架构实战营 模块三

第6课：消息队列备选架构选择和细化实战

李运华

前阿里资深技术专家（P9）

教学目标

1. 通过案例学习备选架构评估方法
2. 通过案例学习如何细化架构方案

橘生淮南则为橘，生于淮北则为枳！

目录

1. 备选架构评估

技术背景

1. 中间件团队规模不大，大约6人左右。

2. 中间件团队熟悉 Java 语言，但有一个同事 C/C++ 很牛。

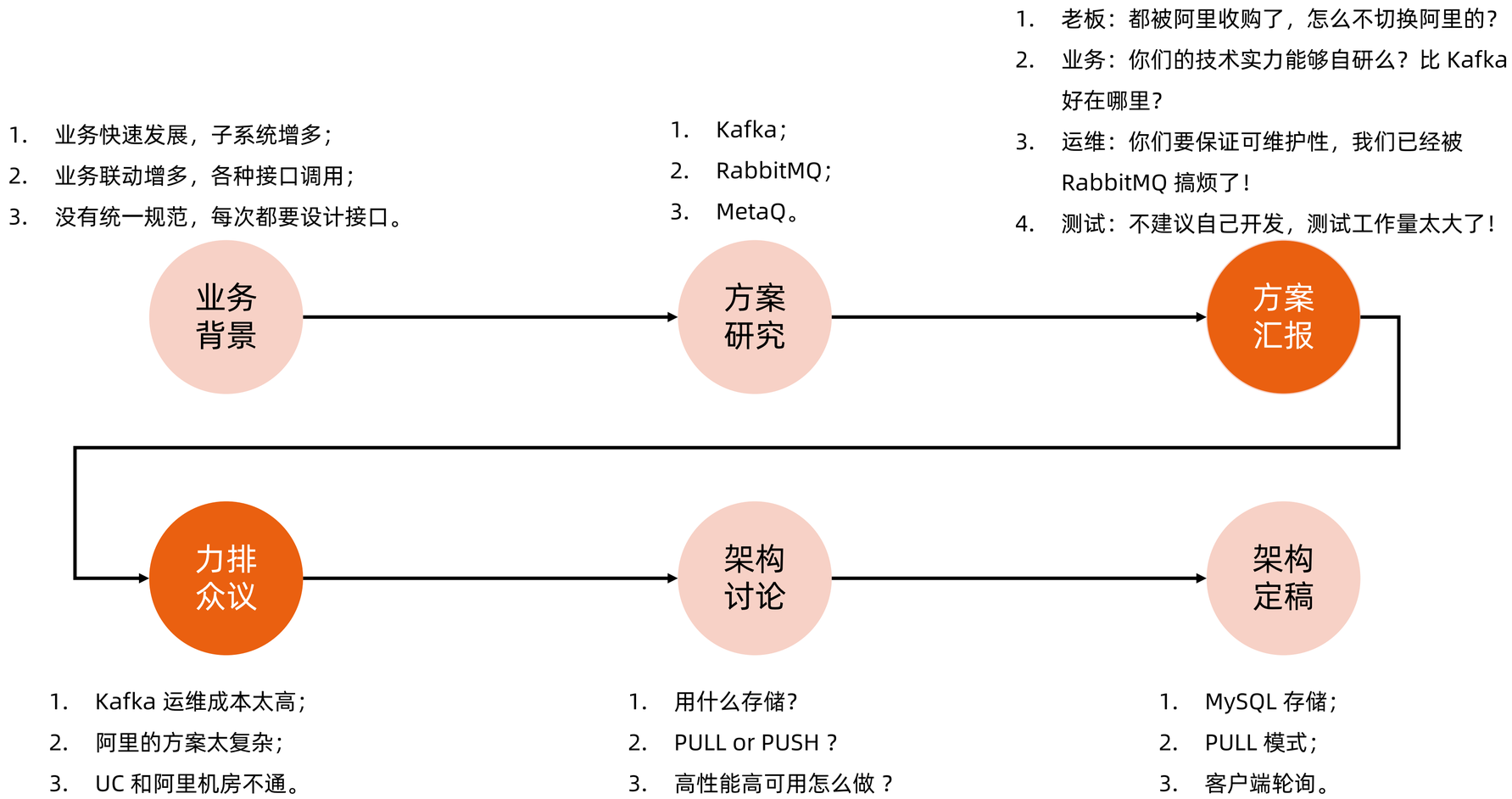
3. 开发平台是 Linux，数据库是 MySQL。

4. 目前整个业务系统是单机房部署，没有双机房。

5. 刚刚被阿里以创纪录的金额收购。

这些背景条件，每一条都可能影响架构设计。

飞鸽消息队列架构设计过程

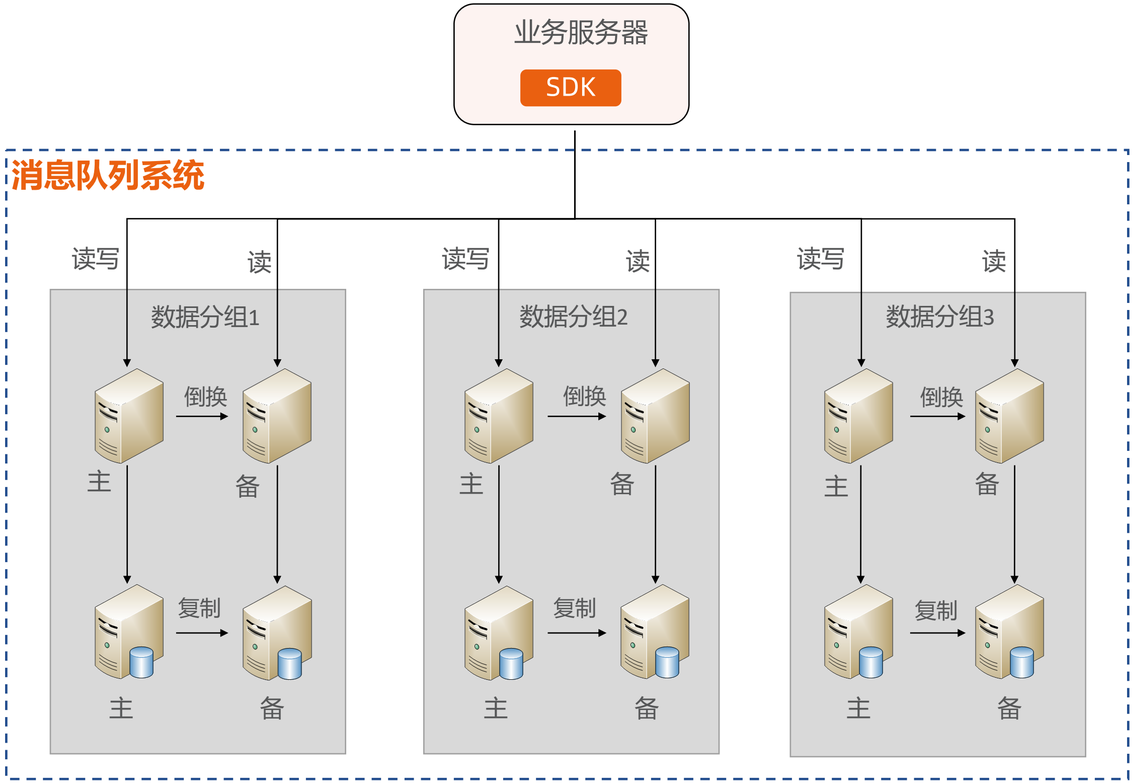


备选架构1 - 开源方案评估

- 人力成本
  - 测试代表倾向于引入 Kafka，因为 Kafka 比较成熟，无须太多测试投入。
  - 中间件团队部分研发人员也支持使用 Kafka，因为使用 Kafka 能节省大量的开发投入。
- 可维护性
  - Kafka 是 Scala 语言编写的，运维团队没有维护 Scala 语言开发的系统的经验，出问题后很难快速处理。
  - 运维团队已经有一套成熟的运维体系，包括部署、监控、应急等，使用 Kafka 无法融入这套体系，需要单独投入运维人力。
- 业务场景
  - 部分人员认为 Kafka 可能并不适合我们的业务场景，Kafka 是大容量的日志消息传输，而我们的消息队列是为了业务数据的可靠传输。
- 学习成本
  - 业务主管倾向于采用 Kafka 方案，因为 Kafka 已经比较成熟，各个业务团队或多或少都了解过 Kafka。

前面课程说“可维护性”是等备选方案选出来后再设计，为何这里要评估“可维护性”？

备选架构2 - 自研集群 + MySQL 存储



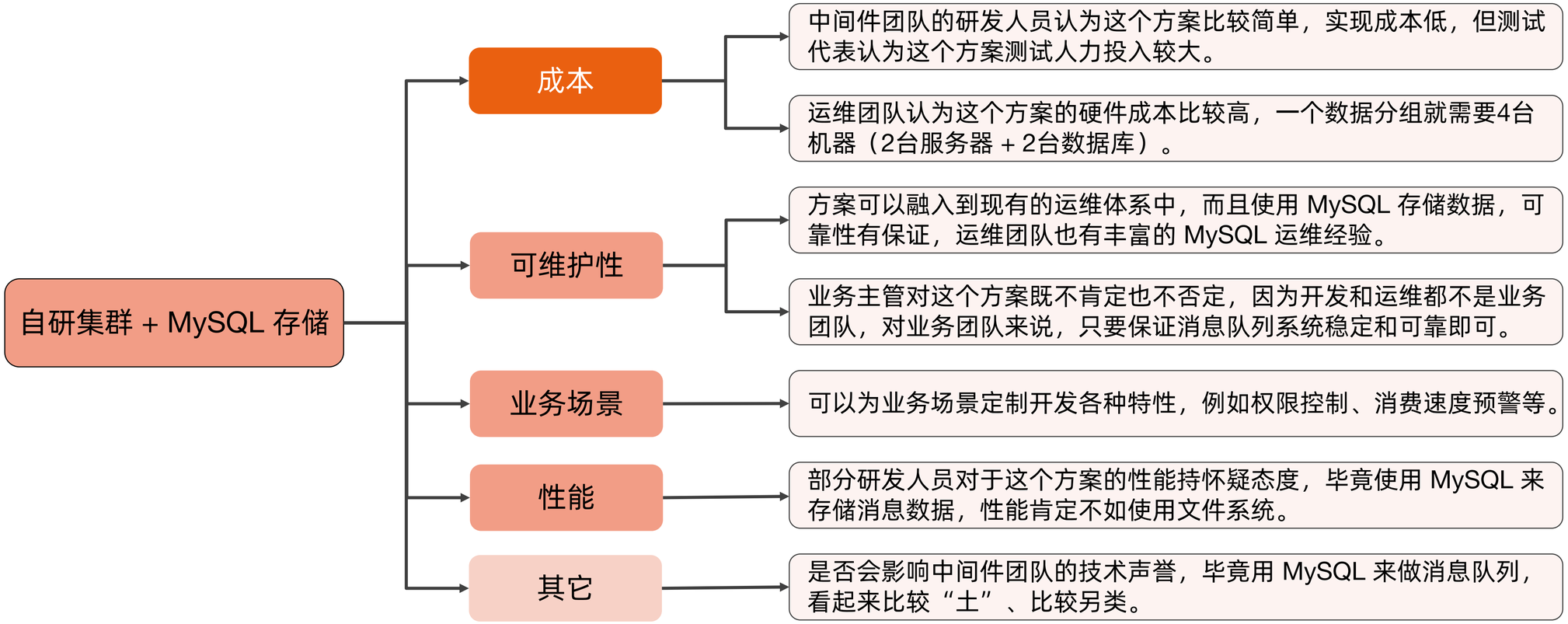
【简单描述】

1. Java 语言编写消息队列服务器；
2. 消息存储采用 MySQL；
3. SDK 轮询服务器进行消息写入；
4. SDK 轮询服务器进行消息读取；
5. MySQL 双机保证消息尽量不丢；
6. 使用 Netty 自定义消息格式，并且支持 HTTP 接口。

【更多说明】

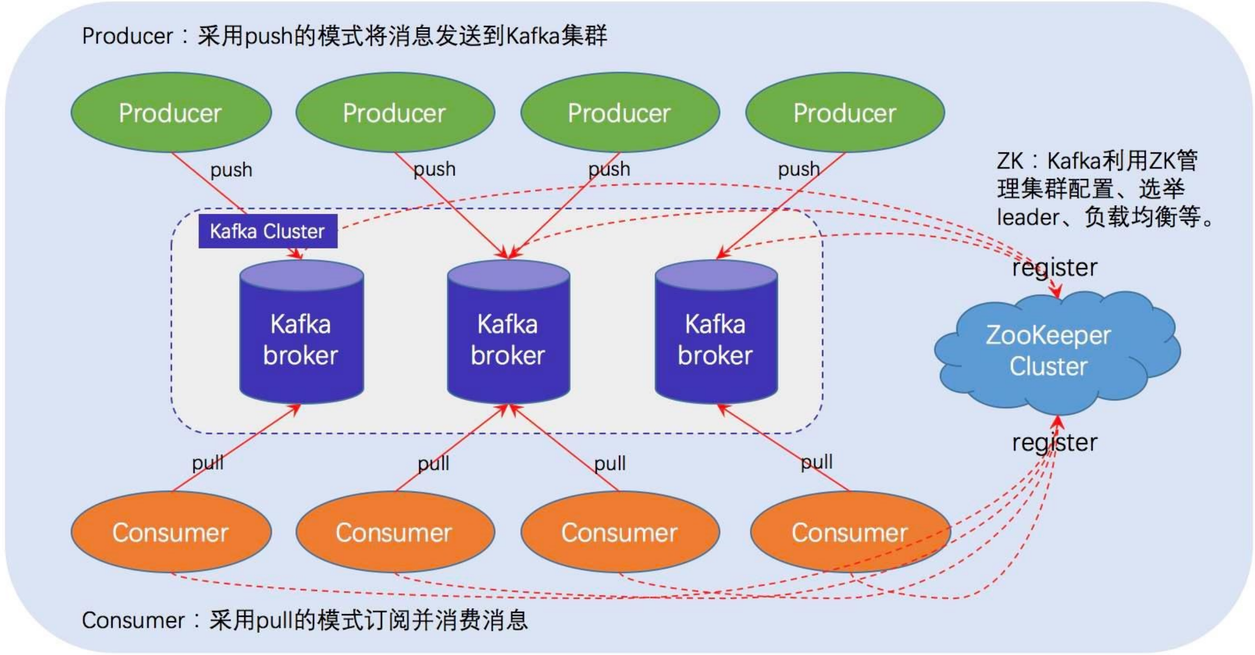
1. 方案可以变化，例如底层存储用 HBase（类似 OpenTSDB）、Redis。

备选架构2评估



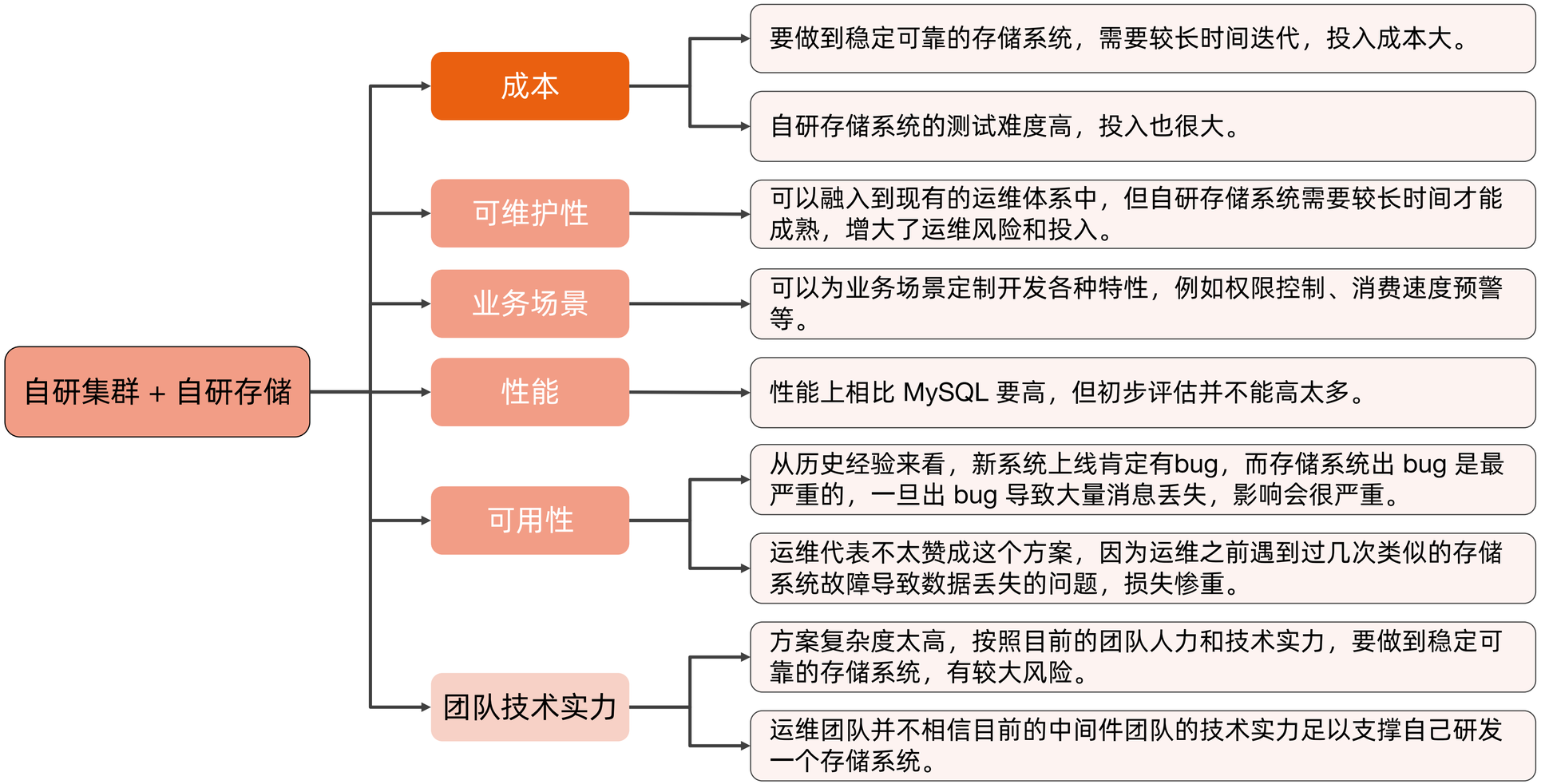
用 MySQL 来做消息队列存储，会影响技术声誉么？

备选架构3 - 自研集群 + 自研存储

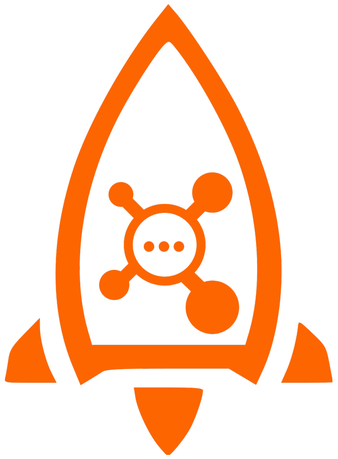


1. 模拟 Kafka 的原理，用 Java 语言实现，也可以用 LSM 数据结构来存储消息。
2. 可以保证高可用高性能。
3. 加上可维护性的各种能力，嵌入到已有的运维体系。

备选架构3评估


自研集群 + 自研存储
成本
要做到稳定可靠的存储系统，需要较长时间迭代，投入成本大。
自研存储系统的测试难度高，投入也很大。
可维护性
可以融入到现有的运维体系中，但自研存储系统需要较长时间才能成熟，增大了运维风险和投入。
业务场景
可以为业务场景定制开发各种特性，例如权限控制、消费速度预警等。
性能
性能上相比 MySQL 要高，但初步评估并不能高太多。
可用性
从历史经验来看，新系统上线肯定有bug，而存储系统出 bug 是最严重的，一旦出 bug 导致大量消息丢失，影响会很严重。
运维代表不太赞成这个方案，因为运维之前遇到过几次类似的存储系统故障导致数据丢失的问题，损失惨重。
团队技术实力
方案复杂度太高，按照目前的团队人力和技术实力，要做到稳定可靠的存储系统，有较大风险。
运维团队并不相信目前的中间件团队的技术实力足以支撑自己研发一个存储系统。

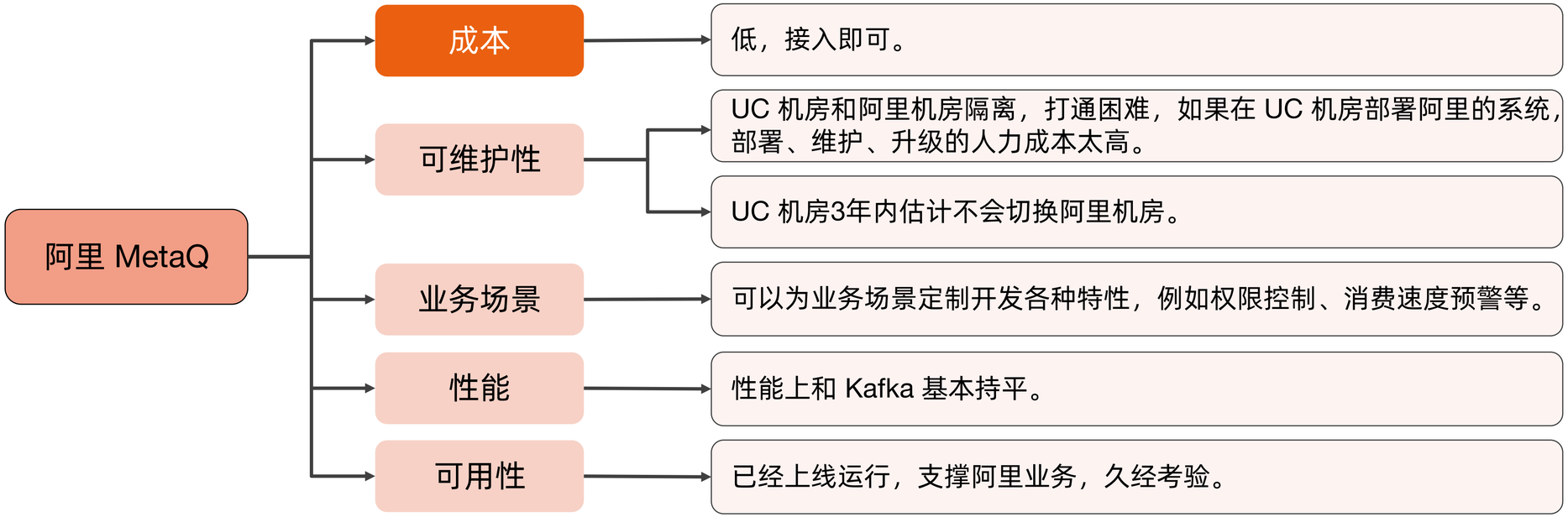
备选架构4 - 直接用阿里的 MetaQ



RocketMQ

1. 模拟 Kafka 的原理，用 Java 语言实现。
2. 刚刚被阿里收购，“自己人”，有什么需求可以提给他们来改。
3. 加了很多牛逼的功能，比 Kafka 更强大。

备选架构4评估



和阿里有关的几个评估结论，事后回过头来看是错的，你能看出是哪几个么？

360度评估结果汇总

极客时间 训练营

评估维度	1. 引入 Kafka	2. 自研集群 + MySQL 存储	3. 自研集群 + 自研存储	4. 接入 MetaQ	最优方案
性能	高	较高	高	高	1、3、4
复杂度（团队技术实力）	低	中	高	低	1
硬件成本	低	较高	低	低	1、3、4
人力成本	低	中	高	低	1、4
可维护性	低	高	中	低	2
可用性	高	高	中	高	1、2、4
业务契合度	低	高	高	中	2、3
团队声誉	低	中	高	低	3

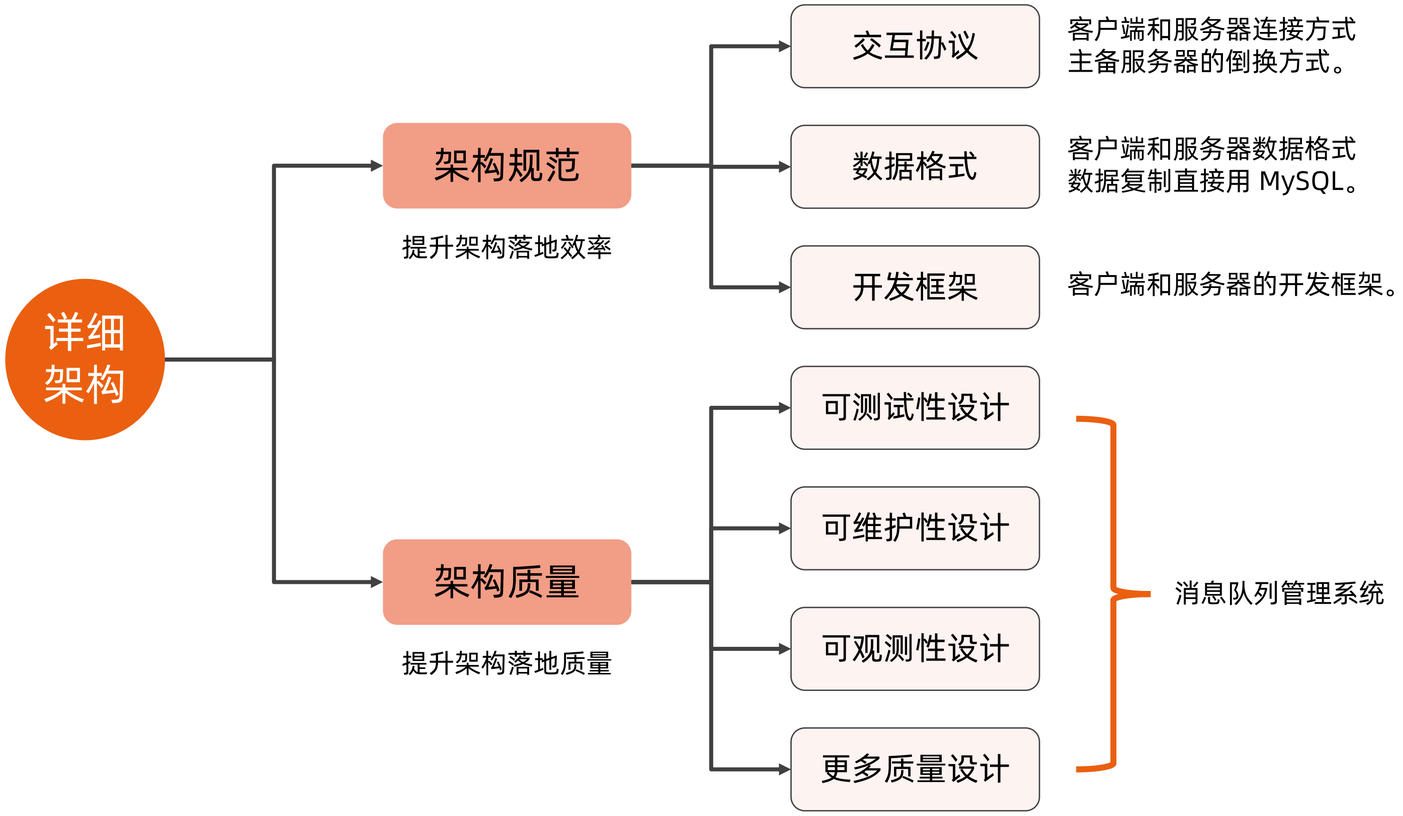
架构决策

确定排序规则：1. 可用性；2. 可维护性；3. 人力成本。

评估维度	1. 引入 Kafka	2. 自研集群 + MySQL存储	3. 自研集群 + 自研存储	4. 接入 MetaQ	架构决策
性能	高	较高	高	高	按照合适原则，系统不需要太高性能，方案2能够满足
复杂度(团队技术实力)	低	中	排除，自研存储的复杂度和风险太大	低	排除3
硬件成本	低	较高	低	低	实际计算下来，方案2只需要不到20台机器
人力成本	低	中	高	低	这里要给老板一个大概的评估
可维护性	排除，无法嵌入到已有运维体系	高	中	排除，因为机房网络问题和机房规划问题没法解决或者成本太大	排除1、4
可用性	高	高	排除，自研存储的复杂度和风险太大	高	排除3
业务契合度	低	高	高	中	NA
团队声誉	低	中	高	低	NA

2. 架构方案细化

详细架构内容


详细架构
架构规范
提升架构落地效率
交互协议
客户端和服务器连接方式
主备服务器的倒换方式。
数据格式
客户端和服务器数据格式
数据复制直接用 MySQL。
开发框架
客户端和服务器的开发框架。
架构质量
提升架构落地质量
可测试性设计
可维护性设计
可观测性设计
更多质量设计
消息队列管理系统

详细架构设计1 - Role & Relation

极客时间 训练营

【客户端 Role 设计】

1. 客户端采用Java语言开发，基于Netty实现与服务端交互。

【客户端和服务器的 Relation 设计】

1. 客户端与服务端采用 TCP 连接，采用 JSON 传递数据。
2. 为了兼容非 Java 系统，服务端同时提供 HTTP 接口。

【服务器 Role 设计】

1. 服务器基于Netty开发，采用 Reactor 网络模型。
2. 两台服务器组成一个 sharding，整个系统可以多个 sharding，每个 sharding 包含一主一从两台服务器（可以对比 MongoDB shard）。
3. 主服务器提供消息读写操作，从服务器只提供消息读取操作。
4. 服务器基于 ZooKeeper 进行主从切换。

【MySQL 的 Role 和 Relation 设计】

1. 采用 MySQL 主从同步。
2. 每个消息队列对应一个表。
3. 消息表最多存储30天内的消息，过期的自动清除。
4. 直接用 MySQL 的主从复制来实现数据复制。

为何用 JSON 作为数据格式，是不是应该用 Protocol Buffer 这种二进制高性能格式？

详细架构设计2 - Rule

极客时间 训练营

【消息发布】

1. 消息队列系统设计两个角色：生产者和消费者，每个角色都有唯一的名称。
2. 消息队列系统提供 SDK 供各业务系统调用，SDK 从配置中读取所有消息队列系统的服务器信息，SDK 采取轮询算法发起消息写入请求给主服务器。
3. 如果某个主服务器无响应或者返回错误，SDK 将发起请求发送到下一台主服务，相当于在客户端实现了分片的功能。

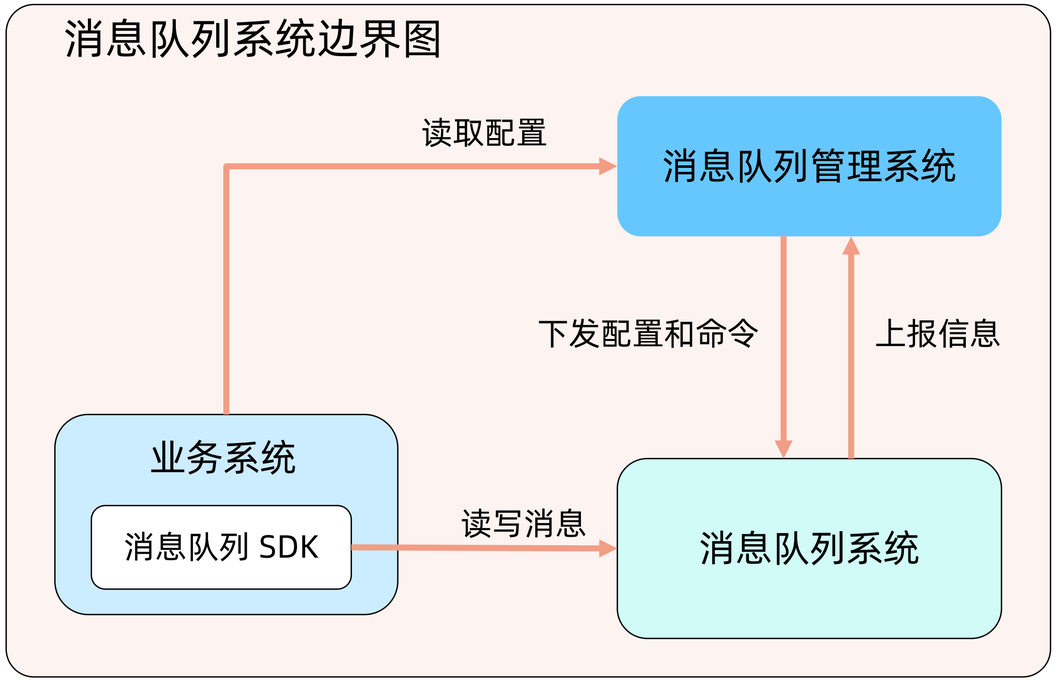
【消息读取】

1. 消息队列系统提供 SDK 供各业务系统调用，SDK 从配置中读取所有消息队列系统的服务器信息，轮流向所有服务器发起消息读取请求。
2. 消息队列服务器需要记录每个消费者的消费状态，即当前消费者已经读取到了哪条消息，当收到消息读取请求时，返回下一条未被读取的消息给消费者。
3. 默认情况下主服务器提供读写服务，当主服务器挂掉后，从服务器提供读消息服务。

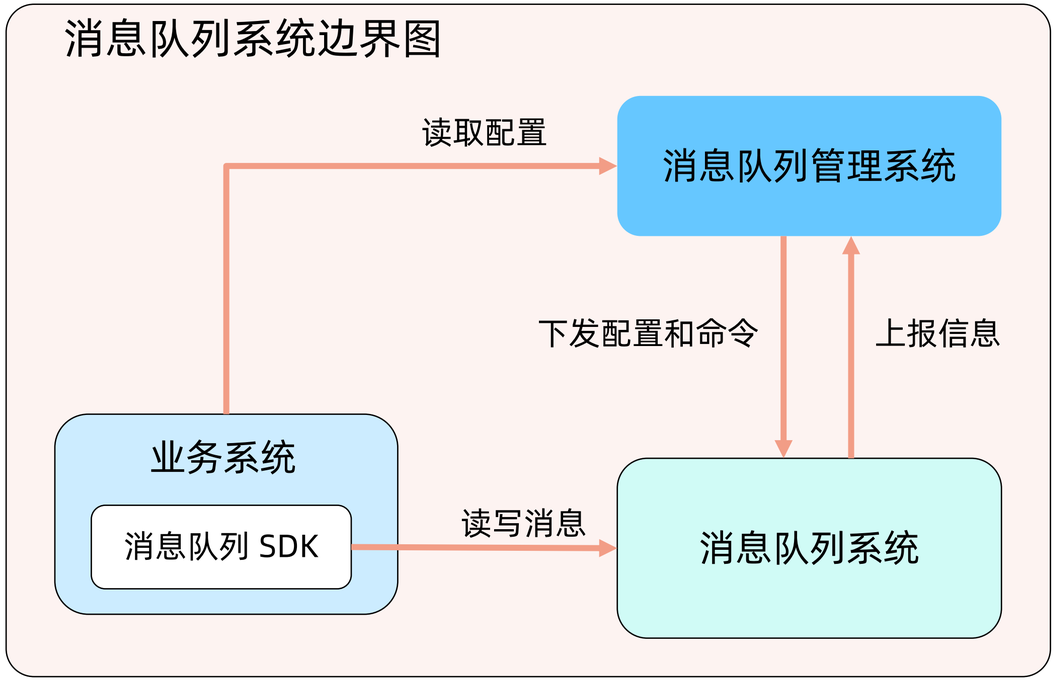
【服务器主从切换】

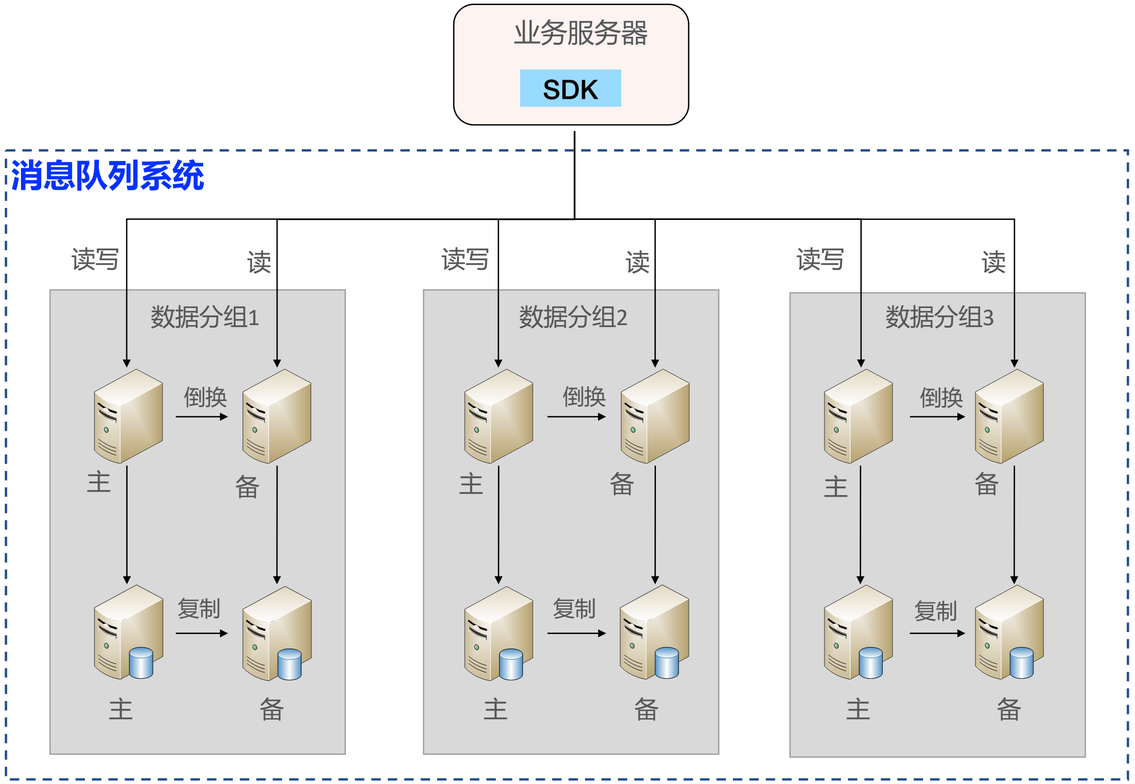
1. 同一组的主从服务器配置相同的 group 名称，在 ZooKeeper 建立对应的 PERSISENT 节点。
2. 主从服务器启动后，在 ZooKeeper 对应的 group 节点下建立 EPHEMERAL 节点，名称分为为 master 和 slave。
3. 从服务器 watch 主服务器的 master 节点状态，当 master 节点超时被删除后，从服务器接管读消息，收到客户端 SDK 的读消息请求后返回消息，收到客户端 SDK 的写请求直接拒绝。

详细架构设计3 - 消息队列管理系统



考考你 - 两幅架构图要合并么？





随堂测验

【判断题】

1. 开源方案比较成熟，拿来即用，开发、测试、运维成本都有优势。
2. 不同的公司环境，对技术复杂度的要求不同，不一定都选简单的。
3. 自研方案因为代码都是自己团队写的，各方面质量肯定更有保证。
4. 备选架构决策的时候，哪个架构的优点多就选哪个。
5. 消息队列管理系统也是架构的一个 Role，但不影响整个架构的复杂度。

【思考题】

如果你现在的团队做消息队列架构选型，你觉得会优选哪个方案，理由是什么？

THANKS

极客时间 | 训练营