设计模式

框架中的设计模式

项目中的设计模式

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

面试官怎么问

你之前项目中用过设计模式吗？

个人技能

1、具备扎实的编程基础，精通 Java 语言，熟悉 JVM，内存模型，并发编程
2、可以灵活运用设计模式，如：单例、工厂、策略、责任链、模板方法等设计模式进行项目开发
3、熟悉 Spring、 SpringMVC、SpringBoot、SpringCloud、Mybaits、Mybaits-Plus 等开发技术。
4、熟悉分布式常见解决方案：分布式事务、分布式一致性、分布式锁
5、熟悉关系型数据库 MySQL，有一定的 SQL 优化经验
6、熟练使用 Redis 等非关系型数据库。
7、熟练运用 Freemarker 模板技术和 nginx 反向代理服务器
8、熟练运用 RabbitMQ、Kafka 等主流消息中间件
9、熟练运用 git、Linux 操作系统等基本命令。
10、有高并发、高性能、高可用系统架构设计实践、并有性能调优经验

目录

Contents

工厂方法模式

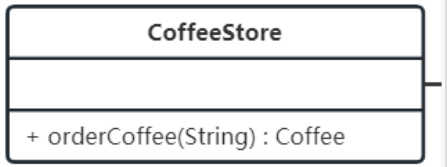
- 简单工厂模式
- 工厂方法模式
- 抽象工厂模式

工厂模式

需求：设计一个咖啡店点餐系统。

设计一个咖啡类（Coffee），并定义其两个子类（美式咖啡【AmericanCoffee】和拿铁咖啡【LatteCoffee】）；再设计一个咖啡店类（CoffeeStore），咖啡店具有点咖啡的功能。

具体类的设计如下：




```
/**
 * 根据类型选择不同的咖啡
 * @param type
 * @return
 */
public static Coffee orderCoffee(String type){
    Coffee coffee = null;
    if("american".equals(type)){
        coffee = new AmericanCoffee();
    }else if ("latte".equals(type)){
        coffee = new LatteCoffee();
    }
    //添加配料
   coffee.addMilk();
    coffee.addSuqar();
    return coffee;
}
```

1.类图中的符号

- +：表示public
- -：表示private
- #：表示protected

2.泛化关系(继承)用带空心三角箭头的实线来表示

3.实现关系用带空心的三角箭头的虚线来表示

4.依赖关系使用带箭头的虚线来表示

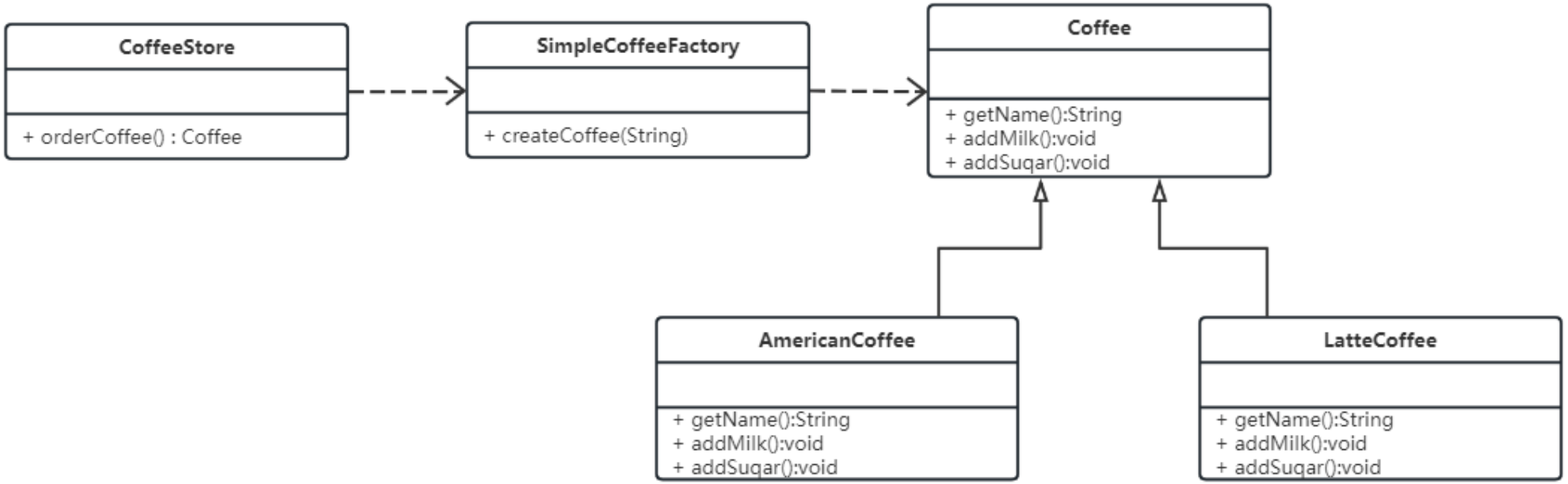
开闭原则：扩展开放，对修改关闭

工厂设计模式：**解耦**

简单工厂模式

简单工厂包含如下角色：

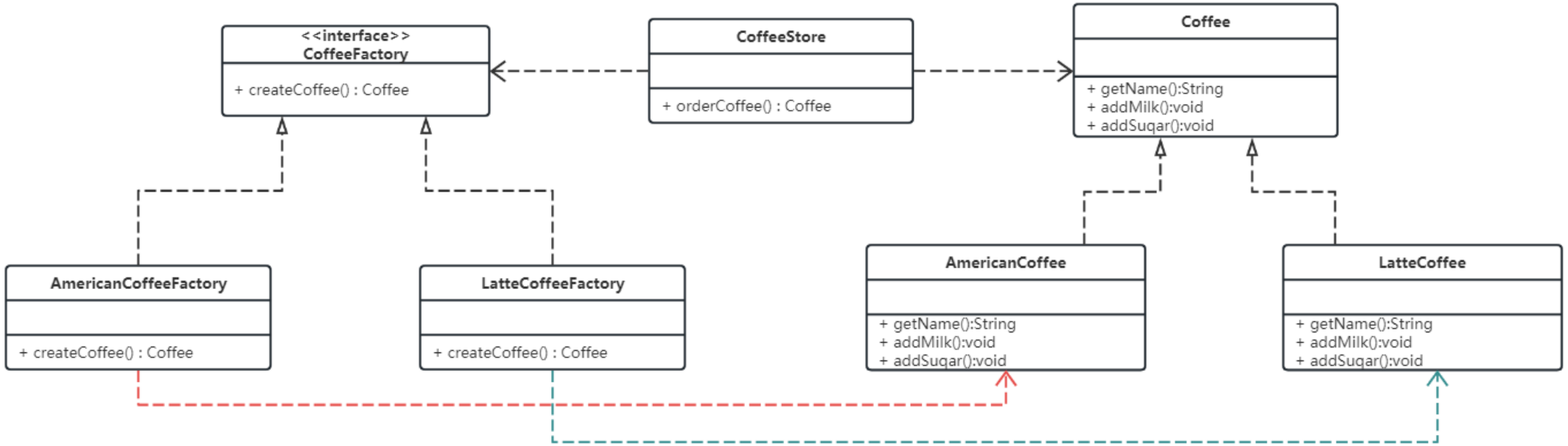
- 抽象产品 ：定义了产品的规范，描述了产品的主要特性和功能。
- 具体产品 ：实现或者继承抽象产品的子类
- 具体工厂 ：提供了创建产品的方法，调用者通过该方法来获取产品。



工厂方法模式

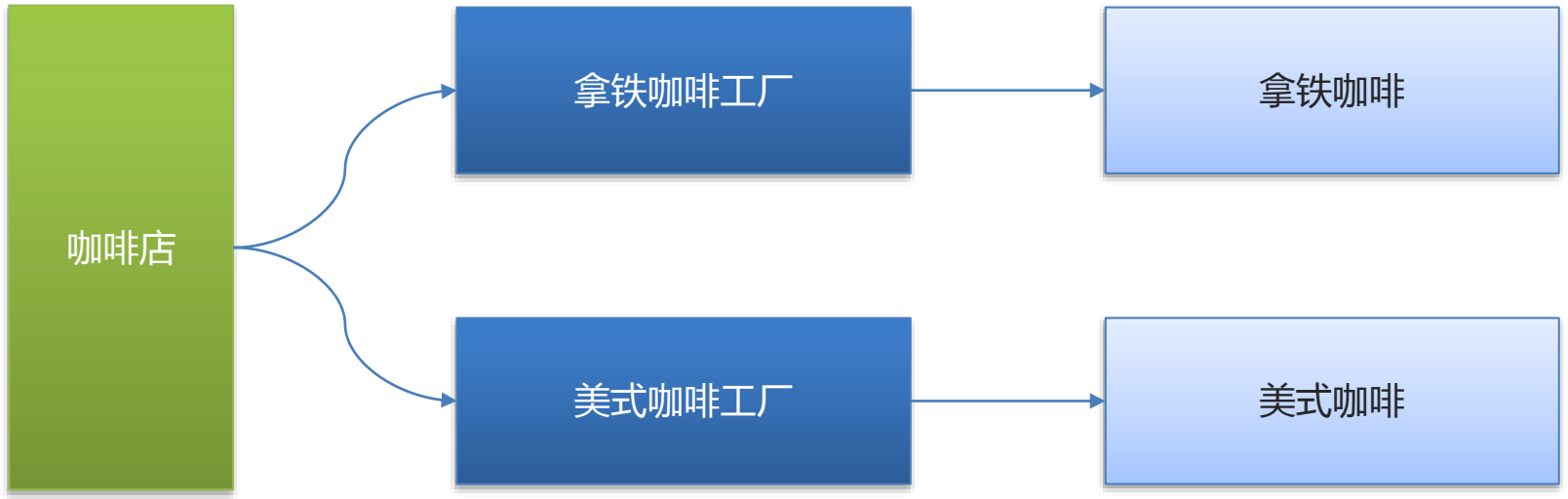
工厂方法模式的主要角色：

- 抽象工厂（Abstract Factory）：提供了创建产品的接口，调用者通过它访问具体工厂的工厂方法来创建产品。
- 具体工厂（ConcreteFactory）：主要是实现抽象工厂中的抽象方法，完成具体产品的创建。
- 抽象产品（Product）：定义了产品的规范，描述了产品的主要特性和功能。
- 具体产品（ConcreteProduct）：实现了抽象产品角色所定义的接口，由具体工厂来创建，它同具体工厂之间一一对应。



工厂方法模式

调用关系



优点：

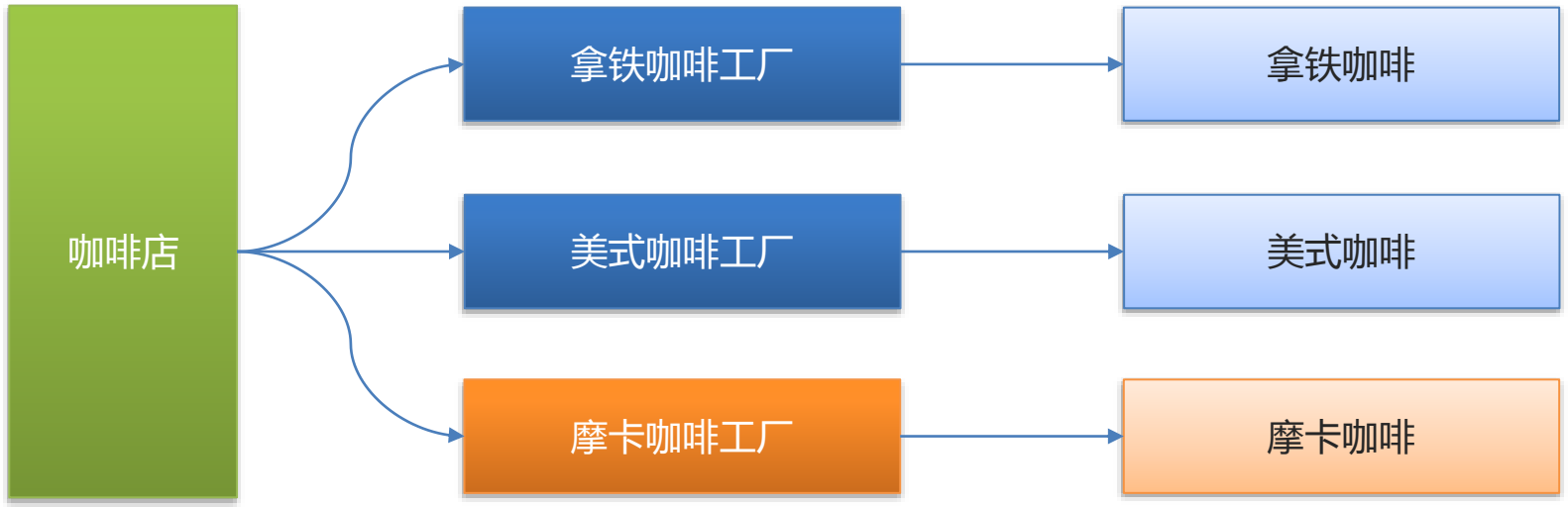
- 用户只需要知道具体工厂的名称就可得到所要的产品，无须知道产品的具体创建过程；
- 在系统增加新的产品时只需要添加具体产品类和对应的具体工厂类，无须对原工厂进行任何修改，满足开闭原则；

缺点：

- 每增加一个产品就要增加一个具体产品类和一个对应的具体工厂类，这增加了系统的复杂度。

工厂方法模式

调用关系



优点：

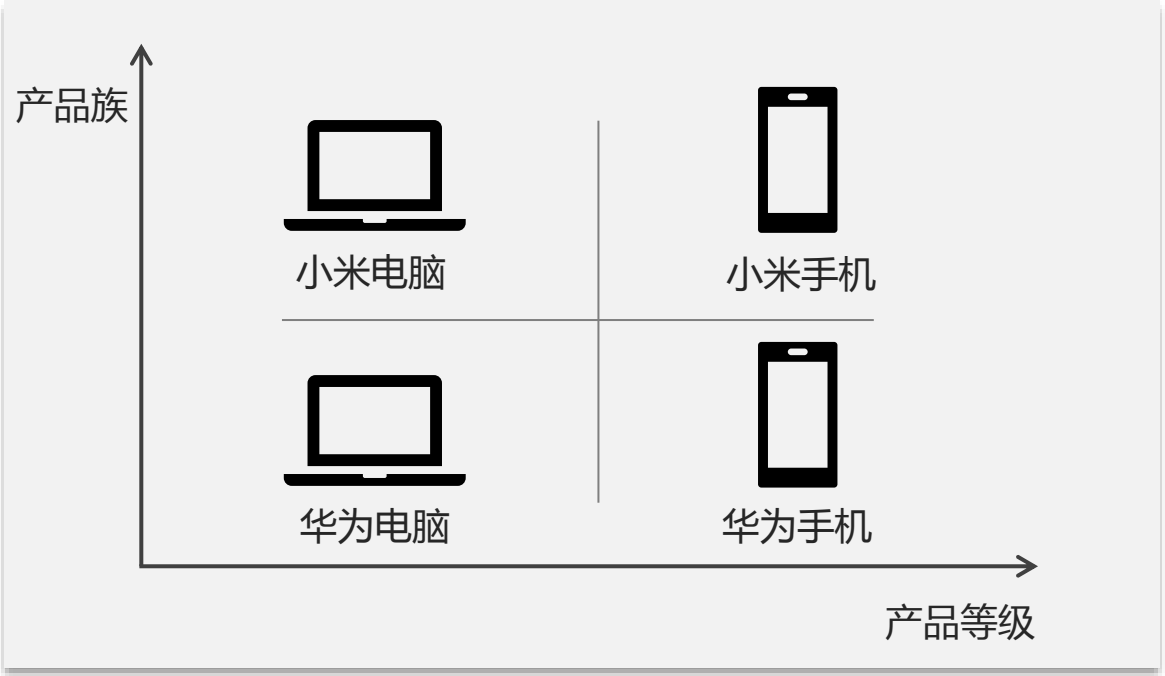
- 用户只需要知道具体工厂的名称就可得到所要的产品，无须知道产品的具体创建过程；
- 在系统增加新的产品时只需要添加具体产品类和对应的具体工厂类，无须对原工厂进行任何修改，满足开闭原则；

缺点：

- 每增加一个产品就要增加一个具体产品类和一个对应的具体工厂类，这增加了系统的复杂度。

抽象工厂模式

工厂方法模式只考虑生产同等级的产品，抽象工厂可以处理多等级产品的生产

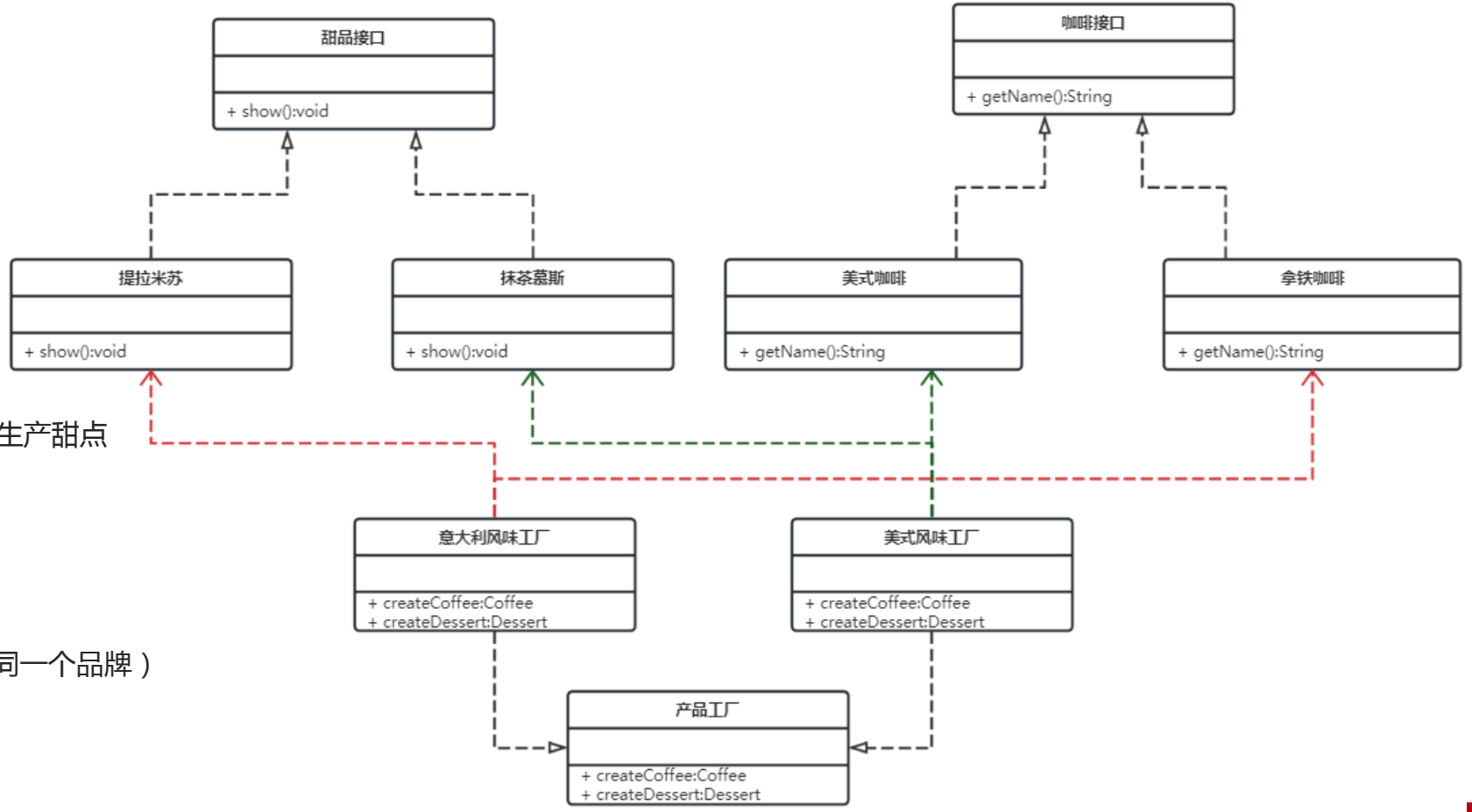


- 产品族：一个品牌下面的所有产品；例如华为下面的电脑、手机称为华为的产品族；
- 产品等级：多个品牌下面的同种产品；例如华为和小米都有手机电脑为一个产品等级；

抽象工厂模式

抽象工厂模式是工厂方法模式的升级版本，工厂方法模式只生产一个等级的产品，而抽象工厂模式可生产多个等级的产品。

一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂

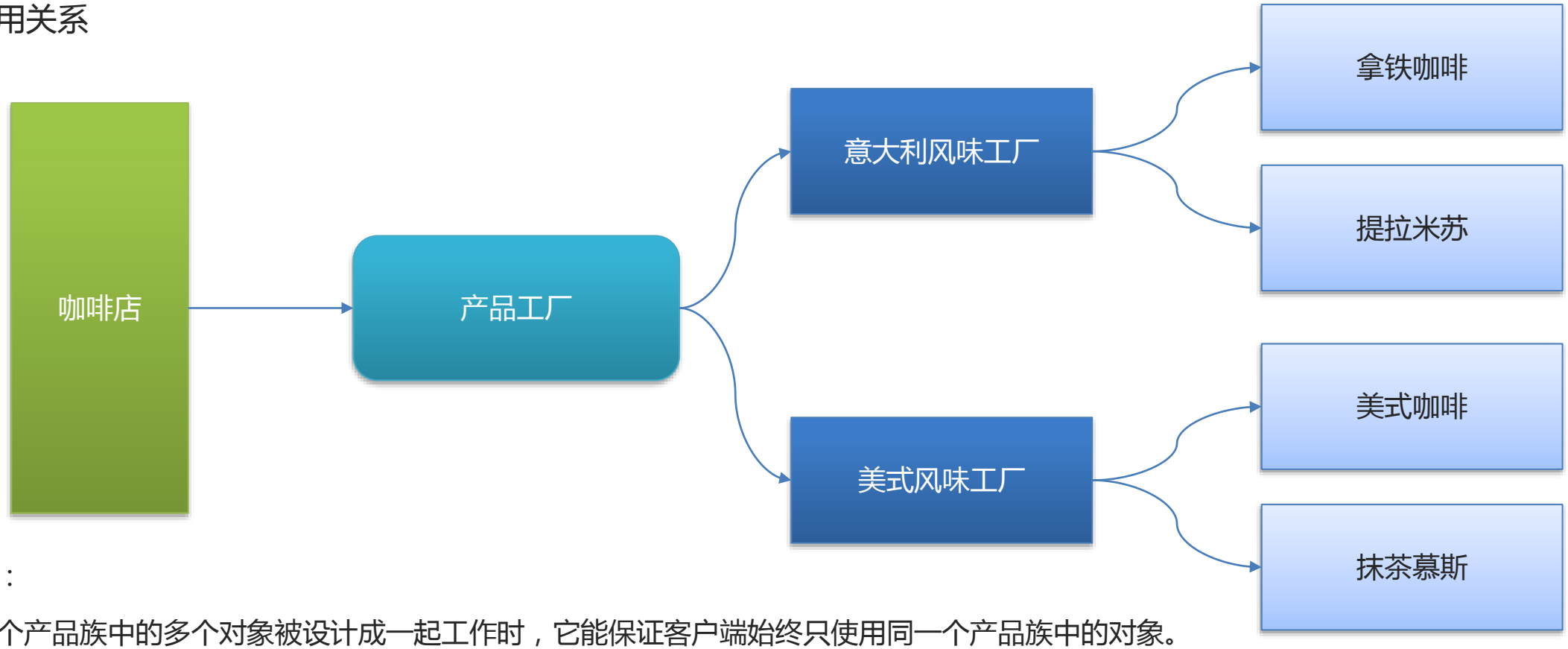


现咖啡店业务发生改变，不仅要生产咖啡还要生产甜点

- 同一个产品等级（产品分类）
  - 咖啡：拿铁咖啡、美式咖啡
  - 甜点：提拉米苏、抹茶慕斯
- 同一个风味，就是同一个产品族（相当于同一个品牌）
  - 美式风味：美式咖啡、抹茶慕斯
  - 意大利风味：拿铁咖啡、提拉米苏

抽象工厂模式

调用关系

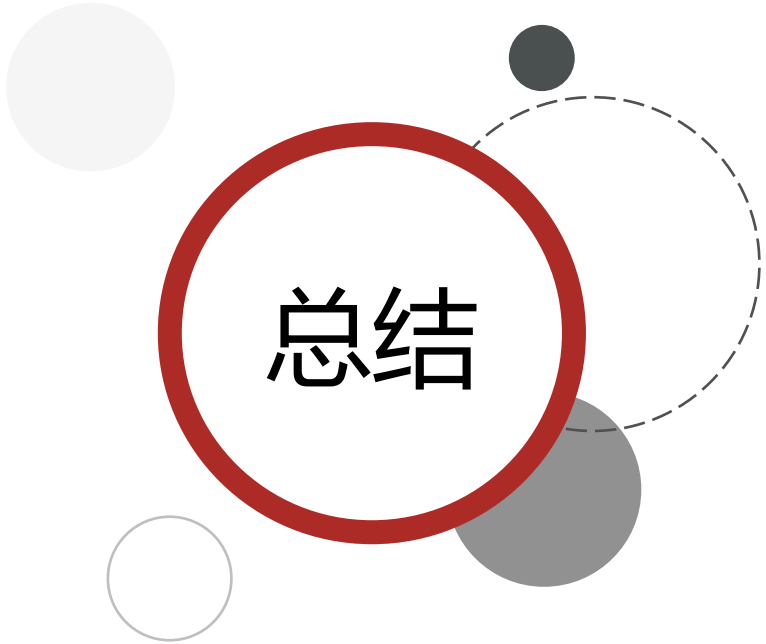


优点：

当一个产品族中的多个对象被设计成一起工作时，它能保证客户端始终只使用同一个产品族中的对象。

缺点：

当产品族中需要增加一个新的产品时，所有的工厂类都需要进行修改。



1. 简单工厂

- 所有的产品都共有一个工厂，如果新增产品，则需要修改代码，违反开闭原则
- 是一种编程习惯，可以借鉴这种编程思路

2. 工厂方法模式

- 给每个产品都提供了一个工厂，让工厂专门负责对应的产品的生产，遵循开闭原则
- 项目中用的最多

3. 抽象工厂方法模式

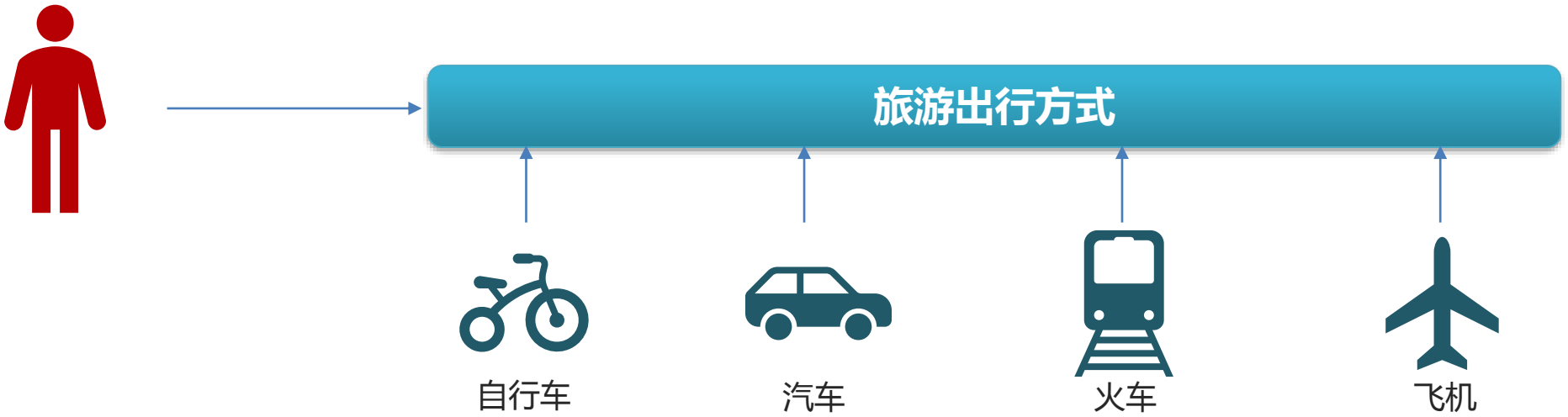
- 如果有多个纬度的产品需要配合生产时，优先建议采用抽象工厂（工厂的工厂）
- 一般的企业开发中的较少

策略模式

- 策略模式
- 案例（策略+工厂）

策略模式

- 该模式定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的变化不会影响使用算法的客户
- 它通过对算法进行封装，把使用算法的责任和算法的实现分割开来，并委派给不同的对象对这些算法进行管理

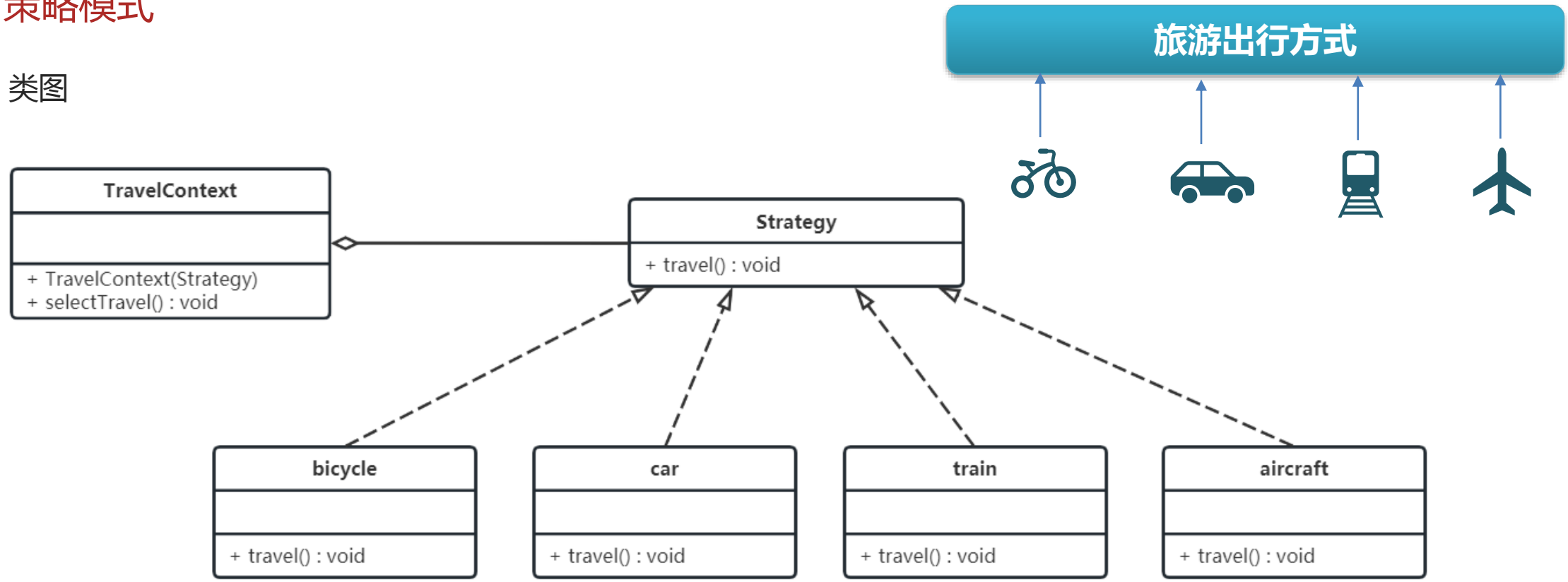


策略模式的主要角色如下：

- 抽象策略（Strategy）类：这是一个抽象角色，通常由一个接口或抽象类实现。此角色给出所有的具体策略类所需的接口。
- 具体策略（Concrete Strategy）类：实现了抽象策略定义的接口，提供具体的算法实现或行为。
- 环境（Context）类：持有一个策略类的引用，最终给客户端调用。

策略模式

类图



优点：

- 策略类之间可以自由切换
- 易于扩展
- 避免使用多重条件选择语句（if else），充分体现面向对象设计思想。

缺点：

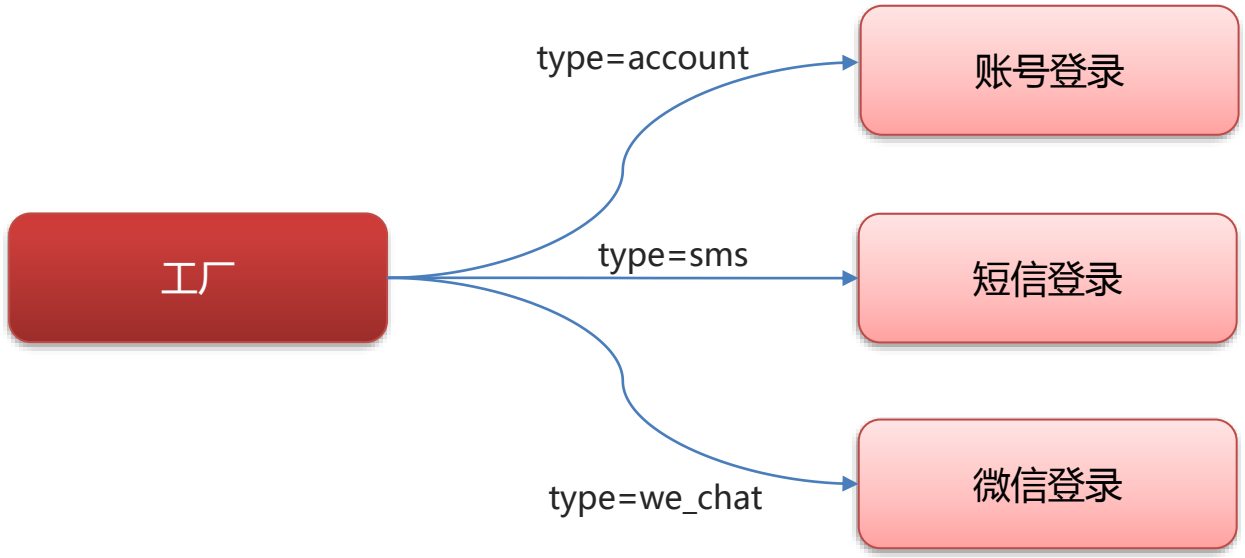
- 客户端必须知道所有的策略类，并自行决定使用哪一个策略类。
- 策略模式将造成产生很多策略类

登录案例（工厂模式+策略模式）

右图是gitee的登录的入口，其中有多种方式可以进行登录

- 用户名密码登录
- 短信验证码登录
- 微信登录
- QQ登录
- ....

登录案例（工厂模式+策略模式）

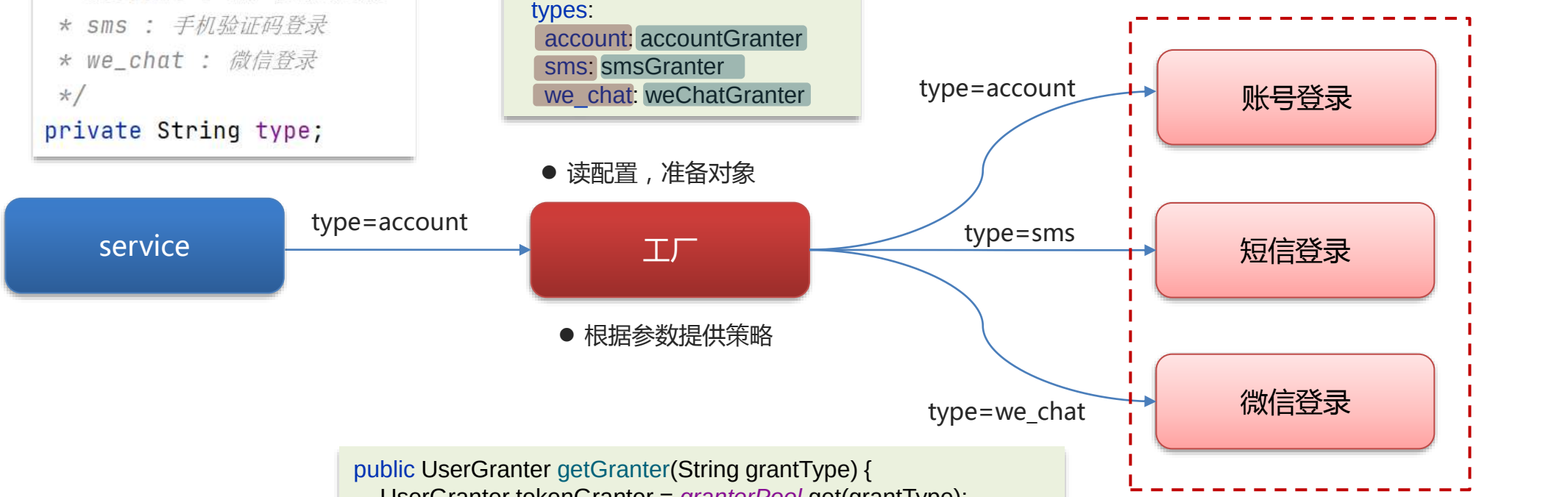


登录案例（工厂模式+策略模式）

```
/**
 * account : 用户名密码登录
 * sms : 手机验证码登录
 * we_chat : 微信登录
 */
private String type;
```

```
login:
  types:
    account: accountGranter
    sms: smsGranter
    we_chat: weChatGranter
```

```
@Component
public class AccountGranter implements UserGranter{
```




```
public UserGranter getGranter(String grantType) {
    UserGranter tokenGranter = granterPool.get(grantType);
    return tokenGranter;
}
```

登录案例（工厂模式+策略模式）

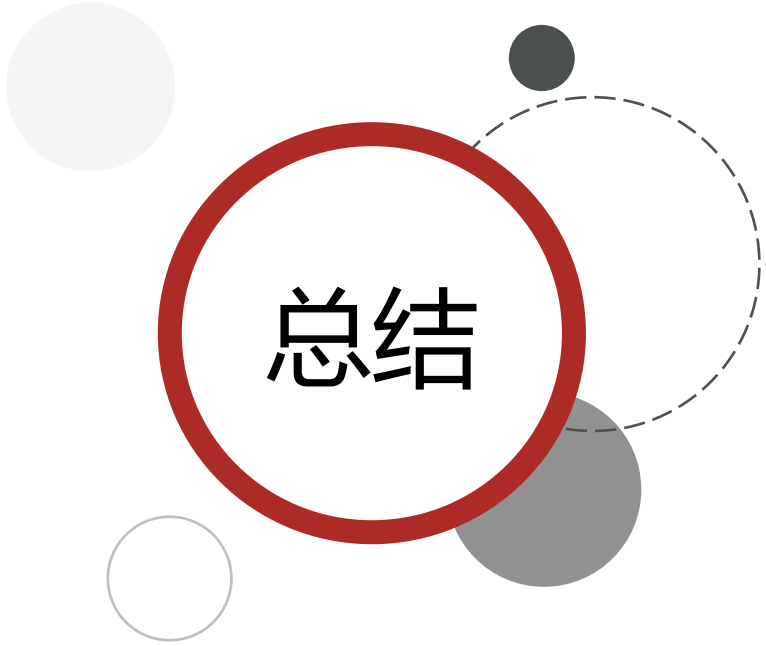
```
login:
  types:
    account: accountGranter
    sms: smsGranter
    we_chat: weChatGranter
    qq: qqGranter
```


```
public UserGranter getGranter(String grantType) {
    UserGranter tokenGranter = granterPool.get(grantType);
    return tokenGranter;
}
```

举一反三

- 订单的支付策略（支付宝、微信、银行卡...）
- 解析不同类型excel（xls格式、xlsx格式）
- 打折促销（满300元9折、满500元8折、满1000元7折...）
- 物流运费阶梯计算（5kg以下、5-10kg、10-20kg、20kg以上）

一句话总结：只要代码中有冗长的 if-else 或 switch 分支判断都可以采用策略模式优化



1. 什么是策略模式

- 策略模式定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的变化不会影响使用算法的客户
- 一个系统需要动态地在几种算法中选择一种时，可将每个算法封装到策略类中

2. 案例（工厂方法+策略）

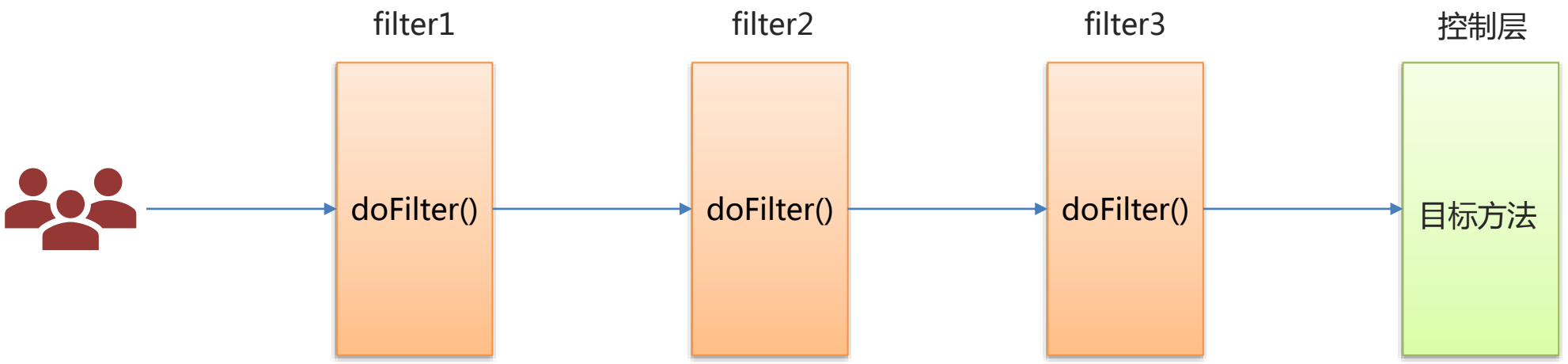
- 介绍业务（登录、支付、解析excel、优惠等级...）
- 提供了很多种策略，都让spring容器管理
- 提供一个工厂：准备策略对象，根据参数提供对象

责任链设计模式

- 概述
- 使用场景

责任链设计模式

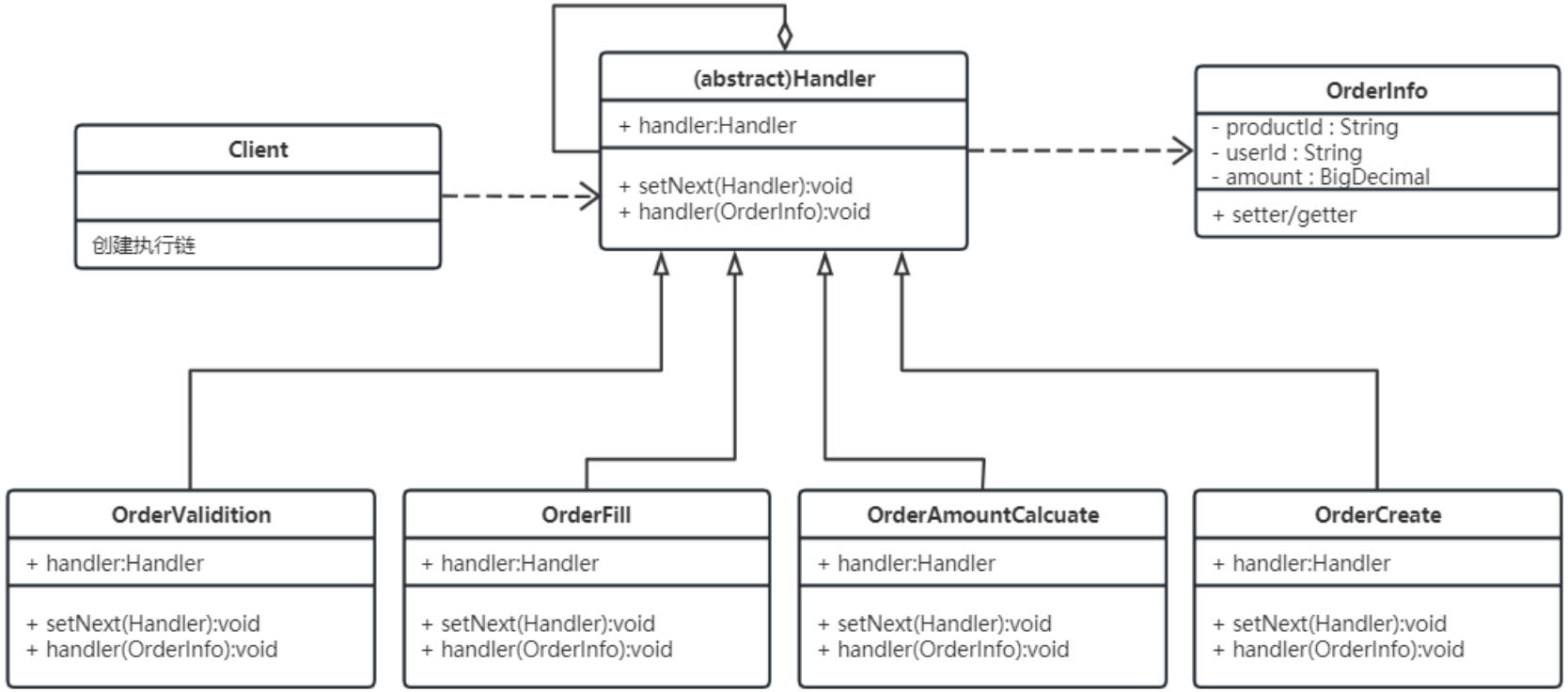
责任链模式：为了避免请求发送者与多个请求处理者耦合在一起，将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止。



责任链设计模式

- 抽象处理者（Handler）角色：定义一个处理请求的接口，包含抽象处理方法和一个后继连接。
- 具体处理者（Concrete Handler）角色：实现抽象处理者的处理方法，判断能否处理本次请求，如果可以处理请求则处理，否则将该请求转给它的后继者。
- 客户类（Client）角色：创建处理链，并向链头的具体处理者对象提交请求，它不关心处理细节和请求的传递过程。

处理订单的操作



责任链优缺点

优点：

- 降低了对象之间的耦合度
- 增强了系统的可扩展性
- 增强了给对象指派职责的灵活性
- 责任链简化了对象之间的连接
- 责任分担

缺点：

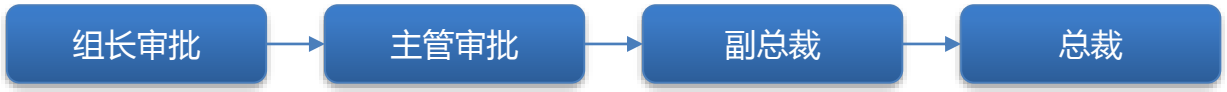
- 对比较长的职责链，请求的处理可能涉及多个处理对象，系统性能将受到一定影响。
- 职责链建立的合理性要靠客户端来保证，增加了客户端的复杂性，可能会由于职责链的错误设置而导致系统出错，如可能会造成循环调用。

举一反三

- 内容审核（视频、文章、课程....）

- 订单创建

- 简易流程审批




黑马程序员
www.itheima.com
传智教育旗下高端IT教育品牌