**1．浅复制与深复制概念**   
  
⑴浅复制（浅克隆）   
  
    被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。换言之，浅复制仅仅复制所考虑的对象，而不复制它所引用的对象。   
  
⑵深复制（深克隆）   
  
    被复制对象的所有变量都含有与原来的对象相同的值，除去那些引用其他对象的变量。那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。换言之，深复制把要复制的对象所引用的对象都复制了一遍。   
  
**2．Java的clone()方法**   
  
⑴clone方法将对象复制了一份并返回给调用者。一般而言，clone（）方法满足：   
①对任何的对象x，都有x.clone() !=x//克隆对象与原对象不是同一个对象   
②对任何的对象x，都有x.clone().getClass()= =x.getClass()//克隆对象与原对象的类型一样   
③如果对象x的equals()方法定义恰当，那么x.clone().equals(x)应该成立。   
  
⑵Java中对象的克隆   
  
①为了获取对象的一份拷贝，我们可以利用Object类的clone()方法。   
②在派生类中覆盖基类的clone()方法，并声明为public。   
③在派生类的clone()方法中，调用super.clone()。   
④在派生类中实现Cloneable接口。   
  
请看如下代码：

[?](http://www.jb51.net/article/62909.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | public class Student implements Cloneable  {    String name;   int age;    Student(String name,int age)    {    this.name=name;    this.age=age;    }   public Object clone()    {     Object o=null;    try     {     o=(Student)super.clone();//Object 中的clone()识别出你要复制的是哪一个对象。     }    catch(CloneNotSupportedException e)     {      System.out.println(e.toString());     }    return o;    }     public static void main(String[] args)    {    Student s1=new Student("zhangsan",18);    Student s2=(Student)s1.clone();    s2.name="lisi";    s2.age=20;    //修改学生2后，不影响学生1的值。    System.out.println("name="+s1.name+","+"age="+s1.age);    System.out.println("name="+s2.name+","+"age="+s2.age);   }  } |

说明：   
①为什么我们在派生类中覆盖Object的clone()方法时，一定要调用super.clone()呢？在运行时刻，Object中的clone()识别出你要复制的是哪一个对象，然后为此对象分配空间，并进行对象的复制，将原始对象的内容一一复制到新对象的存储空间中。   
②继承自java.lang.Object类的clone()方法是浅复制。以下代码可以证明之。

[?](http://www.jb51.net/article/62909.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | class Professor  {    String name;    int age;    Professor(String name,int age)    {    this.name=name;    this.age=age;    }  }  public class Student implements Cloneable  {    String name;// 常量对象。    int age;    Professor p;// 学生1和学生2的引用值都是一样的。    Student(String name,int age,Professor p)    {    this.name=name;    this.age=age;    this.p=p;    }   public Object clone()    {     Student o=null;    try     {      o=(Student)super.clone();     }    catch(CloneNotSupportedException e)     {      System.out.println(e.toString());     }     o.p=(Professor)p.clone();    return o;    }   public static void main(String[] args)   {    Professor p=new Professor("wangwu",50);    Student s1=new Student("zhangsan",18,p);    Student s2=(Student)s1.clone();    s2.p.name="lisi";    s2.p.age=30;    System.out.println("name="+s1.p.name+","+"age="+s1.p.age);    System.out.println("name="+s2.p.name+","+"age="+s2.p.age);    //输出结果学生1和2的教授成为lisi,age为30。    }  } |

那应该如何实现深层次的克隆，即修改s2的教授不会影响s1的教授？代码改进如下。   
改进使学生1的Professor不改变（深层次的克隆）

[?](http://www.jb51.net/article/62909.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | class Professor implements Cloneable  {    String name;    int age;    Professor(String name,int age)    {    this.name=name;    this.age=age;    }   public Object clone()    {     Object o=null;    try     {      o=super.clone();     }    catch(CloneNotSupportedException e)     {      System.out.println(e.toString());     }    return o;    }  }  public class Student implements Cloneable  {    String name;    int age;    Professor p;    Student(String name,int age,Professor p)    {    this.name=name;    this.age=age;    this.p=p;    }   public Object clone()    {     Student o=null;    try     {      o=(Student)super.clone();     }    catch(CloneNotSupportedException e)     {      System.out.println(e.toString());     }     //对引用的对象也进行复制     o.p=(Professor)p.clone();    return o;    }   public static void main(String[] args)    {    Professor p=new Professor("wangwu",50);    Student s1=new Student("zhangsan",18,p);    Student s2=(Student)s1.clone();    s2.p.name="lisi";    s2.p.age=30;    //学生1的教授不 改变。    System.out.println("name="+s1.p.name+","+"age="+s1.p.age);    System.out.println("name="+s2.p.name+","+"age="+s2.p.age);   }  } |

**3．利用串行化来做深复制**（主要是为了避免重写比较复杂对象的深复制的clone（）方法，也可以程序实现断点续传等等功能）   
    把对象写到流里的过程是串行化（Serilization）过程，但是在Java程序师圈子里又非常形象地称为“冷冻”或者“腌咸菜（picking）”过程；而把对象从流中读出来的并行化（Deserialization）过程则叫做 “解冻”或者“回鲜(depicking)”过程。   
    应当指出的是，写在流里的是对象的一个拷贝，而原对象仍然存在于JVM里面，因此“腌成咸菜”的只是对象的一个拷贝，Java咸菜还可以回鲜。   
    在Java语言里深复制一个对象，常常可以先使对象实现Serializable接口，然后把对象（实际上只是对象的一个拷贝）写到一个流里（腌成咸菜），再从流里读出来（把咸菜回鲜），便可以重建对象。   
如下为深复制源代码。

[?](http://www.jb51.net/article/62909.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public Object deepClone()  {   //将对象写到流里   ByteArrayOutoutStream bo=new ByteArrayOutputStream();   ObjectOutputStream oo=new ObjectOutputStream(bo);   oo.writeObject(this);   //从流里读出来   ByteArrayInputStream bi=new ByteArrayInputStream(bo.toByteArray());   ObjectInputStream oi=new ObjectInputStream(bi);   return(oi.readObject());  } |

这样做的前提是对象以及对象内部所有引用到的对象都是可串行化的，否则，就需要仔细考察那些不可串行化的对象或属性可否设成transient，从而将之排除在复制过程之外。上例代码改进如下。

[?](http://www.jb51.net/article/62909.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | class Teacher implements Serializable{    String name;    int age;    public void Teacher(String name,int age){    this.name=name;    this.age=age;    }  }  public class Student implements Serializable{   String name;//常量对象   int age;   Teacher t;//学生1和学生2的引用值都是一样的。   public void Student(String name,int age,Teacher t){    this.name=name;    this.age=age;    this.p=p;   }   public Object deepClone() throws IOException,      OptionalDataException,ClassNotFoundException{//将对象写到流里    ByteArrayOutoutStream bo=new ByteArrayOutputStream();    ObjectOutputStream oo=new ObjectOutputStream(bo);    oo.writeObject(this);//从流里读出来    ByteArrayInputStream bi=new ByteArrayInputStream(bo.toByteArray());    ObjectInputStream oi=new ObjectInputStream(bi);    return(oi.readObject());   }   public static void main(String[] args){    Teacher t=new Teacher("tangliang",30);    Student s1=new Student("zhangsan",18,t);    Student s2=(Student)s1.deepClone();    s2.t.name="tony";    s2.t.age=40;    //学生1的老师不改变    System.out.println("name="+s1.t.name+","+"age="+s1.t.age);   }  } |