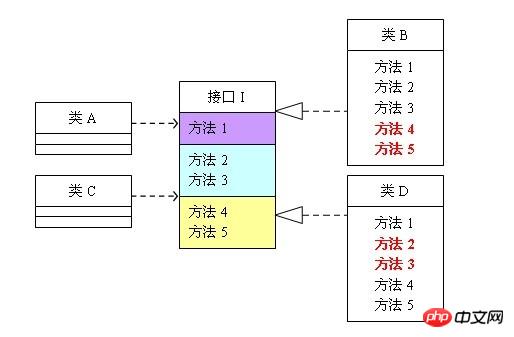
这篇文章主要介绍了接口隔离原则，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧

**定义：**客户端不应该依赖它不需要的接口；一个类对另一个类的依赖应该建立在最小的接口上。

**问题由来：**类A通过接口I依赖类B，类C通过接口I依赖类D，如果接口I对于类A和类B来说不是最小接口，则类B和类D必须去实现他们不需要的方法。

**解决方案：**将臃肿的接口I拆分为独立的几个接口，类A和类C分别与他们需要的接口建立依赖关系。也就是采用接口隔离原则。  
举例来说明接口隔离原则：

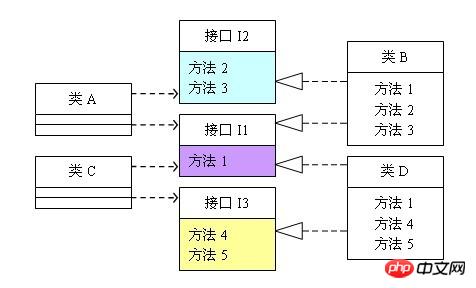


（图1 未遵循接口隔离原则的设计）

这个图的意思是：类A依赖接口I中的方法1、方法2、方法3，类B是对类A依赖的实现。类C依赖接口I中的方法1、方法4、方法5，类D是对类C依赖的实现。对于类B和类D来说，虽然他们都存在着用不到的方法（也就是图中红色字体标记的方法），但由于实现了接口I，所以也必须要实现这些用不到的方法。对类图不熟悉的可以参照程序代码来理解，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | interface I {    public void method1();    public void method2();    public void method3();    public void method4();    public void method5();  }    class A{    public void depend1(I i){      i.method1();    }    public void depend2(I i){      i.method2();    }    public void depend3(I i){      i.method3();    }  }    class B implements I{    public void method1() {      System.out.println("类B实现接口I的方法1");    }    public void method2() {      System.out.println("类B实现接口I的方法2");    }    public void method3() {      System.out.println("类B实现接口I的方法3");    }    //对于类B来说，method4和method5不是必需的，但是由于接口A中有这两个方法，    //所以在实现过程中即使这两个方法的方法体为空，也要将这两个没有作用的方法进行实现。    public void method4() {}    public void method5() {}  }    class C{    public void depend1(I i){      i.method1();    }    public void depend2(I i){      i.method4();    }    public void depend3(I i){      i.method5();    }  }    class D implements I{    public void method1() {      System.out.println("类D实现接口I的方法1");    }    //对于类D来说，method2和method3不是必需的，但是由于接口A中有这两个方法，    //所以在实现过程中即使这两个方法的方法体为空，也要将这两个没有作用的方法进行实现。    public void method2() {}    public void method3() {}      public void method4() {      System.out.println("类D实现接口I的方法4");    }    public void method5() {      System.out.println("类D实现接口I的方法5");    }  }    public class Client{    public static void main(String[] args){      A a = new A();      a.depend1(new B());      a.depend2(new B());      a.depend3(new B());        C c = new C();      c.depend1(new D());      c.depend2(new D());      c.depend3(new D());    }  } |

可以看到，如果接口过于臃肿，只要接口中出现的方法，不管对依赖于它的类有没有用处，实现类中都必须去实现这些方法，这显然不是好的设计。如果将这个设计修改为符合接口隔离原则，就必须对接口I进行拆分。在这里我们将原有的接口I拆分为三个接口，拆分后的设计如图2所示：



（图2 遵循接口隔离原则的设计）

照例贴出程序的代码，供不熟悉类图的朋友参考：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | interface I1 {    public void method1();  }    interface I2 {    public void method2();    public void method3();  }    interface I3 {    public void method4();    public void method5();  }    class A{    public void depend1(I1 i){      i.method1();    }    public void depend2(I2 i){      i.method2();    }    public void depend3(I2 i){      i.method3();    }  }    class B implements I1, I2{    public void method1() {      System.out.println("类B实现接口I1的方法1");    }    public void method2() {      System.out.println("类B实现接口I2的方法2");    }    public void method3() {      System.out.println("类B实现接口I2的方法3");    }  }    class C{    public void depend1(I1 i){      i.method1();    }    public void depend2(I3 i){      i.method4();    }    public void depend3(I3 i){      i.method5();    }  }    class D implements I1, I3{    public void method1() {      System.out.println("类D实现接口I1的方法1");    }    public void method4() {      System.out.println("类D实现接口I3的方法4");    }    public void method5() {      System.out.println("类D实现接口I3的方法5");    }  } |

接口隔离原则的含义是：建立单一接口，不要建立庞大臃肿的接口，尽量细化接口，接口中的方法尽量少。也就是说，我们要为各个类建立专用的接口，而不要试图去建立一个很庞大的接口供所有依赖它的类去调用。本文例子中，将一个庞大的接口变更为3个专用的接口所采用的就是接口隔离原则。在程序设计中，依赖几个专用的接口要比依赖一个综合的接口更灵活。接口是设计时对外部设定的“契约”，通过分散定义多个接口，可以预防外来变更的扩散，提高系统的灵活性和可维护性。

说到这里，很多人会觉的接口隔离原则跟之前的单一职责原则很相似，其实不然。其一，单一职责原则原注重的是职责；而接口隔离原则注重对接口依赖的隔离。其二，单一职责原则主要是约束类，其次才是接口和方法，它针对的是程序中的实现和细节；而接口隔离原则主要约束接口接口，主要针对抽象，针对程序整体框架的构建。

采用接口隔离原则对接口进行约束时，要注意以下几点：

* 接口尽量小，但是要有限度。对接口进行细化可以提高程序设计灵活性是不挣的事实，但是如果过小，则会造成接口数量过多，使设计复杂化。所以一定要适度。
* 为依赖接口的类定制服务，只暴露给调用的类它需要的方法，它不需要的方法则隐藏起来。只有专注地为一个模块提供定制服务，才能建立最小的依赖关系。
* 提高内聚，减少对外交互。使接口用最少的方法去完成最多的事情。

运用接口隔离原则，一定要适度，接口设计的过大或过小都不好。设计接口的时候，只有多花些时间去思考和筹划，才能准确地实践这一原则