面向对象分析与设计

Object Oriented Analysis and Design

——软件设计概述

Introduction to Software Design

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

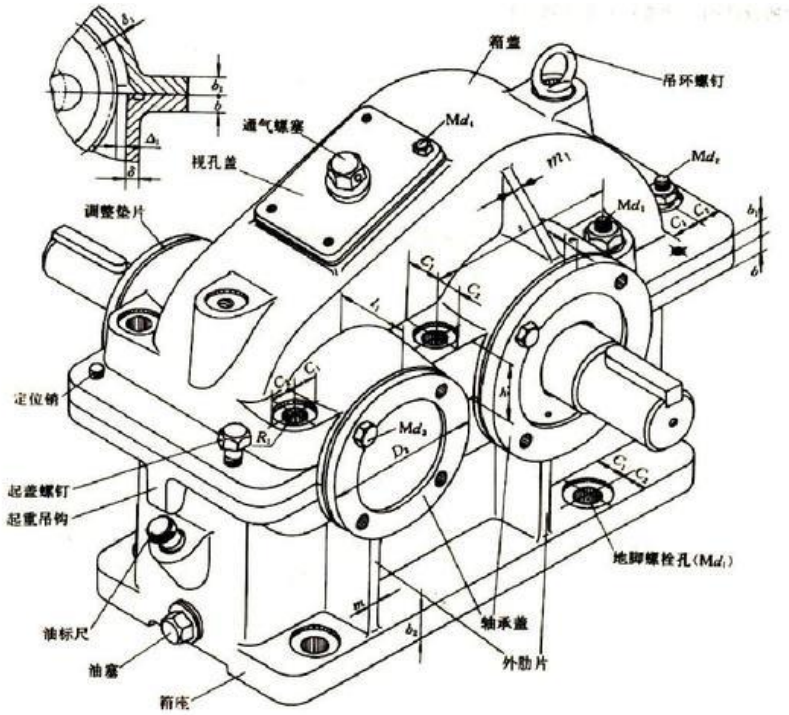
2024年秋季学期

目录

- 软件设计
- 编程范式
- 命令式编程
- 面向对象编程
- 函数式编程
- 统一建模语言UML

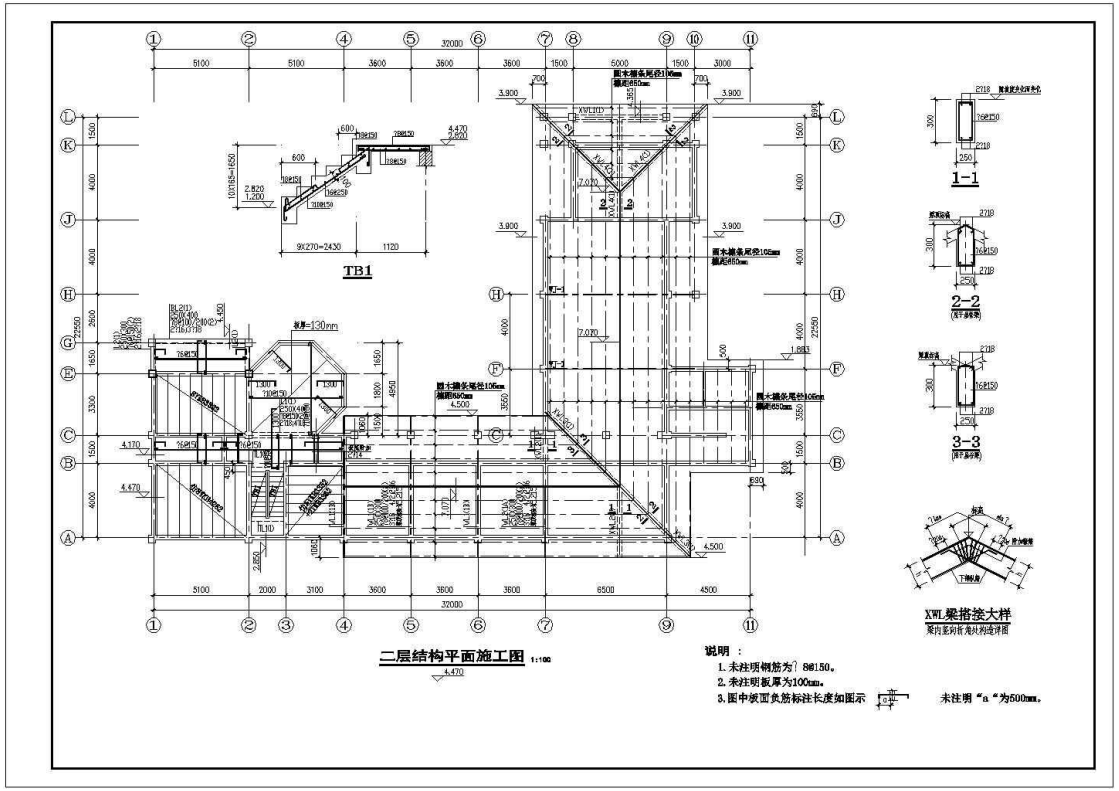
1. 软件设计
Software Design

- 设计是创造可用于解决某个问题的解决方案



1. 软件设计
Software Design

- 设计的基本思想是将复杂的问题分解成若干模块



1. 软件设计
Software Design

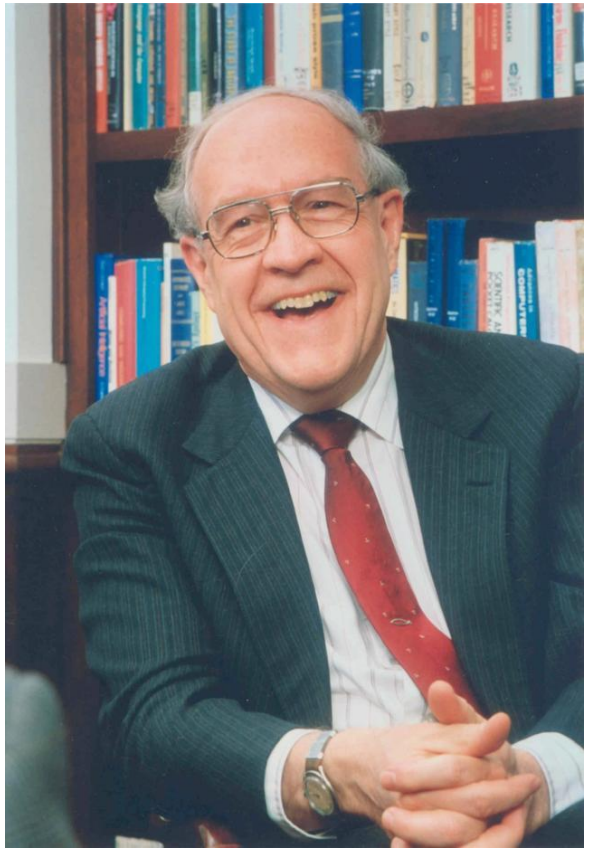
1. 软件设计
Software Design

- 软件固有的复杂性（Essential Complexity）
  - 问题域的复杂性
    - 软件是赋能产业，它需要和各行各业结合，从而使得软件的需求多样且不断变化
  - 管理开发过程的困难性
    - 软件开发是纯粹的智力活动，需要协调各种资源，应对不同的约束条件
  - 软件中的灵活性
    - 不同与传统行业，软件行业还未实现标准化生产，缺乏各类行业标准。
  - 描述离散系统行为
    - 软件系统中的状态转换多是离散系统行为，不能用连续函数描述。

1. 软件设计
Software Design

Brook’s Law

Adding manpower to a late software project makes it later.



Frederick P.Brooks Jr.
(1931/4/19 – 2022/11/17)
北卡罗来纳大学教堂山分校教授
1999年图灵奖得主
《人月神话》作者

1. 软件设计
Software Design

- 软件设计的目的是用“分而治之”的方式，给出满足以下条件的解决方案
  - 满足给定的功能规格说明
  - 满足性能和资源使用需求
  - 满足产品形式的设计限制条件
  - 满足对设计过程的限制条件，如时间、费用或可用工具等

2. 编程范式
Programming Paradigm

- 编程范式
  - 来自 Robert Floyd 在 1978 年12月4日图灵奖的颁奖演说（The Paradigms of Programming）
  - 是指程序的设计方法，即程序应该如何被构建和执行
    - 结构化编程（Structured Programming）
    - 面向对象编程（Object-Oriented Programming）
    - 函数式编程（Functional Programming）

Robert Floyd
(1936/6/8 –2001/09/25)
斯坦福大学教授
1978年图灵奖得主
Algol 60的开发者，HeapSort,
Floyd算法

3. 命令式编程
Imperative Programming

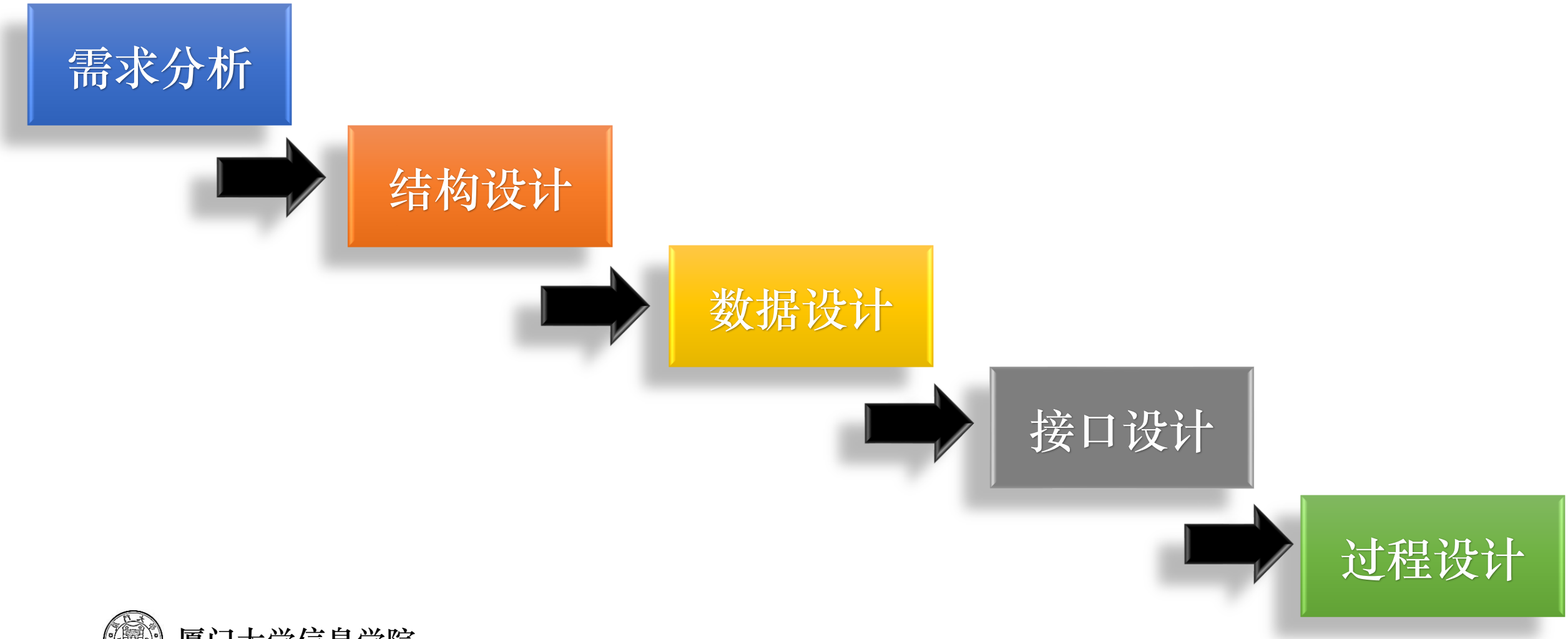
- 面向过程设计
  - 将解决问题的方案抽象为一系列步骤，通过编程的方式将这些步骤转化成程序指令
  - 这些指令用于处理输入数据，得到输出

3. 命令式编程
Imperative Programming

- 结构化编程的例子

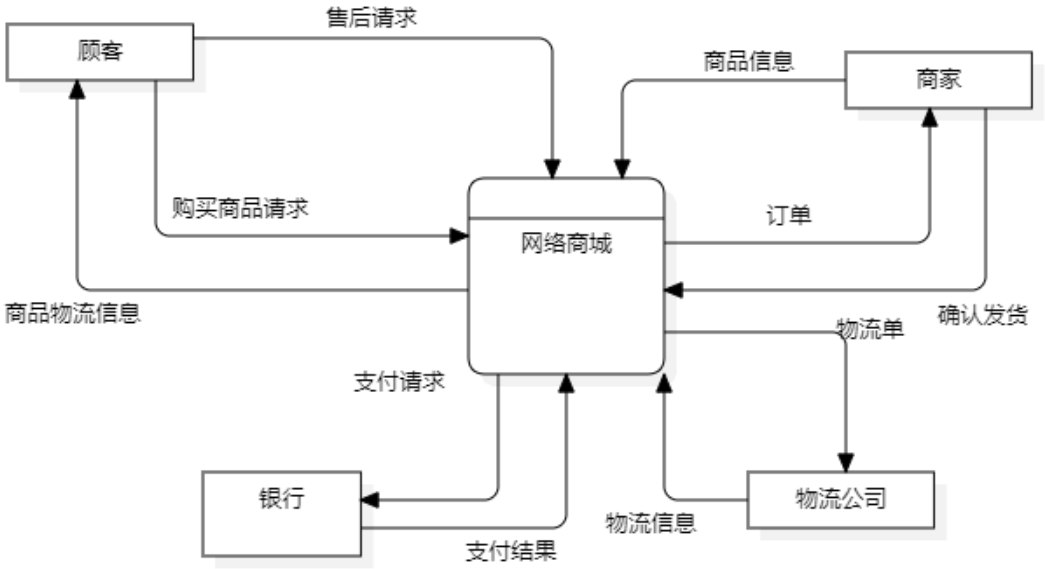
```
public void updateOnsale(Long shopId, Long id, Integer price, LocalDateTime beginTime, LocalDateTime endTime, Integer quantity, Byte type, UserDto user) {
        Onsale onsale = onsaleDao.findById(id);
        if(Constants.PLATFORM!=onsale.getShopId() && shopId!=onsale.getShopId()){
                throw new BusinessException(ReturnNo.RESOURCE_ID_OUTSCOPE, String.format(ReturnNo.RESOURCE_ID_OUTSCOPE.getMessage(), "价格浮动", id, shopId));
        }
        if(price!=null) onsale.setPrice(Long.valueOf(price));
        if(beginTime!=Constants.BEGIN_TIME) {
        if(beginTime!=Constants.BEGIN_TIME) onsale.setBeginTime(beginTime);
        onsale.setBeginTime(beginTime);
        if(endTime!=Constants.END_TIME) onsale.setEndTime(endTime);
        beginTime=onsale.getBeginTime();
        if(quantity!=null) onsale.setQuantity(quantity);
        if(type!=null) onsale.setType(type);
        if(endTime!=Constants.END_TIME) {
                onsale.setEndTime(endTime);
                endTime=onsale.getEndTime();
        }
        List<Onsale> onsaleByProductId = onsaleDao.findOnsaleByProductId(id);
        //判断商品时间是否冲突
        for(Onsale o:onsaleByProductId){
                LocalDateTime beginTimeExist = o.getBeginTime();
                LocalDateTime endTimeExist = o.getEndTime();
                if(!(endTimeExist.isBefore(beginTime)||beginTimeExist.isAfter(endTime))){
                        throw new BusinessException(ReturnNo.getByCode(299), "商品时间冲突");
                }
        }
        onsaleDao.save(onsale, user);
```

3. 命令式编程
Imperative Programming

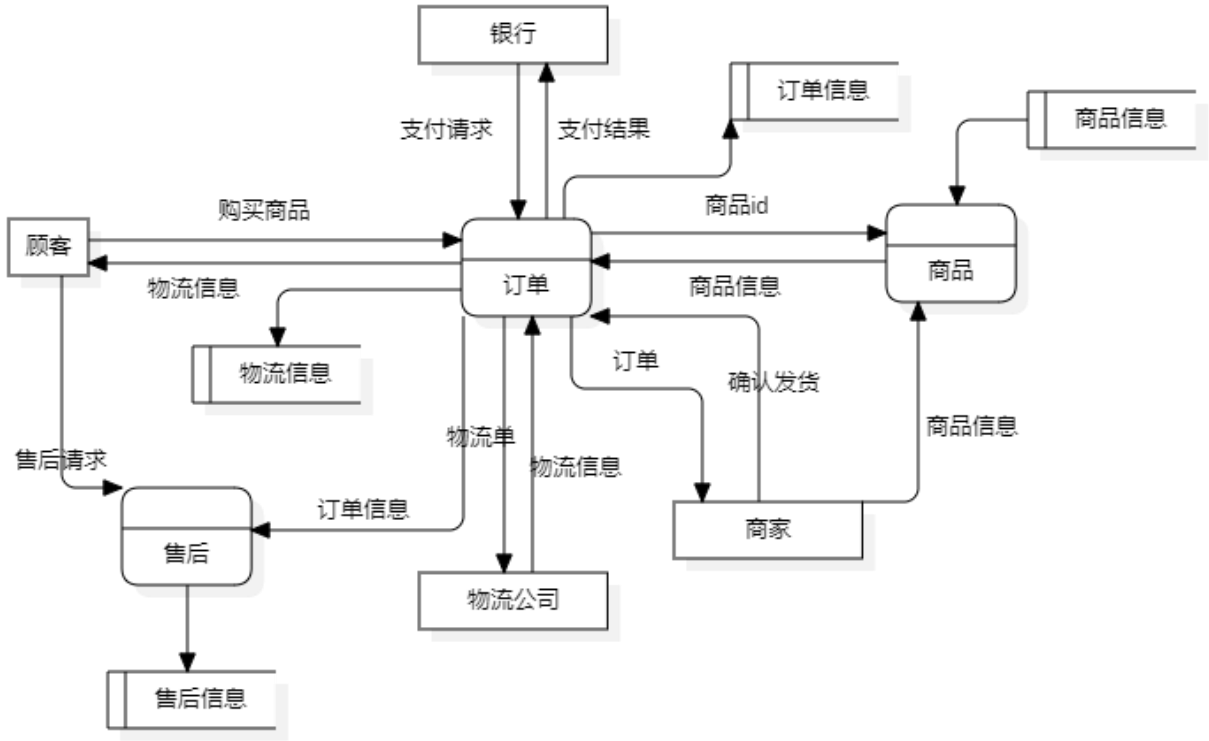


3. 命令式编程
Imperative Programming

- 主要的设计工具为数据流图



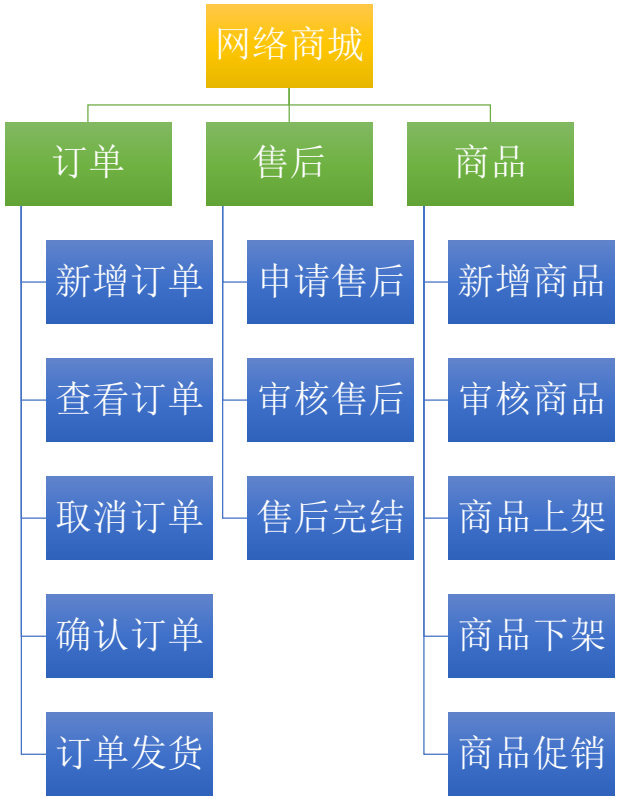
0级数据流图



1级数据流图

3. 命令式编程
Imperative Programming

- 设计结果必然是一个设计树



4. 面向对象编程
Object-Oriented Programming

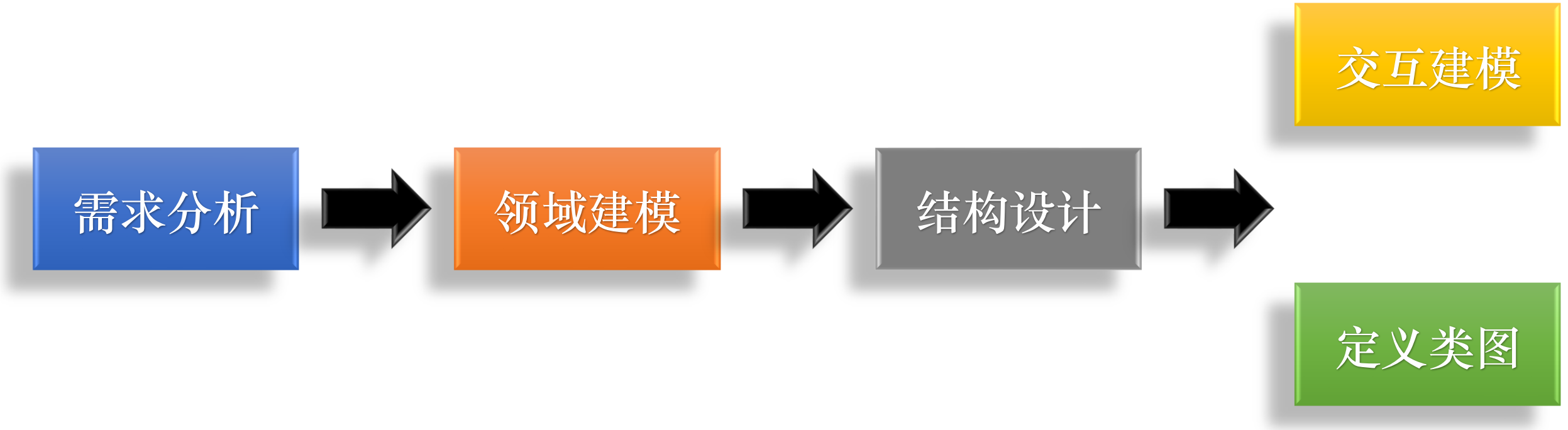
- 面向对象设计—领域驱动设计
  - 从业务需求产生反映事物抽象特征的领域模型
  - 以领域模型产生对象模型（类和接口）
  - 将数据包裹在类和对象中，通过对象的协作来完成所需要的功能

4. 面向对象编程
Object-Oriented Programming

- 面向对象编程的例子

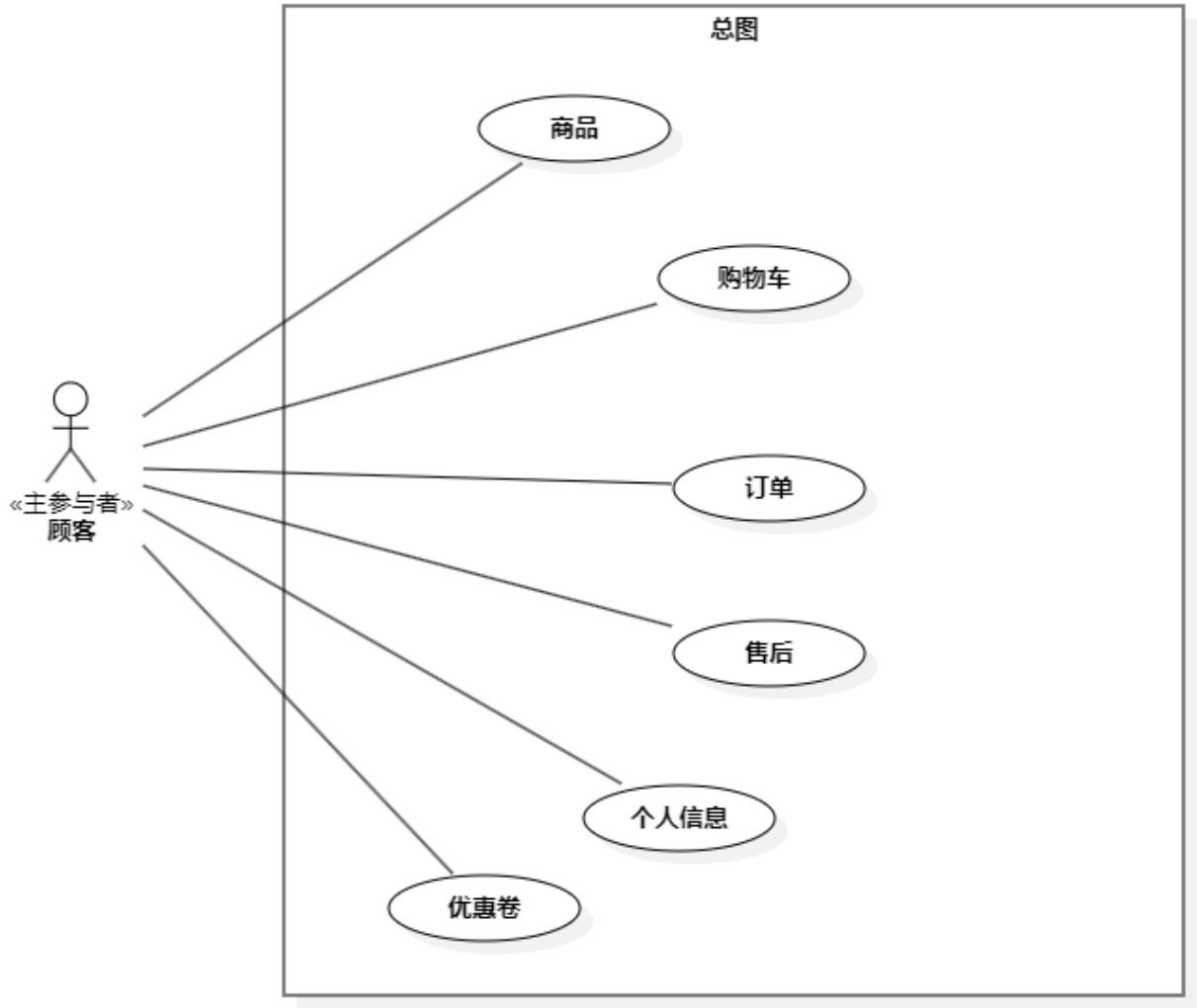
```
public void save(Long shopId, Long id, Onsale onsale, UserDto user) {
    Onsale oldOnsale = onsaleDao.findById(id);
    if(PLATFORM != shopId && shopId != oldOnsale.getShopId()){
        throw new BusinessException(ReturnNo.RESOURCE_ID_OUTSCOPE, String.format(ReturnNo.RESOURCE_ID_OUTSCOPE.getMessage(), "销售", id, shopId));
    }
    Product product = this.productDao.findProductByBeginEnd(oldOnsale.getProductId(), onsale.getBeginTime(), onsale.getEndTime());
    Onsale validOnsale = product.getValidOnsale();
    if(null!=validOnsale&&validOnsale.getId()!=id){
        throw new BusinessException(GOODS_PRICE_CONFLICT,String.format(ReturnNo.GOODS_PRICE_CONFLICT.getMessage(),id));
    }
    onsale.setId(id);
    String key = this.onsaleDao.save(onsale, user);
    if (redisUtil.hasKey(key)){
        redisUtil.del(key);
    }
}
```

4. 面向对象编程
Object-Oriented Programming



4. 面向对象编程
Object-Oriented Programming

- 需求分析
  - 定义用例：用故事的方式描述使用者怎样使用系统



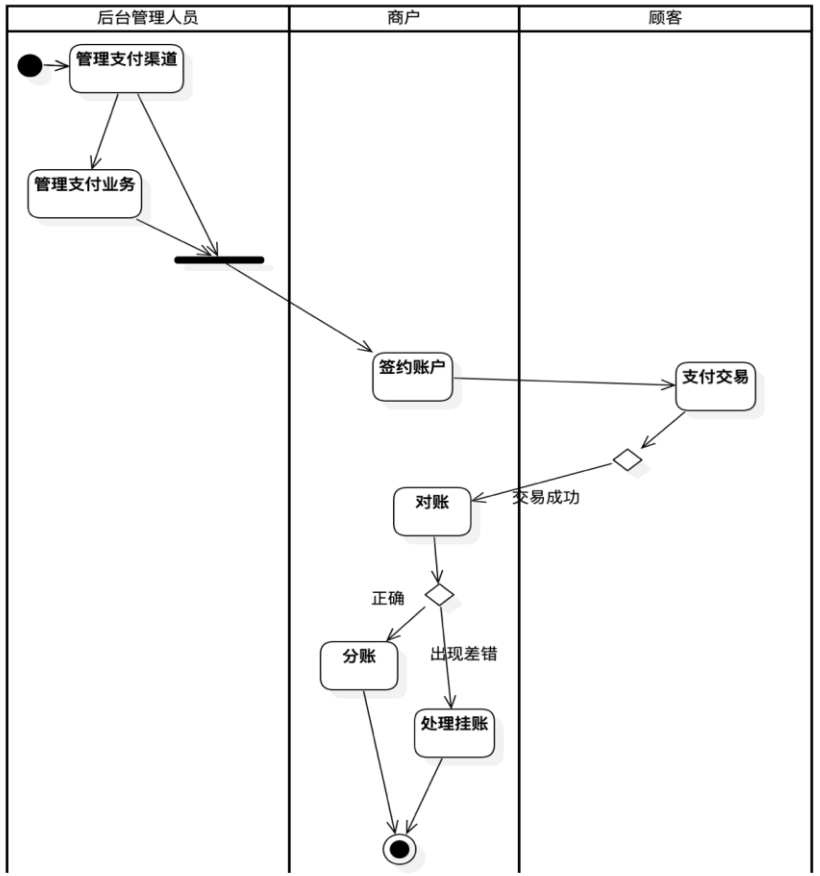
4. 面向对象编程
Object-Oriented Programming

- 用例描述

用例编号：MALL-004	用例名称：购买商品
级别：用户目标	
包： 商品	
参与者：顾客	
描述： 顾客购买商品，	
触发事件：顾客购买商品	
主成功场景	信息
1. 顾客购买商品。	商品id，数量
2. 系统计算总价、运费，推荐的优惠卷，选择默认的开票方式。	总价，运费，推荐的优惠卷，默认开票方式
3. 顾客提交订单	
4. 系统进入支付子用例（MALL-PAY-001）	
扩展场景	信息
2a. 商品售罄	
1. 系统显示商品售罄，终止后续步骤	
3a. 顾客选择其他优惠卷	
1. 系统显示所有适用的优惠卷	适用优惠卷
2．顾客选择其中一张优惠卷	优惠卷
3. 返回第2步重新计算总价	总价
3b. 顾客选择其他开票方式	
1. 系统显示顾客的所有开票方式	所有开票方式
2．顾客选择其中一种开票方式	开票方式
3. 返回第2步重新显示新选的开票方式	
4a. 商品售罄	
1. 系统显示商品售罄，终止后续步骤	
其他：若是第三方商户的交易，需通过分账模式完成	

4. 面向对象编程
Object-Oriented Programming

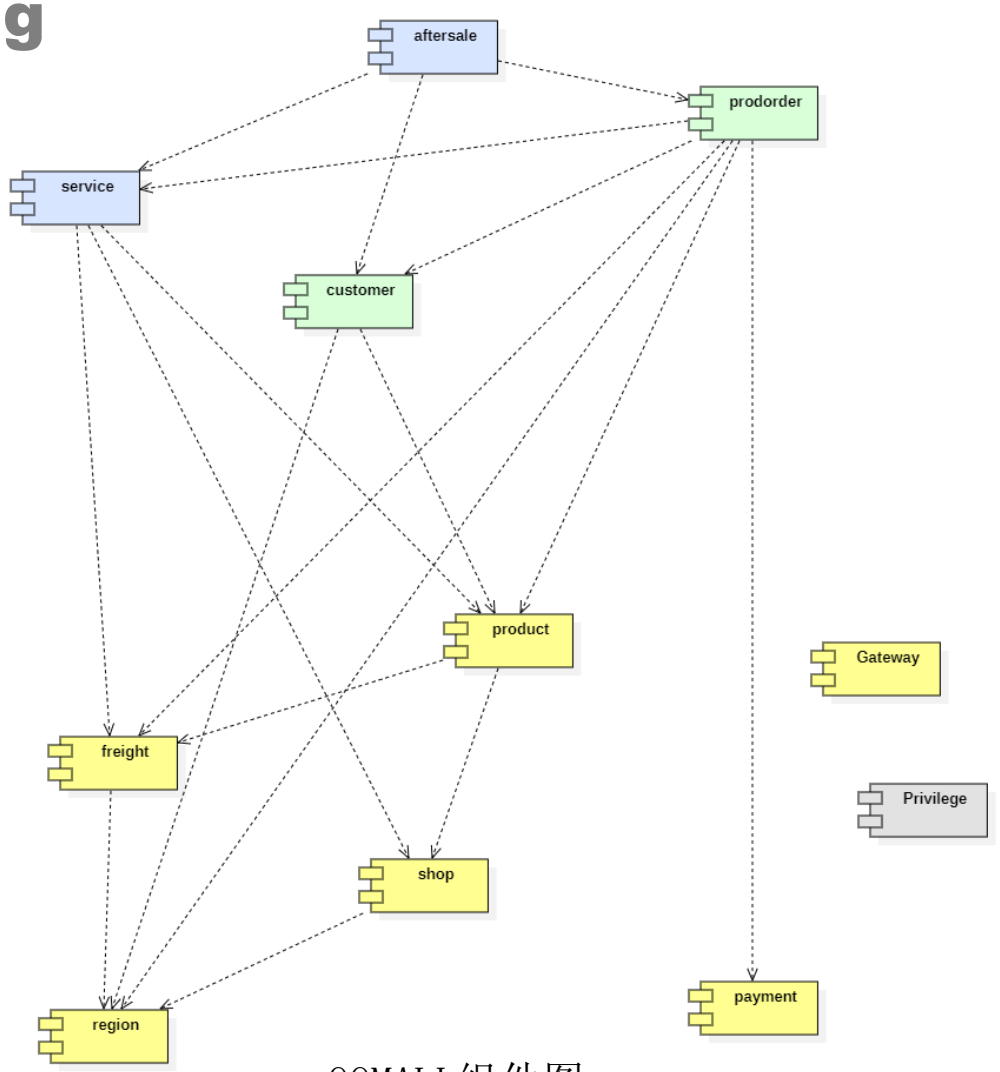
- 业务流程



支付业务流程

4. 面向对象编程
Object-Oriented Programming

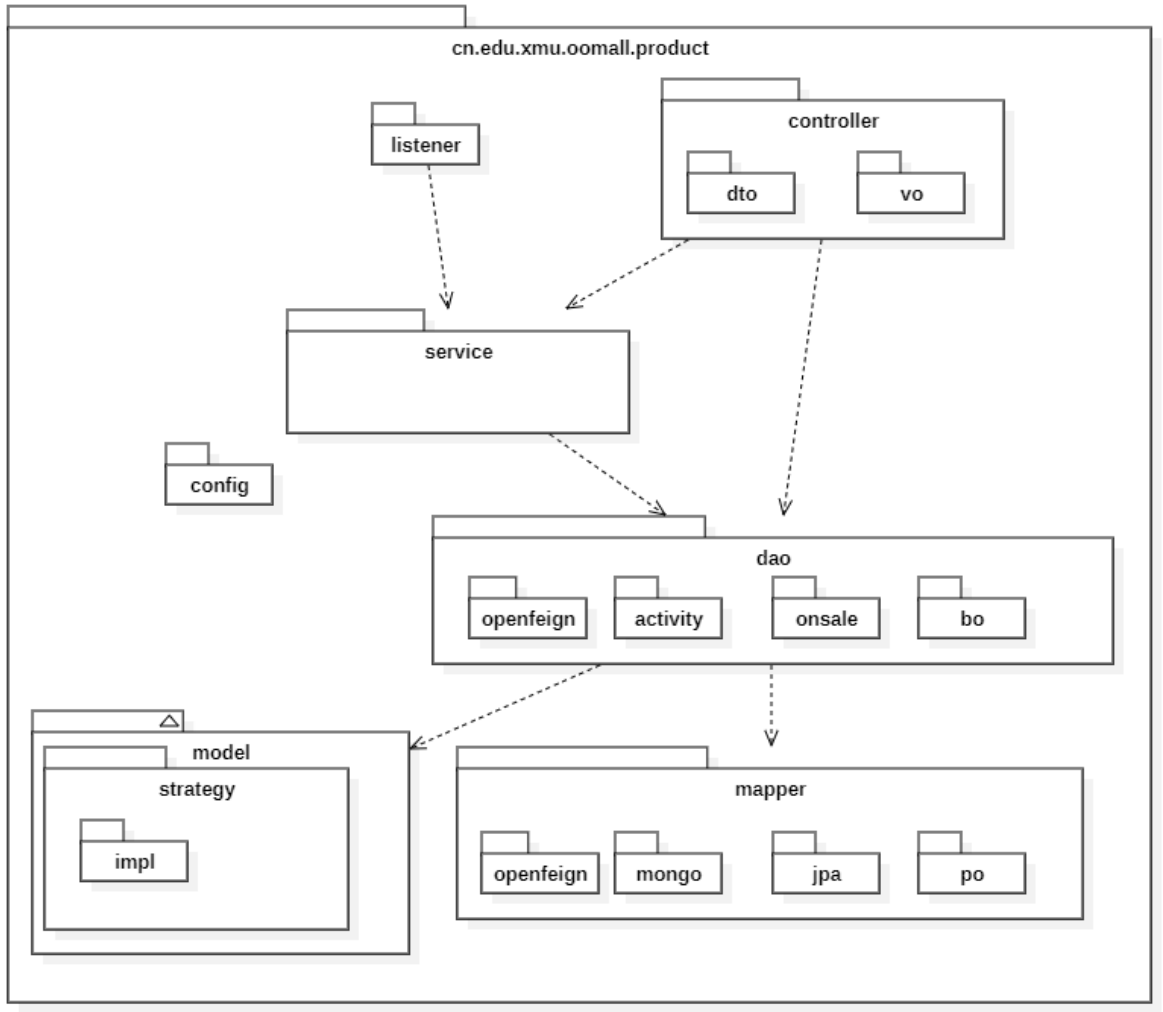
- 结构设计
  - 按照高内聚低耦合的原则将系统分割成若干模块，确定每个模块的体系结构



OOMALL组件图

4. 面向对象编程
Object-Oriented Programming

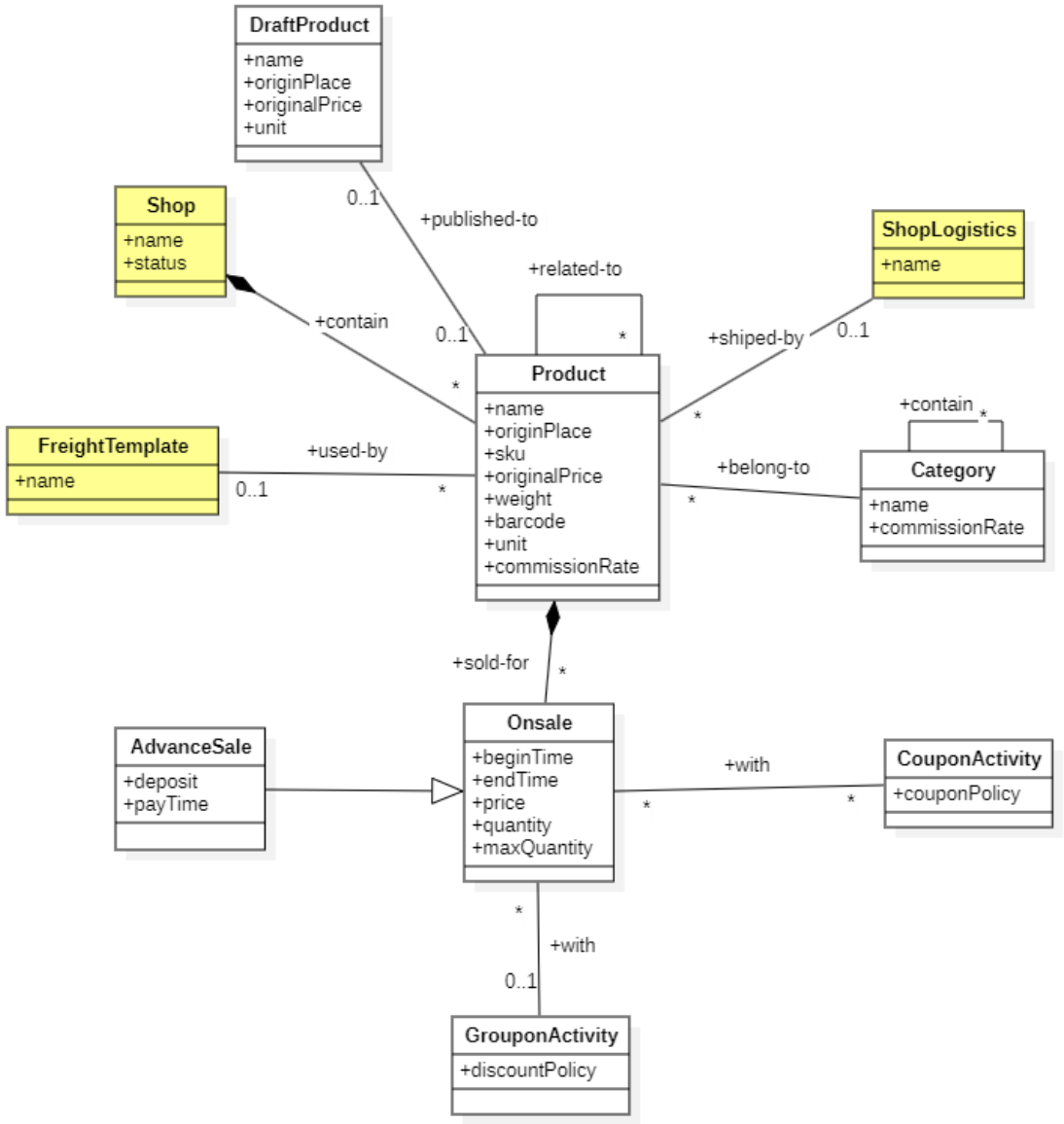
- 结构设计
  - 按照高内聚低耦合的原则将系统分割成若干模块，确定每个模块的体系结构



商品模块包图

4. 面向对象编程
Object-Oriented Programming

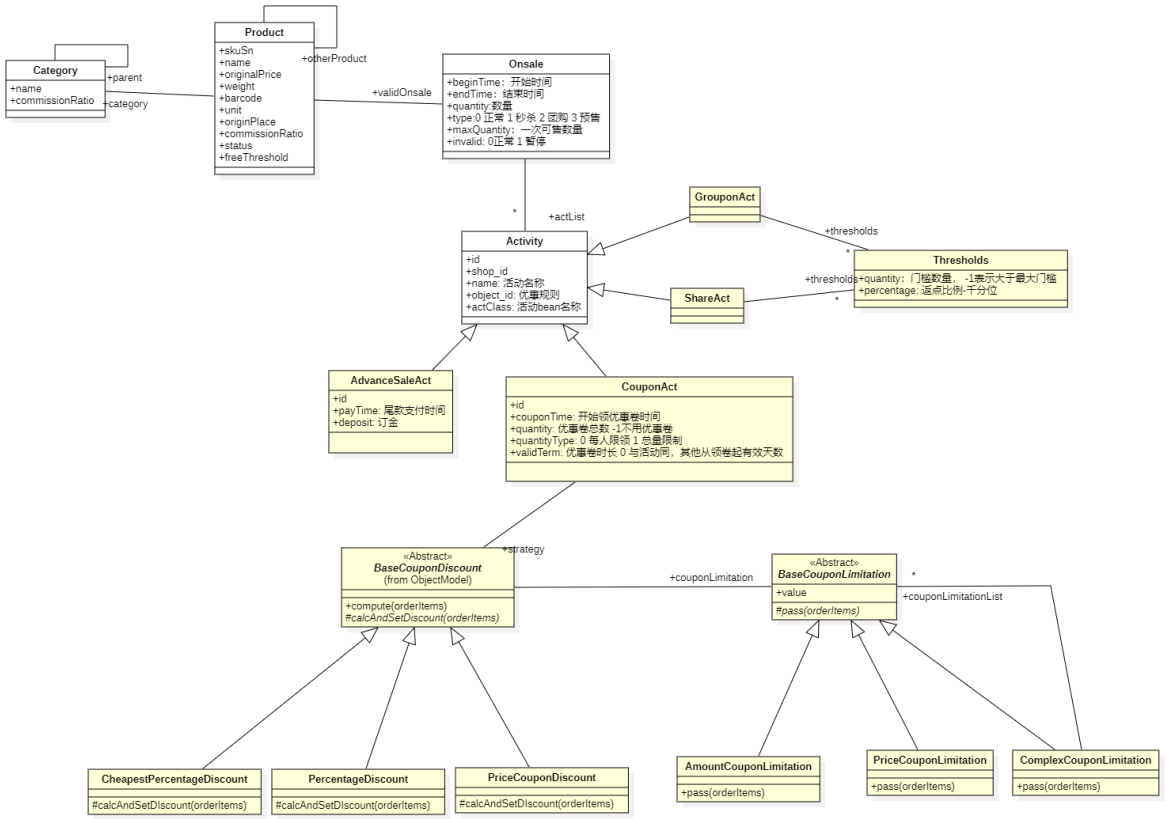
- 由业务需求产生领域模型



商品模块领域模型

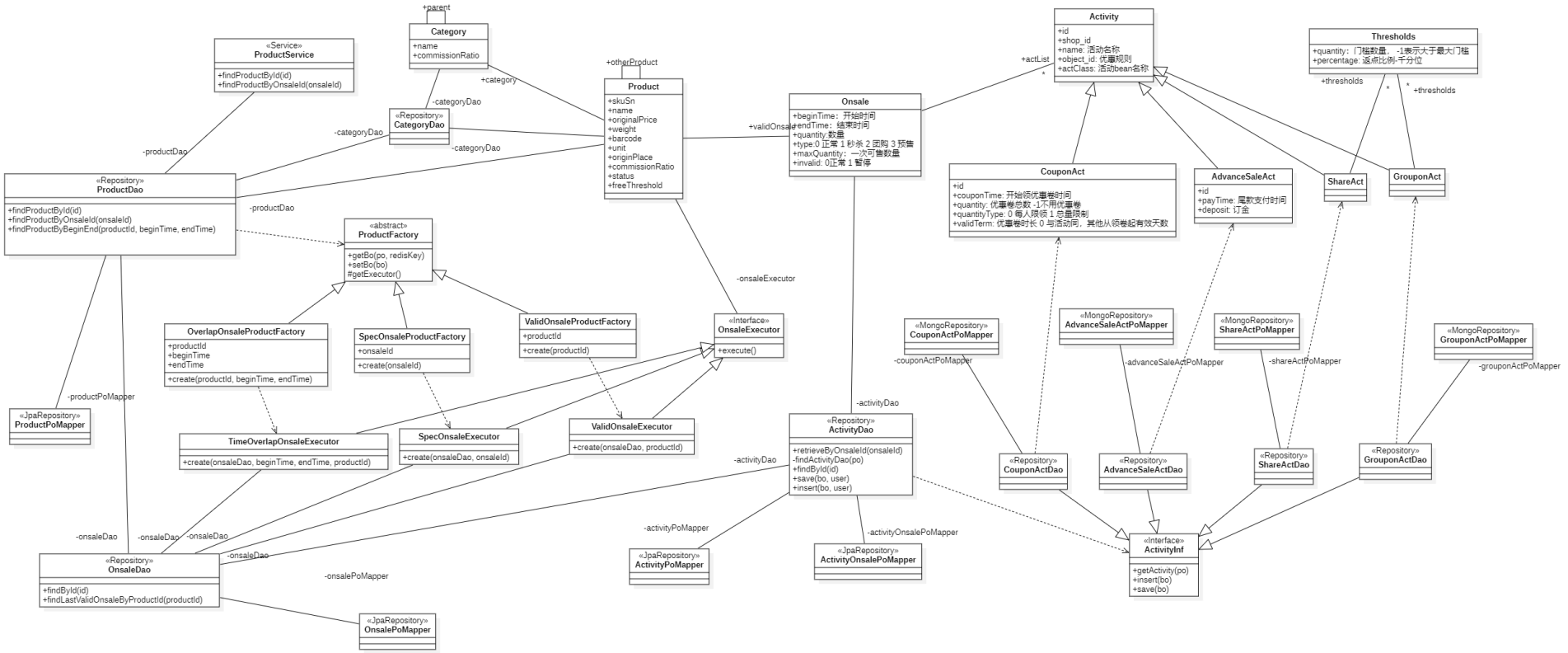
4. 面向对象编程
Object-Oriented Programming

- 由领域模型产生对象模型



商品模块业务对象模型

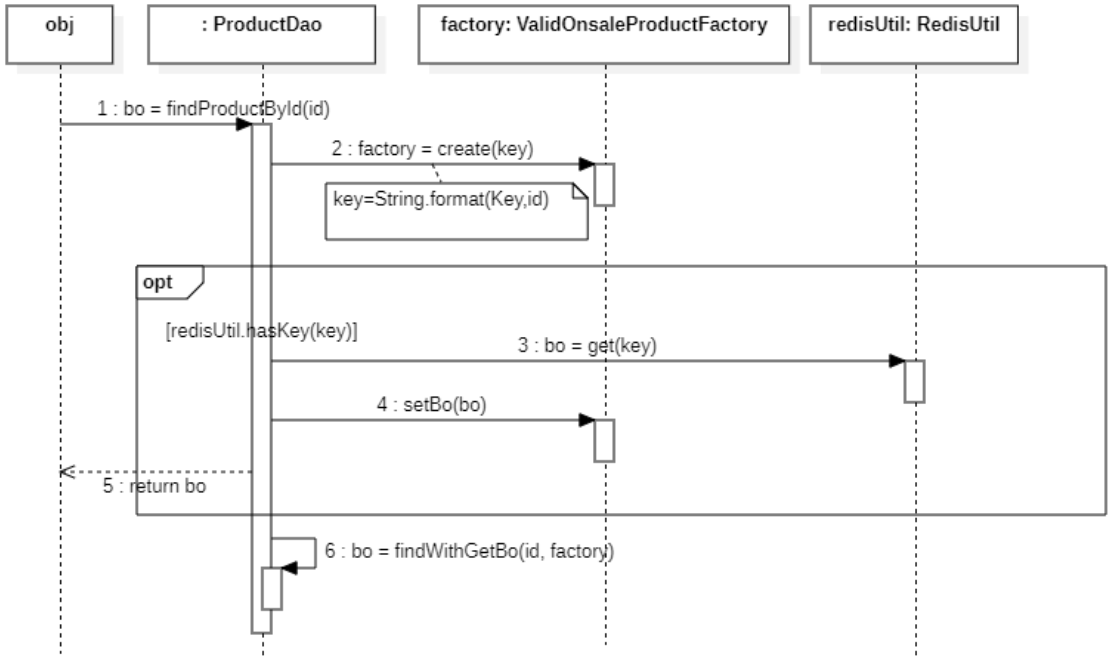
4. 面向对象编程
Object-Oriented Programming

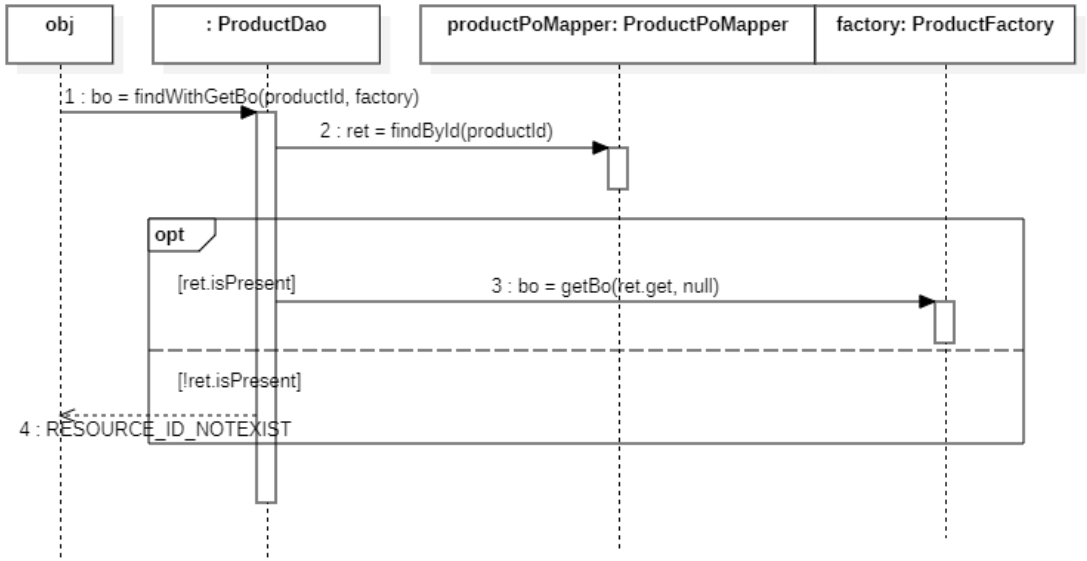


商品模块Bean对象模型

4. 面向对象编程
Object-Oriented Programming

- 运用GRASP原则将职责分配给对象





5.函数式编程
Functional Programming

- 面向函数设计
  - 依靠list，set，map等少数关键数据结构，把程序视作表达式和转换的集合，用函数实现这些表达式和转换，
    - 将程序看成表达式（Expression）和转换（Transformation）
    - 用函数完成计算和转换
    - 闭包（Closure）：函数将其所有使用的变量形成一个独立的上下文环境。
    - 惰性计算（Lazy Evaluation）

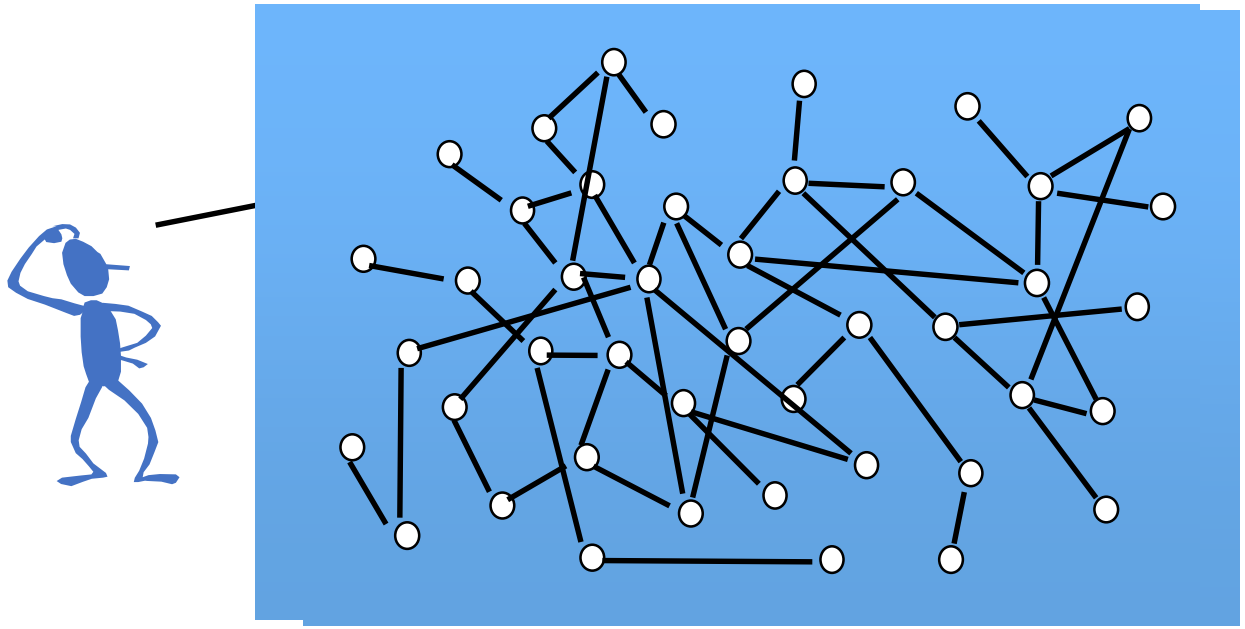
5.函数式编程
Functional Programming

- 函数式编程例子

```
/**
 * 按照活动id返回Onsale
 * @param actId 活动id
 * @param page 页数，从第1页开始
 * @param pageSize 每页数据
 * @return 返回onsale对象
 */
public List<OnSale> retrieveByActId(Long actId, Integer page, Integer pageSize) throws RuntimeException {
    logger.debug("retrieveByActId: actId = {}, page = {}, pageSize = {}",actId, page, pageSize);
    if (null == actId) {
        return null;
    }
    Pageable pageable=PageRequest.of(page - 1,pageSize);
    List<OnsalePo> actOnsalePos = this.onsalePoMapper.findByActIdEquals(actId, pageable);
    return actOnsalePos.stream().map(po -> this.build(po, Optional.empty())).collect(Collectors.toList());
}
```

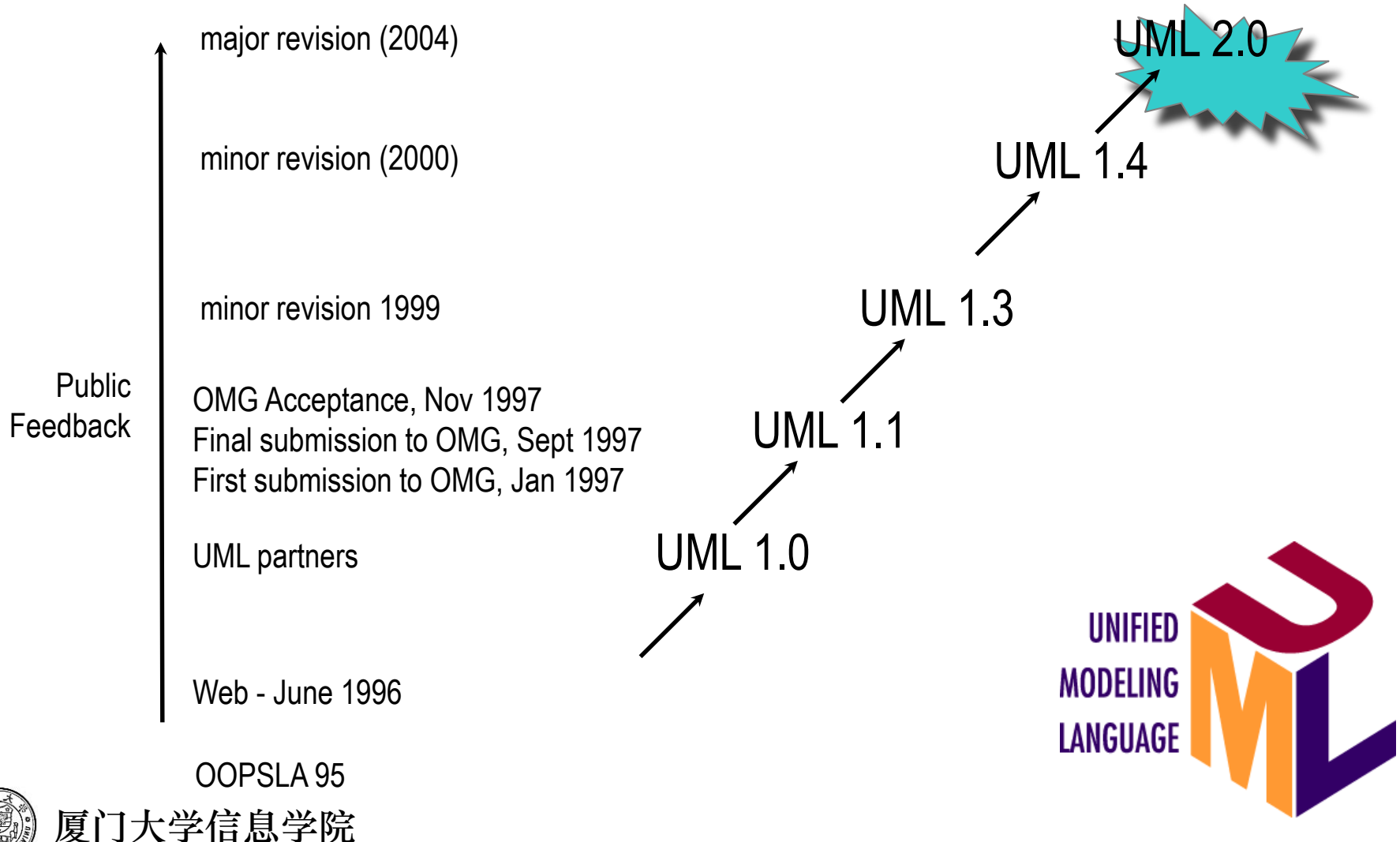
6. 统一建模语言UML
The Unified Modeling Language

是一种用图形和文字来传递信息的建模语言



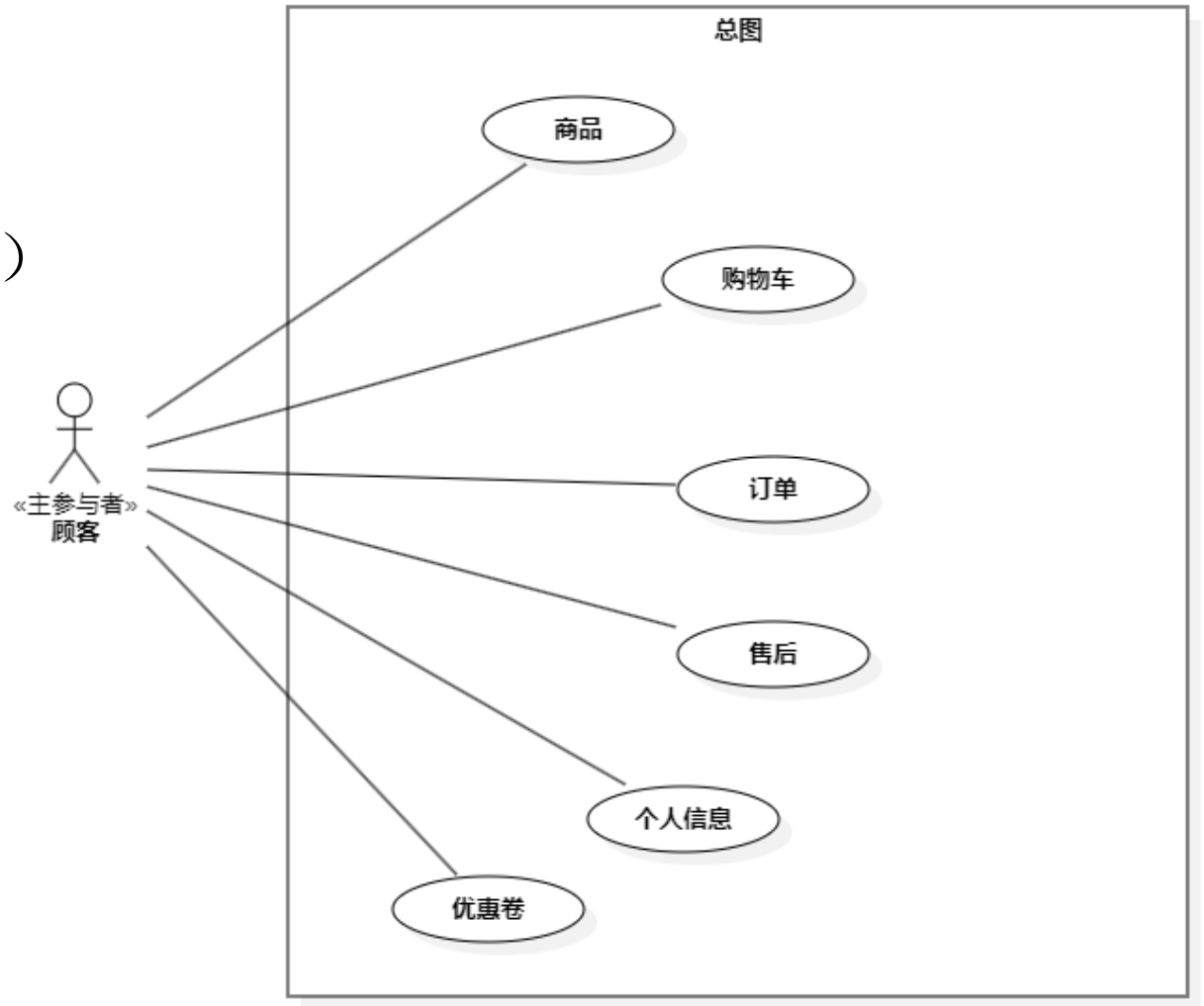
6. 统一建模语言UML

The Unified Modeling Language



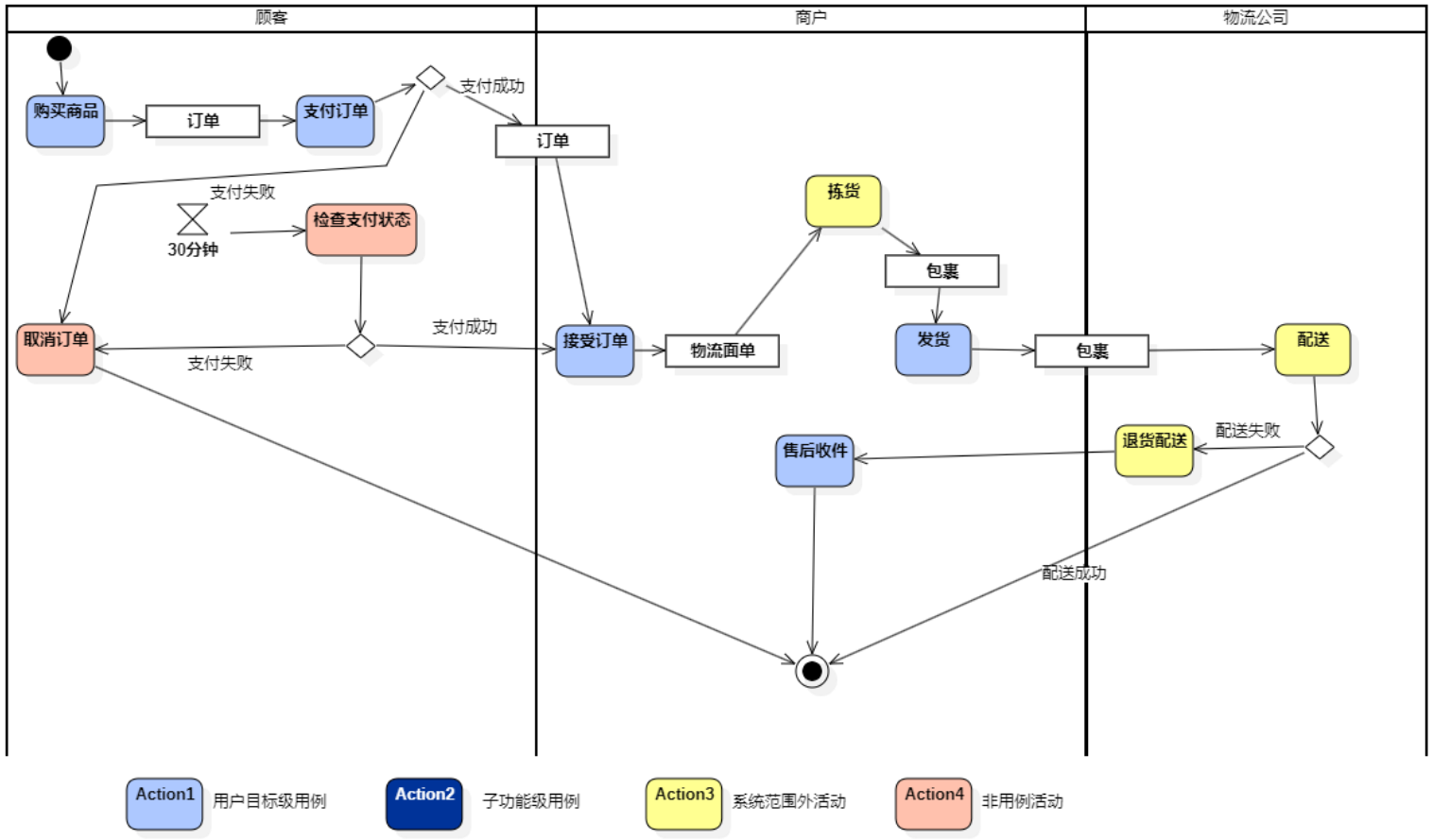
6. 统一建模语言UML
The Unified Modeling Language

- 需求分析
  - 用例图
    (Use Case Diagram)



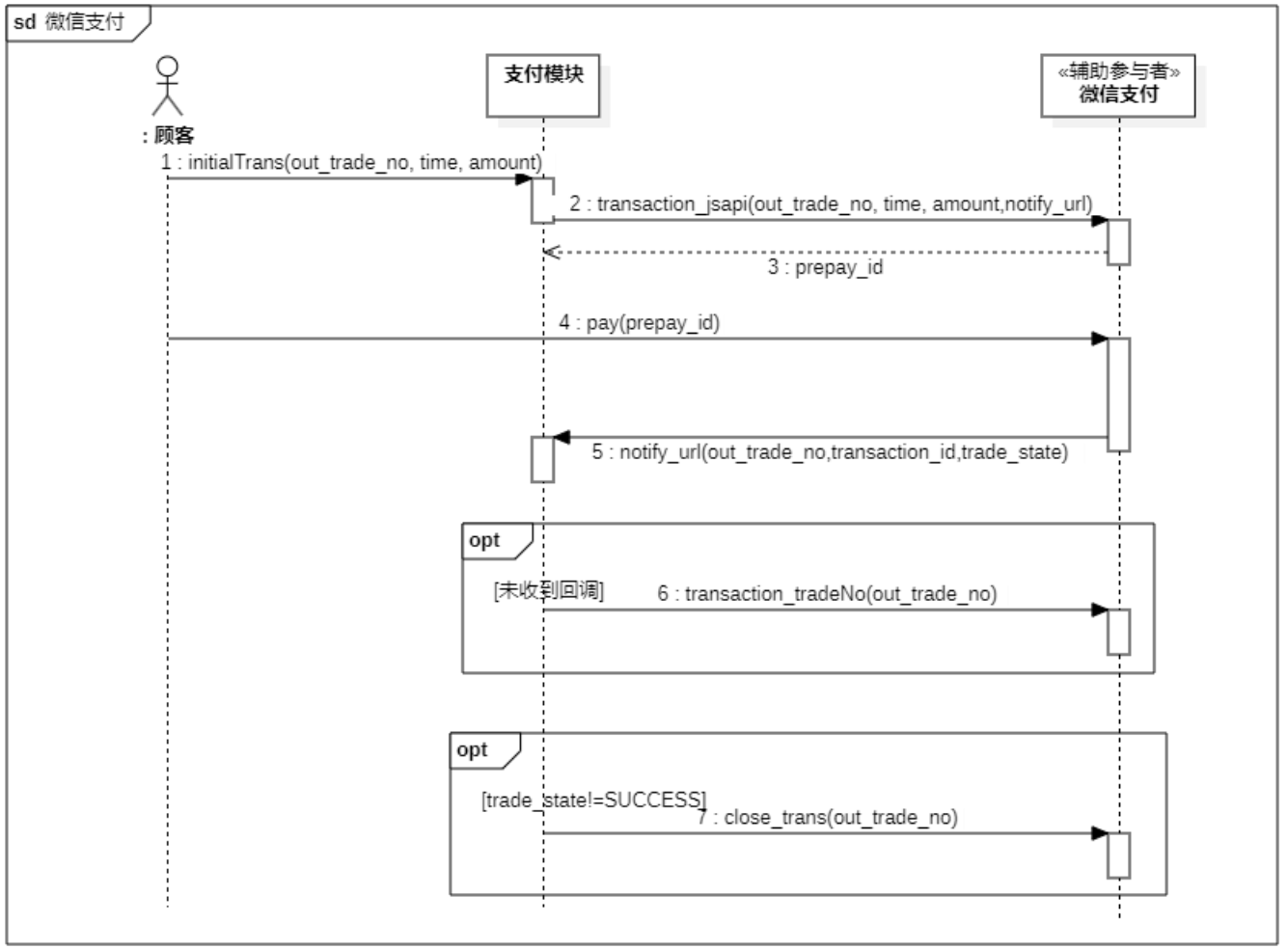
6. 统一建模语言UML
The Unified Modeling Language

- 需求分析
  - 活动图
    （Activity Diagram）



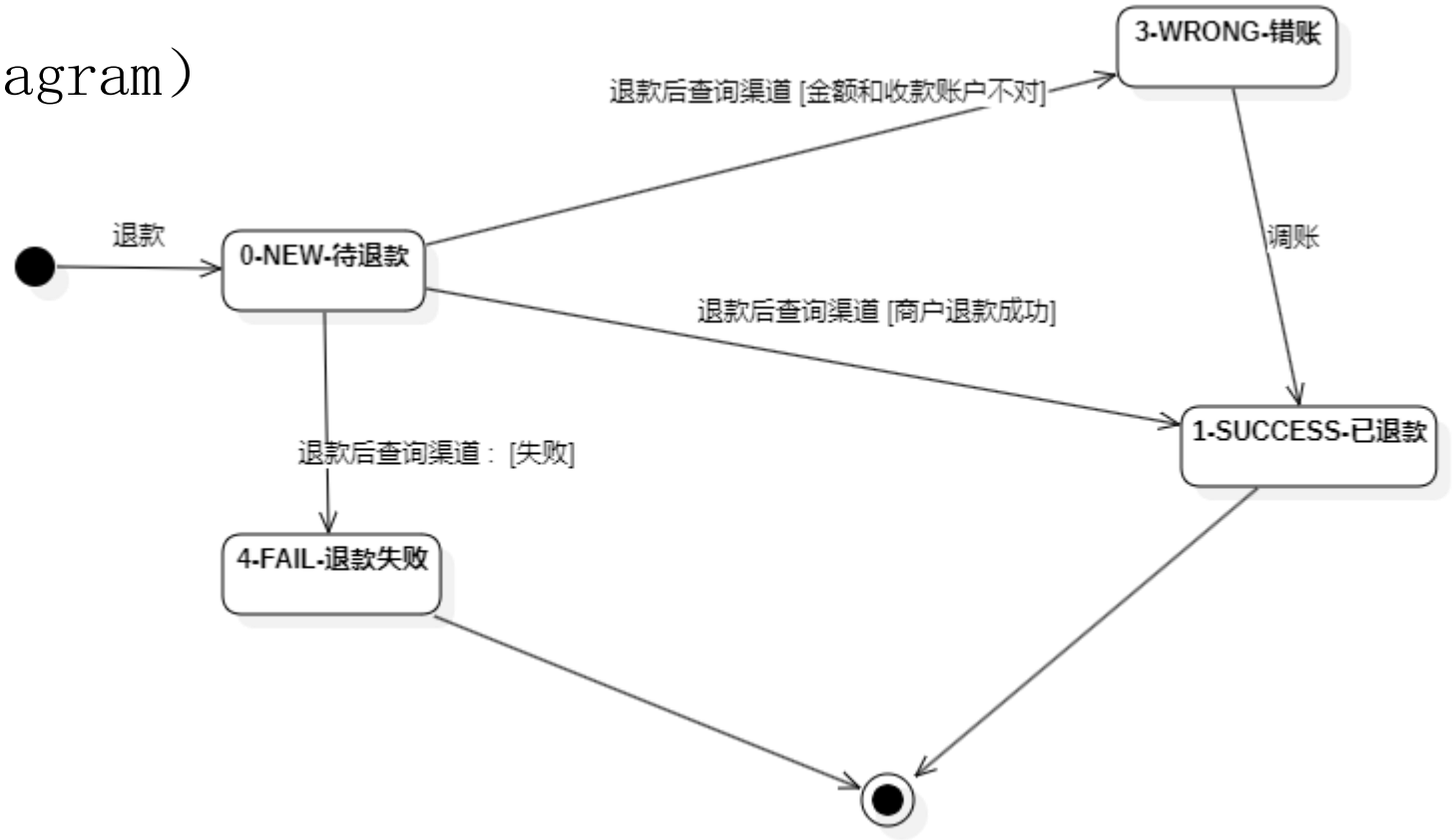
6. 统一建模语言UML
The Unified Modeling Language

- 需求分析
  - 顺序图
    （Sequence Diagram）



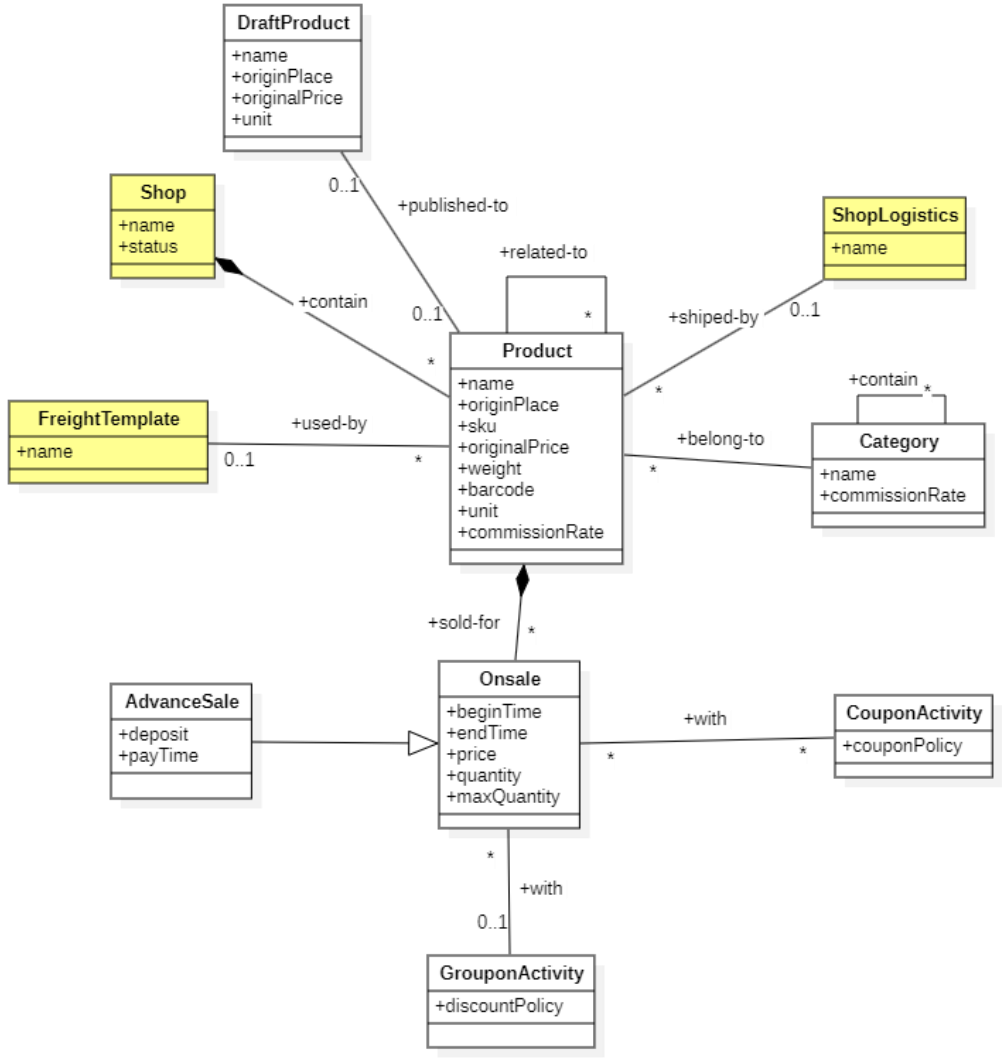
6. 统一建模语言UML
The Unified Modeling Language

- 需求分析
  - 状态机图
    （State Machine Diagram）



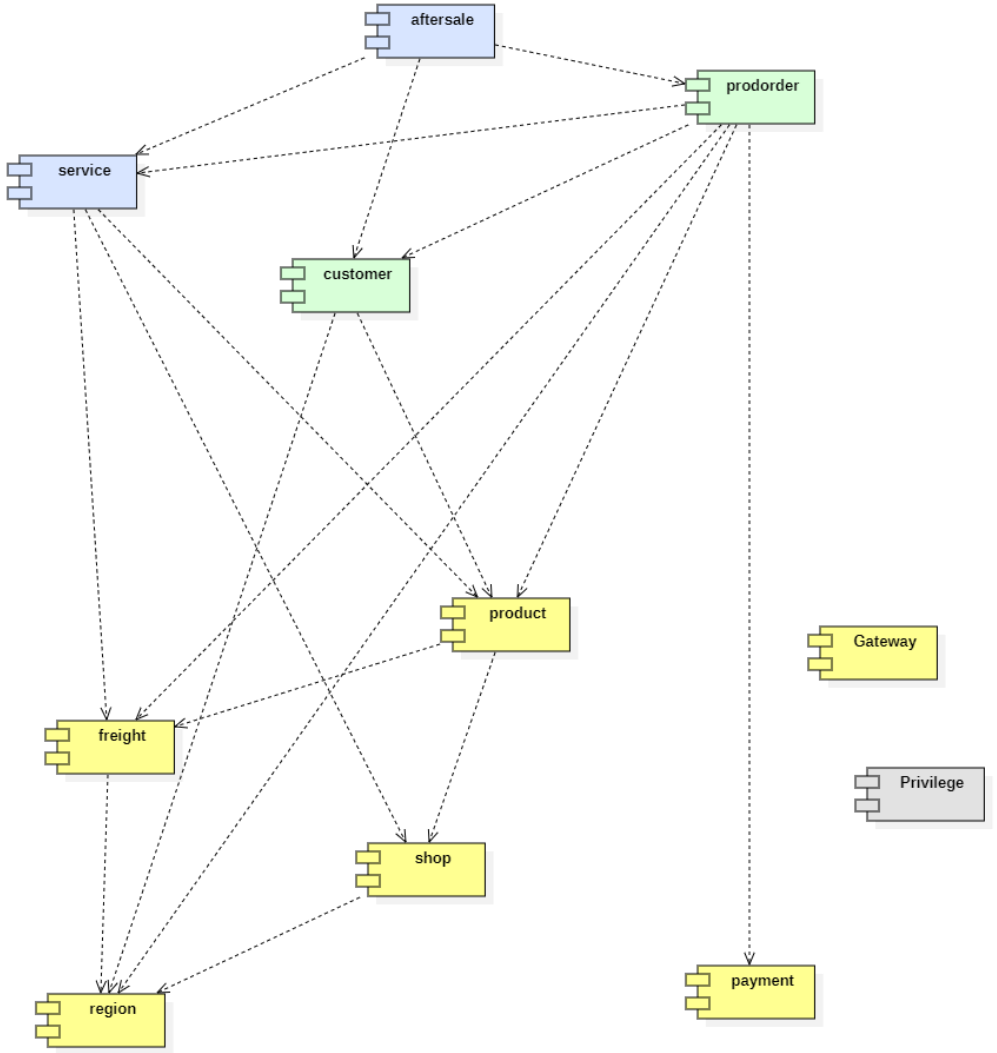
6. 统一建模语言UML
The Unified Modeling Language

- 领域建模
  - 类图
    （Class Diagram）



6. 统一建模语言UML
The Unified Modeling Language

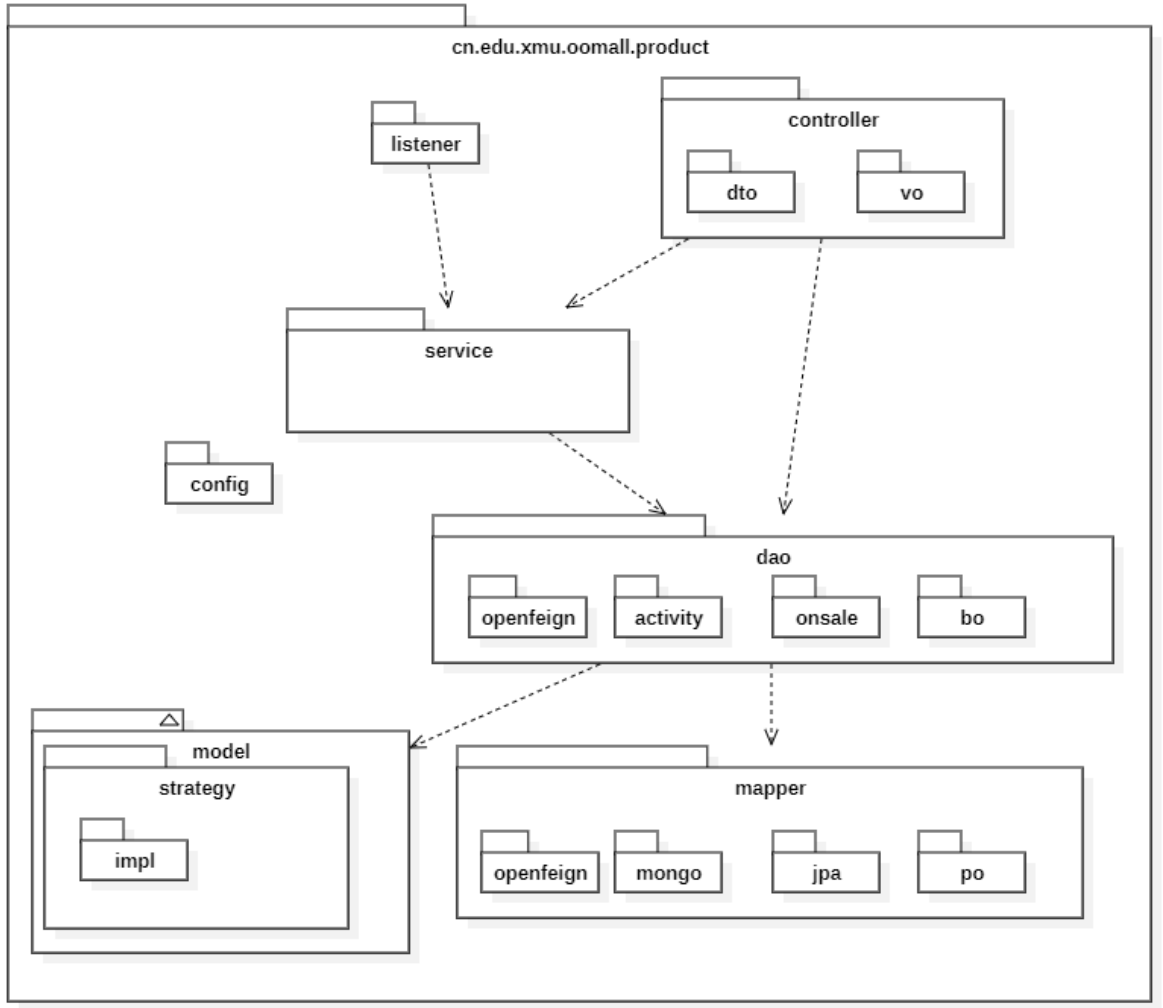
- 结构设计
  - 组件图
    （Component Diagram）



6. 统一建模语言UML
The Unified Modeling Language

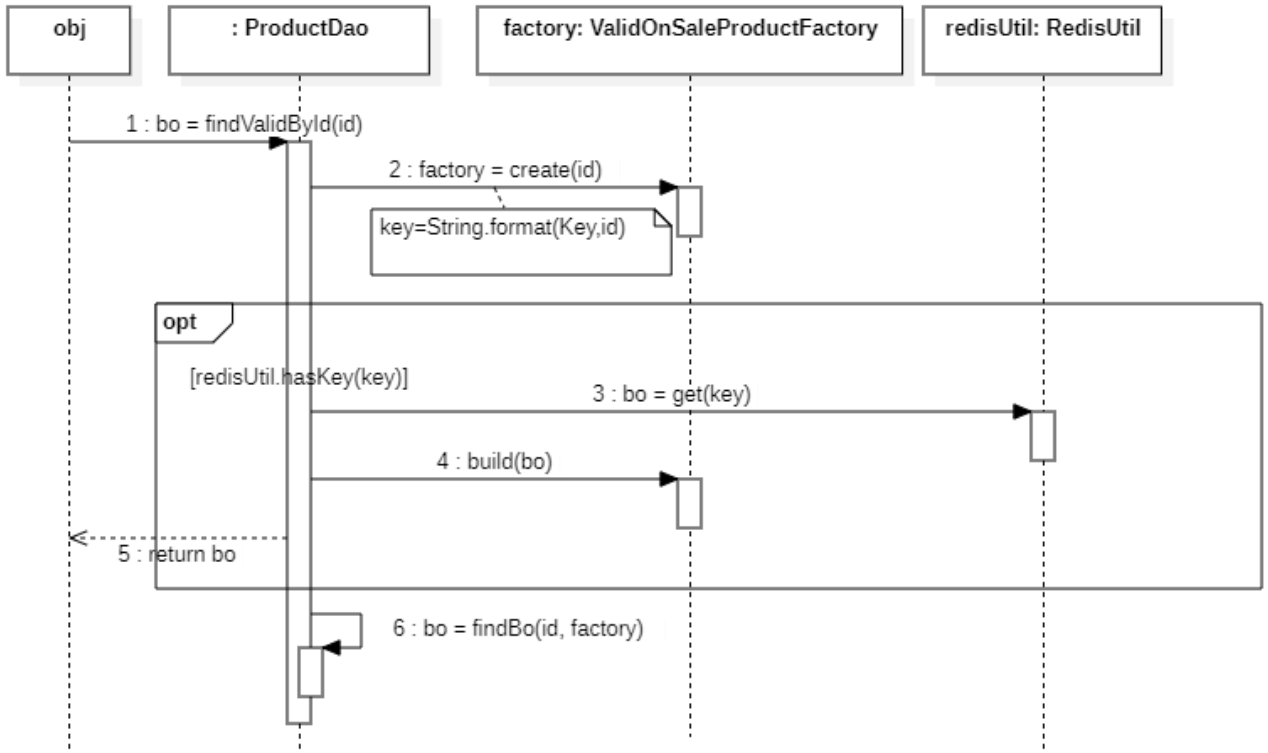
- 结构设计
  - 包图

  Package Diagram



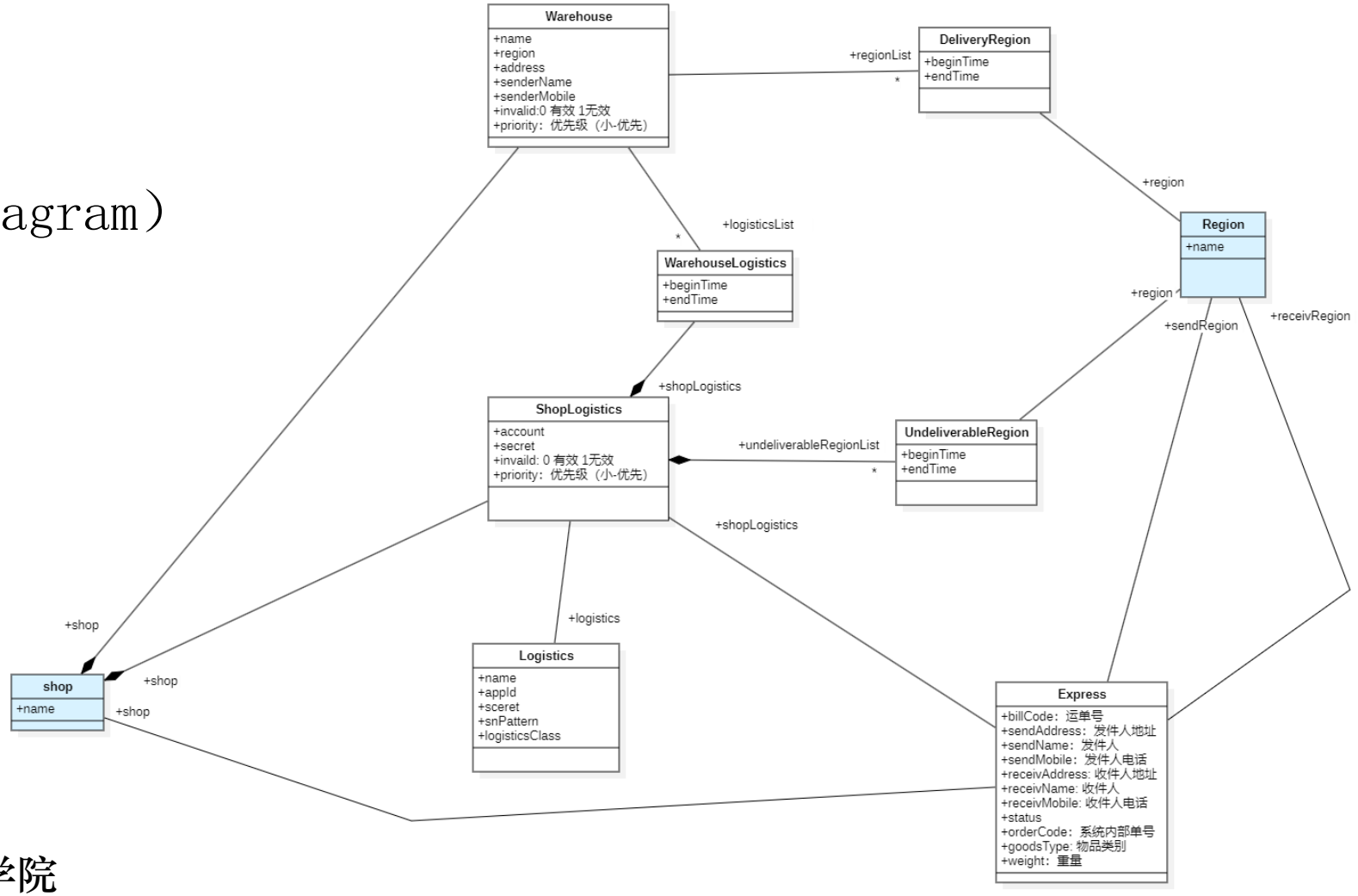
6. 统一建模语言UML
The Unified Modeling Language

- 交互设计
  - 顺序图（Sequence Diagram）



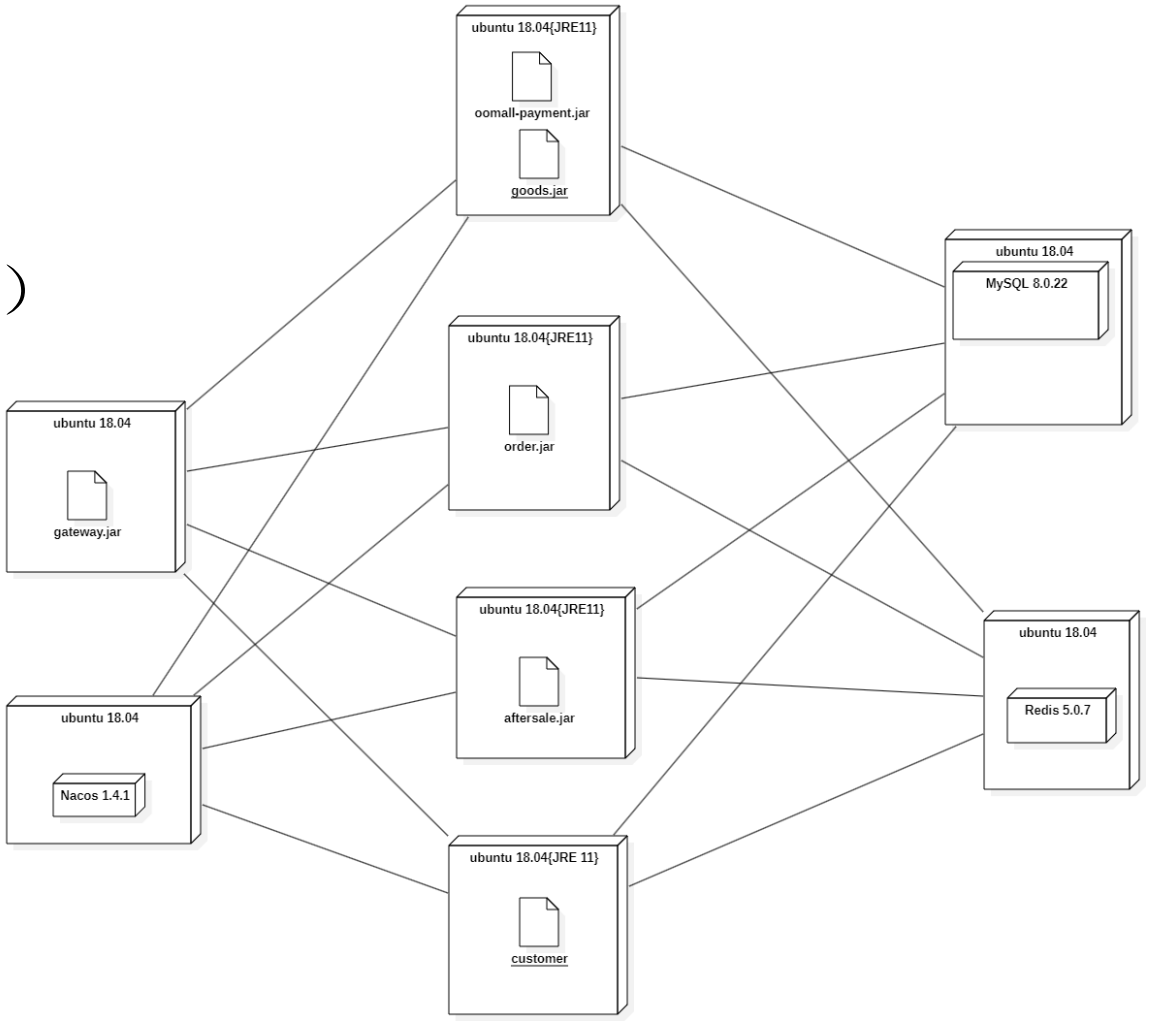
6. 统一建模语言UML
The Unified Modeling Language

- 定义类图
  - 类图
    （Class Diagram）



6. 统一建模语言UML
The Unified Modeling Language

- 部署运维
  - 部署图
    （Deployment Diagram）



6. 统一建模语言UML
The Unified Modeling Language

- UML的三种用法
  - 用UML做草图
  - 用UML做蓝图　(反向工程/代码生成)
  - 用UML做编程语言 (MDA)
- UML主要的工具
  - Rational Rose
  - PowerDesigner
  - MS Visio
  - ArgoUML
  - StarUML