关于java对象复制 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/645/2021_2022__E5_85_B3_E4_BA_8Ejava_c104_645091.htm 我们在编码过程经常会碰到将一个对象传递给另一个对象，java中对于基本型变量采用的是值传递，而对于对象比如bean传递时采用的是应用传递也就是地址传递，而很多时候对于对象传递我们也希望能够象值传递一样，使得传递之前和之后有不同的内存地址，在这种情况下我们一般采用以下两种情况。 1 对象克隆 什么是"clone"？ 在实际编程过程中，我们常常要遇到这种情况：有一个对象A，在某一时刻A中已经包含了一些有效值，此时可能会需要一个和A完全相同新对象B，并且此后对B任何改动都不会影响到A中的值，也就是说，A与B是两个独立的对象，但B的初始值是由A对象确定的。在Java语言中，用简单的赋值语句是不能满足这种需求的。要满足这种需求虽然有很多途径，但实现clone（）方法是其中最简单，也是最高效的手段。 Java的所有类都默认继承java.lang.Object类，在java.lang.Object类中有一个方法clone()。JDK API的说明文档解释这个方法将返回Object对象的一个拷贝。要说明的有两点：一是拷贝对象返回的是一个新对象，而不是一个引用。二是拷贝对象与用new操作符返回的新对象的区别就是这个拷贝已经包含了一些原来对象的信息，而不是对象的初始信息。 怎样应用clone()方法？ 一个很典型的调用clone()代码如下： class CloneClass implements Cloneable{ public int aInt. public Object clone(){ CloneClass o = null. try{ o = (CloneClass)super.clone(). }catch(CloneNotSupportedException

e){ e.printStackTrace(). } return o. } } 有三个值得注意的地方，一是希望能实现clone功能的CloneClass类实现了Cloneable接口，这个接口属于java.lang包，java.lang包已经被缺省的导入类中，所以不需要写成java.lang.Cloneable。另一个值得请注意的是重载了clone()方法。最后在clone()方法中调用了super.clone()，这也意味着无论clone类的继承结构是什么样的，super.clone()直接或间接调用了java.lang.Object类的clone()方法。下面再详细的解释一下这几点。 应该说第三点是最重要的，仔细观察一下Object类的clone()一个native方法，native方法的效率一般来说都是远高于java中的非native方法。这也解释了为什么要用Object中clone()方法而不是先new一个类，然后把原始对象中的信息赋到新对象中，虽然这也实现了clone功能。对于第二点，也要观察Object类中的clone()还是一个protected属性的方法。这也意味着如果要应用clone()方法，必须继承Object类，在Java中所有的类是缺省继承Object类的，也就不用关心这点了。然后重载clone()方法。还有一点要考虑的是为了让其它类能调用这个clone类的clone()方法，重载之后要把clone()方法的属性设置为public。 那么clone类为什么还要实现Cloneable接口呢？稍微注意一下，Cloneable接口是不包含任何方法的！其实这个接口仅仅是一个标志，而且这个标志也仅仅是针对Object类中clone()方法的，如果clone类没有实现Cloneable接口，并调用了Object的clone()方法（也就是调用了super.Clone()方法），那么Object的clone()方法就会抛出CloneNotSupportedException异常。 什么是影子clone？下面的例子包含三个类UnCloneA，CloneB，CloneMain。CloneB类包含了一个UnCloneA的实例和一个int类型变量，

并且重载clone()方法。CloneMain类初始化UnCloneA类的一个实例b1，然后调用clone()方法生成了一个b1的拷贝b2。最后考察一下b1和b2的输出： package clone. class UnCloneA { private int i. public UnCloneA(int ii) { i = ii. } public void doubleValue() { i *= 2. } public String toString() { return Integer.toString(i). } } class CloneB implements Cloneable{ public int aInt. public UnCloneA unCA = new UnCloneA(111). public Object clone(){ CloneB o = null. try{ o = (CloneB)super.clone(). }catch(CloneNotSupportedException e){ e.printStackTrace(). } return o. } } public class CloneMain { public static void main(String[] a){ CloneB b1 = new CloneB(). b1.aInt = 11. System.out.println("before clone,b1.aInt = " b1.aInt). System.out.println("before clone,b1.unCA = " b1.unCA). CloneB b2 = (CloneB)b1.clone(). b2.aInt = 22. b2.unCA.doubleValue(). System.out.println("============================== ==="). System.out.println("after clone,b1.aInt = " b1.aInt). System.out.println("after clone,b1.unCA = " b1.unCA). System.out.println("============================== ==="). System.out.println("after clone,b2.aInt = " b2.aInt). System.out.println("after clone,b2.unCA = " b2.unCA). } } /** RUN RESULT: before clone,b1.aInt = 11 before clone,b1.unCA = 111 ================================== after clone,b1.aInt = 11 after clone,b1.unCA = 222 ================================== after clone,b2.aInt = 22 after clone,b2.unCA = 222 */ 输出的结果说明int类型的变量aInt和UnCloneA的实例对象unCA的clone结果

不一致，int类型是真正的被clone了，因为改变了b2中的aInt变量，对b1的aInt没有产生影响，也就是说，b2.aInt与b1.aInt已经占据了不同的内存空间，b2.aInt是b1.aInt的一个真正拷贝。相反，对b2.unCA的改变同时改变了b1.unCA，很明显，b2.unCA和b1.unCA是仅仅指向同一个对象的不同引用！从中可以看出，调用Object类中clone()方法产生的效果是：先在内存中开辟一块和原始对象一样的空间，然后原样拷贝原始对象中的内容。对基本数据类型，这样的操作是没有问题的，但对非基本类型变量，我们知道它们保存的仅仅是对象的引用，这也导致clone后的非基本类型变量和原始对象中相应的变量指向的是同一个对象。 大多时候，这种clone的结果往往不是我们所希望的结果，这种clone也被称为"影子clone"。要想让b2.unCA指向与b2.unCA不同的对象，而且b2.unCA中还要包含b1.unCA中的信息作为初始信息，就要实现深度clone。 怎么进行深度clone？ 把上面的例子改成深度clone很简单，需要两个改变：一是让UnCloneA类也实现和CloneB类一样的clone功能（实现Cloneable接口，重载clone()方法）。二是在CloneB的clone()方法中加入一句o.unCA = (UnCloneA)unCA.clone(). 程序如下： package clone.ext. class UnCloneA implements Cloneable{ private int i. public UnCloneA(int ii) { i = ii. } public void doubleValue() { i *= 2. } public String toString() { return Integer.toString(i). } public Object clone(){ UnCloneA o = null. try{ o = (UnCloneA)super.clone(). }catch(CloneNotSupportedException e){ e.printStackTrace(). } return o. } } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com