Spring Cloud微服务实践

链家网基础架构部 刘思贤

个人简介

- 刘思贤

- 曾在金山、新浪微博
- 前爱油科技架构师
- 现就职链家网基础架构部

传统业务如何微服务化...

爱油科技的故事

回到2015年

业务的变迁

- 一个网站打遍天下
- 线上收单线下处理
- 系统管理员用后台
- 网站、App、微信
- 业务流程全搬至线上
- 业务人员用后台

研发犯难

- 需求变化侵蚀架构，像熔化的反应堆芯
- 被技术平台绑架，大炮打蚊子
- 协作困难，牵一发而动全身

“架构就是确定系统的职责边界”

微服务架构

- 架构分层和基本原则
- Spring Cloud快速搭建微服务框架
- 领域驱动设计方法指导业务服务开发
- 基础设施和DevOps的支撑

微服务架构

分层和基本原则

期望目标

- 不绑定到特定的语言和框架
- 松散耦合，便于多种语言框架的集成
- 简单、容易落地、方便扩展

架构分层


接入层
Rune
业务服务层
Epic
基础设施层
Infrastructure
H5
PC Web
微信 / 钉钉
APP
Mobile API
API网关服务
成员服务
工作流服务
通知服务
商品库服务
订单服务
X业务服务
Y业务服务
Z业务服务
计算
RDB
消息服务
日志服务
网络
NoSQL
对象存储
容器服务

原则

- 业务服务层服务完全对等
- 业务服务层服务必须是无状态的
- 接入层服务之间禁止相互调用
- 接入层服务不能包含业务逻辑
- 业务接口必须是RESTful风格
- 所有服务必须运行在容器里

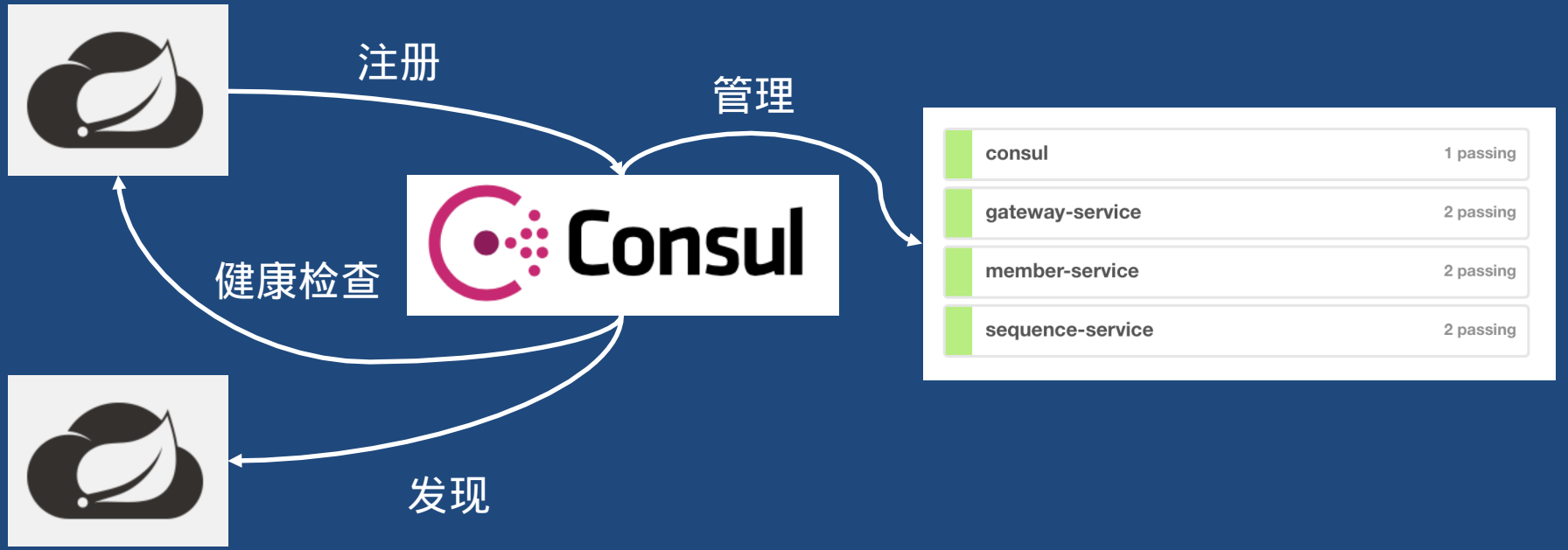
微服务架构

SPRING CLOUD快速搭建微服务框架

服务注册和发现

- Consul
- Spring Cloud Consul

服务注册和发现



Service　　Consul　　Consul UI

仍待解决

- Consul服务注册发现与七层Nginx联动
- 灰度发布

Consul作为配置中心

- 统一管理Spring Cloud项目的配置文件
- 为接入层服务下发配置

```yaml
spring:
  application:
    name: sequence-service
  cloud:
    consul:
      host: 127.0.0.1
      config:
        format: yaml
```

config/sequence-service/data

```
server:
  port: 8082
```

仍待解决

- 配置的版本化管理
- 灰度发布

服务集成

服务集成

Feign → Hystrix → Ribbon → Consul

RESTful自封装接口

```
@RequestMapping(value = "{id}", method = RequestMethod.GET, produces = "application/json")
public UserRepresentation findOne(@PathVariable("id") String id) {
    User user = this.userRepository.findByUserId(new UserId(id));

    if (user == null || user.getDele
        throw new NotFoundException(
    }

    return new UserRepresentation(us
}
```

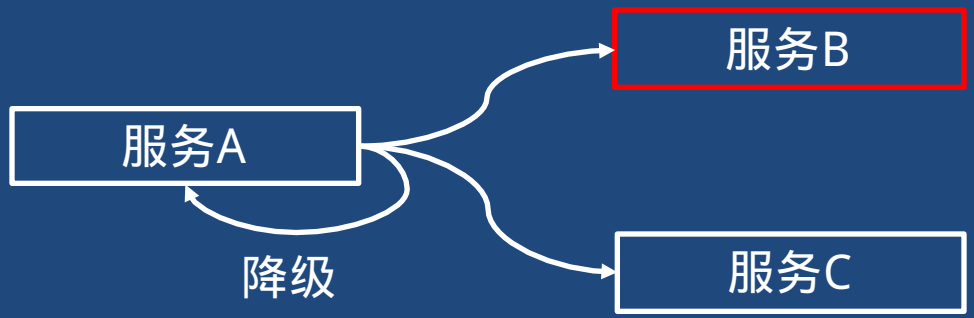
```
@FeignClient("epic-member-microservice")
public interface UserClient extends UserRepository {

    @Override
    @RequestMapping(value = "/users/{id}", method = RequestMethod.GET,
    User findOne(@PathVariable("id") String id);
}
```

服务集成

Feign → Hystrix → Ribbon → Consul

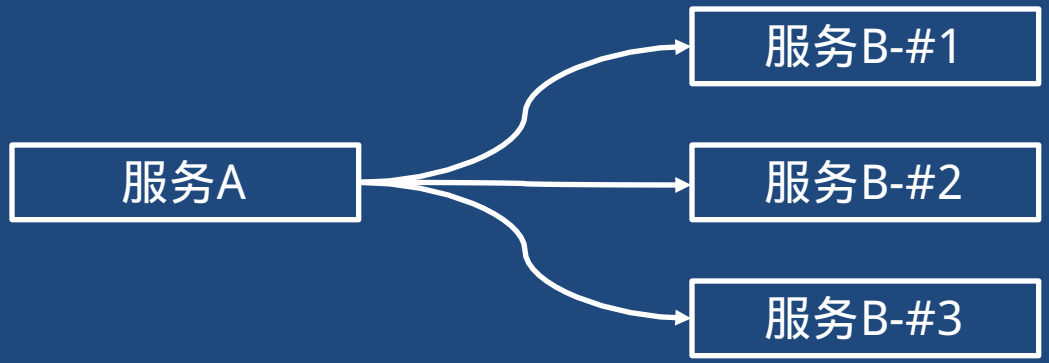
隔离、限流与降级



服务集成

Feign → Hystrix → Ribbon → Consul

负载均衡



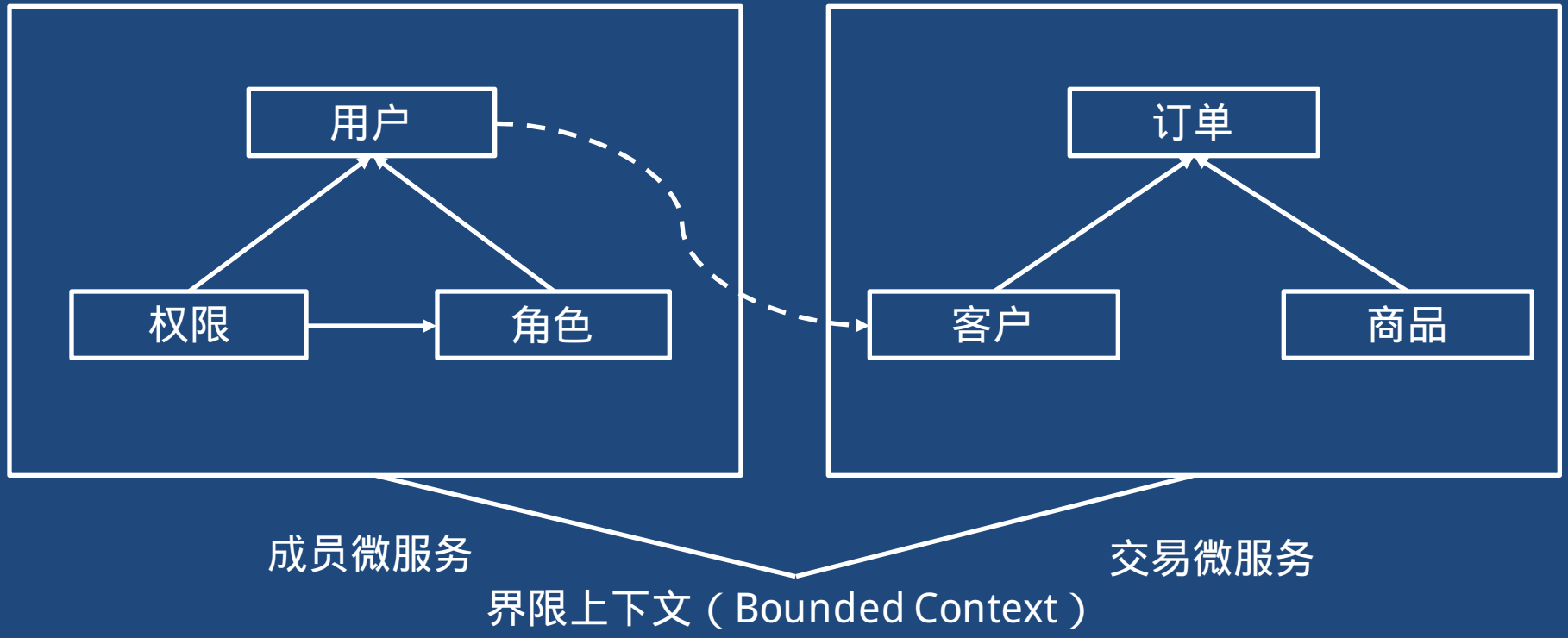
服务集成

- 服务层Node：Consul、Brake
- 接入层Node：request to zuul
- PHP接入层：guzzle to zuul

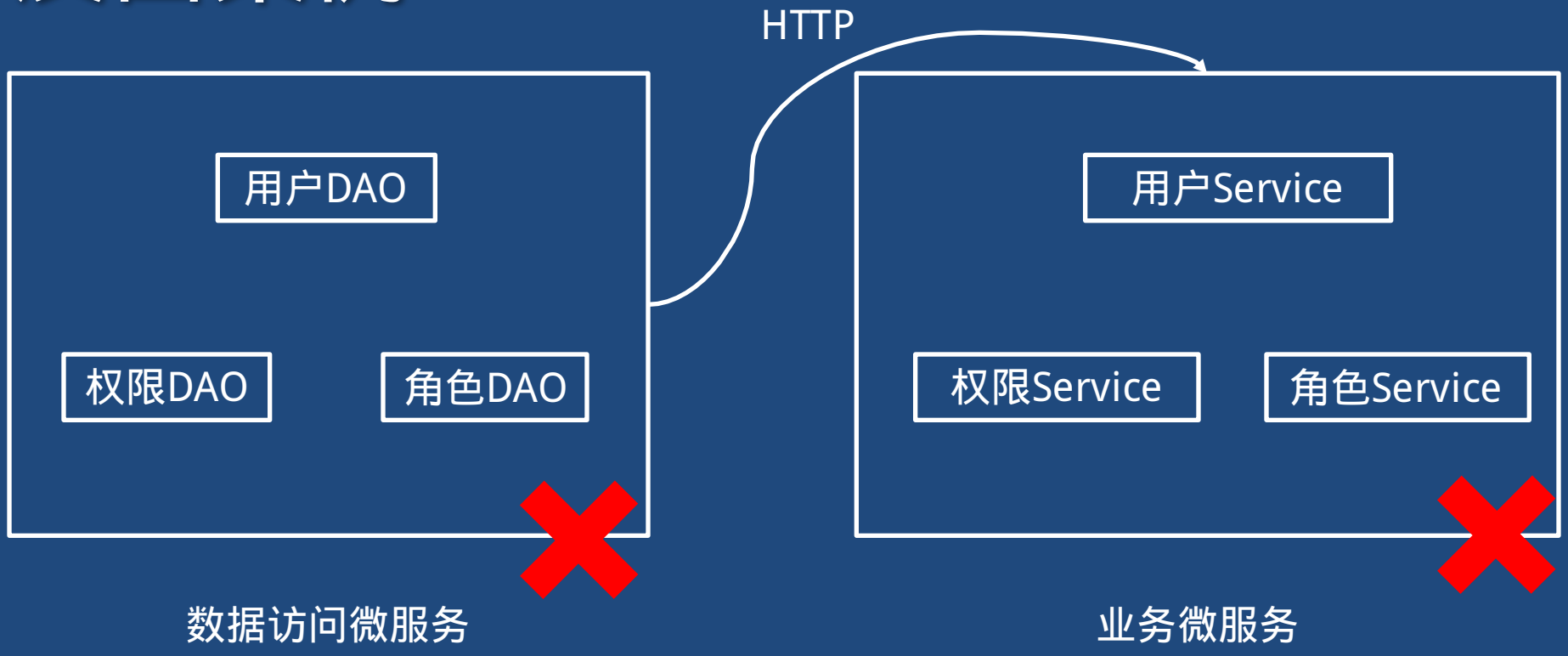
微服务架构

领域驱动设计指导业务服务开发

从业务到微服务



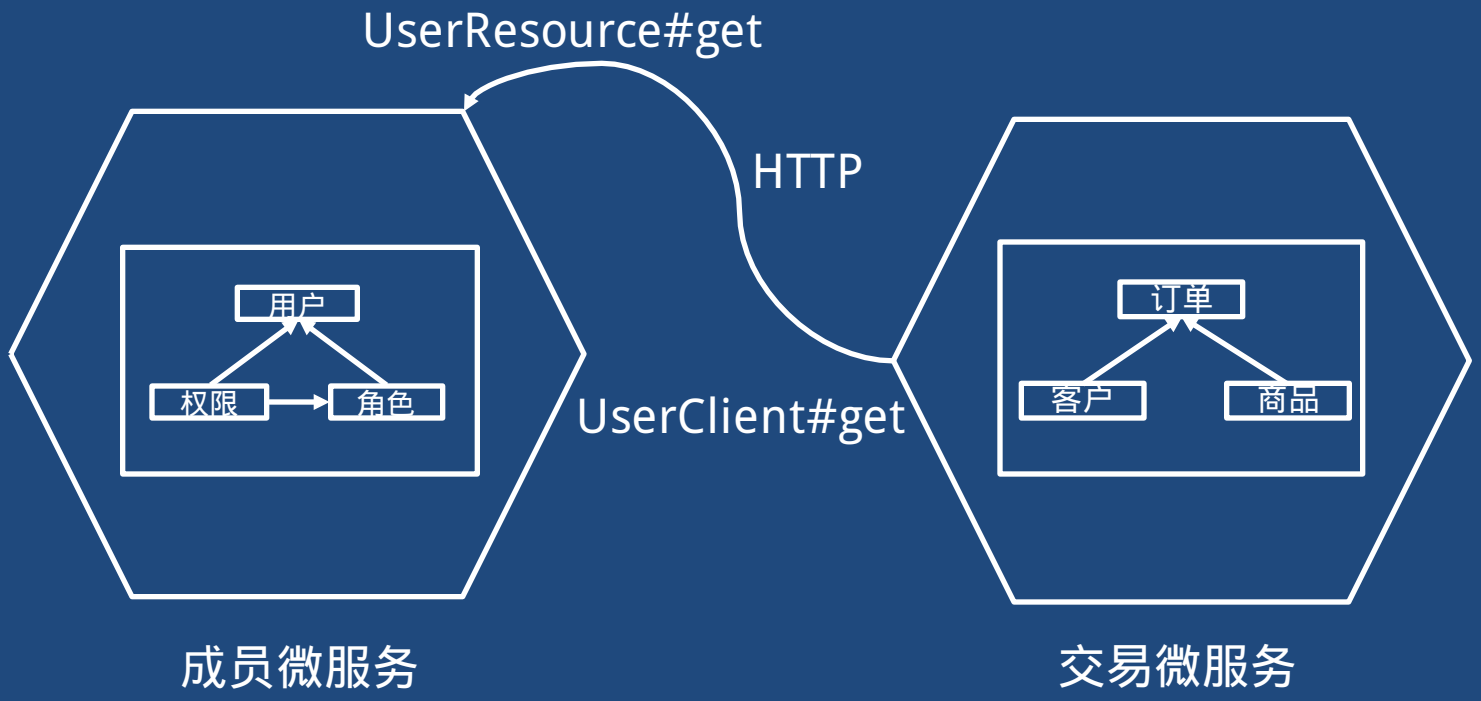
反面案例


HTTP
用户DAO
权限DAO
角色DAO
用户Service
权限Service
角色Service
数据访问微服务
业务微服务

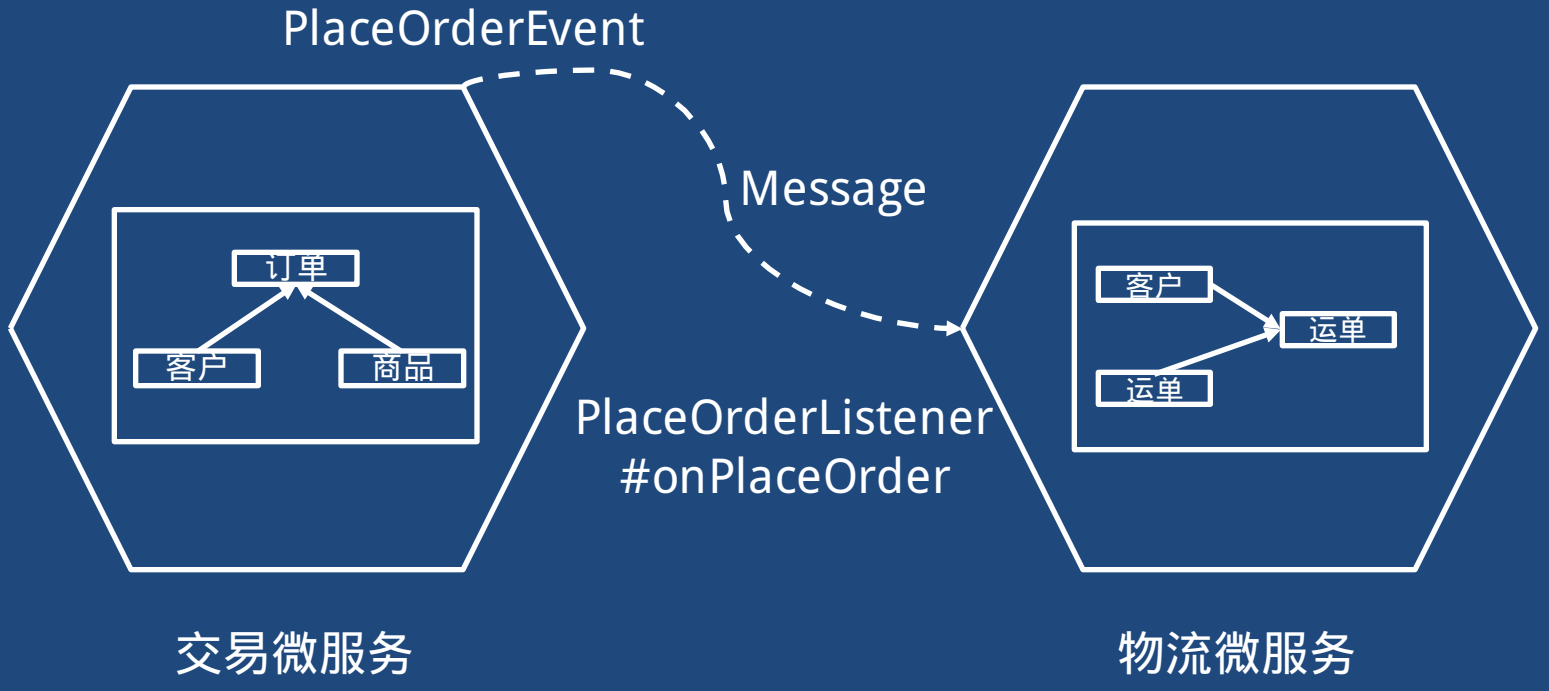
“分布对象设计第一定律：不要分布使用对象。”

——《企业应用架构模式》，Martin Fowler

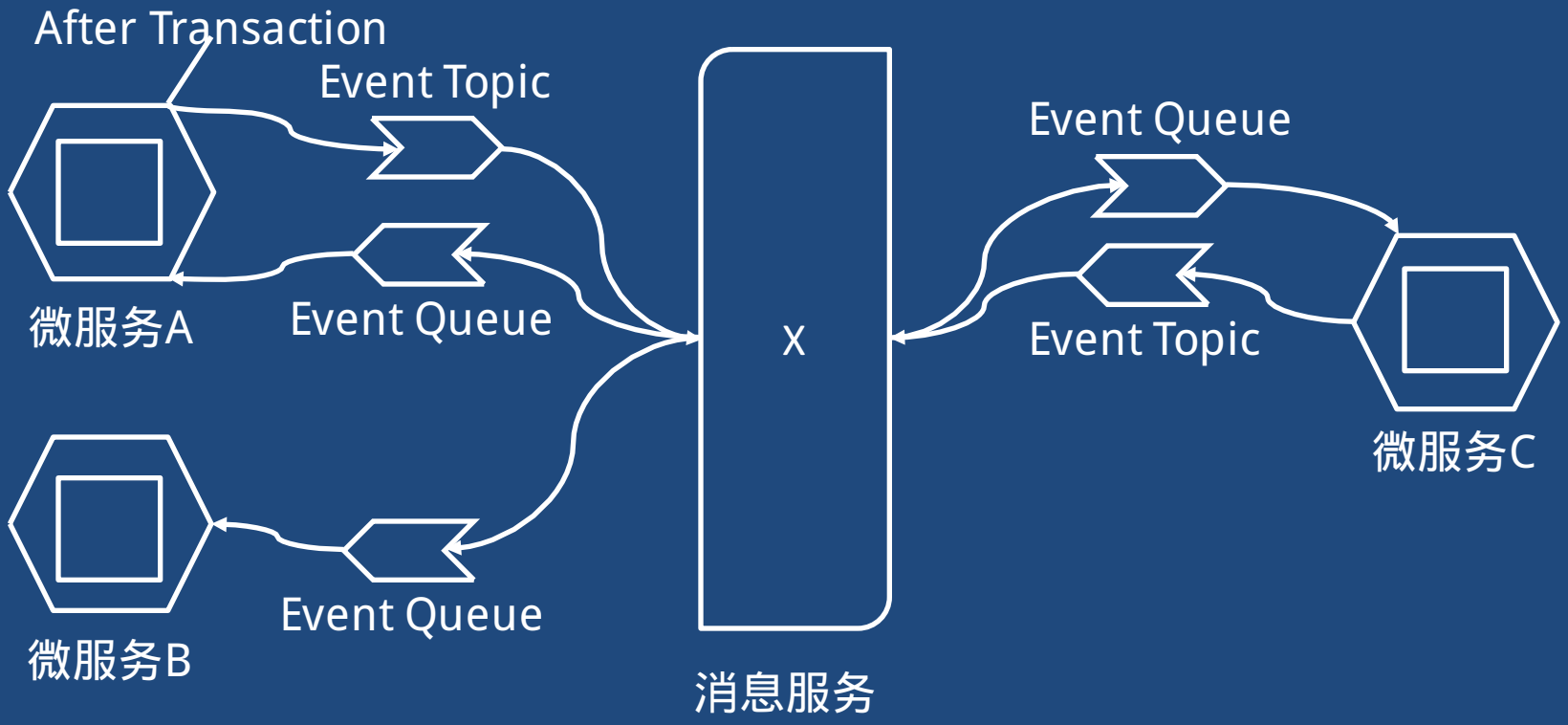
从业务到微服务


UserResource#get
HTTP
用户
权限
角色
UserClient#get
订单
客户
商品
成员微服务
交易微服务

从业务到微服务


PlaceOrderEvent
Message
订单
客户
商品
PlaceOrderListener
#onPlaceOrder
客户
运单
运单
交易微服务
物流微服务

领域事件的传递


After Transaction
Event Topic
Event Queue
微服务A
Event Queue
微服务B
X
消息服务
Event Queue
Event Topic
微服务C

消息服务

主题名称	消息数	消息最大长度(Byte)	消息存活时间(秒)	开启logging	操作
prod-epic-member-domain...	0	65536	86400	false	配置 | 发布消息 | 删除 | 获取地址 | 订阅详情

队列名称	消息生命周期(秒)	消息延时(秒)	活跃消息数	非活跃消息数	延迟消息数	创建/最后修改时间	开启logging	操作
prod-epic-notification-domain-event	345600	0	0	0	0	2017-05-04 21:38:24 2017-05-04 21:38:24	false	修改设置 | 删除 发送消息 | 接收消息

订阅名称	接收端地址	重试策略	消息推送格式	推送类型	操作
notification-subscribe	acs:mns:cn-beijing:1960064396887970:queu...	BACKOFF_RETRY	SIMPLIFIED	队列	配置 | 删除 | 获取地址

包结构

- com.abc.example
  - controller
    - UserController
  - common
  - dao
    - UserDao
    - UserDaoMySQLImpl
  - entity
    - User
  - service
    - UserService
  - util

- com.abc.example
  - user
    - controller
      - UserController
    - service
      - UserService
    - dao
      - UserDao
      - UserDaoMySQLImpl
    - entity
      - User
  - role

更好的包结构

- com.abc.example
  - application
    - command
    - representation
  - domain
    - User
    - UserService
    - UserRepository
  - port
    - persistence
      - JpaUserRepository
  - resource
    - UserResource

https://github.com/VaughnVernon/IDDD_Samples

更好的包结构

- 分包/模块原则：高内聚、低耦合
- 领域模型表达领域知识提高内聚性
- 应用层使用领域模型
- 防腐层提供领域层所需的实现，防止实现耦合到领域

领域模型表达领域知识

```
User userInfo = userClient.get(userId);
Product productInfo = productClient.get(productId);

if (productInfo == null) return false;

Map orderInfo = new HashMap();
orderInfo.put("cId", userInfo.getId());
orderInfo.put("cname", userInfo.getName());
…
connection.insert("t_order", orderInfo);
…
smtp.sendMail(…);
```

- 失血的领域模型
- 缺乏对领域知识的表达

领域模型表达领域知识

```
Customer customer = customerClient.get(userId);
Product product = productClient.get(prodcutId);

Order order = customer.buy(product);
```

领域对象依赖注入

- 自己new的对象
- 从ORM中获取的对象
- 从工厂中取得的对象

spring-instrument

build.gradle

```
compile 'org.springframework:spring-aspects'
javaagent 'org.springframework:spring-instrument'
javaagent 'org.aspectj:aspectjweaver:+'
```

```
task copyAgent(type: Copy) {
    from {
        configurations.javaagent
    }
    into "$buildDir/libs"
    rename 'aspectjweaver-(.+?).jar', 'aspectjweaver.jar'
    rename 'spring-instrument-(.+?).jar', 'spring-instrument.jar'
}
```

spring-instrument

Application

```
@EnableSpringConfigured
@EnableTransactionManagement(mode = AdviceMode.ASPECTJ)
@EnableLoadTimeWeaving(aspectjWeaving = EnableLoadTimeWeaving.AspectJWeaving.ENABLED)
```

Run

```
java -jar app.jar -javaagent:build/libs/spring-instrument.jar  \
  -javaagent:build/libs/aspectjweaver.jar
```

领域对象依赖注入

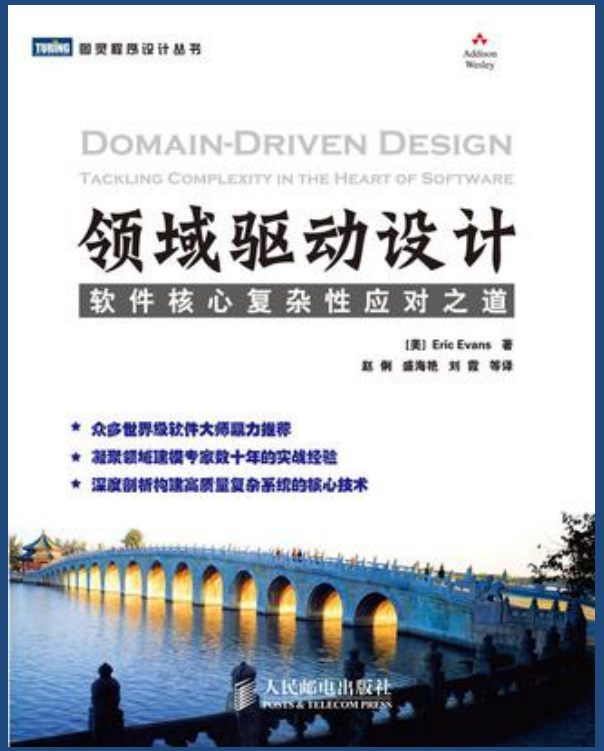
```
@Entity
@Configurable(autowire = Autowire.BY_TYPE)
public class Channel {
    private transient Client client;
    @Autowired
    public void setClient(Client client) {
        this.client = client;
    }

    public void sendMessage(Message message) {
       this.client.send(message);
    }
}
```

原则

- 以界限上下文划分微服务
- 领域模型表达领域知识，高内聚
- 服务与服务之间通过Rest和MQ集成，低耦合

领域驱动设计


DOMAIN-DRIVEN DESIGN
TACKLING COMPLEXITY IN THE HEART OF SOFTWARE
领域驱动设计
软件核心复杂性应对之道
[美] Eric Evans 著
人民邮电出版社


DDD Community
Broadview
PEARSON
实现领域驱动设计
IMPLEMENTING DOMAIN-DRIVEN DESIGN
Vaughn Vernon 著
DDD之父Eric Evans作序

微服务架构

基础设施与DEVOPS

基础设施

- 对象存储：S3/OpenSwift，存储与计算分离
- 消息服务：动态定义消息订阅及路由
- 容器资源管理：Rancher、SwarmKit
- 日志服务：Docker、Rsyslog、Logstash

DevOps

- 分支构建
- 镜像交付
- Dev、QA、Production环境统一

价值

- 新业务最短1周即上线
- 10人团队维护20余种微服务
- 每日数次线上发布，每周里程碑发布
- 全年核心业务可用性99.95%

微服务架构落地技术体系的建设至关重要。

Thank you!

链家网基础架构部 刘思贤