**对象的拷贝**

　　关于clone[对象拷贝]——在实际编程过程，有时候我们会遇到一种情况：当你有一个对象A，**在某一个时刻，A已经保存了对应的属性值，而且这些值本身是有效的，这个时候可能需要一个和A完全相同的对象B**，并且当B里面的属性值发生变化的时候，A中的属性值不受影响，可以理解为A和B独立，但是B的初始化不是按照我们平时创建该对象的时候的初始化操作，B的初始化数据完全来自A。

对Java存储模型了解的人都明白，**在Java里面如果针对两个对象引用采取赋值操作的时候，仅仅是让两个引用指向了同一对象**（也即值传递，非引用调用），如果其中一个引用里面的对象属性改变的时候会影响另外一个对象属性跟着改变，所以Java语言本身的对象赋值语句是不能完成上边的需求的。在这种时候，就需要用到Object类里面的通用方法clone()，这里需要说明的是：

* **通过clone()方法创建的对象是一个新对象，它可以认为是源对象的一个拷贝，但是在内存堆中，JVM会为这个拷贝分配新的对象存储空间来存放该对象的所有状态**
* 该拷贝和普通对象使用new操作符创建的对象**唯一的区别**在于初始值，这个拷贝的初始值不是对象里面成员的默认值，而是**和源对象此刻状态的成员的值是一样的**

　　下边这段代码是clone方法的运用：  
public class Testing {  
    public static void main(String args[]){  
        AClass class1 = new AClass();  
        class1.a = 12;  
        AClass class2 = (AClass)class1.clone();  
        System.out.println(class2.a);  
        System.out.println(class1==class2);

    }  
}

class AClass implements Cloneable{  
    public int a = 0;  
    public Object clone(){  
        AClass o = null;  
        try{  
            o = (AClass)super.clone();  
        }catch(CloneNotSupportedException ex){  
            ex.printStackTrace();  
        }  
        return o;  
    }  
}  
　　上边这段代码运行结果输出为：  
12  
false  
　　可以知道的就是成功复制了一个AClass的对象，该对象的引用为class1，而拷贝对象的引用为class2，这两个引用通过==比较输出为false，证明这两个引用不是指向了同一个对象，而且拷贝对象里面的a的值和class1引用的对象里面的a的值是一样的，都是12，这样就成功完成了对象的拷贝过程。若你对上边这段代码还有不了解的地方可以尝试将下边的代码修改掉：  
AClass class2 = (AClass)class1.clone() 修改为：AClass class2 = class1；  
　　改了过后，输出结果应该为：  
12  
true

　　**所以在对象的clone过程，需要注意的几点有：**　　[1]希望能够提供对象clone功能的类必须实现Cloneable接口，这个接口位于java.lang包里面  
　　[2]希望提供对象clone功能的类必须重载clone()方法，在重载过程可以看到这句话：super.clone()；也就是说，不论clone类的继承结构如何，我们在对象拷贝的时候都直接或简介调用了Object的clone()方法。而且细心留意可以看到Object的clone方法是protected域的，也就是说这个方法只有Object的子类可以调用，而在重载的时候将clone方法修饰符改为public  
　　**[3]还有一点很重要就是Object源代码里面的clone()方法是native方法，一般而言，对JVM来说，native方法的效率远比java中的普通方法高，这就是为什么我们在复制一个对象的时候使用Object的clone()方法，而不是使用new的方式。**  
　　**[4]Cloneable接口和我们在编写IO程序的时候序列化接口一样，只是一个标志，这个接口是不包含任何方法的，这个标志主要是为了检测Object类中的clone方法**，若我们定义的类想要实现拷贝功能，但是没有实现该接口而调用Object的clone方法，那么就会出现语句中catch块里面的异常错误，抛出CloneNotSupportedException。

**浅拷贝**

在对象clone的过程中，浅拷贝又称为“影子clone”，先看一段代码：  
//这里先定义一个类  
class AClass{  
    public int a;  
    public AClass(int a){ this.a = a;} //构造函数  
    public void change(){ a += 12;}  
    public String toString(){ return "A Value is " + this.a;}  
}  
//定义一个clone类，里面包含了AClass的对象引用  
class BClass implements Cloneable{  
    public int a = 12;  
    public AClass obj = new AClass(11);

//重写clone方法时必须super.clone()调用Object类的clone方法，而且该方法是native方法   
    public Object clone(){  
        BClass object = null;  
        try{  
            object = (BClass)super.clone();  
        }catch(CloneNotSupportedException ex){  
            ex.printStackTrace();  
        }  
        return object;  
    }

}

public class TestClone {  
    public static void main(String args[]){  
        BClass class1 = new BClass();  
        class1.a = 15;

        System.out.println(class1.a);  
        System.out.println(class1.obj);  
        BClass class2 = (BClass)class1.clone();  
        class2.a = 22;  
        class2.obj.change();  
        System.out.println(class1.a);  
        System.out.println(class1.obj);  
        System.out.println(class2.a);  
        System.out.println(class2.obj);  
    }  
}  
　　运行上边这段代码会有以下输出：  
15  
A Value is 11  
15                    //这里拷贝成功了  
A Value is 23     //!!!不对，根本没有调用class1里面的obj的change方法，所以不应该修改class1里面的obj里面的变量a的值【初衷】  
22  
A Value is 23  
　　不知细心的读者有没有发现输出和我们预期的拷贝不一样，虽然class2引用的对象是从class1拷贝过来的，class2里面的引用obj和class1里面的引用obj实际上还是指向了同一个对象，其含义在于，拷贝的初衷是要复制一个一模一样的对象，包括对象里面的对象也应该实现的是复制操作，它最终的目的是保证class1和class2本身的属性以及class1和class2里面的对象引用的属性在拷贝过后的各种相关操作里面相互独立，上边输出证明了class1和class2里面的变量a确实已经拷贝成功，但是class1和class2里面的AClass对象的引用obj在拷贝过后还是指向了同一个对象，所以拷贝结束过后，调用class2的obj的change方法的时候，也修改了class1里面的obj指向的对象里面的值。所以在Java里面我们把上边的拷贝过程称为“浅拷贝”，同样又称为“影子clone”。  
　　从这里可以知道，**在JVM的对象复制里面，实际上基本数据类型可以直接通过这种方式来进行拷贝工作，而非原始类型这样操作了过后拷贝的对象仅仅拷贝了对象里面的基本数据类型的成员变量**，而比较复杂的类型的成员变量并没有像预期一样产生拷贝效果，这种拷贝我们就称之为“浅拷贝”。

**深拷贝**

    如果要实现我们预期的对象拷贝效果，就需要使用深拷贝操作，其实在浅拷贝基础上实现深拷贝有两个步骤，以上边的代码为例：  
　　[1]第一步：让AClass实现同样的clone功能  
　　[2]第二步：在BClass的clone操作中多写入一句话：object.obj = (AClass)obj.clone();  
　　修改过后的代码，在这里就不写了，请读者自己去尝试深拷贝的改动，至于String和StringBuffer有关拷贝的特性，我将留到Java复合数据类型的讲解里面来详解。

**关键字this、super**  
　　在Java语言的类定义中，有两个内置对象，一个是this，一个是super，这两个对象的含义如下：  
　　this是一个指向对象自己的引用，该引用指向的对象为该类实例化的对象本身；  
　　super是一个指向该类父类的引用，如果该类不继承于任何类，那么该类调用super引用的就是Object类  
　　这里用一小段代码说明：  
class AClass{  
    public AClass(int a){     
        System.out.println("A Init "+ a);     
    }

    public void fun(int a){     
        System.out.println("fun int "+a);     
    }

}  
class BClass extends AClass{  
    private int a = 24;

    private static int b = 11;

    public BClass(){  
        this(12);

        //this.b = 12;

        super.fun(12);

    }  
    public BClass(int a)

    {  
        super(12);

        System.out.println("B init " + this.a);

    }

}  
　　针对以上代码段做几个详细的说明：  
　　super.fun(12)：在子类中访问父类的成员的时候使用super关键字，但是super不能访问父类的private成员；  
　　super(12)：子类如果要调用父类的构造函数，可以直接使用super(param)这中调用方式，但是如果是调用父类的构造方法，是不能够在子类的其他函数块里面进行的，只能在构造函数中进行，而且一般不能和this(param)同时出现，关于它们在构造函数中的使用，将在小节7中介绍；  
　　//this.b=12：虽然这句话注释掉了，但是这句话是可以通过JVM编译的，这里需要区分的是在static块中调用非static成员和在普通块中调用static成员，在普通语句中调用static成员是可以使用this.VAR的方式进行操作的，但是一般不提倡这样做，因为一般调用static成员在编程的时候都是使用独一无二的方式Class.VAR，而不使用this.VAR。  
　　this(12)/this.a：this在调用自己的成员的构造函数的时候，直接使用this(param)方式调用构造函数，如果是调用普通的成员就使用后者。  
　　\*：this和super都不能放在static块中运行，不论是static的静态初始化块，还是static的静态方法块，里面都不能出现this和super的关键字，这个道理很简单：**static块是Class域的，就是说static块里面的内容是在JVM的类加载器第一次加载类的时候就被初始化了，而且整个过程只会初始化一次。而this和super都是Object域的操作，this指代的是实例本身，而不是类**，同样的super指代的是父类的实例本身，也不是类，所以this和super是不能够出现在static的静态方法块里面的。其实从概念上讲，static和this、super操作的对象不一样，static是属于Class的，而super和this是属于Object的，所以编程过程必须注意这点。