**反射**

**57. 什么是反射？**

反射主要是指程序可以访问、检测和修改它本身状态或行为的一种能力

Java反射：

在Java运行时环境中，对于任意一个类，能否知道这个类有哪些属性和方法？对于任意一个对象，能否调用它的任意一个方法

Java反射机制主要提供了以下功能：

* 在运行时判断任意一个对象所属的类。
* 在运行时构造任意一个类的对象。
* 在运行时判断任意一个类所具有的成员变量和方法。
* 在运行时调用任意一个对象的方法。

**58. 什么是 java 序列化？什么情况下需要序列化？**

简单说就是为了保存在内存中的各种对象的状态（也就是实例变量，不是方法），并且可以把保存的对象状态再读出来。虽然你可以用你自己的各种各样的方法来保存object states，但是Java给你提供一种应该比你自己好的保存对象状态的机制，那就是序列化。  
  
什么情况下需要序列化：

a）当你想把的内存中的对象状态保存到一个文件中或者数据库中时候；  
b）当你想用套接字在网络上传送对象的时候；  
c）当你想通过RMI传输对象的时候；

**59. 动态代理是什么？有哪些应用？**

动态代理：

当想要给实现了某个接口的类中的方法，加一些额外的处理。比如说加日志，加事务等。可以给这个类创建一个代理，故名思议就是创建一个新的类，这个类不仅包含原来类方法的功能，而且还在原来的基础上添加了额外处理的新类。这个代理类并不是定义好的，是动态生成的。具有解耦意义，灵活，扩展性强。

动态代理的应用：

* Spring的AOP
* 加事务
* 加权限
* 加日志

**60. 怎么实现动态代理？**

首先必须定义一个接口，还要有一个InvocationHandler(将实现接口的类的对象传递给它)处理类。再有一个工具类Proxy(习惯性将其称为代理类，因为调用他的newInstance()可以产生代理对象,其实他只是一个产生代理对象的工具类）。利用到InvocationHandler，拼接代理类源码，将其编译生成代理类的二进制码，利用加载器加载，并将其实例化产生代理对象，最后返回。

**对象拷贝**

**61. 为什么要使用克隆？**

想对一个对象进行处理，又想保留原有的数据进行接下来的操作，就需要克隆了，Java语言中克隆针对的是类的实例。

**62. 如何实现对象克隆？**

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，代码如下：

import java.io.ByteArrayInputStream;import java.io.ByteArrayOutputStream;import java.io.ObjectInputStream;import java.io.ObjectOutputStream;import java.io.Serializable;  
public class MyUtil {  
 private MyUtil() { throw new AssertionError(); }  
 @SuppressWarnings("unchecked") public static <T extends Serializable> T clone(T obj) throws Exception { ByteArrayOutputStream bout = new ByteArrayOutputStream(); ObjectOutputStream oos = new ObjectOutputStream(bout); oos.writeObject(obj);  
 ByteArrayInputStream bin = new ByteArrayInputStream(bout.toByteArray()); ObjectInputStream ois = new ObjectInputStream(bin); return (T) ois.readObject();  
 *// 说明：调用ByteArrayInputStream或ByteArrayOutputStream对象的close方法没有任何意义* *// 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源，这一点不同于对外部资源（如文件流）的释放* }}

下面是测试代码：

import java.io.Serializable;  
*/\*\* \* 人类 \* @author nnngu \* \*/*class Person implements Serializable { private static final long serialVersionUID = -9102017020286042305L;  
 private String name; *// 姓名* private int age; *// 年龄* private Car car; *// 座驾*  
 public Person(String name, int age, Car car) { this.name = name; this.age = age; this.car = car; }  
 public String getName() { return name; }  
 public void setName(String name) { this.name = name; }  
 public int getAge() { return age; }  
 public void setAge(int age) { this.age = age; }  
 public Car getCar() { return car; }  
 public void setCar(Car car) { this.car = car; }  
 @Override public String toString() { return "Person [name=" + name + ", age=" + age + ", car=" + car + "]"; }  
}

*/\*\* \* 小汽车类 \* @author nnngu \* \*/*class Car implements Serializable { private static final long serialVersionUID = -5713945027627603702L;  
 private String brand; *// 品牌* private int maxSpeed; *// 最高时速*  
 public Car(String brand, int maxSpeed) { this.brand = brand; this.maxSpeed = maxSpeed; }  
 public String getBrand() { return brand; }  
 public void setBrand(String brand) { this.brand = brand; }  
 public int getMaxSpeed() { return maxSpeed; }  
 public void setMaxSpeed(int maxSpeed) { this.maxSpeed = maxSpeed; }  
 @Override public String toString() { return "Car [brand=" + brand + ", maxSpeed=" + maxSpeed + "]"; }  
}

class CloneTest {  
 public static void main(String[] args) { try { Person p1 = new Person("郭靖", 33, new Car("Benz", 300)); Person p2 = MyUtil.clone(p1); *// 深度克隆* p2.getCar().setBrand("BYD"); *// 修改克隆的Person对象p2关联的汽车对象的品牌属性* *// 原来的Person对象p1关联的汽车不会受到任何影响* *// 因为在克隆Person对象时其关联的汽车对象也被克隆了* System.out.println(p1); } catch (Exception e) { e.printStackTrace(); } }}

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，这项检查是编译器完成的，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好过把问题留到运行时。

**63. 深拷贝和浅拷贝区别是什么？**

* 浅拷贝只是复制了对象的引用地址，两个对象指向同一个内存地址，所以修改其中任意的值，另一个值都会随之变化，这就是浅拷贝（例：assign()）
* 深拷贝是将对象及值复制过来，两个对象修改其中任意的值另一个值不会改变，这就是深拷贝（例：JSON.parse()和JSON.stringify()，但是此方法无法复制函数类型）