Java中的Externalizable接口

1、知识回顾-Java序列化

Java序列化是指把Java对象转换为字节序列的过程，Java反序列化是指把字节序列恢复为Java对象的过程。通过徐序列化和反序列化实现网络传输、本地存储的目的。

1.1 Serializable实现Java序列化

要实现Java对象的序列化，只要**将类实现标识接口——Serializable接口**即可，不需要我们重写任何方法就可以实现序列化。

> 案例：Java实现Serializable接口进行序列化案例。

1.1.1 编写实体类

```
package com.kkb.pojo;
import java.io.Serializable;
import java.util.List;

/**
 * 学生实体类
 */
public class Student implements Serializable{
    /**
     * 学号
     */
    private String stuNum;
    /**
     * 姓名
     */
    private String stuName;
    /**
     * 教师姓名：一个学生可以有多个老师
     */
    private List<String> teacherList;

    //无参数构造方法
    public Student() {
    }

    //全参构造方法
    public Student(String stuNum, String stuName, List<String> teacherList) {
        this.stuNum = stuNum;
        this.stuName = stuName;
        this.teacherList = teacherList;
    }

    @Override
    public String toString() {
        return "Student{" +
                "stuNum='" + stuNum + '\'' +
```

```
                ", stuName='" + stuName + '\'' +
                ", teacherList=" + teacherList +
                '}';
    }

    public String getStuNum() {
        return stuNum;
    }

    public void setStuNum(String stuNum) {
        this.stuNum = stuNum;
    }

    public String getStuName() {
        return stuName;
    }

    public void setStuName(String stuName) {
        this.stuName = stuName;
    }

    public List<String> getTeacherList() {
        return teacherList;
    }

    public void setTeacherList(List<String> teacherList) {
        this.teacherList = teacherList;
    }
}
```

1.1.2 编写Java对象序列化和反序列化工具类

```
package com.kkb.util;
import java.io.*;
/**
 * 序列化和反序列化的工具类
 */
public class MySerializeUtil {

    /**
     * 将对象序列化到指定文件中
     * @param obj
     * @param fileName
     */
    public static void mySerialize(Object obj,String fileName) throws
IOException {
        OutputStream out=new FileOutputStream(fileName);
        ObjectOutputStream objOut=new ObjectOutputStream(out);
        objOut.writeObject(obj);
        objOut.close();
    }

    /**
     * 从指定文件中反序列化对象
     * @param fileName
     * @return
```

```
     */
    public static Object myDeserialize(String fileName) throws IOException,
ClassNotFoundException {
        InputStream in=new FileInputStream(fileName);
        ObjectInputStream objIn=new ObjectInputStream(in);
        Object obj=objIn.readObject();
        return obj;
    }
}
```

1.1.3 测试对象的序列化和反序列化

```
package com.kkb.test;
import com.kkb.pojo.Student;
import com.kkb.util.MySerializeUtil;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

/**
 * 测试类
 */
public class MainTest {

    public static void main(String[] args) {
        List<String> teacherList=new ArrayList<>();
        teacherList.add("空空道人");
        teacherList.add("贾代儒");
        Student stu1=new Student("1001", "贾宝玉", teacherList);
        System.out.println("原始对象："+stu1);

        String fileName="stu01.txt";
        try {
            //对象序列化
            MySerializeUtil.mySerialize(stu1, fileName);
            System.out.println("序列化原始对象完成！OK！");
            //对象的反序列化
            Object obj=MySerializeUtil.myDeserialize(fileName);
            if(obj instanceof  Student){
                Student stuNew= (Student) obj;
                System.out.println("反序列化之后的对象："+stuNew);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行结果：

```
Tests passed: 1 of 1 test – 31 ms
"C:\Program Files\Java\jdk1.8.0_66\bin\java.exe" ...
原始对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=[空空道人, 贾代儒]}
序列化原始对象完成！OK！
反序列化之后的对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=[空空道人, 贾代儒]}

Process finished with exit code 0
```

1.2 部分属性的序列化

实现部分字段序列化的方式：

- 使用transient修饰符
- 使用static修饰符
- 默认方法writeObject和readObject。
- 卖个关子，稍后讲解。

1.2.1 使用transient修饰符

修改实体类，将实体类中不想序列化的属性添加transient修饰词。

```
public class Student implements Externalizable {
    private String stuNum;
    private transient String stuName;
    private transient List<String> teacherList;
    ......
```

重新运行测试类的结果：

我们将实体类中的stuName和teacherList属性添加了transient修饰词，因此对象被序列化的时候忽略这两个属性。通过运行结果可以看出。

```
Tests passed: 1 of 1 test – 20 ms
"C:\Program Files\Java\jdk1.8.0_66\bin\java.exe" ...
原始对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=[空空道人, 贾代儒]}
序列化原始对象完成！OK！
反序列化之后的对象：Student{stuNum='1001', stuName='null', teacherList=null}
```

1.2.2 使用static修饰符

static修饰符修饰的属性也不会参与序列化和反序列化。

修改实体类，将实体类中不想序列化的属性添加static修饰词。

```
public class Student implements Externalizable {
    private String stuNum;
    private static String stuName;
    private List<String> teacherList;
    ......
```

修改测试类，在完成原始对象的序列化之后再对static修饰的变量进行一次赋值操作：

```
public class MainTest {

    public static void main(String[] args) {
        List<String> teacherList=new ArrayList<>();
        teacherList.add("空空道人");
        teacherList.add("贾代儒");
        Student stu1=new Student( stuNum: "1001",  stuName: "贾宝玉", teacherList);
        System.out.println("原始对象: "+stu1);

        String fileName="stu01.txt";
        try {
            //对象序列化
            MySerializeUtil.mySerialize(stu1, fileName);
            System.out.println("序列化对象完毕! OK! ");
            stu1.setStuName("薛宝钗");//在序列化后再对static修饰的变量进行一次赋值操作
            //对象的反序列化
            Object obj=MySerializeUtil.myDeserialize(fileName);
            if(obj instanceof  Student){
                Student stuNew= (Student) obj;
                System.out.println("反序列化之后的对象: "+stuNew);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

重新运行测试类的结果：

我们将实体类中的stuName属性添加了transient修饰词，因此对象被序列化的时候忽略这个属性。通过运行结果可以看出。

```
"C:\Program Files\Java\jdk1.8.0_66\bin\java.exe" ...
原始对象: Student{stuNum='1001', stuName='贾宝玉', teacherList=[空空道人, 贾代儒]}
序列化对象完毕! OK!
反序列化之后的对象: Student{stuNum='1001', stuName='薛宝钗', teacherList=[空空道人, 贾代儒]}

Process finished with exit code 0
```

1.2.3 默认方法writeObject和readObject

修改实体类，将transient修饰词去掉，添加两个方法。

```
public class Student implements Serializable {
    private String stuNum;
    private String stuName;
    private List<String> teacherList;

    private void writeObject(ObjectOutputStream objOut) throws IOException {
        System.out.println("writeObject-----------");
        objOut.writeObject(stuNum);
        objOut.writeObject(stuName);
    }
    private void readObject(ObjectInputStream objIn) throws IOException,
ClassNotFoundException {
        System.out.println("readObject-----------");
        stuNum= (String) objIn.readObject();
        stuName= (String) objIn.readObject();
    }
```

```
    .....
```

重新运行测试类的结果：

我们在添加的方法中只对stuNum和stuName属性做了序列化和反序列化的操作，因此只有这个两个属性可以被序列化和反序列化。

```
Tests passed: 1 of 1 test – 25 ms
"C:\Program Files\Java\jdk1.8.0_66\bin\java.exe" ...
原始对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=[空空道人, 贾代儒]}
writeObject-----------
序列化原始对象完成！OK！
readObject-----------
反序列化之后的对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=null}
```

1.2.3.1 源码分析：

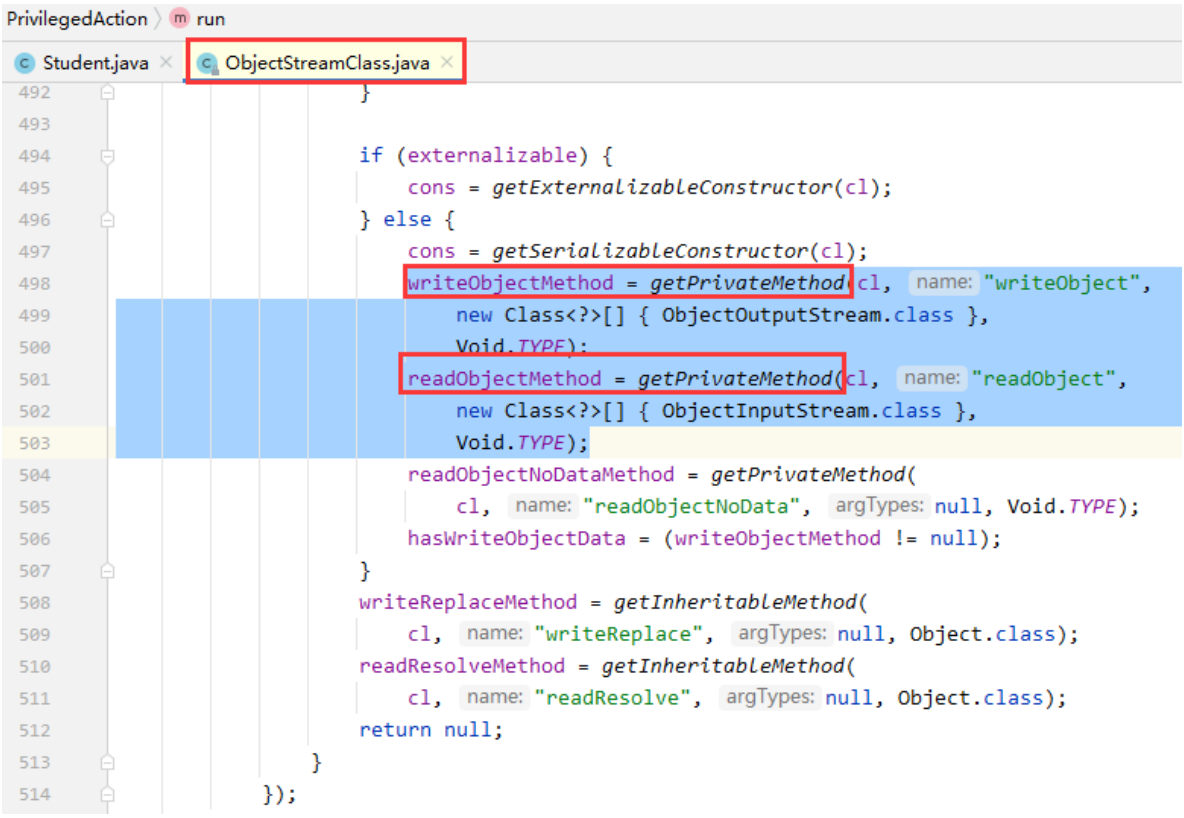
> 注意：**添加的两个方法必须是private void，否则不生效。**
>
> Java调用ObjectOutputStream类检查其是否有私有的、无返回值的writeObject方法，如果有，其会委托该方法进行对象序列化。

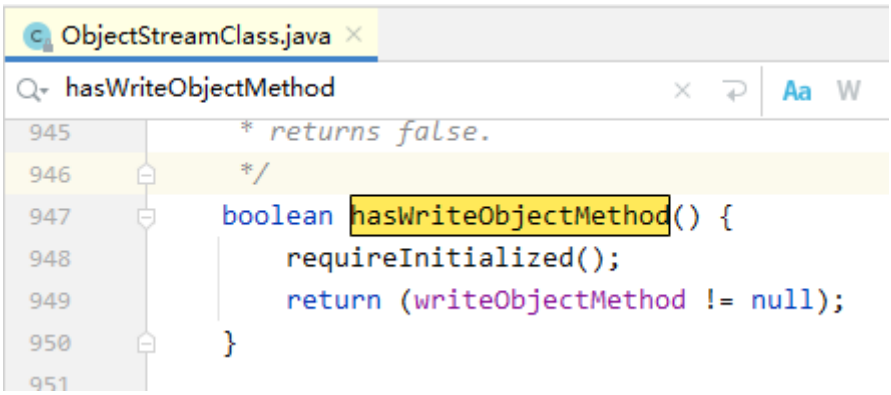
Serializable接口的注释：

Serializable.java

```
60      * <PRE>
61      * private void writeObject(java.io.ObjectOutputStream out)
62      *     throws IOException
63      * private void readObject(java.io.ObjectInputStream in)
64      *     throws IOException, ClassNotFoundException;
65      * private void readObjectNoData()
66      *     throws ObjectStreamException;
67      * </PRE>
68      *
69      * <p>The writeObject method is responsible for writing the state of the
70      * object for its particular class so that the corresponding
71      * readObject method can restore it.  The default mechanism for saving
72      * the Object's fields can be invoked by calling
73      * out.defaultWriteObject. The method does not need to concern
74      * itself with the state belonging to its superclasses or subclasses.
75      * State is saved by writing the individual fields to the
76      * ObjectOutputStream using the writeObject method or by using the
77      * methods for primitive data types supported by DataOutput.
78      *
79      * <p>The readObject method is responsible for reading from the stream and
80      * restoring the classes fields. It may call in.defaultReadObject to invoke
81      * the default mechanism for restoring the object's non-static and
82      * non-transient fields.  The defaultReadObject method uses information in
83      * the stream to assign the fields of the object saved in the stream with the
84      * correspondingly named fields in the current object.  This handles the case
85      * when the class has evolved to add new fields. The method does not need to
86      * concern itself with the state belonging to its superclasses or subclasses.
87      * State is saved by writing the individual fields to the
88      * ObjectOutputStream using the writeObject method or by using the
89      * methods for primitive data types supported by DataOutput.
90      *
```

ObjectStreamClass类：在序列化（反序列化）的时候，ObjectOutputStream（ObjectInputStream）会寻找目标类中的私有的writeObject（readObject）方法，赋值给变量writeObjectMethod（readObjectMethod）。





通过上面这两段代码可以知道，如果writeObjectMethod != null（目标类中定义了私有的writeObject方法），那么将调用目标类中的writeObject方法，如果如果writeObjectMethod == null，那么将调用默认的defaultWriteFields方法来读取目标类中的属性。

readObject的调用逻辑和writeObject一样。

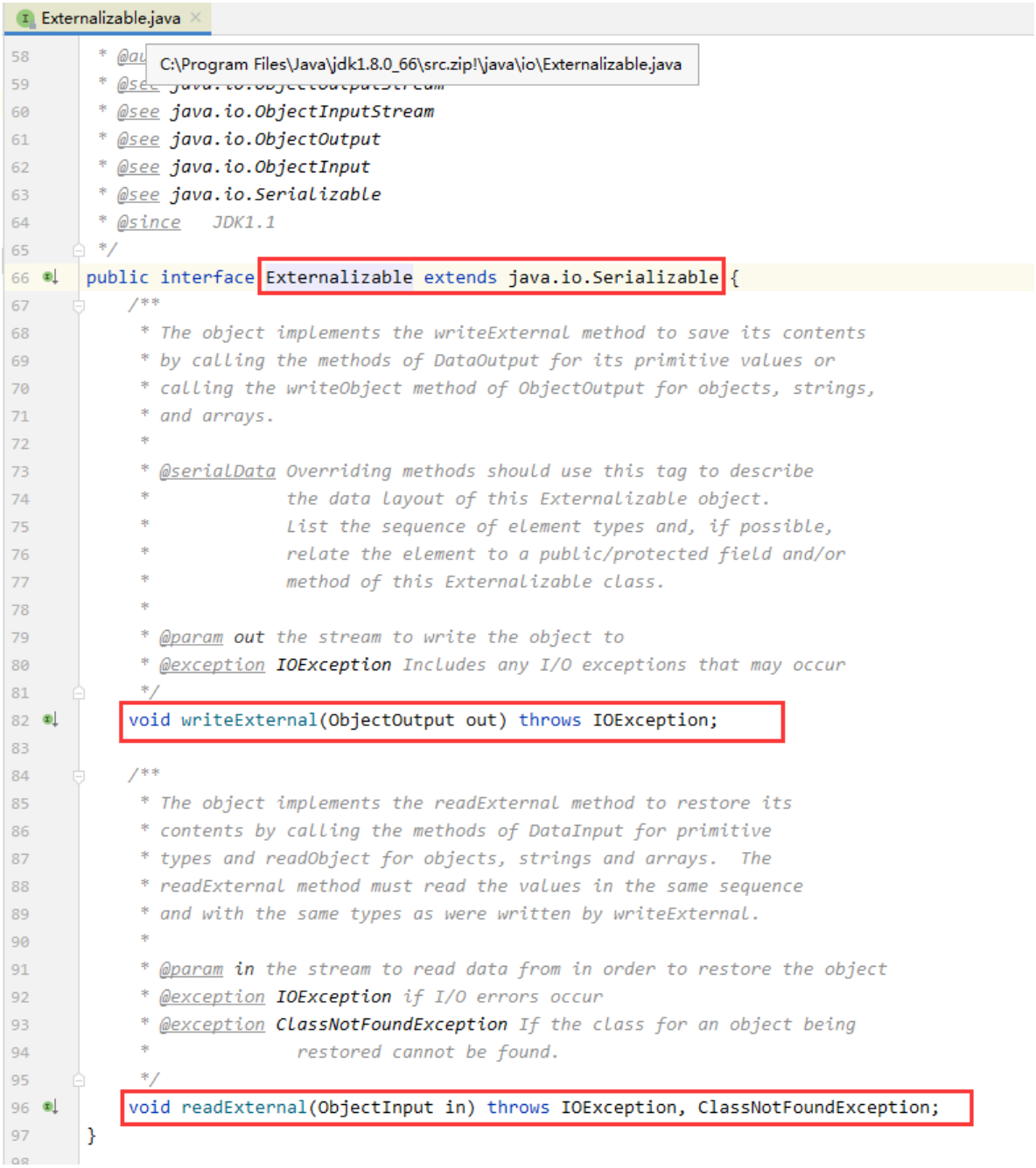
总结一下，如果目标类中没有定义私有的writeObject或readObject方法，那么序列化和反序列化的时候将调用默认的方法来根据目标类中的属性来进行序列化和反序列化，而如果目标类中定义了私有的writeObject或readObject方法，那么序列化和反序列化的时候将调用目标类指定的writeObject或readObject方法来实现。

2、Externalizable实现Java序列化

刚刚我们说实现部分属性序列化的方式有多种，最后一种来啦！就是通过实现Eexternalizable接口。

Externalizable继承自Serializable，使用Externalizable接口需要实现readExternal方法和writeExternal方法来实现序列化和反序列化。

来看看Externalizable接口的说明：



代码实现：

```
package com.kkb.pojo;

import java.io.*;
import java.util.List;

/**
 * @Author wanglina
 * @Version 1.0
 */
public class Student implements Externalizable {
    /**
     * 学号
     */
    private String stuNum;
    /**
     * 姓名
     */
    private String stuName;
```

```
    /**
     * 教师姓名：一个学生可以有多个老师
     */
    private List<String> teacherList;

    //无参数构造方法
    public Student() {
    }

    //全参构造方法
    public Student(String stuNum, String stuName, List<String> teacherList) {
        this.stuNum = stuNum;
        this.stuName = stuName;
        this.teacherList = teacherList;
    }

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(stuNum);
        out.writeObject(stuName);
        //out.writeObject(teacherList);
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
        stuNum= (String) in.readObject();
        stuName= (String) in.readObject();
       //teacherList= (List<String>) in.readObject();
    }

    @Override
    public String toString() {
        return "Student{" +
                "stuNum='" + stuNum + '\'' +
                ", stuName='" + stuName + '\'' +
                ", teacherList=" + teacherList +
                '}';
    }


    public String getStuNum() {
        return stuNum;
    }

    public void setStuNum(String stuNum) {
        this.stuNum = stuNum;
    }

    public String getStuName() {
        return stuName;
    }

    public void setStuName(String stuName) {
        this.stuName = stuName;
    }

    public List<String> getTeacherList() {
```

```
        return teacherList;
    }

    public void setTeacherList(List<String> teacherList) {
        this.teacherList = teacherList;
    }
}
```

测试结果：

```
Tests passed: 1 of 1 test – 69 ms
"C:\Program Files\Java\jdk1.8.0_66\bin\java.exe" ...
原始对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=[空空道人, 贾代儒]}
序列化原始对象完成！OK！
反序列化之后的对象：Student{stuNum='1001', stuName='贾宝玉', teacherList=null}
```

Externalizable接口继承了Serializable接口，所以实现Externalizable接口也能实现序列化和反序列化。
Externalizable接口中定义了writeExternal和readExternal两个抽象方法，这两个方法其实对应Serializable接口的writeObject和readObject方法。可以这样理解：Externalizable接口被设计出来的目的就是为了抽象出writeObject和readObject这两个方法，但是目前这个接口使用的并不多。

3、Serializable VS Externalizable

区 别	Serializable	Externalizable
实现复杂度	实现简单，Java对其有内建支持	实现复杂，由开发人员自己完成
执行效率	所有对象由Java统一保存，性能较低	开发人员决定哪个对象保存，可能造成速度提升
保存信息	保存时占用空间大	部分存储，可能造成空间减少
使用频率	高	偏低