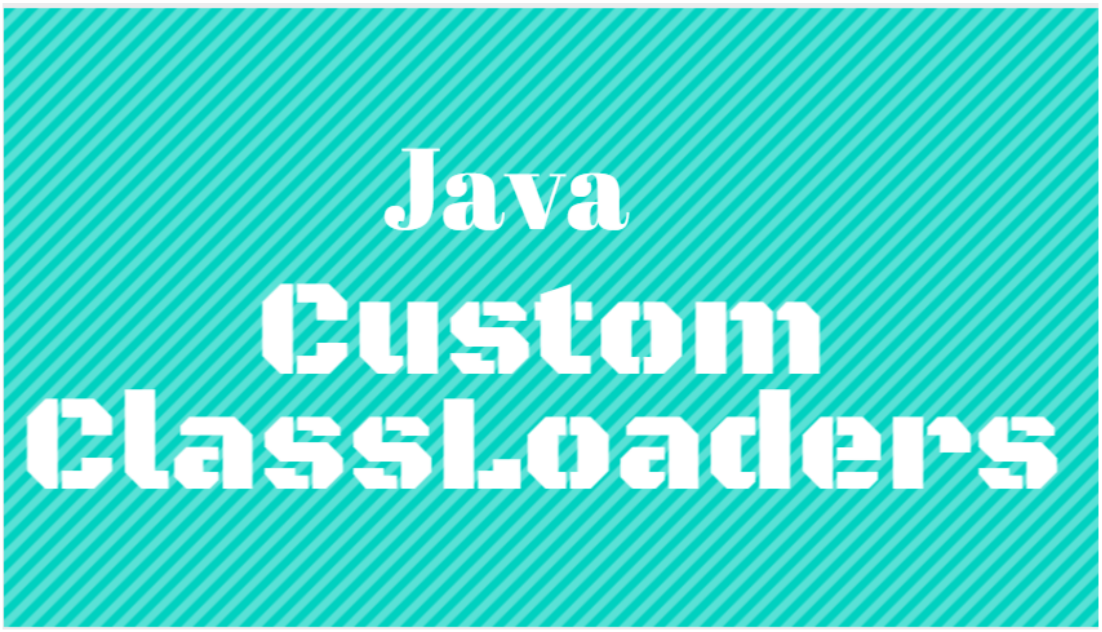
Custom ClassLoaders In Java

By **frugalis_blog** - January 25, 2018



We are going to Write our Own Custom ClassLoaders in Java, but before we start on this topic.Let's take a while to understand Why do We need Custom ClassLoaders?

We are going to check Simple Architecture of Class loaders In Java .

Why Custom ClassLoaders:-

As you know from my previous post JVM has its own ClassLoaders, but this classloader's loads files from a local file system or a hard drive location. Here are Some of the Use Cases Where we need ClassLoader's

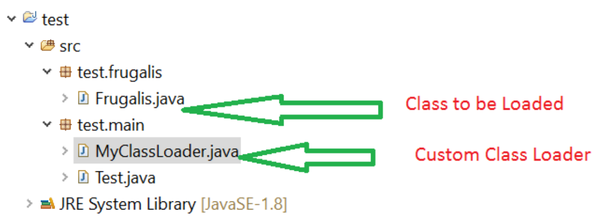
1. What if, Some of the Classes in your application is getting used for a finite period of time. How do we do Unloading of classes after a finite period of time? .This way it helps us for better memory management.
2. Load classes from anywhere Classes can be loaded from anywhere, for example, Database, Networks, or even define it on the fly.

Steps Writing Custom ClassLoaders:-

1. you need to extend your Custom ClassLoader from *java.lang.ClassLoader.*
2. Override the loadClass*()* method of ClassLoader, this method is responsible for loading our class.
3. If the Class is already loaded it returns that class
4. If the class that is requested to load is not present it delegates to parent class loader.
5. If Parent class loader is not able load the class, then loadClass*()* calls *findClass()* to find and load class
6. The class to be loaded must have a public constructor with no arguments.

Custom ClassLoader Example:-

1.Project Structure :-



2. Create a Simple Class:- We create a simple class **Frugalis.java**, this is the class we load here using our custom ClassLoader.

Frugalis.Java

```java
package test.frugalis;

public class Frugalis {

    public void printMyName(){
        System.out.println("Welcome Frugalis Minds");
    }
}
```

3. Custom ClassLoader :- We create Our custom ClassLoader which extends ClassLoader and override ***loadClass() .***

```java
package test.main;

import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;

public class MyClassLoader extends ClassLoader {

    public MyClassLoader(ClassLoader parent) {
        super(parent);
    }

    @Override
    public Class<?> loadClass(String name) throws ClassNotFoundException {
        System.out.println("Class Loading Started for " + name);
        if (name.startsWith("test")) {
            return getClass(name);
        }
        return super.loadClass(name);
    }

    /**
     * Loading of class from .class file
     * happens here You Can modify logic of
     * this method to load Class
     * from Network or any other source
     * @param name
     * @return
     * @throws ClassNotFoundException
     */
    private Class<?> getClass(String name) throws ClassNotFoundException {
        System.out.println("*********Inside getClass*********");

        String file = name.replace('.', File.separatorChar) + ".class";
        System.out.println("Name of File" + file);
        byte[] byteArr = null;
        try {
            // This loads the byte code data from the file
            byteArr = loadClassData(file);
            System.out.println("Size of byte array "+byteArr.length);
            Class<?> c = defineClass(name, byteArr, 0, byteArr.length);
            resolveClass(c);
            return c;
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Loads a given file and converts
     * it into a Byte Array
     * @param name
     * @return
     * @throws IOException
     */
    private byte[] loadClassData(String name) throws IOException {

        System.out.println("<<<<<<<<<Inside loadClassData>>>>>>");

        InputStream stream = getClass().getClassLoader().getResourceAsStream(
                name);
        int size = stream.available();
        byte buff[] = new byte[size];
        DataInputStream in = new DataInputStream(stream);
        // Reading the binary data
        in.readFully(buff);
        in.close();
        return buff;
    }

}
```

4. Running the Example:-

We create a Test.java and create an instance of our MyclassLoader, we load a binary class named as test*.frugalis.*Frugalis and invoke a method *printMyName .*

Test.java

```
package test.main;

import java.lang.reflect.InvocationTargetException;

public class Test {

    public static void main(String[] args) throws ClassNotFoundException, InstantiationException, Il

        MyClassLoader loader = new MyClassLoader(
                Test.class.getClassLoader());

        System.out.println("loader name---- " +loader.getParent().getClass().getName());

        //This Loads the Class we must always
        //provide binary name of the class
        Class<?> clazz =
                loader.loadClass("test.frugalis.Frugalis");

        System.out.println("Loaded class name: " + clazz.getName());

        //Create instance Of the Class and invoke the particular method
        Object instance = clazz.newInstance();

        clazz.getMethod("printMyName").invoke(instance);
    }
}
```

Output:-

```
loader name---- sun.misc.Launcher$AppClassLoader
Class Loading Started for test.frugalis.Frugalis
*********Inside getClass*********
Name of Filetest\frugalis\Frugalis.class
<<<<<<<<<Inside loadClassData>>>>>>
Size of byte array 519
Class Loading Started for java.lang.Object
Loaded class name: test.frugalis.Frugalis
Class Loading Started for java.lang.System
Class Loading Started for java.io.PrintStream
Welcome Frugalis Minds
```

5. Download Code :-

This was an example of simple example of Custom ClassLoader. Please add your comments and suggestions .

Related Posts-

1. Whats are ClassLoaders and internal working ?

frugalis_blog