Java 泛型 Java URL处理

Java 序列化

Java 提供了一种对象序列化的机制，该机制中，一个对象可以被表示为一个字节序列，该字节序列包括该对象的数据、有关对象的类型的信息和存储在对象中数据的类型。

将序列化对象写入文件之后，可以从文件中读取出来，并且对它进行反序列化，也就是说，对象的类型信息、对象的数据，还有对象中的数据类型可以用来在内存中新建对象。

整个过程都是 Java 虚拟机（JVM）独立的，也就是说，在一个平台上序列化的对象可以在另一个完全不同的平台上反序列化该对象。

类 ObjectInputStream 和 ObjectOutputStream 是高层次的数据流，它们包含反序列化和序列化对象的方法。

ObjectOutputStream 类包含很多写方法来写各种数据类型，但是一个特别的方法例外：

```
public final void writeObject(Object x) throws IOException
```

上面的方法序列化一个对象，并将它发送到输出流。相似的 ObjectInputStream 类包含如下反序列化一个对象的方法：

```
public final Object readObject() throws IOException,
ClassNotFoundException
```

该方法从流中取出下一个对象，并将对象反序列化。它的返回值为Object，因此，你需要将它转换成合适的数据类型。

为了演示序列化在Java中是怎样工作的，我将使用之前教程中提到的Employee类，假设我们定义了如下的Employee类，该类实现了Serializable 接口。

Employee.java 文件代码：

```
public class Employee implements java.io.Serializable
{
public String name;
public String address;
public transient int SSN;
public int number;
public void mailCheck()
{
System.out.println("Mailing a check to " + name
+ " " + address);
}
}
```

请注意，一个类的对象要想序列化成功，必须满足两个条件：

该类必须实现 java.io.Serializable 对象。

该类的所有属性必须是可序列化的。如果有一个属性不是可序列化的，则该属性必须注明是短暂的。

如果你想知道一个 Java 标准类是否是可序列化的，请查看该类的文档。检验一个类的实例是否能序列化十分简单， 只需要查看该类有没有实现 java.io.Serializable接口。

序列化对象

ObjectOutputStream 类用来序列化一个对象，如下的 SerializeDemo 例子实例化了一个 Employee 对象，并将该对象序列化到一个文件中。

该程序执行后，就创建了一个名为 employee.ser 文件。该程序没有任何输出，但是你可以通过代码研读来理解程序的作用。

注意： 当序列化一个对象到文件时， 按照 Java 的标准约定是给文件一个 .ser 扩展名。

SerializeDemo.java 文件代码：

```
import java.io.*;
public class SerializeDemo
{
public static void main(String [] args)
{
Employee e = new Employee();
e.name = "Reyan Ali";
e.address = "Phokka Kuan, Ambehta Peer";
e.SSN = 11122333;
e.number = 101;
try
{
FileOutputStream fileOut =
new FileOutputStream("/tmp/employee.ser");
ObjectOutputStream out = new ObjectOutputStream(fileOut);
out.writeObject(e);
out.close();
fileOut.close();
System.out.printf("Serialized data is saved in /tmp/employee.ser");
}catch(IOException i)
{
i.printStackTrace();
}
}
}
```

反序列化对象

下面的 DeserializeDemo 程序实例了反序列化，/tmp/employee.ser 存储了 Employee 对象。

DeserializeDemo.java 文件代码：

```
import java.io.*;
public class DeserializeDemo
{
public static void main(String [] args)
{
Employee e = null;
try
{
FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
ObjectInputStream in = new ObjectInputStream(fileIn);
e = (Employee) in.readObject();
in.close();
fileIn.close();
}catch(IOException i)
```

```
{
i.printStackTrace();
return;
}catch(ClassNotFoundException c)
{
System.out.println("Employee class not found");
c.printStackTrace();
return;
}
System.out.println("Deserialized Employee...");
System.out.println("Name: " + e.name);
System.out.println("Address: " + e.address);
System.out.println("SSN: " + e.SSN);
System.out.println("Number: " + e.number);
}
}
```

以上程序编译运行结果如下所示：

```
Deserialized Employee...
Name: Reyan Ali
Address:Phokka Kuan, Ambehta Peer
SSN: 0
Number:101
```

这里要注意以下要点：

readObject() 方法中的 try/catch代码块尝试捕获 ClassNotFoundException 异常。对于 JVM 可以反序列化对象，它必须是能够找到字节码的类。如果JVM在反序列化对象的过程中找不到该类，则抛出一个 ClassNotFoundException 异常。

注意，readObject() 方法的返回值被转化成 Employee 引用。

当对象被序列化时，属性 SSN 的值为 111222333，但是因为该属性是短暂的，该值没有被发送到输出流。所以反序列化后 Employee 对象的 SSN 属性为 0。

← Java 泛型 Java URL处理 →