


设计原则

2025年6月10日 星期二 10:19

- 1. 开闭原则OCP: Open-Closed Principle 实体可扩展, 不可修改
- 2. 里氏代换原则LSP: Liskov Substitution Principle 父类适用的子类也适用
- 3. 依赖倒置原则DIP: Dependency Inversion Principle 针对接口编程
- 4. 接口隔离原则ISP: Interface Segregation Principle 不使用单一的总接口
- 5. 组合复用原则CRP: Composition Reuse Principle 优先使用组合而非继承
- 6. 迪米特法则LoD: Law of Demeter 每个单元应对其其他单元尽可能少的了解
- 7. 单一职责原则 (SRP) 一个类只负责一方面的职责




### 设计原则：里氏代换原则

凡是父类适用的地方子类应当也适用

- Liskov Substitution Principle
- 一个软件如果使用的是一个父类的话，如果把该父类换成子类，它不能察觉到父类对象和子类对象的区别
- 凡是父类适用的地方子类也适用
- 继承只有满足里氏代换原则才是合理的

如果两个具体的类A，B之间的关系违反了LSP的设计，(假设是从B到A的继承关系)那么根据具体的情况可以在下面的两种重构方案中选择一种。

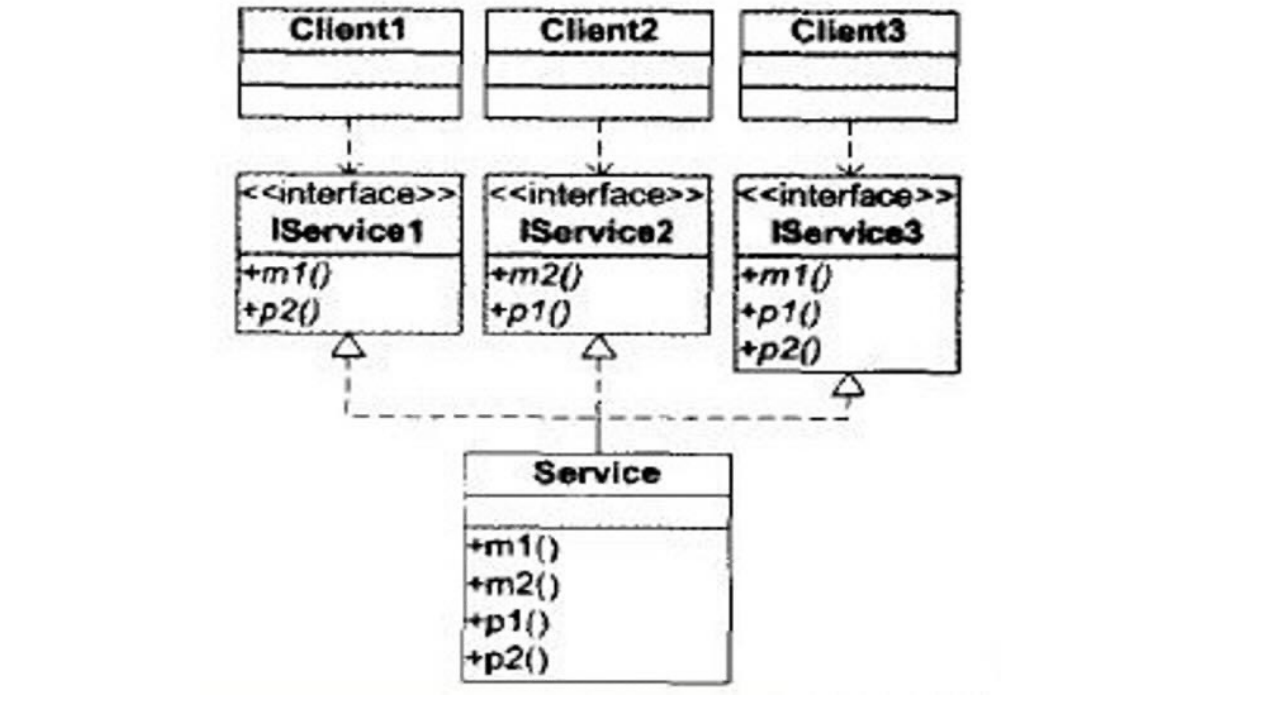
- 创建一个新的抽象类C，作为两个具体类的超类，将A，B的共同行为移动到C中来解决问题。
- 从B到A的继承关系改为委派关系。



### 设计原则：接口隔离原则

使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口。

- 角色的合理划分
  - 一个接口应当简单的只代表一个角色
- 接口污染
  - 过于臃肿的接口是对接口的污染
  - 不要把没什么关系的接口合并在一起




### 设计原则-单一职责原则 (SRP)

- 单一职责原则是最简单的面向对象设计原则，它用于控制类的粒度大小。单一职责原则定义如下：
- 一个类只负责一个功能领域中的相应职责，或者可以定义为：就一个类而言，应该只有一个引起它变化的原因。

问题由来：类 T 负责两个不同的职责：职责 P 1、职责 P 2。当由于职责 P 1 需求发生改变而需要修改类 T 时，有可能会导导致原来运行的职责 P 2 功能发生故障。解决方法：分别建立两个类完成对应的功能。

解决方案：遵循单一职责原则。分别建立两个类 T1、T2，使T1完成职责P1功能，T2完成职责P2功能。这样，当修改类T1时，不会使职责P2发生故障风险；同理，当修改T2时，也不会使职责P1发生故障风险。



### 设计原则：针对接口编程

要针对接口编程  
不要针对实现编程

- “Program to an interface, not an implementation”


不把变量声明为某个特定的具体类的实例对象，而让其遵从抽象类定义的接口

### 1. 开-闭原则OCP

- 软件组成实体应该是可扩展的，但是不可修改的。（Software Entities Should Be Open For Extension, But Closed For Modification）
- 开放-封闭法则认为应该试图去设计出永远也不需要改变的模块。

### 4. 组合复用原则

- 优先使用(对象)组合，而非(类)继承
- Favor Composition Over Inheritance
- 合成复用原则(Composite Reuse Principle, CRP) 又称为组合/聚合复用原则(Composition/Aggregate Reuse Principle, CARP)，其定义如下：
- 尽量使用对象组合，而不是继承来达到复用的目的。



### 设计原则7：迪米特法则

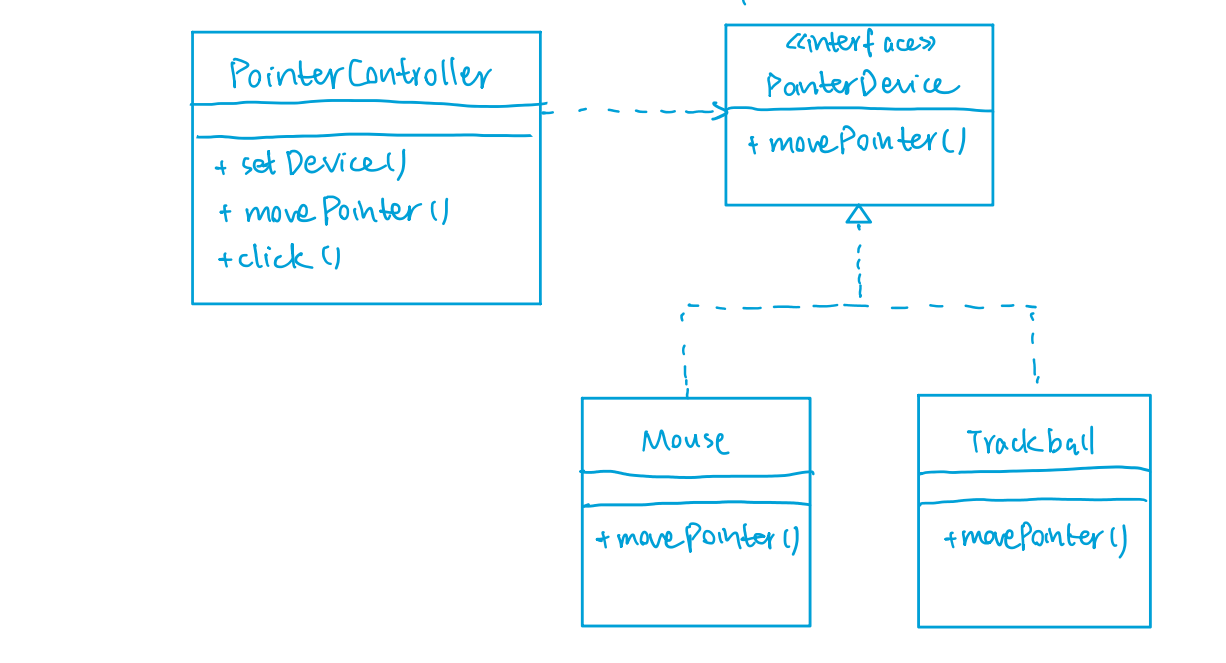
每个软件单元对其它单元尽可能少了解而且仅限于那些与自己密切相关的单元

- Law of Demeter
- 又叫做最少知识原则
- “只与你直接的朋友们通信”
- 不要跟“陌生人”说话
- 也就是信息隐藏、封装

1. 我们知道轨迹球可以用来定位图形界面的指针的位置，鼠标也可以定位图形界面指针的位置，而且两者的实现非常相似，我们现在让轨迹球类继承鼠标类。请问这样的设计模式有什么问题，违反了哪个设计原则，并说明你的理由。对于这样的问题，你认为应该怎样改进，画出类图和写出代码框架。

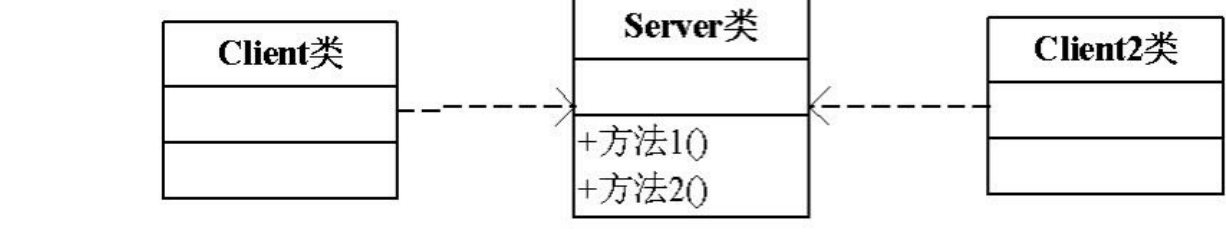
违反 LSP，Trackball 继承 Mouse，但重写了 MovePointer() 方法，其内部实现与父类逻辑不同

违反 CRP，错误地使用继承而非组合实现复用



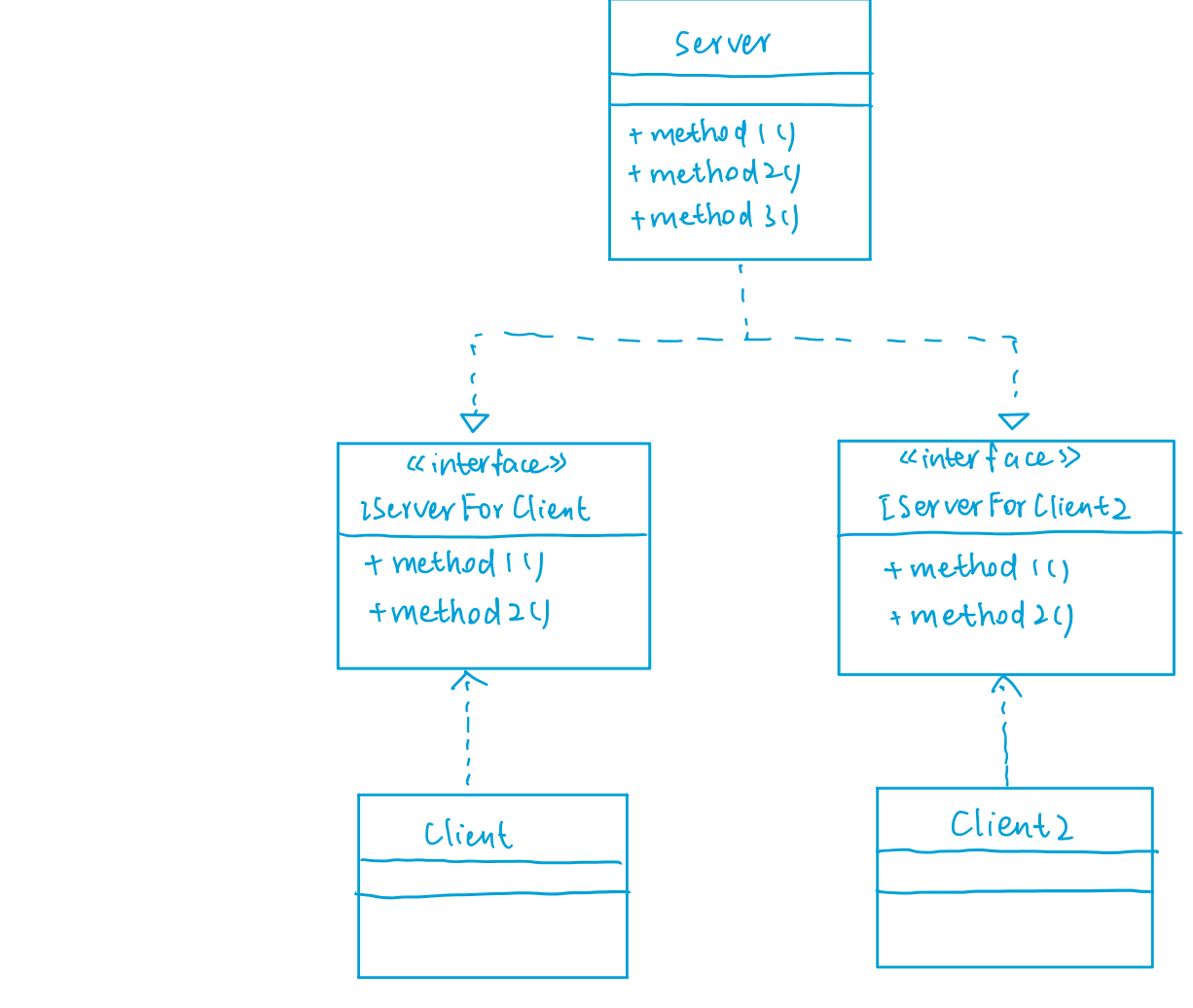
eg:

2. 如下所示类图结构，具体类 Client 调用了具体类 Server 中的方法，来实现业务逻辑。同时 Server 类还被具体类 Client2 类调用，这样如果 Client2 类需要 Server 类中的方法 1() 和方法 2() 的方法实现发生变动时，将影响 Client 类的业务逻辑。请设计一个好的方案，使 Server 类因 Client2 类要求发生变更的时候，不影响 Client 类的业务逻辑。请画出调整后的类图。



使用接口隔离原则 (ISP)

通过拆分接口，让 Client 和 Client2 依赖不同的抽象，避免相互影响



与 LSP 进行区分：SRP 主要用于一个类中有太多方法的情况

- 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。
- 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，主要针对抽象，针对程序整体框架的构建。

```
// 指针设备接口（策略接口）
public interface PointerDevice {
    void movePointer(int dx, int dy); // 移动指针
    void click(); // 点击（可选）
}

// 鼠标：通过位移移动指针
public class Mouse implements PointerDevice {
    @Override
    public void movePointer(int dx, int dy) {
        System.out.println("[Mouse] 移动指针: (" + dx + ", " + dy + ")");
    }

    @Override
    public void click() {
        System.out.println("[Mouse] 点击");
    }
}

// 轨迹球：通过旋转移动指针
public class Trackball implements PointerDevice {
    @Override
    public void movePointer(int dx, int dy) {
        System.out.println("[Trackball] 旋转轨迹球 -> (" + dx + ", " + dy + ")");
    }

    @Override
    public void click() {
        System.out.println("[Trackball] 按压轨迹球");
    }
}

// 指针控制器（上下文）
public class PointerController {
    private PointerDevice device; // 组合策略接口

    // 动态切换设备
    public void setDevice(PointerDevice device) {
        this.device = device;
    }

    // 委托给具体策略
    public void movePointer(int dx, int dy) {
        device.movePointer(dx, dy);
    }

    public void click() {
        device.click();
    }
}

public class Client {
    public static void main(String[] args) {
        PointerController controller = new PointerController();

        // 使用鼠标
        controller.setDevice(new Mouse());
        controller.movePointer(10, 20); // 输出: [Mouse] 移动指针: (10, 20)
        controller.click(); // 输出: [Mouse] 点击

        // 切换为轨迹球
        controller.setDevice(new Trackball());
        controller.movePointer(5, 5); // 输出: [Trackball] 旋转轨迹球 -> (5, 5)
        controller.click(); // 输出: [Trackball] 按压轨迹球
    }
}
```