Java理论与实践:伪typedef反模式 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_Java_E7_90_86_E8_AE_BA_c104_144740.htm 将泛型添加到 Java 语言中增加了类型系统的复杂性，提高了许多变量和方法声明的冗长程度。因为没有提供“ typedef ”工具来定义类型的简短名称，所以有些开发人员转而把扩展当作“ 穷人的 typedef ”，结果收到了良好的效果。 对于 Java 5.0 中新增的泛型工具，一个常见的抱怨就是，它使代码变得太冗长。原来用一行就够的变量声明不再存在了，与声明参数化类型有关的重复非常讨厌，特别是还没有良好地支持自动补足的 IDE。例如，如果想声明一个 Map，它的键是 Socket，值是 Future，那么老方法就是： Map socketOwner = new HashMap(). 比新方法紧凑得多： Map> socketOwner = new HashMap>(). 当然，新方法内置了更多类型信息，减少了编程错误，提高了程序的可读性，但是确实带来了更多声明变量和方法签名方面的前期工作。类型参数在声明和初始化中的重复看起来尤其没有必要；Socket 和 Future 需要输入两次，这迫使我们违犯了“ DRY ”原则（不要重复自己）。 合成类似于 typedef 的东西 添加泛型给类型系统增加了一些复杂性。在 Java 5.0 之前，“ type ”和“ class ”几乎是同义的，而参数化类型，特别是那些绑定的通配类型，使子类型和子类的概念有了显著区别。类型 ArrayList、ArrayList 和 ArrayList 是不同的类型，虽然它们是由同一个类 ArrayList 实现的。这些类型构成了一个层次结构；ArrayList 是 ArrayList 的超类型，而 ArrayList 是 ArrayList 的超类型。 对于原来的简单类型系统，像 C 的 typedef 这样的特

性没有意义。但是对于更复杂的类型系统，typedef 工具可能会提供一些好处。不知是好还是坏，总之在泛型加入的时候，typedef 没有加入 Java 语言。 有些人用作“ 穷人的 typedef ”的一个（坏的）做法是一个小小的扩展：创建一个类，扩展泛型类型，但是不添加功能，例如 SocketUserMap 类型，如清单 1 所示： 清单 1. 伪 typedef 反模式 不要这么做 public class SocketUserMap extends HashMap> { } SocketUserMap socketOwner = new SocketUserMap(). 我将这个技巧称为伪 typedef 反模式，它实现了将 socketOwner 定义简化为一行的这一（有问题的）目标，但是有些副作用，最终成为重用和维护的障碍。（对于有明确的构造函数而不是无参构造函数的类来说，派生类也需要声明每个构造函数，因为构造函数没有被继承。） 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com