类加载及类加载相关

类加载 及 类加载器相关

三种类加载器

jvm内置类加载器

我们想要查看Appclassloader具体加载了哪些路径的类，可以使用一下方法。

```
     String classpath = System.getProperty("java.class.path");
            System.out.println(classpath);
            for (String path : classpath.split(";")) {
                System.out.println(path);
            }
```

输出的值为

```
    F:\apache-maven-3.1.1\repo\org\junit\platform\junit-platform-
launcher\1.3.1\junit-platform-launcher-1.3.1.jar
    F:\apache-maven-3.1.1\repo\org\apiguardian\apiguardian-api\1.0.0\apiguardian-
api-1.0.0.jar
    F:\apache-maven-3.1.1\repo\org\junit\platform\junit-platform-
engine\1.3.1\junit-platform-engine-1.3.1.jar
    F:\apache-maven-3.1.1\repo\org\junit\platform\junit-platform-
commons\1.3.1\junit-platform-commons-1.3.1.jar
    F:\apache-maven-3.1.1\repo\org\opentest4j\opentest4j\1.1.1\opentest4j-
1.1.1.jar
    F:\apache-maven-3.1.1\repo\org\junit\jupiter\junit-jupiter-engine\5.3.1\junit-
jupiter-engine-5.3.1.jar
    F:\apache-maven-3.1.1\repo\org\junit\jupiter\junit-jupiter-api\5.3.1\junit-
jupiter-api-5.3.1.jar
    F:\apache-maven-3.1.1\repo\org\junit\vintage\junit-vintage-engine\5.3.1\junit-
vintage-engine-5.3.1.jar
    F:\apache-maven-3.1.1\repo\junit\junit\4.12\junit-4.12.jar
    F:\apache-maven-3.1.1\repo\org\hamcrest\hamcrest-core\1.3\hamcrest-core-
1.3.jar
    D:\java8\jdk8\jre\lib\charsets.jar
    D:\java8\jdk8\jre\lib\deploy.jar
    D:\java8\jdk8\jre\lib\ext\access-bridge-64.jar
    D:\java8\jdk8\jre\lib\ext\cldrdata.jar
    D:\java8\jdk8\jre\lib\ext\dnsns.jar
    D:\java8\jdk8\jre\lib\ext\jaccess.jar
    D:\java8\jdk8\jre\lib\ext\jfxrt.jar
    D:\java8\jdk8\jre\lib\ext\localedata.jar
    D:\java8\jdk8\jre\lib\ext\nashorn.jar
    D:\java8\jdk8\jre\lib\ext\sunec.jar
    D:\java8\jdk8\jre\lib\ext\sunjce_provider.jar
```

```
    D:\java8\jdk8\jre\lib\ext\sunmscapi.jar
    D:\java8\jdk8\jre\lib\ext\sunpkcs11.jar
    D:\java8\jdk8\jre\lib\ext\zipfs.jar
    D:\java8\jdk8\jre\lib\javaws.jar
    D:\java8\jdk8\jre\lib\jce.jar
    D:\java8\jdk8\jre\lib\jfr.jar
    D:\java8\jdk8\jre\lib\jfxswt.jar
    D:\java8\jdk8\jre\lib\jsse.jar
    D:\java8\jdk8\jre\lib\management-agent.jar
    D:\java8\jdk8\jre\lib\plugin.jar
    D:\java8\jdk8\jre\lib\resources.jar
    D:\java8\jdk8\jre\lib\rt.jar
    D:\myworkspace\note\note\target\test-classes
    D:\myworkspace\note\note\target\classes
    F:\apache-maven-3.1.1\repo\org\slf4j\slf4j-api\1.7.25\slf4j-api-1.7.25.jar
    F:\apache-maven-
3.1.1\repo\org\apache\httpcomponents\httpclient\4.5.5\httpclient-4.5.5.jar
    F:\apache-maven-3.1.1\repo\org\apache\httpcomponents\httpcore\4.4.9\httpcore-
4.4.9.jar
    F:\apache-maven-3.1.1\repo\commons-logging\commons-logging\1.2\commons-
logging-1.2.jar
    F:\apache-maven-3.1.1\repo\commons-codec\commons-codec\1.10\commons-codec-
1.10.jar
    F:\apache-maven-3.1.1\repo\redis\clients\jedis\2.9.0\jedis-2.9.0.jar
    F:\apache-maven-3.1.1\repo\org\apache\commons\commons-pool2\2.4.2\commons-
pool2-2.4.2.jar
    D:\Program Files\IntelliJ IDEA 2018.2.1\lib\idea_rt.jar
```

总结一下： Appclassloader 加载了:

```
1.pom文件中的jar包中的类
2.类路径中jre\lib\和jre\lib\ext中的jar包中的类（由于双亲委托机制，jre\lib\ext中的会交给
extclassloader加载）
3.当前项目中所有的类
```

ExtClassLoader的加载路径

```
String extDirs = System.getProperty("java.ext.dirs");
            for (String path : extDirs.split(";")) {
                System.out.println(path);
            }
```

输出结果为：

```
D:\java8\jdk8\jre\lib\ext
C:\WINDOWS\Sun\Java\lib\ext
```

BootstrapClassloader的加载路径

```
```

```
URL[] urLs = sun.misc.Launcher.getBootstrapClassPath().getURLs();
      for (URL url : urLs) {
          System.out.println(url.toExternalForm());
      }
```

输出结果为：

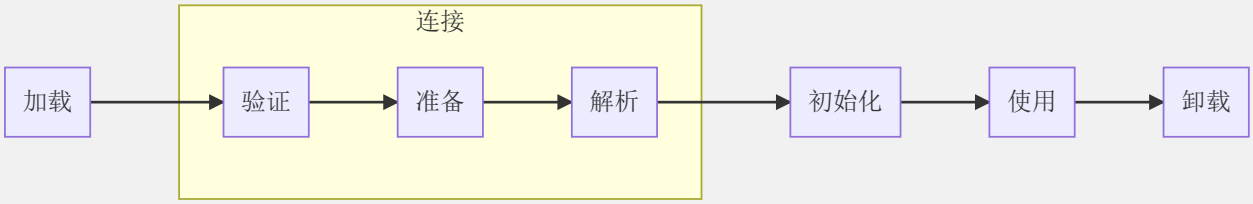
```
file:/D:/java8/jdk8/jre/lib/resources.jar
file:/D:/java8/jdk8/jre/lib/rt.jar
file:/D:/java8/jdk8/jre/lib/sunrsasign.jar
file:/D:/java8/jdk8/jre/lib/jsse.jar
file:/D:/java8/jdk8/jre/lib/jce.jar
file:/D:/java8/jdk8/jre/lib/charsets.jar
file:/D:/java8/jdk8/jre/lib/jfr.jar
file:/D:/java8/jdk8/jre/classes
```

总结： bootstrap类加载器主要负责jre/lib下的所有jar包

类加载相关

类加载的时机

类从被加载到虚拟机开始，到卸出到内存位置，整个生命周期包括了



什么情况下开始类加载过程的的第一个阶段: 加载，虚拟机中没有强行约束。 **但是对于初始化阶段,虚拟机规定类有且只有四中情况必须对类进行 初始化**(加载、验证、准备自然在此之前进行)

1. 遇到new、getstatic、 putstatic、 invokestatic这四条指令的时候，如果类没有初始化则先触发初始化。 生成这四条指令最常见的java场景是：
    - new实例化对象
    - 读取或者设置一个类的静态字段（被final修饰、已在编译器把结果放入常量池的静态字段除外）
    - 调用一个类的静态方法的时候
2. 使用 java.lang.reflect包的方法对类进行反射调用的时候，如果类没有初始化，则需要先触发初始化
3. 当初始化一个类的时候，如果发现父类还没有初始化，则需要先触发父类的初始化
4. 当虚拟机启动时，用户需要指定一个要执行的主类(包含 main() 方法的那个类)， 虚拟机会先初始化这个类

这四种场景成为对一个类的主动引动，除此之外的所有引用方法，都不会被触发初始化，成为被动引用。