面向对象分析与设计

Object Oriented Analysis and Design

——概要设计

Architectural Design

邱明 博士

厦门大学信息学院

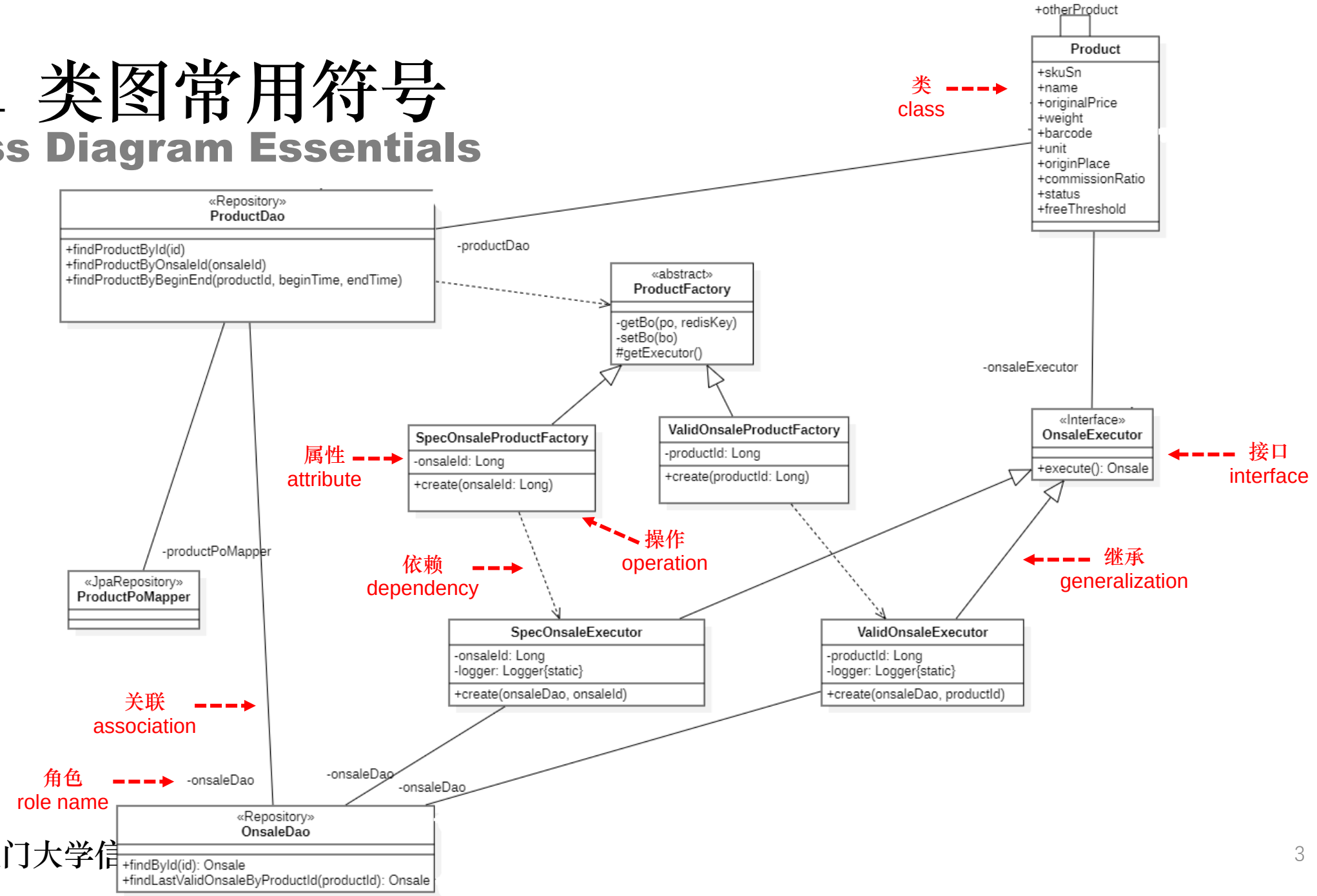
mingqiu@xmu.edu.cn

2023年秋季学期

1. UML类图和包图

UML Class Diagram and Package Diagram

1.1 类图常用符号
Class Diagram Essentials



1.1 类图常用符号
Class Diagram Essentials

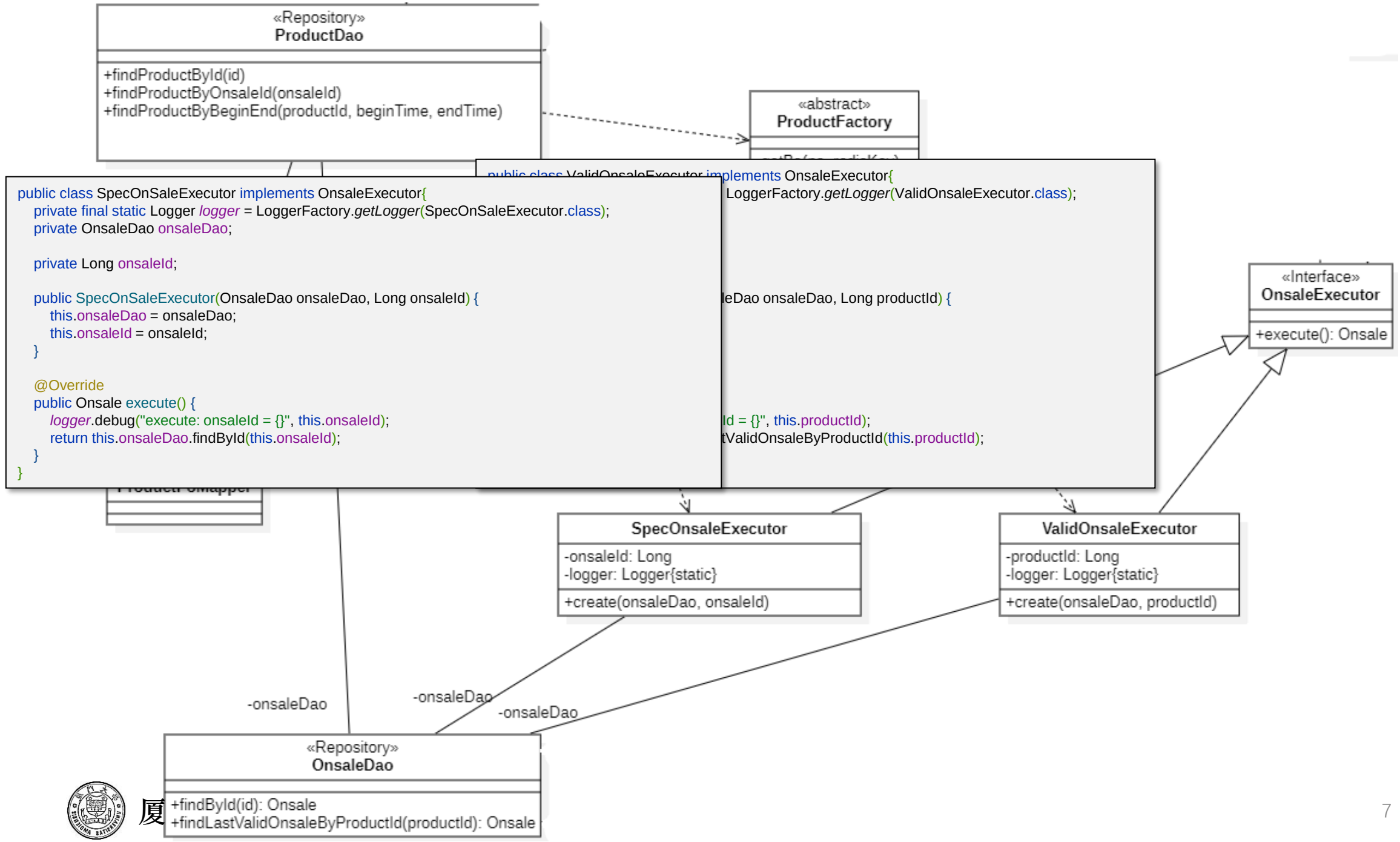
- 属性（Attributes）
  - 类的原始数据类型属性
  - visibility name : type multiplicity = default {property-string}
  - - onsaleId: Long [1] = null {auto-incr}
  - Long onsaleId;

1.1 类图常用符号
Class Diagram Essentials

- 操作（Operation）
  - visibility name (parameter-list) : return-type {property-string}
  - + getPlayer(name : String) : Player {exception IOException}
  - public Player getPlayer(String name) throws IOException;
  - Create用于描述创建对象（new）
  - get/set方法可以省略

1.2 元素间的关系
Relationship

- 关联（Association）
  - 自定义的类型的属性
- 继承（Generalization）
  - 元素之间的继承关系
- 依赖关系（Dependency）
  - 一个对象的修改会影响另一个对象

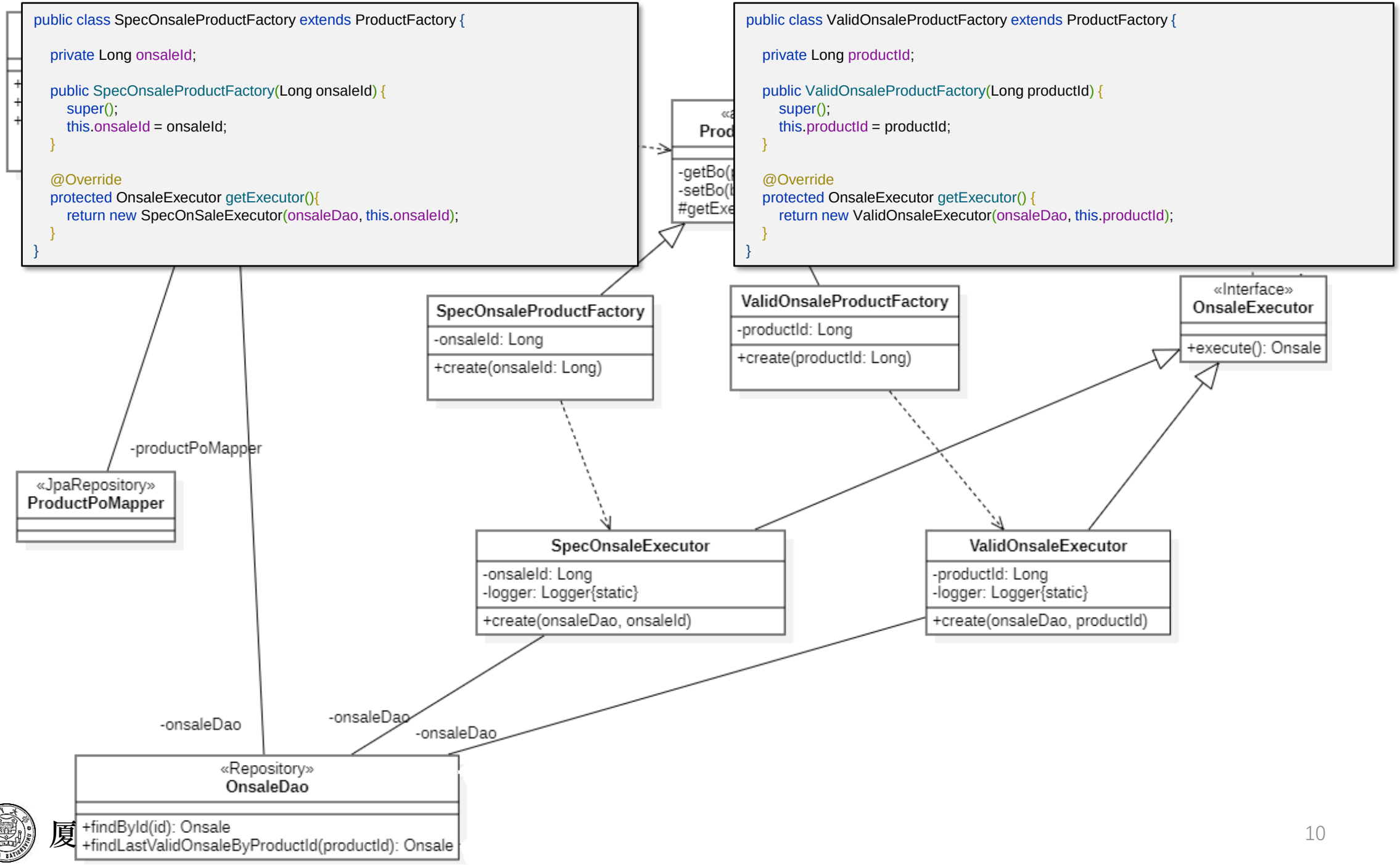

«Repository»
ProductDao
+findProductById(id)
+findProductByOnsaleId(onsaleId)
+findProductByBeginEnd(productId, beginTime, endTime)
«abstract»
ProductFactory
public class SpecOnSaleExecutor implements OnsaleExecutor{
private final static Logger logger = LoggerFactory.getLogger(SpecOnSaleExecutor.class);
private OnsaleDao onsaleDao;
private Long onsaleId;
public SpecOnSaleExecutor(OnsaleDao onsaleDao, Long onsaleId) {
this.onsaleDao = onsaleDao;
this.onsaleId = onsaleId;
}
@Override
public Onsale execute() {
logger.debug("execute: onsaleId = {}", this.onsaleId);
return this.onsaleDao.findById(this.onsaleId);
}
}
LoggerFactory.getLogger(ValidOnsaleExecutor.class);
«Interface»
OnsaleExecutor
+execute(): Onsale
SpecOnsaleExecutor
-onsaleId: Long
-logger: Logger{static}
+create(onsaleDao, onsaleId)
ValidOnsaleExecutor
-productId: Long
-logger: Logger{static}
+create(onsaleDao, productId)
-onsaleDao
-onsaleDao
-onsaleDao
«Repository»
OnsaleDao
+findById(id): Onsale
+findLastValidOnsaleByProductId(productId): Onsale

1.3 泛型和标签
Stereotype and tag

- 泛型（Stereotype）
  - {« »}.对现有UML元素的重新定义
- 标签 tag
  - 用于描述UML元素的性质 {name1=value1, name2=value2}
- 抽象类和方法用 {abstract}tag表示
- Final类用{leaf} tag

1.4 四种依赖关系
Four Kinds of Dependency

- 依赖关系
  - 全局变量
  - 方法的参数
  - 局域变量
  - 静态方法

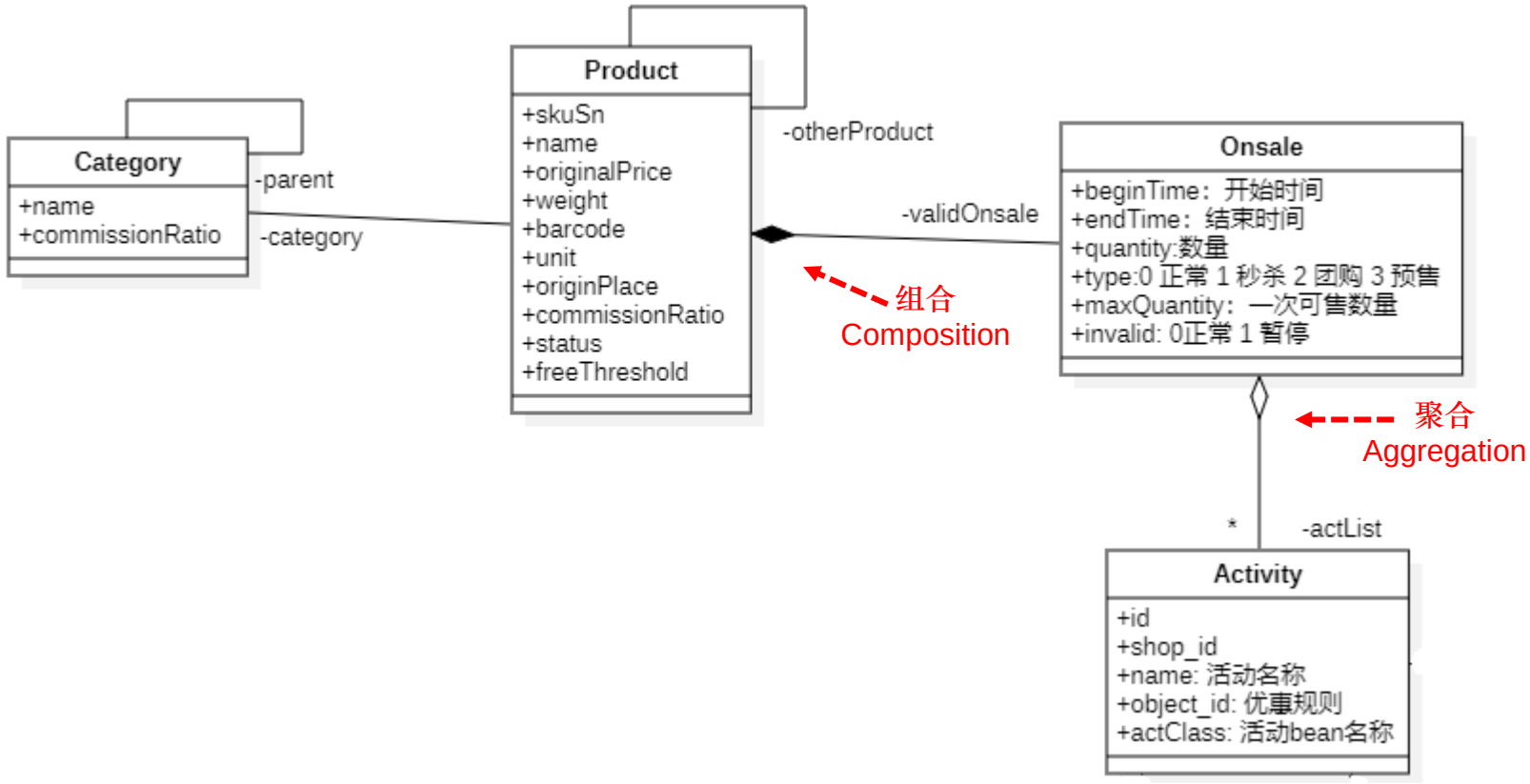

public class SpecOnsaleProductFactory extends ProductFactory {
private Long onsaleId;
public SpecOnsaleProductFactory(Long onsaleId) {
super();
this.onsaleId = onsaleId;
}
@Override
protected OnsaleExecutor getExecutor(){
return new SpecOnSaleExecutor(onsaleDao, this.onsaleId);
}
}
public class ValidOnsaleProductFactory extends ProductFactory {
private Long productId;
public ValidOnsaleProductFactory(Long productId) {
super();
this.productId = productId;
}
@Override
protected OnsaleExecutor getExecutor() {
return new ValidOnsaleExecutor(onsaleDao, this.productId);
}
}
SpecOnsaleProductFactory
-onsaleId: Long
+create(onsaleId: Long)
ValidOnsaleProductFactory
-productId: Long
+create(productId: Long)
«Interface»
OnsaleExecutor
+execute(): Onsale
-productPoMapper
«JpaRepository»
ProductPoMapper
SpecOnsaleExecutor
-onsaleId: Long
-logger: Logger{static}
+create(onsaleDao, onsaleId)
ValidOnsaleExecutor
-productId: Long
-logger: Logger{static}
+create(onsaleDao, productId)
-onsaleDao
-onsaleDao
-onsaleDao
«Repository»
OnsaleDao
+findById(id): Onsale
+findLastValidOnsaleByProductId(productId): Onsale

1.5 聚合和组合
Aggregation and Composition

- 聚合 （Aggregation）
  - 普通的一对多的关联关系.
- 组合 （Composition）
  - 整体和局部的关系
  - 局部对象只能属于一个母体对象
  - 局部对象必属于母体对象
  - 母体对象负责创建和销毁局部对象

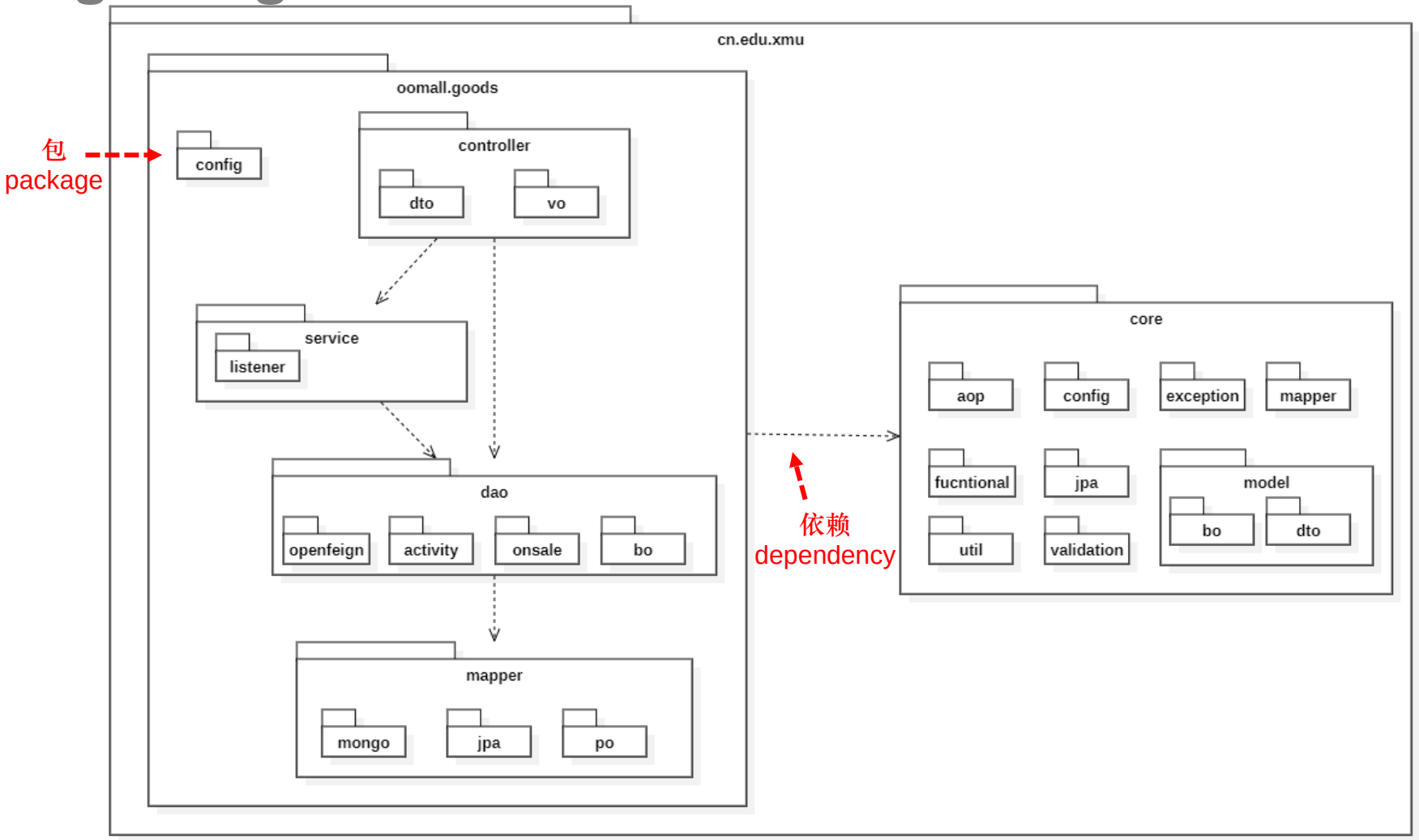
1.5 聚合和组合

Aggregation and Composition



1.6 包图常用符号

Package Diagram Essentials



2. 领域建模

Domain Modeling

2.1 领域模型
Domain Model

- 描述需求中存在的概念以及概念之间的关系，也被称为概念模型（Conceptual Model）
- 用UML类图描述需求中的概念和概念之间的关联
  - 也称可视字典（Visual Dictionary）

2.1 领域模型
Domain Model

- 描述需求中存在的概念以及概念之间的关系，也被称为概念模型（Conceptual Model）
- 以类图描述以下内容
  - 领域对象或概念类
  - 概念之间的关联
  - 概念的属性

2.1 领域模型
Domain Model

- 领域中的概念

符号（Symbol）

Product是电子商城销售的商品，它有名称、产地、价格、重量、条码和单位等属性。

内涵（Intension）

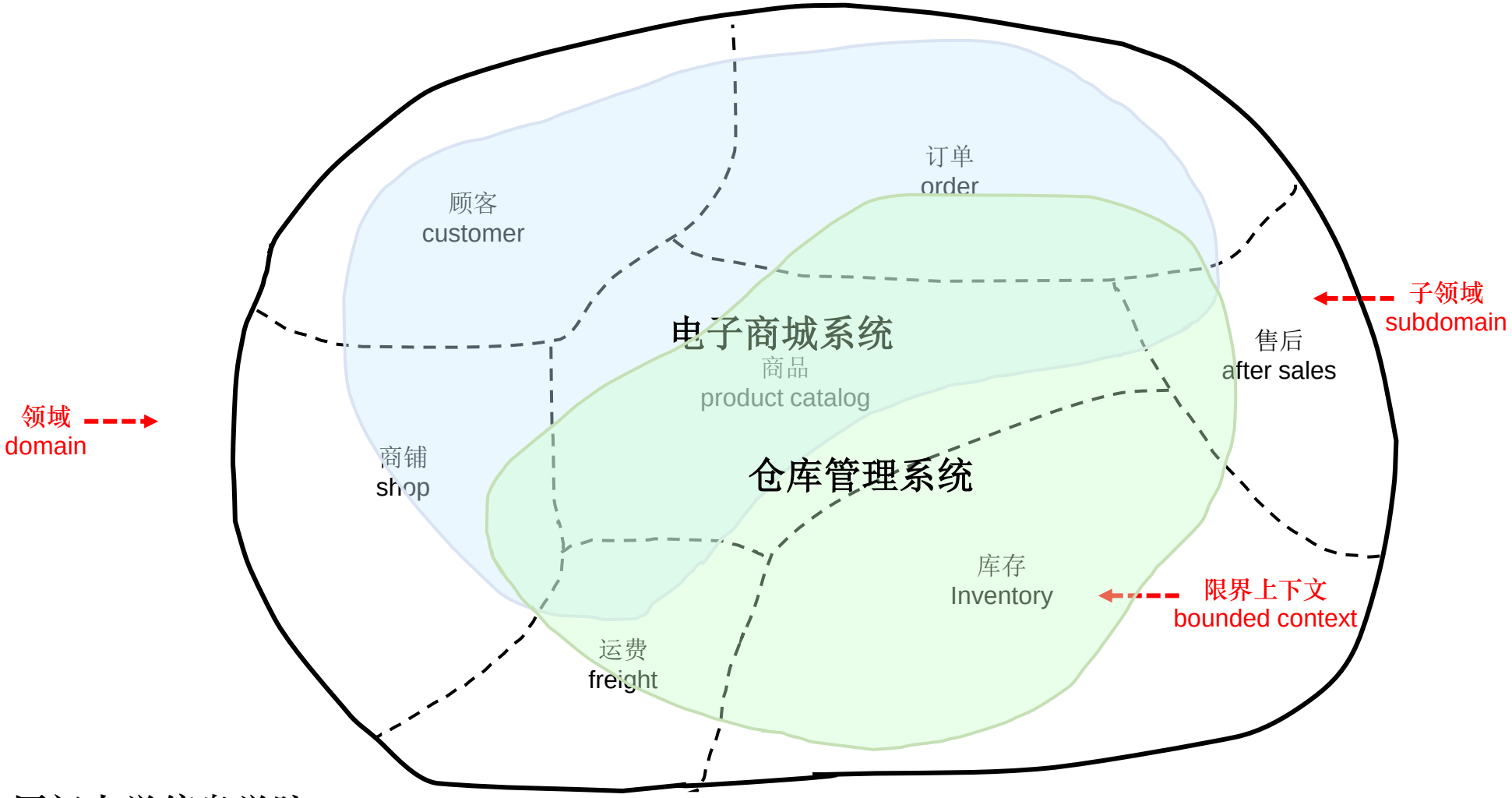
外延（Extension）

2.1 领域模型
Domain Model

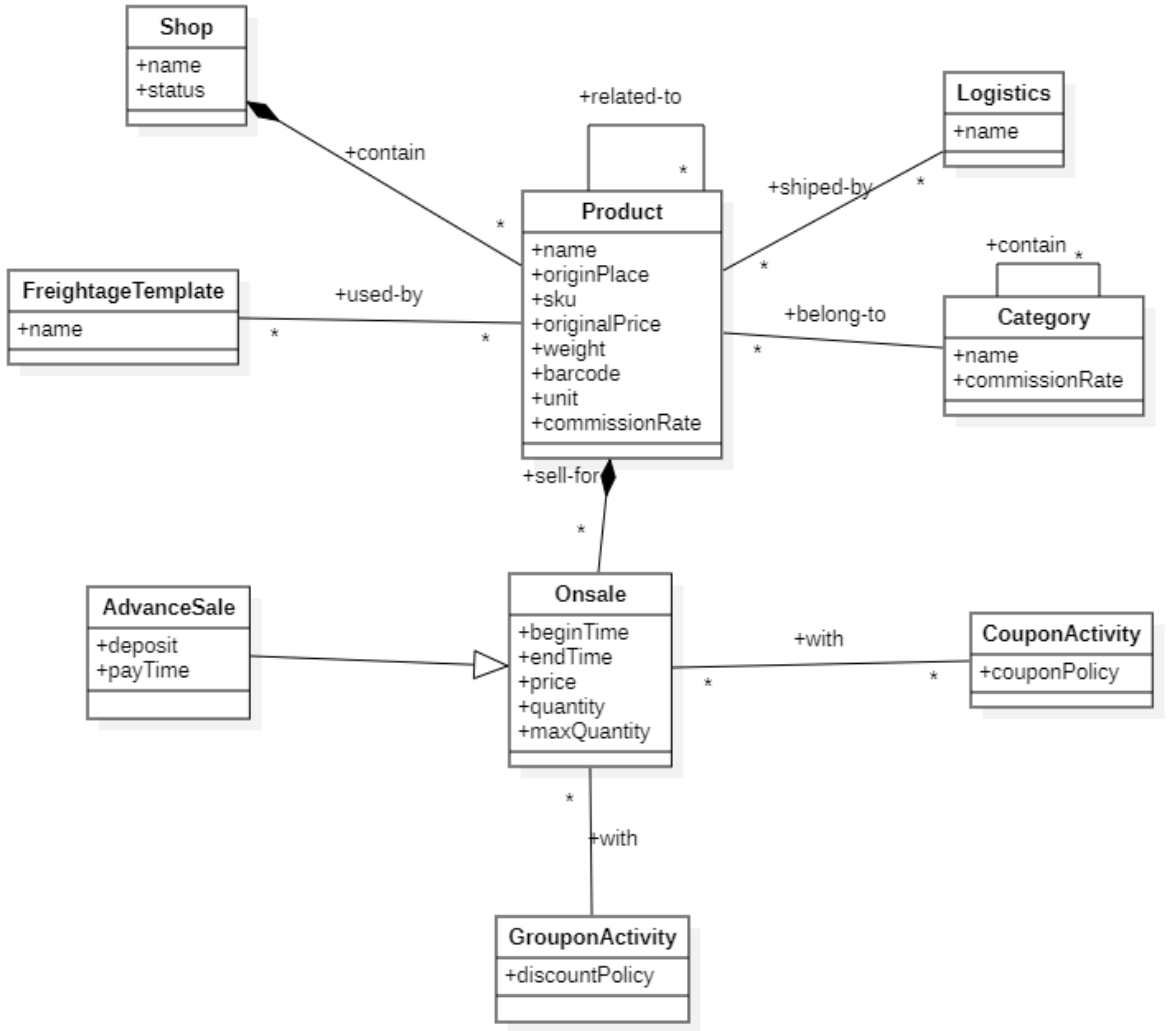
- 领域模型的作用
  - 理解需求
  - 发现需求背后的逻辑关系
  - 是对象模型的中间产品

2.2 领域、子领域和限界上下文

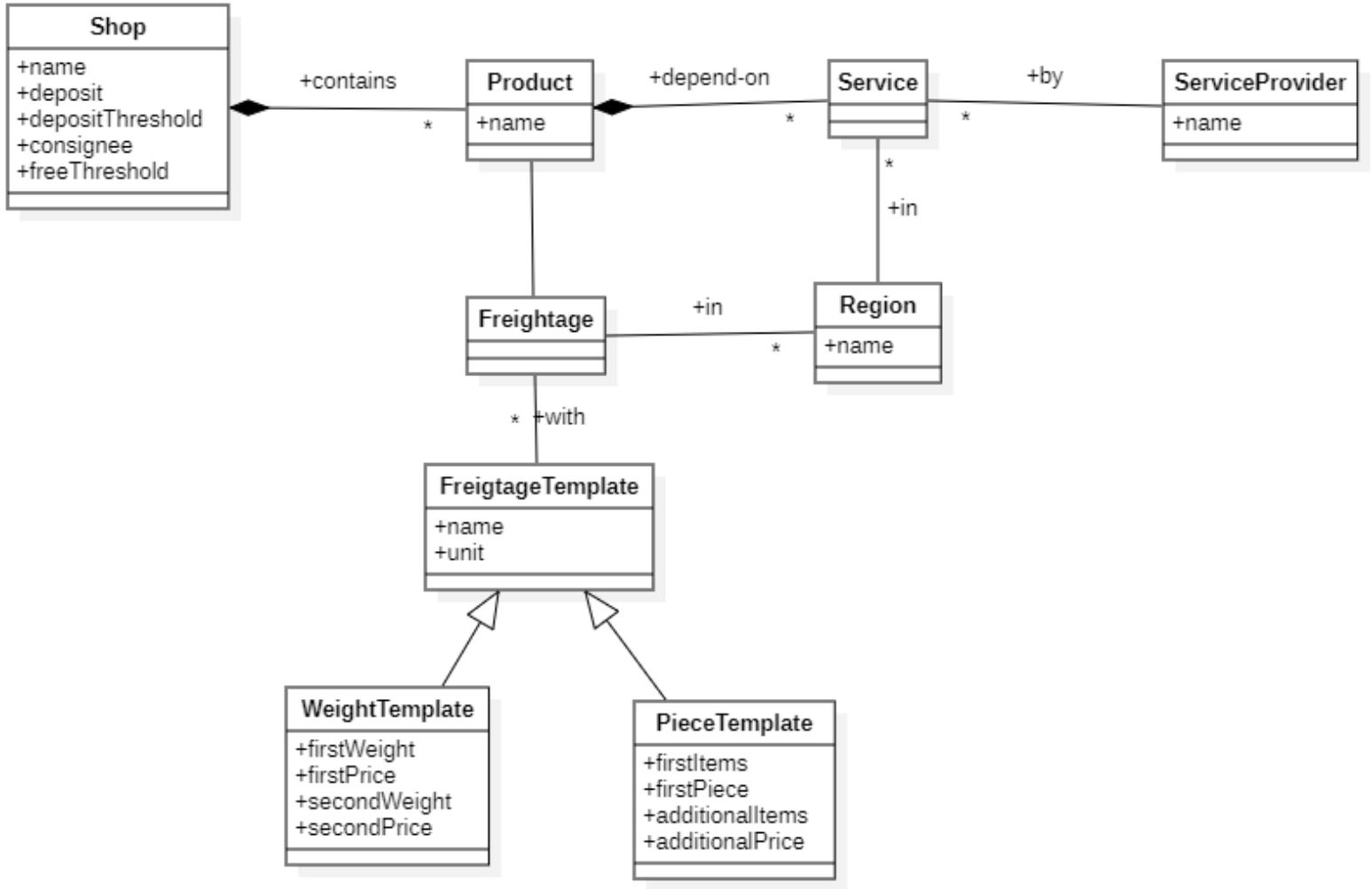
Domain, Subdomain and Bounded Context



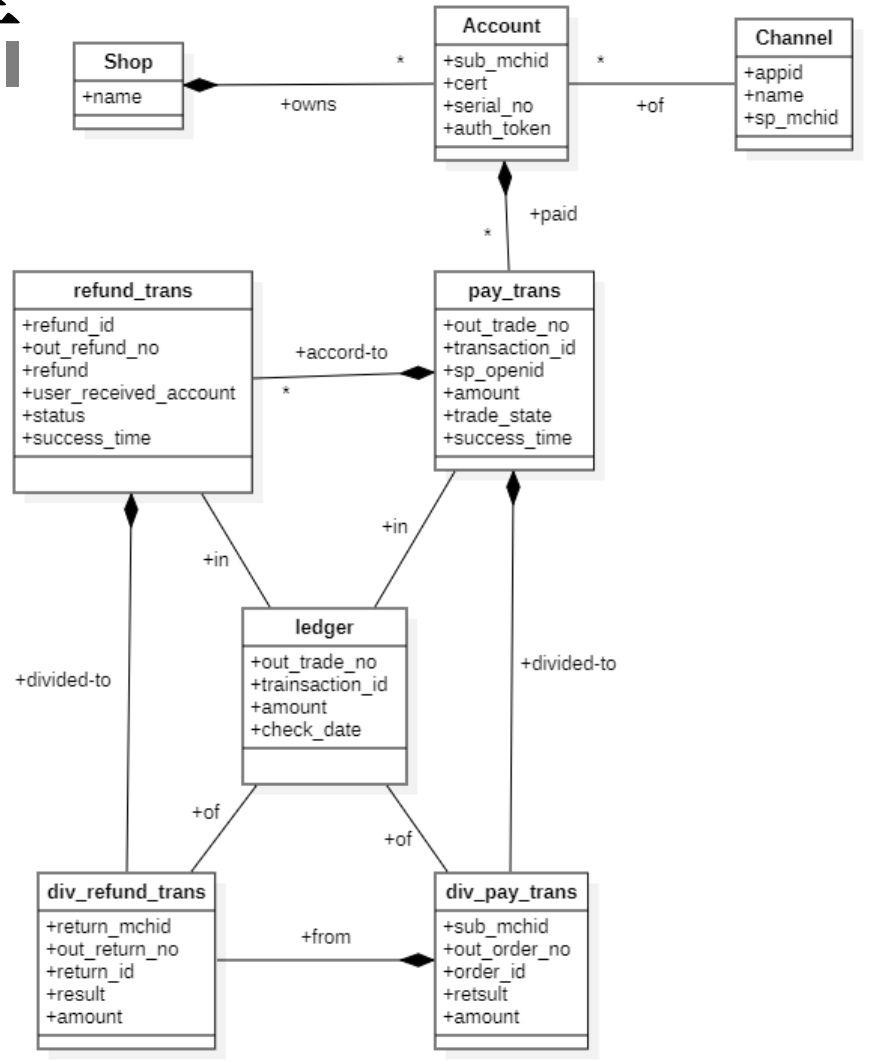
2.3 产品子领域模型
Product Subdomain Model



2.4 商铺子领域模型
Shop Subdomain Model



2.5 支付子领域模型
Payment Subdomain Model



2.5 支付子领域模型
Payment Subdomain Model

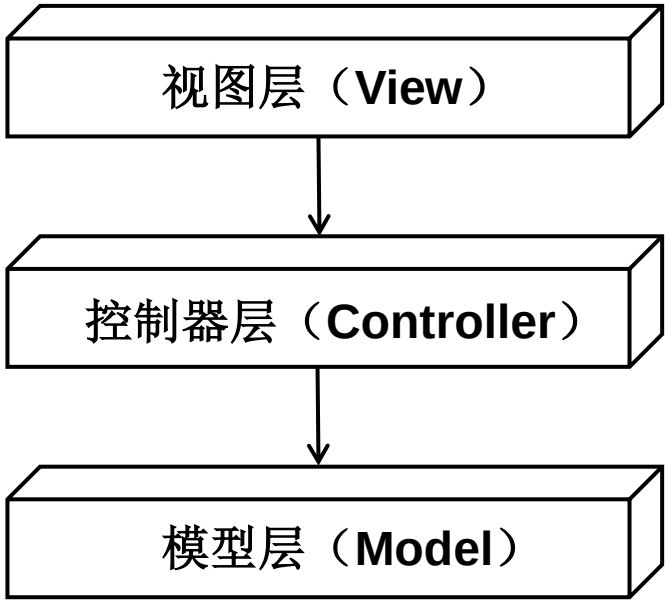
微信	支付宝	描述
appid：服务商应用ID	app_id: 支付宝分配给开发者的应用ID	
sp_mchid：服务商户号		
sub_mchid：子商户号	seller_id: 收款支付宝账号	
out_trade_no:商户订单号	out_trade_no:商户订单号	商户系统内部订单号，只能是数字、大小写字母_
amount：订单金额	total_amount:金额	单位为分
transaction_id: 微信支付订单号	trade_no:支付宝交易号	
trade_state: 交易状态	trade_status: 交易状态	
sp_openid: 支付用户openid	buyer_logon_id：买家支付宝账号	
refund_id: 支付退款单号	trade_no	
out_refund_no: 商户系统内部的退款单号	out_trade_no	
refund: 退款金额	refund_amount: 退款金额	
channel:退款渠道	fund_channel:资金渠道	
user_received_account：退款入账户	buyer_user_id：买家支付宝id	
success_time：退款成功时间	gmt_refund_pay:退款时间	
success_time：支付成功时间	send_pay_date：本次交易打款给卖家的时间	
return_mchid: 回退商户号		
out_order_no: 商户分账单号	out_request_no: 确认结算请求流水号	
out_return_no: 商户回退单号		
return_id: 微信回退单号	trade_no: 支付宝交易号	
order_id: 微信分账单号	trade_no: 支付宝交易号	

3. 体系结构

Architecture

3.1 层次体系结构
Layers Architecture

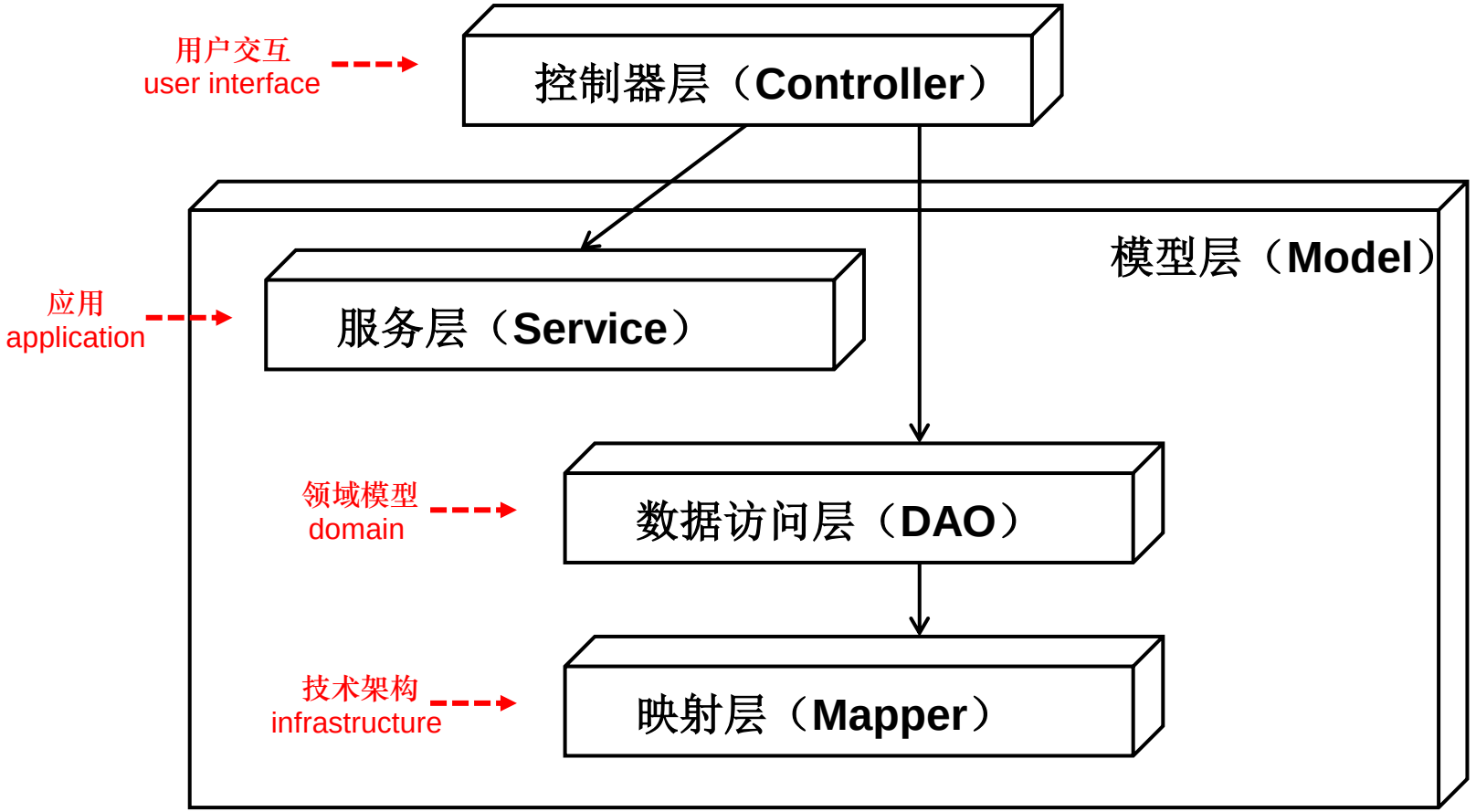
- 系统的逻辑结构
  - 大粒度的系统划分方式，每层具备清晰的职责，具备内聚性
  - 层内耦合大，层间耦合小
    - 变更控制在一层之内
    - 不同的逻辑可以分开



3.1 层次体系结构
Layers Architecture

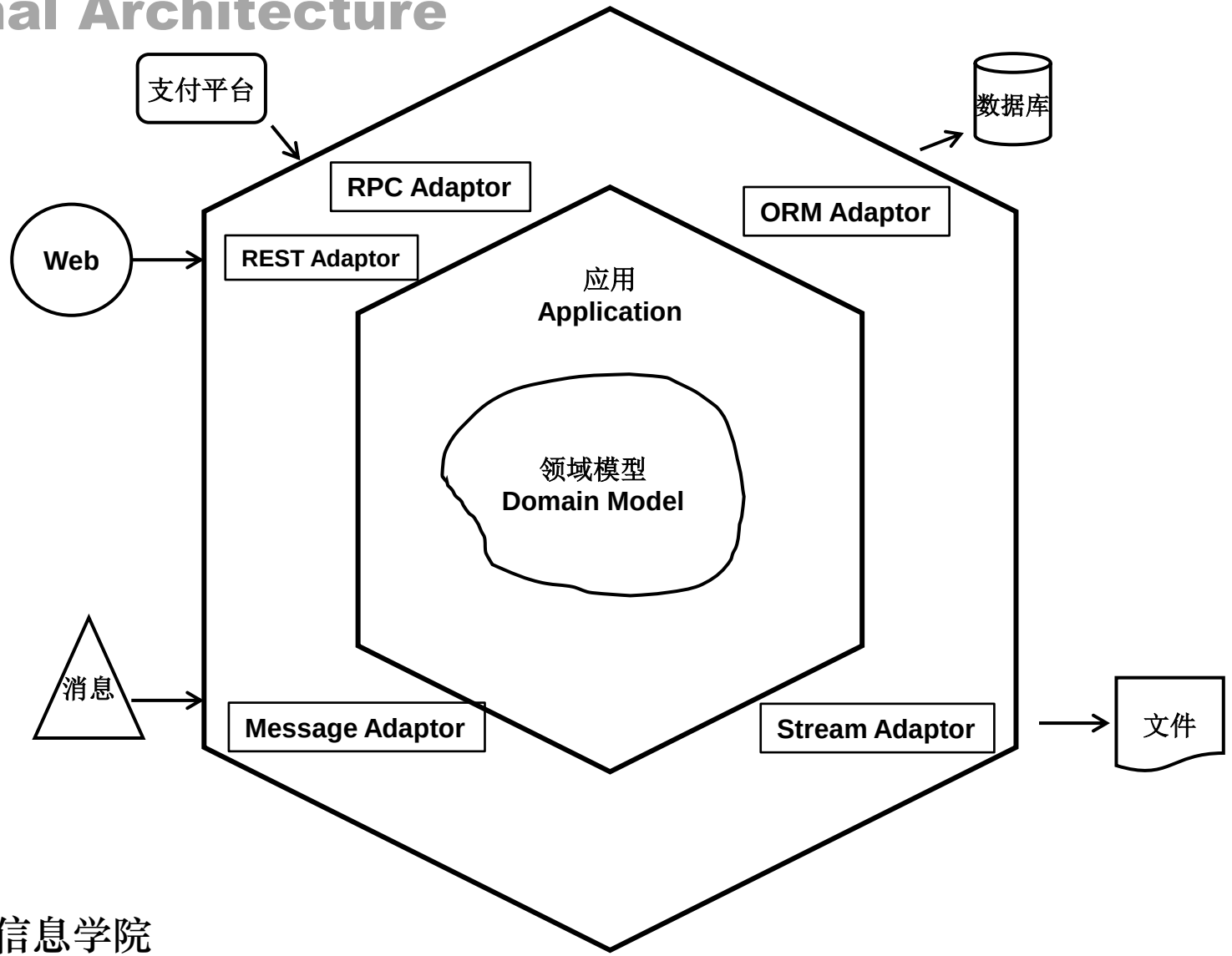
- 严格层次体系结构
  - 上层代码只能调用其直接下层代码
- 松散层次体系结构
  - 上层可以调用其所有的下层代码

3.1 层次体系结构
Layers Architecture



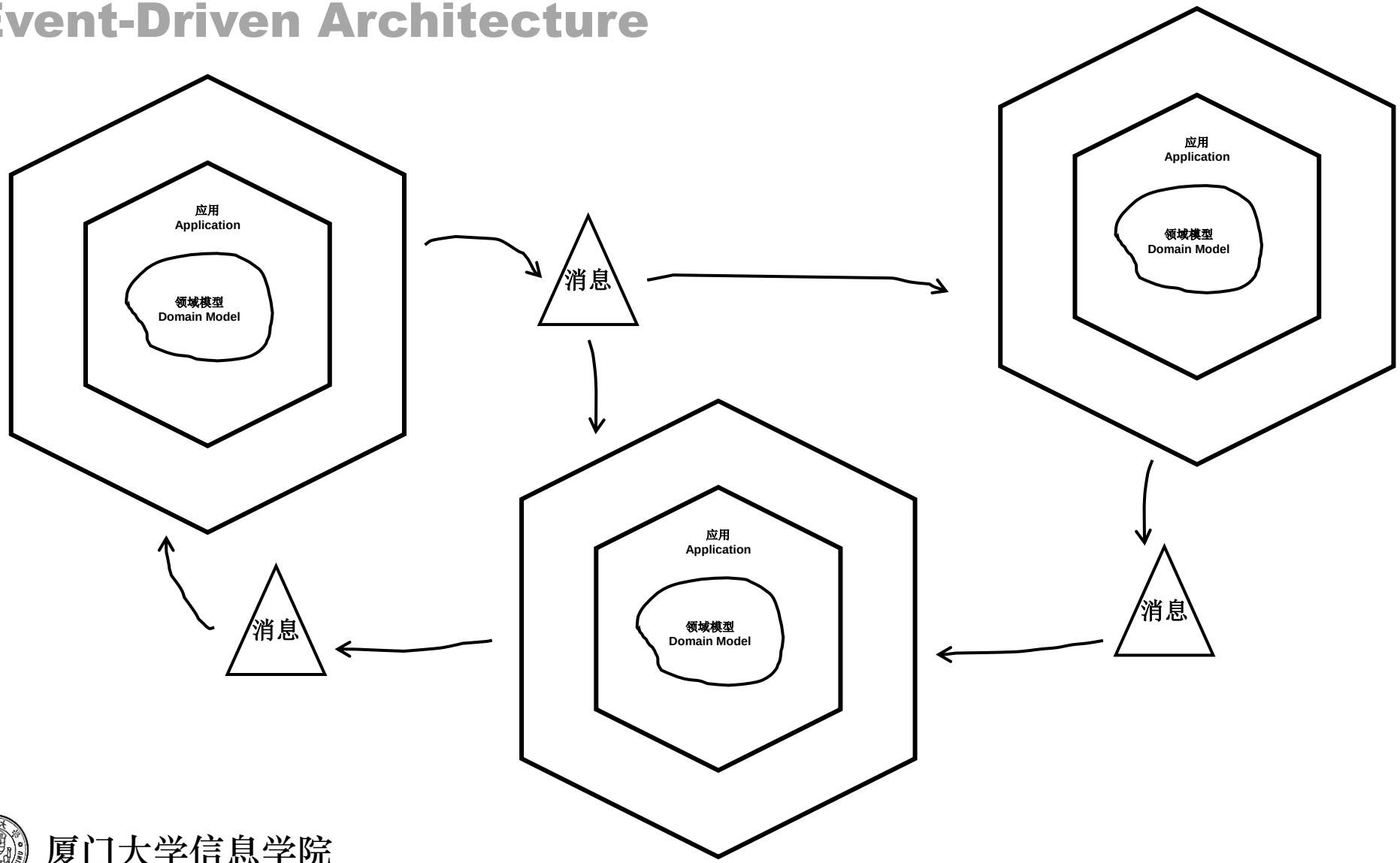
3.2 六边形体系结构

Hexagonal Architecture



3.3 消息驱动体系结构

Event-Driven Architecture



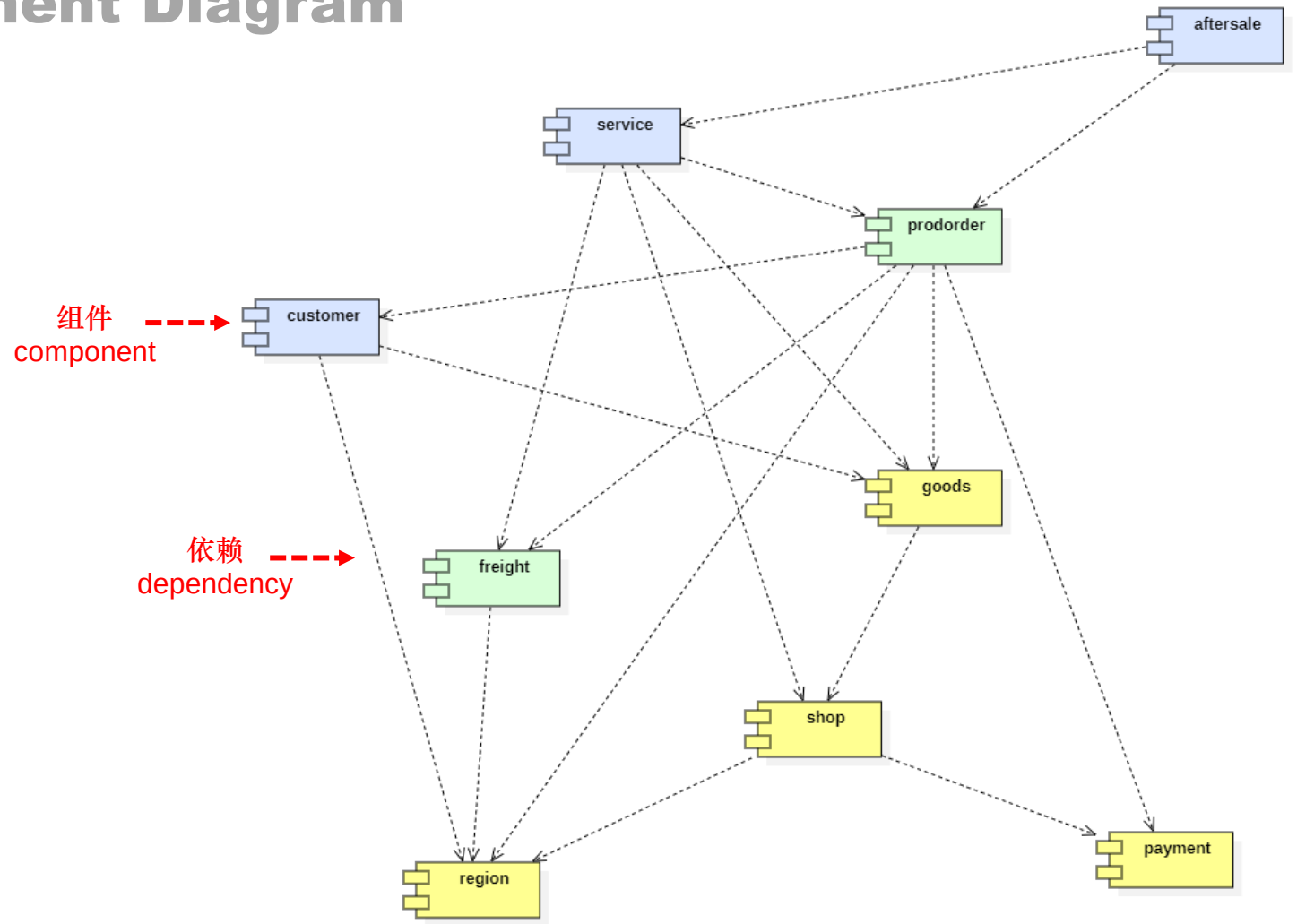
4. 组件图与部署图

Component Diagram and Deployment Diagram

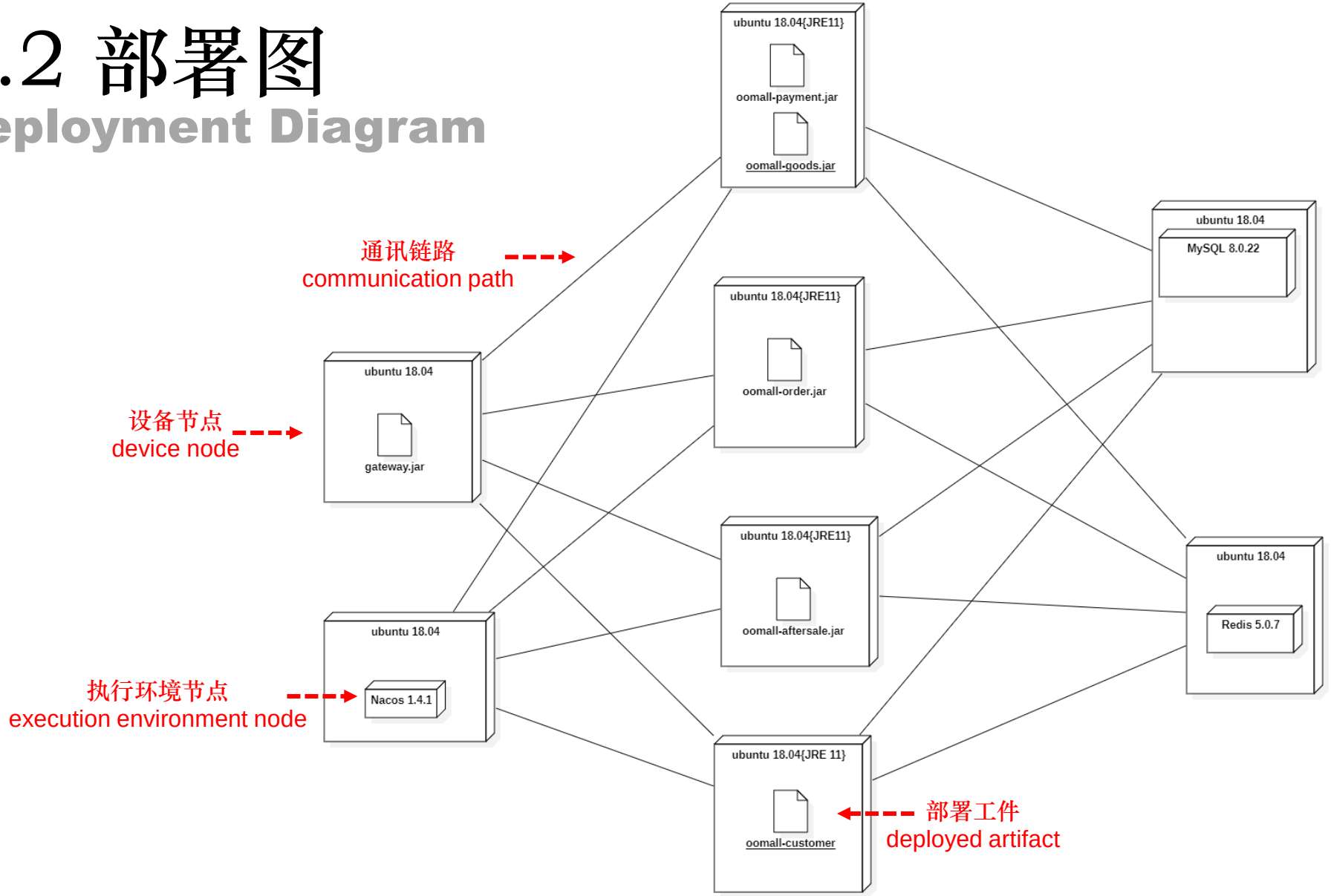
4.1 组件图
Component Diagram

- 组件是软件系统的模块
  - 包含了特定的逻辑，并在执行环境中是可替换的
- UML组件强调以下两点
  - 组件是通过接口被调用的
  - 模块化、自包含、可替换

4.1 组件图
Component Diagram



4.2 部署图
Deployment Diagram



4.2 部署图
Deployment Diagram