通过 Java 的自定义类加载器执行保存在 Web 服务器上的字节码文件

Pass the Java from the definition type to add to carry the machine perfor - mance conservancy in the byte code document on the server of Web

曹大有　刘耀钦　（郧阳师范高等专科学校 计算机科学系 湖北 丹江口 442700）

摘要：Java 的解释器是通过将 Java 类的字节码文件装入 Java 运行环境，然后接受字节码检验器的检验后再运行之。由于 Java 类文件中使用的字节码格式是有很好的文档基础，对于具有汇编程序设计经验并且拥有一个十六进制编辑器的人来说，要想人工为 Java 解释器产生一个包含有效而不安全指令的类文件，这是一件非常容易的事情。本文则从安全角度出发，探讨了如何将 Java 字节码文件预先存入指定的 Web 服务器中，然后通过自定义的 Java 类加载器来运行保存在指定的 Web 服务器中的字节码文件的方法，并对其可行性进行了深入的讨论。

关键词：检验器 字节码 加载器 虚拟机

1 引言

我们通常运行 Java 应用程序时，是通过输入命令行参数：java Application_name，来使用 Java 默认的类加载工具完成对指定的应用程序的加载和解释执行的。但是由于 Java 类的字节码文件中使用的字节码格式是有很好的文档基础，对于具有汇编程序设计经验并且拥有一个十六进制编辑器的人来说，要想人工为 Java 虚拟器产生一个包含有效而不安全指令的类文件，这是一件非常容易的事情。我们能否采取一定的方法来对 Java 的字节码文件进行预先的处理，比如将类的字节码文件预先存入指定的 Web 服务器中，然后从 Web 服务器中来加载并运行该 Java 程序，以此来增强系统的安全性呢？为了解决这一问题，我们必须首先对 Java 应用程序的执行过程和 Java 的类加载机制做一简单概括的分析。

2 Java 虚拟机解释器

Java 编程语言编译器能够将 Java 源程序转换为某种假想机器的机器语言，这种假想机器称为 Java 虚拟机，虚拟机代码存储在扩展名为. class 的类文件中，类文件包含一个类的全部方法的代码，这些类文件必须由一个翻译器进行解释，该翻译器能够将虚拟机的指令集翻译成目标机器的机器语言，这个翻译器称为 Java 虚拟机解释器。

虚拟机解释器通过 Java 默认的类加载器只加载程序运行所需要的类文件，即该类的主类文件和该主类所依赖的其它类的类文件以及 main( ) 方法所调用的更多的类。然而 Java 的类加载机制并非只使用单个的类加载器。每个 Java 应用程序至少要有下面 3 个类加载器：

引导类加载器：负责加载系统类，通常从 rt. jar 文件那里进行加载；

扩展类加载器：用于从 jre/lib/ext 目录加载一个标准的扩展名；

系统类加载器：负责加载应用程序类。

在使用这些类加载器时，又有以下三种使用方式：

隐式加载：即当一个类被解析时，该类所依赖的那些类被加载时所采用的加载方式；

显示加载：它没有命名的类加载器，而是通过调用 Class. forName(className) 方法进行类的加载；

自定义类加载器：即通过调用抽象类 ClassLoader 的 loader. loadClass(className) 方法进行加载。

根据以上的分析，我们如果要想对 Java 的字节码文件进行预先的处理，存入指定的 Web 服务器中，以此来增强系统的安全性，必须通过指定 Java 类的加载器，通过自定义类加载器进行加载。

在我们的 Web 服务器 http://pido. cs. yytc. net. cn/cao 目录下，就事先存放有 Application. java、MyJavaApplication. java 等二个字符界面应用程序的全部字节码文件的内容。下面我们就以此二例来进行讨论之。

3 通过自定义类加载器来加载并运行存放在 Web 服务器中的应用程序

由于我们将类的字节码文件事先存入指定的 Web 服务器中，目的是为了增强系统安全性。那动态加载执行的过程就要复杂一些。首先我们要将存放在指定 Web 服务器中的字节码文件内容进行读出处理，当然我们也可以先读出以形成原始的字节码文件，然后再应用 Java 默认的类加载机制进行动态加载执行。但由于我们要将读出和动态加载执行合为一个过程，所以我们只能通过定制一个类加载器来完成上述过程。

若要定制自己的类加载器，就需要扩展 ClassLoader 抽象类，并重载方法：findClass (String className) 即可完成。findClass(String className) 的方法原型为：

```
protected Class findClass ( String name ) throws ClassNotFoundException
```

超类 ClassLoader 的 loadClass(String name) 方法用于将指定类的加载操作委托给父类去进行，只有当该类尚未加载，并且父类加载器无法加载该类时，才调用 findClass( ) 方法来加载指定的类。经过特殊处理的类文件，如存放在指定的 Web 服务器中的字节码文件，一般要通过重载 findClass(String className) 方法来完成加载过程。

如果要实现 findClass( ) 方法，我们必须完成两件事：一是从本地文件系统或从其他来源，如 Web 服务器上为类加载字节码；二是调用超类 ClassLoader 的 defineClass( ) 方法，给虚拟机提供当前字节码。对第一项任务，我们通过自定义方法：

```
private byte [ ] loadClassBytes ( String name ) throws IOException
```

来完成该任务。该方法从指定的 Web 服务器 http://pido. cs. yytc. net. cn/cao/ 中读出指定名 name 的字节码文件内容，然后将其内容转化为字节数组提供给 defineClass( ) 方法使用。

主要过程是通过建立一个指向指定字节码文件的 URL 对象，再通过 URL 对象的 openStream( ) 方法获得一个输入流 InputStream，通过该输入流将指定的文件内容读出并转换成字节数组。在 HTTP 协议的情况下，调用 openStream( ) 方法将自动地发送一个 HTTP 请求并解析返回的报头，因此返回的流将只读取该对象的内容。该方法的调用等价于首先调用 openConnection ( ) 方法，再调用返回的 URLConnection 对象的 getInputStream( ) 方法。具体实现如下所示：

```
private byte [ ] loadClassBytes ( String name ) throws Exception{
    try{ name = name. replace('.','/') + ". class" ;
      urlAdr + = name;// urlAdr 事先指向 http://pido. cs. yytc. net. cn/cao/
      URL url = new URL( urlAdr) ;
      ByteArrayOutputStream buffer = new ByteArrayOutputStream( ) ;
      InputStream in = url. openStream( ) ;
      BufferedInputStream iin = new BufferedInputStream( in) ; int ch;
      while ( ( ch = iin. read ( ) ) ! = -1) buffer. write( ch) ;
      buffer. flush( ) ; in. close( ) ; return buffer. toByteArray( ) ;
    } finally{ } }
```

方法 defineClass( ) 的原型为：

```
protected final Class defineClass ( String name, byte[ ] b, int off, int len)
    throws ClassFormatError
```

它需要四个参数：一是需要加载的类名，二是存放该类字节码信息的数组名，三是偏移量，四是数组的长度。它的功能是将存放在数组 b 中的字节码信息转换成 Class 对象的一个实例，但是所得到的类在没有通过方法 findClass( ) 调用之前是不能使用的，若数组 b 中的内容不能被译解，则抛出一个 ClassFormatError 异常信息。

有了以上分析，我们对 findClass( ) 方法的重载过程就为：

```
protected Class findClass ( String name ) throws ClassNotFoundException{
    byte[ ]classBytes = null;
    try{ classBytes = loadClassBytes( name ) ;//将文件内容装入字节数组中
    }catch ( IOException exception ){
        throw new ClassNotFoundException ( name ) ;}
    Class cl = defineClass ( name, classBytes, 0, classBytes. length ) ;//定义类
    if( cl = = null ) throw new ClassNotFoundException( name ) ;
    return cl ;}
```

自定义类加载器的代码，具体见程序 ClassFarLoaderTest. java 中的 CryptoClassLoader 类所示。

由于每个应用程序都是以主类中的主方法 main( ) 为执行起点的，所以我们只要从主类中通过 Java 提供的反射功能，寻找并执行 main( ) 方法，从而就可以让整个应用程序执行起来。具体过程要涉及到以下方法：

首先是 Class 类中的 getMethod( ) 方法，其方法原型为：

```
public Method getMethod ( String name, Class[ ] parameterTypes )
        throws NoSuchMethodException, SecurityException
```

该方法从指定的类中寻找并返回由 name 命名并具有指定参数 parameterTypes 的方法对象，当然我们这里寻找的 name 为" main"，而 parameterTypes 为 new Class[ ]{ args. getClass( ) }。

其次是 Method 类中的 invoke( ) 方法，它的方法原型为：

```
public Object invoke( Object obj, Object[ ] args )
throws IllegalAccessException, IllegalArgumentException,
        InvocationTargetException
```

该方法将执行 Method 对象中的方法，其中：args 是该方法需要的参数，而 obj 一般可以为 null。实际过程见以下代码所示，在我们附带的程序 ClassFarLoaderTest. java 中就是通过这种方式来执行字节码文件被存放在指定的 Web 服务器 http://pido. cs. yytc. net. cn/cao/中的类文件。

```
try{
    ClassLoader loader = new CryptoClassLoader( " http://pido. cs. yytc. net. cn/cao/" ) ;
    Class c = loader. loadClass ( name ) ; String [ ] args = new String[ ]{ } ;
    Method m = c. getMethod( " main" ,new Class[ ] { args. getClass( ) } ) ;
    m. invoke( null, new Object[ ] { args } ) ;
    }catch( Exception e ){ e. printStackTrace( ) ;}
```

如我们可以通过在命令行窗口中输入 java ClassFarLoaderTest MyJavaApplication 来运行存放在 Web 服务器：http://pido. cs. yytc. net. cn/cao/中 MyJavaApplication. java 程序的字节码文件。当然程序 MyJavaApplication. java 被编译后的字节码文件必须事先存放在 Web 服务器：http://pido. cs. yytc. net. cn/cao/中。

对存放在 Web 服务器中的字节码文件的动态加载与执行的过程见程序 ClassFarLoaderTest. java 所示，该程序中包括了自定义的类加载器 CryptoClassLoader 的定义过程。

4　可行性讨论

如果我们用 ClassFarLoaderTest. java 动态加载并执行具有字符界面的 Java 应用程序时，只要具有字符界面的应用程序不用命令：System. exit( ) 直接退出 Java 虚拟机，我们就可以多次动态加载并执行它们，但是要解决的是如何在 ClassFarLoaderTest. java 程序中给某些具有字符界面的应用程序提供命令行参数。如：Application. java 应用程序在运行时就可以处理命令行参数，这如何解决呢？解决的办法是在命令窗口中通过以下格式：

java ClassFarLoaderTest ＜被加载并动态执行的应用程序名＞ ＜各参数串＞

直接输入被加载并动态执行的应用程序名后再输入各参数串，其中应用程序名和参数串之间以及各参数串之间用空格分开。然后通过以下具体过程：

```
String cname; cname = args[0] ;
String[ ]arg = new String[ args. length - 1] ; int i =
```

```
0,j=1;
    while(j < args.length) arg[i++] = args[j++];
    ClassLoader loader = new CryptoClassLoader("http://pido.cs.yytc.net.cn/cao/");
    Class c = loader.loadClass(cname);
    Method m = c.getMethod("main",new Class[]{arg.getClass()});
    m.invoke(null, new Object[]{arg});
```

来运行要求命令行参数的字符界面的 Java 应用程序。具体我们可以将以上改进之处放入程序 ClassFarLoaderTest.java 中。这样当我们在命令窗口中输入：

java ClassFarLoaderTest Application 123 234 345 456 时，字符界面程序 Application.java 的运行结果就在控制台窗口中显示出来了，这样我们即指定了被动态加载执行的应用程序又为它指定了运行过程中所需要的参数。当然应用程序 Application.java 事先也要被编译成字节码文件并也要存放在 Web 服务器：http://pido.cs.yytc.net.cn/cao/中。

另外我们也可以通过建立 URL 对象的方式来运行保存在本地但经过加密了的字节码文件。如本文中就提供了程序 Calculator.java 加密后的字节码文件，加密方式见程序 Caesar.java 所示。类加载器中主要过程变为：

```
private byte[] loadClassBytes(String name) throws Exception{
    try{ name = name.replace('.','/') + ".caesar";
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        InputStream in = new File(name).toURL().openStream();
        BufferedInputStream iin = new BufferedInputStream(in);int ch;
        while((ch = iin.read()) != -1){
          byte b = (byte)(ch - 3);//进行解密处理
          buffer.write(b);}
        buffer.flush();in.close();
        return buffer.toByteArray();
    }finally{}}
```

这样当我们在命令窗口中输入：

java CaesarNearLoaderTest Calculator 时，就可以加载并执行程序 Calculator.java 的加密版本。当然如果我们将 MyJavaApplication.java 和 Application.java 二程序加密后的字节码文件存放到指定的 Web 服务器中，也可以仿照此方法对它们进行加载运行。对具有图形用户界面的 Java 应用程序也可仿照此过程进行讨论。本文的运行环境是：Windows XP 和 jdk-1_5_0_04。

参考文献

1 John Zukowski(美)著，邱仲潘译，Java 2 从入门到精通[M]，北京：电子工业出版社，2005.

2 Cay S. Horstmann、Gary Cornell(美)著，京京工作室译，Java 2 核心 卷 I：基础知识[M]. 北京：机械工业出版社，2000.