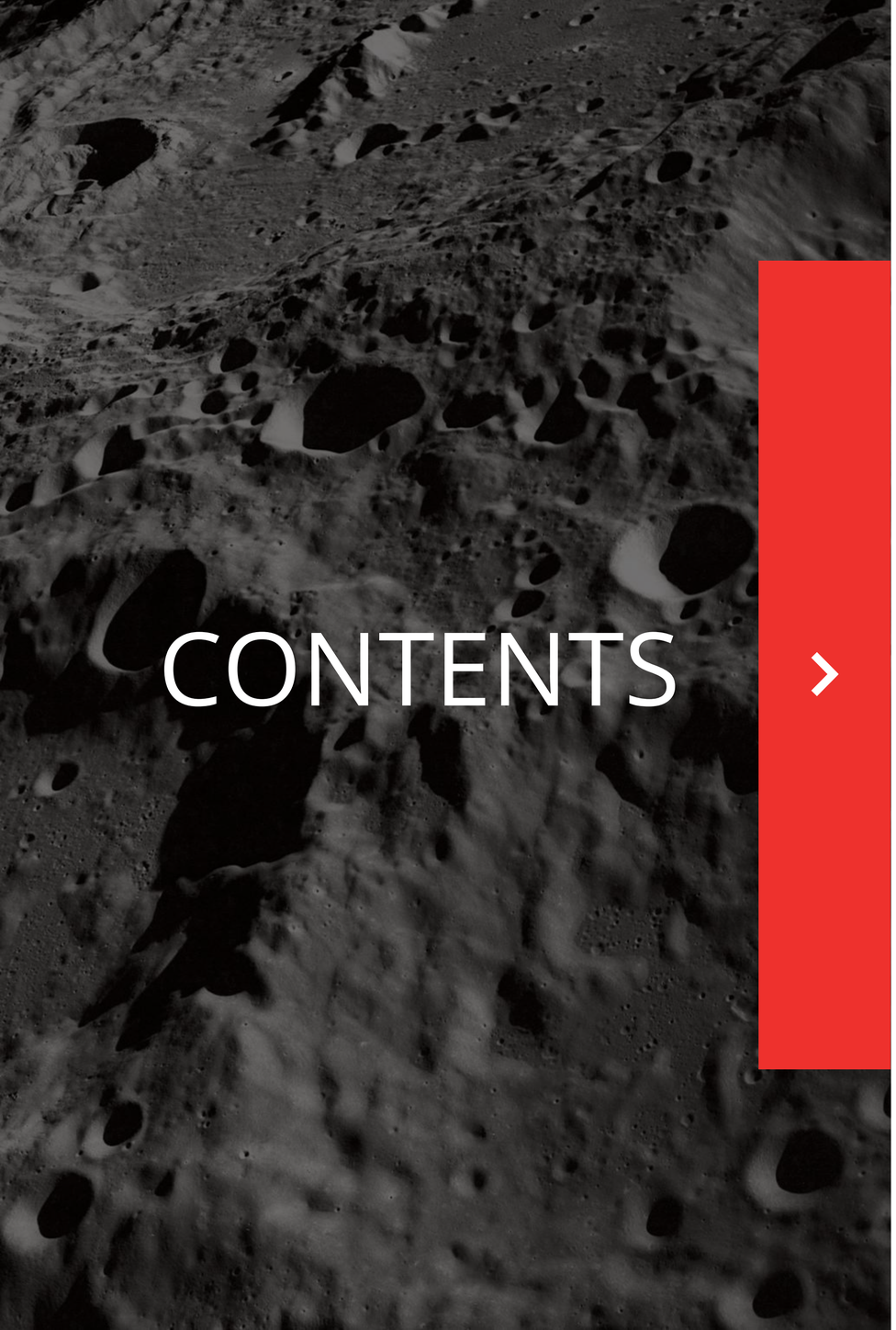
DDD CHINA

领域驱动设计揭秘

张逸



CONTENTS

1 DDD vs. DDD

2 DDD的黑铁时代与黄金时代

3 单体架构是邪恶的吗

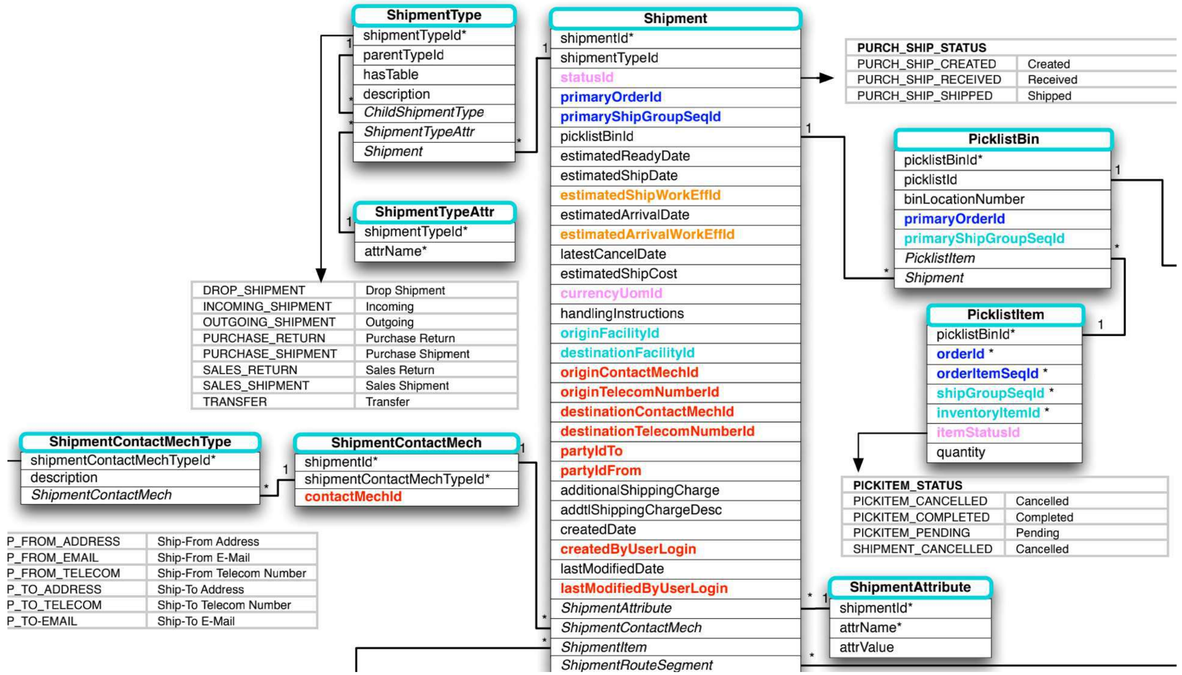
4 DDD的不足与DDD-UP

DDD CHINA

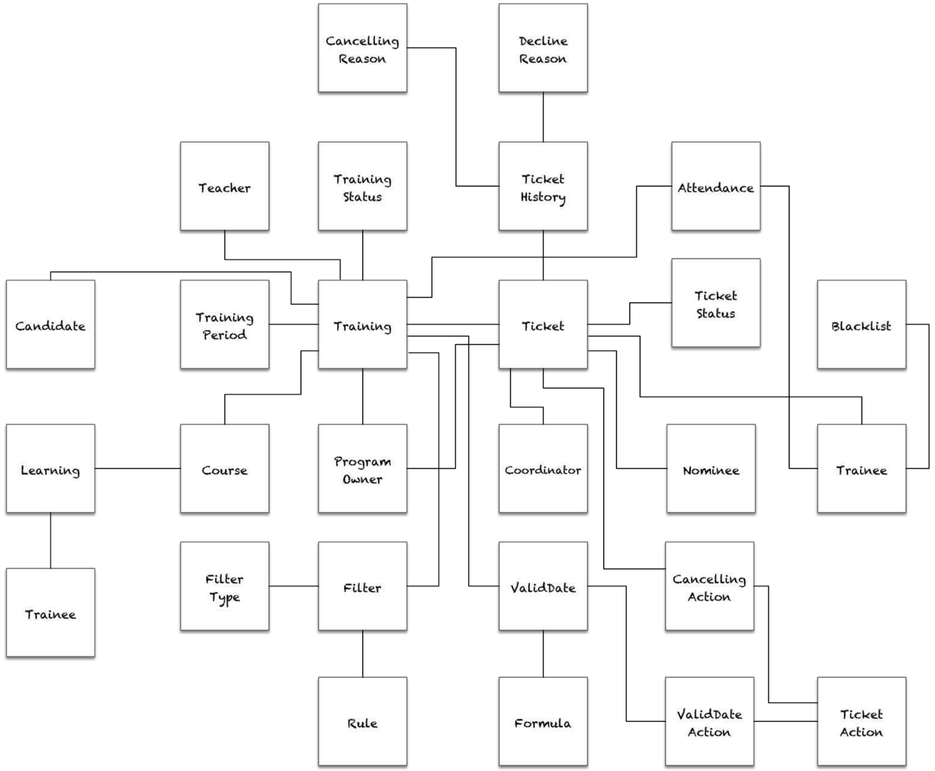
1

DDD vs. DDD

Data Driven Design vs. Domain Driven Design



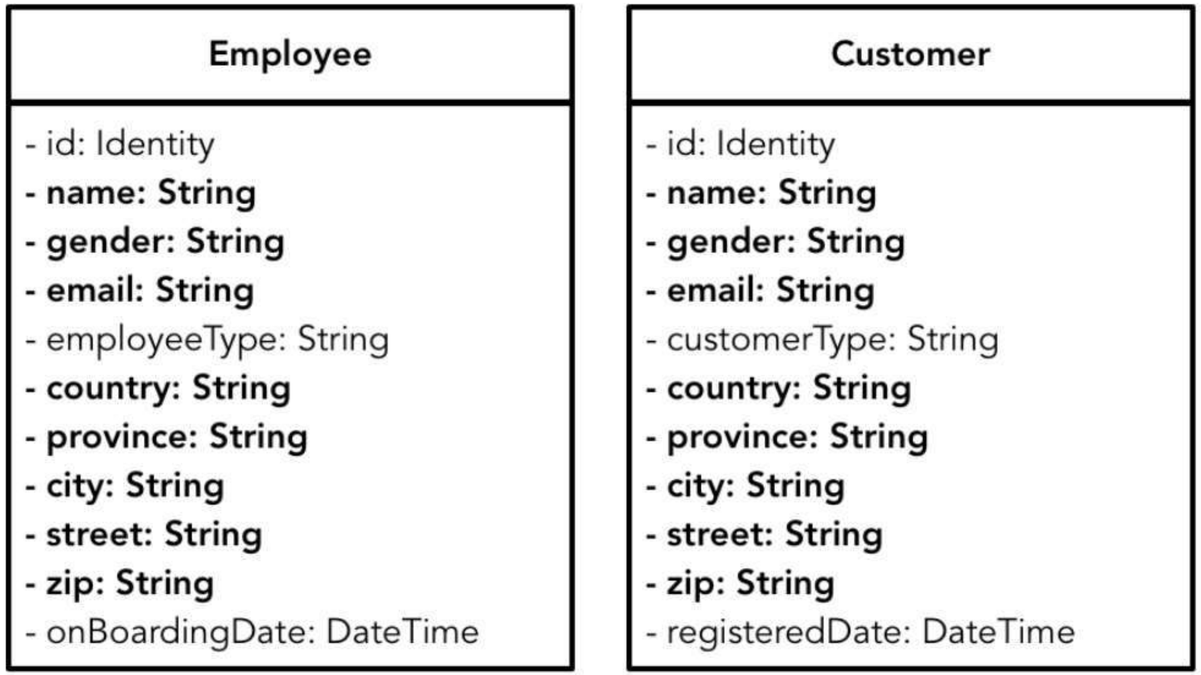
Data Driven Design



Domain Driven Design

数据驱动设计会带来什么

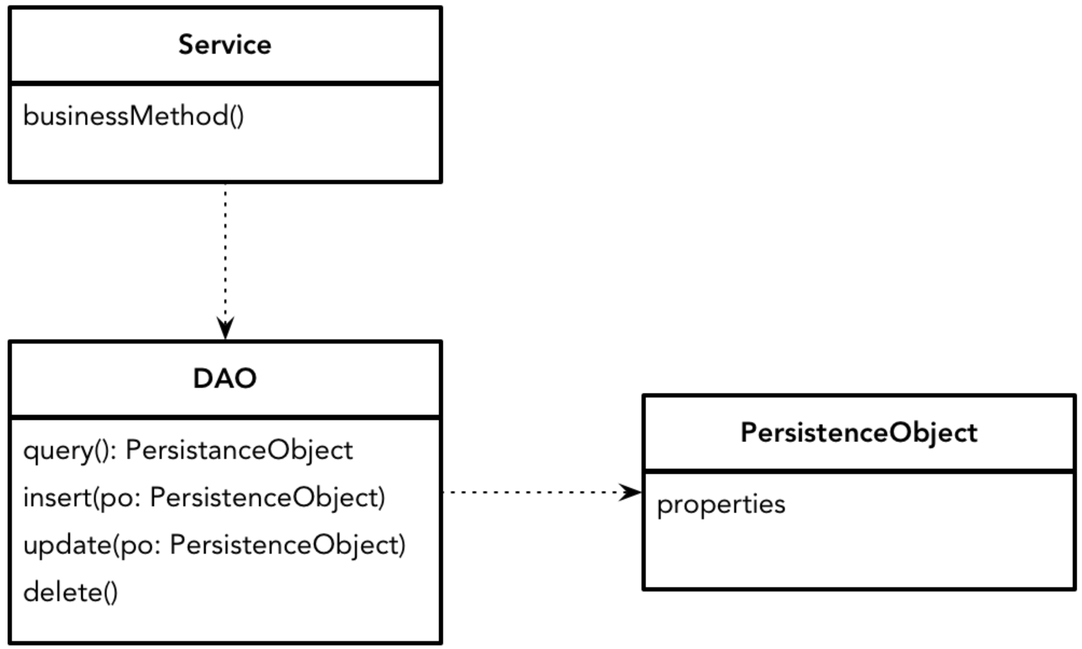
```
# 员工数据表
CREATE TABLE employees(
    id VARCHAR(50) NOT NULL,
    name VARCHAR(20) NOT NULL,
    gender VARCHAR(10),
    email VARCHAR(50) NOT NULL,
    employeeType SMALLINT NOT NULL,
    country VARCHAR(20),
    province VARCHAR(20),
    city VARCHAR(20),
    street VARCHAR(100),
    zip VARCHAR(10),
    onBoardingDate DATE NOT NULL,
    createdTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updatedTime TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY(id)
);
```



贫血模型

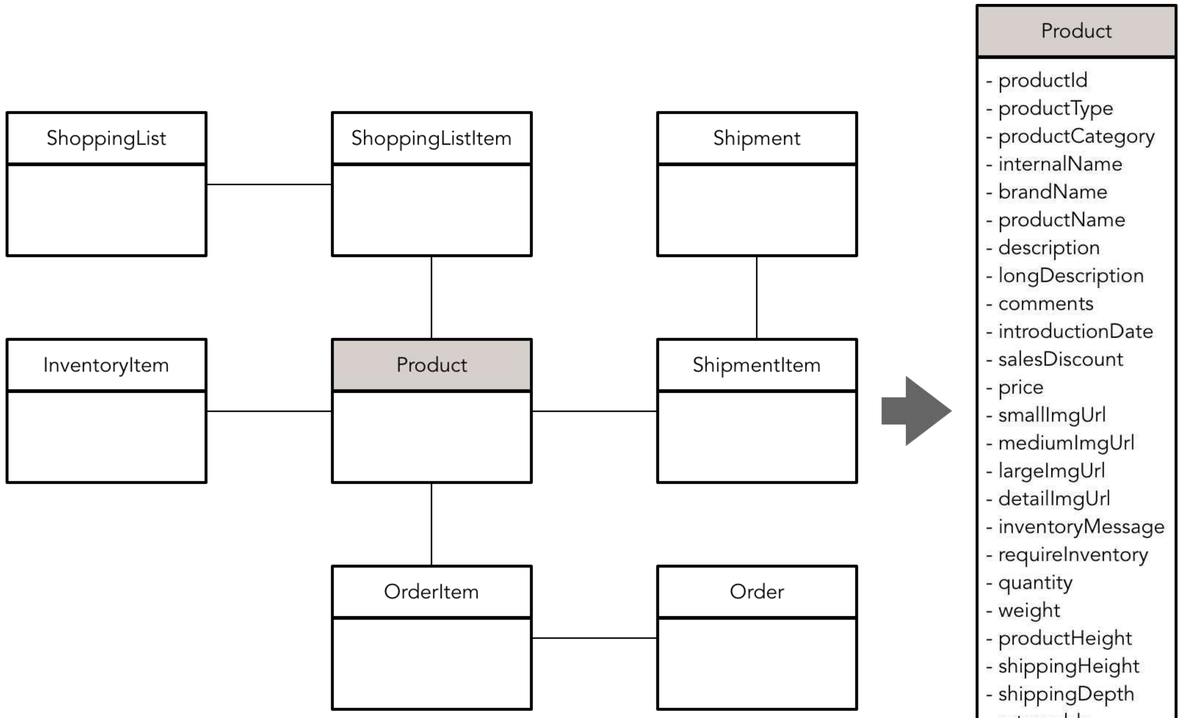
数据驱动设计会带来什么

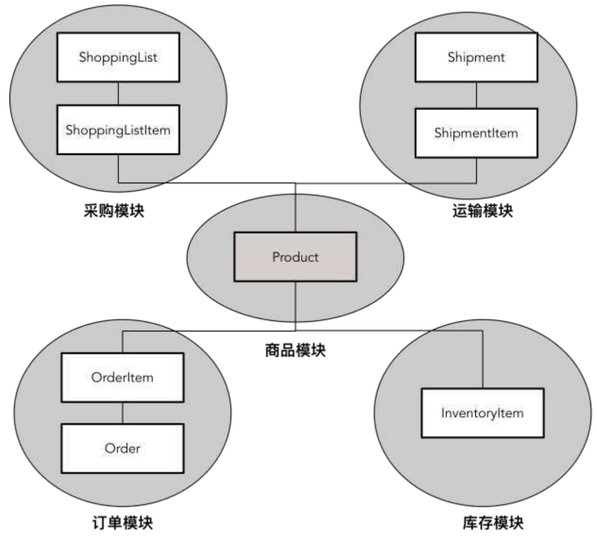
```java
/** Service for creating a new order */
public static Map<String, Object> createOrder(DispatchContext ctx, Map<String, ? extends Object> context)
    Delegator delegator = ctx.getDelegator();
    LocalDispatcher dispatcher = ctx.getDispatcher();
    Security security = ctx.getSecurity();
    List<GenericValue> toBeStored = new LinkedList<GenericValue>();
    Locale locale = (Locale) context.get("locale");
    Map<String, Object> successResult = ServiceUtil.returnSuccess();

    GenericValue userLogin = (GenericValue) context.get("userLogin");
    // get the order type
    String orderTypeId = (String) context.get("orderTypeId");
    String partyId = (String) context.get("partyId");
    String billFromVendorPartyId = (String) context.get("billFromVendorPartyId");

    // check security permissions for order:
    //  SALES ORDERS - if userLogin has ORDERMGR_SALES_CREATE or ORDERMGR_CREATE permission, or if it is sa
    //                 if it is an AGENT (sales rep) creating an order for his customer
    //  PURCHASE ORDERS - if there is a PURCHASE_ORDER permission
    Map<String, Object> resultSecurity = new HashMap<String, Object>();
    boolean hasPermission = OrderServices.hasPermission(orderTypeId, partyId, userLogin, "CREATE", securit
    // final check - will pass if userLogin's partyId = partyId for order or if userLogin has ORDERMGR_CRE
    // jacopoc: what is the meaning of this code block? FIXME
    if (!hasPermission) {
        partyId = ServiceUtil.getPartyIdCheckSecurity(userLogin, security, context, resultSecurity, "ORDER
        if (resultSecurity.size() > 0) {
            return resultSecurity;
        }
    }

    // get the product store for the order, but it is required only for sales orders
    String productStoreId = (String) context.get("productStoreId");
    GenericValue productStore = null;
    if ((orderTypeId.equals("SALES_ORDER")) && (UtilValidate.isNotEmpty(productStoreId))) {
        try {
            productStore = EntityQuery.use(delegator).from("ProductStore").where("productStoreId", product
        } catch (GenericEntityException e) {
            return ServiceUtil.returnError(UtilProperties.getMessage(resource_error,
                    "OrderErrorCouldNotFindProductStoreWithID",UtilMisc.toMap("productStoreId",productStor
        }
    }
```



事务脚本

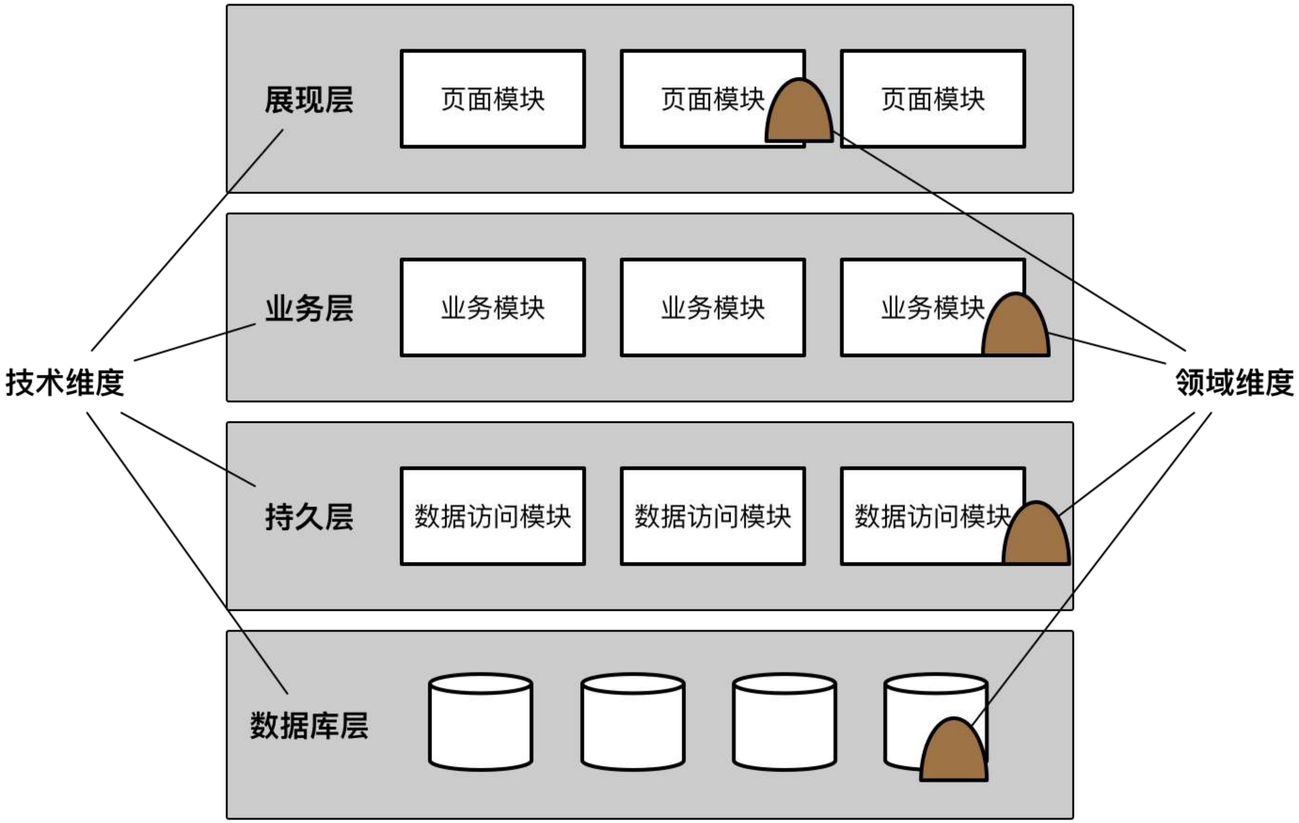
数据驱动设计会带来什么





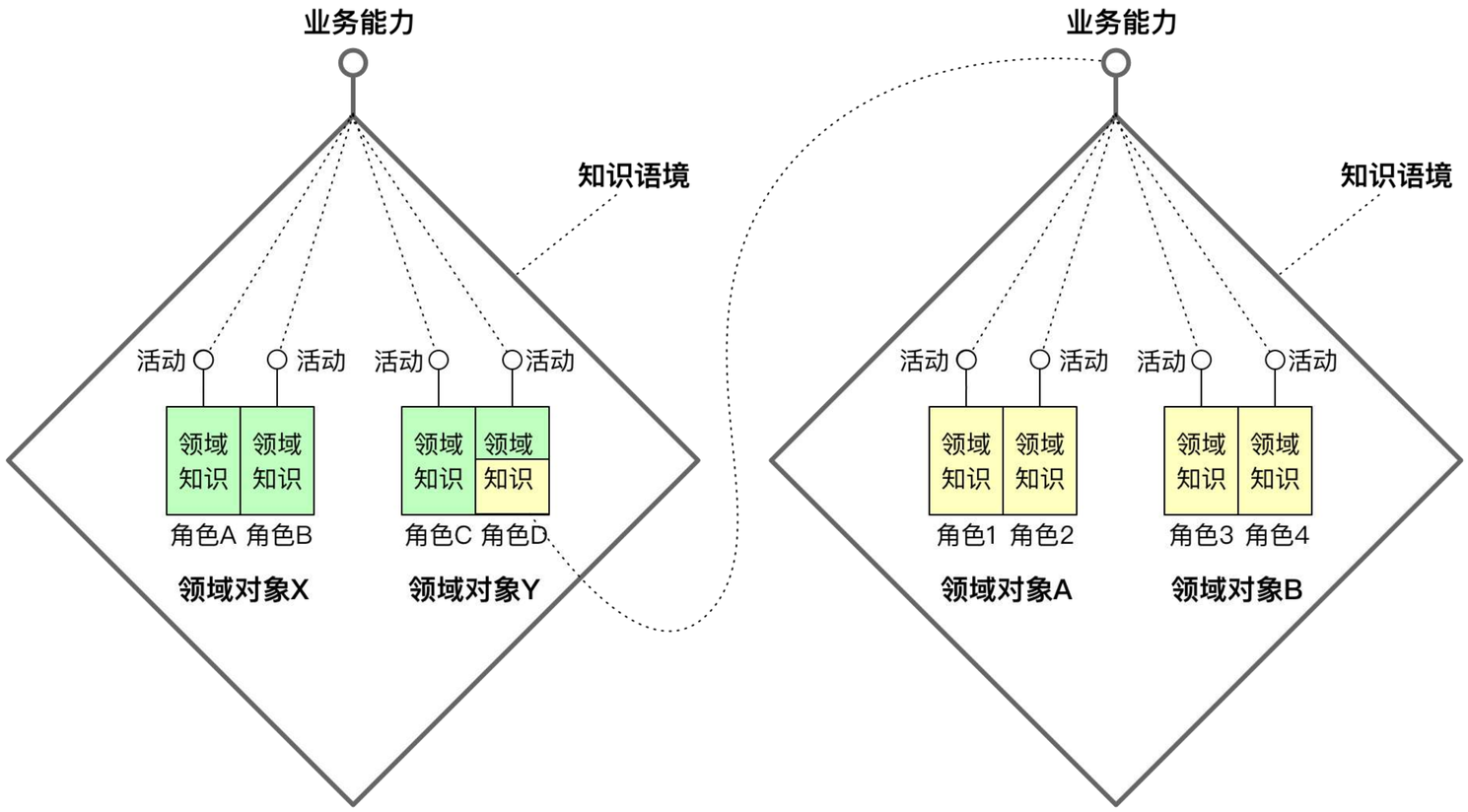
模型没有上下文的边界

数据驱动设计会带来什么



技术维度与领域维度的变化方向不一致

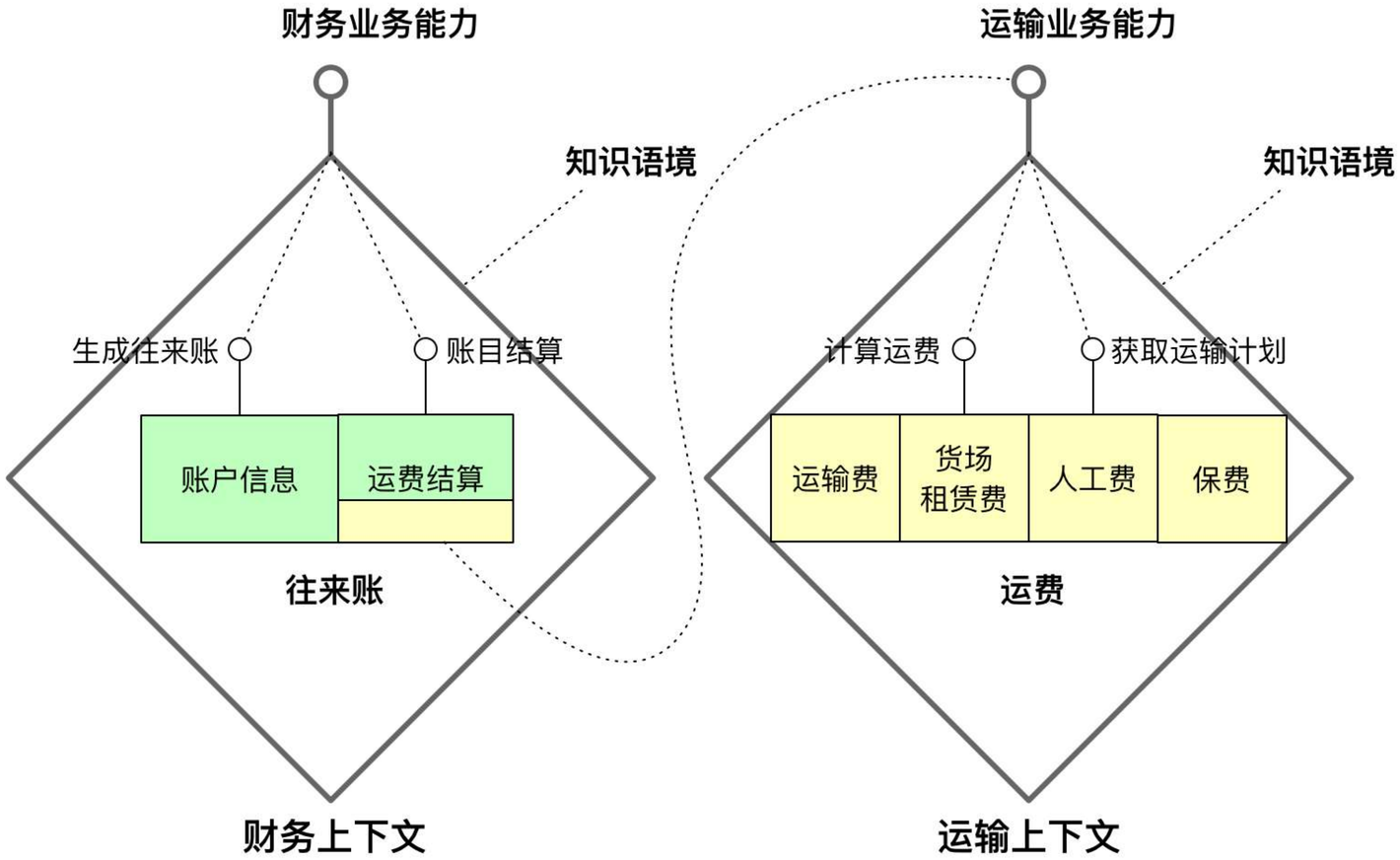
领域驱动设计引入了限界上下文



限界上下文是领域模型的知识语境

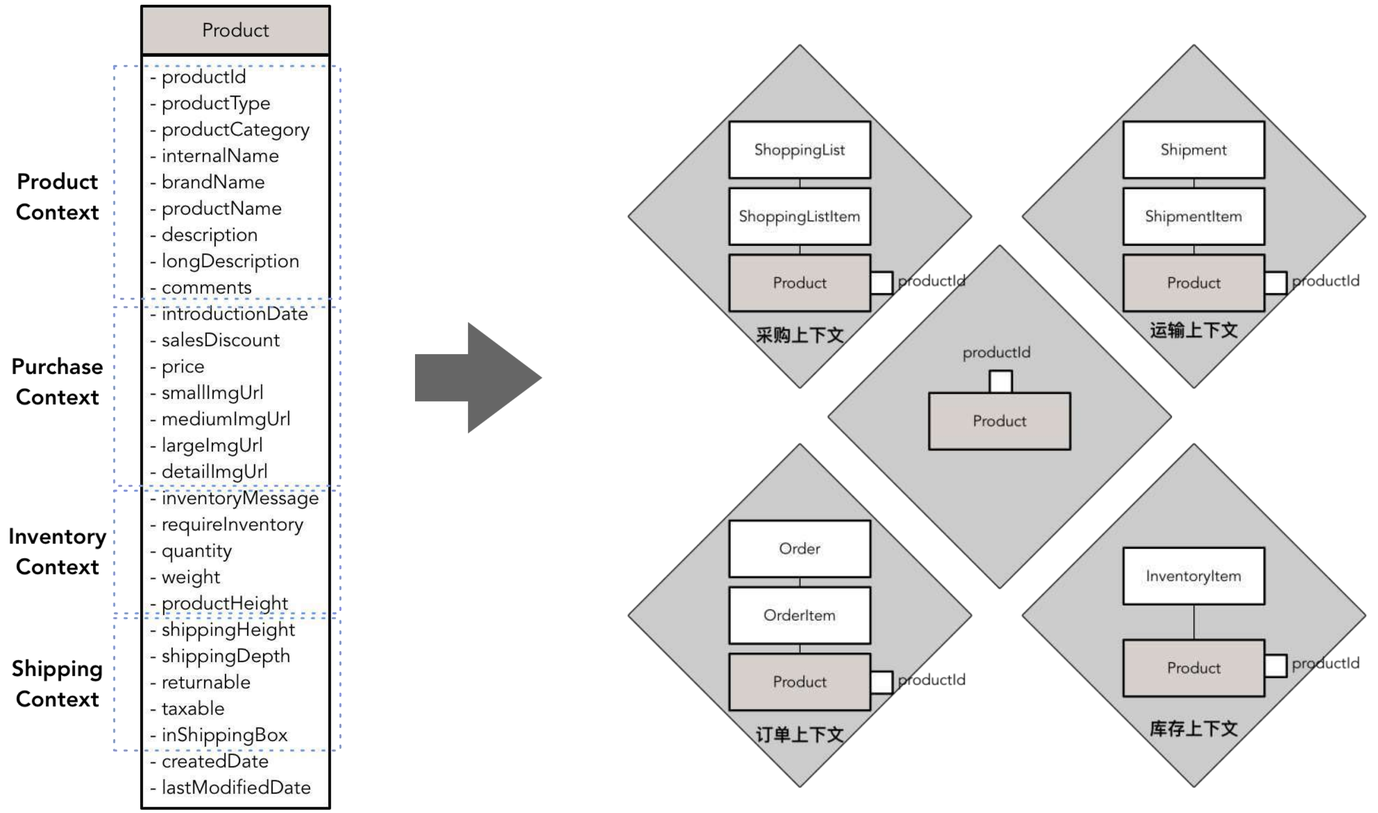
限界上下文是业务能力的纵向切分

领域驱动设计引入了限界上下文



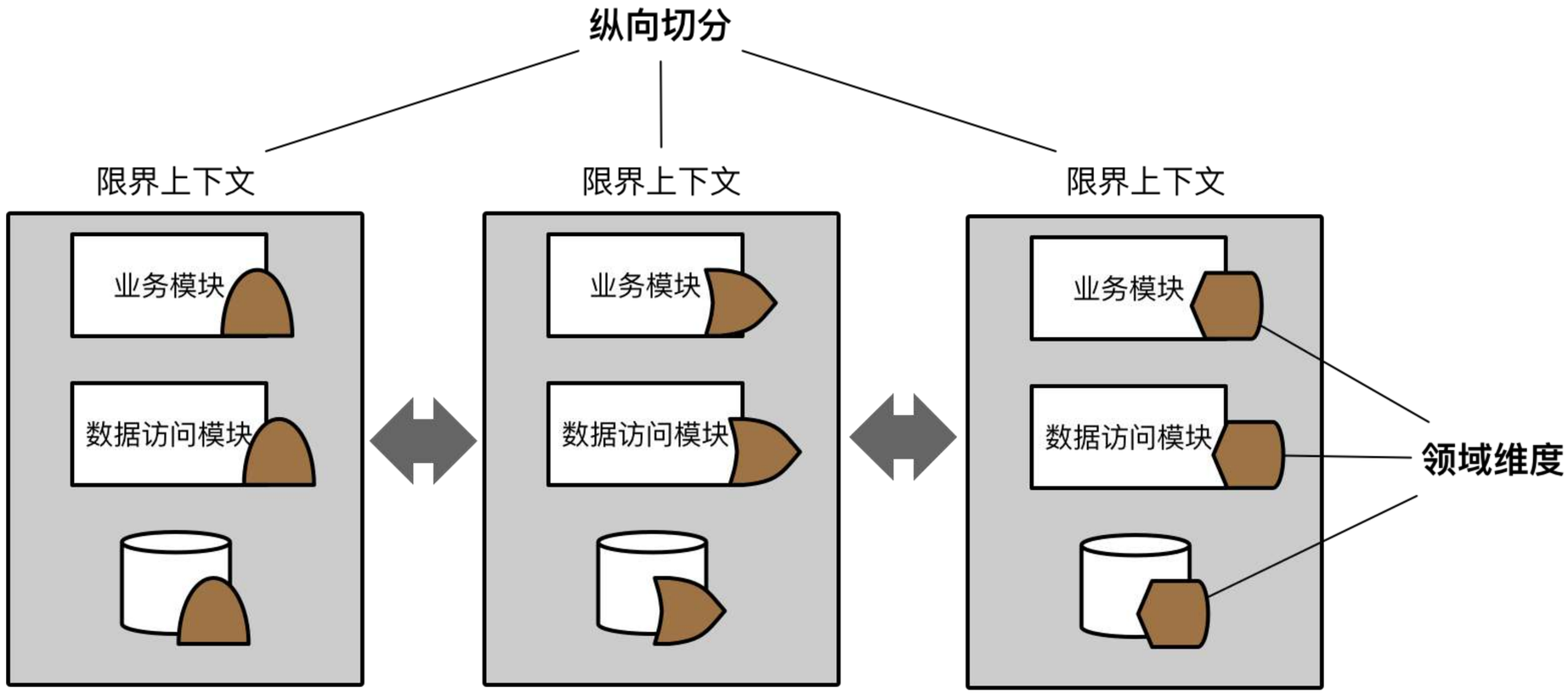
限界上下文之间通过业务能力完成重用

领域驱动设计引入了限界上下文



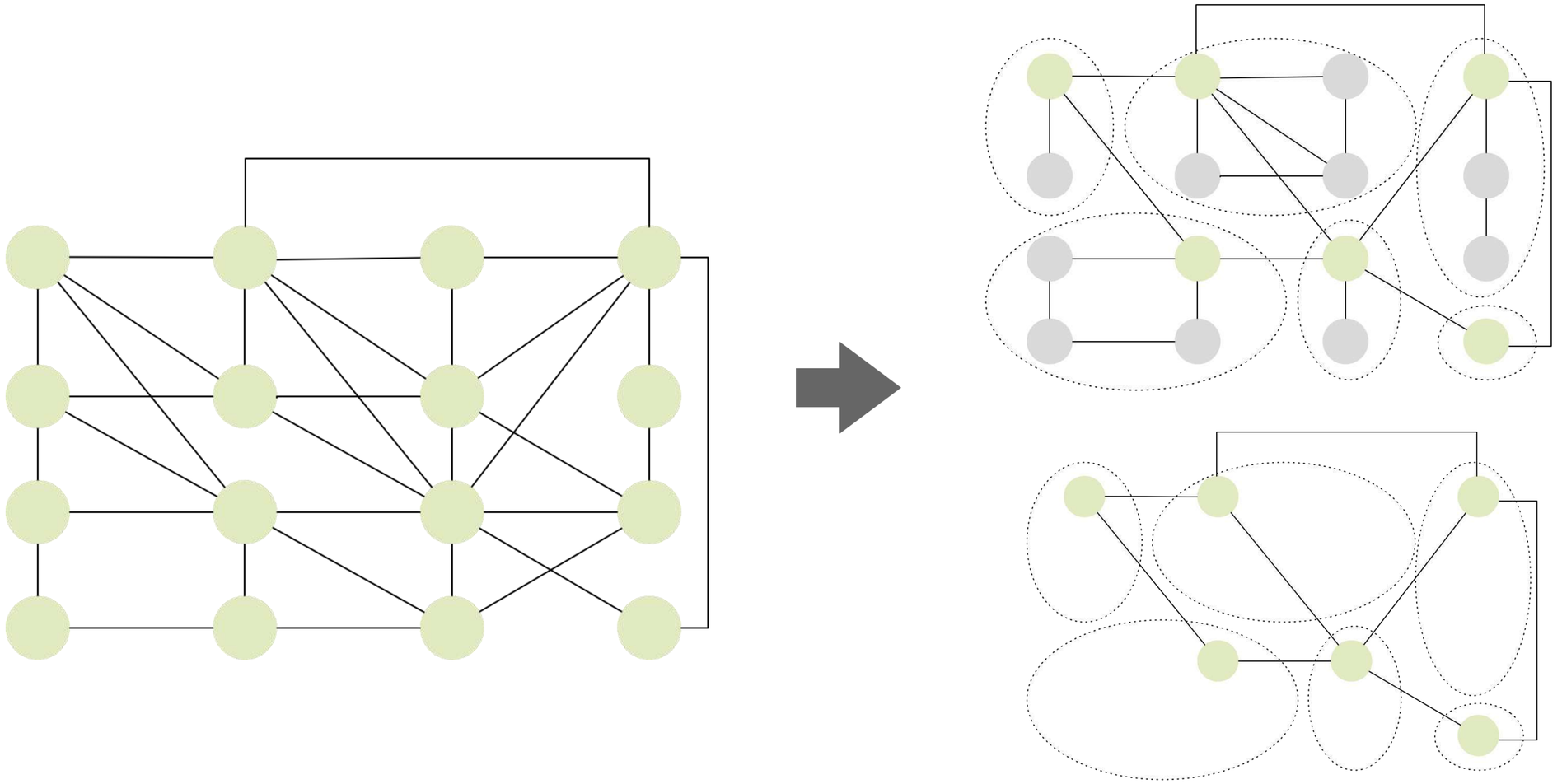
识别上下文，确定领域模型的知识语境

领域驱动设计引入了限界上下文



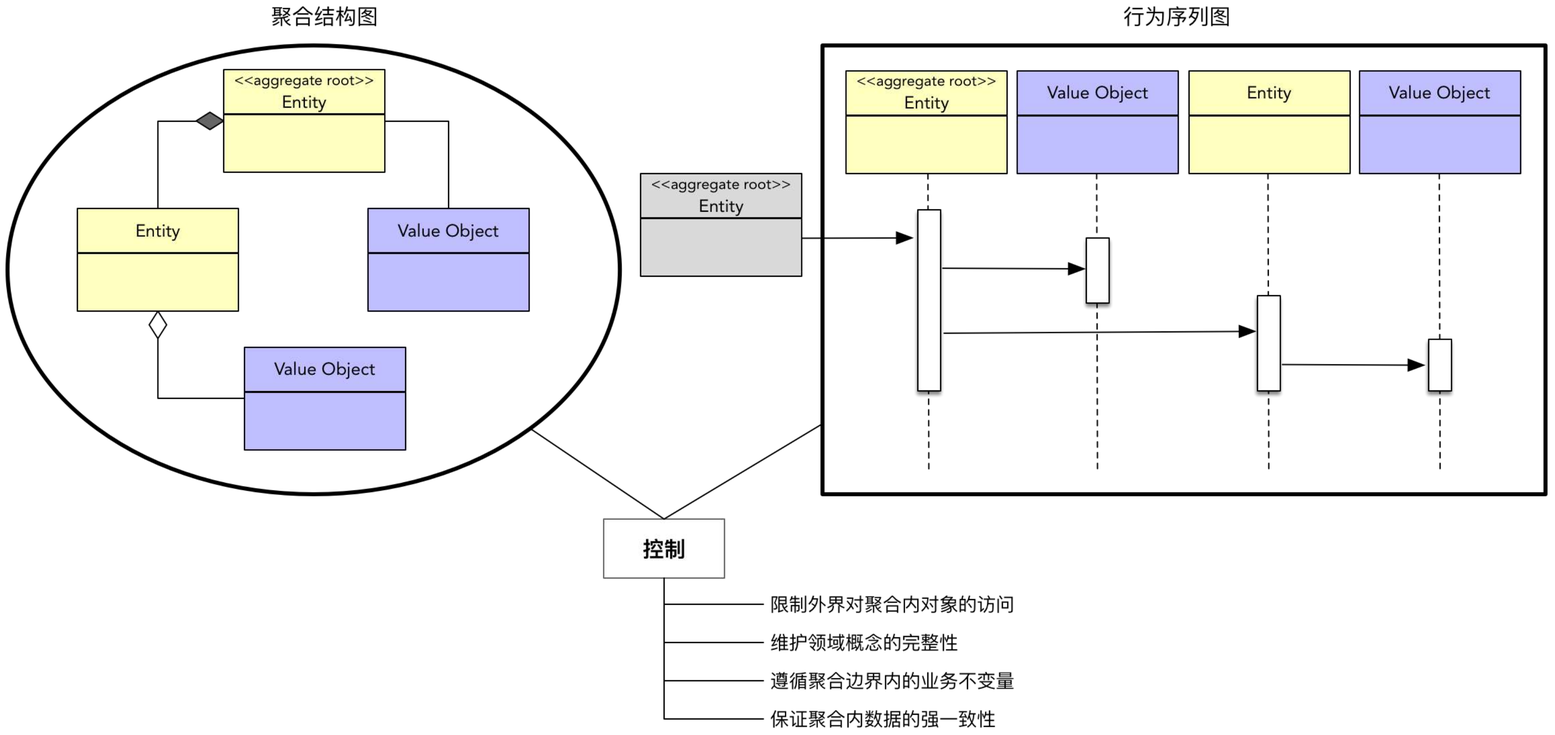
架构顺应业务的变化方向

领域驱动设计引入了聚合



聚合的边界简化了对象图

领域驱动设计引入了聚合



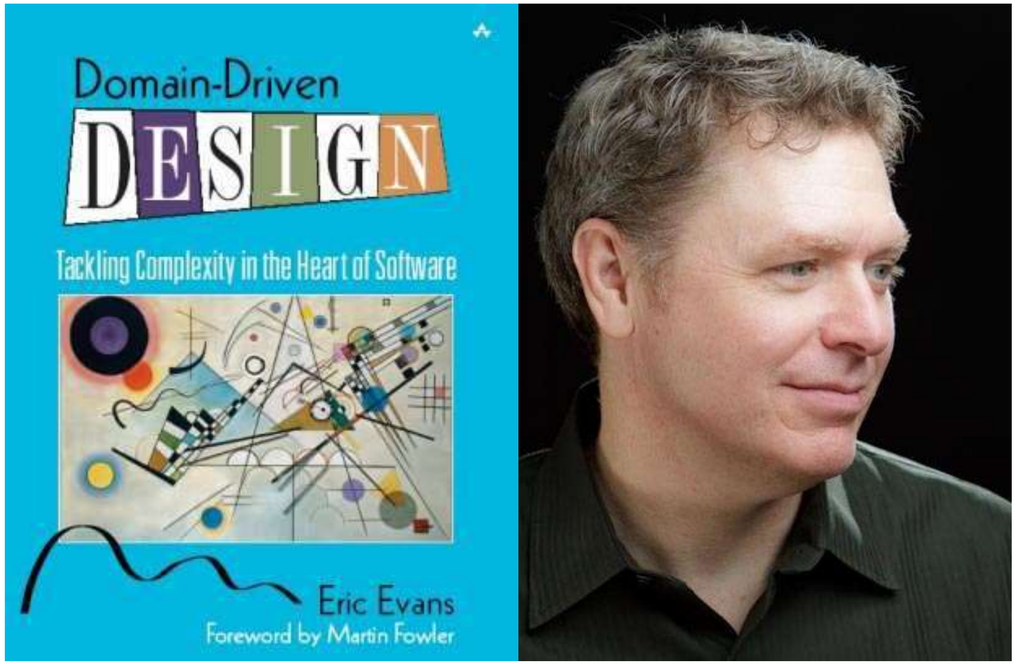
聚合限制了外界对聚合内对象的访问

DDD CHINA

2

DDD的黑铁时代与黄金时代

为什么会成为黑铁时代



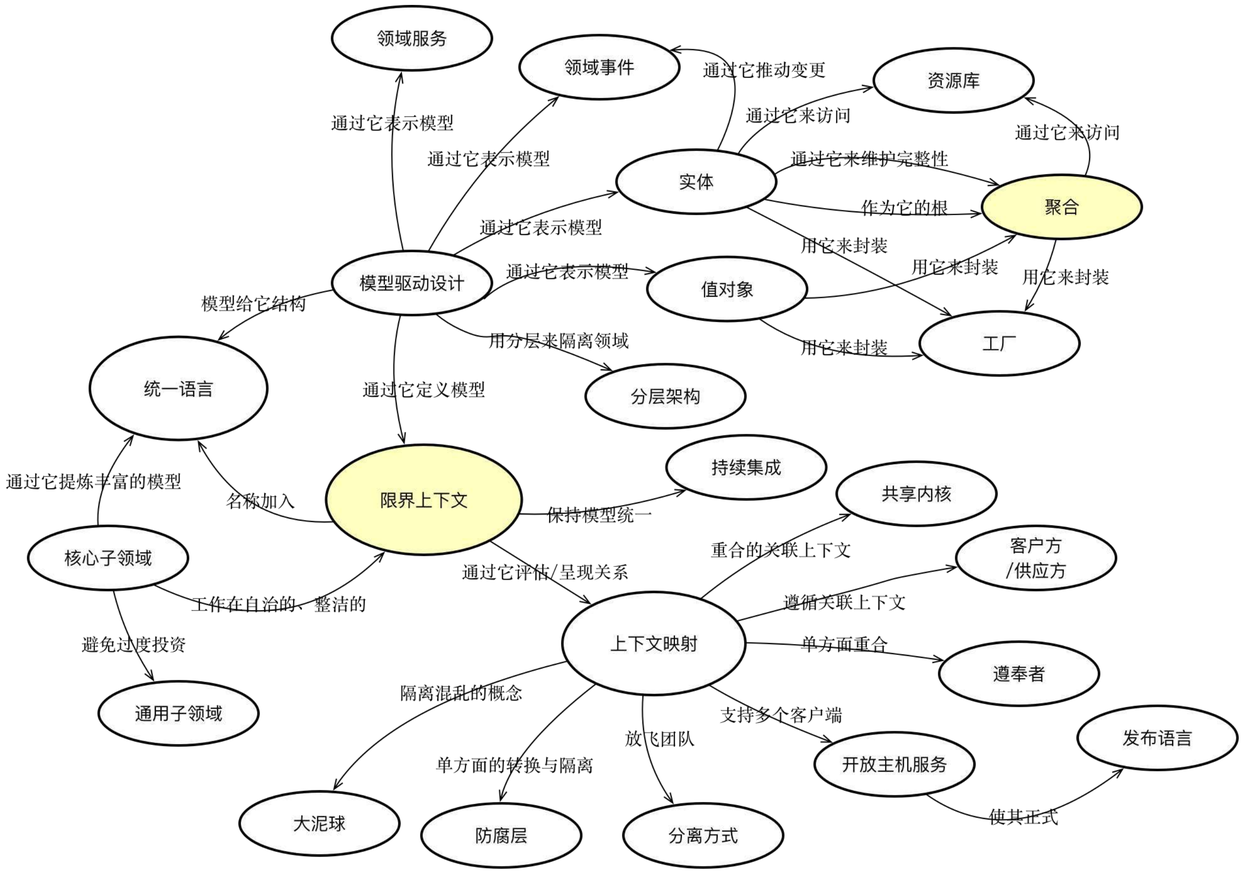
Domain-Driven Design

Tackling Complexity in Software

2003年

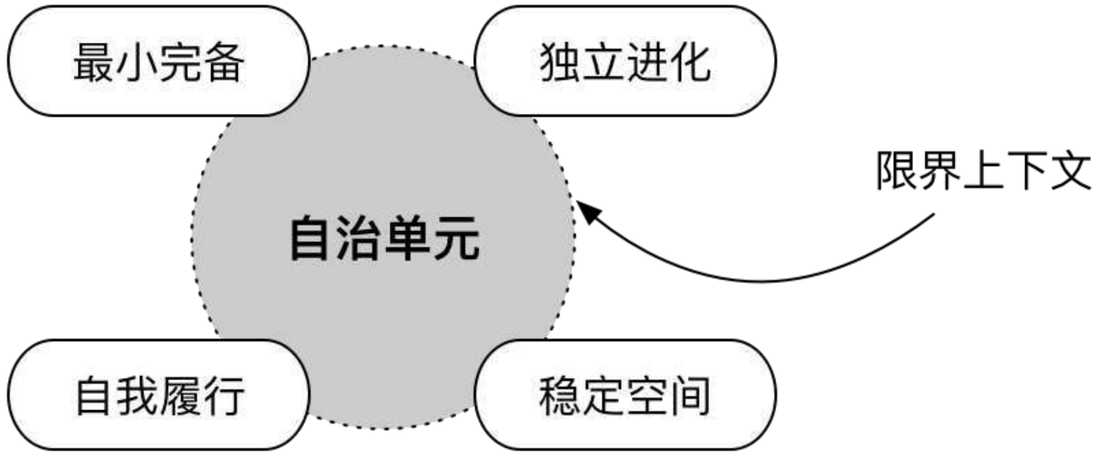
DDD从诞生开始就未成为流行的方法体系

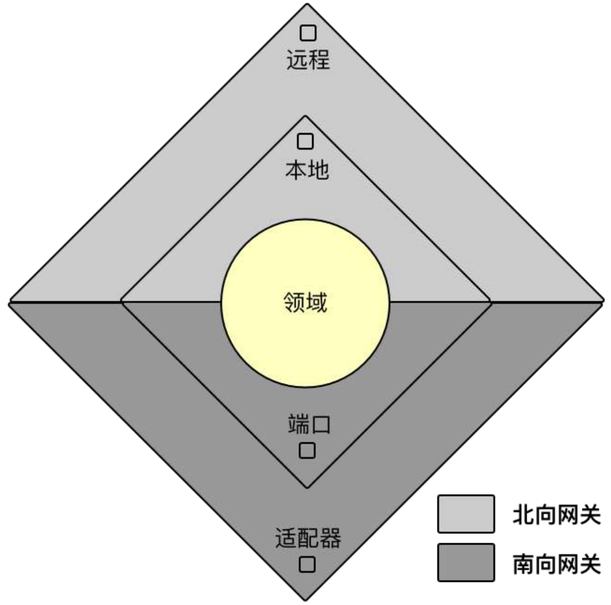
DDD的特点与价值在于它定义的模式



限界上下文与聚合是DDD的核心模式

限界上下文的价值

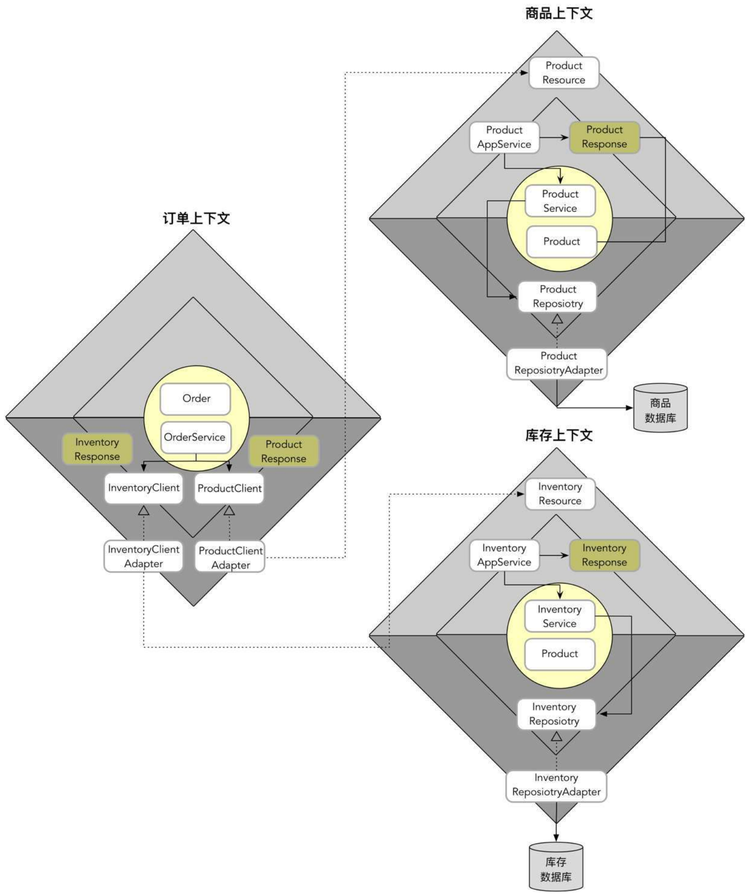




菱形对称架构
Rhomboid Symmetric Architecture

限界上下文是架构层次的自治单元

限界上下文的协作



限界上下文之间是业务能力的重用而非模型的重用

然而...

```java
@Service
@DomainService
public class OrderService {
    @Autowired
    private OrderRepository orderRepository;
    @Autowired
    private InventoryClient inventoryClient;

    public void placeOrder(Order order) {
        if (!order.isInvalid()) {
            throw new InvalidOrderException();
        }

        InventoryReview inventoryReview = inventoryClient.check(order);
        if (!inventoryReview.isAvailable()) {
            throw new NotEnoughInventoryException();
        }

        orderRepository.add(order);
        inventoryClient.lock(order);
    }
}
```

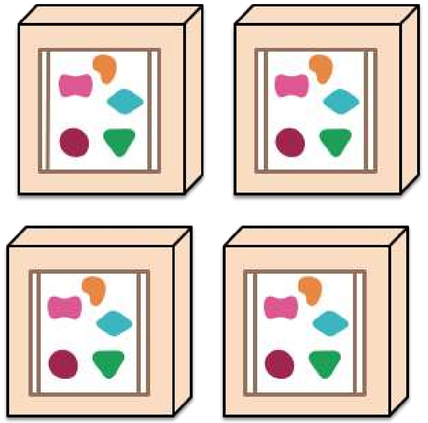
```java
@Component
@Adapter(PortType.Client)
public class InventoryClientAdapter implements InventoryClient {
    @Autowired
    private InventoryAppService inventoryAppService;

    @Override
    public InventoryReview check(Order order) {
        InventoryReviewResponse inventoryReviewResponse = inventoryAppService.checkInventory(convertFromOrder(
        return convertToInventoryReview(inventoryReviewResponse);
    }

    private CheckingInventoryRequest convertFromOrder(Order order) { return null; }

    private InventoryReview convertToInventoryReview(InventoryReviewResponse inventoryReviewResponse) { retur
```

单体架构需要严格遵循限界上下文的边界

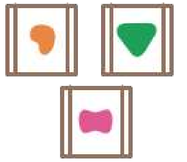
直到...

A monolithic application puts all its functionality into a single process...

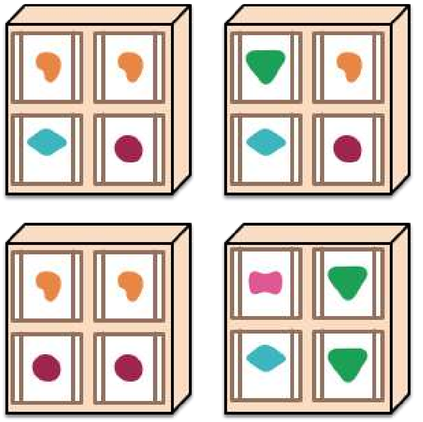
... and scales by replicating the monolith on multiple servers

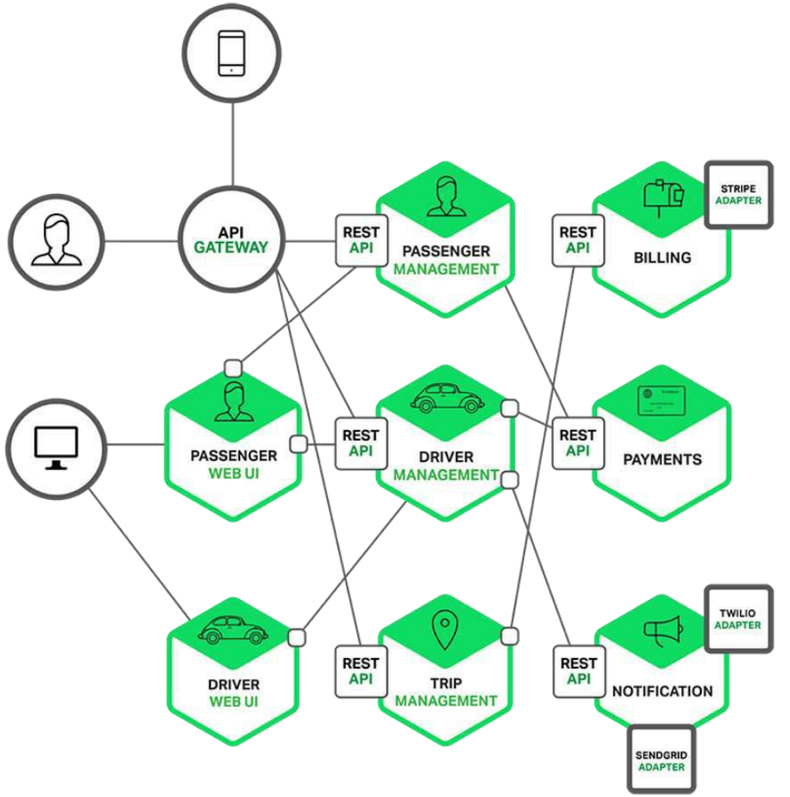


A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.





微服务的提出凸显了限界上下文的重要性

微服务的协作就是限界上下文的协作

```java
@Service
@DomainService
public class OrderService {
    @Autowired
    private OrderRepository orderRepository;
    @Autowired
    private InventoryClient inventoryClient;

    public void placeOrder(Order order) {
        if (!order.isInvalid()) {
            throw new InvalidOrderException();
        }

        InventoryReview inventoryReview = inventoryClient.check(order);
        if (!inventoryReview.isAvailable()) {
            throw new NotEnoughInventoryException();
        }

        orderRepository.add(order);
        inventoryClient.lock(order);
    }
}
```

```java
@Component
@Adapter(PortType.Client)
public class InventoryClientAdapter implements InventoryClient {
    private static final String INVENTORIES_RESOURCE_URL = "http://inventory-service/inventories";

    @Autowired
    private RestTemplate restTemplate;

    @Override
    public InventoryReview check(Order order) {
        CheckingInventoryRequest inventoryRequest = CheckingInventoryRequest.from(order);
        InventoryReviewResponse reviewResponse = restTemplate.postForObject(INVENTORIES_RESOURCE_URL,
        return reviewResponse.to();
    }

    @Override
    public void lock(Order order) {
        LockingInventoryRequest inventoryRequest = LockingInventoryRequest.from(order);
        restTemplate.put(INVENTORIES_RESOURCE_URL, inventoryRequest);
    }
}
```

通过Client防腐层调用上游微服务成为合理的选择

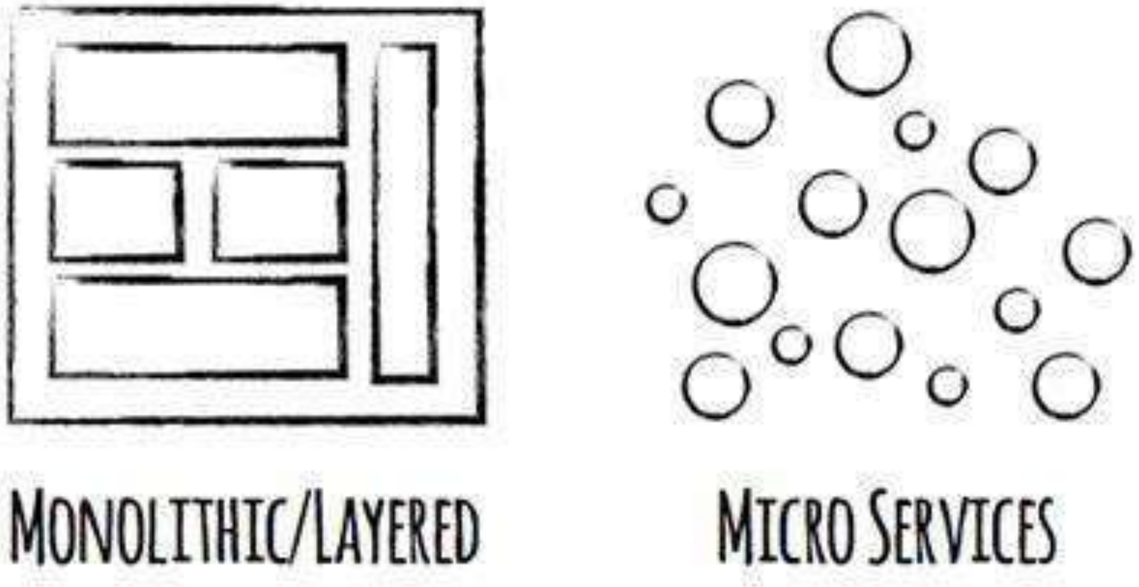
领域驱动设计成为显学，进入黄金时代

DDD CHINA

3

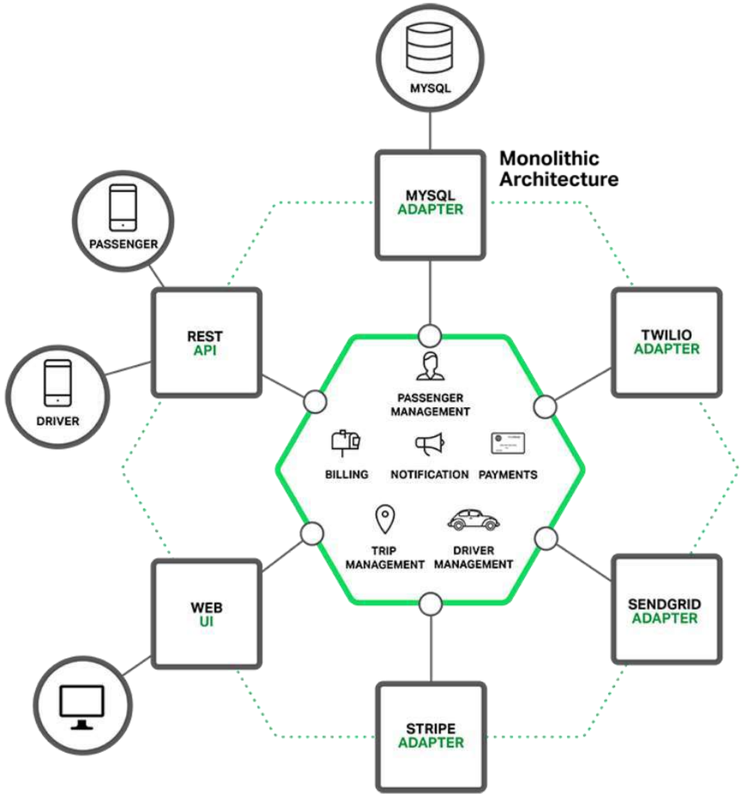
单体架构是邪恶的吗

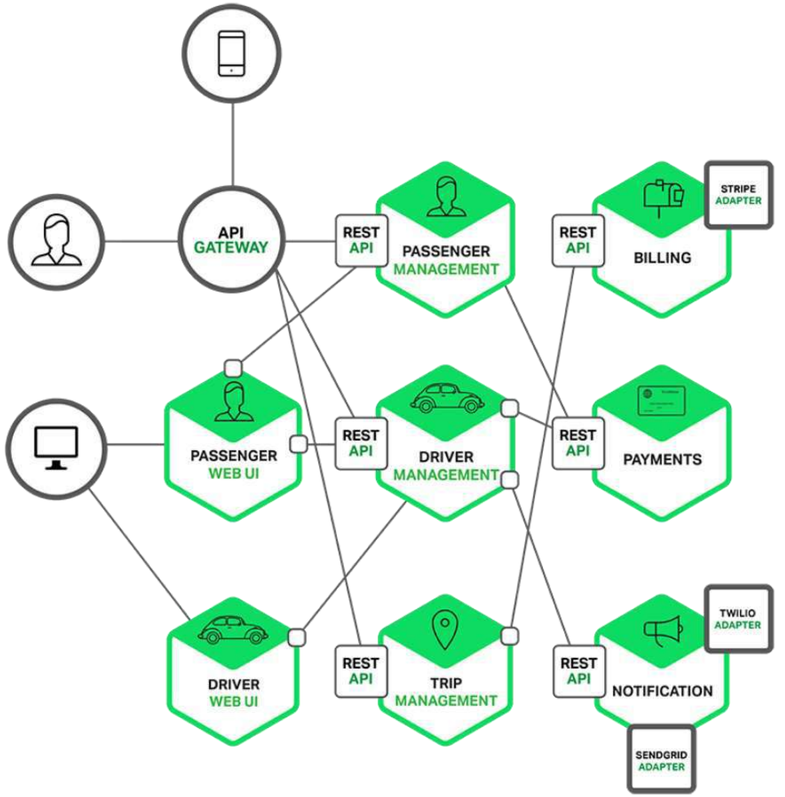
如此区分单体架构和微服务架构并不正确



没有限界上下文的单体架构可能导致大泥球

这样区分单体架构和微服务架构也不正确

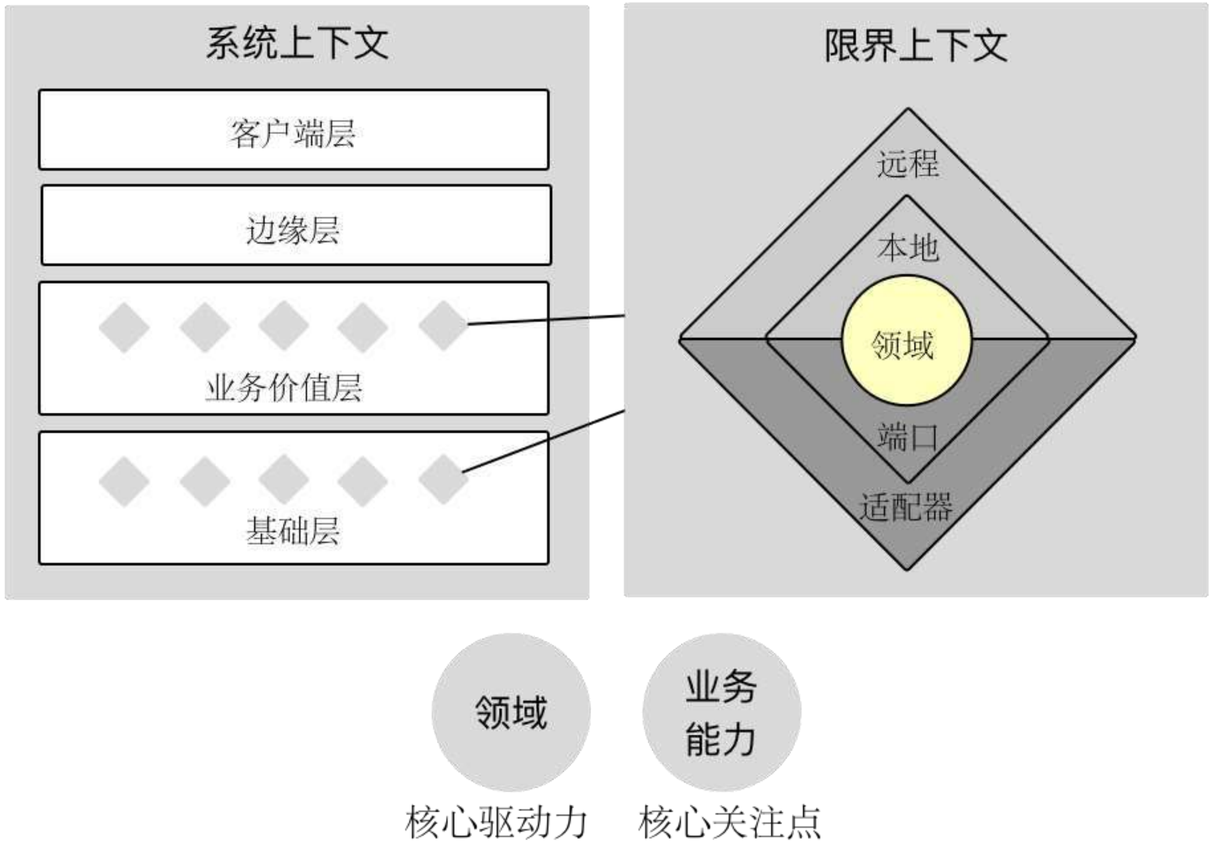




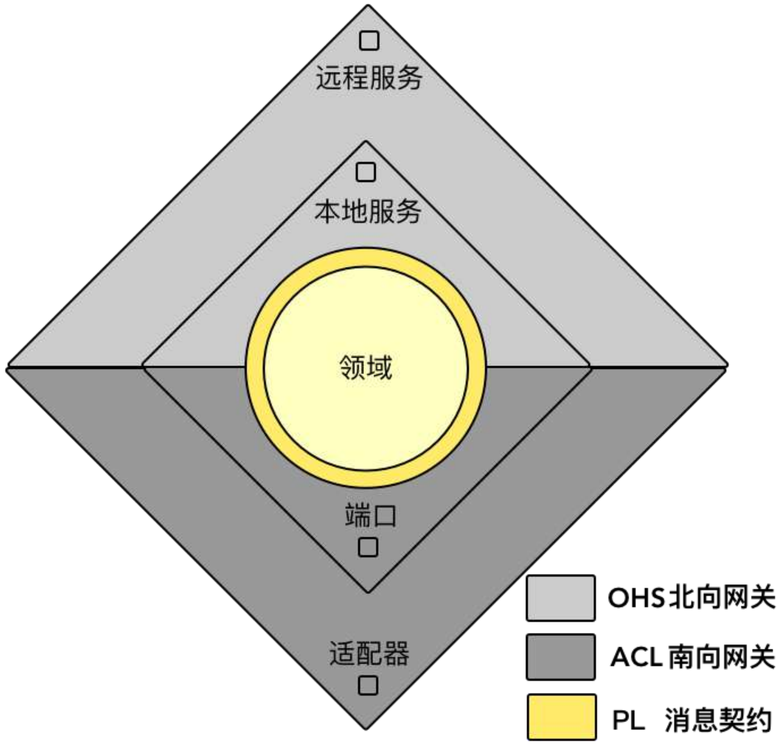
单体架构也要通过业务能力进行纵向切分

领域驱动架构风格

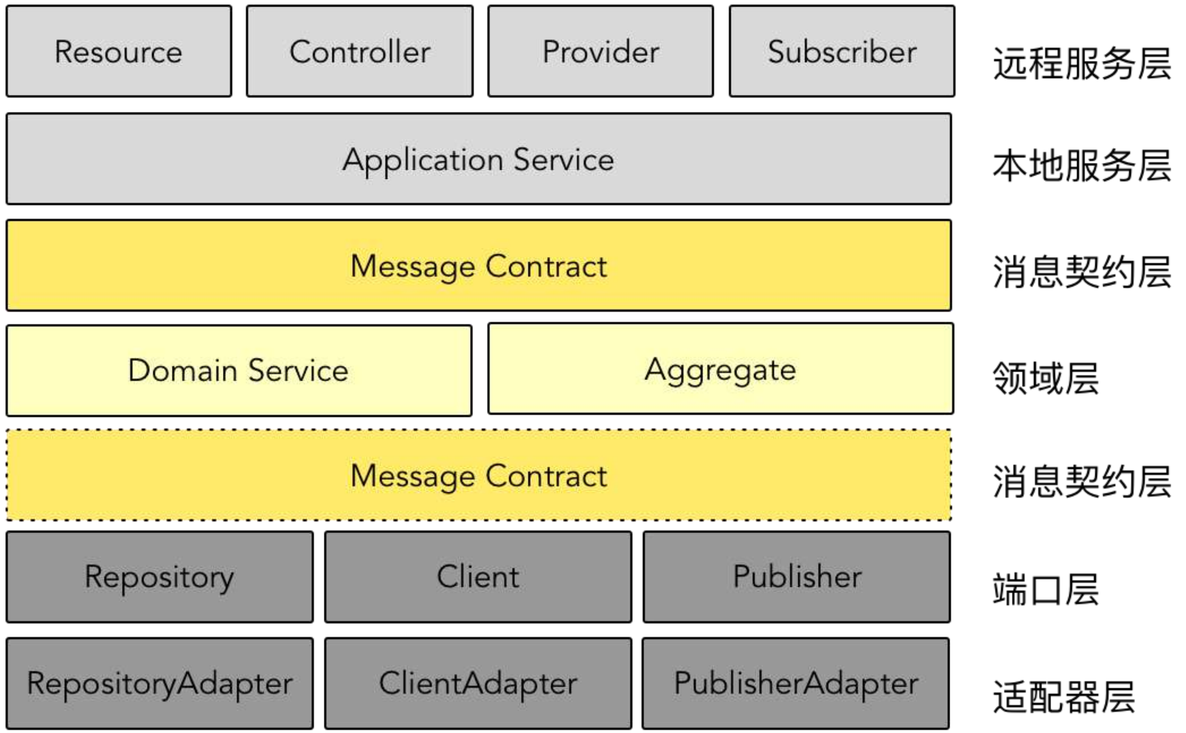
- 领域：核心驱动力
- 业务能力：核心关注点
- 系统上下文层次：系统分层架构模式
- 限界上下文层次：菱形对称架构模式



领域驱动架构风格-菱形对称架构

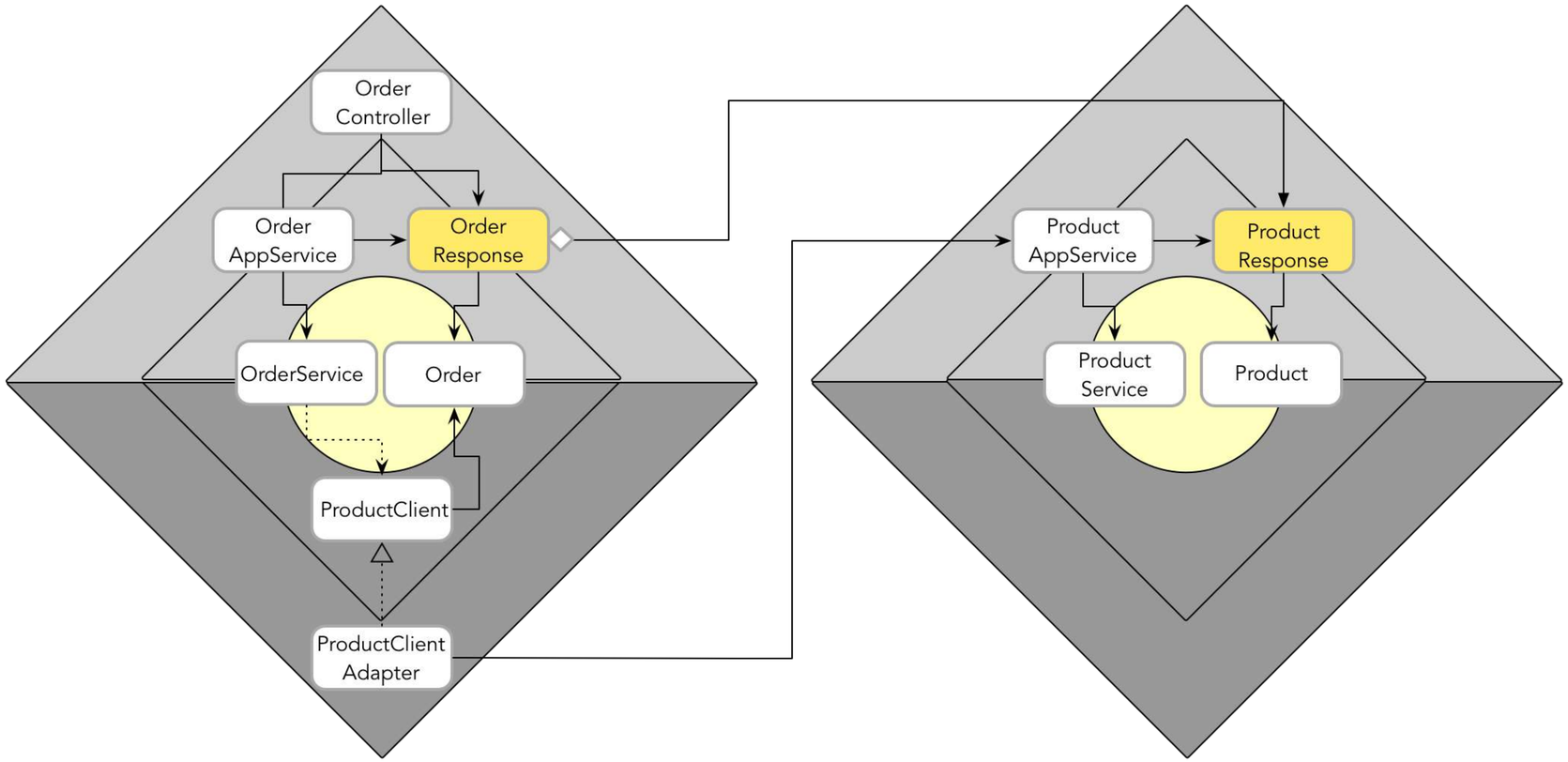


菱形对称架构



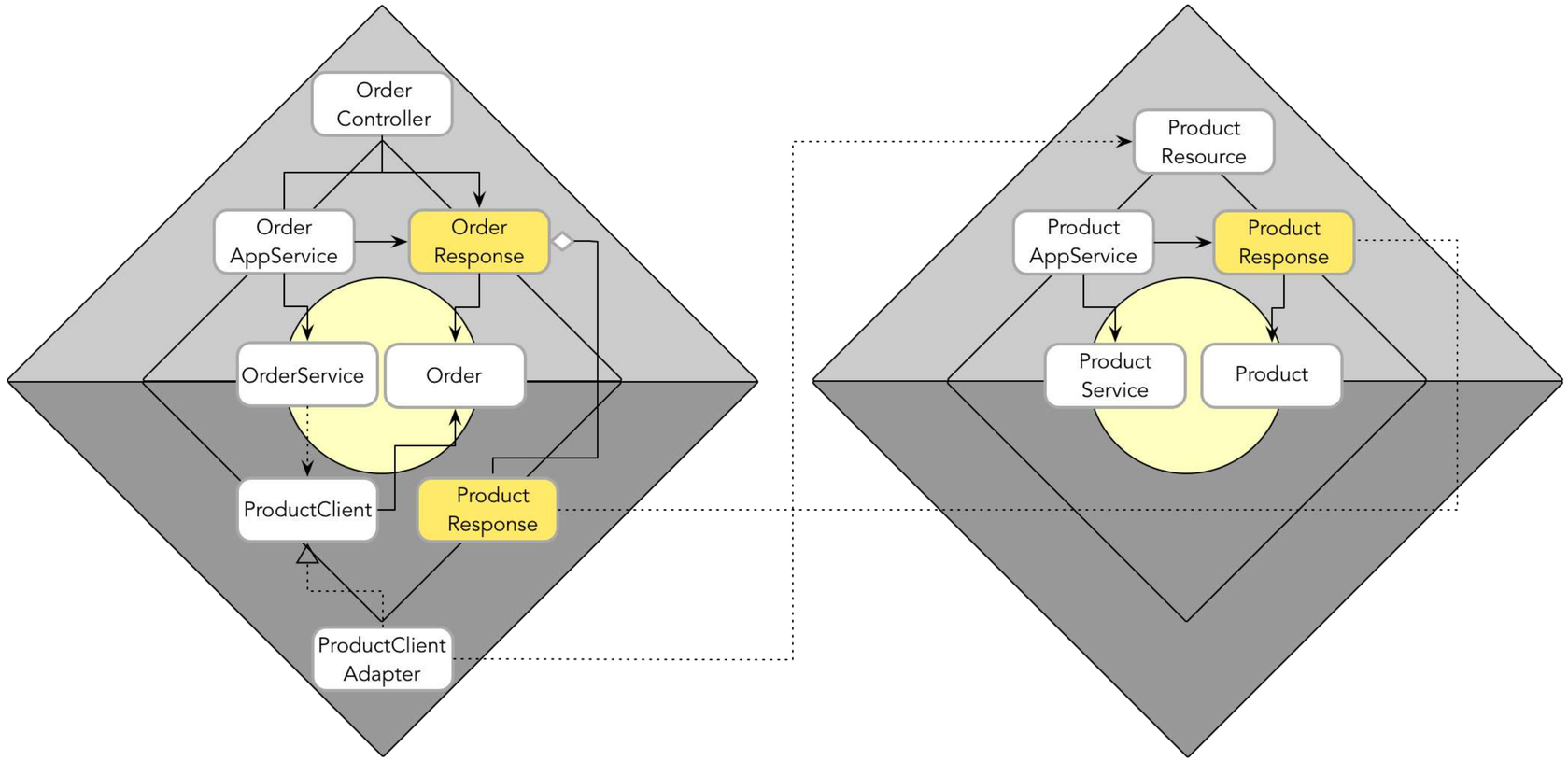
分层架构

领域驱动架构风格-单体架构



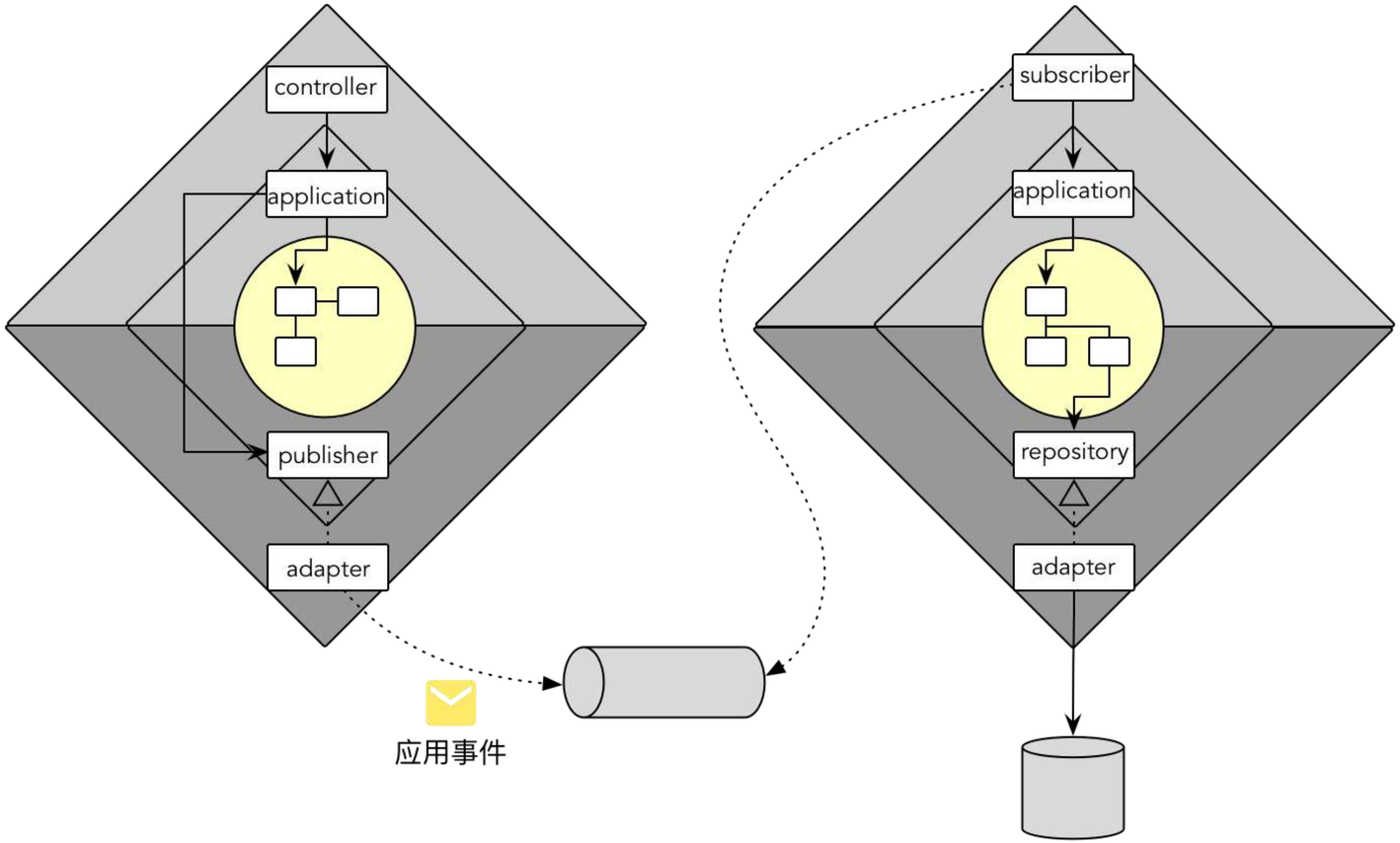
下游限界上下文南向网关的适配器与上游限界上下文的应用服务之间的协作

领域驱动架构风格-微服务架构



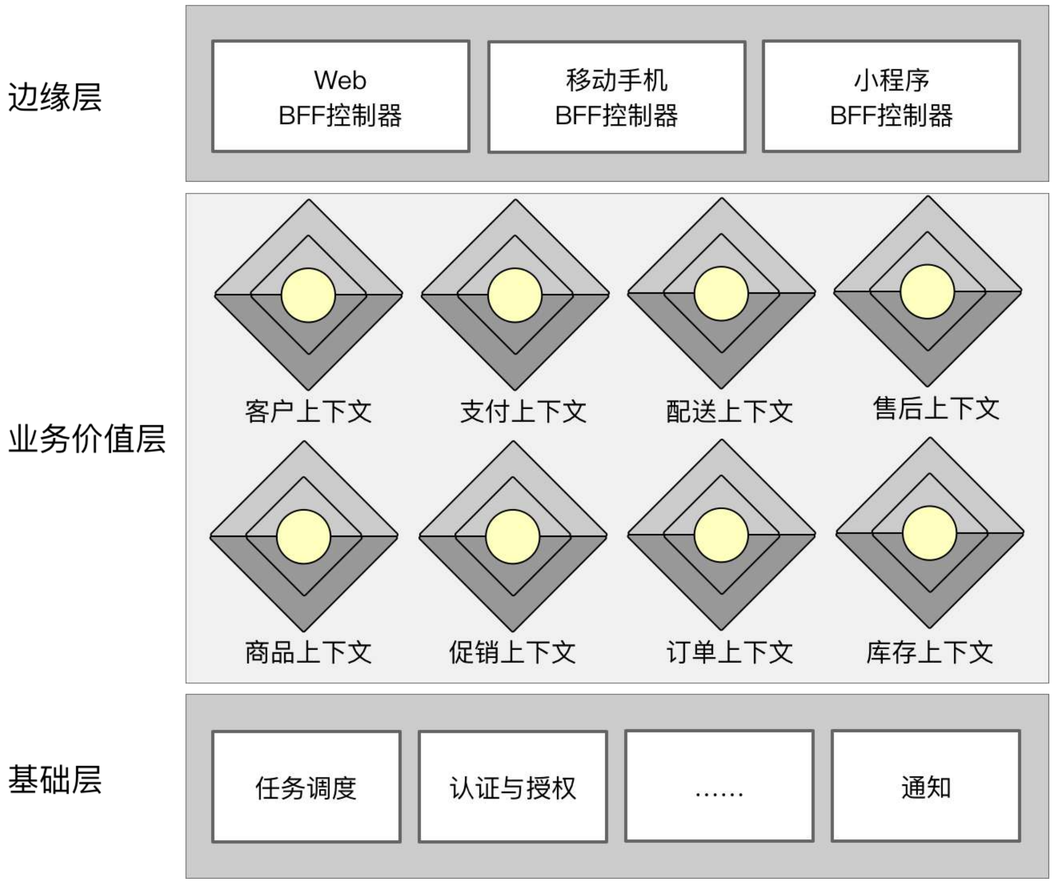
下游限界上下文南向网关的适配器与上游限界上下文的远程服务之间的协作

领域驱动架构风格-事件驱动架构



下游限界上下文南向网关的适配器与上游限界上下文的远程服务之间的协作

领域驱动架构风格-系统分层架构



无论单体架构还是微服务架构，它们的逻辑架构是一致的

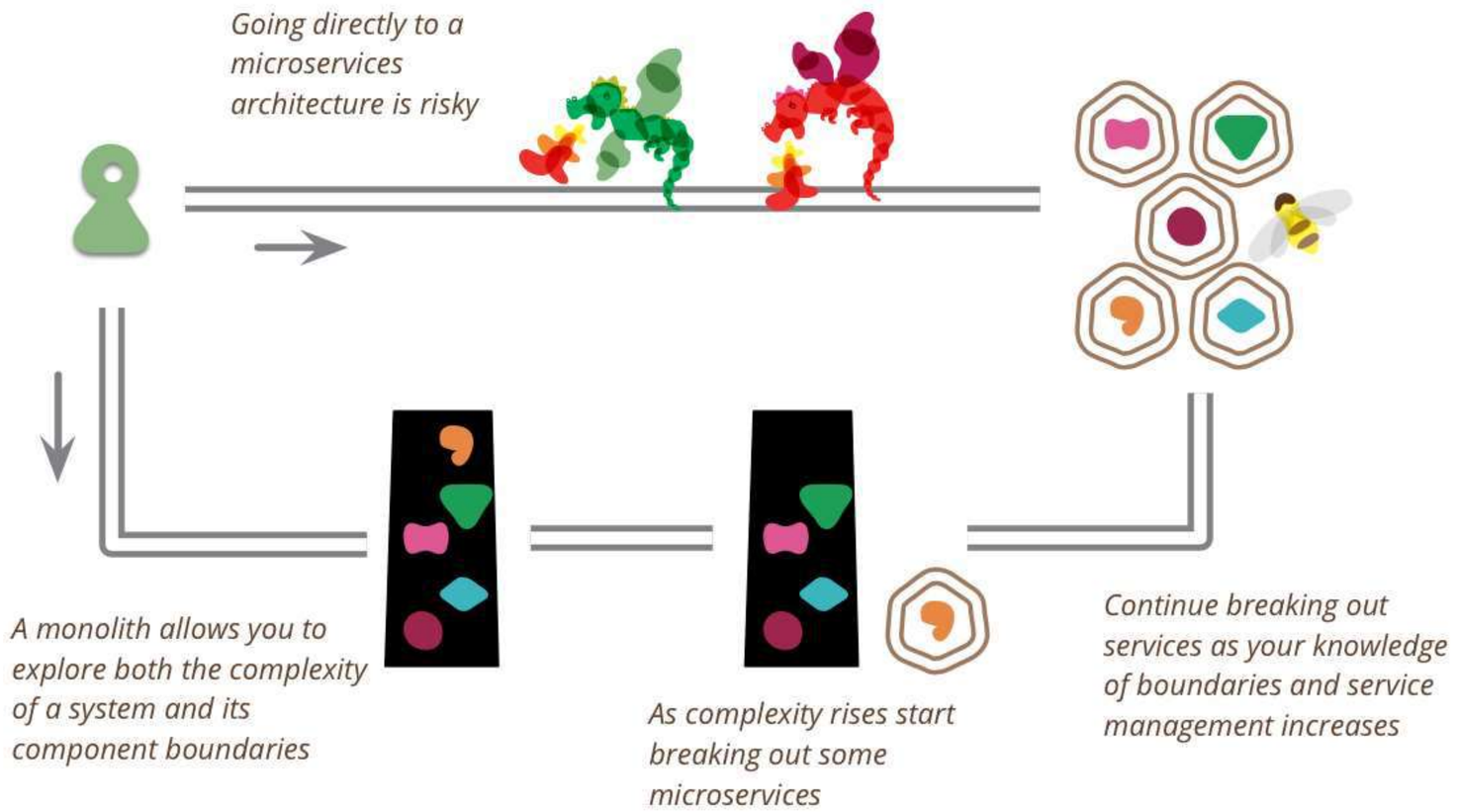
是时候给单体架构正名了

DDD CHINA

It's time

单体架构优先

如果单体架构通过限界上下文进行边界控制，可以降低微服务架构风格的演化成本，也能规避过度微服务化带来的技术风险



4

DDD的不足与DDD-UP

领域驱动设计的不足

不足之一

领域驱动设计缺乏一个规范的过程指导，使得其缺乏可操作性，团队在运用领域驱动设计时，更多取决于设计者的行业知识与设计经验，使得领域驱动设计在项目上的成功存在较大的偶然性。

不足之二

领域驱动设计没有匹配的需求管理体系。不同层次的业务需求贯穿于领域驱动设计过程中的每个环节，识别限界上下文和建立高质量的领域模型都有赖于良好的需求，需求管理体系会直接影响领域驱动设计的质量。

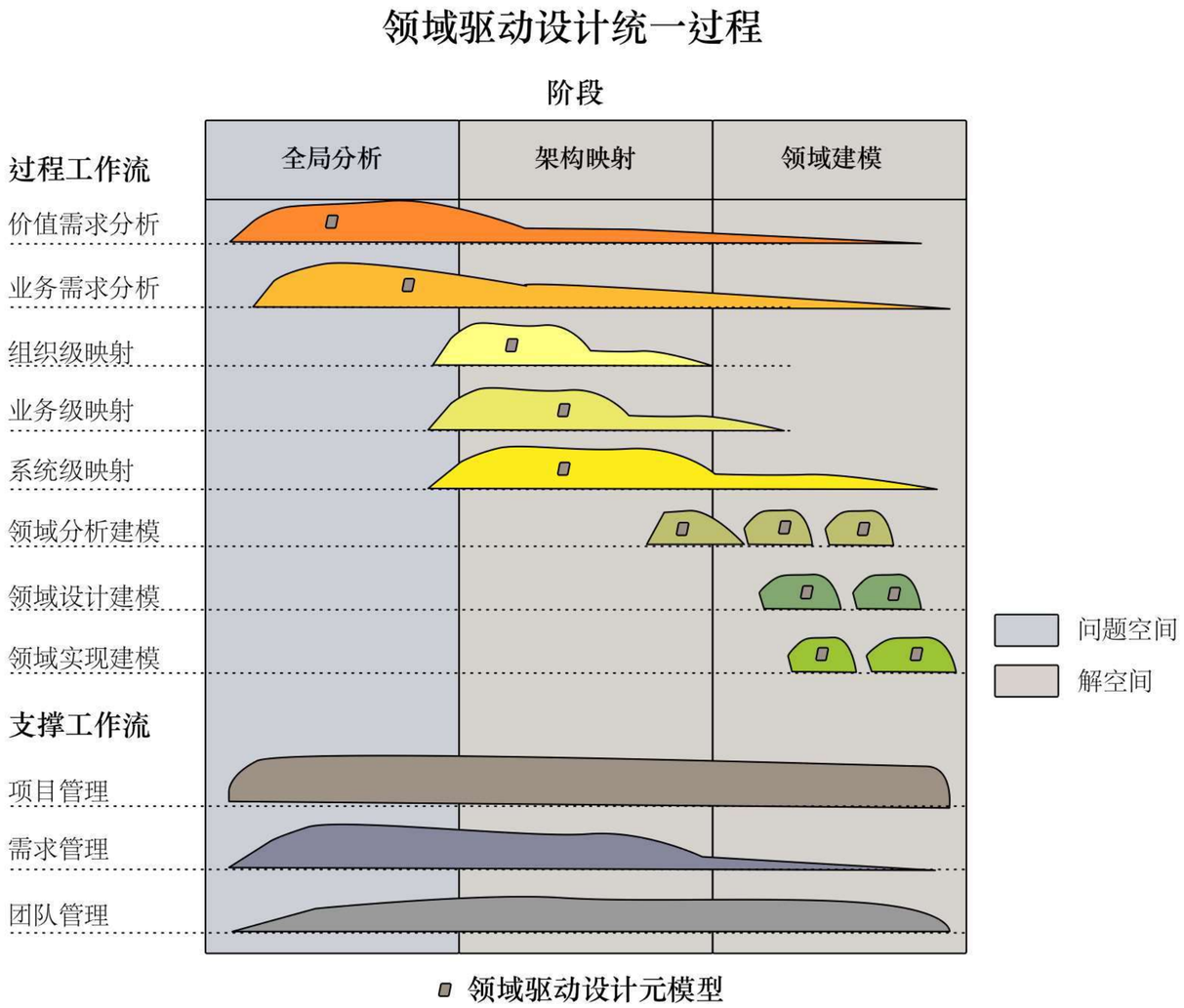
不足之三

领域驱动设计的核心诉求是让业务架构和应用架构形成绑定关系，以面对需求变化时，使得应用架构能够适应业务架构的调整，满足架构的演进性，然而它却缺乏面向领域的架构体系，不足以支撑复杂软件项目的架构需求。

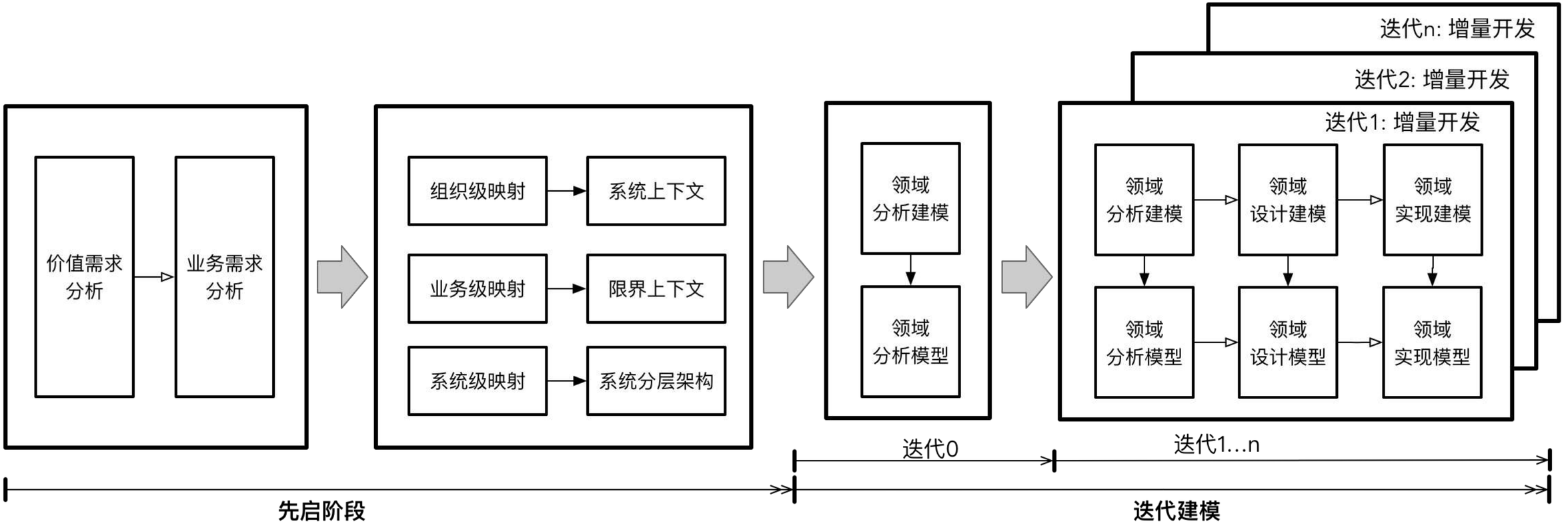
不足之四

领域驱动设计虽然以模型驱动设计为主线，却没有给出明确的领域建模方法。我们需要分别为领域分析建模、领域设计建模和领域实现建模提供对应的方法指导，减轻每个建模活动的知识负担，明确每个建模活动获得的领域模型的验证标准，避免领域建模的随意性。

领域驱动设计统一过程DDD-UP



领域驱动设计统一过程的执行



THANKS

「逸言」
公众号

「TOP DDD」
知识星球

DDD CHINA