

# 设计模式:活用设计模式

疯狂代码 <http://CrazyCoder.cn/>     ĵ:<http://CrazyCoder.cn/Programing/Article49871.html>

、 设计模式隐喻

武功套路是习武门径新手要招式地练习套路烂熟于心的后熟能生巧在实战的中即可见招拆招、运用自如——此时习武的人已从“新手”成长为“好手”“高手”则没有套路实战的中只有自然反应然而招式浑然天成、恰到好处似有似无、无中生有“高手”的上还有“高高手”他们达到境界非我等凭借金氏武侠小说可以揣测

设计模式的于设计好比套路的于武术“新手”要个接个地学习模式“好手”能够活用模式“高手”则没有模式

设计模式“内功”是面向对象基本原则这些原则是“神”模式是“形”高手拼是“内功”对面向对象基本原则有了深刻领悟才能用好设计模式避免“走火入魔”

般在设计模式著作前几章都会介绍面向对象基本原则这几章非常重要学通了这几章后面模式就不过如此了学完了设计模式也最好翻过头来重新看看这几章保证会有新领悟

## 2、 为什么使用设计模式

对任何设计都可以凭主观(对设计很难做出客观评价)判断得出它是个好设计还是个坏设计使用设计模式是为了避免坏设计Martin叔叔在他著作敏捷软件Software开发 原则、模式和实战中描述了拙劣设计症状:

僵化性(Rigidity):设计难以改变

脆弱性(Fragility):设计易于遭到破坏

牢固性(Immobility):设计难以重用

粘滞性(Viscosity):难以做正确事情

不必要复杂性(Needless Complexity):过分设计

不必要重复(Needless Repetition):过多重复

晦涩性(Opacity):混乱表达

## 3、 什么时候使用设计模式

Martin叔叔书中有段话:

在学习它们(设计原则和模式)时候请记住敏捷开发人员不会对个庞大预先设计应用那些原则和模式相反这些原则和模式被应用在次次迭代中力图使代码以及代码所表达设计保持干净

在这段容易被读者忽略文字中我体会到这样几层含义:

代码是设计(这是Martin叔叔强调个观点这个观点可以参考敏捷软件Software开发 原则、模式和实战书附录D);

设计模式是为了使设计适应变化;

设计模式是重构工具;

设计开始就要保持干净、简单以后仍然要保持干净、简单;

不能过度使用设计模式

使用设计模式目的是为了适应未来变化变化的所以存在是它不可预知性——如果可以预知则不能称其为变化如何判断哪些需求可能变化哪些需求可能不变并且在最大程度上保持设计干净、简单这是些工艺问题而不是工程问题既然是工艺问题那么就只能给出原则不能给出标准使用设计模式大体原则可能是:对未来极有可能发生变化问题给出最简单、修改成本最低解

#### 4、避免过度使用设计模式

易维护首先要易理解这点远甚于其他在易理解代码上才好维护过分地使用设计模式会增加复杂性和晦涩性让不易理解从而降低了易维护性

Switch语句曾经遭致诟病许多重构例子就是拿Switch开刀我认为Switch语句是高效语句可以写出极优雅、简单代码在很多情况下直接使用Switch语句比把它拆成若干个Class更“干净”

再比如有段 4百多行代码负责整个系统调度如果未来变化仅仅是修改这 4百行代码而不会大量添加代码那么把这 4百多行代码集中在个里面比将它拆分成十来个Class更加容易维护

#### 5、讨论几个具体模式

## 1、创建模式(Creational Pattern)

工厂(Factory Method)模式是常用模式工厂模式应用情景明确设计思想简单从使用多态到只用个静态思路方法工厂模式变化形式有很多我习惯简单地使用工厂模式也就是使用只有静态思路方法工厂模式下面工厂模式代码简单、干净:

```
MyFactory.GetClassInstance.DoFunction;
```

类厂并不承载业务逻辑需求变化对类厂影响通常很小因此使用重量级工厂模式往往并不划算组包含层次关系重量级工厂类可能意味着过度设计

单例(Singleton)模式和工厂模式关系密切从实现角度讲单例模式是工厂模式个特例但是两个模式应用情景区别因此它们属于区别模式

抽象工厂(Abstract Factory)模式是工厂模式推广抽象工厂模式应用情景更加特殊和严格在个使用抽象工厂设计中如果未来发生区别产品族各自演化情形那么抽象工厂模式就可能崩溃了在实际应用中区别产品族各自演化最终分道扬镳情形是有用户提出这样需求确让人“触目惊心”在使用抽象工厂模式的前定要保证从现在到未来都能够用致方式使用这些产品族

将工厂模式稍加变化可以得到建造(Builder)模式工厂模式“加工工艺”是隐藏而建造模式“加工工艺”是暴露这点区别使建造模式在更加灵活同时也有失优雅

## 2、模板模式(Template Method)和策略(Strategy)模式

模板模式和策略模式应用情景类似但实现方式区别前者使用继承后者使用委托

模板模式有可能是最“古老”模式的在使用面向对象技术早期“继承”大行其道很多设计人员可能不自觉地使用过模板模式模板模式缺点是把具体实现和通用算法紧密地耦合起来使得具体实现只能被个通用算法操纵然而在继承关系中父类信息可以更多地暴露给子类这种(违背面向对象设计原则)微妙沟通在些特定应用中显得更加灵活和方便

策略模式是委托经典使用方法策略模式消除了通用算法和具体实现耦合使得具体实现可以被多个通用算法操纵策略模式也增加了类层次比模板模式复杂

模板模式和策略模式通常可以互相替换它们都像试卷模板模式是填空题策略模式是选择题

### 3、 简化问题模式

门面(Facade)模式把组复杂接口隐藏在个简单且特定接口后面

调停者(Mediator)模式把对象的间引用关系包装在个特定容器里面

组合(Composite)模式描述了整体和部分结构关系并且允许用致方式处理这个结构

上面几个模式对使用者而言都在定程度上起到了简化问题作用

### 4、 扩展功能模式

访问者(Visitor)模式和装饰(Decorator)模式都可以在不改变现有类结构基础上动态地增加功能

访问者模式把现有类结构上对象“分配”到个名为访问者类中在访问者相应思路方法中配置对象、改变对象或扩展功能

装饰模式把现有类结构上对象“注入”个装饰类中在装饰类中扩展它功能

访问者模式和装饰模式在实际效果上是区别访问者模式可以把对象分配到相应思路方法里从而对每个对象分别进行加工或扩展而装饰模式只能用致方式对所有被装饰对象进行加工或扩展要想实现区别加工或扩展只能增加新装饰类

过多“装饰类”有可能使业务逻辑分散并且使结构复杂针对每个具体派生类“访问类”都要有个对应思路方法增加派生类时候也要增加访问类思路方法扩展功能需求是经常发生是否有必要使用上述模式则值得再 3考虑

### 5、 其他常用模式

桥梁(Bridge)模式Class是封装了行为和属性容器然而Class组行为可能独立演化这时最直接想法是使用继承把各不相同行为封装在区别子类里桥梁模式从另外角度解决了这个问题桥梁模式把独立演化行为封装在另外个类体系里和原来类体系分别独立演化两个类体系在抽象层次是“使用”关系在很多OO教材里面用Shape类封装属性和Draw思路方法在桥梁模式里“形状”和“画笔”是两组独立演化类体系在抽象层次“形状”使用“画笔”绘制自己

适配器(Adapter)模式是常用模式它比较简单有时和其他模式配合使用

命令(Command)模式被Martin称为“最简单、最优雅模式的”命令模式魅力在于它为每个类“培训”出了相同技能经过“培训”类“柔性”更强能够产生不可思议能力

6、不太需要模式

观察者(Observer)模式Java和C# 都实现了观察者模式

迭代子(Iterator)模式在Java和C#语言里可以用聚集类代替

备忘录(Memento)模式可以用Class序列化能力代替

责任链(Chain of Responsibility)模式可以用其他方式替代例如观察者模式、语言本身提供消息机制等

解释器(Interpreter)模式个人认为它是个算法不是模式

代理(Proxy)模式如果在开始就知道某些底层策略定会被替换掉那么使用代理来隔离这些策略还是有必要否则几乎没有使用必要

2009-1-9 15:49:08

疯狂代码 <http://CrazyCoder.cn/>