<https://blog.csdn.net/jeffleo/article/details/76737560>

# Java对象克隆——浅克隆和深克隆的区别

2017年08月06日 00:51:35 [JeffCoding](https://me.csdn.net/jeffleo" \t "https://blog.csdn.net/jeffleo/article/details/_blank) 阅读数：3685 标签： [java](http://so.csdn.net/so/search/s.do?q=java&t=blog" \t "https://blog.csdn.net/jeffleo/article/details/_blank)[深克隆](http://so.csdn.net/so/search/s.do?q=%E6%B7%B1%E5%85%8B%E9%9A%86&t=blog" \t "https://blog.csdn.net/jeffleo/article/details/_blank)[浅克隆](http://so.csdn.net/so/search/s.do?q=%E6%B5%85%E5%85%8B%E9%9A%86&t=blog" \t "https://blog.csdn.net/jeffleo/article/details/_blank)更多

个人分类： [Java基础](https://blog.csdn.net/jeffleo/article/category/6358800" \t "https://blog.csdn.net/jeffleo/article/details/_blank)

在***[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "https://blog.csdn.net/jeffleo/article/details/_blank)***中对象的克隆有深克隆和浅克隆之分。有这种区分的原因是*Java*中分为基本数据类型和引用数据类型，对于不同的数据类型在内存中的存储的区域是不同的。基本数据类型存储在栈中，引用数据类型存储在堆中。

## **什么是克隆**

克隆就是依据已经有的数据，创造一份新的完全一样的数据拷贝。

实现克隆有多种方式，可以手工的*new*出一个新的对象，然后将原来的对象信息一个一个的*set*到新的对象中。还有就是使用*clone*方法。使用*clone*方法必须满足：

****实现*Cloneable*接口****

****使用*public*访问修饰符重新定义*clone*方法。****

在不使用克隆方法时，将类*A*的实例*A1*直接赋给类*A*的新实例*A2*时会出现这样的情况，修改*A2*的属性，*A1*的属性也发生了改变。这是因为赋值操作之后，*A1*和*A2*指向同一个对象，就像是使用不同的显示器操作同一个服务器一样，两个显示器显示的都是一个服务器上的内容。

## 浅克隆

对于一个只含有基本数据类型的类来说使用*clone*方法，是完全没有问题的，用图表来表示该情形的*clone*操作：

|  |  |  |
| --- | --- | --- |
| *Customer  customer2=customer1.clone();* | | |
| *customer1* | *ID* | *123* |
| *age* | *23* |
| *customer2* | *ID* | *123* |
| *age* | *23* |
| *customer2.setAge(32);* | | |
| *customer1* | *ID* | *123* |
| *age* | *23* |
| *customer2* | *ID* | *123* |
| *age* | *32* |

在*clone*后*customer1*和*customer2*之间数据互不影响。

但是如果在*Customer*类中有一个引用类型的属性*Address*呢？

****[java]**** [view plain](http://blog.csdn.net/goskalrie/article/details/50790056" \o "view plain" \t "https://blog.csdn.net/jeffleo/article/details/_blank) [copy](http://blog.csdn.net/goskalrie/article/details/50790056" \o "copy" \t "https://blog.csdn.net/jeffleo/article/details/_blank)

1. **public** **static** **void** main(String[] args) **throws** CloneNotSupportedException {
2. Address address = **new** Address("CH" , "SD" , "QD");
3. Customer customer1 = **new** Customer(1 , 23 , address);
4. Customer customer2 = customer1.clone();
5. customer2.getAddress().setCity("JN");
6. customer2.setID(2);
7. System.out.println("customer1:"+customer1.toString());
8. System.out.println("customer2:"+customer2.toString());
9. }
10. }
11. **class** Customer **implements** Cloneable{
12. **public** **int** ID;
13. **public** **int** age;
14. **public** Address address;
15. **public** **int** getID() {
16. **return** ID;
17. }
18. **public** **void** setID(**int** iD) {
19. ID = iD;
20. }
21. **public** **int** getAge() {
22. **return** age;
23. }
24. **public** **void** setAge(**int** age) {
25. **this**.age = age;
26. }
27. **public** Address getAddress() {
28. **return** address;
29. }
30. **public** **void** setAddress(Address address) {
31. **this**.address = address;
32. }
33. **public** Customer(**int** iD, **int** age, Address address) {
34. **super**();
35. ID = iD;
36. **this**.age = age;
37. **this**.address = address;
38. }
39. @Override
40. **public** String toString() {
41. **return** "Customer [ID=" + ID + ", age=" + age + ", address=" + address
42. + "]";
43. }
44. @Override
45. **public** Customer clone() **throws** CloneNotSupportedException {
46. **return** (Customer) **super**.clone();
47. }
48. }
49. **class** Address{
50. **private** String country;
51. **private** String province;
52. **private** String city;
53. **public** String getCountry() {
54. **return** country;
55. }
56. **public** **void** setCountry(String country) {
57. **this**.country = country;
58. }
59. **public** String getProvince() {
60. **return** province;
61. }
62. **public** **void** setProvince(String province) {
63. **this**.province = province;
64. }
65. **public** String getCity() {
66. **return** city;
67. }
68. **public** **void** setCity(String city) {
69. **this**.city = city;
70. }
71. @Override
72. **public** String toString() {
73. **return** "Address [country=" + country + ", province=" + province
74. + ", city=" + city + "]";
75. }
76. **public** Address(String country, String province, String city) {
77. **super**();
78. **this**.country = country;
79. **this**.province = province;
80. **this**.city = city;
81. }
82. }
83. //输出的结果是：
84. //customer1:Customer [ID=1, age=23, address=Address [country=CH, province=SD, city=JN]]
85. //customer2:Customer [ID=2, age=23, address=Address [country=CH, province=SD, city=JN]]

上面分析得到，*clone*后新旧对象互不影响，*customer2*修改了*id*后没有影响到*customer1*，但是修改了*customer2*的*address*属性的*city*值为*JN*后，发现*customer1*的*address*值也发生了改变。****这样就没有达到完全复制、相互之间完全没有影响的目的。****这样就需要进行深克隆。

## **2.深克隆**

深克隆与浅克隆的区别就是，浅克隆不会克隆原对象中的引用类型，仅仅拷贝了引用类型的指向。深克隆则拷贝了所有。也就是说深克隆能够做到原对象和新对象之间完全没有影响。

而深克隆的实现就是在引用类型所在的类实现*Cloneable*接口，并使用*public*访问修饰符重写*clone*方法。

上面的代码做以下修改：

*1*．*Address*类实现*Cloneable*接口，重写*clone*方法*;*

****[java]**** [view plain](http://blog.csdn.net/goskalrie/article/details/50790056" \o "view plain" \t "https://blog.csdn.net/jeffleo/article/details/_blank) [copy](http://blog.csdn.net/goskalrie/article/details/50790056" \o "copy" \t "https://blog.csdn.net/jeffleo/article/details/_blank)

1. @Override
2. **public** Address clone() **throws** CloneNotSupportedException {
3. **return** (Address) **super**.clone();
4. }

*2.*在*Customer*类的*clone*方法中调用*Address*类的*clone*方法。

****[java]**** [view plain](http://blog.csdn.net/goskalrie/article/details/50790056" \o "view plain" \t "https://blog.csdn.net/jeffleo/article/details/_blank) [copy](http://blog.csdn.net/goskalrie/article/details/50790056" \o "copy" \t "https://blog.csdn.net/jeffleo/article/details/_blank)

1. @Override
2. **public** Customer clone() **throws** CloneNotSupportedException {
3. Customer customer = (Customer) **super**.clone();
4. customer.address = address.clone();
5. **return** customer;
6. }

修改后**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "https://blog.csdn.net/jeffleo/article/details/_blank)**代码的输出结果：

*customer1:Customer[ID=1, age=23, address=Address [country=CH, province=SD, city=QD]]*

*customer2:Customer[ID=2, age=23, address=Address [country=CH, province=SD, city=JN]]*

发现*customer2*无论如何修改，*customer1*都没有受到影响。

实现深克隆的另一种方法就是使用****序列化****，将对象写入到流中，这样对象的内容就变成了字节流，也就****不存在什么引用****了。然后读取字节流反序列化为对象就完成了完全的复制操作了。

****[java]**** [view plain](http://blog.csdn.net/goskalrie/article/details/50790056" \o "view plain" \t "https://blog.csdn.net/jeffleo/article/details/_blank) [copy](http://blog.csdn.net/goskalrie/article/details/50790056" \o "copy" \t "https://blog.csdn.net/jeffleo/article/details/_blank)

1. Address address = **new** Address("CH" , "SD" , "QD");
2. Customer customer1 = **new** Customer(1 , 23 , address);
3. Customer customer2 = (Customer) cloneObject(customer1);
4. customer2.getAddress().setCity("JN");
5. customer2.setID(2);
6. System.out.println("customer1:"+customer1.toString());
7. System.out.println("customer2:"+customer2.toString());
8. //customer1:Customer [ID=1, age=23, address=Address [country=CH, province=SD, city=QD]]
9. //customer2:Customer [ID=2, age=23, address=Address [country=CH, province=SD, city=JN]]

*cloneObject*方法的定义：

****[java]**** [view plain](http://blog.csdn.net/goskalrie/article/details/50790056" \o "view plain" \t "https://blog.csdn.net/jeffleo/article/details/_blank) [copy](http://blog.csdn.net/goskalrie/article/details/50790056" \o "copy" \t "https://blog.csdn.net/jeffleo/article/details/_blank)

1. **public** **static** Object cloneObject(Object obj) **throws** IOException, ClassNotFoundException{
2. ByteArrayOutputStream byteOut = **new** ByteArrayOutputStream();
3. ObjectOutputStream out = **new** ObjectOutputStream(byteOut);
4. out.writeObject(obj);
5. ByteArrayInputStream byteIn = **new** ByteArrayInputStream(byteOut.toByteArray());
6. ObjectInputStream in =**new** ObjectInputStream(byteIn);
7. **return** in.readObject();
8. }

个人的理解是*Java*中定义的*clone*没有深浅之分，都是统一的调用*Object*的*clone*方法。为什么会有深克隆的概念？是由于我们在实现的过程中刻意的嵌套了*clone*方法的调用。也就是说深克隆就是在需要克隆的对象类型的类中全部实现克隆方法。就像是*toString*方法一样，假如上面的*Customer*类中重写了*toString*方法，而*Address*类没有进行重写，就会出现这样的输出语句：

*customer1:Customer[ID=1, age=23,******address=com.gos.java.standard.Address@38d8fb2b******]*

只有在Address类也重写了toString方法才会打印出完全的信息：

****customer1:Customer [ID=1, age=23, address=Address [country=CH, province=SD, city=QD]]****

只不过在打印的操作中就默认的调用了对象的*toString*方法，而*clone*方法需要在代码中显式的调用。

## **总结：**

1.浅克隆：只复制基本类型的数据，引用类型的数据只复制了引用的地址，引用的对象并没有复制，在新的对象中修改引用类型的数据会影响原对象中的引用。  
2.深克隆：是在引用类型的类中也实现了clone，是clone的嵌套，复制后的对象与原对象之间完全不会影响。  
3.使用序列化也能完成深复制的功能：对象序列化后写入流中，此时也就不存在引用什么的概念了，再从流中读取，生成新的对象，新对象和原对象之间也是完全互不影响的。  
4.使用clone实现的深克隆其实是浅克隆中嵌套了浅克隆，与toString方法类似