* **处理流之六：对象流**
* **ObjectInputStream**和**ObjectOutputStream**
* **用于存储和读取对象的处理流。它的强大之处就是可以把Java中的对象写入到数据源中，也能把对象从数据源中还原回来**。
* **序列化(Serialize)**：用ObjectOutputStream类将一个Java对象写入IO流中。

/\*\*  
 \* 对象的序列化过程，将内存中的对象通过ObjectOutputStream转换为二进制流，

存储在硬盘文件中  
 \*/  
@Test  
public void test1() {  
  
 Person p1 = new Person("小米", 23,new Pet("花花"));  
 Person p2 = new Person("红米", 21,new Pet("小花"));  
  
 ObjectOutputStream oos = null;  
 try {  
 oos = new ObjectOutputStream(new FileOutputStream("person.txt"));  
  
 oos.writeObject(p1);  
 oos.flush();  
 oos.writeObject(p2);  
 oos.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 if (oos != null) {  
 try {  
 oos.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

* **反序列化(Deserialize)**：用ObjectInputStream类从IO流中恢复该Java对象。
* ObjectOutputStream和ObjectInputStream**不能序列化static和transient修饰的成员变量**。

/\*\*  
 \* 对象的反序列化过程，将硬盘中的文件通过ObjectInputStream转换为相应的对象。  
 \*/  
 @Test  
 public void test2() {  
  
 ObjectInputStream ois = null;  
 try {  
 ois = new ObjectInputStream(new FileInputStream("Person.txt"));  
  
 Person p1 = (Person) ois.readObject();  
 System.***out***.println(p1);  
 Person p2 = (Person) ois.readObject();  
 System.***out***.println(p2);  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 if (ois != null) {  
 try {  
 ois.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

* **对象的序列化**

/\*\*  
 \* 要实现序列化的类：  
 \* 1）要求此类是可序列化的，实现Serializable接口；  
 \* 2）要求类的属性同样的要实现Serializable接口。  
 \* 3）提供一个版本号，private static final long serialVersionUID  
 \* 4）使用static或transient修饰的属性，不可实现序列化。  
 \*/

* **对象序列化机制**
* 允许把内存中的Java对象转换成平台无关的二进制流，从而允许把这种二进制流持久地保存在磁盘上，或通过网络将这种二进制流传输到另一个网络节点。当其它程序获取了这种二进制流，就可以恢复成原来的Java对象。
* 序列化的好处在于可将任何**实现了Serializable接口的对象转化为字节数据**，使其在保存和传输时可被还原。
* 序列化是RMI（Remote Method Invoke–远程方法调用）过程的参数和返回值都必须实现的机制，而RMI是JavaEE的基础。因此序列化机制是JavaEE平台的基础。
* （1）如果需要让某个对象支持序列化机制，则必须**让其类是可序列化**的，为了让某个类是可序列化的，该类必须实现如下两个接口之一：
* **Serializable**
* **Externalizable**
* （2）凡是实现Serializable接口的类都有一个表示序列化版本标识符的静态变量：
* **private static final long serialVersionUID;**
* serialVersionUID用来表明类的不同版本间的兼容性。
* 如果类没有显示定义这个静态变量，它的值是Java运行时环境根据类的内部细节自动生成的。若类的源代码作了修改，serialVersionUID可能发生变化。故建议，显示声明。
* 显示定义serialVersionUID的用途
* 希望类的不同版本对序列化兼容，因此需确保类的不同版本具有相同的serialVersionUID
* 不希望类的不同版本对序列化兼容，因此需确保类的不同版本具有不同的serialVersionUID
* **使用对象流序列化对象**
* 若某个类实现了Serializable接口，该类的对象就是可序列化的：
* 创建一个ObjectOutputStream
* 调用ObjectOutputStream对象的writeObject(对象)方法输出可序列化对象。注意写出一次，操作flush()
* 反序列化
* 创建一个ObjectInputStream
* 调用readObject()方法读取流中的对象
* 强调：如果某个类的字段不是基本数据类型或String类型，而是另一个引用类型，那么这个引用类型必须是可序列化的，否则拥有该类型的Field的类也不能序列化
* 序列化:将对象写入到磁盘或者进行网络传输。
* 要求对象必须实现序列化
* ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream("test3.txt"));
* Person p=new Person("韩梅梅",18,"中华大街",new Pet());
* oos.writeObject(p);
* oos.flush();
* oos.close();
* //反序列化：将磁盘中的对象数据源读出。
* ObjectInputStream ois=new ObjectInputStream(new FileInputStream("test3.txt"));
* Person p1=(Person)ois.readObject();
* System.out.println(p1.toString());
* ois.close();