[转载一篇写的不错的java ClassLoader工作机制的文章](http://blog.csdn.net/openxmpp/article/details/8564598)

2013-02-02 08:45 5301人阅读 [评论](http://blog.csdn.net/openxmpp/article/details/8564598#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/openxmpp/article/details/8564598#report)

 分类：

java（2）



原文： http://longdick.iteye.com/blog/442213

/\*\*

\*  转载请注明作者longdick    http://longdick.iteye.com

\*

\*/

java应用环境中不同的class分别由不同的ClassLoader负责加载。  
一个jvm中默认的classloader有Bootstrap ClassLoader、Extension ClassLoader、App ClassLoader，分别各司其职：

* **Bootstrap ClassLoader**
* 负责加载java基础类，主要是 %JRE\_HOME/lib/ 目录下的rt.jar、resources.jar、charsets.jar和class等
* **Extension ClassLoader**
* 负责加载java扩展类，主要是 %JRE\_HOME/lib/ext 目录下的jar和class
* **App ClassLoader**       负责加载当前java应用的classpath中的所有类。

其中Bootstrap ClassLoader是JVM级别的，由C++撰写；Extension ClassLoader、App ClassLoader都是java类，都继承自URLClassLoader超类。  
Bootstrap ClassLoader由JVM启动，然后初始化sun.misc.Launcher ，sun.misc.Launcher初始化Extension ClassLoader、App ClassLoader。

下图是ClassLoader的加载类流程图，以加载一个类的过程类示例说明整个ClassLoader的过程。

 Bootstrap ClassLoader、Extension ClassLoader、App ClassLoader三者的关系如下：

Bootstrap ClassLoader是Extension ClassLoader的parent，Extension ClassLoader是App ClassLoader的parent。

但是这并不是继承关系，只是语义上的定义，基本上，每一个ClassLoader实现，都有一个Parent ClassLoader。

可以通过ClassLoader的getParent方法得到当前ClassLoader的parent。Bootstrap ClassLoader比较特殊，因为它不是java class所以Extension ClassLoader的getParent方法返回的是NULL。

了解了ClassLoader的原理和流程以后，我们可以试试自定义ClassLoader。

**关于自定义ClassLoader：**

由于一些特殊的需求，我们可能需要定制ClassLoader的加载行为，这时候就需要自定义ClassLoader了.

自定义ClassLoader需要继承ClassLoader抽象类，重写findClass方法，这个方法定义了ClassLoader查找class的方式。

主要可以扩展的方法有：

findClass          定义查找Class的方式

defineClass       将类文件字节码加载为jvm中的class

findResource    定义查找资源的方式

如果嫌麻烦的话，我们可以直接使用或继承已有的ClassLoader实现，比如

* java.net.URLClassLoader
* java.security.SecureClassLoader
* java.rmi.server.RMIClassLoader
* sun.applet.AppletClassLoader

Extension ClassLoader 和 App ClassLoader都是java.net.URLClassLoader的子类。

这个是URLClassLoader的构造方法：

public URLClassLoader(URL[] urls, ClassLoader parent)

public URLClassLoader(URL[] urls)

urls参数是需要加载的ClassPath url数组，可以指定parent ClassLoader，不指定的话默认以当前调用类的ClassLoader为parent。

代码示例：

**Java代码**



1. ClassLoader classLoader = **new** URLClassLoader(urls);
2. Thread.currentThread().setContextClassLoader(classLoader);
3. Class clazz=classLoader.loadClass("com.company.MyClass");//使用loadClass方法加载class,这个class是在urls参数指定的classpath下边。
5. Method taskMethod = clazz.getMethod("doTask", String.**class**, String.**class**);//然后我们就可以用反射做些事情了
6. taskMethod.invoke(clazz.newInstance(),"hello","world");

由于classloader 加载类用的是全盘负责委托机制。*所谓全盘负责，即是当一个classloader加载一个Class的时候，这个Class所依赖的和引用的所有 Class也由这个classloader负责载入，除非是显式的使用另外一个classloader载入。*

所以，当我们自定义的classloader加载成功了com.company.MyClass以后，MyClass里所有依赖的class都由这个classLoader来加载完成。

自定义ClassLoader在某些应用场景还是比较适用，特别是需要灵活地动态加载class的时候。

下面这篇文章列出了其中一种自定义ClassLoader的应用场景，有兴趣的同学可以参考下：

### [运行jar应用程序引用其他jar包的四种方法](http://longdick.iteye.com/blog/332580)

**博客分类：**

* [java 基础](http://longdick.iteye.com/category/85447)

[Java](http://www.iteye.com/blogs/tag/Java)[EXT](http://www.iteye.com/blogs/tag/EXT)[Eclipse](http://www.iteye.com/blogs/tag/Eclipse)[Unix](http://www.iteye.com/blogs/tag/Unix)[Tomcat](http://www.iteye.com/blogs/tag/Tomcat)

大家都知道一个java应用项目可以打包成一个jar，当然你必须指定一个拥有main函数的main class作为你这个jar包的程序入口。

具体的方法是修改jar包内目录META-INF下的MANIFEST.MF文件。

比如有个叫做test.jar的jar包，里面有一个拥有main函数的main class：test.someClassName

我们就只要在MANIFEST.MF里面添加如下一句话：

Main-Class: test.someClassName

 然后我们可以在控制台里输入java -jar  test.jar即可以运行这个jar。

但是我们这个项目需要引用其他第三方的jar包，在eclipse里面以项目jar包的形式引用了这个叫做some.jar的包，当时放在项目的lib子目录下，最后项目打包时把这个some.jar也打进来了，但是用java -jar执行这个test.jar的时候报找不到Class异常，原因就是jar引用不到放在自己内部的jar包。

那怎么办？

运行时将其加入classpath的方式行不行？就是在运行jar的同时加入classpath参数：

java -classpath some.jar  -jar test.jar

这种方式是不行的，因为使用classpath指定的jar是由AppClassloader来加载，java 命令 加了-jar 参数以后，AppClassloader就只关注test.jar范围内的class了，classpath参数失效。

那该怎么引用其他的jar包呢？

**方法一、使用Bootstrap Classloader来加载这些类。**

我们可以在运行时使用如下参数：

-Xbootclasspath:完全取代系统Java classpath.最好不用。  
-Xbootclasspath/a: 在系统class加载后加载。一般用这个。  
-Xbootclasspath/p: 在系统class加载前加载,注意使用，和系统类冲突就不好了.

win32     java -Xbootclasspath/a: some.jar;some2.jar;  -jar test.jar

unix          java -Xbootclasspath/a: some.jar:some2.jar:  -jar test.jar

win32系统每个jar用分号隔开，unix系统下用冒号隔开

**方法二、使用Extension Classloader来加载**

你可以把需要加载的jar都扔到%JRE\_HOME%/lib/ext下面，这个目录下的jar包会在Bootstrap Classloader工作完后由Extension Classloader来加载。非常方便，非常省心。:)

**方法三、还是用AppClassloader来加载，不过不需要classpath参数了**

我们在MANIFEST.MF中添加如下代码：

Class-Path: lib/some.jar

lib是和test.jar同目录的一个子目录，test.jar要引用的some.jar包就在这里面。

然后测试运行，一切正常！

如果有多个jar包需要引用的情况：

Class-Path: lib/some.jar lib/some2.jar

每个单独的jar用空格隔开就可以了。注意使用相对路径。

另：如果META-INF 下包含INDEX.LIST文件的话，可能会使Class-Path配置失效。INDEX.LIST是Jar打包工具打包时生成的索引文件，删除对运行不产生影响。

**方法四、自定义Classloader来加载**

这种方法是终极解决方案，基本上那些知名java应用都是那么干的，如tomcat、jboss等等。

这种方式有点复杂，需要专门开贴讨论.

**总结：**

以上四种方法都可以用，特别是程序运行在非常单纯的环境中时。但是，如果是运行在多任务，多应用的环境中时，最好每个应用都能相互独立，第一种和第二种方案都有可能对其他应用产生影响，因此最好就是选择第三种和第四种。