1. **windows 系统下安 Linux 系统有几种安装方法**

下面总结一下上面的三种方案：

1.虚拟机安装

优点：windows的基础上安装软件，不用担心死机或突发状况。

缺点：用户体验差，无法真正体会ubuntu的好处。

风险系数：★

2.wubi安装

优点：直接双击运行即可，快捷方便。保留了Windows原来的引导，符合大多数人的使用习惯

缺点：容易出现死机，待机容易有问题，联网可能会有问题。

风险系数：★★

3.U盘安装

优点：简单安全，真正意义上的双系统。爱捣弄的同学就来试试吧~

缺点：ubuntu引导不是大多数人想要的结果，每次开机都要选择windows系统才行，如果只是体验linux没有必要。

风险系数：★★★

**2、指针和引用的定义和性质区别：**

(1)指针：指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已。如：

int a=1;int \*p=&a;

int a=1;int &b=a;

上面定义了一个整形变量和一个指针变量p，该指针变量指向a的存储单元，即p的值是a存储单元的地址。

而下面2句定义了一个整形变量a和这个整形a的引用b，事实上a和b是同一个东西，在内存占有同一个存储单元。

(2)可以有const指针，但是没有const引用；

(3)指针可以有多级，但是引用只能是一级（int \*\*p；合法 而 int &&a是不合法的）

(4)指针的值可以为空，但是引用的值不能为NULL，并且引用在定义的时候必须初始化；

(5)指针的值在初始化后可以改变，即指向其它的存储单元，而引用在进行初始化后就不会再改变了。

(6)"sizeof引用"得到的是所指向的变量(对象)的大小，而"sizeof指针"得到的是指针本身的大小；

(7)指针和引用的自增(++)运算意义不一样；

**3、c++ 的虚拟类是什么**

当一个基类被声明为虚基类后，即使它成为了多继承链路上的公共基类，最后的派生类中也只有它的一个备份

**4、管态跟目态区别，以及如何切换**

大多数计算机系统将CPU执行状态分为管态和目态。管态又称为特权状态、系统态或核心态。通常，操作系统在管态下运行。目态又叫做常态或用户态，用户程序只能在目态下运行，如果用户程序在目态下执行特权指令，硬件将发生中断，由操作系统获得控制，特权指令执行被禁止，这样可以防止用户程序有意或无意的破坏系统。用户态切换到内核态的3种方式

a. 系统调用b. 异常 c. 外围设备的中断

1. **ftp 端口号以及主要使用方法**

ftp数据连接端口20，控制连接端口21。使用方法是个什么意思我不明白。答c/s工作方式和数据连接和控制连接的实现

**6、大型数据库表中什么时候使用索引**

[索引](http://baike.baidu.com/item/%E7%B4%A2%E5%BC%95" \t "http://baike.baidu.com/item/_blank)是对[数据库](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93" \t "http://baike.baidu.com/item/_blank)表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。最经常查询的列上使用索引以提高查询效率。

1. **常见的数据备份方法，以及如何导入，为什么要备份等等**

备份文件bak, 数据库还原，为了防止管理员的错误操作导致数据的不可恢复

**8、多线程如何实现？**

java中是继承Thread类或实现Runable接口

1. **.Java的垃圾回收机制？他是自动回收内存么？能不能想办法通知系统进行内存回收**

Java的垃圾回收机制是Java虚拟机提供的能力，用于在空闲时间以不定时的方式动态回收无任何引用的对象占据的内存空间。是自动回收内存

**10、IP地址的分类以及用途**

**11、数据结构中数据的逻辑结构和物理结构的区别**

**12、CSMA/CD用法和退避算法是什么原理和怎么实现？**

CSMA/CD（Carrier Sense Multiple Access/Collision Detect）  
即载波监听多路访问/[冲突检测](http://www.baidu.com/s?wd=%E5%86%B2%E7%AA%81%E6%A3%80%E6%B5%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnhFbnWf4PjfvPycznyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHmdPWf4nW0" \t "https://zhidao.baidu.com/question/_blank)方法  
一、基础篇：  
是一种争用型的[介质访问控制](http://www.baidu.com/s?wd=%E4%BB%8B%E8%B4%A8%E8%AE%BF%E9%97%AE%E6%8E%A7%E5%88%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnhFbnWf4PjfvPycznyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHmdPWf4nW0" \t "https://zhidao.baidu.com/question/_blank)协议。它起源于[美国夏威夷大学](http://www.baidu.com/s?wd=%E7%BE%8E%E5%9B%BD%E5%A4%8F%E5%A8%81%E5%A4%B7%E5%A4%A7%E5%AD%A6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnhFbnWf4PjfvPycznyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHmdPWf4nW0" \t "https://zhidao.baidu.com/question/_blank)开发的ALOHA网所采用的争用型协议，并进行了改进，使之具有比[ALOHA协议](http://www.baidu.com/s?wd=ALOHA%E5%8D%8F%E8%AE%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnhFbnWf4PjfvPycznyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHmdPWf4nW0" \t "https://zhidao.baidu.com/question/_blank)更高的介质利用率。   
CSMA/CD控制方式的优点是：  
原理比较简单，技术上易实现，网络中各工作站处于平等地位 ，不需集中控制，不提供优先级控制。但在网络负载增大时，发送时间增长，发送效率急剧下降。  
CSMA/CD应用在 ISO7层里的数据链路层  
它的工作原理是: 发送数据前 先监听信道是否空闲 ,若空闲 则立即发送数据.在发送数据时,边发送边继续监听.若监听到冲突,则立即停止发送数据.等待一段随即时间,再重新尝试.   
二、进阶篇：  
CSMA/CD控制规程：  
控制规程的核心问题：解决在公共通道上以广播方式传送数据中可能出现的问题（主要是数据碰撞问题）  
控制过程包含四个处理内容：侦听、发送、检测、冲突处理  
（1） 侦听：  
通过专门的检测机构，在站点准备发送前先侦听一下总线上是否有数据正在传送（线路是否忙）？  
若“忙”则进入后述的“退避”处理程序，进而进一步反复进行侦听工作。  
若“闲”，则一定算法原则（“X坚持”算法）决定如何发送。  
（2） 发送：  
当确定要发送后，通过发送机构，向总线发送数据。  
（3） 检测：  
数据发送后，也可能发生数据碰撞。因此，要对数据边发送，边接收，以判断是否冲突了。（参5P127图）  
（4）冲突处理：  
 当确认发生冲突后，进入冲突处理程序。有两种冲突情况：  
 ① 侦听中发现线路忙  
② 发送过程中发现数据碰撞  
① 若在侦听中发现线路忙，则等待一个延时后再次侦听，若仍然忙，则继续延迟等待，一直到可以发送为止。每次延时的时间不一致，由退避算法确定延时值。  
② 若发送过程中发现数据碰撞，先发送阻塞信息，强化冲突，再进行侦听工作，以待下次重新发送（方法同①）  
几个概念：  
上述两种冲突情况都会涉及一个共同算法——退避算法。  
① 退避算法：当出现线路冲突时，如果冲突的各站点都采用同样的退避间隔时间，则很容易产生二次、三次的碰撞。因此，要求各个站点的退避间隔时间具有差异性。这要求通过退避算法来实现。  
截断的二进制指数退避算法（退避算法之一）：  
 当一个站点发现线路忙时，要等待一个延时时间M，然后再进行侦听工作。延时时间M以以下算法决定：  
 M = 2 min{n,16} ms  
其中，n表示连续侦听的次数（记数值）。该表达式的含义是：第一次延迟2ms，再冲突则延迟22ms，以后每次连续的冲突次数记数都比前一次增加一倍的延迟时间，但最长的延迟时间不超过216ms。（即：超过16次做特殊处理）  
② 特殊阻塞信息：是一组特殊数据信息。在发送数据后发现冲突时，立即发送特殊阻塞信息（连续几个字节的全1），以强化冲突信号，使线路上站点可以尽早探测得到冲突的信号，从而减少造成新冲突的可能性。  
③ [冲突检测](http://www.baidu.com/s?wd=%E5%86%B2%E7%AA%81%E6%A3%80%E6%B5%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnhFbnWf4PjfvPycznyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHmdPWf4nW0" \t "https://zhidao.baidu.com/question/_blank)时间>=2α: α表示网络中最远两个站点的传输线路延迟时间。该式表示检测时间必须保证最远站点发出数据产生冲突后被对方感知的最短时间。在2α时间里没有感知冲突，则保证发出的数据没有产生冲突。（只要保证检测2α时间，没有必要整个发送过程都进行检测）  
④ X-坚持的CSMA算法：当在侦听中发现线路空闲时，不一定马上发送数据，而采用X-坚持的CSMA算法决定如何进行数据发送：  
三种算法及特点：  
- 非坚持的CSMA：线路忙，等待一段时间，再侦听；不忙时，立即发送；减少冲突，信道利用率降低：  
- 1坚持的CSMA：线路忙，继续侦听；不忙时，立即发送；提高信道利用率，增大冲突：  
- p坚持的CSMA：线路忙，继续侦听；不忙时，根据p概率进行发送，另外的1-p概率为继续侦听（p是一个指定概率值）；有效平衡，但复杂：  
（5）控制流程图（右上角图）：  
（6）CSMA控制规程的特征  
① 简单  
② 具有广播功能  
③ 平均带宽： f = F / n  
④ 绝对平等，无优先级  
⑤ 低负荷高效，高负荷低效  
⑥ 延时时间不可预测  
⑦ [传输速率](http://www.baidu.com/s?wd=%E4%BC%A0%E8%BE%93%E9%80%9F%E7%8E%87&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnhFbnWf4PjfvPycznyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHmdPWf4nW0" \t "https://zhidao.baidu.com/question/_blank)与传输距离为一定值

总线型网络拓扑，现在我们看这样一个过程：

      站点A准备给站点B传输数据，应用CSMA/CD机制，在传输之前站点A不断侦听线路上是否空闲，当线路上空闲之时，A给B发出frame，在此同时，其他站点也在不断侦听线路空闲状态，恰好此时站点B也认为线路正好空闲，于是B向C发出frame。frame B和frame C在线路上不期而遇，产生了冲突，冲突给A和B分别一个提示，提示线路上有冲突，这个时候，A和B都会采取一个强化碰撞的措施，向线路发送干扰信号，这个信号会让所有的站点都接收到，认为线路上产生了碰撞。紧接着，A与B都使用退避[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/zhaozy55555/article/details/_blank)，延迟一个随机时间接着重传数据，也就是说，双方获得重传的机会是平等的，没有规定水先谁后，但是具体到先后则是有这个退避算法决定的，那么退避算法到底是怎么进行的呢？为什么不是有的人所认为的先冲突先发送的机制而是平等的机制呢？下面我就给大家讲讲这个退避算法的原理。

      早期以太网使用的退避算法称为二进制指数类型退避算法，具体进行如下：

      1）确定基本退避时间（基数），一般定为2τ，也就是一个争用期时间（争用期在后面给出定义），对于以太网就是51.2μs。

      2）定义一个参数K，为重传次数，K＝min[重传次数，10]，可见K≤10。

      3）从离散型整数集合[0，1，2，……，(2^k－1)]中，随机取出一个数记做R。

       （备注：2^k是2的k次方幂，这里上标表示不出来，敬请原谅）

      那么重传所需要的退避时间为R倍的基本退避时间：即：T＝R×2τ。

      4）同时，重传也不是无休止的进行，当重传16次不成功，就丢弃该帧，传输失败，报告给高层协议。

      如：第一次重传K＝1，R＝0，1；T＝0，2t。T在二者中随机选择。

     那么第二次重传呢？K＝2，R＝0，1，2，3；T＝0，2t，4t，6t。

     如此可见，重传数次越多，则退避的时间就越长，称为动态退避。

     说到这里大家应该所有明白了吧，还有什么问题可以接着跟贴提出！

    附录：什么是争用期？

     在总线传输线路上，单程端到端的传输时延记做t。一个站点在发出数据到接收到线路冲突的时间小于等于2倍t，也就是时延为2t，这个时间被称为争用期。（contention period）

**13、如果给你一个数据库200GB,用户每天更新50MB,请你制定一个备份计划**

**14、请阐述一下线程的几种实现方式以及线程同步的几种方式**

什么是系统进程 进程是指在系统中正在运行的一个应用程序；线程是系统分配处理器时间资源的基本单元，或者说进程之内独立执行的一个单元。对于操作系统而言，其调度单元是线程。一个进程至少包括一个线程，通常将该线程称为主线程。一个进程从主线程的执行开始进而创建一个或多个附加线程，就是所谓基于多线程的多任务。 那进程与线程的区别到底是什么？进程是执行程序的实例。例如，当你运行记事本程序（Nodepad）时，你就创建了一个用来容纳组成 [Notepad.exe](http://www.so.com/s?q=Notepad.exe&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)的代码及其所需调用动态链接库的进程。每个进程均运行在其专用且受保护的地址空间内。因此，如果你同时运行记事本的两个拷贝，该程序正在使用的数据在各自实例中是彼此独立的。在记事本的一个拷贝中将无法看到该程序的第二个实例打开的数据。 以沙箱为例进行阐述。一个进程就好比一个沙箱。线程就如同沙箱中的孩子们。孩子们在沙箱子中跑来跑去，并且可能将沙子攘到别的孩子眼中，他们会互相踢打或撕咬。但是，这些沙箱略有不同之处就在于每个沙箱完全由墙壁和顶棚封闭起来，无论箱中的孩子如何狠命地攘沙，他们也不会影响到其它沙箱中的其他孩子。因此，每个进程就象一个被保护起来的沙箱。未经许可，无人可以进出。 实际上线程运行而进程不运行。两个进程彼此获得专用数据或内存的唯一途径就是通过协议来共享内存块。这是一种协作策略。下面让我们分析一下任务管理器里的进程选项卡。 这里的进程是指一系列进程，这些进程是由它们所运行的可执行程序实例来识别的，这就是进程选项卡中的第一列给出了映射名称的原因。请注意，这里并没有进程名称列。进程并不拥有独立于其所归属实例的映射名称。换言之，如果你运行5个记事本拷贝，你将会看到5个称为Notepad.exe的进程。它们是如何彼此区别的呢？其中一种方式是通过它们的进程ID，因为每个进程都拥有其独一无二的编码。该进程ID由[Windows NT](http://www.so.com/s?q=Windows NT&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)或[Windows 2000](http://www.so.com/s?q=Windows 2000&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)生成，并可以循环使用。因此，进程ID将不会越编越大，它们能够得到循环利用。第三列是被进程中的线程所占用的[CPU时间](http://www.so.com/s?q=CPU%E6%97%B6%E9%97%B4&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)百分比。它不是CPU的编号，而是被进程占用的CPU时间百分比。此时我的系统基本上是空闲的。尽管系统看上去每一秒左右都只使用一小部分CPU时间，但该[系统空闲进程](http://www.so.com/s?q=%E7%B3%BB%E7%BB%9F%E7%A9%BA%E9%97%B2%E8%BF%9B%E7%A8%8B&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)仍旧耗用了大约99%的CPU时间。 第四列，CPU时间，是CPU被进程中的线程累计占用的小时、分钟及秒数。请注意，我对进程中的线程使用占用一词。这并不一定意味着那就是进程已耗用的CPU时间总和，因为，如我们一会儿将看到的，NT计时的方式是，当特定的时钟间隔激发时，无论谁恰巧处于当前的线程中，它都将计算到CPU周期之内。通常情况下，在大多数NT系统中，时钟以10毫秒的间隔运行。每10毫秒NT的心脏就跳动一下。有一些[驱动程序](http://www.so.com/s?q=%E9%A9%B1%E5%8A%A8%E7%A8%8B%E5%BA%8F&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)代码片段运行并显示谁是当前的线程。让我们将CPU时间的最后10毫秒记在它的帐上。因此，如果一个线程开始运行，并在持续运行8毫秒后完成，接着，第二个线程开始运行并持续了2毫秒，这时，时钟激发，请猜一猜这整整10毫秒的时钟周期到底记在了哪个线程的帐上？答案是第二个线程。因此，NT中存在一些固有的不准确性，而NT恰是以这种方式进行计时，实际情况也如是，大多数[32位操作系统](http://www.so.com/s?q=32%E4%BD%8D%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&ie=utf-8&src=wenda_link" \t "http://wenda.so.com/q/_blank)中都存在一个基于间隔的计时机制。请记住这一点，因为，有时当你观察线程所耗用的CPU总和时，会出现尽管该线程或许看上去已运行过数十万次，但其CPU时间占用量却可能是零或非常短暂的现象，那么，上述解释便是原因所在。上述也就是我们在任务管理器的进程选项卡中所能看到的基本信息列。

1. **C++多态机制是什么，什么是虚函数，MFC的消息映射机制**

**面向对象程序设计中的多态性是指向不同的对象发送同一个消息，不同对象对应同一消息产生不同行为。在程序中消息就是调用函数，不同的行为就是指不同的实现方法，即执行不同的函数体。也可以这样说就是实现了“一个接口，多种方法”。**

**从实现的角度来讲，多态可以分为两类：编译时的多态性和运行时的多态性。前者是通过静态联编来实现的，比如C++中通过函数的重载和运算符的重载。后者则是通过动态联编来实现的，在C++中运行时的多态性主要是通过虚函数来实现的，也正是今天我们要讲的主要内容。**

**1.不过在说虚函数之前，我想先介绍一个有关于基类与派生类对象之间的复制兼容关系的内容。它也是之后学习虚函数的基础。我们有时候会把整型数据赋值给双精度类型的变量。在赋值之前，先把整形数据转换为双精度的，在把它赋值给双精度类型的变量。这种不同类型数据之间的自动转换和赋值，称为赋值兼容。同样的，在基类和派生类之间也存在着赋值兼容关系，它是指需要基类对象的任何地方都可以使用公有派生类对象来代替。为什么只有公有继承的才可以呢，因为在公有继承中派生类保留了基类中除了构造和析构之外的所有成员，基类的公有或保护成员的访问权限都按原样保留下来，在派生类外可以调用基类的公有函数来访问基类的私有成员。因此基类能实现的功能，派生类也可以。**

**那么它们具体是如何体现的呢？(1)派生类对象直接向基类赋值，赋值效果，基类数据成员和派生类中数据成员的值相同;(2)派生类对象可以初始化基类对象引用;(3)派生类对象的地址可以赋给基类对象的指针;(4)函数形参是基类对象或基类对象的引用，在调用函数时，可以用派生类的对象作为实参;**

**IMG_256IMG_257View Code**

**[IMG_258](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**1 #include "stdafx.h"  
 2 #include<iostream>  
 3 #include<string>  
 4   
 5 class ABCBase  
 6 {  
 7 private:  
 8 std::string ABC;   
 9 public:  
10 ABCBase(std::string abc)  
11 {  
12 ABC=abc;  
13 }  
14 void showABC();  
15 };  
16   
17 void ABCBase::showABC()  
18 {  
19 std::cout<<"字母ABC=>"<<ABC<<std::endl;  
20 }  
21   
22 class X:public ABCBase  
23 {  
24 public:  
25 X(std::string x):ABCBase(x){}  
26 };  
27   
28 void function(ABCBase &base)  
29 {  
30 base.showABC();  
31 }  
32   
33   
34 int main()  
35 {  
36 ABCBase base("A");  
37 base.showABC();  
38   
39 X x("B");  
40 base=x;  
41 base.showABC();  
42   
43 ABCBase &base1=x;  
44 base1.showABC();  
45   
46 ABCBase \*base2=&x;  
47 base2->showABC();  
48   
49 function(x);  
50   
51 return0;  
52 }**

**[IMG_259](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**结果：**

****

**要注意的是：第一，在基类和派生类对象的赋值时，该派生类必须是公有继承的。第二，只允许派生类对象向基类对象赋值，反过来不允许；**

**2.紧接着来讲一下虚函数，它允许函数调用与函数体之间的联系在运行时才建立，即在运行时才决定如何动作。虚函数声明的格式：**

**virtual 返回类型 函数名(形参表)**

**{**

**函数体**

**}**

**那么定义虚函数有什么用呢？让我们先来看看下面这个示例：**

**IMG_261IMG_262View Code**

**[IMG_263](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**1 #include "stdafx.h"  
 2 #include <iostream>  
 3 #include <string>  
 4   
 5   
 6 class Graph  
 7 {  
 8 protected:  
 9 double x;  
10 double y;  
11 public:  
12 Graph(double x,double y);  
13 void showArea();  
14 };  
15   
16 Graph::Graph(double x,double y)  
17 {  
18 this->x=x;  
19 this->y=y;  
20 }  
21   
22 void Graph::showArea()  
23 {  
24 std::cout<<"计算图形面积"<<std::endl;  
25 }  
26   
27 class Rectangle:public Graph  
28 {  
29 public:  
30 Rectangle(double x,double y):Graph(x,y){};  
31 void showArea();  
32 };  
33   
34 void Rectangle::showArea()  
35 {  
36 std::cout<<"矩形面积为："<<x\*y<<std::endl;  
37 }  
38   
39 class Triangle:public Graph  
40 {  
41 public:  
42 Triangle(double d,double h):Graph(d,h){};  
43 void showArea();  
44 };  
45   
46 void Triangle::showArea()  
47 {  
48 std::cout<<"三角形面积为："<<x\*y\*0.5<<std::endl;  
49 }  
50   
51 class Circle:public Graph  
52 {  
53 public:  
54 Circle(double r):Graph(r,r){};  
55 void showArea();  
56 };  
57   
58 void Circle::showArea()  
59 {  
60 std::cout<<"圆形面积为："<<3.14\*x\*y<<std::endl;  
61 }  
62   
63 int main()  
64 {  
65 Graph \*graph;  
66   
67 Rectangle rectangle(10,5);  
68 graph=&rectangle;  
69 graph->showArea();  
70   
71 Triangle triangle(5,2.4);  
72 graph=&triangle;  
73 graph->showArea();  
74   
75 Circle circle(2);  
76 graph=&circle;  
77 graph->showArea();  
78   
79 return0;  
80 }**

**[IMG_264](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**结果：**

****

**结果似乎和我们想象的不一样，既然Graph类(图形类)的对象graph指针分别指向了Rectangle类(矩形类)对象，Triangle类(三角类)对象，以及Circle类(圆类)对象，那么就应该执行它们自己所对应成员函数showArea(),怎么结果会是Graph类(图形类)的对象graph里的成员函数呢？这好像和我们在[C++之继承与派生(2)](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/09/2131942.html" \t "http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/_blank)一节里所讲到的派生类成员覆盖了基类中使用相同名称的成员(派生类对象调用同名成员函数是来自于自己类中成员函数，而非基类中上的)有所不同啊，其实当基类对象指针指向公有派生类的对象时，它只能访问从基类继承下来的成员，而不能访问派生类中定义的成员。但是使用动态指针就是为了表达一种动态调用的性质即当前指针指向哪个对象，就调用那个对象对应类的成员函数。那要怎么来解决的，这时虚函数就体现出了它的作用。其实我们只需要对上一个示例代码中所有的类里出现的showArea()函数声明之前加一个关键字virtual:**

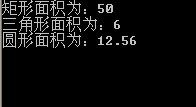
**IMG_266IMG_267View Code**

**[IMG_268](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**1 #include "stdafx.h"  
 2 #include <iostream>  
 3 #include <string>  
 4   
 5   
 6 class Graph  
 7 {  
 8 protected:  
 9 double x;  
10 double y;  
11 public:  
12 Graph(double x,double y);  
13 voidvirtual showArea();//定义为虚函数或virtual void showArea()  
14 };  
15   
16 Graph::Graph(double x,double y)  
17 {  
18 this->x=x;  
19 this->y=y;  
20 }  
21   
22 void Graph::showArea()  
23 {  
24 std::cout<<"计算图形面积"<<std::endl;  
25 }  
26   
27 class Rectangle:public Graph  
28 {  
29 public:  
30 Rectangle(double x,double y):Graph(x,y){};  
31 virtualvoid showArea();//定义为虚函数  
32 };  
33   
34 void Rectangle::showArea()  
35 {  
36 std::cout<<"矩形面积为："<<x\*y<<std::endl;  
37 }  
38   
39 class Triangle:public Graph  
40 {  
41 public:  
42 Triangle(double d,double h):Graph(d,h){};  
43 virtualvoid showArea();//定义为虚函数  
44 };  
45   
46 void Triangle::showArea()  
47 {  
48 std::cout<<"三角形面积为："<<x\*y\*0.5<<std::endl;  
49 }  
50   
51 class Circle:public Graph  
52 {  
53 public:  
54 Circle(double r):Graph(r,r){};  
55 virtualvoid showArea();//定义为虚函数  
56 };  
57   
58 void Circle::showArea()  
59 {  
60 std::cout<<"圆形面积为："<<3.14\*x\*y<<std::endl;  
61 }  
62   
63 int main()  
64 {  
65 Graph \*graph;  
66   
67 Rectangle rectangle(10,5);  
68 graph=&rectangle;  
69 graph->showArea();  
70   
71 Triangle triangle(5,2.4);  
72 graph=&triangle;  
73 graph->showArea();  
74   
75 Circle circle(2);  
76 graph=&circle;  
77 graph->showArea();  
78   
79 return0;  
80 }**

**[IMG_269](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**其它代码原封不动，这样运行出来的结果就是我们所需要的：**

****

**在基类中的某成员函数被声明为虚函数后，在之后的派生类中科以重新来定义它。但定义时，其函数原型，包括返回类型、函数名、参数个数、参数类型的顺序，都必须和基类中的原型完全相同。其实在上述修改后的示例代码里，只要在基类中显式声明了虚函数，那么在之后的派生类中就需要用virtual来显式声明了，可以略去，因为系统会根据其是否和基类中虚函数原型完全相同来判断是不是虚函数。因此，上述派生类中的虚函数如果不显式声明也还是虚函数。最后对虚函数做几点补充说明：(1)因为虚函数使用的基础是赋值兼容，而赋值兼容成立的条件是派生类之从基类公有派生而来。所以使用虚函数，派生类必须是基类公有派生的;(2)定义虚函数，不一定要在最高层的类中，而是看在需要动态多态性的几个层次中的最高层类中声明虚函数;(3)虽然在上述示例代码中main()主函数实现部分，我们也可以使用相应图形对象和点运算符的方式来访问虚函数，如：rectangcle.showArea(),但是这种调用在编译时进行静态联编，它没有充分利用虚函数的特性。只有通过基类对象来访问虚函数才能获得动态联编的特性;(4)一个虚函数无论配公有继承了多少次，它仍然是虚函数;(5)虚函数必须是所在类的成员函数，而不能是友元函数，也不能是静态成员函数。因为虚函数调用要靠特定的对象类决定该激活哪一个函数;(6)内联函数不能是虚函数，因为内联函数是不能在运行中动态确定其位置的即使虚函数在类内部定义，编译时将其看作非内联;(7)构造函数不能是虚函数，但析构函数可以是虚函数;**

**如果在main()主函数中用new建立一个派生类无名对象和定义一个基类对象指针，并将无名对象的地址赋给基类对象指针时，当我们用delete运算符来撤销无名对象时，系统只执行基类析构函数，而不执行派生类析构函数。比如：**

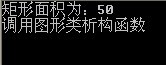
**IMG_271IMG_272View Code**

**[IMG_273](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**1 #include "stdafx.h"  
 2 #include <iostream>  
 3 #include <string>  
 4   
 5   
 6 class Graph  
 7 {  
 8 protected:  
 9 double x;  
10 double y;  
11 public:  
12 Graph(double x,double y);  
13 voidvirtual showArea();//定义为虚函数或virtual void showArea()  
14 ~Graph();  
15 };  
16   
17 Graph::Graph(double x,double y)  
18 {  
19 this->x=x;  
20 this->y=y;  
21 }  
22   
23 void Graph::showArea()  
24 {  
25 std::cout<<"计算图形面积"<<std::endl;  
26 }  
27   
28 Graph::~Graph()  
29 {  
30 std::cout<<"调用图形类析构函数"<<std::endl;  
31 }  
32   
33 class Rectangle:public Graph  
34 {  
35 public:  
36 Rectangle(double x,double y):Graph(x,y){};  
37 virtualvoid showArea();//定义为虚函数  
38 ~Rectangle();  
39 };  
40   
41 void Rectangle::showArea()  
42 {  
43 std::cout<<"矩形面积为："<<x\*y<<std::endl;  
44 }  
45   
46 Rectangle::~Rectangle()  
47 {  
48 std::cout<<"调用矩形类析构函数"<<std::endl;  
49 }  
50   
51 int main()  
52 {  
53 Graph \*graph;  
54 graph=new Rectangle(10,5);  
55 graph->showArea();  
56   
57 delete graph;  
58   
59 return0;  
60 }**

**[IMG_274](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**结果：**

****

**因为在撤销指针graph所指的派生类对象，在调用析构函数时，采用静态联编，只调用了Graph类的析构函数。如果也想调用派生类Rectangle类的析构函数的话，可将Graph类的析构函数定义为虚析构函数。其定义的一般格式：**

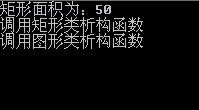
**virtual ~类名()**

**{**

**函数体**

**};**

**虽然派生类的析构函数与基类的析构函数名字不同，但是如果将基类的析构函数定义为虚函数，由该基类派生而来的所有派生类的析构函数都自动成为虚函数。我们把上一示例中的Graph类的析构函数前加上关键字virtual，那么执行结果：**

****

**显然这个结果才是我们所需要的。**

**3.上述示例中用了虚函数后，会发现其实Graph类(图形类)中的虚函数的函数体根本没有被用到过，就算被用到，该基类体现了图形的抽象的概念，并不与具体事物相联系。所以基类中的虚函数也没有实质性的功能。因此我们只需要在基类中留下一个函数名，而具体的实现留给派生类去定义。在C++中就是用纯虚函数来说明的。纯虚函数的一般形式：**

**virtual 返回类型 函数名(形参表)=0;**

**这里的"=0"并不是函数的返回值等于零，它只是起到形式上的作用，告诉编译系统"这是纯虚函数"。纯虚函数不具备函数功能，不能被调用。**

**[IMG_277](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**1 class Graph 2 { 3 protected: 4 double x; 5 double y; 6 public: 7 Graph(double x,double y); 8 voidvirtual showArea()=0;//定义纯虚函数 9 };10 11 Graph::Graph(double x,double y)12 {13 this->x=x;14 this->y=y;15 }**

**[IMG_278](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**4.如果一个类中至少有一个纯虚函数，那么就称该类为抽象类。所以上述中Graph类就是抽象类。对于抽象类有以下几个注意点：(1)抽象类只能作为其他类的基类来使用，不能建立抽象类对象;(2)不允许从具体类中派生出抽象类(不包含纯虚函数的普通类);(3)抽象类不能用作函数的参数类型、返回类型和显示转化类型;(4)如果派生类中没有定义纯虚函数的实现，而只是继承成了基类的纯虚函数。那么该派生类仍然为抽象类。一旦给出了对基类中虚函数的实现，那么派生类就不是抽象类了，而是可以建立对象的具体类;**

**5.最后还是一样，我将用一个实例来总结一下今天所讲的内容(开发工具：vs2010):**

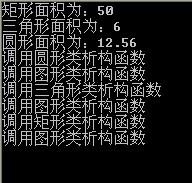
**IMG_279IMG_280View Code**

**[IMG_281](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**1 #include "stdafx.h"  
 2 #include <iostream>  
 3 #include <string>  
 4   
 5   
 6 class Graph //抽象类  
 7 {  
 8 protected:  
 9 double x;  
 10 double y;  
 11 public:  
 12 Graph(double x,double y);  
 13 //void virtual showArea();//定义为虚函数或virtual void showArea()  
 14 voidvirtual showArea()=0;//定义纯虚函数  
 15 virtual~Graph();//定义虚析构函数  
 16 };  
 17   
 18 Graph::Graph(double x,double y)  
 19 {  
 20 this->x=x;  
 21 this->y=y;  
 22 }  
 23   
 24 void Graph::showArea()  
 25 {  
 26 std::cout<<"计算图形面积"<<std::endl;  
 27 }  
 28   
 29 Graph::~Graph()  
 30 {  
 31 std::cout<<"调用图形类析构函数"<<std::endl;  
 32 }  
 33   
 34 class Rectangle:public Graph  
 35 {  
 36 public:  
 37 Rectangle(double x,double y):Graph(x,y){};  
 38 void showArea();//虚函数  
 39 ~Rectangle();//虚析构函数  
 40 };  
 41   
 42 void Rectangle::showArea()  
 43 {  
 44 std::cout<<"矩形面积为："<<x\*y<<std::endl;  
 45 }  
 46   
 47 Rectangle::~Rectangle()  
 48 {  
 49 std::cout<<"调用矩形类析构函数"<<std::endl;  
 50 }  
 51   
 52 class Triangle:public Graph  
 53 {  
 54 public:  
 55 Triangle(double d,double h):Graph(d,h){};  
 56 virtualvoid showArea();//虚函数  
 57 ~Triangle();//虚析构函数  
 58 };  
 59   
 60 void Triangle::showArea()  
 61 {  
 62 std::cout<<"三角形面积为："<<x\*y\*0.5<<std::endl;  
 63 }  
 64   
 65 Triangle::~Triangle()  
 66 {  
 67 std::cout<<"调用三角形类析构函数"<<std::endl;  
 68 }  
 69   
 70   
 71 class Circle:public Graph  
 72 {  
 73 public:  
 74 Circle(double r):Graph(r,r){};  
 75 virtualvoid showArea();//虚函数  
 76 ~Circle();//虚析构函数  
 77 };  
 78   
 79 void Circle::showArea()  
 80 {  
 81 std::cout<<"圆形面积为："<<3.14\*x\*y<<std::endl;  
 82 }  
 83   
 84 Circle::~Circle()  
 85 {  
 86 std::cout<<"调用圆形类析构函数"<<std::endl;  
 87 }  
 88   
 89 int main()  
 90 {  
 91 {  
 92 //Graph g(10,10);//抽象类不能建立对象  
 93   
 94 Graph \*graph;  
 95   
 96 Rectangle rectangle(10,5);  
 97 graph=&rectangle;  
 98 graph->showArea();  
 99   
100 Triangle triangle(5,2.4);  
101 graph=&triangle;  
102 graph->showArea();  
103   
104   
105 Graph \*graph1;  
106 graph1=new Circle(2);//new运算符建立无名对象  
107 graph1->showArea();  
108   
109 delete graph1;//delete运算符撤销派生类Circle无名对象  
110 }  
111   
112 return0;  
113 }**

**[IMG_282](http://www.cnblogs.com/CaiNiaoZJ/archive/2011/08/11/javascript:void(0);" \o "复制代码)**

**结果：**

****

**16、数据如何备份和转储**

**17、数据结构中数据的逻辑结构物理结构和处理方法之间的关系（感觉应该是线性、非线性啊什么链式、顺序啥的具体的你们再看看数据结构啦）**

1.逻辑结构

数据元素之间的逻辑关系称为数据的逻辑结构。数据的逻辑结构可以看作是从具体问题抽象出来的数学模型，它与数据的存储无关。从数学的角度观察，逻辑结构可形式化定义为（D，R），D是数据元素的集合，R是D上关系的有限数据元素的集合。如文章<http://blog.csdn.net/zjsjknd/article/details/7202658> 图1-1（b）线性结构，设4个数据元素之间分别为A、B、C和D。那么，描述的数据结构可以定义为：D={A,B,C,D},R={AB,BC,CD}.

2.物理结构

数据结构在计算机中的表示（又称映像）称为数据的物理结构，或称存储结构。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

如线性结构，既要存储数据元素A，B，C，D又要存储他们之间的关系AB，BC，CD那么，是用一片连续的内存单元来存放这些记录（如用数组表示），还是随机存放各结点数据再用指针进行链接呢？这就是物理结构的问题。根据分析该结构是线性关系，故采用数组来存储。

数据的存储结构可采用顺序存储或链式存储的方法。

顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构。顺序存储结构是一种最基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

链式存储方法是对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示，由此得到的存储表示称为链式存储结构。链式存储结构通常借助于程序设计语言中的指针类型来实现。

除了通常采用的顺序存储方法和链式存储方法外，有时为了查找的方便还采用索引存储方法和散列存储方法。

**18、const是怎么实现的，与宏定义有什么区别？**

(1) 编译器处理方式不同  
　　define宏是在预处理阶段展开。  
　　const常量是编译运行阶段使用。  
(2) 类型和安全检查不同  
　　define宏没有类型，不做任何类型检查，仅仅是展开，这样预处理在C++中成为可能引发错误的隐患。  
　　const常量有具体的类型，在编译阶段会执行类型检查。  
(3) 存储方式不同  
　　define宏仅仅是展开，有多少地方使用，就展开多少次，不会分配内存。  
　　const常量会在内存中分配(可以是堆中也可以是栈中)。  
(4)const 可以节省空间，避免不必要的内存分配

const定义常量从汇编的角度来看，只是给出了对应的内存地址，而不是象#define一样给出的是立即数，所以，const定义的常量在程序运行过程中只有一份拷贝，而 #define定义的常量在内存中有若干个拷贝。   
(5) 提高了效率。 编译器通常不为普通const常量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的常量，没有了存储与读内存的操作，使得它的效率也很高。

解决方案四：

using namespace std;  
#define pi 3.1415  
const float PI=3.14;  
int main()  
{ system("color 4A");  
float s1,s2,r;  
cin>>r;  
s1=PI\*r\*r;  
s2=pi\*r\*r;  
cout<<"s1="<<s1<<endl;  
cout<<"s2="<<s2<<endl;  
system("pause");  
}  
#define是C语言中进行常量定义的方法，const是C++中定义常量的方法

解决方案五：

#define表达式最后不需要加 ;

解决方案六：

define宏是在预处理阶段展开。  
const常量是编译运行阶段使用。

解决方案七：

#define 是预处理阶段执行的. 所谓预处理就是在编译前处理，这个过程没有什么神奇的，就是做一些文本工作。  
例如：#define a A。  
  
编译器注意到这个宏定义，当程序出现a的地方，就会用A代替a。只是简单地做一些文本工作。通常，我们会将一个标示符定义为某个数字，如：  
#define bufsize 1000  
在程序出现bufsize的地方会用1000来代替，一个有趣的地方是，1000是一个字面值常量，编译器会开辟内存用于存储1000这个值的。这也是为什么  
#define bufsize 1000.0和上面的写法是完全不同的，1000会默认作为int型，1000.0默认是浮点型（我忘了是双精度还是单精度）。  
要真正理解#define做了文本工作，可以考虑下列例子。  
#define MAX(a,b) a>b ? a:b  
程序出现,MAX(a,b) 的地方会变成 a>b ? a:b  
可以这样使用： printf("%dn", MAX(3, 2))//会输出3.  
事实上，#define的功能很强大，因为只是做了一些文本工作，所以是类型无关的，比如上面的MAX(a,b)也可以用来比较double类型的数据。

顺便一提，很多标准库中很多东西都是宏定义。

const是一个修饰符，用于表明一个变量不可修改，这是这么简单。一个有趣的问题是，编译器要怎么实现const的功能（也就是怎么让一个变量在  
程序的执行过程中只能读，不能写，这个问题自己想把）。

1. 指针和引用的区别：

指针实际上仍然是值传递，他保存的是某个对象或者变量的地址，指针的值是一个地址。使用指针时实际上先访问指针这个变量取出地址再去取指针指向的对象或者变量，而引用跟被引用的对象或者变量实际上是一样的，只是一个别名。在进行函数调用时，指针是值传递，而引用传递的是实参本身。

2. java垃圾回收机制：

在堆空间中每个对象实例都有一个引用计数器，引用计数器是一个互斥资源。当一个实例被创建时，引用计数器值为1，实例每次被引用时引用计算器+1。当实例的某个引用超过生命周期或者被使用该引用的变量重新赋值时，实例的引用计算器-1。当实例的引用计数器为0时，该实例占用的内存空间将被JAVA虚拟机收回。

3. 对象的创建方法：new、class或者constructor类的newInstance()方法

4. java对象生命期

1.创建阶段

2.应用阶段

3.不可见阶段（程序执行过了对象的作用域）

4.不可达阶段

5.收集阶段（垃圾收集器发现了对象为不可达状态）

6.终结阶段

7.对象控件重新分配阶段

5.多态

存在条件：继承、重写、父类引用指向子类对象

java中通过继承父类或者继承接口实现了多态

6.如何实现多线程：

从高级语言层面上来说：比如java，直接继续Runnable和Thread类就可以了，然后重写相应的方法，就可以实现了。

从OS层面来说：需要使用时间片的概念，CPU需要使用特定的调度算法（必须是多道程序调度算法）。然后使进程成为一个独占资源的单位，而线程成为一个独占CPU却不是独占资源的单位，多线程运行并不是并行，而是并发，当然，这是在单cpu的情况下。

7. 数据如何进行备份和转储

差量转储备份的是数据库变化的部分，而增量转储备份的是改变后的整个数据库。

8.管态和目态的区别

CPU有两种状态，分为管态和目态，在管态方式下CPU可以执行特权指令，而在目态方式下只能执行非特权指令

特权指令：一些跟IO设备有关的指令，访问PSW的指令，存取中断寄存器和时钟寄存器的指令。

9.数据的逻辑结构和物理结构

数据的逻辑结构包括：线性结构（包括栈、队列、串、数组、广义表）、非线性结构（包括树、图）

数据的物理结构包括：顺序储存、连接存储、索引存储、散列储存

10. 抽象类和接口

抽象类将一些实体抽象出来写的，比如将图形抽象出来（图形包括圆、三角形、长方形之类的），抽象类可以有自己的成员变量，可以有非抽象的方法，接口只有有抽象方法，接口只能拥有静态final的成员变量