专业语言：Java类装载体系中的隔离性 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022__E4_B8_93_E4_B8_9A_E8_AF_AD_E8_c104_144380.htm Java中类的查找与装载出现的问题总是会时不时出现在Java程序员面前，这并不是什么丢脸的事情，相信没有一个Java程序员没遇到过ClassNotException,因此不要为被人瞅见自己也犯这样的错误而觉得不自然，但是在如果出现了ClassNotFoundException后异常后一脸的茫然，那我想你该了解一下java的类装载的体制了，同时为了进行下面的关于类装载器之间的隔离性的讨论，我们先简单介绍一下类装载的体系结构。1. Java类装载体系结构装载类的过程非常简单：查找类所在位置，并将找到的Java类的字节码装入内存，生成对应的Class对象。Java的类装载器专门用来实现这样的过程，JVM并不止有一个类装载器，事实上，如果你愿意的话，你可以让JVM拥有无数个类装载器，当然这除了测试JVM外，我想不出还有其他的用途。你应该已经发现到了这样一个问题，类装载器自身也是一个类，它也需要被装载到内存中来，那么这些类装载器由谁来装载呢，总得有个根吧？没错，确实存在这样的根，它就是神龙见首不见尾的Bootstrap ClassLoader. 为什么说它神龙见首不见尾呢，因为你根本无法在Java代码中抓住哪怕是它的一点点的尾巴，尽管你能时时刻刻体会到它的存在，因为java的运行环境所需要的所有类库，都由它来装载，而它本身是C写的程序，可以独立运行,可以说是JVM的运行起点,伟大吧。在Bootstrap完成它的任务后，会生成一个AppClassLoader(实际上之前系统还会使用扩展类装载器ExtClassLoader，它用于装

载Java运行环境扩展包中的类),这个类装载器才是我们经常使用的，可以调用ClassLoader.getSystemClassLoader() 来获得，我们假定程序中没有使用类装载器相关操作设定或者自定义新的类装载器，那么我们编写的所有java类通通会由它来装载，值得尊敬吧。AppClassLoader查找类的区域就是耳熟能详的Classpath，也是初学者必须跨过的门槛，有没有灵光一闪的感觉，我们按照它的类查找范围给它取名为类路径类装载器。还是先前假定的情况，当Java中出现新的类，AppClassLoader首先在类传递给它的父类类装载器，也就是Extion ClassLoader，询问它是否能够装载该类，如果能，那AppClassLoader就不干这活了，同样Extion ClassLoader在装载时，也会先问问它的父类装载器。我们可以看出类装载器实际上是一个树状的结构图，每个类装载器有自己的父亲，类装载器在装载类时，总是先让自己的父类装载器装载(多么尊敬长辈),如果父类装载器无法装载该类时，自己就会动手装载，如果它也装载不了，那么对不起，它会大喊一声：Exception，class not found。有必要提一句，当由直接使用类路径装载器装载类失败抛出的是NoClassDefFoundException异常。如果使用自定义的类装载器loadClass方法或者ClassLoader的findSystemClass方法装载类，如果你不去刻意改变，那么抛出的是ClassNotFoundException。我们简短总结一下上面的讨论：1.JVM类装载器的体系结构可以看作是树状结构。2.父类装载器优先装载。在父类装载器装载失败的情况下再装载，如果都装载失败则抛出ClassNotFoundException或者NoClassDefFoundError异常。那么我们的类在什么情况下被装载的呢？2. 类如何被装载 在java2中，JVM是如何装载类

的呢，可以分为两种类型，一种是隐式的类装载，一种式显式的类装载。2.1 隐式的类装载 隐式的类装载是编码中最常用得方式：A b = new A().如果程序运行到这段代码时还没有A类，那么JVM会请求装载当前类的类装器来装载类。问题来了，我把代码弄得复杂一点点，但依旧没有任何难度，请思考JVM得装载次序：package test.Public class A{ public void static main(String args[]){ B b ＝ new B(). }}class B{C c.}class C{}揭晓答案，类装载的次序为A->B，而类C根本不会被JVM理会,先不要惊讶，仔细想想，这不正是我们最需要得到的结果。我们仔细了解一下JVM装载顺序。当使用Java A命令运行A类时，JVM会首先要求类路径类装载器(AppClassLoader)装载A类，但是这时只装载A，不会装载A中出现的其他类(B类)，接着它会调用A中的main函数，直到运行语句b ＝ new B()时，JVM发现必须装载B类程序才能继续运行，于是类路径类装载器会去装载B类，虽然我们可以看到B中有有C类的声明，但是并不是实际的执行语句，所以并不去装载C类，也就是说JVM按照运行时的有效执行语句，来决定是否需要装载新类，从而装载尽可能少的类，这一点和编译类是不相同的。