

ThoughtWorks®

微服务架构

秦玉林@2017

概览



- 传统单体应用
- 微服务是什么
- 微服务设计原则



- Spring Cloud 介绍
- 常见的微服务公共组件

传统单体应用

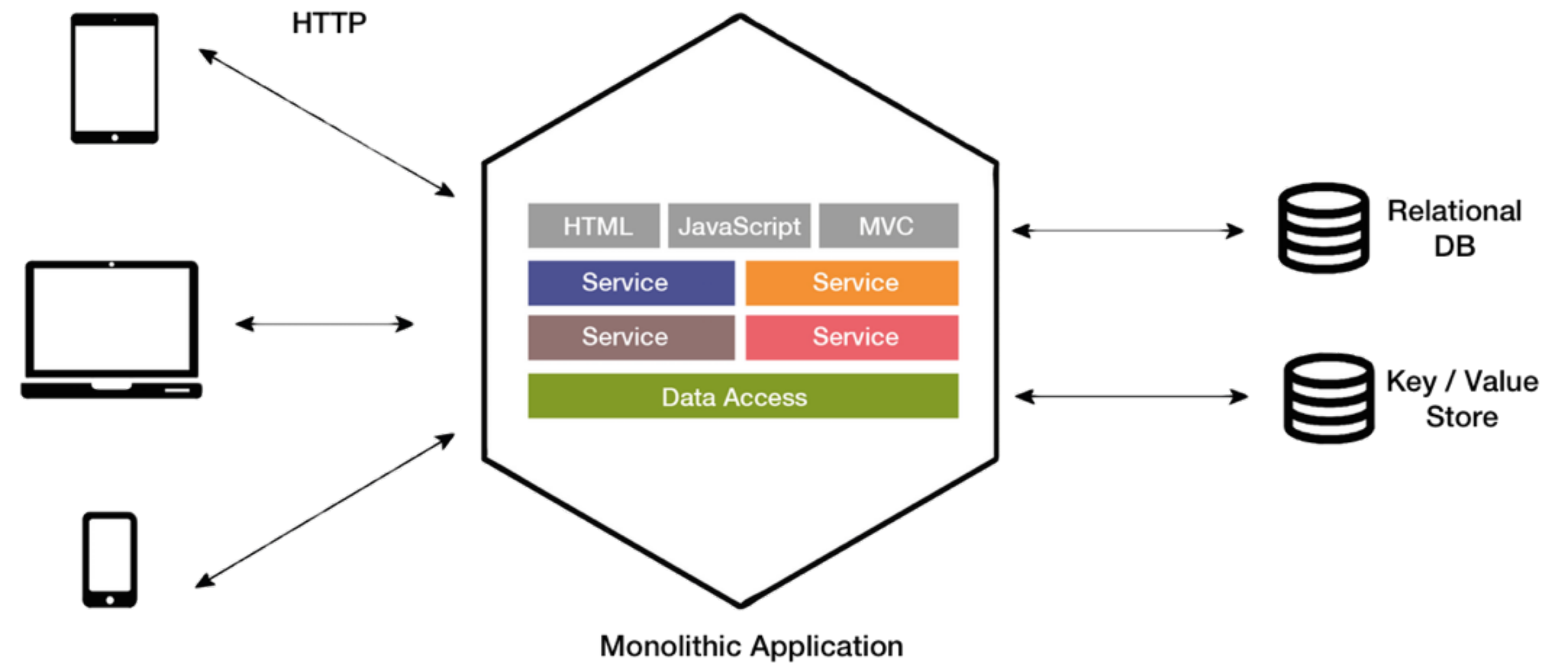
维护成本高

扩展性差

可靠性差

技术选型成本高

交付周期长



微服务是什么

简单来说，微服务架构风格想要开发一种**由多个小服务组成**的应用。每个服务**运行于独立的进程**，并且采用轻量级交互。多数情况下是一个HTTP的资源API。这些服务**具备独立业务能力**并可以通过自动化部署方式**独立部署**。这种风格使**最小化集中管理**，从而可以使用多种不同的编程语言和数据存储技术。

-- James Lewis 和 Martin Fowler

设计原则

高内聚低耦合

单一职责
轻量级通信方式
服务之间的契约

高度自治

能独立的开发，部署，发布
进程隔离
独立的代码库，流水线

以业务为中心

每个服务代表了特定的业务逻辑
能更快的响应业务的变化
围绕业务组织团队

弹性设计

容错
服务降级

日志与监控

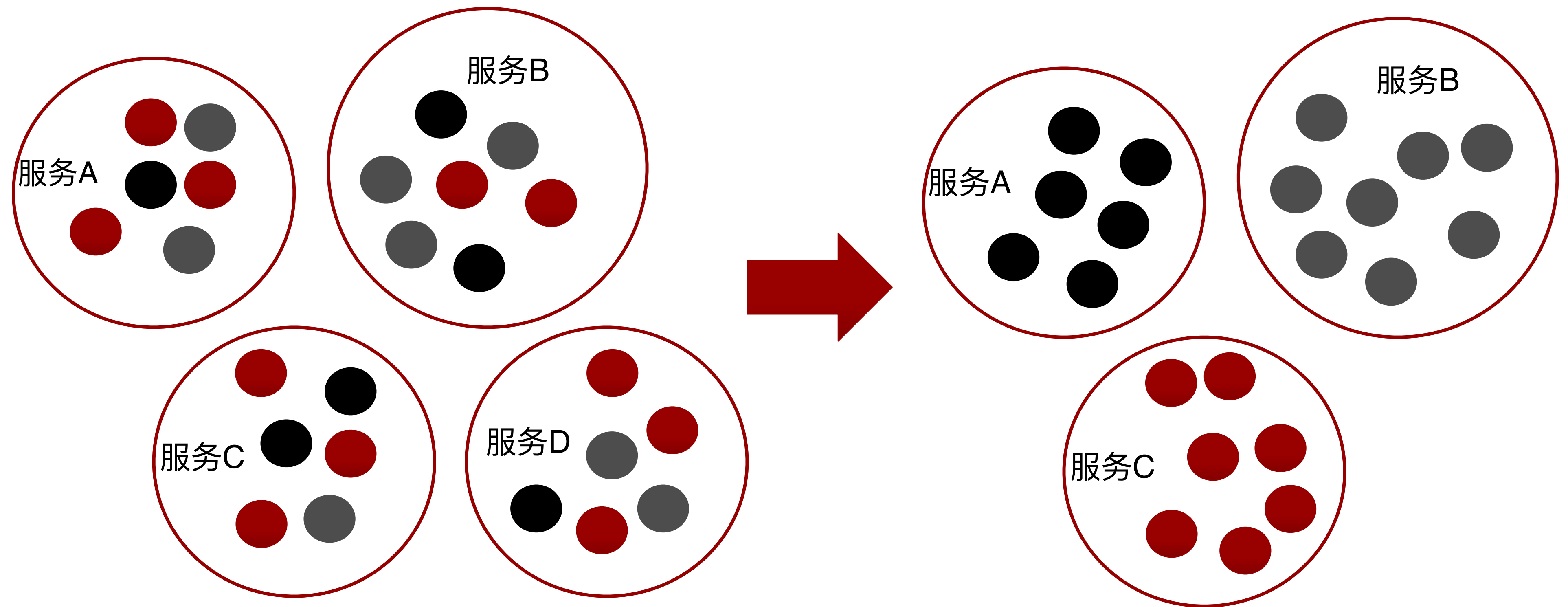
日志聚合
监控与警告

自动化

持续集成
持续交付

高内聚

- 紧密关联的事物应该放在一起
 - Only change for one reason
- Do One Thing, Do It Well



低耦合

- 轻量级的通信方式
 - 同步 RESTful
 - GET
 - PUT
 - POST
 - DELETE
 - 异步
 - 消息队列
 - 发布订阅
- 避免类似RPC或者RMI的通信方式
- 避免在服务与服务之间共享数据库或者库

高度自治

- 独立部署运行和扩展
 - 每个服务能够独立被部署并运行在一个进程内
 - 这种运行和部署方式能够赋予系统灵活的代码组织方式和发布节奏，使得快速交付和应对变化成为可能
- 独立开发和演进
 - 技术选型灵活，不受遗留系统技术栈的约束。
 - 合适的业务问题可以选择合适的技术栈，可以独立的演进
 - 服务与服务之间采取与语言无关的API进行集成
- 独立的团队和自治
 - 团队对服务的整个生命周期负责，工作在独立的上下文中， 谁开发，谁维护。

以业务为中心

- 每个服务代表了特定的业务逻辑
- 有明显的边界上下文
- 围绕业务组织团队
- 能快速的响应业务的变化
- 隔离实现细节，让业务领域可以被重用

弹性设计

- 设计可容错的系统
 - 拥抱失败，为已知的错误而设计
 - 依赖的服务挂掉
 - 网络连接问题
- 设计具有自我保护能力的系统
 - 服务隔离
 - 服务降级
 - 限制使用资源

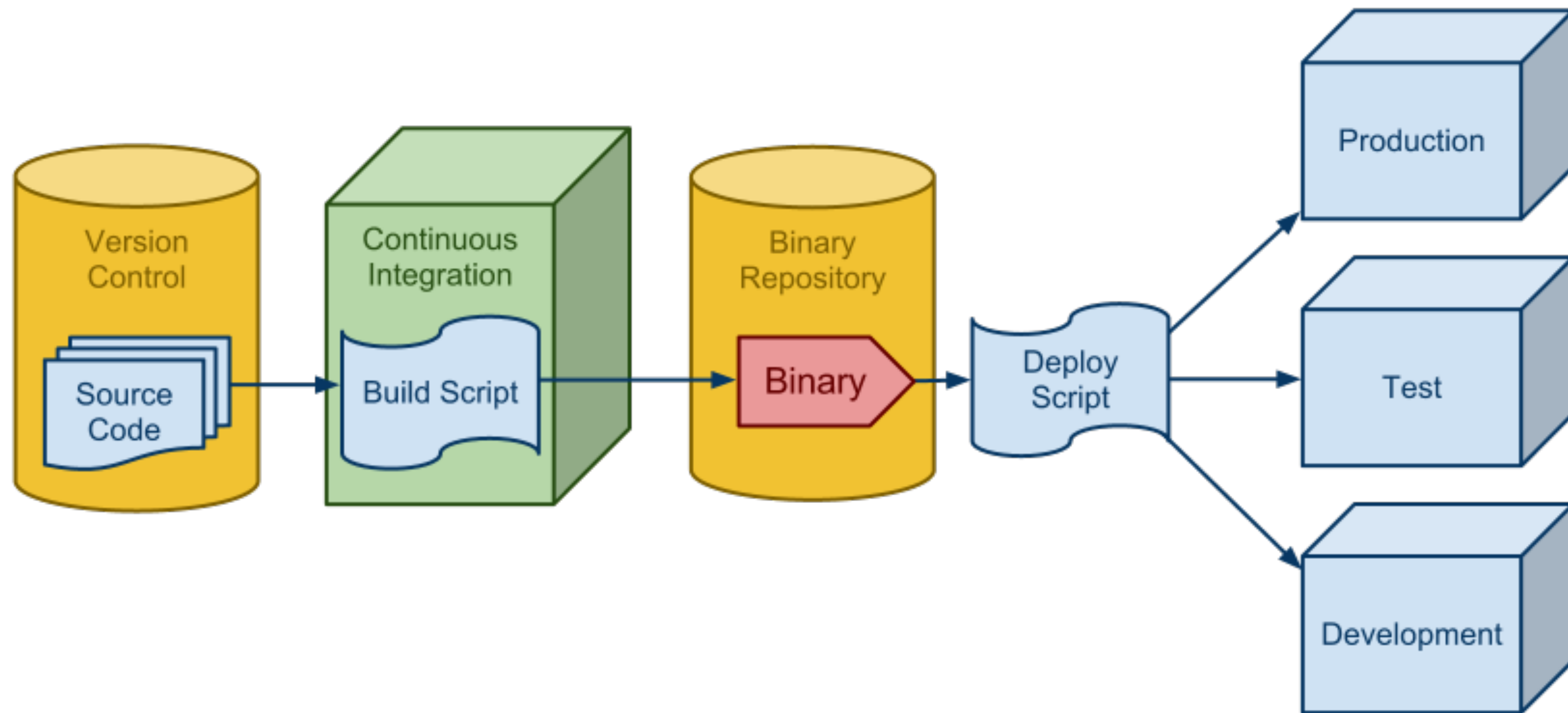
日志与监控

- 日志聚合
- 监控与警告



自动化

- 持续集成
- 持续交付



SPRING CLOUD 介绍

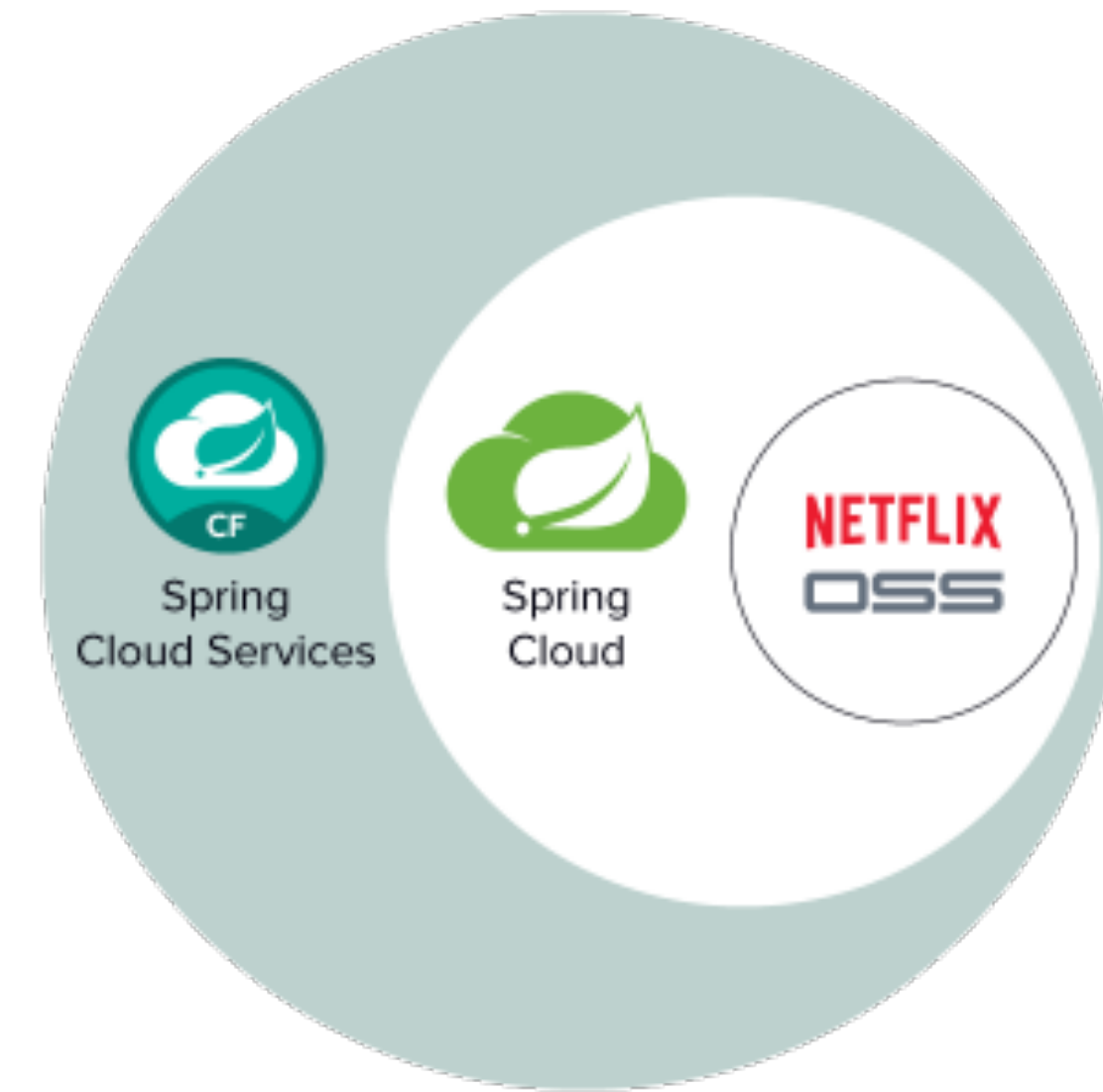
发布于2015年3月

开源

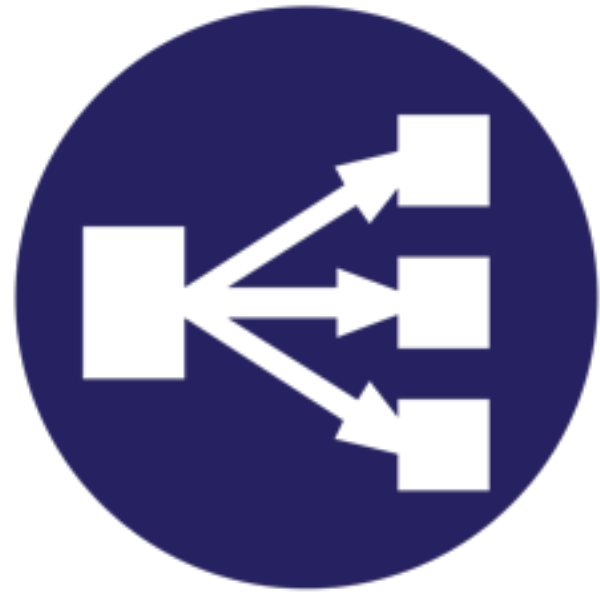
基于SPRING BOOT

构建了分布式系统的公共模式

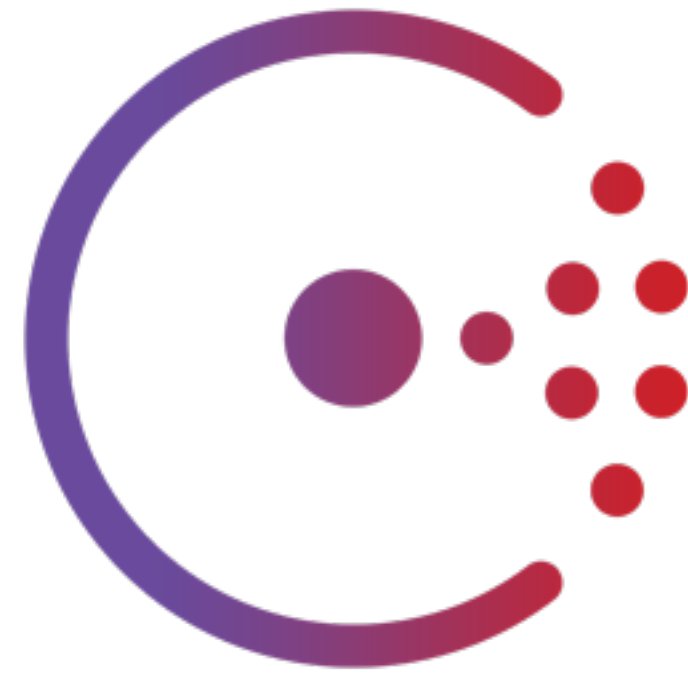
引入了NetflixOSS的开源技术栈



SPRING CLOUD中常见的微服务公共组件



负载均衡



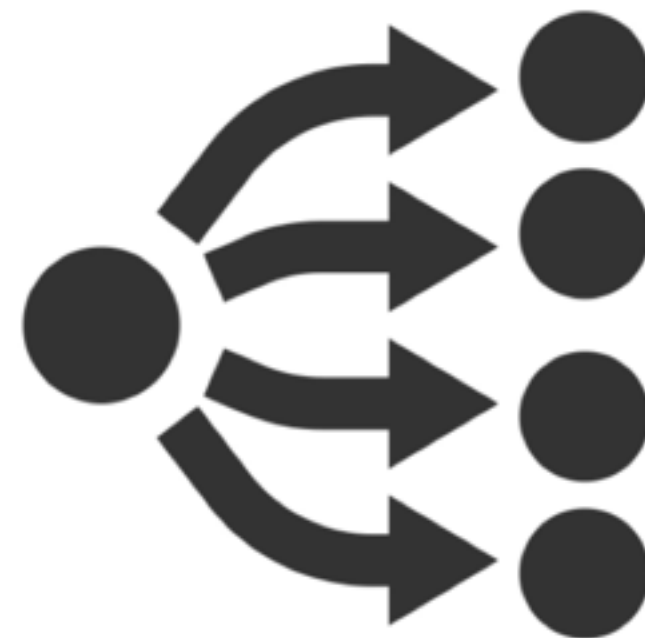
服务注册与发现



监控



分布式配置管理



API网关

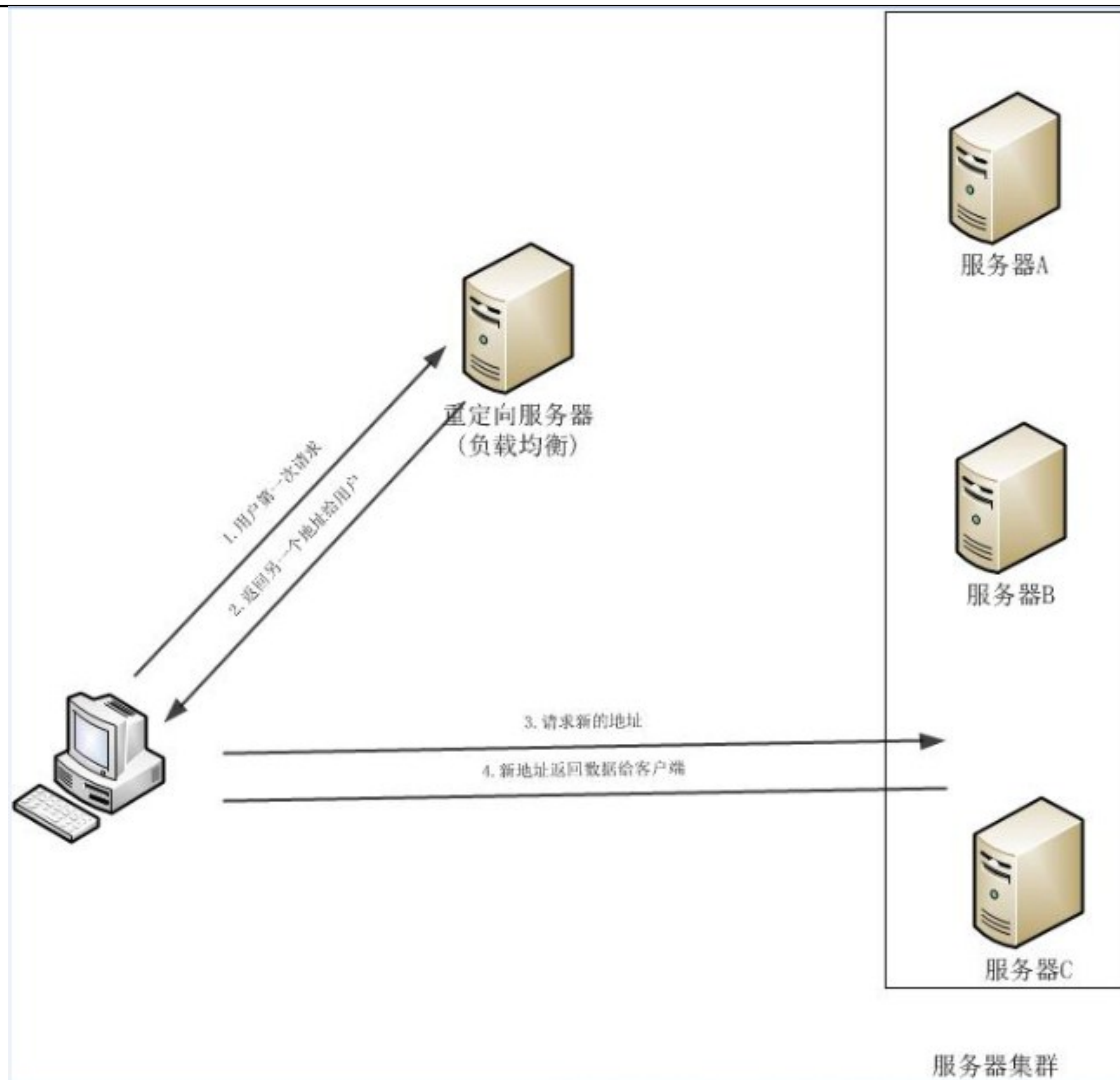


分布式追踪

如何做负载均衡?

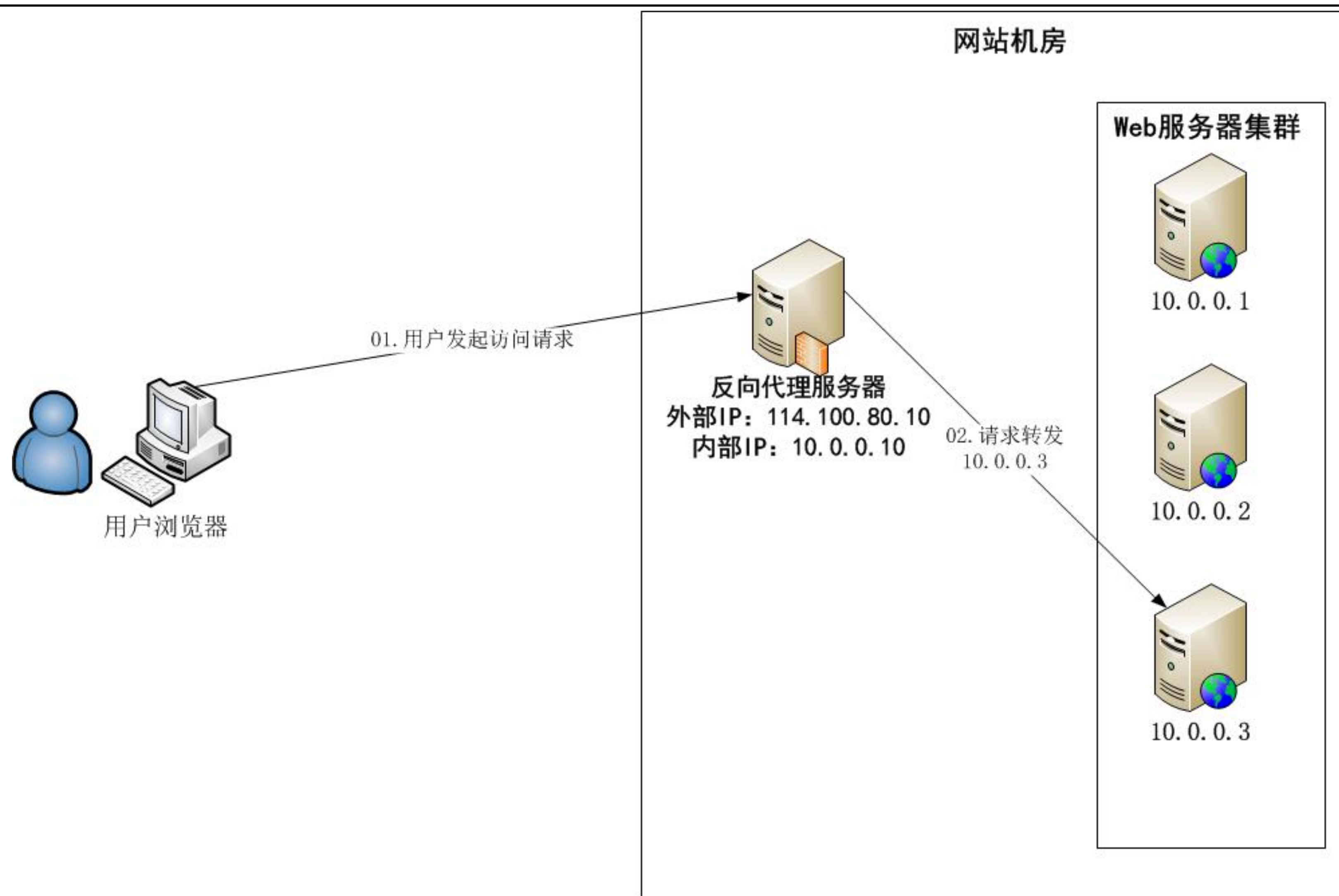
负载均衡

- HTTP重定向负载均衡优点
 - 比较简单
- HTTP重定向负载均衡缺点
 - 浏览器需要两次请求服务器才能完成一次访问
 - 性能较差
 - 重定向服务器自身的处理能力有可能成为瓶颈



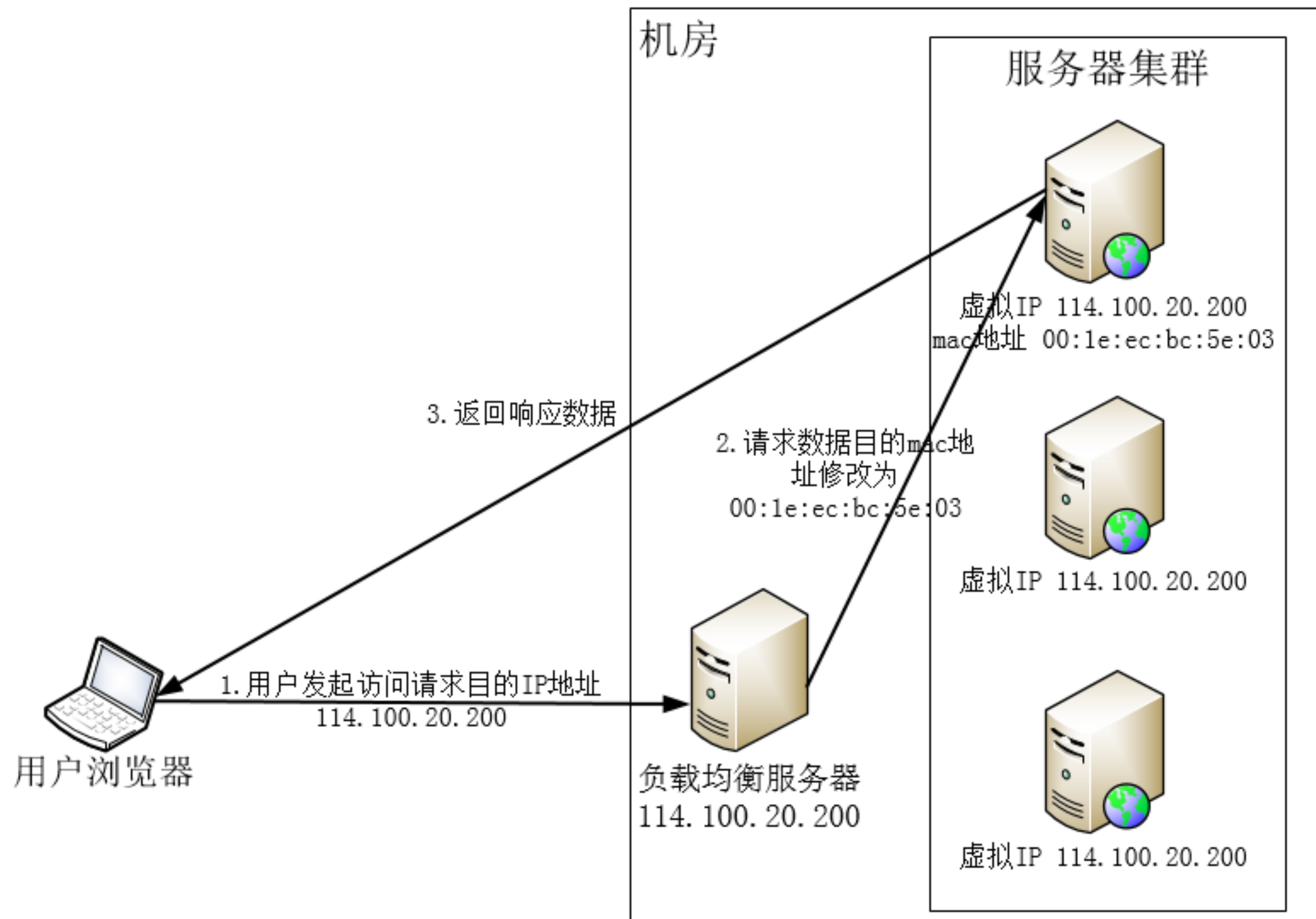
负载均衡

- 反向代理负载均衡优点
 - 比较简单
 - 可以利用反向代理缓存资源
 - 改善网站性能
- 反向代理负载均衡缺点
 - 所有请求和响应的中转站
 - 其性能可能会成为瓶颈



负载均衡

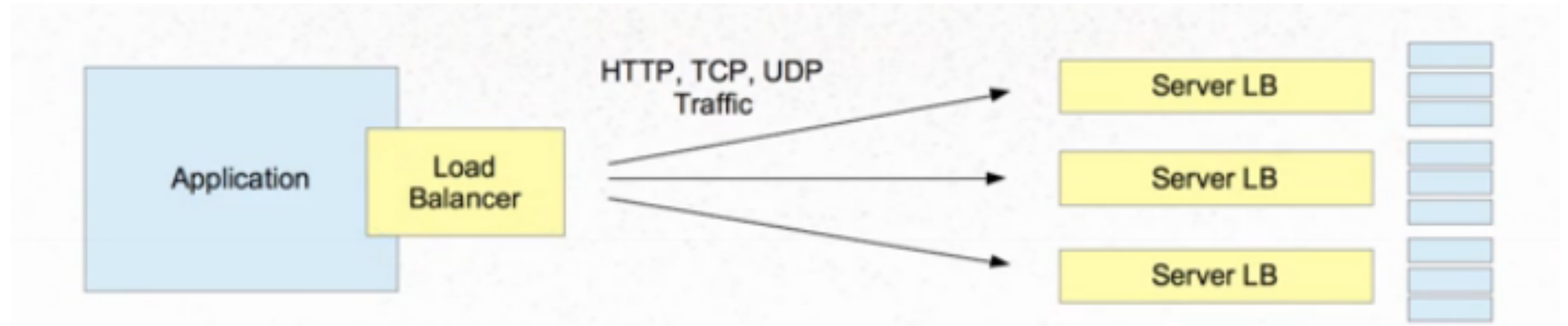
- 数据链路层负载均衡优点
 - 不需要修改数据包的源地址
 - 响应数据不需要通过负载均衡器



负载均衡

- 客户端的负载均衡
 - Netflix Ribbon
- 服务端的负载均衡
 - 硬件F5
 - LVS
 - Nginx
 - HA Proxy
 - ELB

客户端的负载均衡 - NETFLIX RIBBON




```
2017-01-19 02:41:38.425 INFO 7 --- [nio-8080-exec-1] c.n.l.DynamicServerListLoadBalancer :  
DynamicServerListLoadBalancer for client sample-hystrix-aggregate initialized: DynamicServerListLoadBalancer:  
{NFLoadBalancer:name=sample-hystrix-aggregate,current list of Servers=[sampleaggregate2:8081, sampleaggregate1:8081],Load  
balancer stats=Zone stats: {defaultzone=[Zone:defaultzone;Instance count:2;Active connections count: 0;Circuit breaker  
tripped count: 0;Active connections per server: 0.0;]  
,Server stats: [[Server:sampleaggregate2:8081;Zone:defaultZone;Total Requests:0;Successive connection failure:0;Total  
blackout seconds:0;Last connection made:Thu Jan 01 00:00:00 UTC 1970;First connection made: Thu Jan 01 00:00:00 UTC 1970;  
Active Connections:0;total failure count in last (1000) msec:0;average resp time:0.0;90 percentile resp time:0.0;95  
percentile resp time:0.0;min resp time:0.0;max resp time:0.0;stddev resp time:0.0]  
, [Server:sampleaggregate1:8081;Zone:defaultZone;Total Requests:0;Successive connection failure:0;Total blackout  
seconds:0;Last connection made:Thu Jan 01 00:00:00 UTC 1970;First connection made: Thu Jan 01 00:00:00 UTC 1970;Active  
Connections:0;total failure count in last (1000) msec:0;average resp time:0.0;90 percentile resp time:0.0;95 percentile  
resp time:0.0;min resp time:0.0;max resp time:0.0;stddev resp time:0.0]  
]}ServerList:org.springframework.cloud.netflix.ribbon.eureka.DomainExtractingServerList@3e03fb88
```


如何做服务注册？

服务注册与发现

- Netflix Eureka
 - REST接口
 - 支持多节点同步
 - 客户端缓存
 - 自注册模式
- Consul
 - 来自 hashicorp公司
 - REST接口
 - 提供现成的DNS服务器
 - 服务管理器模式



[HOME](#) [LAST 1000 SINCE STARTUP](#)

System Status

Environment	Current time	2017-01-19T07:23:14 +0000
Data center	Uptime	03:04
	Lease expiration enabled	false
	Renews threshold	0
	Renews (last min)	12

DS Replicas

Instances currently registered with Eureka

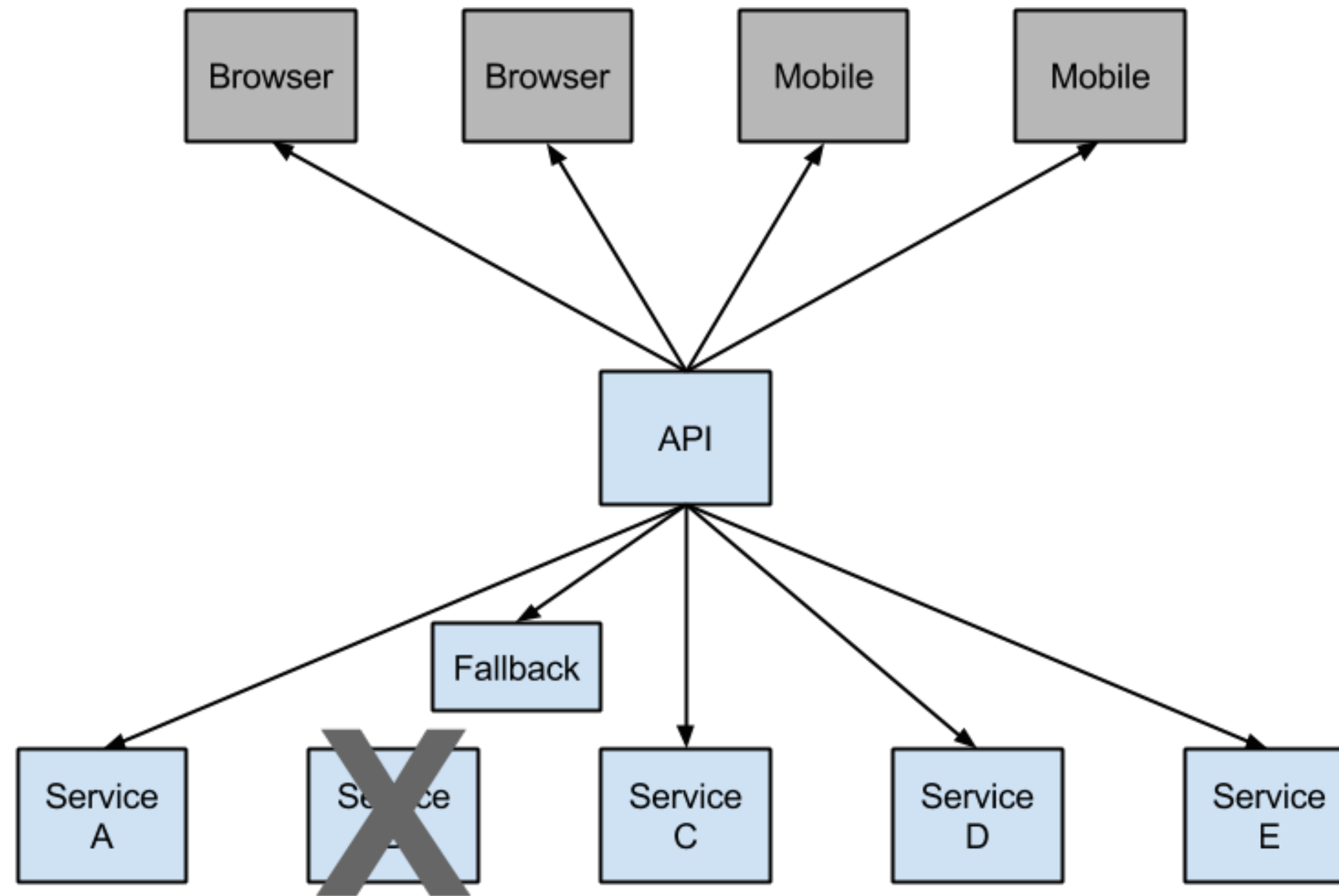
Application	AMIs	Availability Zones	Status
SAMPLE-HYSTRIX-AGGREGATE	n/a (2)	(2)	UP (2) - sampleaggregate1 , sampleaggregate2
SAMPLE-HYSTRIX-CONFIG	n/a (1)	(1)	UP (1) - sampleconfig
SAMPLE-HYSTRIX-GATEWAY	n/a (1)	(1)	UP (1) - samplegateway
SAMPLE-HYSTRIX-MONITOR	n/a (1)	(1)	UP (1) - samplemonitor
SAMPLE-HYSTRIX-SERVICE	n/a (1)	(1)	UP (1) - sample-service

General Info

Name	Value
total-avail-memory	74mb

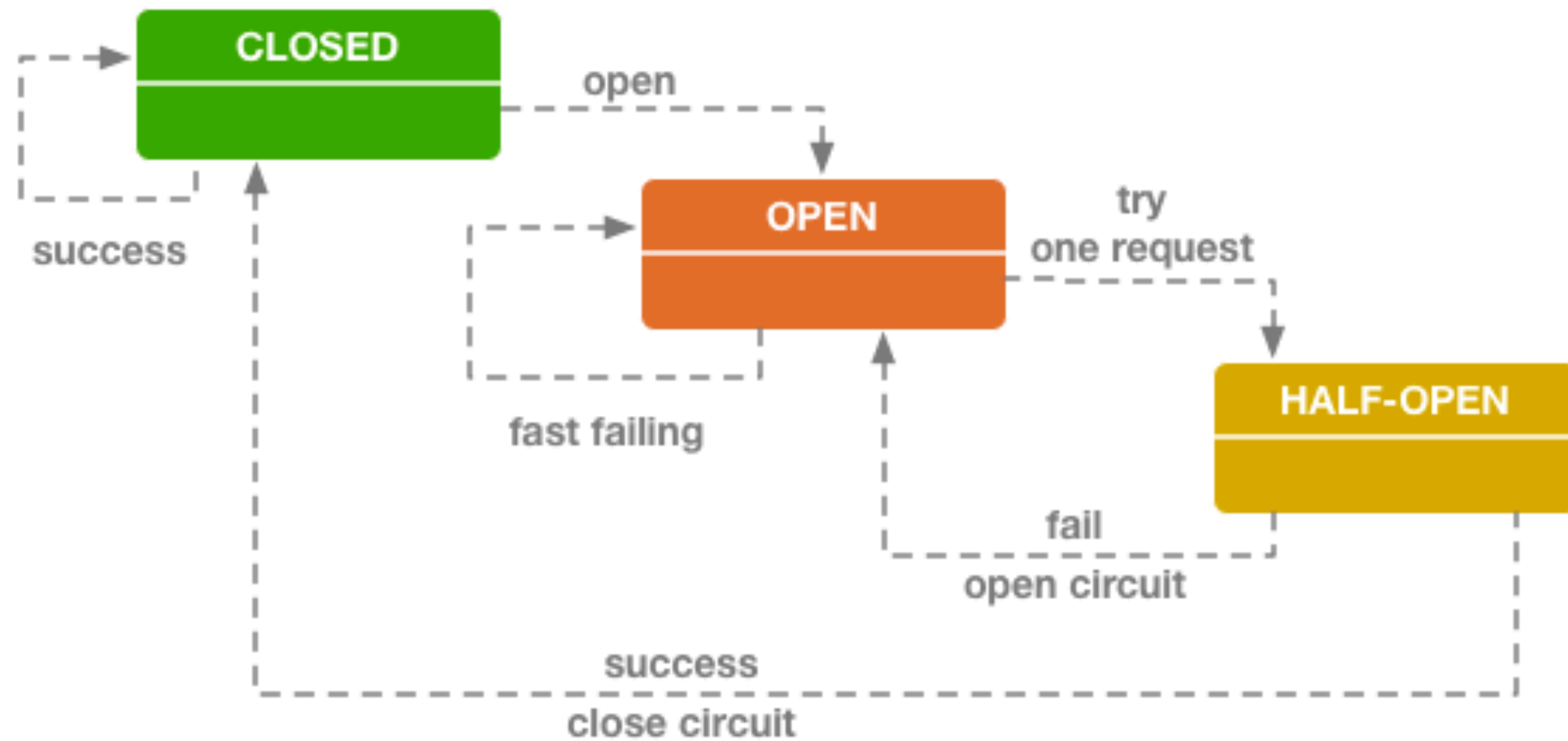
如何做容错?

- 熔断器模式
 - 防止级联错误
 - 问题服务隔离
- Dashboard



熔断器

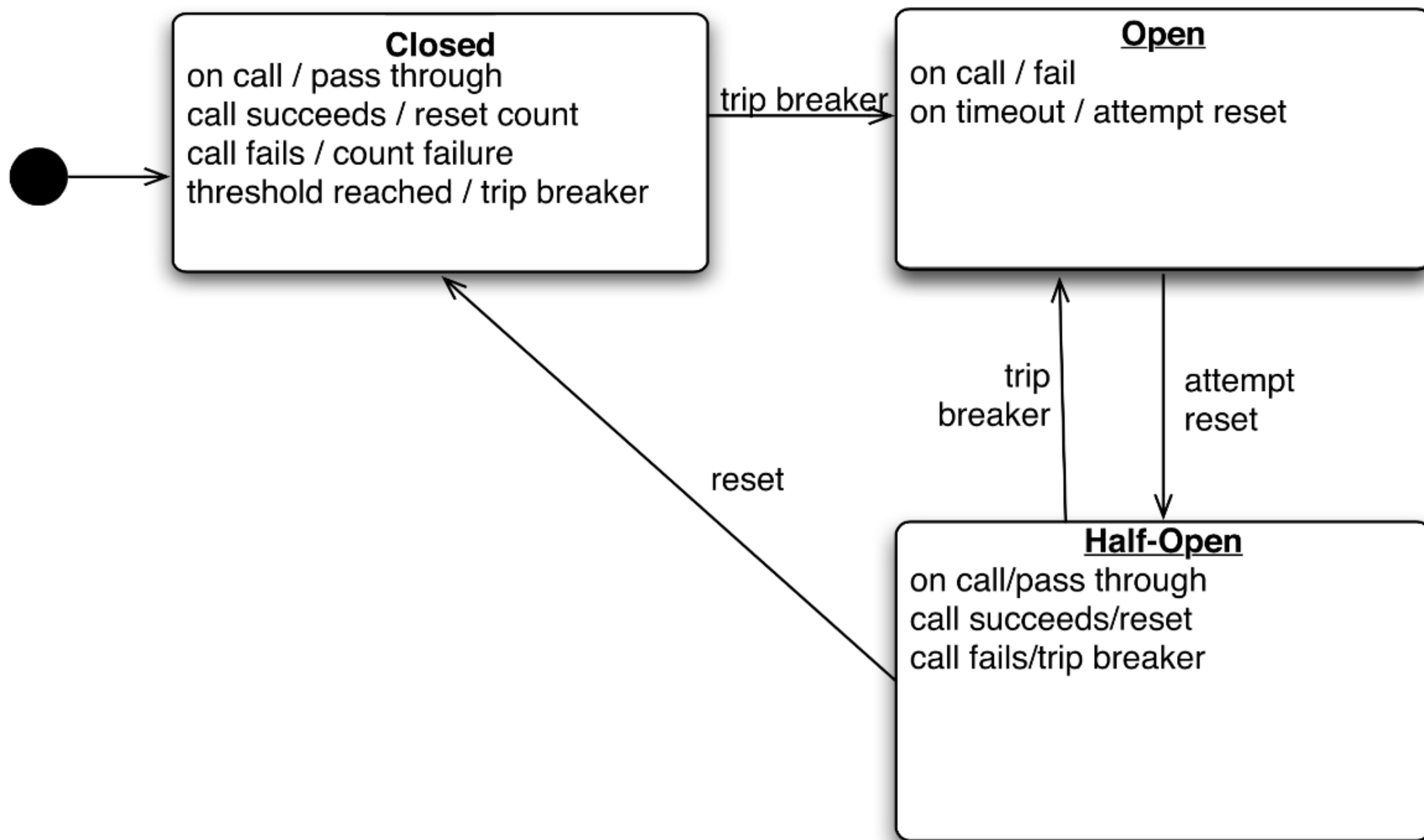
- 三种状态



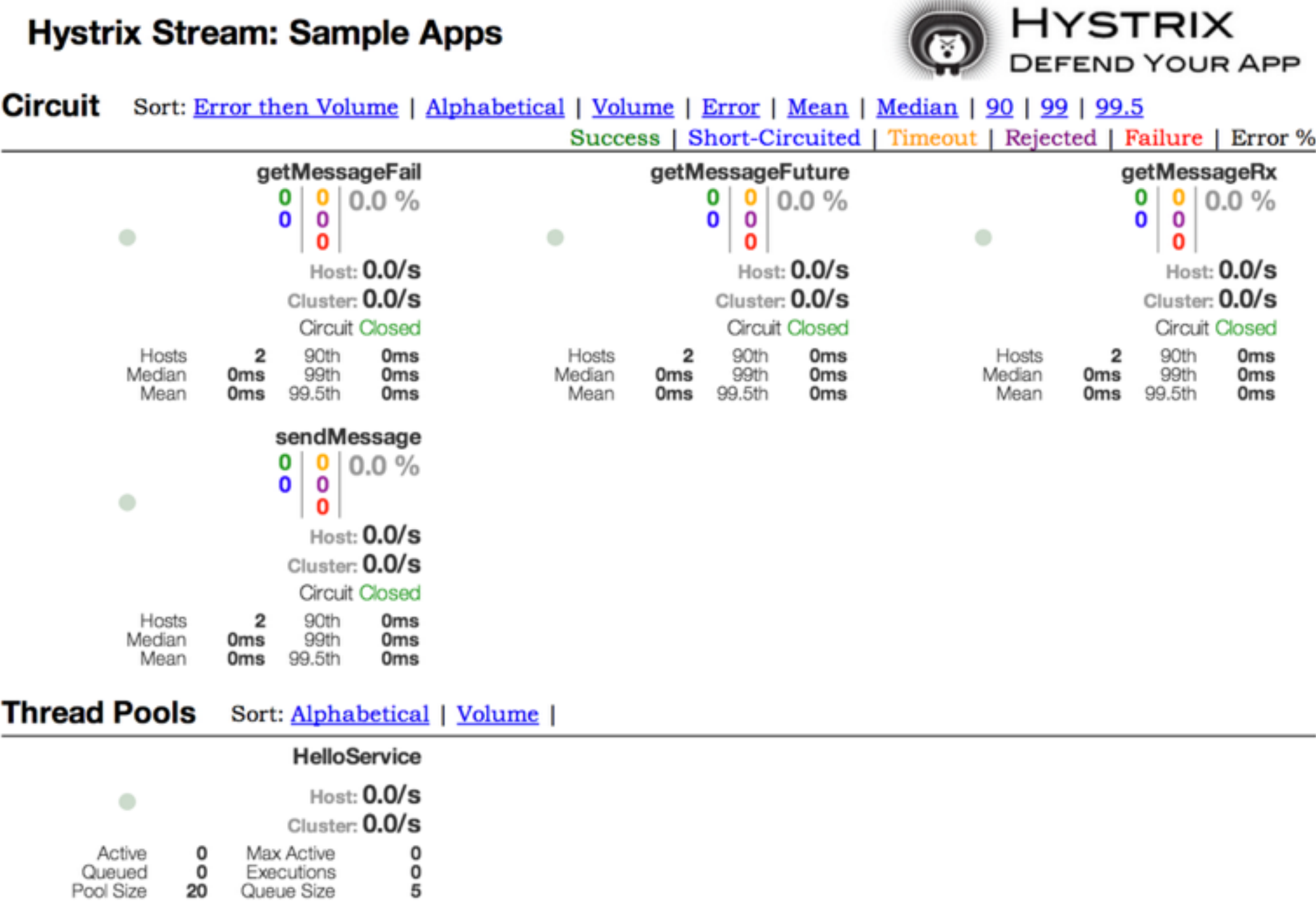
Circuit Breaker State Diagram

熔断器

- 三种状态




```
@Component
public class Processing {
    @HystrixCommand(fallbackMethod = "default")
    public Object doIt(Map<String, Object> args) {
        // do stuff that might fail...
    }
    public Object default(Map<String, Object>args) {
        return someUsefulDefaults;
    }
}
```



如何做配置管理？

SPRING CLOUD CONFIG SERVER

- 默认使用GIT
- 在版本控制之下
- 支持PROPERTY和YAML
- 中心化的动态配置
 - 当配置改变时，一些BEANS会被重新初始化
 - 无须重启，但需要调用/POST /refresh触发



CONFIG SERVER: RESTFUL API

`/ {application} / {profile} [/ {label}]`

`/ {application} - {profile} . yml`

`/ {label} / {application} - {profile} . yml`

`/ {application} - {profile} . properties`

`/ {label} / {application} - {profile} . properties`

CONFIG SERVER: ENDPOINTS

https://github.com/wa-tolls/rates

<branch: master>

- application.properties
- station1
 - s1rates-dev.properties
 - s1rates-qa.properties
 - s1rates.properties
- station2
 - s2rates-dev.properties
 - s2rates.properties

/ {application} / {profile} [/ {label}]

-required-

-required-

-optional-

CONFIG SERVER: ENDPOINTS

`https://github.com/wa-tolls/rates`
<branch: master>

- **application.properties**
- station1
 - s1rates-dev.properties
 - s1rates-qa.properties
 - **s1rates.properties**
- station2
 - s2rates-dev.properties
 - s2rates.properties

`/ {application} / {profile} [/ {label}]`
-required- -required- -optional-

/s1rates/default

CONFIG SERVER: ENDPOINTS

`https://github.com/wa-tolls/rates`
<branch: master>

- **application.properties**
- station1
 - **s1rates-dev.properties**
 - s1rates-qa.properties
 - **s1rates.properties**
- station2
 - s2rates-dev.properties
 - s2rates.properties

`/ {application} / {profile} [/ {label}]`
-required- -required- -optional-

/s1rates/dev

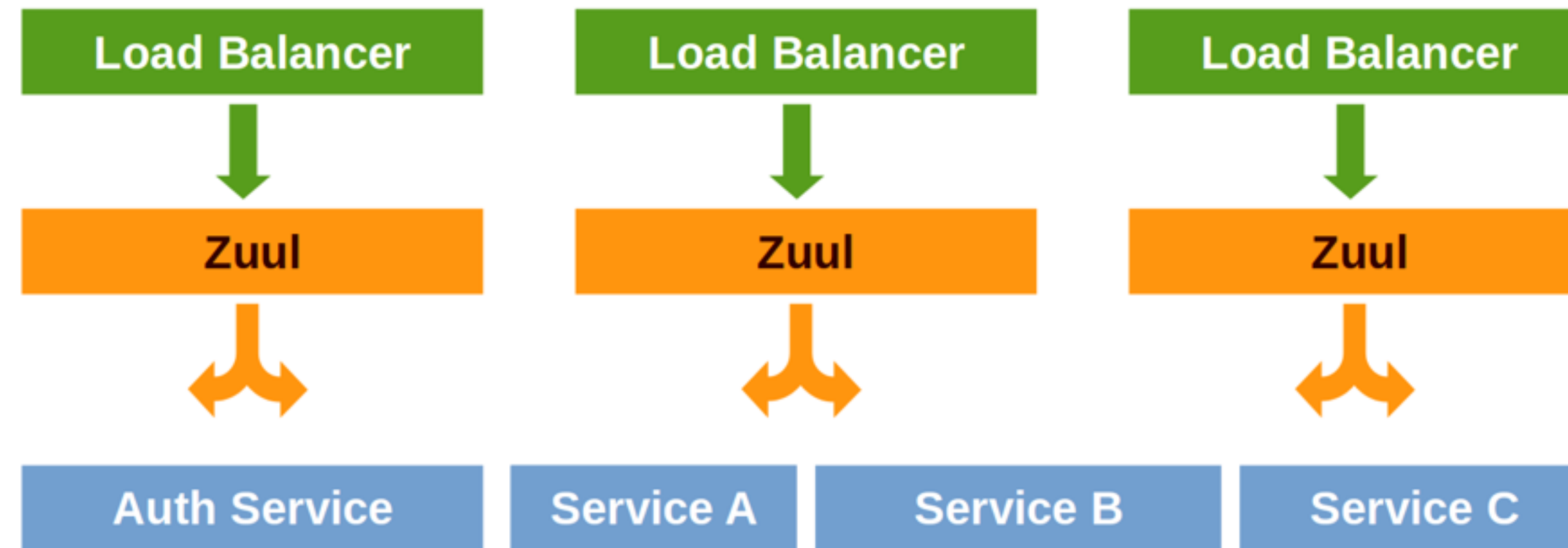
为什么要引入API网关?

API网关

- 客户端与服务端直接通信又有那些弊端？
 - 客户端为了获取某个数据集，可能需要多次请求。
 - PC端和移动端需要的数据量一般不一样，移动端的屏幕更小，需要的数据量更少，我们需要一层来做数据的组装和过滤
 - UI端和微服务直接耦合在一起。
- API网关的好处？
 - 减少API请求次数
 - 限流量
 - 缓存
 - 认证
 - 监控
 - 对外提供统一的接口，使背后的微服务对UI端来说是透明的；对内它方便了以后微服务的重构甚至重写

看门人-ZUUL

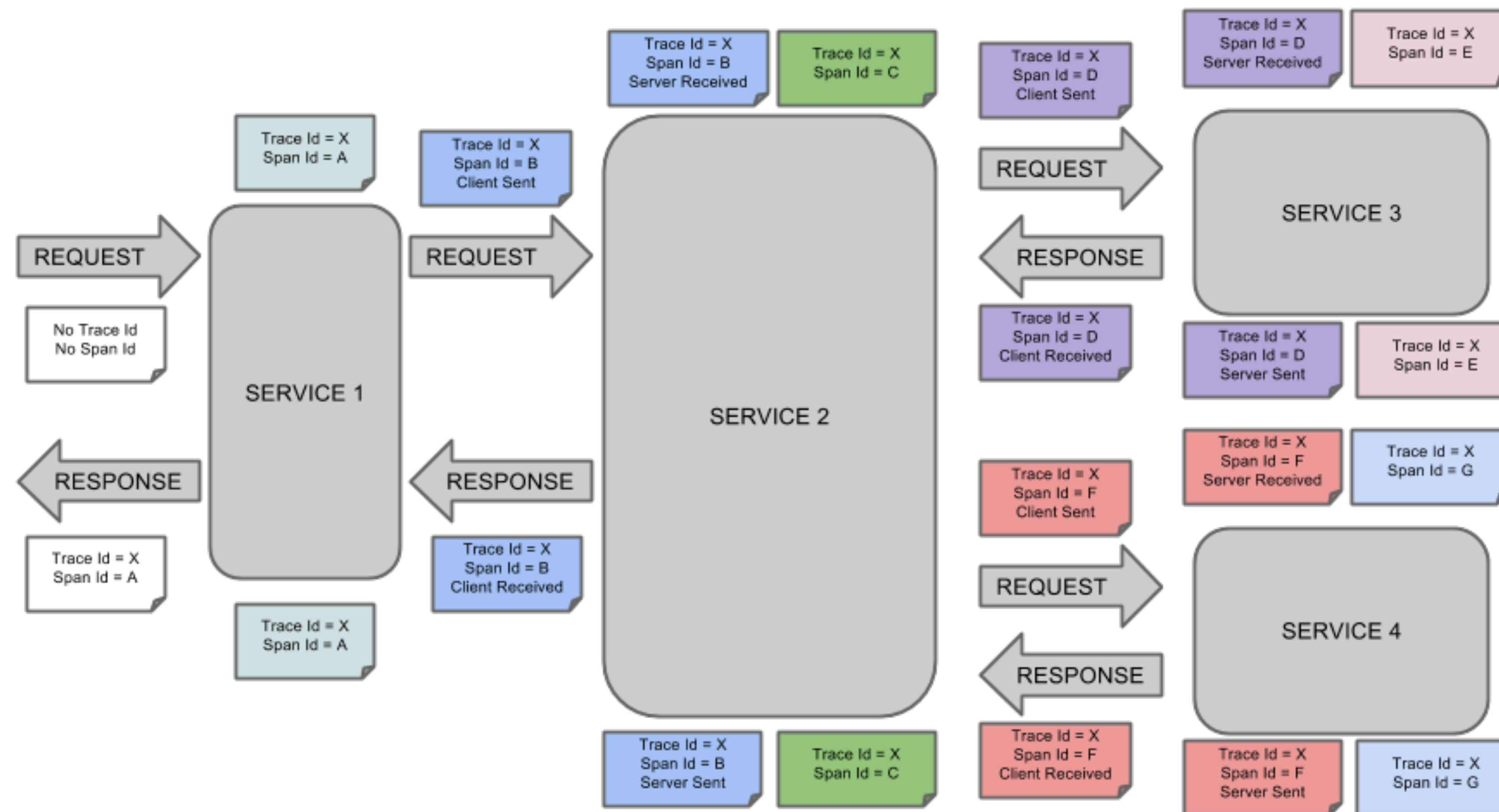
- 它负责请求转发
- 聚合返回的数据集
- 它可以与Eureka配合使用
 - 不需要特别的配置



API调用链追踪?

SPRING CLOUD SLEUTH

- 分布式系统中如何追踪一个Request
- 如何进行性能监控



SPRING CLOUD SLEUTH

- 以下的操作都会被自动追踪

**Runnable /
Callable
operations**

**Spring Cloud
Hystrix, Zuul**

RxJava

**Synchronous /
Asynchronous
RestTemplate**

Spring Integration

**@Async,
@Scheduled
operations**

微服务架构案例演示

Summary



- 传统单体应用
- 微服务是什么
- 微服务设计原则
- Spring Cloud介绍
- 负载均衡
- 服务注册与发现
- 容错机制
- 分布式配置管理
- API网关
- 分布式追踪
- 案例演示





THANK
YOU!