Java语言支持四种类型，接口，类，数组，和基本类型，前三种类型称为引用类型，类实例和数组是对象，而基本类型的值则不是对象,字符串属于对象而非基本数据类型，所以”==”来判断两个字符串是否相等，使用”==”判断的是两个字符串的内存地址，而要判断字符串内容是否相等，就需要通过equal()方法来判断。

基本类型：

基本类型自然不用说了，它的值就是一个数字，一个字符或一个布尔值。

引用类型：

是一个对象类型，值是什么呢？它的值是指向内存空间的引用，就是地址，所指向的内存中保存着变量所表示的一个值或一组值。

引用传递 和 值传递

引用类型：除了在函数传值的时候是"引用传递"，在任何用"＝"向对象变量赋值的时候都是"引用传递"。

值传递：基本类型的传递都属于值传递，和C语言一样，当把Java的基本数据类型（如 int，char，double等）作为入口参数传给函数体的时候，传入的参数在函数体内部变成了局部变量，这个局部变量是输入参数的一个拷贝，所有的函 数体内部的操作都是针对这个拷贝的操作，函数执行结束后，这个局部变量也就完成了它的使命，它影响不到作为输入参数的变量。这种方式的参数传递被称为"值 传递"。

基本类型转换：

**所有整数运算（byte,char，short）都会默认自动转化为int类型再进行运算**，但在整数运算中有long型，则都转化为long类型；**所有浮点型运算（float，double）都自动转换为double类型再进行运算**

byte b1=1,b2=2;

b3=b1+b2;

final byte b4=4,b5=5;

b6=b4+b5;

问b3和b5能否正常赋值？

答案应该是：b3不能正常赋值，需要类型转换，b3=（byte）（b1+b2）；b6能够正常赋值。

原因分析:

在**[Java](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "http://blog.csdn.net/lyric_315/article/details/_blank)**中，在基本类型进行算术运算的时候，会发生小字节类型向大字节类型转换的现象。**对于short，byte,char 比int 字节数小的变量类型来说，运算结果会自动转换为int类型**。Java编译器会在编译期或者运行期将byte和short类型的数据带符号扩展为相应的int类型数据，将boolean和char类型数据零位扩展为相应的int类型数据。因此，在处理boolean 、byte、short 和 char 类型的数组是，也会用相应的int类型的字节码指令来处理。因此，大多数对于上述类型数据的操作，实际上都是使用相应的 int 类型作为运算类型。如下：

short a=1, b=2;  
short c=a+b;  
System.out.println(c);

在编译时，会报出“Exception in thread "main" java.lang.Error: Unresolved compilation problem: Type mismatch: cannot convert from int to short”，可以看出可以看出a+b结果变成了int类型。

**重点来了，如果是final 修饰的变量，进行运算的时候则不会出现类型转换异常。**

final short a=1, b=2;  
short c=a+b;  
System.out.println(c);

这么操作完全OK，究其原因，对于final 修饰的基本类型的变量来说，他们之间的运算直接就被硬编码成了直接赋值语句，连中间结果都没有了，类型转换的异常也就没了。

当两个操作数中没有long类型时，两个操作数中非int类型会先自动转换为int类型，再参与运算，返回结果为int；

当两个操作数中含有long类型时，两个操作数中非long类型会自动转换为long类型，再参与运算，返回结果为long

**int** i1=2;

**long** l1=3;

**int** i2=i1+l1;（error：cannot convert from long to int）等号左边的类型必须比右边 的类型高

**long** l2=i1+l1;

**byte** b1=1;

**long** l3=l1+b1;（l3为long型）

。

**高类型转化为低类型只能进行强制转化，可能会损失精度，丢失数据**

**double** d1 = l1;（long类型直接转化为double型，等于3.0）

**double** d2 = 2.13;

//long l4 = (long) d2;（强制类型转换，l4=2）

**long** l = **new** Double(d2).longValue();（**\*\*\*重要\*\*\***用Double封装下基本类型double， 再调用longValue方法就好了，返回 long类型 ，转化为int类型也可以， 调用intValue()方法）  
**1如何将字串 String 转换成整数 int（用对应的封装类）?**  
  
A. 有两个方法:  
  
1). int i = Integer.parseInt([String]);   
2). int i = Integer.valueOf(my\_str).intValue();  
  
注: 字串转成 double, float, long 的方法大同小异.

**2 如何将整数 int 转换成字串 String（用对应的封装类） ?**

A. 有叁种方法:  
  
1.) String s = String.valueOf(i);  
  
2.) String s = Integer.toString(i);  
  
3.) String s = "" + i;  
  
注: double, float, long 转成字串的方法大同小异.

方法：s=i+"";   //会产生两个String对象  
方法：s=String.valueOf(i); //直接使用String类的静态方法，只产生一个对象

方法：i=Integer.parseInt(s);//直接使用静态方法，不会产生多余的对象，但会抛出异常  
方法：i=Integer.valueOf(s).intValue();//Integer.valueOf(s) 相当于 new Integer(Integer.parseInt(s))，也会抛异常，但会多产生一个对象

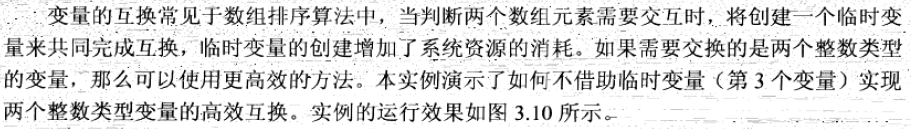
**基本数据类型和对应的封装类**

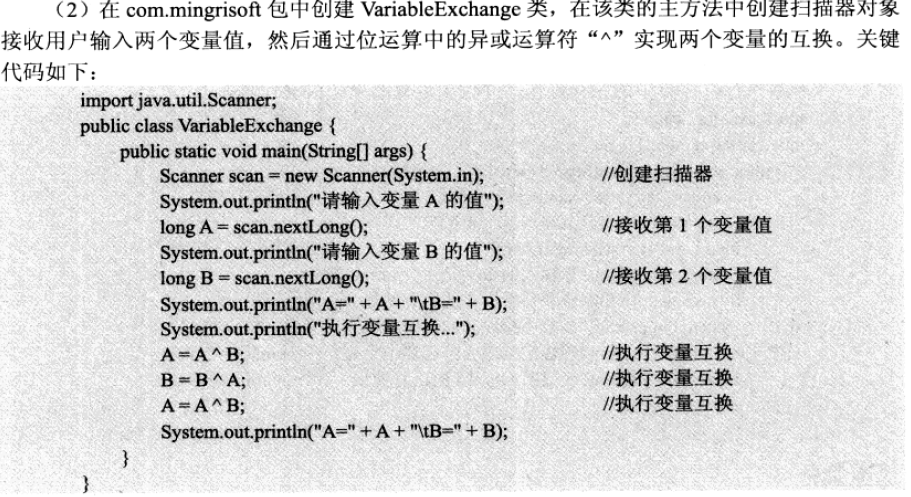
数据类型 默认值 封装类(都在[java.lang](https://www.baidu.com/s?wd=java.lang&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvPWRsnyfvuWT3ujDLmhnL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHfvrHR1rH0v" \t "http://zhidao.baidu.com/_blank)包下)  
byte (byte)0 Byte  
short (short)0 Short  
int 0 Integer  
long 0L Long  
float 0.0f Float  
double 0.0d Double  
char \u0000 Character  
boolean false Boolean

当要求精度很高，或进行超大的浮点数运算时，java的基本数据类型无法保证浮点数的精度（即使用double）也无法对超出其表示范围的数字进行运算，这时候需要用到BgDecimal类来实现。

**实现两个变量的互换（不借助第三个变量）**

关键是理解**异或运算（性质：A^B^A=B）**



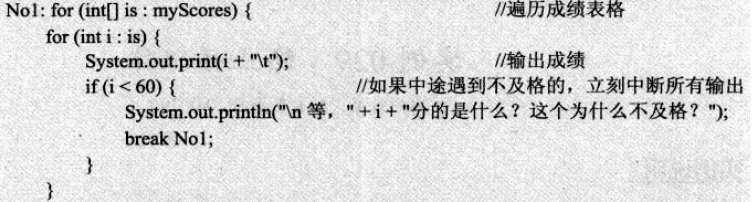


**实现乘法运算的三个方法：**

1. 通过“\*”运算符；
2. 用BigDecimal类的multiply()方法实现
3. 左移运算，左移运算n位，就等于乘以2的n次方 2<<3=2^3=8

**退出两层for循环：**

一般来说用break，程序会退出当前的这一层循环，但是，如果需要在遇到break时退出多层循环，可以在最外层的循环上贴上标签。强制退出多层循环。例如，要退出双层的for循环：

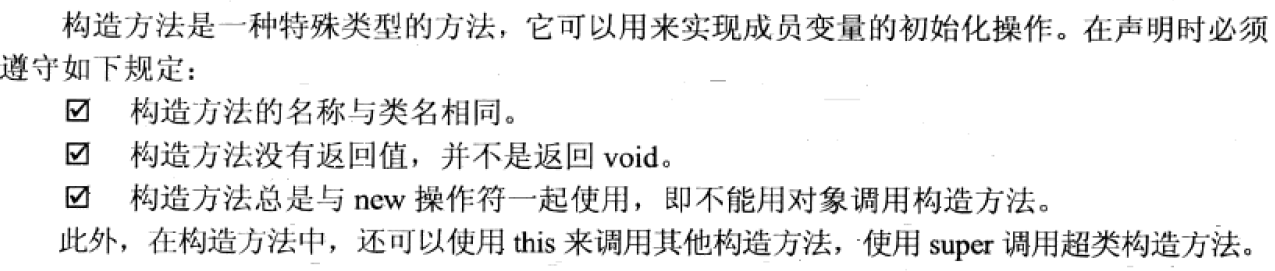


其中，No1就是一个标签。

变量的初始化

一般来说，变量在使用之前需要对其进行初始化，但Java中类的成员变量可以不需要，虚拟机会自动为其进行初始化，对于引用类型的变量，在使用之前一定要进行初始化，不然会抛出NullPointerException的异常，而对于局部变量，在使用之前一定要对其进行初始化，虚拟机不会对其进行初始化工作，否则会报错。

构造方法也可以重载，无参构造和有参构造。



重写equal()方法

如果你重写了equals，比如说是基于对象的内容实现的，而保留hashCode的实现不变，那么很可能某两个对象明明是“相等”，而hashCode却不一样。这样，当你用其中的一个作为键保存到hashMap、hasoTable或hashSet中，再以“相等的”找另一个作为键值去查找他们的时候，则根本找不到。

我们都知道Java语言是完全面向对象的，在java中，所有的对象都是继承于Object类。Ojbect类中有两个方法equals、hashCode，这两个方法都是用来比较两个对象是否相等的。

在未重写equals方法我们是继承了object的equals方法，那里的 equals是比较两个对象的内存地址，显然我们new了2个对象内存地址肯定不一样

对于值对象，==比较的是两个对象的值

对于引用对象，比较的是两个对象的地址

**equals()与‘==’的区别**

默认的equals方法同==，一般来说我们的对象都是引用对象，要重写equals方法。

总结：默认情况下也就是从超类Object继承而来的equals方法与‘==’是完全等价的，比较的都是对象的内存地址，但我们可以重写equals方法，使其按照我们的需求的方式进行比较，如String类重写了equals方法，使其比较的是字符的序列，而不再是内存地址。

hashCode方法也是可以用来比较两个对象是否相等的。但是我们很少使用，应该说是很少直接使用。hashCode方法返回的是一个int值，可以看做是一个对象的唯一编码，如果两个对象的hashCode值相同，我们应该认为这两个对象是同一个对象。一般如果使用java中的Map对象进行存储时，他会自动调用hashCode方法来比较两个对象是否相等。

所以如果我们对equals方法进行了重写，建议一定要对hashCode方法重写，以保证相同的对象返回相同的hash值，不同的对象返回不同的hash值。

1、重写equals方法时需要重写hashCode方法，主要是针对Map、Set等集合类型的使用；

a: Map、Set等集合类型存放的对象必须是唯一的；

b: 集合类判断两个对象是否相等，是先判断equals是否相等，如果equals返回TRUE，还要再判断HashCode返回值是否ture,只有两者都返回ture,才认为该两个对象是相等的。

2、由于Object的hashCode返回的是对象的hash值，所以即使equals返回TRUE，集合也可能判定两个对象不等，所以必须重写hashCode方法，以保证当equals返回TRUE时，hashCode也返回Ture，这样才能使得集合中存放的对象唯一。重写hashCode方法使得相同的对象保存在相同的地址上。

**编写equals方法的建议**

1）显式参数命名为otherObject,稍后需要将它转换成另一个叫做other的变量（参数名命名，强制转换请参考建议5）

2）检测this与otherObject是否引用同一个对象 ：if(this == otherObject) return true;（存储地址相同，肯定是同个对象，直接返回true）

3) 检测otherObject是否为null ，如果为null,返回false.if(otherObject == null) return false;

4) 比较this与otherObject是否属于同一个类 （视需求而选择）

如果equals的语义在每个子类中有所改变，就使用getClass检测 ： if(getClass()!=otherObject.getClass()) return false; (参考前面分析的第6点)

如果所有的子类都拥有统一的语义，就使用instanceof检测 ：

if(!(otherObject instanceof ClassName)) return false;（即前面我们所分析的父类car与子类 bigCar混合比，我们统一了批次相同即相等）

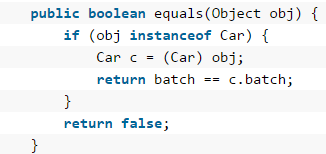
5) 将otherObject转换为相应的类类型变量：ClassName other = (ClassName) otherObject;

6) 现在开始对所有需要比较的域进行比较 。使用==比较基本类型域，使用equals比较对象域。如果所有的域都匹配，就返回true，否则就返回flase。

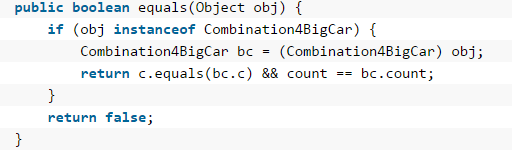
如果在子类中重新定义equals，就要在其中包含调用super.equals(other)

当此方法被重写时，通常有必要重写 hashCode 方法，以维护 hashCode 方法的常规协定，该协定声明 相等对象必须具有相等的哈希码

相同类的不同对象的比较equal重写：



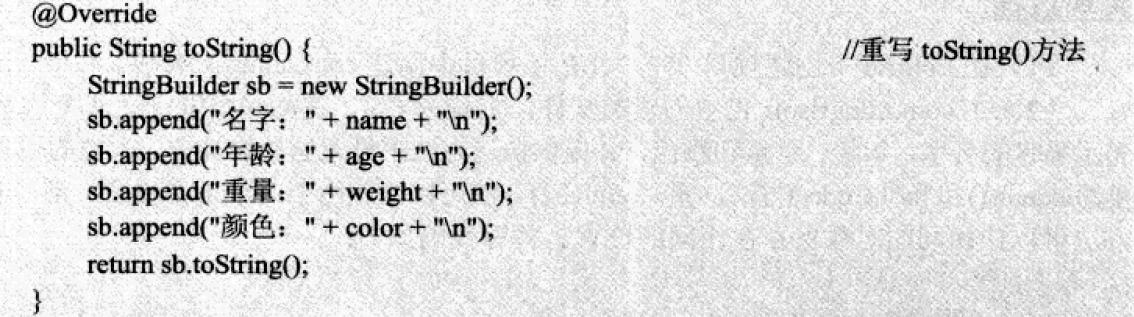
子类继承父类，重写equal方法



<http://blog.csdn.net/javazejian/article/details/51348320>

重写toString()方法

Object类中的toString 方法通常默认会返回一个“以文本方式表示”此对象的字符串。结果应是一个简明但易于读懂的信息表达式。建议所有子类都重写此方法。Object 类的 toString 方法返回一个字符串，该字符串由类名（对象是该类的一个实例）、at 标记符“@”和此对象哈希码的无符号十六进制表示组成。假如没有重写toString()方法，那我们所得到的一个对象，是完全不知道里面拥有什么样的值，只有电脑才知道，大大降低了可读性。我们是希望能按照自己的要求打印出对象的内容，这时，就需要重写toString()方法，因为重写了toString()之后，那么对象在调用toString()方法的时候，会优先调用自己类里的toString()方法。



对象的clone：浅克隆和深克隆

<http://www.2cto.com/kf/201401/273852.html>

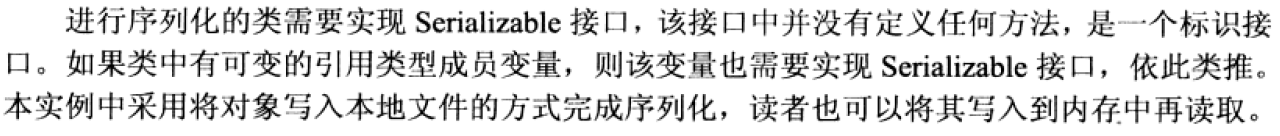
Java中三种变量的拷贝方法比较: 值变量,对象变量,字符串变量

**基本类型**的变量来说,通过简单赋值就可以实现值变量的拷贝.

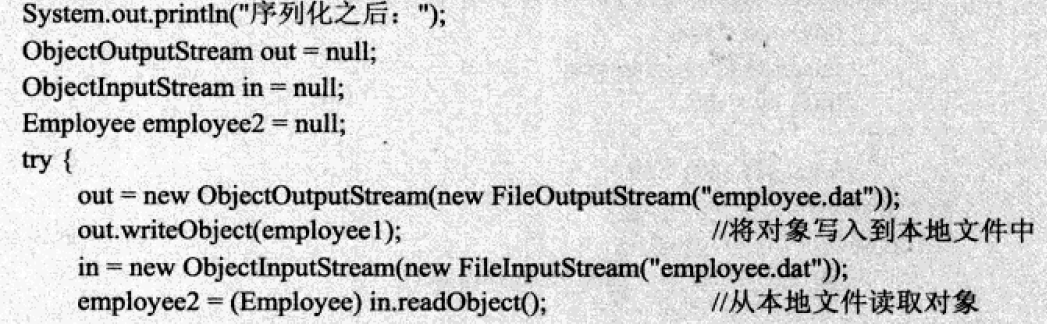
**对象类型** 变量的复制,不能和简单类型变量一般,而必须引用Object.Clone()方法,否则赋予的 只是对象的引用而已. 而直接或者间接的使用Object.Clone()方法,来实现Clone的对 象也有"浅层克隆"和"深层克隆"的说法

**String类型**也是继承自Object,但对于字符串来说,情况显得有点特殊.对于String类型的赋值 是创建一个新的String对象,而不仅仅是将对象 引用赋值给对方.从这个意义上讲, 对String的赋值更类似于基本类型.

克隆对象除了实现cloneable接口然后对象调用clone方法之外，还可以进行序列化的方式进行对象的克隆，特别是当类的成员变量非常复杂，使用了多个可变引用类型的时候，使用clone进行深克隆是很复杂的，这时就可以考虑序列化，但是当需要深克隆很多个对象（eg。10000个对象）时，clone方法的效率就比序列化的高得多。



写入本地文件：



写入内存：

