最大化Java代码的可重用性Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E6_9C_80_E5_A4_A7_E5_8C_96J_c104_644678.htm 在程序员中似乎存在着一种日益普遍的观点，认为重用只是一个神话。或许是传统的面向对象编程方法中所存在的不足增加了重用的困难。本文介绍了从另外一种不同的途径使重用成为可能的三个步骤。 第一步：将功能实现从类实例的方法中移出 由于缺乏精确性，类继承不是非常理想的代码重用机制。换句话说，如果不继承一个类的数据成员和其他的方法，那么你就无法重用这个类的某个单独的方法。这些额外的不必要的负担使方法重用的代码变得复杂。派生类对其父类的依赖性也以入了额外的复杂性：对父类的改动会对子类造成影响；当修改任意一个类的时候，我们很难记得清哪个方法被覆盖，哪个没有；而且被覆盖的方法是否会调用父类中相应的方法并不非常清晰地显现。 任何执行单一概念任务的方法应该能够成为代码用的首选而独立存在。为了达到这个目标，我们必须会到过程化的编程模式，将代码从类实例的方法中移出，形成具有全局可见性的过程。为了提高这种过程的可重用性，过程代码应该象静态的通用方法一样编写：每个过程只能使用自己的输入参数，只能调用其他全局性的过程完成其工作，不能使用任何非本地的变量。这种对外部依赖的简化降低了过程使用的复杂性，也增加了在其他地方使用此过程的可能性。当然，由于其结构通常会变得更为清晰，即使抛开重用的目的不谈我们也可以从这种代码的组织方式中受益。 在Java中，方法不能脱离类而单独存在。因此，你可以对相关

的过程进行组织并使它们成为一个独立的类中的公共静态方法。例如，对于如下所示的一个类： class Polygon{ … public int getPerimeter(){…} public boolean isConvex(){…} public Boolean containsPoint(Point p){…} … } 可以将它改写成下面的形式： class Polygon { … public int getPerimeter() { return pPolygon.computePerimeter(this).} public boolean isConvex() { return pPolygon.isConvex(this).} public boolean containsPoint() { return pPolygon.containsPoint(this, p).} … } 在此处，nPolygon应该是这个样子： class pPolygon { static public int computePerimeter(Polygon polygon) {...} static public boolean isConvex(Polygon polygon) {...} static public boolean containsPoint(Polygon polygon, Point p) {...} } 