可视化的遗留系统微服务改造

毛超

DDD CHINA

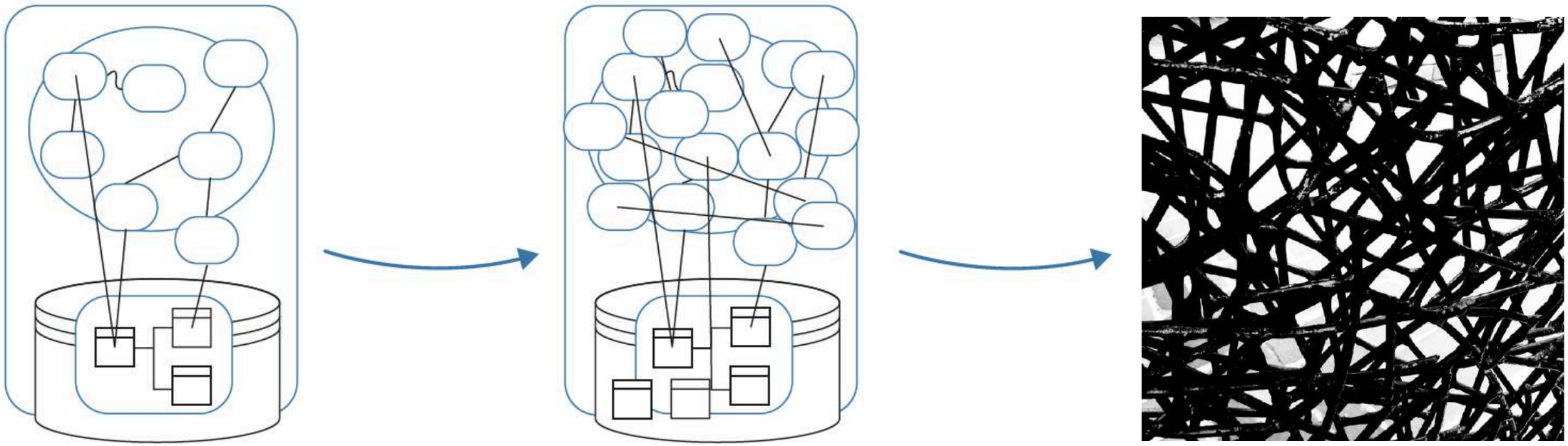
CONTENTS

引言

遗留系统、微服务架构

任何人类的设计都会腐化

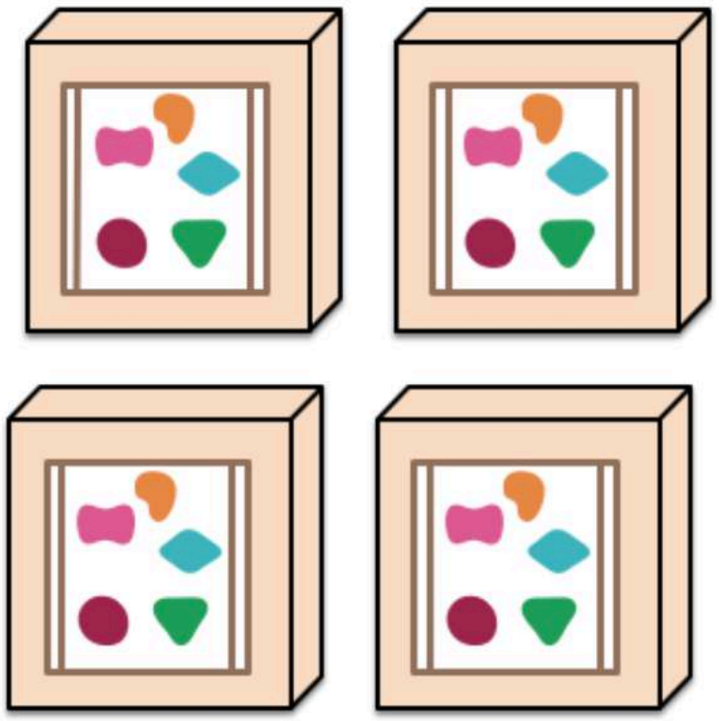
软件当然也不例外



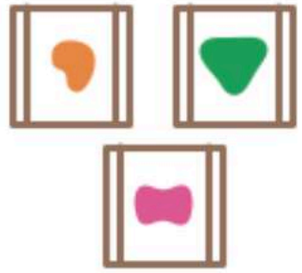
拆成微服务

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

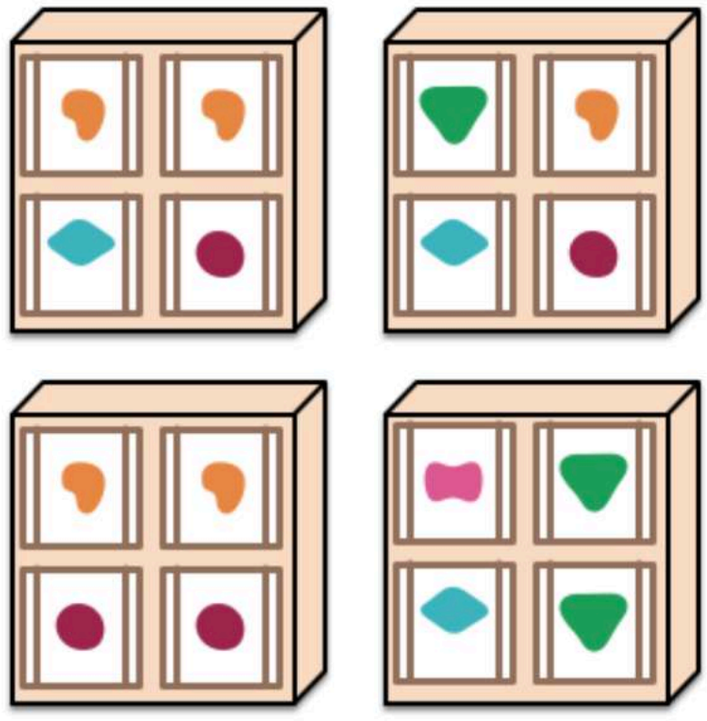
... and scales by replicating the monolith on multiple servers



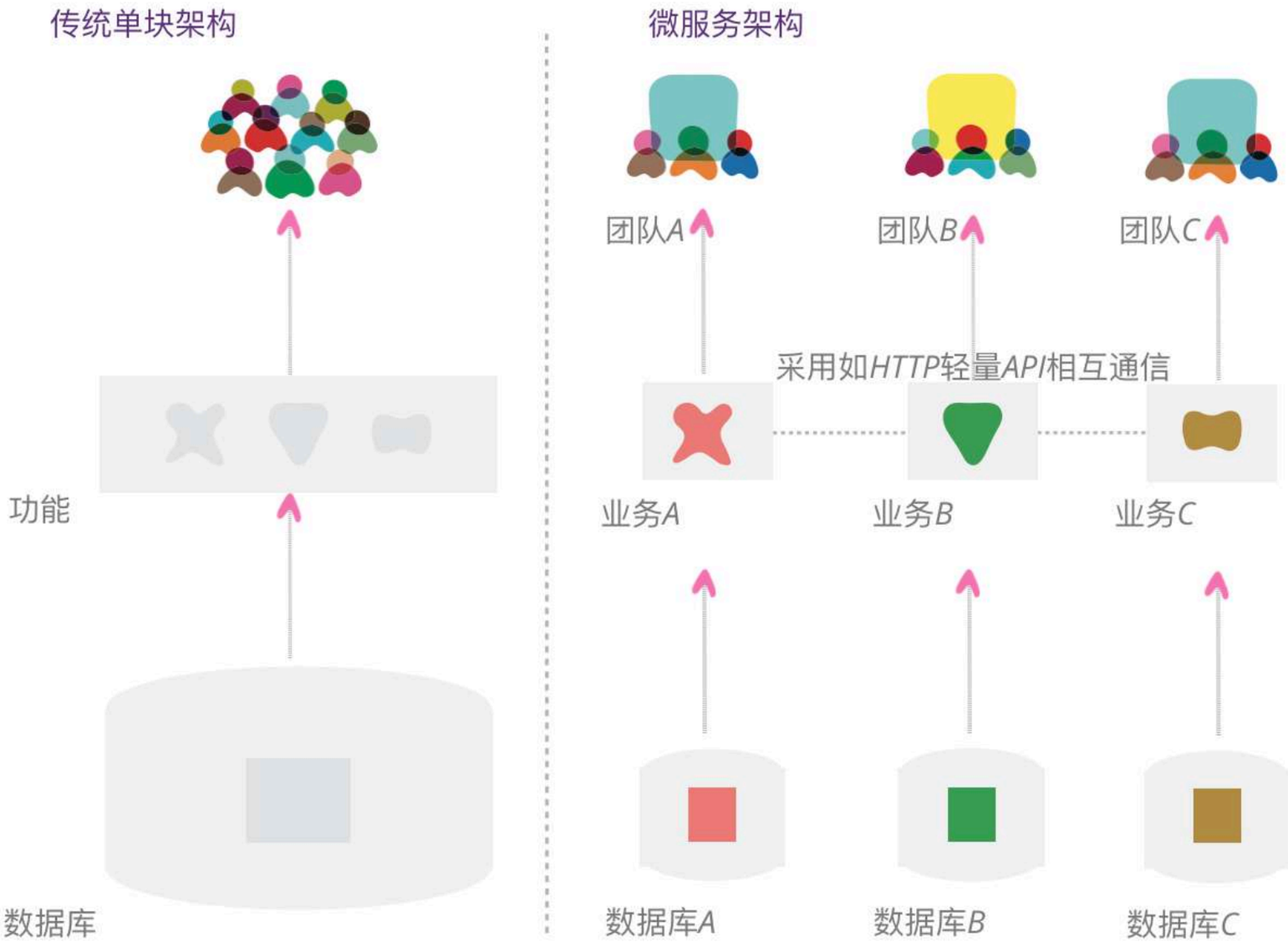
A microservices architecture puts each element of functionality into a separate service...



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



微服务架构的九大特征



- 通过服务进行组件化
- 围绕业务能力组织
- 做产品而不是做项目
- 智能端点与傻瓜管道
- 去中心化地治理技术
- 去中心化地管理数据
- 基础设施自动化
- 容错设计
- 演进式设计

可视化能帮我们什么

掌握系统业务

明确系统边界

小步改造系统

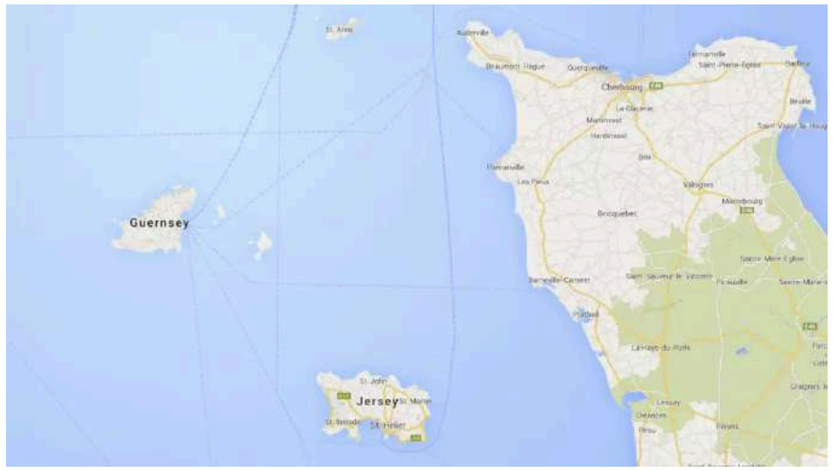
可视化的认识遗留系统

C4模型、用户画像、用户旅程

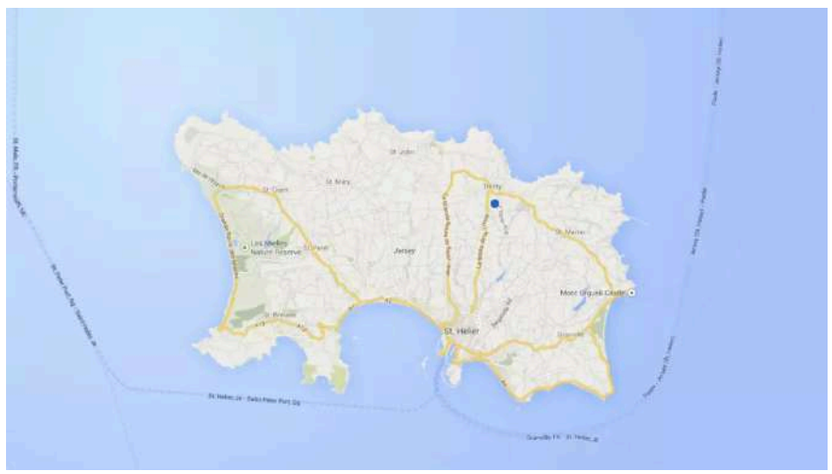
C4模型系统架构可视化

国家级

省级



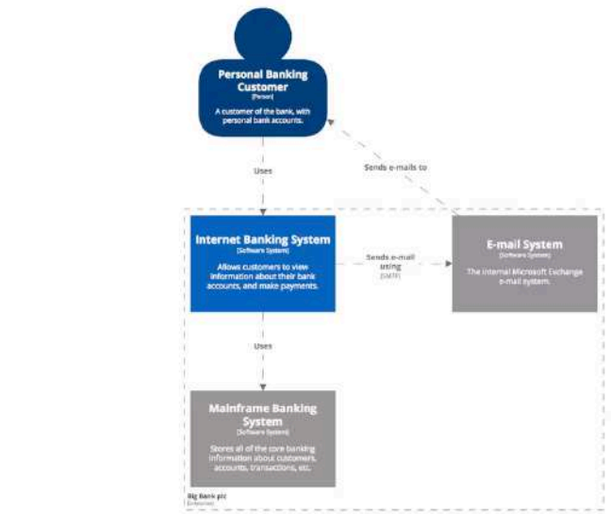
市级



道路级

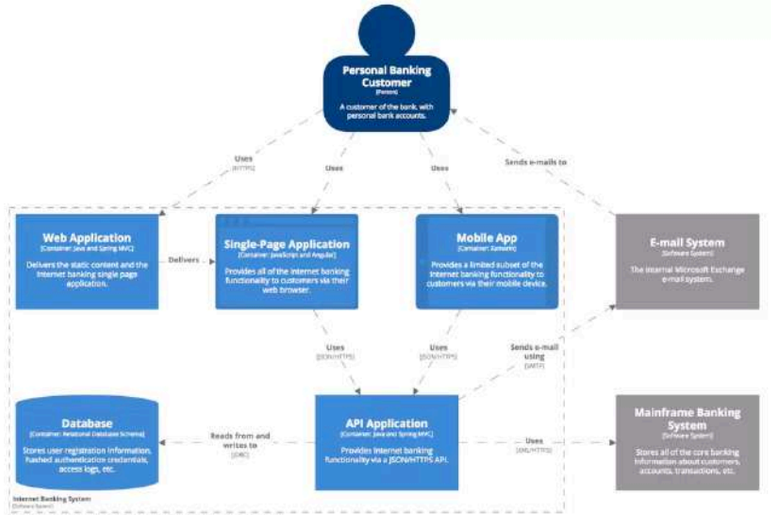
C4模型系统架构可视化

系统上下文图



System Context diagram for Internet Banking System

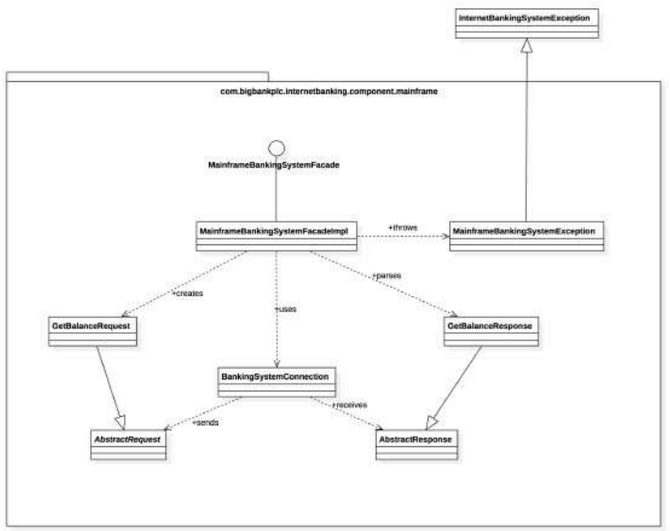
容器图



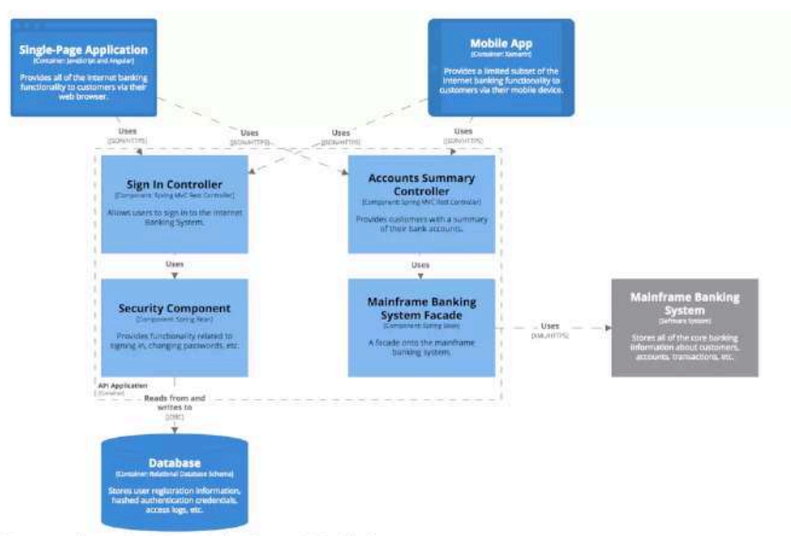
Container diagram for Internet Banking System

已可视化

代码图



组件图



Component diagram for Internet Banking System - API Application

用户画像和旅程系统功能用户可视化

已可视化

用户画像

基本信息

- 28岁，女，未婚
- 入行3年，当客户经理1年
- 客户多为HW公司员工，及周边客户，客户总数800人，核心客户400人
- 整体业绩中等偏上，保险业绩中等偏下
- 认为大多数客户没有购买保险的需求

工作内容

- 维护客户：电话维护，邀约面访
- 生客： 提高服务感受度，提高信任感；熟客： 不定期产品检视调整
- 主要考核内容为：AOM - 管理客户的总资产（最难）；客户数量，月初和月末的精力会主要专注在客群的提升中收 - 即各块的中间收入，保险的占比最大

目标

- **短期目标**
  完成各项KPI
- **中期目标**
  掌握销售技巧，
- **长期目标**
  财务自由，做

她说：

保险只是中收的一部分，只会向分析后觉得有需要的客户推荐保险，不能为了卖保险而破坏客户关系；
客户购买保险的渠道很多，信息也比较透明，万能险要取消了，保险会越来越难卖。

用户旅程

突出用户信息，诉求和价值体现

还原业务场景

可视化的划分遗留系统

领域驱动设计、事件风暴工作坊、服务画布

好的设计

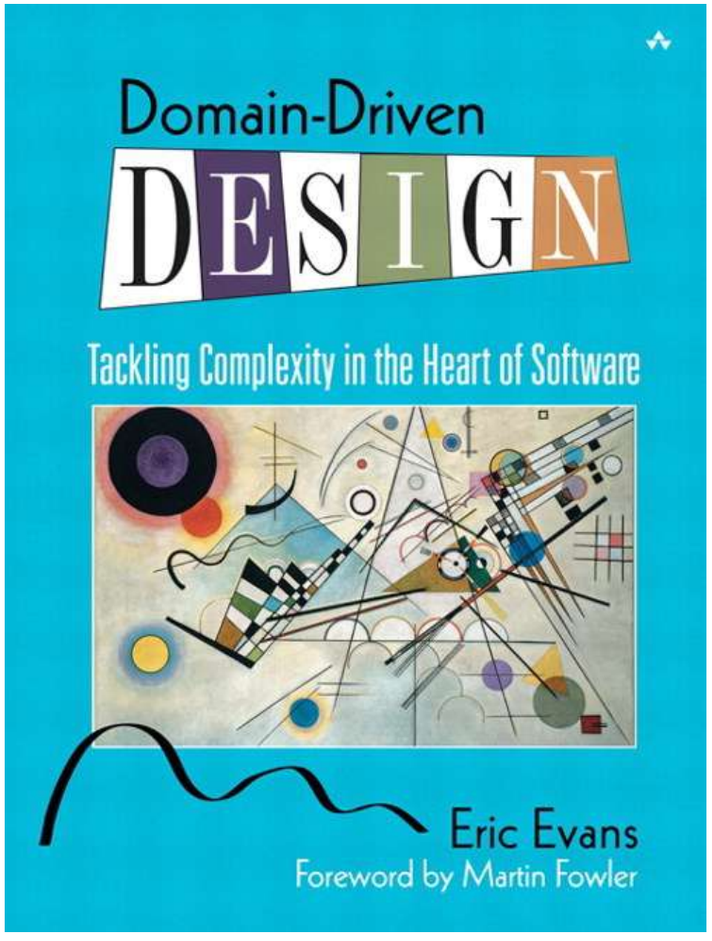
高内聚

就是把相关的行为聚集在一起，把不相关的行为放在别处。如果你要修改某个服务的行为，最好只在一处修改。

低耦合

如果做到了服务之间的松耦合，那么修改一个服务就不需要修改另一个服务。一个松耦合的服务应该尽可能少地知道与之协作的那些服务的信息。

领域驱动设计



- 领域驱动设计是一种处理高度复杂域的设计思想，试图分离技术实现的复杂性，围绕**业务概念**构建**领域模型**来控制业务的复杂性，以解决软件难以理解，难以演化等问题。团队应用它可以成功地开发复杂业务软件系统，使系统在增大时仍然保持敏捷。

事件风暴工作坊

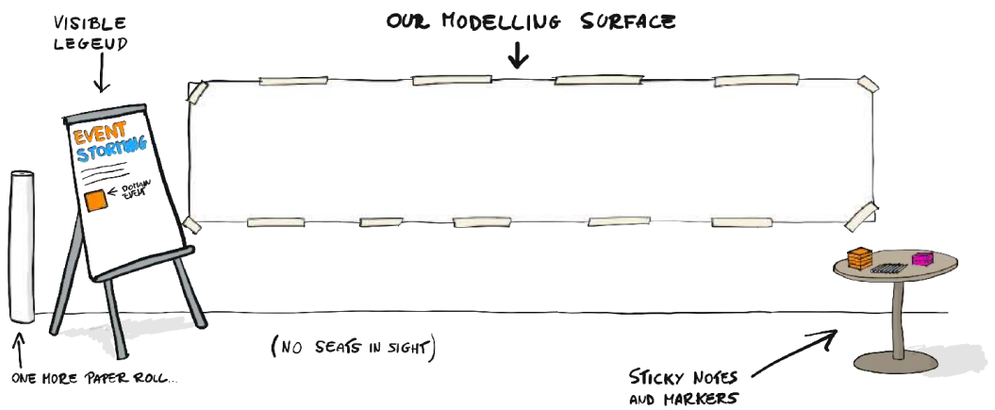
简介

- Event Storming是一种领域建模的实践，是一种快速探索复杂业务领域的方法：
- 最初由Alberto Brandolini 开发，经过DDD社区和TW的很多团队实践验证后，于2015年11月进入ThoughtWorks技术雷达

- Powerful：可以让实践者在数小时内理解复杂的业务模型
- Engaging：把带着问题的人和拥有答案的人共聚一堂构建模型
- Efficient：跟DDD的实现模型高度一致，并能快速发现Aggregate和 Bounded Context
- Easy：标记都很简单，没有复杂的UML
- Fun

活动准备

- 正确的人：业务人员，领域专家，技术人员，架构师，测试人员等关键角色都要参与其中
- 开放空间：有足够的空间可以将事件流可视化，让人们可以交互讨论
- 彩色即时贴：至少三种颜色



寻找领域事件

事件风暴 → 命令风暴 → 寻找聚合 → 划分限界上下文

什么是事件?

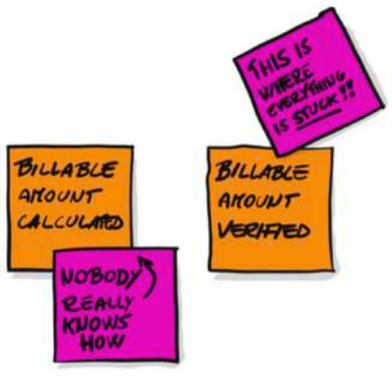
事件：**领域专家关心的**，在**业务上**真实发生的事

例1: 客户订单已提交

例2: 对账已完成，每月末夜间触发

为什么用事件?

通过事件的方式对过去发生的事情进行溯源因为过去所发生的对业务有意义的信息都会通过某种形式保存下来。事件风暴能够让领域专家和工作坊参与者一起明确在业务上究竟是**什么领域模型发生了什么改变**，最终的软件系统需要关注业务过程中发生的业务数据的变化。



如何进行事件风暴?

1. 确定要进行事件风暴的业务场景，场景需要单一而且清晰；

2. 用“XXX已XXX”的格式在橙色便利贴上写下事件，工作坊参与者需要对事件定义达成一致；

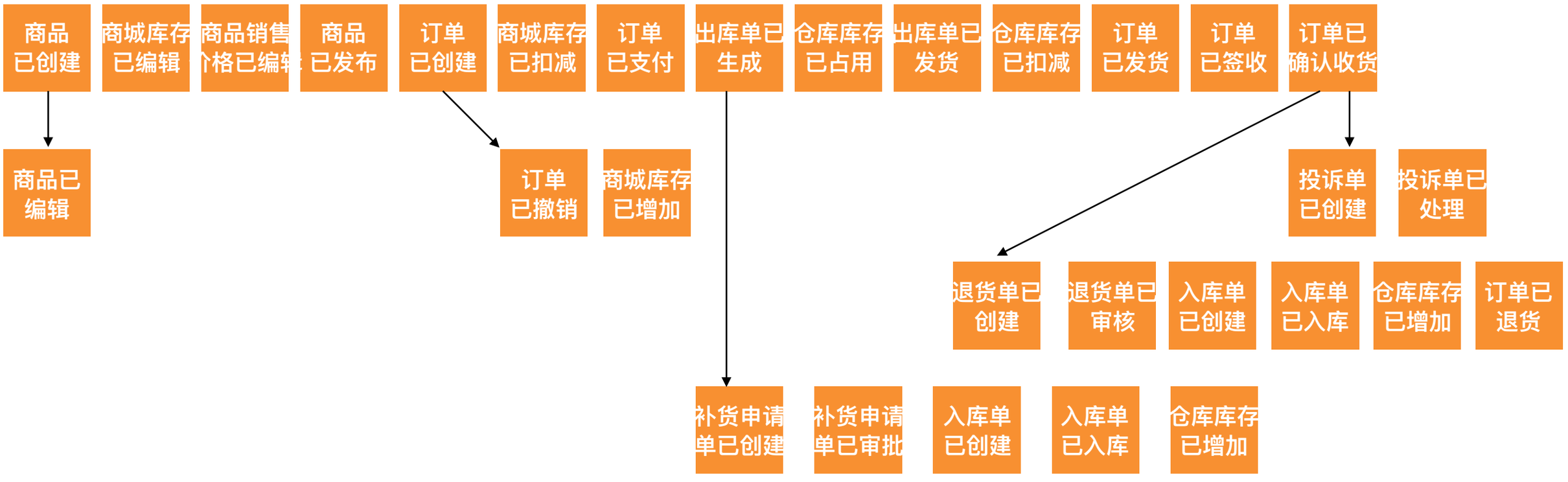
3. 根据时间顺序把事件便利贴贴到白板上；

4. 如果一个事件有同步发生的其它事件，把其它事件放在事件下方；

5. 如果发现了业务中的问题点，用红色便利贴记录为什么是一个问题；

6.以上步骤完成以后，再从最后一个事件开始来反向验证事件的完整性（即：走查前后事件的关联关系，防止有些事件被遗漏）。

事件风暴示例


商品已创建
商城库存已编辑
商品销售价格已编辑
商品已发布
订单已创建
商城库存已扣减
订单已支付
出库单已生成
仓库库存已占用
出库单已发货
仓库库存已扣减
订单已发货
订单已签收
订单已确认收货
商品已编辑
订单已撤销
商城库存已增加
投诉单已创建
投诉单已处理
退货单已创建
退货单已审核
入库单已创建
入库单已入库
仓库库存已增加
订单已退货
补货申请单已创建
补货申请单已审批
入库单已创建
入库单已入库
仓库库存已增加

寻找命令

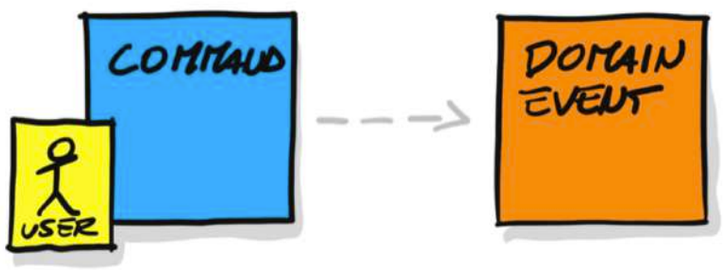
事件风暴 → **命令风暴** → 寻找聚合 → 划分限界上下文

什么是命令?

命令：**什么活动产生了事件**

例1: 提交客户订单

例2: 启动夜间对账



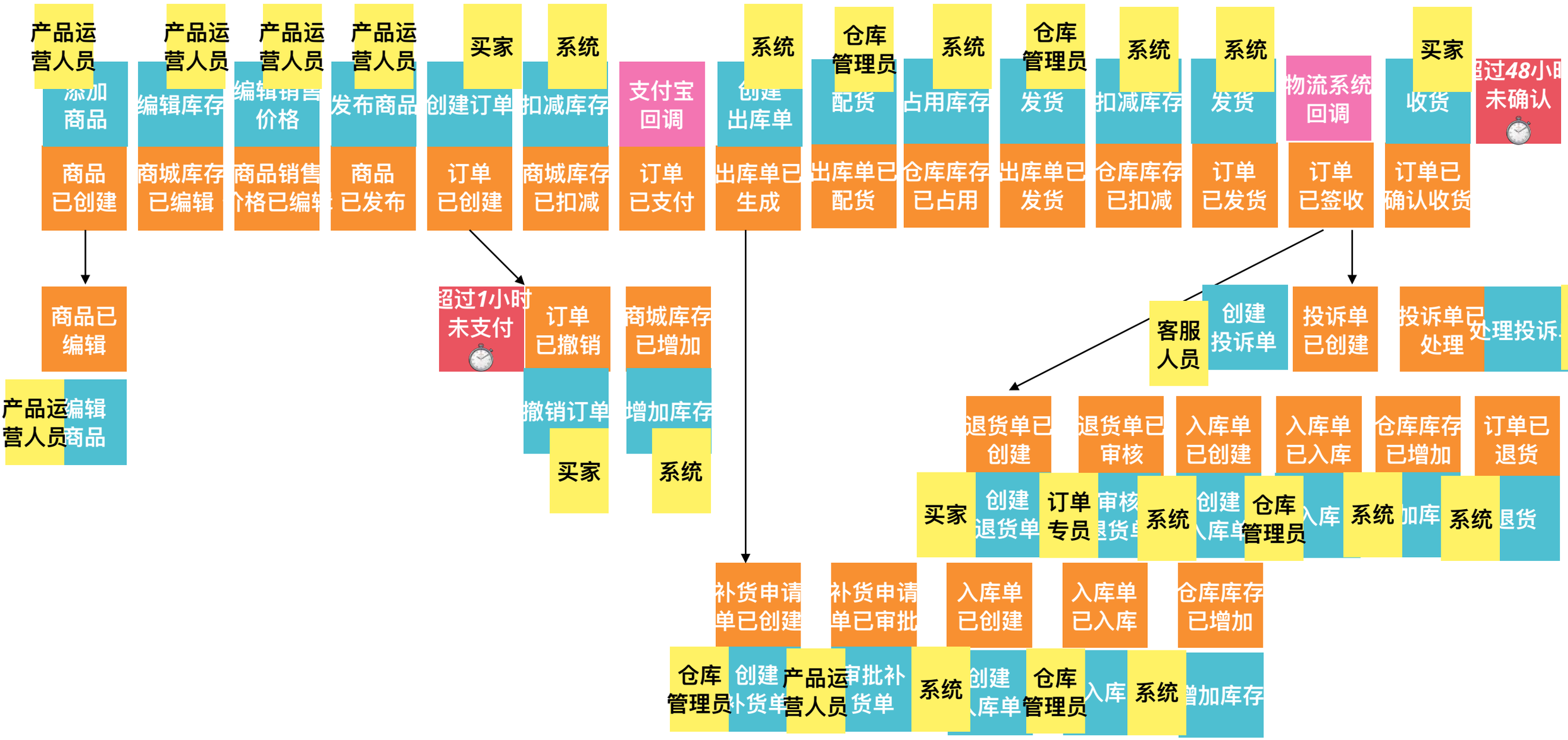
为什么用命令?

事件是业务上的输出，命令是业务上的输入。命令以及相应角色可以明确最终软件系统会有哪些功能给外界使用。命令和事件将会在后续的环节中指导API的设计。

如何进行命令风暴?

1. 将命令写在蓝色即时贴上，可以是
   - 用户从UI界面进行的操作
   - 外部系统触发
   - 定时任务
2. 将命令贴在所产生的事件旁边；
3. 有的命令可能产生多个事件，将它们连接；
4. 在这个过程中可以识别出触发命令的外部系统（粉色即时贴）、persona（黄色即时贴）并进行表示；

命令风暴示例


产品运营人员
添加商品
商品已创建
商品已编辑
产品运营人员
编辑商品
产品运营人员
编辑库存
商城库存已编辑
产品运营人员
编辑销售价格
商品销售价格已编辑
产品运营人员
发布商品
商品已发布
买家
创建订单
订单已创建
超过1小时未支付
订单已撤销
买家
撤销订单
系统
扣减库存
商城库存已扣减
系统
商城库存已增加
增加库存
系统
支付宝回调
订单已支付
系统
创建出库单
出库单已生成
仓库管理员
配货
出库单已配货
系统
占用库存
仓库库存已占用
仓库管理员
发货
出库单已发货
系统
扣减库存
仓库库存已扣减
系统
发货
订单已发货
物流系统回调
订单已签收
买家
收货
订单已确认收货
超过48小时未确认
客服人员
创建投诉单
投诉单已创建
投诉单已处理
处理投诉
客服人员
退货单已创建
买家
创建退货单
退货单已审核
订单专员
审核退货单
入库单已创建
系统
创建入库单
入库单已入库
仓库管理员
入库
仓库库存已增加
系统
加库
订单已退货
系统
退货
补货申请单已创建
仓库管理员
创建补货单
补货申请单已审批
产品运营人员
审批补货单
入库单已创建
系统
创建入库单
入库单已入库
仓库管理员
入库
仓库库存已增加
系统
增加库存

寻找聚合

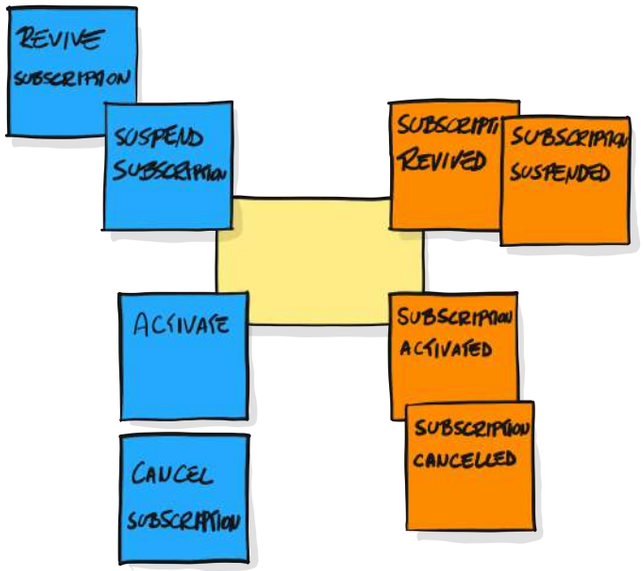
事件风暴 → 命令风暴 → **寻找聚合** → 划分限界上下文

什么是聚合?

聚合是一组相关领域模型的集合是用来封装业务的不变性。确保**关联关系**紧密的领域模型能够内聚在一起。

为什么用聚合?

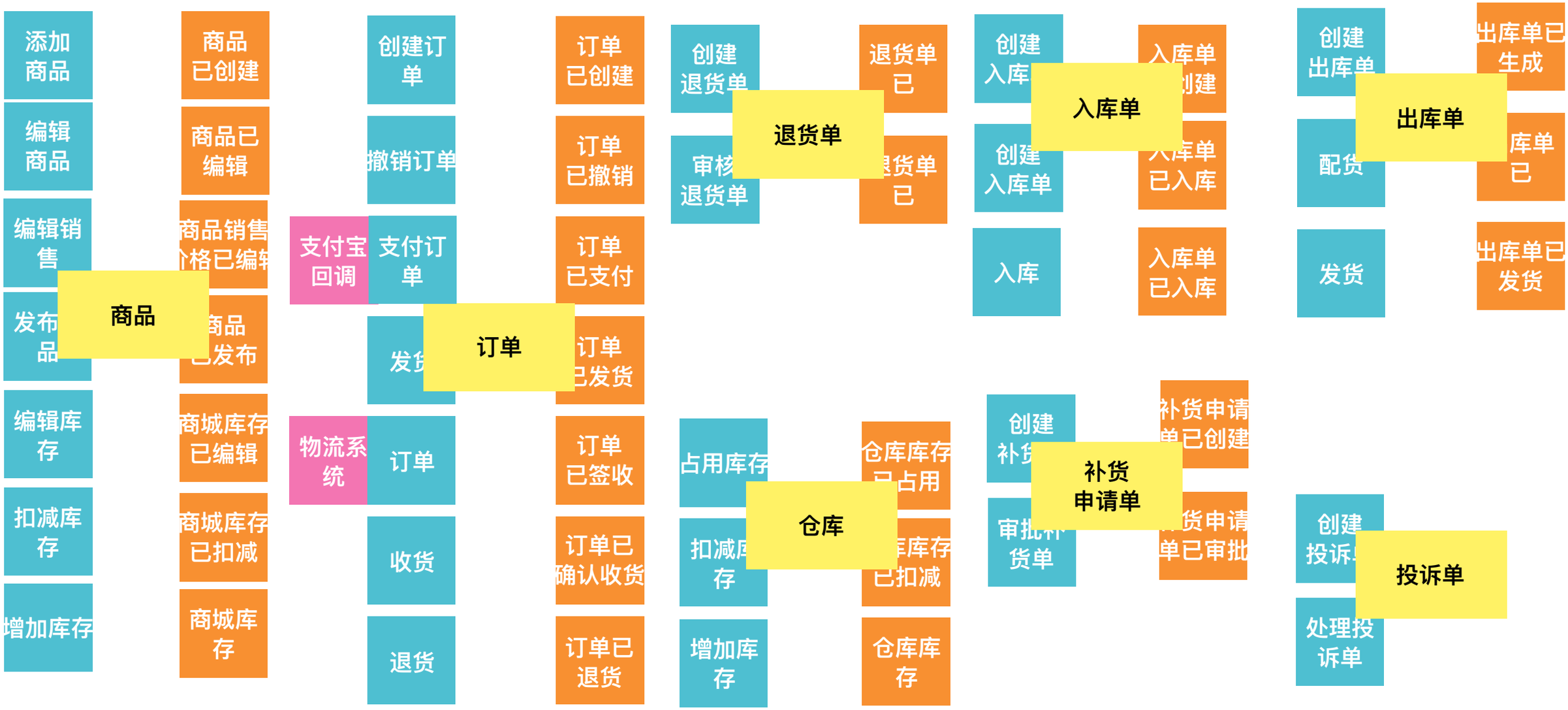
使用聚合的目的是**封装业务的不变性，同时强迫大家尽可能的简化领域模型之间的关联关系**。在业务层面进行高内聚，低耦合的设计。



如何寻找聚合?

1. 按照事件顺序依次通过提问来分析：
   - 这个事件会改变的领域模型是什么？**明确领域模型（简单理解就是事件中的涉及的业务名词）**
   - 这个领域模型是否可以独立访问？**如果是就是聚合**
   - 如果不能独立访问应该需要通过哪个领域模型来访问？**当前领域模型就是与该可独立访问的领域模型为同一个聚合**
2. 将命令贴在聚合的左面，是聚合的输入；事件贴到聚合的右面，是聚合的输出。
3. 再根据聚合的原则（下一页描述）来检验上面的划分结果是否匹配，如不匹配则基于划分原则并结合业务重新调整聚合。

聚合示例


添加商品
编辑商品
编辑销售
发布品
编辑库存
扣减库存
增加库存
商品
商品已创建
商品已编辑
商品销售价格已编辑
商品已发布
商城库存已编辑
商城库存已扣减
商城库存
创建订单
撤销订单
支付宝回调
支付订单
发货
物流系统
订单
收货
退货
订单
订单已创建
订单已撤销
订单已支付
订单已发货
订单已签收
订单已确认收货
订单已退货
创建退货单
审核退货单
退货单
退货单已
退货单已
创建入库
创建入库单
入库
入库单
入库单已入库
入库单已入库
创建出库单
配货
发货
出库单
出库单已生成
出库单已
出库单已发货
占用库存
扣减库存
增加库存
仓库
仓库库存已占用
仓库库存已扣减
仓库库存
创建补货
审批补货单
补货申请单
补货申请单已创建
补货申请单已审批
创建投诉
处理投诉单
投诉单

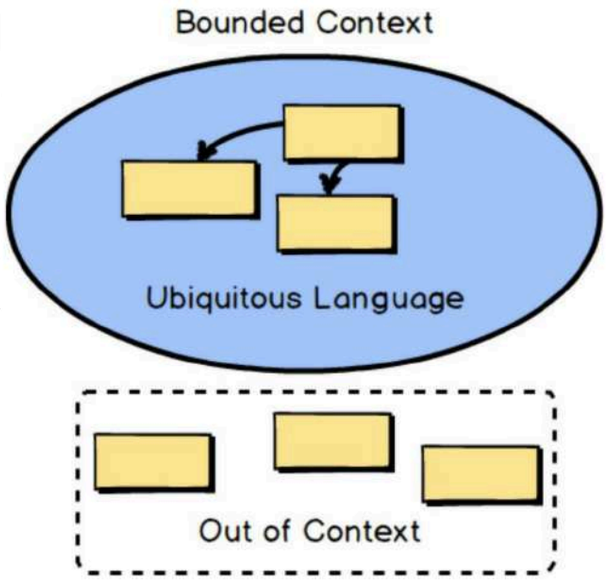
划分限界上下文

事件风暴 → 命令风暴 → 寻找聚合 → 划分限界上下文

什么是限界上下文?

限界上下文可以分为限界和上下文两个词来理解，限界：指一个界限，具体的某一个范围；

上下文：场景、环境；所以限界上下文是在某个场景或环境下的业务边界。该边界就是业务上的职责。



为什么使用限界上下文?

业务的扩展会产生越来越多的领域模型，任何大型项目都会存在很多的领域模型。当不同领域模型对应的软件代码被放在一起后，**软件就变得庞大且复杂，代码难于理解、且容易出现bug，所以需要通过限界上下文来明确定义领域模型的范围和职责。**

如何探索限界上下文?

1. 基于前面输出的聚合和领域模型，判断这些领域模型要解决的业务问题这些问题是否为同一个问题，如果是则放到一个限界上下文中（一个问题对应一个限界上下文），如果为否则拆分到不同的限界上下文中；

2.如果一个聚合（领域模型）同时解决多个问题时，则需要根据限界上下文的划分原则 (后面几页会详细描述) 对聚合（领域模型）进行拆分，拆分后对应的领域模型划分到不同的限界上下文中；

3.上面环节中所定义的问题大小（需考虑问题的变化原因、内在逻辑等）需与领域专家共同讨论完成。

限界上下文示例

每一个限界上下文
都是在明确边界

物流上下文

物流回调

支付上下文

支付宝回调

物流子域

仓储子域

出入库上下文

入库单

出库单

订单上下文

订单

退货单

销售子域

库存上下文

仓库库存

商品上下文

商品

客服子域

投诉上下文

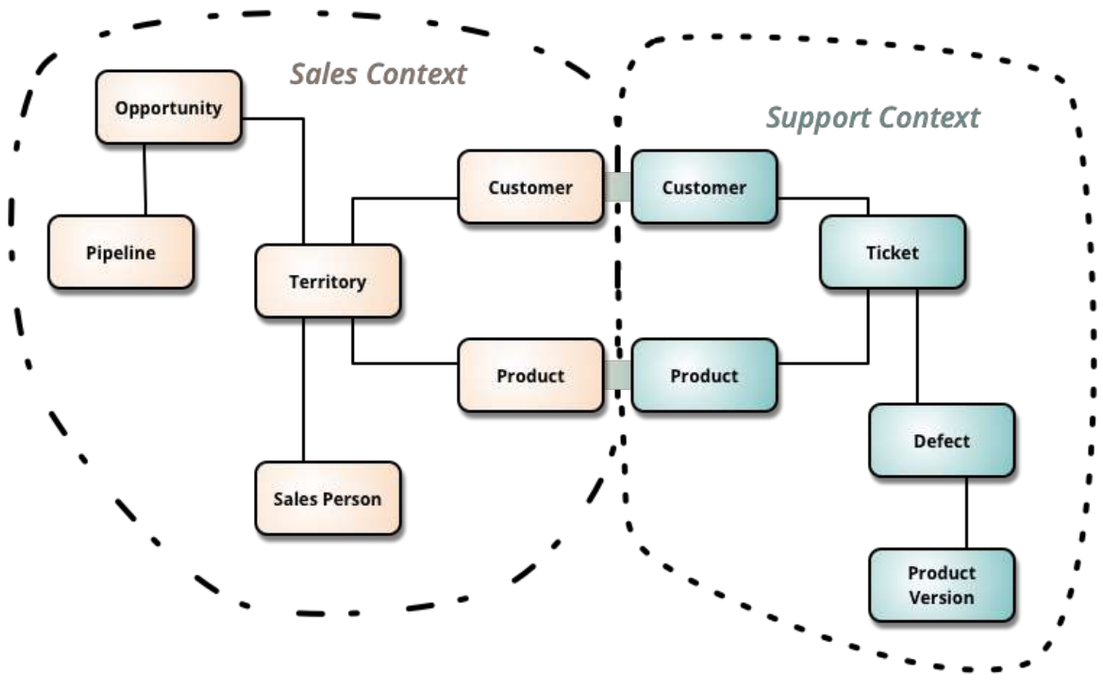
投诉单

补货上下文

补货申请单

微服务设计原则

1. 对齐限界上下文
2. 业务变更频率与相关度
3. 系统非功能性需求
4. 组织结构和康威定律
5. 团队规模
6. 技术异构和现有系统复杂度



服务画布

明确服务的范围 → **明确核心模型** → **明确服务包含的数据表**

子服务定义画布

子服务名称 | Domain

子服务名称 | Domain

子服务名称 | Domain Name

Pipeline

干系人 | Stakeholder

1. 买家
2. 商户

业务价值 | Value Proposition

1. 买家创建订单
2. 商户履行订单

业务行为 | Capabilities

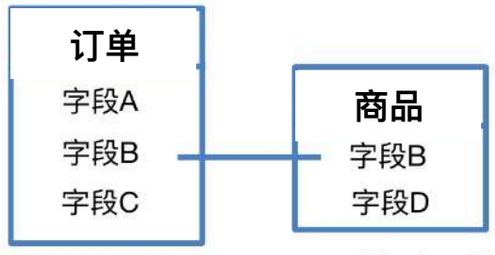
* * * *

已可视化

业务实体 | Key Concept

1. 订单
2. 商品
3. ***

实体模型 | Entity Model



Entity-Relationship Diagram

可视化的拆解遗留系统

微服务架构、绞杀模式、代码依赖分析、数据库依赖分析、
遗留系统拆解评分表、降龙八步

庖丁解牛拆解的最高境界

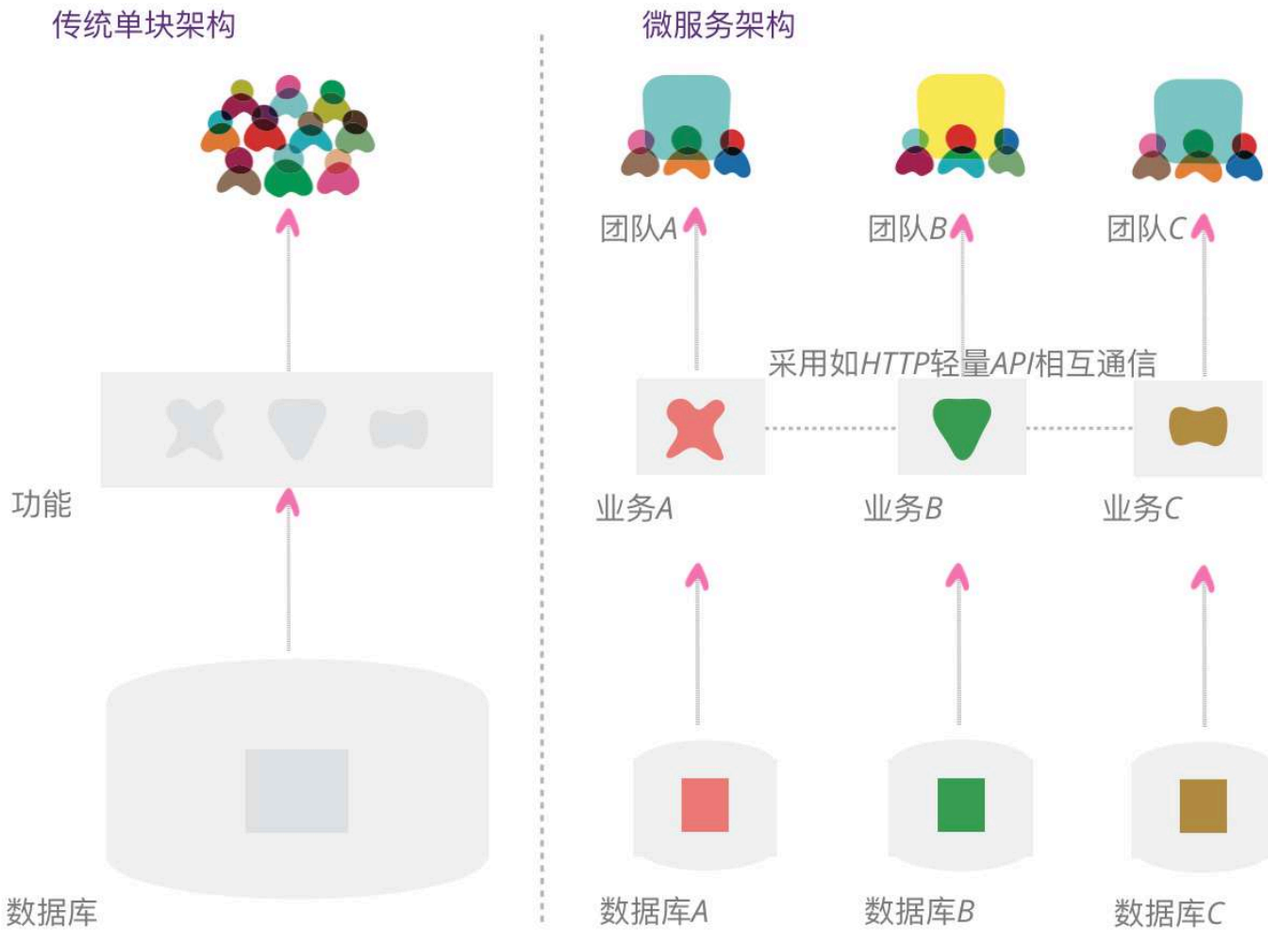
了解牛的生理构造

避开筋腱骨节交错的组织

从骨节的缝隙下手

十九年刀依然锋利

再看一眼微服务架构

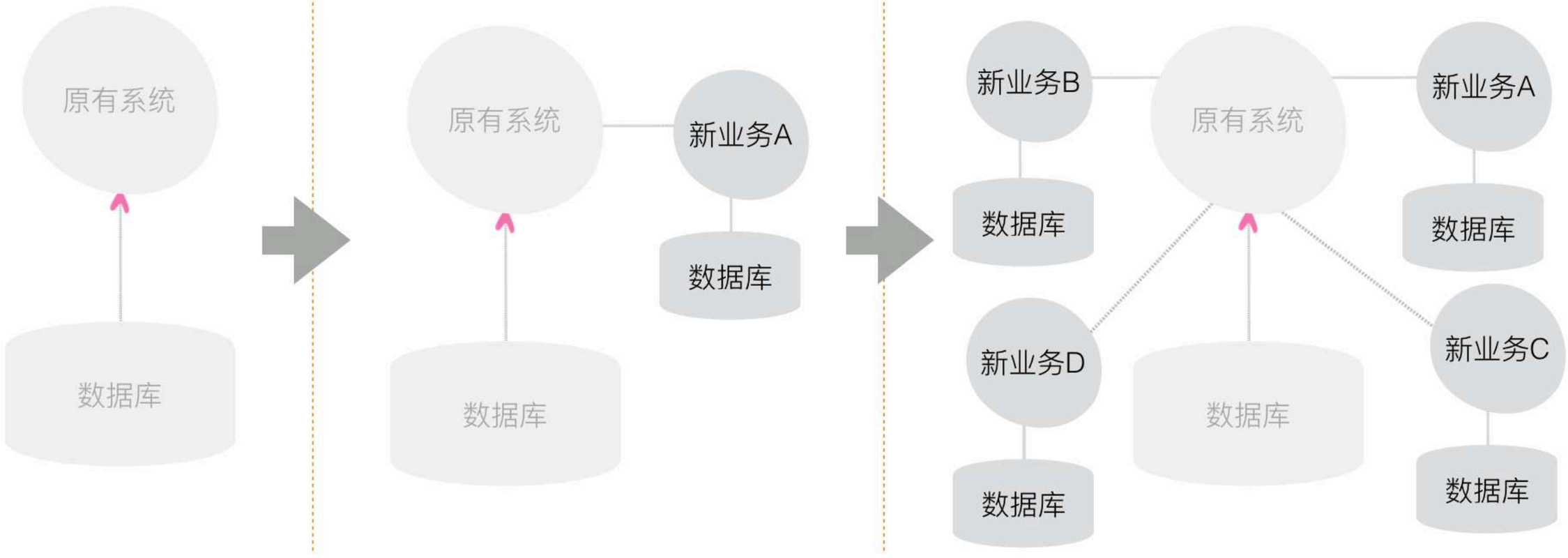


我们要做应用代码拆分

我们要做数据库拆分

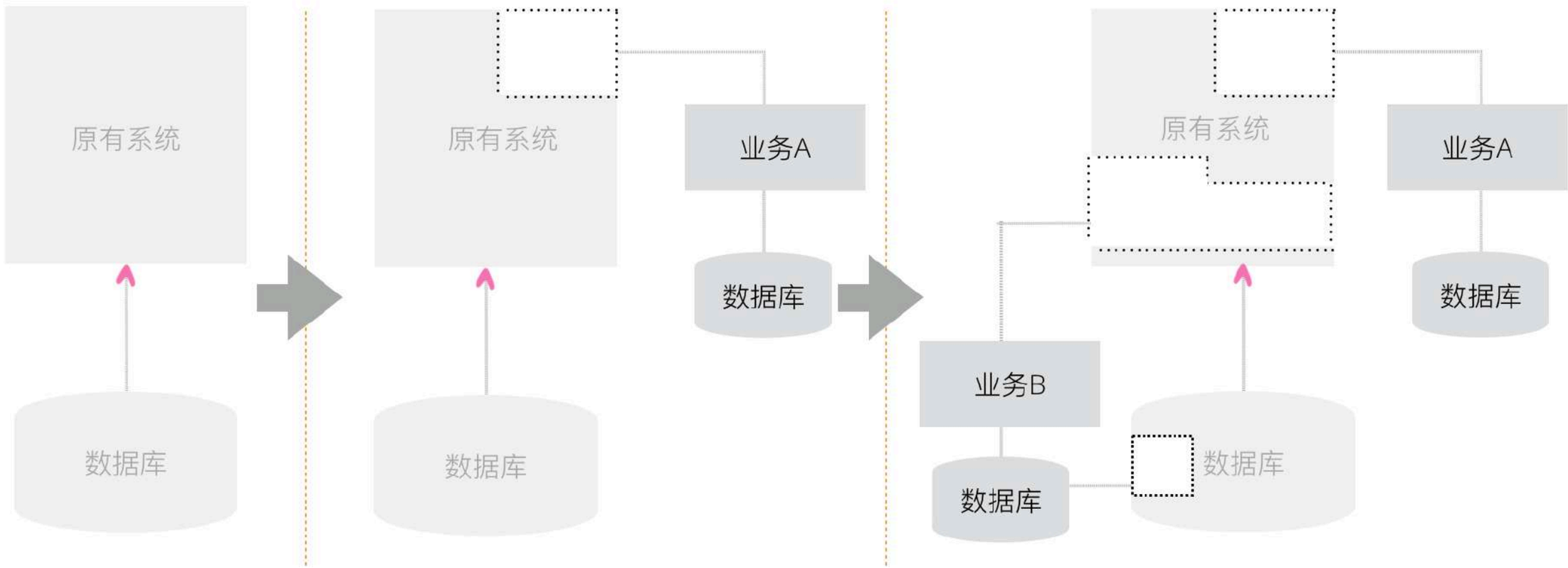
绞杀者模式

- “绞杀者模式”在既有系统资产的基础上实现数字*IT*创新，面对创新的数字*IT*业务更加灵活。
- 通过在新的应用中实现新特性，保持和现有系统的松耦合，仅在必要时将功能从原系统中剥离，以此逐步地替换原有系统。



修缮者模式

- “修缮者模式”在既有系统资产的基础上，通过剥离新业务和功能，逐步“释放”现有系统耦合度，解决遗留系统质量不稳定和*Bug*多的问题。实现传统*IT*性能提升，面对传统的*IT*业务更加稳定灵活，降低维护成本。
- 修缮模式适用于需求变更频率不高的存量系统



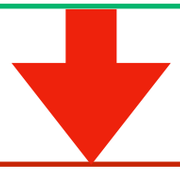
面对巨大复杂的遗留系统，我们该如何开始拆解？

代码依赖模式

我们推荐以模块(java包)为基本单位，从代码依赖的角度看，有三种模式：

依赖其他模块

```
package A
class X {
  public void foo(){
    Y.bar();
  }
}
```



```
package B
class Y {
  public void bar(){}
}
```

被其他模块依赖

```
package A
class X {
  public void foo(){}
}
```

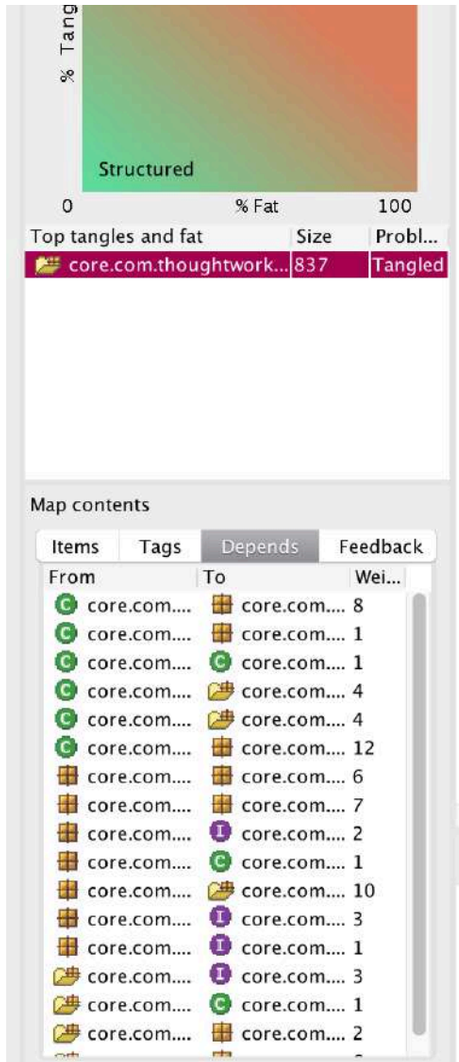
```
package B
class Y {
  public void bar(){
    X.foo();
  }
}
```

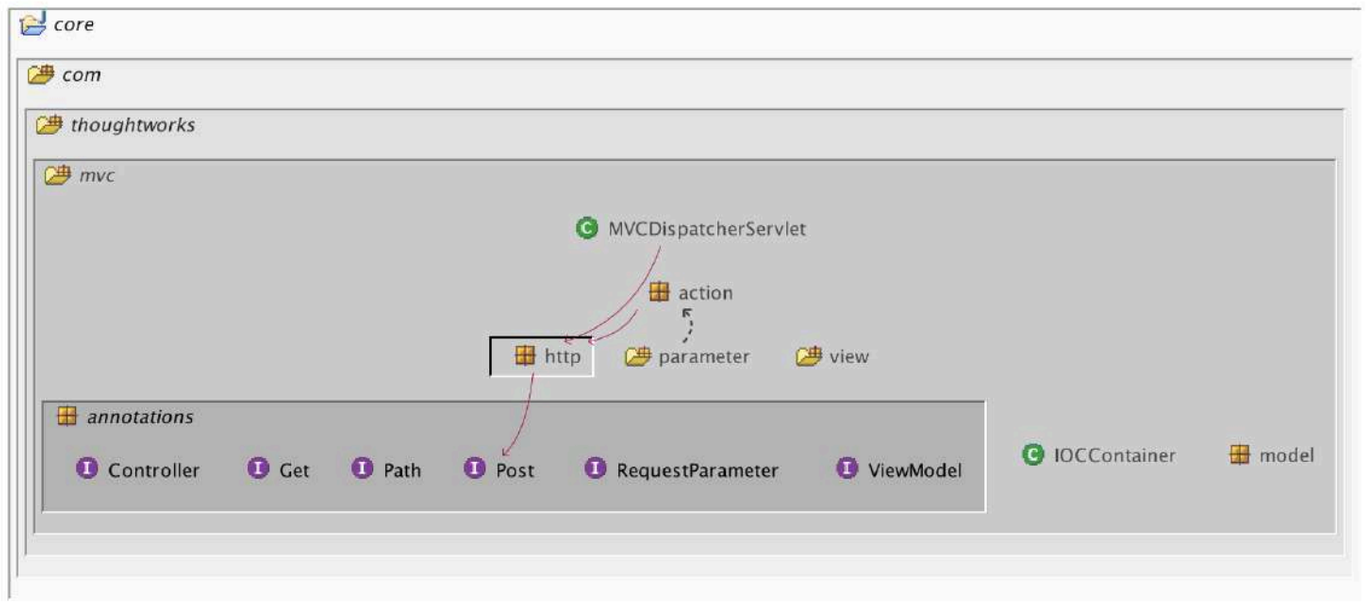
独立存在

```
package A
class X {
  public void foo(){}
}
```

Structure101代码依赖分析

可视化代码地图





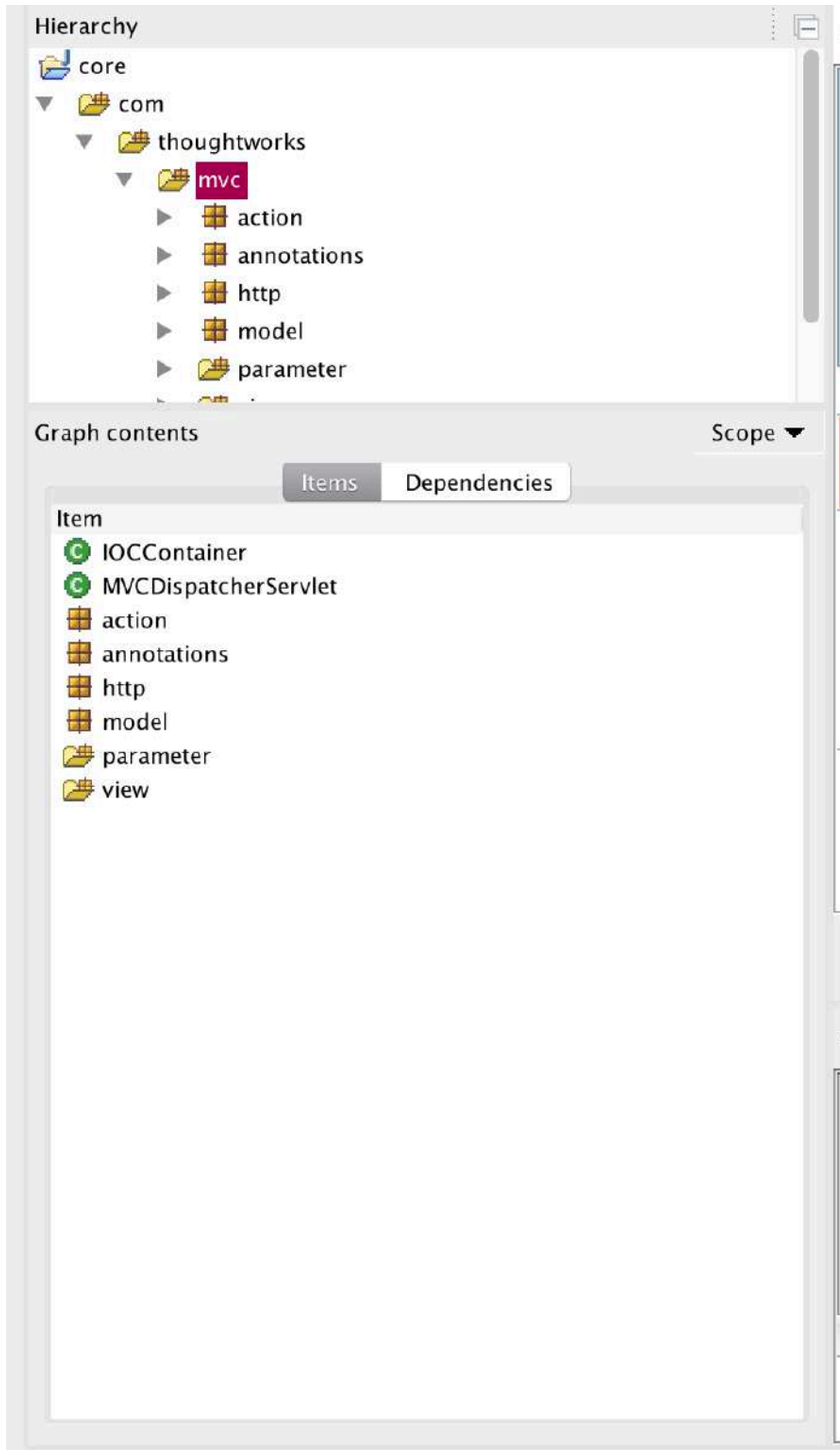
Dependency breakout

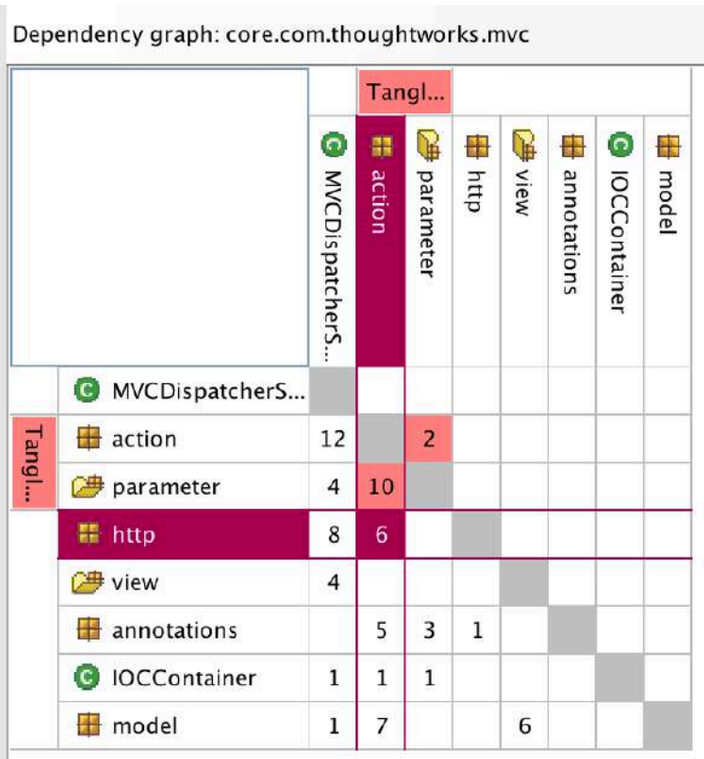
Select a dependency in the graph (or a cell in the matrix) to view its breakout

core.com.thoughtworks.mvc.http

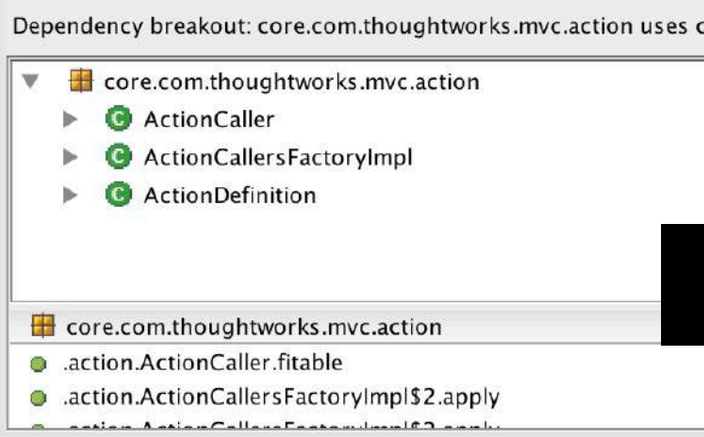
Excluded from project: 0 Excluded from XS: 0 Transformed classes: 0

Structure101代码依赖分析





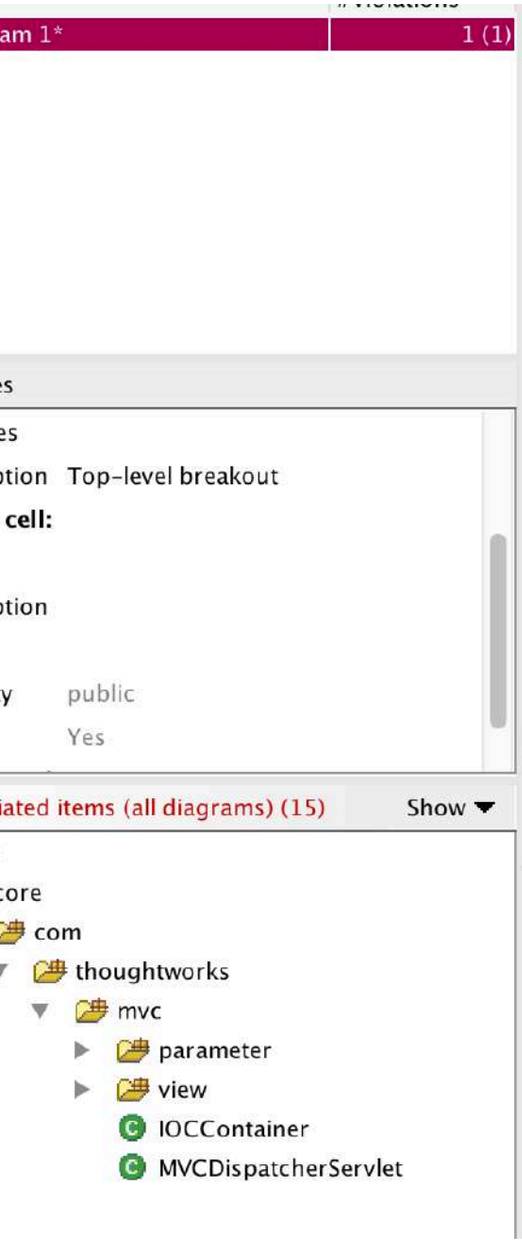
自动分析每一层级包/类之间的依赖生成可视化表格



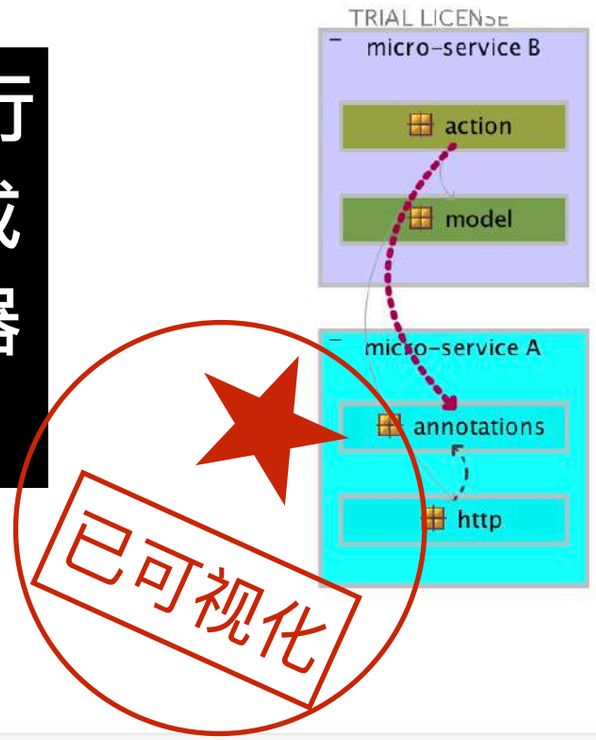
具体依赖细节

Structure101代码依赖分析

子服务定义画布



可以将包/类进行自由组合，形成容器，查看容器之间的依赖

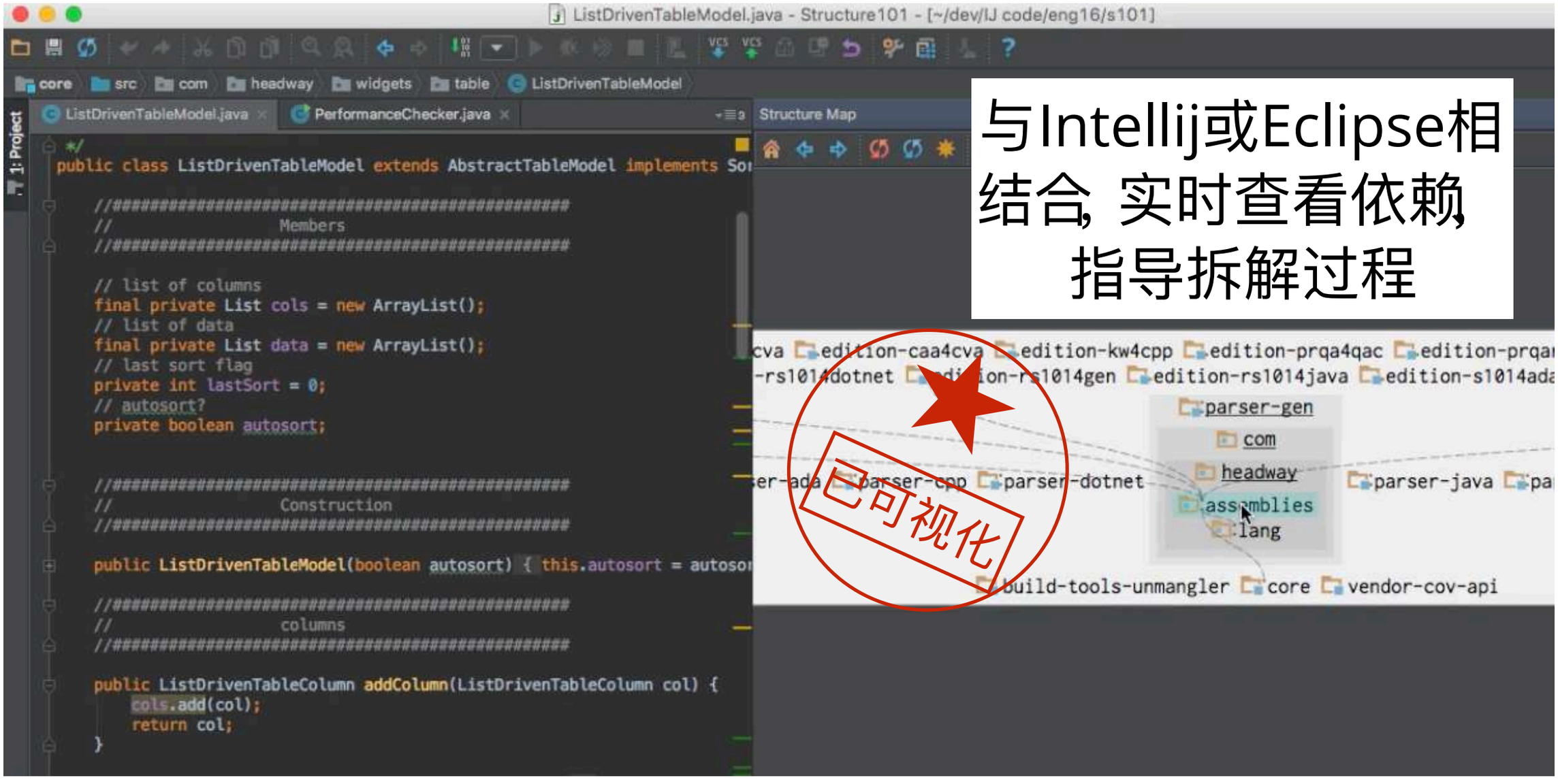


服务A

服务B

具体依赖细节

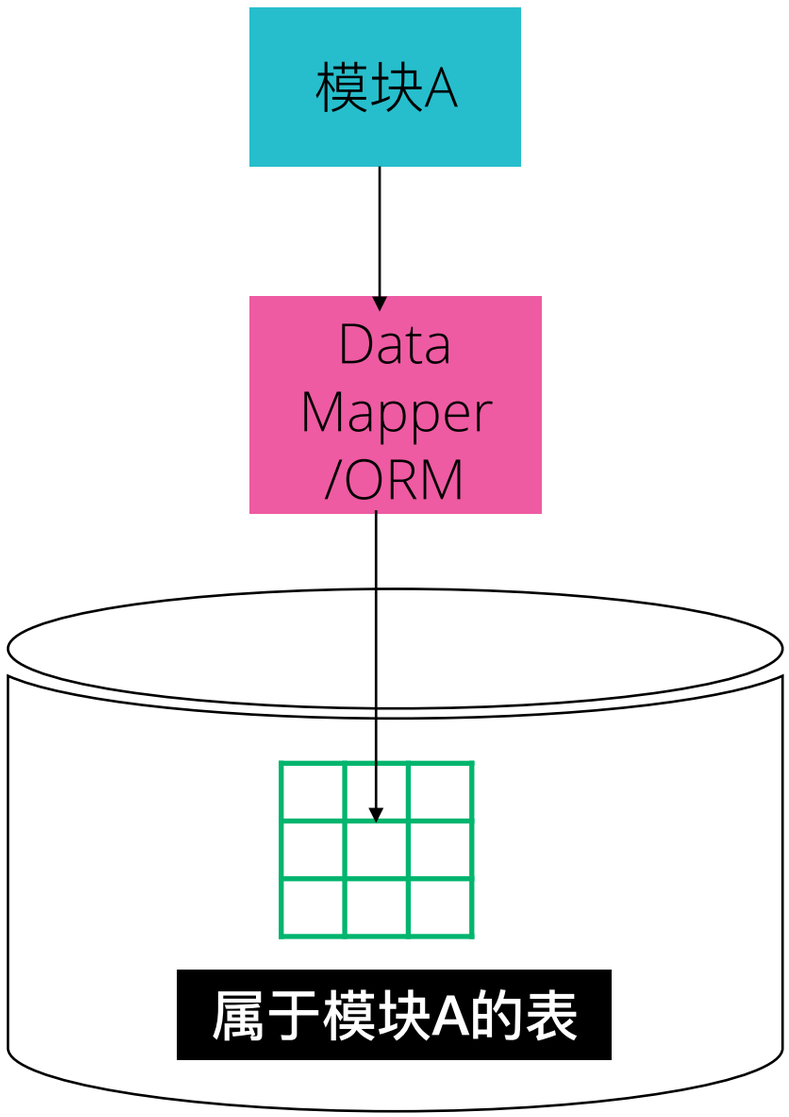
Structure101代码依赖分析

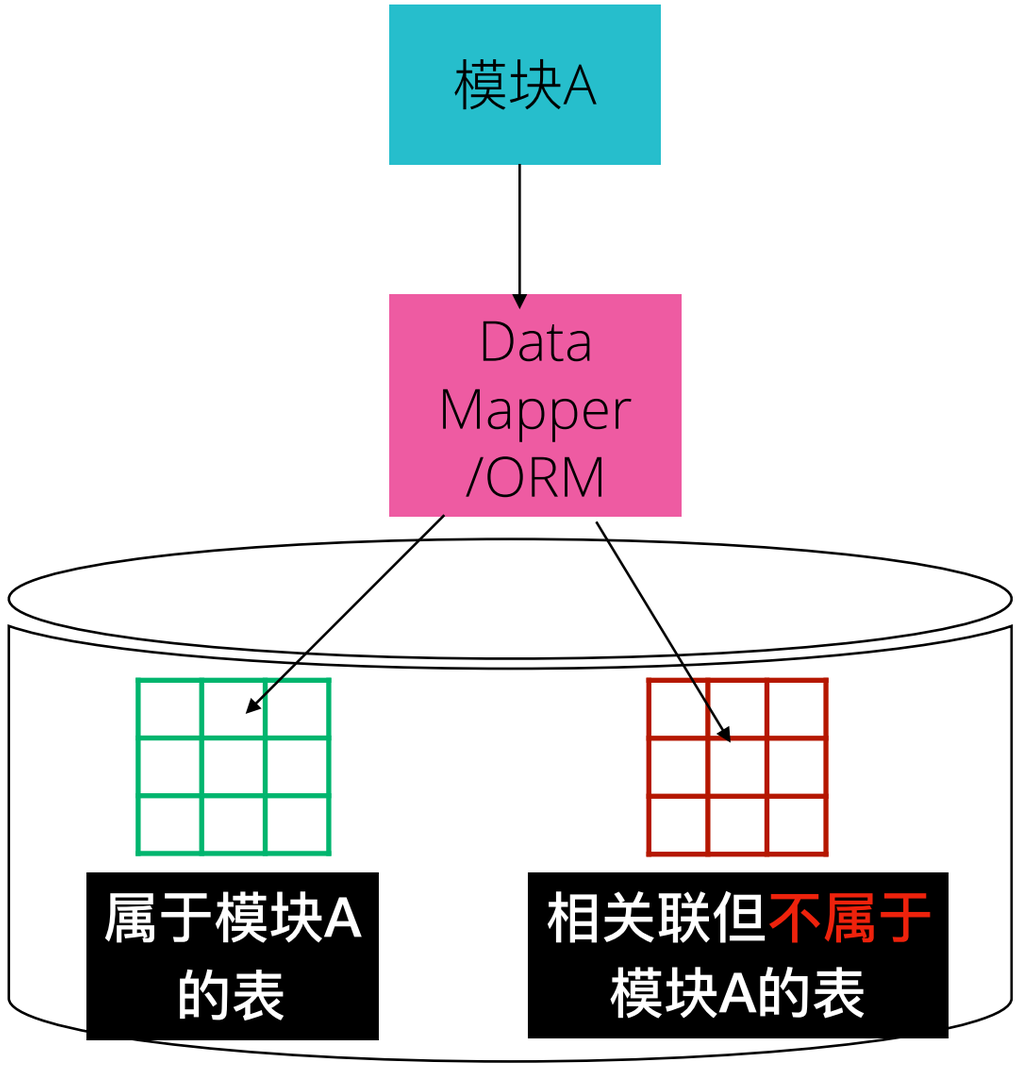


与Intellij或Eclipse相结合，实时查看依赖，指导拆解过程

数据库依赖模式

以模块(java包)为基本单位，从数据库依赖的角度看，有两种模式：





扫描数据库依赖

UserMapper.java

```java
package com.sivalabs.mybatisdemo.mappers;

import java.util.List;
import com.sivalabs.mybatisdemo.domain.User;

public interface UserMapper
{

 public void insertUser(User user);

 public User getUserById(Integer userId);

 public List<User> getAllUsers();

 public void updateUser(User user);

 public void deleteUser(Integer userId);

}
```

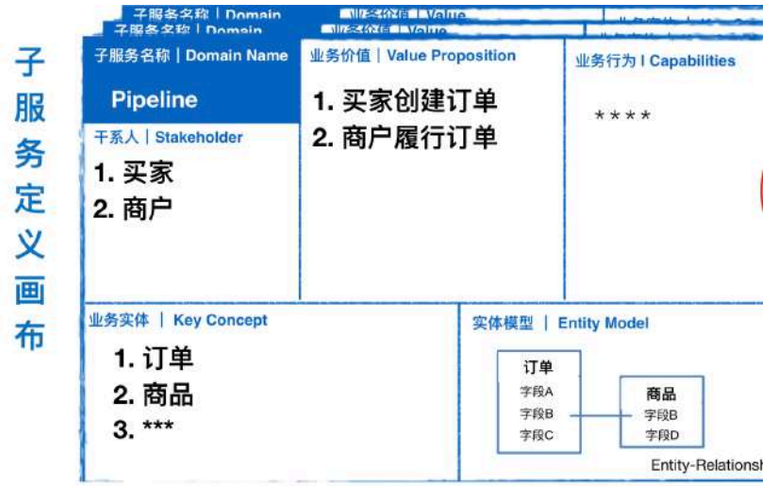
JAVA定义

UserMapper.xml

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE mapper PUBLIC '-//mybatis.org//DTD Mapper 3.0//EN'
  'http://mybatis.org/dtd/mybatis-3-mapper.dtd'>

<mapper namespace='com.sivalabs.mybatisdemo.mappers.UserMapper'

  <select id='getUserById' parameterType='int'
resultType='com.sivalabs.mybatisdemo.domain.User'>
      SELECT
       user_id as userId,
       email_id as emailId ,
       password,
       first_name as firstName,
       last_name as lastName
      FROM USER
      WHERE USER_ID = #{userId}
  </select>
  <!-- Instead of referencing Fully Qualified Class Names we ca
in mybatis-config.xml and use Alias names. -->
   <resultMap type='User' id='UserResult'>
    <id property='userId' column='user_id'/>
    <result property='emailId' column='email_id'/>
    <result property='password' column='password'/>
    <result property='firstName' column='first_name'/>
    <result property='lastName' column='last_name'/>
   </resultMap>

  <select id='getA                    tResult'>
   SELECT * FROM U
  </select>
```

XML实现

扫描数据库依赖



UserMapper.xml

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE mapper PUBLIC '-//mybatis.org//DTD Mapper 3.0//EN'
  'http://mybatis.org/dtd/mybatis-3-mapper.dtd'>

<mapper namespace='com.sivalabs.mybatisdemo.mappers.UserMapper'>

  <select id='getUserById' parameterType='int'
resultType='com.sivalabs.mybatisdemo.domain.User'>
      SELECT
       user_id as userId,
       email_id as emailId ,
       password,
       first_name as firstName,
       last_name as lastName
      FROM USER
      WHERE USER_ID = #{userId}
  </select>
  <!-- Instead of referencing Fully Qualified Class Names we can r
in mybatis-config.xml and use Alias names. -->
    <resultMap type='User' id='UserResult'>
     <id property='userId' column='user_id'/>
     <result property='emailId' column='email_id'/>
     <result property='password' column='password'/>
     <result property='firstName' column='first_name'/>
     <result property='lastName' column='last_name'/>
    </resultMap>

  <select id='getAllUsers' resultMap='UserResult'>
    SELECT * FROM USER
  </select>
```

使用工具扫描

数据库依赖统计表

服务名	mapper名	方法名	正确依赖表名	错误依赖表名
用户服务	com.xxx.UserMapper	getUserById	USER	N/A
用户服务	com.xxx.UserMapper	getUserProducts	User	Product

拆解该从那个服务开始?

拆解后
带来的收益
(业务价值)

- 业务复杂度
- 需求变化频率
- 使用频度

拆解中的
工作量成本
(技术成本)

- 系统集成关系
- 数据迁移量
- 代码改动量

遗留系统拆解评分表

	业务复杂度	需求变化频率	使用频度	系统集成关系	数据迁移量	代码改动量	业务维度总体评分	技术维度总体评分	改造意愿排名
服务A	5	8	8	5	5	8	21	18	3
服务B	8	5	3	8	1	5	16	14	2
服务C	8	5	5	4	5	3	16	13	1
服务D	5	2	3	2	5	3	10	10	4

已可视化

业务维度评分越高，表示越需要微服务化来更快、更好支撑业务需求，改造意义就越高

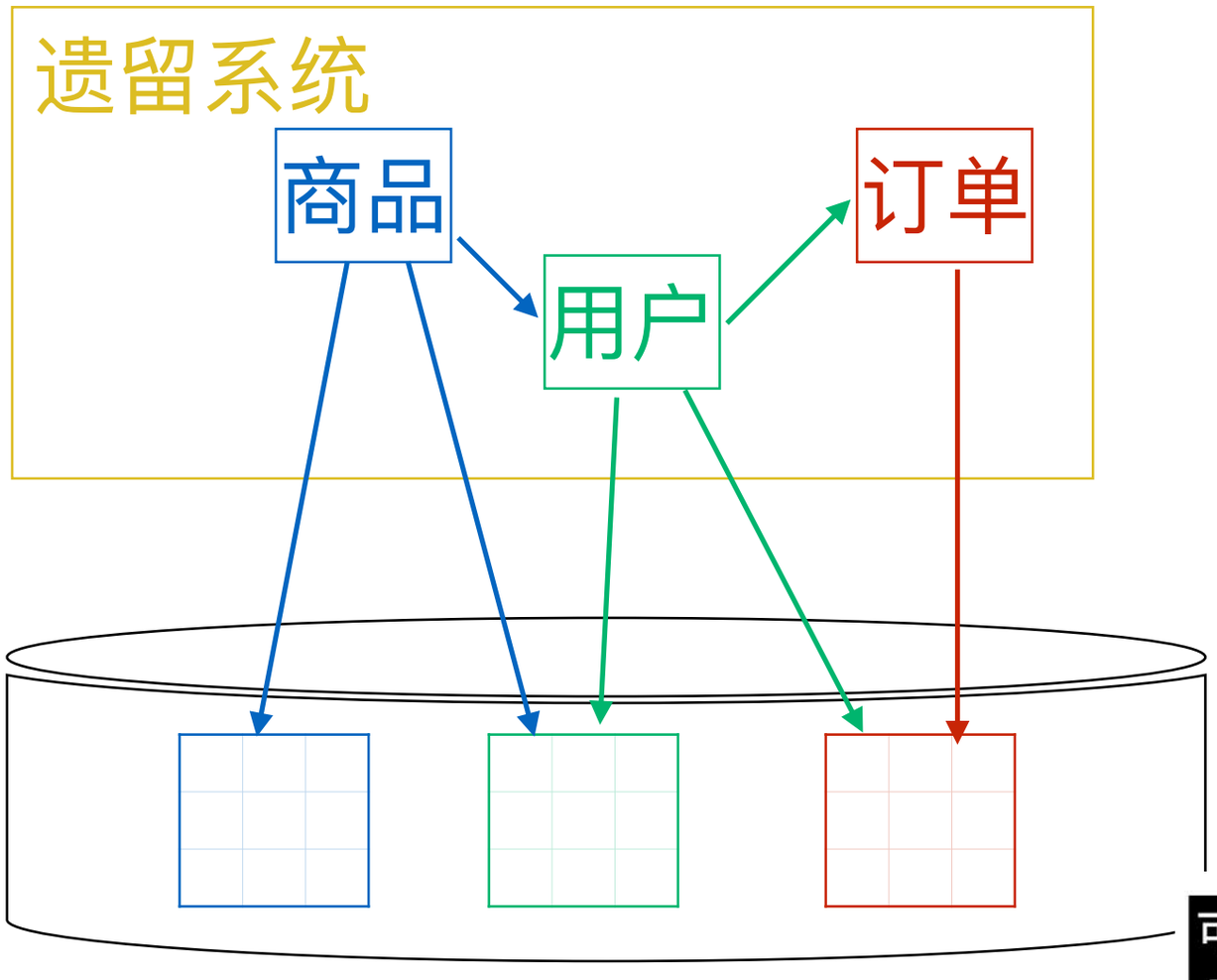
技术维度评分越高，表示技术风险挑战越高，同时反应出微服务化接触的空间越广越深

遗留系统拆解给飞行中的飞机换引擎

友谊第一
比赛第二

图片来源：视觉中国 www.vcg.com

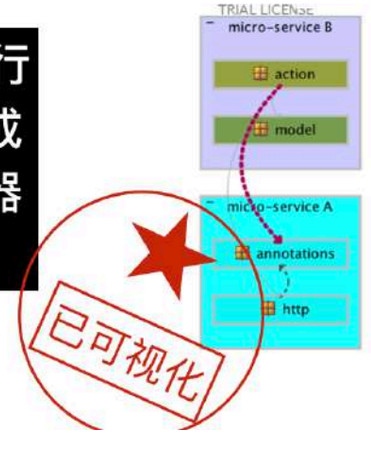
降龙八步第一式



第一式：

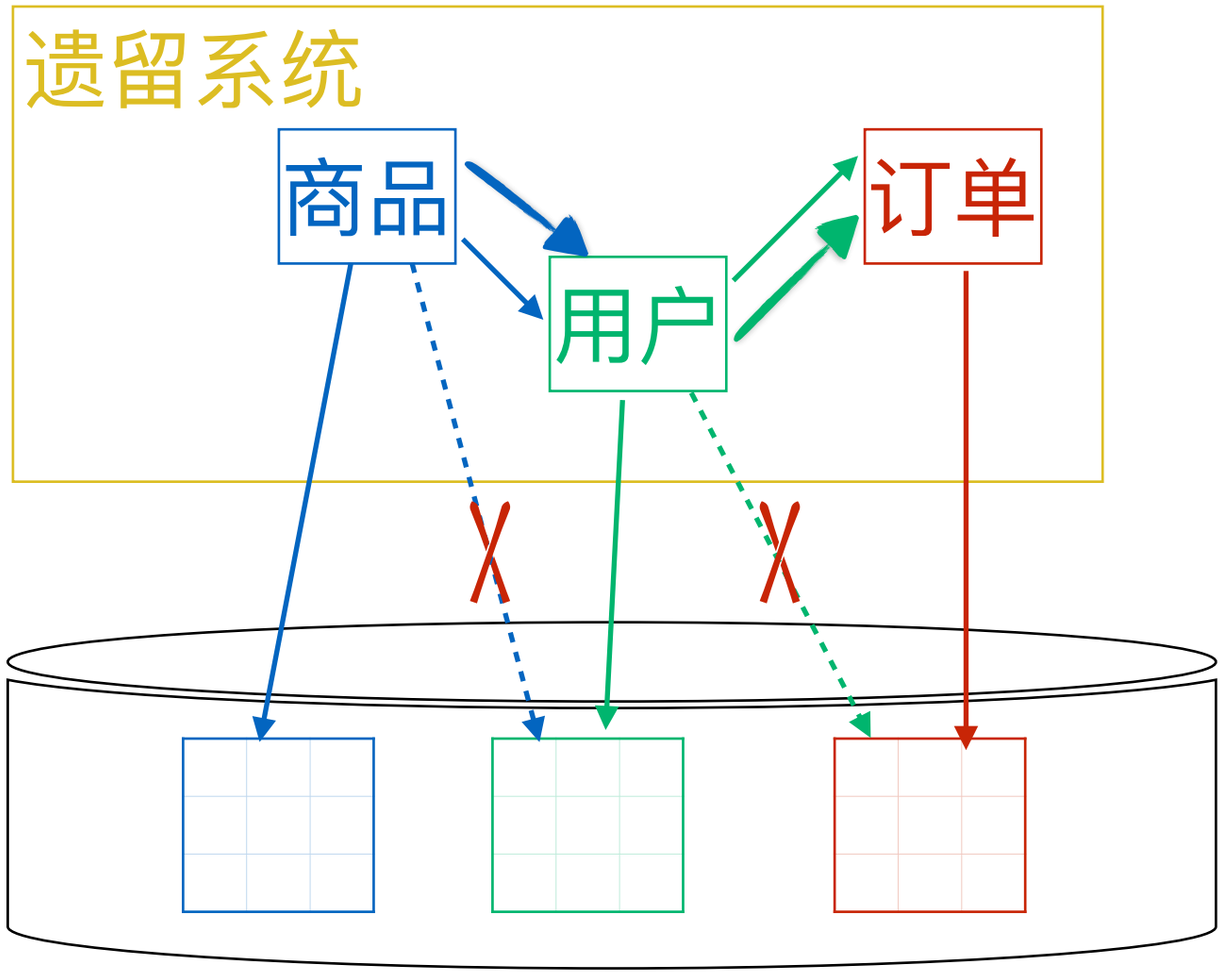
1. 明确要拆解的服务所涉及的模块
2. 验证之前代码依赖分析和数据库分析的结果
3. **约定“同一类型数据只有一处修改”，以数据为中心提高内聚性，同时避免写冲突的出现**

可以将包/类进行自由组合，形成容器，查看容器之间的依赖



服务名	mapper名	方法名	正确依赖表名	错误依赖表名
用户服务	com.xxx.UserMapper	getUserById	USER	N/A
用户服务	com.xxx.UserMapper	getUserProducts	User	Product

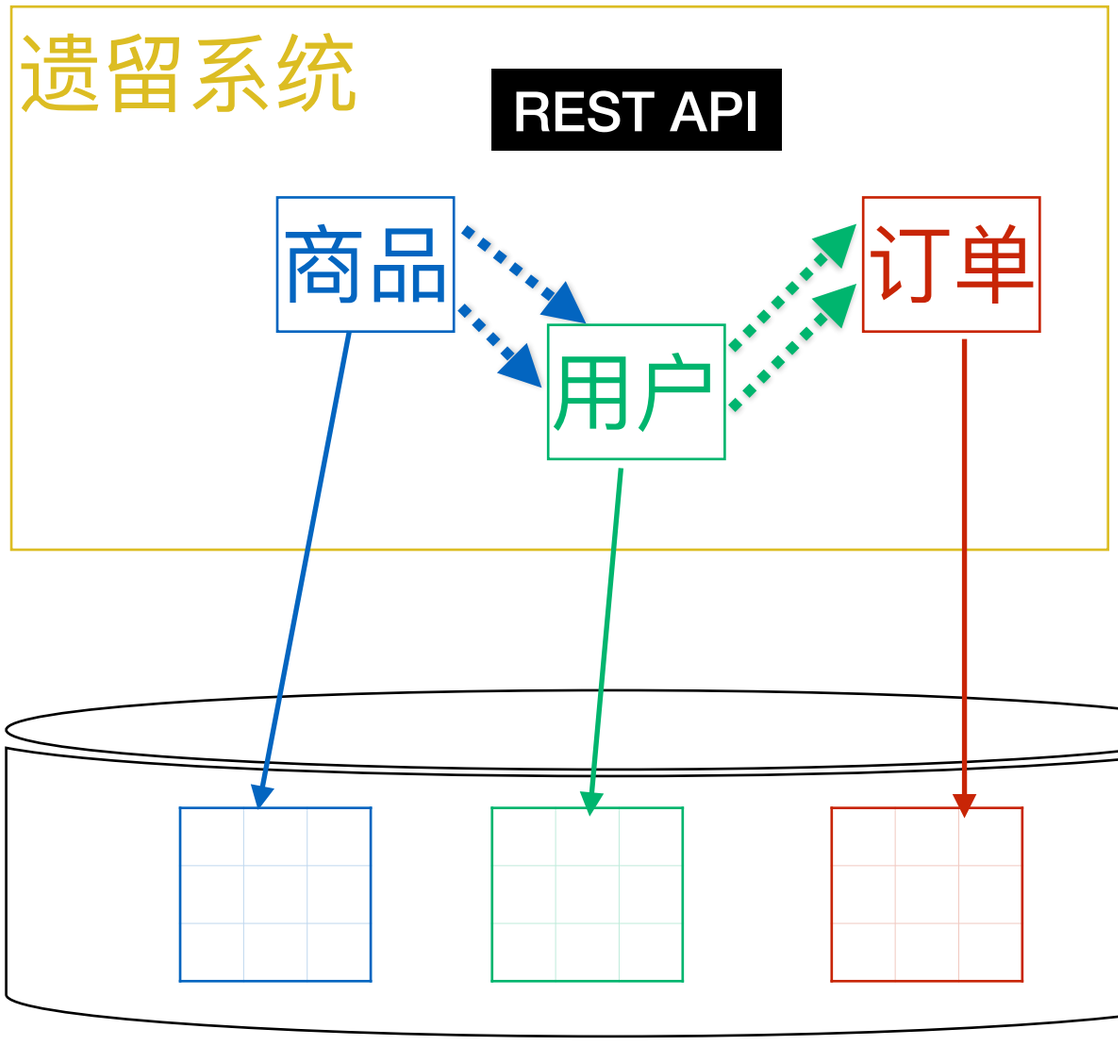
降龙八步第二式



第二式：

1. 在现存订单模块中添加新的方法，提供给用户模块使用，删除用户模块直接使用订单模块的数据表
2. 对用户模块和商品模块做同样的事情
3. 需要在改动点添加自动化测试保证功能正确
4. 将数据库的join推到代码应用层面，势必会比原来慢，需要关注性能问题。考虑添加缓存，数据库索引，甚至调整服务划分来解决

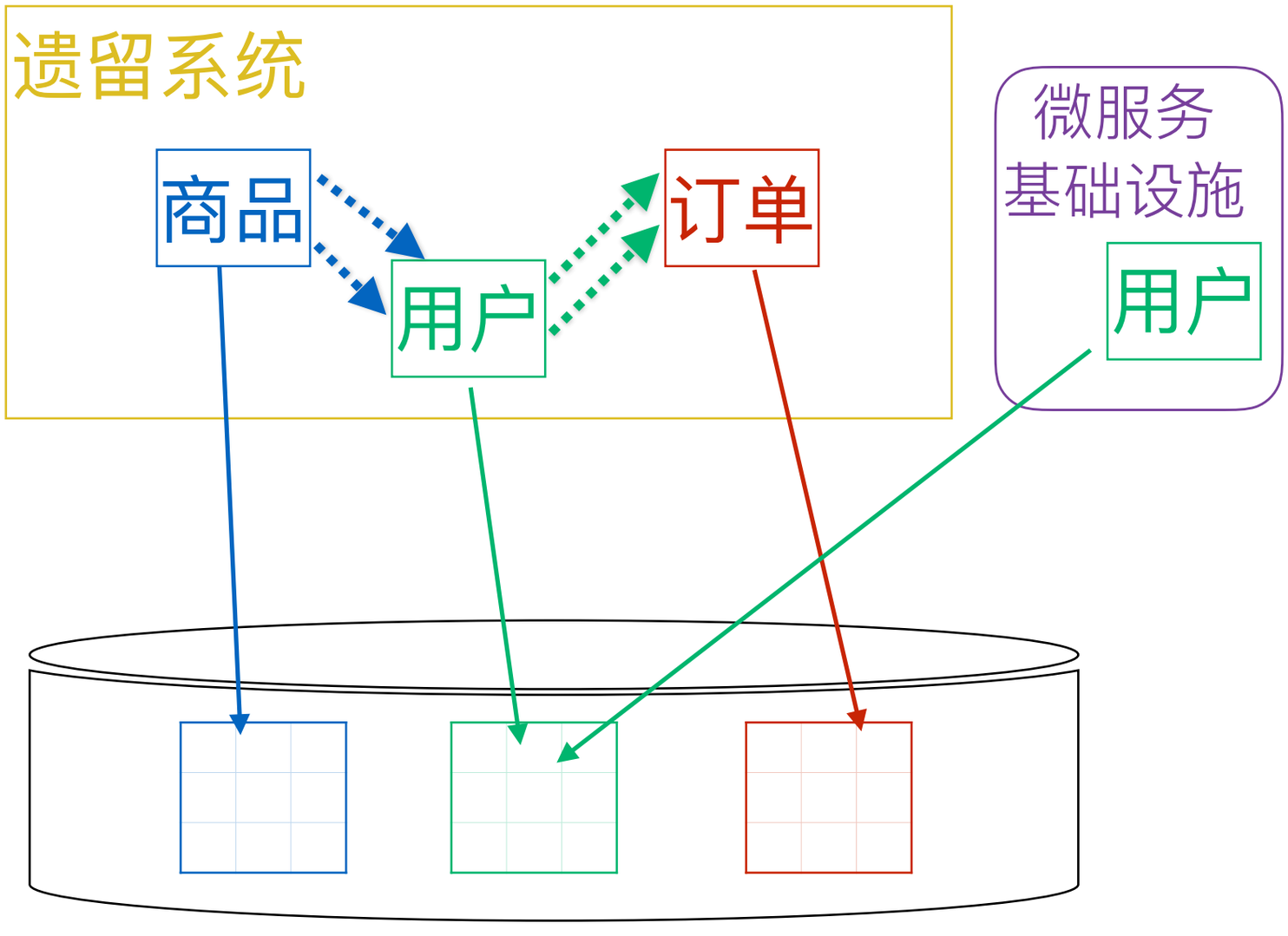
降龙八步第三式



第三式：

1. 在订单模块中添加REST API，提供给用户模块使用，将用户模块对订单模块的代码依赖改成REST API依赖
2. 对用户模块和商品模块做同样的动作
3. 确保所有依赖用户模块，和使用用户模块的依赖点都改成REST API后在进行下一步，否则会导致系统进入中间状态

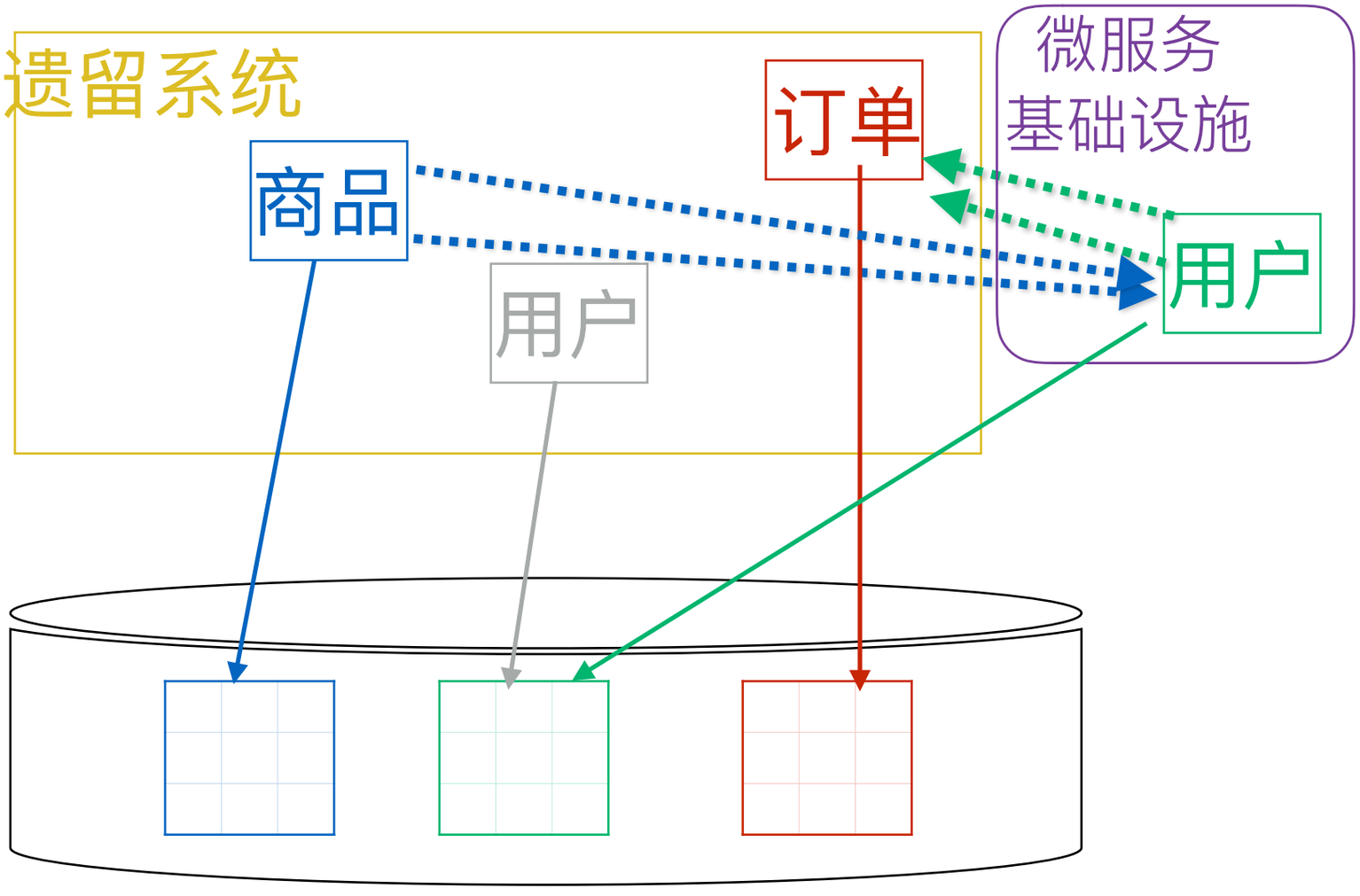
降龙八步第四式



第四式：

1. 将用户模块单独部署成独立运行的用户服务，同时引入微服务基础设施，如服务注册，服务发现，API Gateway等。
2. 确保新的用户服务链接原始数据库

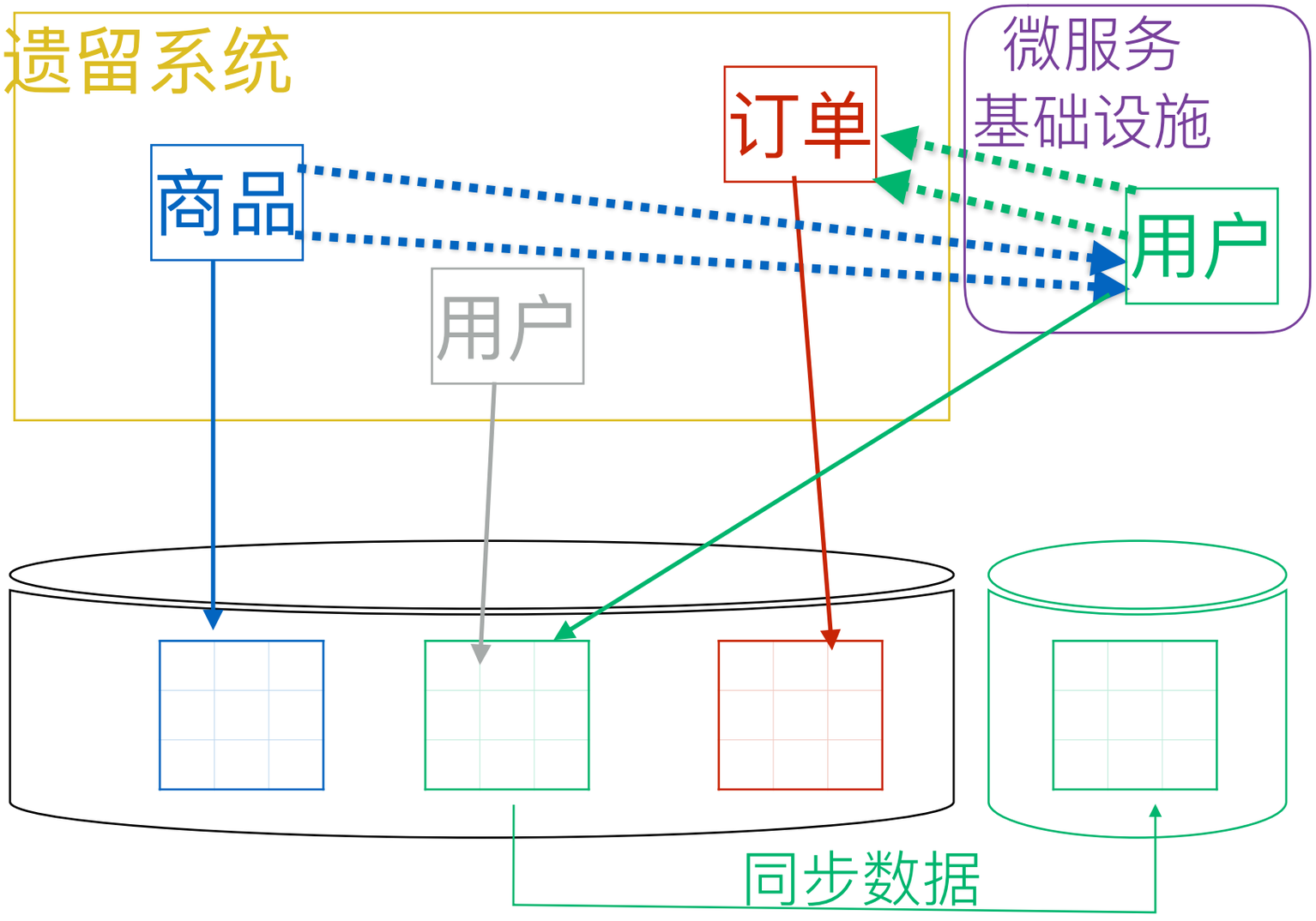
降龙八步第五式



第五式：

1. 修改商品模块，使用新的用户服务，修改用户服务使用订单模块
2. 架空原有用户模块
3. 出现问题随时切换回原始用户模块

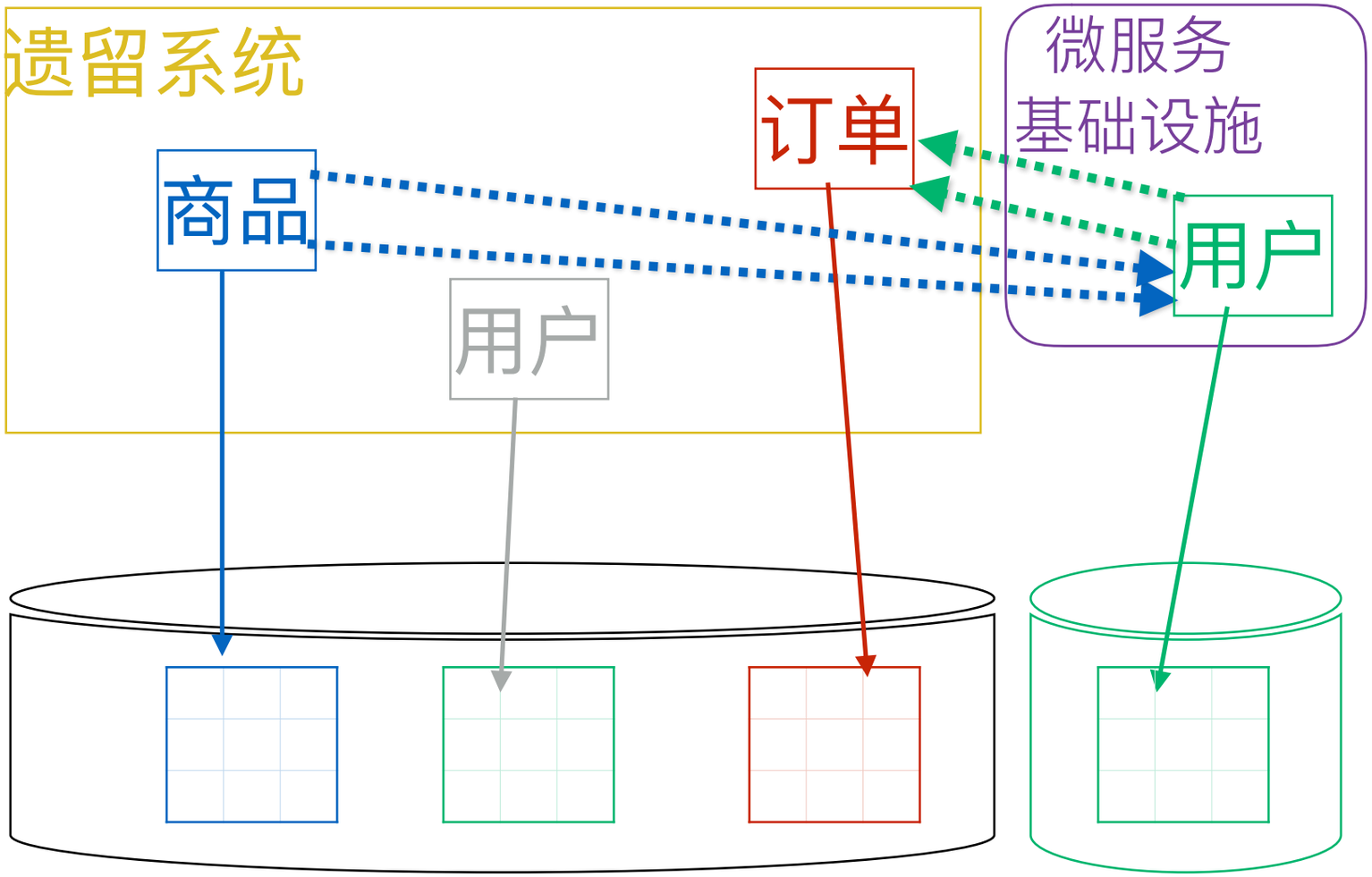
降龙八步第六式



第六式：

1. 给用户服务创建数据库，从原始数据库中同步用户服务的数据

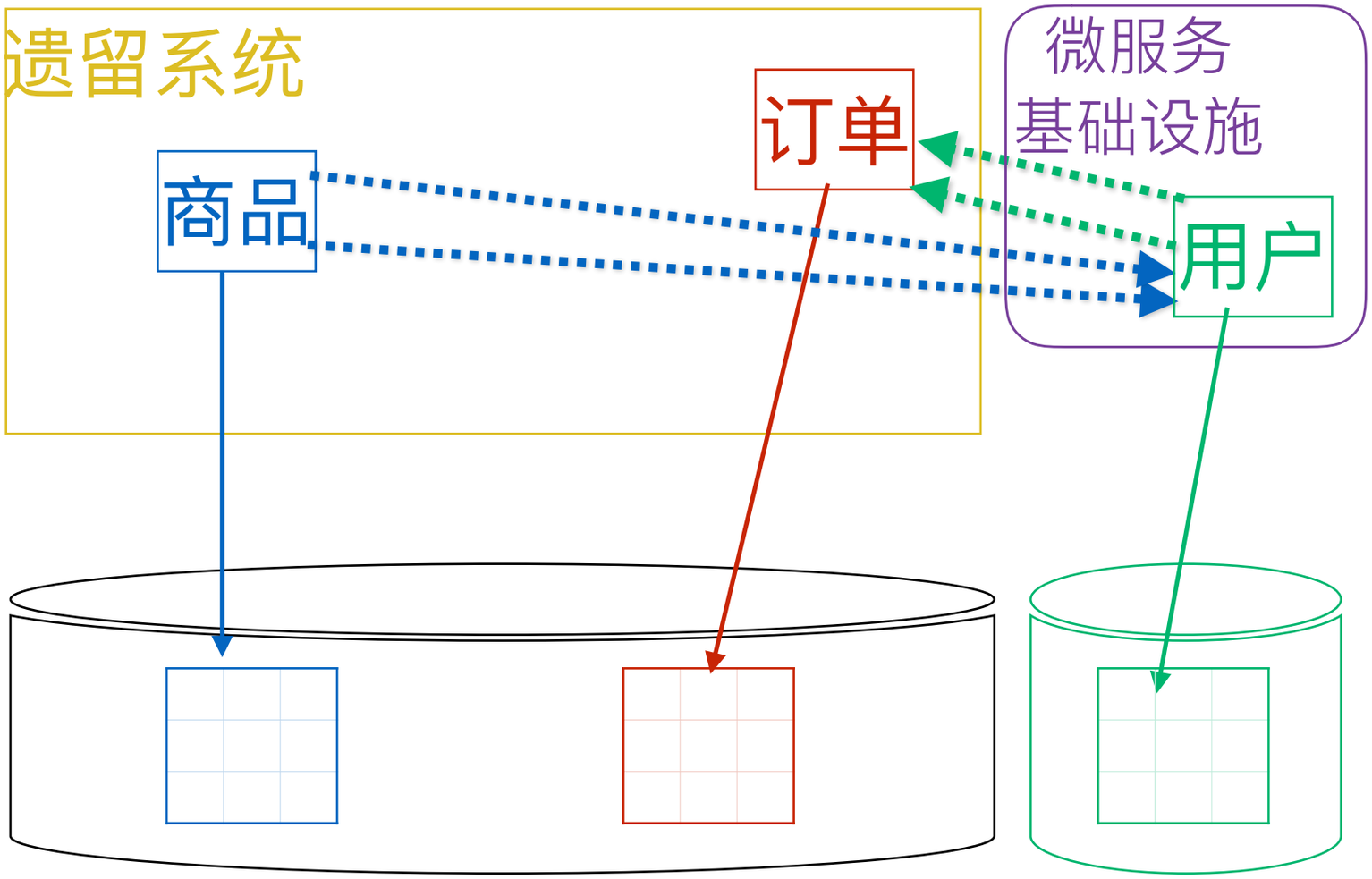
降龙八步第七式



第七式：

1. 配置用户服务，使用用户服务数据库

降龙八步第八式



第八式：

1. 新用户服务运行一段时间后，删除原始用户模块，完成拆解

遗留系统拆解注意事项

1. 遗留系统改造过程中，适当减少新需求的开发，减少改造难度
2. 遗留系统改造过程中，新需求开发必须按照新规则，如只通过REST API进行依赖，同一类型数据只能在某一模块中修改，不依赖其他模块数据库等，减少改造工作量
3. 遗留系统中还会包含前端代码和存储过程，也需要通过代码依赖分析和数据库依赖分析进行拆解
4. 服务间依赖不一定只有REST API，可以通过异步消息、数据冗余等方式，根据实际情况进行选择
5. 部署新微服务后，推荐让新老系统并行一段时间，通过分流让少部分流量进入新微服务，确保改造更加安全

总结

*C4*模型

https://www.jianshu.com/p/1e496225b6b6
https://www.jianshu.com/p/f16aae86713a

微服务架构

https://www.jianshu.com/p/4821a29fa998
https://martinfowler.com/tags/microservices.html

用户画像/用户旅程

https://insights.thoughtworks.cn/redefine-customer-journey/
https://ixdc.org/2016/schedule-articrle.php?id=35

遗留系统拆分

https://martinfowler.com/articles/break-monolith-into-microservices.html
https://martinfowler.com/articles/extract-data-rich-service.html

领域驱动设计

https://insights.thoughtworks.cn/ddd-by-experience/
https://juejin.im/post/59b8d4e06fb9a00a600f4c6e
http://www.ddd-china.com/

重构

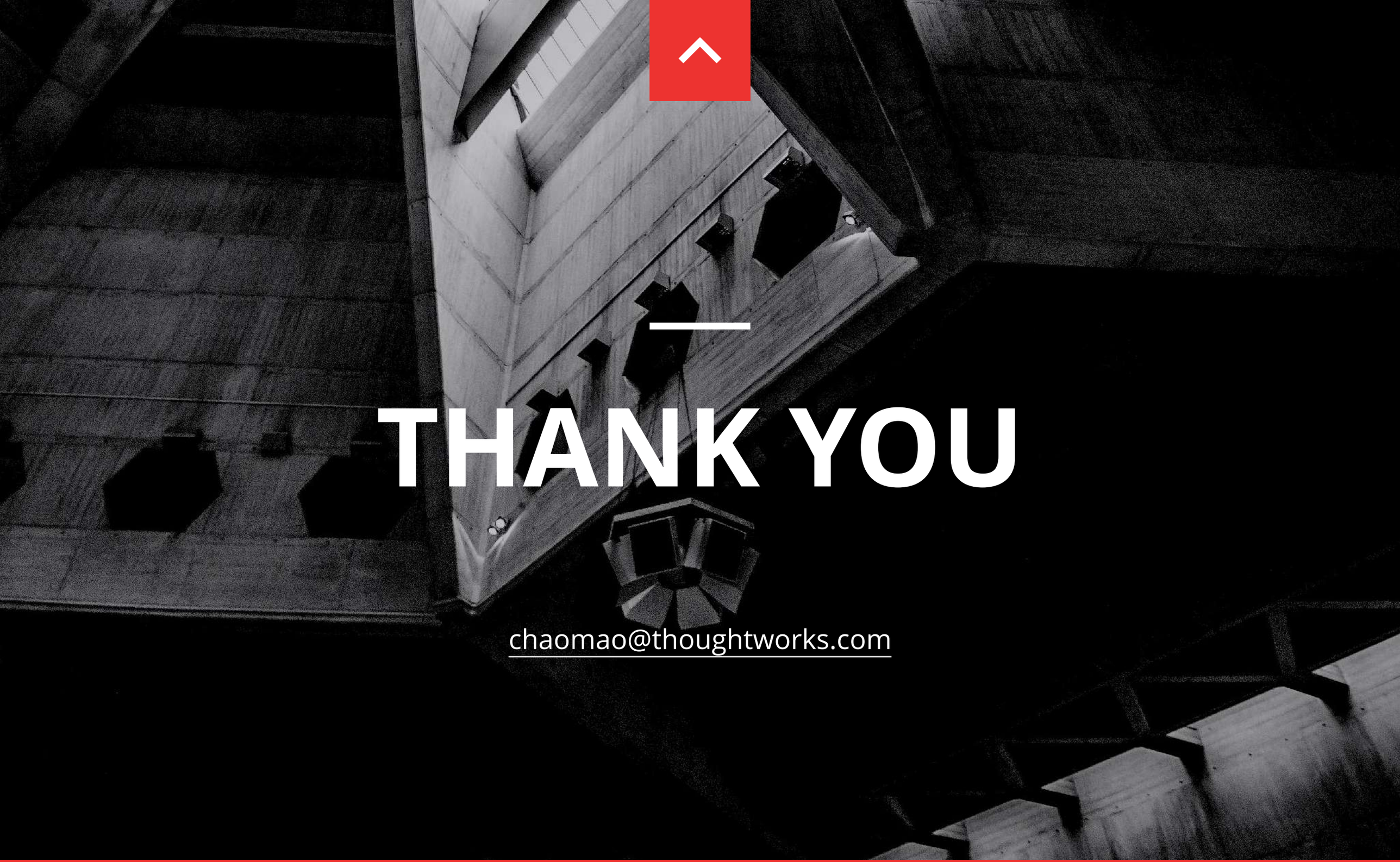
https://martinfowler.com/tags/refactoring.html
https://book.douban.com/subject/30333919/

事件风暴工作坊

https://www.jianshu.com/p/eadbec49fbbc
https://www.eventstorming.com/

Structure 101

http://structure101.com



THANK YOU

chaomao@thoughtworks.com

DDD CHINA