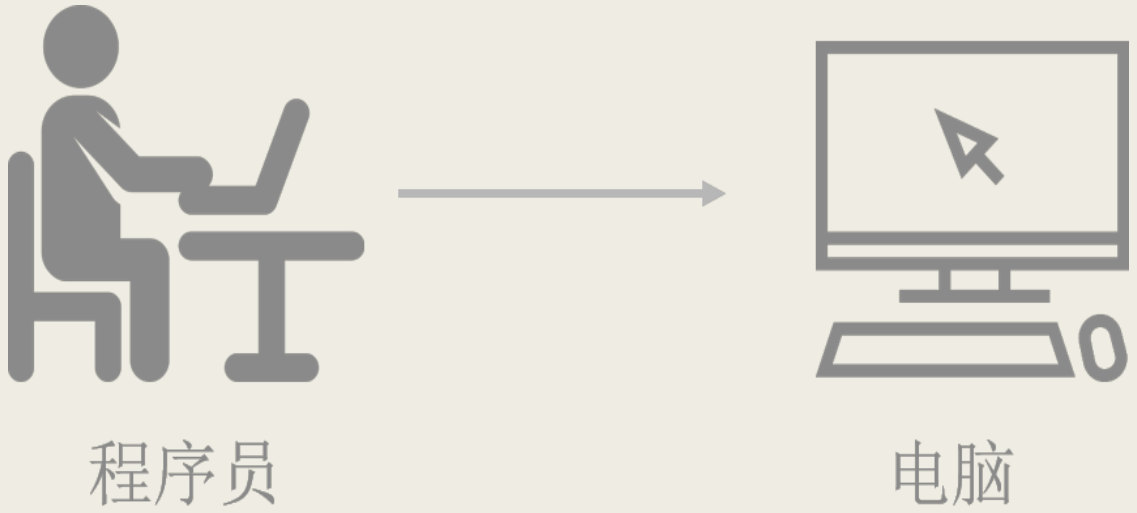
IOC与DI

- 一种设计模式
- 相关术语：
  - *依赖（Depandency）*
  - *依赖倒置原则(Dependency inversion principle)*
  - *控制反转（Inversion of Control）*
  - *依赖注入（Dependency Injection）*
- 在软件开发框架中普遍存在

依赖（Dependency）



依赖倒置原则DIP(Dependency inversion principle)

- 高层模块不应该依赖于低层模块。它们都应该依赖抽象。
- 抽象不应该依赖于细节，细节应该依赖于抽象。

设计模式的六大原则（SOLID）

- *Single Responsibility Principle：单一职责原则*
- *Open Closed Principle：开闭原则*
- *Liskov Substitution Principle：里氏替换原则*
- *Law of Demeter：迪米特法则*
- *Interface Segregation Principle：接口隔离原则*
- ***Dependence Inversion Principle：依赖倒置原则***

DIP概念在java中

- 高层模块—调用者
- 低层模块-被调用者
- 抽象-接口或抽象类
- 细节-具体实现

打破了正常的上层依赖下层的关系—倒置了

依赖倒置实质上是面向接口编程的体现

IoC-依赖倒置原则的一种具体实现方案

- 控制反转 （Inversion of Control）
- DIP是一种 软件设计原则，如何实现呢?
- IoC是一种软件设计模式，可以实现DIP

控制反转原理

将被依赖对象（低层模块）的生产交给第三者（IoC容器）来控制，而不是由高层模块来控制生成---反转了控制方式

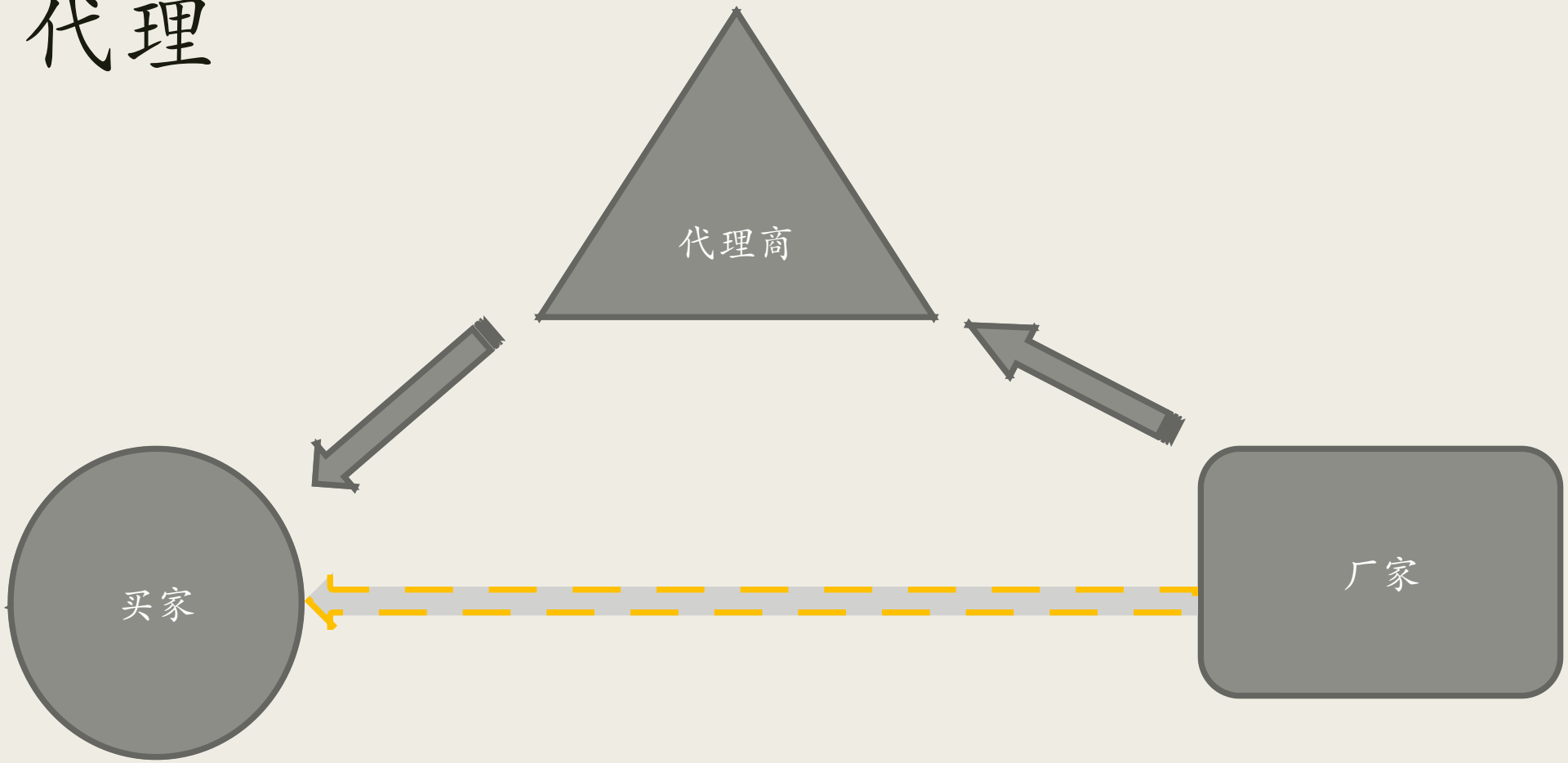
IoC模式的具体解决方案

- 依赖注入DI（Dependency Injection）：所依赖的对象由外部IoC容器注入进去
- DI的方法：
  - 1. 构造函数中注入
  - 2. setter 方式注入
  - 3. 接口注入
- 控制反转IoC和依赖注入DI的关系
  - DI ⊑ IoC
  - *IoC=DI*

IoC的意义

- **面向对象设计法则之一——好莱坞法则：“别找我们，我们找你”**
- **由IoC容器帮对象找相应的依赖对象并注入，而不是由对象主动去找。**
- 编程思想中各模块的主从关系的一次颠覆
  - ***老套路：应用程序是老大，要获取什么资源都是主动出击，编写代码主动调用相关的类库***
  - ***新思想：应用程序就变成被动的了，被动的等待IoC容器来创建并注入它所需要的资源***
- 其应用的集大成者：Spring

代理



目标：获得一个可以全程包办，任意灵活定制的中间人

代理模式（Proxy Pattern）

- 设计模式的一种，又称委托模式

- Java代理模式的实现：
  - *静态代理*
  - *动态代理*

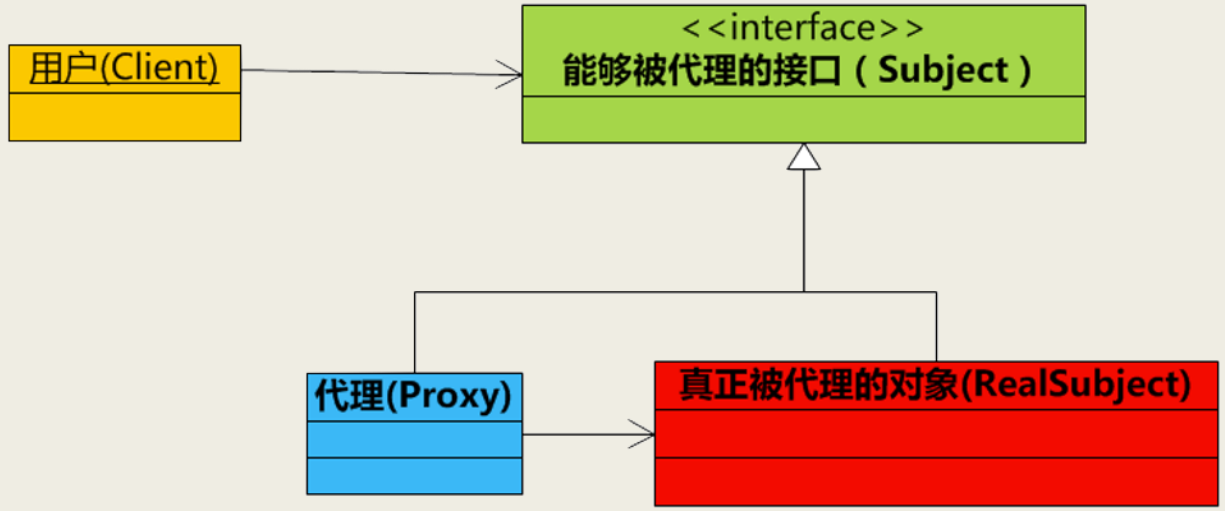
一种面向接口的间接访问对象的编程思想

代理模式的目的

- 灵活扩展原有功能模块
  - 对功能进行增强或修改
  - 在原有方法的前后进行前置处理和后置处理
- 降低代码模块之间的耦合度

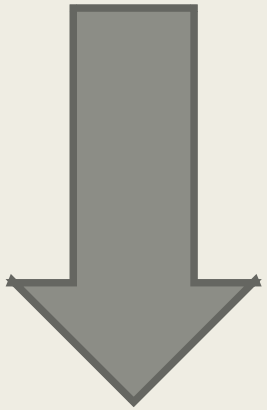
静态代理java实现

- 所有调用目标对象的方法，都调用代理对象的方法
- 对每个方法，需要静态编码
- 理解简单，但代码繁琐



静态代理

如何实现代理任意一个类，任意一个方法?



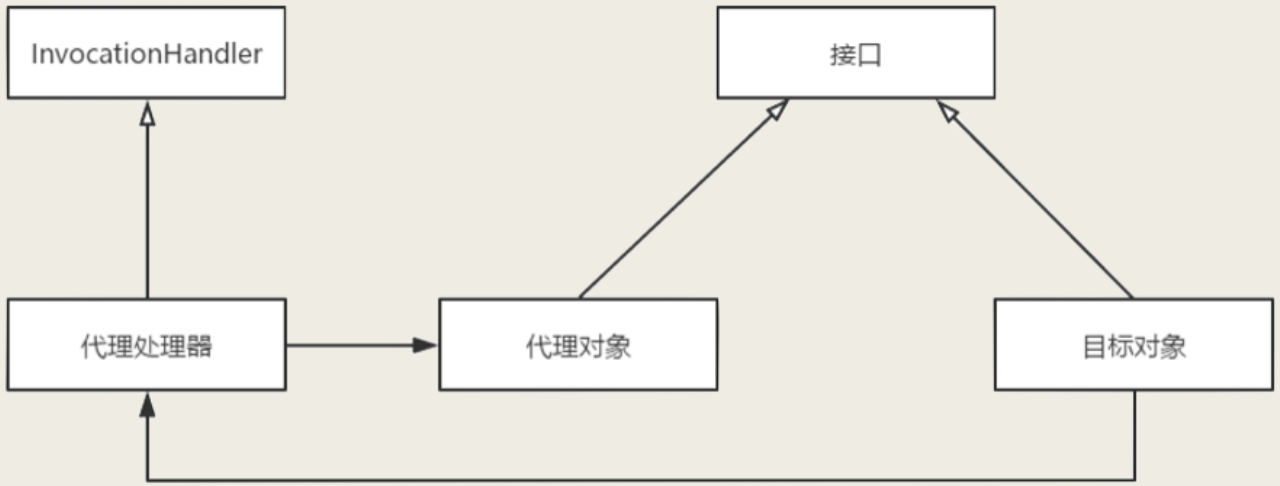
动态代理

Java动态代理实现关键

- 两个关键问题：
  - *代理对象的生成*
  - *代理对象的相关方法的执行*

- JDK中的 Proxy类的静态方法newProxyInstance生成代理对象
- Handler 接口 代理对象方法的调用，会自动转发到InvocationHandler接口的invoke方法

动态代理java实现



- 逻辑结构理解相对比较复杂，但代码简洁灵活，易于扩充

Jdk 动态代理实现步骤

- 编写Handler类实现InvocationHandler接口
- 创建InvocationHandler对象
- 使用Proxy.newProxyInstance创建代理对象
- 执行代理对象的相关方法，

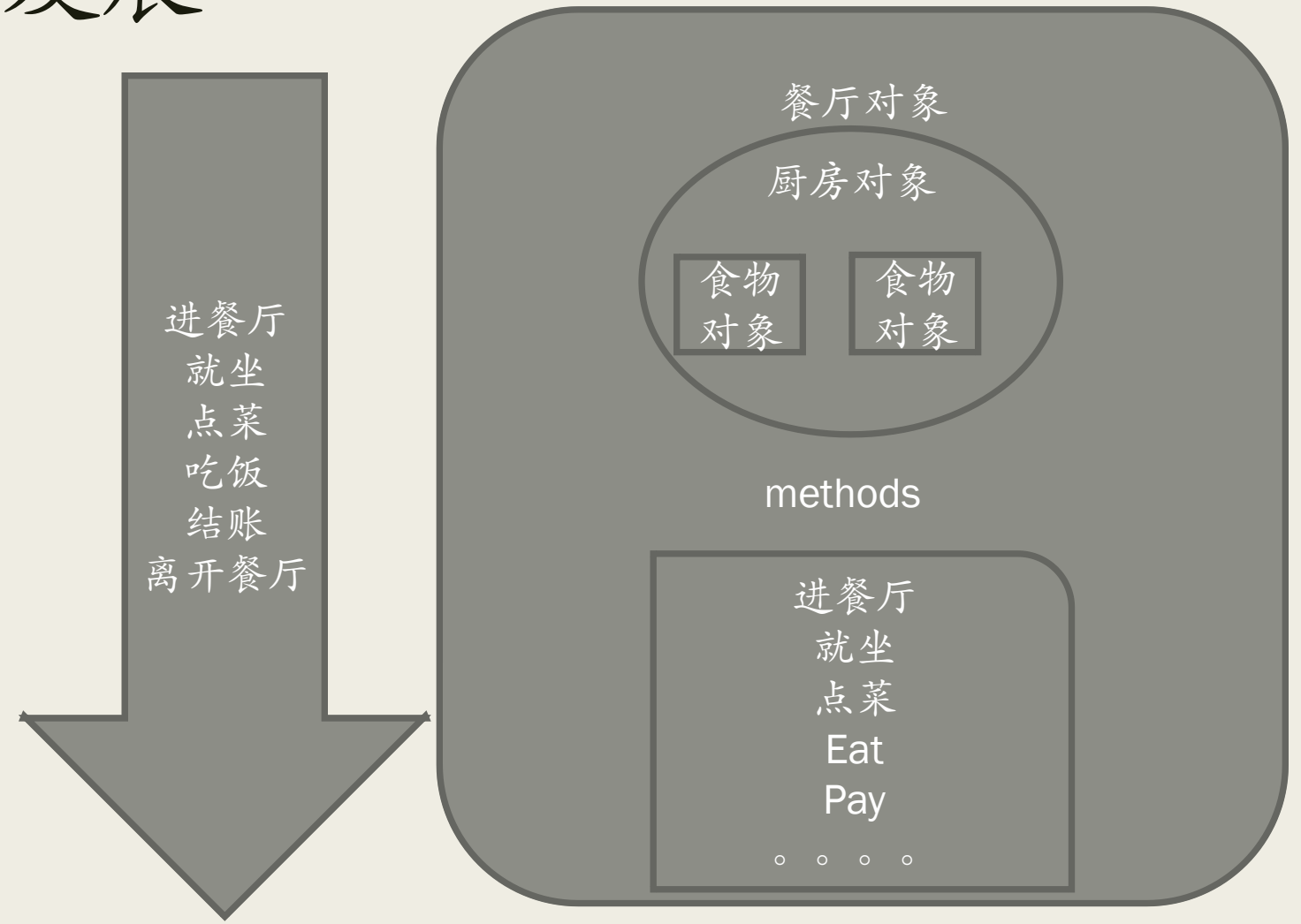
动态代理模式的应用

- 常用与添加监控、审查处理等功能扩展
- 分离代码的耦合(高内聚，低耦合)

- 应用集大成者：
  - *AOP（Aspect Oriented Programming）面向切面编程*
  - *从此：编程就像搭积木*
  - *Spring*

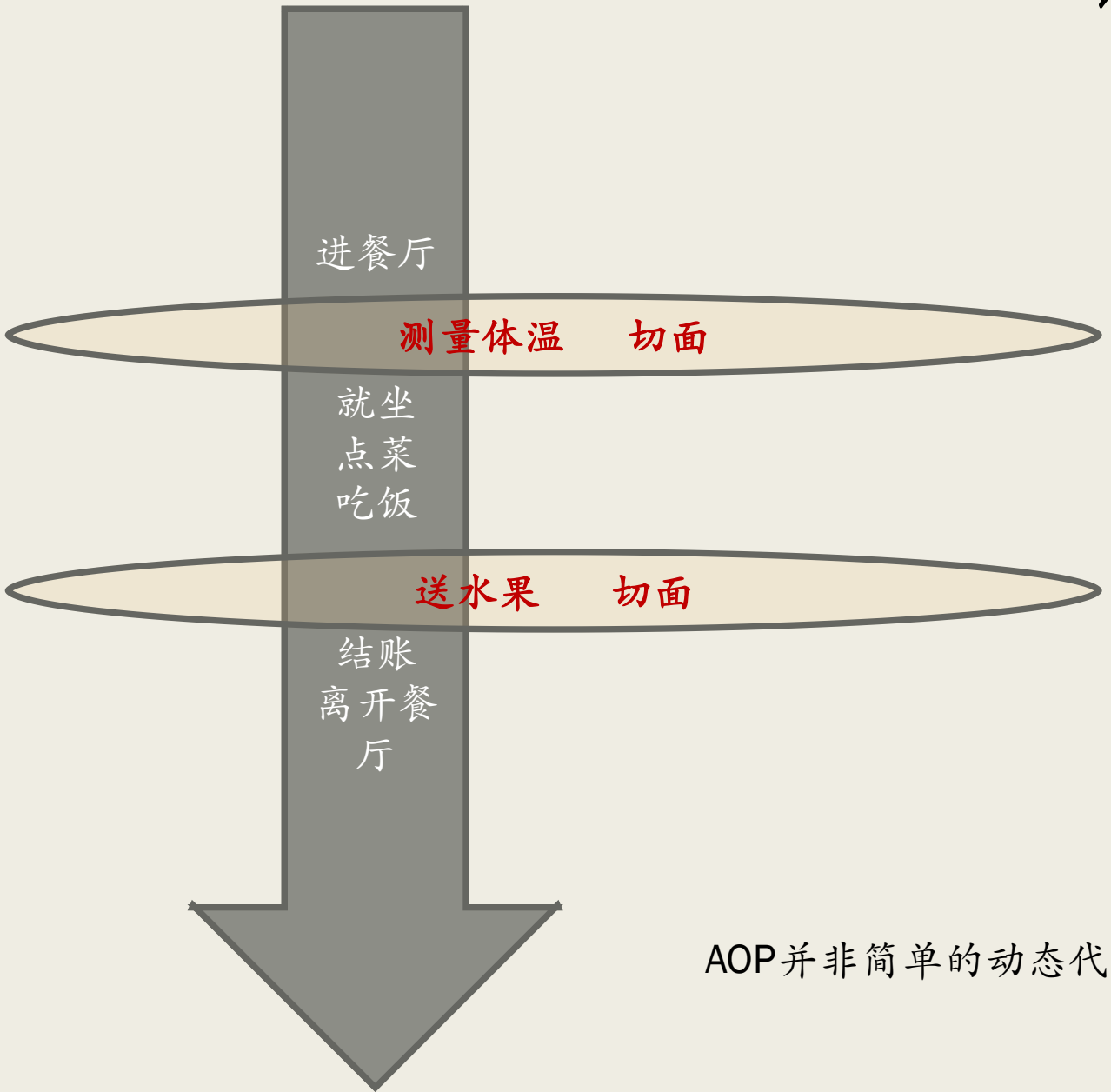
编程思想的发展

- 面向过程
- 面向对象
- 面向接口
- 面向切面



AOP

记得Filter过滤器?



AOP并非简单的动态代理的实现

AOP的应用场景

- 需要在方法前后进行功能增强和扩展
  - *日志*
  - *安全权限*
  - *事务管理*
- 需要对方法的执行顺序进行动态调整
  - *Before*
  - *After*
  - *Around*
  - *.....*

优势

- 将横切业务和主业务逻辑剥离
- 扩展功能不破坏原有业务逻辑
- 专注主要业务逻辑
- 代码复用
- 模块之间解藕

AOP的相关框架

- Spring的核心
  - *IoC*
  - *AOP—jdk动态代理&CGlib*
- AspectJ

Spring中的AOP基本概念

- 通知（Advice）
- 连接点（JoinPoint）
- 切入点（Pointcut）
- 切面（Aspect）
  - 引入（introduction）
  - 目标（target）
  - 代理(proxy)
  - 织入(weaving)

近期小结

- 反射-一切动态编程方法的基础
- 面向接口（抽象类）的编程-面向对象编程的精髓和生命力
- IoC和DI设计模式-颠覆系统各层次之间对象依赖关系
- 动态代理模式-系统功能扩展解藕的神器
- AOP-新的编程思想

- 都是目前各类框架采用的主流技术