件来配置的，以便在产品交付使用后不用修改源程序就可以更换驱动类名。

10、内连接与外连接、全连接的区别

内连接：两表连接，相同部分才显示

外连接：两表连接，一表为主表，主表全部显示，一表为从表，从表中与主表不相等的部分不显示

全连接：两表连接，两表平等，连接结果全部显示，包括A表中有而B表中没有，A表有B表有，A表没有而B表

11、Oracle中查询前几条记录

select \* from table where rownum <= n；

附加：

查询大于2的记录

select rownum,id,age,name from(select rownum no ,id,age,name from loaddata) where no > 2;

查询n到m行记录的通用公式：select top m \* from table where id is not in (select top n \* from table)

分页：select \* from (select rownum r ,t1.\* from tablename t1 where rownum <EndRownum) where r > StartRownum;

12、某一表经常死锁，分析其原因以及解决方案

sql server死锁表现一:

一个用户A 访问表A(锁住了表A),然后又访问表B ；另一个用户B 访问表B(锁住了表B),然后企图访问表A 。这时用户A由于用户B已经锁住表B，它必须等待用户B释放表B,才能继续，好了他老人家就只好老老实实在这等了。同样用户B要等用户A释放表A才能继续这就死锁了。

sql server死锁解决方法：

这种死锁是由于你的程序的BUG产生的，除了调整你的程序的逻辑别无他法，仔细分析你程序的逻辑：①尽量避免同时锁定两个资源②必须同时锁定两个资源时，要保证在任何时刻都应该按照相同的顺序来锁定资源.

sql server死锁表现二：

用户A读一条纪录，然后修改该条纪录；这时用户B也修改该条纪录。这里用户A的事务里锁的性质由共享锁企图上升到独占锁(for update),而用户B里的独占锁由于A有共享锁存在所以必须等A释放掉共享锁，而A由于B的独占锁而无法上升的独占锁也就不可能释放共享锁，于是出现了死锁。这种死锁比较隐蔽，但其实在稍大点的项目中经常发生。

sql server死锁解决方法: 让用户A的事务（即先读后写类型的操作),在select 时就是用Update lock 语法如下： select \* from table1 with(updlock) where ....

13、如果网易通行证现在有1亿用户，怎么样解决登录缓慢问题.

用户登录的问题，也就是对服务器数据库的查询验证问题，所以也就转换为数据库优化问题：

（1）在数据库设计方面

①建立索引

②分区（比如按时间分区）

③尽量使用固定长度的字段

④限制字段长度

（2）在数据库IO方面

①增加缓冲区

②如果涉及表的级联，不同的表存储在不同的磁盘上，以增加IO速度。

（3）在SQL语句方面

①优化SQL语句，减少比较次数。

②限制返回的条目数（MySQL中用limit）

（4）在Java方面

①如果是反复使用的查询，使用PreparedStatement减少查询次数。

14、分页怎么实现的，你在项目中怎么用的，都有什么参数

select \* from (select rownum r ,t1.\* from tablename t1 where rownum <EndRownum) where r > StartRownum;