
Java™
ORACLE

Working with Java class loaders

Vaibhav Choudhary (@vaibhav_c)
Java Platforms Team
https://blogs.oracle.com/vaibhav

Java Your Next (Cloud)

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda of the day …

1. Class and Data
2. Types of class loaders
3. Class loader delegation model
4. Implementation snapshot
5. Parallel Class loading
6. Debugging class loader issues
7. Module Class loader and Jigsaw

Class And Data

- Class represents code. Data represents state.
- State often change, code mostly not.
- Class + State = Instance
- Once a class is loaded in JVM, the same class (?) will not be loaded.
- Same class concept
  - (C1, P1, K1) != (C1, P1, K2)

Class loader types

- Bootstrap class loader (primordial class loader)
  - Special class loader
  - VM runs without verification
  - Access to the repository of “trusted classes”
  - Native implementation
  - **<JAVA_HOME>/jre/lib (rt.jar, i18n.jar)**

Class loader types

- Extension class loader
  - Loads the code in the extension directories
  - <JAVA_HOME>/jre/lib/ext
  - -Djava.ext.dir
  - Java implementation - sun.misc.Launcher$ExtClassLoader

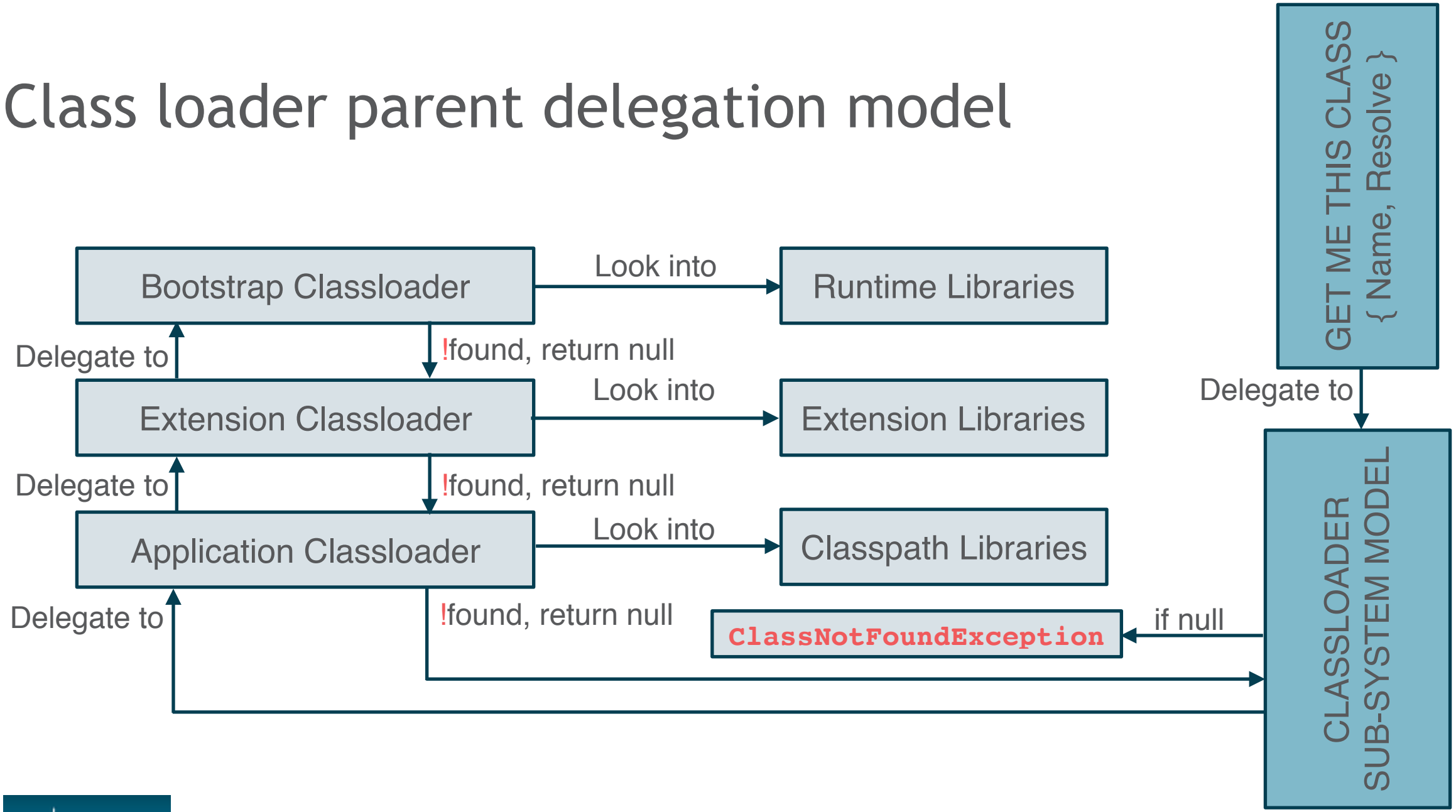
Class loader types

- Application class loader
  - Loads the code found in java.class.path which maps to CLASSPATH.
  - Java implementation - sun.misc.Launcher$AppClassLoader

Writing own class loader

- Why ?
  - Umm, many reasons but the prominent one can be :-
    - class loading from alternate repo - like over network connection.
    - servlet engines - use for partitioning codes.
    - Unloading classes - App Classloader
  - Not going to discuss in detail, how to write own class loader.

Class loader parent delegation model



In short, this is how it works

```
protected synchronized Class<?> loadClass
    (String name, boolean resolved)
    throws ClassNotFoundException {

    // First check if the class is already loaded
    Class c = findLoadedClass(name);
    if (c == null) {
        try {
            if (parent != null) {
                c = parent.loadClass(name, false);  // parent delegation
            } else {
                    c = findBootstrapClass0(name);
            }
        } catch (ClassNotFoundException e) {
            c = findClass(name);                    // if still not found, invoke findClass
        }
    }
    if(resolved) {
           resolveClass(c);
    }
    return c;
}
```

In short, this is how it works

```
- Setting up the parent class loader. Two ways to do :-

Way 1:-
public class MyClassLoader extends ClassLoader{

    public MyClassLoader(){
        super(MyClassLoader.class.getClassLoader());
    }
}

Way 2:-
public class MyClassLoader extends ClassLoader{

    public MyClassLoader(){
        super(getClass().getClassLoader());
    }
}
```

Default implementation of findClass

- Default implementation of findClass will throw ClassNotFoundException.
- Developers are expected to implement this method.

```
protected Class<?> findClass(String name)
        throws ClassNotFoundException {
        throw new ClassNotFoundException(name);
    }
```

Parallel Class loading

- JDK7 made an important enhancement to Multi-threading custom class loader.
- Previously, certain class loader were prone to deadlock.
- JDK7 modifies the lock mechanism.
- Previously as well, it was not an issue which adhere the acyclic delegation model.
- http://openjdk.java.net/groups/core-libs/ClassLoaderProposal.html
- `protected static boolean registerAsParallelCapable()`

Deadlock Scenario

```
Class Hierarchy:
  class A extends B
  class C extends D

ClassLoader Delegation Hierarchy:

Custom Classloader CL1:
  directly loads class A
  delegates to custom ClassLoader CL2 for class B

Custom Classloader CL2:
  directly loads class C
  delegates to custom ClassLoader CL1 for class D
```

```
Thread 1:
  Use CL1 to load class A (locks CL1)
    defineClass A triggers
      loadClass B (try to lock CL2)

Thread 2:
  Use CL2 to load class C (locks CL2)
    defineClass C triggers
      loadClass D (try to lock CL1)
```

JDK7 - Resolution

```
Class Hierarchy:
  class A extends B
  class C extends D

ClassLoader Delegation Hierarchy:

Custom Classloader CL1:
  directly loads class A
  delegates to custom ClassLoader CL2 for class B

Custom Classloader CL2:
  directly loads class C
  delegates to custom ClassLoader CL1 for class D
```

```
Thread 1:
  <if parallel capable>
  Use CL1 to load class A (locks CL1 + A)
    defineClass A triggers
      loadClass B (try to lock CL2 + B)

Thread 2:
  <if parallel capable>
  Use CL2 to load class C (locks CL2 + C)
    defineClass C triggers
      loadClass D (try to lock CL1 + D)
```

Investigating Class Loading Problems

- Can be issue with the class loaders implementation or the way you are using the classes.
- Not a bad idea to enhance toString() method with your cognitive learning.
- JDK toString() implementation may not be enough.
- Understand the class loading is fine or not with -verbose:classes

Issue 1 - ClassNotFoundException

```
java.lang.ClassNotFoundException: Class: com/madplanet/article/classloader/
part2/NotToBeFoundClass could not be found
    at com.madplanet.article.classloader.
    part2.MyClassLoader.findClass
    (MyClassLoader.java:41)
```

- This is an exception

Basic Debugging steps :-

- Find where the class loader loadClass() call get invoked.
- Figure out which class loader used and also all the parent classes.
- Figure out why class is not found by all the class loaders.

Basic Problems :-

- Source not added to the class loaders.
- Class loader parent is not set properly.
- Wrong class loader is used to the class.

Issue 2 - NoClassDefFoundError

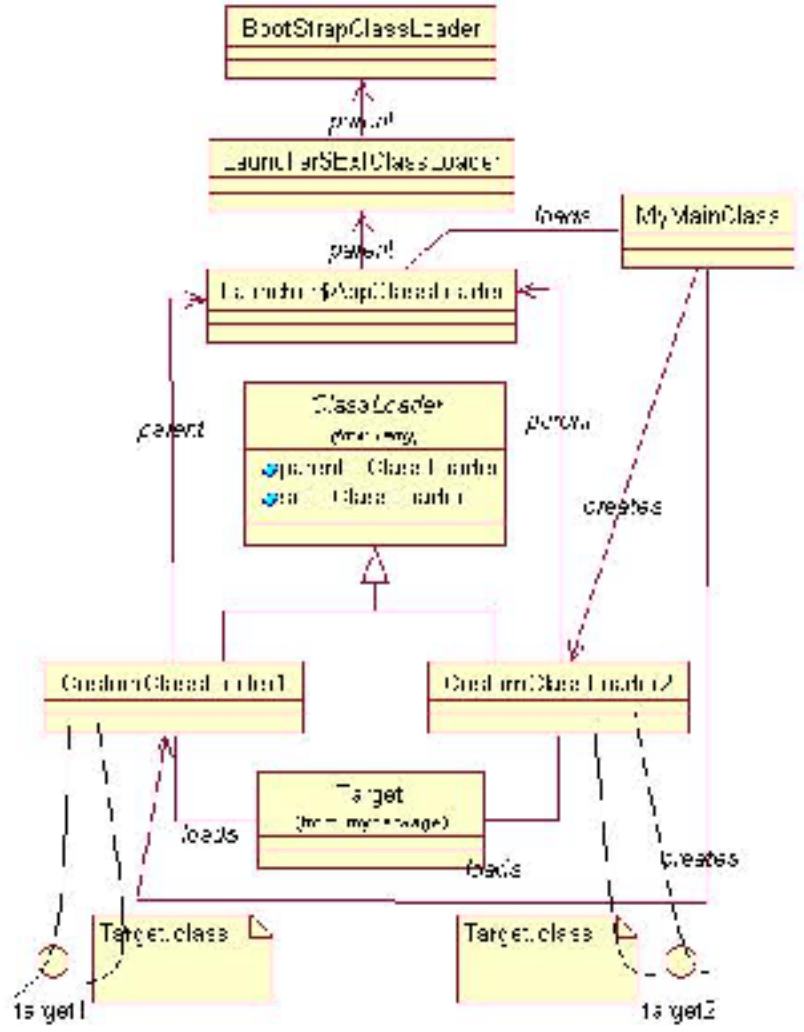
```
part2/NotToBeFoundClass could not be found
    at com.madplanet.article.classloader.
    part2.MyClassLoader.findClass
    (MyClassLoader.java:41)
```

- This is an error.

Basic Debugging steps :-
- Find where the class loader loadClass() call get invoked.
- Figure out which class loader used and also all the parent classes.
- Figure out why class is not found by all the class loaders.

Basic Problems :-
- Source not added to the class loaders.
- Class loader parent is not set properly.
- Wrong class loader is used to the class.

Issue 3 - `ClassCastException`




```
Target target3 = (Target) target2;
```

Lead to ClassCastException

*** shamelessly copied from o'reilly.com**

Project Jigsaw - Module Class loading

- Java SE mandates 2 class loaders.
  - VM Bootstrap class loader
  - System class loader
- In “classpath mode” JDK created 3 class loaders :-
  - VM Bootstrap class loader
  - Extension class loader
  - System class loader
- **In “module mode”, JDK creates m+1 class loaders.**

Project Jigsaw - Module Class loading

- Module Mode class loaders
  - VM bootstrap class loader
  - One loader per one or more modules.
    - A module loader is used to start an application
    - Module loaders load their dependencies, e.g. java.base, that are lazily created when it loads a class
    - A module loader has no parent but instead it does direct class lookup and delegates to the module loader that defines the referenced class
- No extension class loader anymore. Throw -Djava.ext.dir, if any in you code or args.

Important References

- Internals of Java class loading - Oreilly Reference - MUST READ
- Module Class loading - JDK9 - JDK9 interested parties.
- Javadoc for JDK8 - ClassLoader - MUST READ
- Java Language Specification - Chapter 12 - MUST READ

- For any issues, feel free to contact me (@vaibhav_c)
- Jigsaw related queries, feel free to post at - jigsaw-dev@openjdk.java.net (Please review it before posting)