融数微服务架构

基于Spring Cloud和Netflix打造

自我介绍

- 现任融数数据北京研发中心CTO，负责公司大数据平台、微服务框架以及DevOps平台的研发工作；
- 毕业于天津大学，毕业后一直从事软件相关研发和架构设计工作，曾经在普元软件任资深架构师、IBM GBS任咨询经理、亚马逊任架构师等，后加入创业公司，从事研发和管理工作；
- 热爱编程，喜欢钻研新技术，对于微服务、企业架构、大数据以及DevOps有浓厚的兴趣。

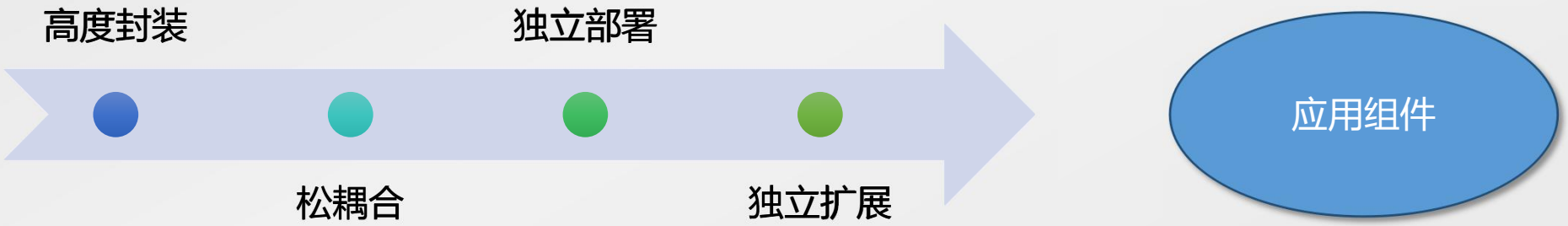
Agenda

- 谈谈微服务
- 微服务技术选型过程
- 微服务架构设计的一些思考点
- 融数微服务架构的核心概念和实现
- 融数DevOps平台对微服务的支撑
- 技术团队的组织
- Operation Excellent

Agenda

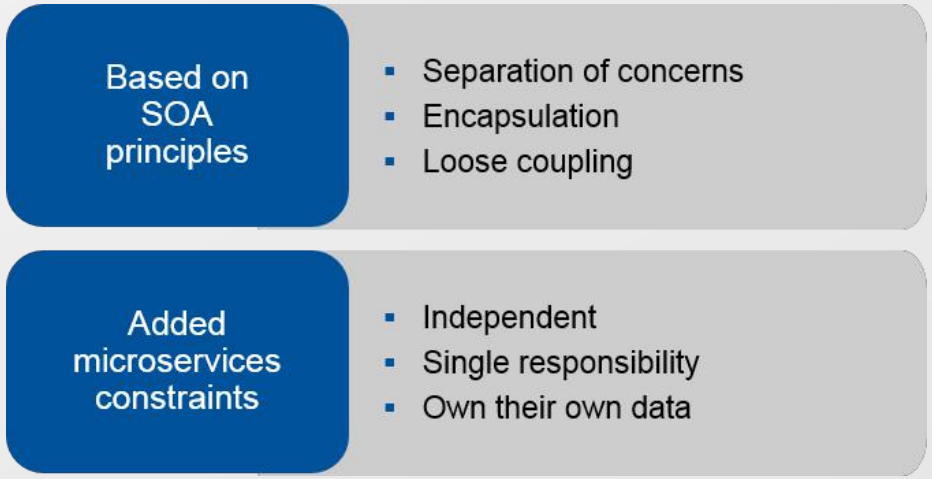
- **谈谈微服务**
- 微服务技术选型过程
- 微服务架构设计的一些思考点
- 融数微服务架构的核心概念和实现
- 融数DevOps平台对微服务的支撑
- 技术团队的组织
- Operation Excellent

从微服务的概念谈起



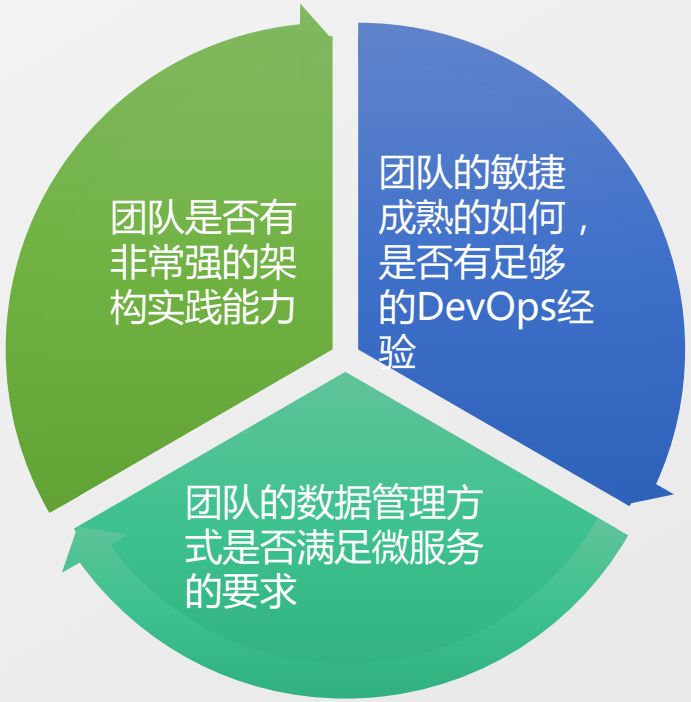
微服务与SOA的异同

- 从设计原则来讲，微服务架构遵循SOA principles
- 小的、可重用的服务并不一定是微服务，微服务架构强调敏捷、独立开发、独立部署、独立扩展，重用在某种程度上范围影响敏捷性
- 微服务架构为了实现其敏捷特性，在SOA约束的基础之上又添加了新的约束
- 微服务之间不能互相依赖，因此要求微服务能够独立部署，独立扩展，微服务之间的依赖越少越好
  - 一个应用只做一件事
  - 不要为外部应用发布API，依赖通过service或者事件搞定
  - 最好通过异步事件交互
  - 每个应用拥有自己独立的数据



采用微服务需要的一些前提条件

- 没有银弹！微服务将单体应用中的复杂性转移到了应用组件之间
- 微服务粒度问题：
  - 如果粒度太细，就需要一个编排服务，其实退回到了小型的SOA架构
  - 如果服务粒度过粗，就不利于独立部署
- 采用微服务之前，需要考虑如下几点
- 微服务架构基础组件：
  - 服务路由
  - 服务注册和发现
  - 配置服务
  - 监控
  - 事件溯源（event sourcing）框架



Agenda

- 谈谈微服务
- **微服务技术选型过程**
- 微服务架构设计的一些思考点
- 融数微服务架构的核心概念和实现
- 融数DevOps平台对微服务的支撑
- 技术团队的组织
- Operation Excellent

微服务技术选型过程

- 微服务架构技术选型的考虑点

社区热度	架构成熟度	学习曲线	可维护性
• 文档多 • 坑少 • 比较容易找到人	• 方便开发 • 方便迁移 • 多协议支持 • 多语言支持	• 基于成熟技术 • 现有知识传承	• 监控能力 • 运维能力

微服务技术选型过程

功能点/服务框架	备选方案				
	Netflix/Spring cloud	Motan	gRPC	Thrift	Dubbo/DubboX
功能定位	完整的微服务框架	RPC框架,但整合了ZK或Consul,实现集群环境的基本的服务注册/发现	RPC框架	RPC框架	服务框架
支持Rest	是 Ribbon支持多种可插拔的序列化选择	否	否	否	否
支持RPC	否	是(Hession2)	是	是	是
支持多语言	是(Rest形式)?	否	是	是	否
服务注册/发现	是(Eureka) Eureka服务注册表，Karyon服务端框架支持服务自注册和健康检查	是(zookeeper/consul)	否	否	是
负载均衡	是(服务端zuul+客户端Ribbon) Zuul-服务,动态路由 云端负载均衡 Eureka(针对中间层服务器)	是(客户端)	否	否	是(客户端)
配置服务	Netflix Archaius Spring cloud Config Server 集中配置	是(zookeeper提供)	否	否	否
服务调用链监控	是(zuul) Zuul提供边缘服务,API网关	否	否	否	否
高可用/容错	是(服务端Hystrix+客户端Ribbon)	是(客户端)	否	否	是(客户端)
典型应用案例	Netflix	Sina	Google	Facebook	
社区活跃程度	高	一般	高	一般	已经不维护了
学习难度	中等	低	高	高	低
文档丰富度	高	一般	一般	一般	高
其他	Spring Cloud Bus为我们的应用程	支持降级	Netflix内部在开发集成gRPC	IDL定义	实践的公司比较多

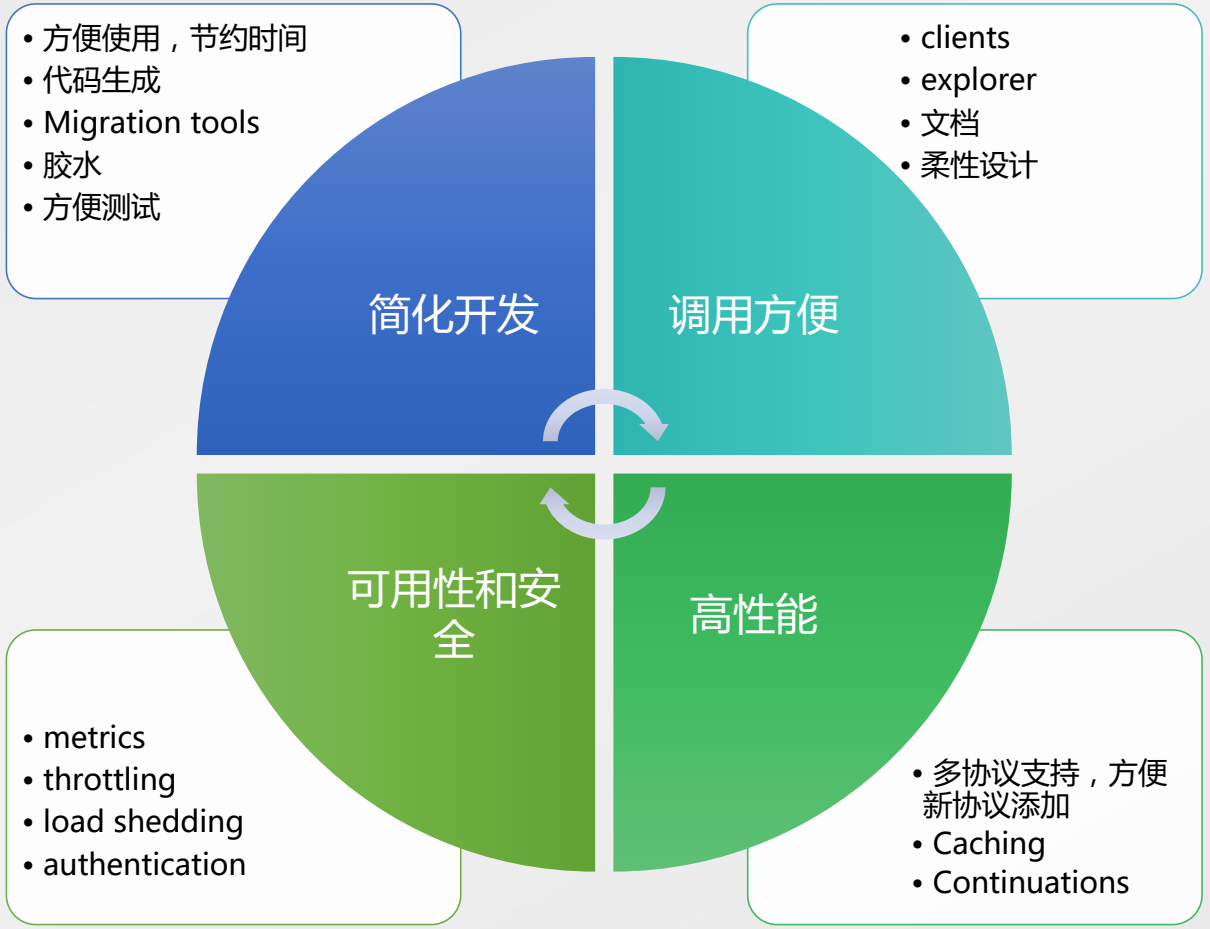
微服务技术选型过程

- 目前团队主要采用Spring Boot + RestEasy的方式实现服务化
  - 首先支持rest
  - 现有业务代码的迁移不希望改动太大
  - 小团队，希望能够有一个比较全面的解决方案
- 结论
  - Netflix提供了比较全面的解决方案
  - Spring Cloud对于Netflix的封装比较全面
  - Spring Cloud基于Spring Boot，团队有基础
  - Spring Cloud提供了Control Bus能够帮助实现监控埋点
  - 业务应用部署在阿里云，Spring Cloud对12factors以及Cloud-Native的支持，有利于在云环境下使用

Agenda

- 谈谈微服务
- 微服务技术选型过程
- **微服务架构设计的一些思考点**
- 融数微服务架构的核心概念和实现
- 融数DevOps平台对微服务的支撑
- 技术团队的组织
- Operation Excellent

微服务架构设计的一些思考点

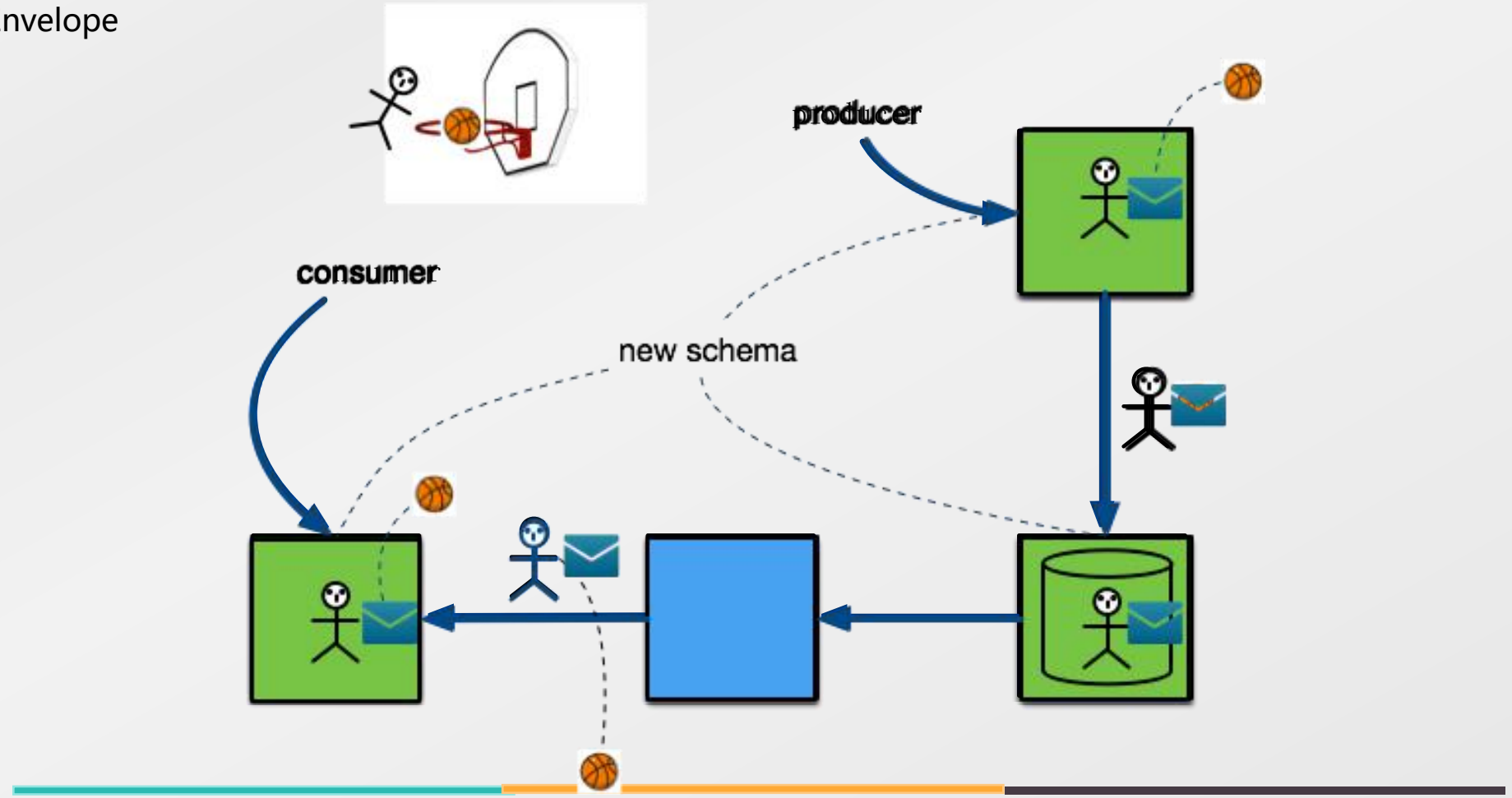


Agenda

- 谈谈微服务
- 微服务技术选型过程
- 微服务架构设计的一些思考点
- **融数微服务架构的核心概念和实现**
- 融数DevOps平台对微服务的支撑
- 技术团队的组织
- Operation Excellent

核心模型

- Envelope



核心模型

Envelope API

```
public interface Envelope extends Serializable {
  public <T> void set(Class<T> clazz, T o);
  public <T> void set(T o);
  public <T> T get(Class<T> clazz);
  public <T> T get();
  ...
}
```

Producer

```
Customer customer = ...;
// Seal in envelope
Envelope e = factory.newEnvelope();
e.set(Customer.class, customer);
```

Consumer

```
Envelope e = …;

// Open Envelope
Customer customer = e.get(Customer.class);
```

核心模型

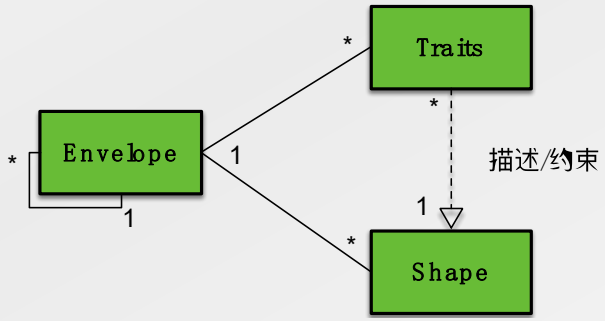
- Shapes（定义数据类型）
  - Simple
  - List
  - Map
  - Structure

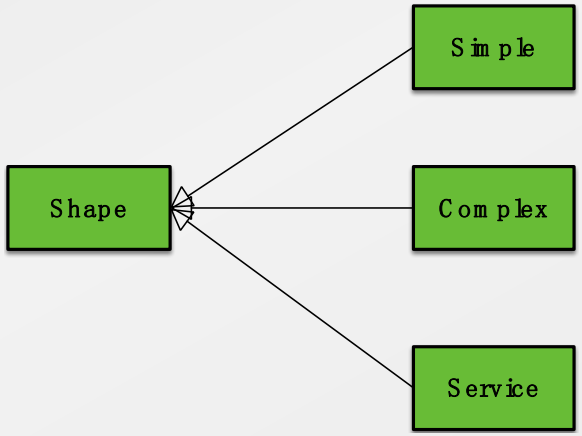
核心模型

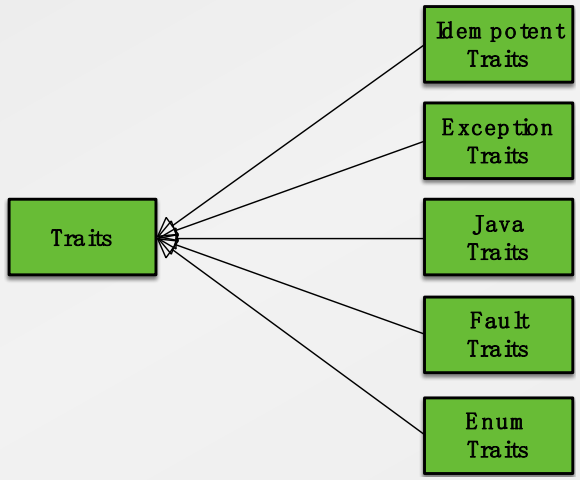
- Traits（定义Shaps的行为）

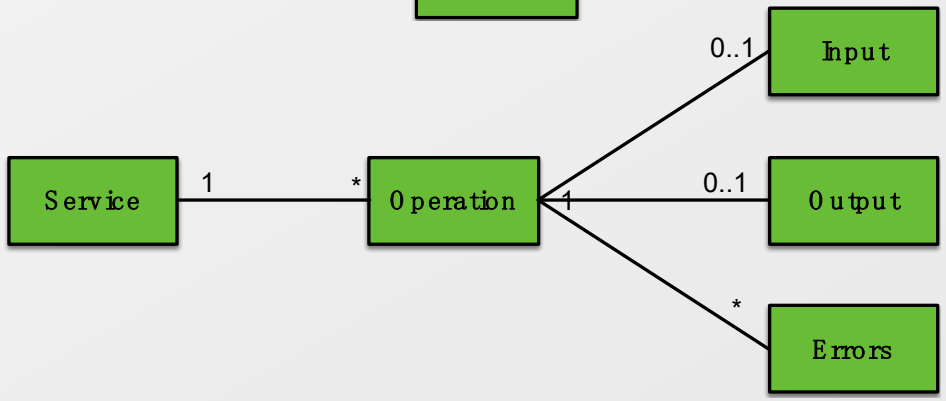
```
<pattern target=“location” >
    <regex value=“[0-9]{5}(?:\-[0-9]{4})?” />
</pattern>
<string name=“location” />
```

核心模型

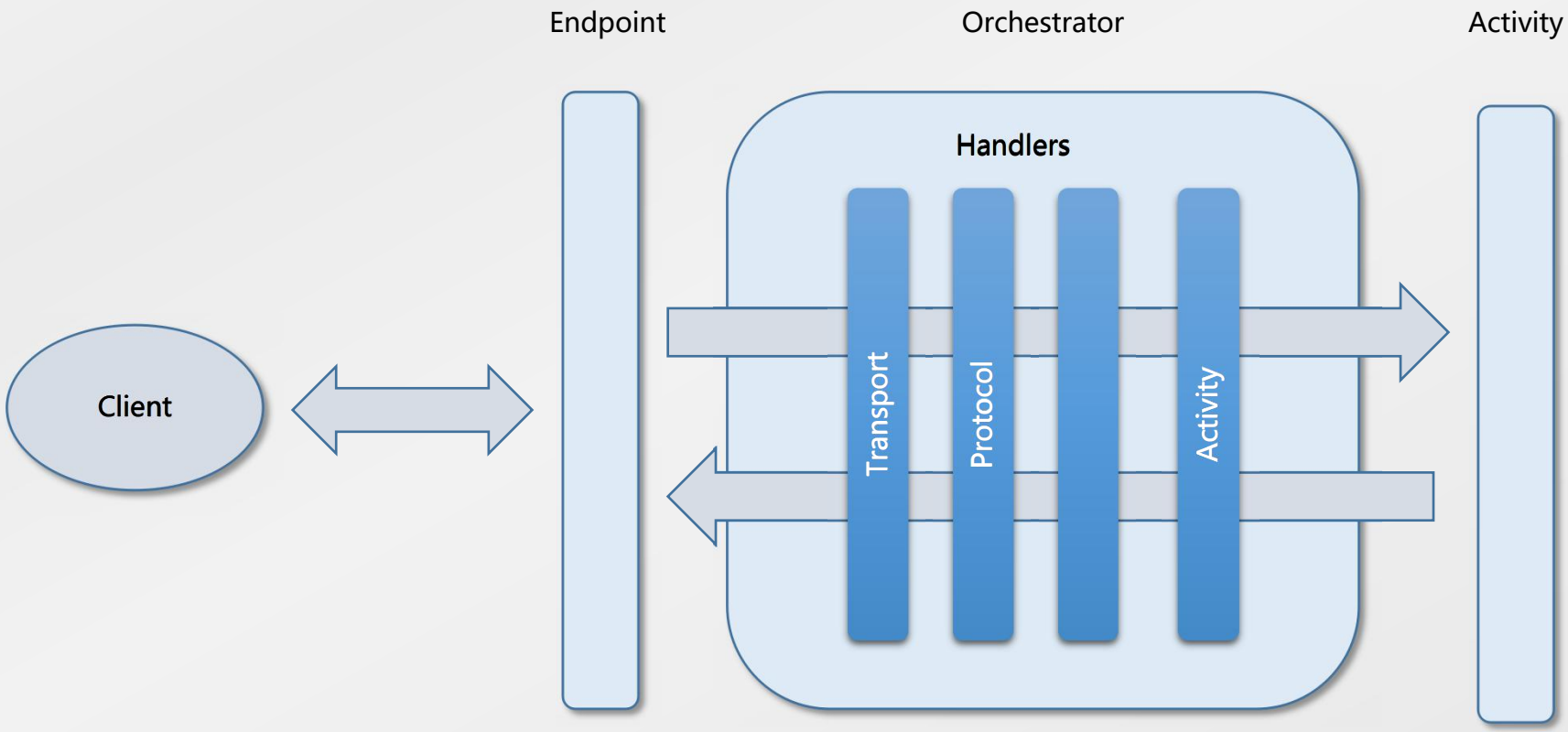




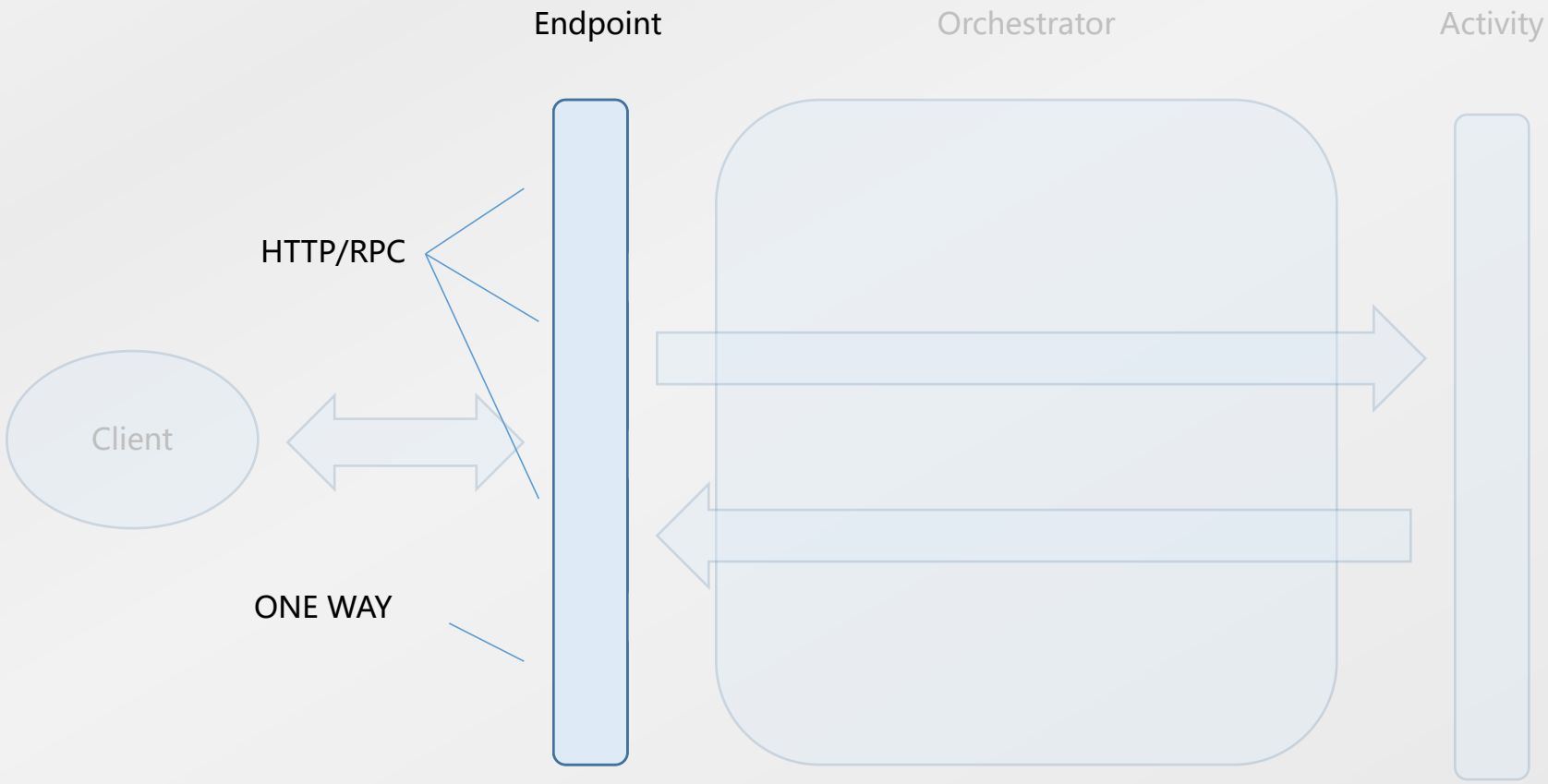




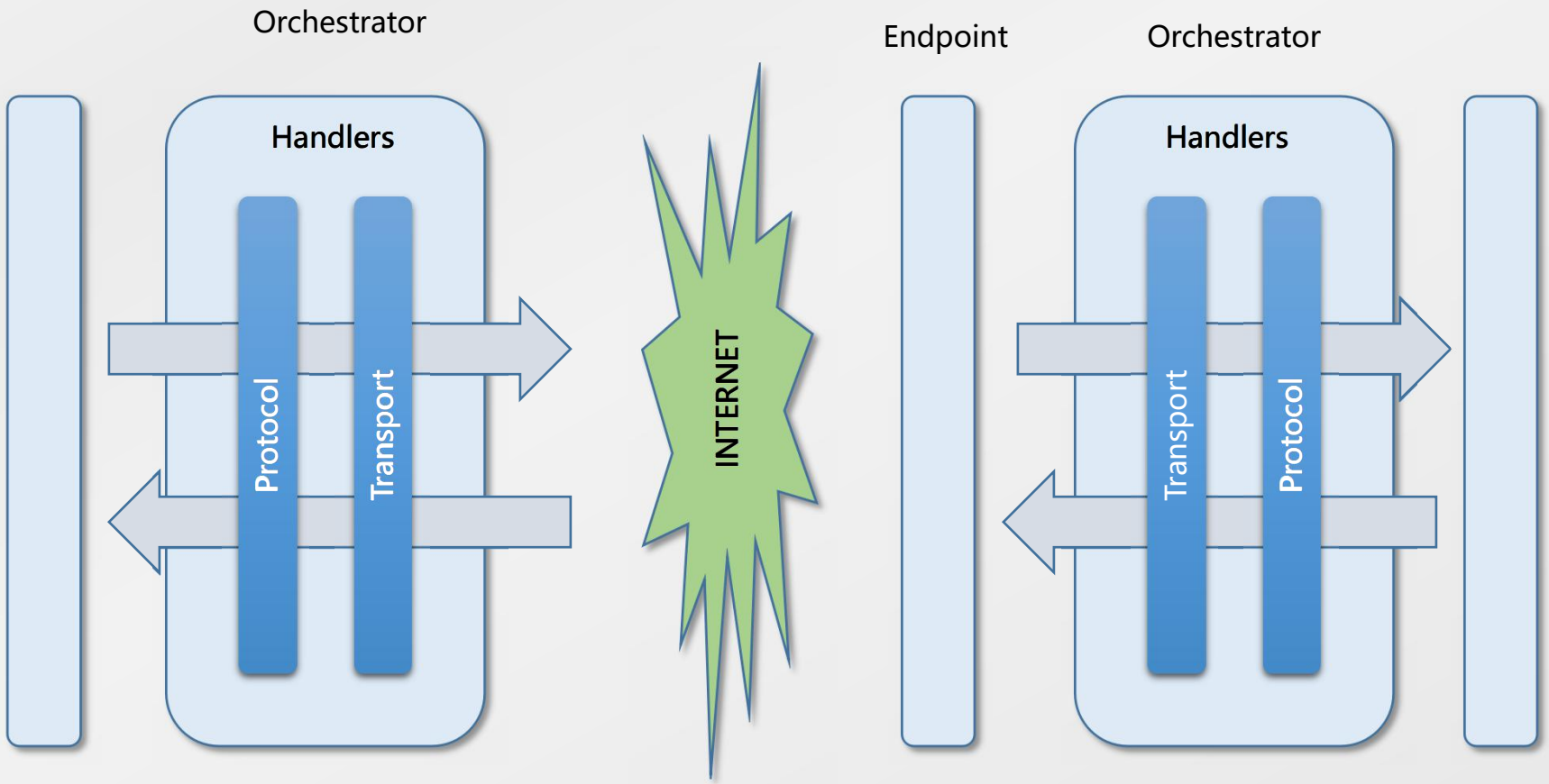
服务处理实现



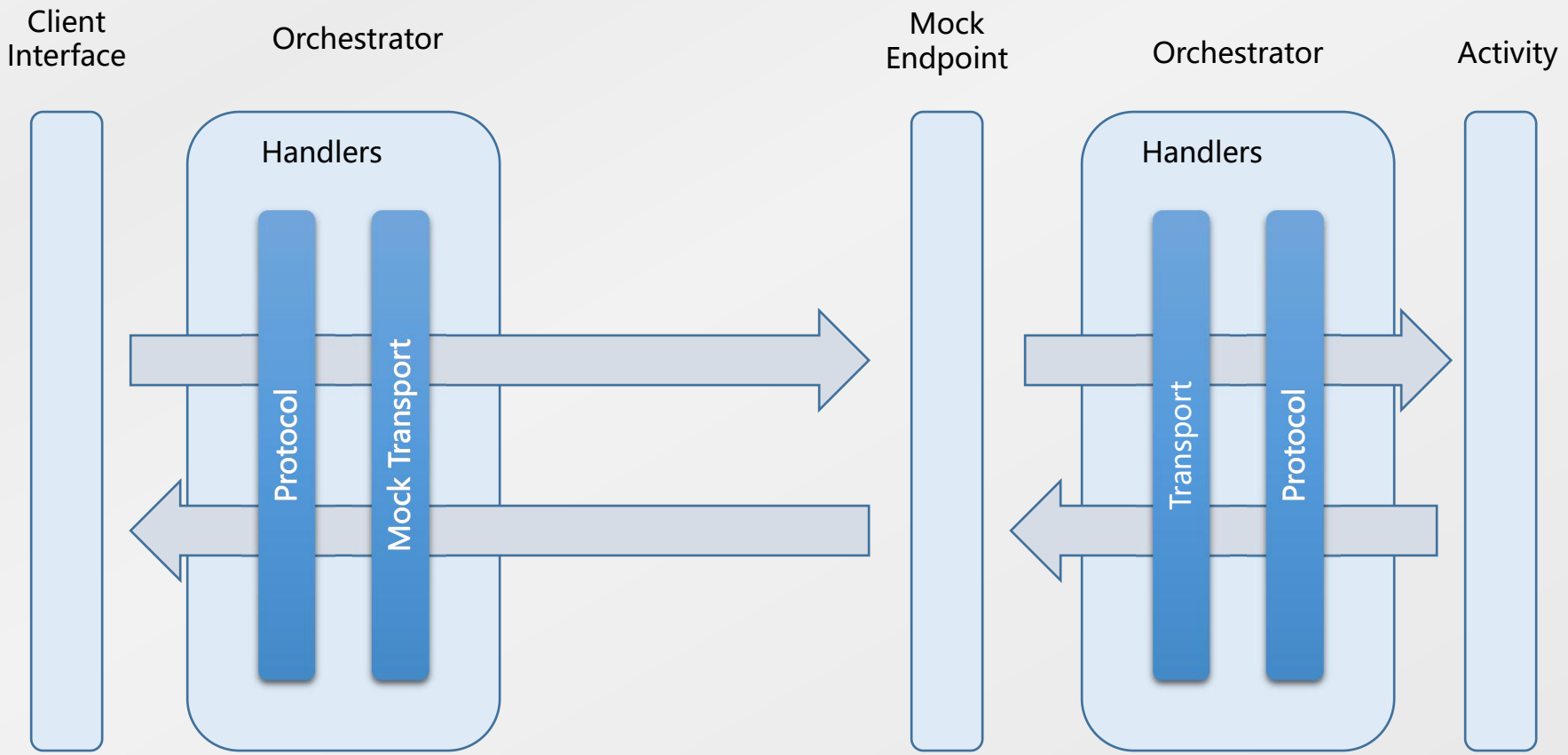
插件式EndPoint



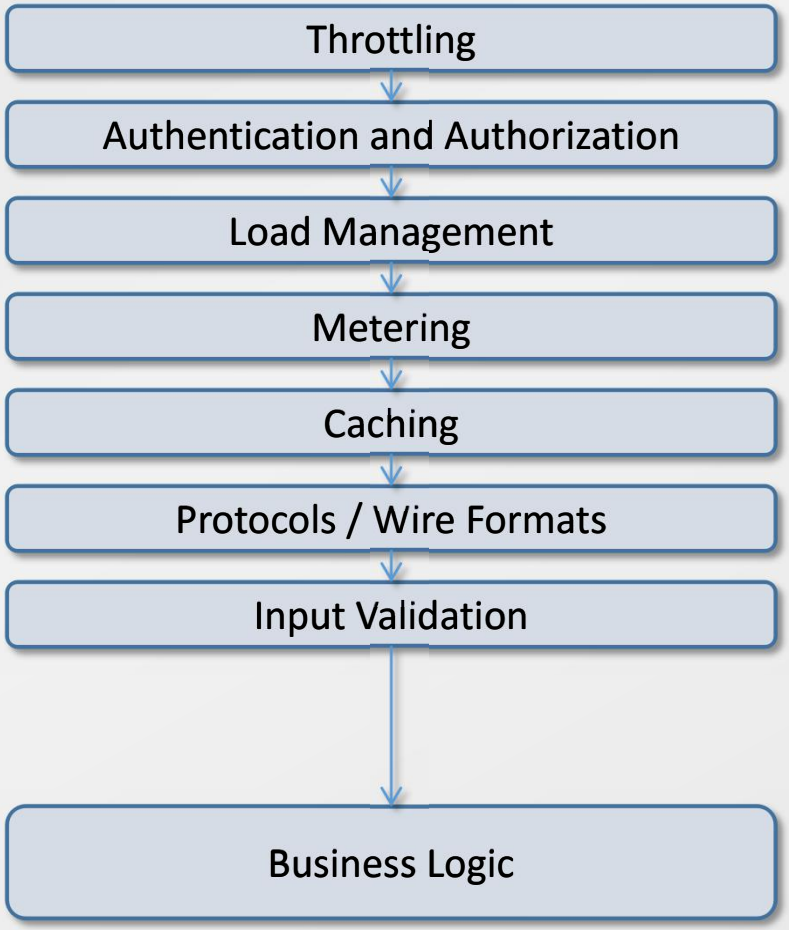
Orchestrator重用



集成测试



服务处理责任链



编写并配置服务

```
<bean id="ochestrator" class="cn.rongcapital.msp.service.helper.OchestratorHelper">
        <constructor-arg>
                <ref bean="handlerChain"/>
        </constructor-arg>
</bean>
<bean id="handlerChain" class="cn.rongcapital.msp.service.helper.ChainHelper">
        <properties>
                <list>
                        <bean class="cn.rongcapital.msp.service.HttpHandler"/>
                        <bean class="cn.rongcapital.msp.service.PintHandler"/>
                        <bean class="cn.rongcapital.msp.service.ThrottlingHandler"/>
                </list>
        </properties>
</bean>
<bean id="server" class="cn.rongcapital.msp.service.server.UndertowServer">
        <constructor-arg>
                <bean class="cn.rongcapital.msp.service.EndpoingConfig">
                        <property name="metricsFactory" ref="metricsFactory"/>
                        <property name="ochestrator" ref="ochestrator"/>
                        <property name="uri" ref="http://0.0.0.0:8080"/>
                </bean>
        </constructor-arg>
</bean>
```

Spring Cloud

- 遵循12 Factors
- 模式
  - Configuration Management
  - Service discovery
  - Circuit breakers
  - Intelligent routing
  - Control bus
  - One-time tokens
  - Global locks
  - Leadership election
  - Distributed sessions

配置管理

•配置信息统一管理

- 提供RESTfulAPI
- 配置信息允许动态变更
- 支持多种数据类型的配置内容

•分级配置

- 将配置分为多个层级：全局、系统、应用、阶段
- 最终的配置集合，是所有层级的配置的并集
- 同名的配置，末端会覆盖上层的配置（可配置）

•版本控制

- 配置项和配置集合具有版本信息，发生变更后会保留历史记录
- 配置集合可以随时回退到历史版本

•配置变更主动通知

- 客户端订阅配置变更信息，在配置发生变更后会立即收到通知和新的配置集合

•客户端与应用程序集成

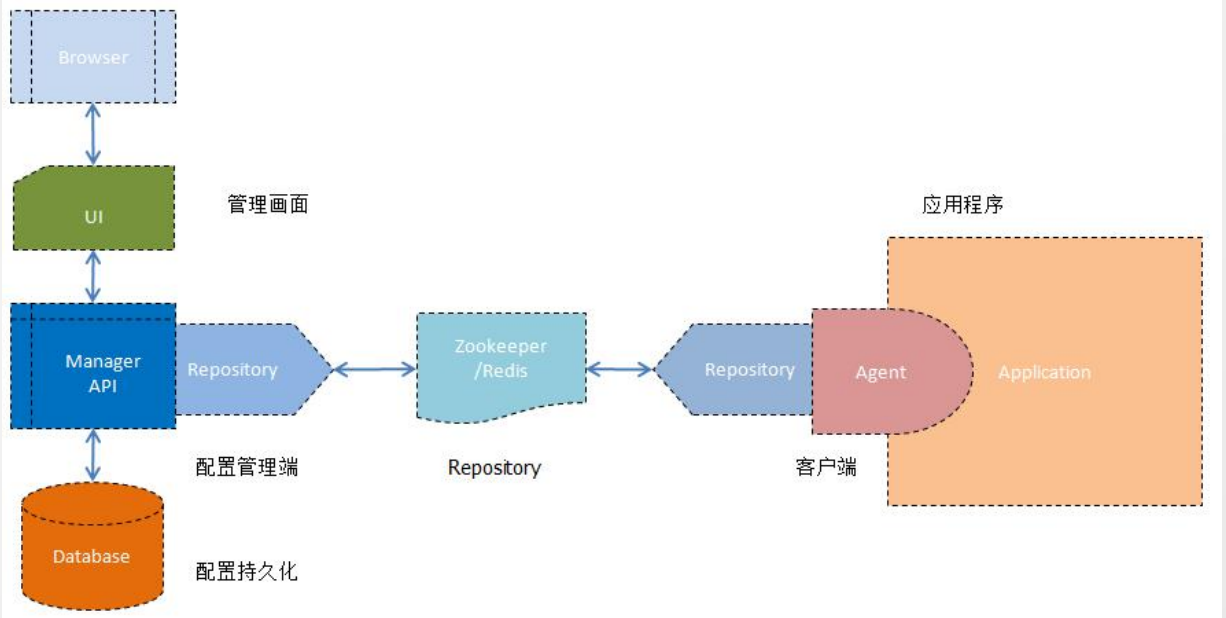
- 集成spring-boot
- 提供接口，允许在运行期间动态获取配置

•本地存储

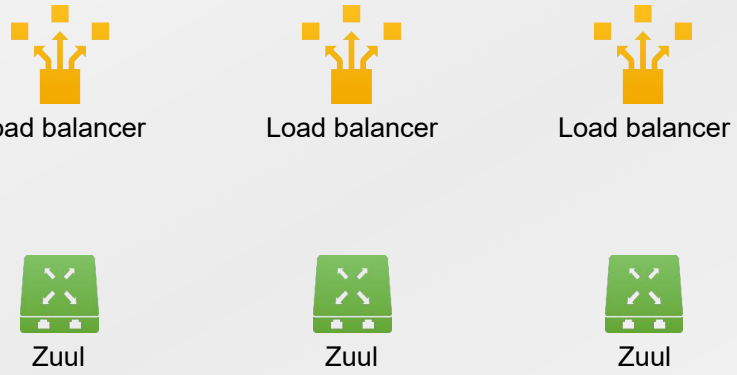
- 在客户端本地保存当前配置，保证应用程序在不能访问Repository下也可以运行

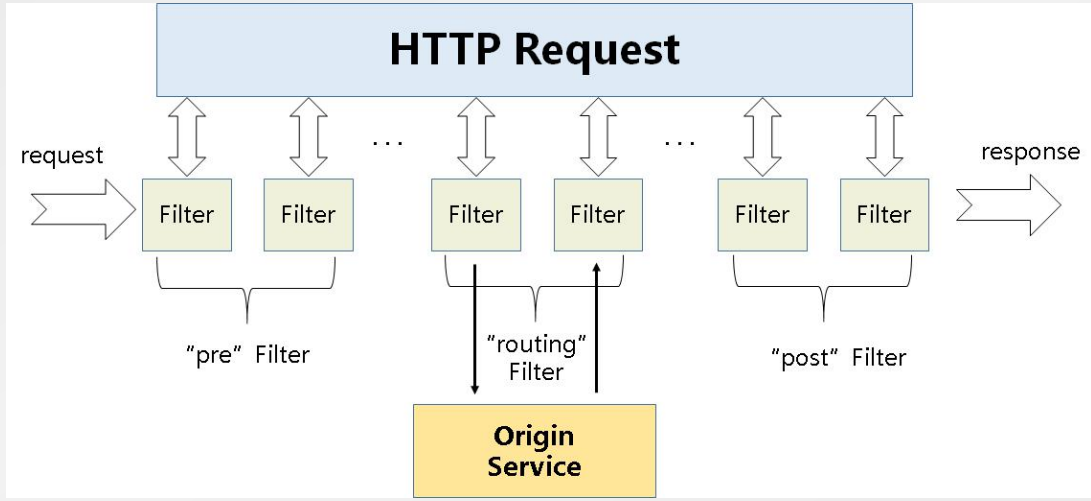
•充分抽象，允许多实现

- 配置信息可以保存在数据库里，也可以保存在NoSQL里
- Repository可以使用Zookeeper或Redis等，也可以自行编写
- 允许自定义配置信息的序列化方式，默认使用 JDK 的序列化方式



Zuul





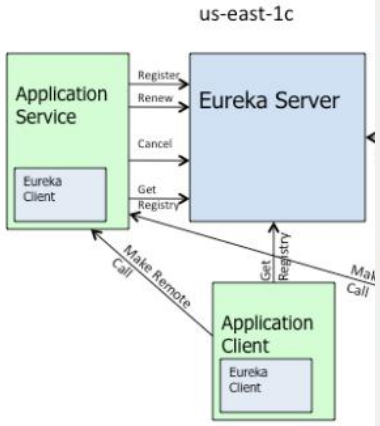

```java
@SpringBootApplication
@Controller
@EnableZuulProxy
@EnableAutoConfiguration(exclude = {
        org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration.class,
        org.springframework.boot.actuate.autoconfigure.ManagementWebSecurityAutoConfiguration.class})
public class Application {

    public static void main(String[] args) {
        new SpringApplicationBuilder(Application.class).web(true).run(args);
    }

}
```

服务发现: Eureka Server

- Eureka client会缓存服务
- Eureka server的注册信息
- Eureka的注册只针对app
- 服务每隔30秒向Eureka
- 如果在15分钟内有85%的
- Eureka Server之间的数据



spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2016-10-06T16:05:59 +0800
Uptime	00:01
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	2

DS Replicas

Instances currently registered with Eureka

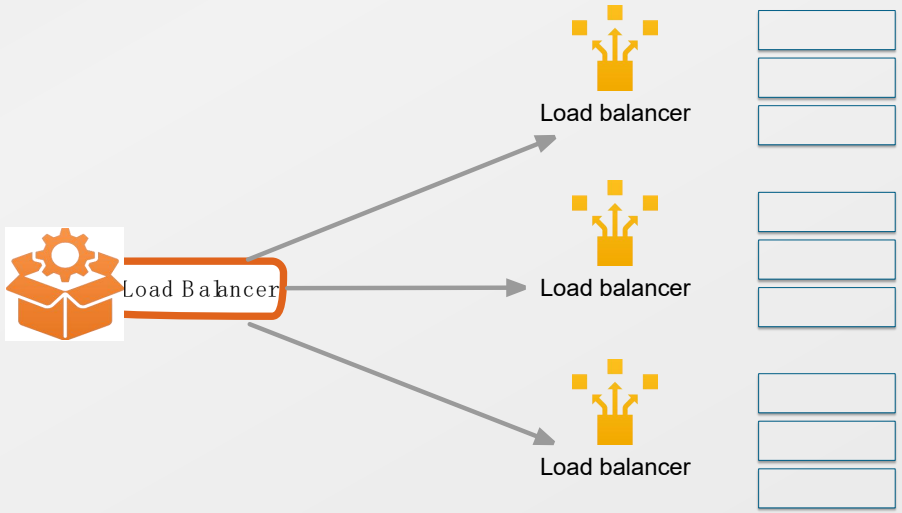
Application	AMIs	Availability Zones	Status
TURBINE	n/a (1)	(1)	UP (1) - localhost:turbine:9090
ZUUL-1	n/a (1)	(1)	UP (1) - localhost:zuul-1:8080

General Info

Name	Value
total-avail-memory	396mb
environment	test

Ribbon

策略名	策略描述	实现说明
BestAvailableRule	选择一个最小的并发请求的server	逐个考察Server，如果Server被tripped了，则忽略，在选择其中ActiveRequestsCount最小的server。
AvailabilityFilteringRule	过滤掉那些因为一直连接失败的被标记为circuit tripped的后端server，并过滤掉那些高并发的的后端server（active connections 超过配置的阈值）	使用一个AvailabilityPredicate来包含过滤server的逻辑，其实就就是检查status里记录的各个server的运行状态
WeightedResponseTimeRule	根据响应时间分配一个weight，响应时间越长，weight越小，被选中的可能性越低。	一个后台线程定期的从status里面读取评价响应时间，为每个server计算一个weight。当刚开始运行，没有形成statas时，使用roubine策略选择server。
Retry Rule	对选定的负载均衡策略机上重试机制。	在一个配置时间段内当选择server不成功，则一直尝试使用subRule的方式选择一个可用的server 使用举例： IRule rule = new RetryRule(new RoundRobinRule(), 200); 200表示retry的时间间隔
RoundRobinRule	roundRobin方式轮询选择server	轮询index，选择index对应位置的server
RandomRule	随机选择一个server	在index上随机，选择index对应位置的server
ZoneAvoidanceRule	复合判断server所在区域的性能和server的可用性选择server	使用ZoneAvoidancePredicate和AvailabilityPredicate来判断是否选择某个server，前一个判断判定一个zone的运行性能是否可用，剔除不可用的zone（的所有server），AvailabilityPredicate用于过滤掉连接数过多的Server。



扩展点：自定义负载均衡策略：

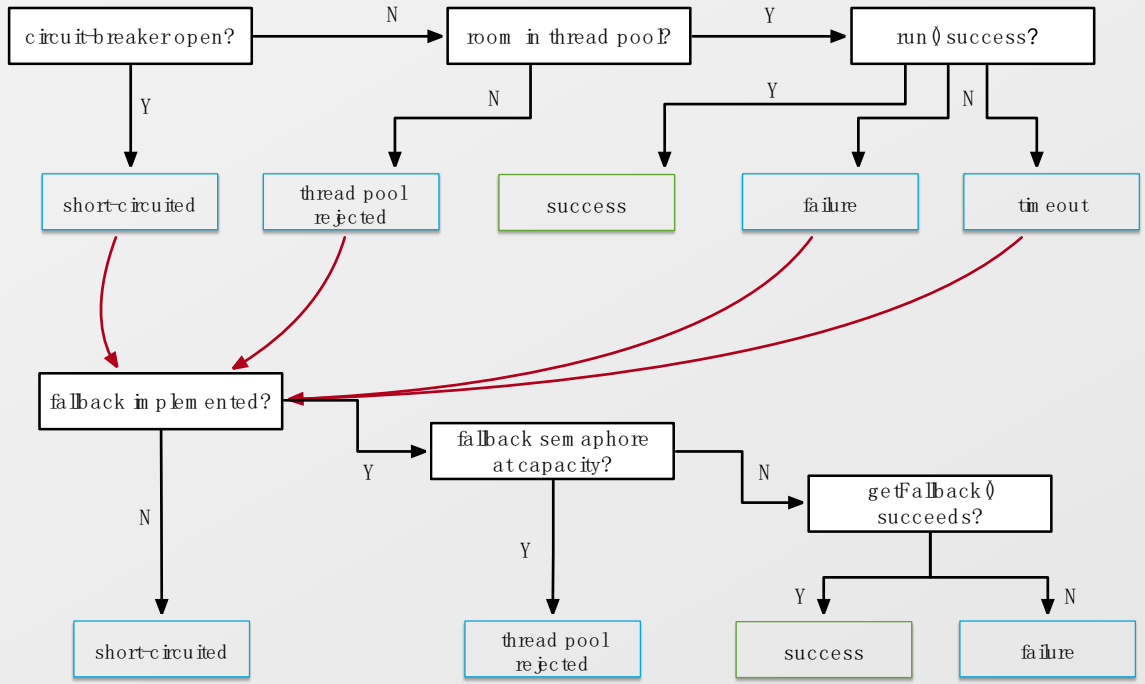
ribbon.client.name = myclient

myclient.ribbon.NFLoadBalancerClassName=均衡器类名

myclient.ribbon.NFLoadBalancerRuleClassName=均衡策略类名

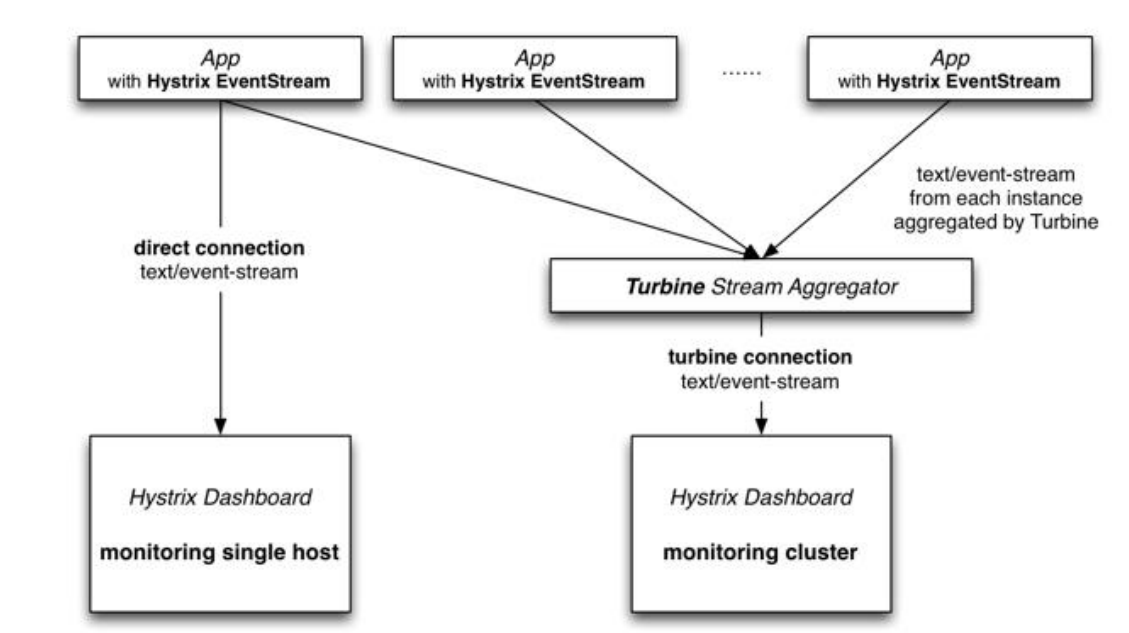
熔断器：Hystrix

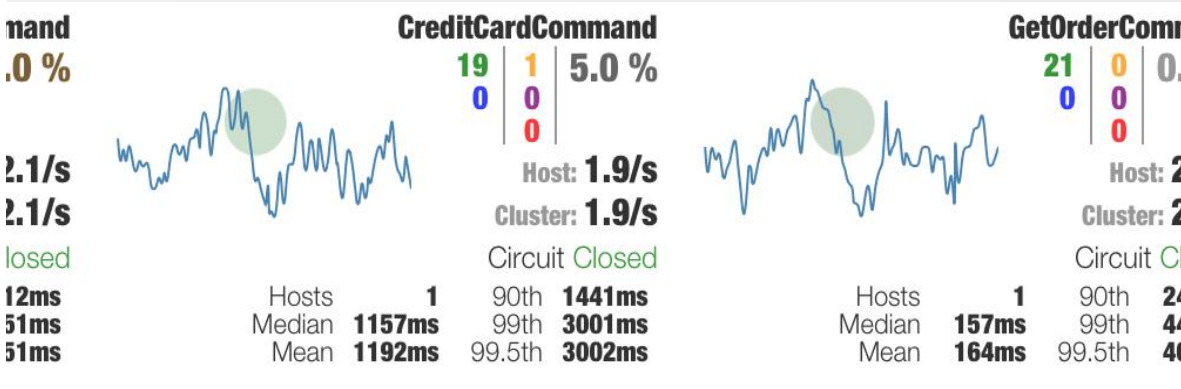
- Hystrix核心由RxJava驱动，是一个基于观察者模式的事件回调库；
- Hyxtrix的核心处理逻辑是将调用包装成Command，将对依赖的调用转换成Command API调用；
- circuitBreaker.allowRequest() 判定熔断是否开启;
- Hystrix熔断器本质是一组状态机，是fast-fail设计思想的体现;
- 处理请求时判定熔断器是否开启，开启使用备选方案(如定义的fallback方法)往下执行，未开启按正常逻辑执行;
- 熔断器依赖metrics收集的health指标，对错误请求数及错误百分比进行条件判定。



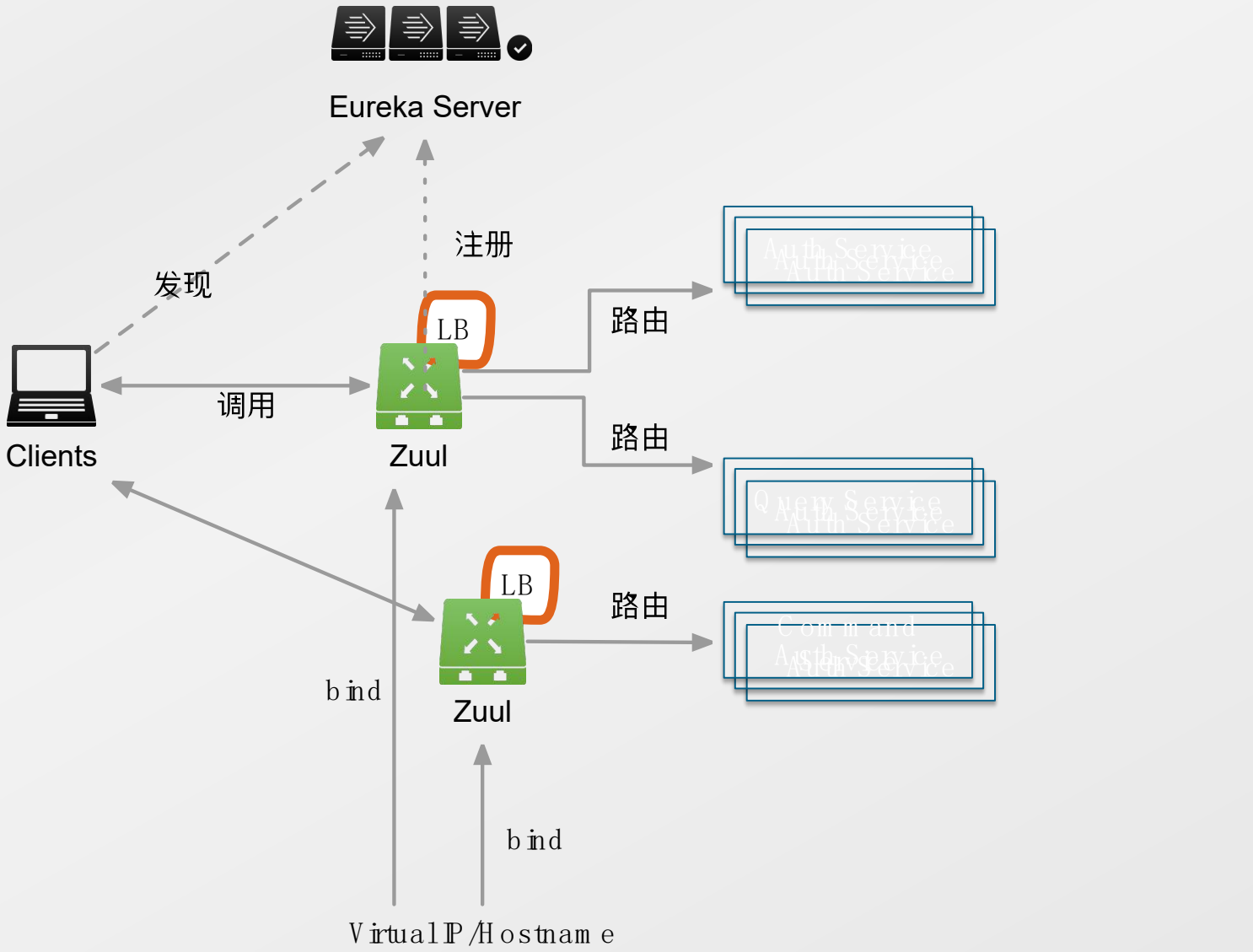
Hystrix UI & Turbine

- Netflix通过turbine聚合Hystrix的监控流信息
- Hystirx的dashboard监控信息只支持实时监控
- Netflix内部会把收集到的数据写入到Atlas系统，我们通过Kafka + ELK收集和存储





服务端治理

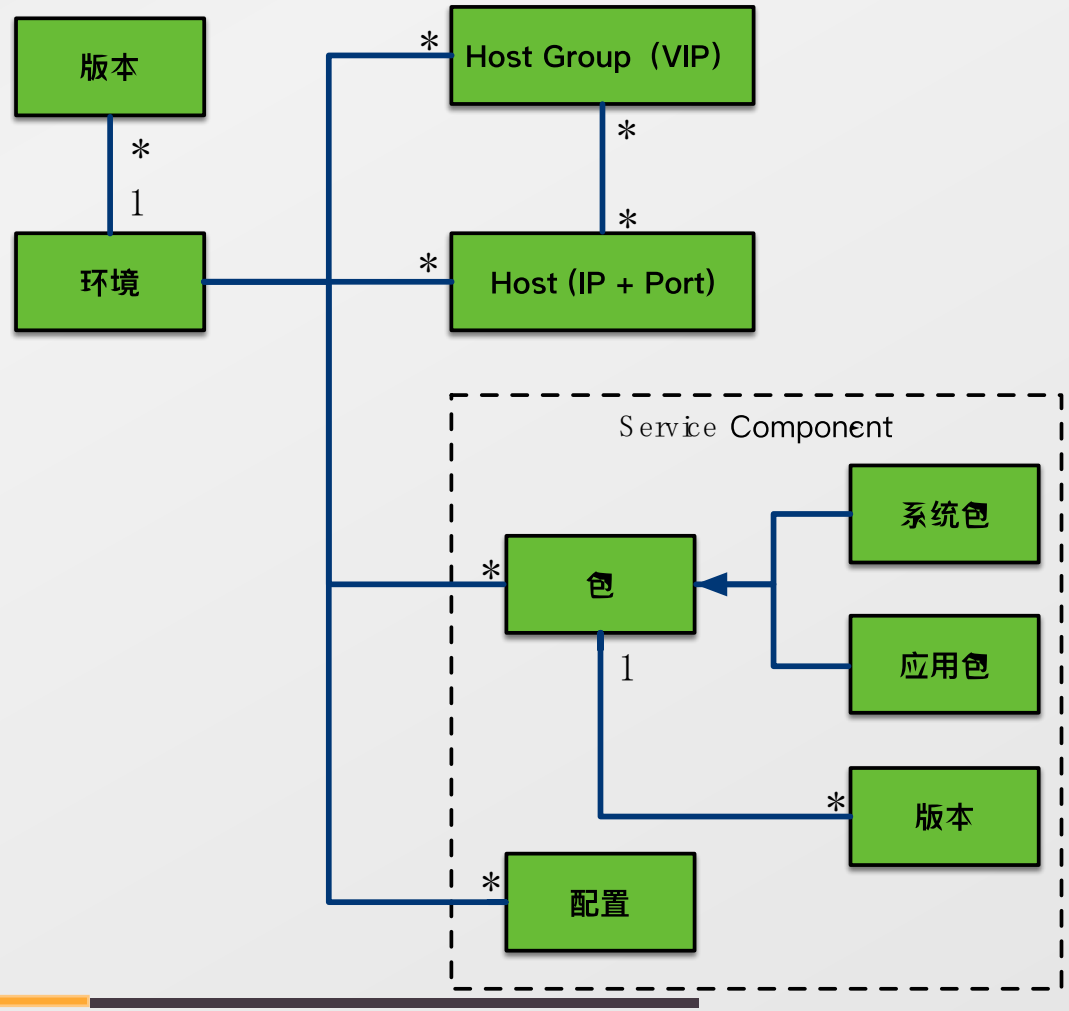


Agenda

- 谈谈微服务
- 微服务技术选型过程
- 微服务架构设计的一些思考点
- 融数微服务架构的核心概念和实现
- **融数DevOps平台对微服务的支撑**
- 技术团队的组织
- Operation Excellent

部署概念的抽象

- 包是部署最小单位
- 服务组件由包、配置组成
- 环境包含服务组件以及运行它所依赖的Host或者Host Group
- 有版本才能回滚



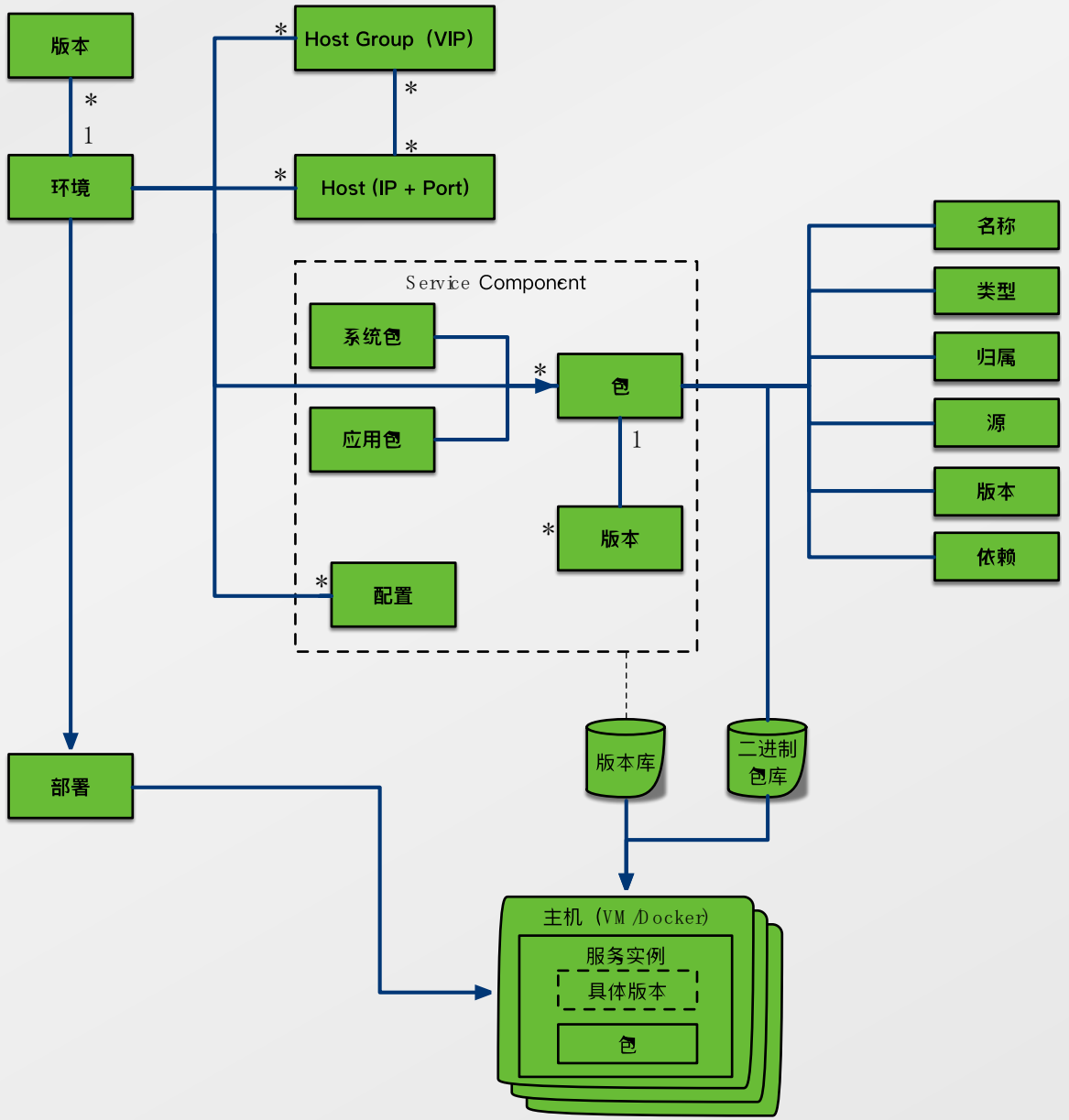
服务的部署

服务组件 =

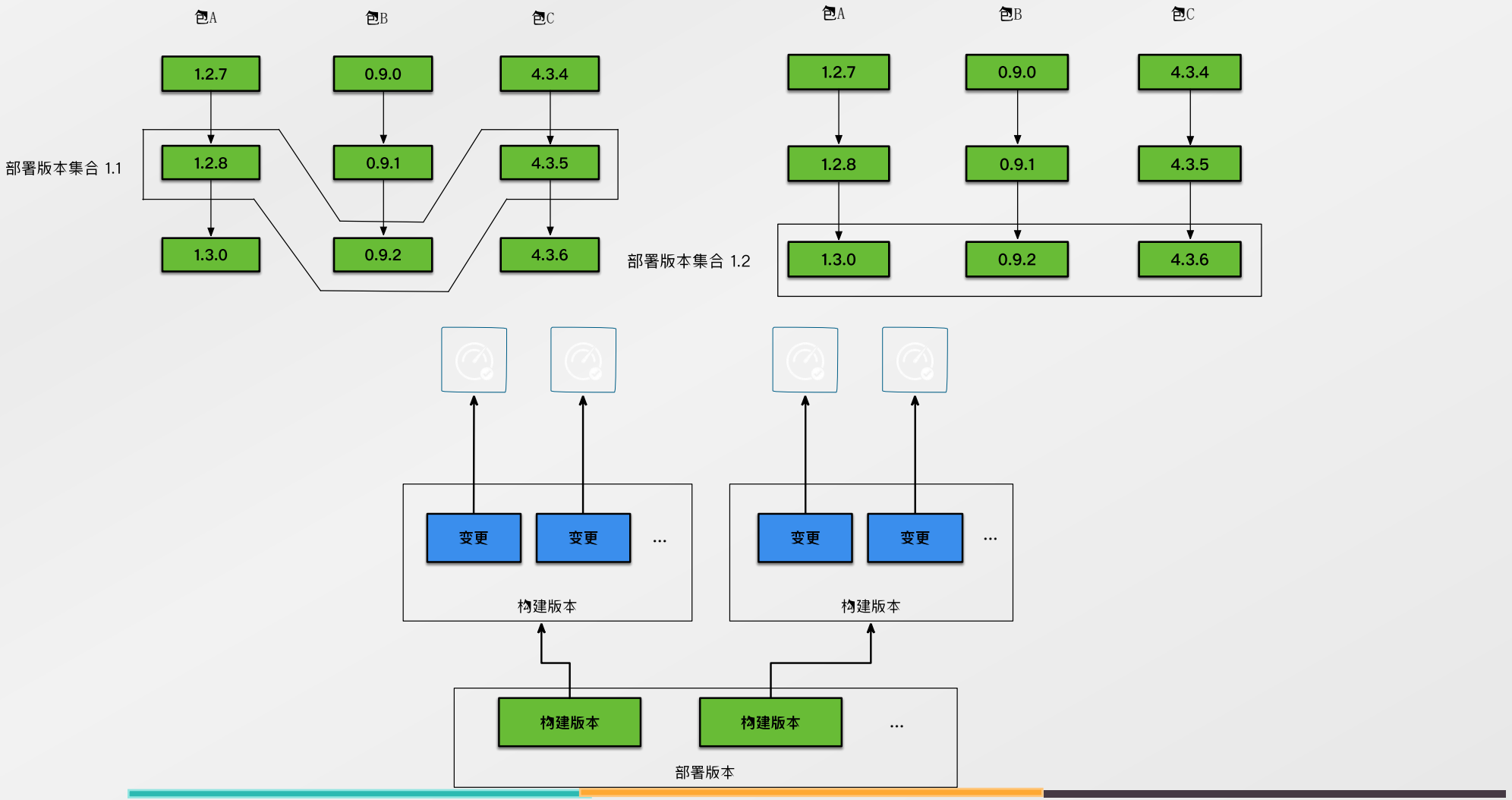
（可运行代码 + 配置）&

（依赖 + 配置）&

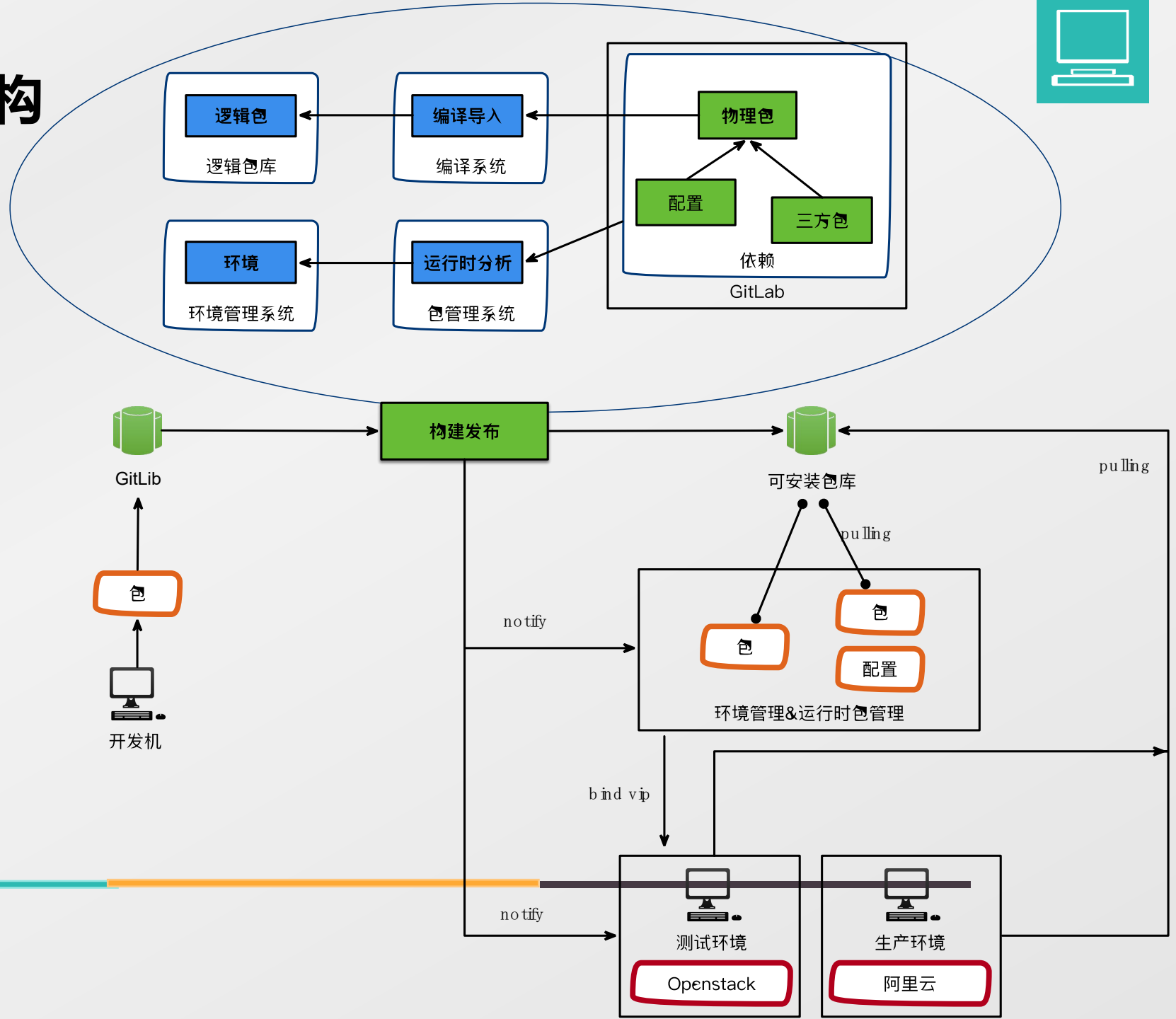
（基础设施 + 配置）



部署结构



DevOps总体架构

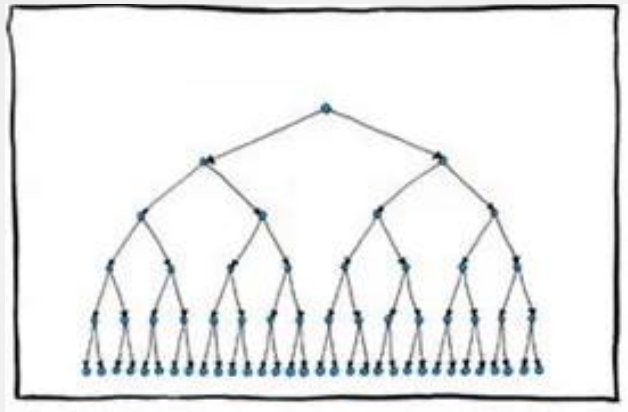


Agenda

- 谈谈微服务
- 微服务技术选型过程
- 微服务架构设计的一些思考点
- 融数微服务架构的核心概念和实现
- 融数DevOps平台对微服务的支撑
- **技术团队的组织**
- Operation Excellent

技术团队组织 – 小团队

- 康威定律
- Two-Pizza Team

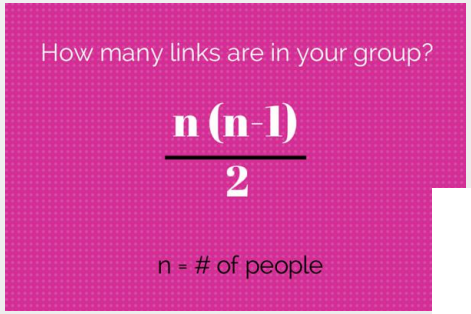


魔数

- 团队成员以7+/-2人为最佳，团队成员水平相当
- 保持团队规模在个位数，如果超过，拆分团队

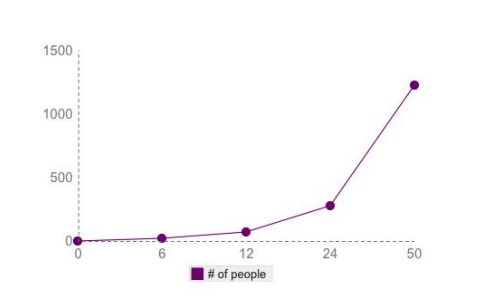
干杯规则

- 团队成员能够近距离的沟通，增进彼此的了解



团队信息透明化

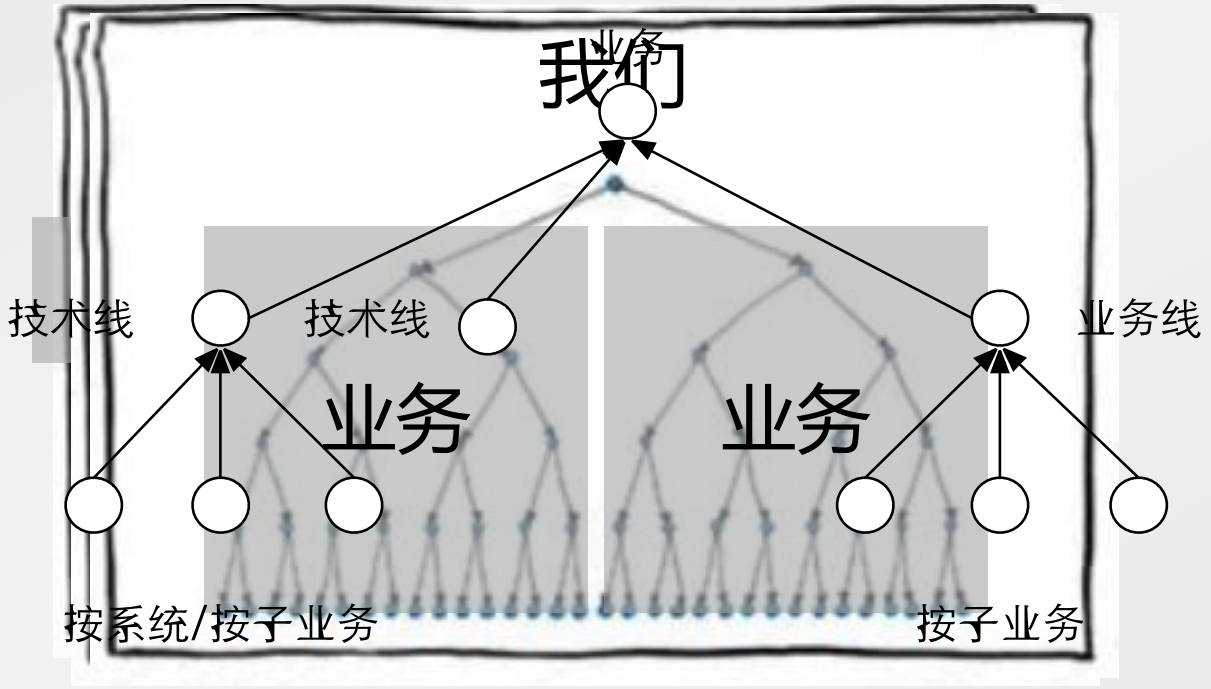
- 团队成员能够非常清楚的了解团队的目标以及其他团队成员的工作



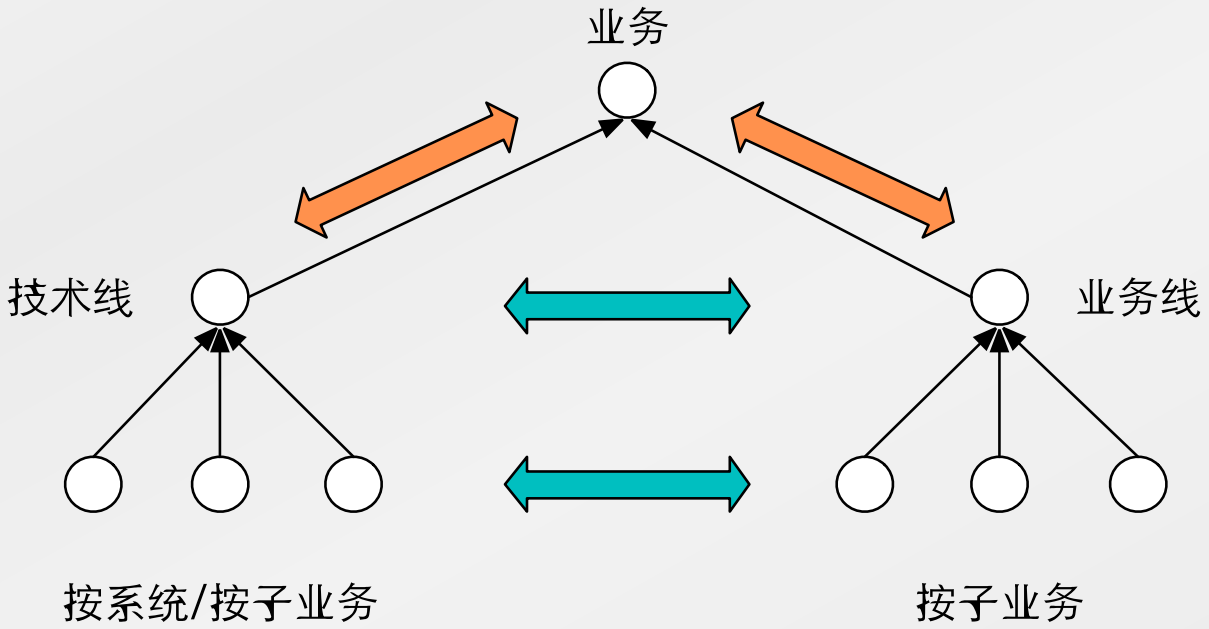
去中心化

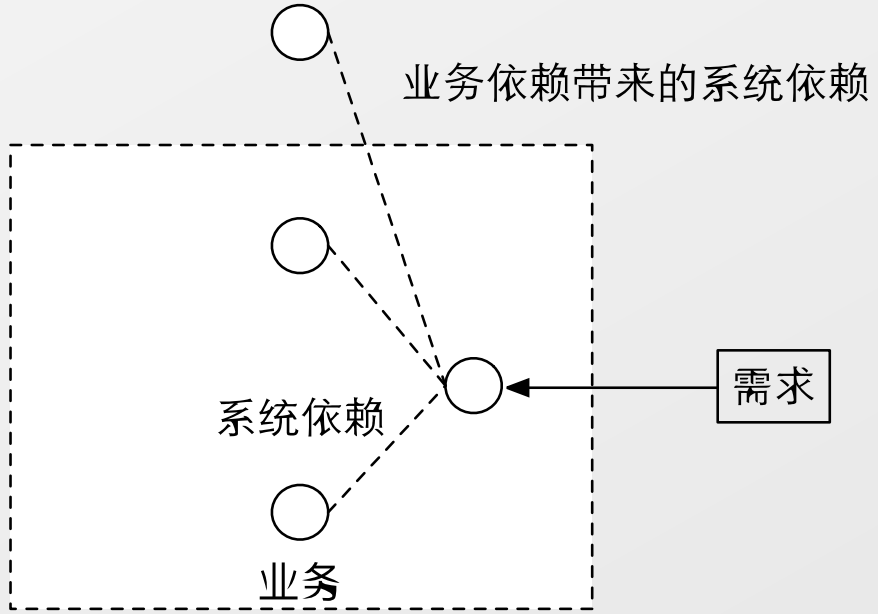
- 鼓励创新和自治，团队自己决定使用的技术，鼓励团队间的技术竞争
- 胜出的团队的方案会被基础架构部门采纳并全公司推广

技术团队组织 – 团队划分



技术团队组织 – 团队间合作





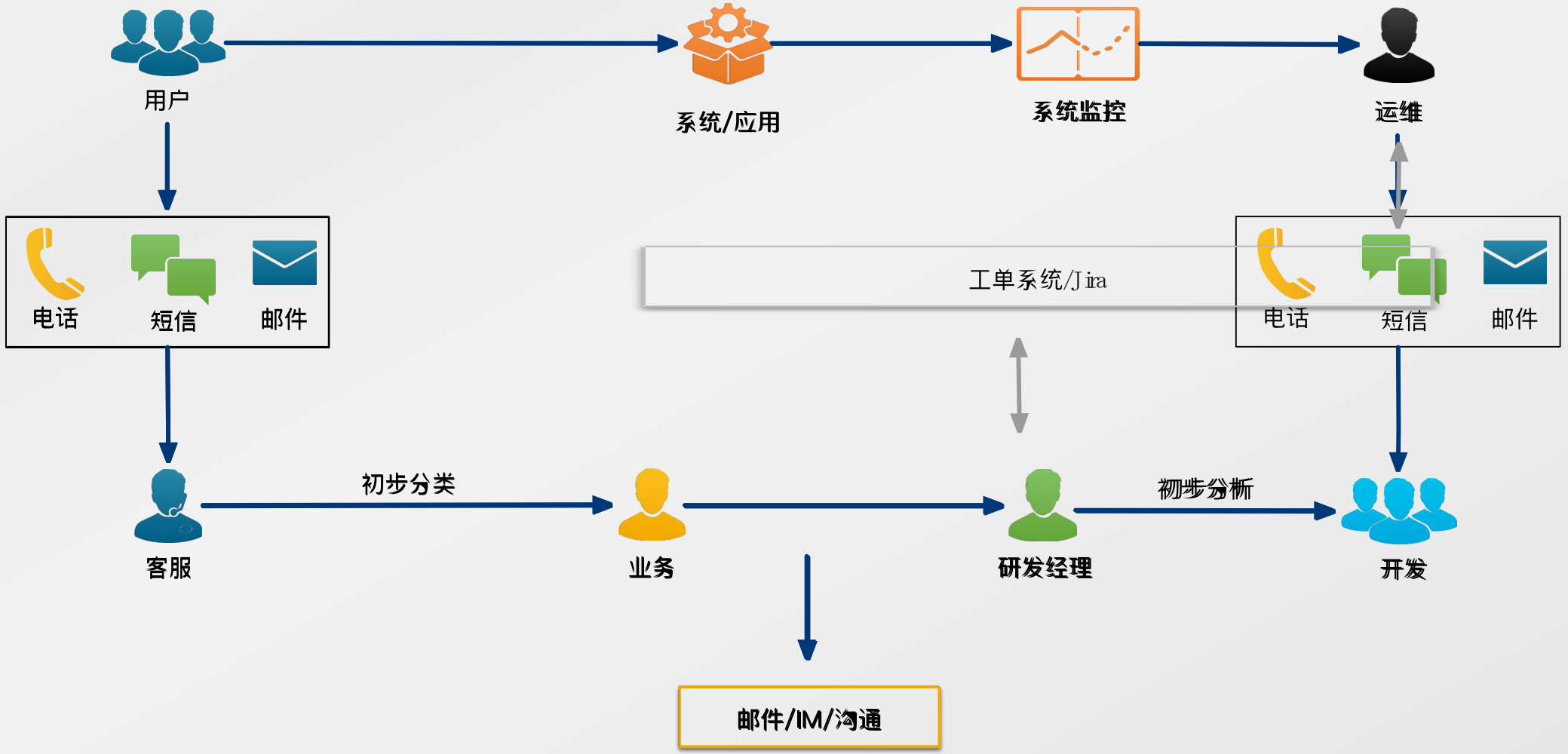
技术团队组织 – 结果导向

- 主人翁意识（Ownership）
- 行动力（Bias for Action）
- 吃自己的狗粮（Eat your dog food）
  - 工程师负责从需求调研、设计、开发、测试、部署、维护、监控、功能升级等一系列的工作，也就是说软件工程师负责应用或者服务的全生命周期的所有工作
  - 运维是团队成员的第一要务，在强大的自动化运维工具的支撑下，软件工程师必须负责服务或者应用的SLA
- 让开发人员参与架构设计，而不是架构师参与开发

Agenda

- 谈谈微服务
- 微服务技术选型过程
- 微服务架构设计的一些思考点
- 融数微服务架构的核心概念和实现
- 融数DevOps平台对微服务的支撑
- 技术团队的组织
- **Operation Excellent**

传统的运维流程

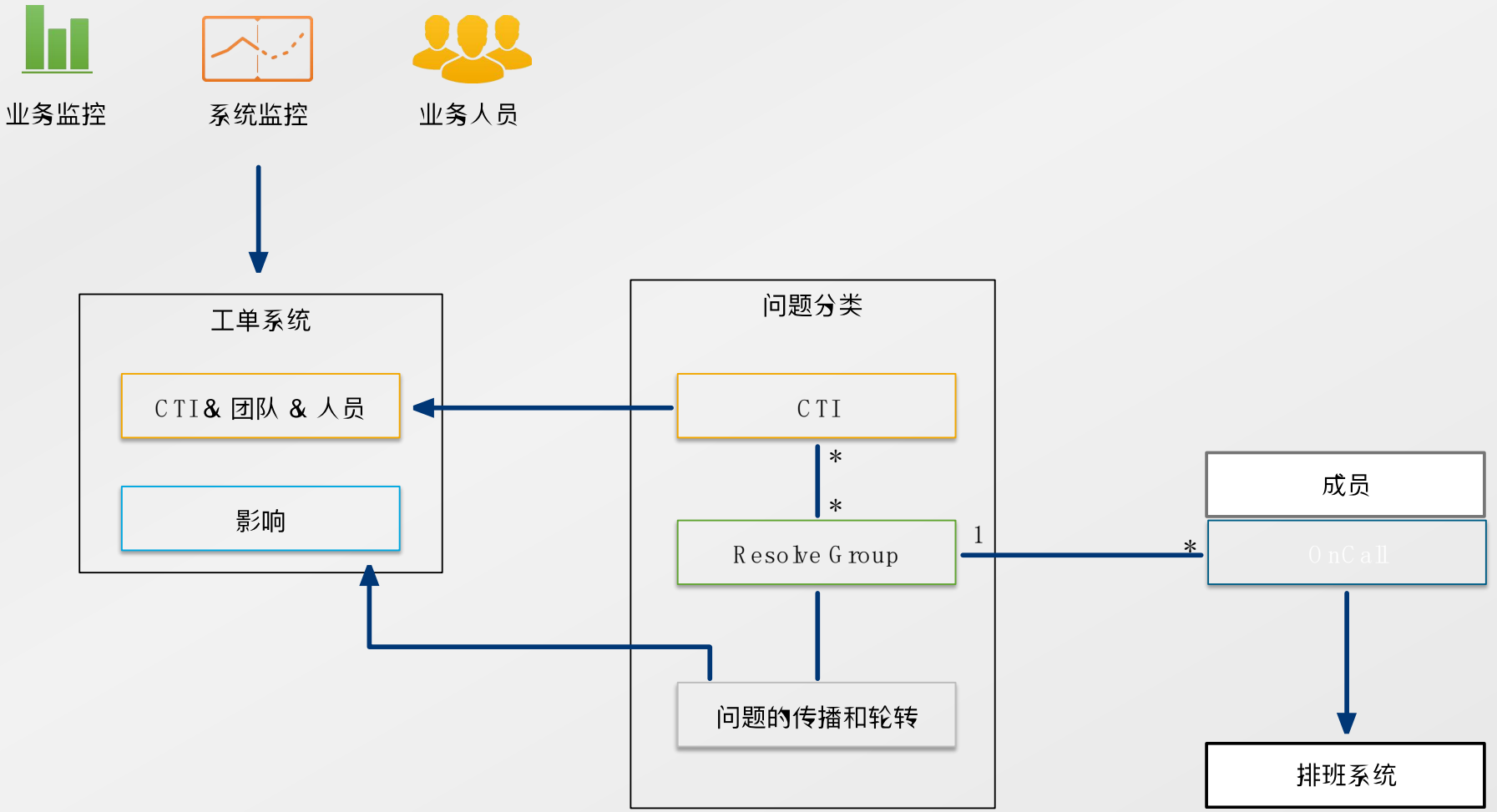


传统的运维流程

- 如何定位问题
- 如何定位人员
- 时效性监控
- 自动化问题反馈

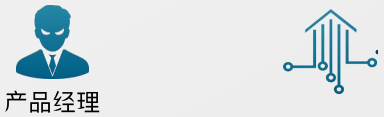
- 一旦发现问题，我们希望能够迅速定位并安排相应的业务和开发人员及时跟进，直到问题解决

新的运维机制



Operation Excellent

- 业务驱动改进
- 研发经理监控
- Sev1 & Sev2
- 每周分析
- COE



谢谢！