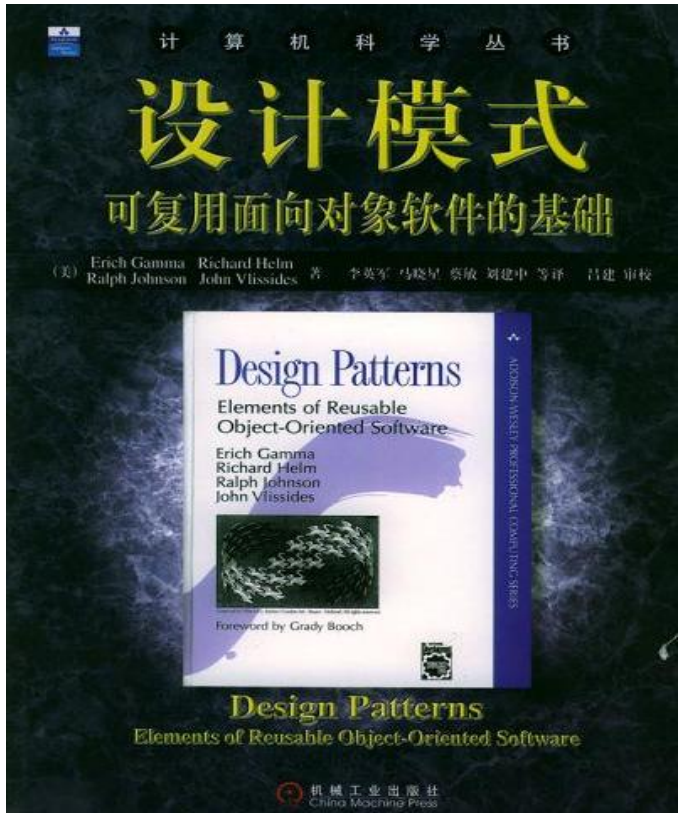
设计模式浅析

2012年3月

设计模式的概念

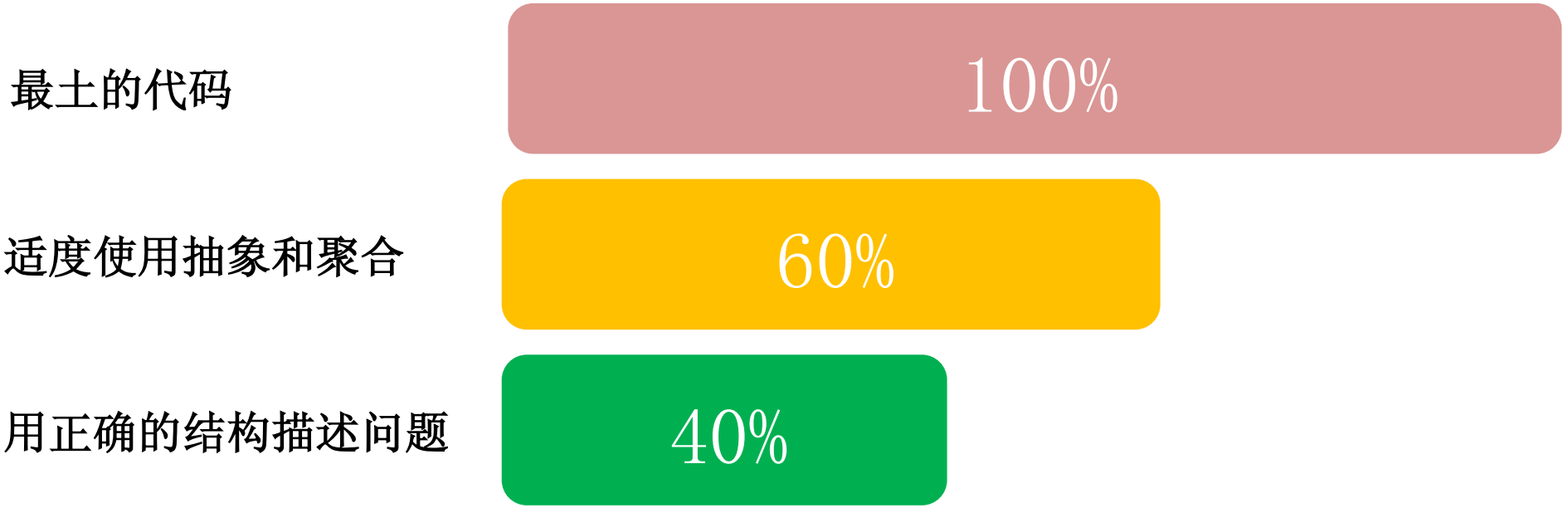
- 设计模式（Design Pattern）
  —一些代码设计经验的总结

- 反复的被使用
- 较大的推广范围
- 经过分类



为什么要了解设计模式

- **目的：通过适度的抽象和复用进而少写代码**



为什么要了解设计模式

- 目的：通过问题的有效分类，“举一反三”，降低思维复杂度。

结构？

为什么是浅析

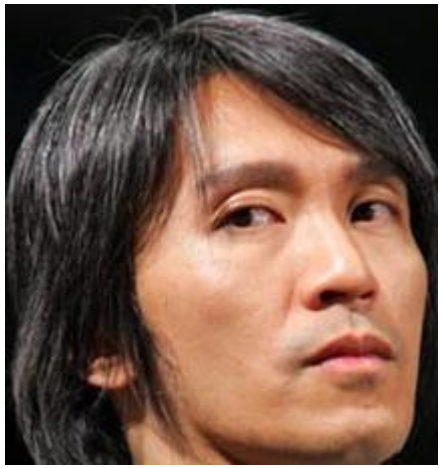
伤不起啊？

对初级程序员来说，设计模式有的时候有点天书？

架构师在说什么？太“饶舌”了！

浅析 = 少讲几个 + 讲的透彻点儿+ 举例子

进入正题



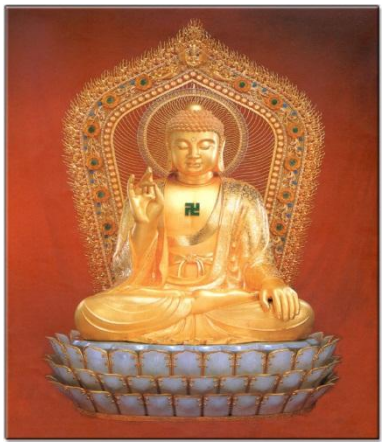
CUT 等一下…

原则上说呢？

组织内部也要讲原则……

原则？

虽然你都是“模式”了，但是你也要遵循一些软件开发和设计的“普世价值”。



设计模式遵循的原则 - 开闭原则

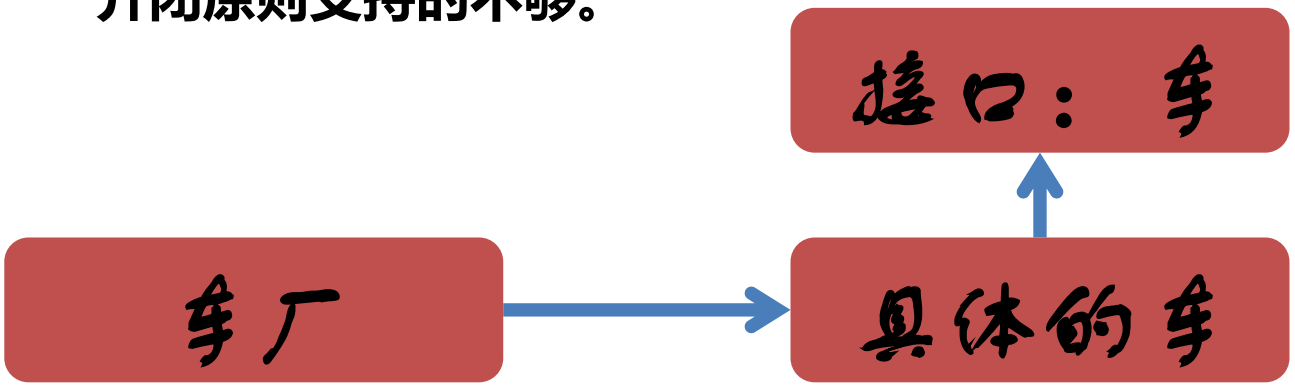
开闭原则（Open – Closed Principle OCP)

Open ---- 对扩展开放

Close ---- 对修改关闭

格言：针对需要适应变化的地方“必须”进行“抽象”

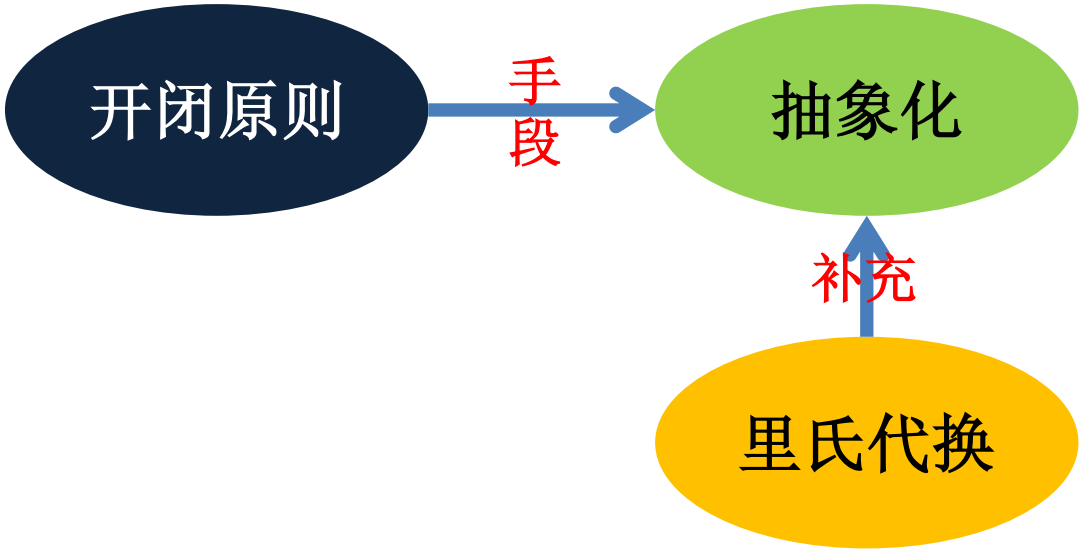
反例：在扩展度很高的场景不合适的使用了简单工厂模式，导致对开闭原则支持的不够。



设计模式遵循的原则 - 里氏代换原则

里氏代换原则（Liskov Subsitution Principle LSP)

所有**父类**的代码，**都**可以用其**子类**进行**替换**。**任何**继承关系**都不能违反此原则**。



设计模式遵循的原则 - 里氏代换原则

类 A 和 类B （B是A的子类）如果违法了里氏代换原则，如何处理？

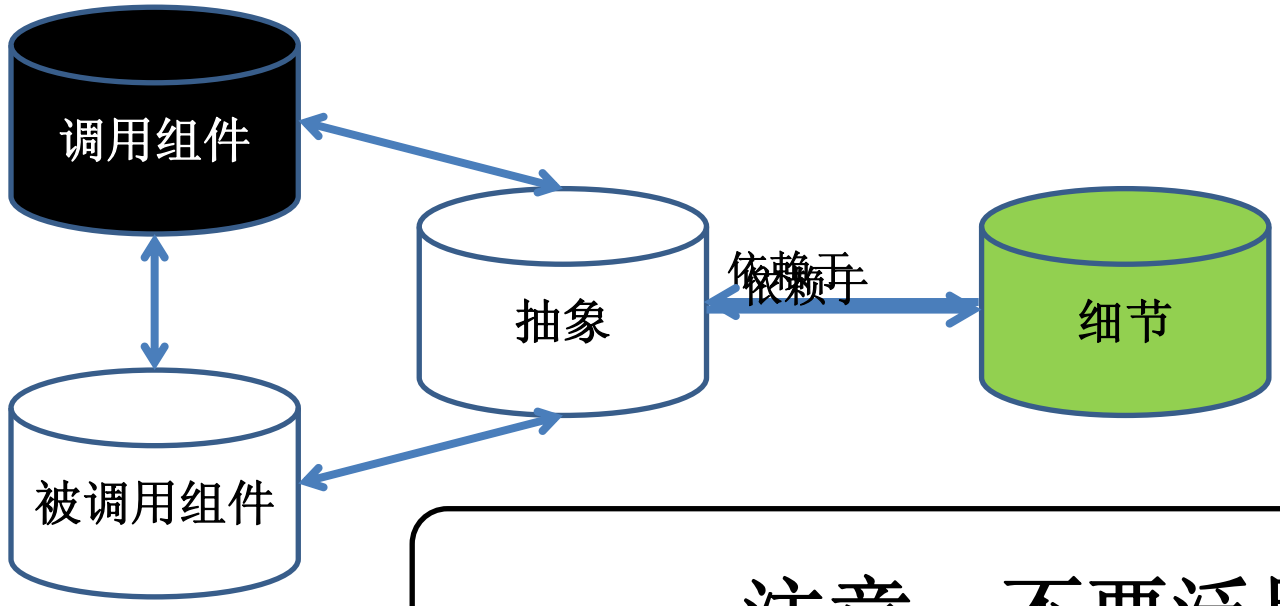
方法1：

方法2：

注意： UnsupportedOperationException

设计模式遵循的原则 - 依赖倒置原则

依赖倒置原则（Dependency Inversion Principle DIP)



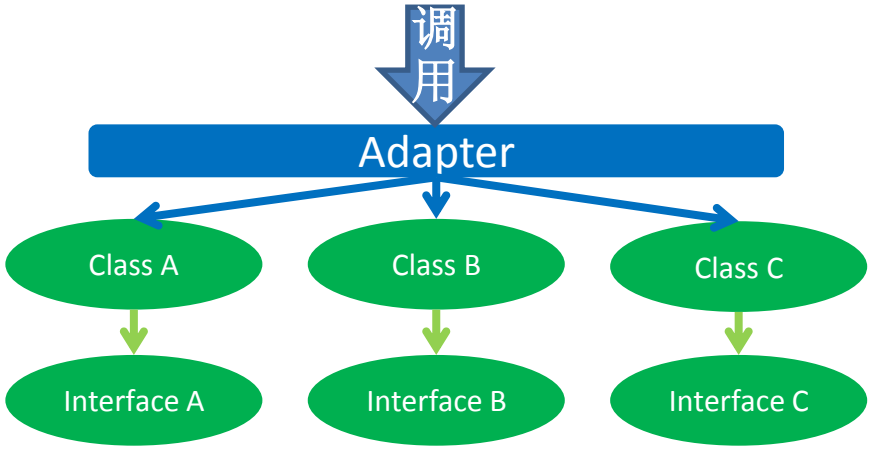
注意：不要泛用

设计模式遵循的原则 - 接口隔离原则

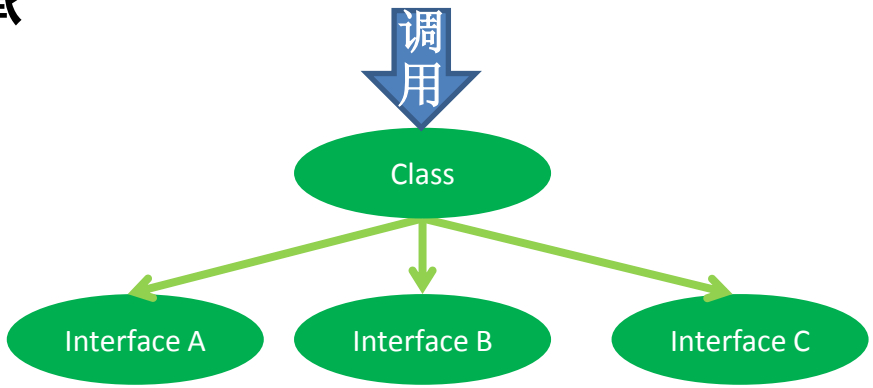
接口隔离原则（Interface Segregation Principle ISP)

提供尽量是**单一方面能力**的接口，而不是具有多重复杂属性的接口。 当需要多重能力时：

手段1 ：Adapter



手段2 ：多重继承



设计模式遵循的原则 - 组/聚合原则

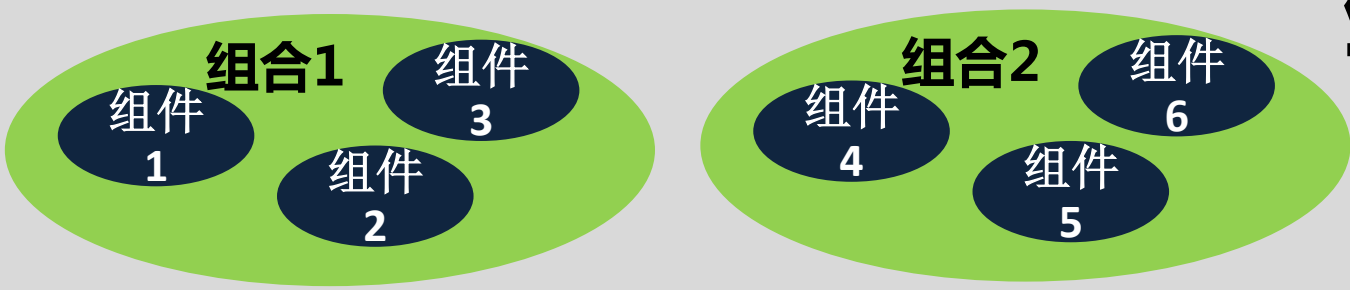
组合/聚合原则（Composition/Aggregation Principle CARP）

少用慎用继承，在能够用合成关系来描述一个问题时，尽量使用合成关系。

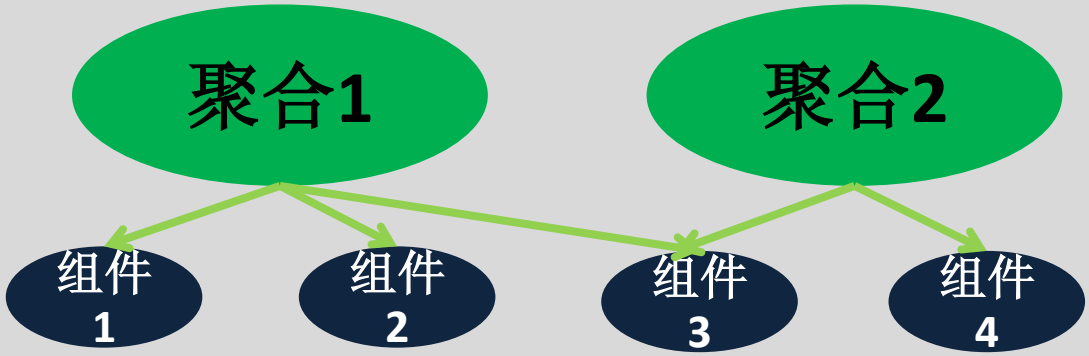
设计模式遵循的原则 – 组/聚合原则

组合与聚合的区别在于：

在组合中：



在聚合中：



生命周期不同

设计模式遵循的原则 - 迪米特法则

迪米特法则（Law of Demeter LoD）

最少知识原则

“不要和陌生人说话”

门面模式

调用者模式

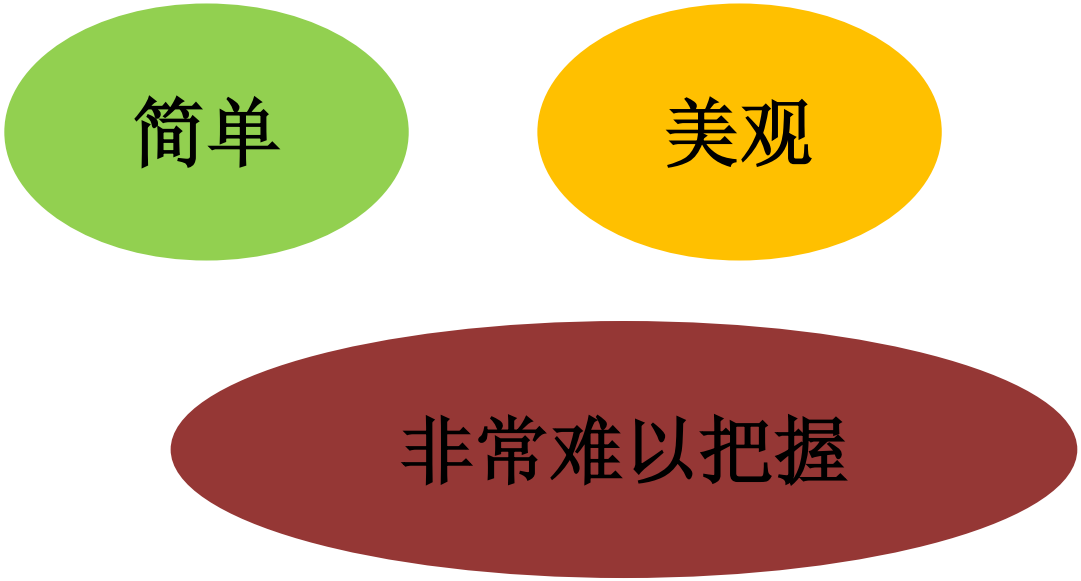
实现类和实现类之间
实现实体和实现实体之间
尽量少的发生依赖关系。

注意：不要泛用

设计模式遵循的原则 - 单一职责原则

单一职责原则（Single Responsibility Principle SRP）

每一个软件实体之负责某一种或者某一方面的职责的实现



离开原则

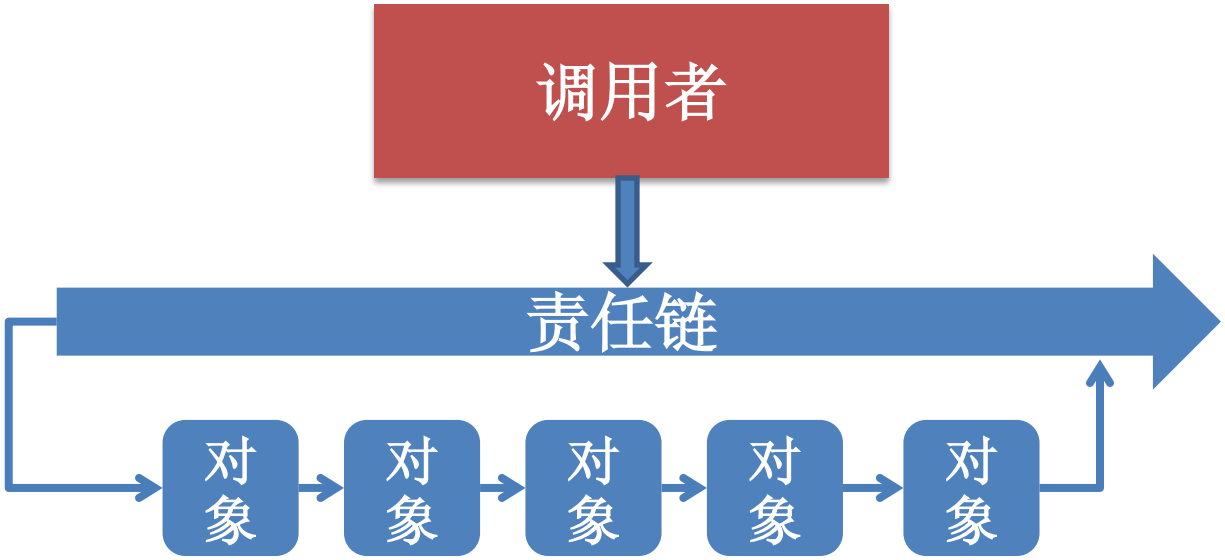
接下来：

让我们进入一些实际的模式和例子。

责任链模式

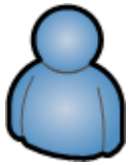
Chain of Responsibility：

为解除请求的调用者和被调用者之间耦合，而使多个对象都有机会处理这个调用请求。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它。



责任链模式

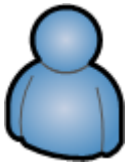
他最好什么都能做.....



调用者

职能鸿沟

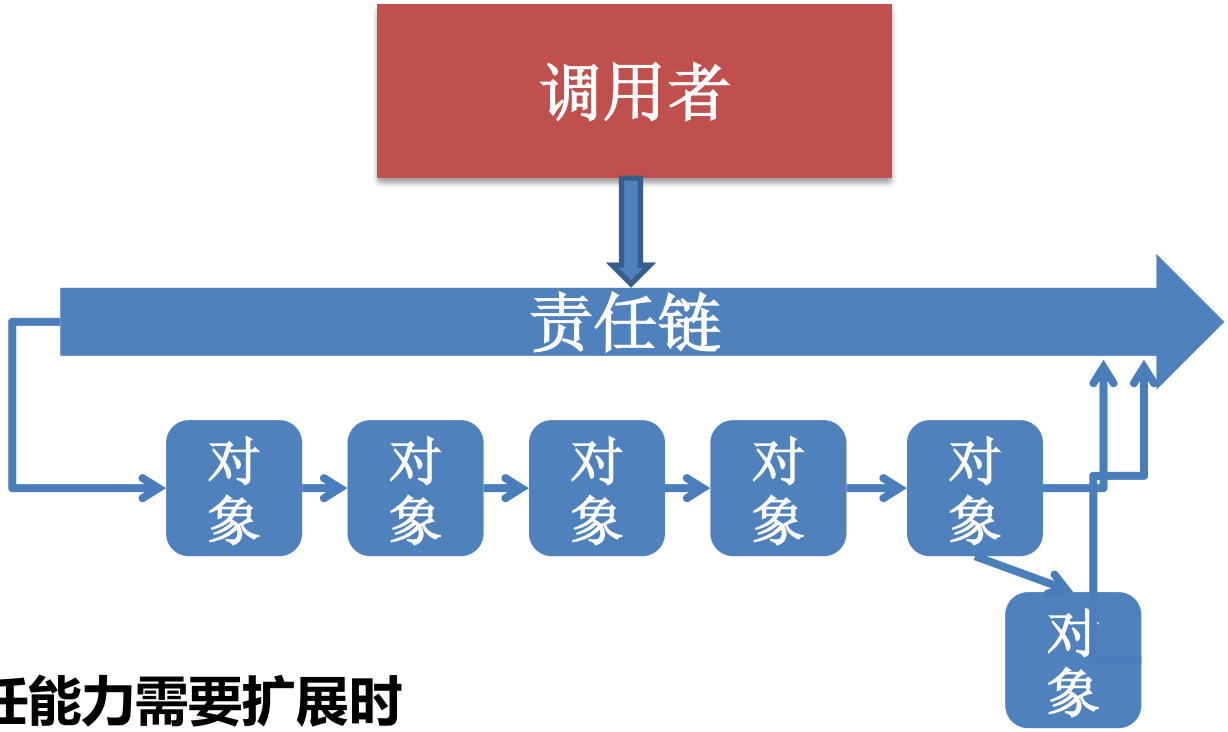
我最好只做单一的工作.....



被调用者

责任链模式

场景： 请求响应机制中，且响应需要提供扩展性。



当责任能力需要扩展时

责任链模式 – 例子

```
try{

}catch( 异常 1  ){

}catch(异常  2){

}catch(异常 3 ) {

}
```

注意：

1 有顺序。

2 标准责任链模式强调排他性。

优点:

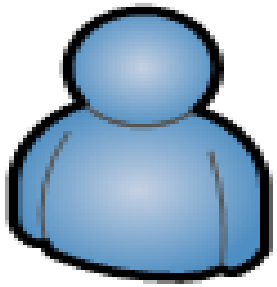
1 降低耦合度。

2 增强对象指派职责角度的灵活扩展性。

缺点：

1 不保证一定被处理。

责任链模式



哥！　　恕我直言...

你讲了半天，这个东西对我完全没有用

完全不够看.....

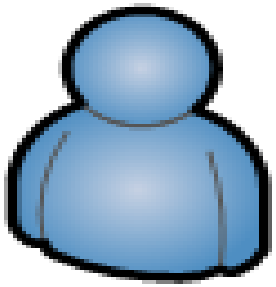
标准责任链有的时候会有点鸡肋.....

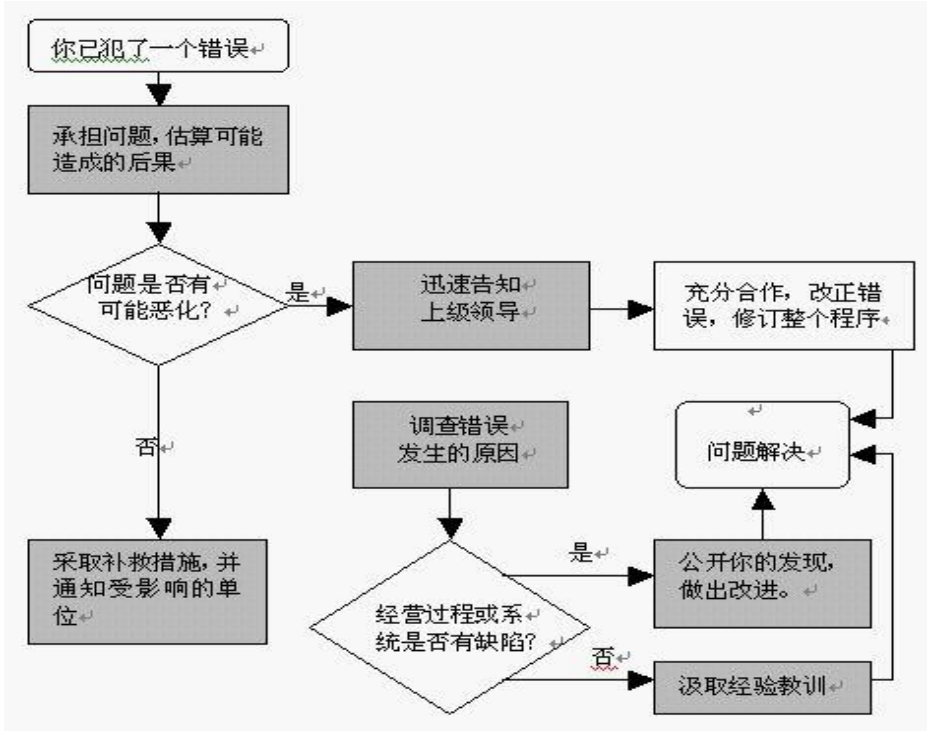
但是没有什么可以难倒勤劳的程序员.....

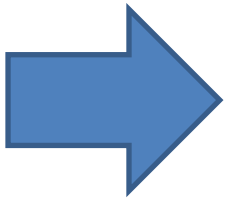
责任链模式

广义责任链

它能干点啥呢？

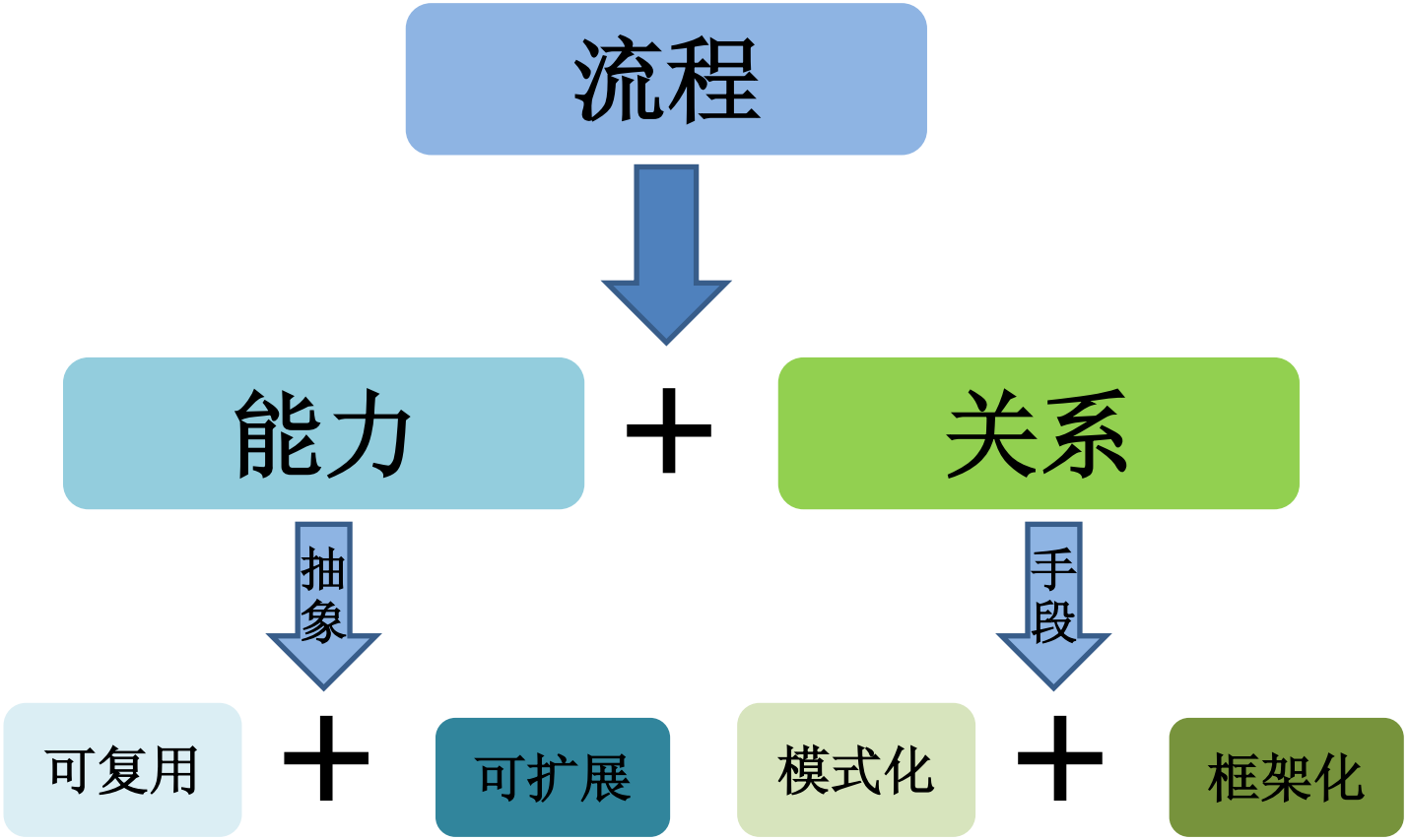






责任链

责任链模式



别急，来个例子.....

责任链模式

都有什么关系？

我是问步骤之间都有什么关系？

顺序

循环

条件

责任链模式

com.ifeng.common.plugin

我们用代码来模式化和框架化各种“步骤”之间的关系

先看看如何来实现标准责任链

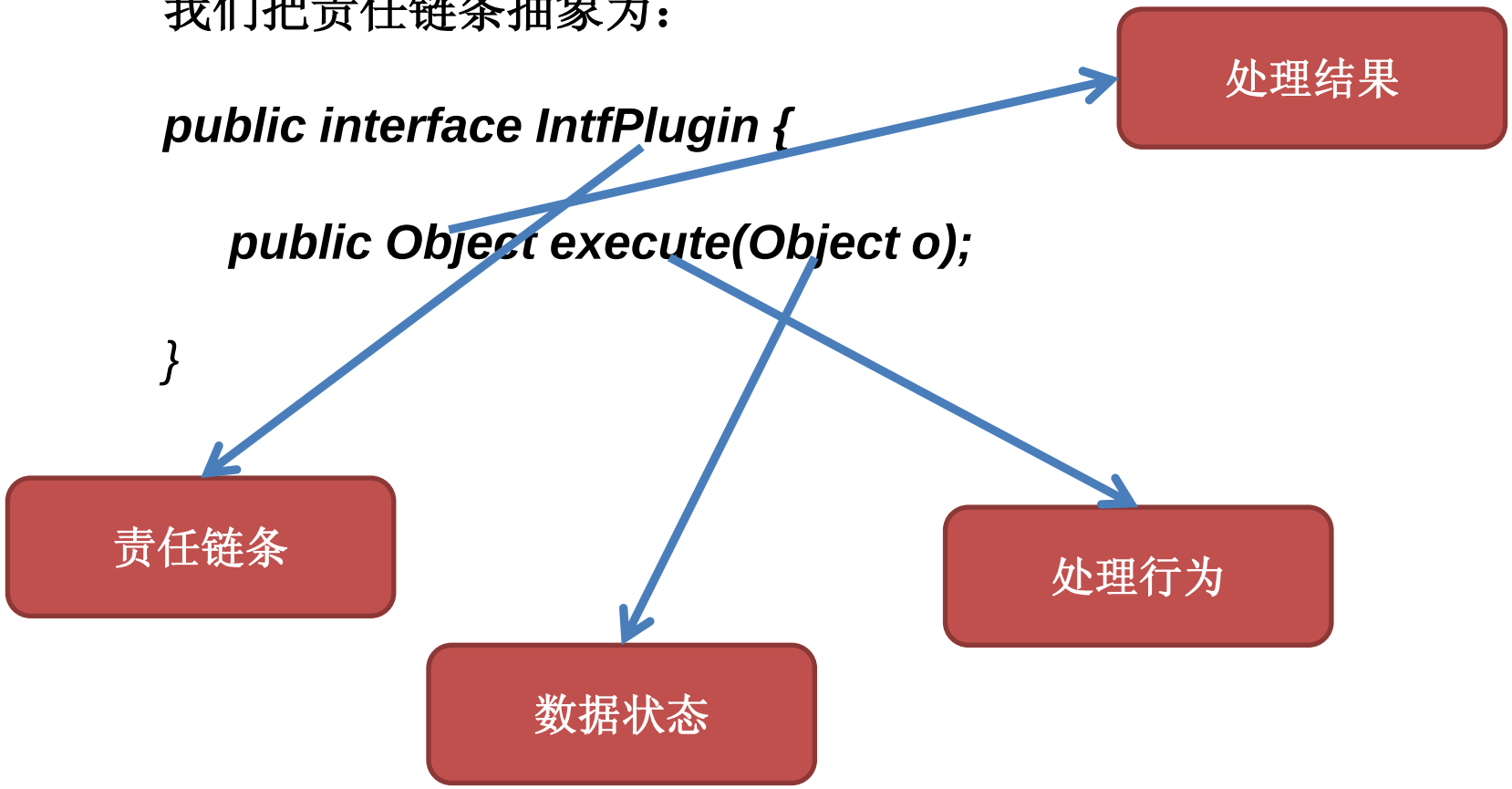
责任链模式

我们把责任链条抽象为：

```
public interface IntfPlugin {

    public Object execute(Object o);

}
```



责任链模式

标准责任链中，有成功和失败的场景

为其扩展判断能力

```
public abstract class AbstLogicPlugin implements
    IntfPlugin{

    public boolean isSuccess(Object o){
        if(o instanceof Boolean){
            return ((Boolean) o).booleanValue();
        }else{
            return o!=null;
        }
    }
}
```

责任链模式

标准责任链中，有组合能力

为其扩展责任链组合能力

参见AbstSuite.java

标准责任链的实现

其他关系的实现

例子

责任链模式 - 小结

让面向业务变化的设计过程不再那么坑爹！

只制造原子，不制造分子！

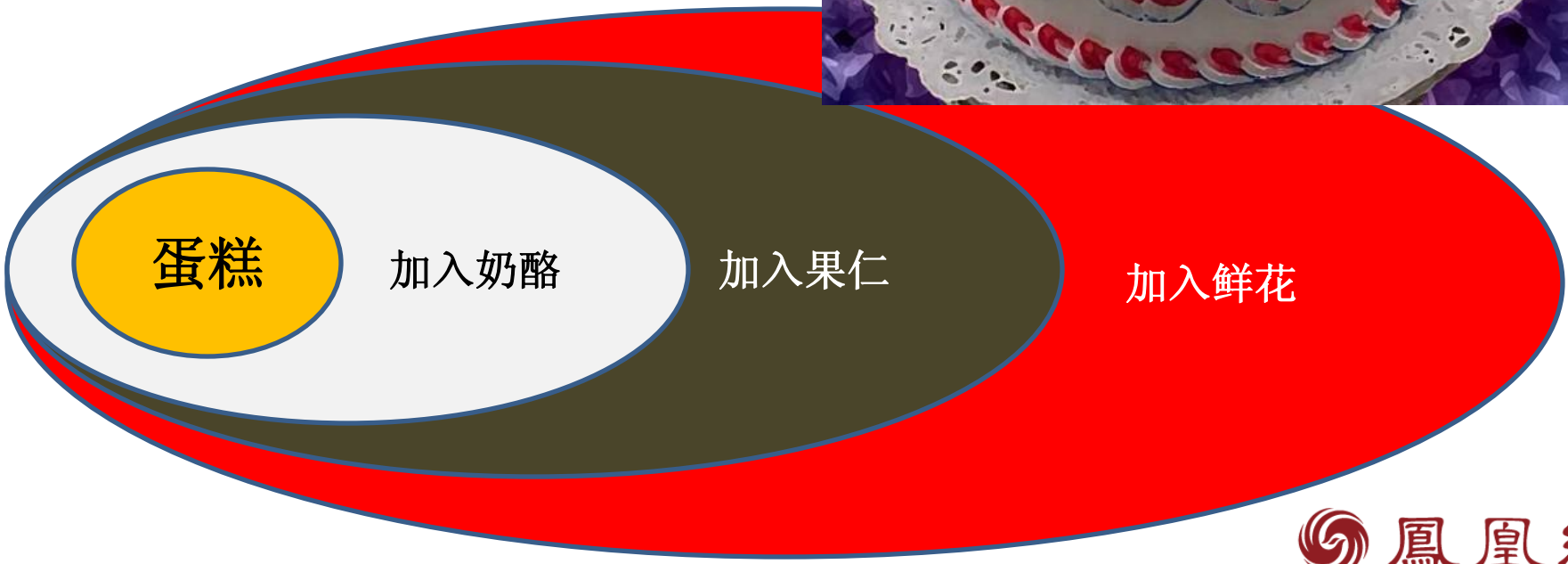
对设计边界有了准确的控制，避免设计模式的泛用。

非可变流程，不需要这么干！

放弃数据和流程的聚合能力会付出一些复杂度代价！

装饰模式

送给女朋友精美的
生日蛋糕



装饰模式

出门化个妆……

我先描个眉…

我再画个眼…

抹点粉…

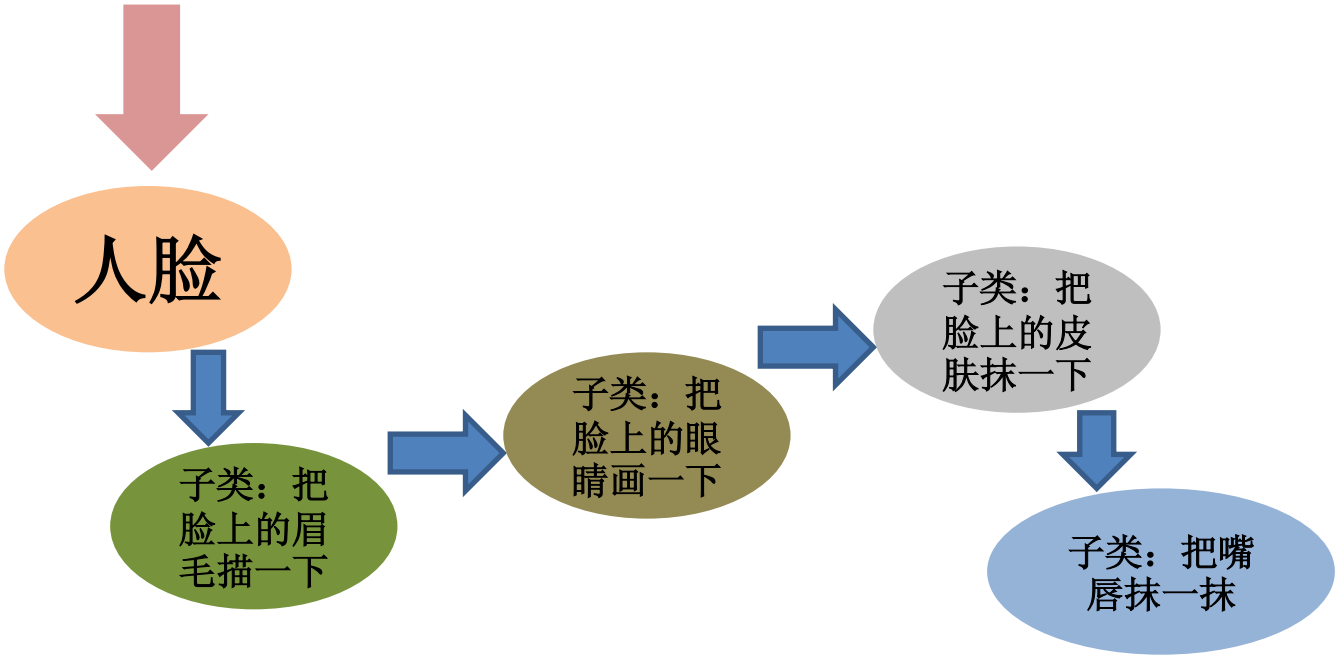
涂点唇彩…

……

大兄弟，还是我！

装饰模式

将类似这样的过程用程序来描述？



装饰模式

我的眉毛挺好的，能不能不画眉毛？

我只想弄弄睫毛，抹点防晒霜？

我不习惯你这个顺序，我平时喜欢别的顺序？

继承的局限性，影响了这样的需求的实现！

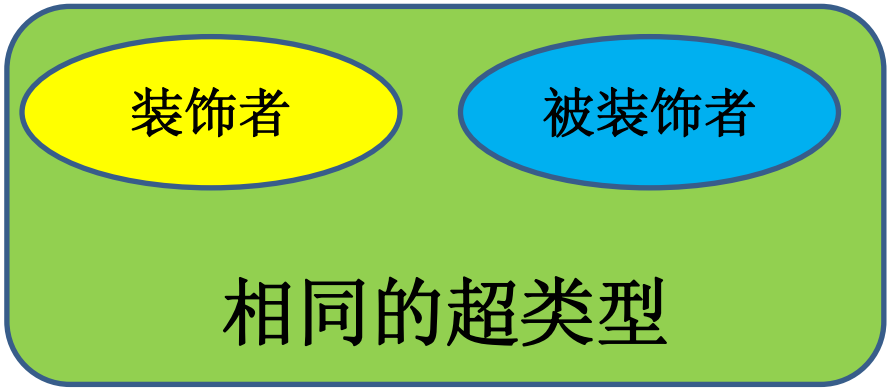
装饰模式

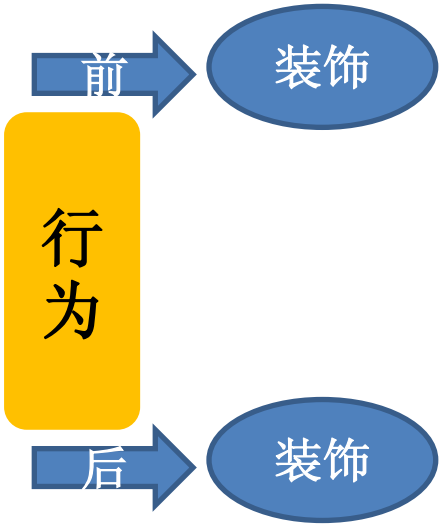
Decorator模式（别名Wrapper）

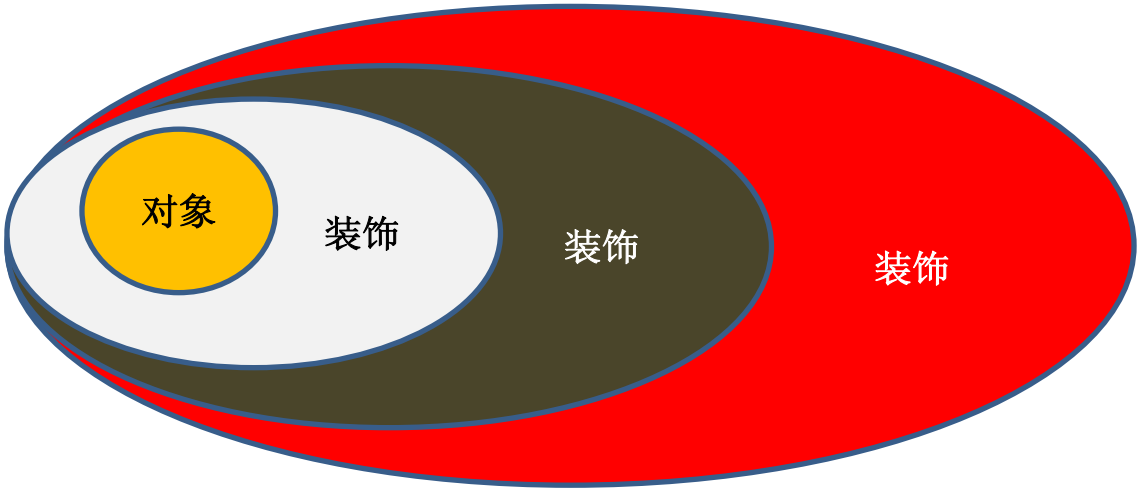
动态的将“职责”附加在对象上

代替继承 + 更加灵活 + 容易扩展 + 顺序无关

装饰模式 – 要点







装饰模式 – 要点

装饰模式的用意是保持接口并增加对象的职责。

上面的红色字体是装饰模式的关键！

装饰模式 – 例子

Struts2中利用sitemesh实现页面装饰效果

装饰页面：

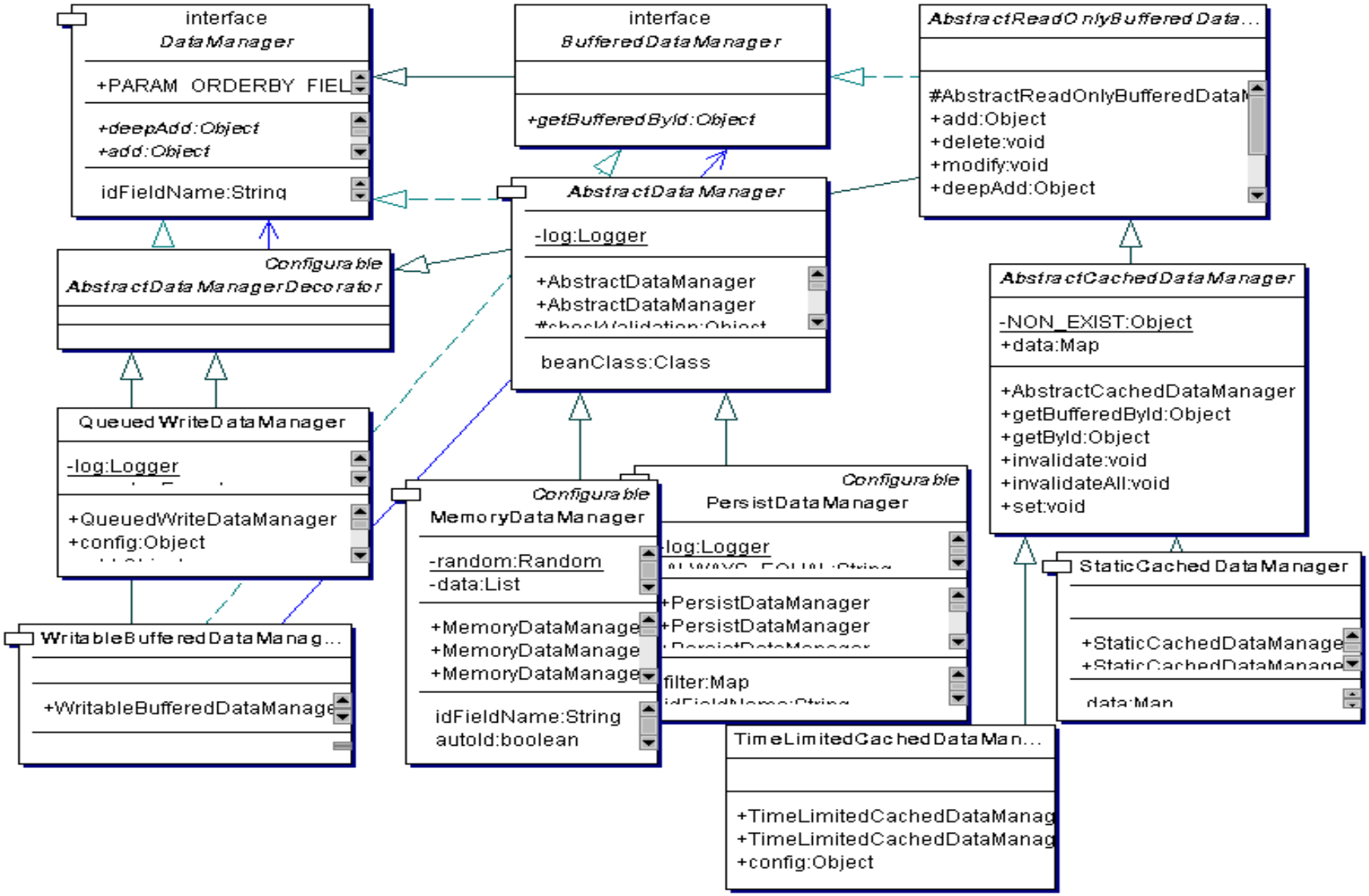
被装饰页面：

```
<%@ page contentType="text/html; charset=GBK"%>
<html>
 <head>
  <title>Agent Testtitle>
 head>  <body>
  <p>My Test Body.<p>
 </body>
</html>
```

```
  <decorator:body />
   <hr>sitemesh example:Footer
  </body>
  </html>
```

装饰模式 – 例子

icommon中 DataManager装饰模式的实现



装饰模式 – 例子

扩展一个带有数据变化通知能力的装饰。

留个作业.....

装饰模式 – 小结

装饰模式和继承都在扩展对象的能力，但是装饰模式更加灵活！

不同的装饰类和其组合关系，可以创造更多的行为组合！

装饰模式比继承更加容易出错

装饰模式 和 适配器模式

欢迎提问

Q & A