



南京大学

逸言

领域驱动设计 项目实战讲解

张逸



张逸

Bruce Zhang



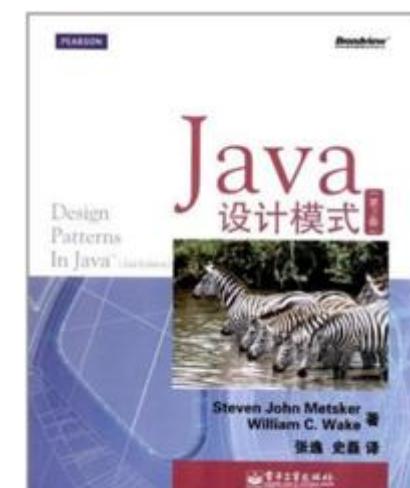
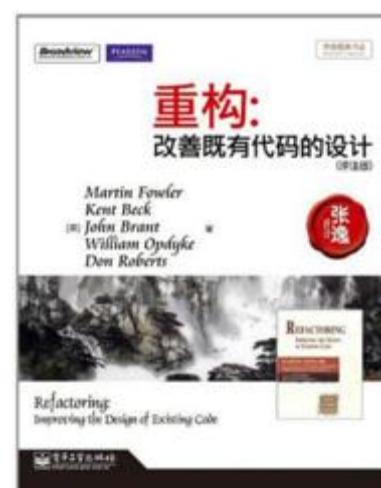
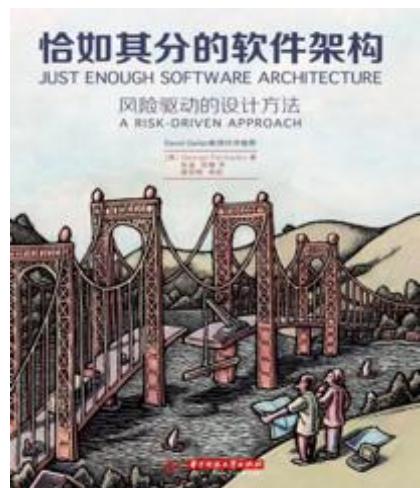
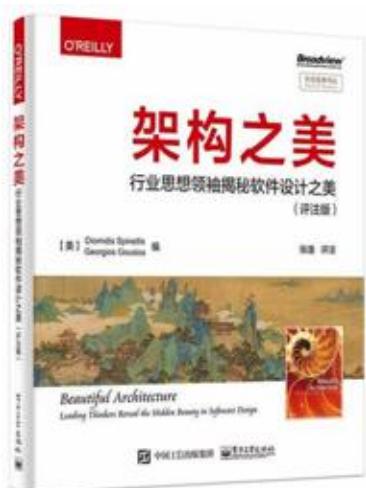
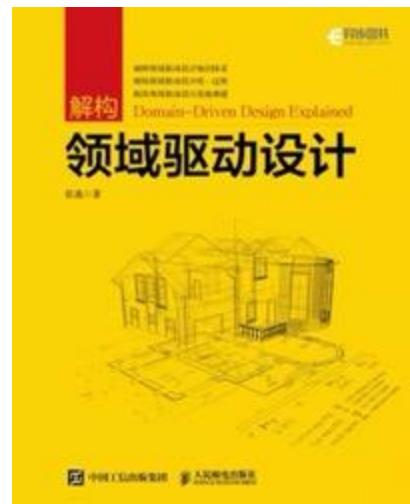
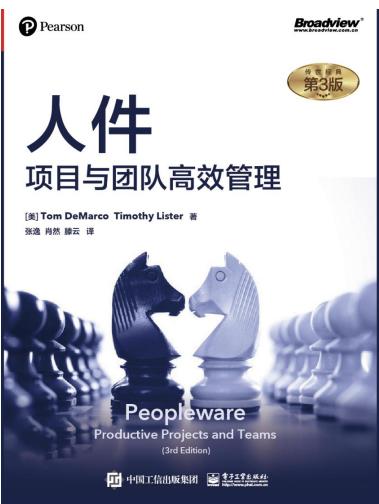
领域驱动设计布道师

数字现代化首席顾问

DDD研修会创始人

信通院应用现代化推进中心专家委员会委员，微软最有价值专家，南京大学DevOps+ Research Lab企业导师，南京大学软件工程卓越技术讲堂讲师，四川大学软件工程硕士，重庆工程学院特聘企业专家，民航总局第二研究所科技创新拔尖人才，K+全球软件研发行业创新峰会联席主席，阿里研发效能峰会出品人，DDD China社区卓越贡献者，2021年度影响力作者奖获得者，人民邮电出版社优秀作译者。

|| 著译作



|| 课程目录

1

领域驱动设计
基础

2

通过案例学习
领域驱动设计

3

领域建模的
重要性

4

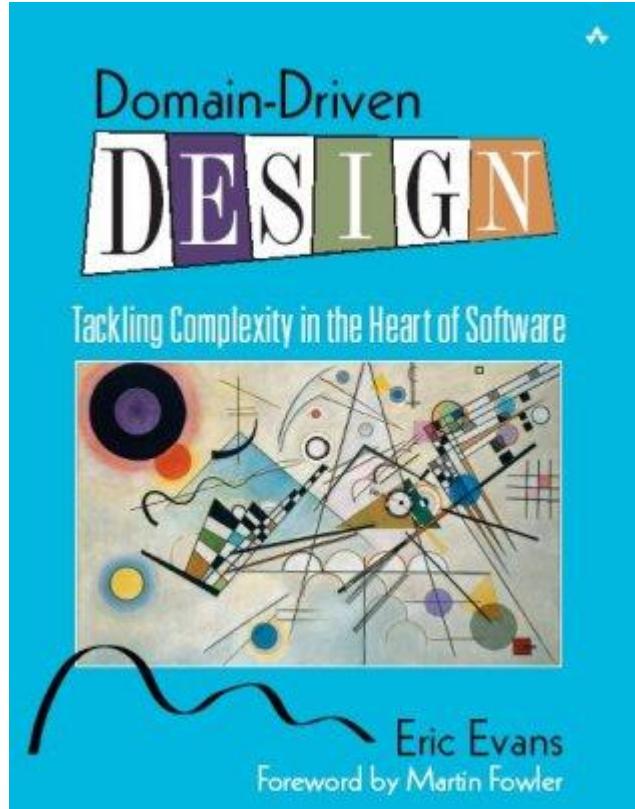
AI与DDD的
双向赋能



A paved road leads through a dense forest of tall, thin trees with green and yellow autumn foliage. The road is marked with white dashed lines and has fallen leaves scattered along its edges. The perspective of the road creates a sense of depth, leading towards a bright opening in the trees at the top of the frame.

1 领域驱动设计基础

|| 领域驱动设计的诞生



Domain-Driven Design

Tackling Complexity in the Heart of Software

2003年

|| DDD在行业软件的运用

领域驱动设计DDD在B端营销系统的实践

原创 HZ 美团技术团队 2024年05月23日 19:58 北京

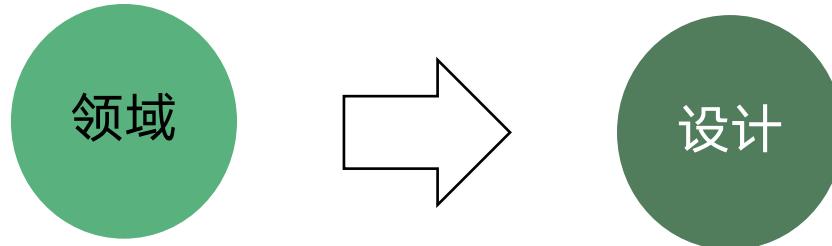


点击关注“美团技术团队”，阅读更多技术干货

总第590篇 | 2024年第010篇

本文整理自美团技术沙龙第73期《基于领域驱动设计（DDD）的架构演进和实践》，系统复杂性根源
于隐晦（难理解），耦合（难改动）和变化（难扩展），DDD正是应对系统复杂性的重要方法。本文针
对B端营销系统设计中的复杂性，从战略设计，战术设计到代码架构，详细介绍了DDD在各个阶段的
实践，期望为大家提供一些可供参考和借鉴的思路。

|| 解读领域驱动设计



领域驱动设计的立意是建立以领域为驱动力的过程体系，在这一核心驱动力的设计思想指导下，并没有死板僵化的构建过程来约束你。



©2025 张逸

|| 领域驱动设计的价值



运用分而治之思想，通过子领域与限界上下文对领域的划分降低**业务复杂度**

通过领域建模与统一语言，为核电行业提炼由领域模型构成的企业资产，通过复用降低**技术复杂度**

遵循领域驱动设计统一过程，打造标准而固化的软件构建过程，降低**工程复杂度**

|| 领域驱动设计的适用范围

适用于

- 业务复杂度高的产品型软件系统
- 面向企业开展应用架构设计
- 稳态系统的架构重构与升级
- 敏态系统的架构设计

不适用于

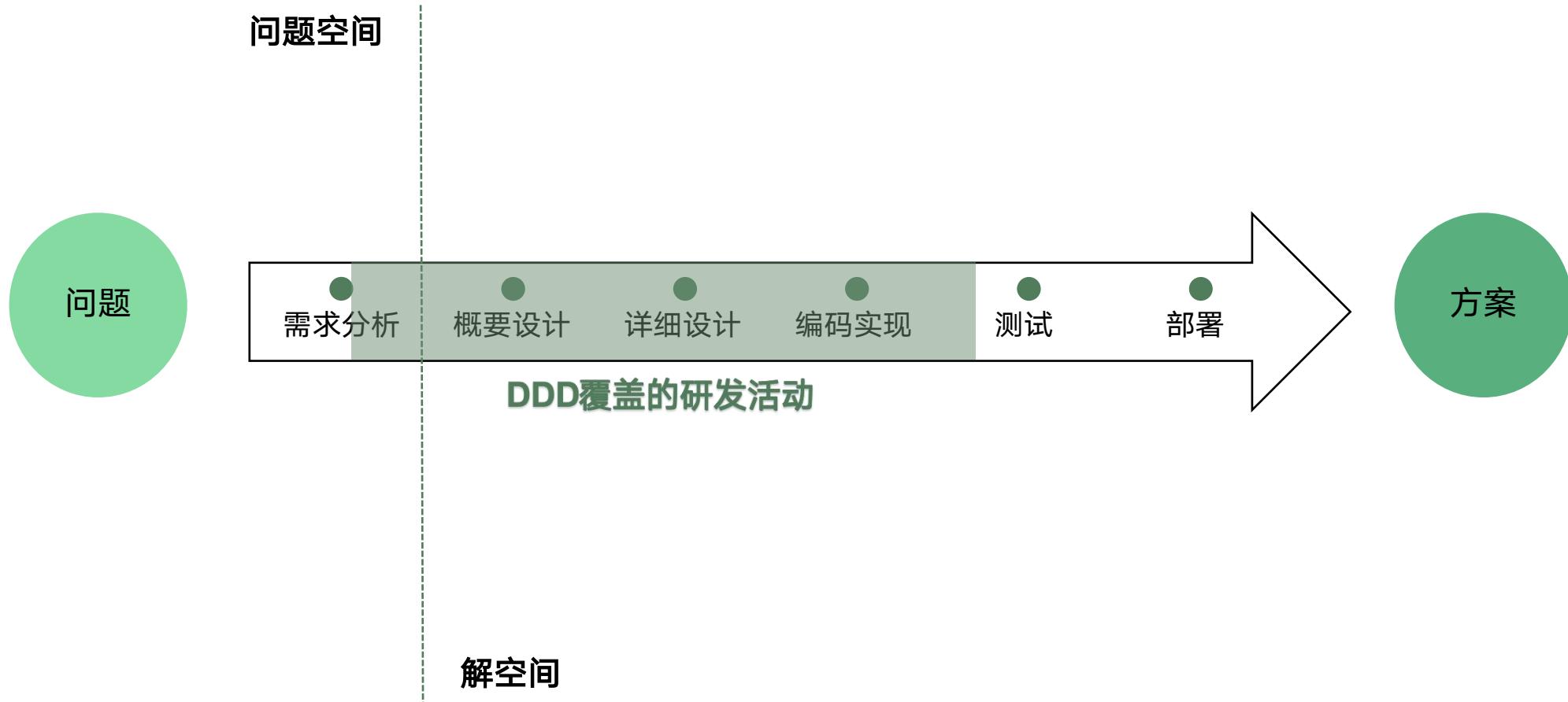
- 业务复杂度低的软件系统
- 以数据为中心的分析平台
- 前端UI设计与开发
- 解决和降低技术复杂度

项目制交付通常注重短期效率，资金来源通常也是基于合同制，比如某具体客户的转型预算；通常由临时团队负责交付，项目结束后交付团队就分散去负责下一个项目。

产品制交付注重长期提效，一个业务组件由固定的团队开发运维，迭代优化，持续改进，产品设计的理念也可以被用于组件设计，比如使用设计思维寻找组件的业务价值，业务变化点。

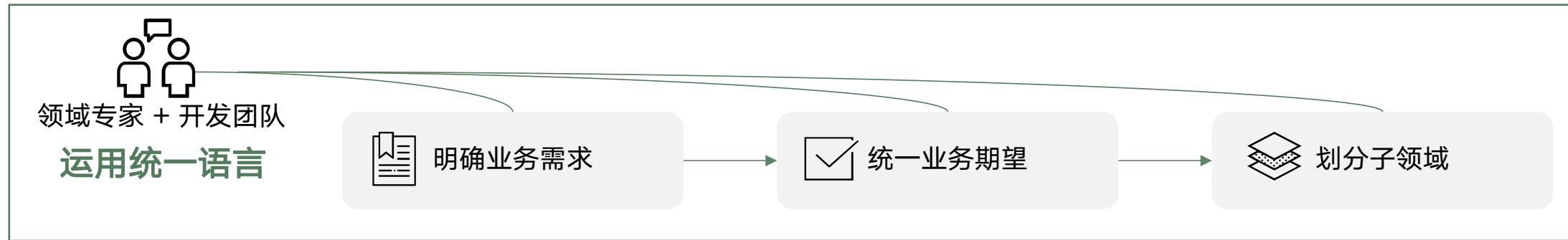


|| 软件研发的目的

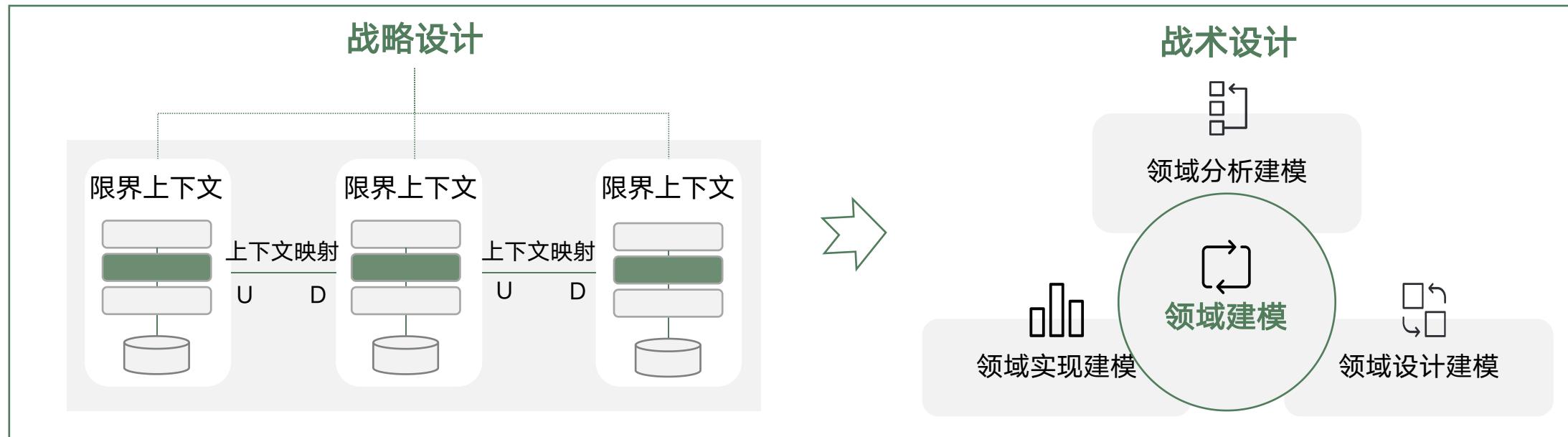


|| 领域驱动设计的研发过程

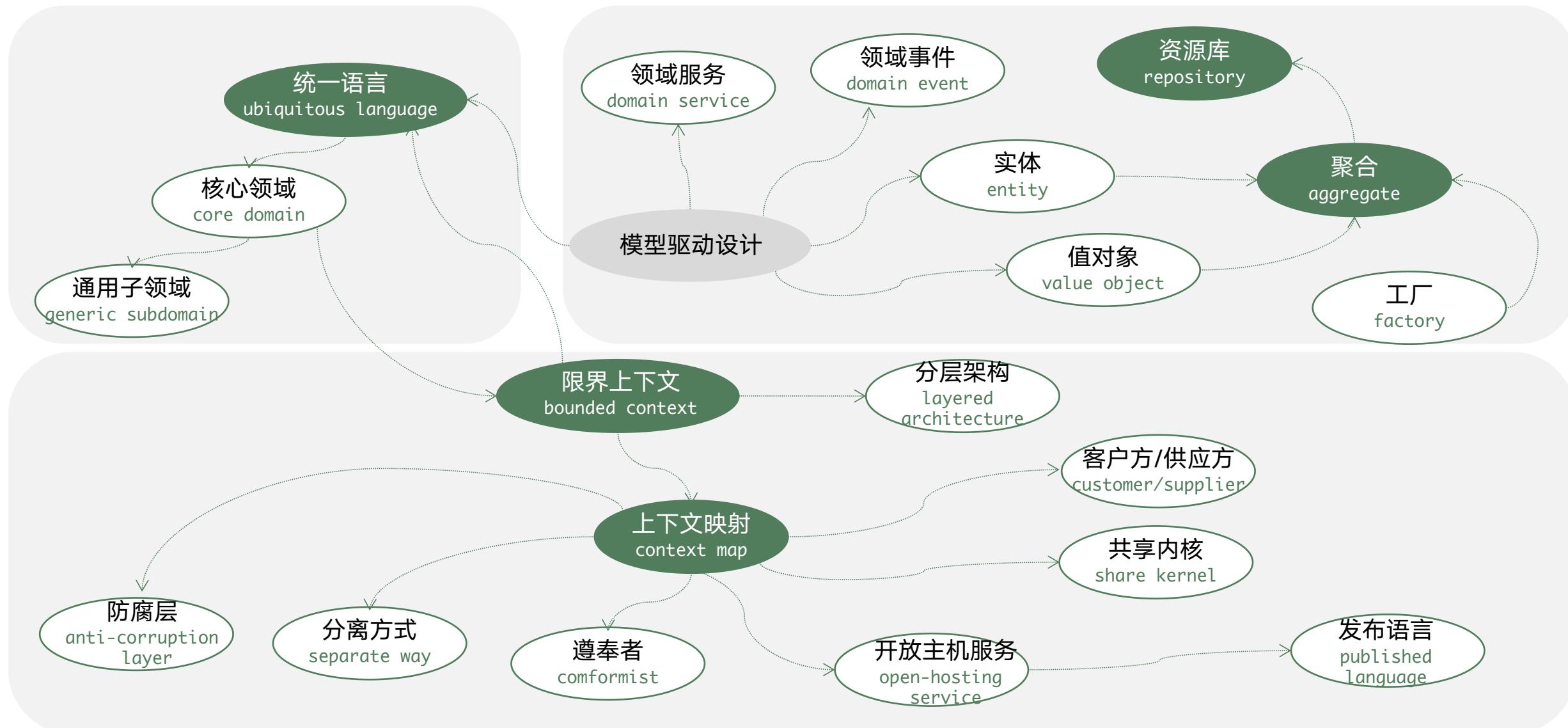
问题空间



解空间



|| 领域驱动设计的核心模式





通过案例学习领域驱动设计

2

01

从理想的OO到现实的DDD

|| 案例一：超市收银员

Cashier

```
public class Cashier {  
    public void charge(Customer myCustomer, float payment) {  
        Wallet theWallet =  
myCustomer.getWallet();  
        if  
(theWallet.getTotalMoney() >=  
payment) {  
  
theWallet.subtractMoney(paymen  
t);  
        } else {  
            throw new  
NotEnoughMoneyException();  
        }  
    }  
}
```

Customer

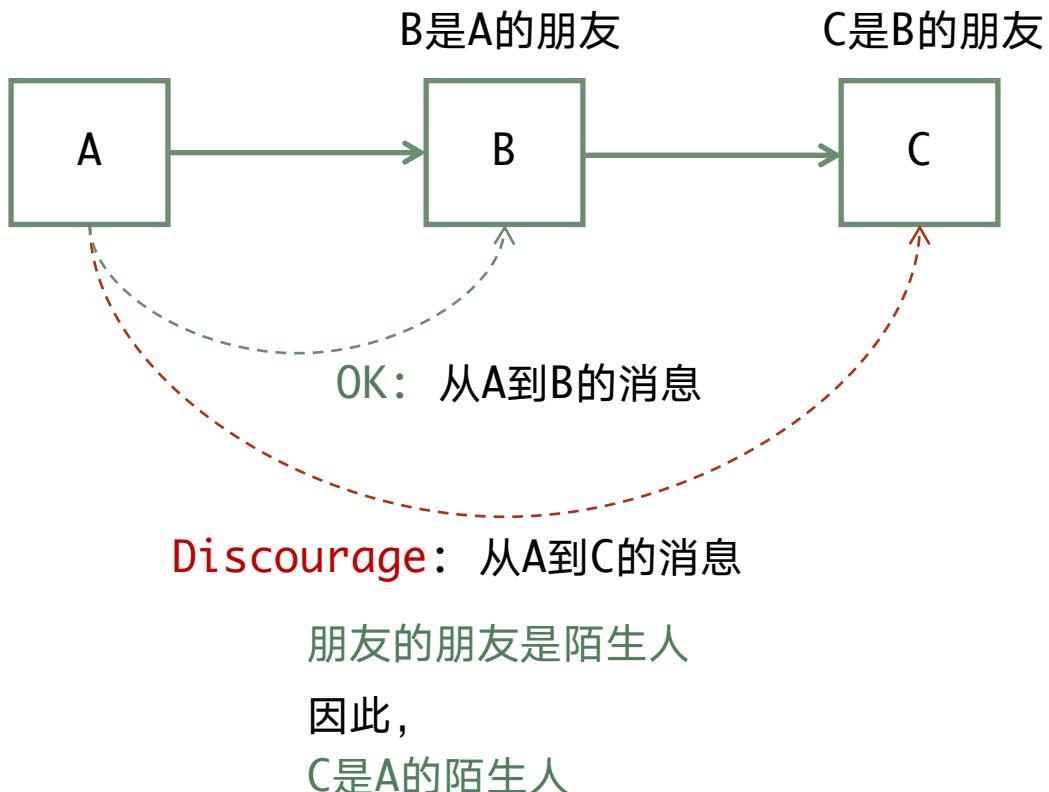
```
public class Customer {  
    private String firstName;  
    private String lastName;  
    private Wallet myWallet;  
  
    public Customer(String  
firstName, String lastName,  
Wallet myWallet) {  
        this.firstName =  
firstName;  
        this.lastName = lastName;  
        this.myWallet = myWallet;  
    }  
  
    public String getFirstName(){  
        return firstName;  
    }  
    public String getLastname(){  
        return lastName;  
    }  
    public Wallet getWallet(){  
        return myWallet;  
    }  
}
```

Wallet

```
public class Wallet {  
    private float value;  
    public Wallet(float value) {  
        this.value = value;  
    }  
    public float getTotalMoney()  
{  
        return value;  
    }  
    public void addMoney(float  
deposit) {  
        value += deposit;  
    }  
    public void  
subtractMoney(float debit) {  
        value -= debit;  
    }  
}
```

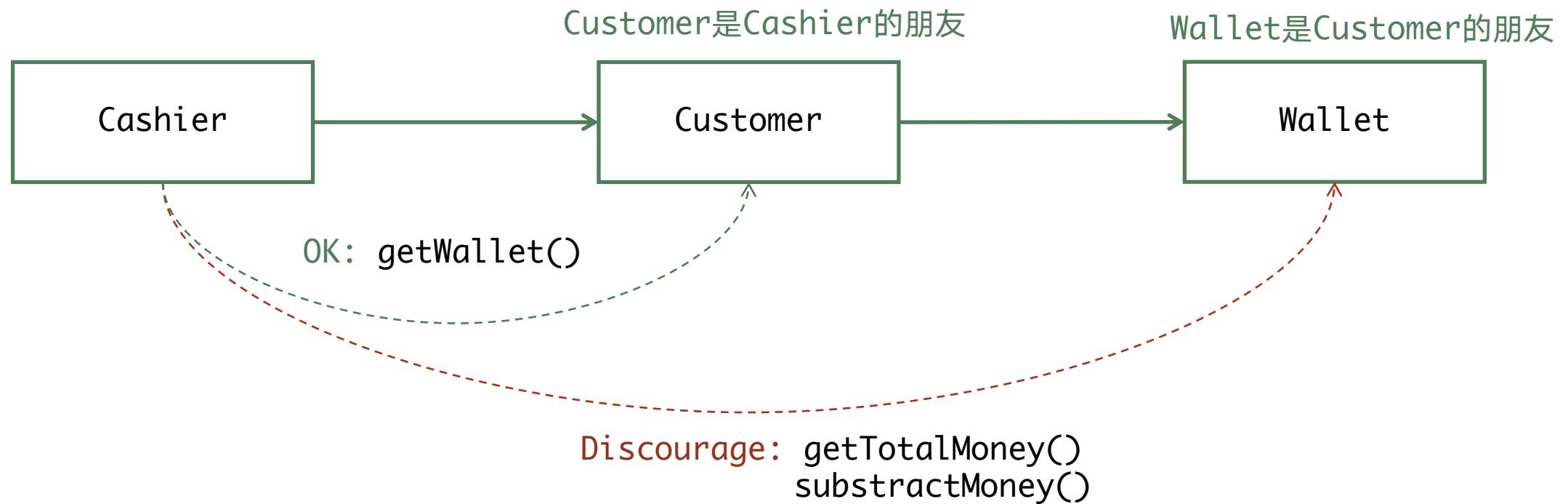


|| 迪米特法则



- 两个对象是否是好朋友，特征：
- 一个对象是另一个对象方法的参数
 - 一个对象是另一个对象的属性
 - 一个对象被另一个对象创建

|| 迪米特法则



|| 修改后的超市收银员

Cashier

```
public class Cashier {  
    public void charge(Customer  
myCustomer, float payment) {  
        myCustomer.pay(payment);  
    }  
}
```

Customer

```
public class Customer {  
    private String firstName;  
    private String lastName;  
    private Wallet myWallet;  
  
    public Customer(String  
firstName, String lastName,  
Wallet myWallet) {  
        this.firstName =  
firstName;  
        this.lastName = lastName;  
        this.myWallet = myWallet;  
    }  
    public void pay(float payment)  
{  
    if  
(myWallet.isEnough(payment)) {  
        myWallet.subtractMoney(payment);  
    } else {  
        throw new  
NotEnoughMoneyException();  
    }  
}
```

Wallet

```
public class Wallet {  
    private float value;  
    public Wallet(float value) {  
        this.value = value;  
    }  
    public float getTotalMoney()  
{  
    return value;  
}  
    public void addMoney(float  
deposit) {  
    value += deposit;  
}  
    public void  
subtractMoney(float debit) {  
    value -= debit;  
}  
    public boolean  
isEnough(float payment) {  
    return value >= payment;  
}
```



|| 隐私法则与最小知识法则

隐私法则

- Customer: Wallet是Customer的隐私

最小知识法则

- Cashier: 对Wallet的操作增加了Cashier的负担

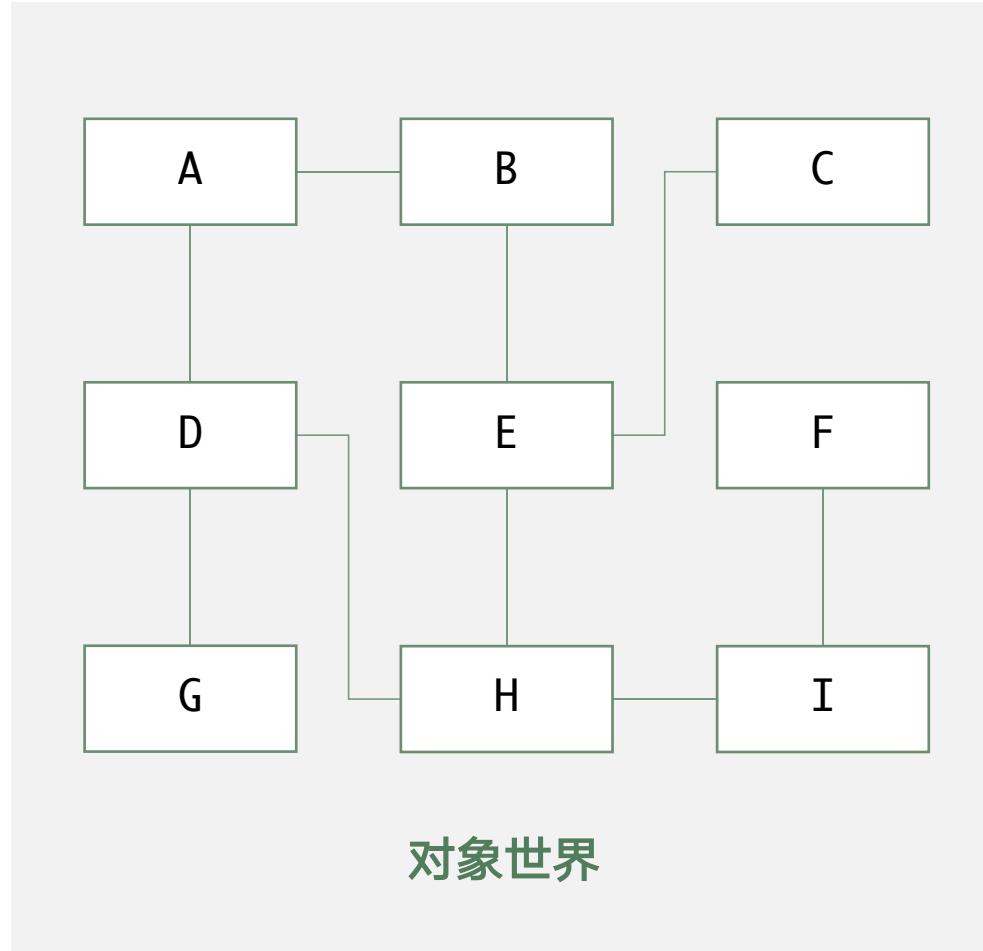
两个协作对象：

- 如果同时违背这两个原则，就会破坏封装，使得一个对象成为另一个对象的数据提供者
- 如果同时遵循这两个原则，就能最大程度降低依赖（耦合），使得两个对象之间产生良好的行为协作



©2025 张逸

|| 对象世界的哲学问题



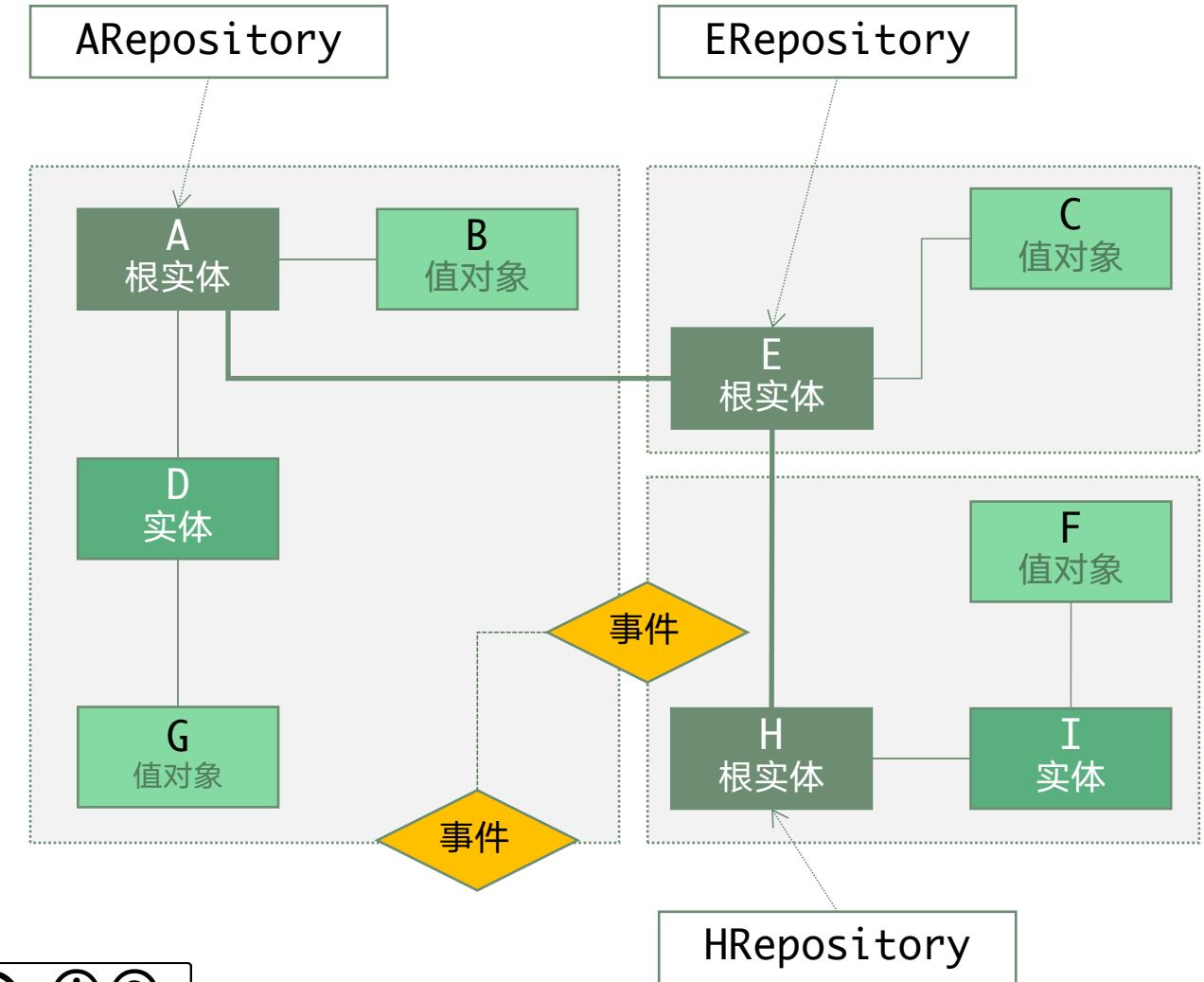
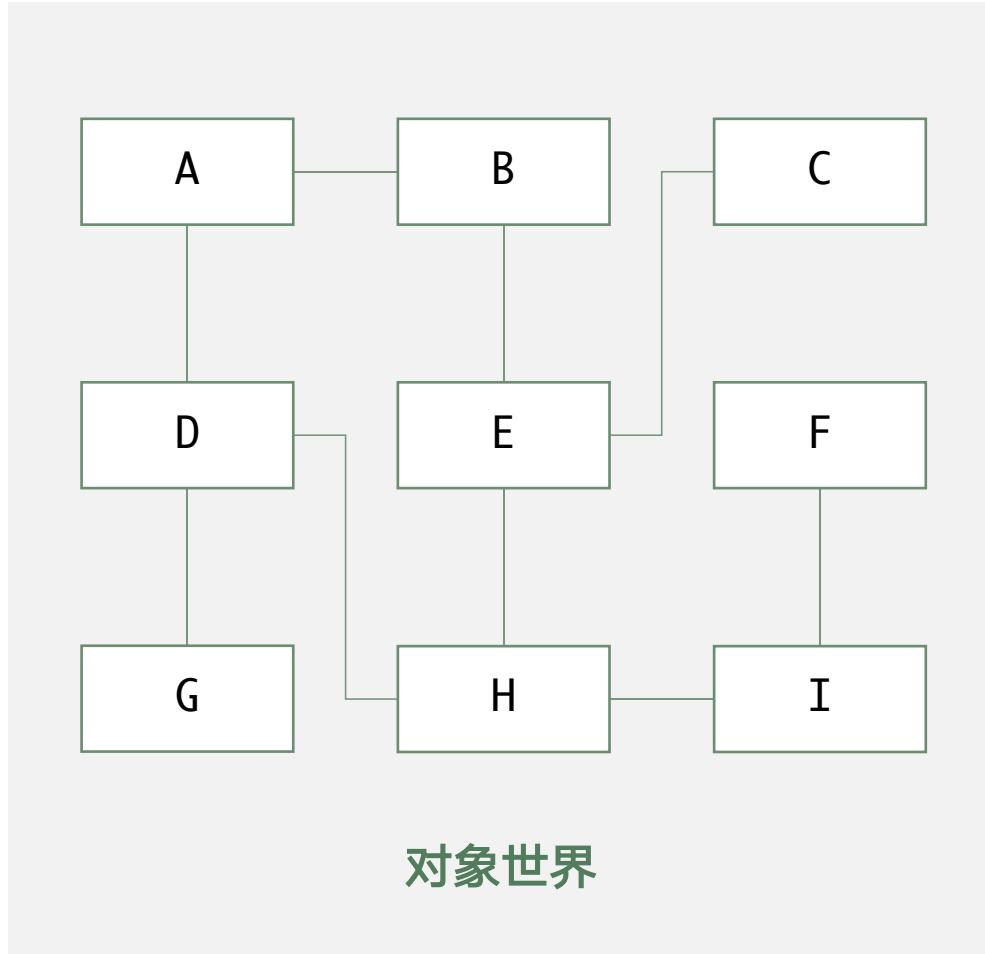
领域对象的哲学三问题：

- 我是谁：对象代表什么身份，它拥有的信息是完整的吗？
- 从哪里来：对象诞生自何处？
- 到哪里去：对象消亡了，会回到哪里？

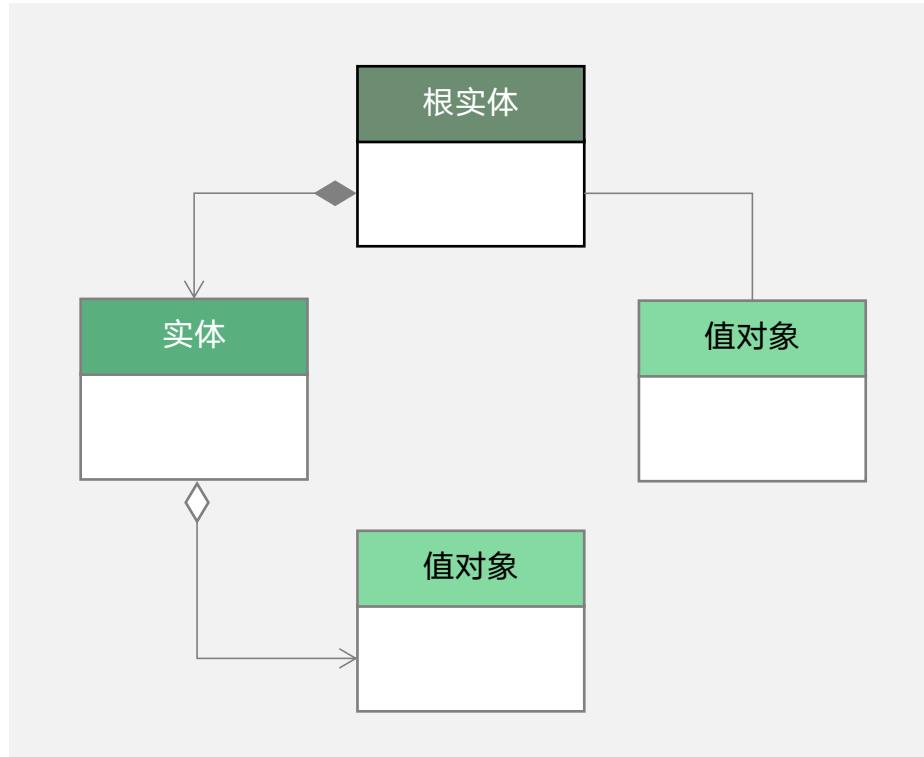
领域对象的现实问题：

- 问题一：领域模型对象如何与数据库协作？
- 问题二：领域模型对象的加载以及对象间的关系该如何处理？
- 问题三：领域模型对象在身份上是否存在泾渭分明的差别？
- 问题四：领域模型对象彼此之间如何能弱依赖地完成状态的变更通知？

|| 施加设计约束的领域设计模型

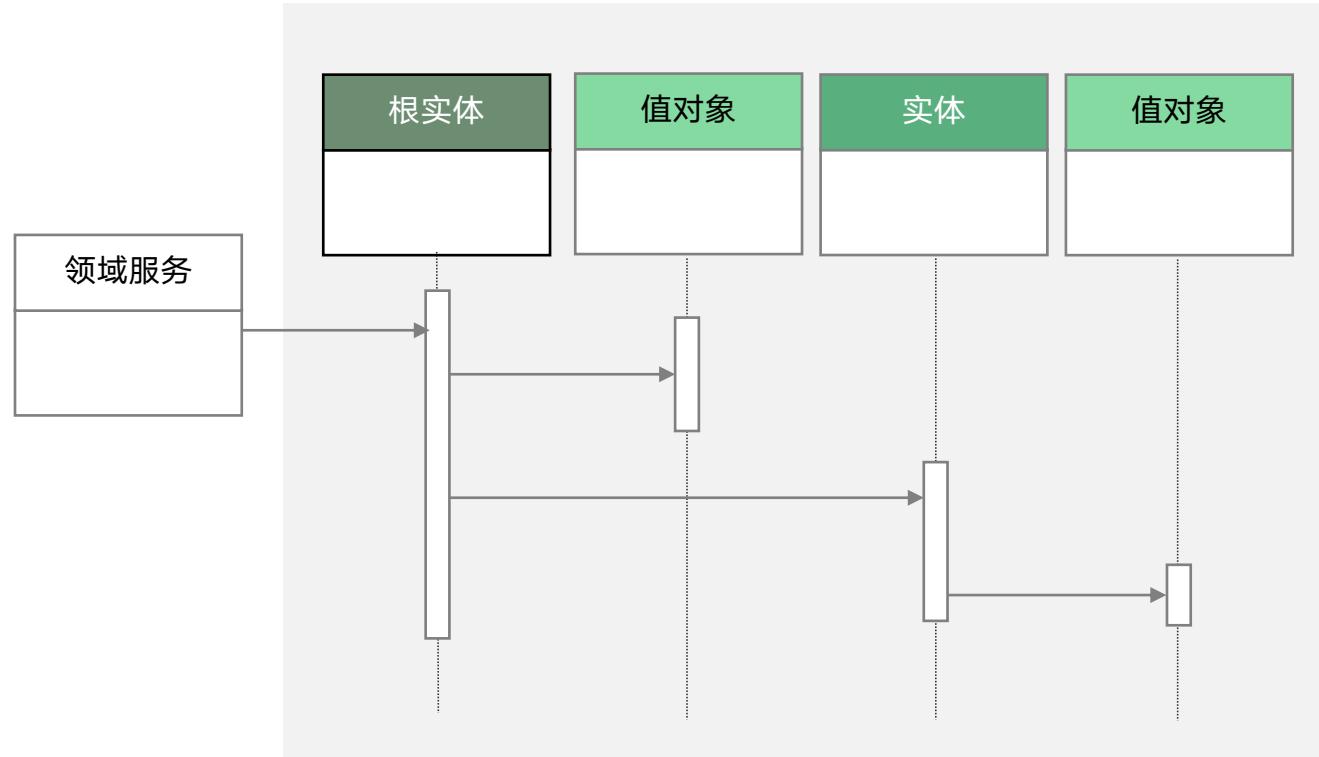


|| 聚合是领域模型的概念边界



聚合的静态结构

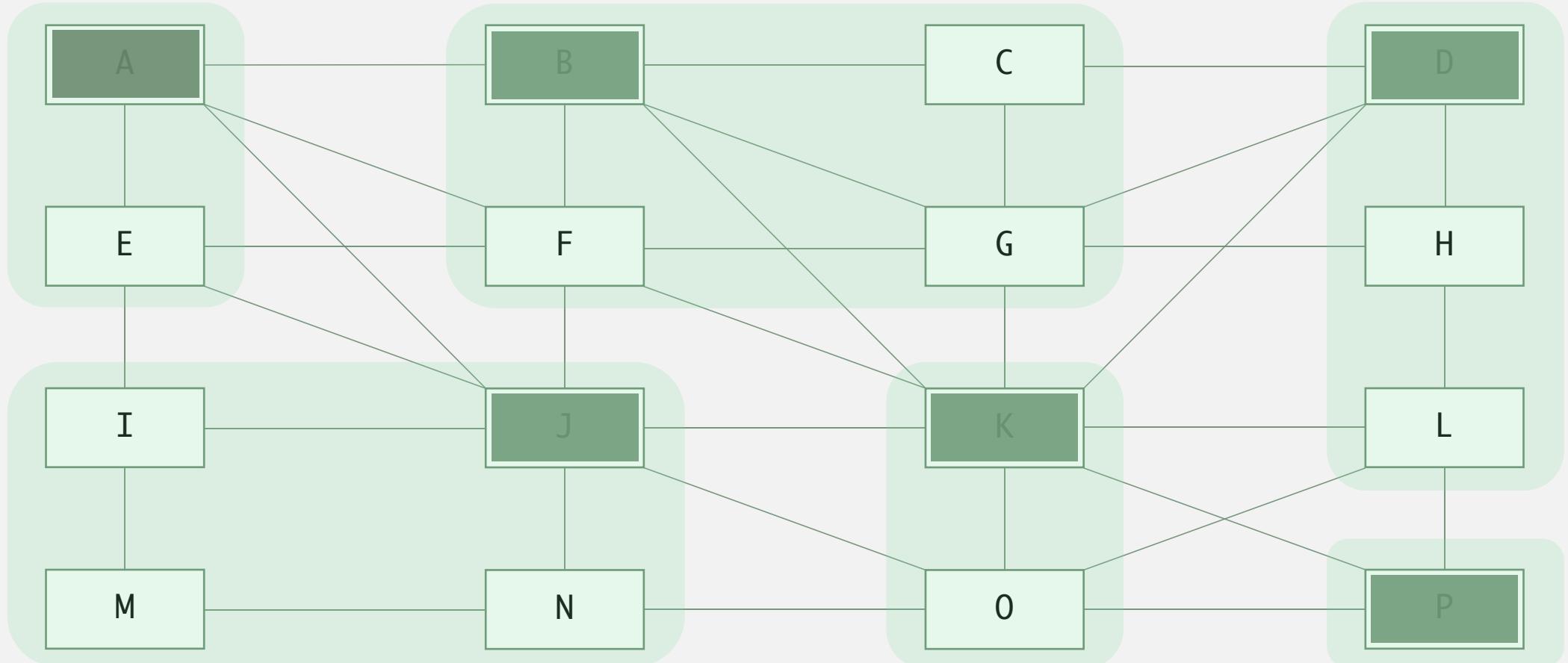
聚合是领域模型的概念边界。封装在聚合边界内的领域模型包括实体和值。它们组成一棵树，每棵树只能由一个根，且必须为实体，可称为**根实体**。



聚合的动态协作

在聚合的内部，实体与值对象的协作完全遵循**面向对象的行为协作模式**，避免设计为贫血模型。**根实体**作为聚合的唯一出口与入口，通过它与外界协作。

|| 为什么引入聚合?



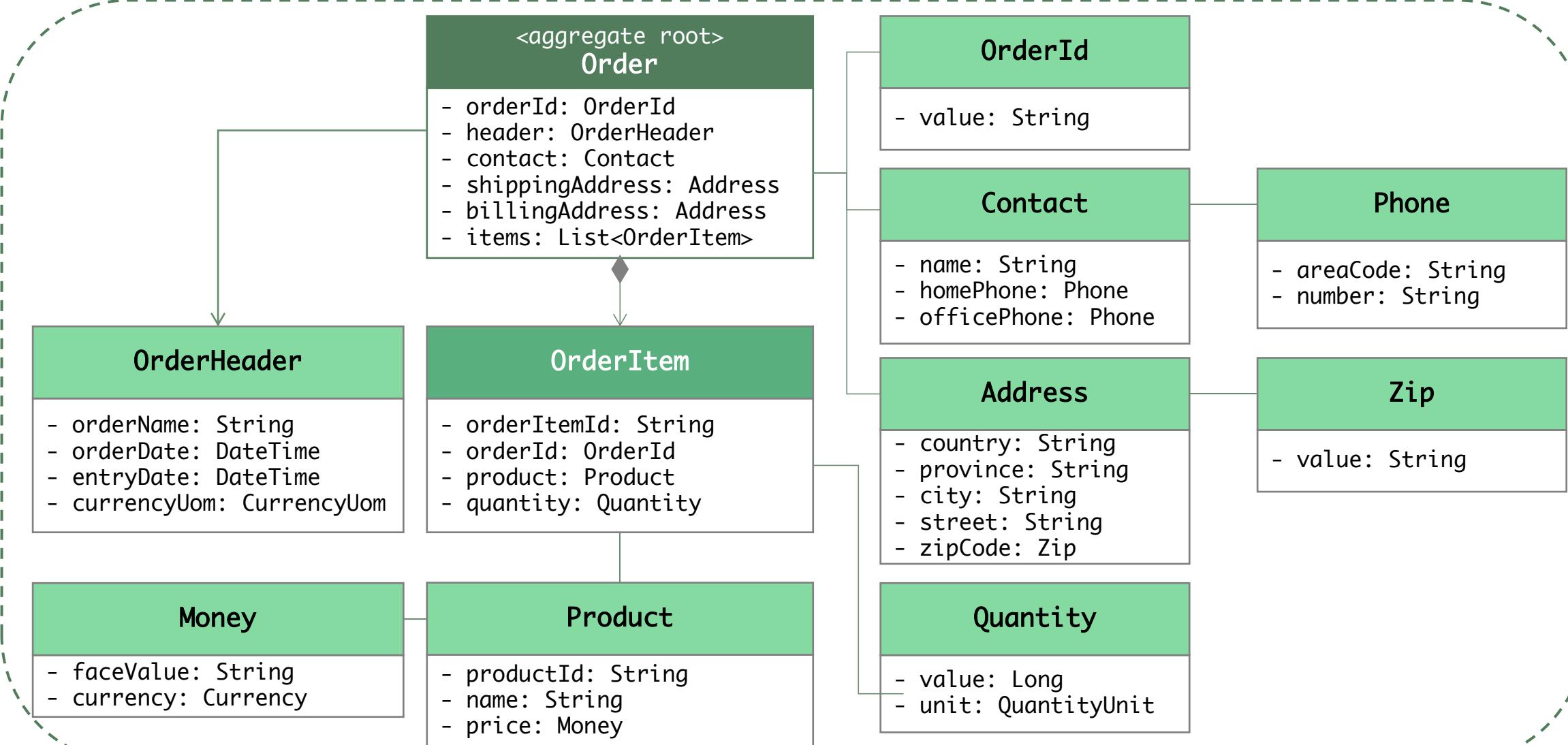
为领域模型带来复杂度的原因:

- 领域对象的数量
- 领域对象之间依赖的数量

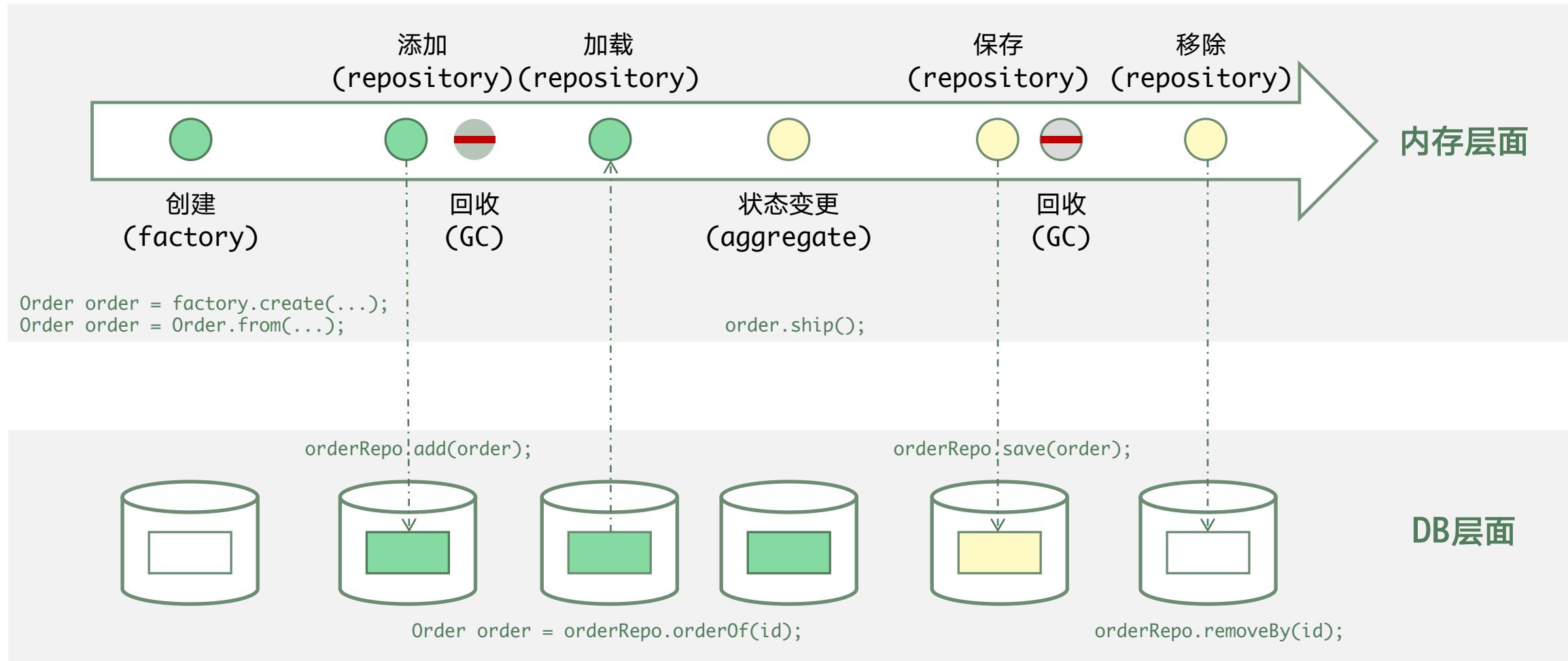
为聚合建立约束条件:

- 聚合之间只能通过聚合根建立关系
- 根实体是聚合唯一的入口和出口

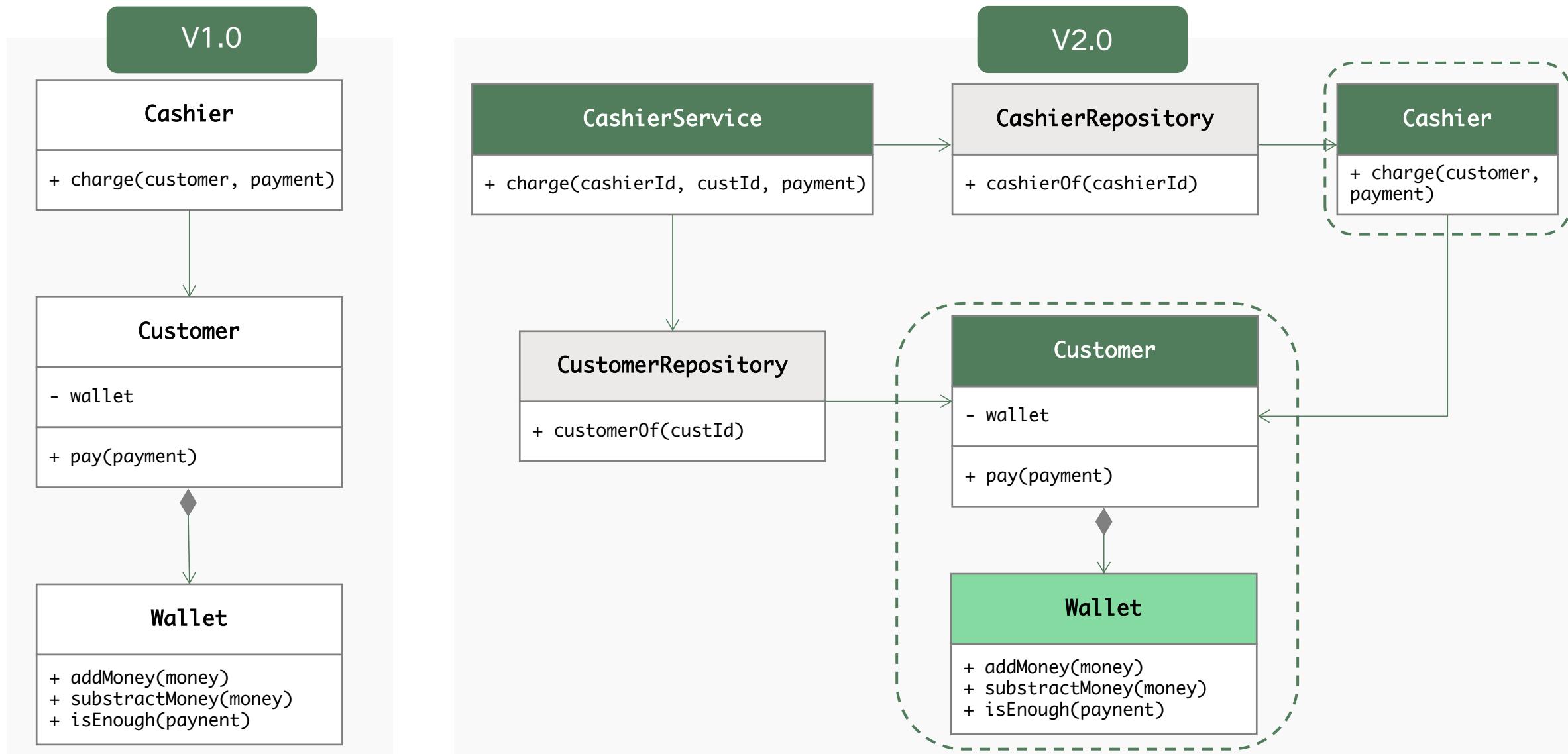
|| 订单聚合



|| 管理聚合的生命周期



|| 超市收银员：引入聚合

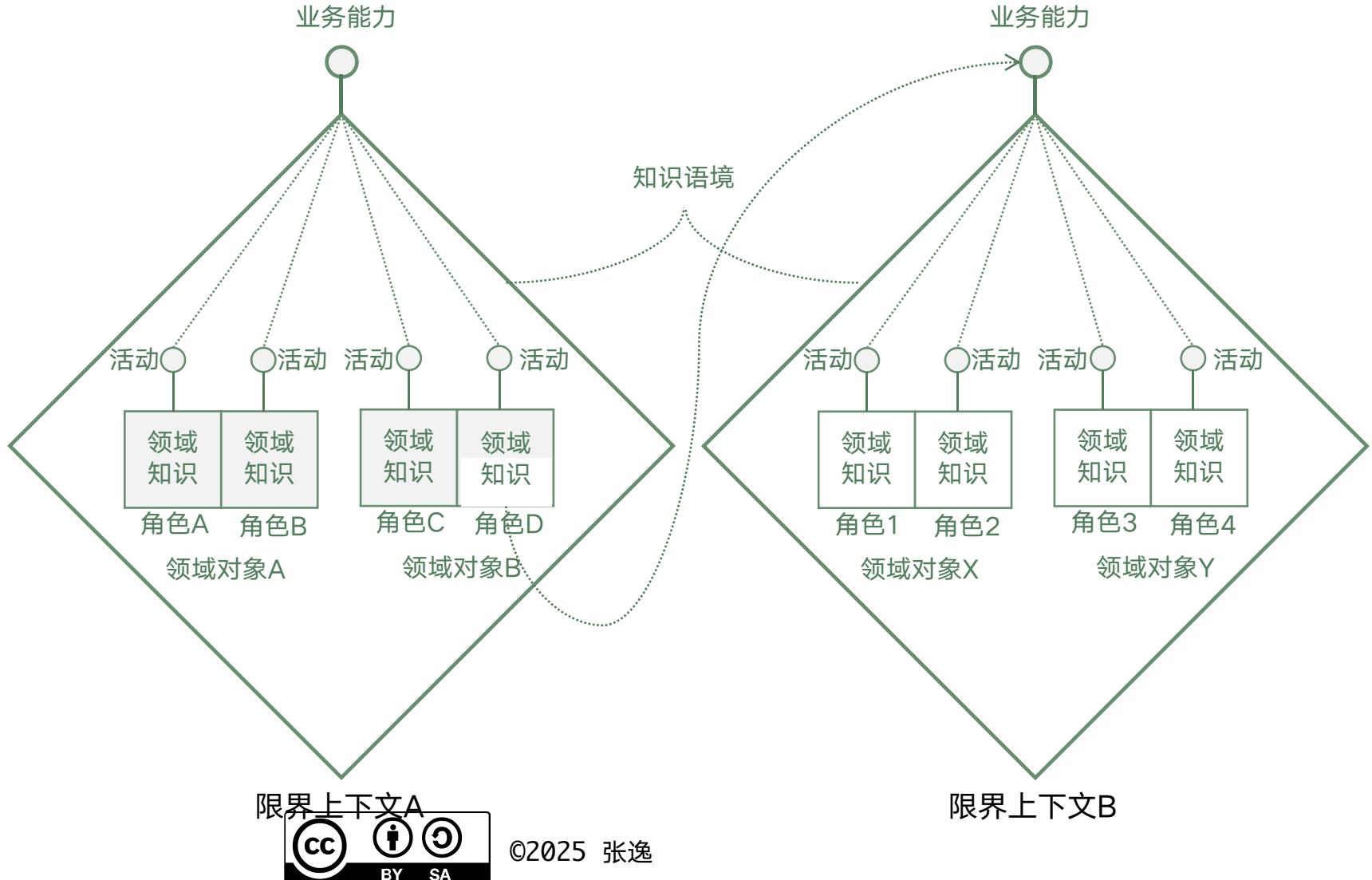


|| 限界上下文

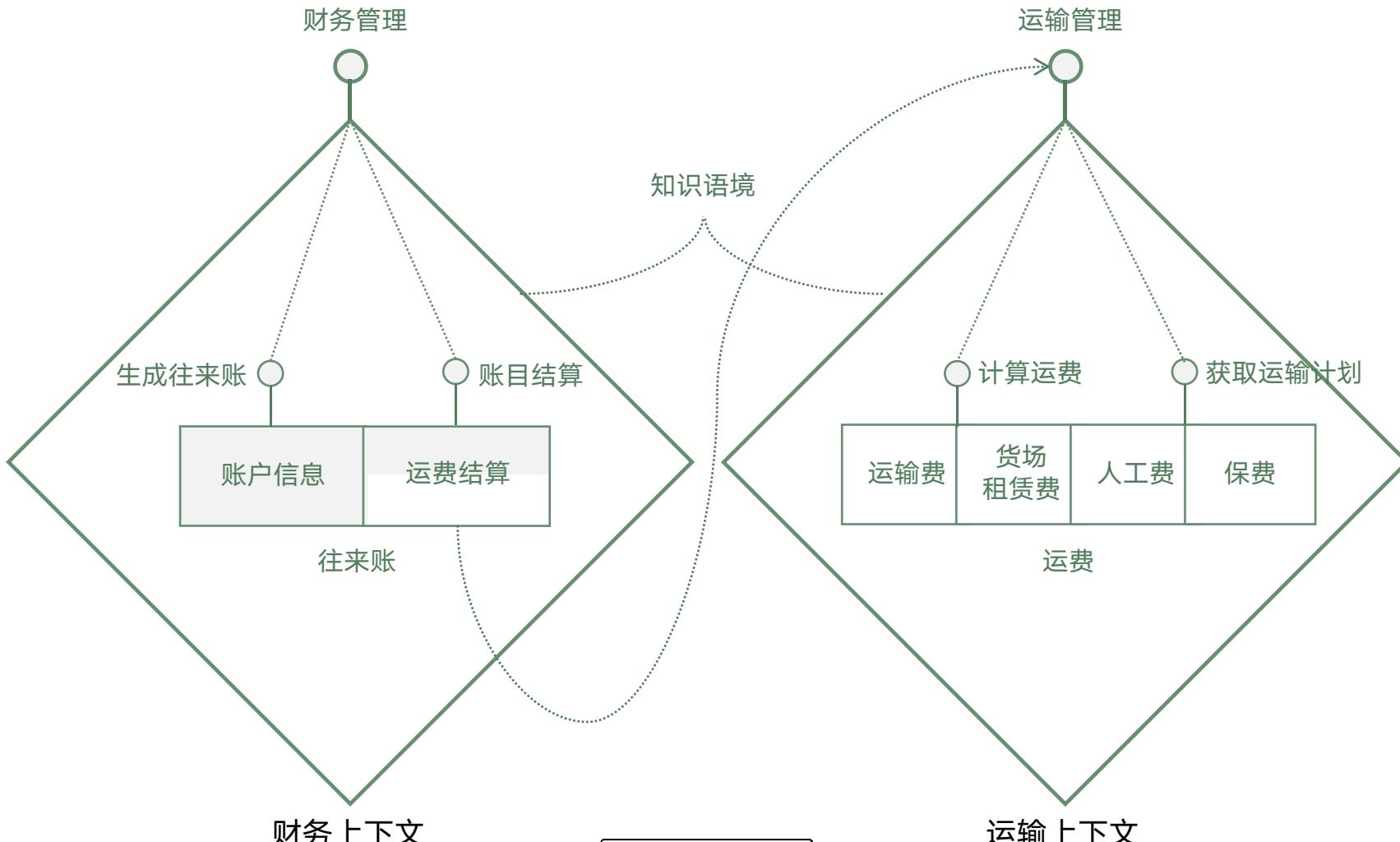


|| 限界上下文的定义

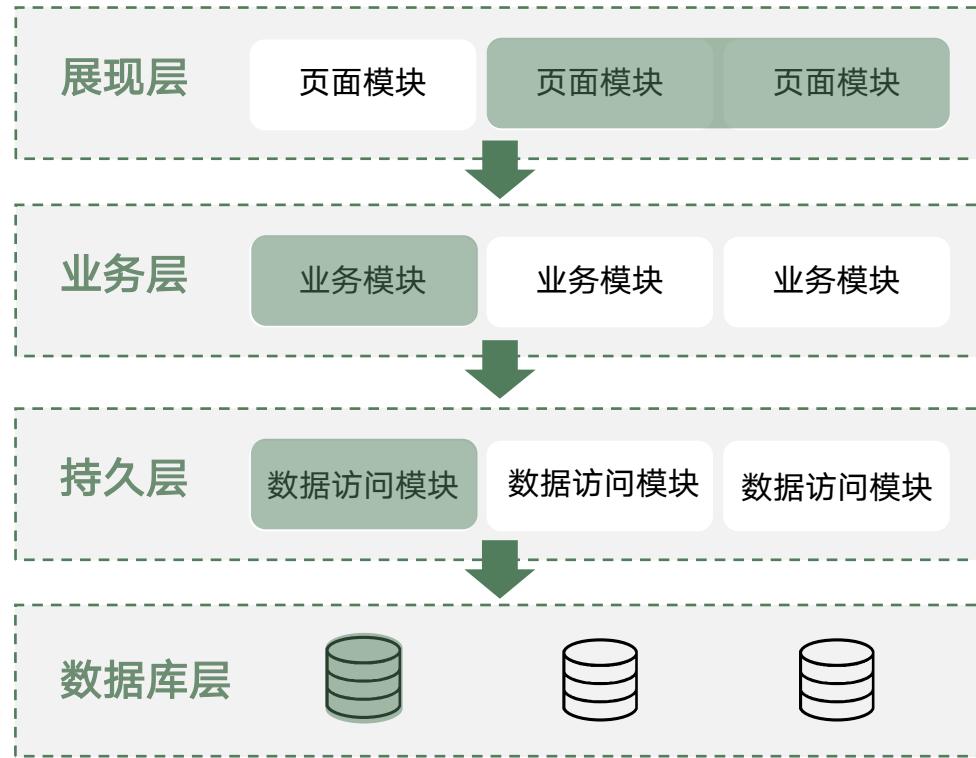
封装了领域知识的领域对象在知识语境的界定下，扮演不同的角色，执行不同的活动，共同对外公开内聚的业务能力。



|| 案例：限界上下文的定义

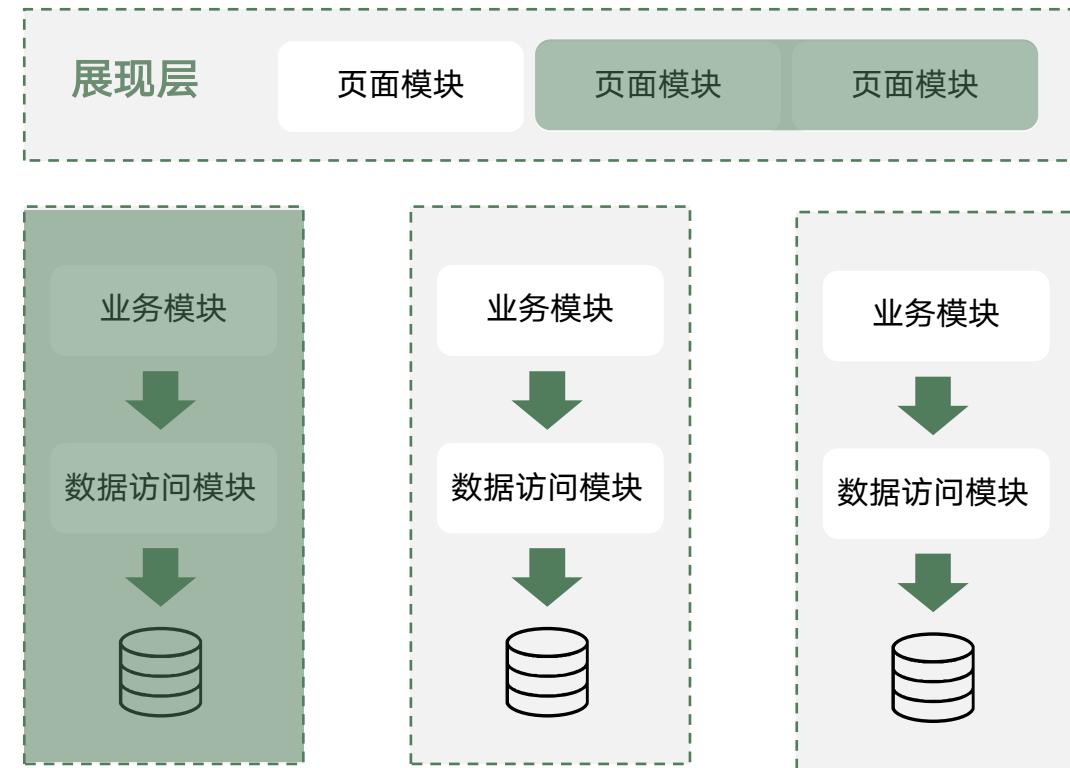


|| 模块与限界上下文



业务模块

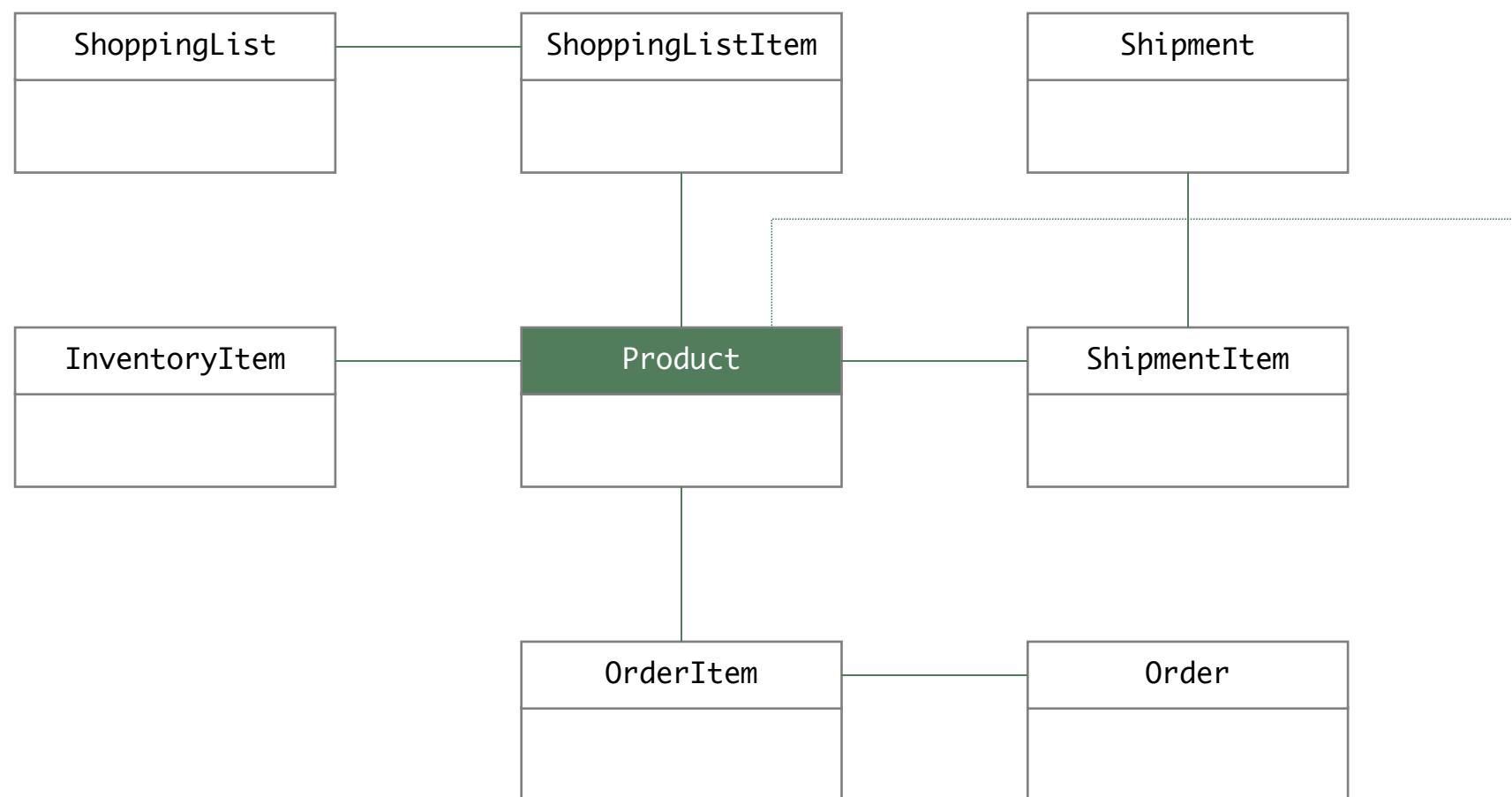
模块首先按照**技术维度**对整个架构进行**水平分层**，业务模块只是业务层的一部分，不能提供完整的业务能力。当业务需求发生变化时，**架构的每一层都可能会受到影响**。



限界上下文

限界上下文首先按照**领域维度**对整个架构进行**纵向切分**，然后再按照技术关注点进行**水平分层**，并形成知识语境的边界，使得**限界上下文在保证概念完整性的同时，能够提供完整的业务能力**。

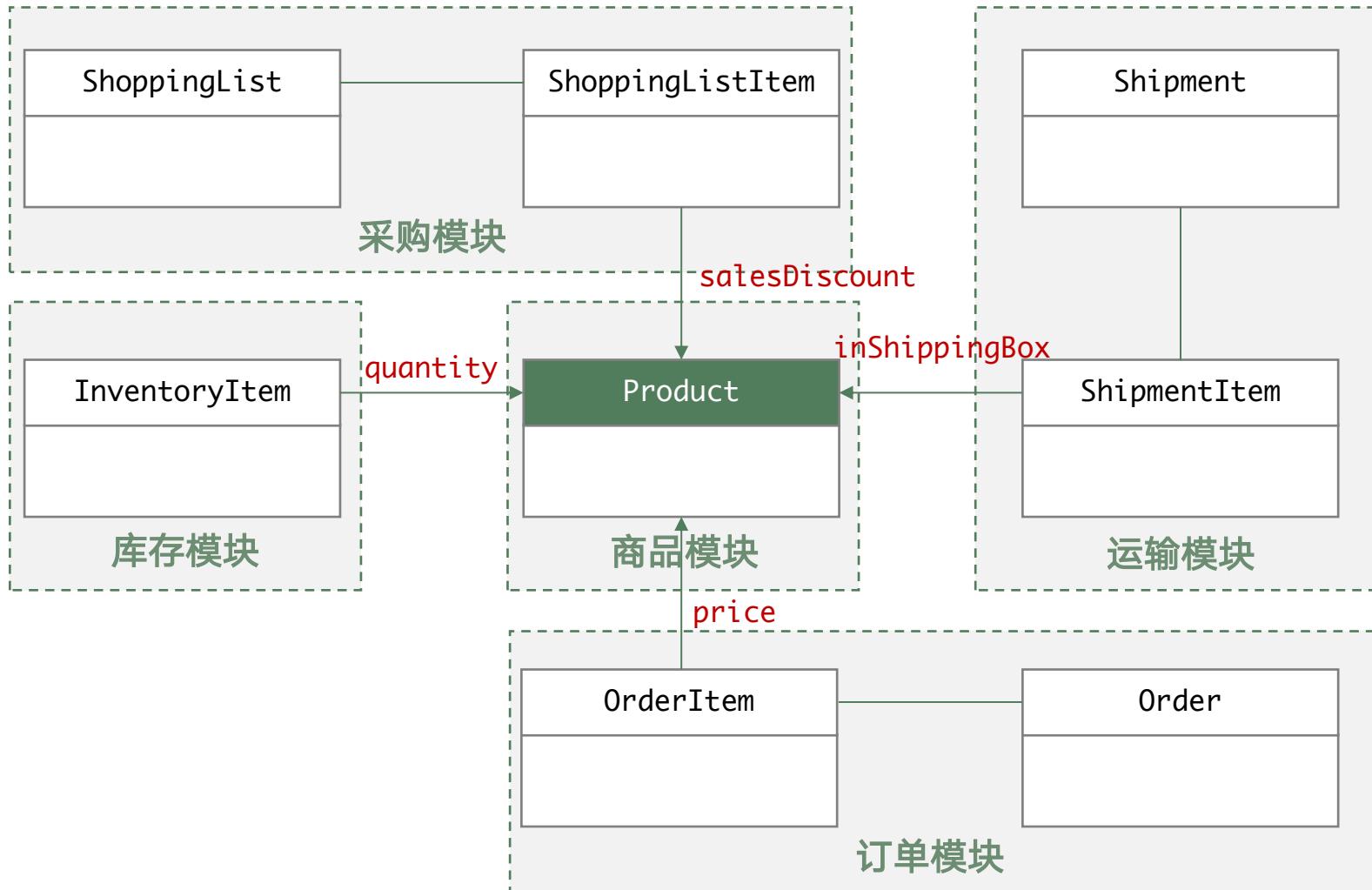
|| 案例：供应链中的商品



Product
<ul style="list-style-type: none">- productId- productType- productCategory- internalName- brandName- productName- description- longDescription- comments- introductionDate- salesDiscount- price- imgUrl- inventoryMessage- requireInventory- quantity- weight- productHeight- shippingHeight- shippingDepth- returnable- taxable- inShippingBox



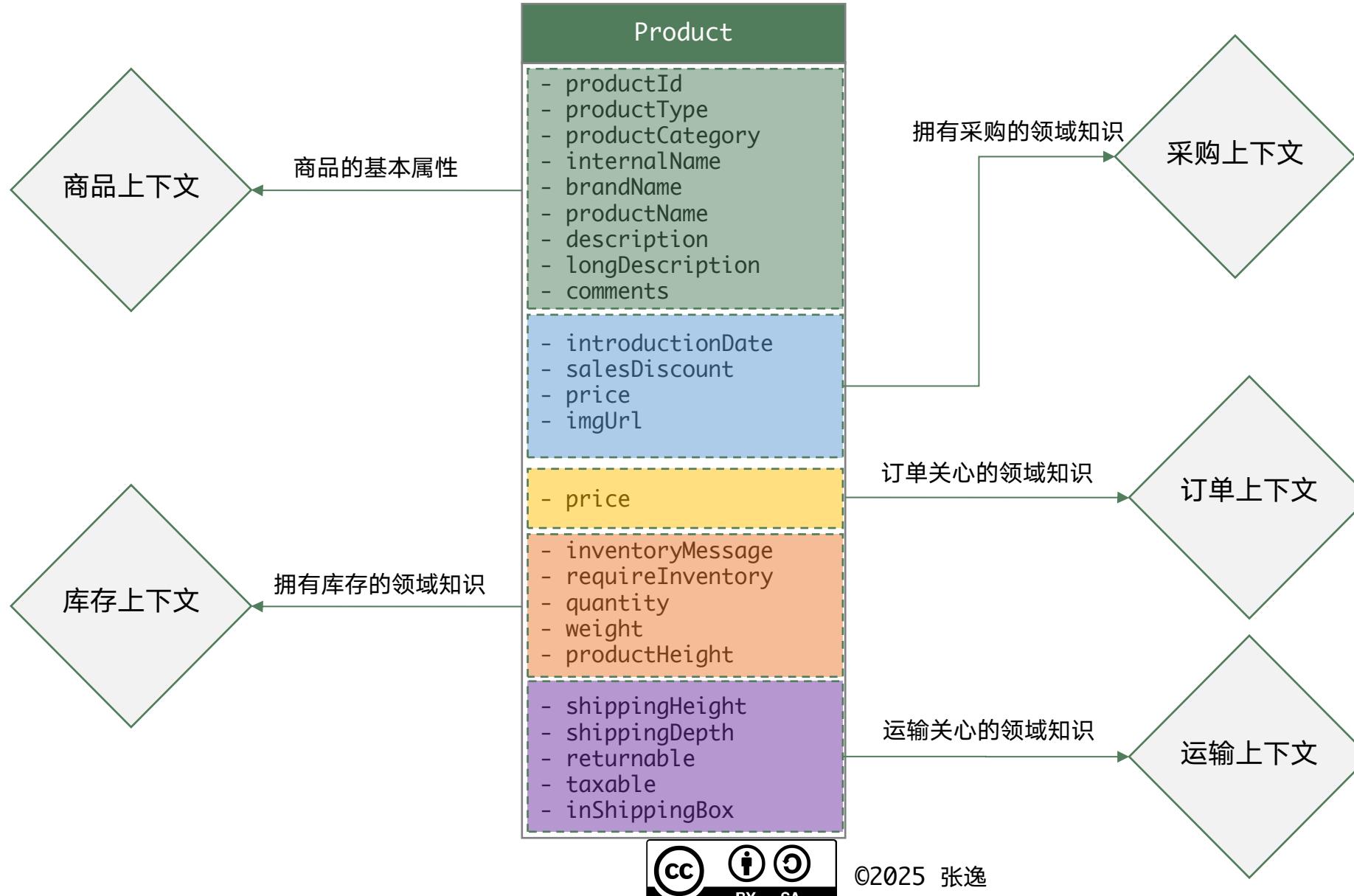
|| 引入模块封装概念



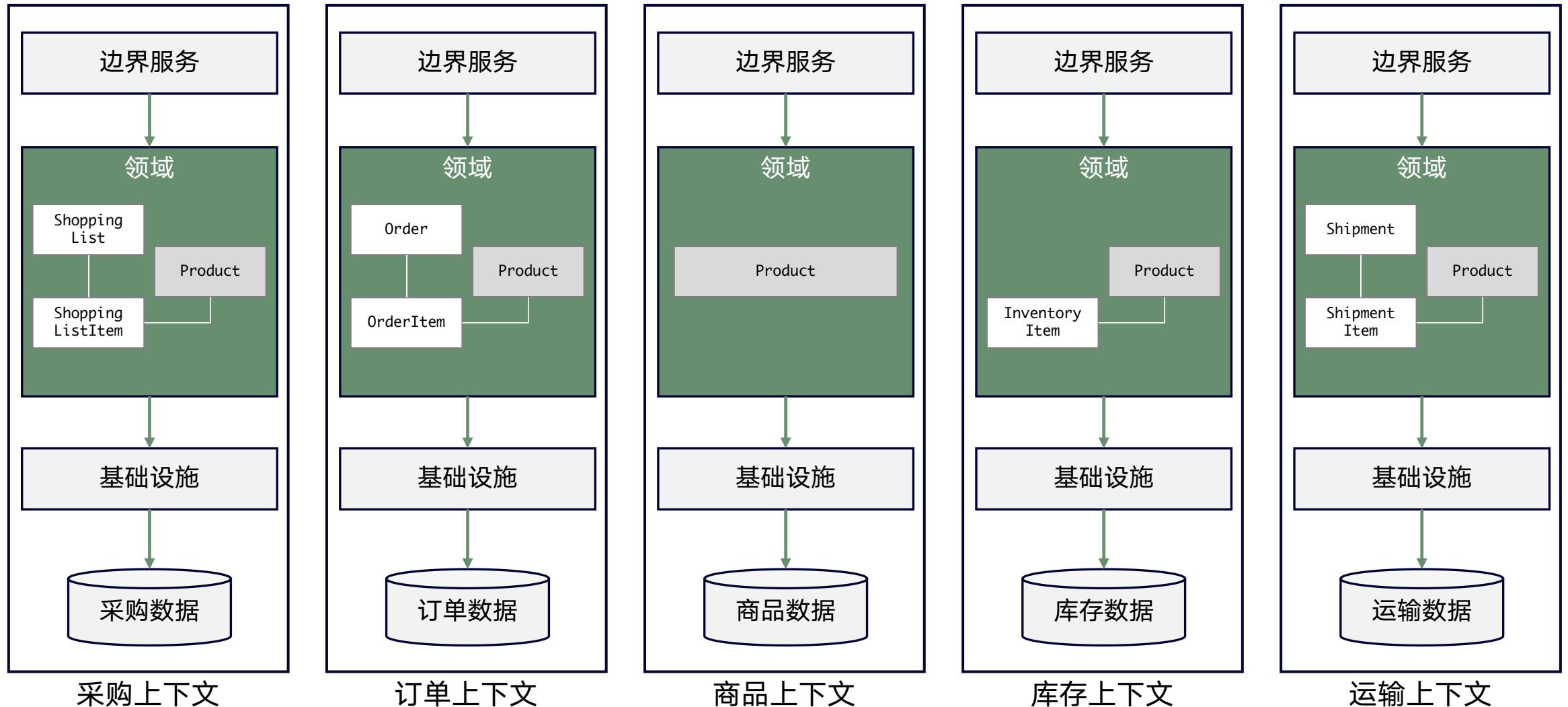
当我们引入模块对领域模型进行划分时，由于模块：

- 没有考虑知识语境的边界，仅仅按照业务特性对其进行类别的划分，将与商品有关的属性都分配给了 `Product` 领域对象，一旦其他模块需要调用商品的属性，就会产生模块之间的依赖。
- 没有进行业务能力的纵向切分，导致每个业务模块仅仅包含与其类别有关的领域知识，无法对外提供完整的业务能力。

|| 引入限界上下文

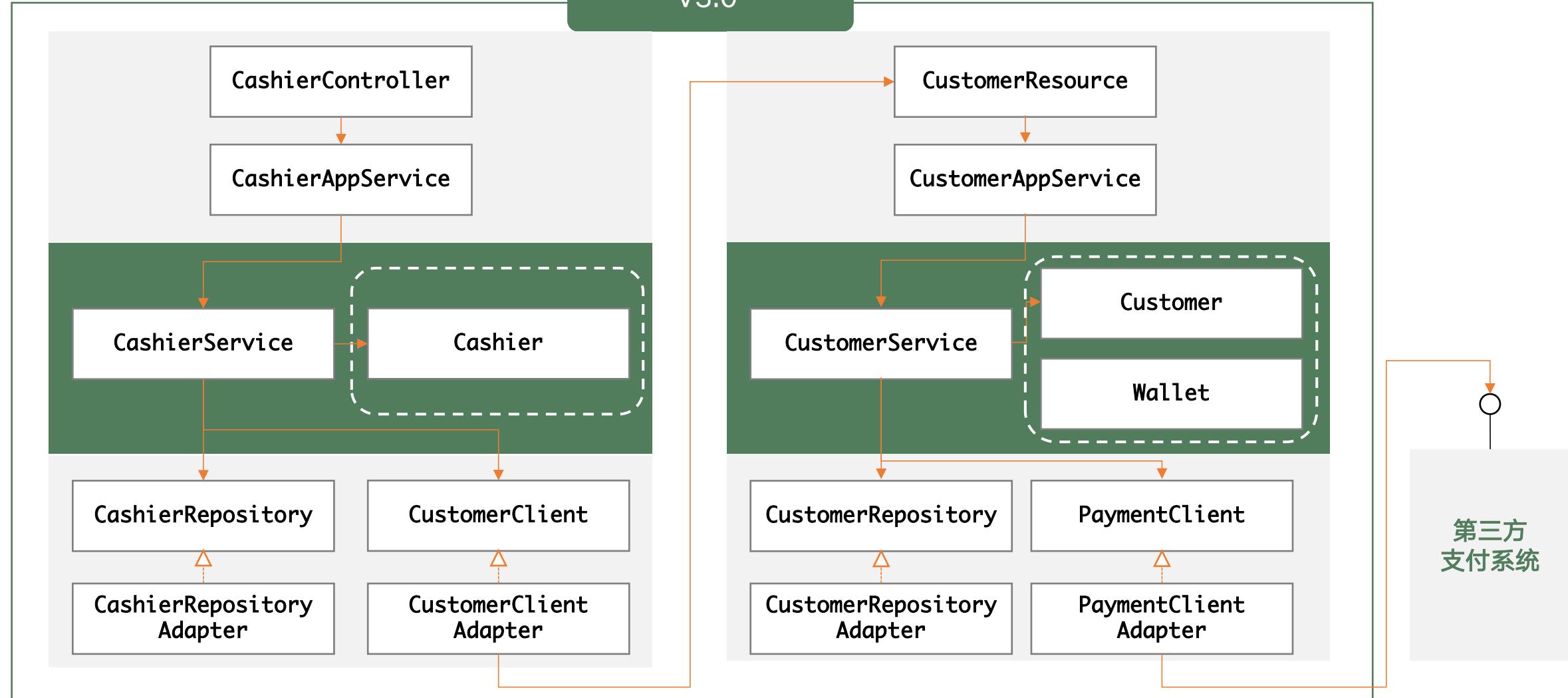


|| 引入限界上下文



|| 超市收银员：引入限界上下文

V3.0



02

DDD设计，从错误到正确

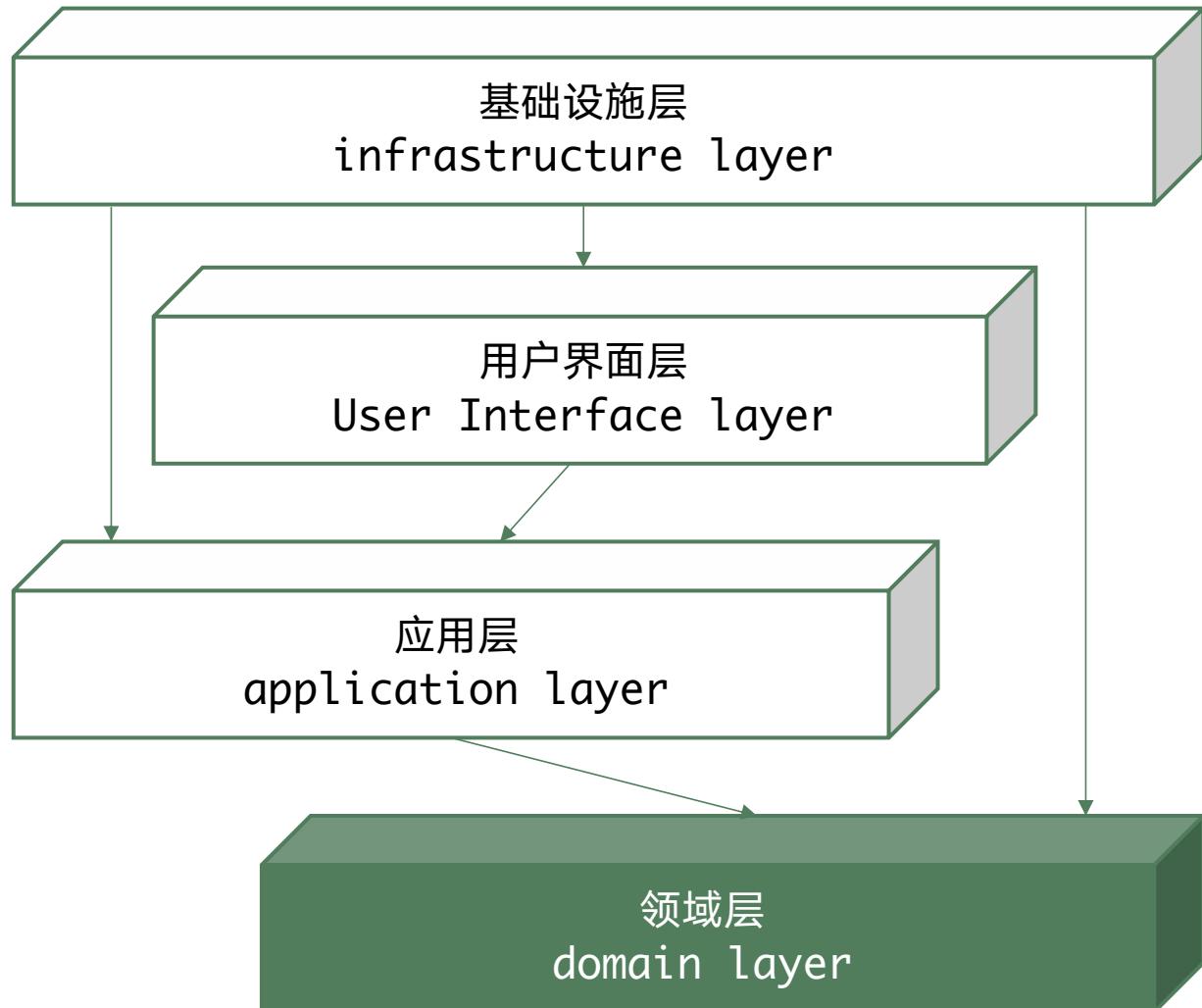
|| 案例：结算系统



在一个结算系统中，业务需求要求能够导入各种结算账单的Excel模板文件，然后通过具体的值填充这些报表模板，生成最终客户需要的Excel报表。过程为：

- 导入一个结算账单模板的Excel工作薄
 - Excel工作薄模板中对应的单元格中定义了一些变量值，系统需要从数据库中读取结算账单的信息，然后基于结算账单信息中的值去替换定义在模板中的这些变量
 - 导出替换了变量值的Excel工作薄
- 结算账单有多种，例如内部结算账单等。不同账单的模板并不相同，需要填充的变量值也不相同。

|| DDD四层架构



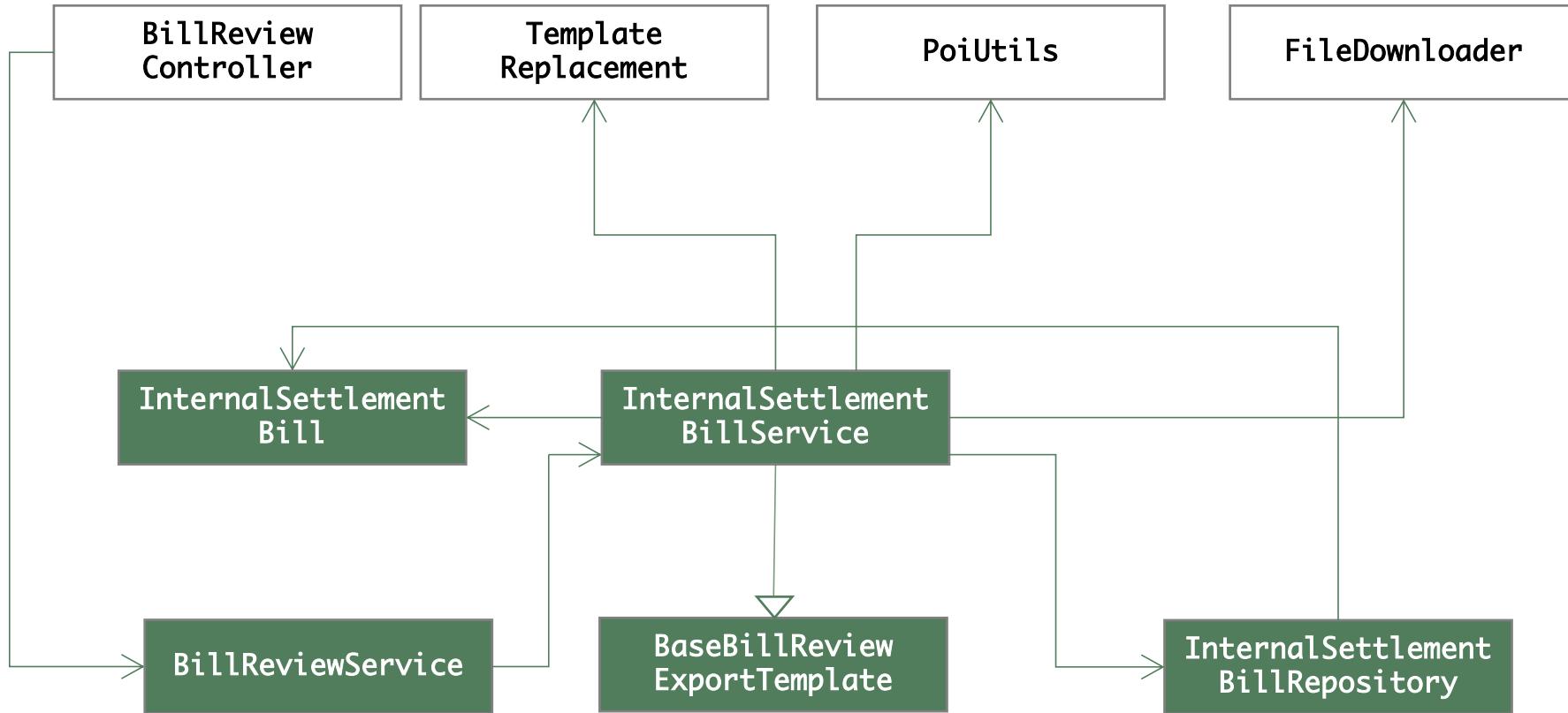
DDD四层架构遵循的原则：

- 依赖倒置原则：高层不应该依赖于低层，而应该依赖于抽象
- 稳定依赖原则：尽可能保证被依赖的层（模块/类）是稳定的
- 整洁架构思想：该思想将领域逻辑与技术实现分为内外两层，并认为越向内的领域层越稳定，越向外的技术实现越不稳定

DDD四层架构的定义中：

- 领域层：封装领域服务、聚合（包含了实体与值对象）以及资源库的抽象接口
- 应用层：定义应用服务
- 基础设施层：包括资源库的实现以及面向Web端的控制器

|| 修改前的设计模型



|| 领域层的定义

```
public class InternalSettlementBill {  
    private String billNumber;  
    private String newAndOldBillNumber;  
    private String flightIdentity;  
    private String flightNumber;  
    private String flightRoute;  
    private String scheduledDate;  
    private String passengerClass;  
    private List<Passenger> passengers;  
    private String serviceReason;  
    private List<CostDetail> costDetails;  
    private BigDecimal totalCost;  
}  
  
public interface InternalSettlementBillRepository {  
    InternalSettlementBill queryByBillNumber(String billNumber);  
}  
  
abstract class BaseBillReviewExportTemplate<T> {  
    public final List<TemplateReplacement> queryAndComposeTemplateReplacementsBy(String billNumber) {  
        T t = queryFilledDataBy(billNumber);  
        return composeTemplateReplacements(t);  
    }  
    protected abstract T queryFilledDataBy(String billNumber);  
    protected abstract List<TemplateReplacement> composeTemplateReplacements(T t);  
}
```



|| 领域层的定义（续）

```
@Service
public class InternalSettlementBillService extends BaseBillReviewExportTemplate<InternalSettlementBill> {
    @Resource
    private InternalSettlementBillRepository internalSettlementBillRepository;
    @Override
    protected InternalSettlementBill queryFilledDataBy(String billNumber) {
        return internalSettlementBillRepository.queryByBillNumber(billNumber);
    }
    @Override
    protected List<TemplateReplacement> composeTemplateReplacements(InternalSettlementBill t) {
        List<TemplateReplacement> templateReplacements = new ArrayList<>();
        templateReplacements.add(new TemplateReplacement(0, 0, t.getNewAndOldBillNumber()));
        templateReplacements.add(new TemplateReplacement(1, 0, t.getFlightIdentity()));
        templateReplacements.add(new TemplateReplacement(1, 2, t.getFlightRoute()));
        return templateReplacements;
    }
}

package settlement.infrastructure.file;

@Data
@AllArgsConstructor
public class TemplateReplacement {
    private int rowIndex;
    private int cellNum;
    private String replaceValue;
}
```

组装TemplateReplacement的职责应该优先分配给持有该领域信息的领域对象

TemplateReplacement 定义在基础设施层，而它事实上是一个领域概念
TemplateReplacement 并没有准确地体现它所对应的领域概念



|| 领域层的定义（续）

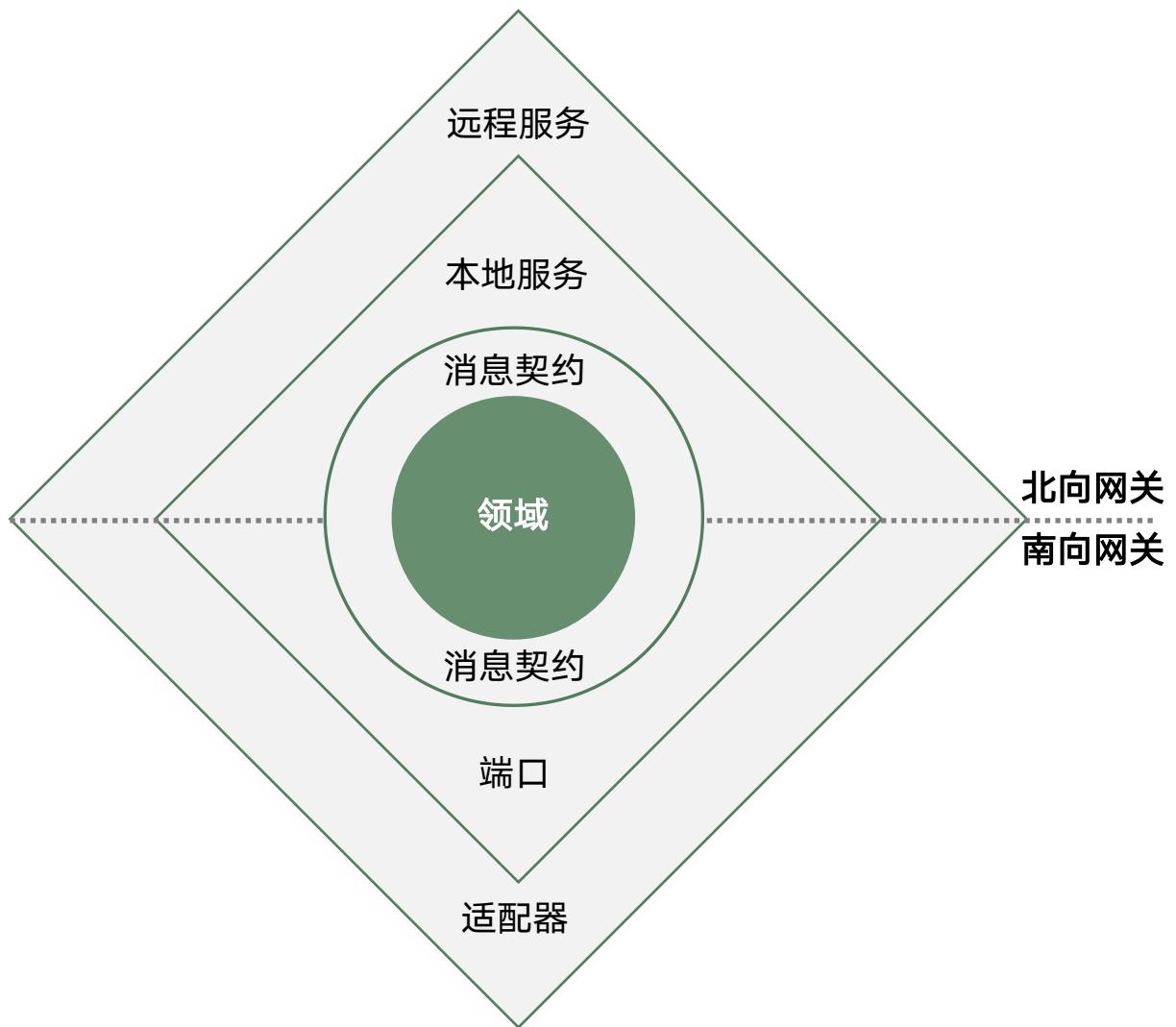
```
@Service
public class BillReviewService {
    @Value("${file-path.bill-templates-dir}")
    private String billTemplatesDirPath;
    @Resource
    private PoiUtils poiUtils;
    @Resource
    private FileDownloader fileDownloader;
    @Resource
    private InternalSettlementBillService internalSettlementBillService;
    @Resource
    private ExportBillReviewConfiguration configuration;
    public void exportBillReviewByTemplate(HttpServletRequest response, String billNumber, String templateName) {
        try {
            String className = fetchClassNameFromConfigBy(templateName);
            List<TemplateReplacement> replacements = templateReplacementsBy(billNumber, className);
            HSSFWorkbook workbook = poiUtils.getHssfWorkbook(billTemplatesDirPath + templateName);
            poiUtils.fillCells(workbook, DEFAULT_SHEET_INDEX, DEFAULT_REPLACE_PATTERN, replacements);
            fileDownloader.downloadHSSFFile(response, workbook, templateName);
        } catch (Exception e) {
            logger.error("Export bill review by template failed, templateName: {}", templateName);
            e.printStackTrace();
        }
    }
}
```

获取工作簿与下载文件的逻辑属于基础设施层

HSSFWorkbook不是领域对象，却被糅合到领域服务的实现中



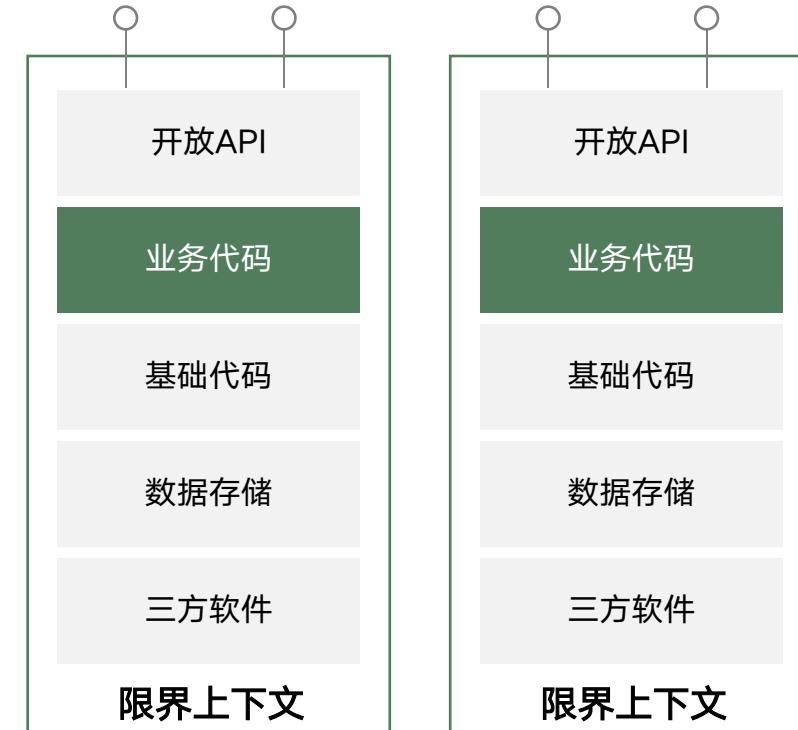
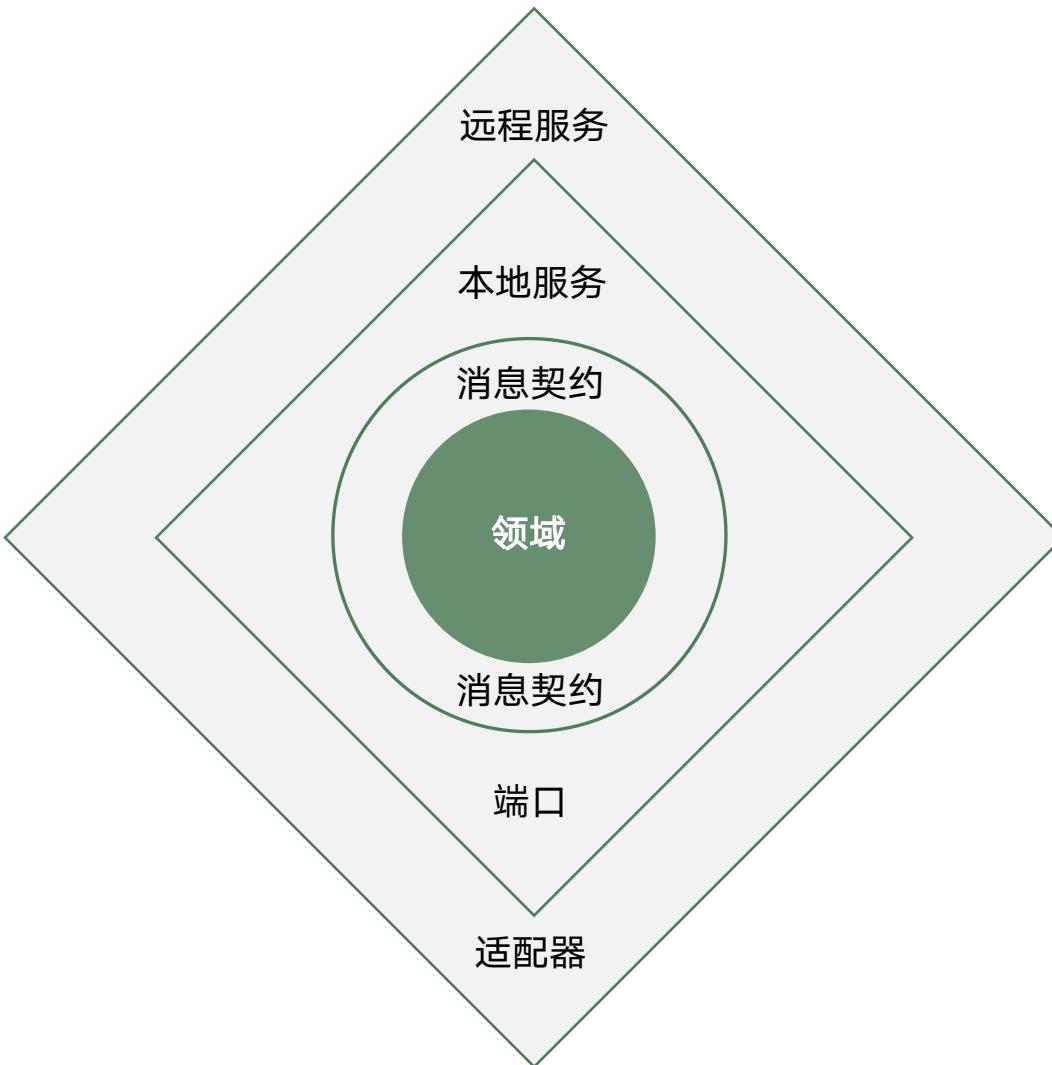
|| 菱形对称架构



为了保障限界上下文的自治性，限界上下文的内部架构应采用菱形对称架构。菱形对称架构由内部的领域层与外部的网关层构成：

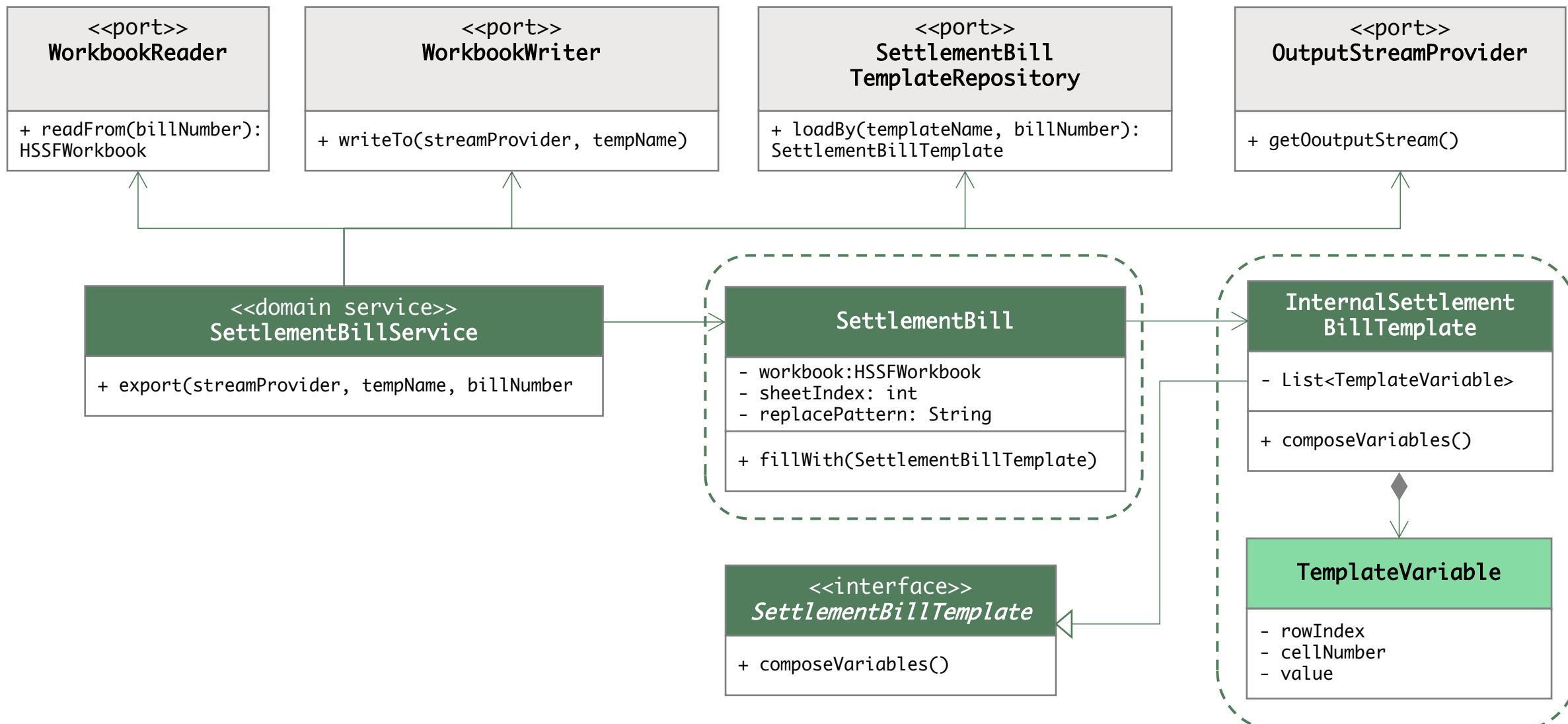
- **内外分离：** 内部领域层的领域逻辑与外部网关层的技术实现分开
- **南北对称：** 南向网关体现了**抽象**的设计原则，负责隔离内部领域层对外部资源的访问，北向网关体现了**封装**的设计原则，封装了内部的领域逻辑，定义远程服务、本地服务与消息契约，形成对外公开的服务契约

|| 限界上下文与菱形对称架构



每个限界上下文都是一个自治单元，且满足业务与技术的正交性。

|| 明确领域概念和领域层的边界



|| 结算账单模板的定义

```
public class InternalSettlementBillTemplate implements SettlementBillTemplate {  
    private String billNumber;  
    private String newAndOldBillNumber;  
    private String flightIdentity;  
    private String flightNumber;  
    private String flightRoute;  
    private String scheduledDate;  
    private String passengerClass;  
    private List<Passenger> passengers;  
    private String serviceReason;  
    private List<CostDetail> costDetails;  
    private BigDecimal totalCost;  
  
    public List<TemplateVariable> composeVariables() {  
        return Lists.newArrayList(  
            new TemplateVariable(0, 0, this.newAndOldBillNumber()),  
            new TemplateVariable(1, 0, this.flightIdentity()),  
            new TemplateVariable(1, 2, this.flightRoute())  
        );  
    }  
}  
  
public class TemplateVariable {  
    private int rowIndex;  
    private int cellNum;  
    private String replaceValue;  
}
```



|| 结算账单的定义

```
public class SettlementBill {  
    private HSSFWorkbook workbook;  
    private int sheetIndex;  
    private String replacePattern;  
    public SettlementBillTemplate(HSSFWorkbook workbook, int sheetIndex, String replacePattern) {  
        this.workbook = workbook;  
        this.sheetIndex = sheetIndex;  
        this.replacePattern = replacePattern;  
    }  
    public void fillWith(SettlementBillTemplate billTemplate) {  
        HSSFSheet sheet = workbook.getSheetAt(sheetIndex);  
        billTemplate.composeVariables().foreach( v -> {  
            HSSFCell cell = sheet.getRow(v.getRowIndex()).getCell(v.getCellNum());  
            String cellValue = cell.getStringCellValue();  
            String replaceValue = v.getReplaceValue();  
            if (replaceValue == null) {  
                logger.warn("{} -> {} 替换值为空, 未从数据库中查出相应字段值", cellValue, replaceValue);  
                continue;  
            }  
            logger.info("{} -> {}", cellValue, replaceValue);  
            if (cellValue.toLowerCase().contains(replacePattern)) {  
                cell.setCellValue(cellValue.replace(replacePattern, replaceValue));  
            } else {  
                cell.setCellValue(replaceValue);  
            }  
        });  
    }  
}
```



|| 领域服务的定义

```
public class SettlementBillService {  
    @Service  
    private WorkbookReader reader;  
    @Service  
    private WorkbookWriter writer;  
    @Repository  
    private SettlementBillTemplateRepository repository;  
  
    public void export(OutputStreamProvider streamProvider, String templateName, String billNumber) {  
        try {  
            SettlementBillTemplate billTemplate = repository.loadBy(templateName, billNumber);  
            SettlementBill bill = reader.readFrom(templateName);  
            bill.fillWith(billTemplate);  
            writer.writeTo(streamProvider, bill, templateName);  
        } catch (DownloadTemplateFileNotFoundException | OpenTemplateFileNotFoundException ex) {  
            throw new TemplateFileFailedException(ex.getMessage(), ex);  
        }  
    }  
}
```

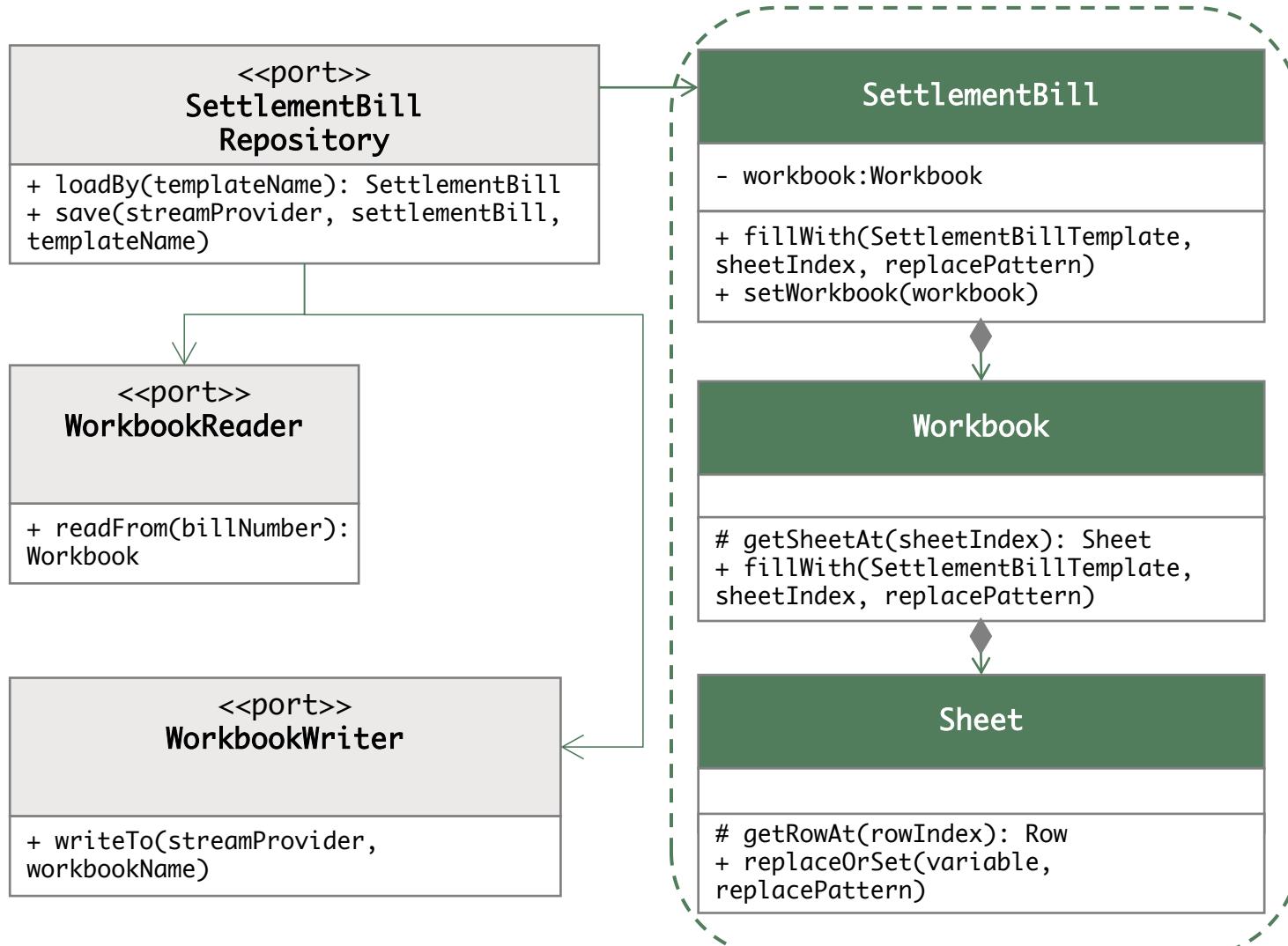
在领域服务SettlementBillService的定义中，除了HSSFWorkbook暴露了部分技术实现细节之外，整个export方法都与具体技术实现无关，保持领域逻辑的纯粹性。具体的技术实现全部转移到了菱形对称架构的南向网关，即代码中WorkbookReader、WorkbookWriter与SettlementBillTemplateRepository这些端口。



|| 抽象与封装

在目前的领域模型中，仍然存在以下两个不足：

- 结算账单SettlementBill的类定义中“渗入”了少量基础设施的内容，HSSFWorkbook与HSSFCell都是外部依赖框架的类；
- 对结算账单的访问没有得到合理的封装，使得领域服务需要通过WorkbookReader读取HSSFWorkbook，并在领域服务中构造SettlementBill领域对象。



|| 修改后的结算账单及其他领域对象

```
public class SettlementBill {  
    private Workbook workbook;  
  
    public void setWorkbook(Workbook workbook) {  
        this.workbook = workbook;  
    }  
  
    public void fillWith(SettlementBillTemplate billTemplate, int SheetIndex, String replacePattern) {  
        Sheet sheet = workbook.fillWith(billTemplate, sheetIndex, replacePattern);  
    }  
}  
  
public abstract class Workbook {  
    protected abstract Sheet getSheetAt(int sheetIndex);  
  
    public void fillWith(SettlementBillTemplate billTemplate, int sheetIndex, String replacePattern) {  
        Sheet sheet = getSheetAt(sheetIndex);  
        if (sheet != null) {  
            billTemplate.composeVariables().foreach(v -> {  
                sheet.replaceOrSet(v, replacePattern);  
            });  
        }  
    }  
}
```

注意观察本页以及下一页定义的Workbook等领域对象，它们都提供了满足自身要求的replaceOrSet()方法，且在这些方法的实现中，无需再调用get或set访问器，符合面向对象的迪米特法则。



|| 修改后的结算账单及其他领域对象

```
public abstract class Sheet {  
    protected abstract Row getRowAt(int rowIndex);  
    public void replaceOrSet(Variable variable, String replacePattern) {  
        Row row = getRowAt(variable.getRowIndex());  
        row.replaceOrSet(variable, replacePattern);  
    }  
}  
public abstract class Row {  
    protected abstract Cell getCellAt(int cellIndex);  
    public void replaceOrSet(Variable variable, String replacePattern) {  
        Cell cell = getCellAt(variable.getCellNum());  
        cell.replaceOrSet(variable.getReplaceValue(), replacePattern);  
    }  
}  
public class Cell {  
    private String cellValue;  
    public void replaceOrSet(String replaceValue, String replacePattern) {  
        if (replaceValue == null) {  
            logger.warn("{} -> {} 替换值为空，未从数据库中查出相应字段值", cellValue, replaceValue);  
            return;  
        }  
        logger.info("{} -> {}", cellValue, replaceValue);  
        if (cellValue.toLowerCase().contains(replacePattern)) {  
            this.cellValue = cellValue.replace(replacePattern, replaceValue);  
        } else {  
            this.cellValue = replaceValue;  
        }  
    }  
}
```



|| 修改后的领域服务

```
public class SettlementBillService {  
    @Service  
    private SettlementBillRepository billRepo;  
    @Repository  
    private SettlementBillTemplateRepository templateRepo;  
  
    public void export(OutputServiceProvider streamProvider, String templateName, String billNumber) {  
        try {  
            SettlementBillTemplate billTemplate = templateRepo.loadBy(templateName, billNumber);  
            SettlementBill bill = billRepo.loadBy(templateName);  
            bill.fillWith(billTemplate);  
            billRepo.save(bill, templateName);  
        } catch (BillTemplateException ex) {  
            throw new TemplateFileFailedException(ex.getMessage(), ex);  
        }  
    }  
}
```

SettlementBillService领域服务通过资源库对象获得领域对象，由于资源库得到了合理的封装，领域服务的业务逻辑也变得更加清晰。



©2025 张逸

|| 修改后的资源库

```
public interface SettlementBillRepository {  
    SettlementBill loadBy(String templateName);  
    void save(SettlementBill bill, String templateName);  
}  
public class SettlementBillRepositoryHSSFAdapter implements SettlementBillRepository {  
    private WorkbookReader reader;  
    private WorkbookWriter writer;  
    private PoiUtils poiUtils;  
    public SettlementBill loadBy(String templateName) {  
        SettlementBill bill = reader.readFrom(templateName);  
        HSSFWorkbook hssfWorkbook = poiUtils.getHssfWorkbook(billTemplatesDirPath + templateName);  
        bill.setWorkbook(new HSSFWorkbookAdapter(hssfWorkbook));  
        return bill;  
    }  
    public void save(OutputStreamProvider streamProvider, SettlementBill bill, String templateName) {  
        writer.writeTo(streamProvider, bill, templateName);  
    }  
}
```

Repository封装了对SettlementBill的访问（加载与保存）逻辑，隐藏了通过WorkbookReader和WorkbookWriter对模板文件读写的技术细节。从对外暴露的接口来看，它遵循了资源库管理聚合生命周期的约定，虽然实现上是对文件的操作，但看起来和访问数据库没有任何差别。



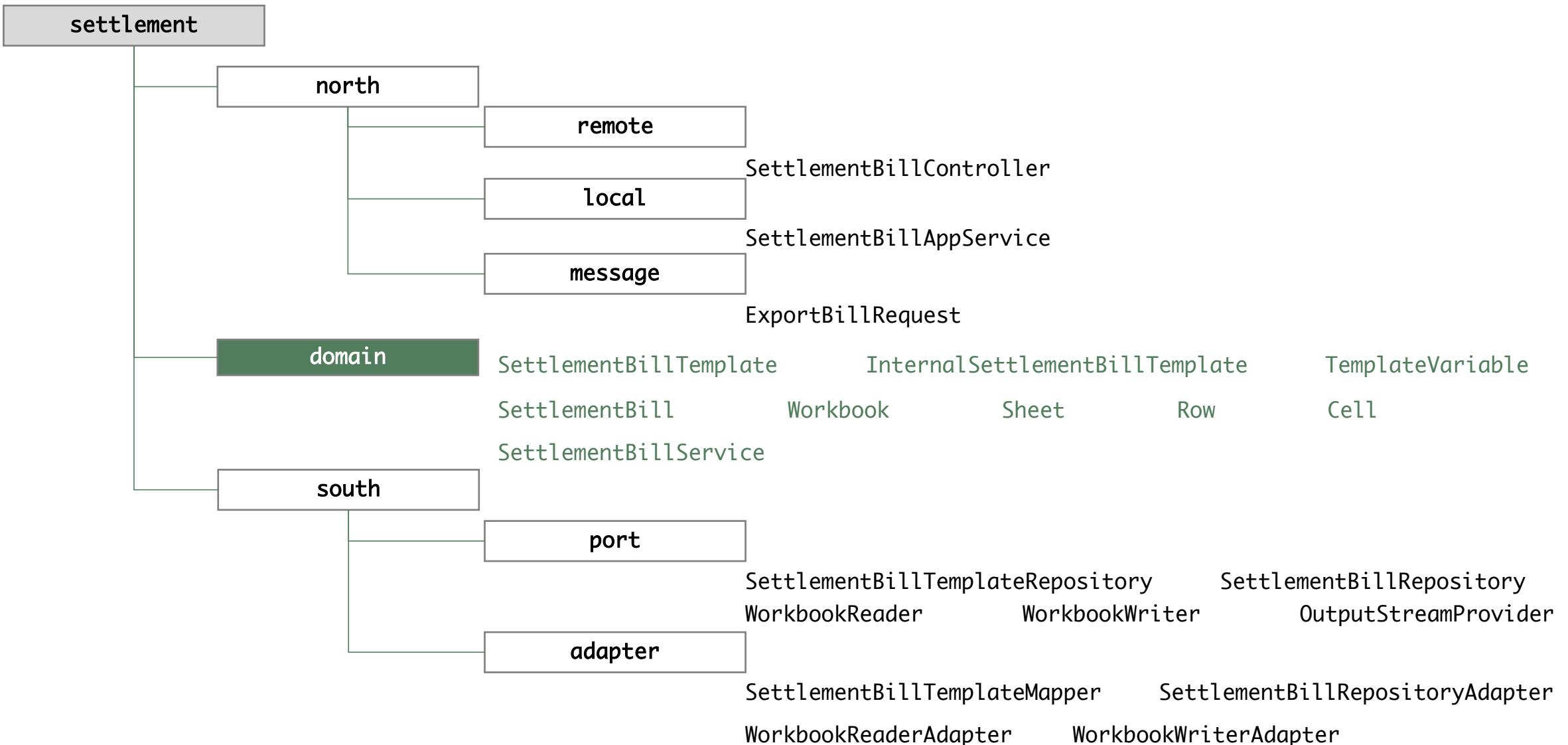
|| 领域模型的适配

```
public class HSSFWorkbookAdapter implements Workbook {  
    private HSSFWorkbook hssfWorkbook;  
    public HSSFWorkbookAdapter(HSSFWorkbook hssfWorkbook) {  
        this.hssfWorkbook = hssfWorkbook;  
    }  
    @Override  
    protected Sheet getSheetAt(int sheetIndex) {  
        HSSFSheet sheet = workbook.getSheetAt(sheetIndex);  
        return new HSSFSheetAdapter(sheet);  
    }  
}  
  
public class HSSFSheetAdapter implements Sheet {  
    private HSSFSheet hssfSheet;  
    public HSSFSheetAdapter(HSSFSheet hssfSheet) {  
        this.hssfSheet = hssfSheet;  
    }  
    @Override  
    protected Row getRowAt(int rowIndex) {  
        HSSFRow hssfRow = hssfSheet.getRow(rowIndex);  
        return new HSSFRowAdapter(hssfRow);  
    }  
}
```

通过PoiUtils获得的HSSFWorkbook等对象，并非Workbook、Sheet、Row、Cell等领域模型类型，因此，需要引入对应的适配器类，将它们转换为对应领域模型类的子类。



|| 代码模型





3 领域建模的重要性

01

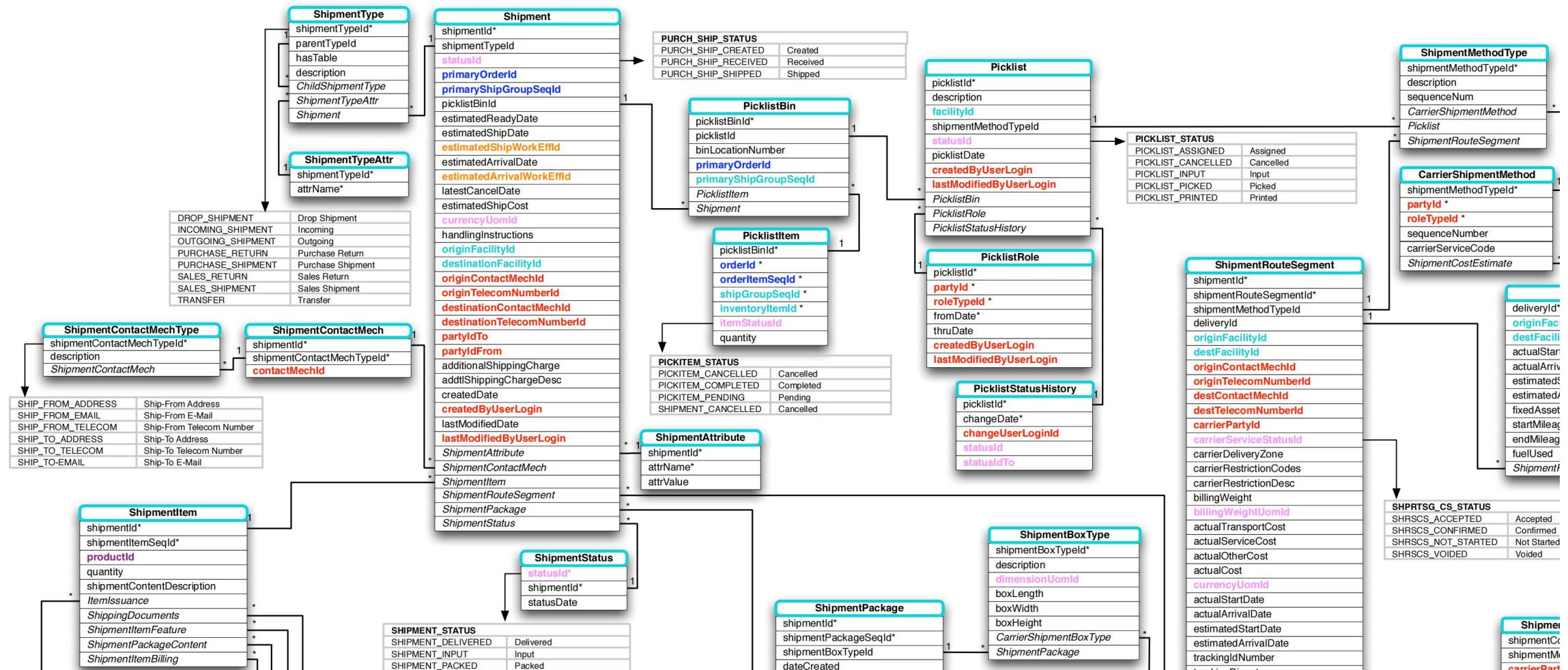
领域建模 vs 数据建模

|| 什么是建模



软件建模是一种解决问题的抽象方法，通过对关键概念、关键行为、关键特征的识别，然后从设计角度对识别出来的概念、行为与特征进行归类、定义和分解，并以开发人员和领域专家都可以理解的语言与形式表示出来，实现复杂问题的简单化，为获得最终的实现代码提供有效输入。

|| 数据驱动的建模方式

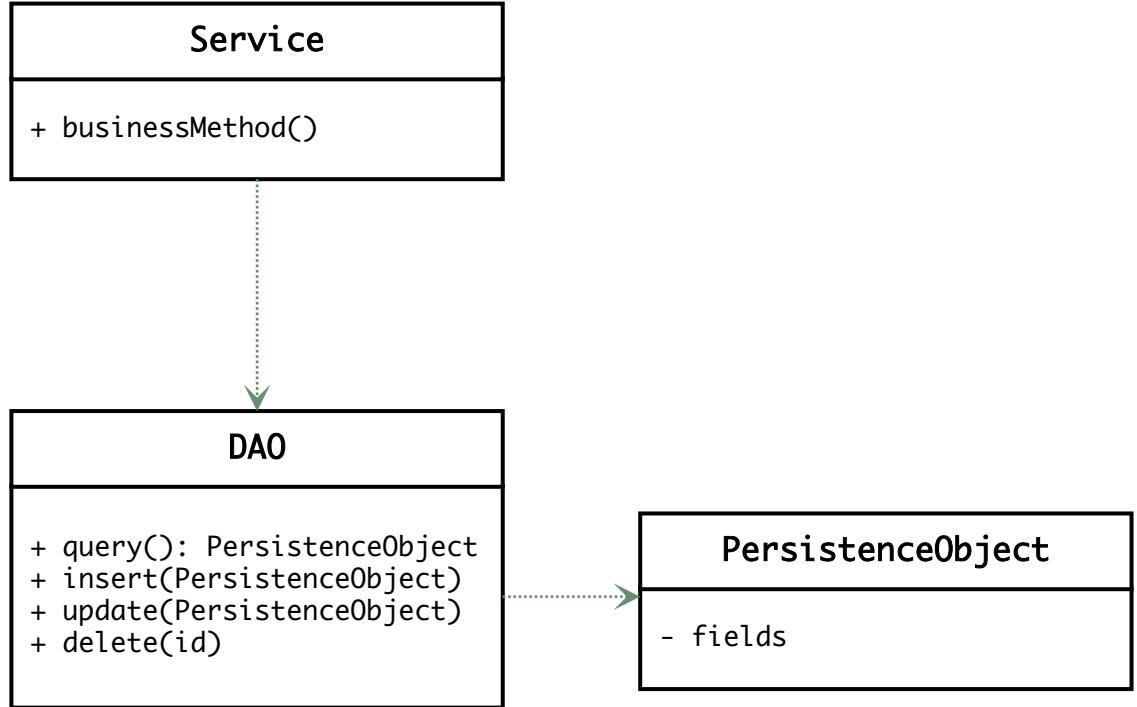


|| 数据驱动导致程式设计

```
public class ShipmentServices {  
    // 共361行代码  
    public static Map<String, Object> calcShipmentCostEstimate(DispatchContext dctx, Map<String, ? extends Object> context) {  
        // 准备数据  
        String productStoreShipMethId = (String) context.get("productStoreShipMethId");  
        ... 以下共26行代码  
  
        // 获得ShipmentCostEstimate(s)  
        Map<String, String> estFields = UtilMisc.toMap("productStoreId", productStoreId, "shipmentMethodTypeId", shipmentMethodTypeId,  
            "carrierPartyId", carrierPartyId, "carrierRoleTypeId", carrierRoleTypeId);  
        EntityCondition estFieldsCond = EntityCondition.makeCondition(estFields, EntityOperator.AND);  
        ... 以下共30行代码  
        // 获得PostalAddress  
        // 获得可能的估算值  
        // 获得ShippableItem的size列表和feature字典  
        // 根据相关数据计算优先级  
  
        // 计算  
        BigDecimal itemFlatAmount = shippableQuantity.multiply(orderItemFlat);  
        BigDecimal orderPercentage = shippableTotal.multiply(orderPercent.movePointLeft(2));  
    }  
}
```

|| 数据驱动导致过程式设计

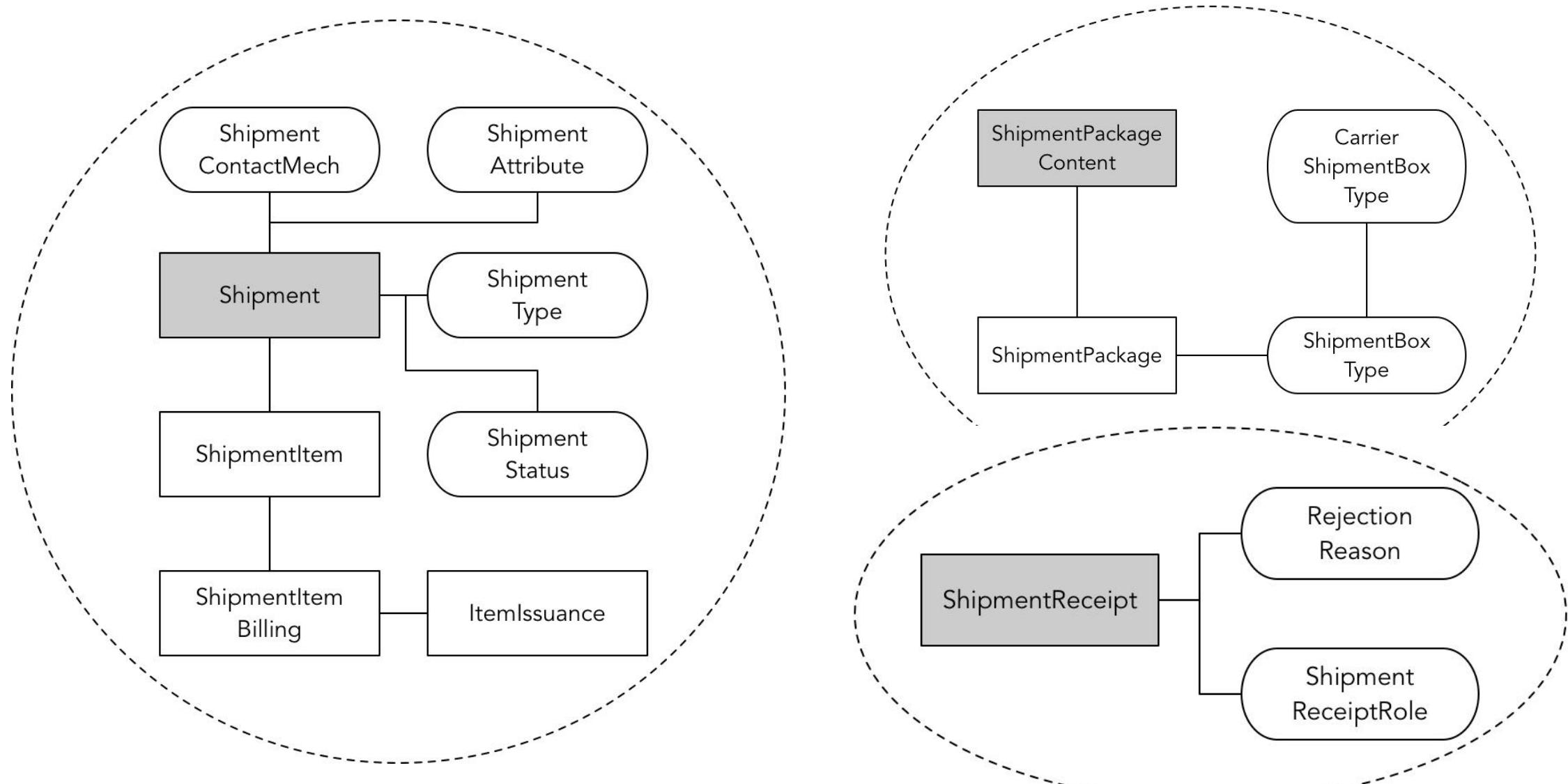
```
public class ShipmentServices {  
    // 共361行代码  
    public static Map<String, Object>  
calcShipmentCostEstimate(DispatchContext dctx, Map<String, ?  
extends Object> context) {  
    // 准备数据  
    String productStoreShipMethId = (String)  
context.get("productStoreShipMethId");  
    ... 以下共26行代码  
  
    // 获得ShipmentCostEstimate(s)  
    Map<String, String> estFields =  
UtilMisc.toMap("productStoreId", productstoreId,  
"shipmentMethodTypeId", shipmentMethodTypeId,  
    "carrierPartyId", carrierPartyId,  
"carrierRoleTypeId", carrierRoleTypeId);  
    EntityCondition estFieldsCond =  
EntityCondition.makeCondition(estFields, EntityOperator.AND);  
    ... 以下共30行代码  
}
```



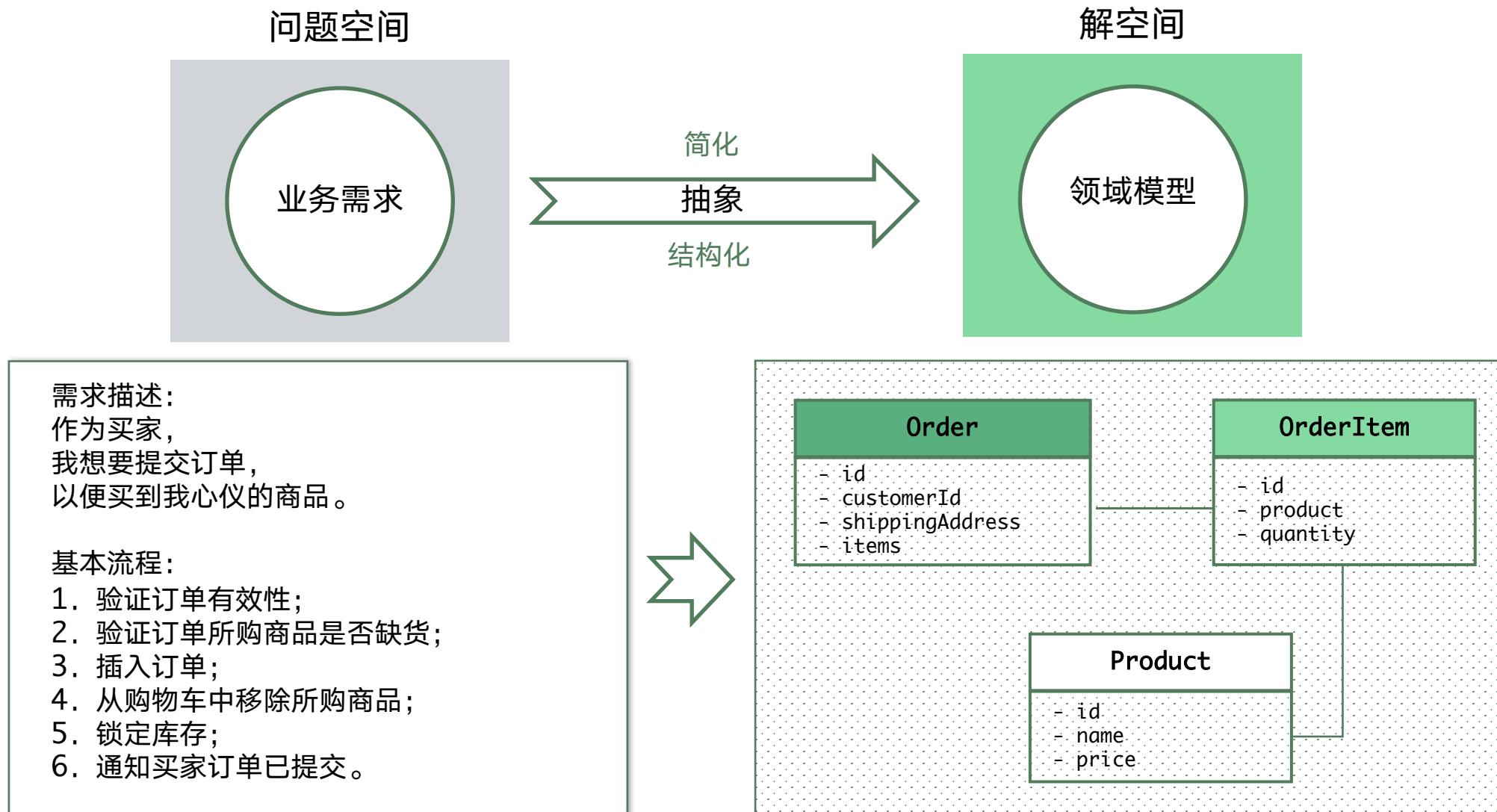
所有业务逻辑都可以通过三种角色完成：

- 服务：采用事务脚本，所有领域逻辑都集中在服务
- DAO：负责与DB通信，实现数据对象的增删改查
- PO：数据表映射的贫血领域模型

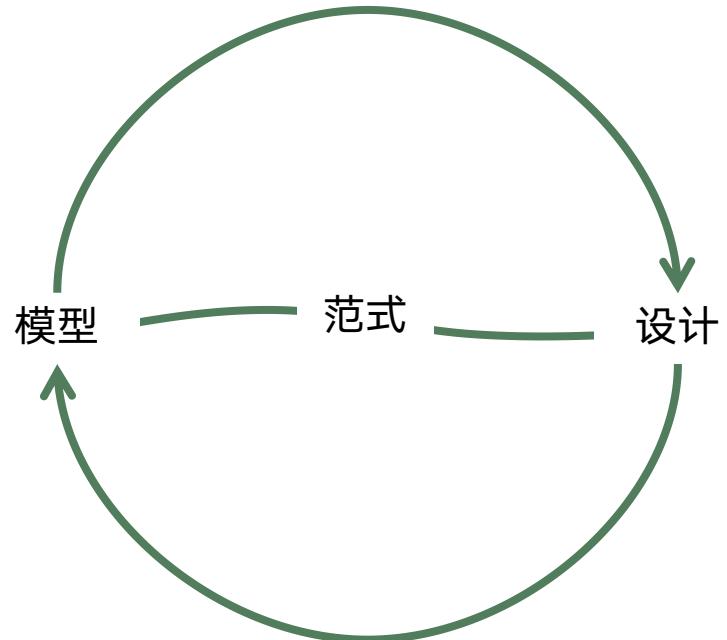
|| 领域驱动的建模方式



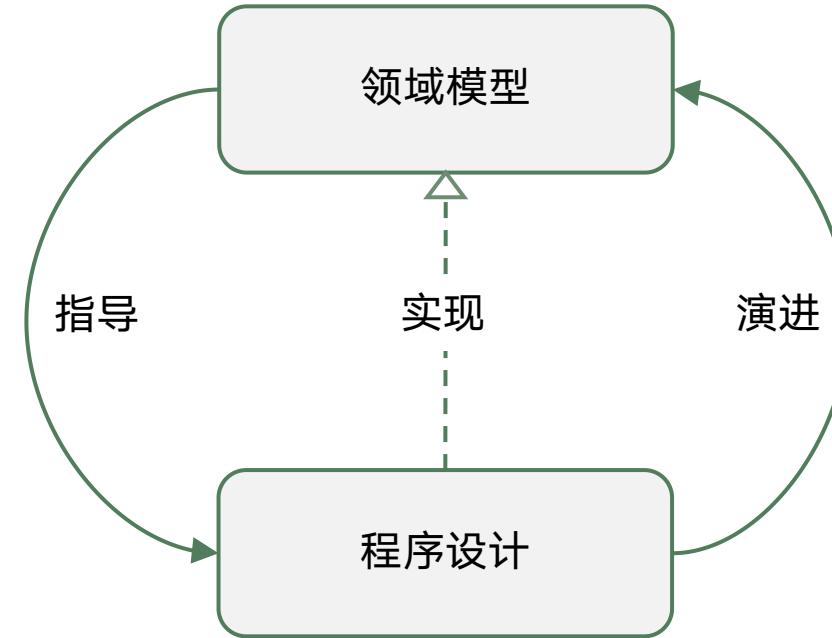
|| 领域驱动设计的核心是领域建模



|| 领域模型与程序设计的关系

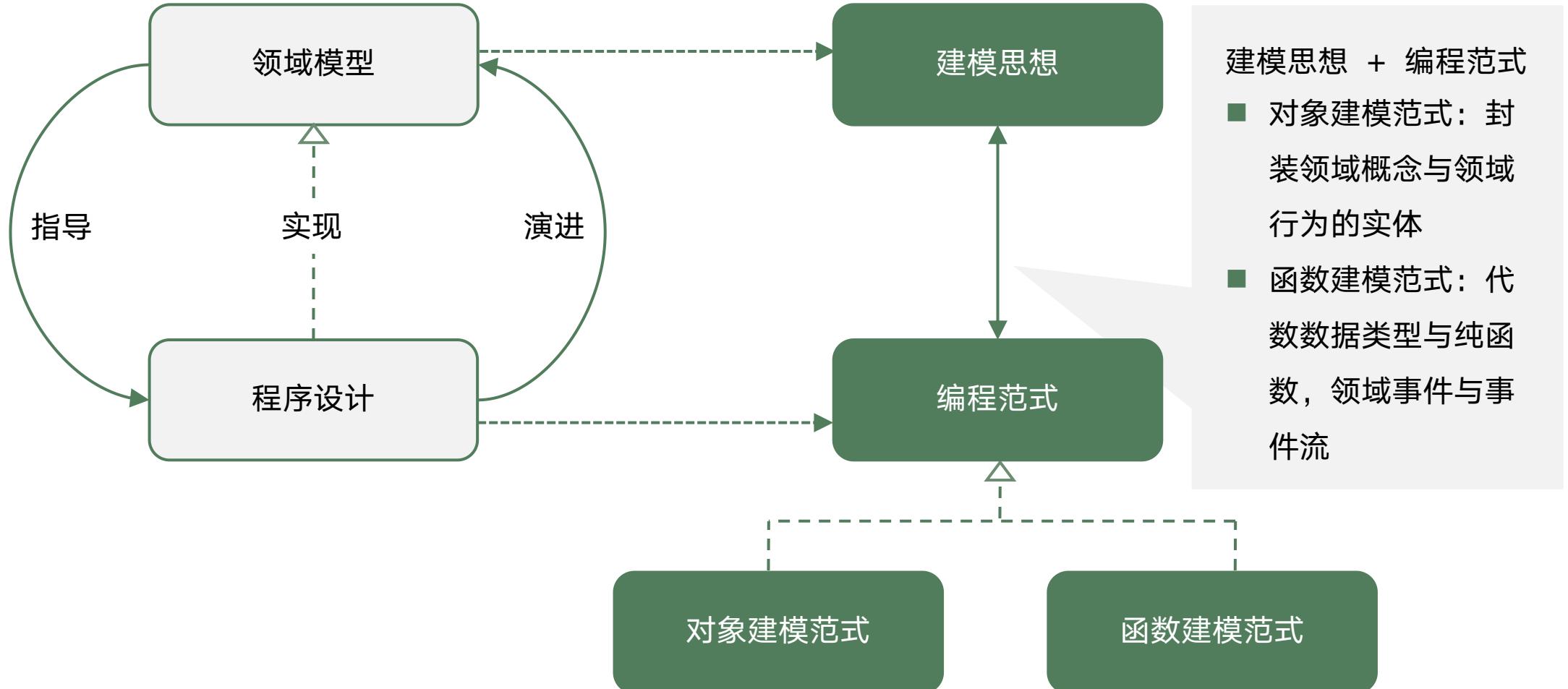


编程语言根据建模范式为模型构造提供实现方式，这要求编程语言能够支持建模者运用代码表达模型概念。



软件系统各个部分的设计应该忠实地反映领域模型，以便体现出这二者之间的明确对应关系。我们应该反复检查并修改模型，以便软件可以更加自然地实现模型。

|| 领域模型与程序设计的关系



02

通过事件风暴开展领域建模

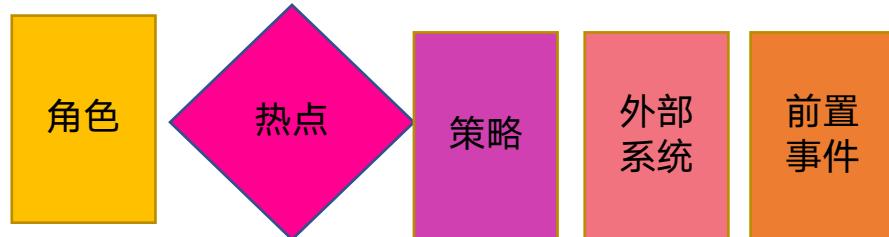
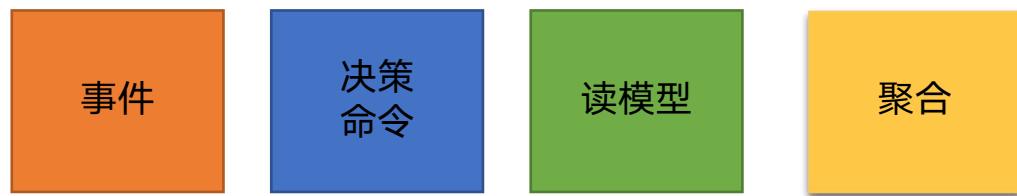
|| 事件风暴

EVENT STORMING

The smartest approach to collaboration
beyond silo boundaries

TO SAVE TIME, LET'S JUST ASSUME
I'M NEVER WRONG

|| 事件风暴图例



|| 识别事件

事件

- 领域事件是过去发生的与业务有关的事实
- 领域事件具有时间点的特征，所有事件连接起来会形成明显的时间轴
- 领域事件是管理者和运营者重点关心的内容，若缺少该事件，会对管理与运营产生影响
- 领域事件会导致目标对象状态的变化



©2025 张逸

|| 标记热点



在识别事件的过程中，若有以下情况，可以为事件标记热点（HotSpot）：

- 暂不考虑的事件流分支
- 出现分歧和争执的事件
- 需要强调的事件或事件对应的领域逻辑

|| 事件的命名



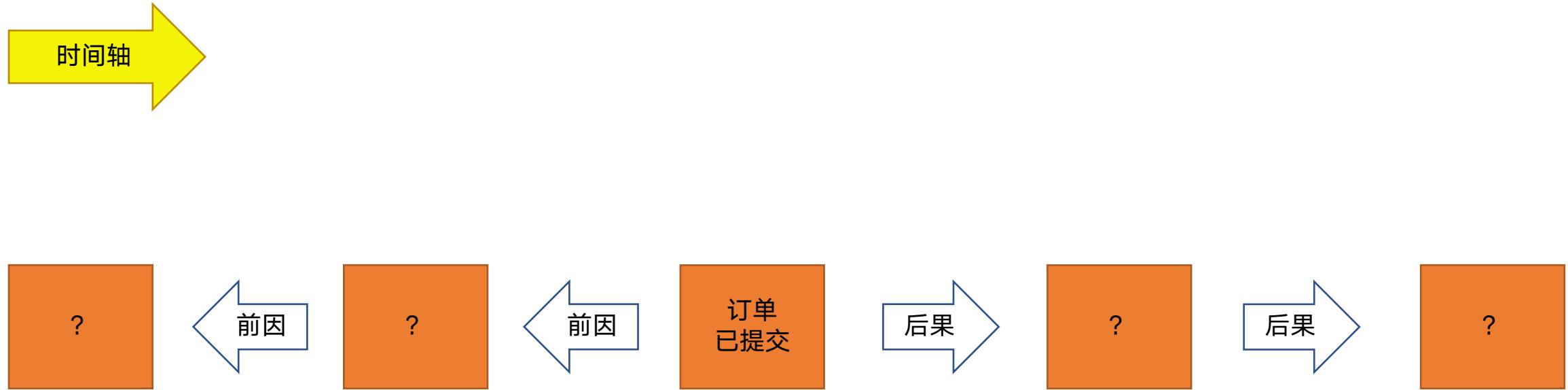
- 领域事件的组成：名称+动词过去时态，如OrderCreated
命名时，需要充分沟通和交流，提炼出统一语言

|| 识别第一个事件



©2025 张逸

|| 事件的驱动力



|| 识别参与者

事件一共有四种参与者：

- 角色（Role）：触发事件的人
- 策略（Policy）：触发事件的规则
- 外部系统（External System）
- 事件（Event）：即当前事件的前置事件

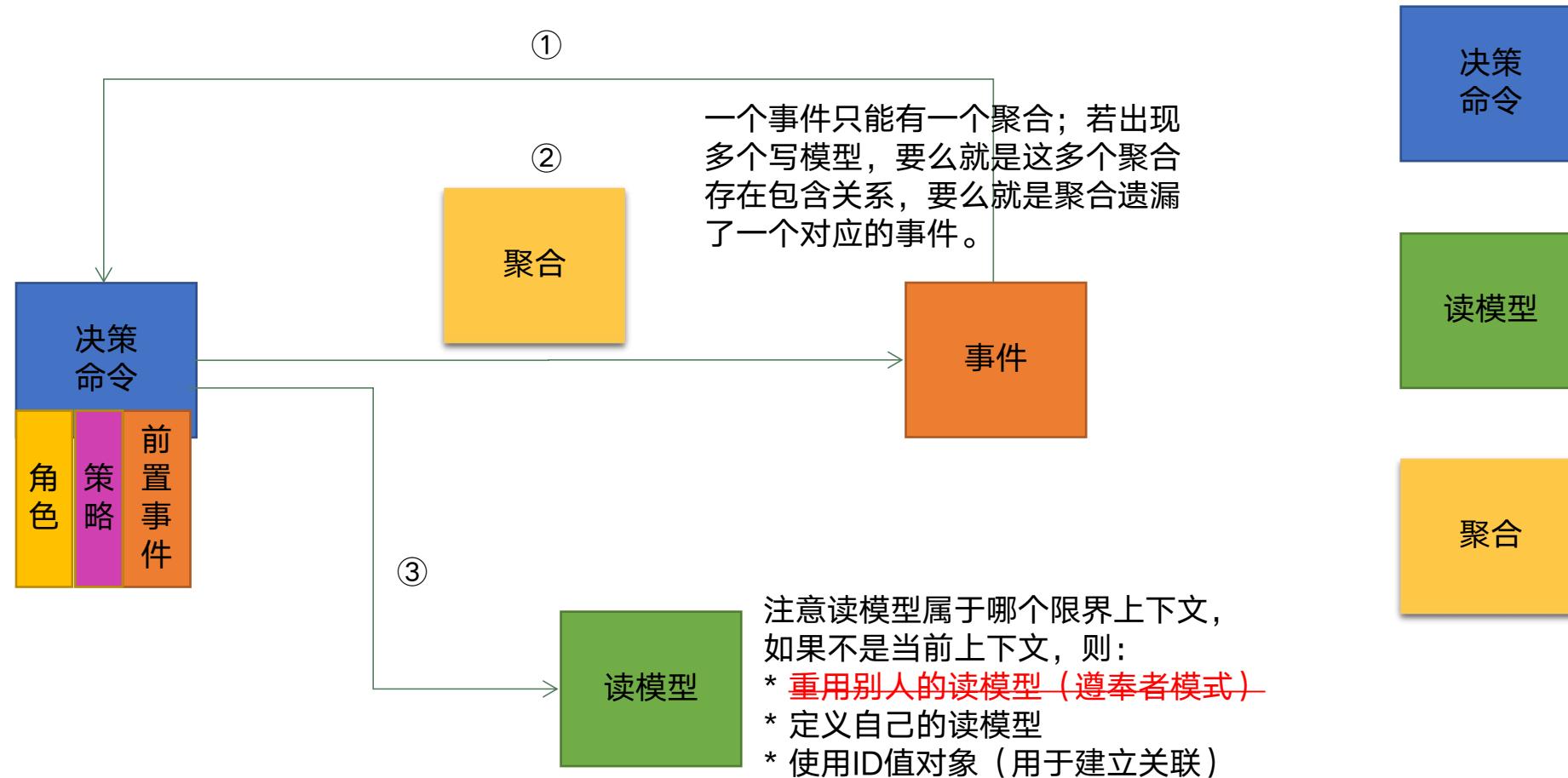


注意：策略是规则，但规则不是策略。可以认为策略是规则+定时器的组合。



©2025 张逸

|| 领域分析建模



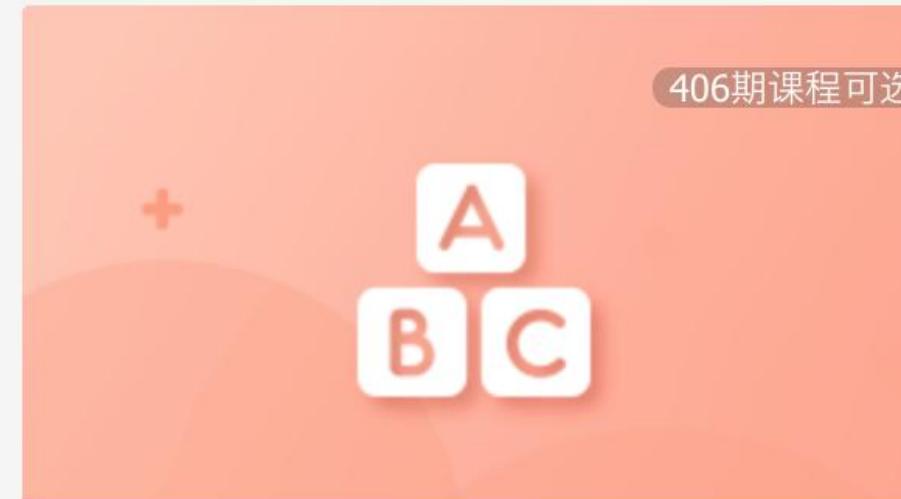
|| 案例演练：为学课堂



数 六年级大数学直播班专区

10000+人已报名

¥1200 起



英 六年级英语直播班专区

10000+人已报名

¥1160 起

|| 案例演练：为学课堂

年级 中班 大班 一年级 二年级 三年级 四年级 五年级 六年级 初一 初二 初三 高一 高二 高三

类别 同步课程 专题课程 素养课程

学科 全部 语文 数学 英语 编程

难度 全部 人教引航班(2星) 北师引航班(2星) 勤学班(3星) 启航班(3星) 致慧班(4星) 探航班(5星)

学期 全部 暑 秋 秋(下)

|| 案例演练：为学课堂

数 【2019-暑】五年级升六年级数学勤学班全国版（六年制）

⌚ 三期：8月11日-8月27日每天9:00-11:10
难度：★★★★☆



授课
赵晓思



辅导
曹晟莹

余1名额

¥ 1500

【暑假课报名中】还剩3天
共15讲

数 【2019-暑】五年级升六年级数学勤学班全国版（六年制）

⌚ 三期：8月11日-8月27日每天18:20-20:30
难度：★★★★☆



授课
屠鸣伟



辅导
晁溪

余12名额

¥ 1500

【暑假课报名中】还剩3天
共15讲

|| 案例演练：为学课堂

【暑假课报名中】还剩3天12时59分49秒

数 三期：8月11日-8月27日每天9:00-11:10

【2019-暑】五年级升六年级数学勤学班全国版（六年制）

直播授课 随时退款 随时看回放 自主调班 及时答疑 课堂巩固一对一批改 学习诊断 学习报告 纸质讲义 学期家长会
设备说明 教具邮寄

难度：★★★★☆

课程有效期：2019-04-04至2022-08-31

授课老师



赵晓思
三期：8月11日-8月27
日每天9:00-11:10

辅导老师



曹晟莹
全程跟踪辅导

剩余1个名额

课程介绍

课程大纲(15)

体验直播课

价 格 ￥1500

活 动
组合优惠 赠送课程
精美讲义邮寄

声 明
若退课时教辅已签收，需扣除教辅工本费用。

直播课不支持Mac os10.11、Windows xp及以下电脑操作系统，不支持Android 5.0以下版本、iOS10.0以下APP版本、不支持iPad mini1、iPad 3、iPhone 5s及以前机型，更多设备信息请查看设备说明

报名多个课程时请尽量避免课程时间段相重合。

咨询

诊断后报名

|| 案例演练：为学课堂



诊断名称: 【2019年暑秋】六年级数学全国版原体系入学诊断

诊断时长: 30分钟

分数线:
启航班(3星) 勤学班(3星) 致慧班(4星)
0分 30分 70分 100分

A horizontal score scale with numerical markers at 0, 30, 70, and 100. Above each marker is a vertical line segment followed by the class name and star rating. The segments for 30 and 70 are longer than the one for 0, and the segment for 100 is the longest.

开始诊断

常见问题

|| 案例演练：为学课堂

试卷总分: 100

试卷题数: 7

考生: 15023157626@talwx.com

诊断时间: 30分钟

剩余时间: 29:49

完成进度: 0%

交卷

在线答题

填空题 (共7题, 共 100 分)

1、计算: $\frac{3}{5} \times 17 + 0.6 \times 44 + 39 \div \frac{5}{3} = \underline{\hspace{2cm}}$.

①

请输入内容

|| 案例演练：为学课堂

57
分

恭喜, 15023157626@talwx.com (2344267) 同学已通过诊断

立即报名

根据诊断结果, 这些课程更适合您:

查看更多课程

数 【2019-暑】五年级升六年级数学勤学班全国版 (六年制)

⌚ 三期: 8月11日-8月27日每天13:00-15:10

难度: ★ ★ ★ ☆ ☆



授课
隋松泉



辅导
薛梅

余11名额

数 【2019-暑】五年级升六年级数学勤学班全国版 (六年制)

⌚ 三期: 8月11日-8月27日每天9:00-11:10

难度: ★ ★ ★ ☆ ☆



授课
赵晓思



辅导
曹晟莹

余1名额

数 【2019-暑】五年级升六年级数学勤学班全国版 (六年制)

⌚ 三期: 8月11日-8月27日每天18:20-20:30

难度: ★ ★ ★ ☆ ☆



授课
屠鸣伟



辅导
晁溪

余12名额

|| 案例演练：为学课堂

【暑假课报名中】还剩3天12时37分25秒

数 三期：**8月11日-8月27日每天9:00-11:10**

【2019-暑】五年级升六年级数学勤学班全国版（六年制）

直播授课 随时退款 随时看回放 自主调班 及时答疑 课堂巩固一对一批改 学习诊断 学习报告 纸质讲义 学期家长会
设备说明 教具邮寄

难度：★★★☆☆

课程有效期：2019-04-04至2022-08-31

授课老师



赵晓思

三期：8月11日-8月27
日每天9:00-11:10

辅导老师



曹晟莹

全程跟踪辅导

剩余1个名额

课程介绍

课程大纲(15)

体验直播课

价 格 **¥ 1500**

活 动
**组合优惠 赠送课程
精美讲义邮寄**

声 明
若退课时教辅已签收，需扣除教辅工本费用。

直播课不支持Mac os10.11、Windows xp及以下电脑操作系统，不支持Android 5.0以下版本、iOS10.0以下APP版本、不支持iPad mini1、iPad 3、iPhone 5s及以前机型，更多设备信息请查看设备说明

报名多个课程时请尽量避免课程时间段相重合。

咨询

立即报名

|| 案例演练：为学课堂

结算 ×

三期：8月11日-8月27日每天9:00-11:10
【2019-暑】五年级升六年级数学勤学班全国版（六年制）
¥ 1500

一起报名，立享更多优惠

三期：8月11日-8月27日每天9:00-11:10
【2019-暑】五年级升六年级数学勤学班全国版（六年制）
¥ 1500

9月7日-9月21日每周六15:30-17:40
【秋（上）】六年级数学勤学班全国版（六年制）
¥ 285 ¥ 300

价格：¥ 1785

需支付：**1785** 结算

|| 案例演练：为学课堂

请选择收货地址 (建议您填写公司/单位地址，以便收货)

收货人 请准确填写真实姓名 X

所在地区 省份 城市 区县

详细地址 请填写详细路名及门牌号

手机号码 用于接收发货通知短信和送货前通知

保存收货人信息

商品清单

	商品详情	商品金额
--	------	------

数 【2019-暑】五年级升六年级数学勤学班全国版（六年制）

三期：8月11日-8月27日每天9:00-

¥1500

授课：赵晓思 辅导：曹晨莹老师

11:10

注：如您需开具发票，支付成功后，请在订单管理-我的发票处进行申请。

若商品申请退费时教辅已签收，需根据签收的教辅按分类扣除工本费。

提交订单

|| 案例演练：为学课堂



订单提交成功，请您尽快付款！ 订单号：2019080411264177617567

应付金额：¥ 1500

订单将为你保留 14:46，
请抓紧时间支付

微信支付

支付宝支付

京东支付

随机立减

网上银行支付



微信支付

使用微信APP扫码支付，请使用5.0以上版本微信



|| 案例演练：为学课堂

订单详情	总计	状态及操作
2019-08-04 11:26:41 订单号：2019080411264177617567 请在15分钟内完成此订单支付，否则订单会自动取消。		
数 【2019-暑】五年级升六年级数学勤学班全国版（六年制） 三期：8月11日-8月27日每天9:00-11:10 授课：赵晓思	¥1500	¥1500 待支付 查看详情
福利 教材邮寄 ×9		立即支付 取消订单

|| 案例演练：为学课堂

全部订单 1 已支付 0 未完成 0 待支付 1 已取消 0

订单详情	总计	状态及操作
2019-08-04 11:26:41 订单号: 2019080411264177617567 数 【2019-暑】五年级升六年级数学勤学班全国版（六年制） 三期：8月11日-8月27日每天9:00-11:10 授课：赵晓思	¥1500 ¥1500	已取消 查看详情 重新报名
福利 教材邮寄 ×9		

|| 案例演练：为学课堂

全部订单 1 已支付 1 未完成 0 待支付 0 已取消 0

订单详情	总计	状态及操作
2019-07-10 23:32:09 订单号: 2019071023320921600115		
<p>语 【2019-暑】四年级升五年级大语文直播班 三期: 7月30日-8月9日每天18:20-20:30 授课: 于晴1</p> <p>调换课</p>	¥1000 ¥1000	已支付 查看详情 查看物流 申请退费
福利 教材邮寄 已签收 ×11		

|| 案例演练：为学课堂

订单跟踪 ×

包裹1

货物运品：贴纸1 【2019-暑】五年级大语文直播班 *1, 课堂巩固1 【2019-暑】五年级大语文直播班 *1, 基础知识手册1 【2019-暑】五年级大语文直播班 *1, 讲义1 【2019-暑】五年级大语文直播班 *1, 小高通用随材 *1, 笔记本1通用笔记本 *1, 产品使用说明书 *1, 给学员的一封信 *1, 给家长的一封信 *1, 学习日历 *1, 读书笔记本 *1

快递公司：中通速递

快递单号：

配送地址：四川省

● 【成都市】已签收, 签收人凭取货码签收, 如有疑问请电联: **18030680360 / 18030680360**, 您的快递已经妥投。风里来雨里去, 只为客官您满意。上有老下有小, 赏个好评好不好? 【请在评价快递员处帮忙点亮五颗星星哦~】
2019-07-22 16:23:55

【成都市】快件已送达【快递超市的凯德风尚代理点】, 如有问题请电联 (18030680360 / 18030680360), 感谢使用中通快递, 期待再次为您服务!
2019-07-19 16:26:02

【成都市】【成都文家场外光华】的路上稍等 (18030680360) 正在第1次派件, 请保持电话畅通, 并耐心等待 (95720为中通快递员外呼专属号码, 请放心接听)
2019-07-19 15:11:36

【成都市】快件已经到达【成都文家场外光华】
2019-07-19 15:11:36

|| 案例演练：为学课堂

我的发票

网校课程开发票的有效期是365天，需要发票的童鞋抓紧哦！

发票详情	订单金额	状态
2019-07-10 23:32:50 订单号: 2019071023320921600115		
语 【2019-暑】四年级升五年级大语文直播班	¥ 1000.00	未开发票 申请开票

|| 识别事件

方案A

引入“订单已创建”事件



方案B

引入“报名”或“报名单”领域概念

诊断
已开始

试卷
已生成

试卷
已提交

诊断
已完成

课程
已推荐

课程
已加入报
名单

订单
已提交

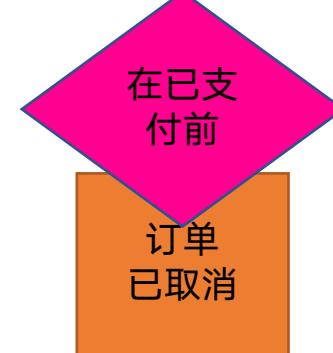
课程名额
已锁定

往来账
已生成

支付
已完成

订单
已支付

报名
已完成



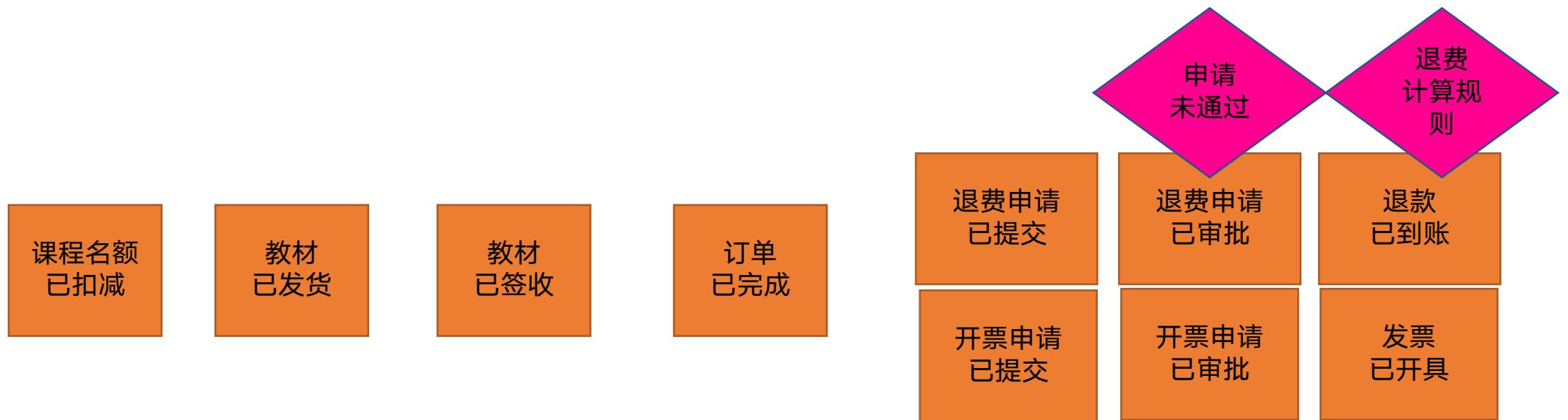
课程名额
已释放

课程
已移出
报名单

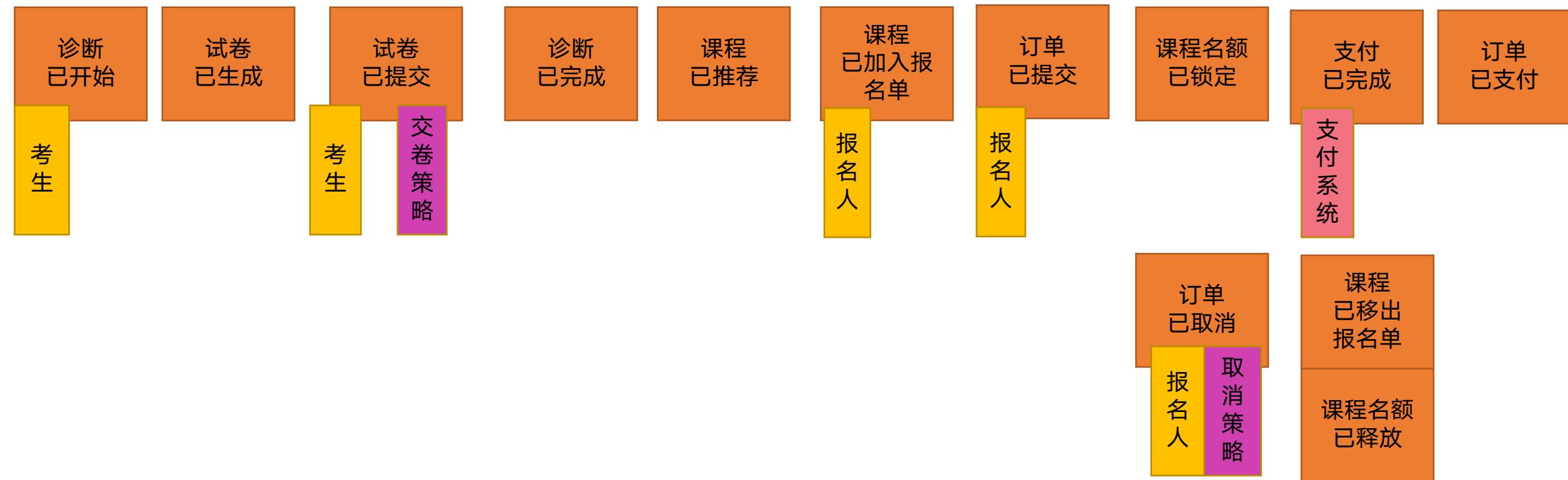
考虑支付失败



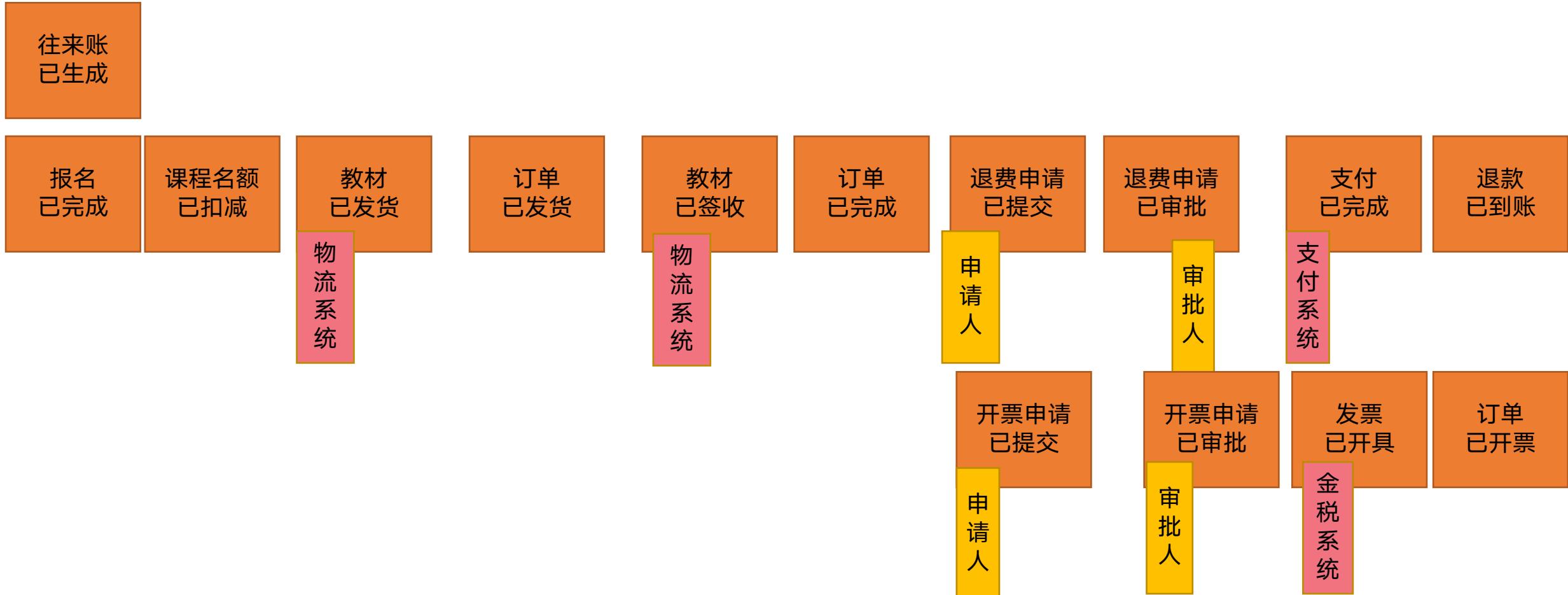
|| 识别事件



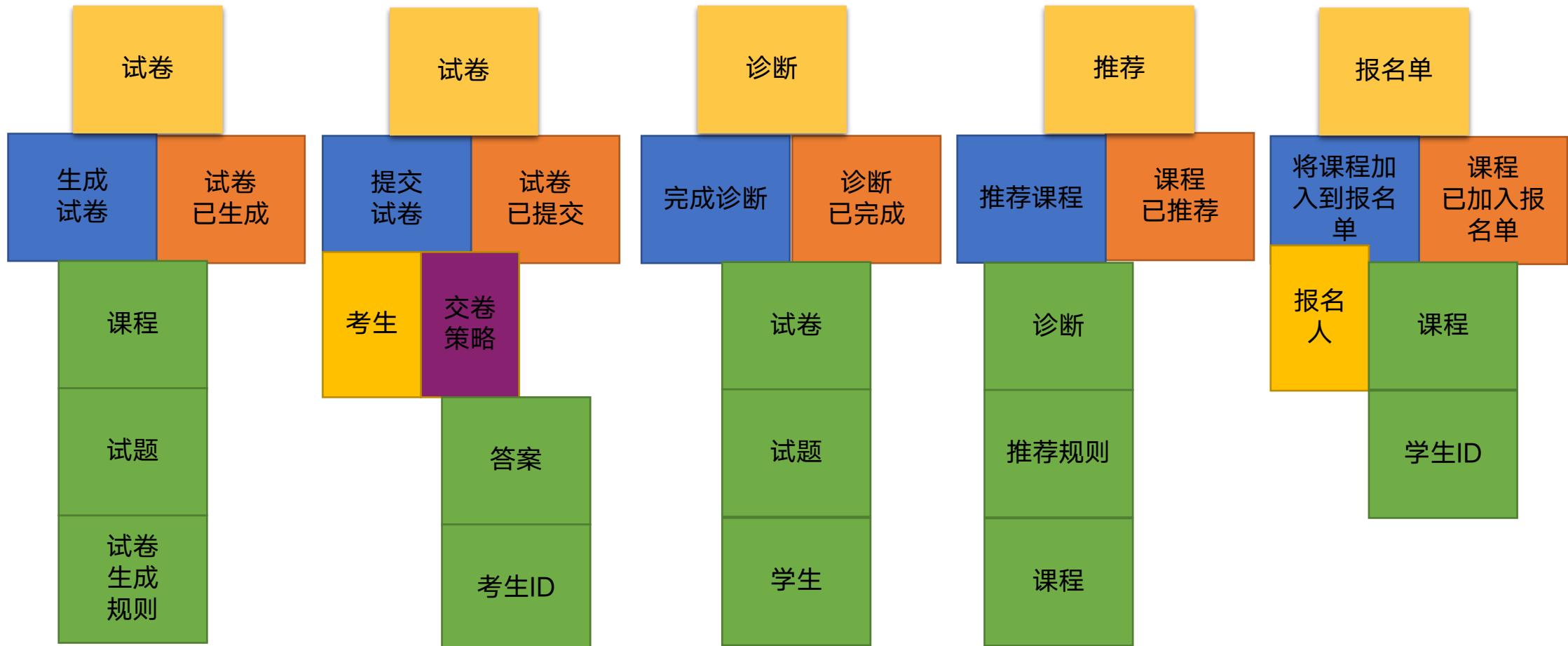
|| 识别参与者



|| 识别参与者



|| 领域分析建模



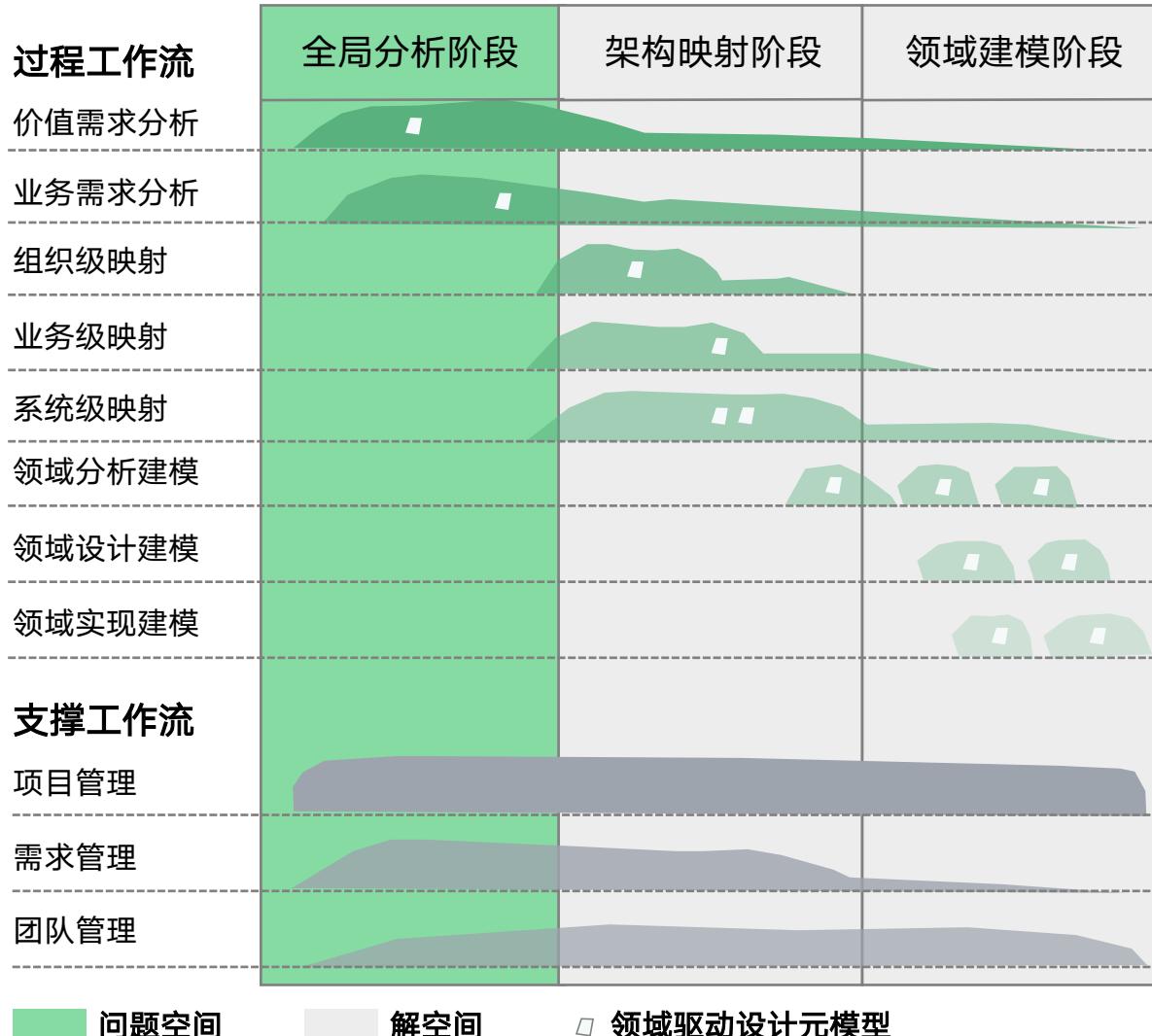
|| 领域分析建模



03

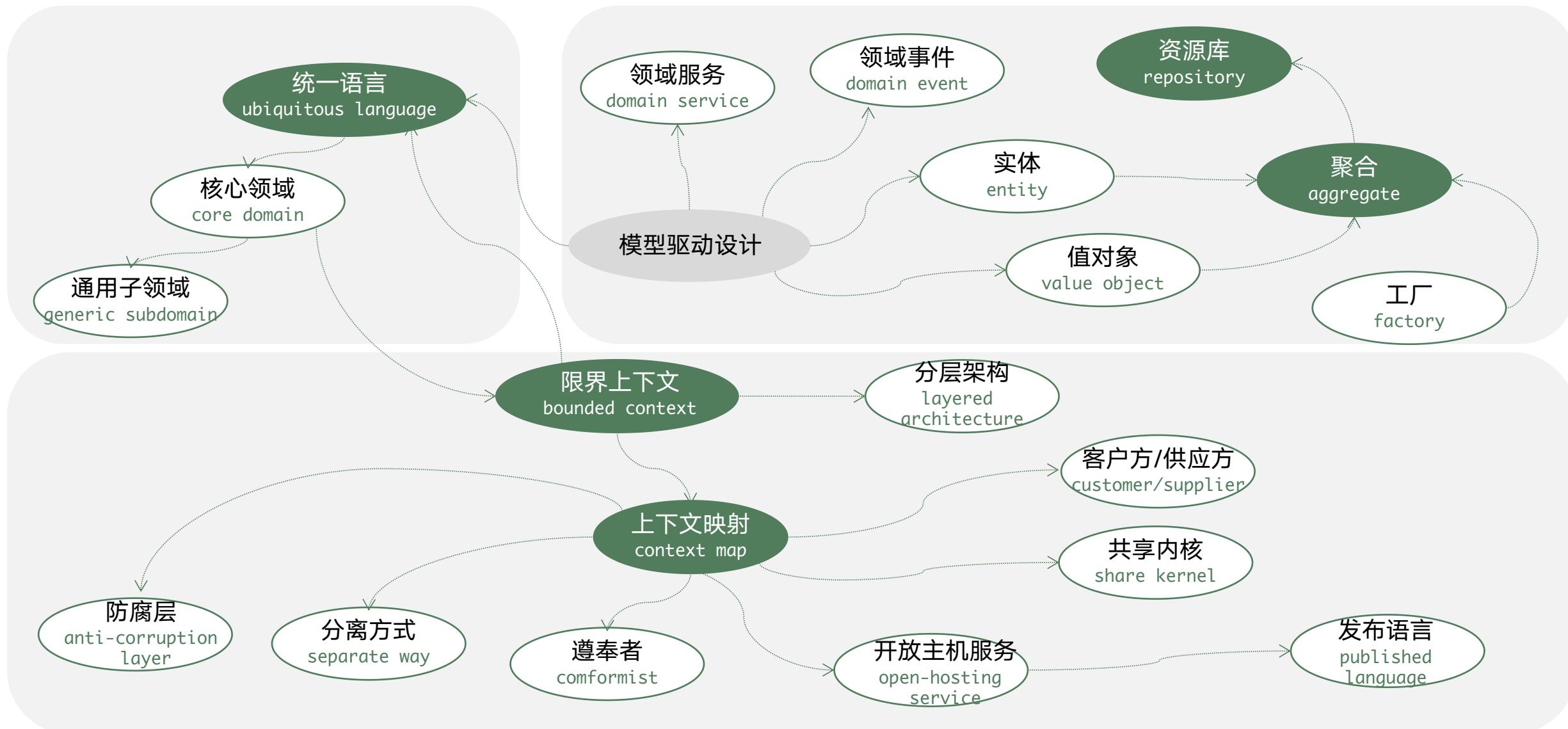
领域驱动设计回顾

|| 领域驱动设计统一过程

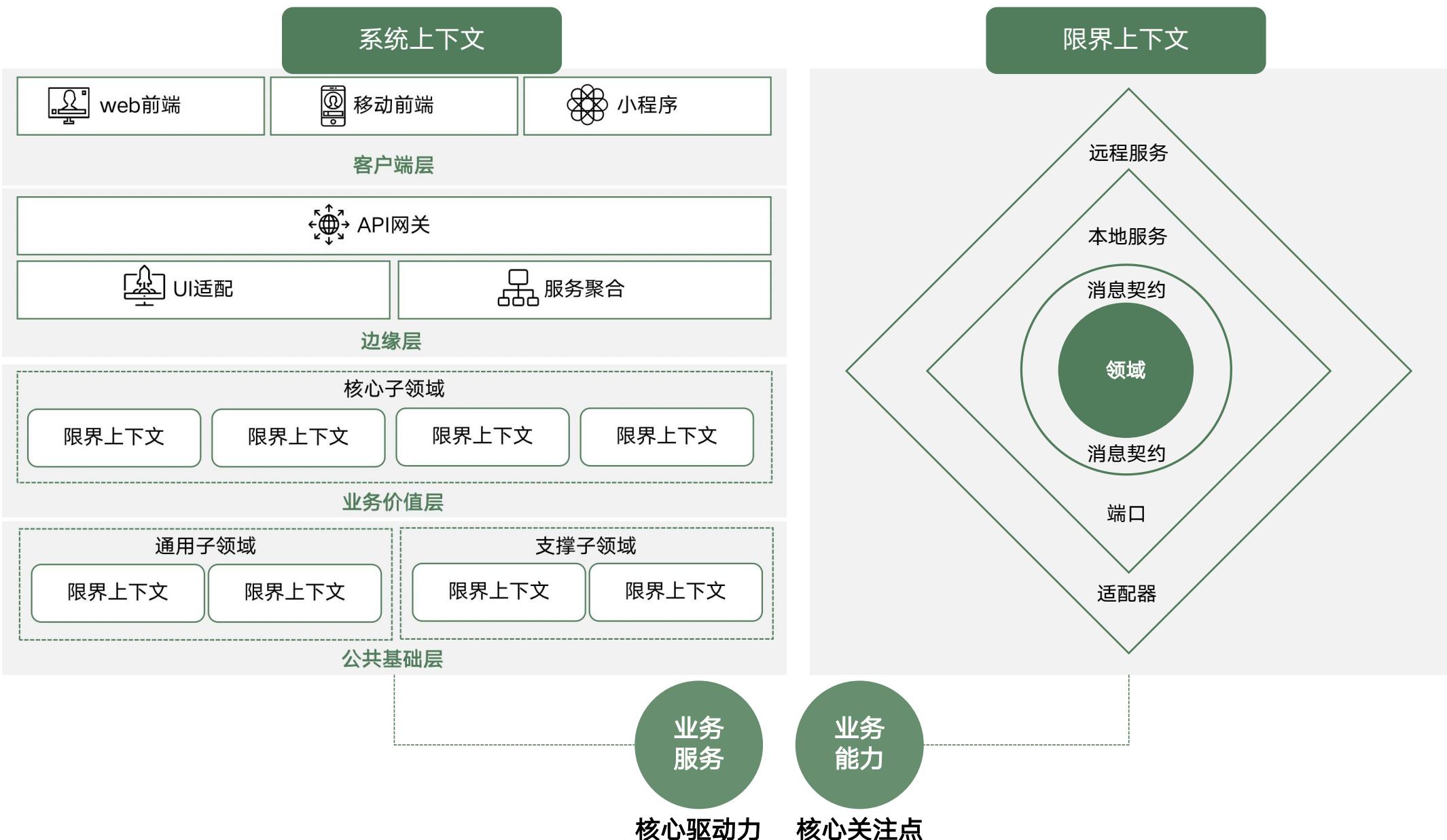


软件构建的过程就是不断对问题空间求解获得解决方案，进而组成完整的解空间的过程。在这个过程中，若要构建出优良的软件系统，就需要不断控制软件的复杂度。因此，我对领域驱动设计的完善，就是在问题空间与解空间背景下，能够定义能够控制软件复杂度的领域驱动设计过程，并将该过程执行的工作流限定在领域关注点的边界之内，避免该过程的扩大化。我将这一过程称为**领域驱动设计统一过程 (domain-driven design unified process, DDDUP)**。

|| 领域驱动设计的主要模式



|| 领域驱动架构风格

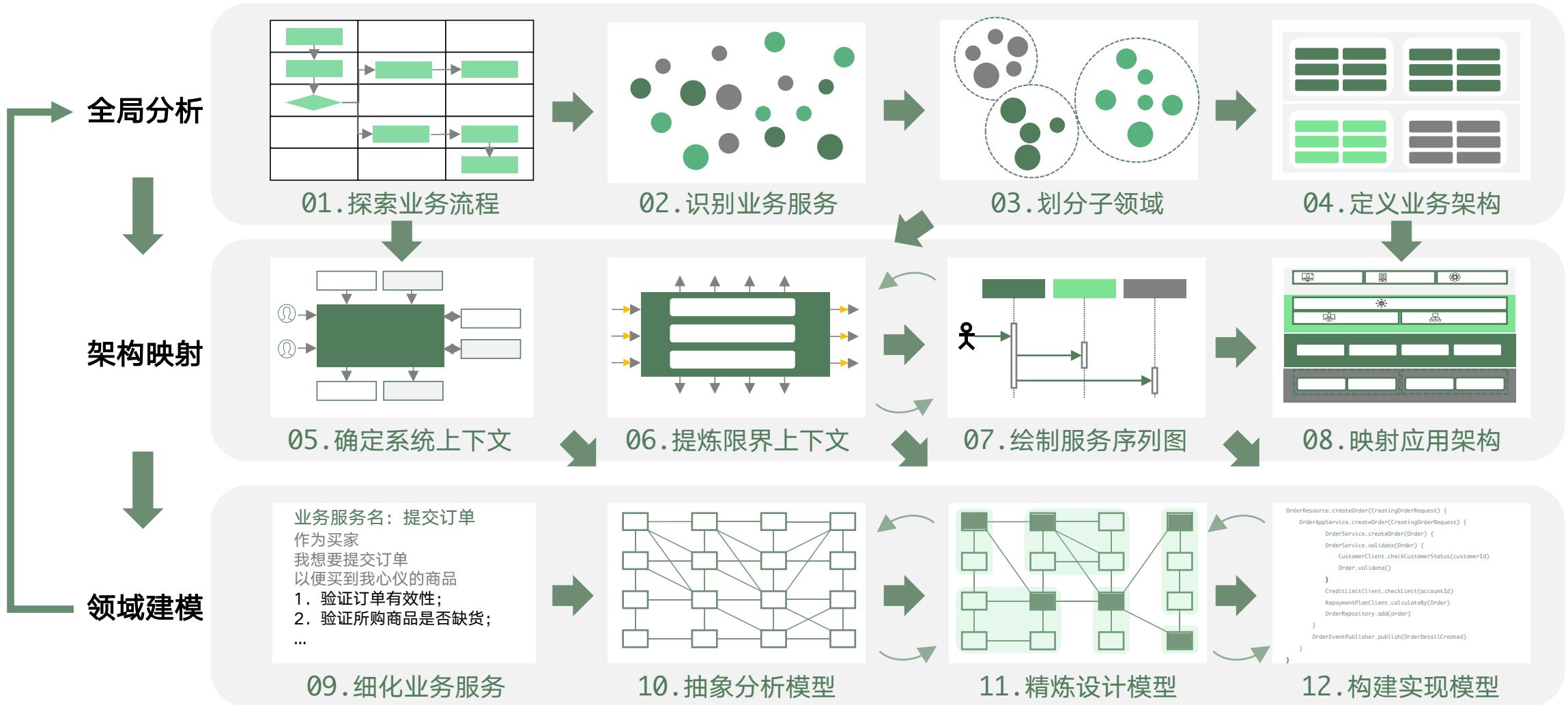




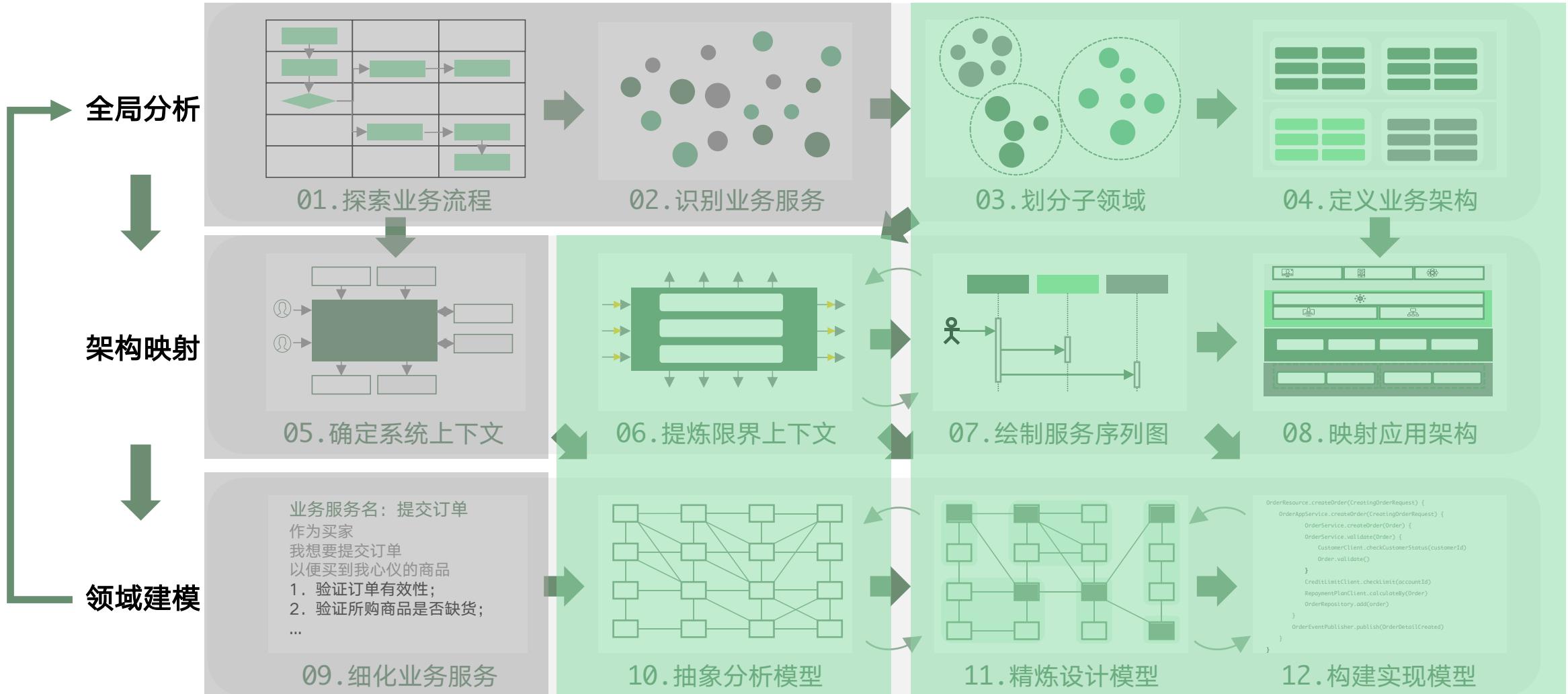
4

AI与DDD的双向赋能

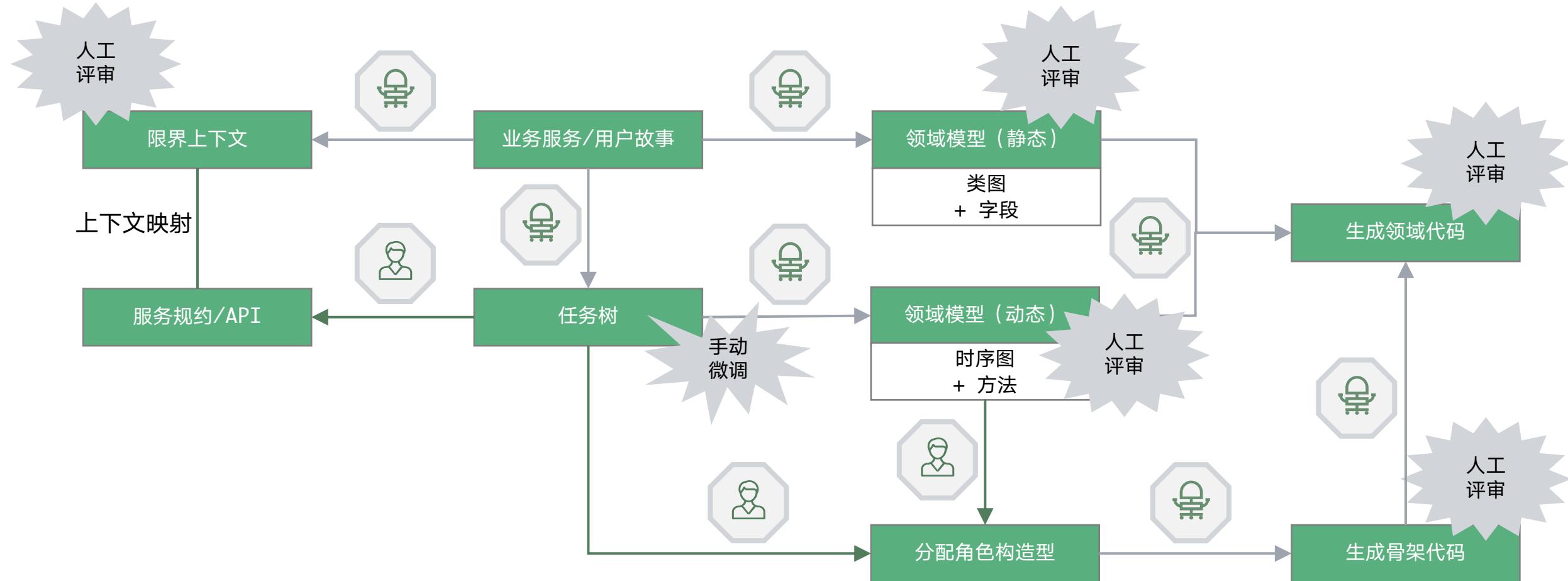
|| 服务风暴的平行模型



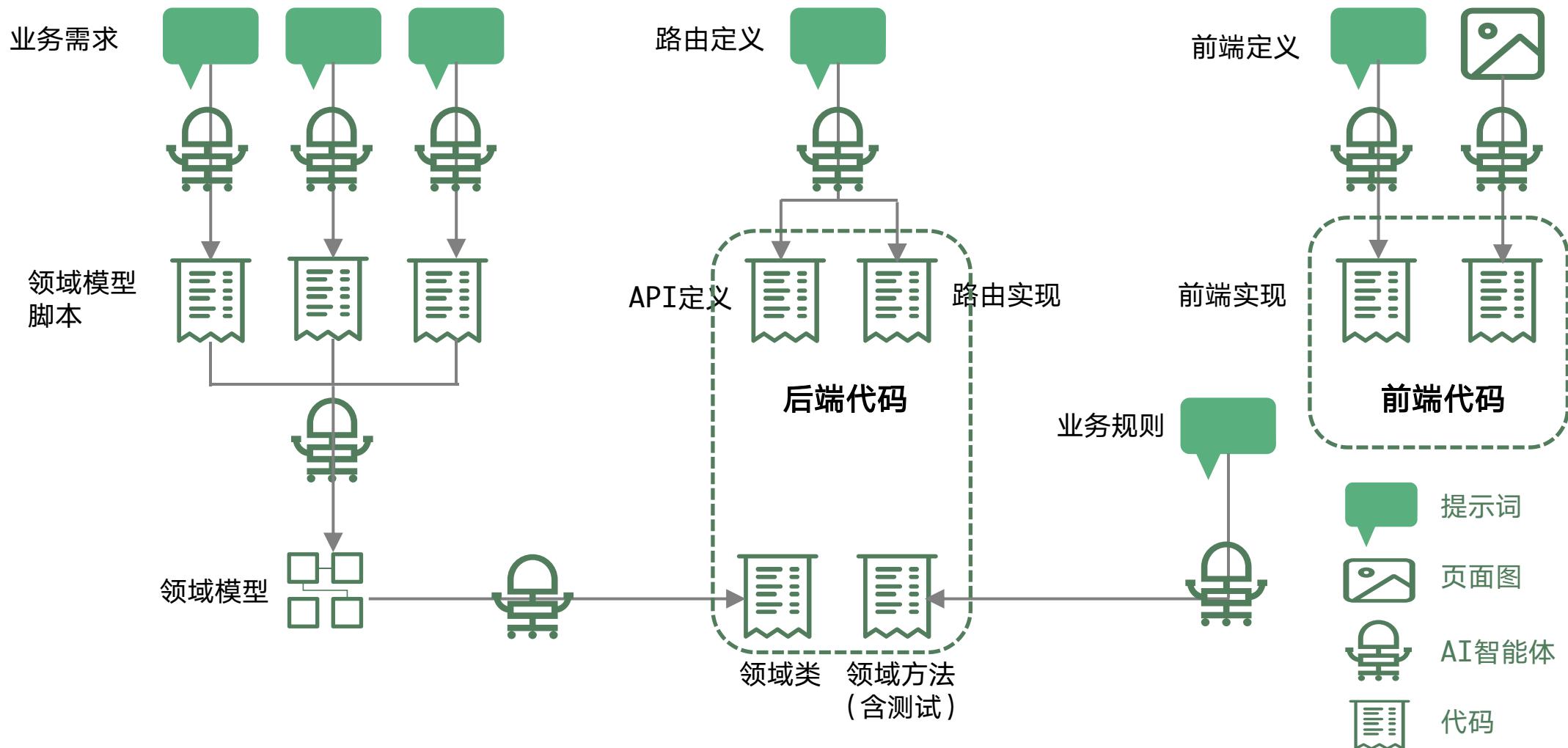
|| 服务风暴的平行模型



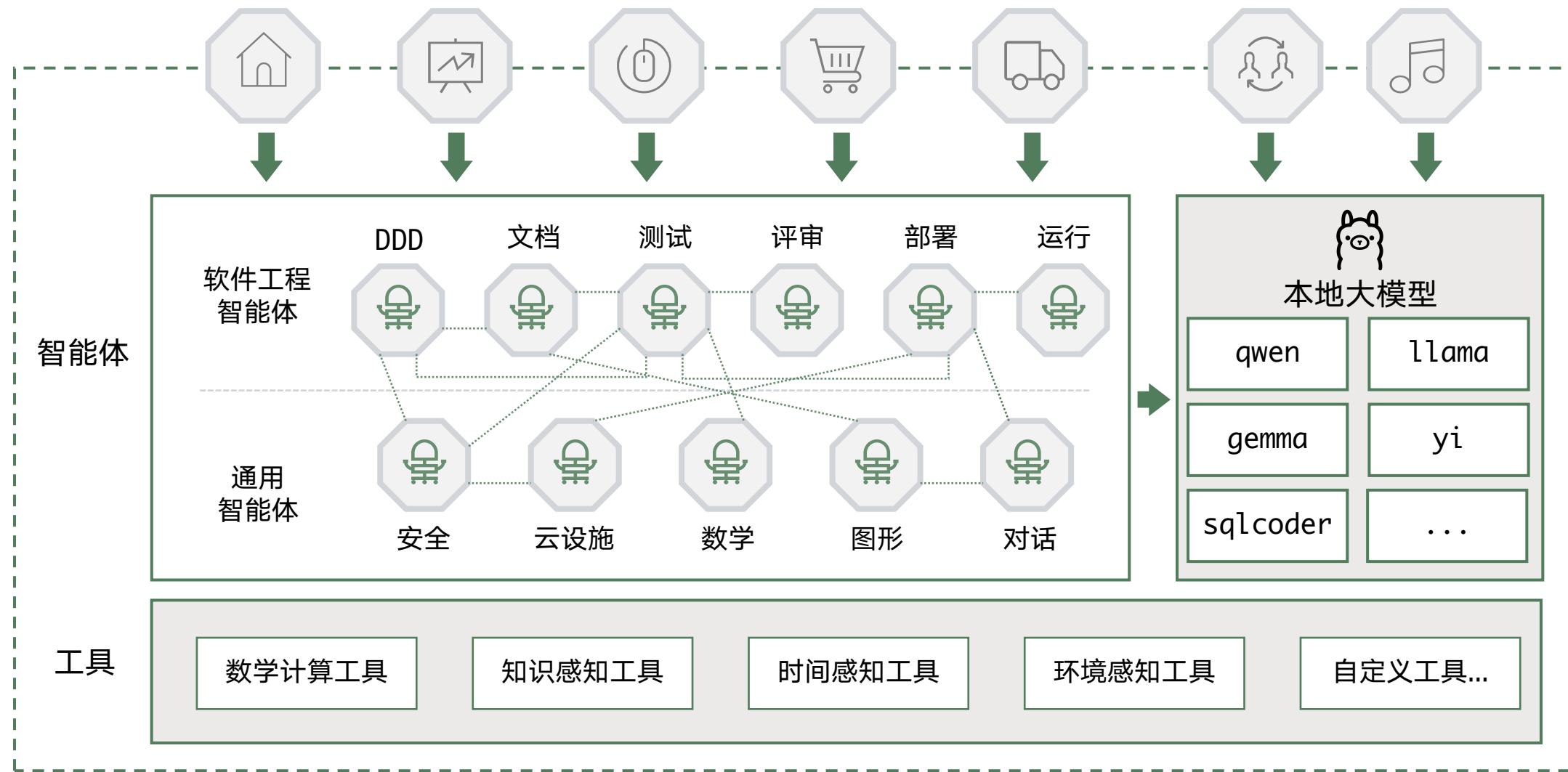
|| AI赋能的DDD设计流程



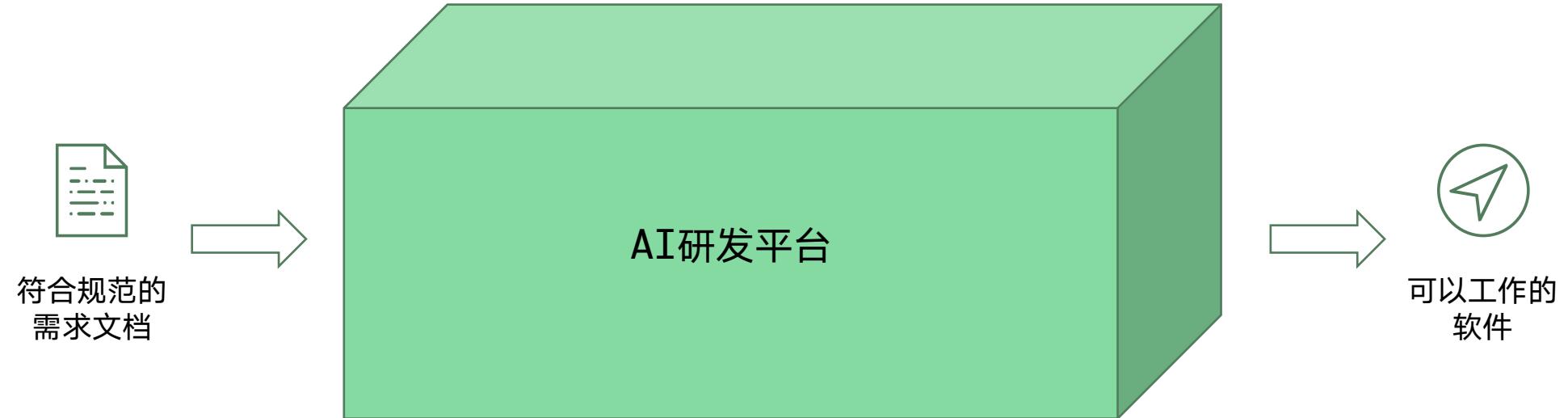
|| AI为DDD流程赋能



|| Agent-Based Architecture



|| DDD赋能的AI4SE



|| AI辅助领域建模



请输入规定格式的用户需求：

需求描述：

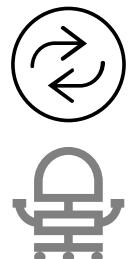
作为买家，
我想要提交订单，
以便买到我心仪的的商品。

基本流程：

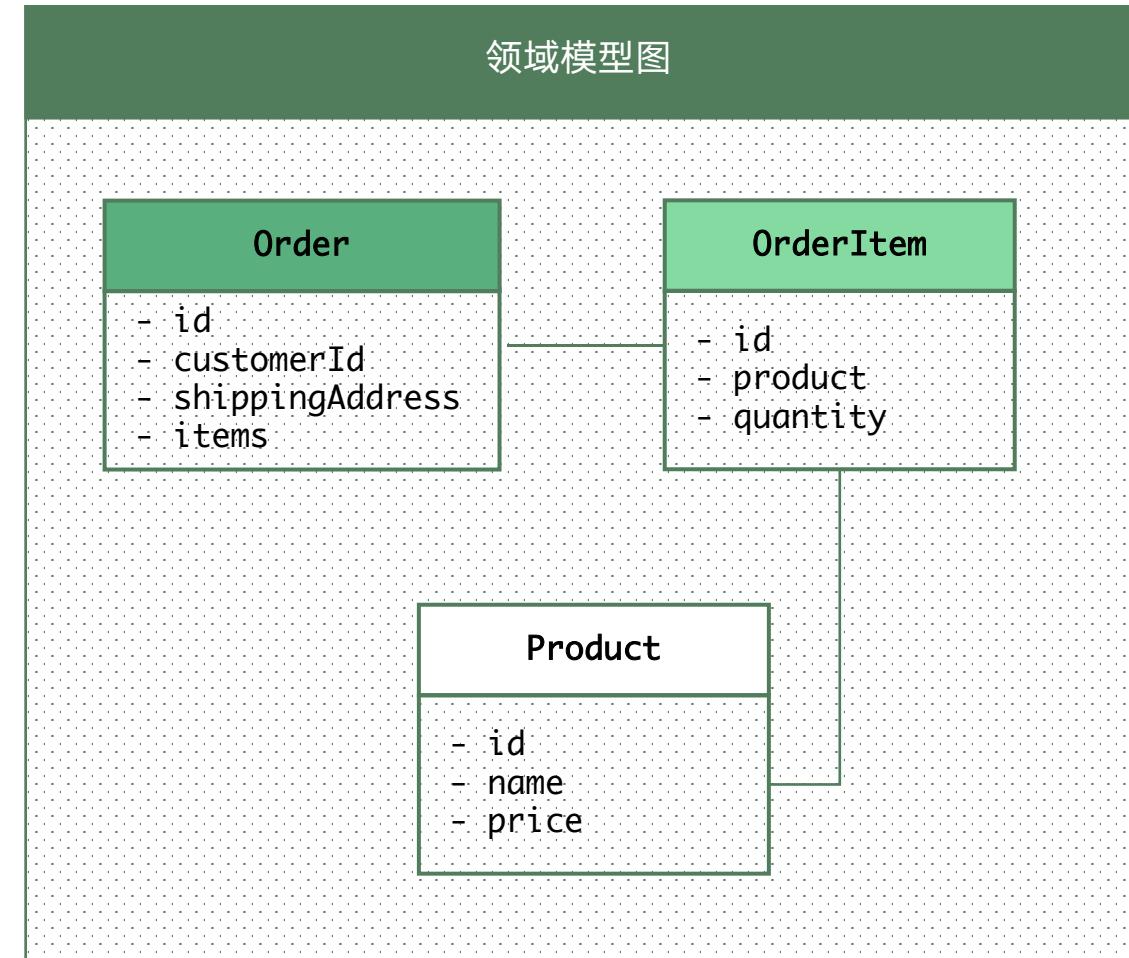
1. 验证订单有效性；
2. 验证订单所购商品是否缺货；
3. 插入订单；
4. 从购物车中移除所购商品；
5. 锁定库存；
6. 通知买家订单已提交。

替代流程：

- 1a. 如果订单无效，给出提示信息；
- 2a. 如果所购商品缺货，给出提示信息；
- 3a. 若订单提交失败，给出失败原因。



领域模型图



|| AI辅助代码生成

Order

- id
- customerId
- shippingAddress
- items



为领域类的方法描述业务规则

订单需要包含客户ID、配送地址、联系信息及已购商品的订单项；

订单项中商品的购买数量要小于或等于库存量；

领域层测试代码

```
// Unit Test 1: Check if order ID is required
@Test
public void testOrderIDRequired() {
    Order order = new Order();
    try {
        validate(order);
        fail("IllegalArgumentException should be thrown
when order ID is not provided");
    } catch (IllegalArgumentException e) {
        // Expected behavior
    }
}
```



领域层产品代码

```
public void validate(Order order) {
    if (order.getId() == null) {
        throw new IllegalArgumentException("Order ID is
required");
    }

    String title = order.getTitle();
    if (title == null || title.length() > 50) {
        throw new IllegalArgumentException("Title is
required and should not exceed 50 characters");
    }

    if (order.getItems().isEmpty()) {
        throw new IllegalArgumentException("Order item
list cannot be empty");
    }
}
```



——
**THANK
YOU**

