

# 深入浅出Java的类加载机制



六尺帐篷 (/u/f8e9b1c246f1)  
2017.07.17 23:15 字数 1756 阅读 246 评论 1 喜欢 9  
(/u/f8e9b1c246f1)

[编辑文章 \(/writer#/notebooks/10012174/notes/14607911\)](#)

Java只有当需要使用类的时候，才会将类载入。java类的载入，是通过类加载器进行的。

在命令模式下，执行java 指令之后，java程序会找到JRE安装的所在目录，然后找到jvm.dll，通过他启动JVM并进行虚拟机的初始化的操作。  
JVM启动并初始化完成之后，就会产生Bootstrap Loader这个类加载器，这个类加载器通常是c或c++写的。Bootstrap Loader随后又会继续载入Extended Loader这个类加载器，并且设定Extended Loader的父加载器为自己Bootstrap Loader。接着Bootstrap Loader继续载入System Loader，并且将System Loader的parent设定为Extended Loader。

Bootstrap Loader通常由C语言写成；Extended Loader是由Java所成，如果是Sun JDK，实际是对应sun.misc.Launcher\$ExtClassLoader（Launcher中的内部类别）；System Loader是由Java成，实际对应于sun.misc.Launcher\$AppClassLoader（Launcher中的内部类别）。

加载器的加载过程如下图描述：

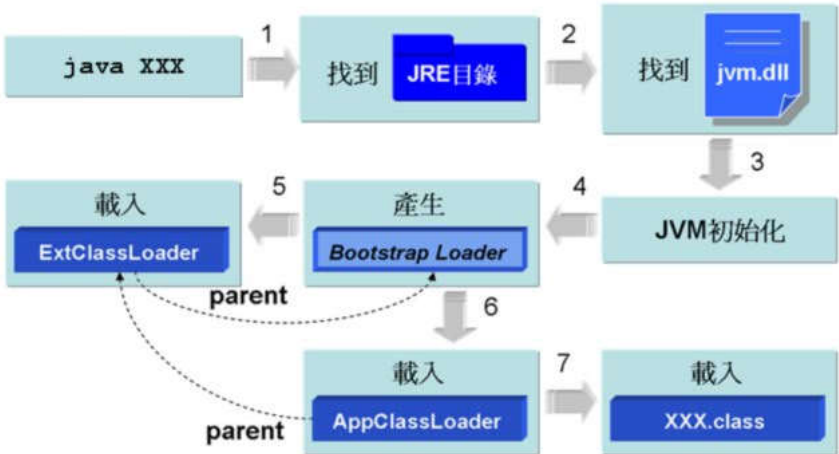
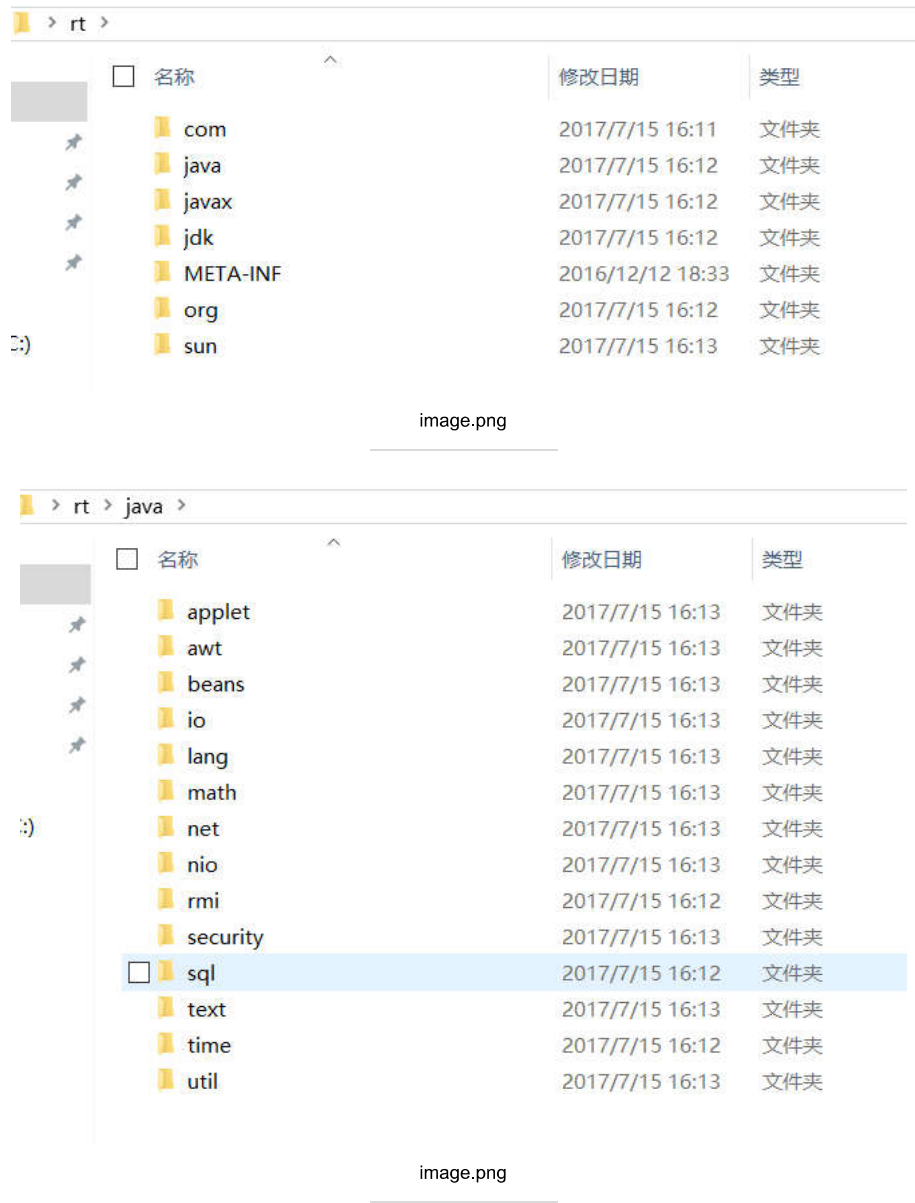


image.png

Bootstrap Loader加载器进行类加载时会搜索sun.boot.class.path中指定位置中的类，一般预设路径是在JRE所在目录的classes下之.class文件，或者lib目录下.jar档案中（例如rt.jar）中的类载入。可以使用System.getProperty("sun.boot.class.path")来获取sun.boot.class.path中指定的路径，例如在我的笔记本中显示的是以下的路径：

```
C:\Program Files\Java\jre1.8.0_131\lib\resources.jar
C:\Program Files\Java\jre1.8.0_131\lib\rt.jar
C:\Program Files\Java\jre1.8.0_131\lib\sunrsasign.jar
C:\Program Files\Java\jre1.8.0_131\lib\jsse.jar
C:\Program Files\Java\jre1.8.0_131\lib\jce.jar
C:\Program Files\Java\jre1.8.0_131\lib\charsets.jar
C:\Program Files\Java\jre1.8.0_131\lib\jfr.jar
C:\Program Files\Java\jre1.8.0_131\classes
```

我们解压rt.jar看下里面到底是哪些class



我们可以很熟悉这个结构了，可以确认Bootstrap Loader类加载器加载的基本是java api 的内容，基础类库都在这里。

Extended Loader (`sun.misc.Launcher$ExtClassLoader`) 是由Java写的，会搜索 `java.ext.dirs` 中指定位置的类，一般初始设定是JRE目录下的`lib\ext\classes`目录下的.class，或`lib\ext`目录下的.jar包中的类。可以使用`System.getProperty("java.ext.dirs")` 来显示`java.ext.dirs`中指定的路径，例如在我的笔记本中显示的是以下的路径：

```
C:\Program Files\Java\jre1.8.0_131\lib\ext
C:\Windows\Sun\Java\lib\ext
```

正如这个加载器的名字所示，它主要负责java的一些扩展类库。

System Loader (`sun.misc.Launcher$AppClassLoader`) 是由Java写的，会查找系统参数`java.class.path`中指定位置中的类，也就是Classpath所指定的路径，初始设定是目前工作路径下的.class类。可以使用`System.getProperty("java.class.path")`来显示`java.class.path`中指定的路径，在使用java执行java程序时，也可以加上`-cp`来覆盖原有的Classpath设置。

例如，我们编写一个简单的java程序输出系统加载器的路径：

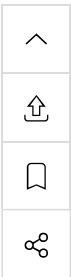




image.png

我们发现系统加载器会去加载当前程序project所在的class路径。

我们来总结一下类加载器载入类的过程。

- Bootstrap Loader会在JVM启动和初始化之后产生
- 随后它会载入Extended Loader并将其parent设为Bootstrap Loader
- 然后Bootstrap Loader再载入System Loader并将其parent设定为ExtClassLoader
- 接着System Loader开始载入您指定的类

在载入类时，每个类载入器会先将载入类的任务交由他的parent去执行，如果parent找不到，才由转到由自己载入，所以在载入类时，会以Bootstrap Loader→Extended Loader→System Loader的顺序来寻找类，如果都找不到，就会丢出NoClassDefFoundError。

类别载入器在Java中是以java.lang.ClassLoader类型存在，每一个类被载入后，都会有一个Class的实例来代表这个类，而每个由这个类生成的实例都会记得自己是由哪个ClassLoader载入的，java中提供了getClass方法可以从实例获取它的class实例，然后可以由Class的getClassLoader()获得载入该类的ClassLoader，而从ClassLoader的getParent()方法可以取得自己的parent。

假如我们有一个someclass类，那么类别载入器之间的关系是：

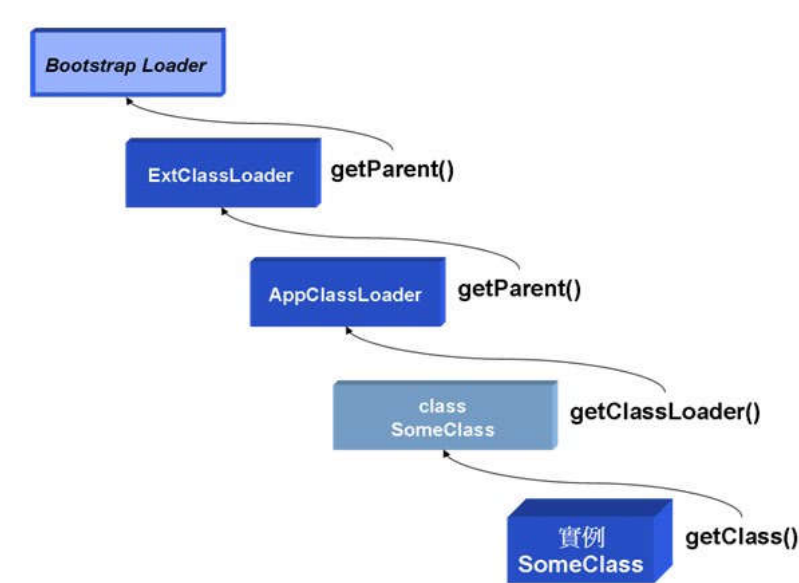
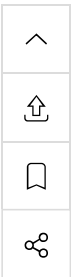


image.png



我们可以写个程序来获取载入器的信息

```
// 取得SomeClass的Class实例
Class c = Class.forName("SomeClass");
// 取得ClassLoader
ClassLoader loader = c.getClassLoader();
System.out.println(loader);
// 取得父ClassLoader
System.out.println(loader.getParent());
// 再取得父ClassLoader
System.out.println(loader.getParent().getParent());
```

所展示的结果是:

```
sun.misc.Launcher$AppClassLoader@19821f
sun.misc.Launcher$ExtClassLoader@adbf1
null
```

SomeClass 是个自定义的类, 你在目前的工作目录下执行java程序, 首先 AppClassLoader会将载入的任务交给ExtClassLoader, 而 ExtClassLoader会将载入的类交给Bootstrap Loader, 由于Bootstrap Loader在它的路径设定 (sun.boot.class.path) 下找不到类, 所以由ExtClassLoader来寻找, 而 ExtClassLoader在它的的路径中 (java.ext.dirs) 下也找不到该类, 所以由 AppClassLoader来寻找, AppClassLoader最后在Classpath (java.class.path) 设定的路径寻找载入的类, 并且找到了, 如果还没找到就会抛出classnotfoundexception。

载入 SomeClass的ClassLoader是AppClassLoader, 而AppClassLoader的parent是 ExtClassLoader, 而ExtClassLoader的parent是null, null并不是表示ExtClassLoader没有设置 parent, 而是因为Bootstrap Loader通常由C写的, 所以在Java中并没有一个实际的类别来表示它, 所以才会显示为null。

如果把SomeClass的.class文件移至JRE目录下的lib\ext\classes下, 并重新执行程序, 你会看到以下的信息:

```
sun.misc.Launcher$ExtClassLoader@adbf1
null
Exception in thread "main" java.lang.NullPointerException
    at Main.main(Main.java:12)
```

由于这次可以在extclassloader指定的路径找到, 所以会有extclassloader来载入 someclass的实例, 而extclassloader的parent为null, 指的是他的父亲是 bootstraploader。

我们再实验, 将class文件放到bootstraploader的目录下, 也就是jre的classes目录下, 那么运行结果就会如下:

```
null
Exception in thread "main" java.lang.NullPointerException
    at Main.main(Main.java:10)
```

这次直接就可以在bootstrap中找到了所以就显示null。

**loadclass方法载入实例时, 不会执行静态区域, 而是会等到真正使用类来初始化实例的时候的执行**

## 使用自己的类别载入器

由同一个classloader载入的类, 会只有一个class对象的实例, 如果同一个类是由不同的 loader载入的话, 就会有两个不同的classloader实例。注意这个说法。如果两个不同的类别在搜索同一个类, 如果在appclassloader就搜到的话, 那么class实例就只有一个。



如果上面三个loader都找不到，最后在各自的classloader里找到了，class实例才会有两份。

在不同的环境中，應用程式可能會設定自己的類別載入器，例如在Tomcat的類別載入器，會找尋Tomcat目錄中lib中的jar檔案之類別，而Web應用程式也會從WEB-INF的lib中找尋jar檔案，以及從WEB-INF/classes中找尋.class檔，搞清楚類別載入器載入檔案的位置與順序，遇到ClassNotFoundException或是NoClassDefFoundError時，才會知道要在哪邊確認類別檔案是否存在。

Java (/nb/10012174)

© 著作权归作者所有




六尺帐篷 (/u/f8e9b1c246f1)


写了 245418 字，被 15240 人关注，获得了 1429 个喜欢

(/u/f8e9b1c246f1)

如果觉得我的文章对您有用，请随意赞赏。您的支持将鼓励我继续创作！


赞赏支持

 喜欢 9



更多分享

(http://cwb.assets.jianshu.io/notes/images/146079'



写下你的评论...

1条评论

只看作者 关闭评论

按喜欢排序 按时间正序 按时间倒序



wxweven (/u/29e5e35e2689)

2楼 · 2017.08.20 10:49

(/u/29e5e35e2689)


你这个类加载图不太对哦，ext的parent并不是bootstrap，而是null

 1人赞

 回复


被以下专题收入，发现更多相似内容

投稿管理

- + 收入我的专题
-  Android知识 (/c/3fde3b545a35?utm\_source=desktop&utm\_medium=notes-included-collection)

 Android... (/c/5139d555c94d?utm\_source=desktop&utm\_medium=notes-included-collection)

 Android开发 (/c/d1591c322c89?utm\_source=desktop&utm\_medium=notes-included-collection)

 程序员 (/c/NEt52a?utm\_source=desktop&utm\_medium=notes-included-collection)

 首页投稿 (/c/bDHhpK?utm\_source=desktop&utm\_medium=notes-included-collection)

