走进Java7模块系统Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/644/2021_2022__E8_B5_B0_E8_BF_9BJava_c104_644556.htm 笔者在观看过Devoxx关于Jigsaw的一段演示后，我很兴奋，觉得它应该会是针对复杂类路径版本问题和JAR陷阱等问题的解决方案。开发者最终能够使用他们所期望的任何Xalan版本，而无需被迫使用授权机制。不幸的是，通往更加有效的模块系统的征途并不是很清晰。 在研究确实问题之前，我们先来看一些基本概念： 模块化是解决复杂性问题很重要的工具。把应用分成不同的部分(模块、库、包、子项目和组件)，再分别进行计算，是行之有效的方式。模块化的最终目的是能定义出一套API用于模块间的沟通。 如果模块间所有的通讯都只通过这种API来实现，那么模块是松耦合的，于是：改变某个模块的实现会很容易，开发和测试各个模块能很容易独立开来，面向对象模式也是类似的道理。在OOP中，理想的状况是拥有大量小的、可重用的、简单并分离良好的对象。在模块系统中，就可以完美地实现小的、可重用的、简单并分离良好的模块。它们的想法和最初的动机是完全一样的，只是规模有所不同。 逻辑分离传统上，Java中有两种办法来实现模块化。逻辑分离是最自然的方式。它包括将应用程序分割成逻辑模块(子项目)，最后再部署成一个完整的应用。通过定义正确的包来实现逻辑分离也是可能的，但更通用的办法是把应用分割成一些存档文件(也就是JAR包)。逻辑分离里能促进模块的重用，有助实现模块间的松耦合。你甚至有可能定义一个API，然后宣布所有模块间的通讯都要通过这个给定的API来实现。这样的想

法有个大问题，那就是你很难强破大家都采用这种限制性用法，而且没有任何一种机制能够确保这个API的用法。你也没法把那些应用通过给定模块来使用的类和作为公共API一部分的类区分开来。如果一个类是“ 公共的 ”，那它就可以被任何其他类使用，无论调用它的那个类属于哪个模块。另一方面，受保护的或者包级可见性的类在其模块内部的调用也有限制。通常来说，涵盖了一些包以及包中类的模块需要能够互相调用。因此即使某个应用是由一些逻辑模块组成，但如果这些模块是耦合的，那么分离也根本没有用处。 物理分离另外一个传统的办法就是物理上的分离。你可以通过将应用分割成不同的组件，然后把每个组件部署到不同的JVM上而实现分离。这些组件间通过远程访问机制进行通讯，比如RMI、CORBA或者WebServices。物理分离实现了分离，也实现了松耦合，但负面影响是开支很大。为实现分离而专门采用远程访问机制，有点杀鸡用牛刀的味道。这会增加开发和部署不必要的复杂性，性能上所受到的影响也不能忽视的。 模块系统的作用位于逻辑分离和物理分离之间。它强调模块分离，但各个模块仍然部署到同一个JVM中，而且模块间的通讯由简单传统的方法调用组成，因此不会有运行 时的开支负担。在Java生态系统中最流行的模块框架是OSGi。它是一个成熟的规范，具有几个不同的实现。在OSGi中，模块被称作bundle，每个 bundle等同于一个JAR。每个bundle也包含一个META-INF/MANIFEST.MF文件，这个文件会宣布导出哪些包(package) 以及导入哪些包。只有那些导出包中的类才能被其他bundle所使用，而其他包都只面向包的内部成员，包里的类也只能在自身bundle中使用。 比如下面这个声明：

Manifest-Version: 1.0 Import-Package: net.krecan.spring.osgi.common Export-Package: net.krecan.spring.osgi.dao Bundle-Version: 1.0.0 Bundle-Name: demo-spring-osgi-dao Bundle-SymbolicName:net.krecan.spring-osgi.demo-spring-osgi-dao 