DDD的为与不为

滕云@ThoughtWorks

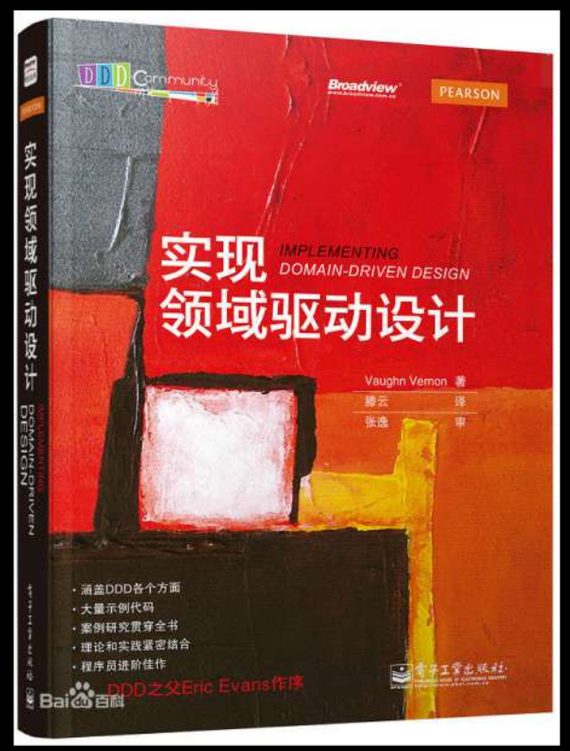
DDD CHINA

关于我

01 ThoughtWorks架构师&编码者

02 Java/TechOps/DDD

03 《实现领域驱动设计》/《人件》译者



我眼中的DDD

道

原则

弱约束

最佳实践

编码强相关

面向对象进阶

数据驱动之反面

优先处理业务逻辑

不是架构师专属标榜

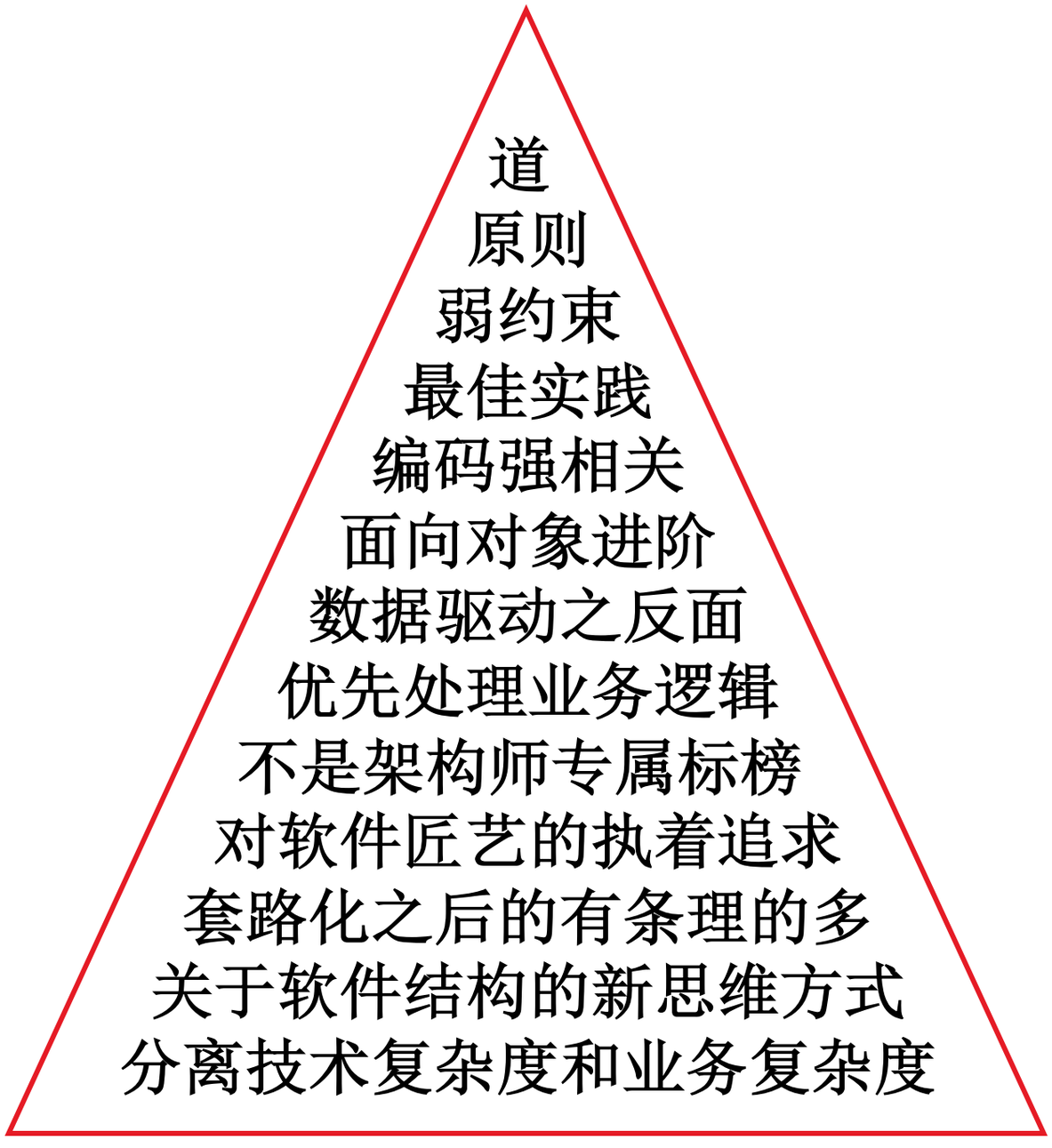
对软件匠艺的执着追求

套路化之后的有条理的多

关于软件结构的新思维方式

分离技术复杂度和业务复杂度

我眼中的DDD


道
原则
弱约束
最佳实践
编码强相关
面向对象进阶
数据驱动之反面
优先处理业务逻辑
不是架构师专属标榜
对软件匠艺的执着追求
套路化之后的有条理的多
关于软件结构的新思维方式
分离技术复杂度和业务复杂度

DDD为何？

高内聚

分层

KIS
S

抽象

纯函数

依赖倒置

DRY

YAGNI

模块化

迪米特

里氏替换

单一职责

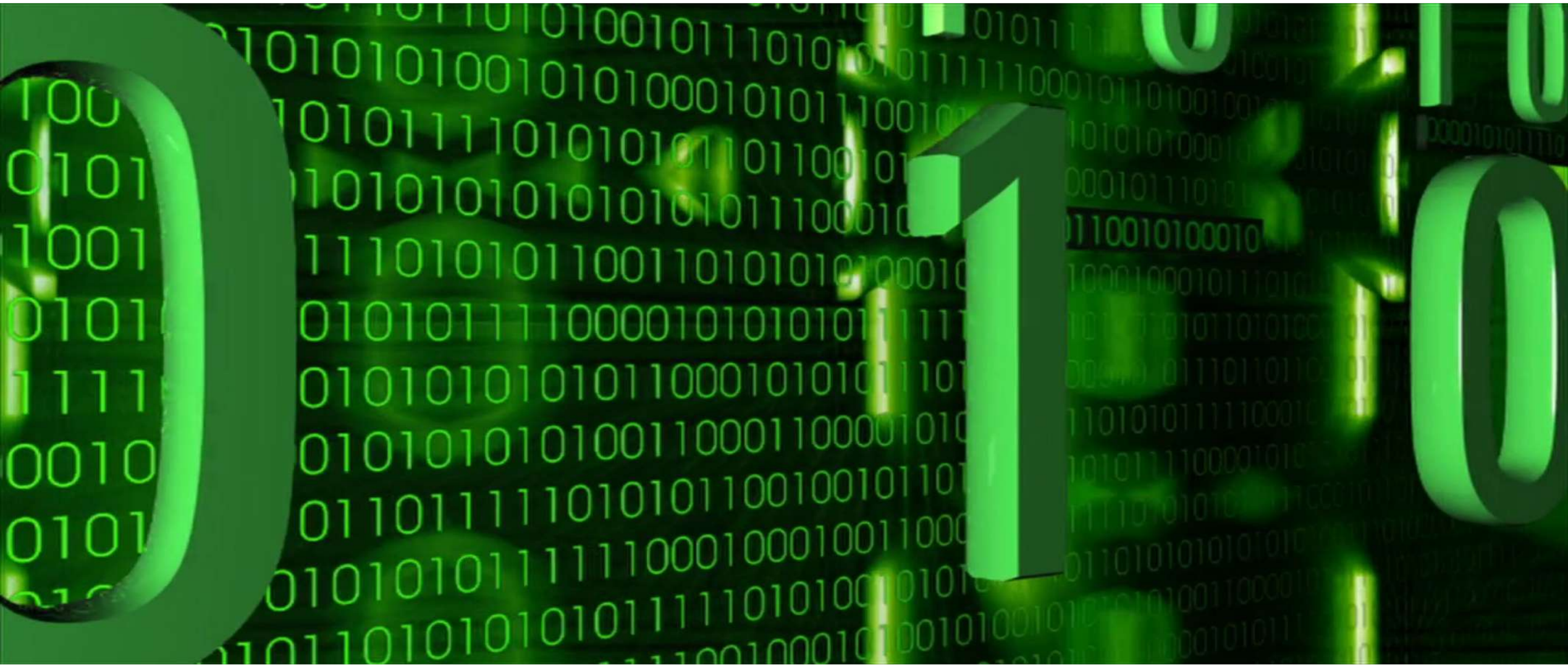
关注点分离

面向接口

不变性

开闭原则

DDD为何？



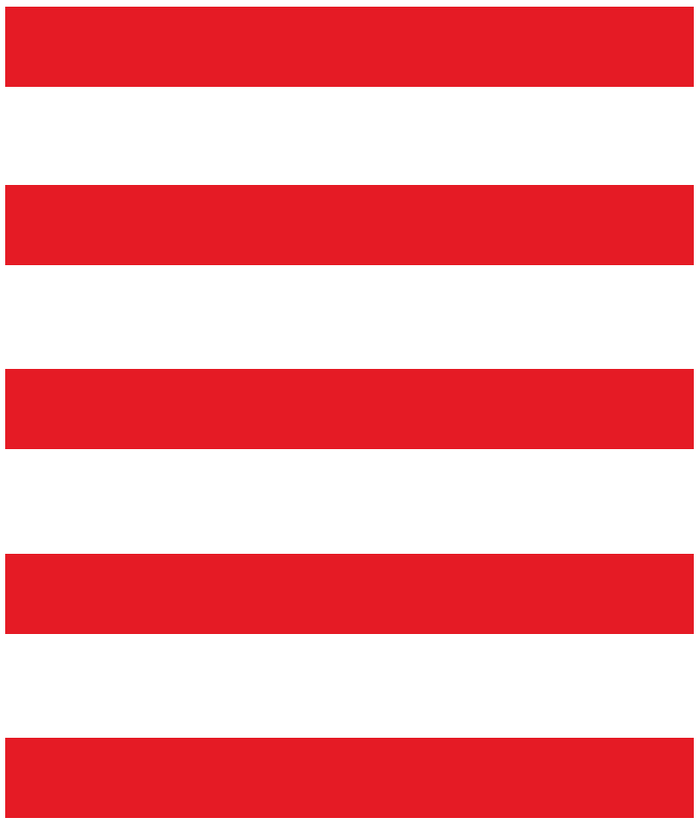
DDD为何？

VS

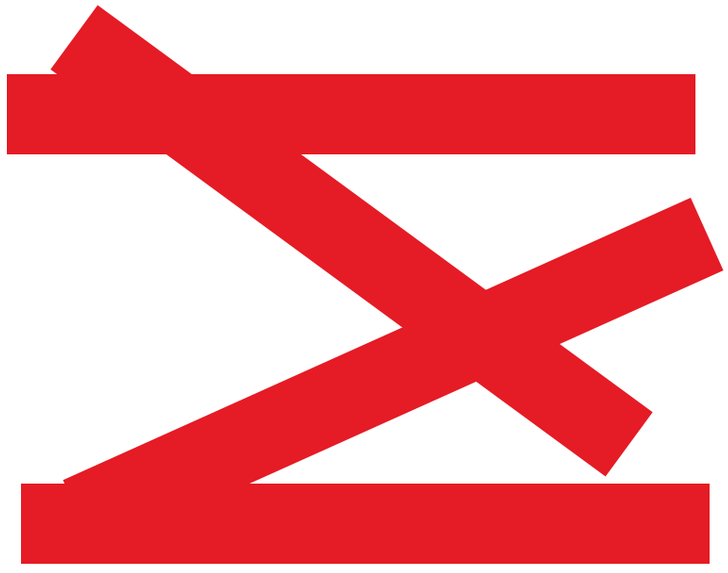
DDD为何？

为了简单

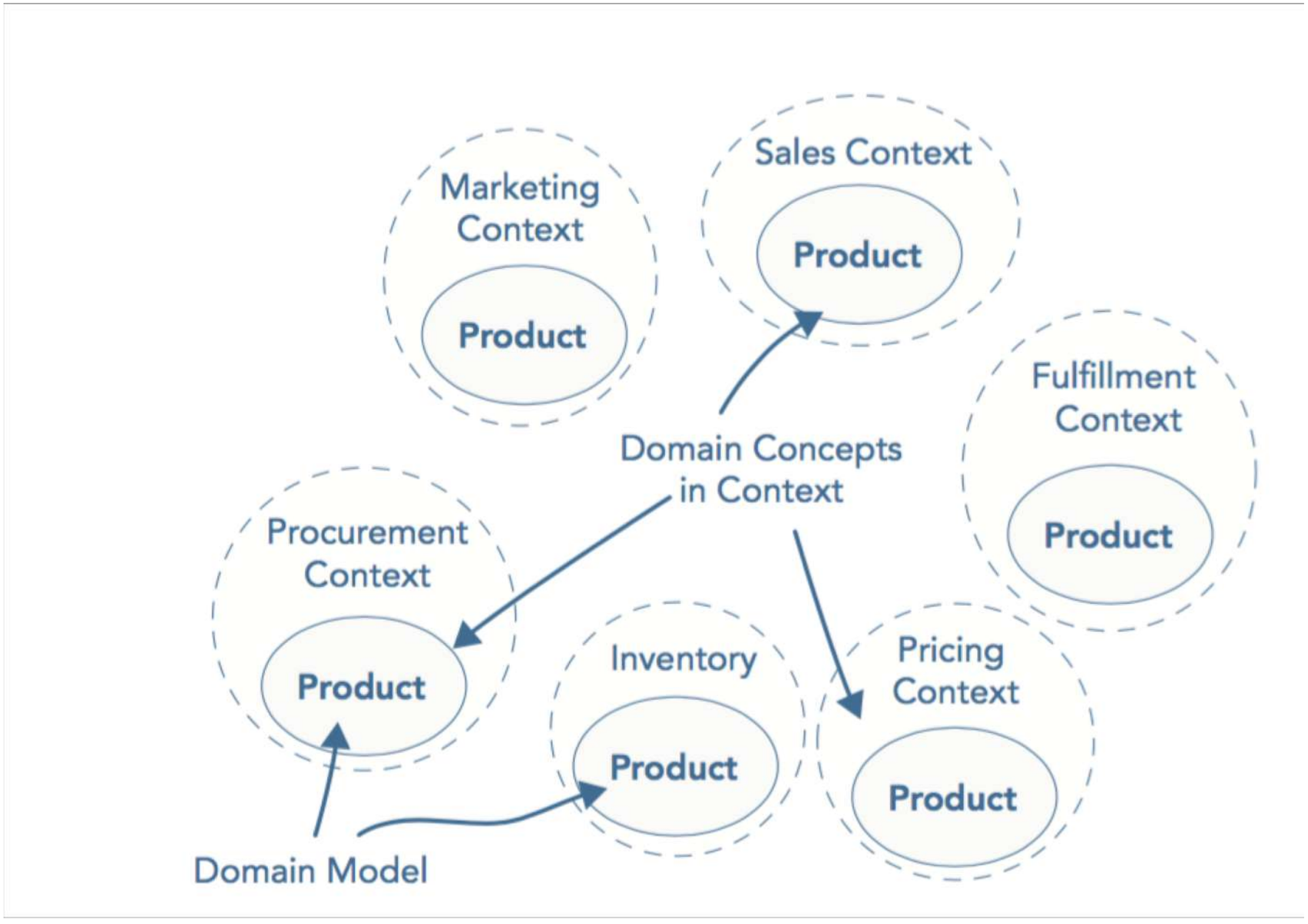
套路化之后的有条理的多



V
S



如何做战略设计？



一个领域概念在一个限界上下文中不应有二义性

事件风暴？


PACT SEALED
NAME CHOSEN
CONTRACTS FOUND
WEBSITE DESIGNED
DOMAIN REGISTERED
Designer
Back office

电商系统

退换货	推荐	支付	客服
订单	用户	促销	会员
物流	商品	库存	商户

如何做战略设计？

高大上 vs 朴素

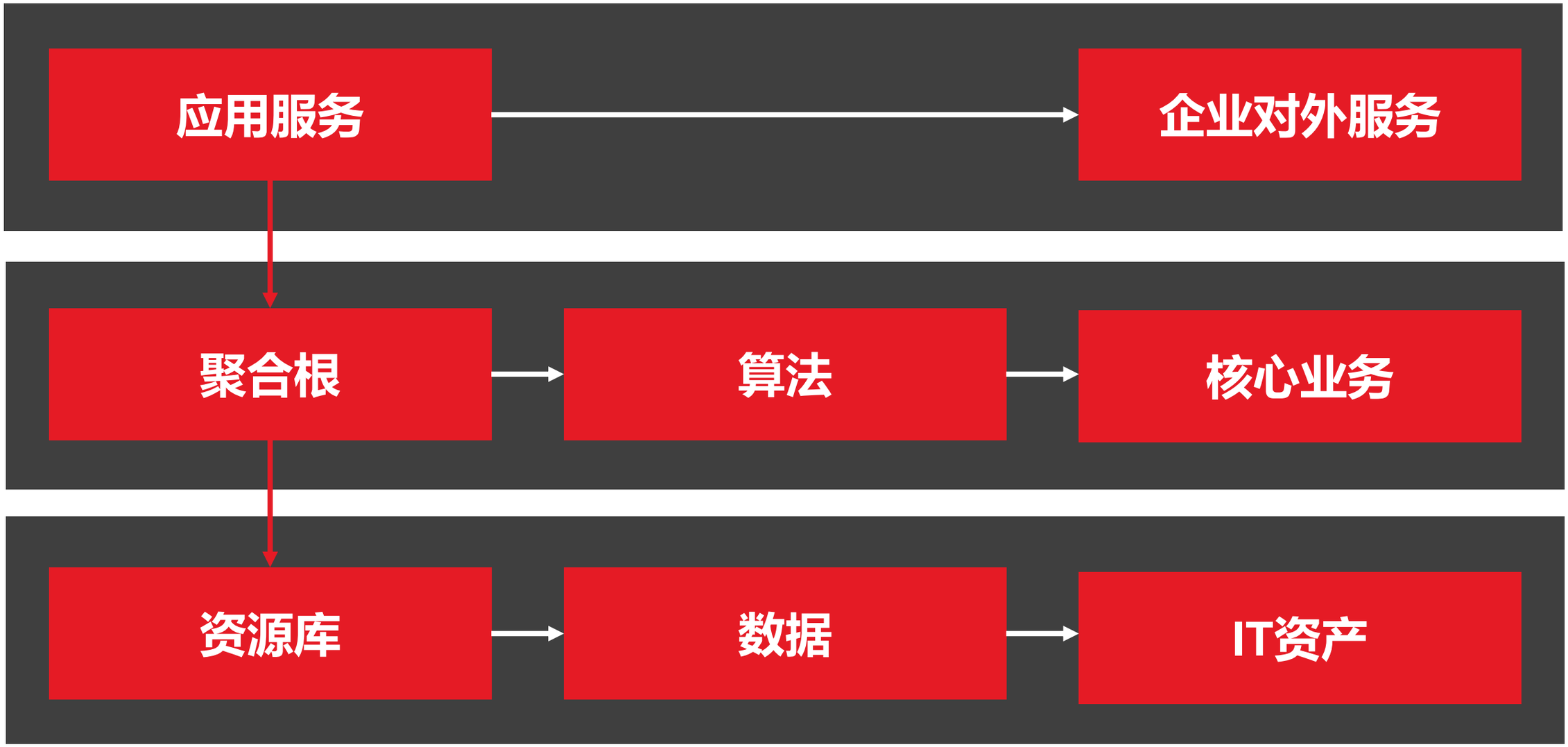
限界上下文的形态

进程

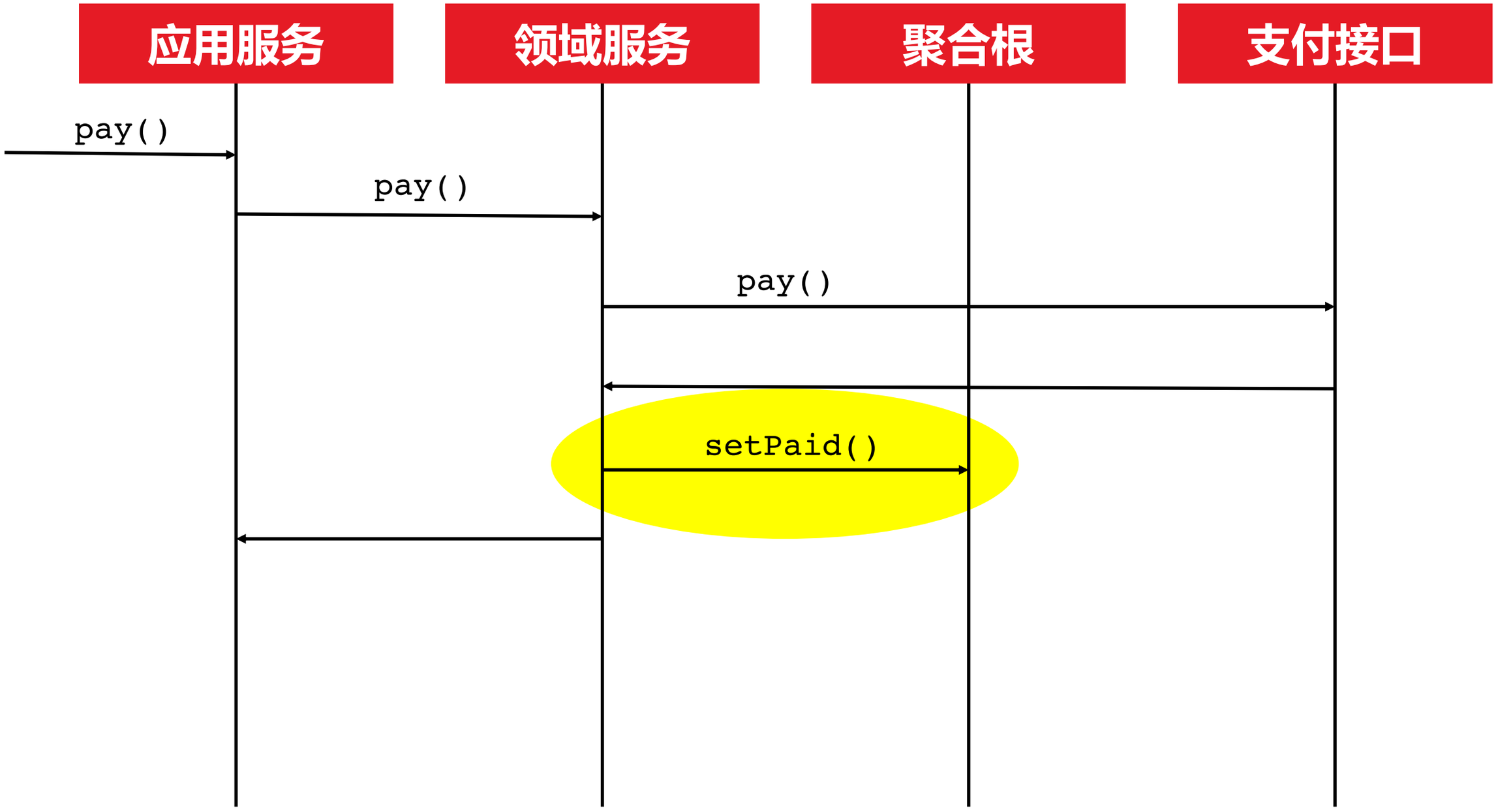
软件包（jar）

命名空间

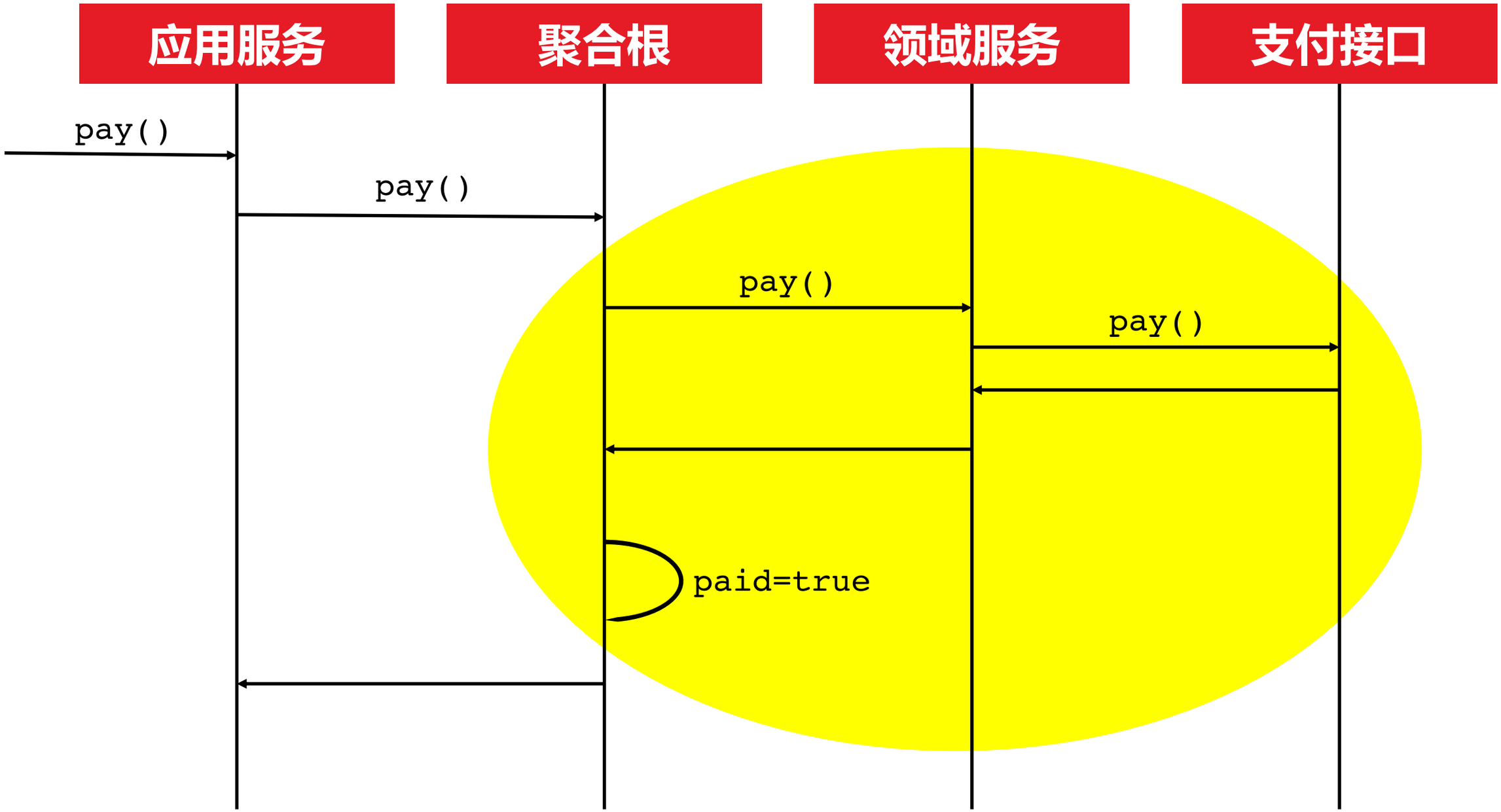
理想的战术设施



无法避免的贫血模型



无法避免的贫血模型



无法避免的贫血模型

```
public class Order {
    private PayService payService;
    private boolean paid;

    public void pay() {
        payService.pay();
        paid = true;
    }
}
```

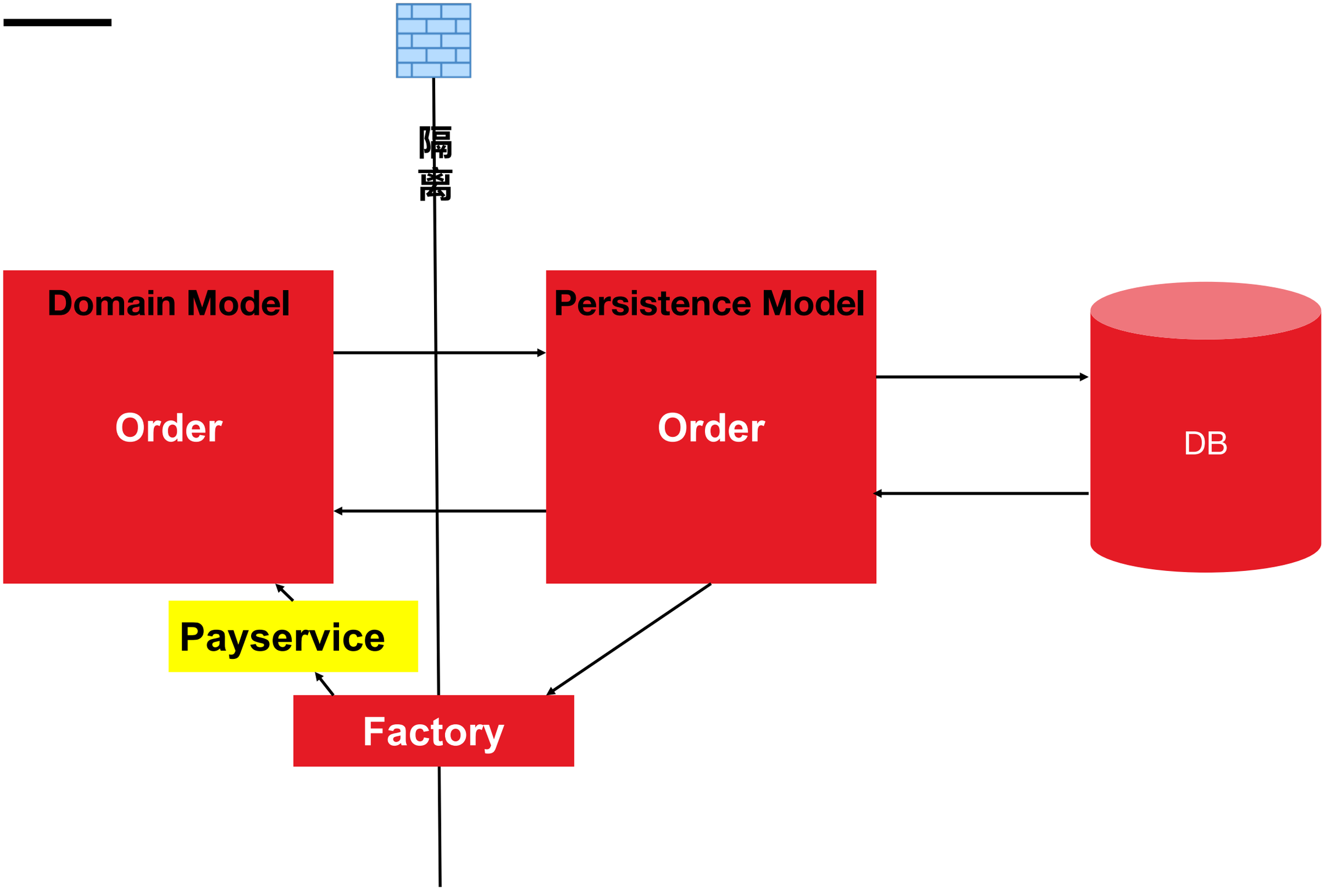
分离领域模型和持久化模型

```
@Entity
@Table(name = “App.Orders”)
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
     ……
}
```

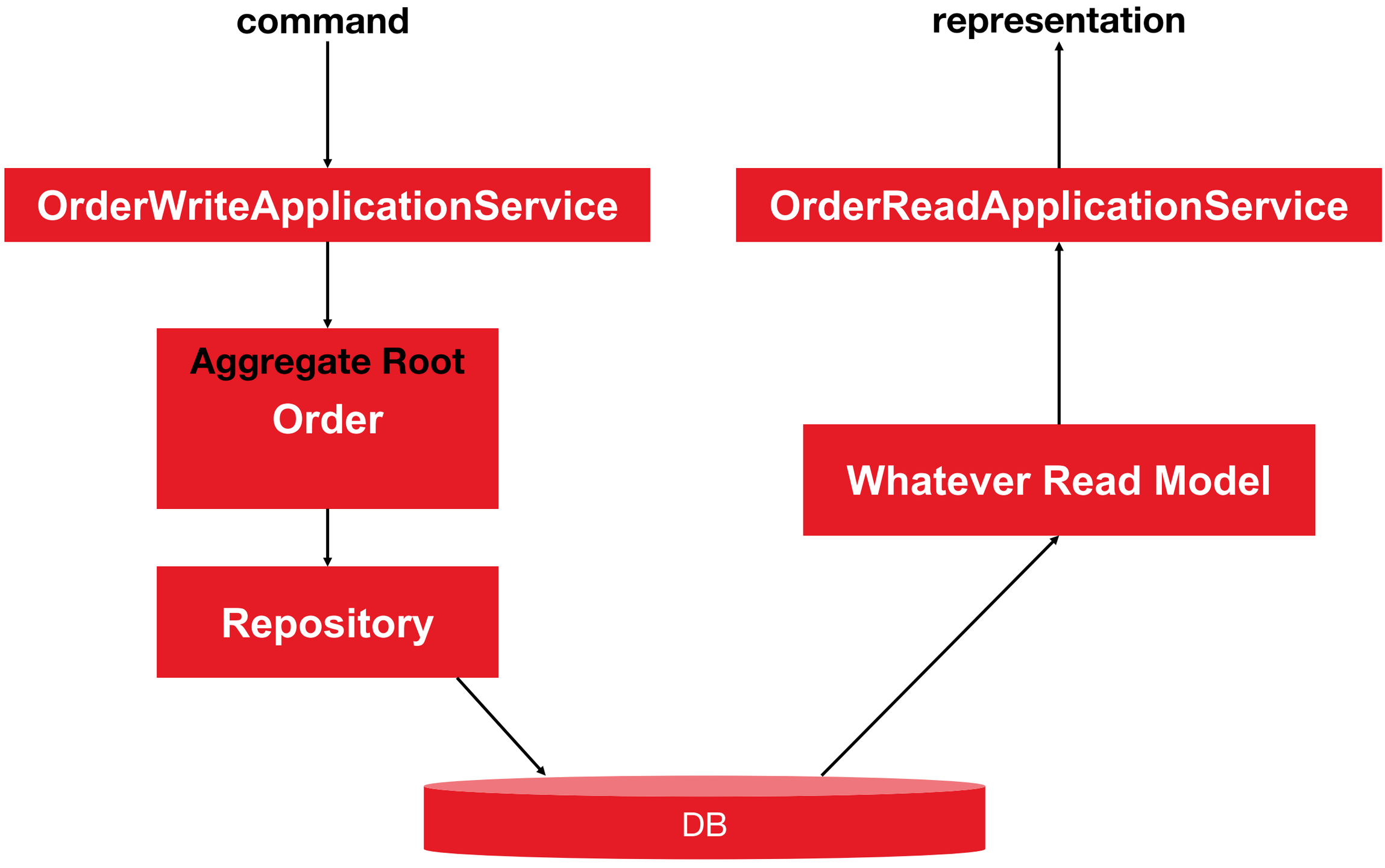
分离领域模型和持久化模型

```
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY, property = "ticketType")
@JsonSubTypes({
        @JsonSubTypes.Type(value = OilInspectionTicket.class, name = "OIL_INSPECTION"),
        @JsonSubTypes.Type(value = FieldServiceTicket.class, name = "FIELD_SERVICE"),
        @JsonSubTypes.Type(value = PublicTicket.class, name = "PUBLIC")
})
public abstract class Ticket {
    private String id;
    private int distributorId;
    private String factoryId;
    private String factoryName;
}
```

分离领域模型和持久化模型



读写分离


command
representation
OrderWriteApplicationService
OrderReadApplicationService
Aggregate Root
Order
Whatever Read Model
Repository
DB

显式化业务逻辑

```
public void sendMessage(List<User> users, String message){
    if(users.size() == 1){
        users.add(users.get(0));
    }
    doSend(users, message);
}
```

什么鬼？

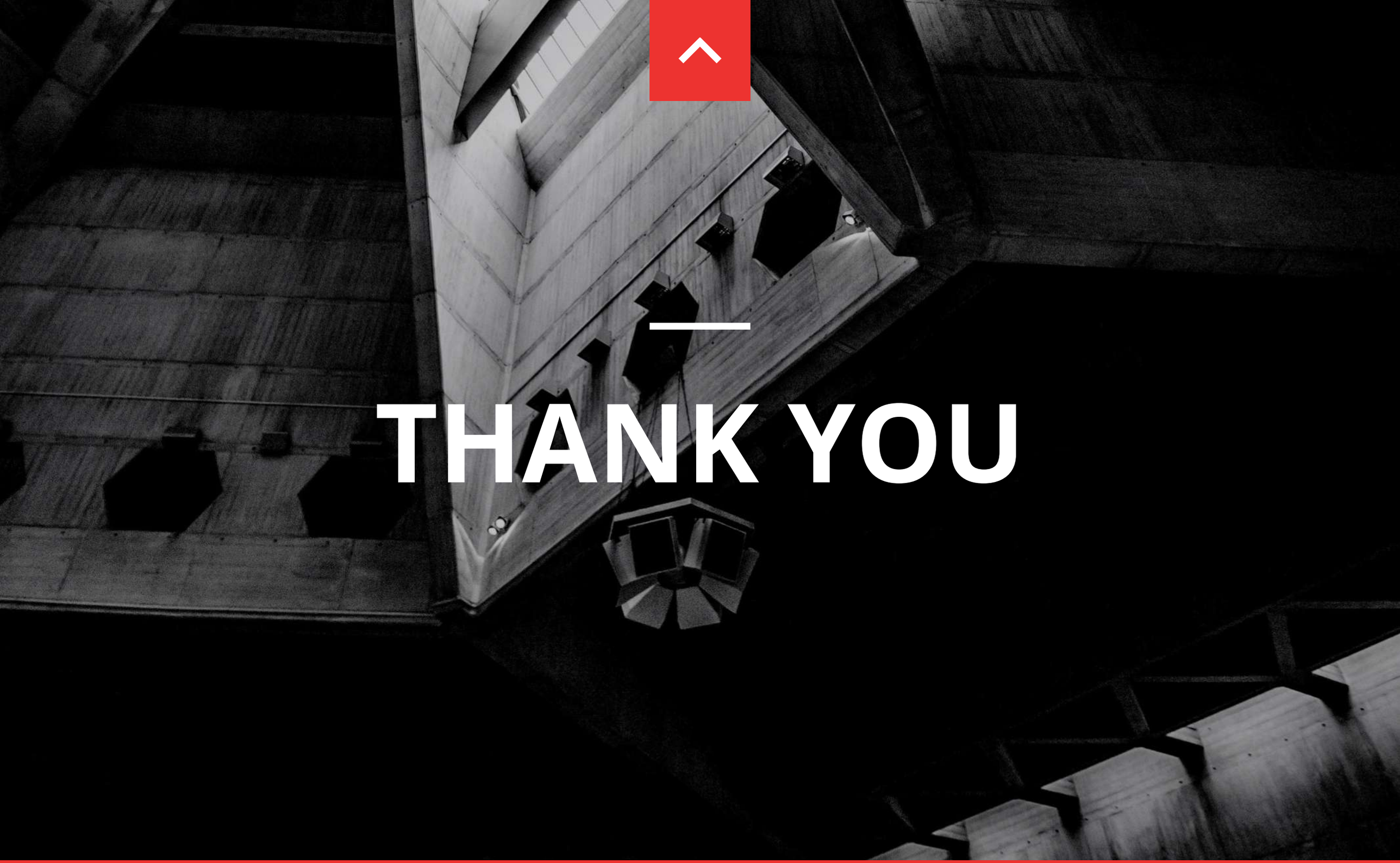
显式化业务逻辑

```
public class PurchaseOrder {
    private PurchaseOrderId id;
}
```

```
public class PurchaseOrderId {
    private final String id;
}
```

vs

```
public class PurchaseOrder {
    private String id;
}
```



THANK YOU

DDD CHINA