Java高级谜题90：荒谬痛苦的超类 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_E9_AB_98_E7_BA_A7_c97_138702.htm 下面的程序实际上不会做任何事情。更糟的是，它连编译也通不过。为什么呢？又怎么来订正它呢？ public class Outer { class Inner1 extends Outer{} class Inner2 extends Inner1{} } 这个程序看上去简单得不可能有错误，但是如果你尝试编译它，就会得到下面这个有用的错误消息： Outer.java:3: cannot reference this before supertype constructor has been called class Inner2 extends Inner1{} 好吧，可能这个消息不那么有用，但是我们还是从此入手。问题在于编译器产生的缺省的Inner2的构造器为它的super调用找不到合适的外部类实例。让我们来看看显式地包含了构造器的该程序： public class Outer { public Outer() {} class Inner1 extends Outer{ public Inner1() { super(). // 调用Object()构造器 } } class Inner2 extends Inner1{ public Inner2() { super(). // 调用Inner1()构造器 } } } 现在错误消息就会显示出多一点的信息了： Outer.java:12: cannot reference this before supertype constructor has been called super(). // 调用Inner1()构造器 ^ 因为Inner2的超类本身也是一个内部类，一个晦涩的语言规则登场了。正如大家知道的，要想实例化一个内部类，如类Inner1，需要提供一个外部类的实例给构造器。一般情况下，它是隐式地传递给构造器的，但是它也可以以expression.super(args)的方式通过超类构造器调用(superclass constructor invovation)显式地传递[JLS 8.8.7]。如果外部类实例是隐式传递的，编译器会自动产生表达式：它使用this来指代最内部的其超类是一个成员变

量的外部类。这确实有点绕口，但是这就是编译器所作的事情。在本例中，那个超类就是Inner1。因为当前类Inner2间接扩展了Outer类，Inner1便是它的一个继承而来的成员。因此，超类构造器的限定表达式直接就是this。编译器提供外部类实例，将super重写成this.super。 解释到这里，编译错误所含的意思可扩展为： Outer.java:12: cannot reference this before supertype constructor has been called this.super(). ^ 现在问题就清楚了：缺省的Inner2的构造器试图在超类构造器被调用前访问this，这是一个非法的操作[JLS 8.8.7.1]。解决这个问题的蛮力方法是显式地传递合理的外部类实例： public class Outer { class Inner1 extends Outer {} class Inner2 extends Inner1{ public Inner2() { Outer.this.super(). } } } 这样可以通过编译，但是它太复杂了。这里有一个更好的解决方案：无论何时你写了一个成员类，都要问问你自己，是否这个成员类真的需要使用它的外部类实例？如果答案是否定的，那么应该把它设为静态成员类。内部类有时是非常有用的，但是它们很容易增加程序的复杂性，从而使程序难以被理解。它们和泛型（谜题89）、反射（谜题80）以及继承（本谜题）都有着复杂的交互方式。在本例中，如果你将Inner1设为静态的便可以解决问题了。如果你将Inner2也设为静态的，你就会真正明白这个程序做了什么：确实是一个相当好的意外收获。 