



淘宝应用架构升级 反应式架构的探索与实践

淘宝 泽彬(许泽彬)



个人介绍

- 花名：泽彬
- 就职于阿里，经历：
 - 负责淘宝应用架构升级
 - 核心开发 – 建设用户增长设施与平台建设
 - 负责过分布式调用链跟踪框架 & 系统
 - 核心开发 – 分布式数据库同步系统
- Github：<https://github.com/zavakid>
- 开源项目：
 - otter 核心开发者：<https://github.com/alibaba/otter>
 - canal 核心开发者：<https://github.com/alibaba/canal>





CONTENTS

01 架构升级的效果

02 架构升级的思考

03 架构升级的实践



CONTENTS

01 架构升级的效果

02 架构升级的思考

03 架构升级的实践



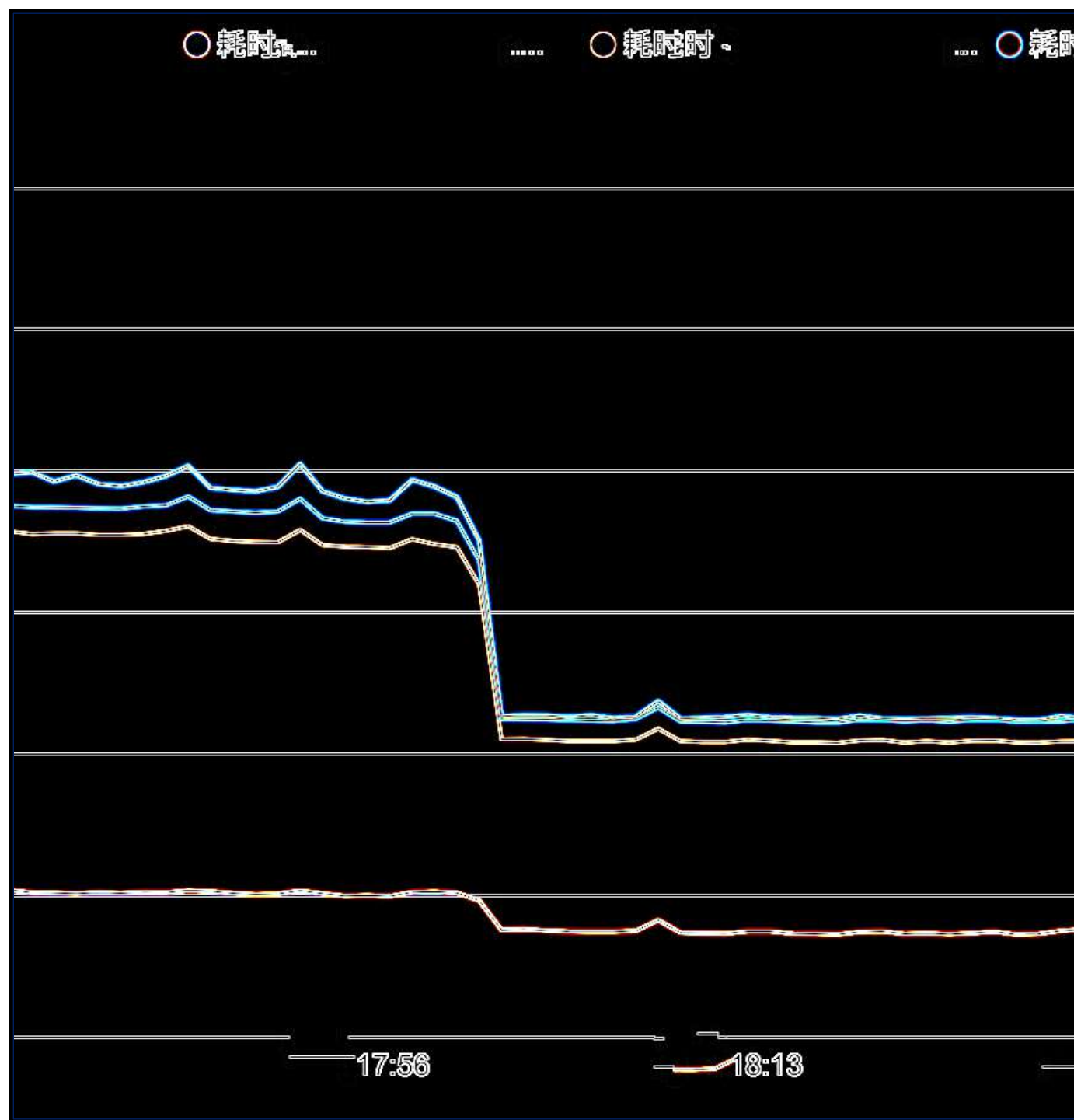
架构升级的效果

我的淘宝

- RT 降低 40%+
- QPS 提升 30%

猜你喜欢

- QPS 提升 90%+
- LOAD 下降 70%+



到底是怎样的**架构升级**
能达到这种效果？



CONTENTS

01 架构升级的效果

02 架构升级的思考

03 架构升级的实践



架构升级的思考

现有架构的问题？



现有架构的问题

同步等待

- 现有同步模型，线程多 load 高
- 资源利用率

应用本身的解决方案？

并行度有限

- 无法纯业务依赖并发
- 微服务化让问题更凸显
- RT 累积

RT 与 用户增长

RT 累积带来成本

- 过早引入 cache
- 每个服务都在设置超时

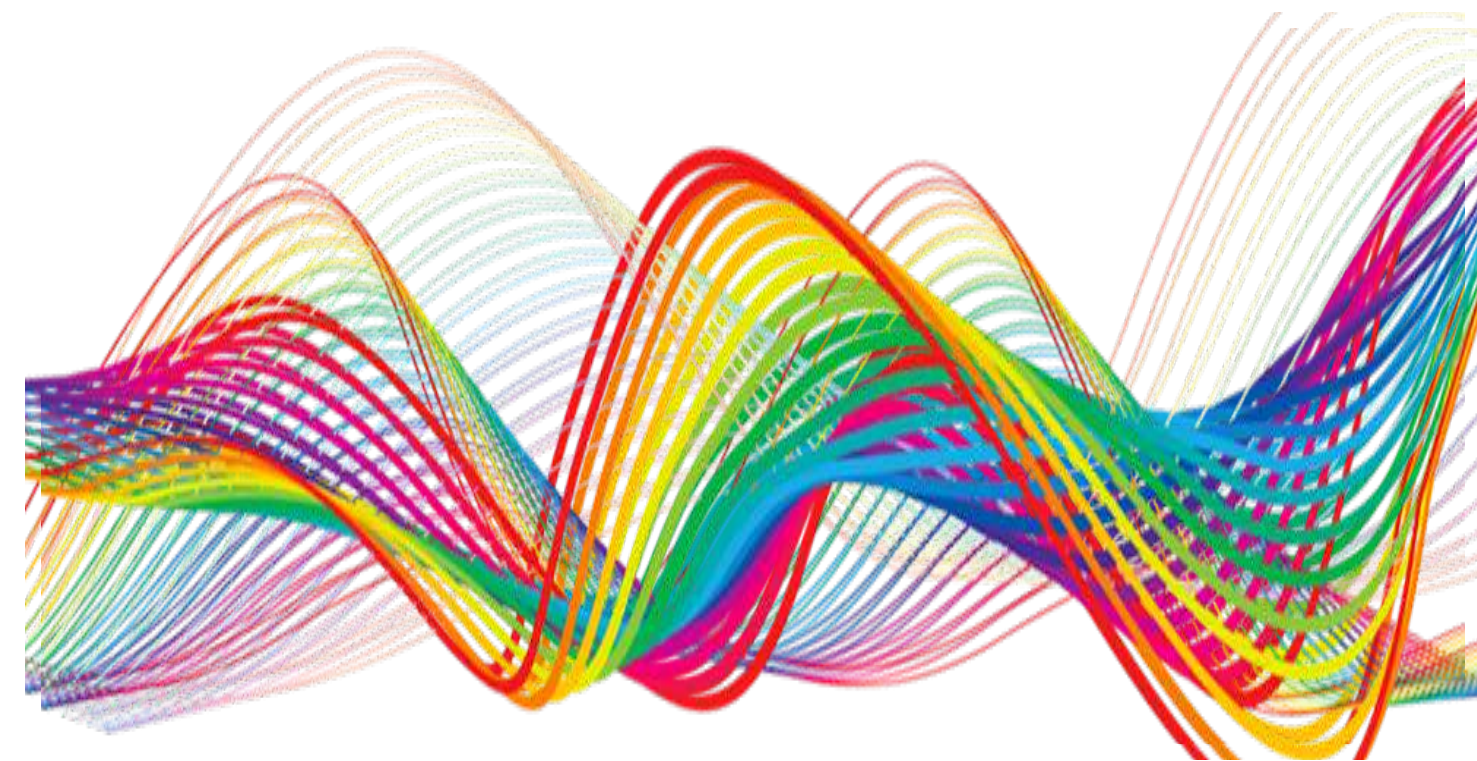
维护成本、业务实现复杂化



天生异步的架构

流

Stream / Flow / Flux





什么是流

流定义

- 一个流是顺序串行执行的
- 多个流之间才可能是并行的

面向流

- 面向数据
- 操作、组合

数据 vs 逻辑

- 业务逻辑 → 数据变换
- 数据变换 → 业务逻辑

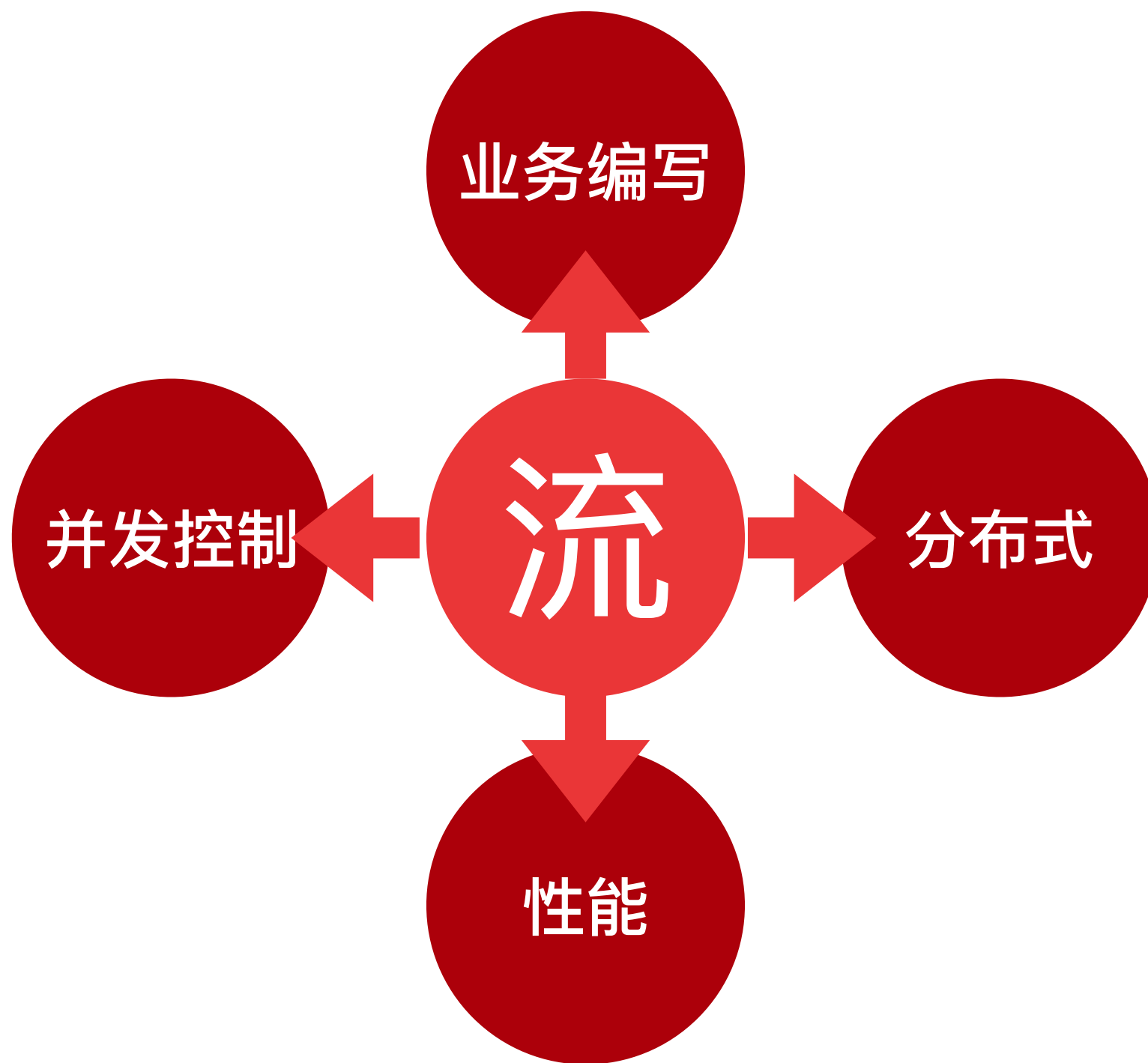
面向流编程是
面向数据编程



Your Mouse is a Database!
— Erik Meijer



流式架构





流 – 业务编写

大量强大的操作符

创建

just / from* / range /
repeat / interval / timer

过滤

filter / take / skip / distinct
sample / debounce / throttle

转换

map / buffer / window / scan /
flatMap / groupBy

组合

merge / concat / startWith /
zip / switch*

声明式表达

完备、更高级、更快捷

```
requestFlow  
  .buffer(10, MILLISECONDS, 16)  
  .flatMap(service::batchOps)
```



流 – 并发控制

多流并发

- 不同的流
- 无依赖
- 切过 Scheduler

就可以自动并发

优雅控制

- 业务按照语义顺序编写
- 更友好的控制并发

维护性 和 性能 更优!

业务顺序编写 并发方式执行

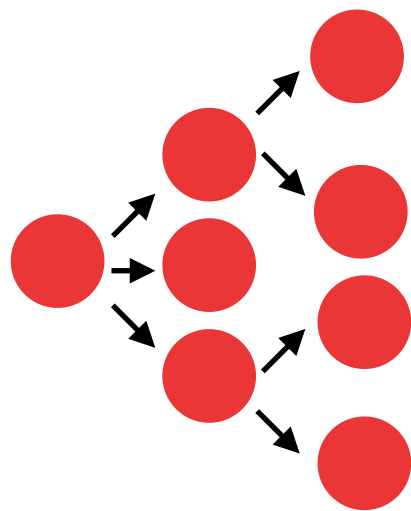
切过 Scheduler是指:

- 远程调用已经异步化, 所以是已经且过 Scheduler
- 可以手工切 Scheduler (subscribeOn / observeOn)

```
processors.flatMap(processor -> processor.process(request))  
  .filter(Result::isSuccess)  
  .map(Result::getData)  
  .reduce(0, (acc, data) -> acc + data);
```

> 流 - 性能 全异步、流式

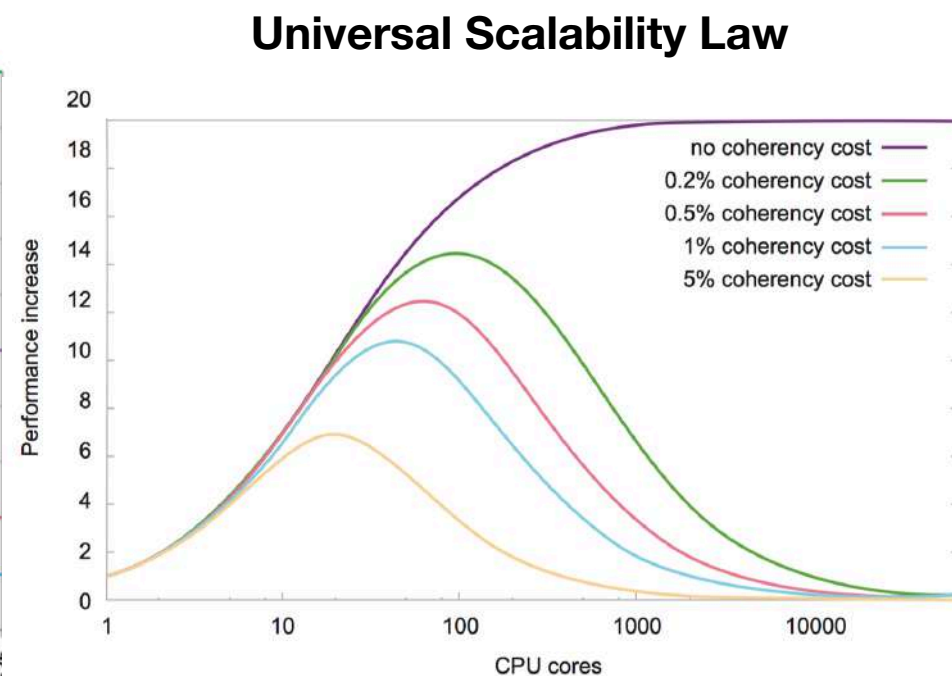
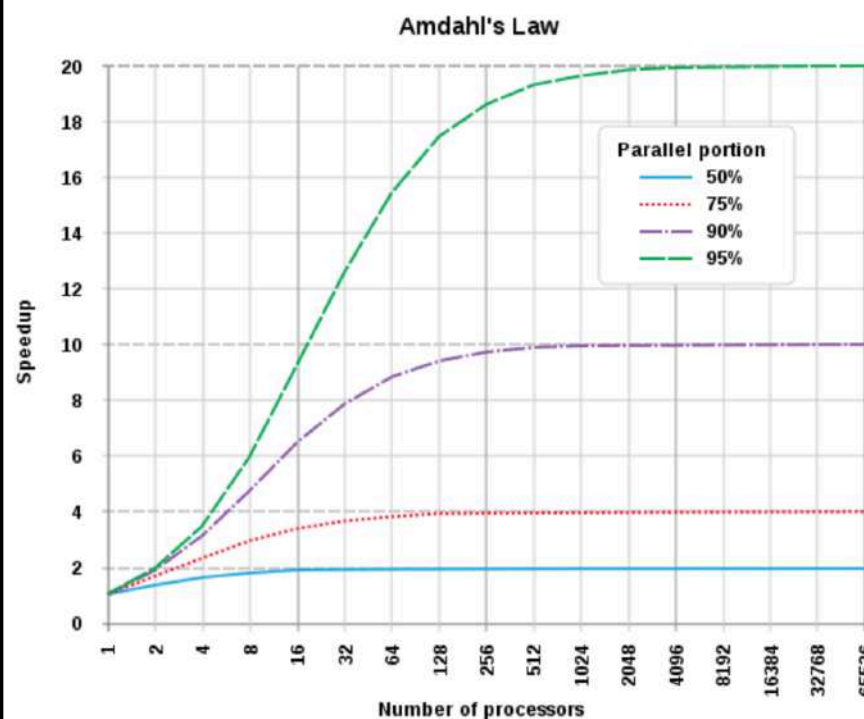
业务等效依赖的异步并发



系统流水线并行处理



更高效的资源利用率



$$S(n) = \frac{T(1)}{T(N)} = \frac{1}{\alpha + \frac{1-\alpha}{N}} = \frac{N}{1 + \alpha(N-1)}$$

CPU数个业务线程

更少的上下文切换、更少(无)的竞争、更低的LOAD

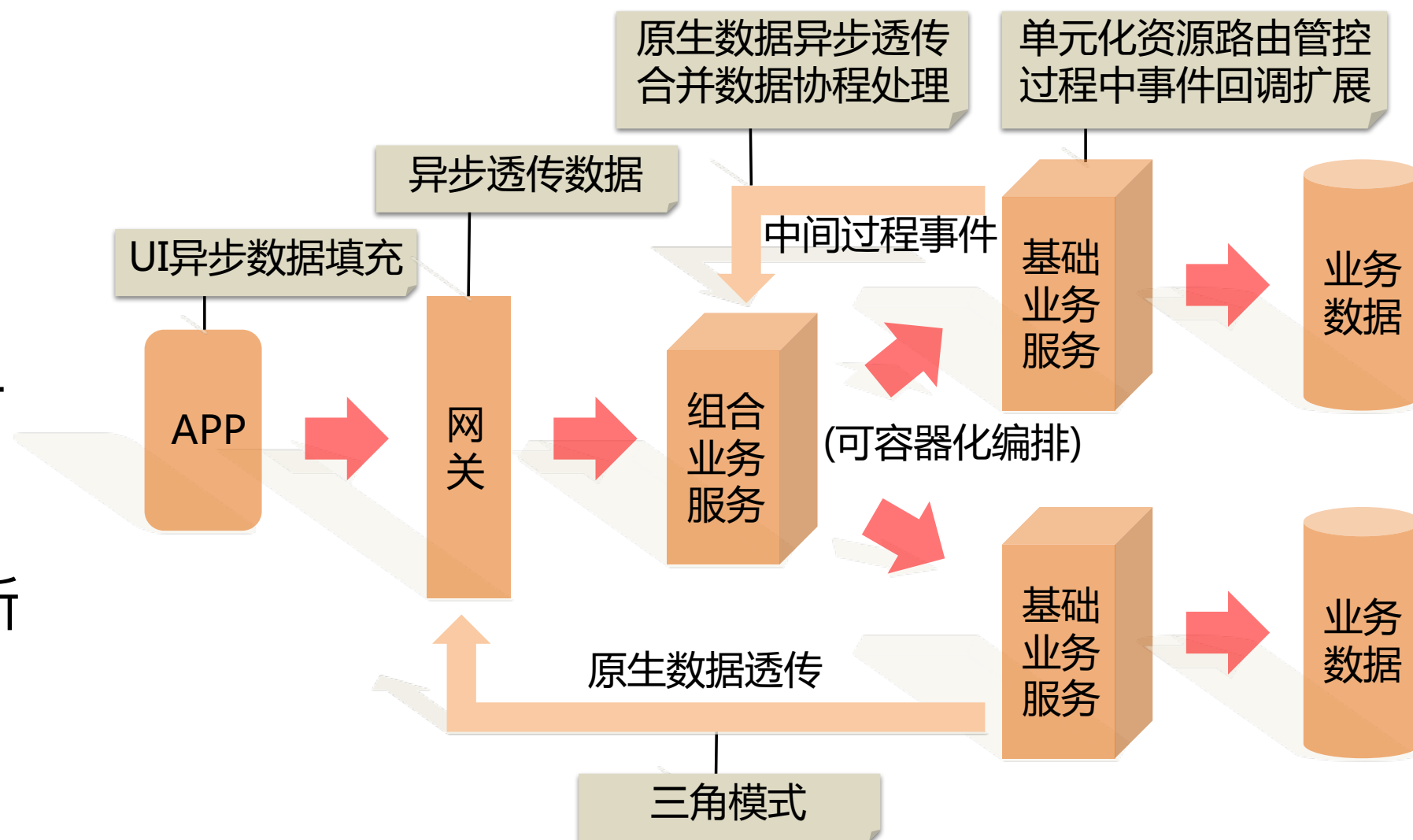


流 - 分布式

流引用可被远程化 系统级的流式贯通

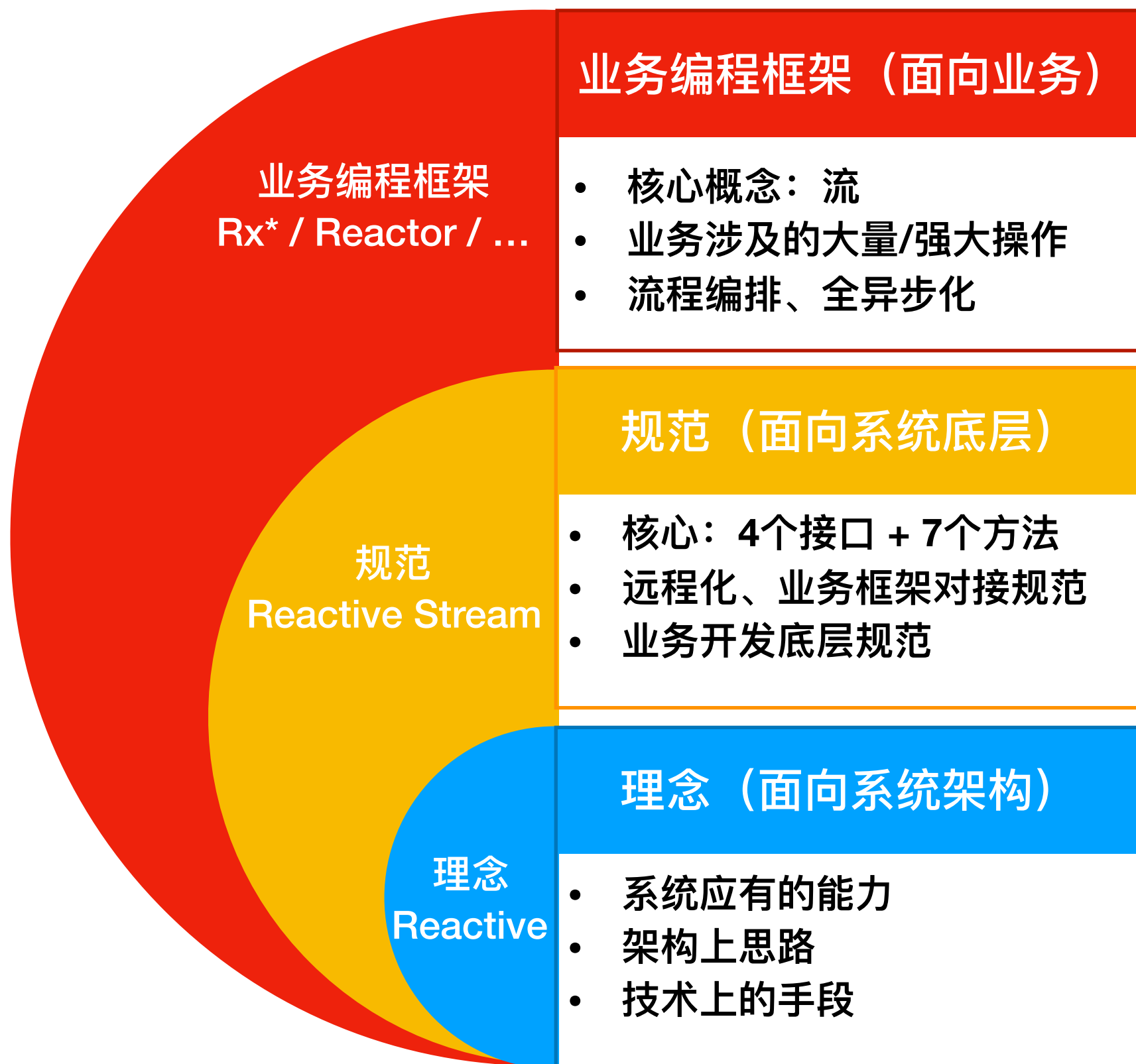
架构治理能力加强:

- 回压
- 三角模式透传
- 业务快速截面创新
- ...





流的概念分层



> 为什么现在可以做升级

语言的支持: Java 8 普及

- * lambda 支持
- * 业务开发人员对 lambda 的接受度大大提高

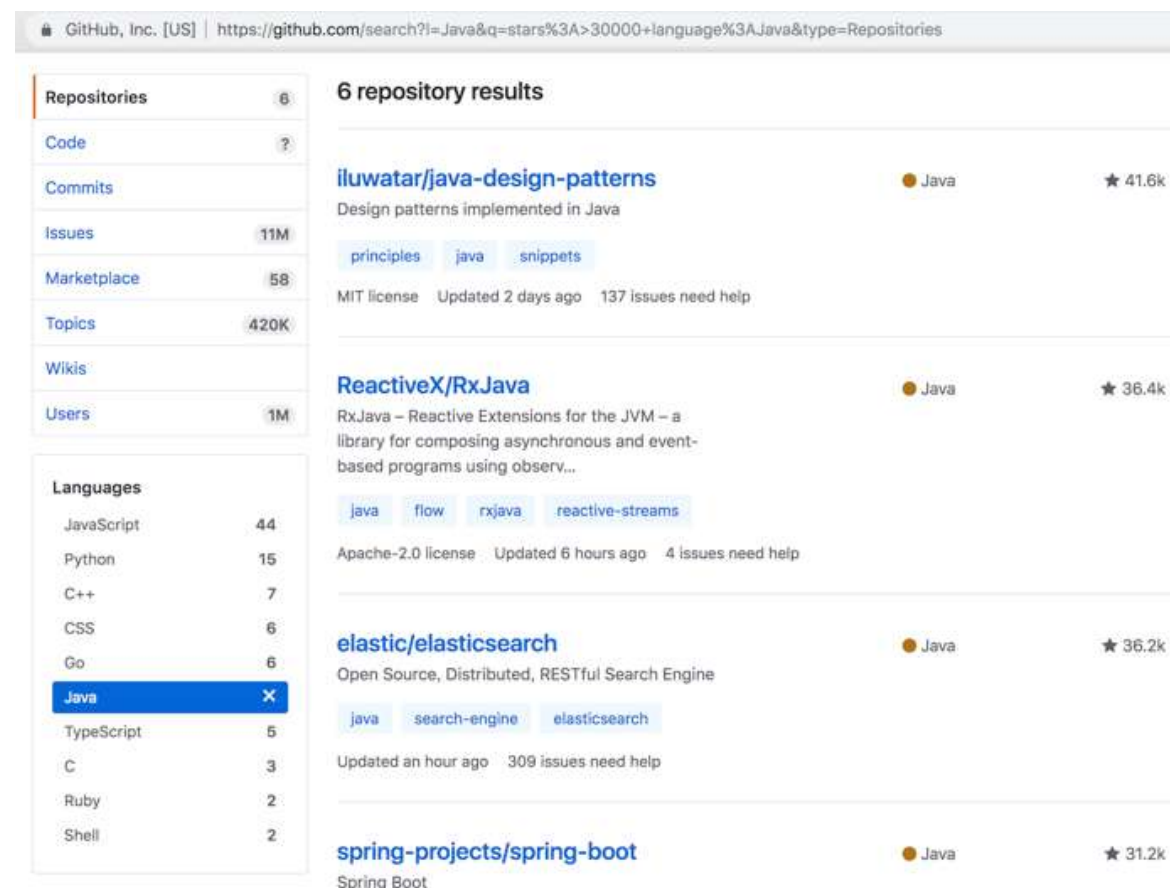


Reactive 业务框架成熟

- * Github 上 RxJava Star 36k+, Java 语言 Star No.2 项目

业界拥抱 Reactive 理念

- * Reactive Stream 已经转正
已经纳入在 Java 9 的 `java.util.concurrent.Flow` 中
- * Spring5 、 Spring Boot 2 拥抱 Reactive



[github 11月份数据](#)



CONTENTS

01 架构升级的效果

02 架构升级的思考

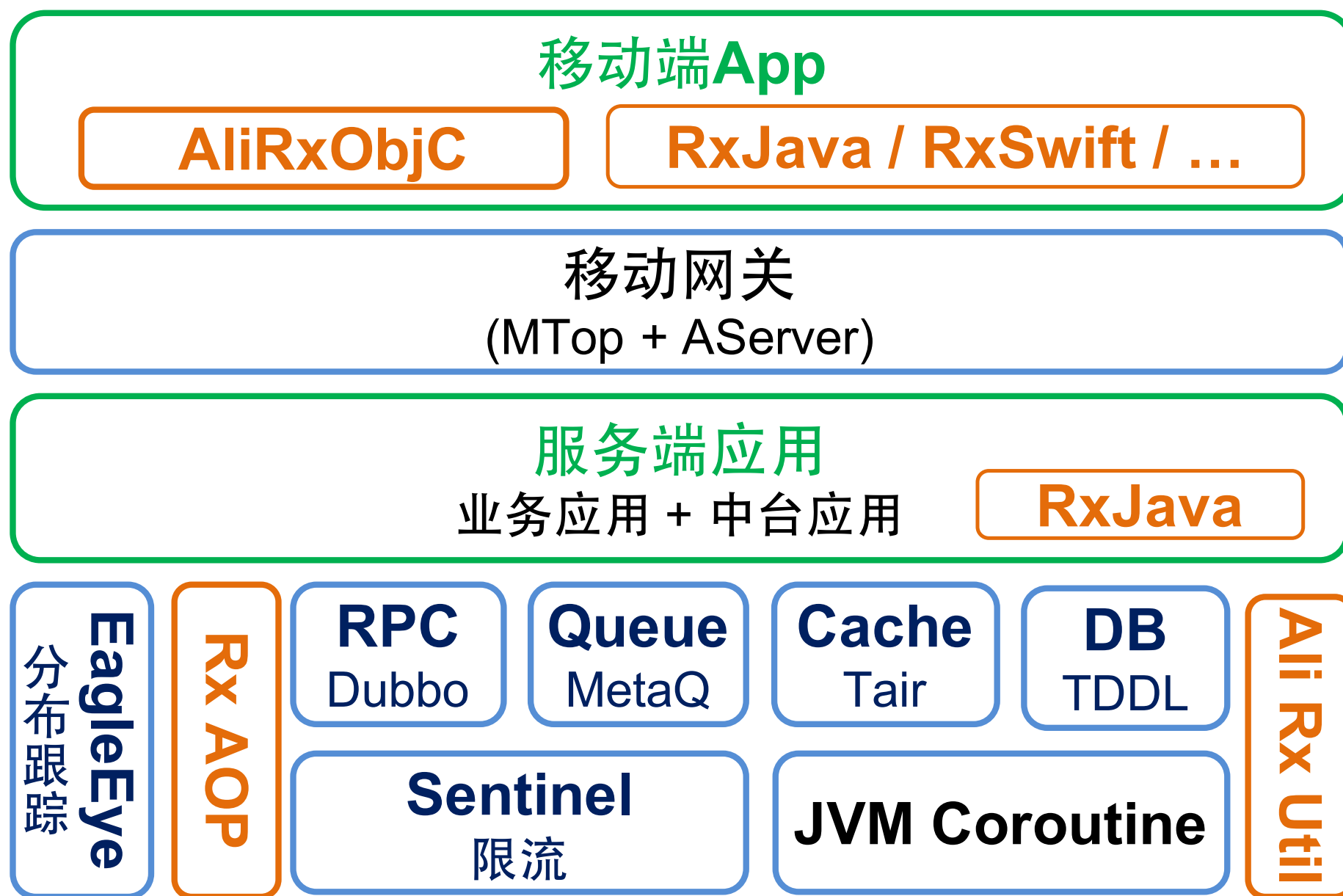
03 架构升级的实践



应用技术架构图







升级事项:

- 编程框架
- 中间件
- 业务





编程框架选型

	Rx* (RxJava/RxJS/RxSwift)	Project Reactor	Akka Stream
特点	<ul style="list-style-type: none">* 出现早（7年前）、已发布 V2* 在 Android 广泛使用，有用户基础* 全栈：客户端 + 服务端 + 前端* 活跃度：RxJava Github Java 排名 No. 2	<p>出现较晚（3年） Spring 加持，有发展前景</p>	<p>Scala，用户基础少、普及难度大</p>
稳定性			
普及性			
切换成本	<p>业务框架不会锁定：有 Reactive Streams 规范 / 不同业务框架可互通 后期业务框架切换成本不高：不同业务框架之间互相模仿 / 使用和概念上差别不大</p>		



中间件升级

1. 服务框架(RPC)

流式支持会在 **开源的 Dubbo3** 放出

2. (移动)网关(Geatway)

- * 先使用适配

接口不变、分段实施、不返回 Flowable

- * 更自然的流式

支持业务直接返回 Flowable

3. 缓存(Cache)

4. 消息(Queue)

- * 天然异步

- * 已有集成，或集成成本低

5. DB (JDBC) (Block)

- * 用 Ali JVM协程 异步集成

- * 或用线程池异步集成

6. 限流组件

7. 分布式跟踪系统

解决业务异步/回调 引入的 上下文传递问题

8. iOS Objective-C 的 Rx 框架

实现 AliRxObjC (**会开源**)



业务实施

1. 先面向应用级升级

- * 应用实施升级
- * 聚焦性能做 case
- * 补齐设施能力
- * 积累业务升级改造经验

2. 架构级升级

- * 规模化后，架构级别收益会显现

业务架构升级case 的选择策略

1. 集群/流量大/核心有认知
 - * 方便看升级改造效果
2. 瓶颈 match 升级
(如 CPU 压不满、IO较多、高Load)
 - * 可以**确定获得**大比例收益
3. 业务逻辑简单优先
 - * 尤其在改造起步阶段，降低难度
4. 业务压力不大
 - * 有业务升级改造资源投入

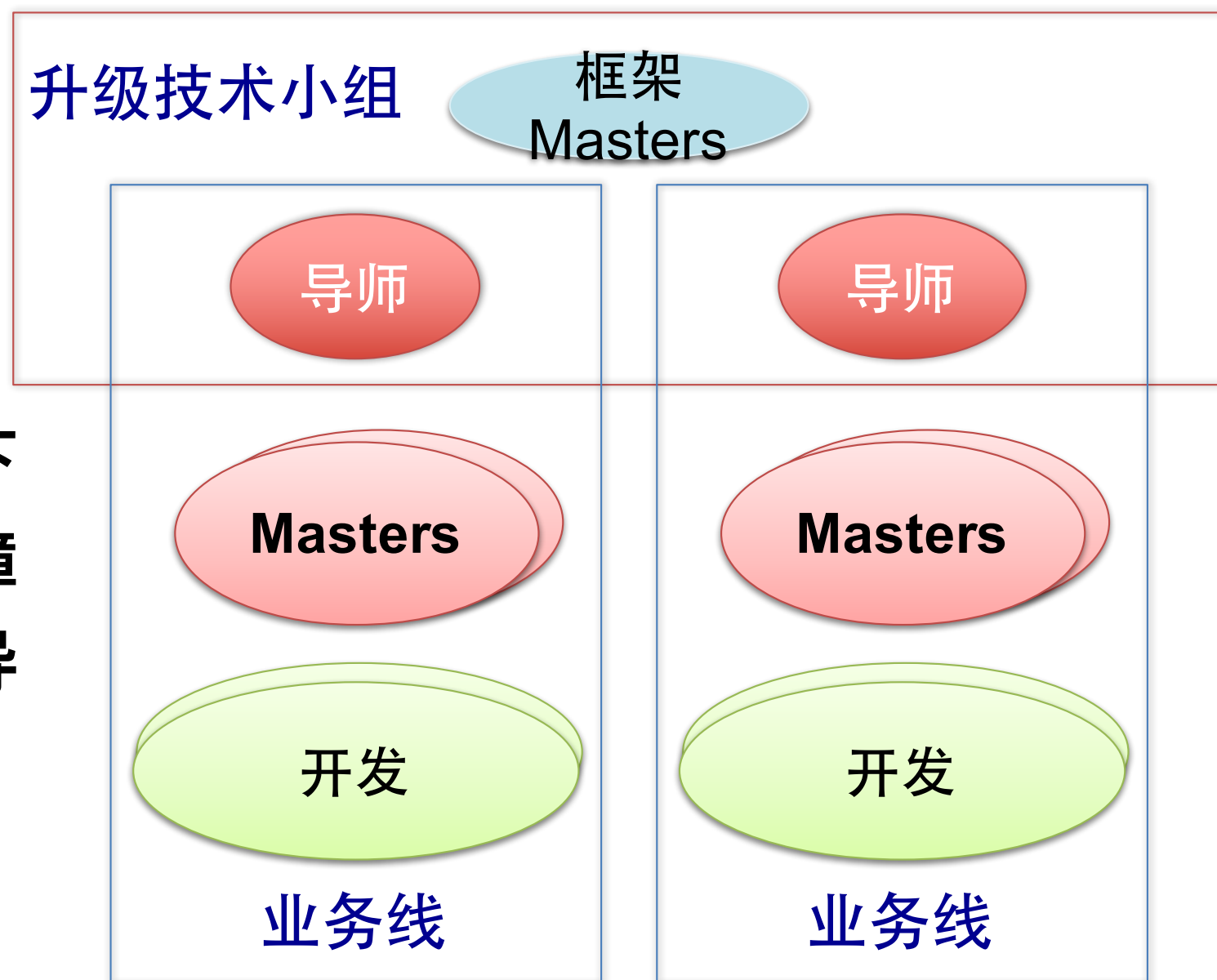


实施难点和策略

- 团队
 - 缺乏技能
 - 缺乏意识
- 业务
 - 缺乏动力
- 技术
 - 缺乏工具



- 自上而下
- 组织保障
- 全员宣导





解决方案支持 — 执行治理（2018双11）

- 线程模型（执行治理）：业务应用 极简高效 的线程模型，统一线程池
 - RPC、缓存等中间件 线程池设置入口，由应用架构统一管控
 - 整个应用 CPU数个业务线程
 - 涉及线程池：EventLoop / Provider / Consumer / 阻塞操作 的线程池
 - 进一步提升性能
- 阻塞检测（异步配套）：提早发现问题，降低全异步升级成本和风险
 - 基于 基础软件 AliJDK(JVM团队) 提供JVM级API的阻塞点检测（业界领先）
 - 基于 非阻塞的任务(计算线程)的执行时长
- 统一的线程上下文传递方案（业务&中间件）
 - 应用不再需要关心 不同的中间件 和 业务提供的不同方案
 - 提供方案，网关、RPC、分布式调用跟踪、RxJava等所有中间件 已统一接入



目标与规划

- 实现**分布式回压**(backpressure)
 - 保证整体系统稳定不过载且充分利用资源
 - 2018双11有试点实验，后续重点
- 通过回压把应用集群压力反馈到基础设施
 - 更精准有效的应用**弹性调度**
- 实现**全异步/流式**为核心的**服务框架**
- 考虑引入 **Kotlin 协程**
 - 符合现在过程式的编程习惯（非 FP 风格）



支撑下一个十年业务的架构!



关注『淘宝技术』公众号

THANKS

DDD CHINA