一、三种类加载器及相应类加载范围

类加载器通常可以分为三种：

启动类加载器(BootstrapClassLoader)

扩展类加载器(ExtClassLoader)

应用程序类加载器(AppClassLoader）

1、启动类加载器

启动类加载器是由c++实现的，是虚拟机的一部分，主要负责加载jvm自身需要的类，即负责加载JAVAHOME下的核心类库。打印下启动类加载器的加载路径，代码如下：

URLClassPath path = Launcher.*getBootstrapClassPath*();  
for(URL url : path.getURLs()){  
 *log*.info(url.getPath());  
}

输出结果

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/resources.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/rt.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/sunrsasign.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/jsse.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/jce.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/charsets.jar

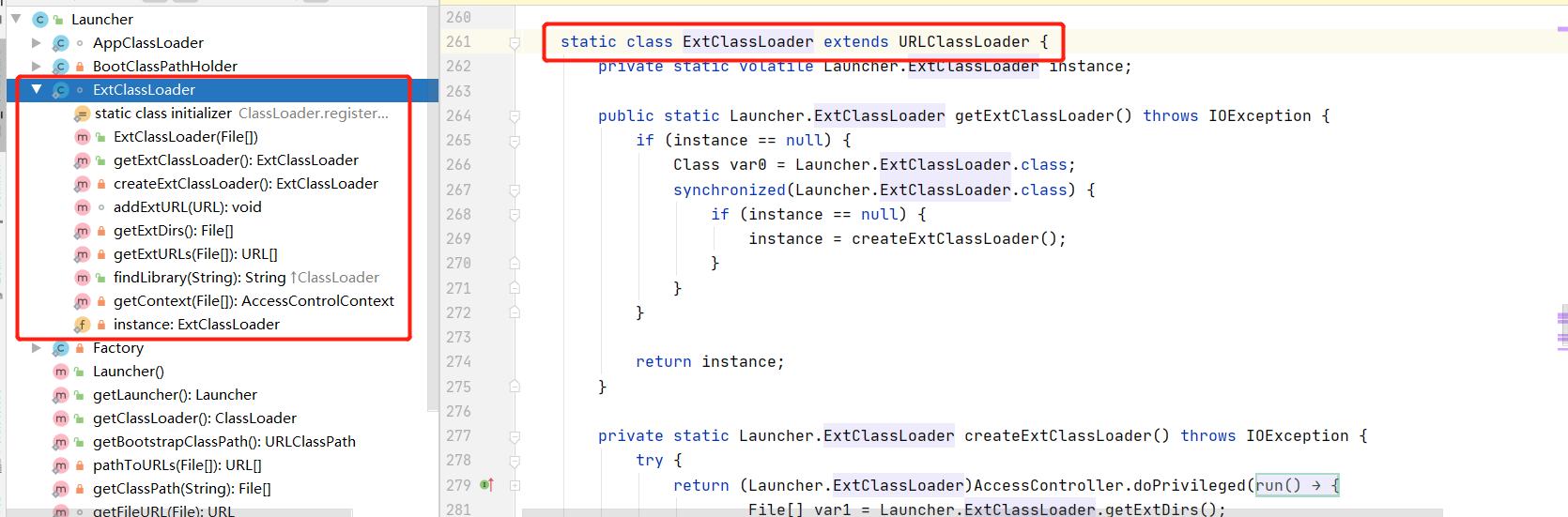
/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/jfr.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/classes

启动类加载是c++实现的，所以没有继承java中的ClassLoder类。接下来的扩展类加载器和应用程序类加载器都继承了ClassLoder

2、扩展类加载器

扩展类加载器是由java实现的，具体实现类是sun.misc. Launcher$ExtClassLoader，如下图：



让我们打印一下扩展类加载器的实例化对象来确认下

URLClassLoader extClassLoader = (URLClassLoader)ClassLoader.*getSystemClassLoader*().getParent();  
System.*out*.println(extClassLoader);

输出结果：sun.misc.Launcher$ExtClassLoader@1753acfe

打印下扩展类加载器的加载路径，代码如下：

URLClassLoader extClassLoader = (URLClassLoader)ClassLoader.*getSystemClassLoader*().getParent();  
for(URL url : extClassLoader.getURLs()) {  
 *log*.info(url.getPath());  
}

输出结果：

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/access-bridge-64.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/cldrdata.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/dnsns.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/jaccess.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/jfxrt.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/localedata.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/nashorn.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/sunec.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/sunjce\_provider.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/sunmscapi.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/sunpkcs11.jar

/C:/Program%20Files/Java/jdk1.8.0\_201/jre/lib/ext/zipfs.jar

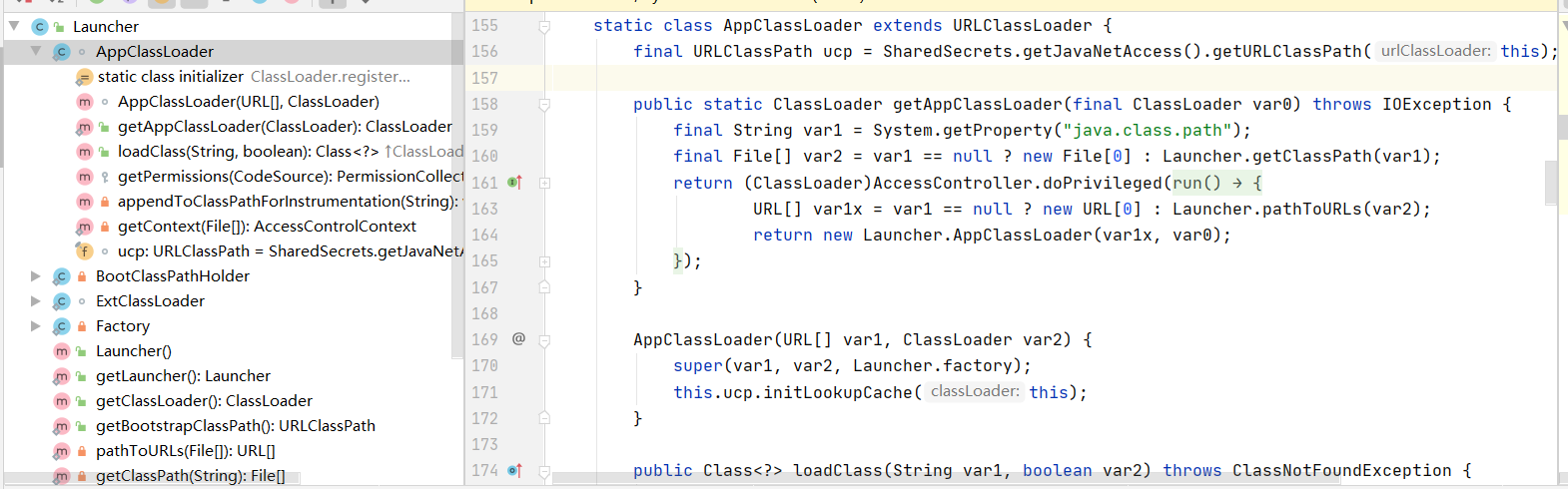
可以看到/jdk1.8.0\_201/jre/lib/ext这个目录下的所有jar包都被加载进来了，那问题来了，扩展类加载器是负责加载这个路径下的所有jar包，还是只负责规定好的jar包呢，让我们试一下，在/jdk1.8.0\_201/jre/lib/ext这个路径下放一个其他的jar包，我放了一个自己的的工具类jar包mg-common-1.0.0.jar到这个目录下，再次执行上边的代码，发现输出结果中包含了新加的jar包。所以**扩展类加载器负责加载/jdk1.8.0\_201/jre/lib/ext这个目录下的所有jar包。**

**注意：扩展类加载器的父级是启动类加载器，不过通过扩展类加载器的**getParent()方法取到的返回结果为null，这是因为启动类加载器不是java实现的。

3、应用程序类加载器

应用程序类记载器同样也是java代码实现的，类加载器是sun.misc. Launcher$AppClassLoader

的实例对象，如下图：



可以通过如下代码来确认

URLClassLoader appClassLoader = (URLClassLoader)ClassLoader.*getSystemClassLoader*();  
System.*out*.println(appClassLoader);

输出结果：sun.misc.Launcher$AppClassLoader@18b4aac2

输出下系统默认的应用程序类加载的加载路径，如下：

URLClassLoader appClassLoader = (URLClassLoader)ClassLoader.*getSystemClassLoader*();  
for(URL url : appClassLoader.getURLs()) {  
 *log*.info(url.getPath());  
}

这个输出结果太多，就不列出来了，感兴趣的可以自己尝试下。

获取应用程序加载器的代码中，我们获取的ClassLoader.*getSystemClassLoader*()这个方法返回的类加载器，这个函数的加载器是从哪里来得呢，可以看下ClassLoder这个类中initSystemClassLoader函数，获取的是sun.misc. Launcher这个类的加载器

private static synchronized void initSystemClassLoader() {  
 if (!*sclSet*) {  
 if (*scl* != null)  
 throw new IllegalStateException("recursive invocation");  
 sun.misc.Launcher l = sun.misc.Launcher.*getLauncher*();  
 if (l != null) {  
 Throwable oops = null;  
 *scl* = l.getClassLoader();  
 try {  
 *scl* = AccessController.*doPrivileged*(  
 new SystemClassLoaderAction(*scl*));  
 } catch (PrivilegedActionException pae) {  
 oops = pae.getCause();  
 if (oops instanceof InvocationTargetException) {  
 oops = oops.getCause();  
 }  
 }  
 if (oops != null) {  
 if (oops instanceof Error) {  
 throw (Error) oops;  
 } else {  
 *// wrap the exception* throw new Error(oops);  
 }  
 }  
 }  
 *sclSet* = true;  
 }  
}

应用程序的类加载器的父级是扩展类加载器可以通过getParent（）函数获取到。

现在三种类加载器准备好了，那为什么要有三种类加载器，如果只有一种类加载器会有什么问题呢，接来下让我们一起来看下双亲委派模式吧。

三、双亲委派模式