java序列化与反序列化

定义

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。

应用场景

当两个进程进行远程通信时，可以相互发送各种类型的数据，包括文本、图片、音频、视频等， 而这些数据都会以二进制序列的形式在网络上传送。那么当两个Java进程进行通信时，如何实现进程间的对象传送呢？这就需要Java序列化与反序列化了。一方面，发送方需要把这个Java对象转换为字节序列，然后在网络上传送；另一方面，接收方需要从字节序列中恢复出Java对象。数据传输便是序列化与反序列化的主要应用场景之一。当然也可用于数据的存储与读取。

实现方式

java的序列化有两种方式：

1. 实现序列化接口Serializable，这个使用的比较多。Serializable接口是一个空的接口，它的主要作用就是标识这个类的对象是可序列化的。
2. 实现接口Externalizable。Exterinable继承了Serializable，是Serializable的一个扩展，对于哪些属性可以序列化，哪些可以反序列化可以做详细地约束。

相关工具类：

1. java.io.ObjectOutputStream：表示对象输出流

   它是OutputStream类的一个子类，对应的ObjectOutputStream.WriteObject(Object object)就要求参数object实现Serializable接口。

   使用方式如下：

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;

String fileName = "test.txt";  //文件名
LoginInfo info = new LoginInfo("chen","123"); //某个待序列化对象，LoginInfo类须实现Serializable接口
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName)); //创建输出流
oos.writeObject(info);   //序列化
oos.close();   //关闭流
```

2. java.io.ObjectInputStream：表示对象输入流

   相应地，它的readObject(Object object)方法从输入流中读取字节序列，再把它们反序列化成为一个对象，并返回。

   使用方式如下：

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;

String fileName = "test.txt";
ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName));
LoginInfo info2 = (LoginInfo)ois.readObject();
ois.close();
```

方式一：实现Serializable接口

Serializable源码

```
/*
 * Copyright (c) 1996, 2005, Oracle and/or its affiliates. All rights reserved.
 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 */
package java.io;
/**
 * Serializability of a class is enabled by the class implementing the
 * java.io.Serializable interface. Classes that do not implement this
 * interface will not have any of their state serialized or
 * deserialized.  All subtypes of a serializable class are themselves
 * serializable.  The serialization interface has no methods or fields
 * and serves only to identify the semantics of being serializable.
 *
 * @author  unascribed
 * @see java.io.ObjectOutputStream
 * @see java.io.ObjectInputStream
 * @see java.io.ObjectOutput
 * @see java.io.ObjectInput
 * @see java.io.Externalizable
 * @since   JDK1.1
 */
public interface Serializable {
}
```

中间还省略了很多注释，再忽略上面留下的注释，发现这个接口是空的！没有字段，没有方法，只是标识一个类的对象是否可序列化（实现了这个接口，就表示可以序列化）。

serialVersionUID的作用

实现Serializable接口前后并没有增加新方法，只是多了一个serialVersionUID，其实这个字段也不是必须的。若不写serialVersionUID。程序也能运行成功，不过eclipe会显示警告。

其实，Java的序列化机制是通过在运行时判断类的serialVersionUID来验证版本一致性的。在进行反序列化时，JVM会把传来的字节流中的serialVersionUID与本地相应实体类的serialVersionUID进行比较，如果相同就认为是一致的，可以进行反序列化，否则就会出现序列化版本不一致的异常(InvalidCastException)。

所以最好有这个字段，那么要如何添加呢？鼠标移动到警告处，eclipse在给出警告的同时也给出了解决方法：

1. 添加默认值，即1L;
2. 添加自动产生的ID值，我的是8685376332791485990L;（推荐）
3. 通过@SuppressWarnings 批注取消特定代码段（即，类或方法）中的警告。

java实现

```
package serialize;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Date;

public class LoginInfo implements Serializable{
    private static final long serialVersionUID = 8685376332791485990L;
    private String username;
    private String password;
    private Date logindate;
    public LoginInfo(){
        System.out.println("non-parameter constructor");
    }
    public LoginInfo(String username, String password){
        System.out.println("parameter constructor");
        this.username = username;
        this.password = password;
        this.logindate = new Date();
    }
    public String toString(){
        return
"username="+username+",password="+password+",logindate="+logindate;
    }
    public static void main(String[] args) throws Exception{
        LoginInfo info = new LoginInfo("chen","123");
        System.out.println(info);
        String fileName = "info_serializable.txt";
        System.out.println("Serialize object");
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName));
        oos.writeObject(info);
        oos.close();
        System.out.println("Deserialize object");
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName));
        LoginInfo info2 = (LoginInfo)ois.readObject();
        ois.close();
        System.out.println(info2);
    }
}
```

运行结果：

```
parameter constructor
username=chen,password=123,logindate=Sun Feb 19 09:33:38 CST 2017
Serialize object
Deserialize object
username=chen,password=123,logindate=Sun Feb 19 09:33:38 CST 2017
```

之后会在项目目录下生成文件info_serializable.txt，虽然存成了txt格式，但并不能使用普通文本格式打开，会出现乱码。

transicent的作用

如果，不希望某些字段被序列化，如LoginInfo中的username，只需在对应字段的定义时使用关键词transient：

```
// private String password;
private transient String password;
```

再次运行程序，结果如下：

```
parameter constructor
username=chen,password=123,logindate=Sun Feb 19 09:50:34 CST 2017
Serialize object
Deserialize object
username=chen,password=null,logindate=Sun Feb 19 09:50:34 CST 2017
```

password反序列化后的结果为null，达到了保护密码的目的。

方式二：实现Externalizable接口

Externalizable源码

```
/*
 * Copyright (c) 1996, 2004, Oracle and/or its affiliates. All rights reserved.
 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 */

package java.io;
import java.io.ObjectOutput;
import java.io.ObjectInput;

/**
 * @author  unascribed
 * @see java.io.ObjectOutputStream
 * @see java.io.ObjectInputStream
 * @see java.io.ObjectOutput
 * @see java.io.ObjectInput
 * @see java.io.Serializable
```

```
 * @since   JDK1.1
 */
public interface Externalizable extends java.io.Serializable {
    /**
     * @param out the stream to write the object to
     * @exception IOException Includes any I/O exceptions that may occur
     */
    void writeExternal(ObjectOutput out) throws IOException;
    /**
     * @param in the stream to read data from in order to restore the object
     * @exception IOException if I/O errors occur
     * @exception ClassNotFoundException If the class for an object being
     *              restored cannot be found.
     */
    void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException;
}
```

Externalizable继承了Serializable接口，并添加了两个新的方法，分别表示在哪些字段能被序列化和哪些字段能够反序列化。

java实现

erialVersionUID和之前一样，最好添加。（但不添加的话，eclipse竟然连警告都没有！！无奈，只能自己随便写个数了。）

```
package serialize;

import java.io.Externalizable;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.util.Date;

public class LoginInfo2 implements Externalizable{
    private static final long serialVersionUID = 4297291454171868241L;
    private String username;
    private String password;
    private Date logindate;
    public LoginInfo2(){
        System.out.println("non-parameter constructor");
    }
    public LoginInfo2(String username, String password){
        System.out.println("parameter constructor");
        this.username = username;
        this.password = password;
        this.logindate = new Date();
    }
    public String toString(){
        return "username="+username+
                ",password="+password+
```

```
                ",logindate="+logindate;
    }
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(logindate);
        out.writeUTF(username);

    }
    @Override
    public void readExternal(ObjectInput in) throws IOException,
            ClassNotFoundException {
        logindate = (Date)in.readObject();
        username = (String)in.readUTF();
    }

    public static void main(String[] args) throws Exception{
        LoginInfo2 info = new LoginInfo2("chen","123");
        System.out.println(info);
        String fileName = "info_externalizable.txt";

        System.out.println("Serialize object");
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName));
        oos.writeObject(info);
        oos.close();
        System.out.println("Deserialize object");
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName));
        LoginInfo2 info2 = (LoginInfo2)ois.readObject();
        ois.close();
        System.out.println(info2);
    }
}
```

运行结果：

```
parameter constructor
username=chen,password=123,logindate=Sun Feb 19 10:53:57 CST 2017
Serialize object
Deserialize object
non-parameter constructor
username=chen,password=null,logindate=Sun Feb 19 10:53:57 CST 2017
```

实现了和Serializable+transient同样的目的。

无参构造函数

仔细分析运行结果，发现这种方式反序列化的时候竟然调用了无参构造函数。对于恢复Serializable对象，完全以它存储的二进制为基础来构造，而不调用构造函数。而对于一个Externalizable对象，public的无参构造函数将会被调用。如果没有public的无参构造函数，运行时会报异常(Invalid Class Exception : LoginInfo2 ; no valid constructor...),之后会调用readExternal 读取数据。源码的注释中也做了如下说明：

> Object Serialization uses the Serializable and Externalizable interfaces. Object persistence mechanisms can use them as well. Each object to be stored is tested for the Externalizable interface.
>
> - If the object supports Externalizable, the writeExternal method is called. If the object does not support Externalizable and does implement Serializable, the object is saved using ObjectOutputStream.
> - When an Externalizable object is reconstructed, an instance is created using the public no-arg constructor, then the readExternal method called. Serializable objects are restored by reading them from an ObjectInputStream.

transcient还有效果吗

对于externalizable实现方式的代码做如下修改：

```
// private Date logindate;
private transient Date logindate;
```

运行结果与代码改动之前一样，说明transcient在externalizable实现的类中失效了

总结

1. 两个流：objectOutputStream、ObjectInputStream

```
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName));
oos.writeObject(info);
oos.close();

ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName));
LoginInfo info2 = (LoginInfo)ois.readObject();
ois.close();
```

2. serialVersionUID的作用

   Ja**va的序列化机制是通过在运行时判断类的serialVersionUID来验证版本一致性的，所以实现Serializable或Externalizable接口都最好有serialVersionUID这一字段，没有会报警告。**

3. Serializable与Externalizable的关系

   Serializable是个空接口，用于指示某类的对象是否可以序列化。Externalizable接口继承了Serializable接口，添加了writeExternal、readExternal方法。

4. transient关键字

   在Serializable接口实现的类中，transient修饰的字段不参与序列化过程；在Externalizable接口实现的类中，transient无效。即：transient只能与Serializable搭配使用。

```
public class LoginInfo implements Serializable{
 private static final long serialVersionUID = 8685376332791485990L;
 private String username;
 private transient String password; //不参与序列化过程
     ......
}
```

5. writeExternal、readExternal 与无参构造函数

   Externalizable实现的类对象，序列化与反序列化由writeExternal与readExternal两个函数控制（内部操作的字段要对应），而且反序列时会先调用无参构造函数创建实例对象，再通过readExternal读取数据。所以Externalizable实现的类若想反序列化，必须有无参构造函数。而恢复Serializable对象，完全以之前存储的二进制为基础来构造，不调用构造函数。

来自： https://www.jianshu.com/p/e554c787c286