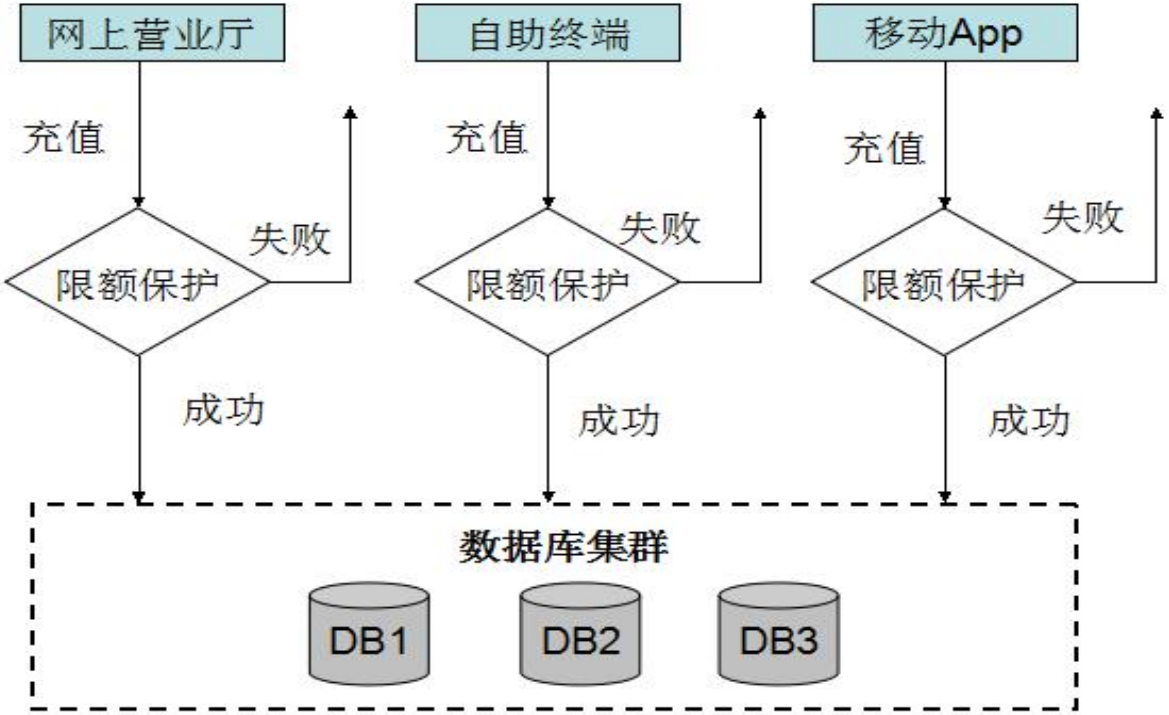
基于微服务架构改造单体架构的实践总结

李林锋，2007年毕业于东北大学，2008年加入华为，从事电信软件的架构设计和开发。8年Java NIO通信框架、网关平台和中间件设计和开发经验，精通Java NIO、Java多线程编程和分布式服务框架等，《分布式服务框架原理与实践》作者，目前从事云平台相关的架构设计和开发。

公众号：Netty之家，分享Netty、服务化相关的各种案例和实践

- 传统单体架构的弊端
- 服务化架构的演进历史
- 微服务架构的实施
- 最佳实践



研发成本高：

- 代码重复率高
- 需求变更困难
- 无法满足新业务快速上线和敏捷交付

运维效率低：

- **测试、部署成本高：**业务运行在一个进程中，因此系统中任何程序的改变，都需要对整个系统重新测试并部署

- **可伸缩性差：**水平扩展只能基于整个系统进行扩展，无法针对某一个功能模块按需扩展

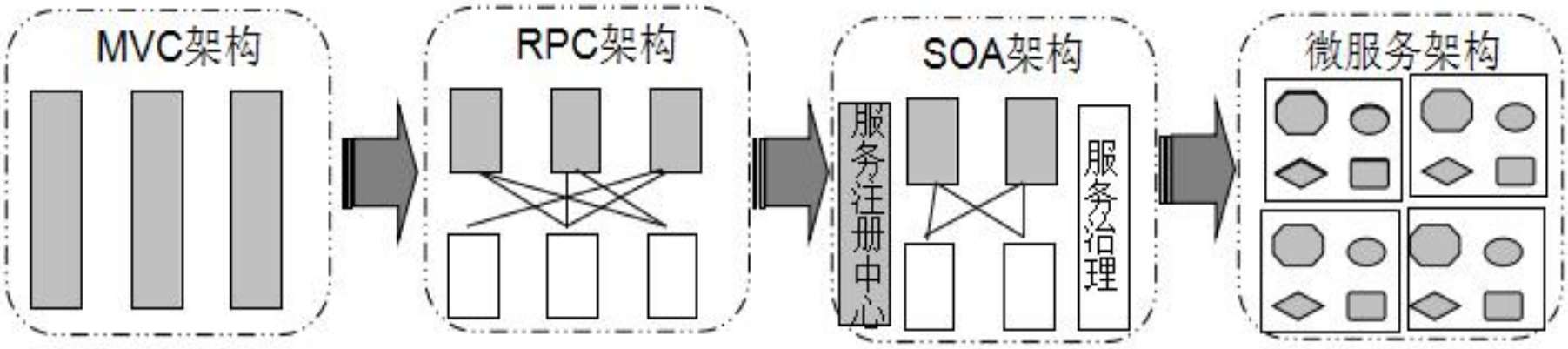
- **可靠性差：**某个应用BUG，例如死循环、OOM等，会导致整个进程宕机，影响其它合设的应用

- **代码维护成本高：**本地代码在不断的迭代和变更，最后形成了一个个垂直的功能孤岛，只有原来的开发者才理解接口调用关系和功能需求，新加入人员或者团队其它人员很难理解和维护这些代码

- **依赖关系无法有效管理：**服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系

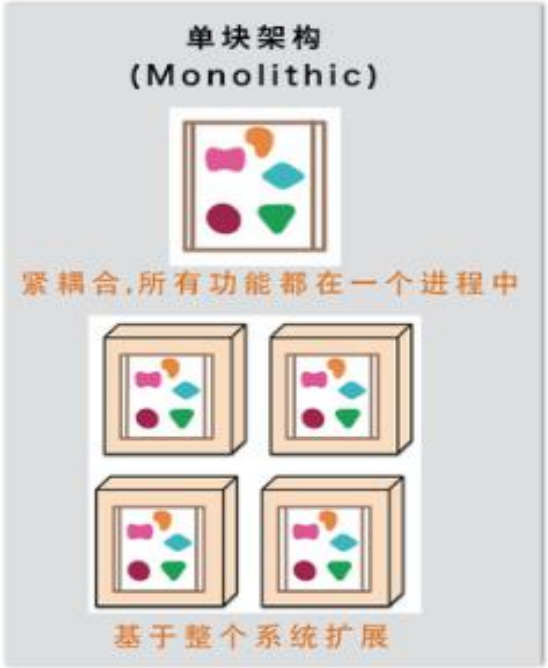
服务化：

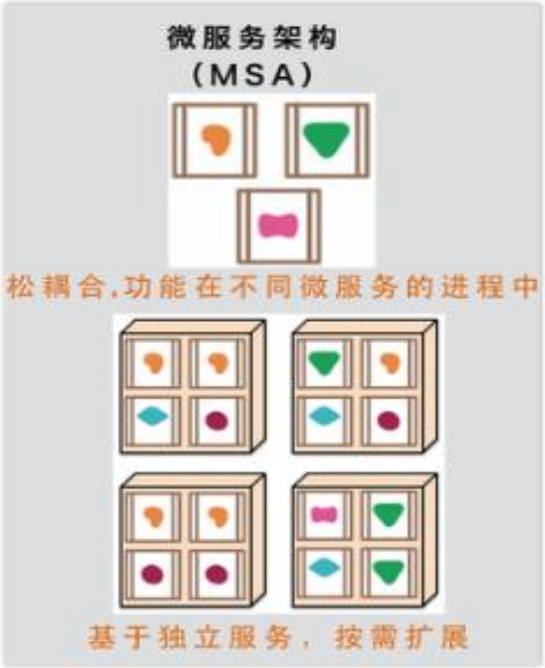
- **拆分：**对应用进行水平和垂直拆分
- **解耦：**通过服务化和订阅、发布机制对应用调用关系解耦，支持服务的自动注册和发现
- **透明：**通过服务注册中心管理服务的发布和消费、调用关系
- **独立：**服务可以独立打包、发布、部署、启停、扩容和升级，核心服务独立集群部署
- **分层：**梳理和抽取核心应用、公共应用，作为独立的服务下沉到核心和公共能力层，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求



1、MVC：解决前后端、界面、控制逻辑和业务逻辑分层问题
2、RPC：远程过程调用，本质就是分布式协作和系统间的解耦
3、SOA：服务化架构，企业级资产重用和异构系统间的集成对接
4、微服务化架构：敏捷交付、互联网、容器化发展的产物

定义：微服务（MSA）是一种架构风格，旨在通过将功能分解到各个离散的服务中以实现对解决方案的解耦。





特征：

- 小，且只干一件事情
- 独立部署和生命周期管理
- 异构性
- 轻量级通信，RPC或者Restful

微服务拆分原则：围绕业务功能进行垂直和水平拆分。大小粒度是难点，也是团队争论的焦点。

错误的实践：

- 以代码量作为衡量标准，例如500行以内
- 拆分的粒度越小越好

建议的原则：

- 功能完整性、职责单一性
- 粒度适中，团队可接受
- 迭代演进，非一蹴而就
- API的版本兼容性优先考虑

微服务开发原则：接口先行，语言中立，服务提供者和消费者解耦，并行开发，提升产能。

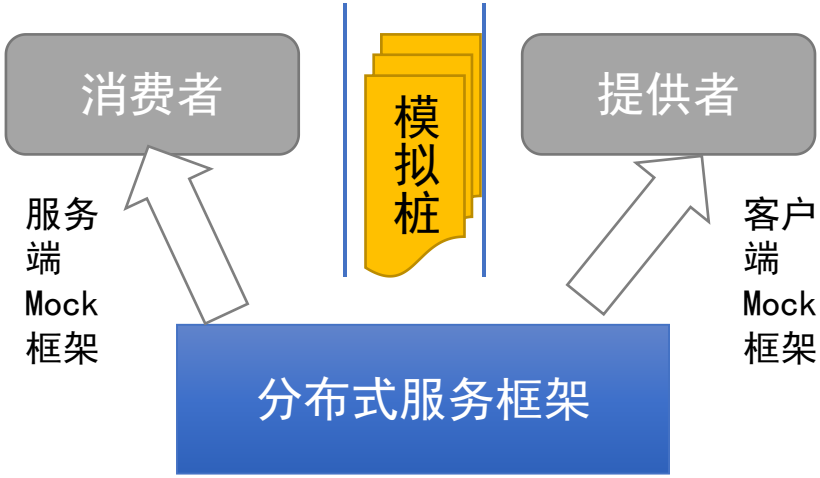
错误的实践：

- 服务提供者专注于内部实现，而不是优先提供契约化的接口
- 担心接口变更，迟迟不提供接口契约，导致消费者无法并行开发

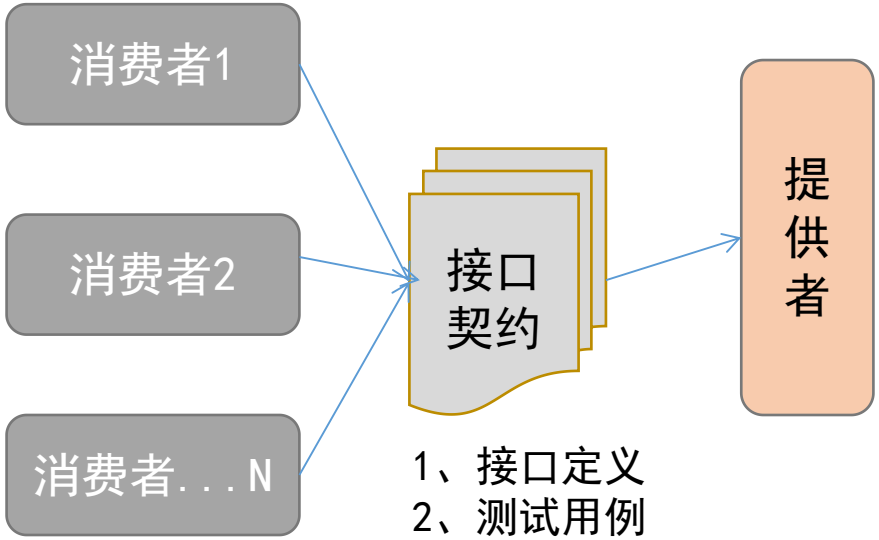
建议的原则：

- 接口优先
- 允许契约的变更，但项目组需就接口的兼容性做约束
- 契约驱动测试，实现服务提供者和消费者解耦
- IDL，代码骨架自动生成

微服务测试原则：单元测试，契约测试（接口测试），行为测试，集成测试。



微服务框架提供服务端和客户端Mock框架

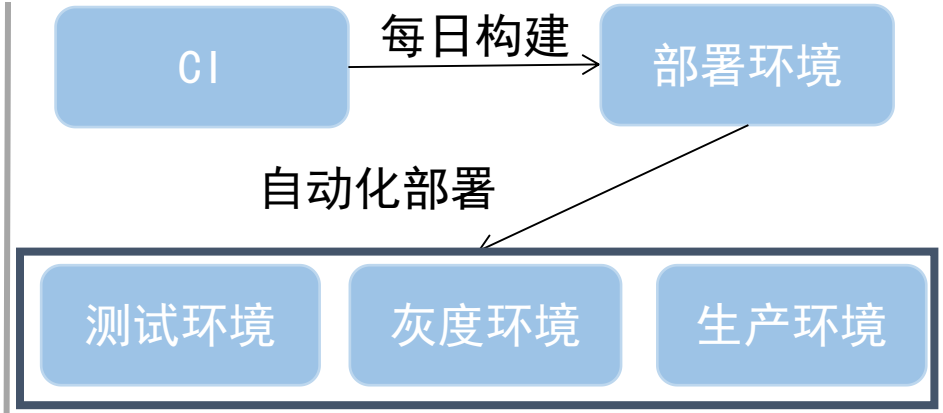


微服务部署原则：独立部署和生命周期管理、基础设施自动化。

启动 停止 升级 回滚 下线

选择	微服务名	版本号	分组信息
○	订单管理	1.0.1	无锡
○	用户管理	1.0.1	南京
○	鉴权认证	1.0.6	镇江

实现关键：微服务独立打包，物理交付件独立，例如jar包

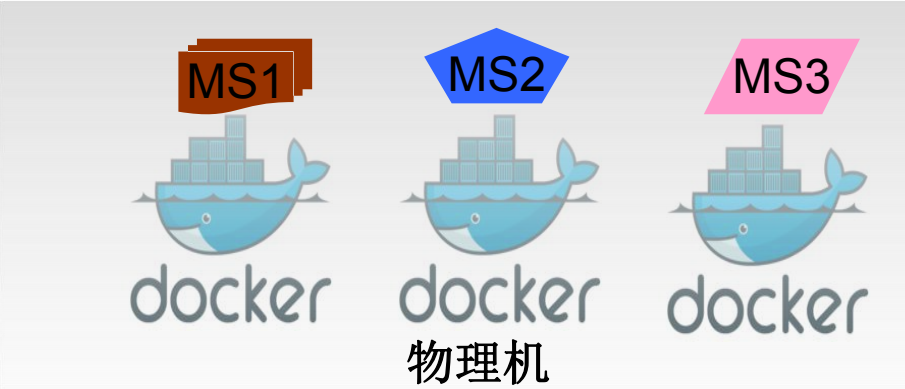


核心技术：

1、持续交付流水线

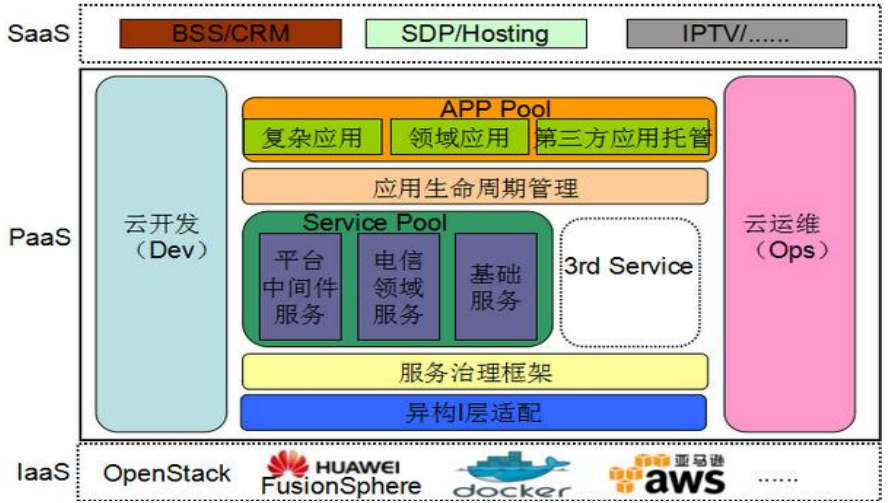
2、Docker或PaaS平台

微服务运行容器：Docker、PaaS平台（VM）



使用Docker部署微服务的优点总结：

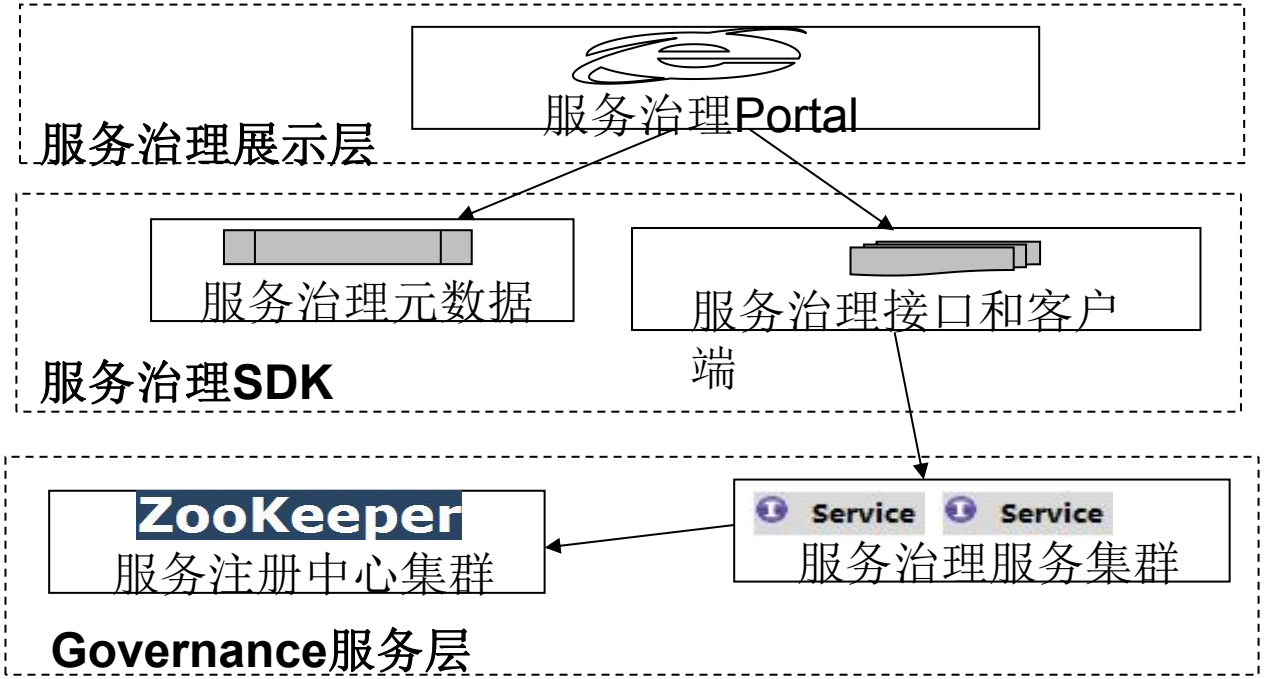
1. 一致的环境：线上线下环境一致
2. 避免对特定云基础设施提供商的依赖
3. 降低运维团队负担
4. 高性能：接近裸机的性能
5. 多租户



微服务云化：

1. 云的弹性和敏捷
2. 云的动态性和资源隔离
3. Dev&Ops

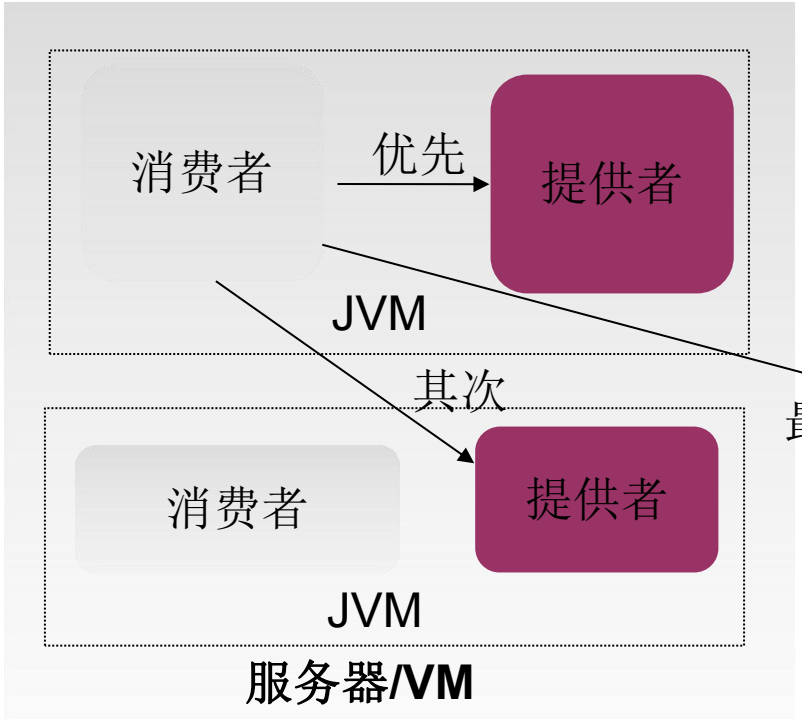
微服务治理原则：线上治理、实时动态生效



微服务治理策略：

1. 流量控制：动态、静态流控
2. 服务降级
3. 超时控制
4. 优先级调度
5. 流量迁移
6. 调用链跟踪和分析
7. 服务路由
8. 服务上线审批、下线通知
9. SLA策略控制...

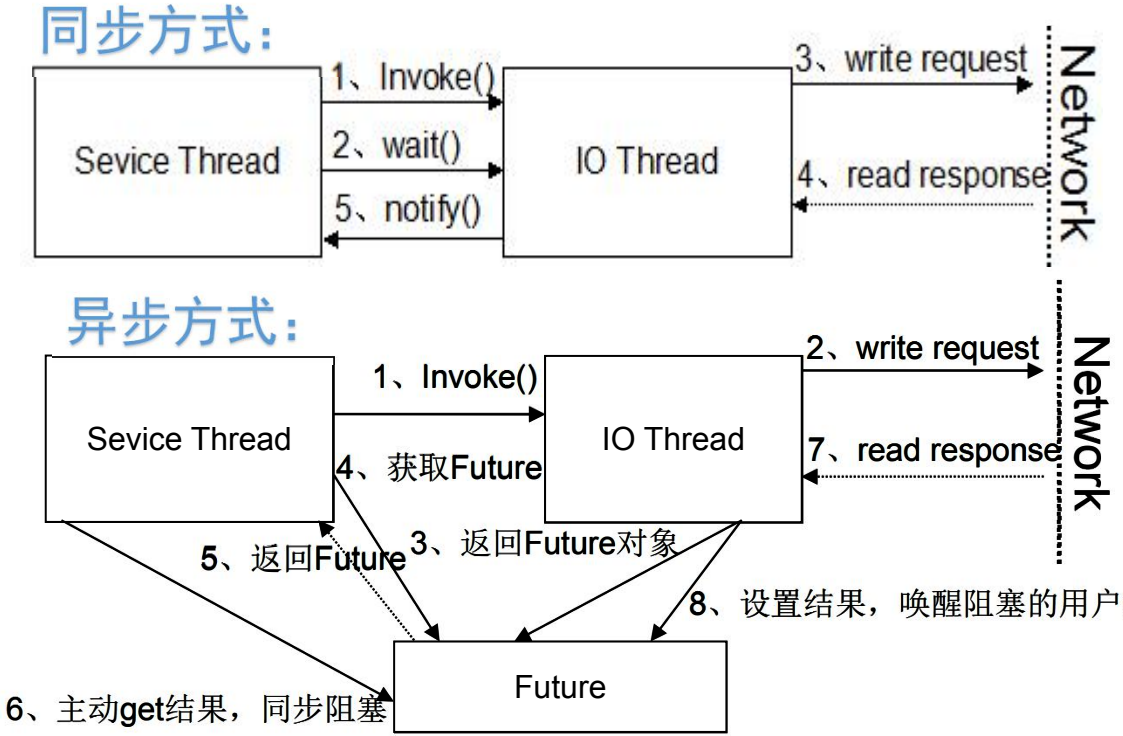
服务路由：本地短路策略

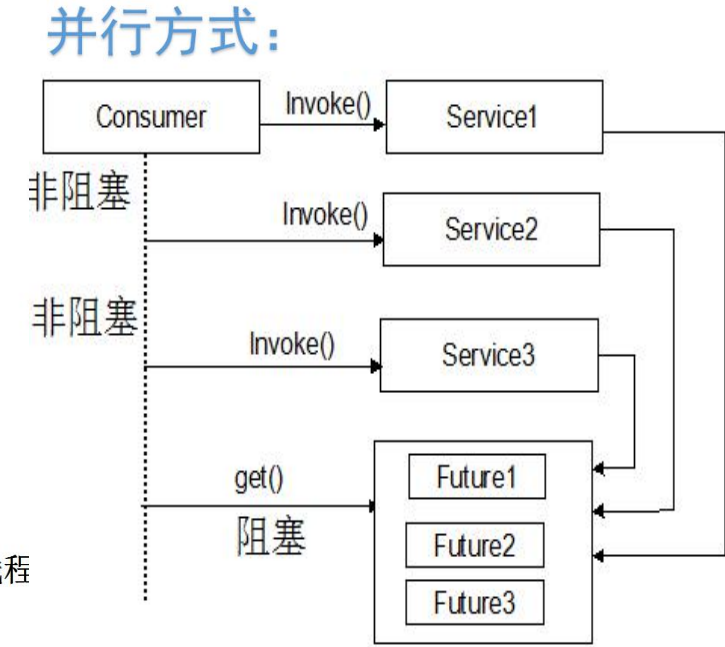


关键技术点：

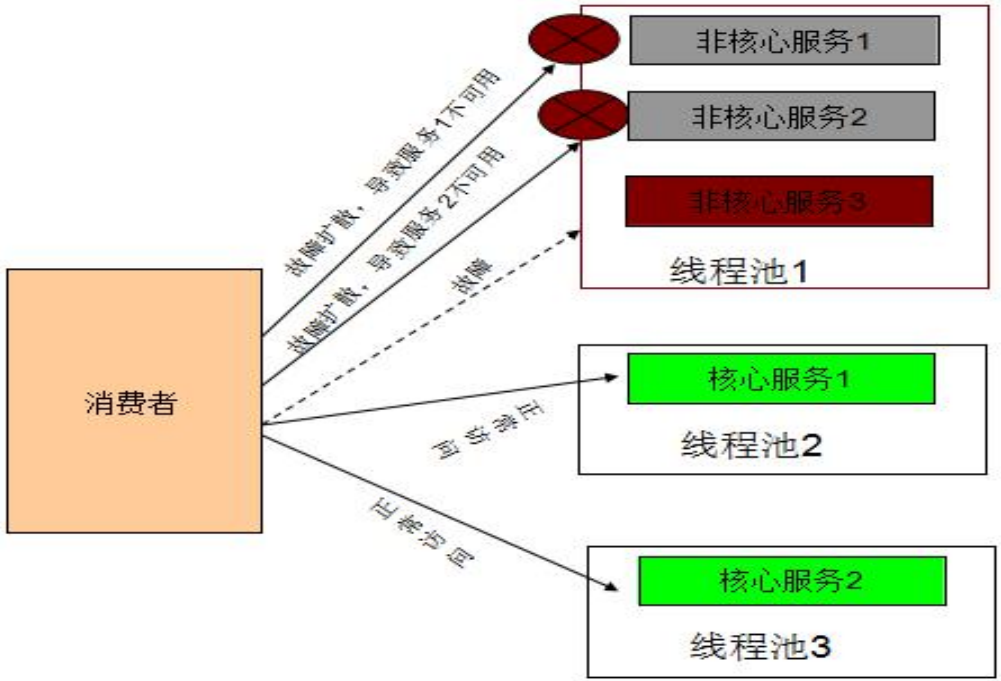
- 优先调用本JVM内部服务提供者、
- 其次是相同主机或者VM的、
- 最后是跨网络调用

服务调用方式：同步调用、异步调用、并行调用





微服务故障隔离：线程级、进程级、容器级、VM级、物理机...

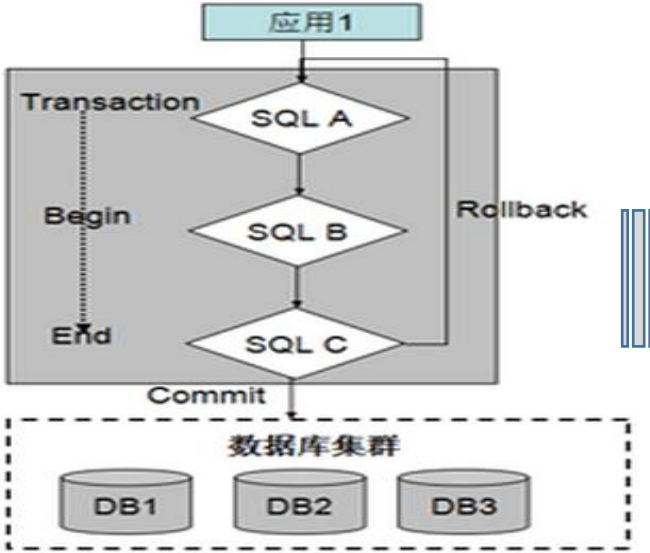


关键技术点：

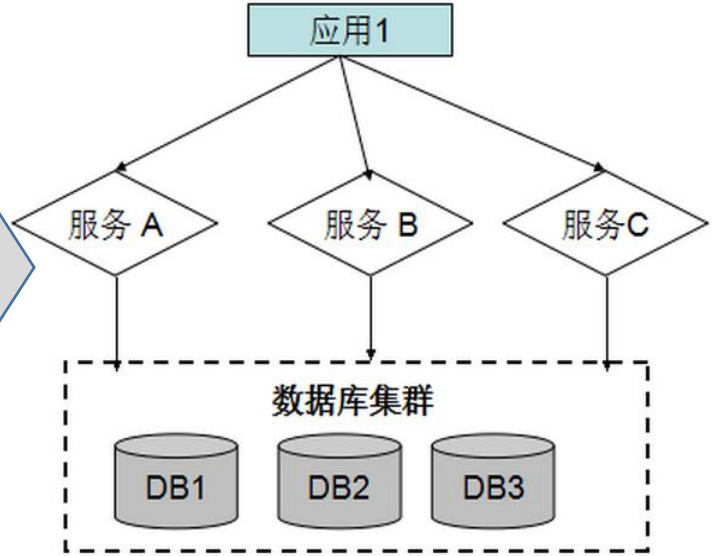
1. 支持服务部署到不同线程/线程池中
2. 核心服务和非核心服务隔离部署

事务一致性：大部分是最终一致性、极少部分需要强一致性

本地事务：



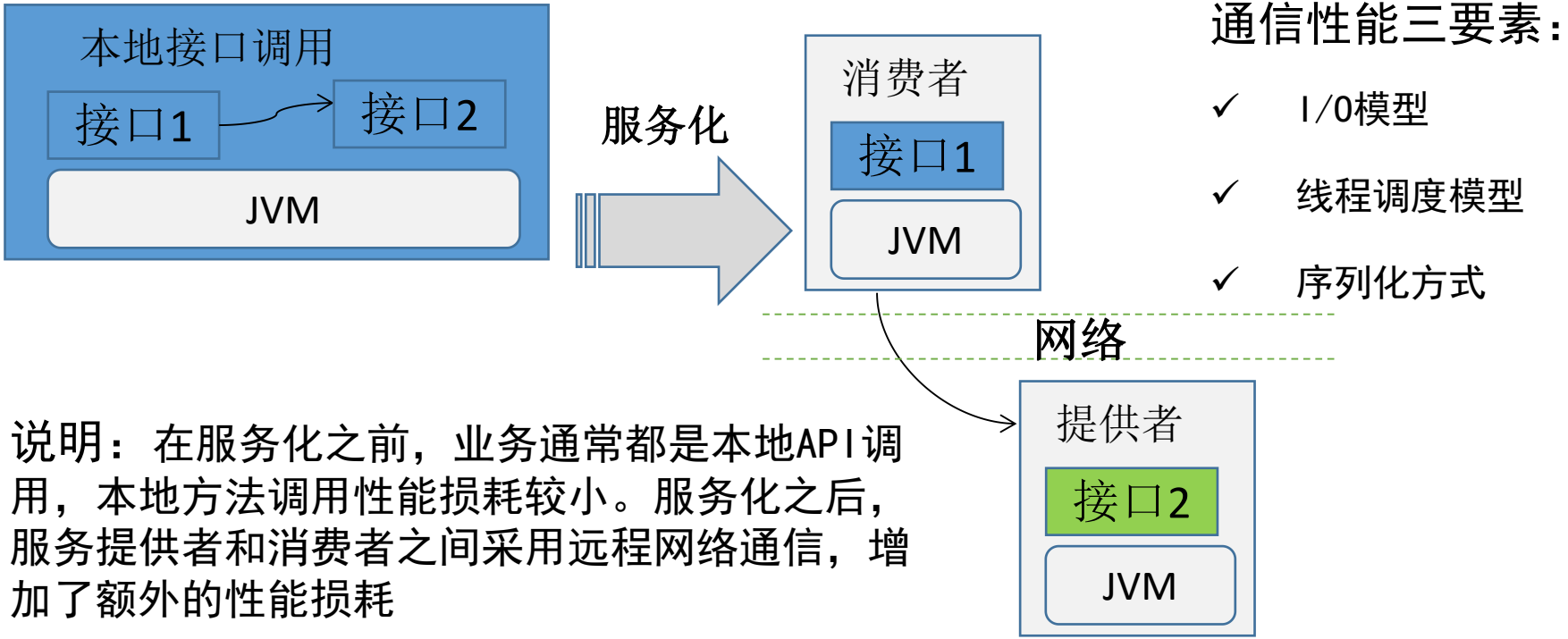
分布式事务：



策略：

1. 最终一致性，消息中间件
2. 强一致性，TCC

时延问题：非阻塞I/O、二进制、长链接、码流压缩



通信性能三要素：

- ✓ I/O模型
- ✓ 线程调度模型
- ✓ 序列化方式

说明：在服务化之前，业务通常都是本地API调用，本地方法调用性能损耗较小。服务化之后，服务提供者和消费者之间采用远程网络通信，增加了额外的性能损耗

微服务接口兼容性：技术保障、管理协同

1. 制定并严格执行《微服务前向兼容性规范》，避免发生不兼容修改或者私自修改不通知周边的情况

2、接口兼容性技术保障：例如Thrift的IDL，支持新增、修改和删除字段，字段定义位置无关性，码流支持乱序等

3、持续交付流水线的每日构建和契约化驱动测试，能够快速识别和发现不兼容


谢谢大家