JAVA基础：了解JAVAclassloader PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/145/2021_2022_JAVA_E5_9F_BA_E7_A1_80_c104_145429.htm 与 C 或 C 编写的程序不同，Java 程序并不是一个可执行文件，而是由许多独立的类文件组成，每一个文件对应于一个 Java 类。 此外，这些类文件并非立即全部都装入内存，而是根据程序需要装入内存。ClassLoader 是 JVM 中将类装入内存的那部分。 而且，Java ClassLoader 就是用 Java 语言编写的。这意味着创建您自己的 ClassLoader 非常容易，不必了解 JVM 的微小细节。 为什么编写 ClassLoader? 如果 JVM 已经有一个 ClassLoader，那么为什么还要编写另一个呢？问得好。缺省的 ClassLoader 只知道如何从本地文件系统装入类文件。不过这只适合于常规情况，即已全部编译完 Java 程序，并且计算机处于等待状态。 但 Java 语言最具新意的事就是 JVM 可以非常容易地从那些非本地硬盘或从网络上获取类。例如，浏览者可以使用定制的 ClassLoader 从 Web 站点装入可执行内容。 有许多其它方式可以获取类文件。除了简单地从本地或网络装入文件以外，可以使用定制的 ClassLoader 完成以下任务： 在执行非置信代码之前，自动验证数字签名使用用户提供的密码透明地解密代码动态地创建符合用户特定需要的定制化构建类任何您认为可以生成 Java 字节码的内容都可以集成到应用程序中。 定制 ClassLoader 示例 如果使用过 JDK 或任何基于 Java 浏览器中的 Applet 查看器，那么您差不多肯定使用过定制的 ClassLoader。 Sun 最初发布 Java 语言时，其中最令人兴奋的一件事是观看这项新技术是如何执行在运行时从远程的 Web 服务器装入

的代码。（此外，还有更令人兴奋的事 -- Java 技术提供了一种便于编写代码的强大语言。）更一些令人激动的是它可以执行从远程 Web 服务器通过 HTTP 连接发送过来的字节码。此项功能归功于 Java 语言可以安装定制 ClassLoader。Applet 查看器包含一个 ClassLoader，它不在本地文件系统中寻找类，而是访问远程服务器上的 Web 站点，经过 HTTP 装入原始的字节码文件，并把它们转换成 JVM 内的类。 浏览器和 Applet 查看器中的 ClassLoaders 还可以做其它事情：它们支持安全性以及使不同的 Applet 在不同的页面上运行而互不干扰。 Luke Gorrie 编写的 Echidna 是一个开放源码包，它可以使您在单个虚拟机上运行多个 Java 应用程序。它使用定制的 ClassLoader，通过向每个应用程序提供该类文件的自身副本，以防止应用程序互相干扰。 我们的 ClassLoader 示例 了解了 ClassLoader 如何工作以及如何编写 ClassLoader 之后，我们将创建称作 CompilingClassLoader (CCL) 的 Classloader。CCL 为我们编译 Java 代码，而无需要我们干涉这个过程。它基本上就类似于直接构建到运行时系统中的 "make" 程序。 注：进一步了解之前，应注意在 JDK 版本 1.2 中已改进了 ClassLoader 系统的某些方面（即 Java 2 平台）。本教程是按 JDK 版本 1.0 和 1.1 写的，但也可以在以后的版本中运行。 Java 2 中 ClassLoader 的变动描述了 Java 版本 1.2 中的变动，并提供了一些详细信息，以便修改 ClassLoader 来利用这些变动。 ClassLoader 的基本目标是对类的请求提供服务。当 JVM 需要使用类时，它根据名称向 ClassLoader 请求这个类，然后 ClassLoader 试图返回一个表示这个类的 Class 对象。 通过覆盖对应于这个过程不同阶段的方法，可以创建定制的

ClassLoader。 在本文的其余部分，您会学习 Java ClassLoader 的关键方法。您将了解每一个方法的作用以及它是如何适合装入类文件这个过程的。您也会知道，创建自己的 ClassLoader 时，需要编写什么代码。 在下文中，您将会利用这些知识来使用我们的 ClassLoader 示例 -- CompilingClassLoader。 方法 loadClass ClassLoader.loadClass() 是 ClassLoader 的入口点。其特征如下： Class loadClass( String name, boolean resolve ). 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 Foo 或 java.lang.Object。 resolve 参数告诉方法是否需要解析类。在准备执行类之前，应考虑类解析。并不总是需要解析。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。 在 Java 版本 1.1 和以前的版本中，loadClass 方法是创建定制的 ClassLoader 时唯一需要覆盖的方法。（Java 2 中 ClassLoader 的变动提供了关于 Java 1.2 中 findClass() 方法的信息。） 方法 defineClass defineClass 方法是 ClassLoader 的主要诀窍。该方法接受由原始字节组成的数组并把它转换成 Class 对象。原始数组包含如从文件系统或网络装入的数据。 defineClass 管理 JVM 的许多复杂、神秘和倚赖于实现的方面 -- 它把字节码分析成运行时数据结构、校验有效性等等。不必担心，您无需亲自编写它。事实上，即使您想要这么做也不能覆盖它，因为它已被标记成最终的。 方法 findSystemClass 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 将原始字节转换成 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。（Java 2 中 ClassLoader 的变动提供了关于

Java 版本 1.2 这个过程变动的详细信息。） 