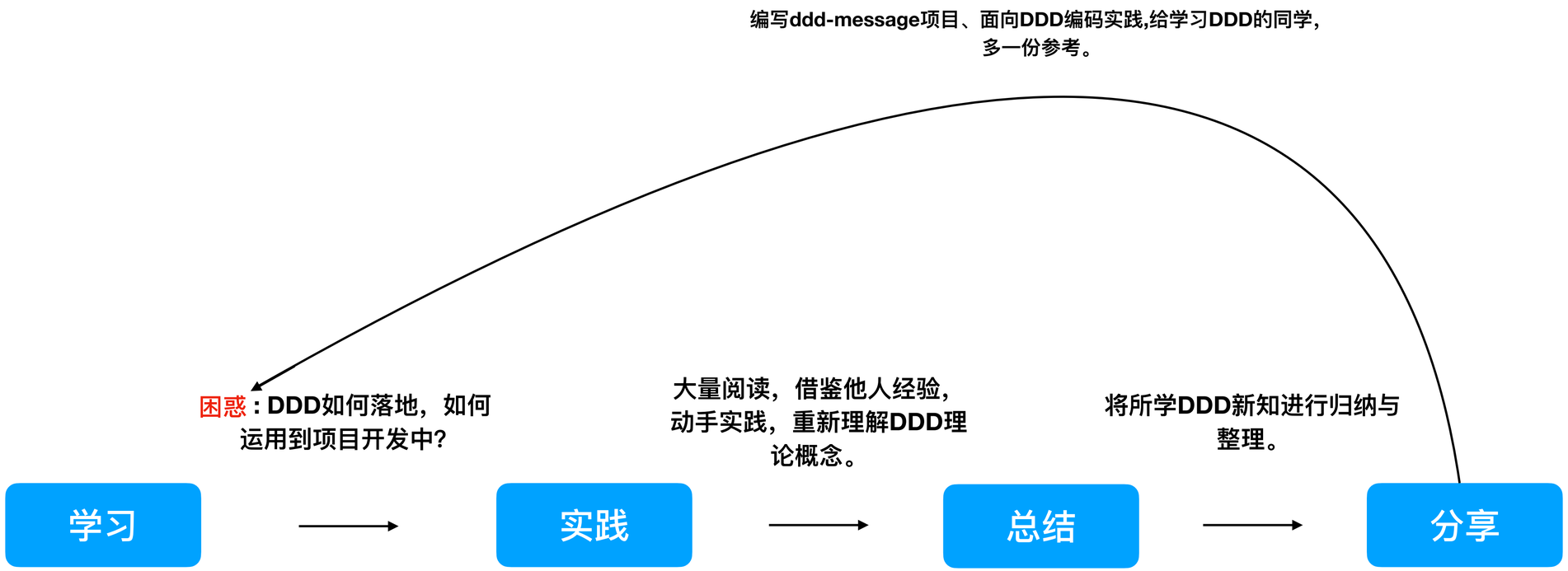
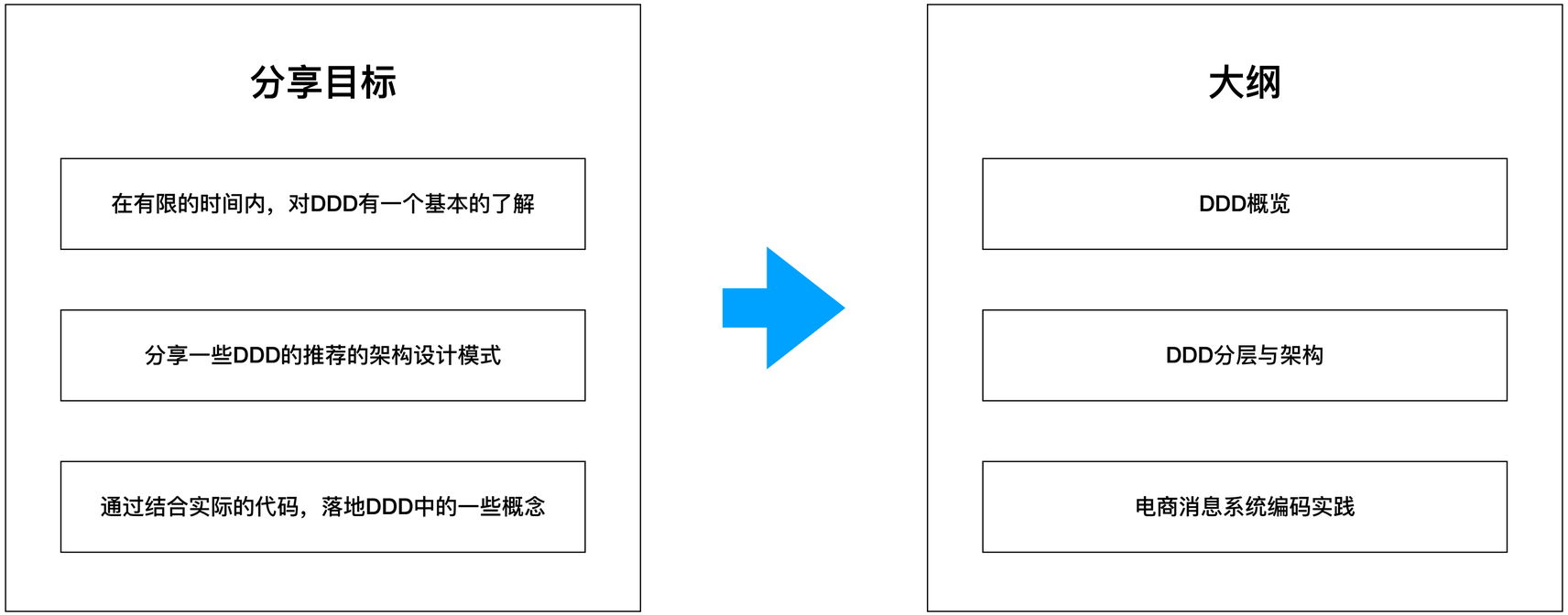
DDD编码实践

游诗成 2024-04-24

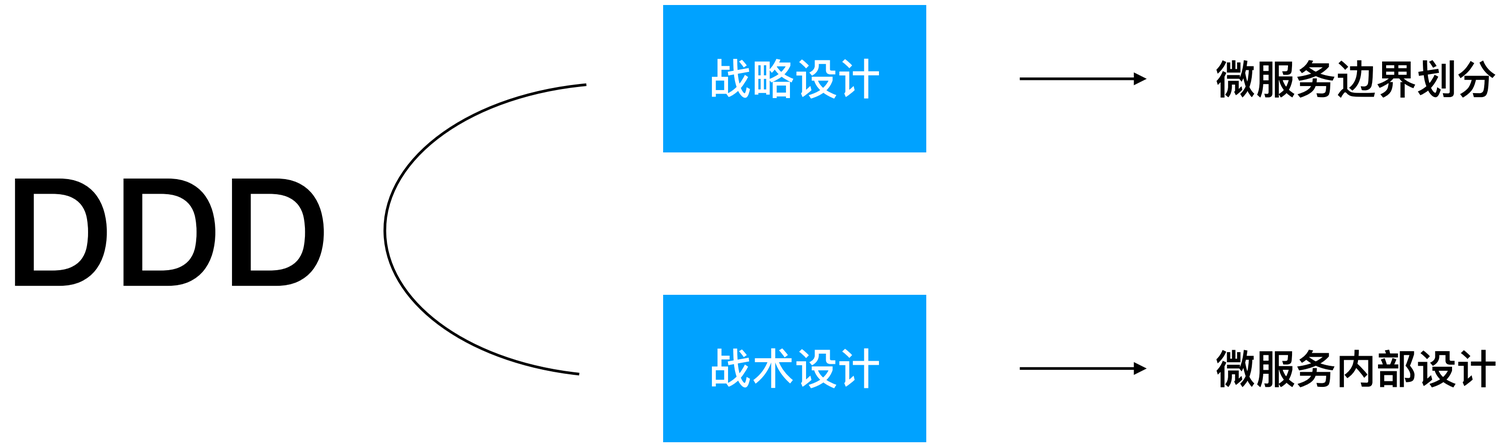
为什么会有这次分享?


编写ddd-message项目、面向DDD编码实践,给学习DDD的同学,
多一份参考。
困惑：DDD如何落地，如何
运用到项目开发中?
大量阅读，借鉴他人经验，
动手实践，重新理解DDD理
论概念。
将所学DDD新知进行归纳与
整理。
学习
实践
总结
分享

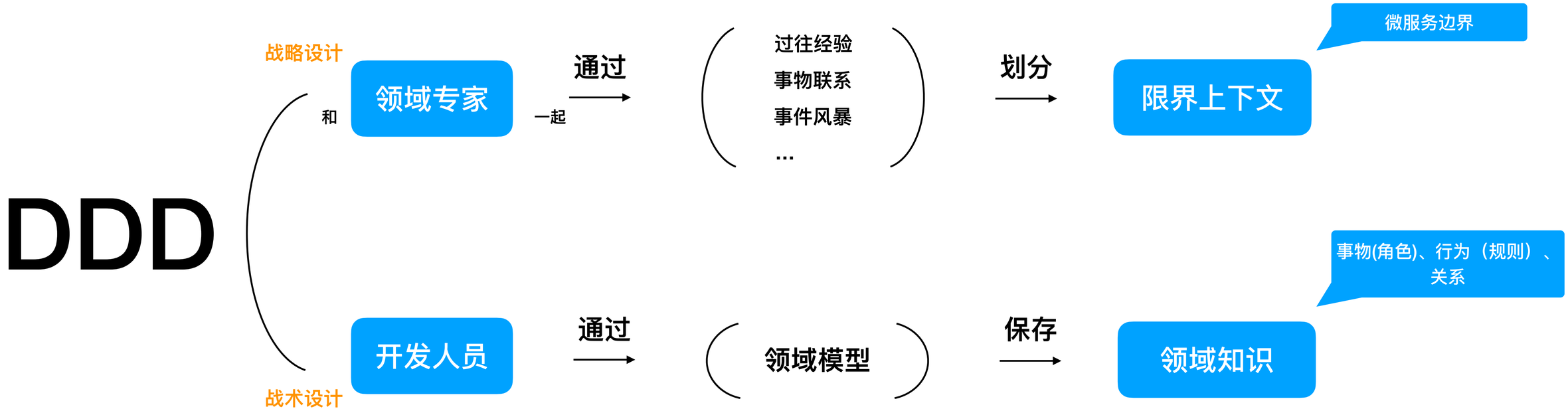
分享目标


分享目标
在有限的时间内，对DDD有一个基本的了解
分享一些DDD的推荐的架构设计模式
通过结合实际的代码，落地DDD中的一些概念
大纲
DDD概览
DDD分层与架构
电商消息系统编码实践

DDD可以做什么？



DDD如何做?

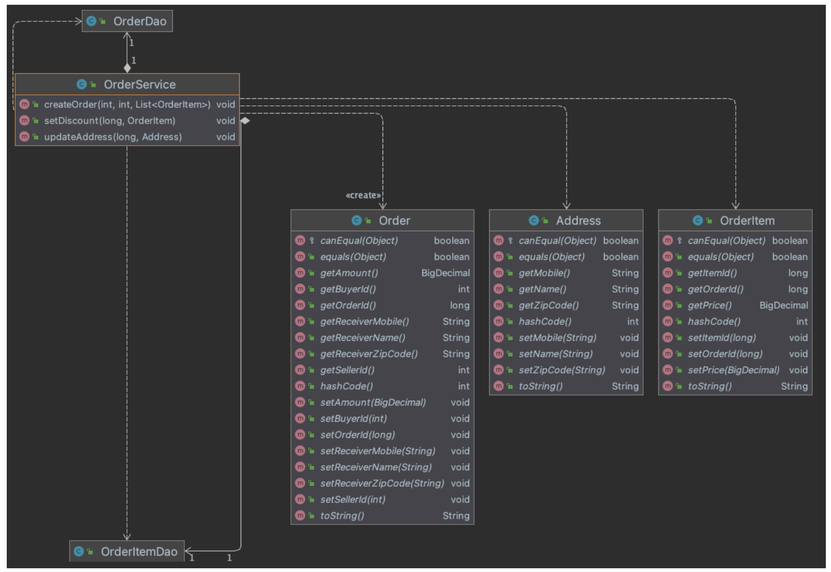


DDD通过领域模型围绕业务进行建模，并将模型与代码进行映射，业务调整影响代码的同时,代码也能直接的反映业务。

💡 按照常规的编码方式,代码就不能直接反映业务了吗?

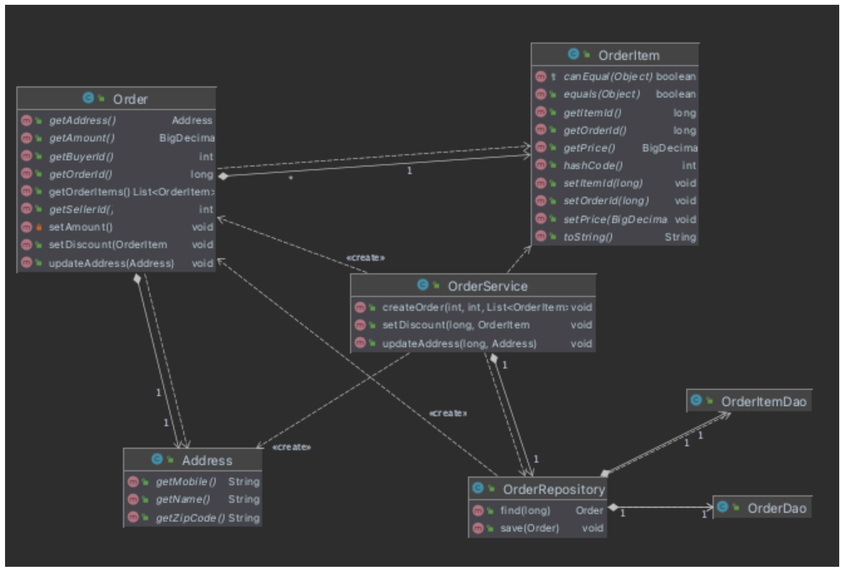
贫血模型到充血模型

贫血模型实现订单相关功能



对象里仅包含属性与get和set方法，没有行为方法，所有的业务逻辑放在业务逻辑层。

充血模型实现订单相关功能



对象包含了业务的行为方法，外部的“业务层”，仅处理一些流程控制，这种方法更符合面向对象设计。

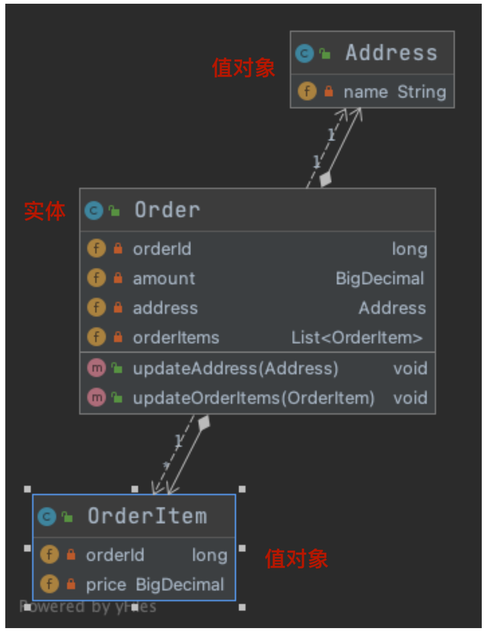
从面向数据开发到面向业务开发的转变

领域模型

领域模型

- **实体**：具有唯一标识，包含着业务知识的充血模型对象，用于对唯一性事物进行建模。
- **值对象**：生成后即不可变对象，通常作为实体的属性，用于描述领域中的事物的某种特征
- **领域服务**：分担实体的功能，承接部分业务逻辑，做一些实体不变处理的业务流程。不是必须的。
- **聚合**：将实体和值对象在一致性边界之内组成聚合,使用聚合划分微服务(限界上下文)内部的边界。
- **领域事件**：表示领域中所发生的事情，通过领域事件可以实现本地微服务(限界上下文)内的信息同步，同时也可以实现对外部系统的解耦。
- **资源库**：保存聚合的地方，将聚合实例存放在资源库（Repository）中，之后再通过该资源库来获取相同的实例。

实体与值对象




```java
public class Order {
    private long orderId;
    private BigDecimal amount;
    private Address address;
    private List<OrderItem> orderItems;

    //更新收货地址，整体覆盖
    public void updateAddress(Address address){
        this.address = address;
    }
    //订单商品发生变更，查找替换
    public void updateOrderItems(OrderItem orderItem){
        //1. 从 orderItems 找到对应的 orderItem 然后替换
        //2. 从新计算总价
    }
}
```

实体的特征

- 唯一标识，对唯一性事物进行建模
- 包含了业务的关键行为，可以随着业务持续变化
- 修改时，因为有唯一标识，所以还是同一个实体

值对象的特征

- 描述事物的某个特征，通常作为实体属性存在
- 创建后即不可变
- 修改时，用另一个值对象予以替换

💡 **Address收货地址可否为一个实体?**

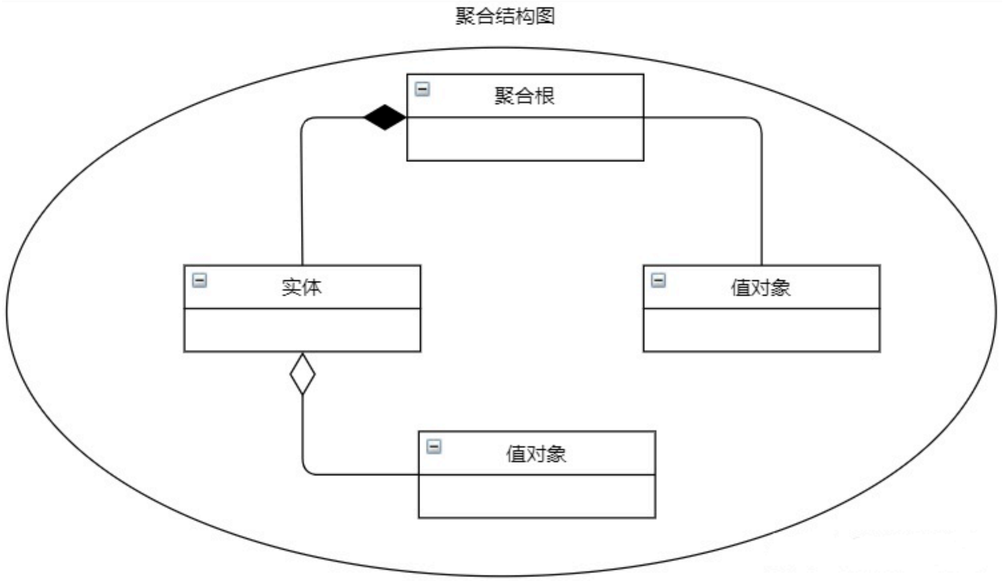
领域服务

分担实体的功能，承接部分业务逻辑，做一些实体不变处理的业务流程,不是必须的。

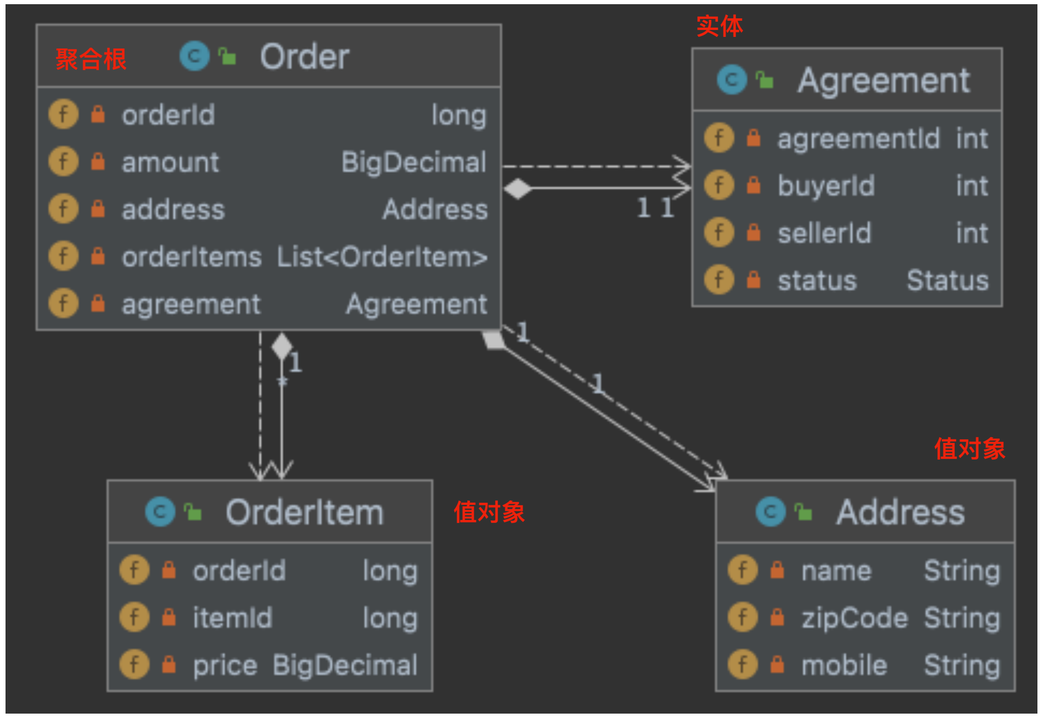
```java
@Service
public class MessageDomainServiceImpl implements MessageDomainService {
    private final UserServiceFacade userServiceFacade;
    private final MessageStatusSpecification messageStatusSpecification;
    private final ReadSpecification readSpecification;
    private final RecallSpecification recallSpecification;

    public MessageDomainServiceImpl(UserServiceFacade userServiceFacade, MessageStatusSpecification messageStatusSpecification, Rea

    @Override
    public Message createMessage(long messageId, MessageCategory category , int senderId, int receiverId, String sourceContent) {
        //通过远程接口获取用户信息值对象
        User sender = userServiceFacade.getUser(senderId);
        User receiver = userServiceFacade.getUser(receiverId);
        //构造消息内容值对象
        Content content = new Content(category,sourceContent);
        //创建实体
        Message message = new Message(messageId,category,sender, receiver, content, new Date());
        //根据规约处理消息状态
        message.handleStatusBy(messageStatusSpecification);
        //根据规约处理消息已读未读状态
        message.handleReadStatusBy(readSpecification);
        return message;
    }

    @Override
    public Result<Void> recall(Message message) {...}
}
```

聚合

将实体和值对象在一致性边界之内组成聚合,使用聚合划分限界上下文(微服务)内部的边界。



聚合根：一种特殊的实体，用于管理聚合内部的实体与值对象，并将自身暴露给外部进行引用。

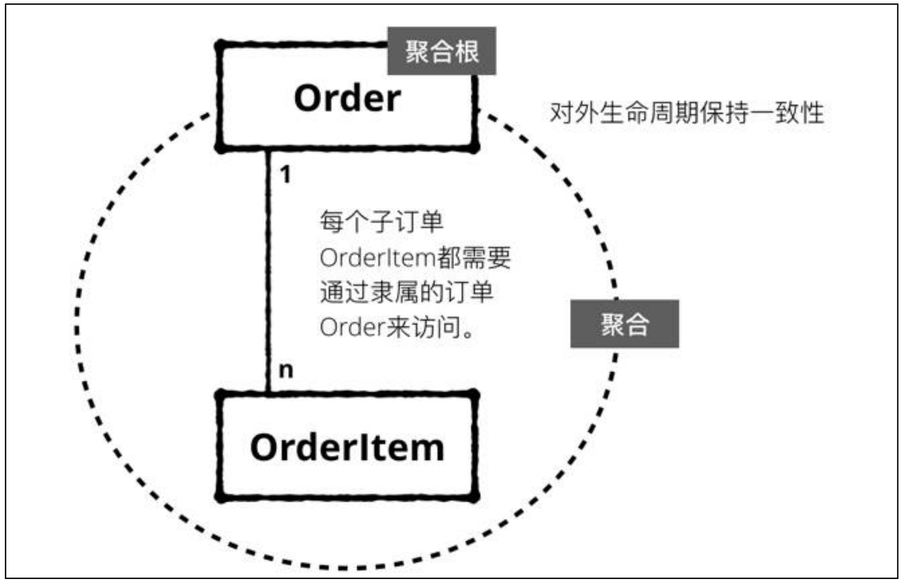


宇宙由一些永恒的物体聚合而成，这些物体通过某种因果关系联系在一起，这种关系独立于物体本身，并且存在于客观的空间和时间中。

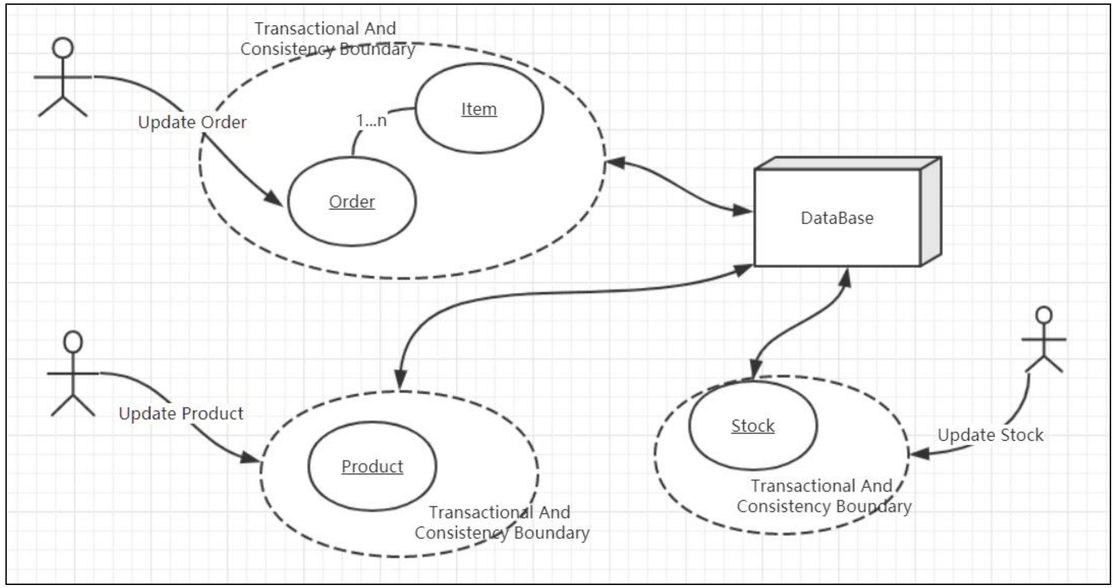
—Jean Piaget

聚合的一致性边界

聚合生命周期的一致性



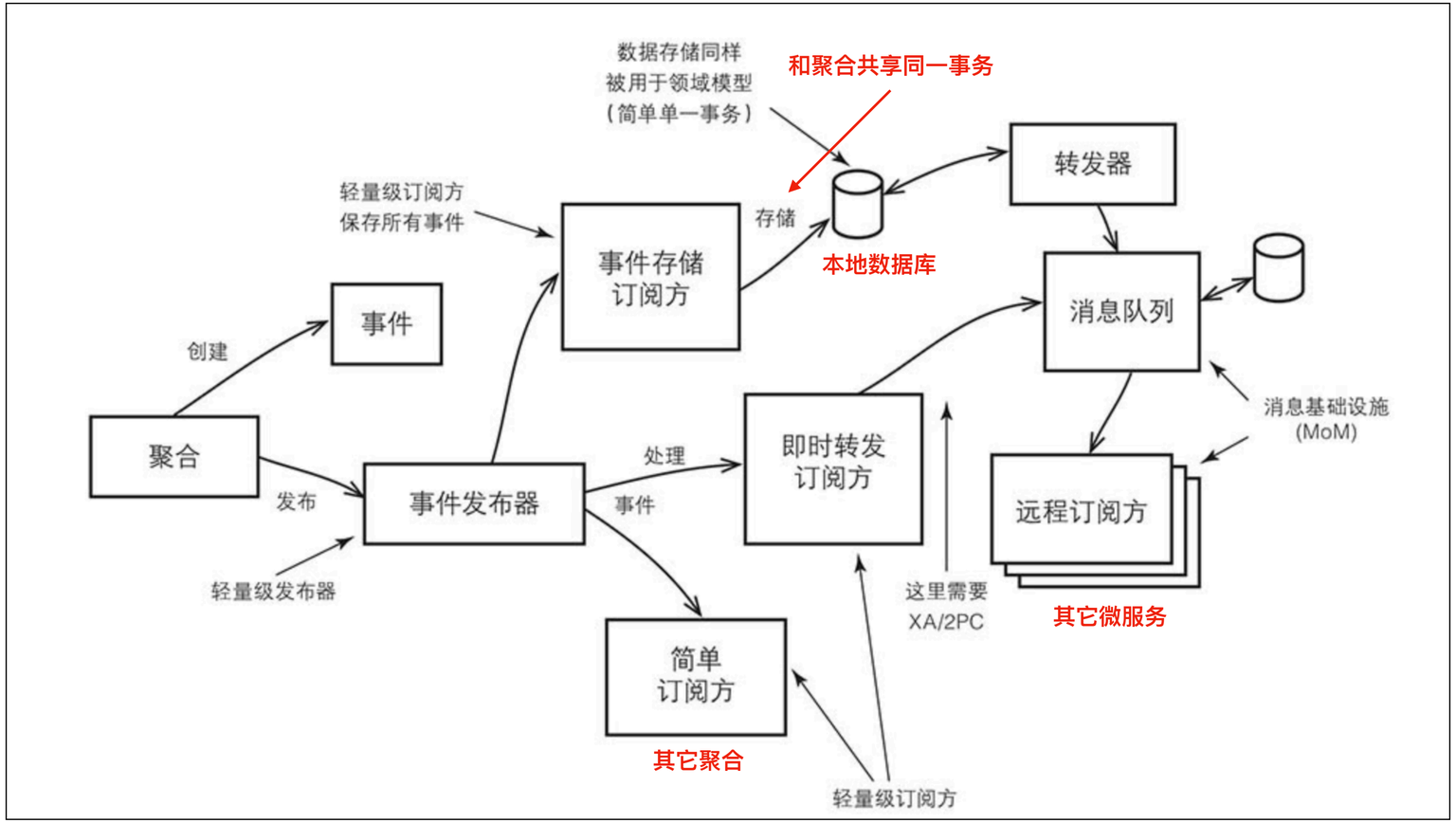
聚合事务的一致性边界



💡 当聚合发生改变，其它聚合如何感知这一变化?

领域事件

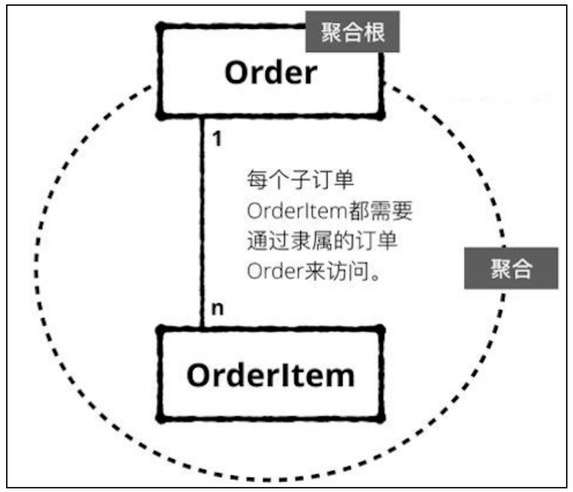
领域事件表示领域中所发生的事情，通过领域事件可以实现微服务内的信息同步，同时也可以实现对外部系统的解耦。

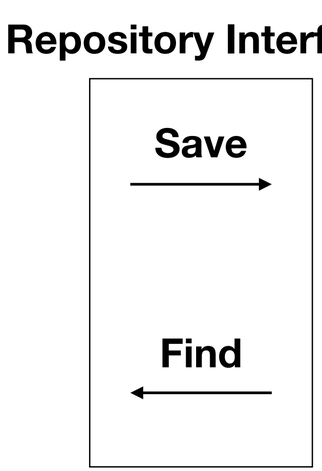


说明：聚合创建发布事件。订阅方可以先存储事件，然后再将其转发到远程的订阅方中；或者不经存储，直接转发，即时转发需要XA（两阶段提交）。

资源库

保存聚合的地方，将聚合实例存放在资源库（Repository）中，之后再通过该资源库来获取相同的实例。





Mysql

orderId	amount
1	100

Order

orderId	itemId	itemName
1	1	黄瓜
1	2	西红柿
1	3	大白菜

OrderItem

ES

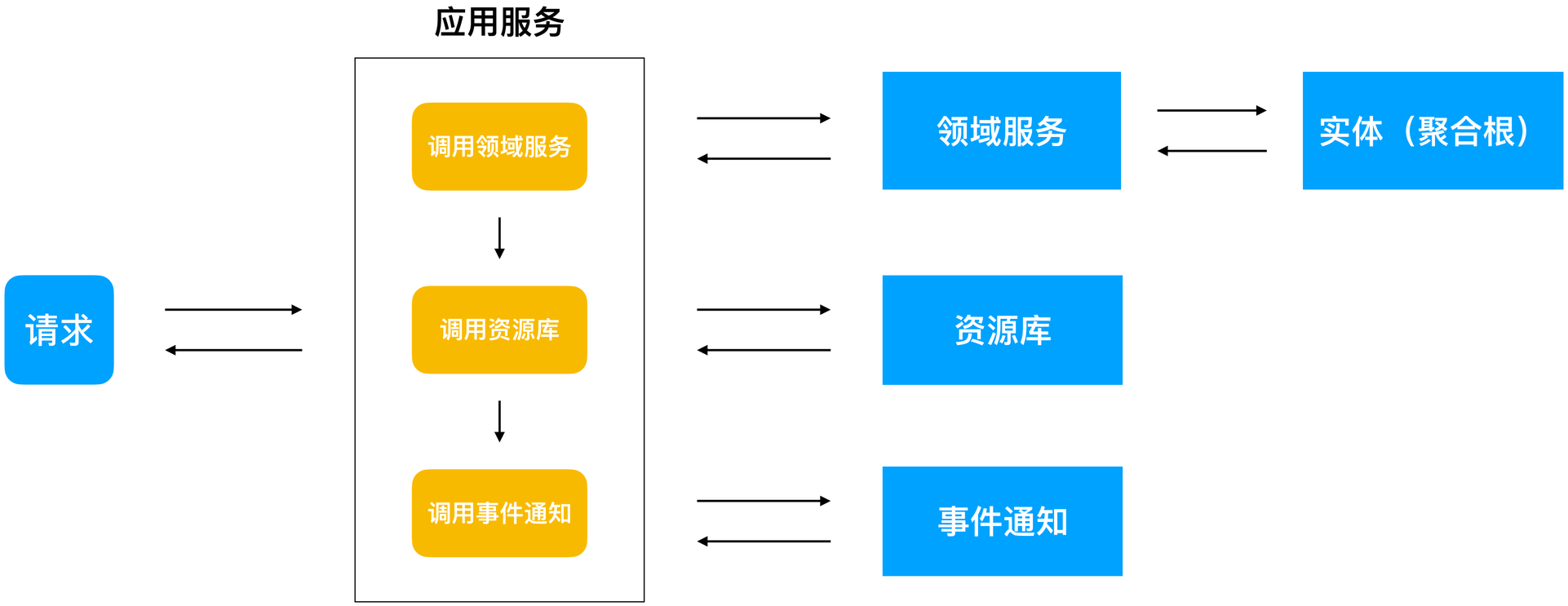
```
{
  "_index": "order",
  "_source": {
    "orderId": 1,
    "amount": 100.00,
    "orderItems":[
        {"itemId":1,"itemName":"黄瓜"},
        {"itemId":2,"itemName":"西红柿"},
        {"itemId":3,"itemName":"大白菜"}
    ]
}
```

Save: 聚合对象由Repository的实现,转换为存储所支持的数据结构进行持久化

Find: 根据存储所支持的数据结构,由Repository的实现转换为聚合对象

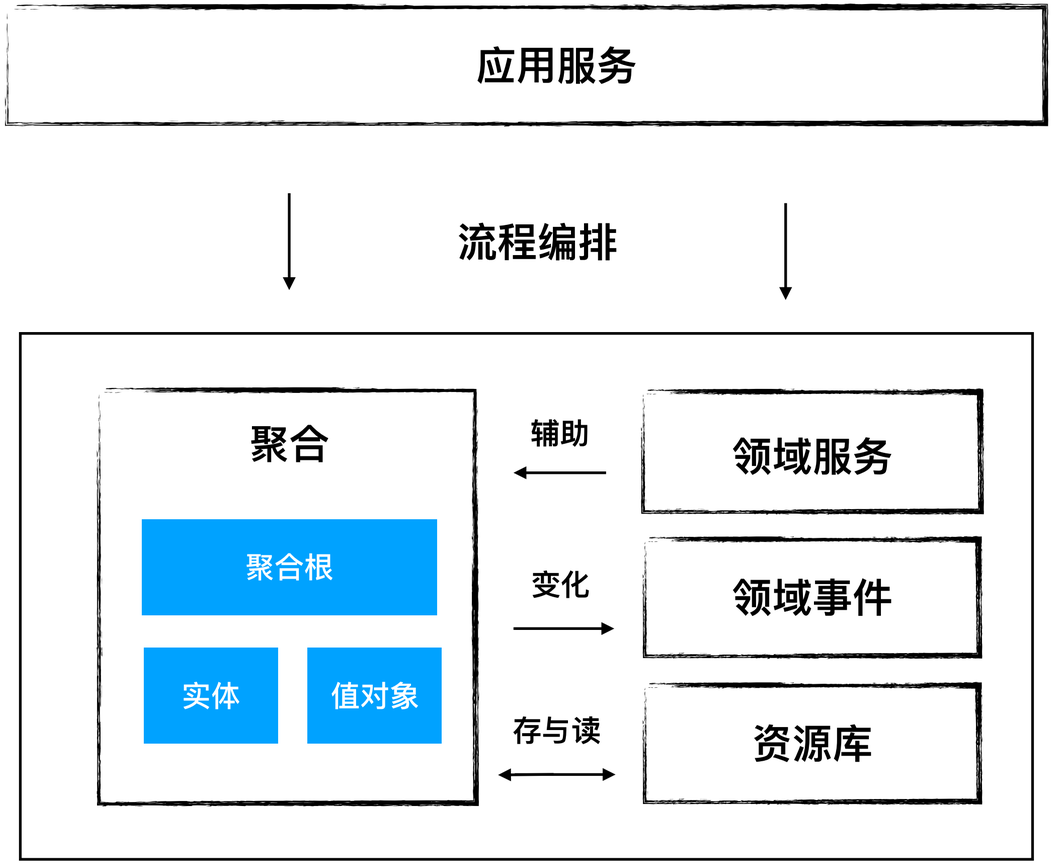
应用服务

负责流程编排，它将要实现的功能委托给一个或多个领域对象来实现，本身只负责处理业务用例的执行顺序以及结果的拼装。



DDD概览总结

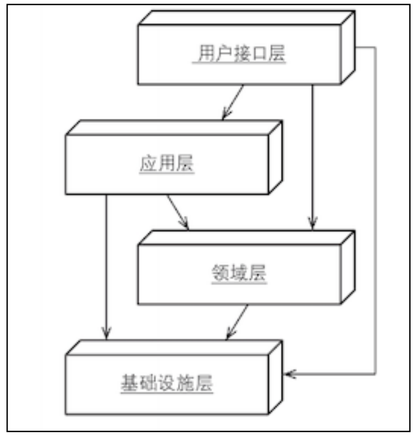

领域事件
聚合
实体
应用服务
聚合根
领域服务
值对象
资源库


应用服务
流程编排
聚合
聚合根
实体
值对象
辅助
领域服务
变化
领域事件
存与读
资源库

DDD分层架构

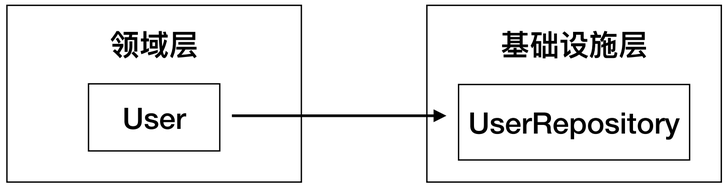
传统架构下的DDD分层

基础层是被其它层所共同依赖，它处于最底层，然而在DDD中领域层才是核心,这会导致**重心偏移**。在这样的分层下，很难避免核心的领域模型对象与基础设施层发生**直接耦合**。因此要改变这种依赖。



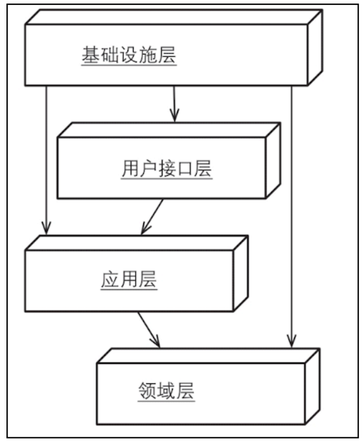
依赖倒置(DIP)

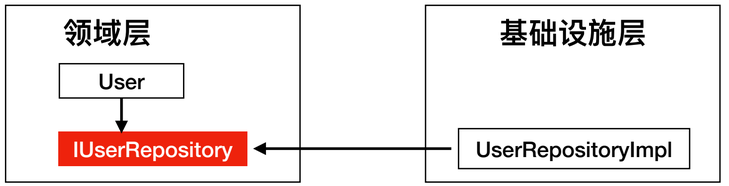
高层模块不应该依赖于底层模块，二者都应该依赖于抽象。抽象不应该依赖于细节，细节应该依赖于抽象。



依赖倒置后的DDD分层

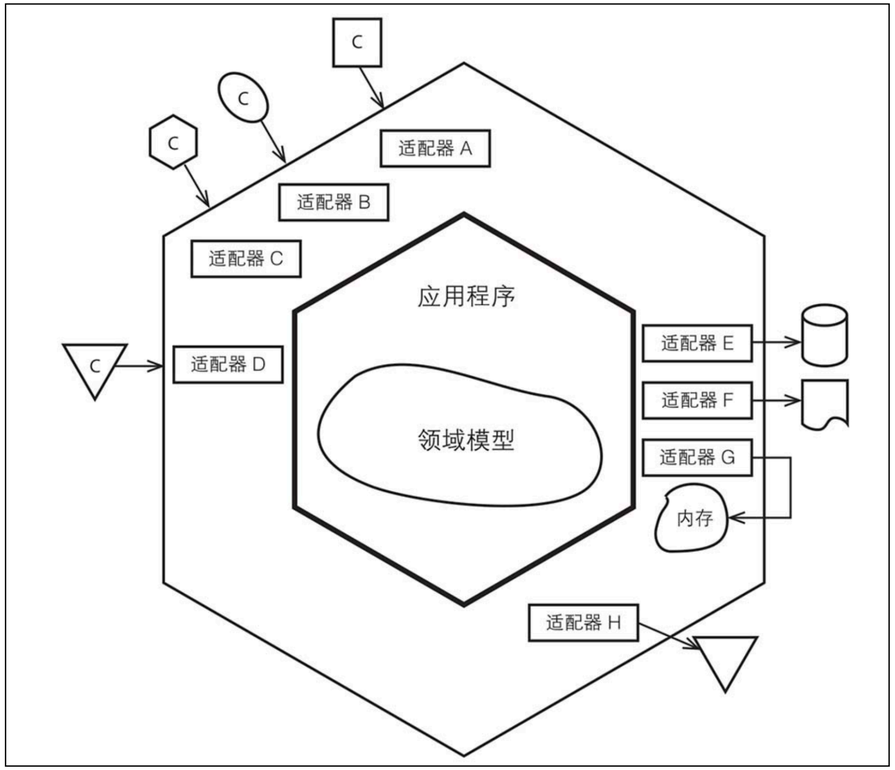
将基础设施层放在所有层的最上方，这样它可以实现所有其他层中定义的接口。对于下层模块而言只需要定义出接口，而不用直接依赖于基础层。

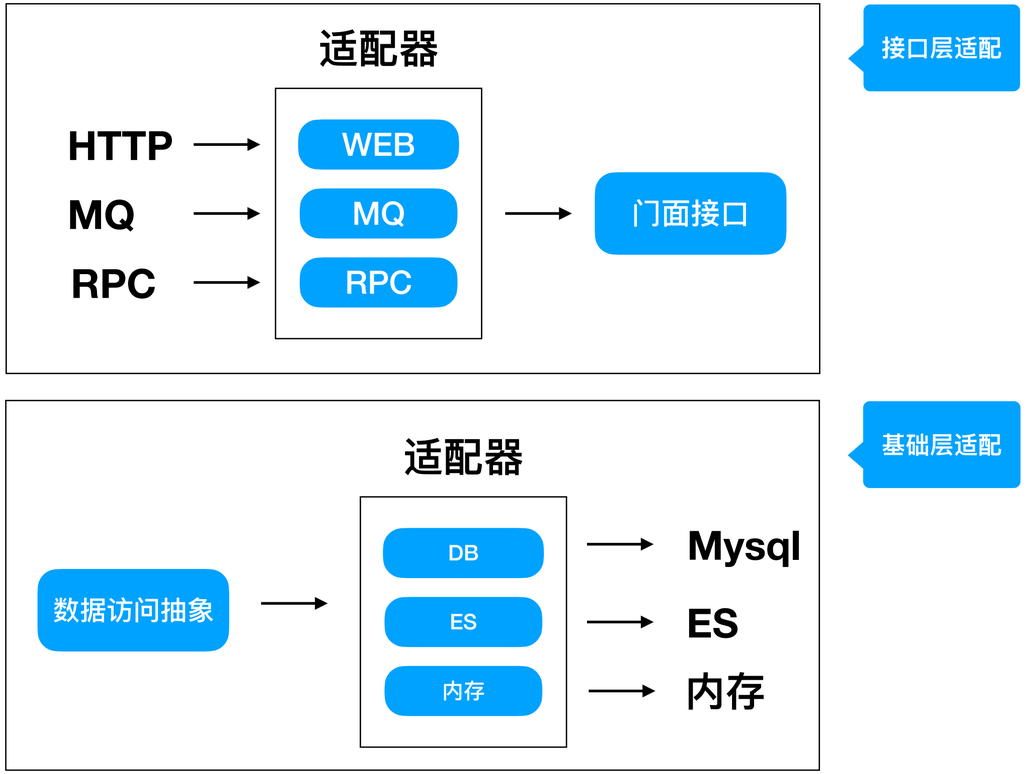




六边形架构（端口适配器架构）

对于每种外界类型，都有一个适配器与之相对应。业务核心逻辑被包裹在内部，外界通过应用层API与内部进行交互，内部的实现无须关注外部的变化，更加聚焦。在这种架构下还可以轻易地开发用于测试的适配器。





在《实现领域驱动设计》一书中,作者认为它是一种具有持久生命力的架构。

DDD电商消息系统编码实践

项目地址：https://gitee.com/izhengyin/ddd-message

Tag

- Part1：了解DDD的项目工程结构,以及各层直接如何配合
- Part2：了解业务规则验证，规约模式的使用
- Part3：了解对DDD中已存在实体如何进行修改
- Part4：了解对值对象进行扩展，适应更复杂的需求
- Part5：了解CQRS，查询与命令分离的架构

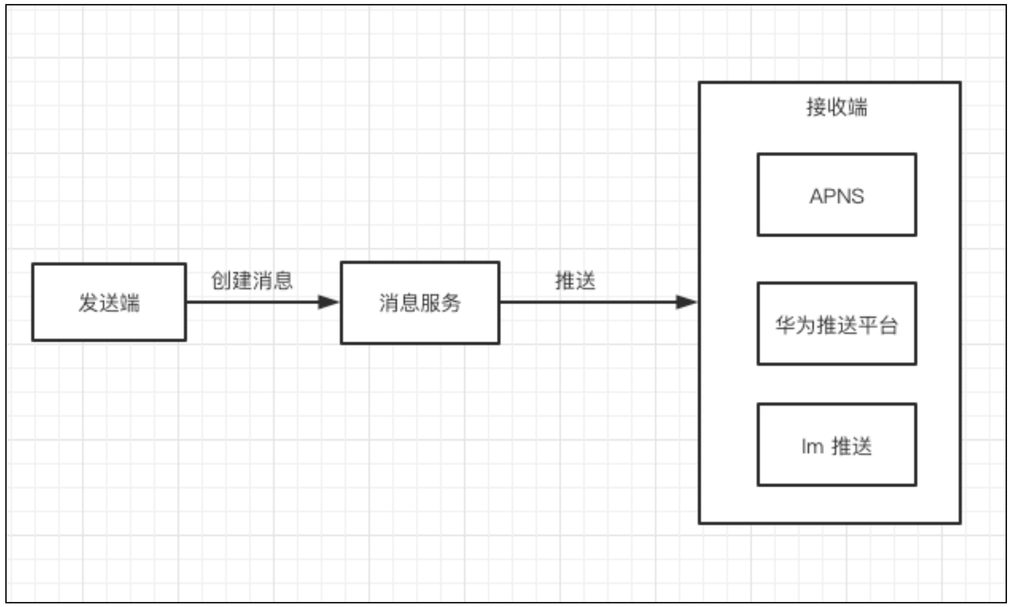
以一个电商消息系统的场景为案例，通过模拟需求分步骤的完成开发，每个Tag对应一个需求，通过检出Tag可以看到项目的变化过程，同时每个步骤还有一篇介绍文章，方便学习。

DDD电商消息系统编码实践（一）

了解一个DDD项目的工程结构，以及各层之间如何配合。

需求概览

实现一个最基本的创建消息并将消息推送到不同终端的需求。



DDD电商消息系统编码实践（二）

了解业务规则验证，规约模式的使用

需求概览

- 未读消息功能，判断消息接收方是否读取了消息。
- 违禁词功能，当消息内容包含违禁词时，消息发送失败，提示发送方原因。
- 安全验证功能，当消息内容包含不安全的信息时，消息发送失败，提示发送方原因。
- 黑名单功能，当接收方将发送方加入黑名单时，消息发送失败，提示发送方原因。

DDD电商消息系统编码实践（三）

了解对已存在实体进行更新时如何实现。

需求概览

增加消息撤回功能，允许5分钟内的消息撤回

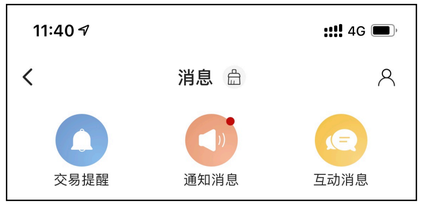
DDD电商消息系统编码实践（四）

丰富消息的类型，涉及对存在属性的调整，看看如何通过对值对象的调整适应这种变化。

需求概览

- 增加消息分类，用于区分不同业务的消息
- 增加消息卡片，用于适应不同消息的展示形态

分类



卡片

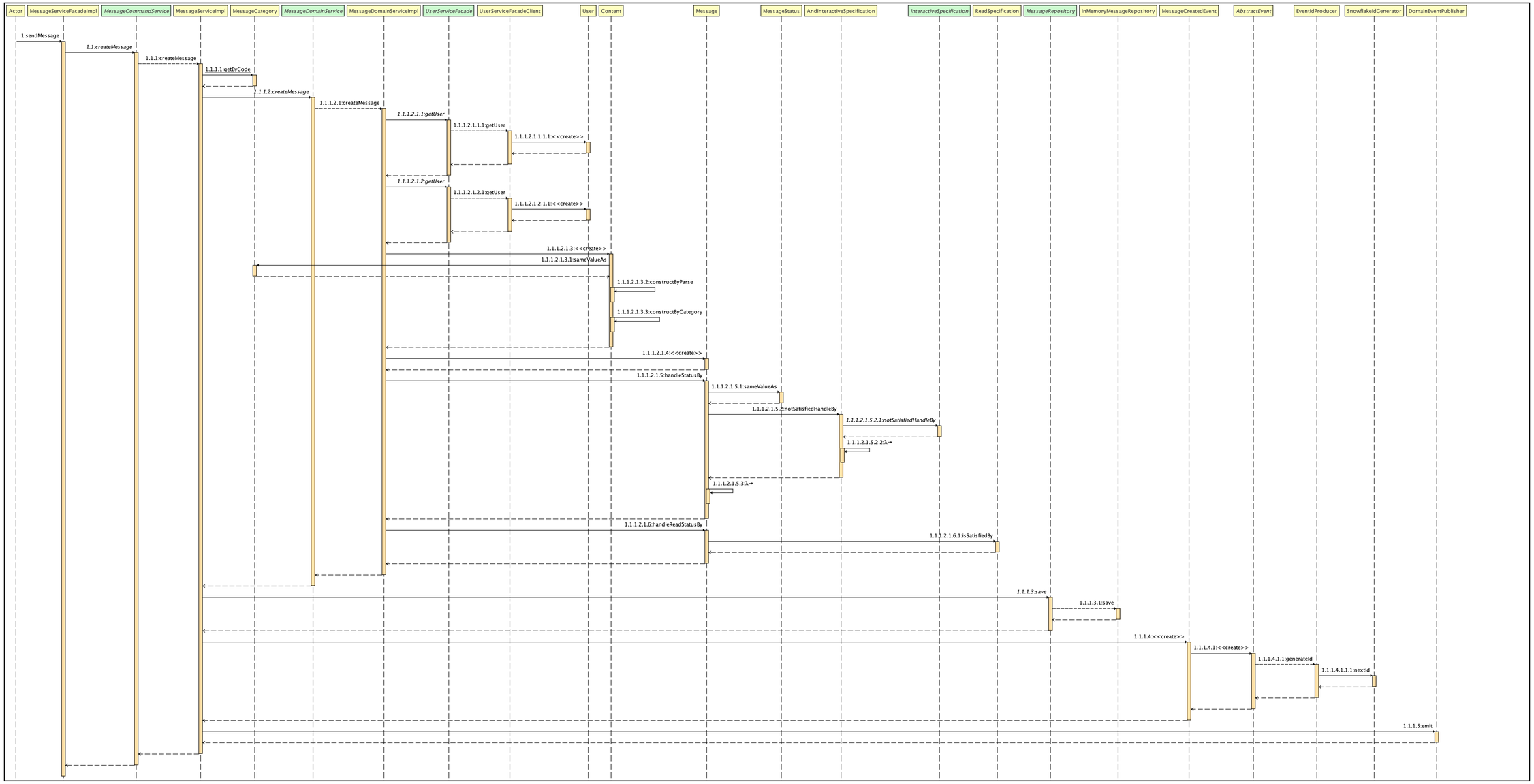
DDD电商消息系统编码实践（五）

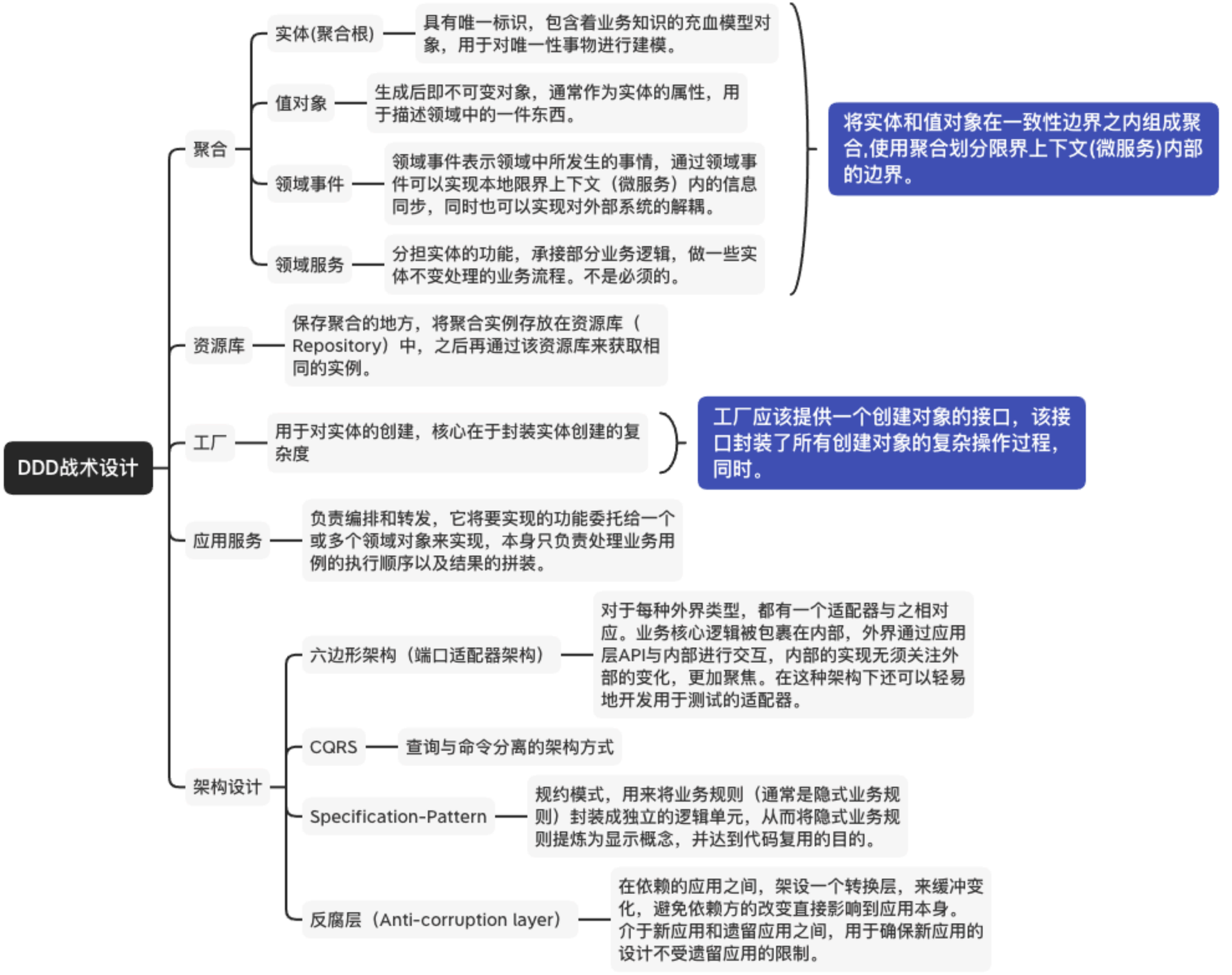
了解CQRS，查询与命令分离的架构

需求概览

- 增加一个查询未读数量的接口
- 增加一个查询往来消息的接口

回顾发消息的流程


Actor
MessageServiceFacadeImpl
MessageCommandService
MessageServiceImpl
MessageCategory
MessageDomainService
MessageDomainServiceImpl
UserServiceFacade
UserServiceFacadeClient
User
Content
Message
MessageStatus
AndInteractiveSpecification
InteractiveSpecification
ReadSpecification
MessageRepository
InMemoryMessageRepository
MessageCreatedEvent
AbstractEvent
EventIdProducer
SnowflakeIdGenerator
DomainEventPublisher
1:sendMessage
1.1:createMessage
1.1.1:createMessage
1.1.1.1:getByCode
1.1.1.2:createMessage
1.1.1.2.1:createMessage
1.1.1.2.1.1:getUser
1.1.1.2.1.1.1:getUser
1.1.1.2.1.1.1.1:<<create>>
1.1.1.2.1.1.2:getUser
1.1.1.2.1.2.1:getUser
1.1.1.2.1.2.1.1:<<create>>
1.1.1.2.1.3:<<create>>
1.1.1.2.1.3.1:sameValueAs
1.1.1.2.1.3.2:constructByParse
1.1.1.2.1.3.3:constructByCategory
1.1.1.2.1.4:<<create>>
1.1.1.2.1.5:handleStatusBy
1.1.1.2.1.5.1:sameValueAs
1.1.1.2.1.5.2:notSatisfiedHandleBy
1.1.1.2.1.5.2.1:notSatisfiedHandleBy
1.1.1.2.1.5.2.2:λ→
1.1.1.2.1.5.3:λ→
1.1.1.2.1.6:handleReadStatusBy
1.1.1.2.1.6.1:isSatisfiedBy
1.1.1.3:save
1.1.1.3.1:save
1.1.1.4:<<create>>
1.1.1.4.1:<<create>>
1.1.1.4.1.1:generateId
1.1.1.4.1.1.1:nextId
1.1.1.5:emit


DDD战术设计
聚合
实体(聚合根)
具有唯一标识，包含着业务知识的充血模型对象，用于对唯一性事物进行建模。
值对象
生成后即不可变对象，通常作为实体的属性，用于描述领域中的一件东西。
领域事件
领域事件表示领域中所发生的事情，通过领域事件可以实现本地限界上下文（微服务）内的信息同步，同时也可以实现对外部系统的解耦。
领域服务
分担实体的功能，承接部分业务逻辑，做一些实体不变处理的业务流程。不是必须的。
将实体和值对象在一致性边界之内组成聚合,使用聚合划分限界上下文(微服务)内部的边界。
资源库
保存聚合的地方，将聚合实例存放在资源库（Repository）中，之后再通过该资源库来获取相同的实例。
工厂
用于对实体的创建，核心在于封装实体创建的复杂度
工厂应该提供一个创建对象的接口，该接口封装了所有创建对象的复杂操作过程，同时。
应用服务
负责编排和转发，它将要实现的功能委托给一个或多个领域对象来实现，本身只负责处理业务用例的执行顺序以及结果的拼装。
架构设计
六边形架构（端口适配器架构）
对于每种外界类型，都有一个适配器与之相对应。业务核心逻辑被包裹在内部，外界通过应用层API与内部进行交互，内部的实现无须关注外部的变化，更加聚焦。在这种架构下还可以轻易地开发用于测试的适配器。
CQRS
查询与命令分离的架构方式
Specification-Pattern
规约模式，用来将业务规则（通常是隐式业务规则）封装成独立的逻辑单元，从而将隐式业务规则提炼为显示概念，并达到代码复用的目的。
反腐层（Anti-corruption layer）
在依赖的应用之间，架设一个转换层，来缓冲变化，避免依赖方的改变直接影响到应用本身。介于新应用和遗留应用之间，用于确保新应用的设计不受遗留应用的限制。

参考资料

- 《重构,改善既有代码的设计》马丁·福勒（Martin Fowler）
- 《领域驱动设计：软件核心复杂性应对之道》（修订版）埃里克 埃文斯（Eric Evans）
- 《实现领域驱动设计》Vaughn.Vernon（沃恩.弗农）
- 《架构整洁之道》Robert C. Martin（罗伯特C.马丁）
- 《DDD实战课》极客时间 , 欧创新
- Thoughworks洞见技术博客,DDD专栏 https://insights.thoughtworks.cn/tag/domain-driven-design/
- 微软Azure技术博客,云原生架构设计模式专栏 https://docs.microsoft.com/en-us/azure/architecture/patterns/
- https://github.com/citerus/dddsample-core
- https://github.com/e-commerce-sample
- https://github.com/ouchuangxin/leave-sample