

Java面试(1)

Java序列化(2)

文章档案(5)

2016年7月 (5)

学习网站

Java面试题

Java入门基础教程

JQuery插件库

菜鸟要飞

码农网！！！！

慕课网

手册网

在线编辑、展示、分享、交流你的

JavaScript 代码

在线手册大全

CSS3.0中文手册

JAVA学习笔记

jQuery 1.8 参考手册

SQL基础教程

在线手册大全汇总

积分与排名

积分 - 82800

排名 - 3587

最新评论

1. Re:Java提高篇——Java 异常处理

腻害大神的分析的飞常到位，看到这篇文章突然想起之前看到过的一个比较好的java学习平台非常适合视频资料学习跟弹窗人工要全套的java资料。希望大家学习了能够对自己有帮助...

--小小牙

2. Re:Java文件操作①——XML文件的读取

对于一个想学习java的朋友实际动手能力是非常重要的，我给大家分享一个对于自学的朋友非常好的资料 学习卡的形势分享 需要人工申请 切记要密码啊...

--小小牙

3. Re:框架基础——全面解析Java注解

不错！

--zzyo

4. Re:Java数据库连接——JDBC调用存储过程,事务管理和高级应用

请教下，如果jdbc调用存储过程时，存储过程里有多commit，在中间失败了，那么jdbc的还能控制全部回滚吗？

--sky_snow

5. Re:Java提高篇——对象克隆（复制）

讲的非常好。浅显易懂。

这里我们自定义了一个学生类，该类只有一个number字段。

我们新建了一个学生实例，然后将该值赋值给stu2实例。(Student stu2 = stu1;)

再看看打印结果，作为一个新手，拍了拍胸腹，对象复制不过如此，

难道真的是这样吗？

我们试着改变stu2实例的number字段，再打印结果看看：

```
stu2.setNumber(54321);

System.out.println("学生1:" + stu1.getNumber());
System.out.println("学生2:" + stu2.getNumber());
```

结果：

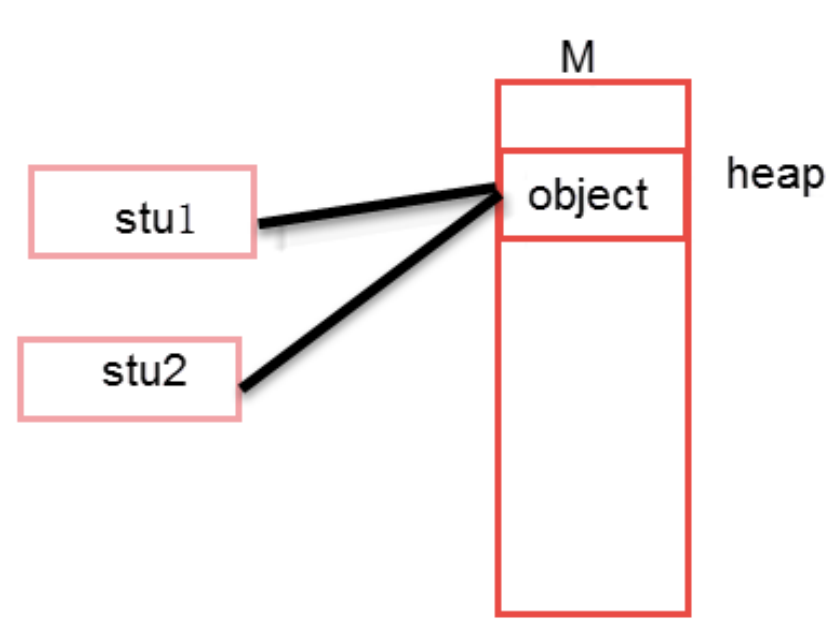
学生1:54321

学生2:54321

这就怪了，为什么改变学生2的学号，学生1的学号也发生了变化呢？

原因出在(stu2 = stu1) 这一句。该语句的作用是将stu1的引用赋值给stu2，

这样，stu1和stu2指向内存堆中同一个对象。如图：



那么，怎样才能达到复制一个对象呢？

是否记得万类之王Object。它有11个方法，有两个protected的方法，其中一个为clone方法。

在Java中所有的类都是缺省的继承自Java语言包中的Object类的，查看它的源码，你可以把你的JDK目录下的src.zip复制到其他地方然后解压，里面就是所有的源码。发现里面有一个访问限定符为protected的方法clone()：

```
/*
 *
 * Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object.
 * The general intent is that, for any object x, the expression:
 * 1) x.clone() != x will be true
 * 2) x.clone().getClass() == x.getClass() will be true, but these are not absolute requirements.
 * 3) x.clone().equals(x) will be true, this is not an absolute requirement.
 *
 */
protected native Object clone() throws CloneNotSupportedException;
```

仔细一看，它还是一个native方法，大家都知道native方法是非Java语言实现的代码，供Java程序调用的，因为Java程序是运行在JVM虚拟机上面的，要想访问到比较底层的与操作系统相关的就没办法了，只能由靠近操作系统的语言来实现。

1. 第一次声明保证克隆对象将有单独的内存地址分配。
2. 第二次声明表明，原始和克隆的对象应该具有相同的类类型，但它不是强制性的。
3. 第三声明表明，原始和克隆的对象应该是平等的equals()方法使用，但它不是强制性的。

因为每个类直接或间接的父类都是Object，因此它们都含有clone()方法，但是因为该方法是protected，所以都不能在类外进行访问。

阅读排行榜

- 1. Java提高篇——对象克隆（复制）(35152)
- 2. Java文件操作①——XML文件的读取(25021)
- 3. window.location.href和window.open的几种用法和区别(21945)
- 4. Java数据库连接——JDBC基础知识（操作数据库：增删改查）(18727)
- 5. Java提高篇——equals()与hashCode()方法详解(13751)
- 6. Oracle数据库相关问题之ORA-12541:TNS:无监听程序(7139)
- 7. Java提高篇——Java 异常处理(5701)
- 8. Java提高篇——JVM加载class文件的原理机制(5082)
- 9. 请求转发（Forward）和重定向（Redirect）的区别(4842)
- 10. java 性能优化：35 个细节，让你提升 java 代码的运行效率(4653)

评论排行榜

- 1. Java提高篇——对象克隆（复制）(9)
- 2. 文件传输基础——Java IO流(9)
- 3. Java数据库连接——JDBC基础知识（操作数据库：增删改查）(9)
- 4. Oracle数据库之SQL基础（二）(6)
- 5. Java提高篇——equals()与hashCode()方法详解(4)

推荐排行榜

- 1. Java提高篇——对象克隆（复制）(13)
- 2. Java提高篇——equals()与hashCode()方法详解(10)
- 3. Java提高篇——静态代码块、构造代码块、构造函数以及Java类初始化顺序(10)
- 4. Java文件操作①——XML文件的读取(9)
- 5. Java数据库连接——JDBC基础知识（操作数据库：增删改查）(9)
- 6. Java提高篇——Java 异常处理(5)
- 7. Java提高篇——JVM加载class文件的原理机制(4)
- 8. 请求转发（Forward）和重定向（Redirect）的区别(4)
- 9. window.location.href和window.open的几种用法和区别(4)
- 10. Java中Native关键字的作用(4)

要想对一个对象进行复制，就需要对clone方法覆盖。

[回到顶部](#)
[回到顶部](#)

为什么要克隆？

大家先思考一个问题，为什么需要克隆对象？直接new一个对象不行吗？

答案是：克隆的对象可能包含一些已经修改过的属性，而new出来的对象的属性都还是初始化时候的值，所以当需要一个新的对象来保存当前对象的"状态"就靠clone方法了。那么我把这个对象的临时属性一个一个的赋值给我新new的对象不也行嘛？可以是可以，但是一来麻烦不说，二来，大家通过上面的源码都发现了clone是一个native方法，就是快啊，在底层实现的。

提个醒，我们常见的Object a=new Object();Object b;b=a;这种形式的代码复制的是引用，即对象在内存中的地址，a和b对象仍然指向了同一个对象。

而通过clone方法赋值的对象跟原来的对象时同时独立存在的。

[回到顶部](#)
[回到顶部](#)

如何实现克隆

先介绍一下两种不同的克隆方法，浅克隆(ShallowClone)和深克隆(DeepClone)。

在Java语言中，数据类型分为值类型（基本数据类型）和引用类型，值类型包括int、double、byte、boolean、char等简单数据类型，引用类型包括类、接口、数组等复杂类型。浅克隆和深克隆的主要区别在于是否支持引用类型的成员变量的复制，下面将对两者进行详细介绍。

一般步骤是（浅克隆）：

- 1. 被复制的类需要实现Cloneable接口（不实现的话在调用clone方法会抛出CloneNotSupportedException异常），该接口为标记接口(不含任何方法)
- 2. 覆盖clone()方法，访问修饰符设为public。方法中调用super.clone()方法得到需要的复制对象。（native为本地方法)

下面对上面那个方法进行改造：



```
class Student implements Cloneable{  
    private int number;  
  
    public int getNumber() {  
        return number;  
    }  
  
    public void setNumber(int number) {  
        this.number = number;  
    }  
  
    @Override  
    public Object clone() {  
        Student stu = null;  
        try{  
            stu = (Student)super.clone();  
        }catch(CloneNotSupportedException e) {  
            e.printStackTrace();  
        }  
        return stu;  
    }  
}  
  
public class Test {  
    public static void main(String args[]) {  
        Student stu1 = new Student();  
        stu1.setNumber(12345);  
        Student stu2 = (Student)stu1.clone();  
  
        System.out.println("学生1:" + stu1.getNumber());  
        System.out.println("学生2:" + stu2.getNumber());  
  
        stu2.setNumber(54321);  
  
        System.out.println("学生1:" + stu1.getNumber());
```



```
        System.out.println("学生2:" + stu2.getNumber());
    }
}
```

结果：

学生1:12345

学生2:12345

学生1:12345

学生2:54321

如果你还不相信这两个对象不是同一个对象，那么你可以看看这一句：

```
System.out.println(stu1 == stu2); // false
```

上面的复制被称为浅克隆。

还有一种稍微复杂的深度复制：

我们在学生类里再加一个**Address**类。

```
class Address {
    private String add;

    public String getAdd() {
        return add;
    }

    public void setAdd(String add) {
        this.add = add;
    }
}

class Student implements Cloneable{
    private int number;
    private Address addr;

    public Address getAddr() {
        return addr;
    }

    public void setAddr(Address addr) {
        this.addr = addr;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    @Override
    public Object clone() {
        Student stu = null;
        try{
            stu = (Student)super.clone();
        }catch(CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return stu;
    }
}
```

```
44     }
45 }
46 public class Test {
47
48     public static void main(String args[]) {
49
50         Address addr = new Address();
51         addr.setAdd("杭州市");
52         Student stu1 = new Student();
53         stu1.setNumber(123);
54         stu1.setAddr(addr);
55
56         Student stu2 = (Student)stu1.clone();
57
58         System.out.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());
59         System.out.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());
60     }
61 }
```






结果：

学生1:123,地址:杭州市

学生2:123,地址:杭州市


乍一看没什么问题，真的是这样吗？

我们在main方法中试着改变addr实例的地址。




```
addr.setAdd("西湖区");


System.out.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());
System.out.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());
```



结果：




```
学生1:123,地址:杭州市
学生2:123,地址:杭州市
学生1:123,地址:西湖区
学生2:123,地址:西湖区
```



这就奇怪了，怎么两个学生的地址都改变了？

原因是浅复制只是复制了addr变量的引用，并没有真正的开辟另一块空间，将值复制后再将引用返回给新对象。

所以，为了达到真正的复制对象，而不是纯粹引用复制。我们需要将Address类可复制化，并且修改clone方法，完整代码如下：





```
1 package abc;
2
3 class Address implements Cloneable {
4     private String add;
5
6     public String getAdd() {
7         return add;
8     }
9
10    public void setAdd(String add) {
11        this.add = add;
12    }
13
14    @Override
```

```
15     public Object clone() {
16         Address addr = null;
17         try{
18             addr = (Address)super.clone();
19         }catch(CloneNotSupportedException e) {
20             e.printStackTrace();
21         }
22         return addr;
23     }
24 }
25
26 class Student implements Cloneable{
27     private int number;
28
29     private Address addr;
30
31     public Address getAddr() {
32         return addr;
33     }
34
35     public void setAddr(Address addr) {
36         this.addr = addr;
37     }
38
39     public int getNumber() {
40         return number;
41     }
42
43     public void setNumber(int number) {
44         this.number = number;
45     }
46
47     @Override
48     public Object clone() {
49         Student stu = null;
50         try{
51             stu = (Student)super.clone();    //浅复制
52         }catch(CloneNotSupportedException e) {
53             e.printStackTrace();
54         }
55         stu.addr = (Address)addr.clone();    //深度复制
56         return stu;
57     }
58 }
59 public class Test {
60
61     public static void main(String args[]) {
62
63         Address addr = new Address();
64         addr.setAdd("杭州市");
65         Student stu1 = new Student();
66         stu1.setNumber(123);
67         stu1.setAddr(addr);
68
69         Student stu2 = (Student)stu1.clone();
70
71         System.out.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());
72         System.out.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());
73
74         addr.setAdd("西湖区");
75
76         System.out.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());
77         System.out.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());
78     }
79 }
```



结果：



学生1:123,地址:杭州市

学生2:123,地址:杭州市

学生1:123,地址:西湖区

学生2:123,地址:杭州市

这样结果就符合我们的想法了。

最后我们可以看看API里其中一个实现了clone方法的类：

java.util.Date:

```
/**
 * Return a copy of this object.
 */
public Object clone() {
    Date d = null;
    try {
        d = (Date)super.clone();
        if (cdate != null) {
            d.cdate = (BaseCalendar.Date) cdate.clone();
        }
    } catch (CloneNotSupportedException e) {} // Won't happen
    return d;
}
```

该类其实也属于深度复制。

参考文档：[Java如何复制对象](#)

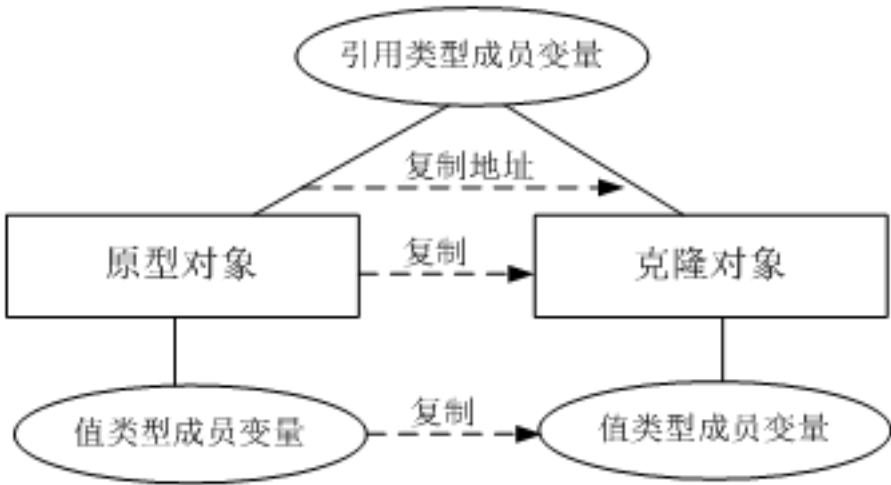
[回到顶部](#)
[回到顶部](#)

浅克隆和深克隆

1、浅克隆

在浅克隆中，如果原型对象的成员变量是值类型，将复制一份给克隆对象；如果原型对象的成员变量是引用类型，则将引用对象的地址复制一份给克隆对象，也就是说原型对象和克隆对象的成员变量指向相同的内存地址。

简单来说，在浅克隆中，当对象被复制时只复制它本身和其中包含的值类型的成员变量，而引用类型的成员对象并没有复制。

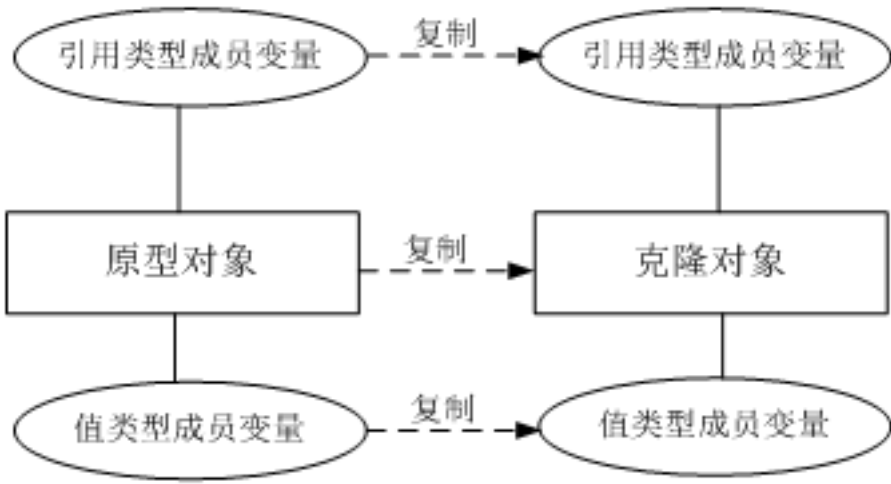


在Java语言中，通过覆盖Object类的clone()方法可以实现浅克隆。

2、深克隆

在深克隆中，无论原型对象的成员变量是值类型还是引用类型，都将复制一份给克隆对象，深克隆将原型对象的所有引用对象也复制一份给克隆对象。

简单来说，在深克隆中，除了对象本身被复制外，对象所包含的所有成员变量也将复制。



在Java语言中，如果需要实现深克隆，可以通过覆盖Object类的clone()方法实现，也可以通过序列化(Serialization)等方式来实现。

（如果引用类型里面还包含很多引用类型，或者内层引用类型的类里面又包含引用类型，使用clone方法就会很麻烦。这时我们可以用序列化的方式来实现对象的深克隆。）

序列化就是将对象写到流的过程，写到流中的对象是原有对象的一个拷贝，而原对象仍然存在于内存中。通过序列化实现的拷贝不仅可以复制对象本身，而且可以复制其引用的成员对象，因此通过序列化将对象写到一个流中，再从流里将其读出来，可以实现深克隆。需要注意的是能够实现序列化的对象其类必须实现Serializable接口，否则无法实现序列化操作。

扩展

Java语言提供的Cloneable接口和Serializable接口的代码非常简单，它们都是空接口，这种空接口也称为标识接口，标识接口中没有任何方法的定义，其作用是告诉JRE这些接口的实现类是否具有某个功能，如是否支持克隆、是否支持序列化等。

[回到顶部](#)
[回到顶部](#)

解决多层克隆问题

如果引用类型里面还包含很多引用类型，或者内层引用类型的类里面又包含引用类型，使用clone方法就会很麻烦。这时我们可以用序列化的方式来实现对象的深克隆。





```
1 public class Outer implements Serializable{
2     private static final long serialVersionUID = 3692852985729411L;    //最好是显式声明ID
3     public Inner inner;
4     //Discription: [深度复制方法,需要对象及对象所有的对象属性都实现序列化]
5     public Outer myclone() {
6         Outer outer = null;
7         try { // 将该对象序列化成流,因为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。所以利用这个特性可以实现对象的深拷贝
8             ByteArrayOutputStream baos = new ByteArrayOutputStream();
9             ObjectOutputStream oos = new ObjectOutputStream(baos);
10            oos.writeObject(this);
11            // 将流序列化成对象
12            ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());
13            ObjectInputStream ois = new ObjectInputStream(bais);
14            outer = (Outer) ois.readObject();
15        } catch (IOException e) {
16            e.printStackTrace();
17        } catch (ClassNotFoundException e) {
18            e.printStackTrace();
19        }
20        return outer;
21    }
22 }
```





Inner也必须实现Serializable，否则无法序列化：





```
1 public class Inner implements Serializable{
2     private static final long serialVersionUID = 872390113109L; //最好是显式声明ID
3     public String name = "";
4
5     public Inner(String name) {
6         this.name = name;
7     }
8
9     @Override
10    public String toString() {
11        return "Inner的name值为: " + name;
12    }
13 }
```





这样也能使两个对象在内存空间内完全独立存在，互不影响对方的值。

[回到顶部](#)

[回到顶部](#)

总结

实现对象克隆有两种方式：

- 1). 实现Cloneable接口并重写Object类中的clone()方法；
- 2). 实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化的，不是在运行时抛出异常，这种是方案明显优于使用Object类的clone方法克隆对象。让问题在编译的时候暴露出来总是优于

文章链接：

[Java进阶系列之对象克隆](#)

[Java基础笔记 - 对象的深复制与浅复制 实现Cloneable接口实现浅复制 序列化实现深复制](#)

[详解Java中的对象克隆](#)

[Java对象克隆指南](#)

[对象的克隆——原型模式（三）](#)

[java提高篇（五）-----使用序列化实现对象的拷贝](#)

-----我是低调的分割线-----

如果对你有帮助，可以点击“推荐”哦`(*^_^*)`



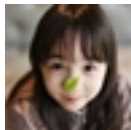
分类: [①Java基础学习笔记](#)

标签: [java基础](#)

好文要顶

关注我

收藏该文



萌小Q

关注 - 6

粉丝 - 262

[+加关注](#)

13

推荐

0

反对

« 上一篇: [Java提高篇——JVM加载class文件的原理机制](#)

» 下一篇: [Java提高篇——静态代码块、构造代码块、构造函数以及Java类初始化顺序](#)

posted @ 2016-07-27 14:00 [萌小Q](#) 阅读(35166) 评论(9) 编辑 收藏

努力加载评论中...

[刷新评论](#) [刷新页面](#) [返回顶部](#)



注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】 [50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库](#)

【活动】 [腾讯云](#) **【云+校园】** [套餐全新升级](#)

【推荐】 [报表开发有捷径：快速设计轻松集成，数据可视化和交互](#)



最新IT新闻:

- [三星手机又陷“冒烟门”：航班起飞后Galaxy J7冒烟](#)
 - [北美城市疯抢亚马逊第二总部 税收优惠最高达70亿美元](#)
 - [趣店受到多家自媒体连续质疑 创始人罗敏回应一切](#)
 - [英国政府投资5100万英镑资助自动驾驶测试设施建造](#)
 - [NASA即将让“黎明号”开启第二次谷神星探测任务](#)
- » [更多新闻...](#)



最新知识库文章:

- [实用VPC虚拟私有云设计原则](#)
 - [如何阅读计算机科学类的书](#)
 - [Google 及其云智慧](#)
 - [做到这一点，你也可以成为优秀的程序员](#)
 - [写给立志做码农的大学生](#)
- » [更多知识库文章...](#)