在FedoraCore上交付Java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_ti2020/493/2021_2022__E5_9C_A8FedoraC_c67_493544.htm 在Fedora的发布版中，Fedora4是首个包含了大量用Java编程语言编写的代码的版本。正是由于GNU类路径（Classpath）和GNU gcj（GNU Compiler for Java）的改进才促成了这些附加部分。 GCJ基础 首先，GNU gcj不是Java. 然而，gcj的目标是实现一个完整的系统，该系统兼容于Java并且将预编译器（ahead-of-time compiler）置于中心。它拥有一个基于GNU类路径的净化类库和一个内置的解释器。其编译器可以将Java源文件、类文件、甚至是整个jar文件编译成目标码。 以前gcj对待Java采取的是一种“ 激进且传统 ”的方式，它认为Java好像是C 的某个不常用的方言似的。这么做有好的一面，然而不幸的是两者运行时的链接模型差异太大因此当遇到一些规模比较大、复杂度比较高的Java应用时，这种方法就无能为力了，特别是面对那些有着复杂的类加载机制的Java应用时，显得尤为突出。 在GCC4.0的发布版中，我们对gcj实现了一种新的编译方式，称之为二进制兼容性ABI（Binary Compatibility Application Binary Interface）。这种编译方式将所有的链接推迟到运行时刻进行并且完全实现了Java的二进制兼容规范正好是让预编译的代码与类装载结合所需要的。 我们还增加了一个类映射数据库。在运行时，只要我们定义好一个类，gcj运行时（叫做“ libgcj ”）就会在数据库中寻找这个类。如果找到该类（注意我们这里使用的是类的“ 内容 ”，而不仅仅是类名），那么libgcj就会映射到该共享库中，该库包含了编译后的类。 这两个改变使得我们可

以做一些更强大的事情：我们可以预编译Java程序而不必要求任何应用级的改变。此外，由于采用了新的方式对字节码进行校验，我们还能确保编译的代码在运行时的类型安全。 构建RPM 在Fedora Core上构建Java应用是很简单的现存的构建方式不会发生变化。Fedora Core舶来了“ Ant ”并且使用来自Eclipse的Java编译器将Java代码编译成字节码。 描述如何编写RPM已经超出了本文讨论的范围，但是Fedora RPM指南上有一些有用的信息，JPackage上也有一些Java特定的指南。 一旦你的程序被编译成字节码，你就可以将他们编译为本地代码。因为gcj尚不包含一个即时编译器（JIT），这就是其获得合理性能的方法。在某些情况下，其性能可能会超过已有即时编译器，因为gcj使用了共享库……当你同时运行应用程序的多个实例时，你就会看到这种巨大的差异。 Fedora提供了两个程序，使得本地编译包的工作变得更简单。我们在构建RPM时会使用到他们。 第一个程序是“ aot-compile-rpm ”。它会搜索jar文件并且使用gcj将他们编译到共享库中。aot-compile-rpm知道一些gcj特定的技巧，例如在编译前将比较大的jar文件分割为若干个小的文件（在运行时编译一个大的jar文件将耗费大量内存资源），在链接结果共享库时使用Bsymbolic（这会导致运行时性能改善）。 假如你没有在构建RPM，那么一个替代方案就是直接将程序中的jar文件编译到共享库中。这里我展示一个最简单的方法（我之前提到过，对于一个大的jar文件，这样做会非常慢）： gcj -fjni -findirect-dispatch -fPIC -shared \ -Wl，-Bsymbolic -o foo.jar.so foo.jar 