理解Java类加载原理计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022__E7_90_86_E8_A7_A3Java_c97_645506.htm 第一部分. 提示 我需要读这篇文章吗？ Java类加载器对Java系统的运行是至关重要的，但是却常常被我们忽略。Java类加载器负载在运行时查找和加载类。自定义类加载器可以完全改变类的加载方式，以自己喜欢的方式来个性化你的Java虚拟机。本文简要的介绍Java类加载器，然后通过一个构造自定义类加载器的例子来说明，这个类加载器在加载类前会自动编译代码。你将学到类加载器到底是干什么的，如何创建你自己的类加载器。只要你有一些基本的Java知识，知道如何创建、编译、运行一个命令行Java程序以及一些 Java类文件的基本概念，你就可以理解本文的内容了。读完本文，你应该能够：* 扩张Java虚拟机的功能 * 创建一个自定义的类加载器 * 如何把自定义的类加载器整合到你的应用程序中 * 修改你的类加载器以兼容Java 2 获得帮助 对本文有任何问题，可以联系作者Greg Travis，油箱：mito@panix.com 。 第二部分. 简介 类加载器是什么？ Java和其他语言不同的是，Java是运行于Java虚拟机(JVM)。这就意味着编译后的代码是以一种和平台无关的格式保存的，而不是某种特定的机器上运行的格式。这种格式和传统的可执行代码格式有很多重要的区别。具体来说，不同于C或者C 程序，Java程序不是一个独立的可执行文件，而是由很多分开的类文件组成，每个类文件对应一个Java类。 另外，这些类文件并不是马上加载到内存，而是当程序需要的时候才加载。类加载器就是Java虚拟机中用来把类加载到内存的工具。而且

，Java类加载器也是用Java实现的。这样你就不需要对Java虚拟机有深入的理解就可以很容易创建自己的类加载器了。 为什么要创建类加载器? 既然Java虚拟金已经有了类加载器，我们还要自己创建其他的呢?问得好。默认的类加载器只知道如何从本地系统加载类。当你的程序完全在本机编译的话，默认的类加载器一般都工作的很好。但是Java中最激动人心的地方之一就是很容易的从网络上而不只是本地加载类。 举个例子，浏览器可以通过自定义的类加载器加载类。 还有很多加载类的方式。除了简单的从本地或者网络外，你还可以通过自定义Java中最激动人心的地方之一: * 执行非信任代码前自动验证数字签名 * 根据用户提供的密码解密代码 * 根据用户的需要动态的创建类 你关心的任何东西都能方便的以字节码的形式集成到你的应用中 自定义类加载器的例子 如果你已经使用过JDK(Java软件开发包)中的appletviewer（小应用程序浏览器）或者其他Java嵌入式浏览器，你就已经使用了自定义类加载器了。Sun刚刚发布Java语言的时候，最令人兴奋的一件事就是观看Java如何执行从远程网站下载的代码。执行从远程站点通过HTTP连接传送来的字节码看起来有点不可思议。之所以能够工作，因为Java有安装自定义类加载器的能力。小应用程序浏览器包含了一个类加载器，这个类加载器不从本地找Java类，而是访问远程服务器，通过HTTP加载原始字节码文件，然后在Java虚拟机中转化为Java类。当然类加载器还做了其他的很多事情：他们阻止不安全的Java类，而且保持不同页面上的不同小程序不会互相干扰。Luke Gorrie写的一个包Echidna是一个开放的Java软件包，他允许在一个Java虚拟机中安全的运行多个Java应用程序。它通过使用自定义类加载器

给每个应用程序一份类文件的拷贝来阻止应用程序之间的干扰。 我们的类加载器例子 我们知道了类加载器是如何工作的，也知道如何定义自己的类加载器了，接下来我们创建一个名字为CompilingClassLoader (CCL)的自定义类加载器。CCL为我们做编译工作，我们就不用自己手动编译了。 这基本上相当于有一个"make"程序构建到我们的运行环境。 注意：我们进行下一步之前，有必要搞清楚一些相关的概念。 系统在JDK版本1.2（也就是我们说的Java 2平台）得到很到改进。本文是在JDK1.0和1.1的版本下写的，但是所有的东西都能在后来的版本工作。ClassLoader也在Java2种有所改进，第五部分有详细介绍。 第三部分.ClassLoader的结构 总揽 类加载器的基本目的是服务于对Java类的请求。Java虚拟机需要一个类的时候，就把一个类名给类加载器,然后类加载器试图返回一个对应的类实例。可以通过在不同的阶段覆盖相应的方法来创建自定义的类加载器。接下来我们将了解到类加载器的一些主要方法。你会明白这些方法是干什么的，他们在加载类文件的时候是如何工作的。你还将知道创建自定义类加载器的时候需要写哪些代码。在下一部分，你将利用这些知识和我们自定义的 CompilingClassLoader一起工作。 方法 loadClass ClassLoader.loadClass() 是ClassLoader的入口点。方法签名如下：Class loadClass(String name, boolean resolve). 参数name指定Java虚拟机需要的类的全名(含包名)，比如Foo或者java.lang.Object。 参数 resolve指定该类是否需要解析 你可以把类的解析理解为完全为运行做好准备。解析一般都不需要。如果Java虚拟机只想知道这个类是否存在或者想知道它的父类的话，解析就完全没有必要了。 在Java1.1和它以前的版本

，如果要自定义类加载器,loadClass方法是唯一需要在子类中覆盖的方法. (ClassLoader在Java1.2中有所改变，提供了方法findClass())。 方法 defineClass defineClass 是ClassLoader中一个很神秘的方法。这个方法通过一个字节数组来构建类实例。这个包含数据的原始字节数组可能来自文件系统，也可能是来自网络。 defineClass 表明了Java虚拟机的复杂性，神秘性和平台依赖性-它通过解释字节码把它转化为运行时数据结构，检查有效性等等。但是不用担心，这些都不用你去实现。其实，你根本不能覆盖它，因为该方法被关键字final修饰。 方法 findSystemClass findSystemClass方法从本地系统加载文件。它在本地系统寻找类文件，如果找到了，调用defineClass把原始字节数组转化成类对象。这是运行Java应用时Java虚拟机加载类的默认机制。对于自定义类加载器，只有在我们无法加载之后才需要用 findSystemClass。 原因很简单: 我们的类加载器负责执行类加载中的某些特定的步骤，但并不是对所有的类。比如，即使我们的类加载器从远程站点加载了某些类，仍然有很多基本的类要从本地系统加载。 这些类不是我们关心的，所以我们让Java虚拟机以默认的方式加载他们：从本地系统。这就是findSystemClass做的事情。整个过程大致如下：* Java虚拟机请求我们自定义的类加载器加载类。 * 我们检查远程站点是否有这个需要加载的类。 * 如果有，我们获取这个类。 * 如果没有，我们认为这个是类在基本类库中，调用findSystemClass从文件系统中加载。 在大多数自定义类加载器中，你应该先调用findSystemClass来节省从远程查找的时间。 实际上，正如我们将在下一部分看到的，只有当我们确定我们已经自动编译完我们的代码后才允许Java虚拟机从本地文

件系统加载类。 方法resolveClass 正如上面说的，类记载可以分为部分加载（不解析）和完全加载（包括解析）。我们创建自定义类加载器的时候，可能要调用resolveClass。