1 clone 方法介绍

- Java Object 类中有一个 native 的 clone() 方法
- 该方法会创建一个新的对象（地址不一样）
- 将被克隆对象的**基本类型的值**（包括String）赋给新对象
- 将**引用类型的地址**赋给新对象

2 浅克隆示例

- 一般情况下，直接调用 clone() 方法只是浅克隆，当字段是引用类型时，克隆的只是地址
- 当改变克隆对象的**引用类型的值**（注意是值！）时，同时会改变原对象对应的值，因为都是同一个地址
- 当改变克隆对象的**引用类型的地址**（即 `set(新实例)` ）时，不会改变原对象的值（因为只改变了克隆对象的引用地址，而原对象的引用地址没有改变）

```
public class Resume implements Cloneable {
    private String name;
    private String sex;
    private String age;
    private WorkExperience we;
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();//直接调用 Object 的 clone 方法
    }
    //省略get/set等方法
}
public class WorkExperience {
    private String timeAArea;
    private String company;
    //省略get/set等方法
}
public static void main(String[] args) throws CloneNotSupportedException {
    Resume resume = new Resume();
    resume.setAge("11");
    resume.setName("wk");
    resume.setSex("男");
    resume.setWe(new WorkExperience("12-16","南京"));
    Resume resume1 = (Resume) resume.clone();
    resume1.setSex("女");
```

```
    resume1.getWe().setCompany("北京");//这时打印的 resume resume1的公司都是 北京
    //resume1.setWorkExperience("12-17","北京");//打印的 resume 的信息没有改变
    //打印resume ，resume1 信息
}
```

3 深克隆简单引用示例

```
public class Resume implements Cloneable {
    private String name;
    private String sex;
    private String age;
    private WorkExperience we;
    @Override
    protected Object clone() throws CloneNotSupportedException {
        Object o = super.clone();
        ((Resume)o).we = (WorkExperience) we.clone();//这个是必须的操作，不然克隆的还是引用
        return o;
    }
}
public class WorkExperience implements Cloneable{
    private String timeAArea;
    private String company;
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
public static void main(String[] args) throws CloneNotSupportedException {
    Resume resume = new Resume();
    resume.setAge("11");
    resume.setName("wk");
    resume.setSex("男");
    resume.setWe(new WorkExperience("12-16","南京"));
    Resume resume1 = (Resume) resume.clone();
    resume1.setSex("女");
    resume1.getWe().setCompany("北京");//这个操作不会修改原对象的对应的值，因为地址就不一样了
}
```

4 深克隆列表示例

- 克隆列表一定要先使列表的地址和原对象不一样，及 new 一个新的列表，在将原列表中的元素克隆进新列表

```
public class Resume0 implements Cloneable {
    private String name;
    private String sex;
    private String age;
    private List<WorkExperience> wks;
    @Override
    protected Object clone() throws CloneNotSupportedException {
        Resume0 o = (Resume0) super.clone();
        //列表没有克隆方法，只能 new 一个了
```

```
        //这一步也是关键，使得克隆对象 wks 的地址与原对象不同，没有这一步的话，因为地址相同，不管怎么修改地址里的值（这个值也可以是地址，新旧两个对象都会同时变化）
        o.wks = new ArrayList<>(wks.size());
        for (int i = 0,iMax = wks.size(); i < iMax; i++) {
            o.wks.add((WorkExperience) wks.get(i).clone());
        }
        return o;
    }
}
```

5 深克隆数组示例

- 数组可以直接 clone

```
public class Resume0 implements Cloneable {
    private String name;
    private String sex;
    private String age;
    private WorkExperience[] wks;
    @Override
    protected Object clone() throws CloneNotSupportedException {
        Resume0 o = (Resume0) super.clone();
        WorkExperience[] ss = new WorkExperience[wks.length];
        o.wks = wks.clone();//直接克隆数组，用system.arraycopy的话，copy 处理的数组中的元素和原数组的元素是同一个地址
        for (int i = 0,iMax = wks.length; i < iMax; i++) {
            o.wks[i] = (WorkExperience) wks[i].clone();
        }
        return o;
    }
}
```

6 序列化

- 当对象比较复杂时，可以直接通过序列化的方式来克隆对象
- 只要所有对象都实现了 Serializable 接口即可

```
@Test
public void 序列化() throws IOException, ClassNotFoundException {
    Resume0 resume = new Resume0();
    resume.setAge("11");
    resume.setName("wk");
    resume.setSex("男");
    WorkExperience[] workExperiences = {new WorkExperience("1-2","nanjing"),new WorkExperience("2-3","beijing")};
    resume.setWks(workExperiences);
    //序列化到 byteArrayOutputStream 流中
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    ObjectOutputStream objectOutputStream = new ObjectOutputStream(byteArrayOutputStream);
    objectOutputStream.writeObject(resume);
```

```
    //从 byteArrayOutputStream 流中 反序列化，得到克隆对象
    ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(byteArrayOutputStream.toByteArray());
    ObjectInputStream objectInputStream = new ObjectInputStream(byteArrayInputStream);
    Resume0 resume0 = (Resume0) objectInputStream.readObject();
    //改变克隆对象的值，原始对象值不变
    resume0.getWks()[0].setTimeAArea("11-22");
}
```

7 深度克隆总结

- 深度克隆核心就是改变所有引用对象的地址，引用对象的地址不一样时，随便怎么修改地址的值，都不会影响原对象

参考

大话设计模式——原型模式 Java-克隆数组 Java - 数组拷贝的几种方式 github 源码地址