服务化架构的演进与实践

李林锋 neu_lilinfeng@sina.com
新浪微博、微信： Nettying
微信公众号:Netty之家

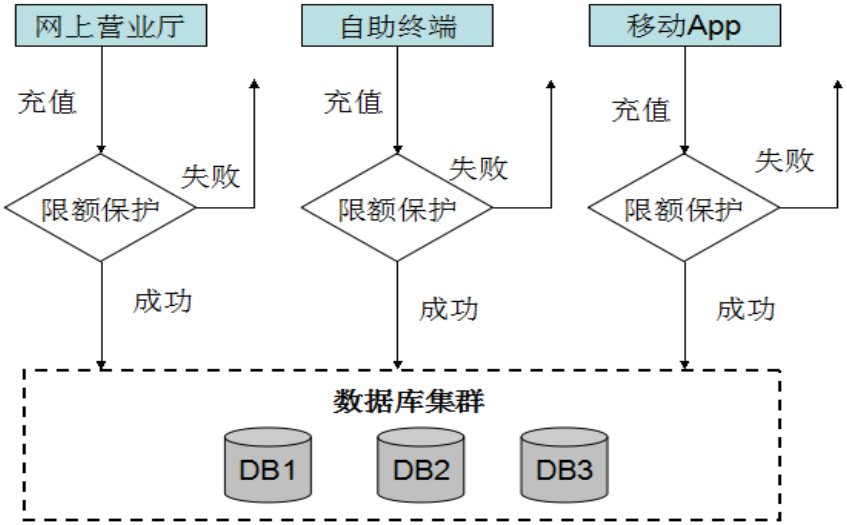
大纲

传统应用开发面临的挑战

服务化实践

服务化架构的演进方向

挑战1-- 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

挑战2-- 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

解决对策

- **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

服务化实践-服务的订阅发布

关键技术点：

1. 服务的订阅、发布机制
2. 服务的健康状态检测
3. 高HA

服务化实践-零侵入

服务提供者：

```
    <bean id = "xxxService" class="edu.neu.xxxServiceImpl" />
    <xxx:service                                                    interface=
"edu.neu.xxxService" ref="xxxService" />
```

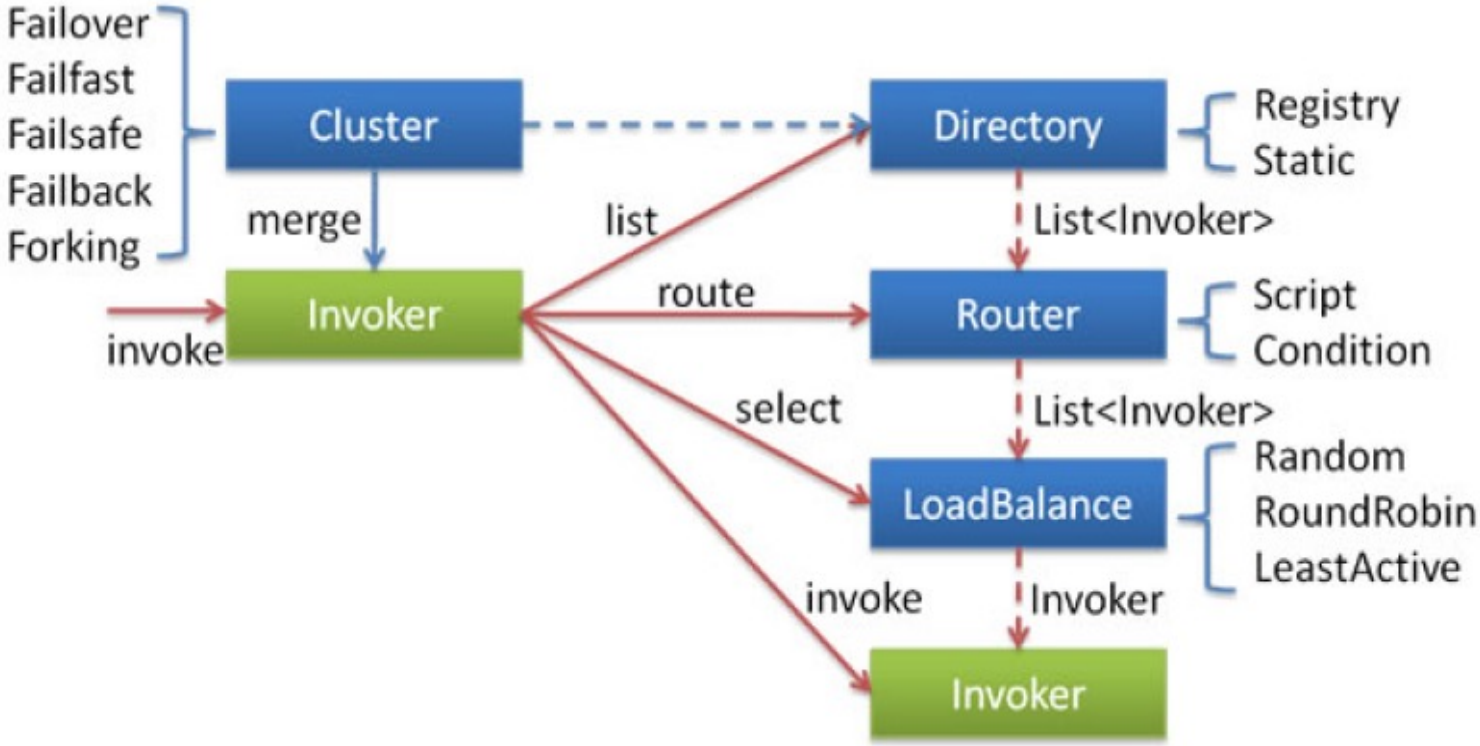
服务消费者：

```
<xxx:reference id="xxxService" interface="edu.neu.xxxService" />
<bean class="edu.neu.xxxAction" init-method="start">
        <property name="xxxService" ref="xxxService" />
</bean>
```

关键技术点：

1. 配置化发布和消费服务（例如Spring扩展机制）
2. 尽量不要直接提供API接口给应用
3. 技术门槛和学习成本要低

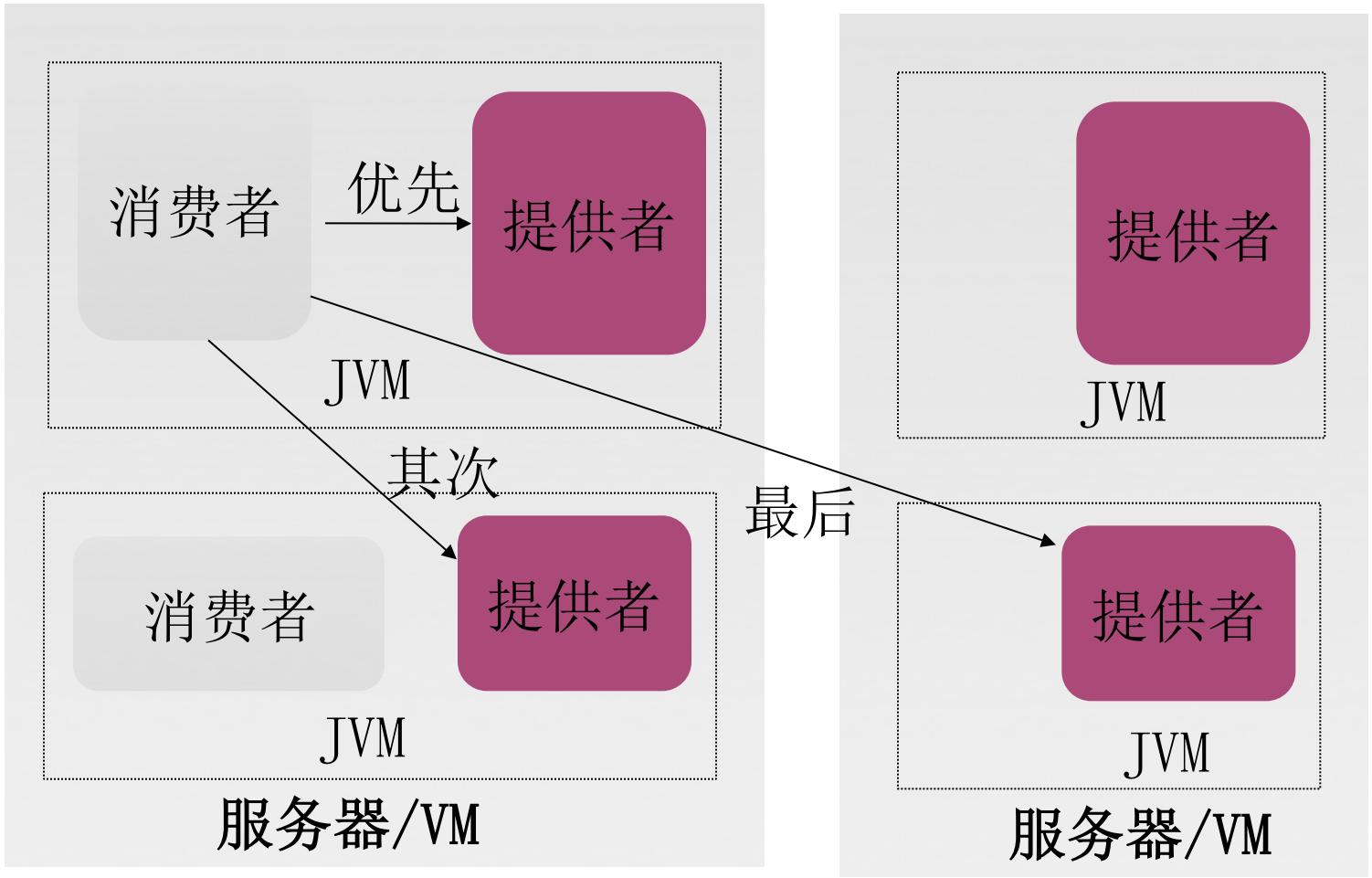
服务化实践-容错和路由



关键技术点：

1. 提供常用的路由策略，路由策略可扩展
2. 集群容错应用无需感知，但要考虑可用性
3. 支持动态路由（例如脚本语言）

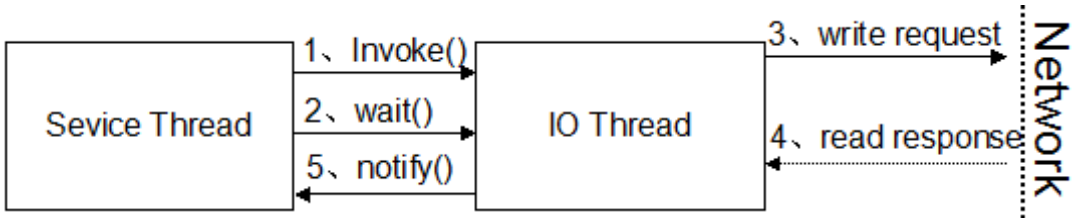
服务化实践-本地短路策略

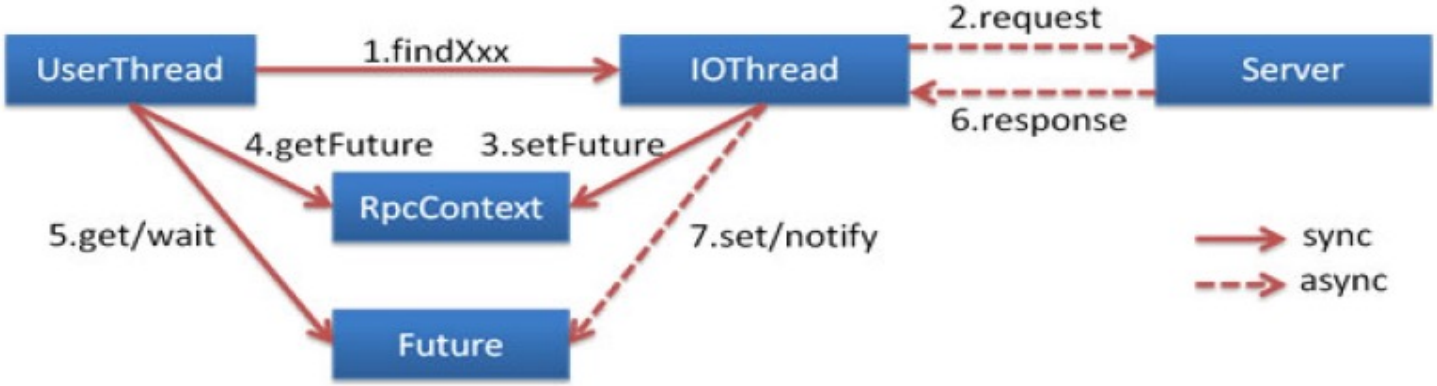


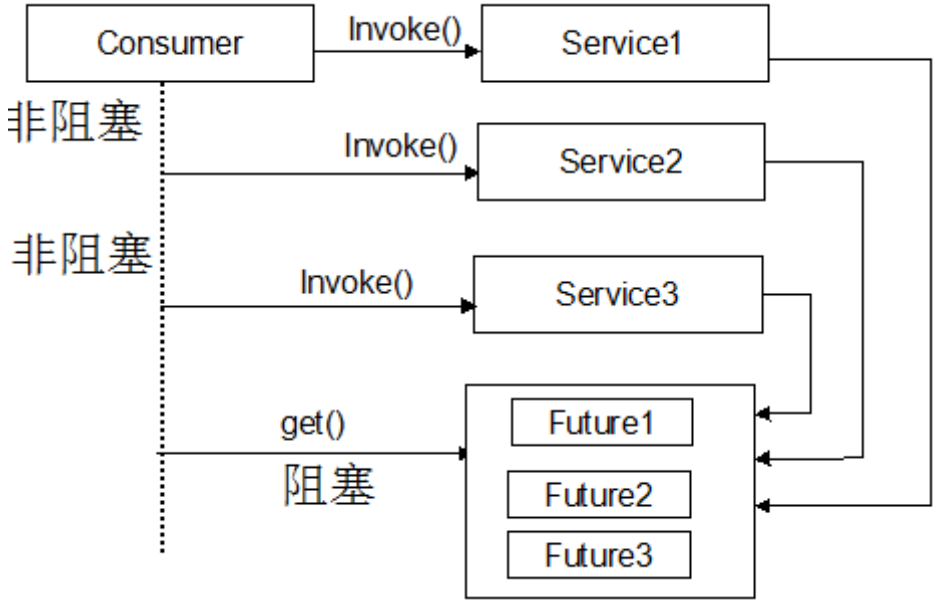
关键技术点：

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

服务化实践-多样化调用方式



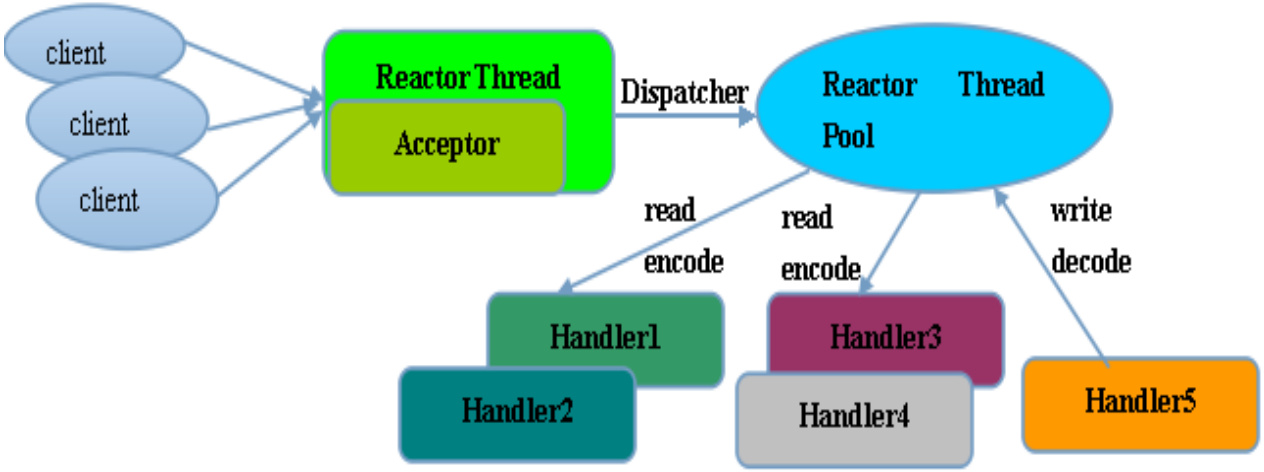


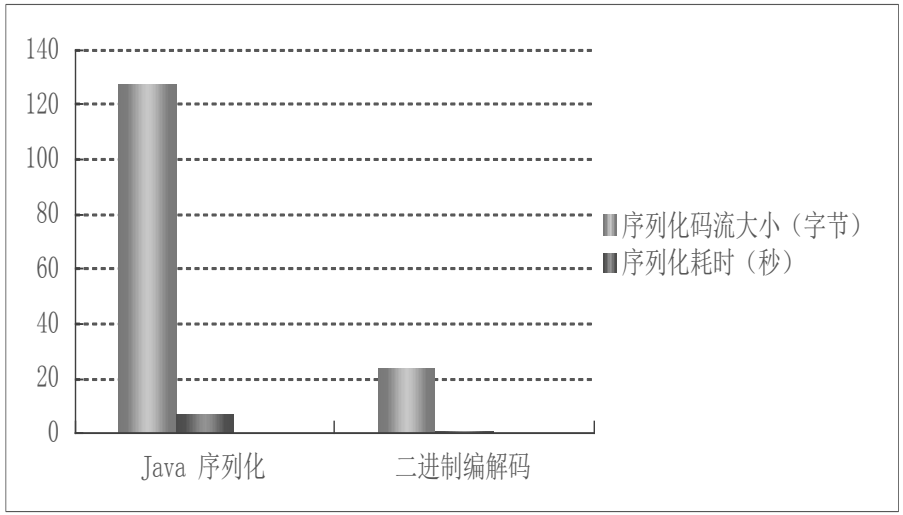


关键技术点：

1. 同步调用
2. 异步调用
3. 并行调用

服务化实践-高性能、低时延



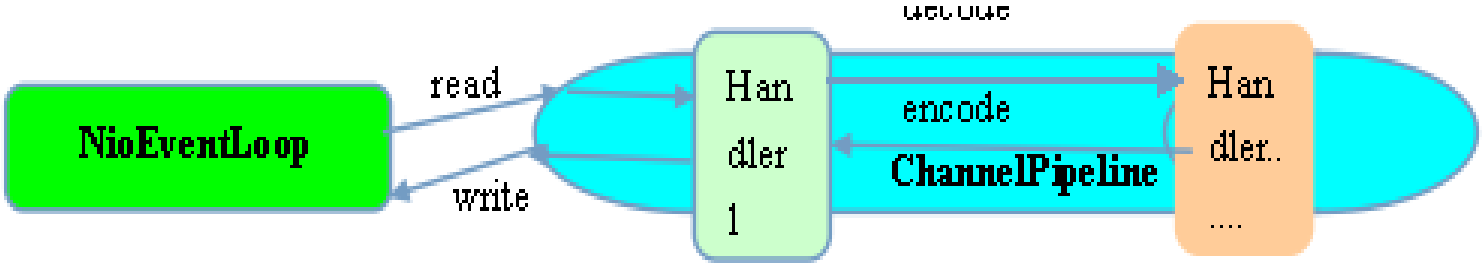


protobuf
Protobuf

关键技术点：

影响性能的关键因素：I/O 、CodeC、Thread

服务化实践-高效线程模型

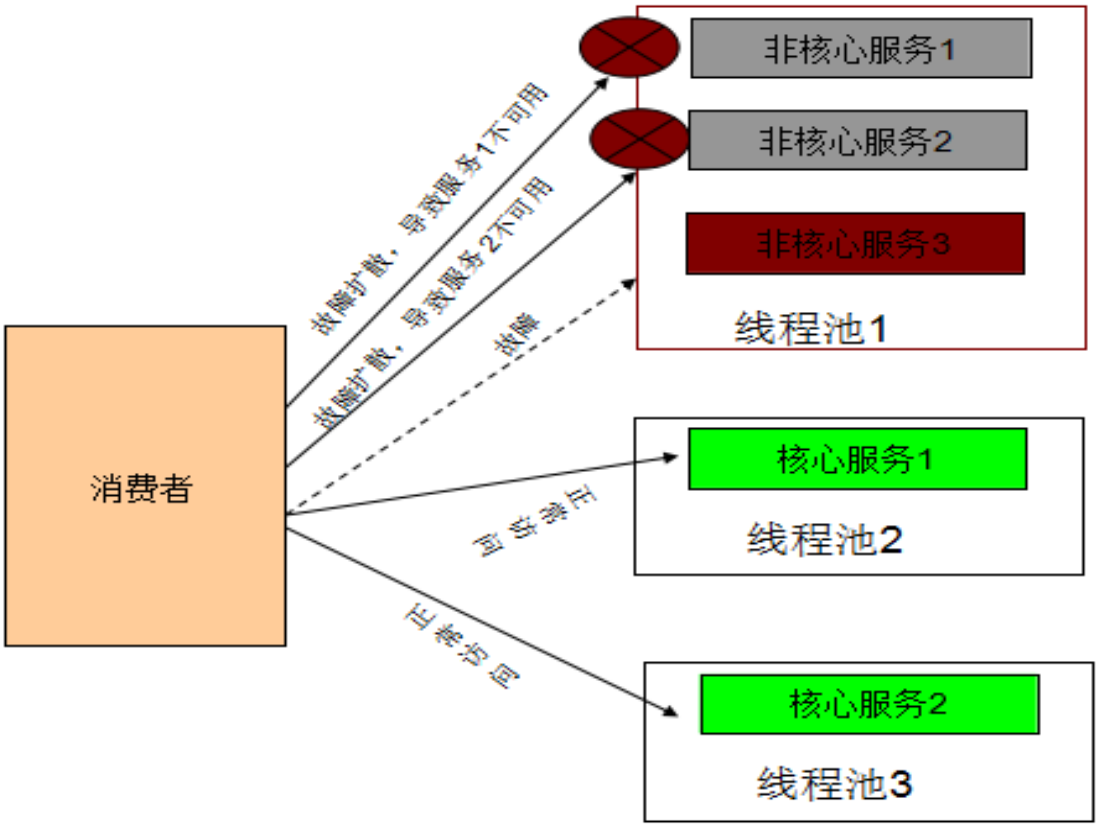


关键技术点：

1. **无锁化串行设计**： 避免ChannelHandler被并发调用，加锁会降低性能

2. **单线程线程池模型，性能更优**：1个线程1个队列。避免JDK线程池的多线程-单阻塞队列导致的激烈锁竞争

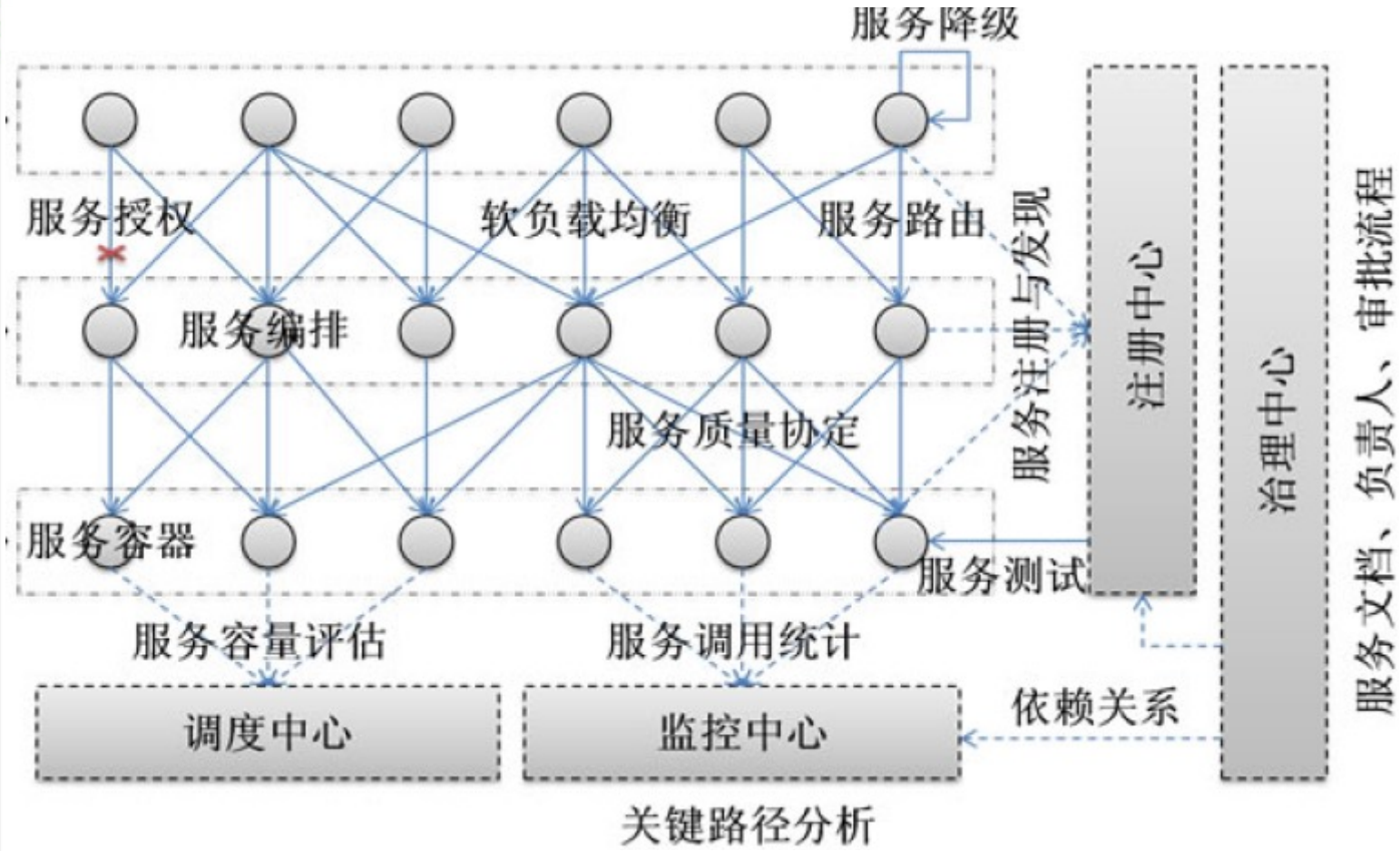
服务化实践-故障隔离



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

服务化实践-服务治理



关键技术点：

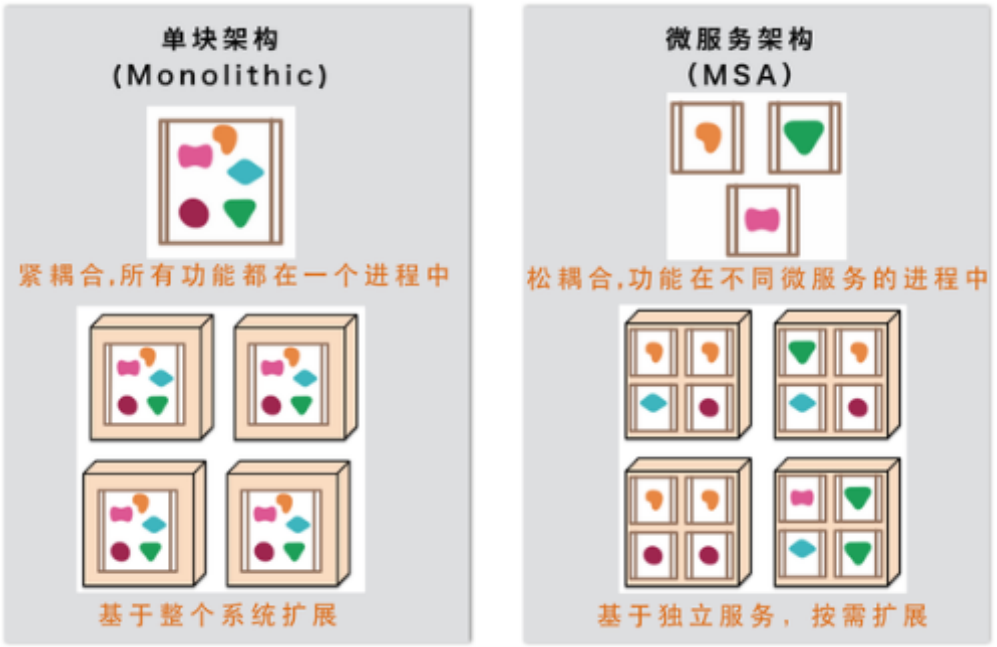
1. 线上治理，不需要重启，秒级生效，故障快速恢复
2. 线下治理，服务全生命周期管理，例如上线审批

服务化实践-高可靠性

关键技术点：

1. 服务无状态设计
2. 服务注册中心集群，宕机不影响业务运行
3. 服务提供者集群，集群容错屏蔽服务提供者故障
4. 服务健康状态检测，基于时延等性能KPI指标
5. 服务治理中心集群，宕机不影响业务运行
6. 服务级故障隔离
7. 核心服务独立部署和集群
8. 跨机房路由和异地容灾

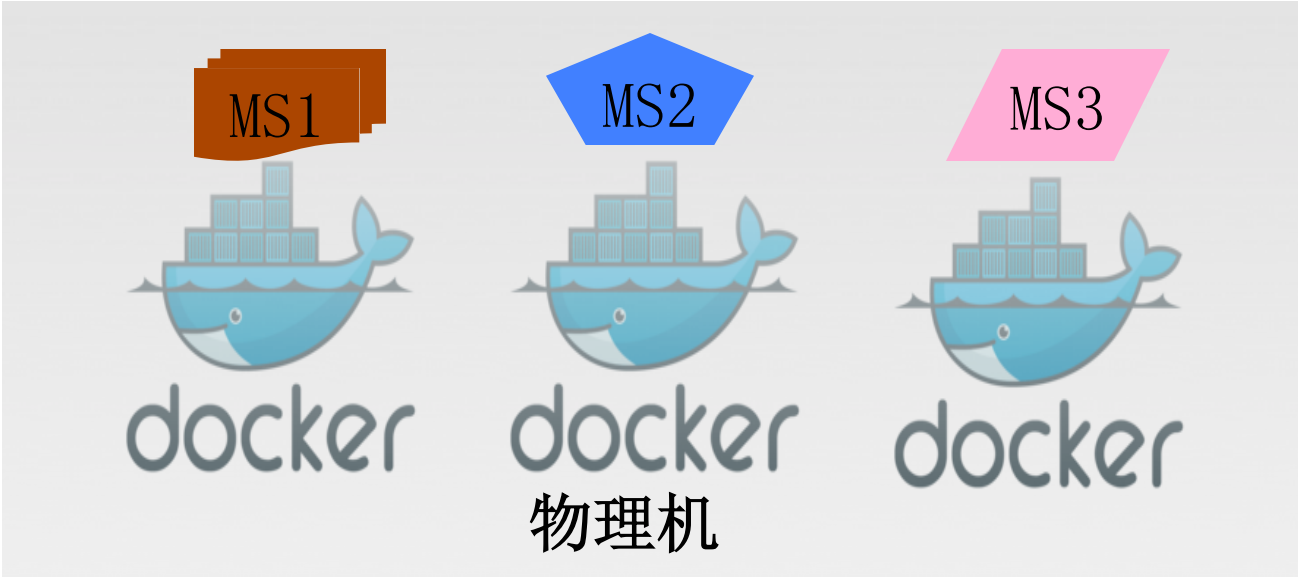
未来演进方向-微服务架构



微服务架构优势总结：

1. 异构性
2. 灵活性：独立生命周期管理
3. 按需伸缩
4. 故障隔离
5. 2 Pizza Team（构建全栈小分队）

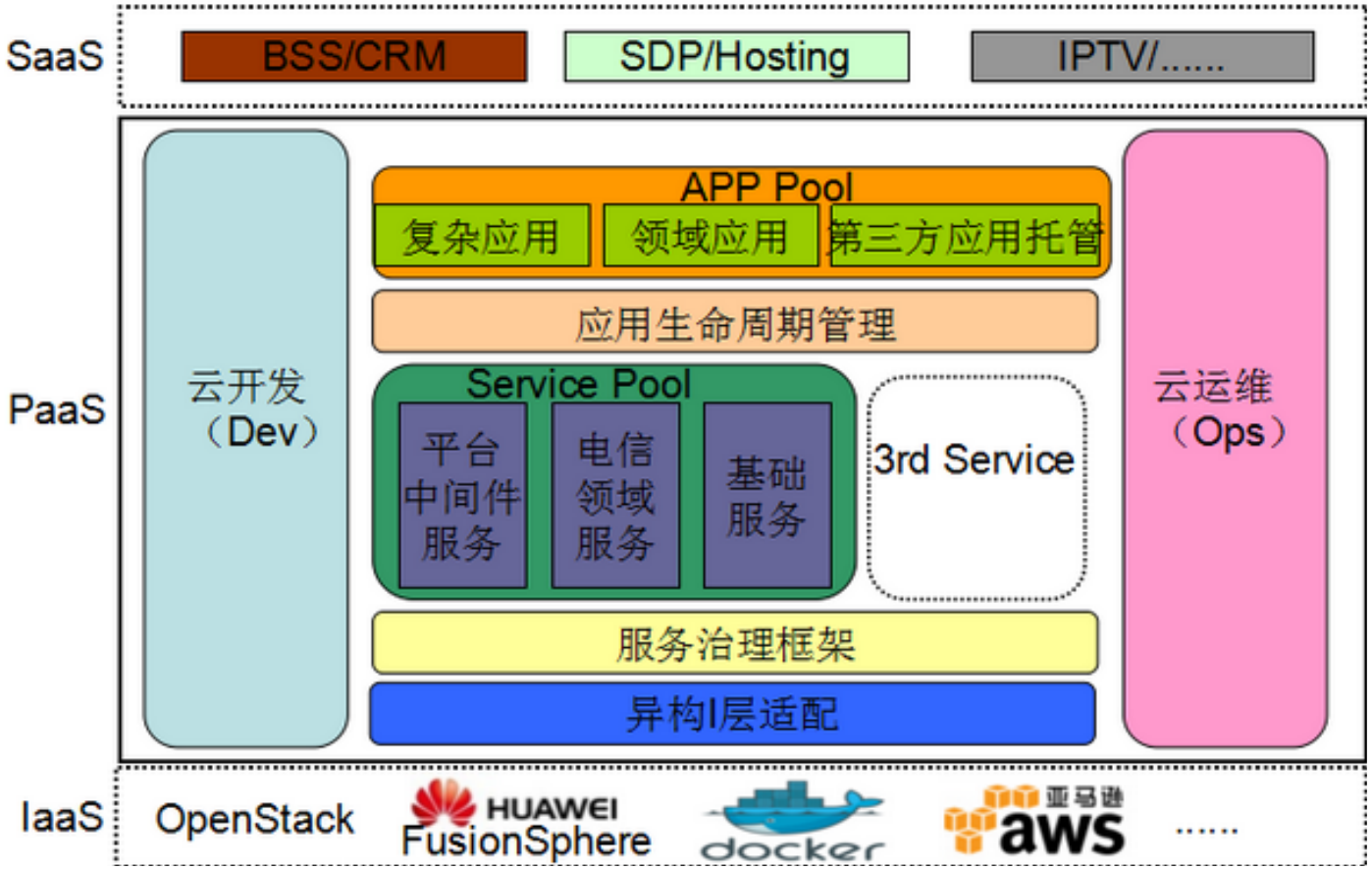
未来演进方向-基于Docker部署微服务



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

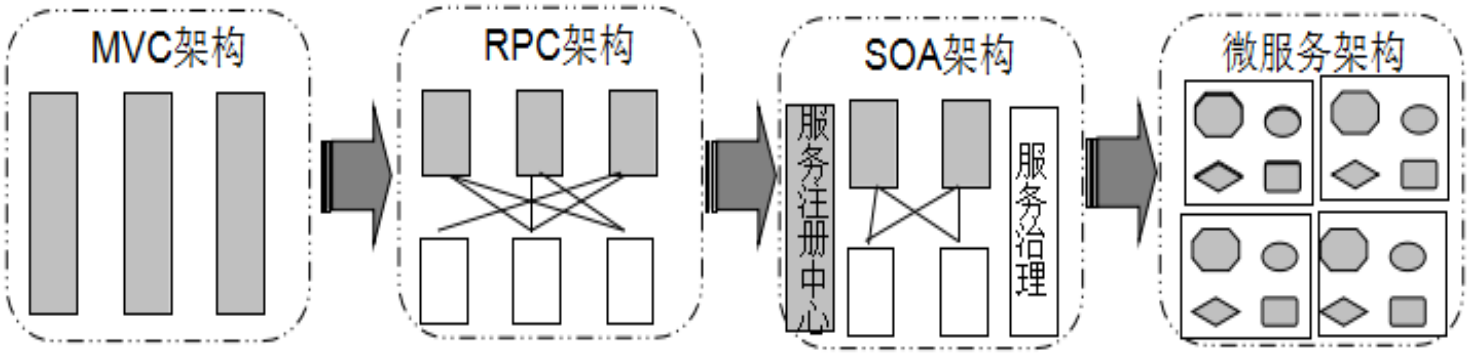
未来演进方向-云上的微服务



微服务云化：

1. 云的弹性和敏捷
2. 云的动态性
3. Dev&Ops

服务化架构演进总结



服务化架构演进总结：

1. MVC架构
2. RPC架构
3. SOA架构
4. 微服务架构

学习资料

Q & A

谢谢聆听！

李林锋 neu_lilinfeng@sina.com
新浪微博、微信：Nettying
微信公众号：Netty之家