[JAVA 对象拷贝](http://blog.csdn.net/jdluojing/article/details/6963112)

1.java里的clone分为：   
A:浅拷贝： 浅复制仅仅复制所考虑的对象，而不复制它所引用的对象。   
b:深拷贝：深复制把要复制的对象所引用的对象都复制了一遍。

Java中对象的克隆,为了获取对象的一份拷贝，我们可以利用Object类的clone()方法。必须要遵循下面三点   
1.在派生类中覆盖基类的clone()方法，并声明为public【Object类中的clone()方法为protected的】。   
2.在派生类的clone()方法中，调用super.clone()。   
3.在派生类中实现Cloneable接口。   
  
Object类里的clone方法是浅拷贝   
  
浅拷贝的例子如下：

1. **public** **class** CloneTest
2. {
4. **public** **static** **void** main(String[] args) **throws** Exception
5. {
6. // teacher对象将被clone出来的Student对象共享.
7. Teacher teacher = **new** Teacher();
8. teacher.setAge(40);
9. teacher.setName("Teacher zhang");
11. Student student1 = **new** Student();
12. student1.setAge(20);
13. student1.setName("zhangsan");
14. student1.setTeacher(teacher);
16. // 复制出来一个对象student2
17. Student student2 = (Student) student1.clone();
18. System.out.println(student2.getAge());
19. System.out.println(student2.getName());
21. System.out.println("~~~~~~~~~~~~~~~~~~~~~~");
22. System.out.println(student1.getTeacher().getAge());
23. System.out.println(student1.getTeacher().getName());
25. // 修改student2的引用对象
26. student2.getTeacher().setAge(50);
27. student2.getTeacher().setName("Teacher Li");
29. System.out.println("~~~~~~~~~~~~~~~~~~~~~~");
30. System.out.println(student1.getTeacher().getAge());
31. System.out.println(student1.getTeacher().getName());
32. }
33. }
35. **class** Teacher
36. {
37. **public** **int** age;
38. **public** String name;
40. **public** **int** getAge()
41. {
42. **return** age;
43. }
45. **public** **void** setAge(**int** age)
46. {
47. **this**.age = age;
48. }
50. **public** String getName()
51. {
52. **return** name;
53. }
55. **public** **void** setName(String name)
56. {
57. **this**.name = name;
58. }
60. }
62. **class** Student **implements** Cloneable
63. {
65. **public** **int** age;
66. **public** String name;
67. **public** Teacher teacher;
69. **public** **int** getAge()
70. {
71. **return** age;
72. }
74. **public** **void** setAge(**int** age)
75. {
76. **this**.age = age;
77. }
79. **public** String getName()
80. {
81. **return** name;
82. }
84. **public** **void** setName(String name)
85. {
86. **this**.name = name;
87. }
89. **public** Teacher getTeacher()
90. {
91. **return** teacher;
92. }
94. **public** **void** setTeacher(Teacher teacher)
95. {
96. **this**.teacher = teacher;
97. }
99. @Override
100. **public** Object clone() **throws** CloneNotSupportedException
101. {
102. **return** **super**.clone();
103. }
104. }
105. 输出结果为：
106. 20
107. zhangsan
108. ~~~~~~~~~~~~~~~~~~~~~~
109. 40
110. Teacher zhang
111. ~~~~~~~~~~~~~~~~~~~~~~
112. 50
113. Teacher Li

2.深复制（深Clone）例子：

1. **public** **class** DeepCloneTest
2. {
4. **public** **static** **void** main(String[] args) **throws** Exception
5. {
6. // teacher对象将不被clone出来的Student对象共享.
7. Teacher teacher = **new** Teacher();
8. teacher.setAge(40);
9. teacher.setName("Teacher zhang");
11. Student student1 = **new** Student();
12. student1.setAge(20);
13. student1.setName("zhangsan");
14. student1.setTeacher(teacher);
16. // 复制出来一个对象student2
17. Student student2 = (Student) student1.clone();
18. System.out.println(student2.getAge());
19. System.out.println(student2.getName());
21. System.out.println("~~~~~~~~~~~~~~~~~~~~~~");
22. System.out.println(student1.getTeacher().getAge());
23. System.out.println(student1.getTeacher().getName());
25. // 修改student2的引用对象
26. student2.getTeacher().setAge(50);
27. student2.getTeacher().setName("Teacher Li");
29. System.out.println("~~~~~~~~~~~~~~~~~~~~~~");
30. System.out.println(student1.getTeacher().getAge());
31. System.out.println(student1.getTeacher().getName());
32. }
33. }
35. **class** Teacher **implements** Cloneable
36. {
37. **public** **int** age;
38. **public** String name;
40. **public** **int** getAge()
41. {
42. **return** age;
43. }
45. **public** **void** setAge(**int** age)
46. {
47. **this**.age = age;
48. }
50. **public** String getName()
51. {
52. **return** name;
53. }
55. **public** **void** setName(String name)
56. {
57. **this**.name = name;
58. }
60. @Override
61. **public** Object clone() **throws** CloneNotSupportedException
62. {
63. **return** **super**.clone();
64. }
66. }
68. **class** Student **implements** Cloneable
69. {
71. **public** **int** age;
72. **public** String name;
73. **public** Teacher teacher;
75. **public** **int** getAge()
76. {
77. **return** age;
78. }
80. **public** **void** setAge(**int** age)
81. {
82. **this**.age = age;
83. }
85. **public** String getName()
86. {
87. **return** name;
88. }
90. **public** **void** setName(String name)
91. {
92. **this**.name = name;
93. }
95. **public** Teacher getTeacher()
96. {
97. **return** teacher;
98. }
100. **public** **void** setTeacher(Teacher teacher)
101. {
102. **this**.teacher = teacher;
103. }
105. @Override
106. **public** Object clone() **throws** CloneNotSupportedException
107. {
108. Student student = (Student) **super**.clone();
109. // 将引用的对象teacher也clone下
110. student.setTeacher((Teacher) (student.getTeacher().clone()));   // student.getTeacher().clone()
111. **return** student;
112. }
113. }
115. 输出结果为：
116. 20
117. zhangsan
118. ~~~~~~~~~~~~~~~~~~~~~~
119. 40
120. Teacher zhang
121. ~~~~~~~~~~~~~~~~~~~~~~
122. 40
123. Teacher zhang

3.利用序列化来做深复制,把对象写到流里的过程是序列化（Serilization）过程，而把对象从流中读出来的过程则叫做反序列化（Deserialization）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。，利用这个特性，可以做深拷贝 。

**//妙不可言！！！nice**

1. **import** java.io.ByteArrayInputStream;
2. **import** java.io.ByteArrayOutputStream;
3. **import** java.io.ObjectInputStream;
4. **import** java.io.ObjectOutputStream;
5. **import** java.io.Serializable;
6. //利用序列化来做深复制
7. //深clone
9. **public** **class** DeepCloneTest
10. {
12. **public** **static** **void** main(String[] args) **throws** Exception
13. {
14. // teacher对象将不被clone出来的Student对象共享.
15. Teacher teacher = **new** Teacher();
16. teacher.setAge(40);
17. teacher.setName("Teacher zhang");
19. Student student1 = **new** Student();
20. student1.setAge(20);
21. student1.setName("zhangsan");
22. student1.setTeacher(teacher);
24. // 复制出来一个对象student2
25. Student student2 = (Student) student1.deepCopy();
26. System.out.println(student2.getAge());
27. System.out.println(student2.getName());
29. System.out.println("~~~~~~~~~~~~~~~~~~~~~~");
30. System.out.println(student1.getTeacher().getAge());
31. System.out.println(student1.getTeacher().getName());
33. // 修改student2的引用对象
34. student2.getTeacher().setAge(50);
35. student2.getTeacher().setName("Teacher Li");
37. System.out.println("~~~~~~~~~~~~~~~~~~~~~~");
38. System.out.println(student1.getTeacher().getAge());
39. System.out.println(student1.getTeacher().getName());
40. }
41. }
43. **class** Teacher **implements** Serializable
44. {
46. **private** **static** **final** **long** serialVersionUID = -8834559347461591191L;
48. **public** **int** age;
49. **public** String name;
51. **public** **int** getAge()
52. {
53. **return** age;
54. }
56. **public** **void** setAge(**int** age)
57. {
58. **this**.age = age;
59. }
61. **public** String getName()
62. {
63. **return** name;
64. }
66. **public** **void** setName(String name)
67. {
68. **this**.name = name;
69. }
71. }
73. **class** Student **implements** Serializable
74. {
76. // serialVersionUID
77. // 如果你的对象序列化后存到硬盘上面后，**可是后来你却更改了类的field**(增加或减少或改名)，当你反序列化时，就会出现Exception的，这样就会造成不兼容性的问题。
78. //**但当serialVersionUID相同时，它就会将不一样的field以type的缺省值赋值**(如int型的是0,String型的是null等)，这个可以避开不兼容性的问题。所以最好给serialVersionUID赋值  所以这只是用来赋缺省值
79. **private** **static** **final** **long** serialVersionUID = 7991552226614088458L;
81. **public** **int** age;
82. **public** String name;
83. **public** Teacher teacher;
84. **public** **int** getAge()
85. {
86. **return** age;
87. }
88. **public** **void** setAge(**int** age)
89. {
90. **this**.age = age;
91. }
92. **public** String getName()
93. {
94. **return** name;
95. }
96. **public** **void** setName(String name)
97. {
98. **this**.name = name;
99. }
101. **public** Teacher getTeacher()
102. {
103. **return** teacher;
104. }
106. **public** **void** setTeacher(Teacher teacher)
107. {
108. **this**.teacher = teacher;
109. }
110. /\*\*
111. \* 因为对象序列化会自然地对对象深拷贝，再利用“序列化-反序列化”读出
112. \*/
113. **public** Object deepCopy() **throws** Exception
114. {
115. // 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
116. ByteArrayOutputStream bos = **new** ByteArrayOutputStream();
118. ObjectOutputStream oos = **new** ObjectOutputStream(bos);
119. //this很关键，引用当前对象，当然，这是值传递
120. oos.writeObject(**this**);
122. // 将流序列化成对象
123. ByteArrayInputStream bis = **new** ByteArrayInputStream(bos.toByteArray());
125. ObjectInputStream ois = **new** ObjectInputStream(bis);
127. **return** ois.readObject();
128. }
129. }
131. 输出结果为：
132. 20
133. zhangsan
134. ~~~~~~~~~~~~~~~~~~~~~~
135. 40
136. Teacher zhang
137. ~~~~~~~~~~~~~~~~~~~~~~
138. 40
139. Teacher zhang