一.  
&:与，其中一个为假就为假。  
|：或，两个为假才为假。  
^:异或，和|有点不一样，当true^true=false,true^false=true,false^true=true,false^false=false,  
总结:两边相同结果是假，两边不同结果是真。  
位运算符：  
<<:左移,3<<2=12,即3\*2的2次方=12, 3<<1=6,即3\*2的一次方=6.  
>>：右移：跟左移相反，右移后补什么由最高位的数值而定，如果最高位是0话，就补0如果是1的话  
就补1.  
>>>：无符号右移,右移后，都是用0补空位.  
二.  
&：与运算,比如6&3等于几呢？6用二进制表示是110,3用二进制表示是011,由于1&0或0与1都是假，即0，所以  
把他们二进制作&运算后的二进制值为010,即为2.  
|：或运算，比如6|5等于几呢？6用二进制表示是110，5用二进制表示是101,由于1|0或0|1都是真，所以  
把他们二进制作|运算后的二进制值为111,即为7。  
^：异或运算，比如6^5，6用二进制表示是110，5用二进制表示是101,异或由于是两边相同即为假，即1^1=0,  
0^0=0,而两边不相同即为真，即1^0=1,0^1=1,所以把他们的二进制作异或运算后的值为011.即为3.  
一个数异或同一个数两次，结果还是那个数。  
~：反码，其实就是取反，比如要求6的反码，先把6用二进制表示就是 000....000110,然后将其取反就是  
111....111001,将它减个一再取反，1用二进制表示就是111....111001,再将它们相减得到000....000111  
将它转成十进制数就是-7.

虽然之前在学校学过这方面的知识，但是现在对这方面的知识的重点理解的不是特别的清晰，所以今天  
把视频从头到尾看了一遍，把其之前不太理解的总结如下：  
一.  
数组一旦被创建，他就有默认的值，  
int类型的数组的默认值为0，double的为0.0,float的为0.0f.boolean默认为false.  
二.  
练习:获取数组中的最大值.  
思路:  
1.获取最值需要进行比较，每一次比较都会有一个较大的值，因为该值不确定。通过 一个变量进行临储。  
2.让数组中的每一个元素都和这个变量中的值进行比较，如果大于变量的值，就用该变量记录较大值。

[第三篇 黑马程序员-条件判断语句](http://blog.csdn.net/ljj810875/article/details/7888843)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 08:51 38人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888843#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888843#report)

[android](http://blog.csdn.net/tag/details.html?tag=android)[n2](http://blog.csdn.net/tag/details.html?tag=n2)[java](http://blog.csdn.net/tag/details.html?tag=java)[string](http://blog.csdn.net/tag/details.html?tag=string)[byte](http://blog.csdn.net/tag/details.html?tag=byte)[class](http://blog.csdn.net/tag/details.html?tag=class)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一.  
顺序结构,如下代码所示：  
package circulate;

public class IfDemo {  
 public static void main(String[] args)  
 {  
  System.out.println("Hello World");  
  System.out.println("Hello World");  
  System.out.println("Hello World");  
  System.out.println("Hello World");  
 }  
}  
二.  
注意：  
如果if控制的语句只有一条，那么"｛ "可以不写.  
if...else语句和三元运算符的区别(代码演示):  
三元运算符：  
int a = 9,b;  
b=(a>1)?100:200;  
用if...else表达就是:  
if(a>1)  
  b=100;  
else  
  b=200;  
小结：这里写成三元运算符更加简便。  
像下面的代码是不能写成三元运算符的：  
int x = 1;  
  if (x > 1)   
  {

   System.out.println("yes");  
  }  
  else   
  {  
     
   System.out.println("a");

  }  
如果写成三元运算符就是：b=(x>1)?System.out.println("yes"):System.out.println("a");  
这样子那肯定不行，因为System.out.println("yes")是直接打印到屏幕上的。  
三.  
if...else结构的简写格式:变量=(条件表达式)?表达式1:表达式2；  
if...else的代码演示:  
    int n=3;  
    
  if(n>1)  
   System.out.println("a");  
  else if(n>2)  
   System.out.println("b");  
  else if(n>2)  
   System.out.println("c");  
  else  
   System.out.println("d");  
    
 }  
以上代码打印结果为a,因为当n=3时执行第一个语句时满足了，他就不会往后执行了，所以打印结果为a.  
三元运算符的特点:可以简化if...else代码，弊端就是因为是一个运算符，所以  
运算完必须要有一个结果。  
以下代码中 if(n>1)  
               System.out.println("a");和  
                                           if(n>2)  
                                              System.out.println("b");是独立的。  
  if(n>1)  
   System.out.println("a");  
  if(n>2)  
   System.out.println("b");  
  if(n>3)  
   System.out.println("c");  
  else  
   System.out.println("d");  
练习：根据用户指定的月份，打印该月份所属的季节  
分析：  
//3,4,5春   6,7,8夏   9,10,11秋  12,1,2东  
  int x = 4;  
    
  if(x==3||x==4||x==5)  
     System.out.println("春季");  
  else if(x==6||x==7||x==8)  
   System.out.println("夏季");  
  else if(x==9||x==10||x==11)  
   System.out.println("秋季");  
  else if(x==12||x==1||x==2)  
   System.out.println("冬季");  
  else  
   System.out.println("月份不存在");  
     
 }  
如果if(x==3||x==4||x==5)中的(x==3||x==4||x==5)可以表示为(x>=3&&x<=5),  
那么else if(x==12||x==1||x==2)中的(x==12||x==1||x==2)怎样这样表示呢？  
方法是把1-12以外的数全部判断出来，然后我再判断1-12里面的数：  
  if(x>12||x<1)  
   System.out.println("月份不存在");  
     else if(x>=3&&x<=5)  
       System.out.println("春季");  
     else if(x>=6&&x<=8)  
       System.out.println("夏季");  
     else if(x>=9&&x<=11)  
       System.out.println("秋季");  
     else   
       System.out.println("冬季");   
    
 }  
四.  
switch语句  
格式:  
    switch(表达式)  
    {  
      case 取值1:  
       执行语句;  
      break;  
        
       case 取值2:  
       执行语句;  
        break;  
       .....  
        default:  
         执行语句;  
         break;  
      }  
当case不对的话，他就往下执行，如果判断所有的都不对的话，那就是默认即执行default:  
                                   执行语句;  
                                   break;  
特点：  
1.switch语句选择的类型只有四种:byte,short,int,char.  
2.case之间与default没有顺序,先执行第一个case,没有匹配的case执行default.  
3.结束switch语句的两种情况：遇到break，执行到switch语句结束  
4.如果匹配的case或者default没有对应的break，那么程序会继续向下执行，运行可以执行的  
语句，直到遇到break或者switch的结尾结束。  
小结：  
if和switch语句很像，具体什么场景下应用哪个语句呢？如果判断的具体数值不多，而是符合byte  
short int ,char这四种类型，虽然两个语句都可以使用，建议使用switch语句，因为  
效率稍高，其他情况，对区间判断，对结果为boolean类型判断，使用if,if的使用范围更广

[第四篇 黑马程序员-循环语句](http://blog.csdn.net/ljj810875/article/details/7888858)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 08:54 31人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888858#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888858#report)

[string](http://blog.csdn.net/tag/details.html?tag=string)[class](http://blog.csdn.net/tag/details.html?tag=class)[android](http://blog.csdn.net/tag/details.html?tag=android)[java](http://blog.csdn.net/tag/details.html?tag=java)[图形](http://blog.csdn.net/tag/details.html?tag=%e5%9b%be%e5%bd%a2)[存储](http://blog.csdn.net/tag/details.html?tag=%e5%ad%98%e5%82%a8)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

一.  
while语句  
格式：  
while(条件表达式)  
{  
  执行语句;  
}  
while的特点：先判断条件，只有条件满足才执行循环体  
do...while:先执行循环体，在判断条件，条件满足，在继续执行循环体  
小结：  
do...while:无论条件是否满足，循环体至少执行一次。  
二.  
for循环语句  
格式：  
for(初始化表达式:循环条件表达式：循环后的表达式)  
｛  
   执行语句;  
 }  
for循环语句执行的过程如下面代码所示，先初始化x=0,再判断x是否小于3，这里0<3,下面  
就执行System.out.println("x="+x);然后再执行x++,即为1，然后再来判断1是否小于3，满足了，  
所以就打印为1，然后执行x++,即为2，在判断2是否小于3，结果满足，所以打印2，再执行  
x++,即为3，判断3是否小于3，结果不满足，即跳出循环。  
public class ForDemo   
{  
 public static void main(String[] args)  
 {  
 for(int x=0;x<3;x++)  
 {  
   System.out.println("x="+x);   
   }

   }  
}  
注意：变量在定义的时候除了初始化值才能被使用以外，它还有个作用域的问题，即作用  
范围，在以上for循环语句中，x是读到for语句以后创建出x,那么这个x就只在  
{  
   System.out.println("x="+x);  
}  
这里面有效，这就叫变量的作用域。而while循环中，变量是定义在循环语句外面，故可以。  
小结：

如果变量只为循环的增量而存在，控制循环次数的话，用for循环，内存会相对于while  
循环语句用的少一点，  
看以下代码可知 System.out.println("x====="+x);这条语句打印会报找不到x  
这个变量的错误。  
public class ForDemo   
{  
 public static void main(String[] args)  
 {  
 for(int x=0;x<3;x++)  
 {  
   System.out.println("x="+x);   
 }  
 System.out.println("x====="+x);  
for循环里面的初始化表达式和循环后的表达式都可以不写，条件表达式如果也不写的话  
那默认为true.  
无限循环的最简单表现形式:  
for语句是：  
for(;;){}  
while语句是：  
while(true){}  
练习获取1-10的和并打印  
分析：  
该和具有规律：后面的第一个加数总是前面两个数的和，且前面的第二个加数逐1递增都等于后面的第二个加数：  
  0+1  
  1+2  
    3+3  
      6+4  
根据此规律编写代码如下：  
public class ForTest2   
{   
 public static void main(String[] args)  
 {  
  //1.定义变量用于存储不断变化的和  
  int sum=0;  
    
        //2.定义变量。记录住不断变化的被加的数   
  int x=1;  
  //定义循环，重复加法的过程  
  while(x<=10)  
  {  
   sum=sum+x;  
   x++;  
   
  }  
  System.out.println("sum="+sum);  
    
 }  
}  
循环注意：哪些语句要参与运算，哪些语句不需要。  
练习：1-100之间7的倍数的个数，并打印  
分析：  
1.先对1-100进行循环（遍历）通过循环的方式  
2.在循环的过程中，定义条件，只对7的倍数进行操作  
步骤：  
1.定义循环语句，选择for语句  
2.在循环中定义判断，只要是7的倍数即可，使用if语句。  
条件：7的倍数，x%7==0  
三.  
嵌套循环，如下代码所示说一下以下嵌套循环的执行过程，for循环一执行，在内存中产生了一个  
x变量块，初始化值为0，满足x<3以后，它就下来执行循环体，循环体还是一个循环for(int y=0;y<4;y++)  
初始化一个变量y=0满足y<4,所以打印一次ok,然后读y++,所以必须循环体执行完毕以后再执行for(int   
x=0;x<3;x++)  
所以先打印了4次ok以后 for(int y=0;y<3;y++)结束，即循环体结束，然后读 for(int x=0;x<3;x++)  
先x++,它的值为1，而1<3满足条件，故又执行循环体for(int y=0;y<4;y++)，故打印12次ok.  
public class ForForDemo {  
 public static void main(String[] args)  
 {  
  for(int x=0;x<3;x++)  
  {  
   for(int y=0;y<4;y++)  
   {  
    System.out.println("OK");  
   }  
   System.out.println();  
  }  
 }

}  
以上代码打印结果如下：                                        
okokokok  
okokokok  
okokokok  
小结：  
以上结果像个长方形，外循环控制的行数，内循环控制的每一行的列数，也就是一行中元素  
的个数。  
发现图形有很多行，每一行有很多列，故要使用嵌套循环  
原理：形象说法，大圈套小圈。由于这里的列数在变化，就用  
一个变量来记录这个列数的变化，如下代码表示：  
public class ForForDemo {  
 public static void main(String[] args)  
 {  
    int z=5;  
    
  for(int x=0;x<5;x++)  
  {  
   for(int y=0;y<z;y++)  
   {  
    System.out.print("\*");  
      
   }  
   System.out.println();  
   z--;  
  }  
 }

}  
尖朝上，可以改变条件，让条件随着外循环变化，尖朝下，可以初始化值  
让初始化随着外循环变化。  
break语句：跳出它所在的循环，在嵌套循环里，如果break要跳出某个循环的话，那就要对for 循环分别  
起名，就要用到标号的概念，比如在FOR循环前面加w:等只要是合法的标识符都行以示区别。

[第五篇 黑马程序员-函数、类与对象](http://blog.csdn.net/ljj810875/article/details/7888863)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 08:56 44人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888863#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888863#report)

[string](http://blog.csdn.net/tag/details.html?tag=string)[java](http://blog.csdn.net/tag/details.html?tag=java)[代码分析](http://blog.csdn.net/tag/details.html?tag=%e4%bb%a3%e7%a0%81%e5%88%86%e6%9e%90)[生活](http://blog.csdn.net/tag/details.html?tag=%e7%94%9f%e6%b4%bb)[android](http://blog.csdn.net/tag/details.html?tag=android)[class](http://blog.csdn.net/tag/details.html?tag=class)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

一.  
如何定义一个函数呢?  
1.既然函数是一个独立的功能，那么该功能的运算结果是什么先明确  
因为这是在明确函数的返回值类型  
2.再明确在定义该功能的过程中是否需要未知的内容参与运算  
因为是在明确函数的参数列表（参数的类型和参数的个数）  
小结：  
方法返回值为void的时候不要写输出语句，直接调用就可以了。  
方法里面只定义本方法要操作的功能，不要在定义其他的功能。  
什么时候用重载呢？  
当定义的功能相同，但是参与运算的未知内容不同  
那么，这时候就定义一个函数名称以表示其功能，方便阅读，而通过参数列表的不同来区分多个同名函数  
重载里面的参数是有顺序的。重载和返回值类型没关系，重载中只要方法名相同，然后参数类型和参数  
个数只要有一方不同它就重载了。  
面向对象  
将复杂的事物变得简单化  
类与对象  
类就是对现实生活中事物的描述，对象就是实实在在存在的个体。  
匿名对象  
匿名对象两种使用情况  
当对对象方法仅进行一次调用的时候  
匿名对象可以作为实际参数进行传递  
只所以对外提供访问方式，就因为可以在访问方式中加入逻辑判断等语句  
对访问的数据进行操作，提高代码的健壮性  
封装原则：  
将不需要对外提供的内容都隐藏，提供公共方法对其访问，隐藏的最低权限是  
private.  
对外提供的方法用get和set  
set方法一般都是void,因为设置了不需要返回且set一般都带参数  
而get没有参数。且其返回类型跟它获取的变量一致，你如果看到某个类里  
有setxxx或getxxx,那么这个类里一定有私有的属性xxx,  
抽象类和方法的特点：  
1.抽象方法一定定义在抽象类中  
2.抽象方法和抽象类都必须被abstract修饰  
3.抽象类不可以用new创建对象，因为调用抽象方法没意义  
4.抽象类中的方法要被使用，必须由子类复写所有的抽象方法后，建立子类对象调用。  
小结：  
如果子类只覆盖了部分抽象方法。那么该子类还是一个抽象类。  
抽象类和一般类没太大不同，该怎么描述事物就如何描述，只不过该事务中出现了不确定的部分，  
这些部分也是该事物的功能，需要明确出现，但是无法定义主体。通过抽象方法来表示。  
抽象类比一般类多了个抽象函数，就是在类中可以定义抽象方法，抽象类中也可以不定义抽象方法，它的  
作用是不要该类建立对象。  
内部类：  
内部类的特点：  
内部类可以直接访问外部类中的成员，包括私有成员  
原因：之所以可以直接访问外部类中的成员，是因为内部类中持有了一个外部类的引用，  
格式：外部类名.this  
外部类要访问内部类中的成员必须要建立内部类的对象。  
当内部类定义在外部类的的成员位置上时，而且非私有，可以在外部其他类中  
可以直接建立内部类对象：  
格式：  
外部类名.内部类名  变量名=外部类对象.内部类对象；  
构造函数的特点：  
1.函数名与类名相同  
2.不用定义返回值类型  
3.不可以写return语句  
作用：  
给对象进行初始化  
注意：  
1.默认构造函数的特点  
2.多个构造函数是以重载的形式存在的  
对象一建立就会调用与之对应的构造函数  
构造函数的作用可以用于给对象进行初始化  
构造函数的小细节：  
当一个类中没有定义构造函数时，那么系统会默认给该类加入一个空参数的  
当在类中自定义了一个构造函数后，默认的构造函数就没有了。  
一个对象建立，构造函数只运行一次，而一般方法可以被该对象调用多次。  
构造代码块需要注意的地方：  
作用：给对象进行初始化  
对象已建立就运行，而且优先于构造函数执行  
构造代码块是给所有对象进行统一初始化  
构造代码块中定义的是不同对象共性的初始化内容  
二.  
类与对象的关系  
类就是对现实生活中事物的描述，对象就是这类事物实实在在存在的个体  
举例：  
现实生活中的对象：张三，李四  
想要描述：提取对象中共性的内容,对具体对象的抽象  
描述时，这些对象的共性有：姓名，年龄，性别，学习Java的功能  
这个描述在Java中是用类表示，而这个对象就是在Java中用new操作符所产生的一个实体，这个实体在堆内  
存当中，把这些东西映射到内存里的话一个是类，一个就是实体。  
描述事物无非就是两部分，一部分就是事物的属性，另一部分就是事物的行为  
属性对应类中的变量，行为对应类中的方法  
其实定义类就是在描述事物，就是再定义属性和行为，属性和行为称为类中的成员(成员变量和成员函数)

举例：  
人开门(名词提炼法):  
人  
｛  
    开门（门）  
    ｛  
         门.开（）;  
    ｝  
｝

门  
｛  
     开（）｛操作门轴等。｝

｝  
三.  
面向对像的三个特征：封装，继承，多态  
以后的开发其实就是找对象使用，没有对象就创建个对象  
简单来说就是找对象，建立对象，使用对象，维护对象的关系  
成员变量和局部变量：  
成员变量作用于整个类中，局部变量作用于函数中，或者语句中  
在内存中的位置：  
成：在堆内存。因为对象的存在，才在内存中存在  
局：存在栈内存中.  
数据类型分为两种：1.基本数据类型  2.引用数据类型（数组引用，类引用，接口引用）  
堆内存中的变量都有一个特点，那就是默认初始化值。那就是null.  
匿名对象使用方式一：当对对象的方法只调用一次时，可以用它来写，这样比较简化，如果  
对一个对象进行多个成员调用，必须给这个对象起个名字。  
静态代码块：  
格式：  
static{  
   静态代码快中的执行语句  
   ｛  
 特点：  
1.随着类的加载而执行 只执行一次，并优先于主函数  
2.用于给类进行初始化的  
注意：静态代码块是给类初始化的，构造代码块是给对象初始化的。  
private:私有权限修饰符，用于修饰类中的成员（成员变量，成员函数）。  
注意：私有只在本类中有效。  
四.  
分析以下代码：  
class Person  
{  
   private String name="hah";  
   private int age;  
   private static String country="CN";  
   Person(String name,int age)  
   {  
            this.name=name;  
            this.age=age;  
    }  
    {  
             System.out.println(name+".."+age);         
    }  
    public void setName(String name)  
    {  
             this.name=name;  
    }  
      
    public void speak()  
    {  
             System.out.println(this.name+":::"+this.age);  
    }  
      
    public static void showCountry()  
    {  
             System.out.println("country="+country);  
    }  
}

class PersonDemo  
{  
    public static void main(String[] args)  
    {  
                Person p=new Person("zhangsan",20);  
    }  
}  
1.通过分析以上代码总结几点需要注意的地方：  
2.将age私有化以后，类以外即使建立了对象也不能直接访问了  
3.人应该有年龄，就需要提供对应的访问方式  
4.私有仅仅是封装的一种表现形式  
5.之所以对外提供访问方式，就因为可以在访问方式中加入逻辑判断等语句  
6.当分析事物时，该事物具备一些特殊行为，那么将这些内容定义在构造函数中  
7.静态是一个修饰符，用于修饰成员（成员变量和成员函数）  
8.静态修饰内容被对象所共享  
9.静态方法只能访问静态成员(方法和变量）  
10.非静态方法既可以访问静态也可以访问非静态  
11.静态方法中不可以定义this ,super关键字 ，因为静态优先于对象存在，所以静态方法中不可以出现  
12.静态有利有弊  
利：对对象的共享数据进行单独空间的存储  节省空间，没有必要每个对象中都存储一份  
可以直接被类名调用。  
弊：生命周期过长  访问出现局限性（静态虽好，只能访问静态）  
对象的建立过程：  
分析以上代码中的这段"Person p=new Person("zhangsan".20);"代码分析对象的建立过程：  
1.因为new用到了Person.class文件，所以先找到Person.class文件并加载到内存中  
2.执行该类中的static代码块，如果有的话，给Person.class类进行初始化。  
3.在堆内存中开辟空间，分配内存地址  
4.在堆内存中建立对象特有属性 并进行默认初始化  
5.对属性进行显示初始化  
6.对对象进行构造代码快初始化  
7.对对象进行对应的构造函数初始化  
8.将内存地址赋给栈内存中的P变量。

[第六篇 黑马程序员-继承](http://blog.csdn.net/ljj810875/article/details/7888870)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 08:57 40人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888870#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888870#report)

[java](http://blog.csdn.net/tag/details.html?tag=java)[android](http://blog.csdn.net/tag/details.html?tag=android)[设计模式](http://blog.csdn.net/tag/details.html?tag=%e8%ae%be%e8%ae%a1%e6%a8%a1%e5%bc%8f)[扩展](http://blog.csdn.net/tag/details.html?tag=%e6%89%a9%e5%b1%95)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

一.  
继承需要注意的地方：  
1.继承提高了代码的复用性  
2.让类与类之间产生关系，有了这个关系，才有了多态的特性  
3.千万不要为了获取其他类的功能，简化代码继承  
4.必须是类与类之间有所属关系才可以继承，  
5.java中只支持单继承，不支持多继承，因为多继承容易带来安全隐患。当多个父类中定义了相同功能，当功能内容不同时，不确定  
要运行哪个， 但是Java保留了这个机制，并用另一种体现形式来完成表示，即多实现   
6.Java支持多层继承，也就是一个继承体系  
7.想要使用体系，先查阅体系父类的描述，因为父类中定义的是该体系中共性功能  
通过了解共性功能 就可以知道该体系的基本功能，那么这个体系基本就可以使用了  
8.在具体调用时，要创建子类的对象，为什么？  
一是因为有可能父类不能创建对象  
二是因为创建子类对象可以使用更多的功能，包括基本的也包括特有的  
简单一句话：查阅父类功能，创建子类对像使用功能  
9.如果子类中出现非私有的同名成员变量时，子类要访问本类中的变量  
用this，子类要访问父类中的同名变量，用super.  
this代表的是本类对象的引用  
super代表的是父类对象中的引用  
子父类中函数的特点：  
当子类出现和父类一模一样的函数时，当子类对象调用该函数，会  
运行子类函数的内容，如同父类的函数被覆盖一样,这种情况是函数的另一个特性：重写（覆盖）  
当子类继承父类，沿袭了父类的功能到子类中，但是子类虽具备该功能，  
但是功能的内容却和父类不一致，这时，没有必要定义新功能，而是使用覆盖特性，  
保留父类功能定义，并重写功能内容  
覆盖注意：  
子类覆盖父类，必须保证子类权限大于等于父类权限，才可以覆盖，否则编译失败  
静态只能覆盖静态.  
重载和覆盖的区别：  
重载只看同名函数的参数列表，  
覆盖：子父类方法要一模一样  
子父类中的构造函数:  
在对子类对象进行初始化时，父类的构造函数也会运行，那是因为子类的构造函数默认第一行有一条隐式  
的语句super();它会访问父类中空参数的构造函数，而且子类中所有的构造函数默认第一行都是super();  
为什么子类一定要访问父类中的构造函数？  
因为父类中的数据子类可以直接获取，所以子类对象在建立时，需要先查看父类是如何对这些数据  
进行初始化的。所以子类在对象初始化时，要先访问一下父类的构造函数。如果要访问父类中指定的构造函数，可以通过手动定义的super方式来指定。  
注意：  
super语句一定定义在子类构造函数的第一行，但也有特殊情况，如果我在子类的第一行写上  
this();语句，那第一行语句就没有super了，但是子类里至少一条语句第一行是super来供其访问。  
结论：子类的所有的构造函数默认都会访问父类中空参数的构造函数，因为子类每一个构造函数  
内的第一行都有一句隐式的super();当父类中没有空参数的构造函数时，子类必须手动通过super  
语句形式来指定要访问的父类中的构造函数，当然，子类的构造函数第一行也可以手动指定this语句  
来访问本类中的构造函数，子类中至少会有一个构造函数会访问父类中的构造函数。这叫做子类的实例化  
过程。  
二.  
final关键字：  
作为一个修饰符，它有以下特点：  
1.可以修饰类，函数，变量。  
2.被final修饰的类不可以被继承。  
3.被final修饰的方法不能被复写。  
4.被final修饰的变量是一个常量只能赋值一次，  
注意：  
1.当在描述事物时，一些事物的出现值是固定的，这时为了增强阅读性，都给这些值起个名字  
方便于阅读，而这个值不需要改变，所以加上final修饰。  
2.作为常量， 常量的书写规范所有字母  
都大写，如果由多个单词组成，单词间通过\_连接.  
3.内部类定义在类中的局部变量时，只能访问该局部变量被final修饰的局部变量。  
4.当多个类中出现了相同功能  但是功能主体不同，这时可以进行向上抽取，这时 只抽取功能定义而不抽取功能主体。  
模版方法设计模式：  
在定义功能时，功能的一部分是确定的，有一部分是不确定的，确定的部分在使用不确定的部分  
这时就将不确定的部分暴露出去，让该类的子类去完成。  
三.  
接口：初期理解，可以认为是一个特殊的抽象类  
当抽象类中的方法都是抽象的，那么该类可以通过接口的形式来表示。  
常见定义:  
常量：public static final  
方法:public abstract  
记住：接口是不可以创建对象的，因为有抽象方法  
需要被子类实现，子类对接口中的抽象方法全部覆盖后，子类才可以实例化。  
否则子类是一个抽象类。  
接口可以被类多实现,也是对多继承不支持的转换形式，Java支持多实现  
Java为什么不支持多继承而支持多实现呢？  
因为多实现里面的方法没有方法主体，可以由子类任意定义.  
注意：只有接口与接口之间支持多继承，其他的情况除外。  
总的来说，接口具有以下特点：  
1.接口是对外暴露的规则  
2.接口是程序的功能扩展  
3.接口可以用来多实现  
4.类与接口之间是实现关系，而且类可以继承一个类的同时实现一个类  
5.接口与接口之间可以有继承关系

[第七篇 黑马程序员-多态与String类以及基本数据类型的包装类](http://blog.csdn.net/ljj810875/article/details/7888874)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 08:59 47人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888874#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888874#report)

[string](http://blog.csdn.net/tag/details.html?tag=string)[integer](http://blog.csdn.net/tag/details.html?tag=integer)[equals](http://blog.csdn.net/tag/details.html?tag=equals)[byte](http://blog.csdn.net/tag/details.html?tag=byte)[object](http://blog.csdn.net/tag/details.html?tag=object)[android](http://blog.csdn.net/tag/details.html?tag=android)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

一.  
多态的概念：  
可以理解为事物存在的多种体现形态。  
打个比方：  
人：男人，女人

动物：猫，狗。

猫 x=new 猫();

动物 x=new 猫();  
多态的一种形式：  
父类指向子类对象，当我想使用子类特有内容的时候，就向下转型，如下代码所示:  
package circulate;

public class Demo4   
{  
 private int num;  
 Demo4(int num)  
 {  
  this.num=num;  
 }  
 public boolean equals(Object obj)  
 {  
  Demo4 d=(Demo4)obj;//父类指向子类对象，当我想使用子类特有内容的时候，就向下转

型  
    
  return this.num==d.num;  
    
    
 }

}

public class ObjectDemo   
{  
 public static void main(String[] args)  
 {  
  Demo4 d1=new Demo4(4);  
  Demo4 d2=new Demo4(4);  
    
    
  System.out.println(d1.equals(d2));  
 }  
}  
1.多态的体现  
父类的引用指向了自己的子类对象  
父类的引用也可以接受自己的子类对象  
2.多态的前提  
必须是类与类之间有关系，要不是继承或实现  
通常还有一个前提 那就是覆盖  
3.多态的好处  
多态的出现大大提高了程序的扩展性  
4.多态的弊端  
提高了扩展性，但是只能使用父类中的引用访问父类中的成员  
5.多态的出现代码中的特点（多态使用的注意事项）  
在编译时期，参阅引用型变量所属的类中是否有调用的方法，如果有，编译通过。反之，失败  
在运行时期参阅对象所属的类中是否y9ou调用的方法。  
简单总结：成员函数在多态调用时，编译看左边，运行看右边。  
在多态中，无论编译和运行，都参考左边（引用型变量所属的类）  
在多态中，静态成员函数的特点：  
无论编译和运行，都参考左边。  
字符串String：  
字符串最大特点，一旦被初始化，就不可以改变。  
字符串是否是以指定内容开头  
boolean startsWith(str)  
字符串是否以指定内容结尾  
boolean endsWith(str);  
字符串中是否有内容  
boolean isEmpty();此方法的原理就是判断长度是否为0，0和""以及null有区别，""表示一个对象，null  
是指向空  
字符串中是否包含一个子串  
boolean contains(str);  
判断字符串内容是否相同  
boolean equals(str);  
判断内容是否相同，并忽略大小写  
boolean equalsIgnoreCase();  
转换:  
将字符数组转成字符串  
构造函数：String(char[])  
String(char[],offset.count):将字符数组值的一部分转换成字符串  
将基本数据类型转成字符串  
static String valueOf(int)  
static String valueOf(double)  
举例:比如3+"";是一个字符串，那么String.valueOf(3)和它一样  
将字节数组转成字符串  
String(byte[])  
String(byte[],offset,count):将字节数组中的一部分转成字符串  
将字符串转成字符数组  
char[] toCharArray();  
将字符串转成字节数组  
byte[] getBytes();  
替换：  
String replace(oldchar,newchar);  
切割：  
String[] split(regex);  
子串：获取字符串中的一部分  
String subString(begin);  
String subString(begin,end);  
转换，去除空格，比较  
1.将字符串转成大写或者小写。  
String toUpperCase();  
String toLowerCase();  
String trim();  
3.对两个字符串进行自然顺序的比较  
int comparTo(String);  
StringBuffer类:  
什么时候用呢？当数据类型不确定，而且最终变成字符串，而且数据的个数不确定，用这个比较方便，比  
数组方便  
字符串的组成原理就是通过该类实现的  
StringBuffer可以对字符串内容进行增删  
StringBuffer是一个容器,且是可变化的，字节可以操作多个数据类型，最终会通过toString方法变成字符

串，这就是StringBuffer的三个特点  
很多方法与String相同  
StringBuffer是可变长度的  
它是一个容器,所以就有以下功能：  
1.存储  
StringBuffer append():将指定数据作为参数添加到已有数据结尾  
StringBuffer insert(index，数据):可以将数据插入指定index  
2.删除  
StringBuffer delete(start,end):删除缓冲区中的数据，包含start,不包含end.  
StringBuffer  deleteCharAt(index):删除指定位置的字符  
3.修改  
StringBuffer replace(start,end,要替换的数据String)  
void setCharAt(int index,char ch)：替换  
4.反转  
StringBuffer reverse();  
5.void getChars(int srcBegin,int srcEnd,char[] dst,int dstBegin)  
将缓冲区中指定数据存储到指定数组中  
StringBuilder类:  
在jdk1.5之后出现的  
不同之处是StringBuffer是线程同步  
StringBuilder:是线程不同步的  
升级三个因素：  
提高效率  
简化书写  
提高安全性  
基本数据类型的对象包装类：  
byte-Byte short-Short int-Integer long-Long  boolean-Boolean  float-Float  
double-Double  char-Character  
基本数据类型对象包装类的最常见作用，就是用于基本数据类型和字符串类型之间作转换  
1.基本数据类型转字符串  
基本数据类型+""  
基本数据类型.toString(基本数据类型值)  
如：Integer.toString(34):就是将34整数变成"34"  
2.字符串转基本数据类型  
基本数据类型包装类.parse基本数据类型,比如：  
xxx a=xxx.parsexxx(String)  
int a=Integer.parseInt("123");  
double b=Double.parseDouble("12.23");  
boolean b=Boolean.parseBoolean("true");  
十进制转成其他进制  
toBinaryString();  
toHexString();  
toOctalString();  
其他进制转成十进制  
parseInt(String,radix);  
有一个intValue的方法也可以把一个Integer对象变成一个int值，比如  
Integer i=new Integer("123");  
int num=i.intValue();  
注意:Integer本身复写了Object里面的equals方法，它建立了本对象自己比较相同的特有方式，他比较的  
是数值是否相同。

[第八篇 黑马程序员-线程](http://blog.csdn.net/ljj810875/article/details/7888884)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:01 62人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888884#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888884#report)

[thread](http://blog.csdn.net/tag/details.html?tag=thread)[class](http://blog.csdn.net/tag/details.html?tag=class)[input](http://blog.csdn.net/tag/details.html?tag=input)[output](http://blog.csdn.net/tag/details.html?tag=output)[string](http://blog.csdn.net/tag/details.html?tag=string)[多线程](http://blog.csdn.net/tag/details.html?tag=%e5%a4%9a%e7%ba%bf%e7%a8%8b)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

创建线程的两种方式：  
继承Thread类  
实现Runnable接口  
1.定义类实现Runnable接口  
2.覆盖Runnable接口中的run方法  
注意：复写run方法的目的是将自定义的代码存储在run方法中，让线程运行  
将线程要运行的代码存放在该run方法中  
3.通过Thread类建立线程对象  
4.通过Runnable接口的子类对象作为实际参数传递给Thread类的构造函数  
为什么呢？因为自定义的run方法所属的对象是Runnable接口的子类对象，所以要  
让线程去指定指定对象的run方法。就必须明确该run方法所属对象  
5.调用Thread类的start方法开启线程并调用Runnable接口子类的run方法  
start方法有两个作用：  
一是启动线程，二是调用run方法  
线程有自己默认的名称：  
格式：Thread-编号，编号从0开始。  
static Thread currentThread():获取当前线程对象  
getName():获取线程名称  
设置线程名称：setName或者构造函数  
实现方式和继承方式有什么区别呢？  
实现方式好处是：避免了单继承的局限性，建议使用它  
继承方式：线程代码存放在Thread子类run()方法中  
实现方式：存在接口的子类的run方法中  
线程安全：  
我们先来看段代码：  
package thread;

public class Ticket implements Runnable//extends Thread   
{  
 private int tick=100;  
 public void run()  
 {  
  while(true)  
  {  
   if(tick>0)  
   {  
    System.out.println(Thread.currentThread().getName()+"....sale:"+tick--);

   }  
  }  
 }

}

package thread;

public class TicketDemo   
{  
 public static void main(String[] args)  
 {  
  /\*Ticket t1=new Ticket();  
  Ticket t2=new Ticket();  
  Ticket t3=new Ticket();  
  Ticket t4=new Ticket();

  t1.start();  
  t2.start();  
  t3.start();  
  t4.start();\*/  
  Ticket t=new Ticket();  
    
  Thread t1=new Thread(t);  
  Thread t2=new Thread(t);  
  Thread t3=new Thread(t);  
  Thread t4=new Thread(t);

  t1.start();  
  t2.start();  
  t3.start();  
  t4.start();  
    
 }

}  
分析：  
在判断中，假如tick大于1，满足条件，所以Thread-0准备去执行，但是有可能出现一种情况，就是Thread-0  
刚判断完正要执行的时候，但是被线程Thread-1给抢去了，而Thread-1正准备要执行，但是被Thread-2抢去了，Thread-2正要准备  
去执行的时候，但是被Thread-3给抢去了，这四个全部挂着，但是它们都有执行的资格，假设当Thread-1，Thread-2，Thread-3全部挂着的时候  
切换到Thread-0，它就买到1号票了，其他的线程因为不要再判断了，所以他们买到的票tick--就分别是0，-1，-2  
张错票，这显然不对，所以这里线程不安全。  
造成这个问题的原因：  
当多条语句在操作同一个线程共享数据时，一个线程对多条语句只执行了一部分，还没有执行完，  
另一个线程参与进来执行，导致共享数据的错误  
解决办法：  
对多条操作共享数据的语句，只能让一个都执行完，在执行过程中，其他线程不能参与进来  
Java对于多线程安全问题使用同步代码块来解决：  
synchronized(对象)  
{  
     需要被同步的代码       
}  
哪些语句正在操作共享数据，那哪些代码块就需要被同步，如下代码所示：  
package thread;

public class Ticket implements Runnable//extends Thread   
{  
 private int tick=1000;  
 Object obj=new Object();  
 public void run()  
 {  
  while(true)  
  {  
   synchronized(obj)  
   {  
    if(tick>0)  
    {  
     try {  
      Thread.sleep(10);  
     } catch (InterruptedException e) {  
        
     }  
     System.out.println(Thread.currentThread().getName()+"....sale:"+tick--);  
         }  
    }  
  }  
 }

}  
同步代码块的原理：  
用锁的思想去理解  
对象如同锁，持有锁的线程可以在同步中执行，没有锁的线程即使获取cpu的执行权，也进不去，因为没有获取锁。  
同步的前提：  
1.必须要有两个或者两个以上的线程  
2.必须是多个线程使用同一个锁  
必须保证同步中有一个线程在运行就可以了  
弊端：多个线程需要判断锁，较为消耗资源  
多线程-同步函数：  
如何同步？  
1.明确哪些代码是多线程运行代码  
2.明确共享数据  
3.明确多线程运行代码中哪些语句是操作共享数据的  
在函数的返回值前面加上synchronized，它就变成同步函数了  
同步函数用的是哪一个锁呢？  
函数需要被对象调用，函数都有一个所属对象的引用，那就是this  
所以同步函数使用的锁是this。  
如果同步函数被静态修饰后，使用的锁是什么呢？  
通过验证，发现不是this。因为静态方法中也不可以定义this。  
静态进内存时，内存中没有本类对象，但是一定有该类对应的字节码文件对象，所以它的锁可以用字节码文件对象  
格式：类名.class。  
该对象的类型是Class.  
单例设计模式：  
饿汉式：  
class Single  
{  
    private static final Single s=new Single();  
    private Single(){}  
    public static Single getInstance()  
 {  
      return s;  
  }  
}  
懒汉式：  
class Single  
{  
    private static final Single s=null;  
    private Single(){}  
    public static synchronized Single getInstance()  
    {  
         if(s==null)  
               s=new Single();  
         return s;  
 }  
   
class SingleDemo  
{  
     public static void main(String[] args)  
     {  
           System.out.println("Hello World");

class Single2  
{  
    private static final Single2 s=null;  
    private Single2(){}  
    public static  Single getInstance()  
    {  
         if(s==null)  
         ｛  
             synchronized(Single2.class)  
             {  
                 if(s==null)  
                 s=new Single();  
         return s;  
 }  
   
class SingleDemo  
{  
     public static void main(String[] args)  
     {  
           System.out.println("Hello World");

懒汉式面试经常问到。  
第二种方式效率提高了。  
单例设计模式涉及的知识点必须要会！。  
线程间通信，其实就是多个线程在操作同一个资源  
看段代码：  
package thread;

public class Res   
{  
 String name;  
 String sex;  
}

package thread;

public class Input implements Runnable  
{  
 private Res r;  
 Input(Res r)  
 {  
  this.r=r;  
 }

 @Override  
 public void run()   
 {  
  int x=0;  
  while(true)  
  {  
  if(x==0)  
  {  
     r.name="mike";  
     r.sex="man";  
  }  
  else  
  {  
     r.name="丽丽";  
     r.sex="女女女女女女女女女";  
 }  
   
  x=(x+1)%2;  
 }

}  
}

package thread;

public class Output  implements Runnable  
{  
 private Res r;  
 Output(Res r)  
 {  
  this.r=r;  
 }

 public void run()   
 {  
  while(true)  
  {  
   System.out.println(r.name+"....."+r.sex);  
  }  
 }  
}

package thread;

public class InputOutputDemo  
{  
 public static void main(String[] args)  
 {  
  Res r=new Res();  
    
  Input in=new Input(r);  
  Output out=new Output(r);  
    
  Thread t1=new Thread();  
    
  Thread t2=new Thread();  
    
  t1.start();  
  t2.start();

 }

}  
上面代码有安全问题，打印出来的结果出现了mike为女，而丽丽为男的情况。这是怎么造成的呢？  
这是因为当要输出mike名字时，接着准备输出性别时,而被“丽丽”给抢去了，所以会打印出  
丽丽为男。  
这该怎么解决呢？  
出现安全问题，我们用同步。  
注意：凡是出现runnable接口我们一般都得建对象。  
如果两个类里的方法有一个同步了而另一个没有同步，那么也是没有满足第一个前提：至少有两个或两个以上的线程  
它们是处理同一个资源。所以两边都必须同步。但是两边都同步以后还是报错，这里第一个前提已经满足了，  
所以我们再来看第二个前提：必须是多个线程使用同一个锁。目前他们用的还不是同一个锁，那传个this可以吗？答案是no  
因为Input有个this，Output类也有个this，是两个不同的对象，那就要思考，这内存里有没有唯一对象，那传个Input.class对象  
行不行？答案是：yes,因为内存里有四个唯一的类：Input.class  Output.class  Res.class  InputOutputDemo.class.所以传个  
Input.class是靠谱的。但是这太过牵强，我们看下这里哪个对象是唯一的，当然是r了，因为它是唯一对象，在程序里表示唯一的资源，打印的结果是一大片mike.....man或一大片  
丽丽.......女女女女女女女女女女女女，这样看起来不爽，我们必须要它这样打印：一行一行的切换，为什么会产生  
这样的现象呢？分析：  
如果输入的线程即Input的线程获得CPU执行权，它存了一个mike......man,它在存的时候别个是进不来的，存完以后，  
这个时候，它出了同步的话，Input和Output都有可能再次抢到CPU执行权，所以Input还有可能抢到执行权，它就把  
丽丽......女输入进去了。前面的值就被覆盖掉了，它有了CPU执行权以后，就会一直输入，把前面的值覆盖掉，当输入  
到某个时刻，它的执行权被Output抢走了，它会输出多次。所以造成这种现象。这是CPU切换的时候造成的。  
我们现在想实现这种需求：交替执行，且是一次一次的交替执行。怎么做呢？方法是加个标记，如加个boolean flag=false;  
它的原理是：当Input线程在输入数据的时候，就判断这个标记，看里面是否有值，如果为false，那就表示没值，  
那输入线程就往里面存了一个mike....man,输入以后，Input线程还会有执行权，它其实要做一件事情，就是存完数据以后  
就把标记改为true，那就表示有数据了，当它再次判断时候，看到有数据，它就不往里面输了，因为原来的还没输出呢，这时候就需要  
它等着别动，有人会想到用sleep，那睡多长时间合适呢？这是不确定的，而它什么时候应该醒呢？应该是我把数据都输入进去完了，  
然后再输出完了它才能够醒，所以这里最好的办法是用wait方法，它可以等待唤醒，一wait就会冻结，即放弃执行资格，所以Output  
线程就会执行，它本来是具有执行资格，但是没执行权，当Input线程一wait,它就有执行权了，它就开始输出  
它输出之前也要做一次判断，如果判断为true，即里面有数据，它就输出，输出完以后把标记置为false，这时它还有执行权，当它再次判断的时候，  
一看是false，它就不会再输出，这时候它就会等着，这时Input 线程等着，它也等着，程序就挂了，所以必须保证  
一个线程是活着的，这怎么办呢？方法是Output在等之前应该把Input线程叫醒，叫它往里边输入数据，这时Output线程  
是等着的，当Input线程被叫醒了以后，它就去判断，一看是false，即里面没数据，所以它就往里面存数据，它这时  
还有执行权，然后再去判断，一看是true，所以它就又要等着，在等之前就唤醒Output线程。这就是等待唤醒机制，这在程序开发中是  
非常常见的。必须掌握。  
其实线程都是放在线程池里，我们使用notify唤醒线程的时候，其实唤醒的是线程池里面的  
线程，到底是先唤醒哪个线程呢？通常是唤醒第一个被等待的。  
在同步里面必须标识出wait它所操作的那个线程所属的锁。这里r.wait即标识出锁r.这里wait表示持有r这个锁的线程。  
为什么？因为同步会出现嵌套，即两个锁的时候。我notily的时候是唤醒r这个锁上的等待线程。  
为什么wait和notify这些专门操作线程的方法定义在Object当中？  
锁是任意对象，任意对象能调用的方法必然应该定义在Object当中。  
等待和唤醒必须是同一个锁。  
生产者和消费者的问题：  
分析以下代码：  
package thread2;

public class Resource {  
 private String name;  
 private int count = 1;  
 private boolean flag = false;

 public synchronized void set(String name) {  
  if(flag)  
   try {  
    this.wait();  
   } catch (InterruptedException e) {  
    e.printStackTrace();  
   }  
  this.name = name + "--" + count++;// 这里传名字带编号，如张三+编号  
  System.out.println(Thread.currentThread().getName() + "...生产者..."  
    + this.name);// currentThread:返回对当前正在执行的线程对象的引用。返回：当前执行的线程。  
  flag=true;  
  this.notify();

 }

 public synchronized void out() {  
  if(!flag)  
   try {  
    this.wait();  
   } catch (InterruptedException e) {  
    e.printStackTrace();  
   }  
  System.out.println(Thread.currentThread().getName() + "...消费者..."  
    + this.name);  
  flag=false;  
  this.notify();  
 }

}

package thread2;

public class Producer implements Runnable {  
 private Resource res;

 Producer(Resource res) {  
  this.res = res;  
 }

 @Override  
 public void run() {  
  while (true) {  
   res.set("+商品+");  
  }

 }

}

package thread2;

public class Consumer implements Runnable {  
 private Resource res;

 Consumer(Resource res) {  
  this.res = res;  
 }

 @Override  
 public void run() {  
  while (true) {  
   res.out();

  }  
 }

}

package thread2;

public class ProducerConsumeDemo {  
 public static void main(String[] args) {  
  Resource r = new Resource();  
  Producer pro = new Producer(r);  
  Consumer con = new Consumer(r);

  Thread t1 = new Thread(pro);  
  Thread t2 = new Thread(pro);  
  Thread t3 = new Thread(con);  
  Thread t4 = new Thread(con);

  t1.start();  
  t2.start();  
  t3.start();  
  t4.start();  
 }

}  
以上程序会打印出生产两个商品而只消费一个商品，或者生产一个商品而被消费了两次，  
为什么会出现生产两个商品而只消费一个商品呢？分析如下：  
生产者有：t1,t2  
消费者有:t3,t4  
假设当t1获得CPU执行权的时候，t1判断标记为假，它不需要等待，它就往里面赋了一个值，然后count++一次，再打印  
如果它赋的值是130，那么假设它打印的是Thread-0...生产者...+商品+--130（当然，有可能也会是Thread-1...生产者...+商品+--130），  
即表示生产了一次，生产完以后，就把标记置为真了，然后执行notify,notify完以后，它这时是还有执行权的，  
然后去再次判断，一看flag为true,它于是就等着，这时t2,t3,t4都有可能抢到CPU执行权，假设t2抢到了，它于是去判断标记，  
一看为真，于是它也就等着了，这时只剩下t3和t4了，t3一进来判断标记!flag为假(即非真即为假，这里以前我一直弄不明白，这里解释以下，以前  
我总是认为这里判断应该是!flag即为真，其实不是这样的，因为之前t1消费完以后就把flag置为true了，所以t3  
去判断!flag即为假，是假的话那么就表示t3消费了商品了），不需要等，即消费了一次，到目前为止，这属于正常消费，  
即生产一次，消费一次，消费完了以后，它就把标记置为false了，然后notify一次，这时它就把t1唤醒了，这时t1  
有资格，但是不一定有执行权，这时执行权在t3那里，然后它再次判断标记!flag为真，即不能再消费了，这时它就wait  
即等着，这时如果t4抢到执行权，它就去判断标记，一看为真，它也就wait了，这时t1是活着的，它这时它是从程序中的  
“this.name = name + "--" + count++;”这段语句开始运行的，即它又count++了一次，即它又生产了一次，当t1生产完  
一个新的商品以后，然后下来执行flag=true,即把标记flag置为真了，置完真以后然后notify  
一次，把t2唤醒了(因为t2进线程池是先进去的，比t3,t4都早，所以唤醒的是它),这时t2获取资格，但是不一定有执行权，  
这时t1还是有执行权的，然后它再次去判断标记，一看是true，它就wait了，即等着，放弃资格了，这时活着的  
的只有t2,这时要注意了，这时t2不会再去判断标记了，为什么呢？因为它本来是等着的（即它原来已经判断过了，不会再判断了  
，因为if语句置判断一次的，如果是while就可以判断多次)应该就往下执行了，即从this.name = name + "--" + count++;  
这个语句开始执行，这时它又新生产了一个商品，把前一个商品覆盖掉了（它就把t1那时生产的商品给覆盖掉了，之前t1  
生产了一个，由于t3,t4都放弃资格了，所以没有被消费，就被t2新生产的商品覆盖掉了），这就导致生产了两个，即  
Thread-1...生产者...+商品+--130  
Thread-0...生产者...+商品+--131  
即Thread-1...生产者...+商品+--130刚生产完，又被t2又生产了一次，导致在t3消费的时候只消费了  
Thread-0...生产者...+商品+--131一次，。  
当t2生产完，它就notify，就把t3唤醒了，t3就消费t2生产的那个了，t1那个没消费成功，之所以出现这种情况  
是因为t2没去判断标记，那我们怎样做才能要其醒的时候去判断标记呢？方法：  
因为if只判断一次，我们把它改成while就可以判断多次了，但是这样做的话，就会出现他们全部等待的状况  
这不是死锁，而是叫全部等待，你等待我，我等待你，全都动不了。全都冻结了。即t1把t2给唤醒了，即  
把本方给唤醒了，而t3 t4都没有被唤醒。要把对方唤醒才靠谱，所以把notify换成notifyAll就可以了，  
把他们全部都唤醒。  
我们说过了这样做的话会导致全部等待的状况发生，要解决这个问题的话，就需要在全部唤醒的时候，不要唤醒本方，  
只把对方唤醒，这该怎么办呢？我们就要用到jdk5的新特性了，jdk5中出现了Lock,Lock 就把sychronized替代  
了，wait,notify,notifyAll方法 分别被condition里面的await,signal，signall给替代了，  
Condition把wait,notify等对象封装了，这些condition对象怎么来呢？  
答案是通过锁来。为什么呢？因为像wait等方法是定义在同步语句块中，同步语句块中必然有锁，每个wait,notify  
都要标识自己所属的锁，现在是Lock,wait,notify变成了Condition,那么这个Condition怎么获取？同样道理，  
它需要通过锁获取。  
释放资源需要被定义在finally里，即需要把释放锁unlock定义在finally里。  
为了解决这个问题，我们需要在Resource类里面定义两个Condition对象，即  
private Condition condition\_pro = lock.newCondition();  
private Condition condition\_con = lock.newCondition();  
它的执行状况是这样的：  
生产者获取锁，一进来，判断标记，如果为真，生产者就必须等待，即t1,t2都等待，然后消费者被唤醒，即执行condition\_con.singal();  
紧接着消费者获取了锁，一进来，判断标记满足条件的话，消费者许等待，所以它就下来执行condition\_pro.signal();  
把生产者t1唤醒了（即她只唤醒生产者这里重点强调下），这时它还有执行权，然后去判断标记!flag为真，这时它就等着，  
然后t4去判断标记，一看为真，它也就等着，由于这时t1被唤醒了，他就从this.name = name + "--" + count++;// 这里传名字带编号，如张三+编号  
开始执行，当执行到condition\_con.signal();的时候，即唤醒消费者其中一个（这时它没去唤醒同类对象了，即没  
唤醒t2,而是去唤醒消费者），这就搞定了，即生产者里面代码唤醒消费者，消费者代码里面唤醒生产者。  
这就是一个锁里面可以绑定好几个Condition对象，而以前的只能绑定一个，而且挨个拿锁去区分，很麻烦，而锁  
一嵌套，就死锁，这是新特性的好处。新特性中把wait,notify封装成了Condition对象，它一个锁可以对应多个Condition,以前的是一个锁对应一个waot,notify  
停止线程：  
stop方法已经过时，那么停止线程就只有一种方法，那就是run方法结束：  
开启多线程运行，运行代码通常是循环结构，只要控制住循环，就可以要run方法结束，也就是线程结束。  
在Thread类当中有一个interrupt()方法，中断线程的意思，中断线程不是停止线程的意思，当线程处于冻结状态的时候  
就可以用这个方法，强制清除其冻结状态。让其恢复到运行状态中来，因为是强制的，所以会发生异常。  
守护线程：  
setDaemon方法：  
将线程标记为守护线程或用户线程。  
守护线程也叫后台线程，它随着前台线程的结束而结束。  
join方法：  
通俗说就是抢夺CPU执行权，  
它是临时加入线程用的，当一个线程t被设置为join，那么主线程会把执行权给它，主线程这时就处于冻结状态，  
它必须等到t线程结束后，主线程才可以运行。  
当A线程执行到了B线程的.join方法时，A就会等待，等B线程都执行完，A才会执行。  
yield方法：  
暂停当前正在执行的线程对象，并执行其他线程。

[第九篇 黑马程序员-集合](http://blog.csdn.net/ljj810875/article/details/7888892)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:02 27人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888892#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888892#report)

[数据结构](http://blog.csdn.net/tag/details.html?tag=%e6%95%b0%e6%8d%ae%e7%bb%93%e6%9e%84)[equals](http://blog.csdn.net/tag/details.html?tag=equals)[存储](http://blog.csdn.net/tag/details.html?tag=%e5%ad%98%e5%82%a8)[object](http://blog.csdn.net/tag/details.html?tag=object)[hashmap](http://blog.csdn.net/tag/details.html?tag=hashmap)[iterator](http://blog.csdn.net/tag/details.html?tag=iterator)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

为什么出现集合类？  
面向对象语言对事物的体现都是以对象的形式，所以为了方便对多个  
对象的操作，就对对象进行存储，集合就是存储对象最常用的一种方式  
数组和集合有什么不同？  
数组虽然也可以存储对象，但长度是固定的，集合的长度是可变的  
数组中可以存储基本数据类型，集合只能存储对象。  
集合类的特点：  
集合只用于存储对象，集合长度是可变的，集合可以存储不同类型的对象  
列表迭代器:listIterator  
是Iterator的子接口，在迭代时，不可以通过集合对象的方法操作集合中的元素，因为会  
发生并发执行异常，所以在迭代时，只能用迭代器方法操作，但是迭代器不能添加，  
所以用其子接口listiterator。  
枚举就是vector特有的取出方式  
枚举和迭代器很像。是一样的  
枚举的名称和方法名过长，所以被迭代器代替  
ArrayList的底层数据结构是数组结构。  
特点：查询快，增删慢。  
LinkedList的底层数据结构是链表结构  
特点：查询慢，增删较快。  
HashSet:底层是哈希表  
Set集合的功能和Collection是一致的  
hashCode:返回该对象的哈希码值。  
假设有哈希值：  
[Demo@3c](mailto:Demo@3c)  
[Demo@3c](mailto:Demo@3c)  
他们不是同一个对象。  
[Demo@c17164](mailto:Demo@c17164)  
[Demo@lfb8ee3](mailto:Demo@lfb8ee3)  
在哈希表里面当元素重复的时候，它还有一次校验方式：  
如果地址值不一样，你有你的位置，我有我的位置。咱两不是一个元素，如果一样的话，那还要作次判断，  
就是判断元素对象是否相同。即判断位置相同以后，还要判断是否是同一个对象。  
判断是否是同一个对象用equals方法。  
当发现位置是一样的，但是不是同一个对象，这时会在该地址下顺延，又产生一个对象，  
这个对象的位置和其是一样的，这时如果哈希值c17164存进来，它在往里面存的话先要和3c碰一下，  
判断地址是否一样，如果不一样，它就单独放一个位置。这就是哈希表结构的特点  
Set方式取出只有一种方式那就是迭代器。  
hashSet是如何保证元素唯一性的呢？  
答案:通过元素的两个方法，hashCode和equals来完成，如果元素的HashCode值相同，  
才会判断equals是否为true,如果元素的hashCode值不同，不会调用equals.  
ArrayList判断元素和删除元素依赖的是equals方法，而HashSet依赖的是hashCode和equals方法。  
数据结构不同依赖的方法也不一样  
目录：  
1.TreeSet  
2.TreeSet存储自定义对象  
3.二叉树  
4.实现Comparator方式排序  
5.TreeSet练习  
6.泛型概述  
7.泛型使用  
8.泛型类  
9.泛型方法  
10.静态方法泛型  
11.泛型接口  
12.泛型限定  
13.泛型限定2  
TreeSet:  
可以对Set集合中的元素进行排序.底层数据结构是二叉树.  
保证元素唯一性的依据：compareTo方法和return 0.  
TreeSet排序的第一种方式：让元素自身具备比较性，元素需要实现Comparable接口  
覆盖compareTo方法。这种方式也称为元素的自然排序，或者叫做默认顺序  
TreeSet的第二种排序方式：当元素自身不具备比较性时，或者具备的比较性不是所需要的  
这时就需要让集合自身具备比较性，在集合初始化时，就有了比较方式  
Comparable<T>接口：  
此接口强行对实现它的每个类的对象进行整体排序，这种排序称为类的自然排序。  
TreeSet可以进行排序，但是按照什么方式排，你没和它说过，你想说按照年龄排，但是它不知道  
为什么不知道？因为这个学生对象根本就不具备比较性。只有存到它里面的对像具备比较性，它才可以对其进行  
排序，所以为了让其具备比较性，可以让其实现Comparable接口。就具备比较性了。当其  
具备了比较性，那么TreeSet就可以使用其比较性。因为它预先就在使用这个接口中的方法，多态  
该接口中只有一个方法，那就是compareTo方法。此方法的作用是比较此对象与指定对象的顺序  
返回的是正数，零或负数。  
字符串本身具备比较性，但是它的比较方式不是所需要的，这时就只能使用比较器  
泛型：  
泛型:jdk1.5版本以后出现的新特性，用于解决安全问题，是一个安全机制。  
好处  
1.将运行时期出现问题ClassCastException,转移到了编译时期  
方便于程序员解决问题，让运行时期问题减少，安全。  
2,避免了强制转换的麻烦  
泛型格式:通过<>来定义要操作的引用数据类型  
在使用java提供的对象时，什么时候写泛型呢？  
通常在集合框架中很常见，只要见到<>就要定义泛型  
其实<>就是用来接受类型的，当使用集合时，将集合中要存取的数据类型作为参数传递到<>即可。  
泛型类：  
什么时候定义泛型类?  
当类中要操作的引用数据类型不确定的时候  
早期定义Object来完成扩展。  
现在定义泛型来完成扩展  
泛型类定义的泛型，在整个类中有效，如果被方法使用  
那么泛型类的对象明确要操作的具体类型后，所有要操作的类型就已经固定了  
为了要不同方法操作不同类型，而且类型不确定，那么可以将泛型定义在方法上  
特殊之处:  
静态方法不可以访问类上定义的泛型  
如果静态方法操作的应用数据类型不确定，可以将泛型定义在方法上  
?表示通配符，也可以理解为占位符  
泛型的限定:  
? extends E:可以接受E类型或者E的子类型。上限  
? super E:可以接收E类型或者E的父类型  
Map集合：  
该集合存储键值对，往里存，保持键的唯一性。  
1.添加  
    put(K key, V value)  
    putAll(Map<? extends K,? extends V> m)  
2，删除  
    remove(Object key)  
    clear()  
3.判断  
     containsValue(Object value)  
     containsKey(Object key)  
4，获取  
     size()  
     get(Object key)  
     values()  
       
     entrySet()  
     keySet()  
       
Map集合:  
Hashtable:底层是哈希表数据结构，不可以存入null作为键，nullL作为值的情况，该集合是线程同步的  
HashMap：底层是哈希表数据结构，允许使用null键为空值，该集合是不同步的。效率高  
TreeMap:底层是二叉树数据结构，线程不同步，可以用于给map集合中的键排序。  
和set很像  
其实set底层就是使用了MAP集合。  
map集合的两种取出方式:  
1.Set<k> keySet： 将map中所有的键存入set集合。因为set具备迭代器，  
所以可以用迭代方式取出所有的值，在根据get方法获取每一个键对应的值

map集合的取出原理:将map集合转成set集合,再通过迭代取出。  
2.Set<Map.Entry<k,v>> entrySet:将map集合中的映射关系存入到了set集合中,  
而这个关系的数据类型就是:Map.Entry

[第十篇 黑马程序员-IO输入输出流](http://blog.csdn.net/ljj810875/article/details/7888897)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:04 20人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888897#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888897#report)

[null](http://blog.csdn.net/tag/details.html?tag=null)[java](http://blog.csdn.net/tag/details.html?tag=java)[exception](http://blog.csdn.net/tag/details.html?tag=exception)[android](http://blog.csdn.net/tag/details.html?tag=android)[string](http://blog.csdn.net/tag/details.html?tag=string)[io](http://blog.csdn.net/tag/details.html?tag=io)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一：  
1.IO流用来处理设备之间的数据传输。  
  计算机中有很多设备，如硬盘，内存等等，硬盘中的文件，内存中数据的输入输出，在Java中我们把它叫做IO流。  
2.Java对数据的操作方式是通过流的形式。  
  计算机设备中数据的输入输出，在Java中是通过流来操作的。  
3.Java用于操作流的对象都在IO包中。  
4.流按操作数据分为两种：字节流与字符流。根据老师的建议我开始自学下载下来的  
  字节流是由字节组成的，所有的InputStream和OutputStream的子类都是，用在处理二进制数据，主要用来处理图片，它的基类是InputStream,OutputStream.  
  字符流是按虚拟机的encode来处理，也就是要进行字符集的转换，主要用来处理文字,它的基类是Reader,Writer。  
5.流按流向分为：输入流，输出流。

二：  
1.字符流类Writer中的方法：write:将字符串写入到流中。flush:将数据刷新到目的地中。close:关闭流。  
2.try....catch():IO异常处理。  
3.文本文件的两种读取方式：1.字符读取。2.字符数组读取

三：  
1.BufferedWriter:将文本写入字符输出流并缓冲字符以便能提供单字符、数组和行的高效写入。  
BufferedReader:缓冲字符输入流。  
2.通过缓冲区复制文本文件(代码演示)：

import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
public class CopyTextByBuf {  
 public static void main(String[] args) {  
  BufferedReader bufr=null;  
  BufferedWriter bufw=null;  
  try {  
   //创建一个要读取的文件的流  
   bufr=new BufferedReader(new FileReader("D:\\新建文件夹[\\IOtest\\src\\CopyTextByBuf.java](file:///\\iotest\src\CopyTextByBuf.java)"));  
   //创建一个要存放读取的流的文件  
   bufw=new BufferedWriter(new FileWriter("se.txt"));  
   String line=null;//两个流之间的中转站  
   while((line=bufr.readLine())!=null)  
   {  
    bufw.newLine();//读取一个文本行  
             bufw.write(line);  
    bufw.flush();  
   }  
  } catch (Exception e) {  
   throw new RuntimeException("读入失败");  
  }  
  finally{  
   try {  
    if(bufr!=null)  
    bufr.close();  
   } catch (IOException e) {  
    throw new RuntimeException("读取关闭失败");  
   }  
      try {  
       if(bufw!=null)  
     bufw.close();  
   } catch (IOException e) {  
       throw new RuntimeException("写入关闭失败");    
   }     
  }  
 }  
}

[第十一篇 黑马程序员-IO输入输出](http://blog.csdn.net/ljj810875/article/details/7888993)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:24 67人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7888993#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7888993#report)

[null](http://blog.csdn.net/tag/details.html?tag=null)[string](http://blog.csdn.net/tag/details.html?tag=string)[file](http://blog.csdn.net/tag/details.html?tag=file)[exception](http://blog.csdn.net/tag/details.html?tag=exception)[reference](http://blog.csdn.net/tag/details.html?tag=reference)[android](http://blog.csdn.net/tag/details.html?tag=android)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

一：  
1.readLine的原理：读取一个文本行。  
通过下列字符之一即可认为某行已终止：换行（'\n')、回车（'\r')或回车后直接跟着换行。  
返回：包含该行内容的字符串，不包含任何行终止符，如果已到达流末尾，则返回 null。  
2.装饰设计模式：装饰模式是在不必改变原类文件和使用继承的情况下，动态扩展一个对象的功能。它是通过创建一个包装对象，也就是装饰来包裹真实的对象。  
3.装饰模式的特点：（1） 装饰对象和真实对象有相同的接口。  
                  （2） 装饰对象包含一个真实对象的索引（reference）  
                  （3） 装饰对象接受所有的来自客户端的请求，它把这些请求转发给真实的对象。  
                  （4） 装饰对象可以在转发这些请求以前或以后增加一些附加功能。这样就确保了在运行时，不用修改给定对象的结构就可以在外部增加附加的功能。

二：  
装饰模式和继承的区别：  
1.装饰模式：用来扩展特定对象的功能。继承： 用来扩展一类对象的功能。  
2.装饰模式：不需要子类。继承：需要子类。  
3.装饰模式：动态的。继承：静态的。  
4.装饰模式：运行时分配职责。继承：编译时分配职责。  
5.装饰模式：防止由于子类而导致的复杂和混乱。继承：导致很多子类产生，在一些场合，报漏类的层次。  
6.装饰模式：更多的灵活性。继承：缺乏灵活性。

三：  
1.LineNumberReader:维护一个计数器，该计数器表明正在读取的是哪一行。  
2.文件的读写操作（代码演示）：  
读文件代码演示：  
 public static void readFileBytes(String fileName) {  
  File file = new File(fileName);  
  InputStream in = null;  
  try {  
   logger.debug("以字节为单位读取文件内容，一次读多个字节：");  
    //一次读多个字节  
   byte[] tempbytes = new byte[100];  
   int byteread = 0;  
   in = new FileInputStream(file);  
    //读多个字节到字节数组中，byteread为一次读入的字节数  
   while ((byteread = in.read(tempbytes)) != -1) {  
    logger.debug(tempbytes);  
    logger.debug(0);  
    logger.debug(byteread);  
   }  
  } catch (Exception e1) {  
   logger.error("读取文本文件异常!",e1);  
  } finally {  
   if (in != null) {  
    try {  
     in.close();  
    } catch (IOException e1) {  
     logger.error("读取文本文件异常!",e1);  
    }  
   }  
  }  
 }  
 

写文件代码演示：  
public static boolean writerFile(String filePathName,String content){  
  boolean flag=false;  
  OutputStreamWriter os=null;  
  try {  
   if(filePathName!=null&&!"".equals(filePathName)){  
    os = new OutputStreamWriter(new FileOutputStream(filePathName));  
   }  
  } catch (FileNotFoundException e1) {  
   flag=false;  
   e1.printStackTrace();  
  }   
  if(os!=null){  
  BufferedWriter bw=new BufferedWriter(os);   
  try {  
   if(content!=null&&!"".equals(content)){  
    bw.write(content);  
    flag= true;  
   }  
  } catch (IOException e) {  
   flag=false;  
   e.printStackTrace();  
  }finally{  
   try {  
    bw.close();  
    os.close();  
   } catch (IOException e) {  
    flag=false;  
    e.printStackTrace();  
   }      
  }  
  }  
  return flag;  
 }  
}

[第十二篇 黑马程序员-IO输入输出流](http://blog.csdn.net/ljj810875/article/details/7889003)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:25 19人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889003#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889003#report)

[string](http://blog.csdn.net/tag/details.html?tag=string)[integer](http://blog.csdn.net/tag/details.html?tag=integer)[android](http://blog.csdn.net/tag/details.html?tag=android)[java](http://blog.csdn.net/tag/details.html?tag=java)[delete](http://blog.csdn.net/tag/details.html?tag=delete)[file](http://blog.csdn.net/tag/details.html?tag=file)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一：  
1.字节流可以通过定义一个字节数组作为缓冲，这个缓冲在传送文件时是很有必要的，如果没有这个数组作为缓冲，CPU资源消耗会很高。  
2.Java中如何从键盘录入  
  利用 Scanner 实现从键盘读入integer或float 型数据。  
  利用 BufferedReader实现从键盘读入字符串并写进文件test.txt中（代码演示）。  
  import java.io.\*;  
  public class Test{  
  public static void main(String[] args) throws IOException{  
  BufferedReader buf = new BufferedReader(new InputStreamReader(System.in));  
  BufferedWriter buf2 = new BufferedWriter(new FileWriter("test.txt"));  
  String str = buf.readLine();  
  while(!str.equals("exit")){  
   buf2.write(str);  
   buf2.newLine();  
   str = buf.readLine();  
  }  
  buf.close();  
  buf2.close();  
 }  
}  
3.流操作规律：  
通过两个明确来完成:  
明确源和目的  
   源：输入流。InputStream  Reader。  
   目的：输出流。OutputStream Writer。  
操作的数据是否是纯文本。  
   是：字符流。  
   否：字节流。  
当体系明确后，在明确要使用哪个具体的对象。  
   通过设备来进行区分：  
   源设备：内存，硬盘，键盘。  
   目的设备：内存，硬盘，控制台。

二；  
1.File类的常见方法：  
创建文件的方法  
   创建文件：boolean createNewFile()。  
   创建文件夹：boolean mkdir()。  
   创建多级目录：boolean mkdirs()。  
       
删除文件的方法  
   删除文件:boolean delete(), deleteOnExit()。

判断的方法   
   是否是绝对路径：isAbsolute()。  
   是否存在：exists()。

获取的方法  
   文件大小：long length()。  
   文件名：getName()。  
   父目录：getParent()。  
   路径：getPath()。  
   绝对路径：getAbsolutePath()。  
   获取指定目录下当前的文件夹和文件名称：String[] list()。  
   获取指定目录下当前的文件夹和文件对象：File[] listFiles()。  
   获取指定过滤器过滤后的文件或者文件夹对象：File[] listFiles(FileFilter)

[第十三篇 黑马程序员-IO输入输出流](http://blog.csdn.net/ljj810875/article/details/7889013)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:27 177人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889013#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889013#report)

[properties](http://blog.csdn.net/tag/details.html?tag=properties)[import](http://blog.csdn.net/tag/details.html?tag=import)[string](http://blog.csdn.net/tag/details.html?tag=string)[android](http://blog.csdn.net/tag/details.html?tag=android)[java](http://blog.csdn.net/tag/details.html?tag=java)[byte](http://blog.csdn.net/tag/details.html?tag=byte)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一：  
1.Properties类

Properties存取配置文件（代码演示）：

package test.bst;     
    
import java.io.FileNotFoundException;     
import java.io.IOException;     
import java.io.InputStream;     
import java.util.Properties;     
    
public class Test {     
    private static Properties properties = new Properties();     
    
    public static void main(String[] args) {     
        try {     
            InputStream is = Test.class.getClassLoader().getResourceAsStream("cache.properties");     
            properties.load(is);     
            String size = properties.getProperty("cache.size");     
            writeLog("配置成功！" + size);     
        } catch (FileNotFoundException e) {     
            writeLog("配置文件不存在！" + e.getMessage());     
        } catch (IOException e) {     
            writeLog("读取配置文件IO错误！" + e.getMessage());     
        }     
    }     
    
    public static void writeLog(String strLog) {     
        System.out.println(strLog);     
    }     
}

2.PrintWriter和PrintStream类:

PrintWriter类的作用是向文本输出流打印对象的格式化表示形式。实现在 PrintStream 类中的所有 print 方法。  
它不包含用于写入原始字节的方法，对于这些字节，程序应该使用未编码的字节流写入。  
它如果启用了自动刷新，则只有在调用 println、printf 、 format 的其中一个方法时才可能完成此操作，  
而不是每当正好输出换行符时才完成。这些方法使用平台自有的行分隔符概念，而不是换行符

PrintStream类不会抛出 IOException异常，而是，异常情况仅设置可通过checkError方法测试的内部标志。  
为了自动刷新，可以创建一个 PrintStream；这意味着可在写入 byte 数组之后自动调用 flush 方法，可调用其中一个 println 方法，或写入一个换行符或字节。   
它可以使输出流方便的打印各种数据值的表示形式。

3.合并流:

合并流的主要作用是将两个文件的内容合并成一个文件。

SequenceInputStream类常用的方法:

SequenceInputStream(InputStream s1, InputStream s2):  通过记住这两个参数来初始化新创建的 SequenceInputStream,  
以提供从此 SequenceInputStream 读取的字节。

available():返回文件的大小。

4.切割文件:用输入流去读取文件，将读取的规定大小的流输出到指定的文件，直到把整个文件读取完。

[第十四篇 黑马程序员-IO输入输出流](http://blog.csdn.net/ljj810875/article/details/7889025)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:29 20人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889025#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889025#report)

[string](http://blog.csdn.net/tag/details.html?tag=string)[java](http://blog.csdn.net/tag/details.html?tag=java)[android](http://blog.csdn.net/tag/details.html?tag=android)[存储](http://blog.csdn.net/tag/details.html?tag=%e5%ad%98%e5%82%a8)[internet](http://blog.csdn.net/tag/details.html?tag=internet)[byte](http://blog.csdn.net/tag/details.html?tag=byte)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一:  
1.RandomAccessFile类:

此类的实例支持对随机访问文件的读取和写入。随机访问文件的行为类似存储在文件系统中的一个大型 byte 数组。  
存在指向该隐含数组的光标或索引，称为文件指针；输入操作从文件指针开始读取字节，并随着对字节的读取而前移此文件指针。  
如果随机访问文件以读取/写入模式创建，则输出操作也可用；输出操作从文件指针开始写入字节，并随着对字节的写入而前移此文件指针。  
写入隐含数组的当前末尾之后的输出操作导致该数组扩展。该文件指针可以通过 getFilePointer 方法读取，并通过 seek 方法设置。

2.DataInputStream和DataOutputStream类:

DataInputStream类:数据输入流允许应用程序以与机器无关方式从底层输入流中读取基本数据类型。  
DataOutputStream类:数据输出流允许应用程序以适当方式将基本 Java 数据类型写入输出流中。然后，应用程序可以使用数据输入流将数据读入。

3.ByteArrayInputStream类和ByteArrayOutputStream类: 包含一个内部缓冲区，该缓冲区包含从流中读取的字节。  
内部计数器跟踪 read 方法要提供的下一个字节。

ByteArrayInputStream类:此类实现了一个输出流，其中的数据被写入一个 byte 数组。缓冲区会随着数据的不断写入而自动增长。  
可使用 toByteArray() 和 toString() 获取数据。

4.字符编码转换:

所谓编码转换就是一个字符与字节之间的转换，通过子类转换流（InputStreamReader和OutputStreamWriter)来完成，  
因为这两个类是字节流和字符流之间的适配器类。

5.字符编码:

String对象的编码:

Java的String内部有private final char value[] ，用来存储字符，使用的是UTF-16编码。  
也就是说，不管是什么样的字符串，只要是存储在String对象中的，就是UTF-16编码。

Java的class文件的编码:

在class文件中，字符串都是以UTF-8的格式保存的。  
所以，只要你告诉了javac你的源文件的正确编码，那么运行的时候内存中的String常量肯定不会出错。

输入文件和输出文件的编码:

这里涉及到两个类InputStreamReader和OutputStreamWriter。  
他们都有一个带字符编码参数charsetName的构造函数。用来指定一个stream里面的字符是什么编码。  
然后进行相应的转化。不建议直接使用FileReader和FileWriter直接打开文件，这样子的话就没办法指定字符编码，会直接使用系统的默认编码。

6.对象的序列化:

序列化是将对象转换为容易传输的格式的过程。  
例如，可以序列化一个对象，然后使用 HTTP 通过 Internet 在客户端和服务器之间传输该对象。在另一端，反序列化将从该流重新构造对象。

是对象永久化的一种机制。  
确切的说应该是对象的序列化，一般程序在运行时，产生对象，这些对象随着程序的停止运行而消失，但如果我们想把某些对象（因为是对象，所以有各自不同的特性）保存下来，  
在程序终止运行后，这些对象仍然存在，可以在程序再次运行时读取这些对象的值，或者在其他程序中利用这些保存下来的对象。这种情况下就要用到对象的序列化。

对象序列化的最主要的用处就是在传递,和保存对象(object)的时候,保证对象的完整性和可传递性。譬如通过网络传输,或者把一个对象保存成一个文件的时候,要实现序列化接口。

[第十五篇 黑马程序员-网络编程](http://blog.csdn.net/ljj810875/article/details/7889032)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:31 44人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889032#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889032#report)

[网络](http://blog.csdn.net/tag/details.html?tag=%e7%bd%91%e7%bb%9c)[string](http://blog.csdn.net/tag/details.html?tag=string)[socket](http://blog.csdn.net/tag/details.html?tag=socket)[exception](http://blog.csdn.net/tag/details.html?tag=exception)[byte](http://blog.csdn.net/tag/details.html?tag=byte)[android](http://blog.csdn.net/tag/details.html?tag=android)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一:  
1.网络模型:  
OSI参考模型。  
TCP/IP参考模型。  
2.网络通讯要素:  
IP地址。  
端口号。  
传输协议。  
3.IP地址:  
网络中设备的标识，  
不易记忆，可用主机名，  
本地回环地址:127.0.0.1，主机名:localhost。  
4.端口号:  
用于标识进程的逻辑地址，不同进程的标识，  
有效端口:0-65535,其中0-1024系统使用或保留端口。  
5.传输协议：  
通讯的规则。  
常见的协议:TCP，UDP。  
6.UDP:  
将数据及源和目的封装成数据包中，不需要建立连接，  
每个数据报的大小限制在64K内，  
因无连接，是不可靠协议，  
不需要建立连接，速度快。  
7.TCP:  
建立连接，形成传输数据的通道，  
在连接中进行大数据量传输，  
必须建立连接，效率会稍低，  
是可靠协议。  
8.Socket:  
Socket就是为网络服务提供的一种机制，  
通信的两端都有Socket，  
网络通信其实就是Socket间的通信，  
数据在两个Socket间通过IO传输。

二.  
1.UDP发送端与接收端(代码演示):  
UDP发送端:  
package test;

import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.net.SocketException;  
import java.net.UnknownHostException;

public class UdpSend {  
 public static void main(String[] args) throws IOException{  
  DatagramSocket ds=new DatagramSocket(8888);  
  byte[] buf="udp ge men lai le".getBytes();  
  DatagramPacket dp=new DatagramPacket(buf,buf.length,InetAddress.getByName("10.1.39.41"),10000);  
     ds.send(dp);  
     ds.close();  
 }

}

UDP接收端:  
package test;

import java.net.DatagramPacket;  
import java.net.DatagramSocket;

public class UdpRece {  
 public static void main(String[] args) throws Exception {  
  DatagramSocket ds = new DatagramSocket(10000);  
  while (true) {  
     
   byte[] buf = new byte[1024];  
   DatagramPacket dp = new DatagramPacket(buf, buf.length);  
   ds.receive(dp);  
   String ip = dp.getAddress().getHostAddress();  
   // getData:返回数据缓冲区  
   String data = new String(dp.getData(), 0, dp.getLength());  
   int port = dp.getPort();  
   System.out.println(ip + "::" + data + "::" + port);  
  }

 }  
}

[第十六篇 黑马程序员-网络编程](http://blog.csdn.net/ljj810875/article/details/7889157)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:39 49人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889157#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889157#report)

[网络](http://blog.csdn.net/tag/details.html?tag=%e7%bd%91%e7%bb%9c)[socket](http://blog.csdn.net/tag/details.html?tag=socket)[exception](http://blog.csdn.net/tag/details.html?tag=exception)[import](http://blog.csdn.net/tag/details.html?tag=import)[string](http://blog.csdn.net/tag/details.html?tag=string)[byte](http://blog.csdn.net/tag/details.html?tag=byte)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一:  
1.TCP传输:  
tcp分客户端和服务端，  
客户端对应的对象是Socket,服务端对应的对象是ServerSocket，  
通过查阅socket对象,发现在该对象建立时，就可以去连接指定主机，  
因为tcp是面向连接的，所以在建立socket服务时，就要有服务端存在，并连接成功，形成通路后，在该通道进行数据的传输。  
2.网络文件复制(代码演示):  
客户端:  
import java.net.\*;    
import java.io.\*;    
    
public class SendFile {    
    
    /\*\*   
     \* 需求：将文件发送至服务端   
     \*    
     \* 流程：   
     \*      1、建立socket连接 ，指定服务器端口10000   
     \*      2、建立文件读取流 BufferedInputStream链接到文件   
     \*      3、建立输出流链接到 socket 的getOutputStream中   
     \*      4、读取文件 写入socket流中   
     \*      5、建立输入流链接到socket 的getInputStream中，等待成功信息   
     \*        
     \*/    
    public void send()throws Exception{    
        Socket socket =     
                new Socket (InetAddress.getByName("192.168.1.5"),10000);    
        BufferedInputStream bis =     
                new BufferedInputStream(new FileInputStream("D:\\hello.txt"));    
        OutputStream socketOut = socket.getOutputStream();    
            
        byte[] buff = new byte[1024];    
        int len ;    
        while ( (len = bis.read(buff))!=-1)    
        {    
            socketOut.write(buff);    
        }    
        System.out.println("finished");    
        socket.shutdownOutput();    
        BufferedReader socketInput =    
                new BufferedReader( new InputStreamReader(socket.getInputStream()));    
        String s = socketInput.readLine();    
        System.out.println(s);    
        bis.close();    
        socket.close();     
    }    
        
        
    public static void main(String[] args) throws Exception {    
        SendFile s = new SendFile();    
        s.send();    
    }    
    
}

服务器端:  
import java.io.\*;    
import java.net.\*;    
    
public class ReceiveFile {    
    
    /\*\*   
     \* 创建一个ServerSocket服务，监听10000端口   
     \* 创建BufferedOutputstream   
     \* 获取一个socket流，   
     \* 利用socket流中的InputStream创建一个InputStream   
     \* 写入文件   
     \* @throws Exception    
     \*/    
    public void Receive() throws Exception{    
        ServerSocket serSocket = new ServerSocket(10000);    
        Socket socket = serSocket.accept();    
            
        BufferedOutputStream fileOut =     
                new BufferedOutputStream(new FileOutputStream("E:\\hello.txt"));    
            
        InputStream inSocket =     
                socket.getInputStream();    
            
        byte[] buf = new byte[1024];    
        int len =0;    
        System.out.println("Going to While");    
        while ( (len = inSocket.read(buf))!=-1)    
        {    
            System.out.println("inside while");    
            fileOut.write(buf, 0, len);    
                
        }    
        String s = "写入成功！";    
        System.out.println(s);    
        DataOutputStream socketOut = new DataOutputStream(socket.getOutputStream());    
        socketOut.writeBytes(s);    
            
        fileOut.close();    
        socket.close();    
        serSocket.close();          
    }    
        
    public static void main(String[] args) throws Exception {    
        ReceiveFile rec = new ReceiveFile();    
        rec.Receive();    
    }    
    
}

[第十七篇 黑马程序员-Java高新技术](http://blog.csdn.net/ljj810875/article/details/7889163)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:41 72人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889163#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889163#report)

[sun](http://blog.csdn.net/tag/details.html?tag=sun)[constructor](http://blog.csdn.net/tag/details.html?tag=constructor)[string](http://blog.csdn.net/tag/details.html?tag=string)[class](http://blog.csdn.net/tag/details.html?tag=class)[object](http://blog.csdn.net/tag/details.html?tag=object)[exception](http://blog.csdn.net/tag/details.html?tag=exception)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

一:  
1.可变参数:

一个方法接受的的参数个数不固定，例如:  
System.out.println(add(2,3,5));  
System.out.println(add(1,2,3,4));

2.可变参数的特点:

只能出现在参数列表的最后;  
...位于变量类型和变量名之间,前后有无空格都可以;  
调用可变参数的方法时,编译器为该可变参数隐含创建一个数组,在方法体中用数组的形式访问可变参数;

3.for的增强循环

格式:for(type 变量名:集合变量名){...}

4.注意事项:  
迭代变量必须在()中定义  
集合变量可以是数组或实现了Iterable接口的集合类

5.享元模式(flyweight):  
如果有很多很小的对象,他们有很多相同的属性，那你就可以把他们变成一个对象，还有些不同的属性，  
把他们变成方法的参数，称之为外部状态，那些相同的属性称之为这个对象的内部状态。

枚举：  
用普通类模仿枚举：  
package cn.itcast.day1;

public class EnumTest {

 /\*\*  
  \* @param args  
  \*/  
 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  /\*\*  
   \* 这里在赋值时，只能调用类WeekDay里面的这些常量  
   \* 这里MON这个常量是一个对象，是WeekDay里面的这种类型的对象，所以  
   \* 我们可以给它赋值，赋给WeekDay这种类型的引用变量weekDay，而且只能  
   \* 赋在类WeekDay里面定义的常量，这就是枚举。  
   \* \*/  
  WeekDay weekDay=WeekDay.MON;  
  System.out.println(weekDay.nextDay());

 }

}

package cn.itcast.day1;  
/\*\*  
 \* 用普通类模仿枚举  
 \* 类WeekDay创建的实例对象是我规定的那么几个，首先，把构造方法定义成  
 \* 私有的，人家不能创建了，  
 \* \*/  
public class WeekDay {  
 private WeekDay(){}  
   
 public final static WeekDay SUN=new WeekDay();  
 public final static WeekDay MON=new WeekDay();  
   
 //此方法不要定义成静态的，这是SUN这些实实在在的对象上的方法（注意：静态修饰内容被对象所共享）。  
 public WeekDay nextDay(){  
  if(this==SUN){//假如当前对象等于SUN这个引用变量指向的对象  
   return MON;  
  }else{  
   return SUN;  
     
  }  
 }  
 //这里覆盖toString方法，打印诸如SUN这些东东  
 public String toString(){  
  return this==SUN?"SUN":"MON";  
    
 }

}  
这里的SUN等是对象类型的。这在枚举里就是一个对象，每一个枚举元素都是一个对象  
此外，还有一种实现方式：  
将SUN等这些的nextDay都用一个独立的方法去写，现在我要每个元素自己写自己的方法：  
package cn.itcast.day1;  
/\*\*  
 \* 用普通类模仿枚举  
 \* 类WeekDay创建的实例对象是我规定的那么几个，首先，把构造方法定义成  
 \* 私有的，人家不能创建了，  
 \* \*/  
public abstract class WeekDay2{  
 private WeekDay2(){}  
   
 //此类是WeekDay的子类，并且用这个子类创建了一个对象  
 public final static WeekDay2 SUN=new WeekDay2(){

  @Override  
  /\*\*  
   \* 抽象类中的方法要被使用，必须由子类复写所有的抽象方法后，  
   \* 建立子类对象调用。  
   \* \*/  
  public WeekDay2 nextDay() {  
   // TODO Auto-generated method stub  
   return MON;  
  }  
    
 };  
 public final static WeekDay2 MON=new WeekDay2(){

  @Override  
  public WeekDay2 nextDay() {  
   // TODO Auto-generated method stub  
   return SUN;  
  }};  
   
 public abstract WeekDay2 nextDay();  
   
 //此方法不要定义成静态的，这是SUN这些实实在在的对象上的方法（注意：静态修饰内容被对象所共享）。  
 /\*public WeekDay2 nextDay(){  
  if(this==SUN){//假如当前对象等于SUN这个引用变量指向的对象  
   return MON;  
  }else{  
   return SUN;  
     
  }  
 }\*/  
   
   
 //这里覆盖toString方法，打印诸如SUN这些东东  
 public String toString(){  
  return this==SUN?"SUN":"MON";  
    
 }

}

package cn.itcast.day1;

public class EnumTest2 {

 /\*\*  
  \* @param args  
  \*/  
 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  /\*\*  
   \* 这里在赋值时，只能调用类WeekDay里面的这些常量  
   \* 这里MON这个常量是一个对象，是WeekDay里面的这种类型的对象，所以  
   \* 我们可以给它赋值，赋给WeekDay这种类型的引用变量weekDay，而且只能  
   \* 赋在类WeekDay里面定义的常量，这就是枚举。  
   \* \*/  
  WeekDay2 weekDay=WeekDay2.MON;  
  System.out.println(weekDay.nextDay());

 }

}  
采用抽象方法定义nextDay就将大量的if...else语句转移成了一个个独立的类。  
枚举自动可以帮我们实现toString方法。  
枚举的基本应用：  
枚举里面的方法：  
代码演示：  
package cn.itcast.day1;

public class EnumTest3 {

 /\*\*  
  \* @param args  
  \*/  
 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  /\*\*  
   \* 这里在赋值时，只能调用类WeekDay里面的这些常量  
   \* 这里MON这个常量是一个对象，是WeekDay里面的这种类型的对象，所以  
   \* 我们可以给它赋值，赋给WeekDay这种类型的引用变量weekDay，而且只能  
   \* 赋在类WeekDay里面定义的常量，这就是枚举。  
   \* \*/  
  WeekDay3 weekDay=WeekDay3.MON;  
  System.out.println(weekDay.nextDay());  
  //这里定义变量的时候就只能给它枚举WeekDay里面定义的元素，这里FRI是一个对象  
  WeekDay weekDay2=WeekDay.FRI;  
  System.out.println(weekDay2);  
  System.out.println(weekDay2.name());  
  System.out.println(weekDay2.ordinal());  
  System.out.println(WeekDay.valueOf("SUN").toString());  
  System.out.println(WeekDay.values().length);

 }  
 //枚举WeekDay相当于一个对象，而里面的SUN等就是一些实例对象  
 public enum WeekDay{  
  SUN,MON,TUE,WED,THI,FRI,SAT  
 }

}

package cn.itcast.day1;  
/\*\*  
 \* 用普通类模仿枚举  
 \* 类WeekDay创建的实例对象是我规定的那么几个，首先，把构造方法定义成  
 \* 私有的，人家不能创建了，  
 \* \*/  
public abstract class WeekDay3{  
 private WeekDay3(){}  
   
 //此类是WeekDay的匿名子类，并且用这个子类创建了一个对象  
 public final static WeekDay3 SUN=new WeekDay3(){

  @Override  
  /\*\*  
   \* 抽象类中的方法要被使用，必须由子类复写所有的抽象方法后，  
   \* 建立子类对象调用。  
   \* \*/  
  public WeekDay3 nextDay() {  
   // TODO Auto-generated method stub  
   return MON;  
  }  
    
 };  
 public final static WeekDay3 MON=new WeekDay3(){

  @Override  
  public WeekDay3 nextDay() {  
   // TODO Auto-generated method stub  
   return SUN;  
  }};  
   
 public abstract WeekDay3 nextDay();  
   
 //此方法不要定义成静态的，这是SUN这些实实在在的对象上的方法（注意：静态修饰内容被对象所共享）。  
 /\*public WeekDay2 nextDay(){  
  if(this==SUN){//假如当前对象等于SUN这个引用变量指向的对象  
   return MON;  
  }else{  
   return SUN;  
     
  }  
 }\*/  
   
   
 //这里覆盖toString方法，打印诸如SUN这些东东  
 public String toString(){  
  return this==SUN?"SUN":"MON";  
    
 }

}  
打印结果是：  
SUN  
FRI  
FRI  
5  
SUN  
7

name（）  
ordinal（）  
表示元素在枚举里面是第几个  
valueOf（）  
把元素装换成枚举里面的元素对象。如把System.out.println(WeekDay.valueOf("SUN").toString());  
里面的“SUN”转换成枚举里面的元素SUN.即把一个字符串变成所对应的枚举元素。  
values（）：  
返回一个数组，也就是说把这个枚举里面的每个元素一个个逐一装进这个数组看里面，我们什么时候要用这个呢，当我们需要迭代这个枚举里面的所有元素，刚才学过for加强语句，我们要对枚举里面的元素逐一进行遍历，我们不能直接遍历这个枚举类，应该把得到枚举里所有元素的这个数组，再对这个数组进行遍历，这个方法就是把枚举里所有的元素转换成一个数组。  
实现带有构造方法的枚举：  
代码演示：  
package cn.itcast.day1;  
/\*  
 \* 实现带有构造方法的枚举  
\*/  
import cn.itcast.day1.EnumTest3.WeekDay;

public class EnumTest4 {

 /\*\*  
  \* @param args  
  \*/  
 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  /\*\*  
   \* 这里在赋值时，只能调用类WeekDay里面的这些常量  
   \* 这里MON这个常量是一个对象，是WeekDay里面的这种类型的对象，所以  
   \* 我们可以给它赋值，赋给WeekDay这种类型的引用变量weekDay，而且只能  
   \* 赋在类WeekDay里面定义的常量，这就是枚举。  
   \* \*/  
  WeekDay3 weekDay=WeekDay3.MON;  
  System.out.println(weekDay.nextDay());  
  //这里定义变量的时候就只能给它枚举WeekDay里面定义的元素，这里FRI是一个对象  
  WeekDay weekDay2=WeekDay.FRI;  
  System.out.println(weekDay2);  
  System.out.println(weekDay2.name());  
  System.out.println(weekDay2.ordinal());  
  System.out.println(WeekDay.valueOf("SUN").toString());  
  System.out.println(WeekDay.values().length);

 }  
 //枚举WeekDay相当于一个对象，而里面的SUN等就是一些实例对象  
 public enum WeekDay{  
  SUN,MON,TUE,WED,THI,FRI,SAT;  
  private WeekDay(){  
   System.out.println("first");  
  }  
  private WeekDay(int day){  
   System.out.println("second");  
  }  
 }

}

package cn.itcast.day1;  
/\*\*  
 \* 用普通类模仿枚举  
 \* 类WeekDay创建的实例对象是我规定的那么几个，首先，把构造方法定义成  
 \* 私有的，人家不能创建了，  
 \* \*/  
public abstract class WeekDay4{  
 private WeekDay4(){}  
   
 //此类是WeekDay的匿名子类，并且用这个子类创建了一个对象  
 public final static WeekDay4 SUN=new WeekDay4(){

  @Override  
  /\*\*  
   \* 抽象类中的方法要被使用，必须由子类复写所有的抽象方法后，  
   \* 建立子类对象调用。  
   \* \*/  
  public WeekDay4 nextDay() {  
   // TODO Auto-generated method stub  
   return MON;  
  }  
    
 };  
 public final static WeekDay4 MON=new WeekDay4(){

  @Override  
  public WeekDay4 nextDay() {  
   // TODO Auto-generated method stub  
   return SUN;  
  }};  
   
 public abstract WeekDay4 nextDay();  
   
 //此方法不要定义成静态的，这是SUN这些实实在在的对象上的方法（注意：静态修饰内容被对象所共享）。  
 /\*public WeekDay2 nextDay(){  
  if(this==SUN){//假如当前对象等于SUN这个引用变量指向的对象  
   return MON;  
  }else{  
   return SUN;  
     
  }  
 }\*/  
   
   
 //这里覆盖toString方法，打印诸如SUN这些东东  
 public String toString(){  
  return this==SUN?"SUN":"MON";  
    
 }

}

打印结果是：  
SUN  
first  
first  
first  
first  
first  
first  
first  
FRI  
FRI  
5  
SUN  
7  
注意:枚举里面的元素必须位于所有成份之前只要用到了枚举类，它里面的静态变量都会被初始化，只要用到了类，它的静态代码都会被执行，这不用怀疑，此代码里面枚举里的元素等同于静态的成员变量，他们都会被执行，他们执行的时候构造方法被调用了，所以全是first.这是调用空参数的那个构造函数的缘故，如果我们想要其调用那个有参构造函数该怎么做呢，方法就是在元素的前面加个数值，比如在SUN前面加个1，就是SUN(1),那么打印结果就是，即先调用了有参构造函数，打印了一个second：  
SUN  
second  
first  
first  
first  
first  
first  
first  
FRI  
FRI  
5  
SUN  
7  
实现带有抽象方法的枚举：  
注意：一个类的内部类可以用四个修饰符修饰，外部类只能用两个修饰符修饰，一个默认的，一个public,  
这里这些内部类和这里的成员方法和成员变量都是用public修饰的是平级的，那成员变量和成员方法有什么它就有什么。他们是兄弟关系，都是放在类上面的.  
注意：  
new Date（300）｛｝；这段代码表示new了一个Date的子类，子类的构造方法是调用父类的无参构造方法  
我希望子类的构造方法调用父类的有参构造方法，这里表示new一个Date的子类的实例对象，并且调用了父类的有参构造函数。  
枚举只有一个成员时，就可以作为一种单例的实现方式。  
为什么要使用枚举写单例呢，第一，构造方法不用再写了，默认的是私有的，不带参数的，new instance哪个代码也不用自己写了，因为那个元素自然的就new出来了，用枚举创建单例很简单的。

二.  
反射的基石-Class类  
如何得到各个字节码所对应的实例对象：  
类名.class  
对象.getClass  
Class.forName（“类名”），比如Class.forName（“java.util.Date”）  
九个预定义Class实例对象：  
有八个基本数据类型，一个类型就对应一个实例对象。  
一个void.  
数组类型的Class实例对象  
用到的方法就是Class.isArray()  
反射就是把Java类中的各种成份映射成相应的Java类  
构造方法反射的应用：  
代码演示：  
package cn.itcast.day1;

import java.lang.reflect.Constructor;

public class ReflectTest {  
 public static void main(String[] args) throws Exception{  
  String str1="abc";  
  Class cls1=str1.getClass();  
  Class cls2=String.class;  
  Class cls3=Class.forName("java.lang.String");  
    
  System.out.println(cls1==cls2);  
  System.out.println(cls2==cls3);  
  //判断是否是基本基本数据类型的方法  
  System.out.println(cls1.isPrimitive());  
  System.out.println(int.class.isPrimitive());  
  System.out.println(int.class==Integer.class);  
                //这里Integer.TYPE表示这个Integer为基本类型了  
  System.out.println(int.class==Integer.TYPE);  
  System.out.println(int[].class.isPrimitive());  
  System.out.println(int[].class.isArray());  
    
  /\*\*  
   \* 这个StringBuffer是表示选择哪个构造方法。  
   \* 当得到一个构造方法，  
   \* 这里不知道是String类里面的哪个构造方法(这是在编译时,是不知道的)，只知道你是一个  
   \* 构造方法，可以newInstance,要在运行阶段才会知道对应的构造方法  
   \* 这里一定要区分出运行时和编译时，这里运行时才执行  
   \* String.class.getConstructor(StringBuffer.class);这行代码  
   \* 所以在编译时是不知道这个构造方法属于的是哪个类  
   \* \*/  
  Constructor constructor1=String.class.getConstructor(StringBuffer.class);  
  /\*\*  
   \* 使用此 Constructor 对象表示的构造方法来创建该构造方法的声明类的新实例，  
   \* 并用指定的初始化参数初始化该实例  
   \* newInstance可以调用很多次，每调用一次就new一个新的String.  
   \* (new StringBuffer("abc")表示传一个StringBuffer的对象进去  
   \* 这里表示在用一个StringBuffer构造方法的时候在传一个对象进去  
   \*   
   \* \*/  
  String str2=(String)constructor1.newInstance(new StringBuffer("abc"));  
  System.out.println(str2.charAt(2));  
    
 }  
   
}  
打印结果是：  
true  
true  
false  
true  
false  
true  
false  
true  
c  
这里得到一个构造方法，然后new一个Instance，这里用到反射。  
Class.newInstance()方法：  
该方法内部先得到默认的构造方法，然后用该构造方法创建实例对象。  
该方法内部的具体代码是怎样呢？用到了缓存机制来保存默认构造方法的实例对象  
这里的newInstance是Class里面的，也可以创建实例对象，这里的和构造方法里面的newInstance有什么区别没有，这里是一个无参的构造方法，表示为：public T newInstance()throws InstantiationException,是一个无参的构造方法，其实不要这个方法不影响Java的开发，但为什么要提供这样的一个方法呢？是提供一个便利，我们要创建一个类的实例对象的步骤是：由class得到constructor,再new Object,这样比较麻烦，而Class.newInstance可以把中间一步省略，即该方法内部先得到默认的构造方法，然后用该构造方法创建实例对象，而构造方法里面的newInstance表示为：public T newInstance(Object... initargs)  throws InstantiationException,  
反射的弊端：比较占用资源，导致程序性能下降                                                  
注意：因为构造方法是无序的，所以我们想要得到某个类里面的一些构造方法，就必须根据构造方法的参数。  
Constructor类：  
Constructor类代表某个类中的一个构造方法。  
例子：  
得到某个类所有的构造方法：  
Constructor[] constructors=Class.forNmae("java.lang.String").getConstructors();  
得到某一个构造方法：  
例子：  
注意：获得方法时要用到类型  
Constructor constructor=Class.forName("java.lang.String").getConstructor(StringBuffer.class);  
创建实例对象：  
注意：调用获得的方法时要用到与上面相同类型的实例对象  
String str=(String)constructor.newInstance(new StringBuffer("abc"));  
成员变量的反射：  
代码演示：  
package cn.itcast.day1;

public class ReflectPoint {  
 private int x;  
 public int y;  
 public ReflectPoint(int x, int y) {  
  super();  
  this.x = x;  
  this.y = y;  
 }

}

package cn.itcast.day1;

import java.lang.reflect.Constructor;  
import java.lang.reflect.Field;

public class ReflectTest2 {  
 public static void main(String[] args) throws Exception{  
               ReflectPoint pt1=new ReflectPoint(3,5);  
  /\*\*  
   \*这里先得到字节码，因为字节码有字段信息，再得到某个字段，  
   \*类上有很多字段，即有很多成员变量，用变量名来区别  
   \*这里的fieldY的值不是5，这里的fieldY只是代表这个类  
   \*的字节码的一个变量。它没有对应到对象身上，我可以用pt1,pt2或  
   \*pt3等搞出很多个对象，每个对象身上都有一个fieldY,这个fieldY  
   \*在pt1上是5，在pt2上就不知道是多少了，就是说fieldY不代表一个  
   \*具体的值，只代表一个变量，即不代表某个变量身上具体的值.  
   \*举个例子：在Person类中，有对象张三，李四，王五，他们每个人都只有  
   \*一个苹果，假如这些苹果是编号的，张三等他们也是编号的，是不同的对象  
   \*我把这些编号的苹果给他们，我不知道第几号苹果属于他们其中哪个，  
   \*有可能张三，李四都有一个编号为1的苹果，这里的fieldY就代表这些编号的苹果，  
   \*要想得到其中一个人所持有的苹果的编号值，那么就要找到这个具体的  
   \*人，然后得到苹果查看编号  
   \*  
   \* \*/  
  Field fieldY=pt1.getClass().getField("y");  
  //取出变量上的值  
  System.out.println(fieldY.get(pt1));  
  /\*\*  
   \* getDeclaredField()方法设置私有可见，这里x就是私有的  
   \* \*/  
  Field fieldX=pt1.getClass().getDeclaredField("x");  
  //设置可以访问  
  fieldX.setAccessible(true);  
  System.out.println(fieldX.get(pt1));  
    
 }  
   
}  
打印结果是：  
true  
true  
false  
true  
false  
true  
false  
true  
c  
5  
3

Field类:  
Field类代表某个类中的一个成员变量  
按ALT+Shift+S快捷键是创建构造方法。  
成员方法的反射：  
代码演示：  
package cn.itcast.day1;  
/\*\*  
 \* 方法的反射  
 \* \*/  
import java.lang.reflect.Constructor;  
import java.lang.reflect.Field;  
import java.lang.reflect.Method;

public class ReflectTest3{  
 public static void main(String[] args) throws Exception{  
  String str1="abc";  
                /\*\*  
   \* name:方法名      
   \* int.class:一个类里面同一个名字的方法有多种重载形式，那怎样从这些方法中选取一个呢  
   \* 那就要靠方法的参数列表，参数列表就包括了参数类型和参数个数，这里的每一个参数类型都用一  
   \* 个class来表示，到底有多少个参数就在这里写多少个class进去,我们这里只有一个参数，而charAt是  
   \* int,所以这里写成int.class.  
   \* \*/  
    
  Method methodCharAt=String.class.getMethod("charAt", int.class);  
  System.out.println(methodCharAt.invoke(str1, 1));  
 }  
   
}  
打印结果是：  
b  
这里是用反射的方式获得字节码里的方法，再拿这个方法作用于这个对象，invoke方法就是调用这个方法的意思，chatAt方法执行调用的动作。即这个invoke是这个methodCharAt方法对象身上的方法。  
Method类代表某个类中的一个成员方法  
得到类中的某一个方法：  
例子：  
Method charAt=Class.forName("java.lang.String").getMethod("charAt",int.class);  
调用方法：  
通常方式：  
System.out.println(str.charAt(1));  
反射方式：  
注意：  
如果传递给Method对象的invoke()方法的第一个参数为null，说明该Method对象对应的是一个静态方法。  
System.out.println(charAt.invoke(str,1));  
jdk1.4和jdk1.5的invoke方法的区别：  
jdk1.5:public Object invoke(Object obj,Object...args)  
jdk1.4:public Object invoke(Object obj,Object[] args),即按jdk1.4的语法，需要将一个数组  
作为参数传递给invoke方法时，数组中的每个元素分别对应被调用方法中的一个参数，所以，调用chatAt方法的代码也可以用  
jdk1.4改写为charAt.invoke("str",new Object[]{1}形式。

[第十八篇 黑马程序员-Java高新技术](http://blog.csdn.net/ljj810875/article/details/7889212)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:49 29人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889212#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889212#report)

[编译器](http://blog.csdn.net/tag/details.html?tag=%e7%bc%96%e8%af%91%e5%99%a8)[java](http://blog.csdn.net/tag/details.html?tag=java)[integer](http://blog.csdn.net/tag/details.html?tag=integer)[myeclipse](http://blog.csdn.net/tag/details.html?tag=myeclipse)[float](http://blog.csdn.net/tag/details.html?tag=float)[jvm](http://blog.csdn.net/tag/details.html?tag=jvm)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

一:  
1.JavaBean:  
特殊的java类，其方法的名字符合某种规则，例如:  
比如我们要获取 age的属性，其用到的方法是int getAge();  
还可用void setAge(int age)。  
2.用MyEclipse将读取属性和设置属性的流水帐代码分别抽取成方法，  
只要调用这个方法，并给这个方法传递一个对象、属性名和设置值，它就能完成属性修改的功能。  
3.得到BeanInfo最好采用"obj.getClass()"方式，而不要采用"类名.class"方式，这样程序更通用。  
4.我们在JavaBean上操作的时候那个x属性是int,但当我们用BeanUtil去操作的时候，它的类型是String.

二:  
1.注解:  
注解相当于一个标记，加了注解就等于打上了某种标记。没加，就等于没有加标记，以后，javac编译器，开发工具和其他程序可以用反射来了解你的类及各种元素上有无何种标记，  
看你有什么标记，就去干相应的事，标记可以加在包，类,字段，方法的参数以及局部变量上。  
2.定义基本类型的属性和应用属性:  
在注解类中增加String color();  
@MyAnnotation(color="red")  
3.用反射方式获得注解对应的实例对象后，再通过该对象调用属性对应的方法  
MyAnnotation a=(MyAnnotation)AnnotationTest.class.getAnnotation(MyAnnotation.class);  
System.out.println(a.color());

三.  
1.泛型:  
泛型是提供给java编译器使用的，可以限定集合中的输入类型，让编译器挡住源程序中的非法输入，  
，编译器编译带类型说明的集合时会去除掉“类型”信息，使程序运行效率不受影响，对于参数化的泛型  
类型，getClass()方法的返回值和原始类型完全一样。由于编译生成的字节码会去掉泛型  
类型信息，只要能跳过编译器，就可以往某个泛型集合中加入其它类型的数据，例如，用反射得到集合，再调用  
其add方法即可。  
2.泛型中的?通配符  
使用?通配符可以引用其他各种参数化的类型，？通配符定义的变量主要用作引用，可以调用  
参数化无关的方法，不能调用与参数化有关的方法。  
3.泛型中的？通配符的扩展:  
限定通配符的上边界:  
正确:Vector<? extends Number> x=new Vector<Integer>();  
错误:Vector<? extends Number> x=new Vector<String>();  
限定通配符的下边界：  
正确：Vector<? super Integer> x=new Vector<Number>();  
错误：Vector<? super Integer> x=new Vector<Byte>();  
4.由C++的模版函数引入自定义泛型:  
如下函数的结构很相似，仅类型不同:  
int add(int x, int y){  
                    return x+y;  
     }  
     float add(float x,float y){  
                    return x+y;  
     }  
     double add(double x,double y){  
                     return x+y;  
     }

C++用模版函数解决，只写一个通用方法，它可以适应各种类型，示意代码如下:  
        template<class T>  
        T add(T x,T y){  
                     return(T)(x+y);  
        }  
四：  
代理：  
代理的概念与作用：

程序中的代理  
要为已存在的多个具有相同接口的目标类的各个方法增加一些系统功能，例如，异常处理、日志、计算方法的运行时间、事务管理、等等，你准备如何做？  
编写一个与目标类具有相同接口的代理类，代理类的每个方法调用目标类的相同方法，并在调用方法时加上系统功能的代码。如果采用工厂模式和配置文件的方式进行管理，则不需要修改客户端程序，在配置文件中配置是使用目标类、还是代理类，这样以后很容易切换，譬如，想要日志功能时就配置代理类，否则配置目标类，这样，增加系统功能很容易，以后运行一段时间后，又想去掉系统功能也很容易。  
动态代理技术：  
要为系统中的各种接口的类增加代理功能，那将需要太多的代理类，全部采用静态代理方式，将是一件非常麻烦的事情！写成百上千个代理类，是不是太累！  
JVM可以在运行期动态生成出类的字节码，这种动态生成的类往往被用作代理类，即动态代理类。  
JVM生成的动态类必须实现一个或多个接口，所以，JVM生成的动态类只能用作具有相同接口的目标类的代理。  
CGLIB库可以动态生成一个类的子类，一个类的子类也可以用作该类的代理，所以，如果要为一个没有实现接口的类生成动态代理类，那么可以使用CGLIB库。  
代理类的各个方法中通常除了要调用目标的相应方法和对外返回目标返回的结果外，还可以在代理方法中的如下四个位置加上系统功能代码：  
1.在调用目标方法之前  
2.在调用目标方法之后  
3.在调用目标方法前后

五：  
1.类加载器:  
java虚拟机中可以安装多个类加载器，系统默认三个主要的类加载器，每个类负责加载特定位置的类:  
BootStrap, ExtClassLoader,AppClassLoader  
类加载器也是java类,因为其他是Java类的类加载器本身也要被类加载器加载，  
显然必须有第一个类加载器不是Java类，这正是BootStrap。  
2.Java虚拟机中的所有类加载器采用具有父子关系的树形结构进行组织，在实例  
化每个类装载器对象时，需要为其指定一个父级类装载器对象或者默认采用系统  
类装载器为其父级类加载。  
3.模版方法设计模式:  
将一个抽象类的部分逻辑以具体的形式实现，然后申明一些抽象方法来迫使  
子类实现剩余的逻辑。不同子类以不同的方式实现这些抽像方法，进而对不同逻辑可以采取不同的实现方式。

[第十九篇 黑马程序员-交通灯管理系统](http://blog.csdn.net/ljj810875/article/details/7889238)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 09:58 95人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889238#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889238#report)

[交通](http://blog.csdn.net/tag/details.html?tag=%e4%ba%a4%e9%80%9a)[android](http://blog.csdn.net/tag/details.html?tag=android)[java](http://blog.csdn.net/tag/details.html?tag=java)[生活](http://blog.csdn.net/tag/details.html?tag=%e7%94%9f%e6%b4%bb)[编程](http://blog.csdn.net/tag/details.html?tag=%e7%bc%96%e7%a8%8b)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ---------

现将自己所看视频和在网上查的资料然后将自己的理解总结如下:

异步随机生成按照各个路线行驶的车辆:

由东向而来去往南向的车辆 :左转车辆  
由南向而来去往北向的车辆 :直行车辆  
由西向而来去往南向的车辆 :右转车辆  
信号灯忽略黄灯，只考虑红灯和绿灯。

应考虑左转车辆控制信号灯，右转车辆不受信号灯控制。

具体信号灯控制逻辑与现实生活中普通交通灯控制逻辑相同，不考虑特殊情况下的控制逻辑。

注：南北向车辆与东西向车辆交替放行，同方向等待车辆应先放行直行车辆而后放行左转车辆。

1.每辆车通过路口时间为1秒（提示：可通过线程Sleep的方式模拟）。

2.随机生成车辆时间间隔以及红绿灯交换时间间隔自定，可以设置。

3.不要求实现GUI，只考虑系统逻辑实现，可通过Log方式展现程序运行结果。

看完要求就需要开始分析。

总共有12条路线

为了统一编程模型，可以假设每条路线都有一个红绿灯对其进行控制，右转弯的灯可以假设称为常绿状态，另外，其他的8条线路是两两成对的，为4组，所以，程序只需要考虑途中标注了数字号的4条路线的控制灯的切换顺序，这四条路线相反方向的路线的控制灯跟随这4条路线切换，不必另外考虑。

首先想到路，12条路都有名字，路上有车。所以应有Road类，类中有name和vechicles属性。车随时上路，用缓冲池模拟车上路，然后每秒检查等是否绿，绿了就放一辆车，用缓冲池的调度。

其次是信号灯，12条线路12个灯，右转弯可以设置为常亮的不用考虑。其它8个两两相对可以设为四组。灯的设计用枚举来实现，每一个枚举元素都包含相对方向的灯opposite，下一个亮的next和灯是否为绿lighted等三个属性。light（）方法，某个灯变绿时，相对的灯也要变绿，blackout（）方法，某个灯变红时，相应的灯也变红，同时下一个灯变绿。  
然后再构建一个灯的控制类。其中有一个当前等currentLamp属性和一个灯控制方法lampController().

[第二十篇 黑马程序员-银行业务系统](http://blog.csdn.net/ljj810875/article/details/7889273)

分类： [学习日志](http://blog.csdn.net/ljj810875/article/category/1214241)2012-08-21 10:01 527人阅读 [评论](http://blog.csdn.net/ljj810875/article/details/7889273#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ljj810875/article/details/7889273#report)

[android](http://blog.csdn.net/tag/details.html?tag=android)[java](http://blog.csdn.net/tag/details.html?tag=java)[电话](http://blog.csdn.net/tag/details.html?tag=%e7%94%b5%e8%af%9d)

------- [android培训](http://edu.csdn.net/heima)、[java培训](http://edu.csdn.net/heima)、期待与您交流！ ----------

现将自己所看视频和在网上查的资料理解总结如下:

模拟实现银行业务调度系统逻辑，具体需求如下：

银行内有6个业务窗口，1 - 4号窗口为普通窗口，5号窗口为快速窗口，6号窗口为VIP窗口。  
有三种对应类型的客户：VIP客户，普通客户，快速客户（办理如交水电费、电话费之类业务的客户）。  
异步随机生成各种类型的客户，生成各类型用户的概率比例为：  
VIP客户：普通客户：快速客户  =  1 ：6 ：3。

客户办理业务所需时间有最大值和最小值，在该范围内随机设定每个VIP客户以及普通客户办理业务所需的时间，快速客户办理业务所需时间为最小值（提示：办理业务的过程可通过线程Sleep的方式模拟）。  
各类型客户在其对应窗口按顺序依次办理业务。  
当VIP（6号）窗口和快速业务（5号）窗口没有客户等待办理业务的时候，这两个窗口可以处理普通客户的业务，而一旦有对应的客户等待办理业务的时候，则优先处理对应客户的业务。  
随机生成客户时间间隔以及业务办理时间最大值和最小值自定，可以设置。  
不要求实现GUI，只考虑系统逻辑实现，可通过Log方式展现程序运行结果。  
面向对象的分析和设计：  
有三种对应类型的客户：VIP客户，普通客户，快速客户，异步随机生成各种类型的客户，各类型客户在其对应窗口按顺序依次办理业务。  
首先，经常在银行办理业务的人更有利于理解本系统，每一个客户其实就是由银行的一个取号机器产生号码的方式来表示的。所以，我想到要有一个号码管理器对象，让这个对象不断地产生号码，就等于随机生成了客户。  
由于有三类客户，每类客户的号码编排都是完全独立的，所以，我想到本系统一共要产生三个号码管理器对象，各自管理一类用户的排队号码。这三个号码管理器对象统一由一个号码机器进行管理，这个号码机器在整个系统中始终只能有一个，所以，它要被设计成单例。  
各类型客户在其对应窗口按顺序依次办理业务 ，准确地说，应该是窗口依次叫号。  
各个窗口怎么知道该叫哪一个号了呢？它一定是问的相应的号码管理器，即服务窗口每次找号码管理器获取当前要被服务的号码。