如何让DDD落地

和坚

DDD CHINA

AGENDA

01 为什么DDD落地难

02 从需求到代码

03 当需求发生变化时

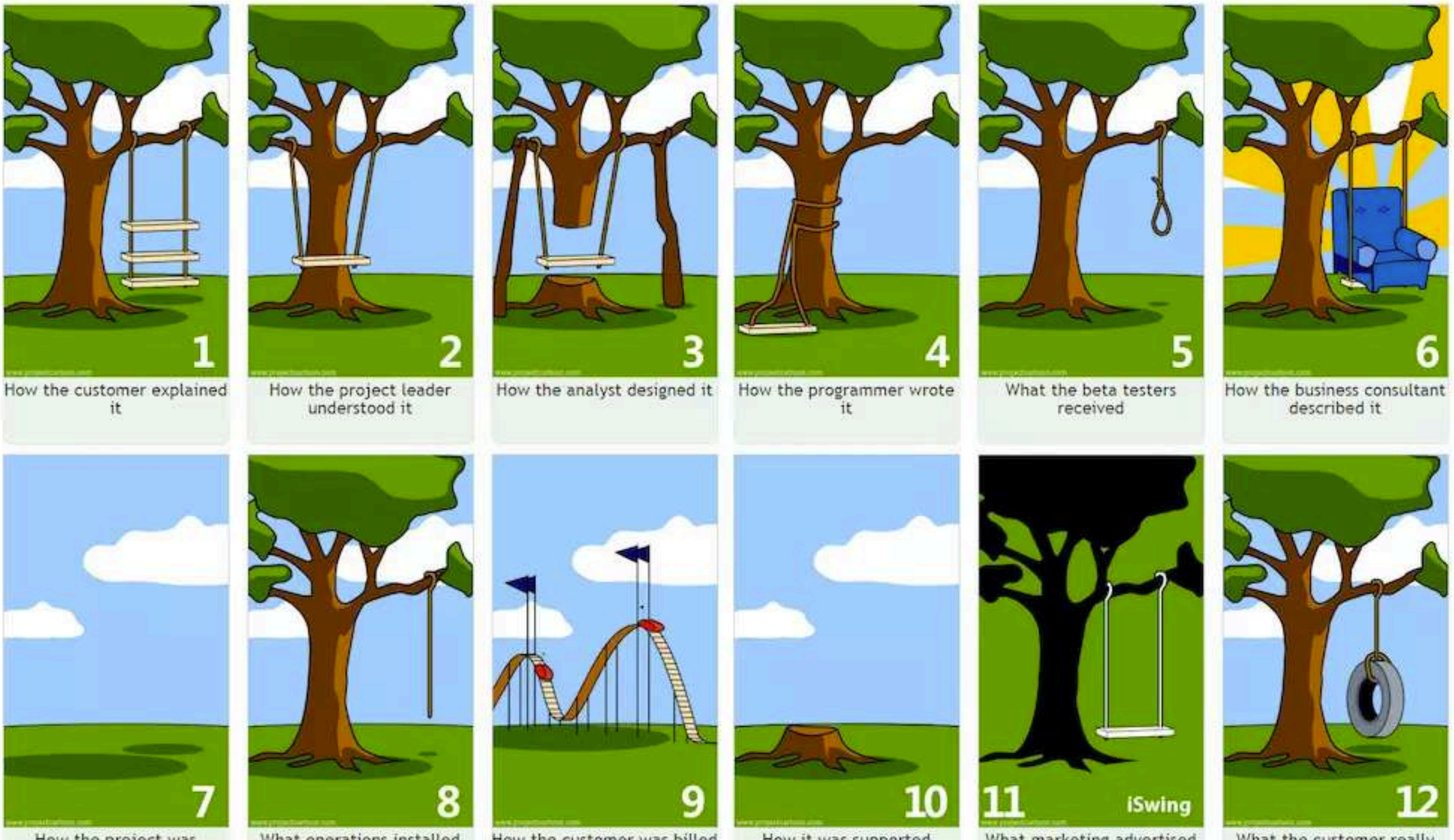
04 把大象塞进冰箱

01 为什么DDD难落地

从2003年由Eric Evans提出DDD以后，在软件开发领域一直都是雷声大，雨点小。这几年之所以开始火起来，主要的功劳也要给队友“微服务”，那么这是为什么呢?

DDD的目标是什么

软件设计的门槛是什么


1
How the customer explained it
2
How the project leader understood it
3
How the analyst designed it
4
How the programmer wrote it
5
What the beta testers received
6
How the business consultant described it
7
How the project was documented
8
What operations installed
9
How the customer was billed
10
How it was supported
11
iSwing
What marketing advertised
12
What the customer really needed

DDD里难懂的概念


push state change with
SERVICES
DOMAIN EVENTS
access with
REPOSITORIES
access with
express model with
express change with
ENTITIES
encapsulate with
act as root of
express identity with
encapsulate with with
Aggregate
express state & computation with
VALUE OBJECTS
MODEL-DRIVEN DESIGN
encapsulate with
model gives structure to
isolate domain expressions with
encapsulate with
define model within
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich model with
keep model unified by
CONTINUOUS INTEGRATION
names enter
BOUNDED CONTEXT
SHARED KERNEL
SHARED KERNEL
CORE DOMAIN
interdependent contexts form
overlap allied contexts through
CUSTOMER /SUPPLIER
relate allied contexts as
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
minimize translation
avoid overinvesting in
CONFORMIST

谁来做领域专家?

产品经理

系统分析师

技术组长

架构师

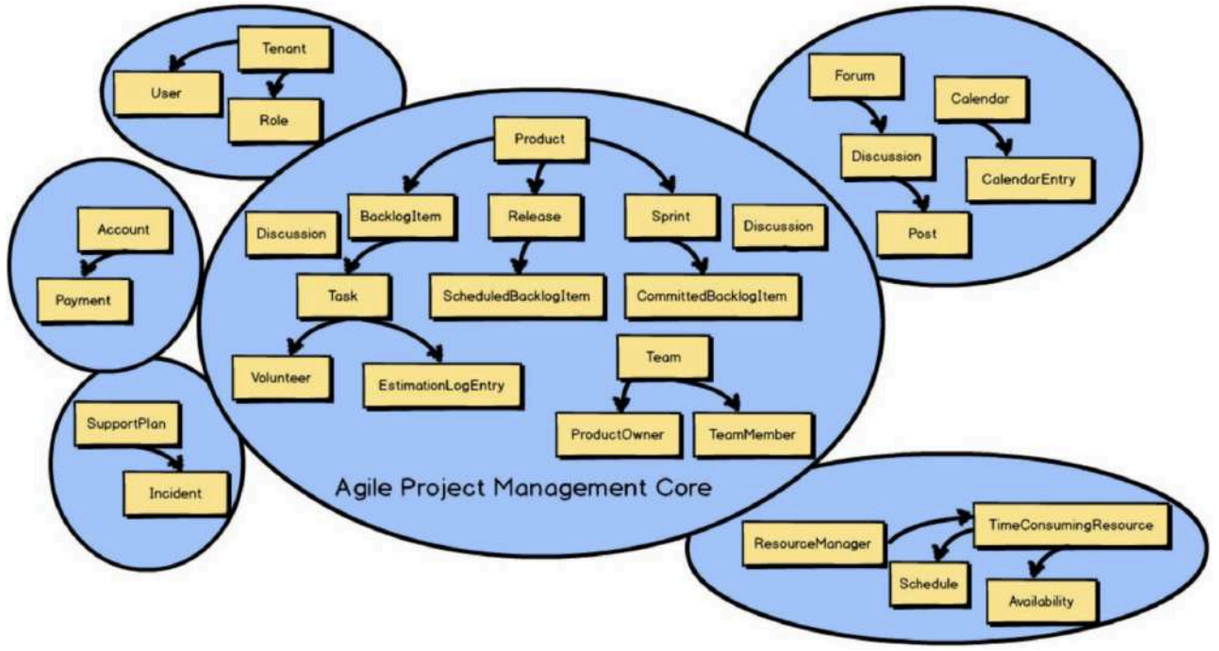
如何从需求到DDD?

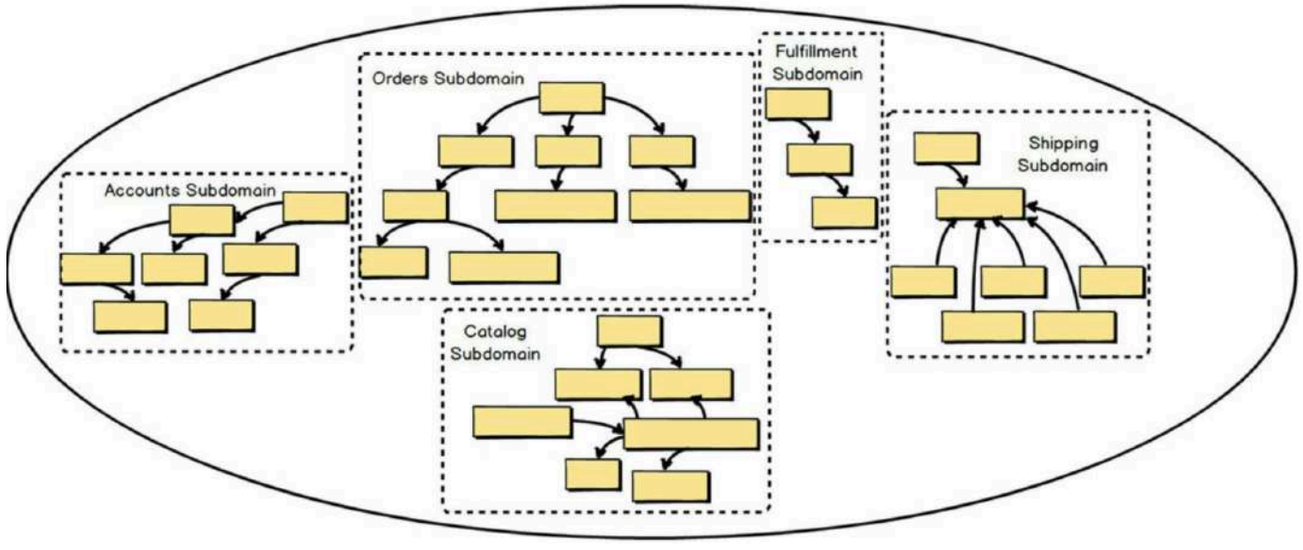
我想要一个电商平台，我可以在上面卖东西

> User Story 1.1
>
> 作为一个“网站管理员”，我想要“统计每天有多少人访问了我的网站”，以便于“我的赞助商了解我的网站会给他们带来什么收益。”

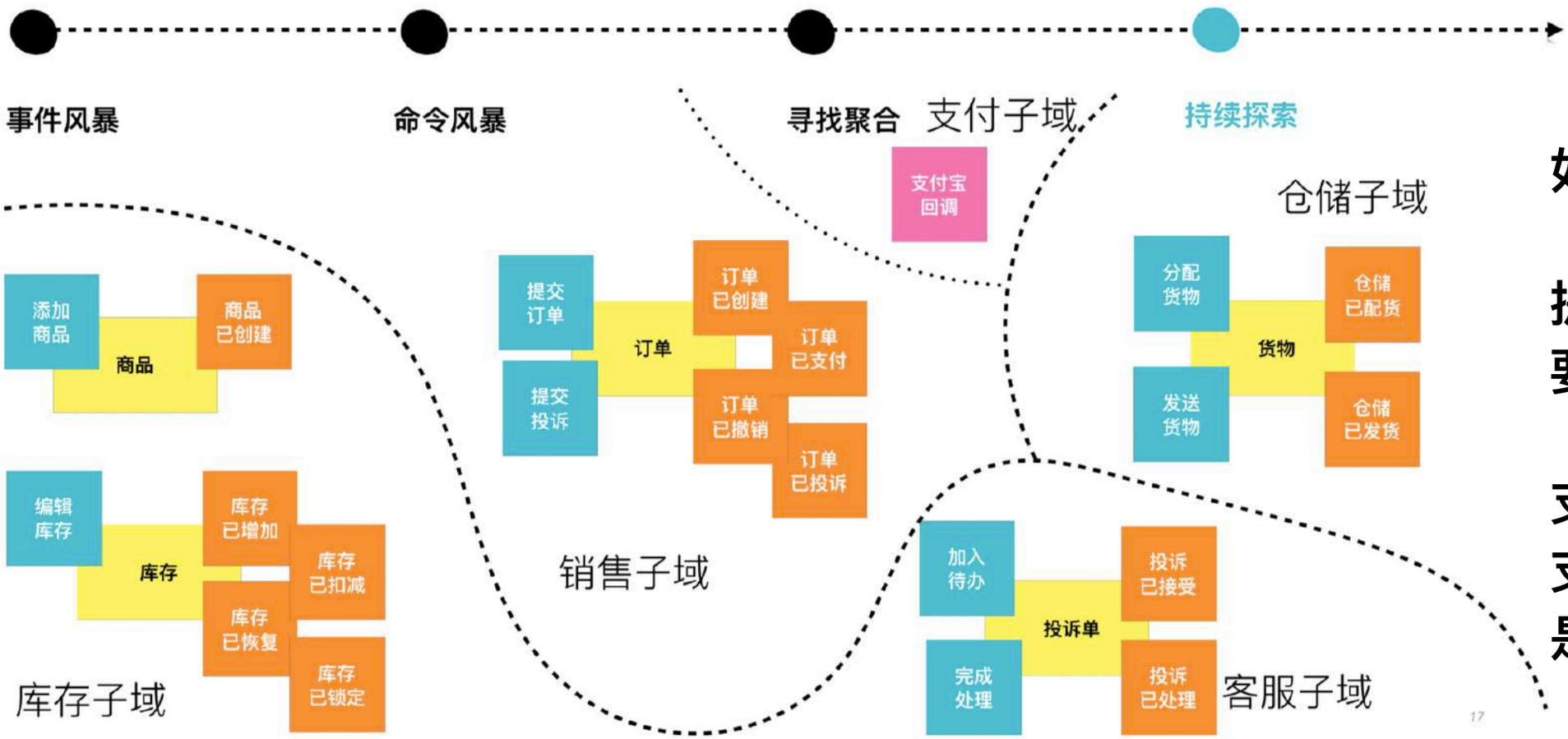
用例 ID	【需要定编码规则】	用例名	
创建者		创建日期	
最后更新者		最后更新日期	
用例体			
参与者(角色)			
功能描述	【概要地对功能进行描述，这一项不能省，它相当于高层用例】		
前置条件			
后置条件			
主干过程	【描述正常的、一般情况下的动作系列】		
分支过程	【从主干过程中由判断点分支出来的动作系列，通过动作的标号（类似于有层次的目录）与主干过程中的动作对应】		
异常	【描述主干过程、分支过程中出现异常时的动作系列】		
包含	【用例包含其他用例】		
业务规则	【用例中用到的业务规则，如计算公式】		
特别需求	【如：对于性能方面有何要求】		
假设	【用例中我们做了哪些假设】		
注意和问题	【还有哪些未确定事项】		

?





Event Storming里缺少了什么信息



如何添加商品?

提交订单的时候
要做什么检查?

支付订单的时候和
支付宝对接的逻辑
是什么样的?

魔鬼隐藏在细节之中

02 从需求到代码

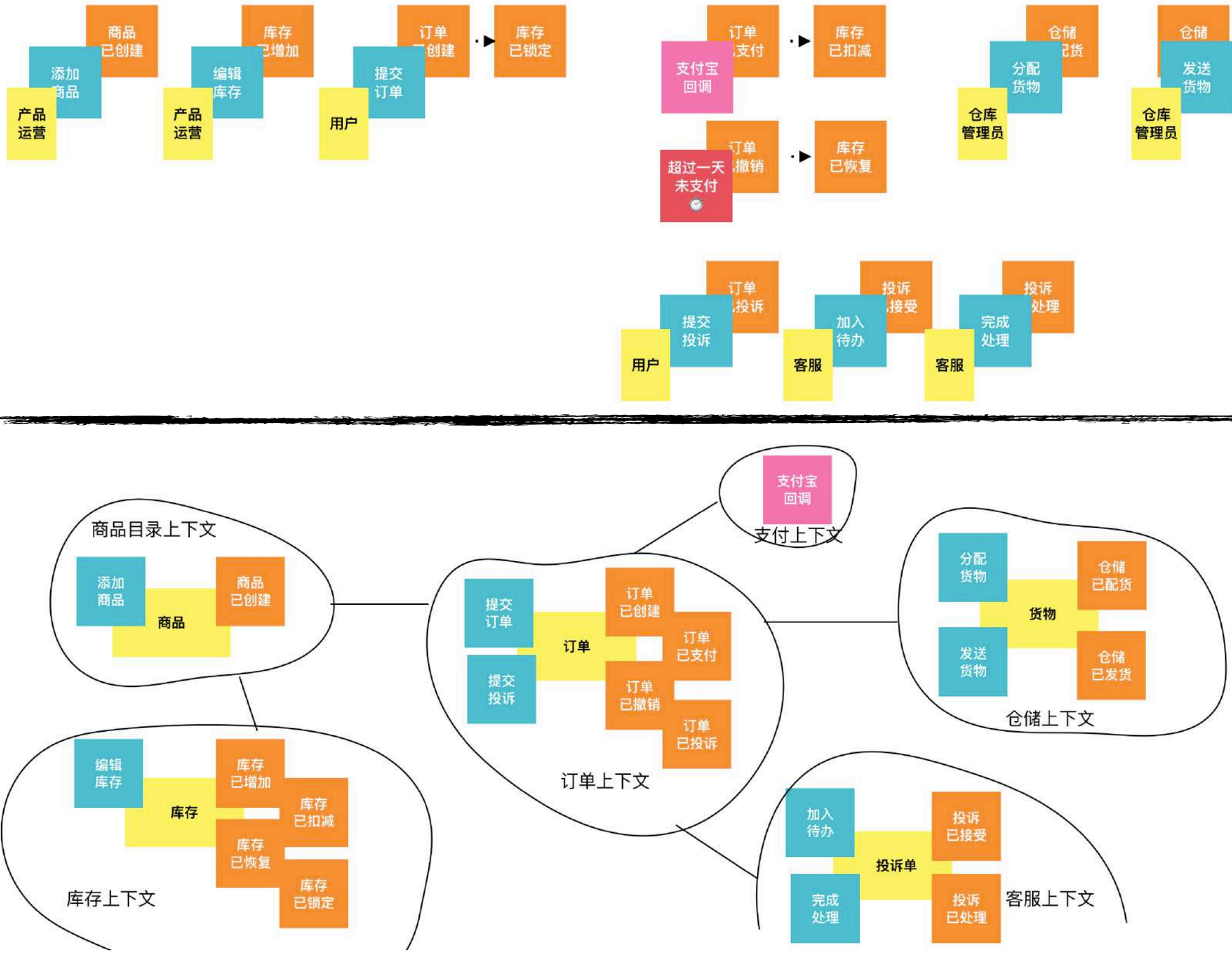
今天既然是讲落地，那么就从一个实际需求出发，我们看看如何把需求拆解成用户故事，然后如何把用户故事转变成领域故事，最后领域故事又如何变成为代码。

京西商城的需求

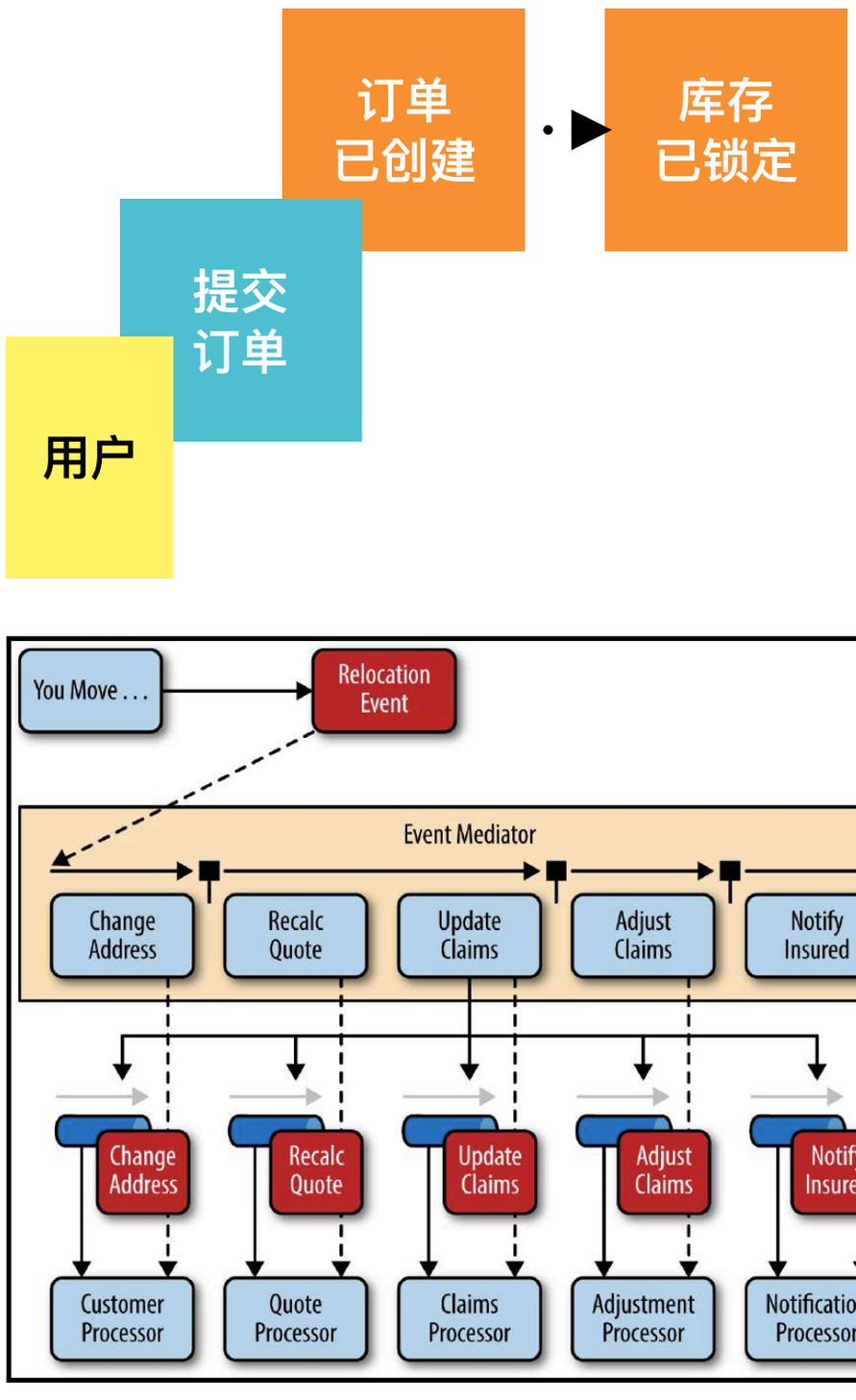
- 产品运营人员可以添加新的商品，编辑产品库存，并发布到京西商城，用户可以进行购买；

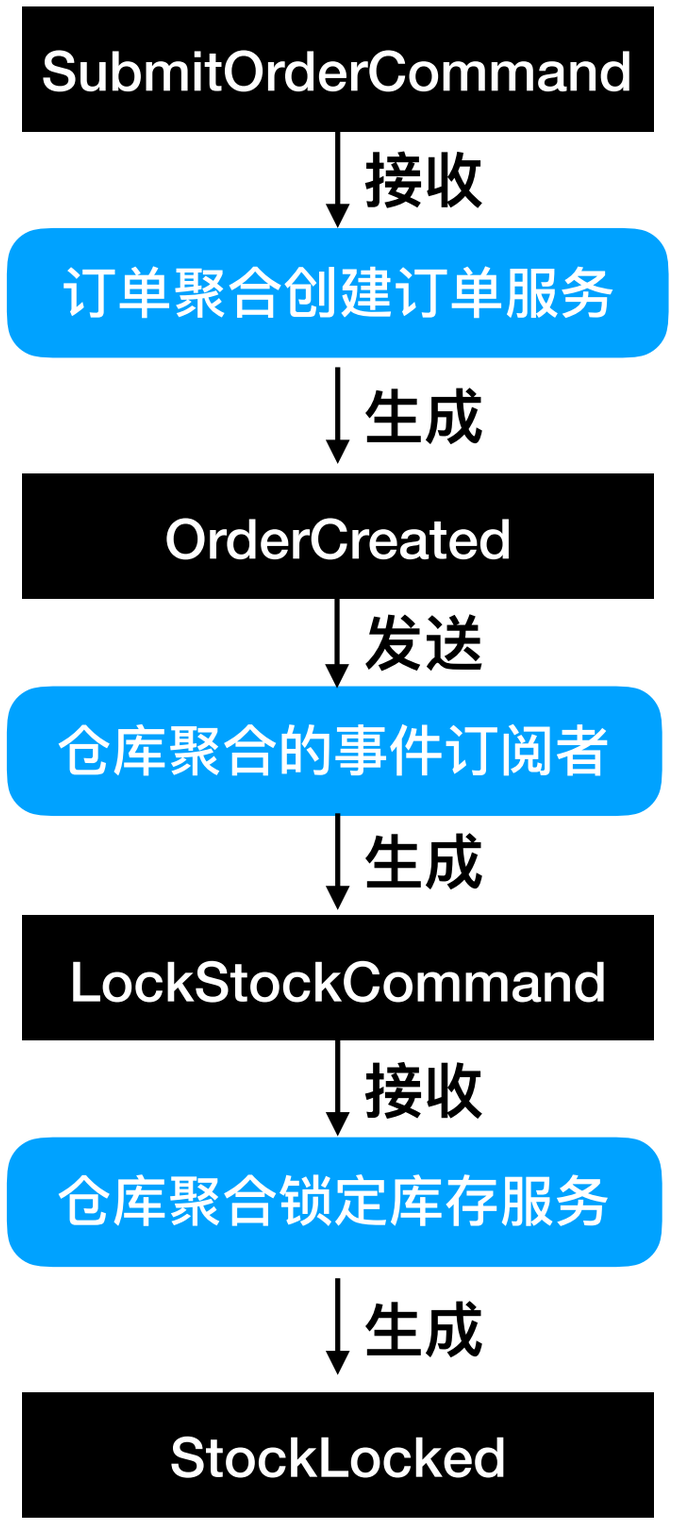
- 京西商城的买家可以查看产品可购买的库存数量，并生成购买订单，买家可以撤单，查看订单的状态，对订单进行支付（支付宝）。支付成功后，仓储管理员会收到出库请求，进行配货并发货。同时，买家可以查看物流状态以及确认收货。

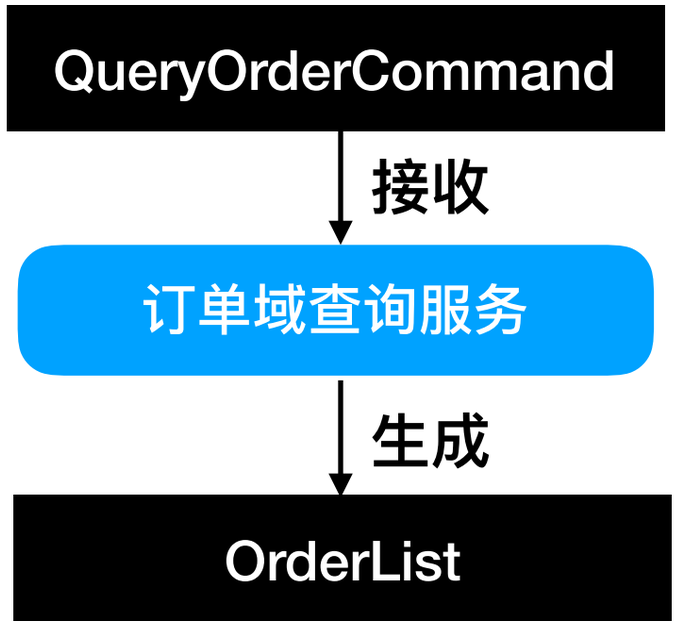
事件风暴的结果


商品
已创建
添加
商品
产品
运营
库存
已增加
编辑
库存
产品
运营
订单
已创建
提交
订单
用户
库存
已锁定
订单
已支付
支付宝
回调
库存
已扣减
订单
已撤销
超过一天
未支付
库存
已恢复
仓储
已配货
分配
货物
仓库
管理员
仓储
发送
货物
仓库
管理员
订单
已投诉
提交
投诉
用户
投诉
已接受
加入
待办
客服
投诉
已处理
完成
处理
客服
商品目录上下文
添加
商品
商品
商品
已创建
支付宝
回调
支付上下文
提交
订单
订单
订单
已创建
订单
已支付
提交
投诉
订单
已撤销
订单
已投诉
订单上下文
分配
货物
仓储
已配货
货物
发送
货物
仓储
已发货
仓储上下文
编辑
库存
库存
库存
已增加
库存
已扣减
库存
已恢复
库存
已锁定
库存上下文
加入
待办
投诉单
投诉
已接受
完成
处理
投诉
已处理
客服上下文

事件风暴得到的命令和事件就没用了吗?







需求分析到用户故事

Story6 创建订单
作为一个商城顾客
我可以确定购物车中的商品和数量，然后提交商品订单
为了在商城上购买商品

Acceptance Criteria 1
- Given：导航【购物车】
- When：确定购物车中要提交的商品和数量，然后点击提交；
- Then：订单成功提交，从商品库存中锁定订单的数量，提示气泡“商品已保存成功”
- And when：如果订单的数量超过商品库存中可以锁定的数量
- Then：订单无法成功提交，提示气泡“库存数量不够，仅剩X件”

用户故事到领域故事

点击提交按钮后

1.**订单API控制器**会收到**创建订单请求**，然后根据请求生成**创建订单命令**，然后让**创建订单应用**根据**创建订单命令**创建商品

2.**创建订单应用**收到命令后，先调用**查询商品库存服务**查找**商品库存**，

如果**商品库存**可以支持订单数量，就让**创建订单服务**根据**创建订单命令**进行订单创建，创建成功后返回成功结果给**订单API控制器**

如果商品库存不支持订单数量，返回出错结果和原因给**订单API适配器**

3.**创建订单服务**收到命令后，会把**创建订单命令**转换成**订单**，然后调用**订单仓库**进行保存，保存成功后会让事件发布器发布**订单已创建事件**

4.**事件发布器**会根据事件类型把**订单已创建事件**转发给**商品库存订阅者**，

5.**商品库存订阅者**收到事件以后会根据事件中的商品ID和订单数量创建**锁定商品库存命令**，然后给**锁定商品库存服务**进行库存锁定

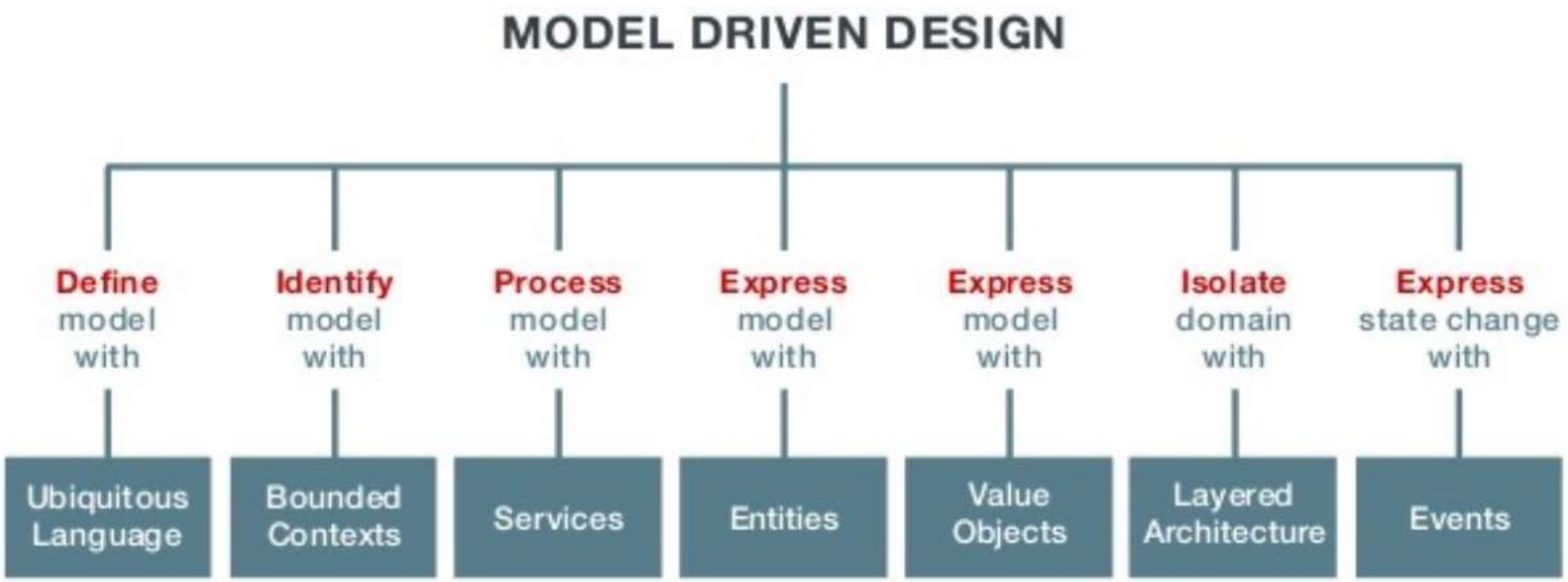
6.**锁定商品库存服务**收到命令后，会把根据命令中的商品ID从**商品库存仓库**中获取**商品库存**，然后根据让**商品库存**锁定订单数量，之后调用**商品库存仓库**进行保存，最后让**事件发布器**发布**商品库存已锁定事件**

.....

领域故事中的主要对象

领域对象	类型	命名
订单API控制器	Adapter	OrderController
创建订单命令	Command	CreateOrderCommand
创建订单应用	ApplicationService	CreateOrderApplication
查询商品库存服务	DomainService	QueryStockService
商品库存	Entity	Stock
订单	Entity	Order
订单仓库	Repository	OrderRepository
订单已创建	Event	OrderCreated
商品库存订阅者	Subscriber	StockEventSubscriber
锁定商品库存命令	Command	LockStockCommand
锁定商品库存服务	DomainService	LockStockService
商品库存仓库	Repository	StockRepository
商品库存已锁定事件	Event	StockLocked

领域模型


MODEL DRIVEN DESIGN
Define model with
Ubiquitous Language
Identify model with
Bounded Contexts
Process model with
Services
Express model with
Entities
Express model with
Value Objects
Isolate domain with
Layered Architecture
Express state change with
Events

如何区分实体和值对象

实体

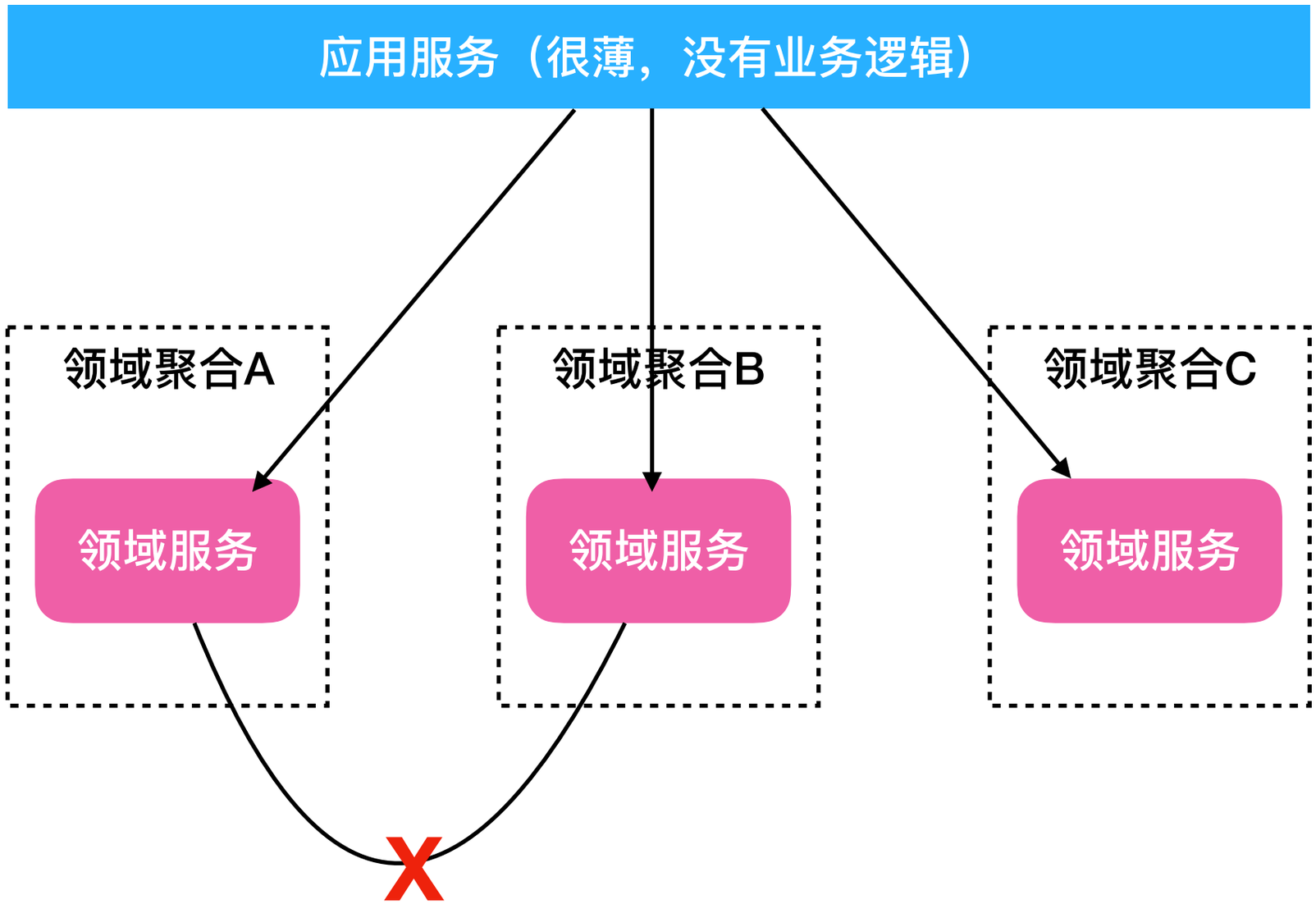
- 具有生命周期
- 有唯一标识
- 通过Id判断相等性
- 增删改查/持久化
- 可变
- 比如Order/Car

VS

值对象

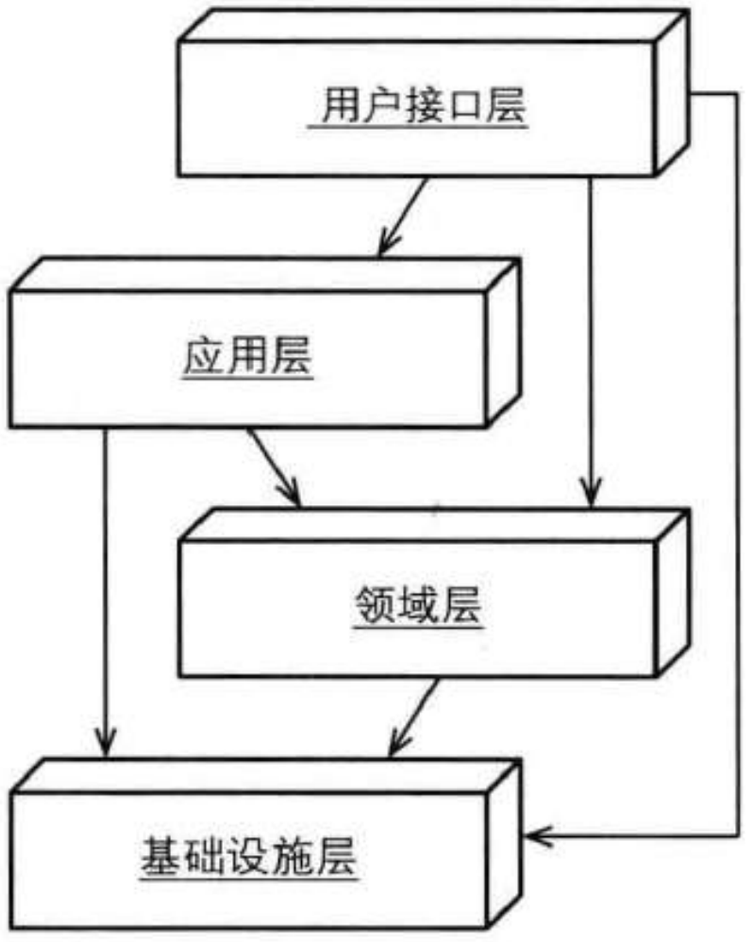
- 起描述性作用
- 无唯一标识
- 通过属性判断相等性
- 实现Equals()方法
- 即时创建/用完即扔
- 不可变(Immutable)
- 比如Address/Color

如何区分应用服务和领域服务



领域服务之间最好不要相互直接调用

使用什么架构



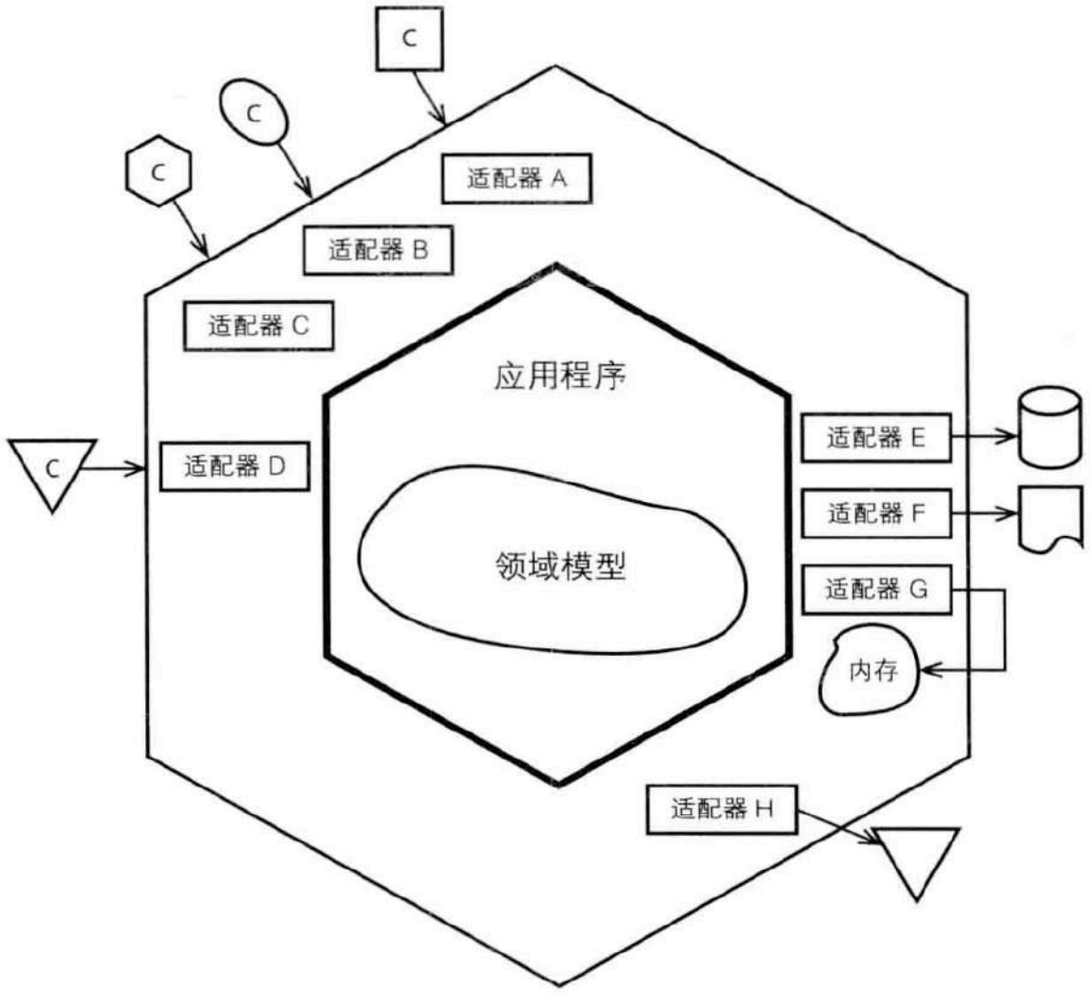
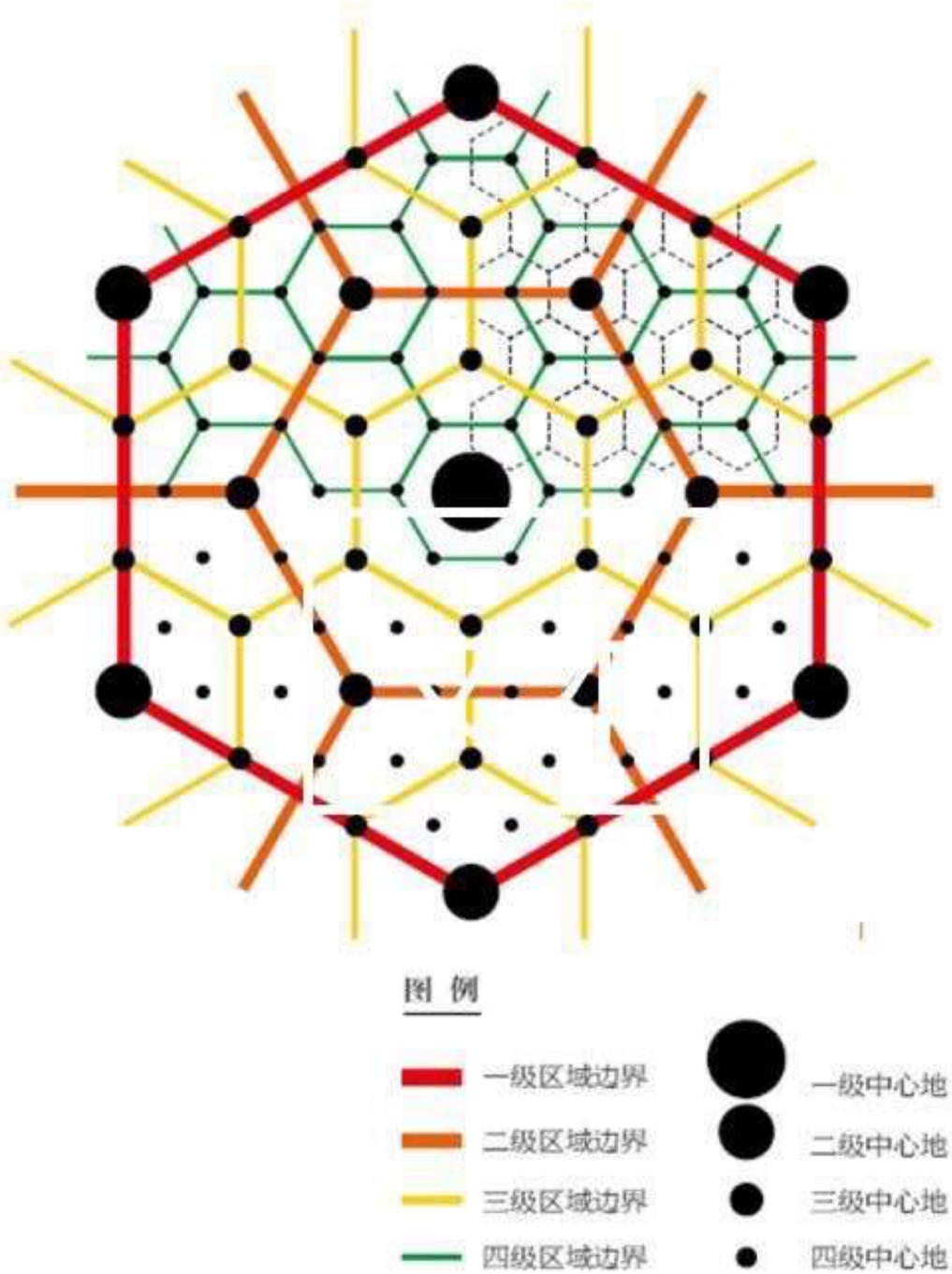
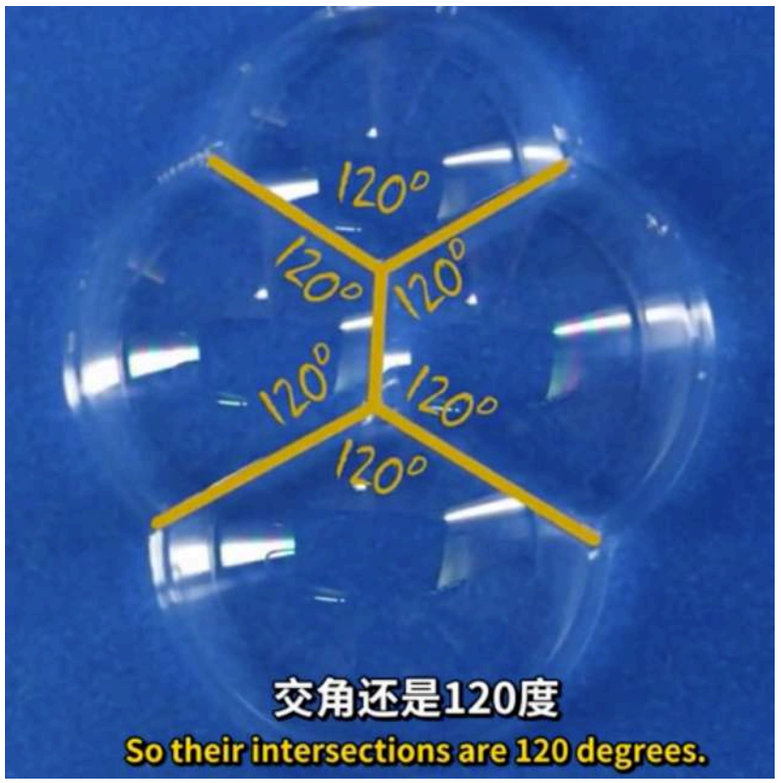


图4.4六边形架构也称为端口与适配器。对于每种外界类型，都有一个适配器与之相对应。外界通过应用层API与内部进行交互。

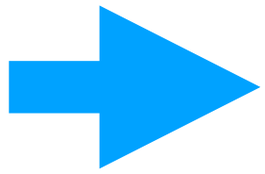
六边形架构为什么是六条边

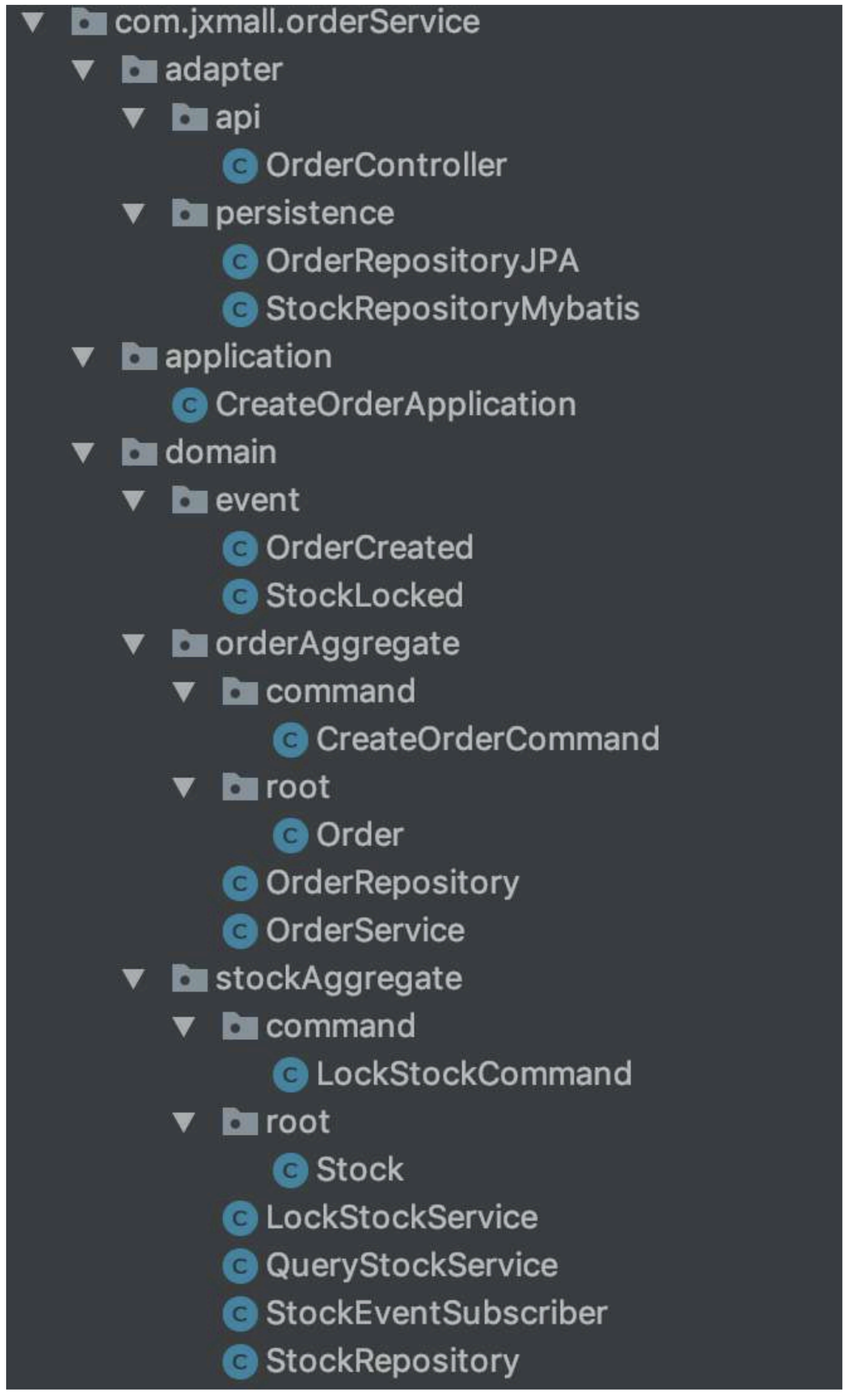




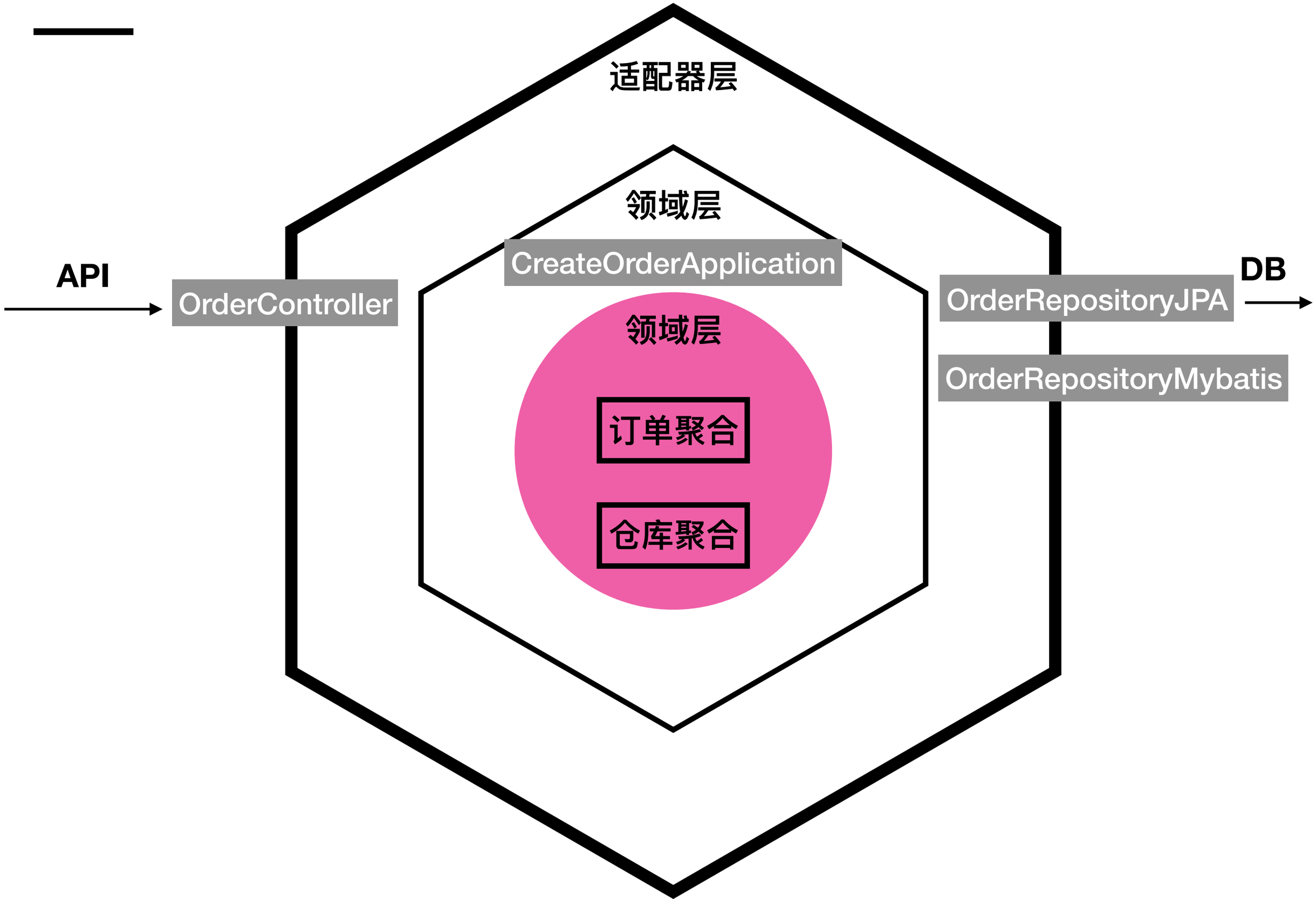
领域故事中的对象怎么对应到架构上

命名
OrderController
CreateOrderCommand
CreateOrderApplication
QueryStockService
Stock
Order
OrderRepository
OrderCreated
StockEventSubscriber
LockStockCommand
LockStockService
StockRepository
StockLocked





领域层不应该有任何外部依赖细节



领域故事如何变成代码

创建订单服务收到命令后，会把**创建订单命令**转换成**订单**，然后调用**订单仓库**进行保存，保存成功后会让事件发布器发布**订单已创建事件**

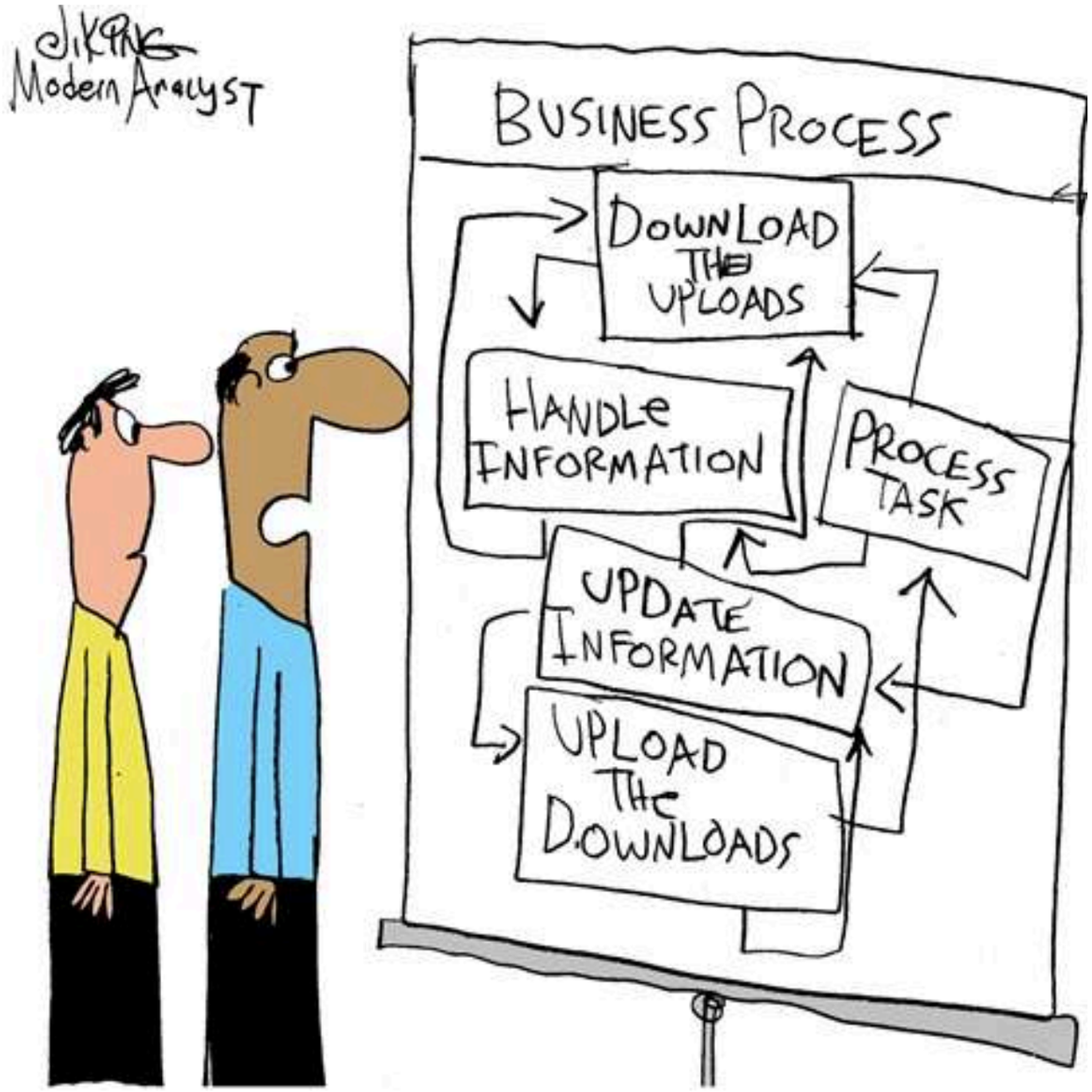
```
public class CreateOrderService {
    OrderRepository orderRepository;

    public CommandResult createOrder(CreateOrderCommand command) {
        Order order = command.toOrder();
        orderRepository.save(order);
        OrderCreated orderCreated = new OrderCreated(order.getId(),
                                                     order.getProductId(),
                                                     order.getOrderCount());
        EventPublisher.publish(orderCreated);
        return new CommandResult();
    }
}
```

03 当需求发生变化时

软件中最痛苦的就是需求经常发生变化，这也是DDD想要解决的问题之一，

为什么需求会变化?



"Charlie, did you really get these details from the customer?"

需求提出者不一定想好了

需求提出者的问题变化了

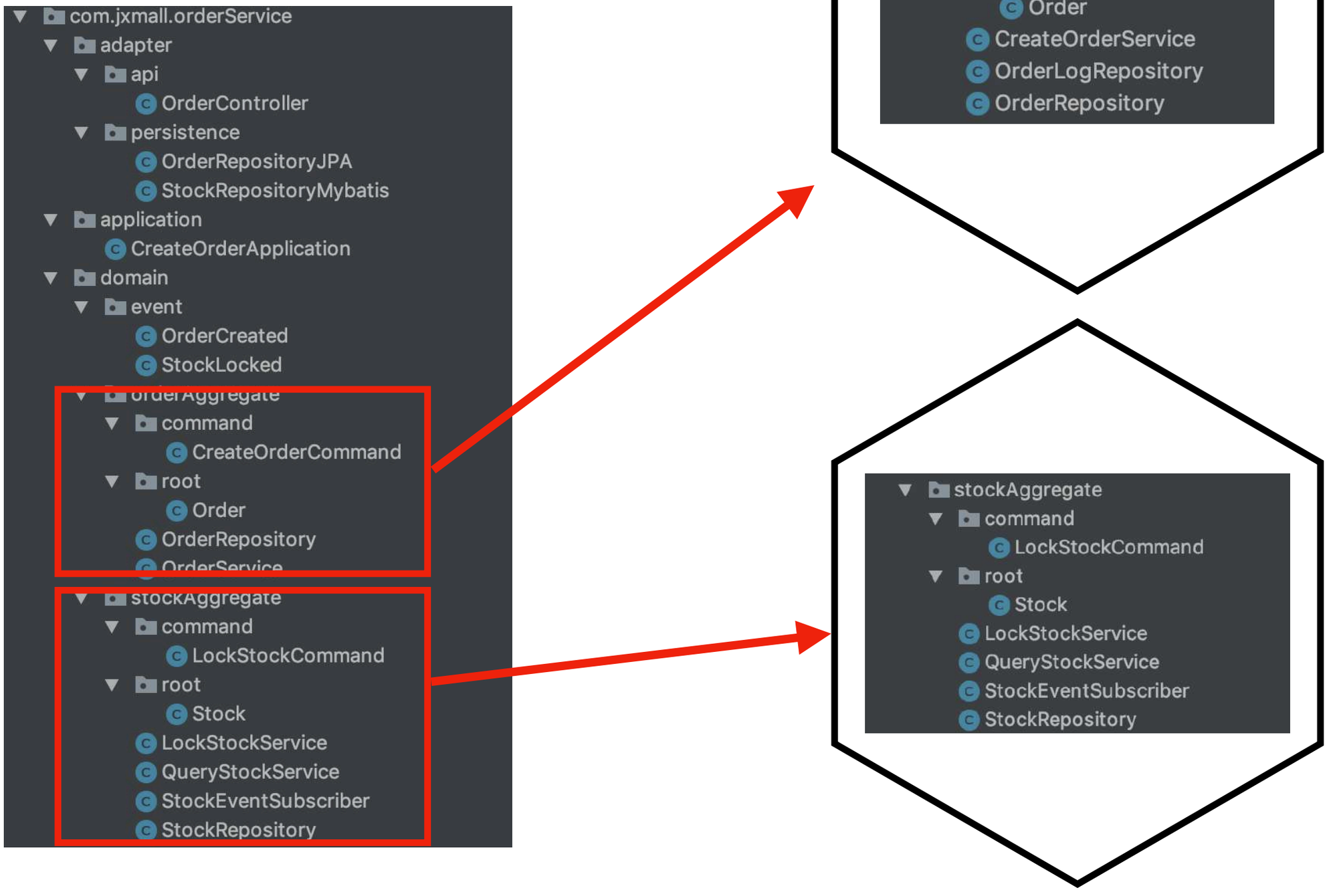
沟通过程信息丢失

需要增加一些业务逻辑时

Acceptance Criteria 1
- Given：导航【购物车】
- When：确定购物车中要提交的商品和数量，然后点击提交；
- Then：订单成功创建，从商品库存中锁定订单的数量，订单日志需要记录这笔订单创建，最后提示气泡“商品已保存成功”
- And when：如果订单的数量超过商品库存中可以锁定的数量
- Then：订单无法成功提交，提示气泡“库存数量不够，仅剩X件”

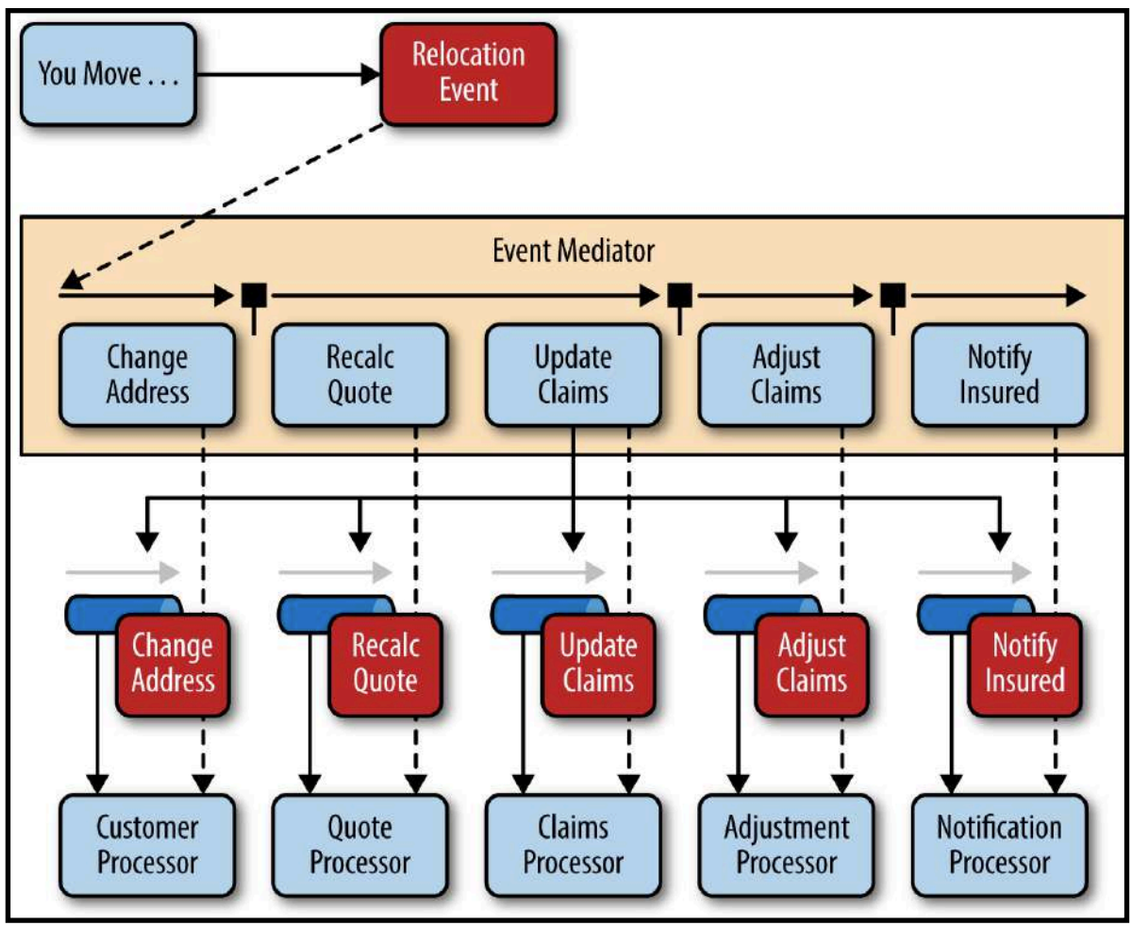
3.**创建订单服务**收到命令后，会把**创建订单命令**转换成**订单**，然后调用**订单仓库**进行保存，保存成功后会让事件发布器发布**订单已创建事件**
4.**事件发布器**会根据事件类型把**订单已创建事件**转发给**商品库存订阅者**，**订单日志订阅者**
5....
6....
7.**订单日志订阅者**收到**订单已创建事件**后，会根据事件创建**添加订单日志命令**，然后调用**添加订单日志服务**来进行添加
8.**添加订单日志服务**收到命令后，会把**添加订单日志命令**转换成**订单日志**，然后调用**订单日志仓库**保存**订单日志**

需要拆分时

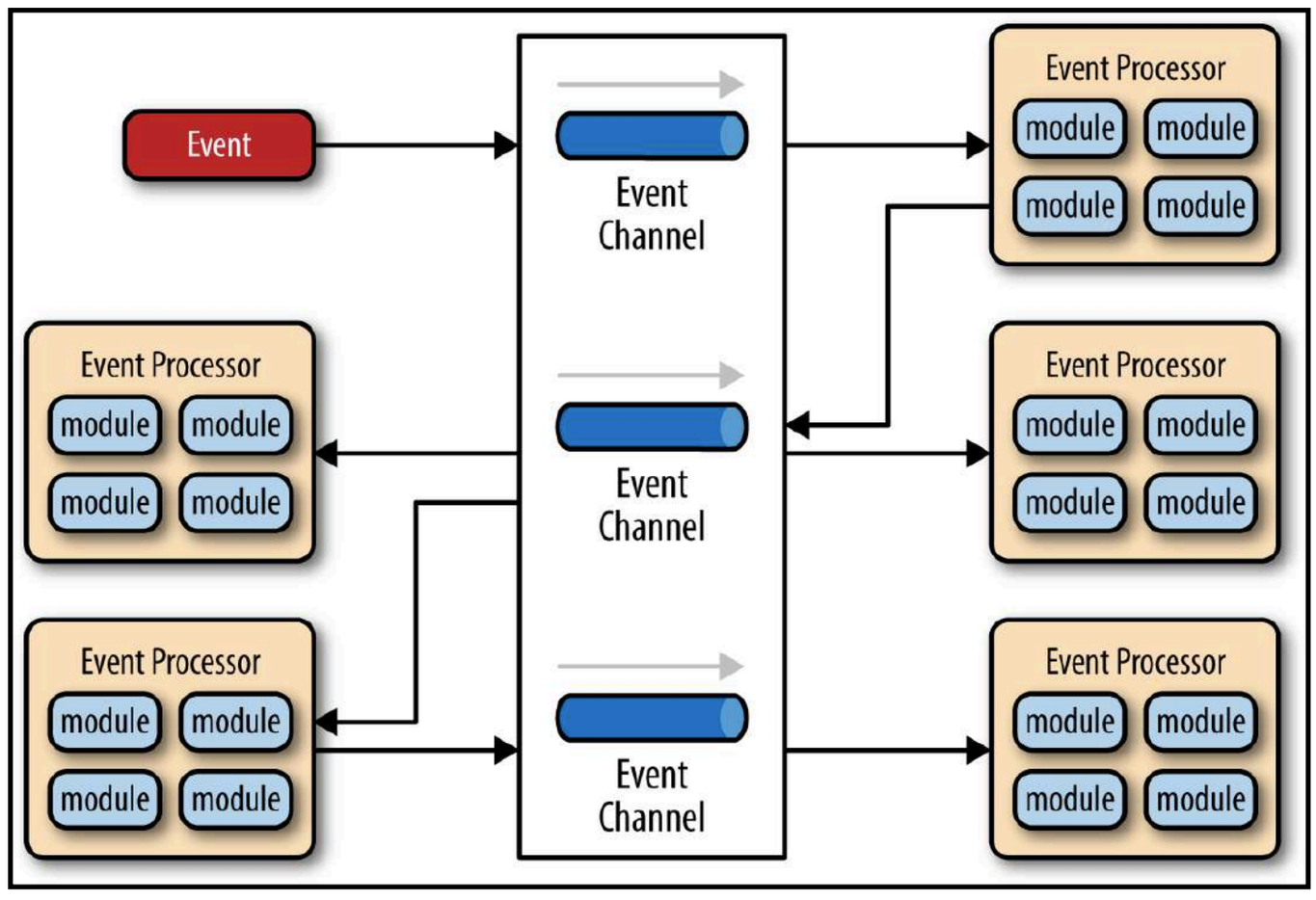

com.jxmall.orderService
adapter
api
OrderController
persistence
OrderRepositoryJPA
StockRepositoryMybatis
application
CreateOrderApplication
domain
event
OrderCreated
StockLocked
orderAggregate
command
CreateOrderCommand
root
Order
OrderRepository
OrderService
stockAggregate
command
LockStockCommand
root
Stock
LockStockService
QueryStockService
StockEventSubscriber
StockRepository
orderAggregate
command
root
Order
CreateOrderService
OrderLogRepository
OrderRepository
stockAggregate
command
LockStockCommand
root
Stock
LockStockService
QueryStockService
StockEventSubscriber
StockRepository

坚持使用事件进行服务间交互

Mediator拓扑架构



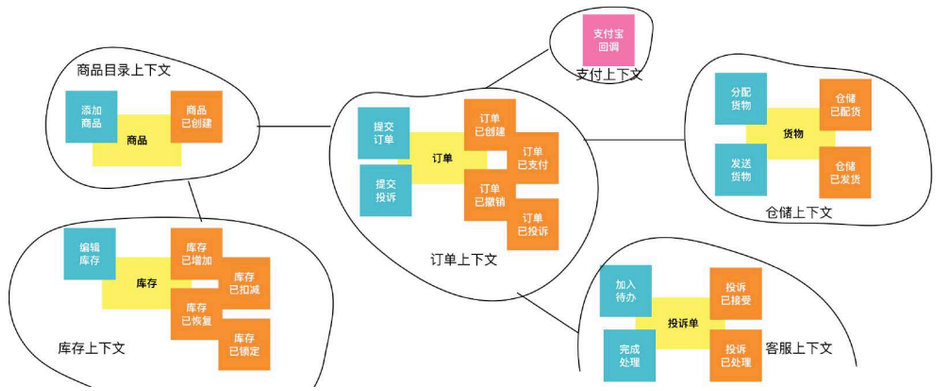
Broker拓扑架构



04 把大象塞进冰箱分几步

打开冰箱，塞进大象，关上冰箱

把需求落地需要几步



创建订单服务收到命令后，会把**创建订单命令**转换成**订单**，然后调用**订单仓库**进行保存，保存成功后会让事件发布器发布**订单已创建事件**

划界

用事件风暴划分业务边界

透视

用实现细节编写领域故事

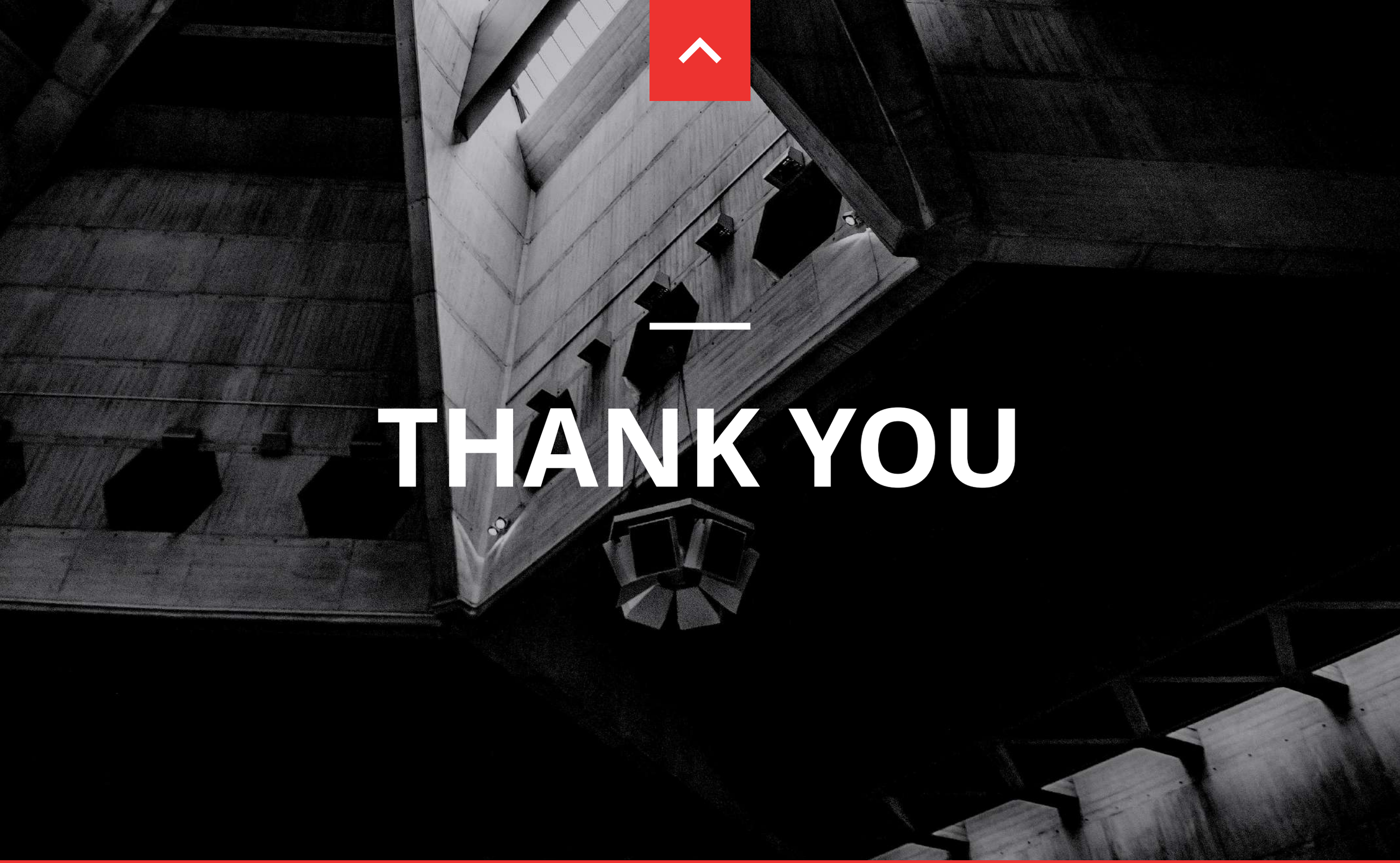
转码

把领域故事转换成代码

```java
public class CreateOrderService {
    OrderRepository orderRepository;

    public CommandResult createOrder(CreateOrderCommand command) {
        Order order = command.toOrder();
        orderRepository.save(order);
        OrderCreated orderCreated = new OrderCreated(order.getId(),
                                                     order.getProductId(),
                                                     order.getOrderCount());

        EventPublisher.publish(orderCreated);
        return new CommandResult();
    }
}
```



THANK YOU

DDD CHINA