课程目标

1、简要分析GoF23种设计模式和设计原则，做整体认知。
2、剖析Spirng的编程思想，启发思维，为之后深入学习Spring 做铺垫。
3、了解各设计模式之间的关联，解决设计模式混淆的问题。

内容定位

1、掌握设计模式的“道”，而不只是“术”。
2、道可道非常道，滴水石穿非一日之功，做好长期修炼的准备。

3、不要为了用设计模式去生搬硬套，而是在业务上到遇到问题时，很自然地想到设计模式作为一种解决方案。

1. 一句话归纳设计模式

设计原则	一句话归纳	目的
开闭原则(OCP) (Open-Close)	对扩展开放，对修改关闭	减少维护带来新的风险
依赖倒置原则(DIP) (Dependence Inversion)	高层不应该低层	更利于代码结构的升级 扩展
单一职责原则(SRP) (Simple Responsibility)	一个类只干一件事	便于理解，提高代码可读性
接口隔离原则(ISP) (Interface Segregation)	一个接口只干一件事	功能解耦，高聚合、低耦合
迪米特法则(LoD) (Law of Demeter)	不该知道的不要知道	只和朋友交流，不和陌生人说话，减少代码臃肿
里氏替换原则(LSP) (Liskov Substitution)	子类重写方法功能发生改变， 不应该影响父类方法的含义	防止继承泛滥
合成复用原则(CARP) (Composite/Aggregate Reuse)	尽量使用组合实现代码复用， 而不使用继承	降低代码耦合

2. GOF23种设计模式简介

《Design Patterns:Elements of Reusable Object-Oriented Software》（即后述《设计模式》一书），由Erich Gamma、Richard Helm、Ralph Johnson和John Vlissides合著（Addison-Wesley，1995）。这几位作者常被称为“四人组（Gang ofFour）"，而这本书也就被称为“四人组（或GoF）"书。

在《设计模式》这本书的最大部分是一个目录，该目录列举并描述了23种设计模式。另外，近来这一清单又增加了一些类别，最重要的是使涵盖范围扩展到更具体的问题类型。例如，Mark Grand在Patterns in Java:A Catalog of Reusable Design Patterns Illustrated with UML（即后述《模式Java版》一书）中增加了解决涉及诸如并发等问题的模式，而由 Deepak Alur、John Crupi 和Dan Malks合著的 CoreJ2EE Patterns:Best Practices and Design Strategies 一书中主要关注使用Java2企业技术的多层应用程序上的模式。

很多人并没有注意到这点，学完Java 基础语言就直接去学J2EE，有的甚至鸭子赶架，直接使用起Weblogic等具体2EE软件，一段时间下来，发现不过如此，挺简单好用，但是你真正理解J2EE了吗？

你在具体案例中的应用是否也是在延伸J2EE的思想？对软件设计模式的研究造就了一本可能是面向对象设计方面最有影响的书籍：《设计模式》。

由此可见，设计模式和J2EE在思想和动机上是一脉相承的，我总结了以下几个原因：
1.设计模式更抽象，J2EE是具体的产品代码，我们可以接触到，而设计模式在对每个应用时才会产生

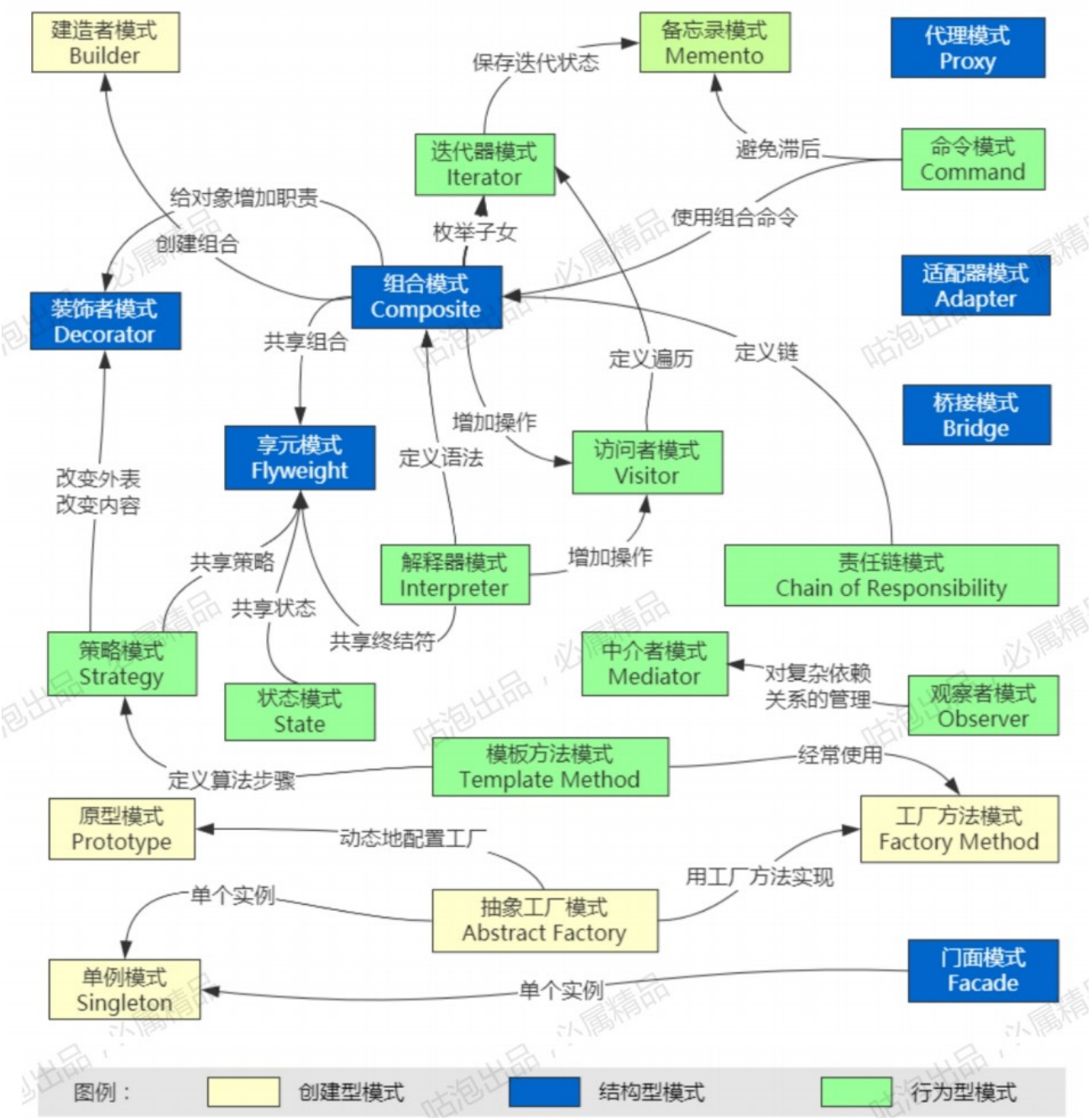
具体代码。
2.设计模式是比J2EE等框架软件更小的体系结构，J2EE中许多具体程序都是应用设计模式来完成的，当你深入到J2EE的内部代码研究时，这点尤其明显，因此，如果你不具备设计模式的基础知识（GoF的设计模式），你很难快速的理解J2EE。不能理解J2EE，如何能灵活应用？
3.J2EE只是适合企业计算应用的框架软件，但是GoF的设计模式几乎可以用于任何应用！因此GoF的设计模式应该是J2EE的重要理论基础之一。

所以说，GoF的设计模式是Java基础知识和J2EE框架知识之间一座隐性的“桥”。

设计模式其实也是一门艺术。设计模式来源于生活，不要为了套用设计模式而去使用设计模式。设计模式是在我们迷茫时提供的一种解决问题的方案，或者说用好设计模式可以防范于未然。

设计模式总结的是经验之谈，总结的是前人的经验，提供给后人去借鉴使用，前人栽树，后人乘凉。设计模式可以帮助我们提升代码的可读性、可扩展性；降低维护成本；解决复杂的业务问题，但是，千万千万不要死记硬背，生搬硬套。下面我们还是先来总体预览一下GOF23种设计模式的归纳和总结。

分类	设计模式
创建型	工厂方法模式（Factory Method）、抽象工厂模式（Abstract Factory）、单例模式(Singleton)、原型模式（Prototype）、建造者模式（Builder）
结构型	代理模式（Proxy）、门面模式（Facade）、装饰器模式（Decorator）、享元模式（Flyweight）、组合模式（Composite）、适配器模式(Adapter)、桥接模式（Bridge）
行为型	模板方法模式（Template Method）、策略模式（Strategy）、 责任链模式（Chain of Responsibility）、迭代器模式（Iterator）、命令模式（Command）、 状态模式（State）、备忘录模式（Memento）、中介者模式（Mediator）、 解释器模式（Interpreter）、观察者模式（Observer）、访问者模式（Visitor）



3. 设计模式使用频次总结

3.1. 创建型模式（Creational）

3.1.1. 高频：

工厂方法模式（Factory Method）、抽象工厂模式（Abstract Factory）、单例模式（Singleton）、建造者模式（Builder）

3.1.2. 低频：

原型模式（Prototype）

3.2. 结构型模式（Structural）

3.2.1. 高频：

代理模式（Proxy）、门面模式（Facade）、
装饰器模式（Decorator）、享元模式（Flyweight）、
适配器模式（Adapter）、组合模式（Composite）

3.2.2. 低频：

桥接模式（Bridge）

3.3. 行为型模式（Behavioral）

3.3.1. 高频：

模板方法模式（Template Method）、策略模式（Strategy）、
责任链模式（Chain of Responsibility）、状态模式（State）

3.3.2. 低频：

备忘录模式（Memento）、
观察者模式（Observer）、迭代器模式（Iterator）、
中介者模式（Mediator）、命令模式（Command）、
解释器模式（Interpreter）、访问者模式（Visitor）

4. 一句话归纳设计模式

设计模式	一句话归纳	目的	生活案例	框架源码举例
工厂模式（Factory）	产品标准化，生产更高效	封装创建细节	实体工厂	LoggerFactory、Calender
单例模式（Singleton）	世上只有一个 Tom	保证独一无二	CEO	BeanFactory、Runtime
原型模式（Prototype）	拔一根猴毛，吹出千万个	高效创建对象	克隆	ArrayList、PrototypeBean
建造者模式（Builder）	高配中配与低配， 想选哪配就哪配	开放个性配置步骤	选配	StringBuilder、BeanDefinitionBuilder
代理模式（Proxy）	没有资源没时间， 得找媒婆来帮忙	增强职责	媒婆	ProxyFactoryBean、JdkDynamicAopProx CglibAopProxy
门面模式(Facade)	打开一扇门，走向全世界	统一访问入口	前台	JdbcUtils、RequestFacade
装饰器模式（Decorator）	他大舅他二舅，都是他舅	灵活扩展、同宗同源	煎饼	BufferedReader、InputStream
享元模式(Flyweight)	优化资源配置， 减少重复浪费	共享资源池	全国社保联网	String、Integer、ObjectPool
组合模式(Composite)	人在一起叫团伙， 心在一起叫团队	统一整体和个体	组织架构树	HashMap、SqlNode
适配器模式（Adapter）	适合自己的，才是最好的	兼容转换	电源适配	AdvisorAdapter、HandlerAdapter
桥接模式(Bridge)	约定优于配置	不允许用继承	桥	DriverManager
委派模式（Delegate）	这个需求很简单， 怎么实现我不管	只对结果负责	授权委托书	ClassLoader、BeanDefinitionParserDeleg
模板模式（Template）	流程全部标准化， 需要微调请覆盖	逻辑复用	把大象装进	JdbcTemplate、HttpServlet
策略模式（Strategy）	条条大道通北京， 具体哪条你来定	把选择权交给用户	选择支付方式	Comparator、InstantiationStrategy
责任链模式(Chain of Responsibility)	各人自扫门前雪， 莫管他人瓦上霜	解耦处理逻辑	踢皮球	FilterChain、Pipeline
迭代器模式(Iterator)	流水线上坐一天，每个包裹扫一遍	统一对集合的访问方式	逐个检票进站	Iterator
命令模式(Command)	运筹帷幄之中， 决胜千里之外	解耦请求和处理	遥控器	Runnable、TestCase
状态模式(State)	状态驱动行为， 行为决定状态	绑定状态和行为	订单状态跟踪	Lifecycle
备忘录(Memento)	给我一剂“后悔药”	备份	草稿箱	StateManageableMessageContext
中介者(Mediator)	联系方式我给你， 怎么搞定我不管	统一管理网状资源	朋友圈	Timer
解释器模式（Interpreter）	我想说“方言”， 一切解释权归我所有	实现特定语法解析	摩斯密码	Pattern、ExpressionParser
观察者模式（Observer）	到点就通知我	解耦观察者与被观察者	闹钟	ContextLoaderListener
访问者模式（Visitor）	横看成岭侧成峰， 远近高低各不同	解耦数据结构和数据操作	KPI 考核	FileVisitor、BeanDefinitionVisitor

5. 设计模式之间的关联关系和对比

5.1. 单例模式和工厂模式

实际业务代码中，通常会把工厂类设计为单例。

5.2. 策略模式和工厂模式

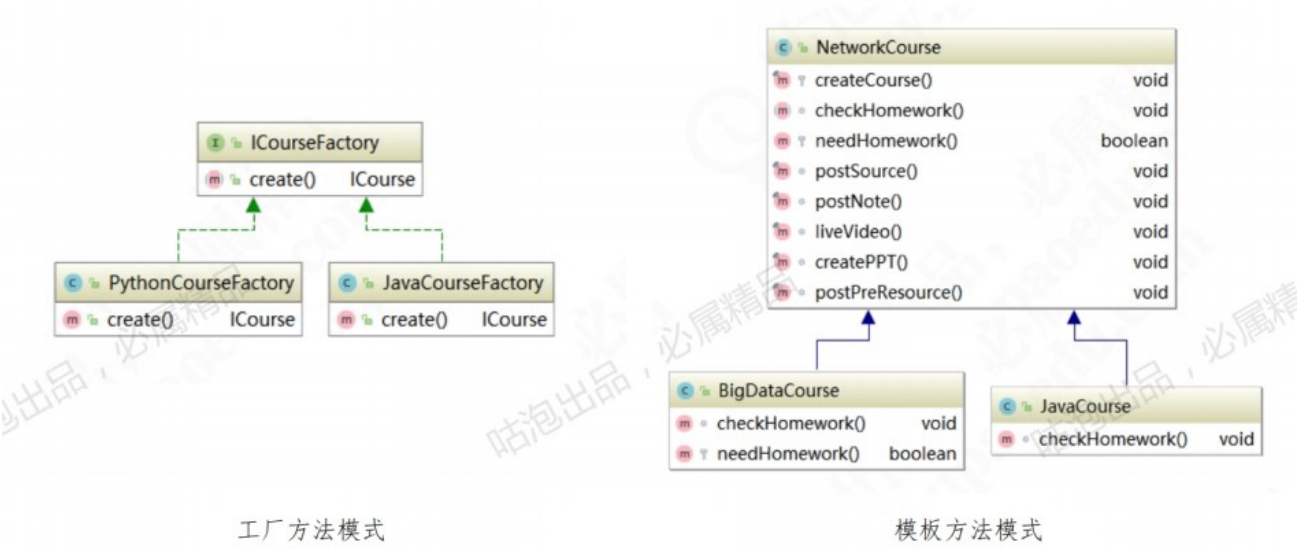
1、工厂模式包含工厂方法模式和抽象工厂模式是创建型模式，策略模式属于行为型模式。
2、工厂模式主要目的是封装好创建逻辑，策略模式接收工厂创建好的对象，从而实现不同的行为。

5.3. 策略模式和委派模式

1、策略模式是委派模式内部的一种实现形式，策略模式关注的结果是否能相互替代。
2、委派模式更关注分发和调度的过程。

5.4. 模板方法模式和工厂方法模式

工厂方法是模板方法的一种特殊实现。

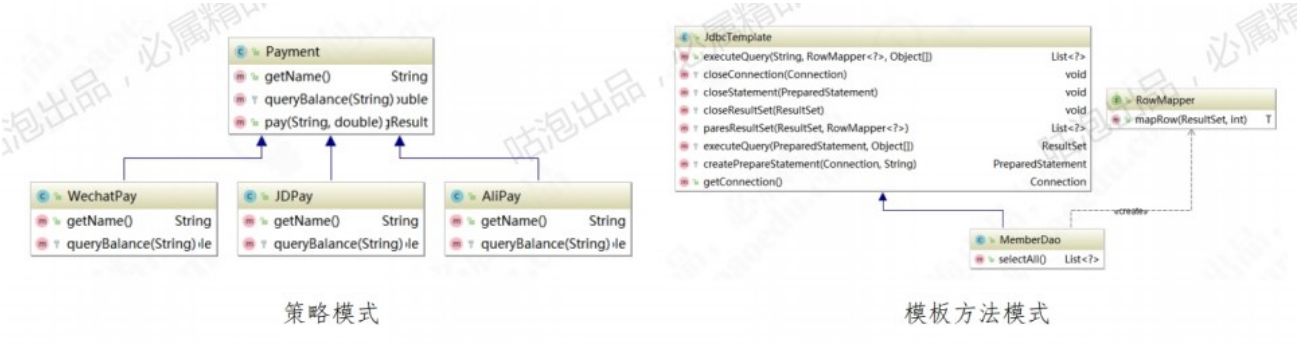


工厂方法模式　　模板方法模式

对于工厂方法模式的create()方法而言，相当于只有一个步骤的模板方法模式。这一个步骤交给子类去实现。而模板方法呢，将needHomework()方法和checkHomework()方法交给子类实现，needHomework()方法和checkHomework()方法又属于父类的某一个步骤且不可变更。

5.5. 模板方法模式和策略模式

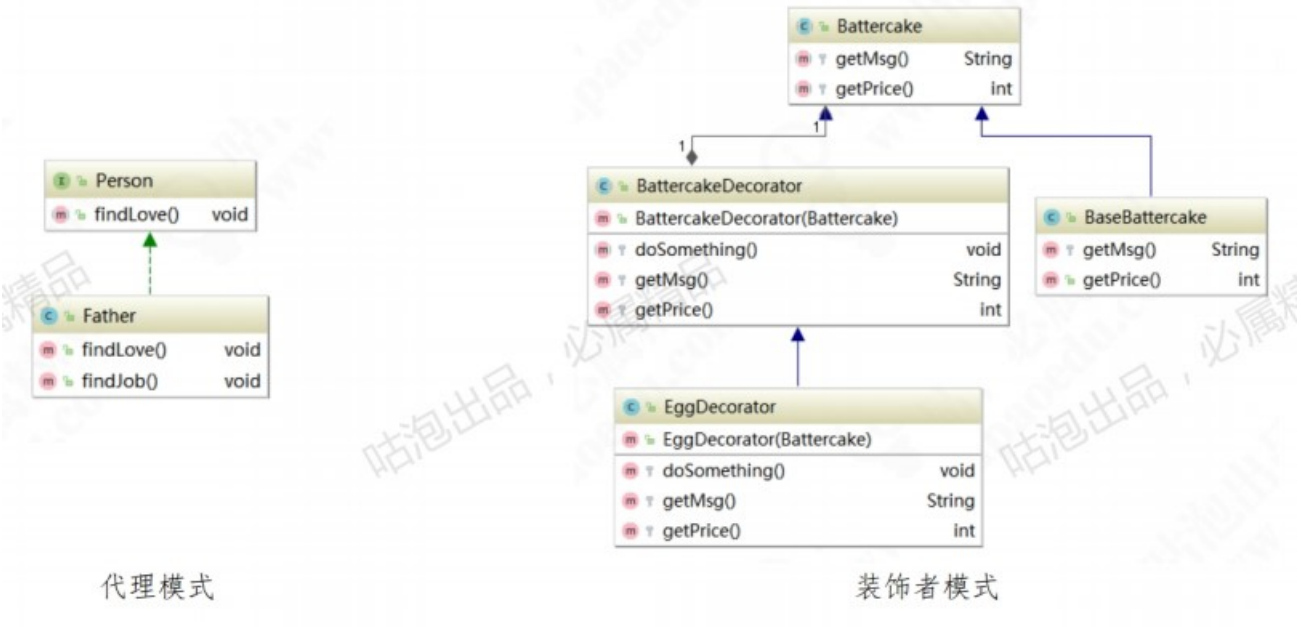
1、模板方法和策略模式都有封装算法。
2、策略模式是使不同算法可以相互替换，且不影响客户端应用层的使用。
3、模板方法是针对定义一个算法的流程，将一些有细微差异的部分交给子类实现。
4、模板方法模式不能改变算法流程，策略模式可以改变算法流程且可替换。策略模式通常用来代替if..else..等条件分支语句。



1、WechatPay、JDPay、AliPay是交给用户选择且相互替代解决方案。而JdbcTemplate 下面的子类是不能相互代替的。
2、策略模式中的 queryBalance()方法虽然在pay()方法中也有调用，但是这个逻辑只是出于程序健壮性考虑。用户完全可以自主调用queryBalance()方法。而模板方法模式中的mapRow()方法一定要在获得 ResultSet之后方可调用，否则没有意义。

5.6. 装饰者模式和静态代理模式

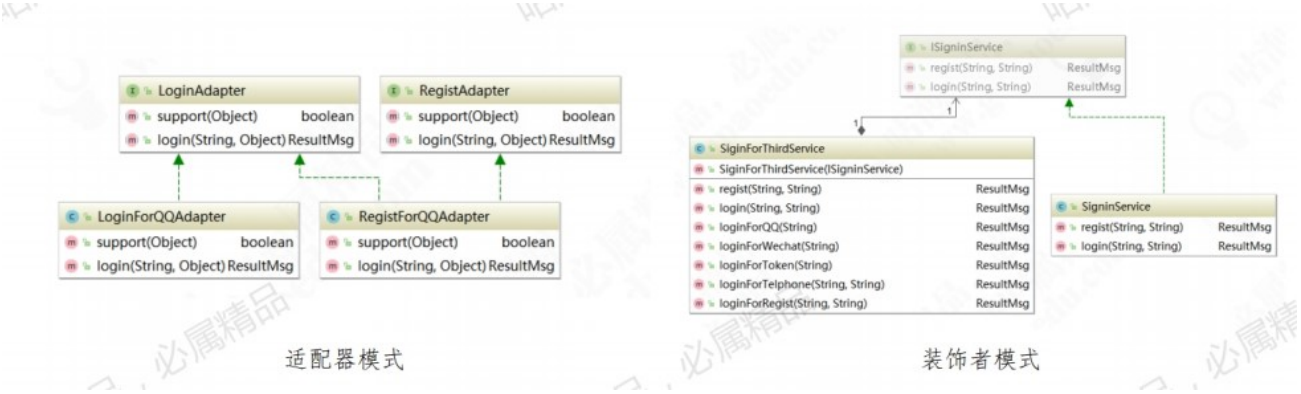
1、装饰者模式关注点在于给对象动态添加方法，而代理更加注重控制对对象的访问。
2、代理模式通常会在代理类中创建被代理对象的实例，而装饰者模式通常把被装饰者作为构造参数。



装饰者和代理者虽然都持有对方引用，但逻辑处理重心是不一样的。

5.7. 装饰者模式和适配器模式

1、装饰者模式和适配器模式都是属于包装器模式（Wrapper Pattern）。
2、装饰者模式可以实现被装饰者与相同的接口或者继承被装饰者作为它的子类，而适配器和被适配者可以实现不同的接口。



装饰者和适配器都是对SiginService的包装和扩展，属于装饰器模式的实现形式。但是装饰者需要满足OOP的is-a关系，我们也讲过煎饼的例子，不管如何包装都有共同的父类。而适配器主要解决兼容问题，不一定要统一父类，上图中LoginAdapter 和RegistAdapter就是兼容不同功能的两个类，但RegistForQQAdapter 需要注册后自动登录，因此既继承了RegistAdpter又继承了LoginAdapter。

5.8. 适配器模式和静态代理模式

适配器可以结合静态代理来实现，保存被适配对象的引用，但不是唯一的实现方式。

5.9. 适配器模式和策略模式

在适配业务复杂的情况下，利用策略模式优化动态适配逻辑。