**serialVersionUID的作用**

简单来说，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)

serialVersionUID有两种显示的生成方式：        一个是默认的1L，比如：private static final long serialVersionUID = 1L;        一个是根据类名、接口名、成员方法及属性等来生成一个64位的哈希字段，比如：        private static final   long     serialVersionUID = xxxxL; 当你一个类实现了Serializable接口，如果没有显示的定义serialVersionUID，Eclipse会提供这个      提示功能告诉你去定义 。在Eclipse中点击类中warning的图标一下，Eclipse就会      自动给定两种生成的方式。如果不想定义它，在Eclipse的设置中也       可以把它关掉的，设置如下：         Window ==> Preferences ==> Java ==> Compiler ==> Error/Warnings ==>         Potential programming problems         将Serializable class without serialVersionUID的warning改成ignore即可。  
当实现java.io.Serializable接口的实体（类）没有显式地定义一个名为serialVersionUID，类型为long的变量时，Java序列化机制会根据编译的class(它通过类名，方法名等诸多因素经过计算而得，理论上是一一映射的关系，也就是唯一的)自动生成一个serialVersionUID作序列化版本比较用，这种情况下，如果class文件(类名,方法明等)没有发生变化(增加空格,换行,增加注释,等等),就算再编译多次,serialVersionUID也不会变化的.

如果我们不希望通过编译来强制划分软件版本，即实现序列化接口的实体能够兼容先前版本，未作更改的类，就需要显式地定义一个名为serialVersionUID，类型为long的变量，不修改这个变量值的序列化实体都可以相互进行串行化和反串行化。

如果你没有考虑到兼容性问题时，就把它关掉，不过有这个功能是好的，只要任何类别实现了Serializable这个接口的话，如果没有加入 serialVersionUID，Eclipse都会给你warning提示，这个serialVersionUID为了让该类别 Serializable向后兼容。

问题一:假设有A端和B端,如果2处的serialVersionUID不一致,会产生什么错误呢?

问题二:假设2处serialVersionUID一致,如果A端增加一个字段,B端不变,会是什么情况呢?

问题三:假设2处serialVersionUID一致,如果B段增加一个字段,A端不变,会是什么情况呢?

问题四:假设2处serialVersionUID一致,如果A端减少一个字段,B端不变,会是什么情况呢?

问题五:假设2处serialVersionUID一致,如果B端减少一个字段,A端不变,会是什么情况呢?

例子如下:写2个类,类名相同,字段相同,方法相同.放在不同的包里,来模仿A端和B端.

实体类:在本例中,在测试类SerialTest执行前代表A端,然后,在测试类DeserialTest执行前代表B端.

package com.test; import java.io.Serializable;

public class Serial implements Serializable{ /\*\*   \*   \*/ private static final long serialVersionUID = 6977402643848374753L; int id;       String name;       public Serial(int id, String name) {           this.id = id;           this.name = name;       }       public String toString() {           return "DATA: " + id + " " +name;         }   }

测试类,代表A端的序列化

package com.test.serializable;

import java.io.FileNotFoundException; import java.io.FileOutputStream; import java.io.IOException; import java.io.ObjectOutputStream;

public class SerialTest {

/\*\*   \* @param args   \*/ public static void main(String[] args) {   // TODO Auto-generated method stub   Serial serial1 = new Serial(1,"song");   System.out.println("Object Serial"+serial1);     try {    FileOutputStream fos = new FileOutputStream("serialTest.txt");    ObjectOutputStream oos = new ObjectOutputStream(fos);    oos.writeObject(serial1);    oos.flush();    oos.close();         } catch (FileNotFoundException e) {    // TODO Auto-generated catch block    e.printStackTrace();   } catch (IOException e) {    // TODO Auto-generated catch block    e.printStackTrace();   } } }

测试类,代表B端的反序列化

package com.test.serializable;

import java.io.FileInputStream; import java.io.FileNotFoundException; import java.io.IOException; import java.io.ObjectInputStream;

public class DeserialTest {

/\*\*   \* @param args   \*/ public static void main(String[] args) {   // TODO Auto-generated method stub   Serial serial2 ;   try {    FileInputStream fis = new FileInputStream("serialTest.txt");    ObjectInputStream ois = new ObjectInputStream(fis);    serial2 = (Serial)ois.readObject();    ois.close();    System.out.println("Object Deserial"+serial2);   } catch (FileNotFoundException e) {    // TODO Auto-generated catch block    e.printStackTrace();   } catch (IOException e) {    // TODO Auto-generated catch block    e.printStackTrace();   } catch (ClassNotFoundException e) {    // TODO Auto-generated catch block    e.printStackTrace();   }   }

}

问题一:假设有A端和B端,如果2处的serialVersionUID不一致,会产生什么错误呢?

答案如下:

1)先执行测试类SerialTest,然后修改serialVersion值(或注释掉serialVersion并编译),再执行测试类DeserialTest,报错:

java.io.InvalidClassException: com.test.serializable.Serial; local class incompatible: stream classdesc serialVersionUID = 1, local class serialVersionUID = 11

2)A端和B端都没显示的写serialVersionUID,实体类没有改动(如果class文件(类名,方法明等)没有发生变化(增加空格,换行,增加注释,等等),).序列化,反序列化正常.

问题二:假设2处serialVersionUID一致,如果A端增加一个字段,B端不变,会是什么情况呢?

答案二: 序列化,反序列化正常,A端增加的字段丢失(被B端忽略).

问题五:假设2处serialVersionUID一致,如果B端减少一个字段,A端不变,会是什么情况呢?

答案:与问题二类似,序列化,反序列化正常,B端字段少于A端,A端多的字段值丢失(被B端忽略).

问题三:假设2处serialVersionUID一致,如果B段增加一个字段,A端不变,会是什么情况呢?

问题四:假设2处serialVersionUID一致,如果A端减少一个字段,B端不变,会是什么情况呢?(与问题三类似,四答案:序列化,反序列化正常,B端字段多余A端,B端多出的字段被赋予对应类型的默认值)

答案三: 序列化,反序列化正常,B端新增加的int字段被赋予了默认值0.

例子如下:

3)先执行SerialTest,然后在实体类增加一个字段age,再执行测试类DeserialTest.

package com.test.serializable;

import java.io.Serializable;

public class Serial implements Serializable {

/\*\*   \*   \*/ private static final long serialVersionUID = -2337937881709830076L;

/\*\*   \*   \*/ //private static final long serialVersionUID = 1L; int id;

    String name;       public Serial(int id, String name) {           this.id = id;           this.name = name;       }       public String toString() {           return "DATA: " + id + " " +name;         }     public int age ;//在B端增加一个新字段 }

相应的修改测试类DeserialTest,打印出age的值.

package com.test.serializable;

import java.io.FileInputStream; import java.io.FileNotFoundException; import java.io.IOException; import java.io.ObjectInputStream;

public class DeserialTest {

/\*\*   \* @param args   \*/ public static void main(String[] args) {   // TODO Auto-generated method stub   Serial serial2 ;   try {    FileInputStream fis = new FileInputStream("serialTest.txt");    ObjectInputStream ois = new ObjectInputStream(fis);    serial2 = (Serial)ois.readObject();    ois.close();    System.out.println("Object Deserial"+serial2+" age="+serial2.age);   } catch (FileNotFoundException e) {    // TODO Auto-generated catch block    e.printStackTrace();   } catch (IOException e) {    // TODO Auto-generated catch block    e.printStackTrace();   } catch (ClassNotFoundException e) {    // TODO Auto-generated catch block    e.printStackTrace();   }   }

}

打印结果如下:

Object DeserialDATA: 1 song age=0

说明序列化,反序列化正常,B端新增加的int字段被赋予了默认值0.

上面的情况对增加/减少 字段/方法 都适用.

//但当serialVersionUID相同时，它就会将不一样的field以type的预设值Deserialize，可避开不兼容性问题。