/

Java EE Support Patterns

Cloud, Java, Middleware, APM & Tools tutorials

java.lang.ClassNotFoundException: How to resolve

11/11/2012 PIERRE-HUGUES CHARBONNEAU 5 COMMENTS

This article is intended for Java beginners facing java.lang.ClassNotFoundException challenges. It will provide you with an overview of this common Java exception, a sample Java program to support your learning process and resolution strategies.

If you are interested on more advanced class loader related problems, I recommended that you review my article series on java.lang.NoClassDefFoundError since these Java exceptions are closely related.

java.lang.ClassNotFoundException: Overview

As per the Oracle documentation, ClassNotFoundException is thrown following the failure of a class loading call, using its string name, as per below:

- The Class.forName method
- The ClassLoader.findSystemClass method
- The ClassLoader.loadClass method

In other words, it means that one particular Java class was not found or could not be loaded at “runtime” from your application current context class loader.

This problem can be particularly confusing for Java beginners. This is why I always recommend to Java developers to learn and refine their knowledge on Java class loaders. Unless you are involved in dynamic class loading and using the Java Reflection API, chances are that the ClassNotFoundException error you are getting is not from your application code but from a referencing API. Another common problem pattern is a wrong packaging of your application code. We will get back to the resolution strategies at the end of the article.

java.lang.ClassNotFoundException: Sample Java program

* A ClassNotFoundException YouTube video is now available.

Now find below a very simple Java program which simulates the 2 most common ClassNotFoundException scenarios via Class.forName() & ClassLoader.loadClass(). Please simply copy/paste and run the program with the IDE of your choice (*Eclipse IDE was used for this example*).

The Java program allows you to choose between problem scenario #1 or problem scenario #2 as per below. Simply change to 1 or 2 depending of the scenario you want to study.

Class.forName()

```
private static final int PROBLEM_SCENARIO = 1;
```

ClassLoader.loadClass()

```
private static final int PROBLEM_SCENARIO = 2;
```

ClassNotFoundExceptionSimulator

```
package org.ph.javaee.training5;

/**
 * ClassNotFoundExceptionSimulator
 * @author Pierre-Hugues Charbonneau
 *
 */
```

I am happy to inform you that I published recently...

DZone's Guide to Building and Deploying Applications on the Cloud
This post is to inform you that DZone has just...

Java 8 - CPU Flame Graph
Brendan Gregg and Martin Spier from Netflix recently shared a...

HEADLINES BY FEEDBURNER

ABOUT THE AUTHOR

PIERRE-HUGUES CHARBONNEAU

P-H is an IT Architect currently working for CGI Inc. in Canada. He has 15 years+ of experience developing and troubleshooting Java & Web enterprise systems. Email: phcharbonneau@hotmail.com.

VIEW MY COMPLETE PROFILE

BLOG ARCHIVE

```
public class ClassNotFoundExceptionSimulator {

    private static final String CLASS_TO_LOAD = "org.ph.javaee.training5.ClassA";
    private static final int PROBLEM_SCENARIO = 1;

    /**
     * @param args
     */
    public static void main(String[] args) {

        System.out.println("java.lang.ClassNotFoundException Simulator - Training 5");
        System.out.println("Author: Pierre-Hugues Charbonneau");
        System.out.println("http://javaeesupportpatterns.blogspot.com");

        switch(PROBLEM_SCENARIO) {

            // Scenario #1 - Class.forName()
            case 1:

                System.out.println("\n** Problem scenario #1: Class.forName() **\n");
                try {
                    Class<?> newClass = Class.forName(CLASS_TO_LOAD);

                    System.out.println("Class "+newClass+" found successfully!");

                } catch (ClassNotFoundException ex) {

                    ex.printStackTrace();

                    System.out.println("Class "+CLASS_TO_LOAD+" not found!");

                } catch (Throwable any) {
                    System.out.println("Unexpected error! "+any);
```

```
                }

                break;

            // Scenario #2 - ClassLoader.loadClass()
            case 2:

                System.out.println("\n** Problem scenario #2: ClassLoader.loadClass() **\n");
                try {
                    ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
                    Class<?> callerClass = classLoader.loadClass(CLASS_TO_LOAD);

                    Object newClassAInstance = callerClass.newInstance();

                    System.out.println("SUCCESS!: "+newClassAInstance);
                } catch (ClassNotFoundException ex) {

                    ex.printStackTrace();

                    System.out.println("Class "+CLASS_TO_LOAD+" not found!");

                } catch (Throwable any) {
                    System.out.println("Unexpected error! "+any);
                }

                break;
        }

        System.out.println("\nSimulator done!");
    }
}
```

ClassA

```
package org.ph.javaee.training5;

/**
 * ClassA
 * @author Pierre-Hugues Charbonneau
 *
 */
public class ClassA {

private final static Class<ClassA> CLAZZ = ClassA.class;

    static {
        System.out.println("Class loading of "+CLAZZ+" from ClassLoader '"+CLAZZ.getClassLoader()+"' in progress...");
    }

    public ClassA() {
        System.out.println("Creating a new instance of "+ClassA.class.getName()+"...");

        doSomething();
    }

    private void doSomething() {
        // Nothing to do...
    }
}
```

If you run the program as is, you will see the output as per below for each scenario:

#Scenario 1 output (baseline)

java.lang.ClassNotFoundException Simulator - Training 5

```
Author: Pierre-Hugues Charbonneau
http://javaeesupportpatterns.blogspot.com

** Problem scenario #1: Class.forName() **

Class loading of class org.ph.javaee.training5.ClassA from ClassLoader
'sun.misc.Launcher$AppClassLoader@bfbdb0' in progress...
Class class org.ph.javaee.training5.ClassA found successfully!

Simulator done!
```

#Scenario 2 output (baseline)

```
java.lang.ClassNotFoundException Simulator - Training 5
Author: Pierre-Hugues Charbonneau
http://javaeesupportpatterns.blogspot.com

** Problem scenario #2: ClassLoader.loadClass() **

Class loading of class org.ph.javaee.training5.ClassA from ClassLoader
'sun.misc.Launcher$AppClassLoader@2a340e' in progress...
Creating a new instance of org.ph.javaee.training5.ClassA...
SUCCESS!: org.ph.javaee.training5.ClassA@6eb38a

Simulator done!
```

For the “baseline” run, the Java program is able to load `ClassA` successfully.

Now let’s voluntary change the full name of `ClassA` and re-run the program for each scenario. The following output can be observed:

#ClassA changed to ClassB

```
private static final String CLASS_TO_LOAD = "org.ph.javaee.training5.ClassB";
```

#Scenario 1 output (problem replication)

```
java.lang.ClassNotFoundException Simulator - Training 5
Author: Pierre-Hugues Charbonneau
http://javaeesupportpatterns.blogspot.com

** Problem scenario #1: Class.forName() **

java.lang.ClassNotFoundException: org.ph.javaee.training5.ClassB
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:186)
        at
org.ph.javaee.training5.ClassNotFoundExceptionSimulator.main(ClassNotFoundExceptionSimulator.java:29)
Class org.ph.javaee.training5.ClassB not found!

Simulator done!
```

#Scenario 2 output (problem replication)

```
java.lang.ClassNotFoundException Simulator - Training 5
Author: Pierre-Hugues Charbonneau
http://javaeesupportpatterns.blogspot.com

** Problem scenario #2: ClassLoader.loadClass() **

java.lang.ClassNotFoundException: org.ph.javaee.training5.ClassB
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
```

```
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    at
org.ph.javaee.training5.ClassNotFoundExceptionSimulator.main(ClassNotFoundExceptionSimulator.java:51)
Class org.ph.javaee.training5.ClassB not found!

Simulator done!
```

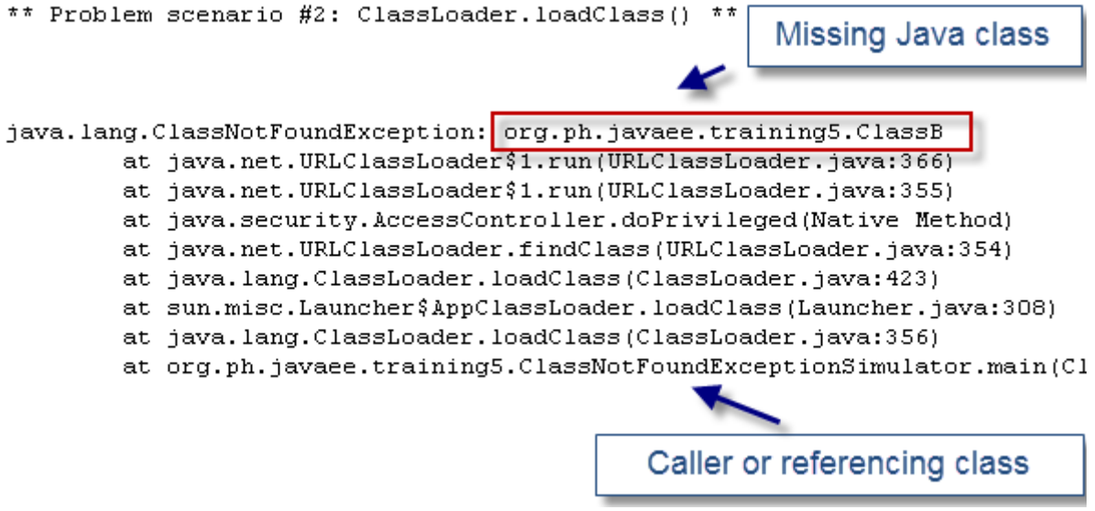
What happened? Well since we changed the full class name to org.ph.javaee.training5.ClassB, such class was not found at runtime (does not exist), causing both Class.forName() and ClassLoader.loadClass() calls to fail.

You can also replicate this problem by packaging each class of this program to its own JAR file and then omit the jar file containing ClassA.class from the main class path Please try this and see the results for yourself…(hint: NoClassDefFoundError)

Now let's jump to the resolution strategies.

java.lang.ClassNotFoundException: Resolution strategies

Now that you understand this problem, it is now time to resolve it. Resolution can be fairly simple or very complex depending of the root cause.



- Don’t jump on complex root causes too quickly, rule out the simplest causes first.
- First review the java.lang.ClassNotFoundException stack trace as per the above and determine which Java class was not loaded properly at runtime e.g. application code, third party API, Java EE container itself etc.
- Identify the caller e.g. Java class you see from the stack trace just before the Class.forName() or ClassLoader.loadClass() calls. This will help you understand if your application code is at fault vs. a third party API.
- Determine if your application code is not packaged properly e.g. missing JAR file(s) from your classpath
- If the missing Java class is not from your application code, then identify if it belongs to a third party API you are using as per of your Java application. Once you identify it, you will need to add the missing JAR file(s) to your runtime classpath or web application WAR/EAR file.
- If still struggling after multiple resolution attempts, this could means a more complex class loader hierarchy problem. In this case, please review my NoClassDefFoundError article series for more examples and resolution strategies

I hope this article has helped you to understand and revisit this common Java exception. Please feel free to post any comment or question if you are still struggling with your java.lang.ClassNotFoundException problem.

Posted in: ClassLoader,Java

Newer Post Home Older Post

5 comments:

Hima P says:
January 3, 2013 at 8:51 AM

nice coding.....

Reply

Anonymous says:
January 10, 2013 at 8:57 AM

This worked really well. Thank you

Reply

Pierre-Hugues Charbonneau says:
January 10, 2013 at 8:59 AM

Thanks, glad that you liked the tutorial and sample Java program.

It is always best to learn these principles via simple Java programs vs. trying to understand complex applications too fast too soon.

Regards,
P-H

Reply

Adnan Alfadly says:

September 6, 2014 at 10:03 AM

This worked really well. Thank you

Reply

Ucky Kit says:

November 11, 2015 at 5:17 AM

Thanks , this article is really nice

Reply

Post a Comment

Enter your comment...

Comment as: Amitabh Mandal

Sign out

Publish Preview

Notify me