简述

一个原型类，只需要实现Cloneable接口，覆写clone方法，就可以进行复制。（此处clone方法可以改成任意的名称，因为Cloneable接口是方法名，如cloneA或者cloneB，因为此处的重点是super.clone()这句话，super.clone()调用的是Object的clone()方法

- 浅复制：将一个对象复制后，基本数据类型的变量都会重新创建，而引用类型，指向的还是原对象所指向的。
- 深复制：将一个对象复制后，不论是基本数据类型还有引用类型，都是重新创建的。简单来说，就是深复制进行了完全彻底的复制，

代码

```
public class Person implements Cloneable, Serializable {

    private Car car;

    public Person(Car car){
        this.car = car;
    }

    /* 浅复制 */
     public Object clone() throws CloneNotSupportedException {
         Person proto = (Person) super.clone();
         return proto;
     }

     /* 深复制 */
     public Object deepClone() throws IOException, ClassNotFoundException {
         /* 写入当前对象的二进制流 */
         ByteArrayOutputStream bos = new ByteArrayOutputStream();
         ObjectOutputStream oos = new ObjectOutputStream(bos);
         oos.writeObject(this);

         /* 读出二进制流产生的新对象 */
         ByteArrayInputStream bis = new ByteArrayInputStream(bos.toByteArray());
         ObjectInputStream ois = new ObjectInputStream(bis);
         return ois.readObject();
     }

     public Car getCar() {
         return car;
     }

     public void setCar(Car car) {
         this.car = car;
     }
}


public class Car implements Serializable {

    private String name;

    public Car(String name){
```

```
43          this.name = name;
44      }
45
46      public String getName() {
47          return name;
48      }
49
50      public void setName(String name) {
51          this.name = name;
52      }
53  }
54
55
56  public class Main {
57
58      public static void main(String[] args) throws CloneNotSupportedException, IOException, ClassNot
59
60          Car car = new Car("大众");
61          Person p1 = new Person(car);
62          Person p2 = (Person) p1.clone();
63          Person p3 = (Person) p1.deepClone();
64
65
66          System.out.println("p1 car : " + p1.getCar().getName());
67          System.out.println("p2 car : " + p2.getCar().getName());
68          System.out.println("p3 car : " + p3.getCar().getName());
69
70
71          car.setName("本田");
72          System.out.println("p1 car : " + p1.getCar().getName());
73          System.out.println("p2 car : " + p2.getCar().getName());
74          System.out.println("p3 car : " + p3.getCar().getName());
75      }
76  }
77
78  /* 输出
79  p1 car : 大众
80  p2 car : 大众
81  p3 car : 大众
82  p1 car : 本田
83  p2 car : 本田
84  p3 car : 大众
85  */
```

总结

- 使用原型模式复制对象不会调用类的构造方法。因为对象的复制是通过调用Object类的clone方法来完成的，它直接在内存中复制数据
但构造方法中的代码不会执行，甚至连访问权限都对原型模式无效。单例模式中，只要将构造方法的访问权限设置为private型，就可以
造方法的权限，所以，单例模式与原型模式是冲突的，在使用时要特别注意

- 使用原型模式创建对象比直接new一个对象在性能上要好的多，因为Object类的clone方法是一个本地方法，它直接操作内存中的二进
差别非常明显