JAVA进阶:提高代码可重用性的三个措施 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/220/2021_2022_JAVA_E8_BF_9B_E9_98_B6_c67_220258.htm 本文介绍了三种修改现有代码提高其可重用性的方法，它们分别是：改写类的实例方法，把参数类型改成接口，选择最简单的参数接口类型。 措施一：改写类的实例方法 通过类继承实现代码重用不是精确的代码重用技术，因此它并不是最理想的代码重用机制。换句话说，如果不继承整个类的所有方法和数据成员，我们无法重用该类里面的单个方法。继承总是带来一些多余的方法和数据成员，它们总是使得重用类里面某个方法的代码复杂化。另外，派生类对父类的依赖关系也使得代码进一步复杂化：对父类的改动可能影响子类；修改父类或者子类中的任意一个类时，我们很难记得哪一个方法被子类覆盖、哪一个方法没有被子类覆盖；最后，子类中的覆盖方法是否要调用父类中的对应方法有时并不显而易见。 任何方法，只要它执行的是某个单一概念的任务，就其本身而言，它就应该是首选的可重用代码。为了重用这种代码，我们必须回归到面向过程的编程模式，把类的实例方法移出成为全局性的过程。为了提高这种过程的可重用性，过程代码应该象静态工具方法一样编写：它只能使用自己的输入参数，只能调用其他全局性的过程，不能使用任何非局部的变量。这种对外部依赖关系的限制简化了过程的应用，使得过程能够方便地用于任何地方。当然，由于这种组织方式总是使得代码具有更清晰的结构，即使是不考虑重用性的代码也同样能够从中获益。
在Java中，方法不能脱离类而单独存在。为此，我们可以把相

关的过程组织成为独立的类，并把这些过程定义为公用静态方法。 例如，对于下面这个类： class Polygon { . . public int getPerimeter() {...} public boolean isConvex() {...} public boolean containsPoint(Point p) {...} . . } 我们可以把它改写成： class Polygon { . . public int getPerimeter() {return pPolygon.computePerimeter(this).} public boolean isConvex() {return pPolygon.isConvex(this).} public boolean containsPoint(Point p) {return pPolygon.containsPoint(this, p).} . } 其中，pPolygon是： class pPolygon { static public int computePerimeter(Polygon polygon) {...} static public boolean isConvex(Polygon polygon) {...} static public boolean containsPoint(Polygon polygon, Point p) {...} }从类的名字pPolygon可以看出，该类所封装的过程主要与Polygon类型的对象有关。名字前面的p表示该类的唯一目的是组织公用静态过程。在Java中，类的名字以小写字母开头是一种非标准的做法，但象pPloygon这样的类事实上并不提供普通Java类的功能。也就是说，它并不代表着一类对象，它只是Java语言组织代码的一种机制。 在上面这个例子中，改动代码的最终效果是使得应用Polygon功能的客户代码不必再从Polygon继承。Polygon类的功能现在已经由pPolygon类以过程为单位提供。客户代码只使用自己需要的代码，无需关心Polygon类中自己不需要的功能。但它并不意味着在这种新式过程化编程中类的作用有所削弱。恰恰相反，在组织和封装对象数据成员的过程中，类起到了不可或缺的作用，而且正如本文接下来所介绍的，类通过多重接口实现多态性的能力本身也带来了卓越的代码重用支持。然而，由于用实例方法封装代码功能

并不是首选的代码重用手段，所以通过类继承达到代码重用和多态性支持也不是最理想的。