Java虚拟机类装载：原理、实现与应用 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022_Java_E8_99_9A_E6_8B_9F_c104_252295.htm 一、引言 Java虚拟机(JVM)的类装载就是指将包含在类文件中的字节码装载到JVM中, 并使其成为JVM一部分的过程。JVM的类动态装载技术能够在运行时刻动态地加载或者替换系统的某些功能模块, 而不影响系统其他功能模块的正常运行。 本文将分析JVM中的类装载系统，探讨JVM中类装载的原理、实现以及应用。 二、Java虚拟机的类装载实现与应用 2.1 装载过程简介 所谓装载就是寻找一个类或是一个接口的二进制形式并用该二进制形式来构造代表这个类或是这个接口的class对象的过程，其中类或接口的名称是给定了的。当然名称也可以通过计算得到，但是更常见的是通过搜索源代码经过编译器编译后所得到的二进制形式来构造。 在Java中，类装载器把一个类装入Java虚拟机中，要经过三个步骤来完成：装载、链接和初始化，其中链接又可以分成校验、准备和解析三步，除了解析外，其它步骤是严格按照顺序完成的，各个步骤的主要工作如下： 装载：查找和导入类或接口的二进制数据； 链接：执行下面的校验、准备和解析步骤，其中解析步骤是可以选择的； 校验：检查导入类或接口的二进制数据的正确性； 准备：给类的静态变量分配并初始化存储空间； 解析：将符号引用转成直接引用； 初始化：激活类的静态变量的初始化Java代码和静态Java代码块。 至于在类装载和虚拟机启动的过程中的具体细节和可能会抛出的错误，请参看《Java虚拟机规范》以及《深入Java虚拟机》，它们在网络上面的资源地址是：

http://java.sun.com/docs/books/vmspec/2nd-edition/html/Preface.doc.html http://www.artima.com/insidejvm/ed2/index.html 由于本文的讨论重点不在此就不再多叙述。 2.2 装载的实现 JVM中类的装载是由ClassLoader和它的子类来实现的,Java ClassLoader是一个重要的Java运行时系统组件。它负责在运行时查找和装入类文件的类。 在Java中，ClassLoader是一个抽象类，它在包java.lang中,可以这样说，只要了解了在ClassLoader中的一些重要的方法，再结合上面所介绍的JVM中类装载的具体的过程，对动态装载类这项技术就有了一个比较大概的掌握，这些重要的方法包括以下几个: ①loadCass方法 loadClass(String name ,boolean resolve)其中name参数指定了JVM需要的类的名称,该名称以包表示法表示,如Java.lang.Object；resolve参数告诉方法是否需要解析类，在初始化类之前,应考虑类解析，并不是所有的类都需要解析，如果JVM只需要知道该类是否存在或找出该类的超类,那么就不需要解析。这个方法是ClassLoader 的入口点。 ②defineClass方法 这个方法接受类文件的字节数组并把它转换成Class对象。字节数组可以是从本地文件系统或网络装入的数据。它把字节码分析成运行时数据结构、校验有效性等等。 ③findSystemClass方法 findSystemClass方法从本地文件系统装入文件。它在本地文件系统中寻找类文件,如果存在,就使用defineClass将字节数组转换成Class对象,以将该文件转换成类。当运行Java应用程序时,这是JVM 正常装入类的缺省机制。 ④resolveClass方法 resolveClass(Class c)方法解析装入的类,如果该类已经被解析过那么将不做处理。当调用loadClass方法时,通过它的resolve 参数决定是否要进行解析。 ⑤findLoadedClass方法 当调

用loadClass方法装入类时,调用findLoadedClass 方法来查看ClassLoader是否已装入这个类,如果已装入,那么返回Class对象,否则返回NULL。如果强行装载已存在的类,将会抛出链接错误。 2.3 装载的应用 一般来说，我们使用虚拟机的类装载时需要继承抽象类java.lang.ClassLoader,其中必须实现的方法是loadClass()，对于这个方法需要实现如下操作:(1) 确认类的名称.(2) 检查请求要装载的类是否已经被装载.(3) 检查请求加载的类是否是系统类.(4) 尝试从类装载器的存储区获取所请求的类.(5) 在虚拟机中定义所请求的类.(6) 解析所请求的类.(7) 返回所请求的类。 所有的Java 虚拟机都包括一个内置的类装载器，这个内置的类库装载器被称为根装载器(bootstrap ClassLoader)。根装载器的特殊之处是它只能够装载在设计时刻已知的类,因此虚拟机假定由根装载器所装载的类都是安全的、可信任的,可以不经过安全认证而直接运行。当应用程序需要加载并不是设计时就知道的类时,必须使用用户自定义的装载器(user-defined ClassLoader)。下面我们举例说明它的应用。public abstract class MultiClassLoader extends ClassLoader{ ... public synchronized Class loadClass(String s, boolean flag) throws ClassNotFoundException { /* 检查类s是否已经在本地内存*/ Class class1 = (Class)classes.get(s). /* 类s已经在本地内存*/ if(class1 != null) return class1. try/*用默认的ClassLoader 装入类*/ { class1 = super.findSystemClass(s). return class1. } catch(ClassNotFoundException _ex) { System.out.println(">> Not a system class."). } /* 取得类s的字节数组*/ byte abyte0[] = loadClassBytes(s). if(abyte0 == null) throw new ClassNotFoundException(). /* 将类字节数组转换为类*/ class1 =

defineClass(null, abyte0, 0, abyte0.length). if(class1 == null) throw new ClassFormatError(). if(flag) resolveClass(class1). /*解析类*/ /*将新加载的类放入本地内存*/ classes.put(s, class1). System.out.println(">> Returning newly loaded class."). /* 返回已装载、解析的类*/ return class1. } ...} 