1. transient的作用及使用方法

我们都知道一个对象只要实现了Serilizable接口，这个对象就可以被序列化，java的这种序列化模式为开发者提供了很多便利，我们可以不必关系具体序列化的过程，只要这个类实现了Serilizable接口，这个类的所有属性和方法都会自动序列化。

然而在实际开发过程中，我们常常会遇到这样的问题，这个类的有些属性需要序列化，而其他属性不需要被序列化，打个比方，如果一个用户有一些敏感信息（如密码，银行卡号等），为了安全起见，不希望在网络操作（主要涉及到序列化操作，本地序列化缓存也适用）中被传输，这些信息对应的变量就可以加上transient关键字。换句话说，这个字段的生命周期仅存于调用者的内存中而不会写到磁盘里持久化。

总之，java 的transient关键字为我们提供了便利，你只需要实现Serilizable接口，将不需要序列化的属性前添加关键字transient，序列化对象的时候，这个属性就不会序列化到指定的目的地中。

示例code如下：

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

/**
 * @description 使用transient关键字不序列化某个变量
 *      注意读取的时候，读取数据的顺序一定要和存放数据的顺序保持一致
 *
 */
public class TransientTest {

    public static void main(String[] args) {

        User user = new User();
        user.setUsername("Alexia");
        user.setPasswd("123456");

        System.out.println("read before Serializable: ");
        System.out.println("username: " + user.getUsername());
        System.err.println("password: " + user.getPasswd());
```

```
        try {
            ObjectOutputStream os = new ObjectOutputStream(
                    new FileOutputStream("C:/user.txt"));
            os.writeObject(user); // 将User对象写进文件
            os.flush();
            os.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            ObjectInputStream is = new ObjectInputStream(new FileInputStream(
                    "C:/user.txt"));
            user = (User) is.readObject(); // 从流中读取User的数据
            is.close();

            System.out.println("\nread after Serializable: ");
            System.out.println("username: " + user.getUsername());
            System.err.println("password: " + user.getPasswd());

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

class User implements Serializable {
    private static final long serialVersionUID = 8294180014912103005L;

    private String username;
    private transient String passwd;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPasswd() {
        return passwd;
```

```
    }

    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }

}
```

输出为：

```
read before Serializable:
username: Linxijun
password: 123456

read after Serializable:
username: Linxijun
password: null
```

密码字段为null，说明反序列化时根本没有从文件中获取到信息。

2. transient使用小结

1）一旦变量被transient修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。

2）transient关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被transient关键字修饰的。变量如果是用户自定义类变量，则该类需要实现Serializable接口。

3）被transient关键字修饰的变量不再能被序列化，一个静态变量不管是否被transient修饰，均不能被序列化。

第三点可能有些人很迷惑，因为发现在User类中的username字段前加上static关键字后，程序运行结果依然不变，即static类型的username也读出来为“Linxijun”了，这不与第三点说的矛盾吗？实际上是这样的：第三点确实没错（一个静态变量不管是否被transient修饰，均不能被序列化），反序列化后类中static型变量username的值为当前JVM中对应static变量的值，这个值是JVM中的不是反序列化得出的，不相信？好吧，下面我来证明：

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

/**
 * @description 使用transient关键字不序列化某个变量
 *       注意读取的时候，读取数据的顺序一定要和存放数据的顺序保持一致
 *
 */
public class TransientTest {

    public static void main(String[] args) {

        User user = new User();
        user.setUsername("Alexia");
        user.setPasswd("123456");

        System.out.println("read before Serializable: ");
        System.out.println("username: " + user.getUsername());
        System.err.println("password: " + user.getPasswd());

        try {
            ObjectOutputStream os = new ObjectOutputStream(
                    new FileOutputStream("C:/user.txt"));
            os.writeObject(user); // 将User对象写进文件
            os.flush();
            os.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            // 在反序列化之前改变username的值
            User.username = "newName";

            ObjectInputStream is = new ObjectInputStream(new FileInputStream(
                    "C:/user.txt"));
            user = (User) is.readObject(); // 从流中读取User的数据
            is.close();

            System.out.println("\nread after Serializable: ");
            System.out.println("username: " + user.getUsername());
            System.err.println("password: " + user.getPasswd());

        } catch (FileNotFoundException e) {
            e.printStackTrace();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

class User implements Serializable {
    private static final long serialVersionUID = 8294180014912103005L;

    public static String username;
    private transient String passwd;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPasswd() {
        return passwd;
    }

    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }

}
```

运行结果为：

```
read before Serializable:
username: Linxijun
password: 123456

read after Serializable:
username: newName
password: null
```

这说明反序列化后类中static型变量username的值为当前JVM中对应static变量的值，为修改后newName，而不是序列化时的值Linxijun。

3. transient使用细节——被transient关键字修饰的变量真的不能被序列化吗?

思考下面的例子：

```
import java.io.Externalizable;
import java.io.File;
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;

/**
 * @descripiton Externalizable接口的使用
 *
 *
 */
public class ExternalizableTest implements Externalizable {

    private transient String content = "是的，我将会被序列化，不管我是否被transient关
键字修饰";

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(content);
    }

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

    public static void main(String[] args) throws Exception {

        ExternalizableTest et = new ExternalizableTest();
        ObjectOutput out = new ObjectOutputStream(new FileOutputStream(
                new File("test")));
        out.writeObject(et);

        ObjectInput in = new ObjectInputStream(new FileInputStream(new File(
                "test")));
        et = (ExternalizableTest) in.readObject();
        System.out.println(et.content);

        out.close();
        in.close();
```

```
    }
}
```

content变量会被序列化吗？是的，运行结果就是：

是的，我将会被序列化，不管我是否被transient关键字修饰

这是为什么呢，不是说类的变量被transient关键字修饰以后将不能序列化了吗？

我们知道在Java中，对象的序列化可以通过实现两种接口来实现，若实现的是Serializable接口，则所有的序列化将会自动进行，若实现的是Externalizable接口，则没有任何东西可以自动序列化，需要在writeExternal方法中进行手工指定所要序列化的变量，这与是否被transient修饰无关。因此第二个例子输出的是变量content初始化的内容，而不是null。