Java更多的库谜题80：更深层的反射 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_Java_E6_9B_B4_E5_A4_9A_c97_138720.htm 下面这个程序通过打印一个由反射创建的对象来产生输出。那么它会打印出什么呢？ public class Outer{ public static void main(String[] args) throws Exception{ new Outer().greetWorld(). } private void greetWorld()throws Exception { System.out.println(Inner.class.newInstance()). } public class Inner{ public String toString(){ return "Hello world". } }}这个程序看起来是最普通的Hello World程序的又一个特殊的变体。Outer中的main方法创建了一个Outer实例，并且调用了它的greetWorld方法，该方法以字符串形式打印了通过反射创建的一个新的Inner实例。Inner的toString方法总是返回标准的问候语，所以程序的输出应该与往常一样，是Hello World。如果你尝试运行这个程序，你会发现实际的输出比较长，而且更加令人迷惑： Exception in thread "main" InstantiationException: Outer$Innerat java.lang.Class.newInstance0(Class.java:335)at java.lang.Class.newInstance(Class.java:303)at Outer.greetWorld(Outer.java:7)at Outer.main(Outer.java:3)为什么会抛出这个异常呢？从5.0版本开始，关于Class.newInstance的文档叙述道：如果那个Class对象“ 代表了一个抽象类（abstract class），一个接口（interface），一个数组类（array class），一个原始类型（primitive type），或者是空（void）；或者这个类没有任何空的[也就是无参数的]构造器；或者实例化由于某些其他原因而失败，那么它就会抛出异常

”[JAVA-API]。这里出现的问题满足上面的哪些条件呢？遗憾的是，异常信息没有提供任何提示。在这些条件中，只有后2个有可能会满足：要么是Outer.Inner没有空的构造器，要么是实例化由于“某些其它原因”而失败了。正如Outer.Inner这种情况，当一个类没有任何显式的构造器时，Java会自动地提供一个不带参数的公共的缺省构造器[JLS 8.8.9]，所以它应该是有一个空构造器的。不过，newInstance方法调用失败的原因还是因为Outer.Inner没有空构造器！一个非静态的嵌套类的构造器，在编译的时候会将一个隐藏的参数作为它的第一个参数，这个参数表示了它的直接外围实例（immediately enclosing instance）[JLS 13.1]。当你在代码中任何可以让编译器找到合适的外围实例的地方去调用构造器的时候，这个参数就会被隐式地传递进去。但是，上述的过程只适用于普通的构造器调用，也就是不使用反射的情况。当你使用反射调用构造器时，这个隐藏的参数就需要被显式地传递，这对于Class.newInstance方法是不可能做到的。要传递这个隐藏参数的唯一办法就是使用java.lang.reflect.Constructor。当对程序进行了这样的修改后，它就可以正常的打印出 Hello World了： private void greetWorld() throws Exception{ Constructor c = Inner.class.getConstructor(Outer.class). System.out.println(c.newInstance(Outer.this)).}作为其他的选择，你可能观察到了，Inner实例并不需要一个外围的Outer实例，所以可以将Inner类型声明为静态的（static）。除非你确实是需要一个外围实例，否则你应该优先使用静态成员类（static member class）而不是非静态成员类[EJ Item 18]。下面

这个简单的修改就可以订正这个程序： public static class Inner{...}Java程序的反射模型和它的语言模型是不同的。反射操作处于虚拟机层次，暴露了很多从Java程序到class文件的翻译细节。这些细节当中的一部分由Java的语言规范来管理，但是其余的部分可能会随着不同的具体实现而有所不同。在Java语言的早期版本中，从Java程序到class文件的映射是很直接的，但是随着一些不能被虚拟机直接支持的高级语言特性的加入，如嵌套类（nested class）、协变返回类型（covariant return types）、泛型（generics）和枚举类型（enums），使得这种映射变得越来越复杂了。 考虑到从Java程序到class文件的映射的复杂度，请避免使用反射来实例化内部类。更一般地讲，当我们在用高级语言特性定义的程序元素之上使用反射的时候，一定要小心，从反射的视角观察程序可能不同与从代码的视角去观察它。请避免依赖那些没有被语言规范所管理的翻译细节。对于平台的实现者来说，这里的教训就是要再次重申，请提供清晰准确的诊断信息。