



对象流(序列化和反序列化)

序列化和反序列化

序 列 化：指把内存中的Java对象数据，通过某种方式把对象存储到磁盘文件中或者传递给其他的节点(在网络上传输)。

我们把整个过程称之为序列化。

反序列化：把磁盘文件中的对象数据或者把网络节点上的对象数据，恢复成Java对象的过程。

为什么要进行序列化

1)在分布式系统中，需要共享的数据的JavaBean对象，得使用序列化。此时需要把对象再网络上传输，此时就得把对象数据转换为二进制形式。

以后在 HttpSession 中的对象，都必须实现序列化接口的类才能做序列化操作。

2)服务器化：如果服务器发现某些对象太久都没有活动了，此时服务器就会把这些对象中的对象持久化在本地磁盘文件中(Java对象->二进制文件)。

如果某些对象需要活动的时候，现在内存中去寻找找不到，找到再读取磁盘文件中，反序列化我们得对象数据，恢复成Java对象。

需要被序列化的对象的类，必须实现序列化接口`java.io.Serializable`接口(标志接口)[没有抽象方法]。

底层会处理，如果当前对象是Serializable的实现类，允许做序列化。boolean ret = Java对象 instanceof Serializable;

在Java中大多数类都已经实现`Serializable`接口。

使用对象流来完成序列化和反序列化操作：

`ObjectOutputStream`：通过`writeObject()`方法做序列化操作的。
`ObjectInputStream`：通过`readObject()`方法反序列化操作的。

`Exception in thread "main" java.io.NotSerializableException: cn.wolfcode.day03_01_object.User`

此时错误：User类没有实现序列化接口`java.io.Serializable`。

反序列化操作必须存在对象的节制对象。

//使用对象流来实现序列化和反序列化

public class ObjectOutputStreamDemo {
 public static void main(String[] args) throws Exception {
 File f = new File("file/obj.txt");
 //写出对象流
 FileOutputStream out = new ObjectOutputStream(new FileOutputStream(f));
 User user = new User("will", "1234", 17);
 out.writeObject(user);
 out.close();
 }
}

反序列化操作

private static void readObject(File f) throws Exception {
 ObjectInputStream in = new ObjectInputStream(new FileInputStream(f));

User user = (User) in.readObject();

System.out.println(user);

in.close();
}

序列化的细节操作的版本:

1)如果某些数据不需要序列化，比如`password`，此时怎么办？

理论上说静态的字段和瞬时的字段是不能被序列化操作的。

`transient private String password;`

序列化操作

序列化操作时，如果两个class文件通过serialVersionUID序列化版本ID来判断字节码是否发生改变。`java.io.InvalidClassException`

如果不显示指定serialVersionUID变量，类变量的值由VM根据相关信息计算，而修改后的类的计算方式和之前往往不同。

从而形成了对象反序列化因为版本不兼容而失败的问题。

解决方案：在类中提供一个固定的serialVersionUID。

/需要序列化的对象

public class User implements java.io.Serializable {
 private static final long serialVersionUID = 1L;

更多教程在www.wolfcode.cn
或扫描页面尾部二维码

打印流

打印流：打印数据的打印流只能是输出流。

`PrintStream`: 字节打印流
`PrintWriter`: 字符打印流

对于`PrintWriter`来说，当启用字符刷新之后，

调用`print()`或`println()`方法，不会立即执行操作。

如果没有开启自动刷新，则需要手动刷新或者在缓冲区满的时候，再自动刷新。

使用`print()`方法作输出流时的操作特别简陋，因为在打印流中：

提供了`print()`方法，但并不执行。

提供了`println()`方法，但并不换行。

print()和println()方法可以支持输出各种数据类型的数据，记住`void println(Object x)`即可。

打印输出的格式化输出`[print|println]`:

`System.out.println()`其实等价于`PrintStream ps = System.out; ps.println()`

baseline `printf(String format, Object... args)`

使用指定格式字符串和参数将格式化的字符串写入此输出的便捷方法。

//Java的格式化输出`[print|println]`:

public class PrintDemo {
 public static void main(String[] args) {
 //打印一句话，效果风格：姓名:will,年龄:17
 String name = "龙17";
 int age = 17;
 //传统做法
 String str = "姓名:" + name + ",年龄:" + age;
 System.out.println(str);
 //格式化输出
 String format = "%s,%d";
 Object[] data = {"龙17", 17};
 System.out.printf(format, data);
 System.out.println();
 //格式化输出
 System.out.printf("姓名:%s,年龄:%d", "Will", 18);
 }
}

标准I/O概述和操作

标准的I/O：通过键盘录入数据给程序。

标准的输出：在屏幕上显示程序数据。

在System类中有两个常量：

`InputStream in = System.in;`

`PrintStream out = System.out;`

标准读流的重定向操作：

标准的输入：通过键盘录入数据给程序。

重新指定输入的源不再是键盘，而是一个文件。

`stdin.setInputStream(new FileInputStream("file/instream.txt"))`

此后`System.in`的数据就是从`instream`读取的数据。

标准的输出：在屏幕上显示程序数据。

重新指定输出的目标不再是屏幕，而是一个文件。

`static void setOut(PrintStream out) 重新分配标准输出流。`

字段摘要

System类

`static final OutputStream out` “标准” 借错输出流。

`static final InputStream in` “标准” 输入流。

`static final OutputStream out` “标准” 输出流。

//重定向标准输入流

`System.setIn(new FileInputStream("file/instream.txt"))`

//重定向标准输出流

`System.setOut(new PrintStream("file/xx.txt"))`

`System.out.println(..begin..);`

`int data = System.in.read(); //接受键盘录入的一个字节`

`System.out.print(data);`

`System.out.println(..end..);`

扫描器类(Scanner)

java.util.Scanner类：扫描器类，表示输入操作。

存在的方法：`xxx表示数据型.getAttribute(boolean)`。

`boolean hasNextXxx()`：判断是否有下一种类型的数据

`Xxx nextXxx();`：获取下一个该类型的数据。

//扫描文件中的数据

`Scanner sc = new Scanner(new File("file/stream.txt"), "GBK");`

//扫描键盘输入的数据

`Scanner sc = new Scanner(System.in);`

//扫描字符串

`Scanner sc = new Scanner("你好啊，真的，我没有骗你，说实话！");`

while (sc.hasNextLine()) {

String line = sc.nextLine();

System.out.println("ECHO:" + line);

}

sc.close();

Properties类和文件

配置文件-资源文件`[properties作为扩展名的文件]/属性文件：`

每个项目开发，为方便使用配置文件。

把所有的数据存进代码中，写死了，硬编码。

比如，在`wolfcode`中需要连接数据库，必须要有数据库的账号和密码。

此时我们在Java代码中编写，类似代码：

`String username="root";`

`String password="admin"`

代码部署运行TOMCAT。

但是，如果部署项目部署在别人电脑/服务器中，别人电脑中的数据库的账号和密码不可以不再是`root`和`admin`，此时我们再去修改，去替换用了账号和密码的地方。

部署项目的账号密码，为了安全操作，不能让直接修改代码。

此时，我们专门对数据库提供一个配置文件，里面专门存放数据库连接相关的信息。

`db.properties`

#key/value

username=root

password=admin

.....

现在数据库的连接信息在`db.properties`文件中，而Java代码需要读取该文件中的信息。

重心转移：Java代码如何加载`properties`文件，如何读取该文件中的数据。

必须使用`Properties类(Hashtable的子类,Map接口的实现)`。

//加载properties文件

`public class LoadResourceDemo {`

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

//创建Properties对象

Properties p = new Properties();

//从文件中读取数据，FileInputStream类，从文件中读取

p.load(new FileInputStream("H:/JavaApp/IO/src/db.properties"));

p.load(inStream);

System.out.println(p);

System.out.println("账号：" + p.getProperty("username"));

System.out.println("密码：" + p.getProperty("password"));

}

}

Properties类和文件

System类

`static final OutputStream out` “标准” 错误输出流。

`static final InputStream in` “标准” 输入流。

`static final OutputStream out` “标准” 输出流。

//重定向标准输入流

`System.setIn(new FileInputStream("file/instream.txt"))`

//重定向标准输出流

`System.setOut(new PrintStream("file/xx.txt"))`

`System.out.println(..begin..);`

`int data = System.in.read(); //接受键盘录入的一个字节`

`System.out.print(data);`

`System.out.println(..end..);`

数据流：提供了可以读/写任意数据类型的方法。

`DataInputStream`：提供了`readXXX(value)`方法。

`DataOutputStream`：提供了`writeXXX()`方法。

注意：`writeXXX()`方法必须要对起来，`writeXXX()`方法写出来数据，此时只能使用`readByte()`读取回来。

//数据流

`public class DataStreamDemo {`

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

File f = new File("file/out.txt");

//写入文件
 BufferedWriter out = new BufferedWriter(new FileWriter(f));

out.write("Hello WolfCode");

out.close();

//读取文件
 DataInputStream in = new DataInputStream(new FileInputStream(f));

byte b = in.readByte();

System.out.print((char)b);

in.close();

}

}

Properties类和文件

System类

`static final OutputStream out` “标准” 错误输出流。