第 8 周 架构案例

秦金卫/KimmKing

资深架构师

KK架构训练营讲师

讲师介绍

- Apache Dubbo/ShardingSphere PMC
- 资深开源参与者/前阿里巴巴/京东架构师
- 《高可用可伸缩微服务架构》作者
- 《JVM 核心技术32讲》专栏作者
- 《面向性能的系统架构设计》作者
- 阿里云MVP/腾讯云TVP/TGO会员

目录

1. 某大型电商平台的微服务架构改造

互联网迭代快，业务要求不能停，怎么办？

重构与重做，改造与复用

取舍与妥协，改造效果

1. A 系统的现状与问题

A系统的背景

业务上：从杭州刚挪到北京，作为重新孵化项目，想要做大做强。行业竞争压力非常之大，做不出成绩，就是生与死的问题。

技术上：遗留的老系统，再一起其他系统的基础上改的单体，团队拥抱变化，换手了4次以上，找不到一个做过这个系统的人了。

关系上：北京新业务定位是新业务的创新和孵化，这是一个大项目，也是要树立标杆的项目。

A系统的问题

当时我写的一封邮件，说的相对委婉：

国内机票旧系统开发于2006年，期间几十名开发人员在代码里留下了痕迹。不久前移交到北京新业务研发团队维护，主要问题有：

1、代码量大(40万行)，质量不高，结构复杂，难以修改和维护。一些代码的具体作用和逻辑细节，没有人清楚。很多注释和代码不一致，文档缺乏，比较混乱。

2、业务处理过程逻辑复杂，中间数据冗余，与外部接口耦合度高，性能较差，稳定性不高，扛不住大促活动压力。

3、对业务过程的数据和状态监控不足，难以管理控制和跟踪，各方使用不方便。

4、刚交接过来，没有人真正懂这个系统的业务，业务方也是空降，这个问题尤其严重。

重构面临的压力和选择

其实当时看来，重做是最好的路线，但不现实：

业务方：不能停止业务发展，死路一条

开发方：没有太多资源，也经不起折腾，半年后大促

2. 如何推动重构 A 系统

核心矛盾点在哪里?

这是一个很常见的矛盾点:

业务不认可重做/重构，伤筋动骨。

一提就说：反正系统还能用。

重做周期太长，成本太大。

重构不带来业务价值，步子大了也会影响业务。

祖传屎山，维护不易。

对于业务方

业务上：

必须要稳定推动业务发展，快速加功能，开发营销促销系统，不然被对手越拉越大。

核心要求：

1、业务不停

2、保障稳定

3、提升性能

4、快速开发

对于产品方

产品上：

提升用户体验，跟竞品在产品层面拉齐，多对接航司和第三方，引流，打通各方。

核心点：

1、用户体验

2、改版引流

3、同业务方

对于开发方

技术上：

最大问题是，资源不够。

最少的资源，最大满足业务和产品需求，同时能够发展北京研发团队。

最终达成一致

各方妥协，但是研发肯定是压力最大的。

不用怕，临危受命的同时，可以提要求，定规则。

不停机，发展业务，改进体验，都没问题。

本着这个大目标，要求按我们的安排来。

1、所有其他项目为这个项目让路，优先级 P0，（后来改成-1）

2、资源主要用来保障这个项目，所有人配合，包括后面招聘的

3、人员管理等方面，特事特办，所有人不管考勤等制度

3. 重构的目标和方式

怎么定重构目标?

平衡短期利益和长期利益。

分阶段，先重构搜索模块，然后是订单交易、产品库存。

国内机票搜索重构启动于6月底，经过近两个月的紧张开发，于8月底完成开发并提测，目前进入测试和准备上线阶段。

预计上线后，能带来更高的性能（+30%以上），更好的用户体验（数据的准确性、缓存策略），更好的代码质量与扩展性（更易于日常维护和添加功能），更好的监控和管理（at-eye）。

为国内机票年底的双12大促（预期投入一亿费用），做好最充足最全面的准备。

怎么定重构目标?

国内机票搜索重构的目标主要有五个:

1、培养机票新人。梳理清楚国内机票搜索部分的业务和各种潜规则，培养懂机票的新人。
2、提高代码质量。设计更合理，结构更清晰更易维护，系统耦合更低、扩展性更高。
3、提升搜索性能。使系统更稳定，更高效，搜索更快，资源消耗更低，大促不宕机。
4、监控搜索过程。统一管理搜索过程中的各种开关和参数调整，管理缓存，跟踪搜索步骤。
5、国内机票搜索去 oracle 化。 在搜索重构的过程中，用 mysql 代替旧系统的 oracle 数据库。

目标的具体说明

① 培养新人问题，主要是XX\XX\XXX\XXXX。在此过程中，熟悉淘宝技术体系和开发过程，掌握国内机票业务，能逐步的 cover 机票系统。

② 代码质量问题，主要是项目结构合理的分层，搜索的分段式处理，code review，高覆盖率的单元测试，使项目易懂易上手易维护。

③ 提升性能方面，主要从简化中间结果和数据结构，理清业务简化旧逻辑，整理各步骤缓存最大化合理利用缓存结果三个方向来提升系统在 CPU 计算和内存、以及网络 IO 等方面的开销。提高系统吞吐量、QPS，降低响应时间。

④ 监控搜索过程，通过哈勃、timetunnel+ateye、memlog、系统 log 等手段埋点，管理系统多种粒度的开关和参数设置，跟踪和调试搜索流程中的数据和状态、各种外部接口数据和缓存。

⑤ 去 Oracle，改用 mysql+tddl，实现分库分表，为进一步的分布式架构打下基础。

重构的组织方式

三个要点：

小黑屋主力，核心+全流程

其他后备力量，业务模块+迭代周期

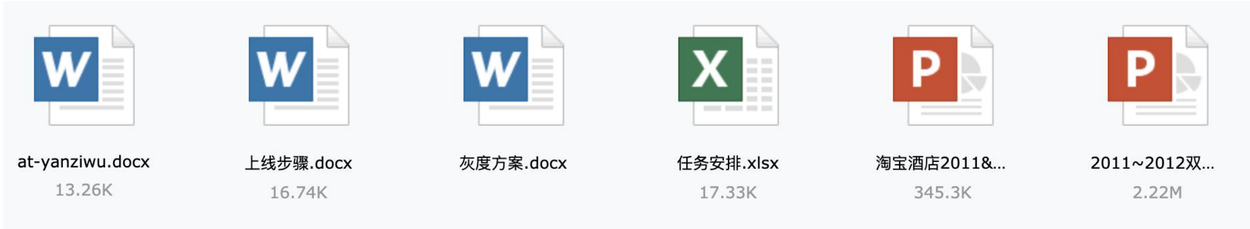
开发期间，试行996（全部算倒休），抢在年底大促前搞定三期重构

4. 重构的过程和结果

重构怎么做--规划、启动

确定规划，分步改造

启动项目，拉齐思想



抄送: 淘宝–淘宝旅行; 淘宝–旅行技术; 淘宝–技术研发部–新业务技术团队
主题: 国内机票搜索重构项目：夺天之战【涅槃】

天柱山，玄玄大陆第一高山。相传山高万仞，横亘在天地之间，绵延数十万里，直插入青天。故老相传，此山为天之尽头！天柱山，山连天，神魔止步不可攀！五千年前，天柱山塌陷，大批异世界敌人在此进入人类世界。

人界三大圣地遁世仙宫、至尊金城、梦幻血海，以及玄兽圣地天罚森林，集合所有强者聚于此地，鼎力合作，与异族一战，封邪魔于异界，大胜而归。

此战名曰“**夺天之战**”。

燕子坞，淘宝仙境第一名胜。据说由数千宛如珍珠般得珊瑚岛组成，蓝天白云，椰林树影，水清沙幼，风景如画，水天如诗。公认为天上地下最美之地。燕子坞，岛如珠，神佛留恋不复出！

[redacted]奋战于此，

变革国内机票搜索系统架构，根治五年以来之沉疴，解耦系统，梳理业务，订正逻辑，精简数据。名为重构，实为重写。六人争分夺秒，日夜奋战，精诚合作，众志成城，呕心沥血，最终于近日功成。

破而后立，浴火重生，复建天界秩序；

勇于前行，乘风再起，中兴淘宝旅行。

史记称之为“**涅槃**”！！

Milestones

M1 上线启动10-11

M2 灰度上线10-20

M3 功能发布10-27

M4 全部提测11-08

M5 所有上线11-29

M6 机票大促12-21

重构怎么做--明确业务

1、重建业务需求文档。

需求是一切的起始点。

2、如何保障业务不停？

架构层面的设计，与产品方达成一致。

重构怎么做--架构调整

分层改造和逐步对接，借鉴与微软的 outlook 团队的案例。

这在当时是一个很大的冒险。

例如，页面的改版需求，我们顺带着把一些后端的业务逻辑，代码结构，数据库都改掉了。

在一个需求周期内，搭车业务需求，做技术改造。

重构怎么做--性能优化

从优化业务处理角度。同步大事务到小事务，异步处理。

从优化系统代码角度。一个8层 for 循环的例子。

从优化数据结构角度。笛卡尔积的例子，一次请求500M内存。

从优化缓存策略角度，从静态有效期到动态有效期。

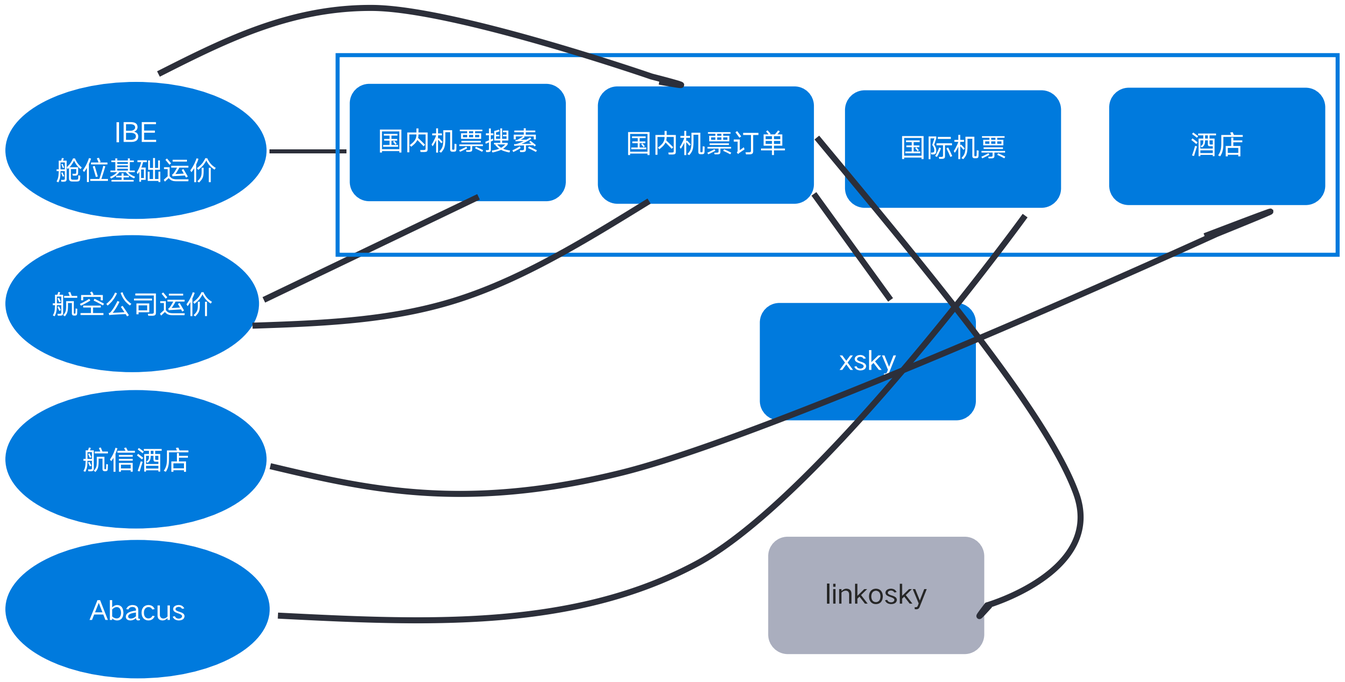
时效性：

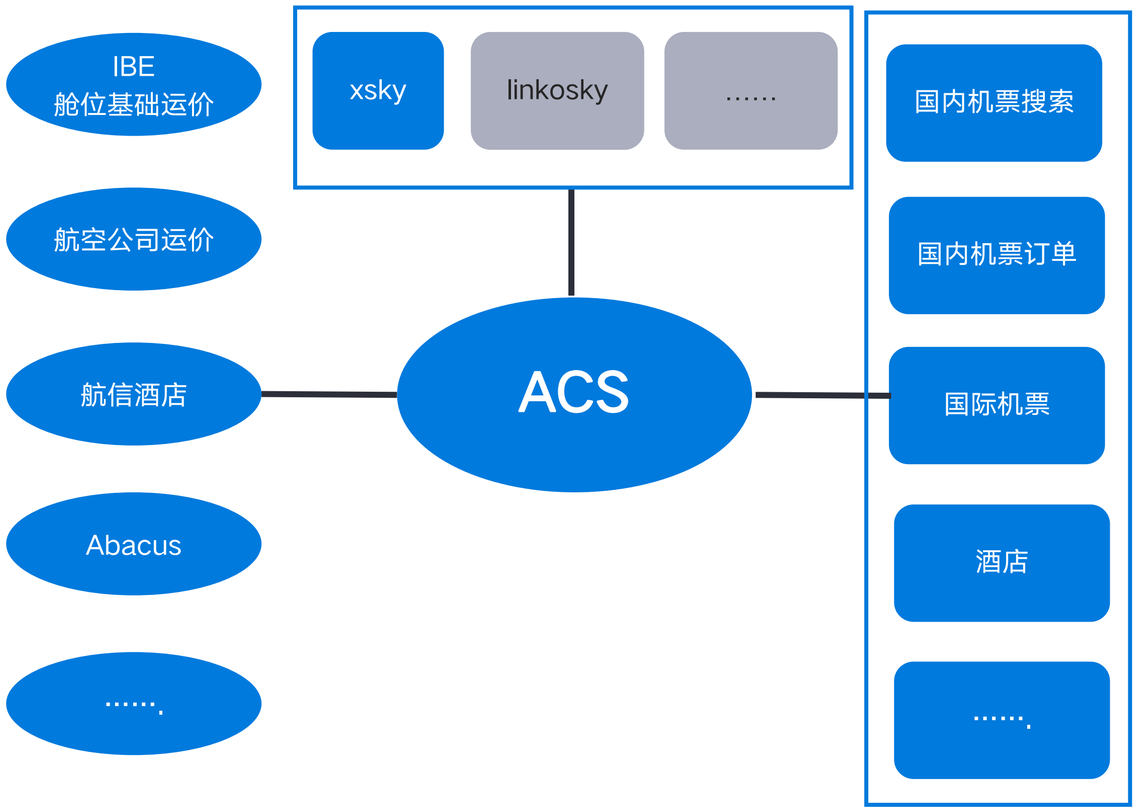
- 明天的，周末的，节假日的，
- 航线的，

重构怎么做--引入标准

借鉴之前的 ESB 经验，解决对接的复杂性问题，标准化集成方式。

不同航司的接口，差异非常大，数据结构不同，数据的有效性不同。





重构怎么做--灰度发布

通过灰度发布+特性开关，保障可靠上线。

先是预发环境上线，办公室内访问，
然后线上灰度，公司内部IP访问，
再接着杭州和北京访问，
最后全国可以访问，完全放开。

重构怎么做--监控运维

实现一套全方位的运维系统，快速发现和处理问题。

实现线上开关和 debug 的机制，快速定位问题。

对每天的业务数据进行自动统计和发邮件，做到所有人对业务情况了然于心。

项目的效果

业务上，支持了大量新功能，包括营销促销系统。
技术上，重构了新系统，完成了去 O 和分库分表，保证了大促。
产品上，优化了用户体验，补齐了系统的各项短板。
团队上，培养了一批懂业务，懂新系统底层的研发核心人员。

新系统文档完善，架构先进，代码只有旧系统的1/3不到，性能提升50倍。
> 做减法比加法难。

此外，在新架构的基础上，打通了各个业务线。
后来我又负责了其他系统的重构，旅行线业务统一项目，关联销售项目等。

5. 对本次重构的复盘

复盘总结

为什么要做重构？

重构的难点是什么？

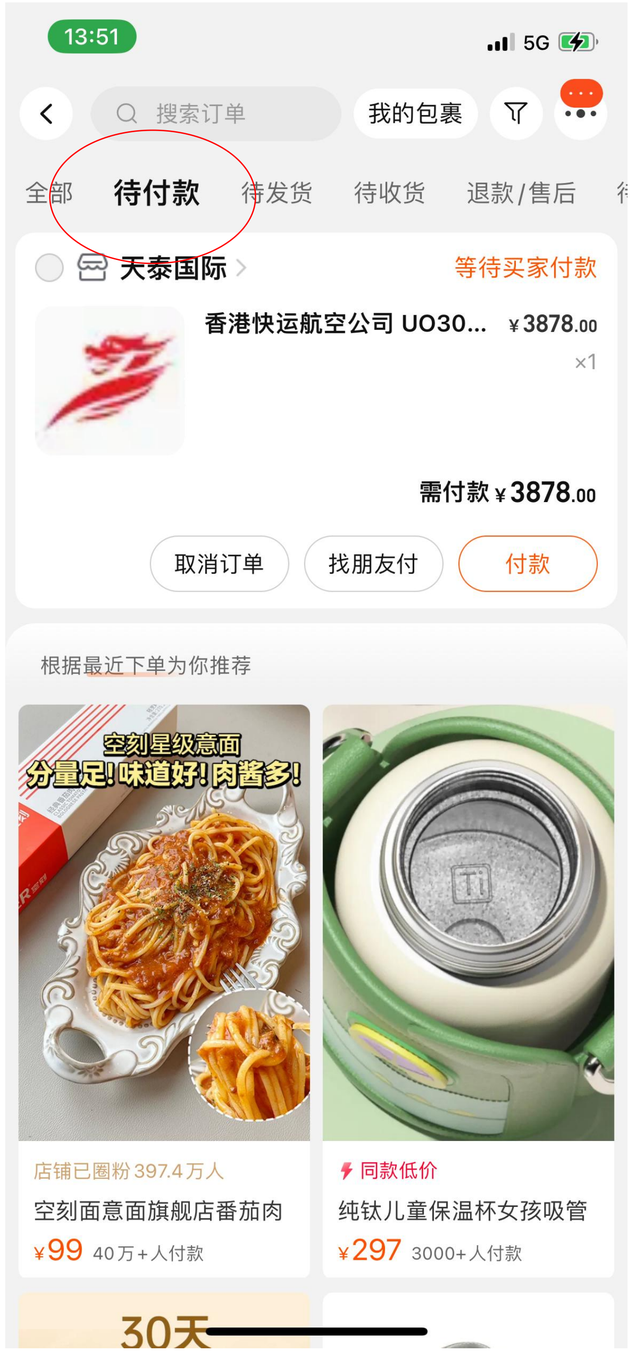
重构要做哪些事儿？

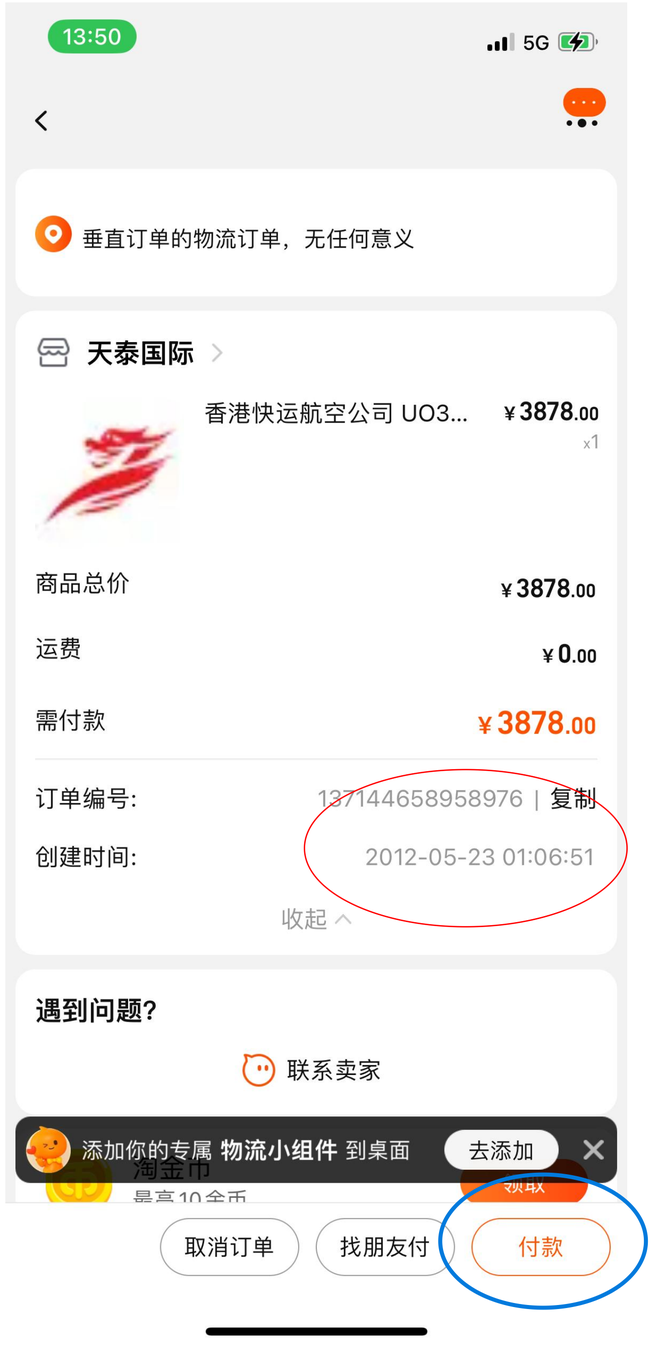
怎么解决重构过程的各种问题？

怎么保证重大改造的稳定上线？

一个彩蛋

机票的数据迁移。





2. 某国有大行核心系统分布式架构设计

分布式新核心建设背景

2019初，XXX公司开始协助XX银行开展新一代个人业务核心系统预研项目。

经过对老核心进行梳理，确定采用分布式技术建设新核心。

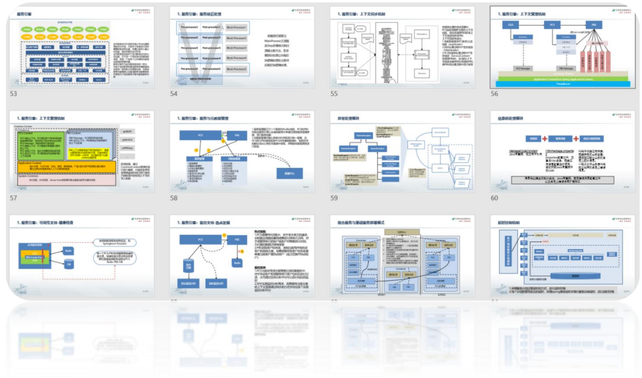
先后调研微众银行，蚂蚁金服，民生银行与百信银行，分别针对各家的分布式核心系统进行调研，确定了分布式核心平台的可行性，并结合业界经验确定了基于单元化+微服务设计的分布式架构。

2020年初，完成总体技术方案，技术选型，原型验证以及相关的设计文档。

分布式新核心建设背景

2021年4月，完成技术平台投产，旁路验证。
并且开始支撑大规模的业务系统开发。

2022年4月，完成新核心个人存款上线。
2023年，完成各业务系统的上线。



分布式新核心建设目标

新一代个人业务核心系统，采用工程方法论进行实施，主要包括：

1. 基于企业建模方法进行业务建模，形成流程、实体、产品模型驱动的企业级业务架构；

2. 基于单元化+微服务架构构建企业级 IT 架构；

3. 基于实施工艺和架构管控指导项目实施，形成企业级工程方法；

应用开发，需基于应用分析和设计结果（分析和设计工艺产出物），结合 IT 架构提供的开发模型进行落地。

分布式新核心建设路线

技术验证

技术平台上线

大规模业务开发

业务系统上线

分布式新核心建设难点

新一代个人业务核心采用单元化微服务架构，主要面临以下的问题和挑战：

1.业界没有同规模的单元化微服务架构实现案例可参考

2.单元化设计难题（单元规划、单元内分库分表、单元容灾）

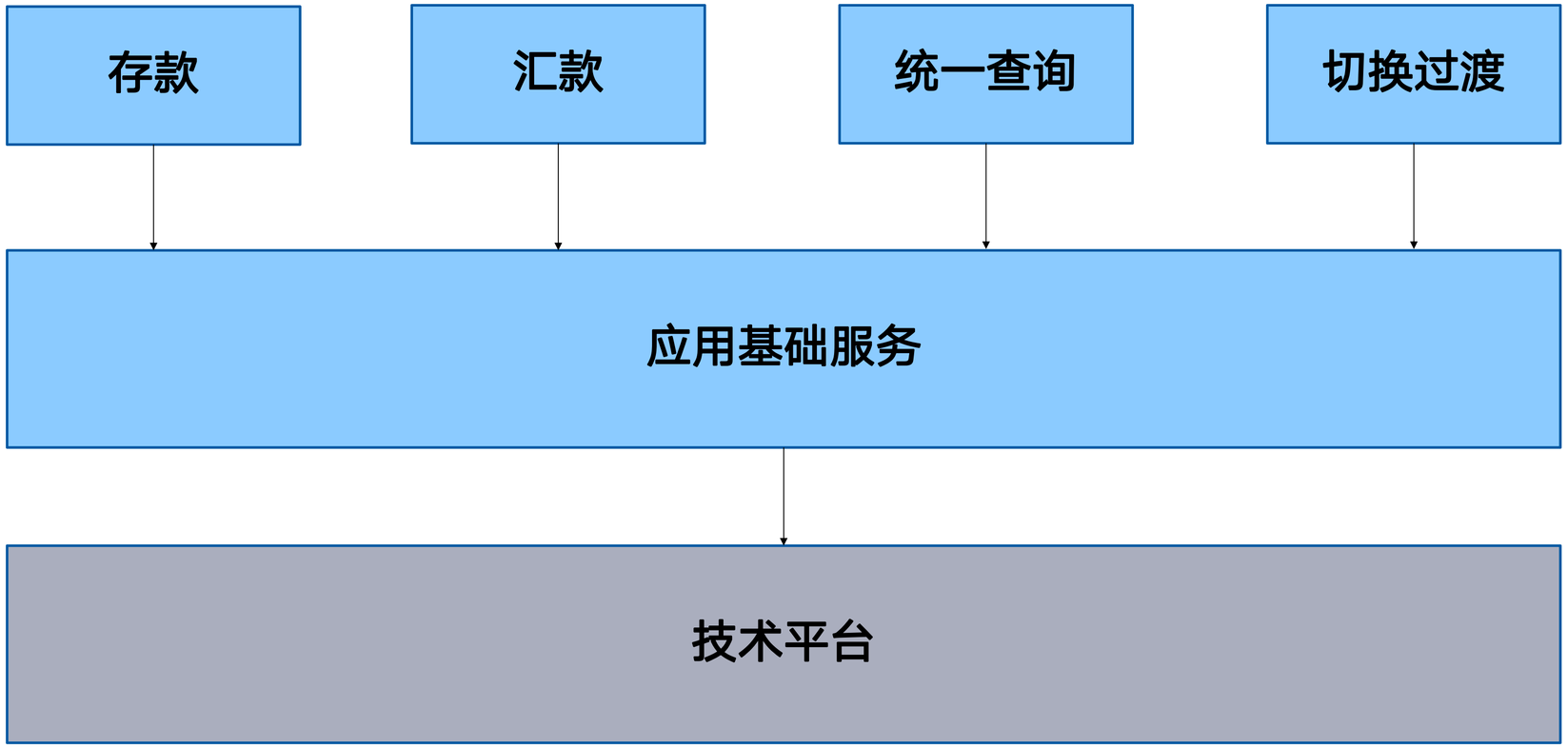
3.微服务拆分和设计难题

4.微服务治理难题（限流、熔断、故障切换、灰度发布）

5.高可用（99.99%）、高性能（> 150,000 TPS）的挑战

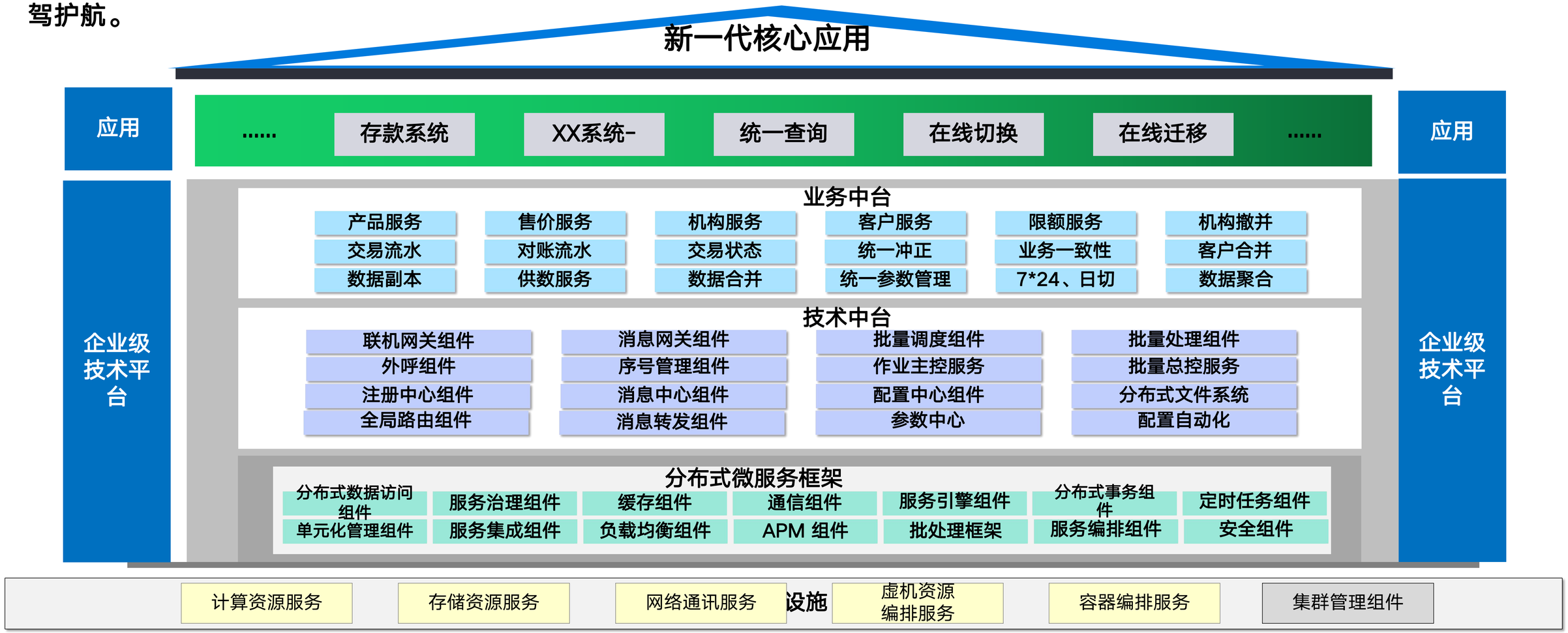
分布式新核心--整体架构设计

整体架构



分布式新核心-整体架构

建设企业级技术平台（技术中台），自下而上划分为微服务框架（框架层）、技术中台（独立服务层）、业务中台三层，为业务发展保驾护航。



分布式新核心--整体架构设计-单元化

单元化架构划分：

1，数据按「客户维度」进行分区，每个分区称为「单元」

2，一个客户关联的所有数据落在同一个单元中

3，每个单元有自己的应用和数据

4，每个单元能满足单元内用户的所有业务操作。

单元化架构好处：

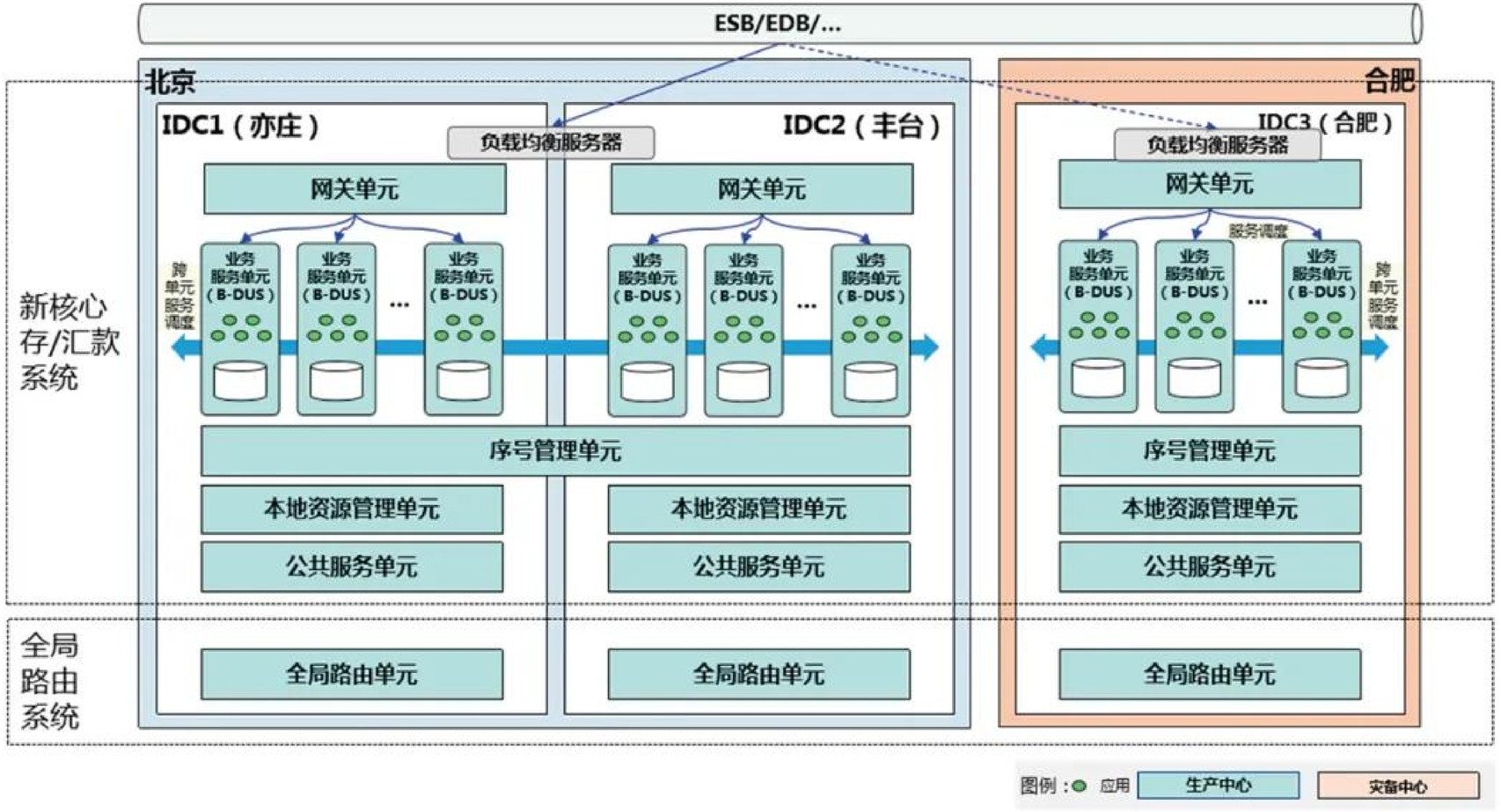
1. 无限可伸缩架构（单元水平扩容、微服务弹性伸缩）

2. 多中心部署架构，提升容灾能力（单元容灾模型）

3. 灰度发布能力，提高版本发布可靠性（按单元灰度、微服务版本灰度）

4. 故障隔离能力，降低故障影响范围，提高系统可用性（单元故障隔离、微服务故障隔离）

分布式新核心--整体架构设计-单元化



Y行新一代个人业务核心系统采用单元化部署架构，采用开源+增强的自主可控研发模式，形成核心系统的快速弹性扩展支持能力。

- 采用16单元64库1024表拆分方式，两地三中心部署。
- 系统可用性超过99.99%，设计容量10亿客户，支持超过5万TPS的交易峰值。

分布式新核心--整体架构设计--微服务

微服务采用Apache Dubbo 2.5.3/2.6.8 作为基础骨架。

Apache Dubbo vs Spring Cloud?

民生过来的，Dubbo、Druid，Shardingsphere 很熟。

分布式新核心--整体架构设计--微服务拆分原则

1.先粗后细原则

- 粒度过细，对机器资源、数据库连接数、运维管控都会有影响
- 通过性能测试分析，决策是否有必要再更细粒度的拆分

2.组合服务 / 原子服务 分离部署原则

- 组合服务 按功能、渠道拆分部署
- 原子服务 按功能拆分部署

3.按部署单元拆分原则

- 不同功能部署在不同 DUS 拆分成不同微服务（应用）

4.按重要程度拆分原则

- 不同 SLA（性能、高可用等） 要求；如：三方支付、柜面功能独立微服务（应用）

为什么选择单元化分布式方案？（1/3）

- 现状梳理、访谈，确定采用**「分布式架构」**与**「企业建模」**结合的方式实现核心系统
- 同业交流，确定基于**「单元化」**设计的分布式架构
- 方案细化，功能梳理，并结合外部调研，确定**「微服务架构」**与**「单元化分布式架构」**结合建设原型技术平台

- 原型技术平台设计和开发
- 原型技术平台功能测试、压力测试

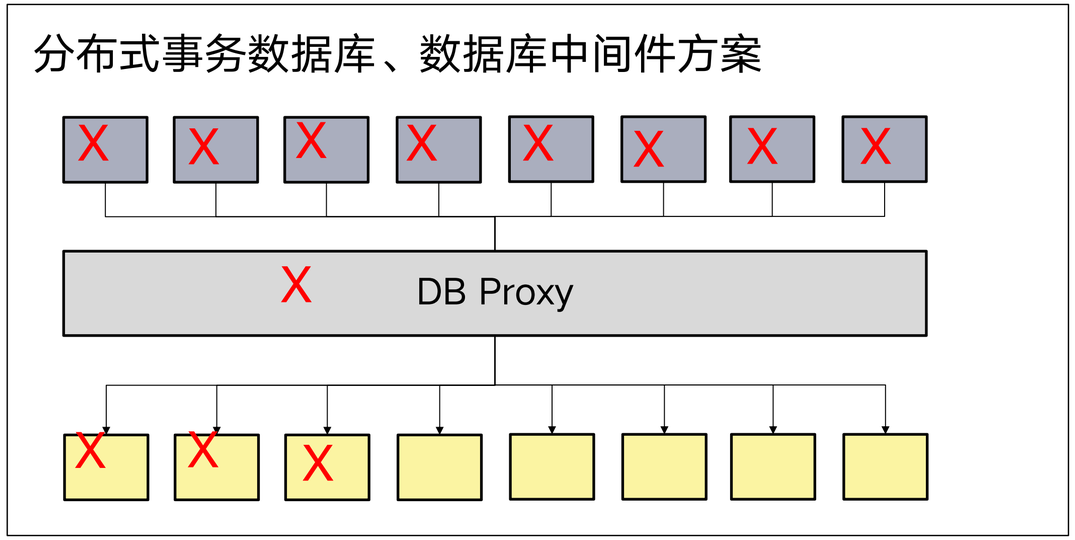
- 能力重新梳理
- 需求分析和设计
- 技术平台工程开发

通过访谈、同业交流，确定**「单元化分布式架构」**、**「微服务架构」**与**「企业建模」**结合方式建设核心系统，并经过原型技术平台论证可行性。

为什么选择单元化分布式？（2/3）

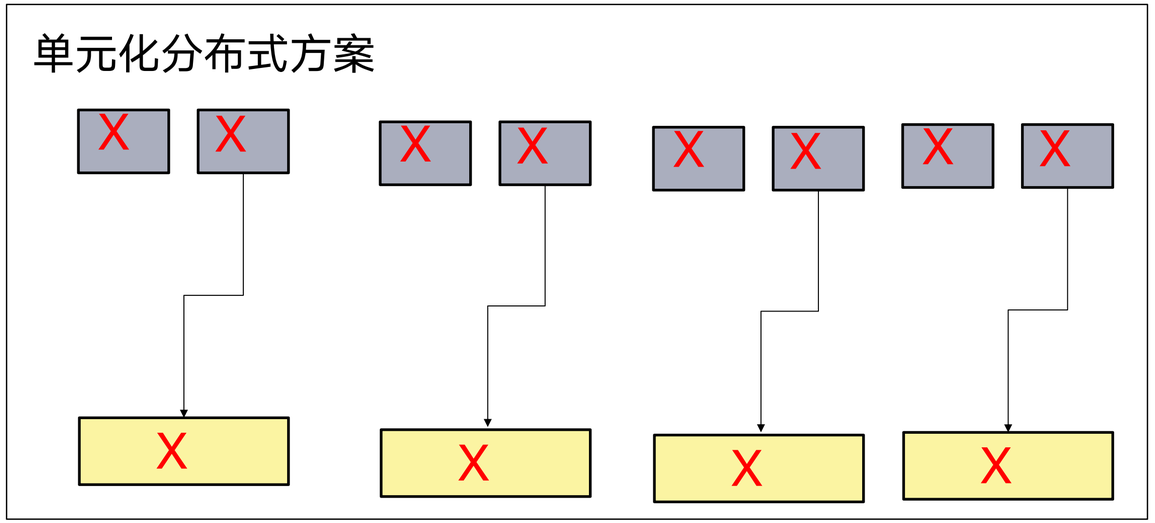
采用数据库中间件方案、分布式事务数据库方案时，碰到如下痛点不好解决：

- 在日常运行、灰度发布、扩容时，一旦出现某个数据库分片未知故障、应用未知故障等，会交叉关联影响到核心系统整体上下游的运行，大面积用户将可能受到影响，即**「爆炸影响半径」**很大。



仅数据层分片时，爆炸半径大，可能会影响所有客户

爆炸半径 = | 所有客户 |



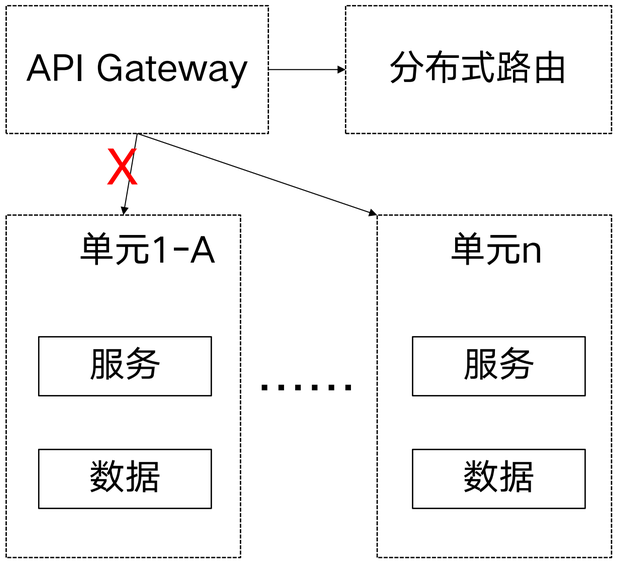
按客户分片时，爆炸半径变小，某个分片故障只影响部分客户。例如分成四个分片其中一个出现故障只影响 25% 客户。

爆炸半径 = | 所有客户 | / | 分片数量 |

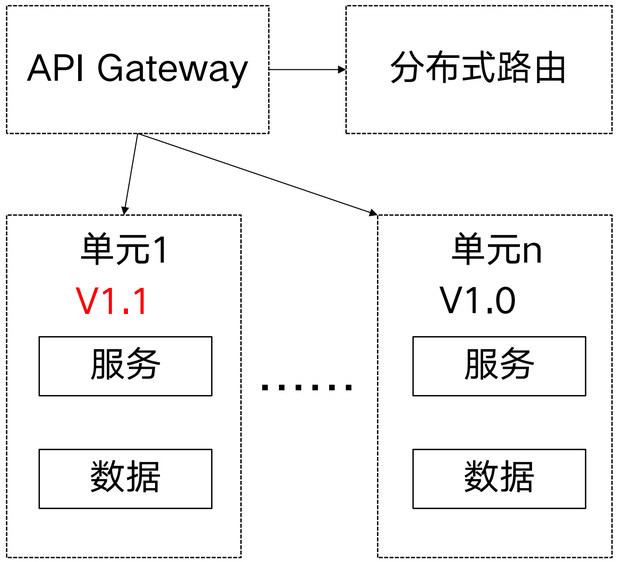
选择单元化分布式？（3/3）

基于「单元化分布式」方案，可以获得**单元故障隔离、单元灰度发布**的架构红利，如下图所示：

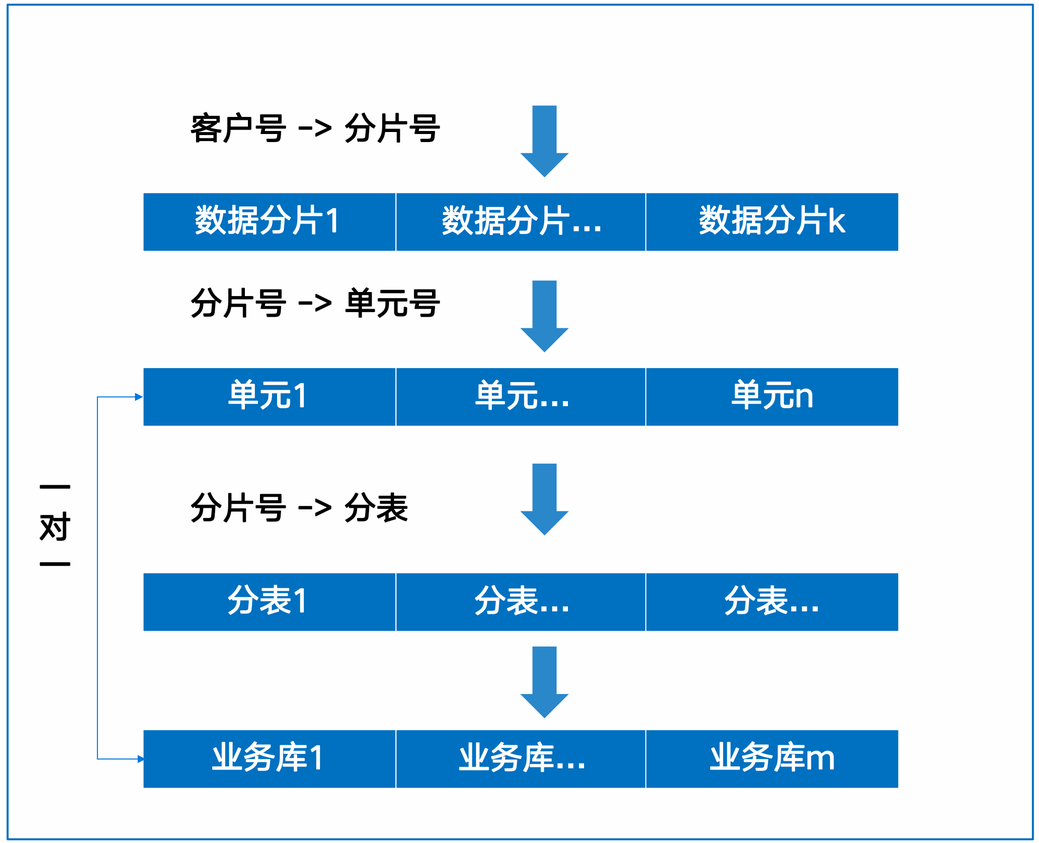
单元分片隔离



单元灰度发布



单元化分布式架构 >> 数据分区



分片数据：

- 客户业务数据，以客户为最小单位进行数据切分

非分片数据：

- 无法按客户维度进行切分的数据，如产品、定价等参数，以副本形式保留在每个单元中以提高访问性能

分片数据，基于「客户维度」进行分区，实现三层映射：

- 分片映射：「客户/账户/卡号」到「分片号」
- 单元映射：「分片号」到「单元号」的映射
- 分表映射：「分片号」到「分表」的映射

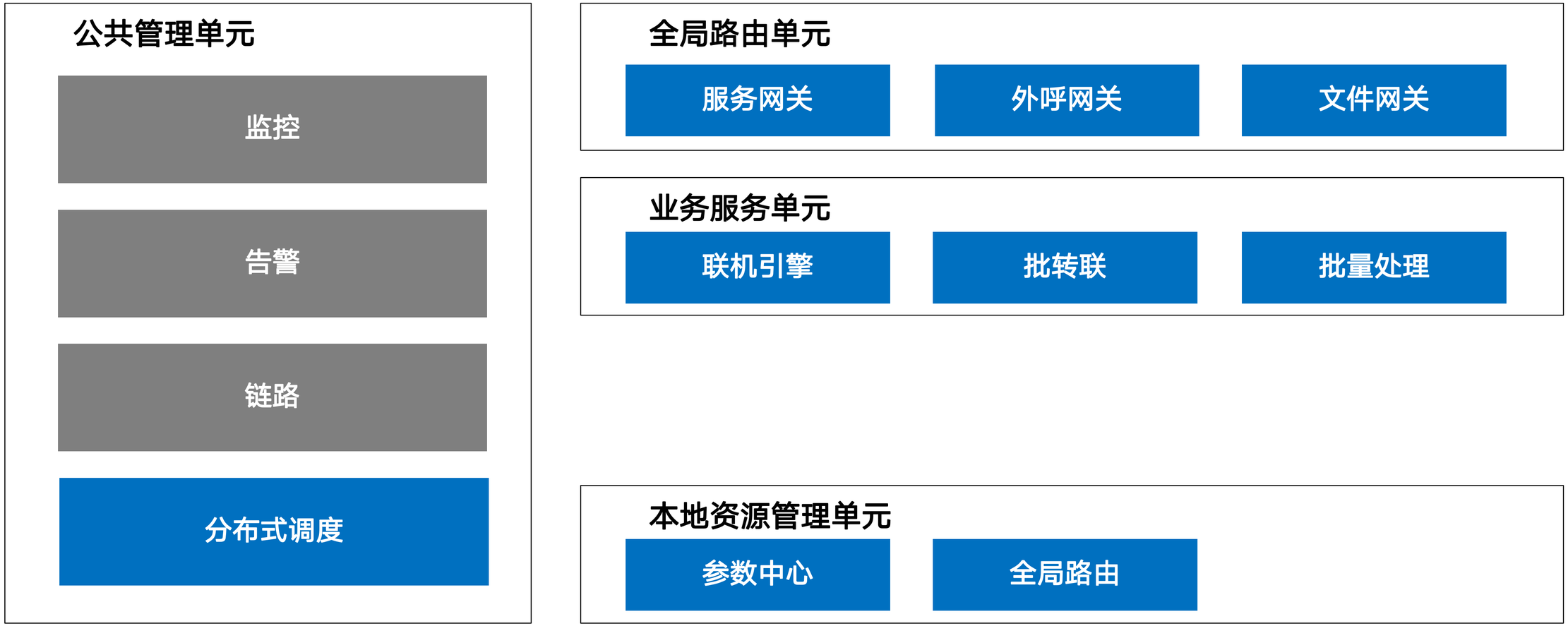
单元化分布式架构 >> 单元定义

通过按单元的标准化部署，实现按单元水平扩展，快速应急切换和故障隔离，定义有「四个单元」。

单元	备注
全局路由单元	外联系统请求与服务路由。
业务服务单元	业务服务单元，包括多个客户的业务数据以及为此类客户提供服务支持的应用实例，完成客户的业务处理。
本地资源管理单元	按 IDC 提供基础资源信息以及该数据的管理服务，包括映射数据、非客户维度数据（业务、技术参数）以及此类数据的管理服务。
公共管理单元	为全业务单元提供支撑的应用服务，以及为自动化运维管理平台相对接的相关数据，包括批量调度服务，日志聚合信息，监控数据等。

单元化分布式架构 >> 单元组件图

总体架构由「四个单元」组成，每个单元部署相关组件服务。



注：上图未包含「微服务框架」、「通用中间件」相关组件。

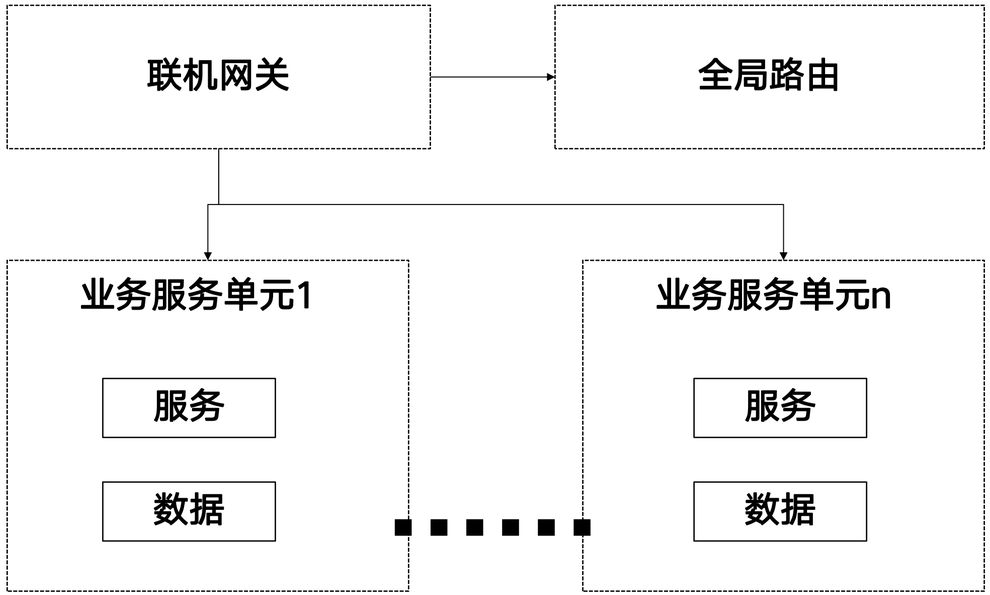
技术组件全景图

基于「微服务架构」和「单元化分布式架构」，按技术平台和应用框架区分技术平台组件。

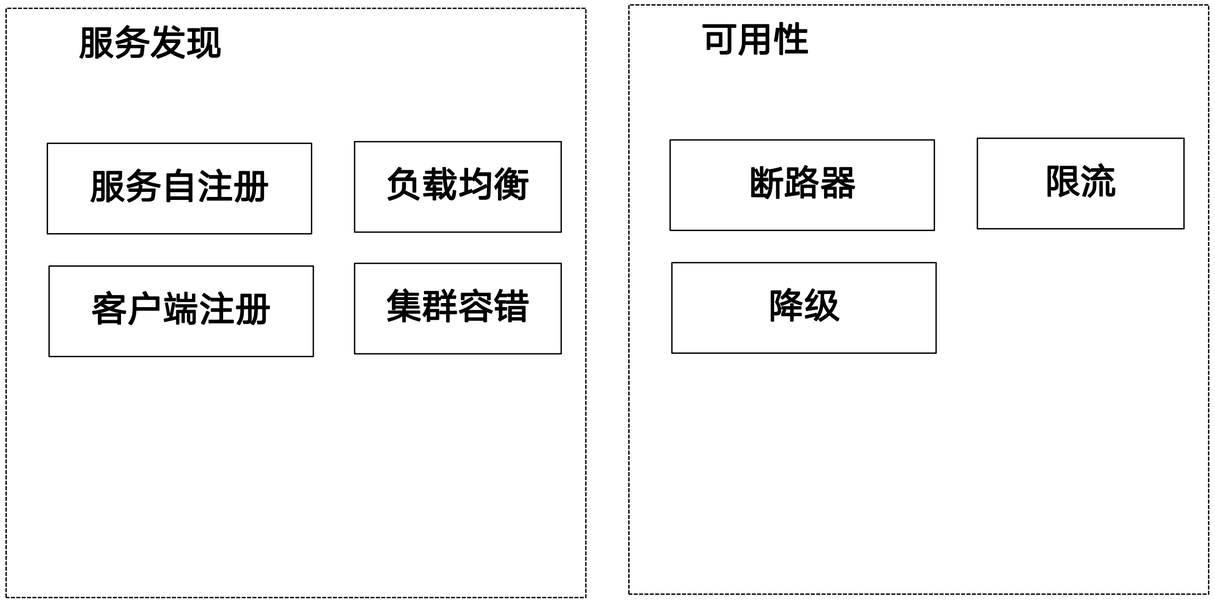
高可用和灰度发布

基于「单元」进行故障隔离、灰度发布

- 某个单元数据库未知故障、应用未知故障等未知故障时，只有本单元的用户会受到影响
- 可以选定灰度发布某个单元



基于「微服务框架」实现熔断、限流、降级、集群容错



技术平台特点

- 组件以开源、开源增强、自研为主
- 基于主流的技术架构，包括「单元化分布式架构」、「微服务架构」
- 承接企业建模落地

Spring 5

Spring Boot

微服务-开发框架

Spring Cloud

微服务生态-编程模型

OpenMessaging
开放消息标准
http://openmessaging.cloud

OpenAPI Specification

The World Standard for RESTful APIs

链路追踪标准
https://opentracing.io/

分布式新核心--整体架构设计--技术栈

微服务开发平台

序号	组件名称	技术栈
1	网关组件	spring cloud gateway 增强
2	外呼组件	自研
3	全局路由组件	自研
4	序号管理组件	自研
5	联机服务组件	自研（包括服务编排、服务引擎、分布式事务、分布式锁）
6	批量处理组件	自研
7	批量调度组件	自研
8	参数中心组件	自研

分布式微服务 + 通用中间件

序号	组件名称	技术栈
1	RPC 通信组件	Apache Dubbo 增强
2	注册中心	Nacos 增强
3	配置中心	Apollo 增强
4	消息中间件	RocketMQ
5	数据库访问中间件	ShardingSphere 增强
6	分布式文件存储	HDFS
7	技术SDK	Spring Boot
8	熔断限流	Sentinel
9	数据库	openGauss

第八周-作业实践

1. 【分组学习讨论环节】讨论这两个案例对自己现在设计或维护系统是否具有参考价值。

Q&A

谢谢各位

kimmking@163.com