Java 动态类加载机制分析及其应用①

崔行臣, 赵 佟

(山东广播电视大学 现代教育技术中心, 济南 250014)

摘 要: Java 虚拟机(JVM)中的类加载机制可以在 Java 应用程序运行期间动态加载类文件, 而不影响其它功能模块的正常运行. 通过对 Java 类加载器的层次体系结构, 动态类加载机制原理、实现过程进行分析, 将 Java 动态类加载机制应用到高校网站管理平台的文件发布模块中, 使得设计好的静态网页和后台管理系统相关联, 通过自定义加载器来加载加密过的 Java 类文件. 这样以可插拔的方式快速的部署二级网站, 增强了网站管理平台的灵活性和可扩展性.

关键词: Java 虚拟机; 类加载器; 委托模型; 动态类加载机制

Java Dynamic Class Loading Mechanism Analysis and Its Application

CUI Xing-Chen, ZHAO Tong

(Center for Educational Technology, Shandong Radio & TV University, Jinan 250014, China)

Abstract: In order to load Java class file during the Java application runtime and without affecting the normal operation of other functional modules, the realization of Java dynamic class loading method is given based on the java virtual machine (JVM) mechanism. Java Class Loader architecture, theory of dynamic class loading mechanism, the implementation process are analyzed, and which are applied to the file release module of web site management platform system. The file release module is designed to make static web pages and website management system linked. Through the custom loader to loading encryption of Java class files. In a hot swap way to implement the deployment of secondary website, enhance the site management platform's flexibility and expansibility.

Key words: Java virtual machine; class loader; delegation model; dynamic class loading mechanism

动态加载是一种在运行时安装程序组件的技术. 许多操作系统如 Unix、Microsoft Windows 使用的动态连接就是一种动态加载技术. 使用动态连接后, 程序中的符号引用可以在程序被加载到内存后才替换成相应的机器地址, 并且直到第一次使用时符号引用才被替换. 与静态连接相比, 动态连接具有增加程序灵活性、节约内存空间的优点[1].

Java 动态类加载是指可以使运行中的程序去调用在源代码中未提及、而在程序运行中所用的类. 一个应用程序总是由 n 多个类组成, Java 程序启动时 , 并不是一次把所有 的类全部加载 后再运行, 它总是先把保证程序运行的基础类 一次性加载到 JVM 中, 其它类等到 JVM 用到的时候再加载, 这样的好处是节省了内存的开销, 因为 Java 最早就是为 嵌入式系统而设计的, 内存宝贵, 而用到时再加载这也是 Java 动态 性的一种体现. 本文介绍了 Java 类加载器的体系架构, 研究探讨了 Java 动态类加载机制的原理及实现, 并将其应用到高校网站通用管理系统的文件发布模块和类文件加密中, 实现了基于网站通用管理平台建设新的二级网站的热插拔扩展, 有效的平衡了平台的扩展性和稳定性之间的关系, 达到了实现高校二级部门网站系统易扩展、部署方便等目的.

1 Java类加载器的层次体系架构

Java 中的所有类, 必须被装载到 jvm 中才能运行, 这个装载工作是由 jvm 中的类加载器完成的, 类加载

① 收稿时间:2012-12-26;收到修改稿时间:2013-01-21

© 中国科学院软件研究所 http://www.c-s-a.org.cn

器所做的工作实质是把 类文件从硬盘读取到内存中. 对于 Java 中的类大致可以分为三种: Java 系统核心类、扩展类和由程序员自定义的类. jvm 的类加载器也至少有三种: Bootstrap classLoader、ExtClassLoader 和 AppClassLoader, 分别负责加载 Java 系统核心类, 扩展类和程序员定义的应用类.

Bootstrap classLoader 是系统中 唯一的、用编写虚拟机的语言编写的类加载器. 如果 JVM 是用 C/C++实现的, 则 Bootstrap classLoader 也是用 C/C++编写, 在 Java 中 Bootstrap ClassLoader 用 null 表示[2]. Bootstrap Loader 用 默认方式从 JRE 中加载 Java 运行环境提供的所有核心类, 如 JRE 目录下的 rt.jar, charsets.jar 等. 这些类是所有应用程序必须的, 因此不是"即用即装", 而是首先装入并永驻 JVM, 直至 JVM 退出的. 类加载器实质上也是 Java 类, 因为 Java 类的类加载器本身也要被类加载器加载, 显然必须要第一个类加载器本身不是 Java 类, 这正是 Bootstrap 根引导类加载器.

Java 核心类之外的类是由 ExtClassLoader 和 AppClassLoader 来完成加载的. ExtClassLoader 负责装载 JRE 扩展目录 ext 下的 jar 类包; AppClassLoader 负责装载 classpath 路径下的类包. JVM 中的所有类加载器采用具有父子关系的树形结构进行组织, 即根加载器是 ExtClassLoader 的父加载器, ExtClassLoader 是 AppClassLoader 的父加载器. 在程序员不显示指定类加载器的情况下使用 AppClassLoader 装载应用程序的类.

除了 JVM 默认的三个加载器以外, 第三方可以编写自己的类加载器, 以实现一些特殊的需求, 例如, 对特定目录的类进行加密, 只有用我们自己的类加载器才能解密并加载.

2 Java动态类加载机制原理

JVM 解释字节码, 必须通过加载 、 连接和初始化三个步骤:

(1) 加载: 类装载就是寻找类 或接口字节码文件进行解析并构造 JVM 内部对象表示的组件.

(2) 连接: 连接是把已经加载的二进制数据合并到虚拟机的运行时状态中去, 包括检查 、 准备和解析三个子阶段, 其中解析步骤 是可以选择的.

(a) 检查: 检查载入的 class 文件数据的正确性;

(b) 准备: 为类的静态变量分配存储空间;

(c) 解析: 将符号引用转成直接引用.

(3) 初始化: 对静态变量, 静态代码块 执行初始化工作. 如图 1 所示.

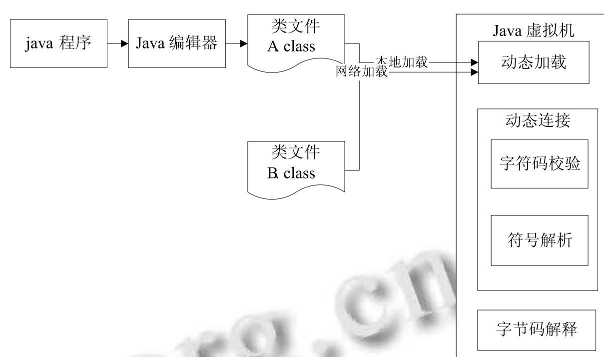

图 1 Java 类的加载与连接过程

由上一节可知, JVM 运行时至少会产生三个类加载器: Bootstrap ClassLoader, Ext ClassLoader 和 App ClassLoader, 当一个类需要加载运行时, 为了协调多个类加载器工作, Java 对类加载器进行了分工和分级, 不同级别的类加载器负责加载不同的类, JVM 采用基于类加载器 层次关系的"全盘负责双亲委托模型"机制实现按需加载, 简单讲, 就是类加载器有载入类的需求时, 会先请示其父类加载器使用其搜索路径来载入, 如果父类加载器找不到, 才由自己的类加载器依照自己的 搜索路径搜索类, 该搜索过程具有递归性. "全盘负责"是指当一个加载器装载一个类时, 除非显式的使用另外一个加载器, 该类所依赖及引用的类也由这个加载器载入. 假如类 A 和类 B 相关联, 当 JVM 加载类 A 后, 系统也会使用 同一个类加载器加载类 B. 关 联的种类有很多, 例如类 A 继承类 B, 又例如类 A 的方法中创建 了类 B 的实例等. "委托模型"是指, 当一个加载器被请求装载某个类时, 首先在本类加载器中检查要加载的类是否已经加载, 如果加载了就直接返回以前加载过的这个类对象. 如果这个类未被本类加载器加载过, 那么它把这个类名交给它的父类加载器, 委托父类 加载器去加载. 若父加载器能装载, 则返回这个类所对应的 class 对象, 否则继续向上级提交, 一直到根引导类加载器, 显然这是一个递归的调用. 只有在根引导类加载器都不能加载的情况下才从自己的类路径中查找并装载目标类. 这一点是从安全方面考虑的, 试想 如果一个人写了一个恶意的基础类(如 java.lang.String)并加载到 JVM 将会引起严重的后果, 但有了委托模型, java.lang.String 永远是由根加

© 中国科学院软件研究所 http://www.c-s-a.org.cn

载器来装载, 避免以上情况发生[3]. “全盘负责双亲委托模型”机制如图 2 所示.

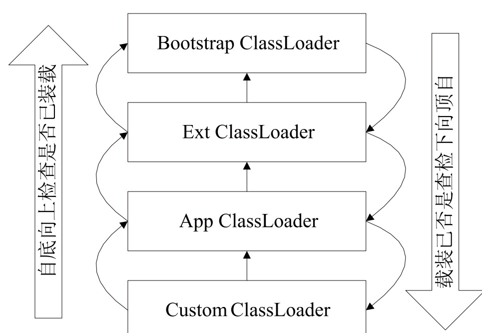

图 2　JVM 动态类加载的委托模型示意图

每个类加载器都可以 用来加载类, 而且同一个类可以在不同的类加载器中多次加载, 但在同一个类加载器上就只允许加载一次. 这是因为当类加载器加载一个类之前会先检查这个类是否被本类加载器加载过. 因此, 在 JVM 中, 每个被加载的类 都由两个参数来确定, 一个是类的名字, 还有一个是加载这个类 的类加载器[4]. 类文件被装载解析后, 在虚拟机中都有一个对应的 java.lang.Class 对象, 提供了类结构信息的描述, 且每个 Class 对象都包含一个对定义它的 ClassLoader 的引用. 数组及基本数据类型, 甚至 void 都拥有对应的 Class 对象. Class 类没有 public 的构造方法, Class 对象是在装载类 时由 JVM 通过调用类加载器中的 defineClass()方法自动构造的.

3　Java动态类加载的实现

类在 JVM 中的加载方式分为两种: 一是隐式加载, 程序在 运行过程中当执行到通过 new 等方式生成对象时, 隐式调用类加载器加载对应的类到 JVM 中; 二是显式装载, 由程序员调用自定义的类装载器把需要的类加载到内存当中, 其方式又分为两种: 一是通过 Class.forName()方法, 二是由类 java.lang.ClassLoader 的方法 loadClass()提供. 显式加载大大 提高了程序的灵活性. 自定义类加载器可以从 Java.lang.ClassLoader 的任何子类创建(可以从构造方法 ClassLoader (ClassLoader parent)可以看出, 需要指定一个父加载器, 创建后即挂到 JVM 加载器层次树中).

任何类的加载都是 通过抽象类 ClassLoader 类及其子类来实现的, 它是 Java 核心 API 的一部分, 属于 Java.lang 包, 所有自定义类加 载器必须继承并实例化该类. 重要方法有[5]:

(1) Class loadClass(String name, boolean resolve) : name 参数指定类加载器需要 装载类的全限定类名. resolve 参数指定类加载器是否解析该类. 在初始化类之前应考虑进行类解析的工作, 但并不是所有的类都需要解析. 该方法使用委托机制来加载类. 首先它执行 findLoadedClass 方法, 在本类加载器中检查要加载的类是否已经加载, 如果加载了就直接返回以前加载过的这个类对象. 否则, 委托父加载器去加载(调用 loadClass 方法), 最后调用 findClass(String)方法查找类.

(2) Class findClass(String name): 使用指定的二进制名称查找类. 在通过父类 加载器检查所请求的类后, 此方法将被 loadClass 方法调用. loadClass 的缺省实现调用这个方法, 并可以通过覆盖来定制它. 比如自定义加载器, 首先继承抽象类 ClassLoader, 然后不必重写 loadClass 方法, 只需重写 findClass 方法即可, 这样自定义的加载器仍然按照委托模型来加载类. 这种设计模式属于模板方法设计模式.

(3) Class defineClass(String name, byte[] b, int off, int len): 该方法将一个字节数组转换为类的字节码.

(4) Class findSystemClass(String name) : 此方法通过系统类加载器来加载该类文件, 如果存在, 就使用 defineClass 将原始字节转换成 Class 对象, 以将该文件转换成类.

(5) Class findLoadedClass(String name), 如果 Java 虚拟机已将此加载器记录为具有给定二进制名称的某个类的加载器, 则返回该二进制名称的类.

动态类加载最直接方式 是使用 Java.lang.Class 的 forName()方法, 它有两种重载形式:

public static Class<?> forName(String className) throws ClassNotFoundException

public static Class<?> forName(String name, boolean initialize, ClassLoader loader)

使用给定的类加载器(第三个参数指定), 返回与带有给定字符串名的类或接口(第一个参数指定)相关联的 Class 对象. 第二个参数指定表示是否被初始化. Class.forName("Foo") 等效于:

Class.forName("Foo",true,this.getClass().getClassLoader())

实现动态类加载, 使用 forName()还是 调用用户自定义类加载器 loadClass() 方法 取决于用户的需要.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

一般情况下使用 forName(), 因为它是动态类加载最直接的方法. 另外, 若需要请求的类型在加载时就初始化, 则不得不使用 forName()方法. 自定义类加载器可以满足一些 forName() 无法满足的需求. 如从网络上下载, 从数据库中获取, 从加密文件中提取, 甚至动态地创建它们, 这时就需要自定义类加载器. 创建自定义类加载器, 主要原因是可以用定制的方式把类型的全限定名转换成一个 Java class 文件格式的字节数组, 并使用自定义类加载器而不是 forName() 方法对代码安全性进行保护[6].

4 应用实例

Java 语言的类加载器功能非常强大, 可以通过继承 ClassLoader 类并重载类方法来实现一些控制程序加载过程的功能. 本节将 Java 动态类加载机制应用到高校二级网站管理系统的文件发布模块中, 并采用自定义加载器来加载已经加密过的 Java 类文件.

4.1 系统应用需求

当前大部分高校由于没有构建统一的网站通用平台, 各高校的二级部门各自为阵, 将大量人力和物力花在网站的重复建设上, 而且各高校二级学院网站的风格不统一、效果差, 不利于展示高校的整体形象. 因此, 构建一个高校网站通用平台应用于高校的各个二级学院具有非常重要的意义. 由于高校各部门网站的功能大致是类似的, 如权限管理、栏目管理、新闻管理等, 所以在开发好网站通用管理平台后, 如果要建设一个二级部门网站, 技术人员可以先做好静态网页, 然后在通用管理平台中为二级网站分配栏目, 公共功能之外的个性差异需求通过文件发布模块以"可插拔"的方式发布到通用管理平台中, 经过简单的配置即可完成一个二级部门网站的建设. 文件发布模块的主要功能是对某二级网站的静态页面动态化, 使得设计好的静态网页和后台管理系统相关联. 通过灵活的发布模块, 增加了系统的可扩展性、灵活性和降低了系统模块之间的耦合性, 再对发布的文件代码进行加密处理, 增强了系统安全性.

4.2 模块实现

本系统采用基于 MVC 模式的 struts2 框架实现. 文件发布模块需要完成以下工作:

(1) 创建二级网站主页的 HomepageAction 类, 这是一个 Struts2 Action, 负责获取主页栏目信息并以把数据封装在集合中返回, 在视图层采用 struts2 标签展示出来. 这也是各个部门的个性差异部分.

(2) 添加 struts2 配置文件: 配置 Action, 添加逻辑视图和物理视图资源之间的映射.

(3) 为保证代码安全性, 需要对上传代码进行加密处理.

因当前 JVM 中正在运行的是网站通用管理平台, 对于为每个部门主页提交的 Homepage 类是系统预先不知道的类型, 并且对于编译后的 class 文件进行加密处理后, 必须使用自定义加载器加载自定目录下的类文件并解密, 否则就不能正确解析原来的类.

(1) 实现代码加密的自定义加载器源代码如下:

```
public class MyClassLoader extends ClassLoader{
private String classDir;  //  需要类加载器加载的类文件的目录
public MyClassLoader(){ }
    public MyClassLoader(String classDir){
        super( null) ;  / /设定父类加载器为 null
         this.classDir = classDir;
    }
    @Override
// /重写父类的 findClass( ) 方法, 用于按照用户自定义要求实现类的加载
    protected Class<?> findClass(String name) throws ClassNotFoundException {
         String classFileName = classDir + "\\" + name.substring(name.lastIndexOf('.')+1) + ".class";
         try {
             FileInputStream fis = new FileInputStream(classFileName);
             ByteArrayOutputStream bos = new ByteArrayOutputStream();
             decrypt(fis,bos);  // 解密处理方法
             fis.close();
              byte[] bytes = bos.toByteArray();
         return defineClass(bytes, 0, bytes.length);
     } catch (Exception e) { e.printStackTrace(); }
         return null;
    }
}
```

(2) 设计文件发布方法. 设计 deployModule(String

© 中国科学院软件研究所 http://www.c-s-a.org.cn

name)方法从数据库读取并生成类文件的方法, 其部分源码如下:

```
public void deployModule ( string classdir ) throws ClassNotFoundException {
……
Class clazz = new MyClassLoader(classdir).loadClass("cn.com.HomepageAction");
Object o = (Object )clazz.newInstance(); / / 创建对象, 利用 Java 反射机制解析类文件中的方法
……
}
```

5 结语

Java 动态类加载是 Java 程序具有动态性的关键机制, 也是 JVM 的一项核心技术. 本文分析了 Java 类加载器的体系结构、动态类加载机制的原理、实现技术, 并把 Java 类加载机制应用到高校部门网站通用管理平台中的文件发布模块和加密类文件加载中, 实现了基于网站通用管理平台建设新的二级网站的热插拔扩展, 使得平台具有更好的扩展性和代码的可重用性, 对加密后的字节码只能用自定义加载器进行解密并加载, 实现一定意义上的类文件安全. Java 动态类加载机制的应用还有很多, Java 中 RMI(远程方法调用)就是 Java 动态类 加载机制的一个典型应用, 而这是一般类加载器无法实现, 只有通过自定义加载策略, 才能完美实现这样的需求.

参考文献

1 左天军,朱智林,韩俊刚,陈平.Java 动态类加载分析.计算机科学,2005,32(4):194–196.

2 周志明.深入理解 Java 虚拟机:JVM 高级特性与最佳实践.北京:机械工业出版社,2010.191–197.

3 邓洋春.Java 虚拟机关键机制研究与实践.长沙:中南大学,2009:3–28.

4 张敦华,刘建.Java 动态加载机制及其应用.计算机工程与设计,2004,25(3):432–441.

5 JDK6.0 API 文档.http://docs.oracle.com/javase/6/docs/api/index.html.

6 王万森,龚文.Java 动态类加载机制研究及应用.计算机工程与设计,2011,32(6):2154–2158.

(上接第 199 页)

3 龙德帆,樊尚春,庞宏冰.用于原木材积检测的图像处理与分析算法.北京航空航天大学学报,2005,31(1):82–85.

4 景林,黄习培.成捆原木计算机图像检尺系统研究及应用计算机应用,2006,26(z2):137–139.

5 黄习培,景林.原木端面图像检尺直径识别算法的研究.林业机械与木工设备,2006,34(1):24–26.

6 Kass M, et al. Snakes: Active Contour Models. International Journal of Computer Vision, 1987,1(4):321–331.

7 Xu CY, et al. Snakes, shapes, and gradient vector flow. IEEE Trans. on Image Processing, 1998,7(3):359–369.

8 Li CM, et al. Level set evolution without re-initialization: A new variational formulation. IEEE Conference on Computer Vision and Pattern Recognition(CVPR05). 2005,1:433–436.

9 Rochery M, et al. Higher Order Active Contours. International Journal of Computer Vision, 2006,69(1):27–42.

10 Horváth P, et al. A higher-order active contour model of a ‘gas of circles’' and its application to tree crown extraction, Pattern Recognition, 2009,42(5):699–709.

11 Li CM, et al. Minimization of Region-Scalable Fitting Energy for Image Segmentation. IEEE Trans. on Image Processing, 2008,17(10):1940–1949.

12 Zhu GP, et al. Gradient vector flow active contours with prior directional information, Pattern Recognition Letters, 2010,31(9):845–856.

13 Hough PVC. A method and means for recognizing complex patterns, U.S. Patent 3, 069, 654.

14 Chung KL, et al. Speed up the computation of randomized algorithms for detecting lines, circles, and ellipses using novel tuning and LUT-based voting platform. Applied Mathematics and Computation, 2007,190(1):132–149.

© 中国科学院软件研究所 http://www.c-s-a.org.cn