Java Class loader

## 动态加载

Java基础类库是一次性载入的，因为基础库中包含java程序执行所必需的类。其它所有的非基础类都是在第一次被用到时才动态加载到JVM的。这句话有两层含义：

1. Java程序在运行时并不一定被完整加载，只有当发现该类还没有加载时，才去本地或远程查找类的.class文件并验证和加载；
2. 当程序创建了第一个对类的静态成员的**引用**（如类的静态变量、静态方法、构造方法——构造方法也是静态的）时，才会加载该类。仅仅声明一个类型是不会被加载的；
3. 如果该类有父类，载入该类之前应确保父类加载，即父类是在子类之前加载，这同样遵循委托机制；
4. 如果要加载的类A中**引用**了类B，Java虚拟机将使用加载类A的类加载器来加载类B。

## 加载原则

Java类加载器基于三个原则：委托、可见性、唯一性。委托原则：把加载类的请求转发给父 类加载器，而且仅加载类当父类加载器无法找到或者不能加载类时；可见性原则：允许子类加载器查看由父类加载器加载的所有的类，但是父类加载器不能查看由子类加载器加载的类；唯一性原则：只允许加载一次类文件，这基本上是通过委托原则来实现的并确保子类加载器不重新加载由父类加载器加载过的类。

## 加载原理

Java中使用的默认类加载器有以下三种：Bootstrap , Extension以及System or Application class loader。每个类加载器都有一个预定义的位置，它们在那里加载类文件。

1） Bootstrap ClassLoader - JRE/lib/rt.jar，负责加载JDK中的核心类库，如：rt.jar、resources.jar、charsets.jar；

2）Extension ClassLoader - JRE/lib/ext 或者任何指向java.ext.dirs的路径，负责加载Java的扩展类库；

3）Application ClassLoader - CLASSPATH环境变量、-classpath or -cp 命令行选项，JAR内部清单文件的类路径属性，负责加载应用程序classpath目录下的所有jar和class文件。

ClassLoader使用的是**双亲委托模型**来搜索类的，每个ClassLoader实例都有一个父类加载器的引用（不是继承的关系，是一个包含的关系）。当一个ClassLoader实例需要加载某个类时，它会试图亲自搜索某个类之前，先把这个任务委托给它的父类加载器，这个过程是由上至下依次检查的，首先由最顶层的类加载器Bootstrap ClassLoader试图加载，如果没加载到，则把任务转交给Extension ClassLoader试图加载，如果也没加载到，则转交给App ClassLoader 进行加载，如果它也没有加载得到的话，则返回给委托的发起者，由它到指定的文件系统或网络等URL中加载该类。如果它们都没有加载到这个类时，则抛出ClassNotFoundException异常。

因为这样可以避免重复加载，当父亲已经加载了该类的时候，就没有必要子ClassLoader再加载一次。考虑到安全因素，我们试想一下，如果不使用这种委托模式，那我们就可以随时使用自定义的String来动态替代java核心API中定义的类型，这样会存在非常大的安全隐患，而双亲委托的方式，就可以避免这种情况，因为String已经在启动时就被引导类加载器（Bootstrcp ClassLoader）加载，所以用户自定义的ClassLoader永远也无法加载一个自己写的String，除非你改变JDK中ClassLoader搜索类的默认算法。

JVM在判定两个class是否相同时，不仅要判断两个类名是否相同，而且要判断是否由同一个类加载器实例加载的。只有两者同时满足的情况下，JVM才认为这两个class是相同的。就算两个class是同一份class字节码，如果被两个不同的ClassLoader实例所加载，JVM也会认为它们是两个不同class。

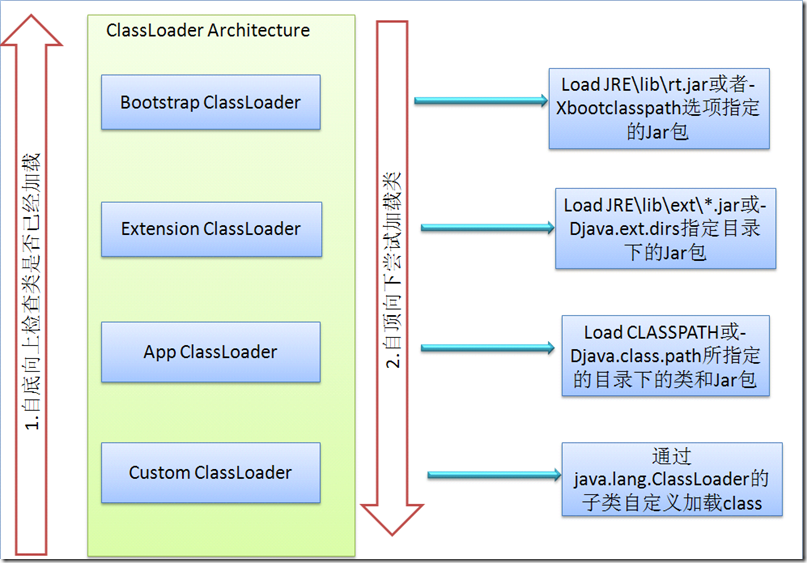


Figure 　ClassLoader 体系

## 定义自已的ClassLoader

**既然JVM已经提供了默认的类加载器，为什么还要定义自已的类加载器呢？**因为Java中提供的默认ClassLoader，只加载指定目录下的jar和class，如果我们想加载其它位置的类或jar时，比如：我要加载网络上的一个class文件，通过动态加载到内存之后，要调用这个类中的方法实现我的业务逻辑。在这样的情况下，默认的ClassLoader就不能满足我们的需求了，所以需要定义自己的ClassLoader。因为JDK已经在loadClass方法中帮我们实现了ClassLoader搜索类的算法，当在loadClass方法中搜索不到类时，loadClass方法就会调用findClass方法来搜索类，所以我们只需重写该方法即可。如没有特殊的要求，一般不建议重写loadClass搜索类的算法。

**定义自已的类加载器分为两步：**

1、继承java.lang.ClassLoader

2、重写父类的findClass方法

## 待完成：

**要自己实现一个Classloader**