java.io.InvalidClassException: local class incompatible:

Asked 11 years, 10 months ago Modified 1 year, 9 months ago Viewed 187k times

66

I created client and server and then added a class in client side for serializing purposes, then simply just went to the folder of the client in my hard drive and copy paste it to the server correponding location, both `classname.class` and `classname.java` respectively.

It worked well in my own laptop but when I wanted to continue my work on other system , when I opened the projects folders and after client tries to connect to the server, the following error appears:

```
Exception in thread "main" java.io.InvalidClassException: projectname.clasname;
local class incompatible: stream classdesc serialVersionUID =
-6009442170907349114, local class serialVersionUID = 6529685098267757690
    at java.io.ObjectStreamClass.initNonProxy(ObjectStreamClass.java:562)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1582)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1495)
    at
java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1731)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1328)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:350)
```

What is going on? Is it because I ran the program with an older version of the IDE?

EDIT

```
import java.io.Serializable;
import java.net.URL;

public class KeyAdr implements Serializable {
  private static final long serialVersionUID = 6529685098267757690L;

  public URL adr;
  public String key;
}
```

Join Stack Overflow

Be part of the community and earn reputation by helping others. Save your favorite posts and try dark mode!

Sign up

10 Answers

Sorted by: Highest score (default)

97

If a class does not explicitly define a `private static final long serialVersionUID` in the code it will be autogenerated, and there is no guarantee that different machines will generate the same id; it looks like that is exactly what happened. Also if the classes are different in any way (using different versions of the class) the autogenerated `serialVersionUID` s will also be different.

From the `Serializable` interface's docs:

> If a serializable class does not explicitly declare a `serialVersionUID` , then the serialization runtime will calculate a default `serialVersionUID` value for that class based on various aspects of the class, as described in the Java(TM) Object Serialization Specification. However, it is *strongly recommended* that all serializable classes explicitly declare `serialVersionUID` values, since the default `serialVersionUID` computation is highly sensitive to class details that may vary depending on compiler implementations, and can thus result in unexpected `InvalidClassExceptions` during deserialization. Therefore, to guarantee a consistent `serialVersionUID` value across different java compiler implementations, a serializable class must declare an explicit `serialVersionUID` value. It is also strongly advised that explicit `serialVersionUID` declarations use the `private` modifier where possible, since such declarations apply only to the immediately declaring class-- `serialVersionUID` fields are not useful as inherited members. Array classes cannot declare an explicit `serialVersionUID` , so they always have the default computed value, but the requirement for matching `serialVersionUID` values is waived for array classes.

You should define a `serialVersionUID` in the class definition, e.g.:

```
class MyClass implements Serializable {
    private static final long serialVersionUID = 6529685098267757690L;
    ...
```

5 try cleaning the project outputs (`.class` files generated at compilation time) and rebuild (recompile) the projects. – yair Apr 30, 2012 at 6:47

1 @EJP trying to match the SUID to a different version of the class isn't advisable. Updating the class at both ends was the right thing to do here. – trutheality Apr 30, 2012 at 18:48

13

One thing that could have happened:

- 1: you create your serialized data with a given library A (version X)
- 2: you then try to read this data with the same library A (but version Y)

Hence, at compile time for the version X, the JVM will generate a first Serial ID (for version X) and it will do the same with the other version Y (another Serial ID).

When your program tries to de-serialize the data, it can't because the two classes do not have the same Serial ID and your program have no guarantee that the two Serialized objects correspond to the same class format.

Assuming you changed your constructor in the mean time and this should make sense to you.

Share Improve this answer Follow

answered Mar 14, 2018 at 9:52
belka
1,500 1 19 31

Possibly, you added a field to the class but in the serialized object it was no such a field.
– Szymon Roziewski Nov 16, 2023 at 13:39

1

If you are using the Eclipse IDE, check your Debug/Run configuration. At Classpath tab, select the runner project and click Edit button. *Only include exported entries* must be checked.

Share Improve this answer Follow

answered Feb 3, 2020 at 8:12
Pekmezli Dürüm
113 6

1

Try deleting/renaming the output file and recreate a new one if you had stored an object during testing, I had this error because I had serialized data to the same file in two different jFrame Classes.

Share Improve this answer Follow

answered May 16, 2022 at 13:13
Tankiso Thebe
170 2 12

0

I believe this happens because you use the different versions of the same class on client and server. It can be different data fields or methods

Share Improve this answer Follow

answered Apr 30, 2012 at 5:29
Mark Bramnik
40.8k 4 59 102

0

The exception message clearly speaks that the class versions, which would include the class meta data as well, has changed over time. In other words, the class structure during serialization is not the same during de-serialization. This is what is most probably "going on".

Share Improve this answer Follow

answered Mar 8, 2015 at 15:39
nawazish-stackoverflow
263 5 14

Serialisation in java is not meant as long term persistence or transport format - it is too fragile

2 I can agree to a degree for persistence but there is nothing wrong with using Java Serialization as a transport format. It's too fragile if one doesn't know the concepts of Java serialization like `serialVersionUID` 's. – Sanjay T. Sharma Apr 30, 2012 at 6:43

It isn't, really. Maybe we just see the 'Java serialization' world with different colored glasses. Plus it's pretty much the only option when using RMI. Also, I see that you had made some comment about 'project manager skinning something' which was edited out later... – Sanjay T. Sharma Apr 30, 2012 at 8:47

"Slightest difference in class bytecode and JVM and your data is not readable anymore". This is simply untrue. See the Object Versioning section of the Object Serialziation Specification. – user207421 Apr 30, 2012 at 10:06

@EJP Here's a link: Versioning of Serializable Objects – jpaugh Sep 20, 2016 at 14:12

0

If you are using oc4j to deploy the ear.

Make sure you set in the project the correct path for deploy.home=

You can fiind deploy.home in common.properties file

The oc4j needs to reload the new created class in the ear so that the server class and the client class have the same serialVersionUID

Share Improve this answer Follow edited Jul 24, 2019 at 8:10 answered Jul 24, 2019 at 7:31

Mr Big
1 2

0

The autogenerated serialVersionUID would be same in different OS or JDK version. But if you add a function or a field, the autogenerated serialVersionUID will change.

Share Improve this answer Follow answered May 11, 2022 at 3:28

Roc King
441 6 18