QCon 全球软件开发大会
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

BEIJING 2018

传统企业DevOps+微服务从0到1

博云 BoCloud

BoCloud博云 / 赵安全

Geekbang. InfoQ

GTLC
GLOBAL
TECH LEADERSHIP
CONFERENCE
全球技术领导力峰会

通往**年薪百万**的CTO的路上，
如何打造自己的技术**领导**力？

扫描二维码了解详情

目录

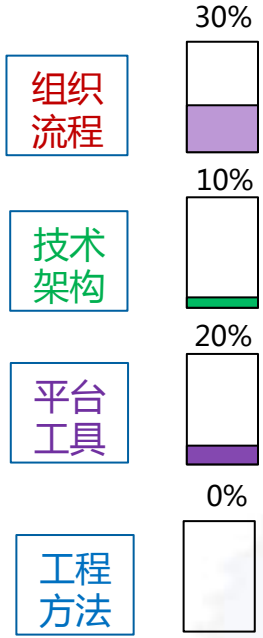
项目背景

博云 BoCloud

困境：业务总规模200多页面，外购件多，需求变更频繁，版本上线太慢，面向客户规模约200万

预计难于应对未来突发大流量需求。

现状和准备度



使用开发测试大组织结构，组织内切分容易，愿意为微服务调整,传统研发规范齐全

只是了解一些微服务知识，无任何试点和开发积累

手工、少量工具

无DevOps理念，无模型设计方法

试点应用选择

最终目标

- 业务重构，能快速响应需求，上线应用，能适应突发访问流量。
- 构建DevOps和微服务体系和技术平台

诉求

- 构建DevOps体系规范并工具落地
- 构建微服务设计理念并具体落地业务微服务拆分和微服务技术框架；

项目背景-初始人员构成

◆10+年的架构师/开发经理/资深开发：6个

◆3-10年的开发人员：8个

◆0-3年的开发：10个

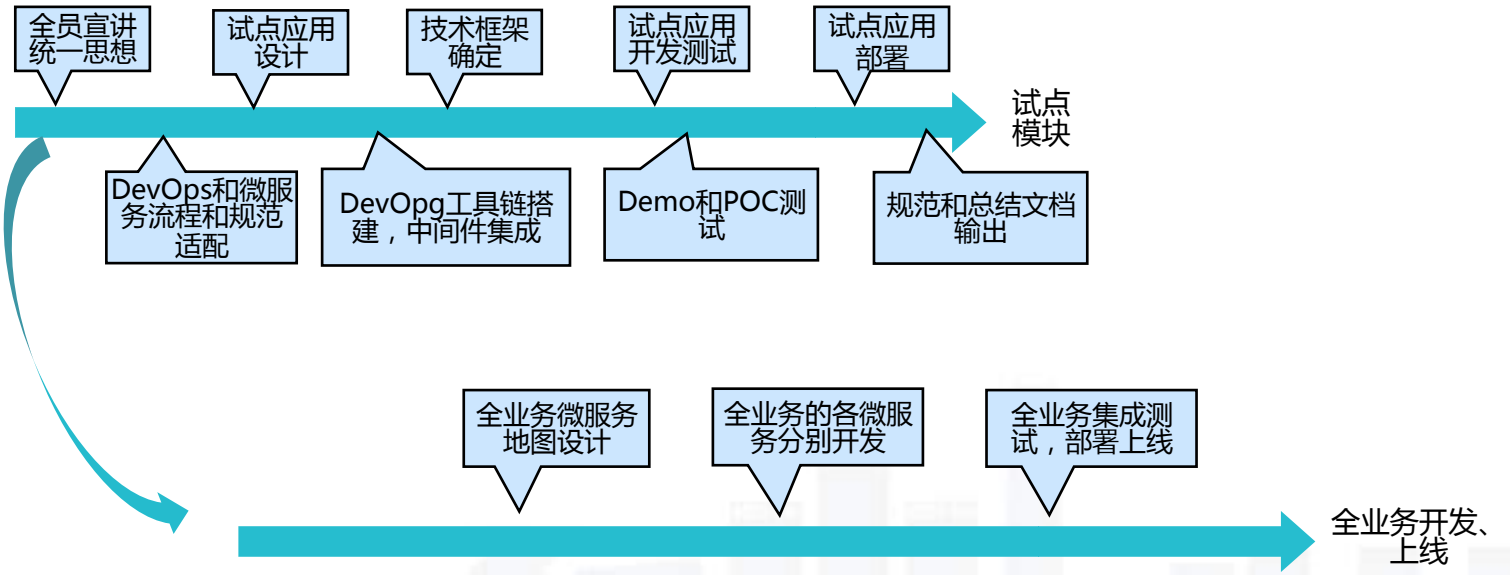
◆测试人员：6个

习惯的开发模式：瀑布模型

习惯的应用架构：传统单体架构

“你们来了之后，我们既不会设计了，也不懂流程了”

实施步骤与工作思路


全员宣讲统一思想
试点应用设计
技术框架确定
试点应用开发测试
试点应用部署
试点模块
DevOps和微服务流程和规范适配
DevOpg工具链搭建，中间件集成
Demo和POC测试
规范和总结文档输出
全业务微服务地图设计
全业务的各微服务分别开发
全业务集成测试，部署上线
全业务开发、上线

总体思路

组织流程

工程方法

DevOps+领域驱动设计

技术架构

微服务

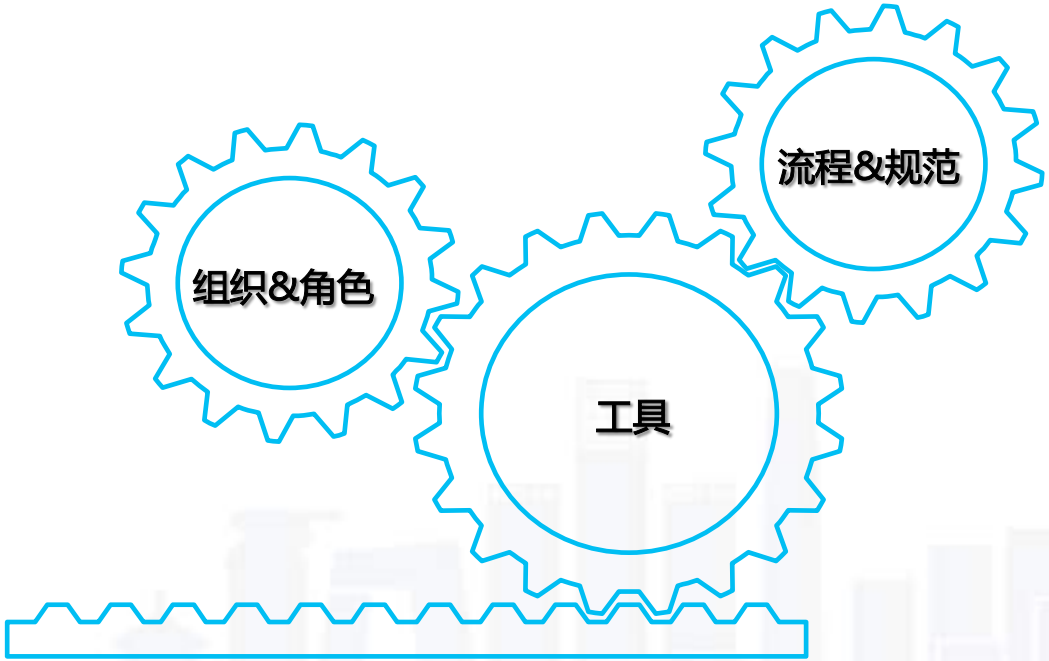
平台工具

20+工具支撑

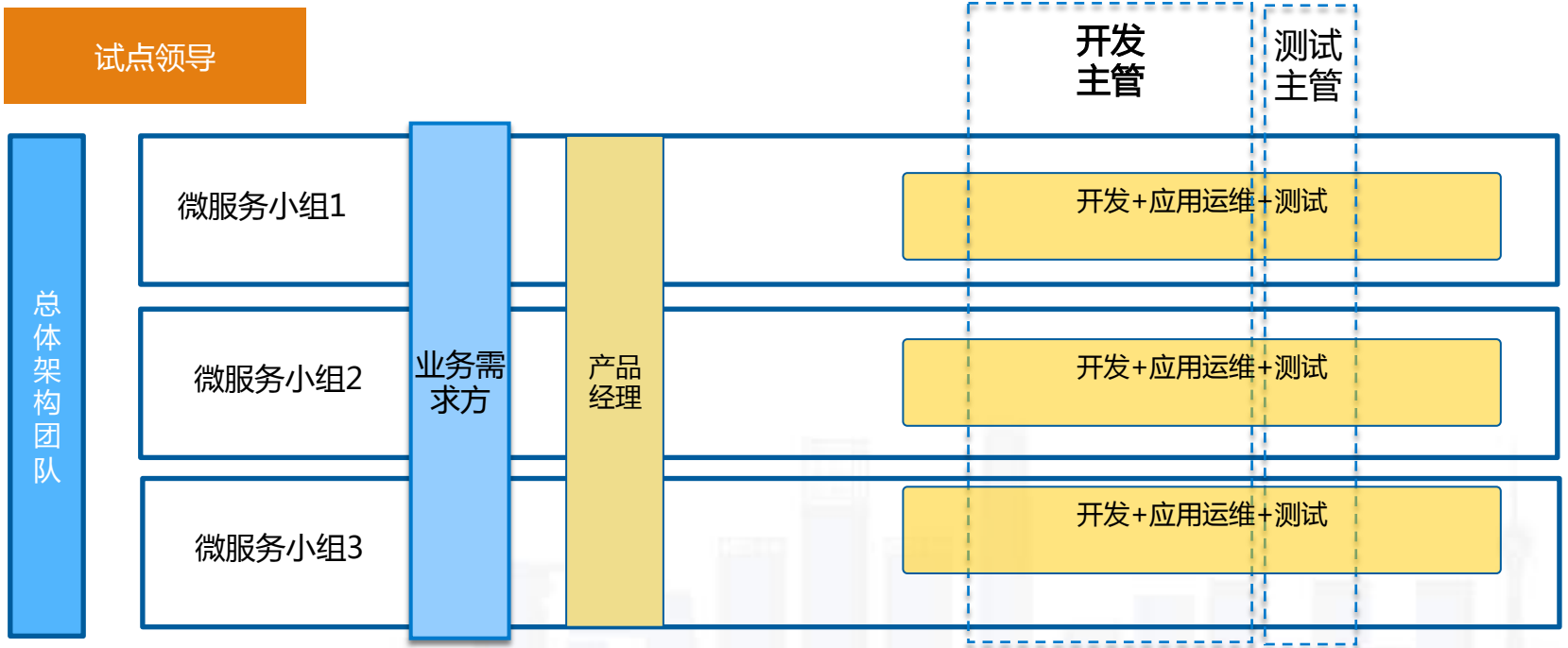
（Jira+Gitlab+Kubernetes+Docker+SpringCloud）

目录

DevOps落地整体思路



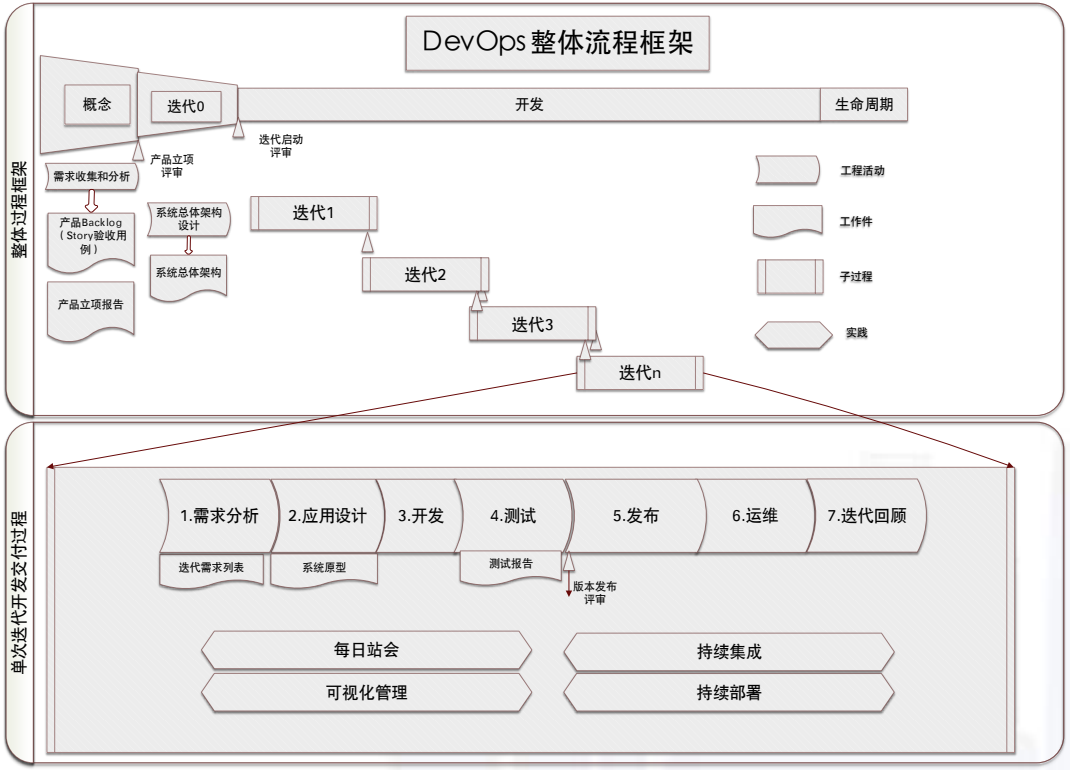
DevOps-组织架构调整



身在心在汉之-周五转测试

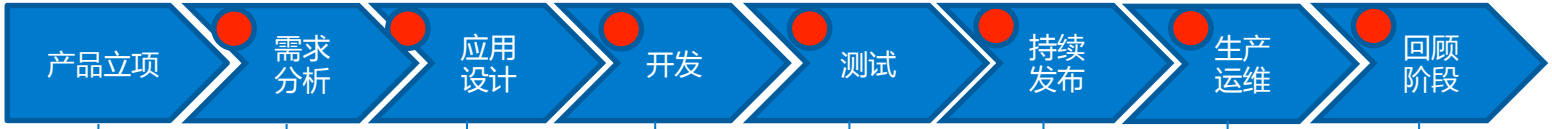
身在曹营心在汉之-跨组织协调

DevOps流程设计-敏捷+持续交付



- 特殊的迭代0：第1个版本-考虑架构设计，业务要求，团队磨合的难度，周期2个月（最小可用版本）
- 后续版本：两周一个迭代
- 有流程，好落地
- 形式引导思维

DevOps-部分规范示例



《产品立项制度》
《 迭代计划会议指南》
《 迭代评审会议指南》
《 敏捷开发过程指南》

《敏捷需求分解规范》
《迭代估算指南》
《用户故事指南》
《需求开发指南》

《微服务业务设计指南》
《数据库设计规范》

《Git 分支管理规范》
《微服务开发规范》
《代码评审指南》
…

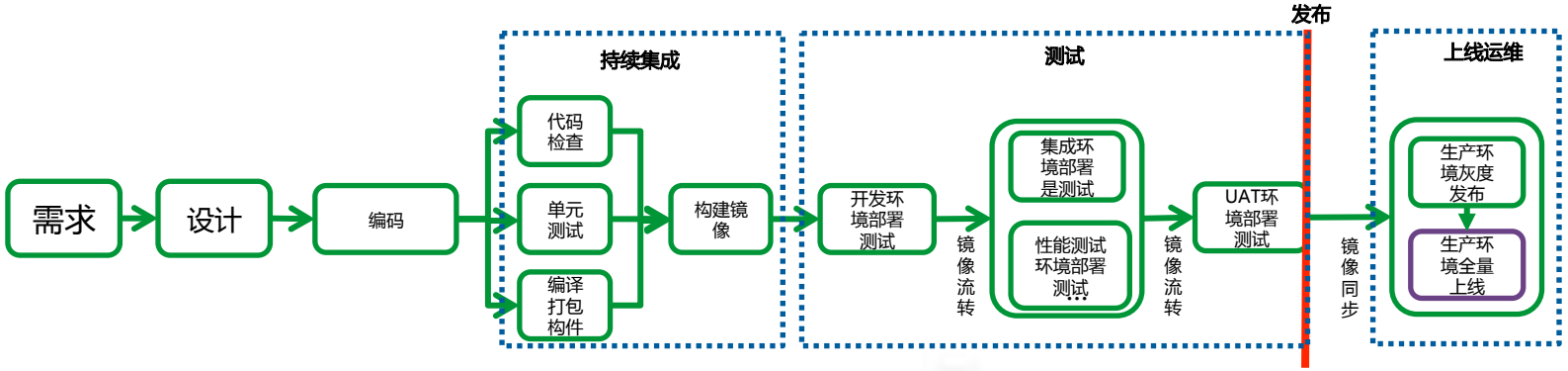
《软件质量标准》
《缺陷管理规范》
《测试指南》

《构建部署规范》
《配置管理规范》

《版本发布规范》

《迭代回顾会议指南》

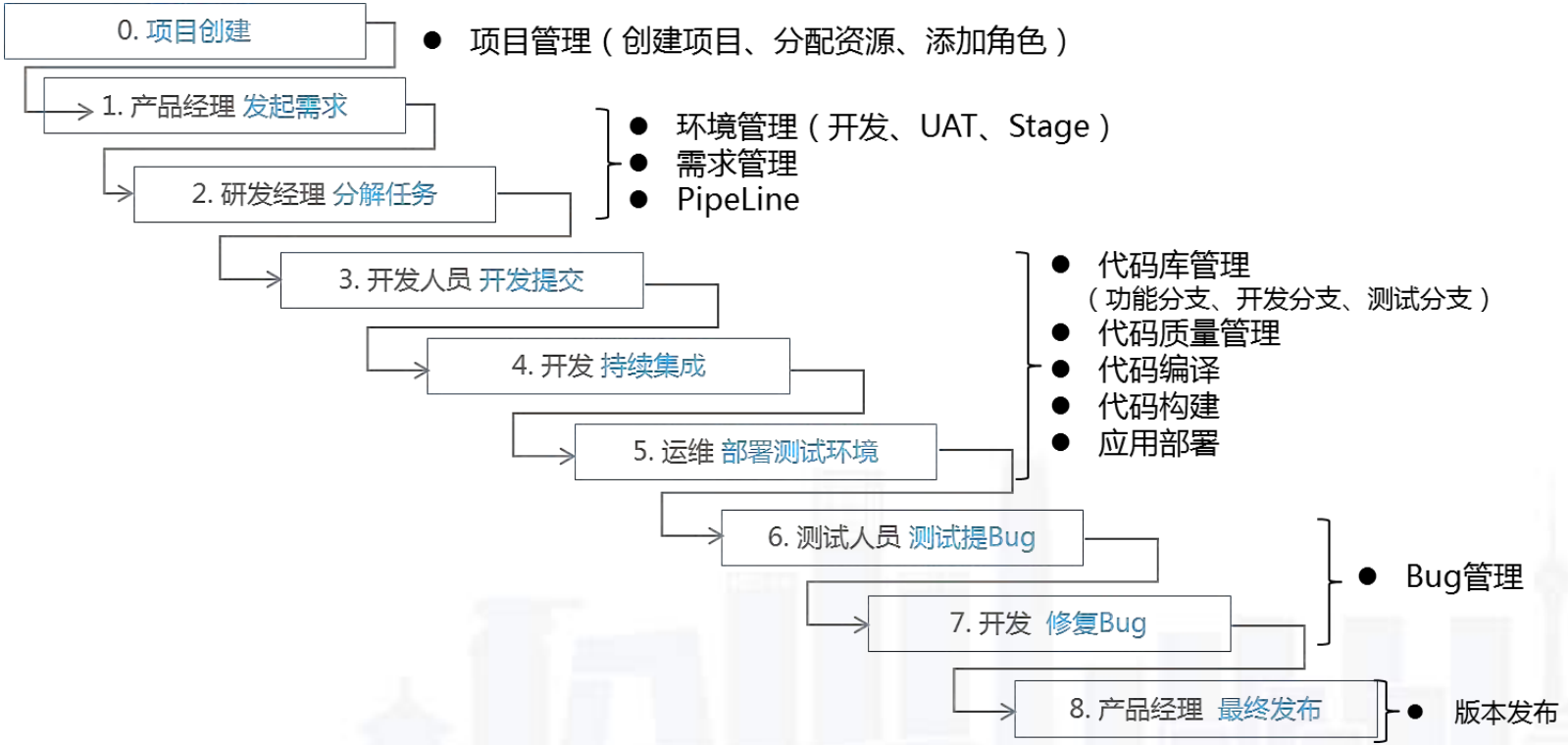
DevOps工具落地全景图



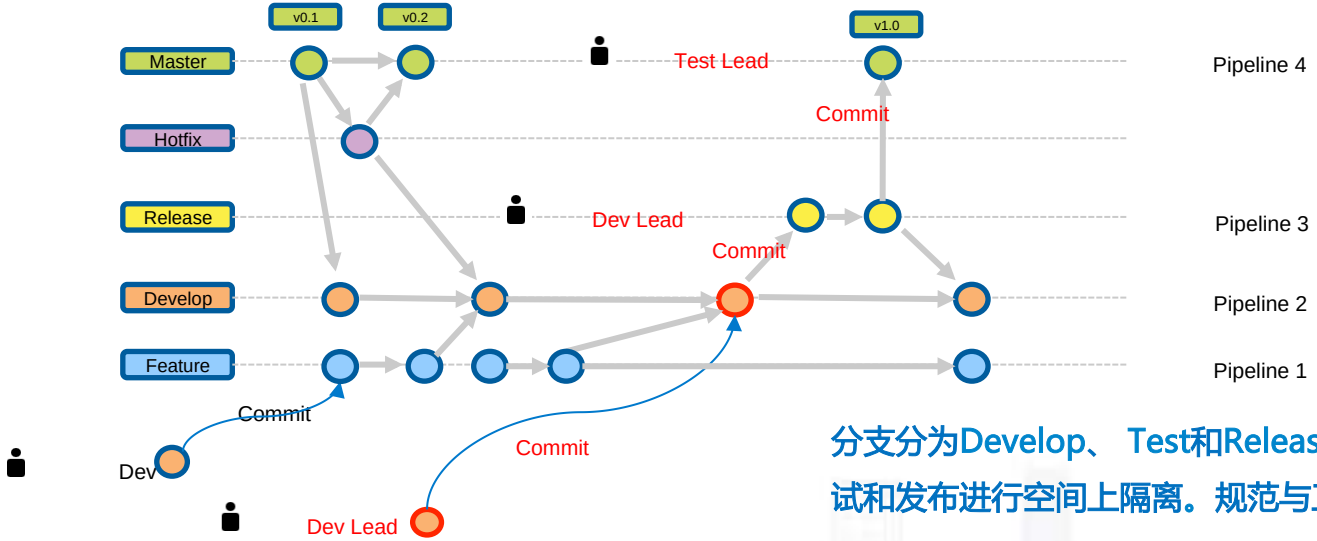
支撑工具

需求管理（禅道/Jira）	计划管理（禅道/Jira）	风险管理（禅道/Jira）	单元测试（JUnit）	自动化测试 selenium	持续集成管理（Jenkins）	代码扫描（Sonar）	镜像扫描（Clair）	微服务框架：Spring Cloud
用户权限管理（Ldap）	代码仓库管理（Gitlab/SVN）	缺陷管理（禅道/Jira）	性能测试（JMeter）	应用实例管理（容器云平台）	自动部署服务（容器云平台）	多环境管理（容器云平台）	日志监控（ELK）	

DevOps-持续交付过程示例



DevOps-代码分支和Pipeline触发关系



分支分为Develop、 Test和Release分支， 将开发、 测试和发布进行空间上隔离。规范与工具集成，保证落地。

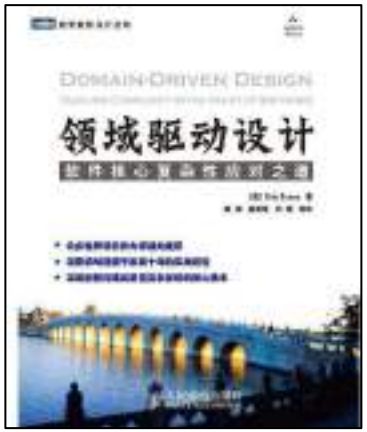
Pipeline	频率	触发点	目的	涉及项目
1	日常	独立开发人员Push 到Feature 分支	功能测试	代码检测，代码规范
2	适中	开发组长Merge 到Develop 分支	Dev集成测试	代码检测，代码规范，代码评审，等
3	不频繁	开发组长commit 到Release 分支（测试通过）	UAT环境测试	代码检测，代码规范，代码评审，自动化测试，手工测试等
4	不频繁	开发组长commit 到Master 分支（测试通过）	预生产环境测试	代码检测，代码规范，代码评审，自动化测试，手工测试，压力测试等

目录

微服务拆分的核心需求

- 高内聚，低耦合：尽可能的减少微服务间的调用，尽可能的减少分布式事务
- 微服务应该足够小，能够分配一个团队（5人左右）去实现，但也不能过细

微服务拆分的指导方法—领域驱动设计



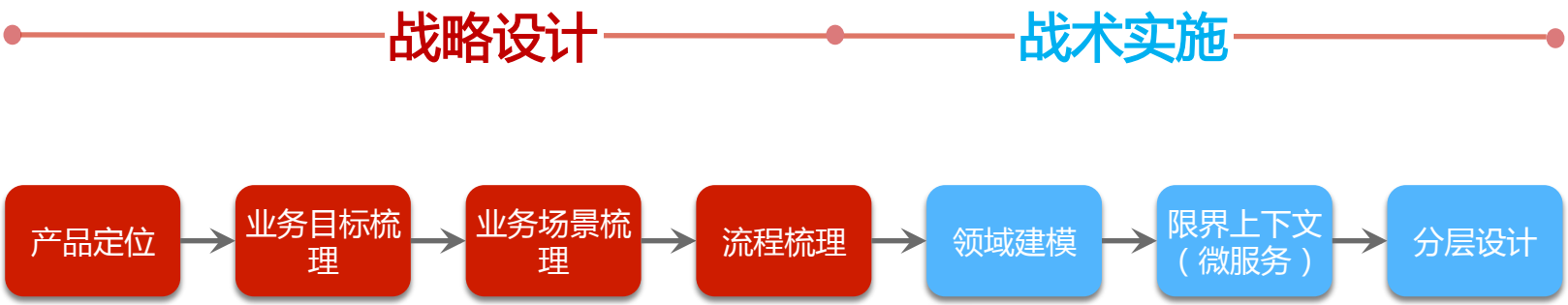
使用场景：

- 针对复杂业务软件
- 增强可扩展性，应对需求变化
- 对设计和开发人员要求高

核心：

- 强调业务领域与系统设计的匹配
- 领域对象和设计复用

整体思路-先业务分解，再领域建模



战略设计-整体思路

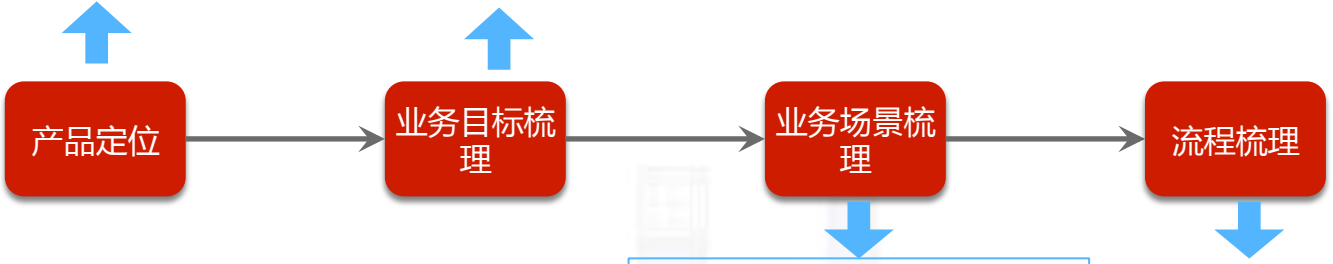
电梯演讲：

- 目标用户
- 用户需要
- 产品名称
- 产品类型
- 关键优点
- 主要竞品
- 主要不同

通过分析用户工作中的痛点和需求来找到需要提供的服务/业务（问题域/子领域）

微服务拆分，业务理解和梳理**非常关键**

关键：业务人员（领域专家）深度参与！

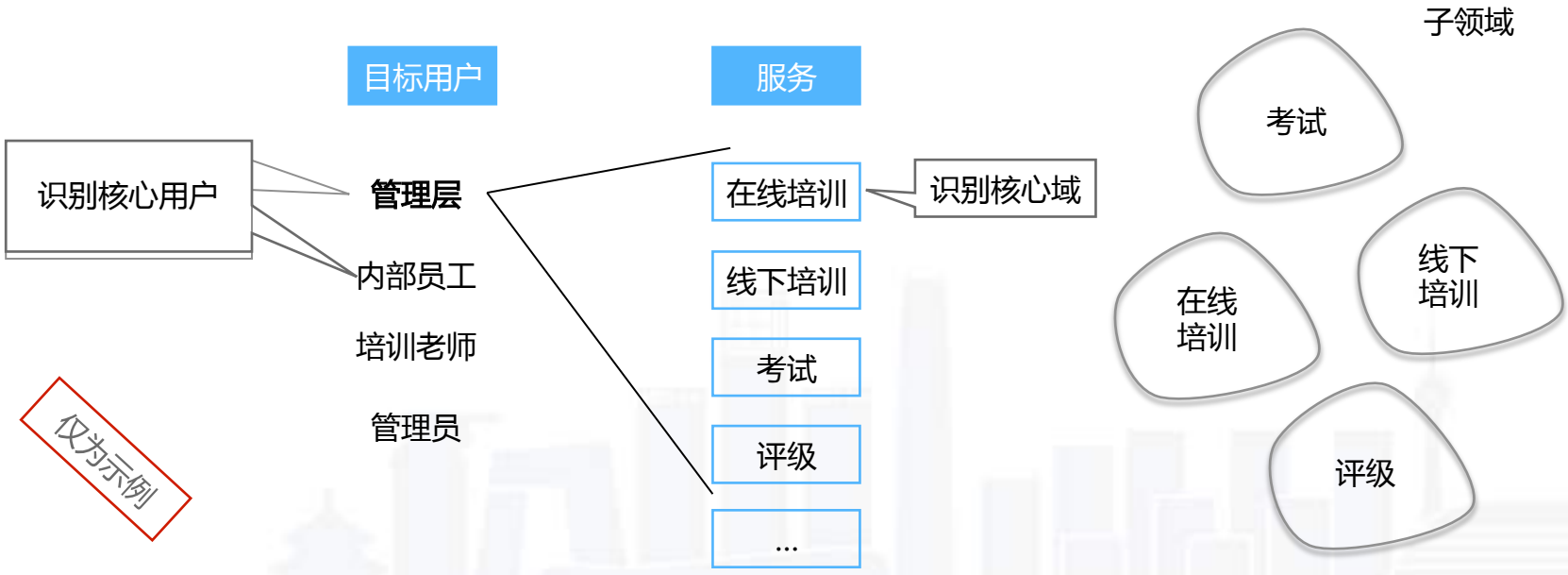


从用户和业务视角对子领域的探索，对产品需要支撑的场景分类及典型场景，用以支撑领域建模。

细化的流程梳理

战略设计举例-业务目标梳理

产品定位：管理层希望建设一个针对内部员工的培训系统，以提升员工技能。该系统需要提供完成培训、考试、评级等功能



找到最小可用版本

战略设计-场景梳理方法

- 选定一块子领域
- 参与者针对所选定的问题域，发散思考各个服务场景

- 对发散所得的服务场景进行整理，得到场景分类

仪式化

领域专家参与

重复的场景即潜在的基础服务

仅为示例

流程梳理和领域建模示例

流程梳理

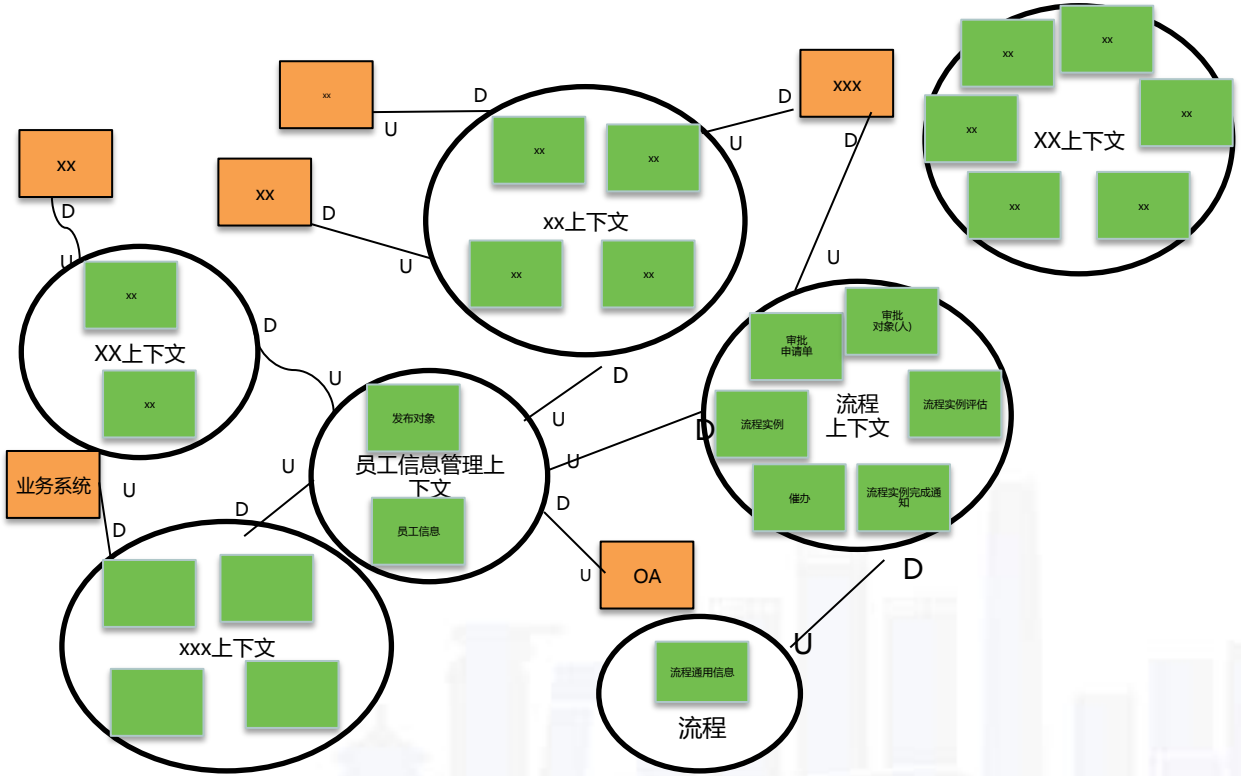
实体（聚合根）

值对象

领域建模-聚合

仅为示例

限界上下文设计示例



设计依据：

- 子领域
- 聚合之间的关系

识别基础服务

识别外部服务

校验场景和流程

确定服务间接口

确定微服务拆分-围绕限界上下文调整

设计因素1：围绕限界上下文边界

◆理想情况下限界上下文与微服务为1:1

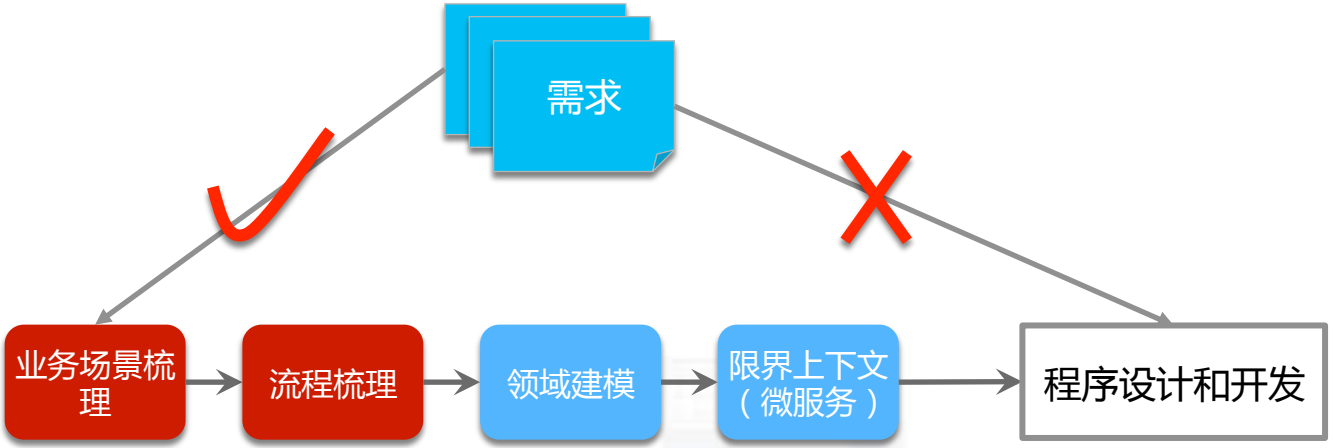
◆微服务拆分的底线是不能打破聚合，打破聚合会破坏事务一致性和业务约束。

设计因素2：对齐不同业务变化速率/相关度

设计因素3：考虑系统非功能性需求：安全性、伸缩性等

设计因素4：其他设计约束：复杂度、团队结构、技术异构

应对需求变更

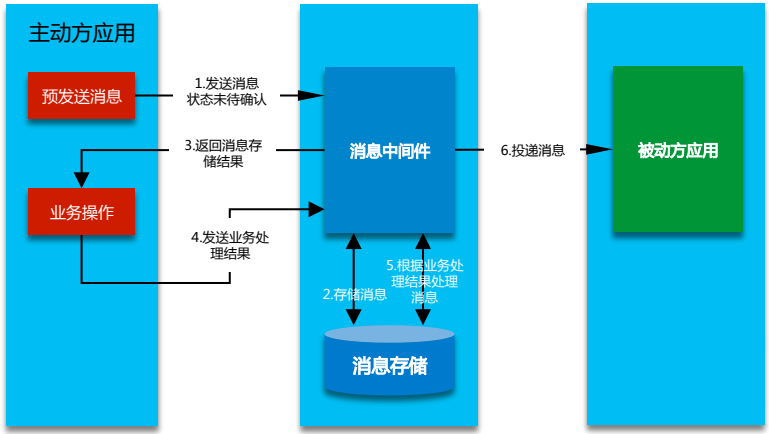


经验分享-分布式事务处理

博云 BoCloud

都存在

可靠消息最终一致性方案



适用于一致性要求较高，实时性要求不高的业务场景
被动方开发成本低，业务系统与消息耦合度低

强一致性方案-TCC

适用于实时性要求较高的业务场景
业务层开发代价高

经验分享—模拟API

- 微服务的开发是有不同的开发小组进行开发，一旦服务消费者需要服务提供者提供的api延期，服务消费者则无法进行相应的开发工作。
- 当服务消费者需要联调时，服务提供者必须是启动状态并且网络通畅。
- 使用MockMvc实现对http请求的模拟，能够直接使用网络的形式转到controller的调用，这样可以使得测试速度快，不依赖网络环境，而且提供了一套验证工具，使得请求的验证统一而且方便。

仅仅是从0到1，还需要持续改进

博云 BoCloud

- 测试自动化、契约测试
- 服务治理深化
- 持续交付流水线优化
- 微服务独立发展
- 跨组协同，团队自组织

总结

1.选择合适应用，组建新团队，有利于迈出DevOps+微服务的第一步；

3.微服务落地的关键不仅仅是技术框架，更重要的是设计。在领域设计中，战略设计和战术设计同样重要。

GMTC 2018

全球大前端技术大会

大前端的下一站

<<扫码了解更多详情>>

极客邦科技
企业培训与咨询

Geekbang.

扫码关注

获取更多培训信息

关注QCon微信公众号，
获得更多干货！

Thanks!

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

Geekbang. InfoQ