

# 第七章

---

## Spring Cloud

—— **Coordinate Anything**

# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# Spring Boot中的RestTemplate

---

- Spring Boot 中没有自动配置 RestTemplate
- Spring Boot 提供了 RestTemplateBuilder
  - **RestTemplateBuilder.build()**

# 常用方法

---

## GET 请求

- `getForObject()` / `getForEntity()`

## POST 请求

- `postForObject()` / `postForEntity()`

## PUT 请求


- `put()`

## DELETE 请求

- `delete()`

```
String result = restTemplate.getForObject(  
    "http://example.com/hotels/{hotel}/bookings/{booking}", String.class, "42", "21");
```

```
URI uri = UriComponentsBuilder
    .fromUriString("http://example.com/hotels/{hotel}")
    .QueryParam("q", "{q}")
    .encode()
    .buildAndExpand("Westin", "123")
    .toUri();
```



```
URI uri = UriComponentsBuilder
    .fromUriString("http://example.com/hotels/{hotel}?q={q}")
    .build("Westin", "123");
```

# 内容提要

- 访问Web资源
- **Cloud Native概述**
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# Cloud Native定义

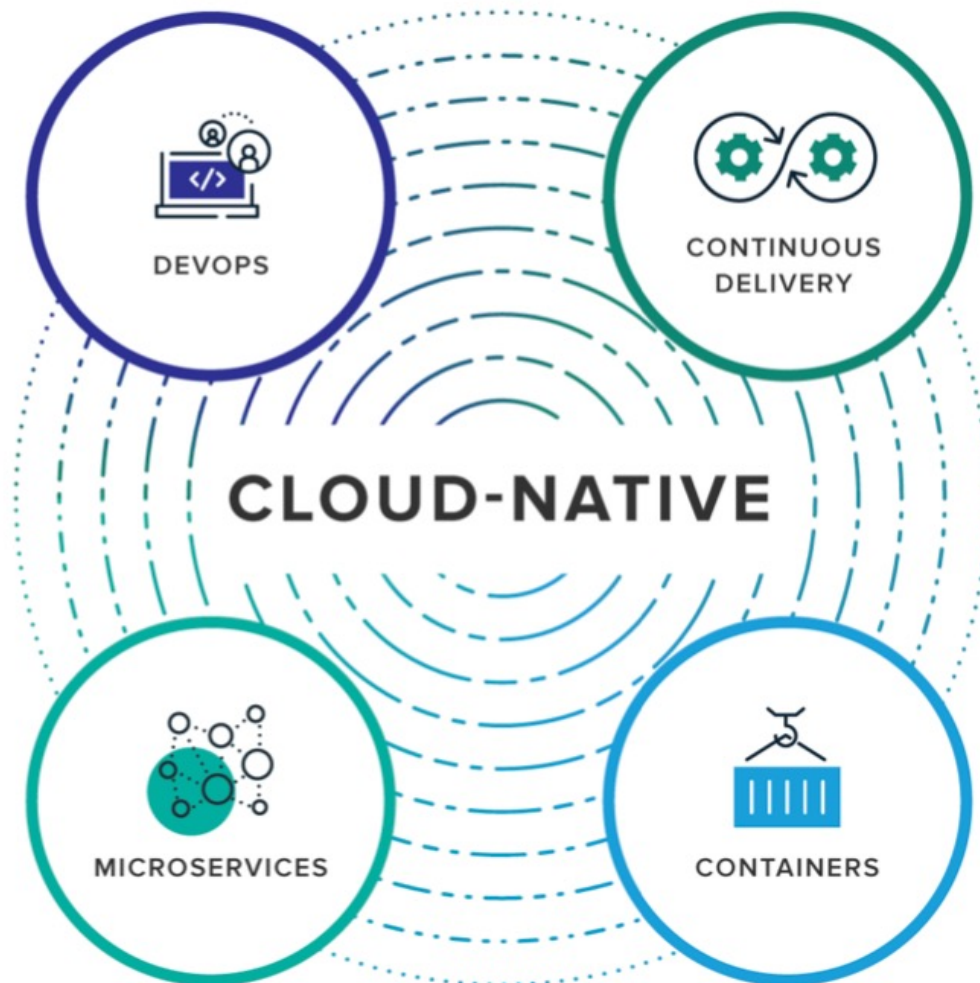
---

“云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。”

– *CNCF Cloud Native Definition v1.0*

# 云原生应用要求.....

---





# 云原生应用要求.....

---

## DevOps

- 开发与运维一同致力于交付高品质的软件服务于客户

## 持续交付

- 软件的构建、测试和发布，要更快、更频繁、更稳定

## 微服务

- 以一组小型服务的形式来部署应用

## 容器

- 提供比传统虚拟机更高的效率

# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# 微服务定义

## 微服务 (SOA架构的一种变体)

★ 收藏

👍 74

🔗 41

- 🔊 播报
- ✎ 编辑
- 💬 讨论 <sup>4</sup>
- 📺 上传视频

什么是微服务？

维基上对其定义为：一种软件开发技术- 面向服务的体系结构（SOA）架构样式的一种变体，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够独立地部署到生产环境、类生产环境等。另外，应尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据上下文，选择合适的语言、工具对其进行构建。

中文名	微服务	所属学科	软件架构
外文名	microservice	目 的	有效的拆分应用，实现敏捷开发和部署

目录	<div>1 简介</div> <div>2 受益方法<ul style="list-style-type: none"><li>▪ 可独立部署</li><li>▪ 正确的工作工具</li><li>▪ 精确缩放</li></ul></div>	<div>3 关键支持技术和工具<ul style="list-style-type: none"><li>▪ 容器，Docker和Kubernetes</li><li>▪ API网关</li></ul></div> <div>4 常见模式</div> <div>5 反模式</div>
----	---	---

# 微服务的优点

---

## 异构性

- 语言、存储.....

## 弹性

- 一个组件不可用，不会导致级联故障

## 扩展

- 单体服务不易扩展，多个较小的服务可以按需扩展

# 微服务的优点

---

- 易于部署
- 与组织结构对齐
- 可组合性
- 可替代性

# 实施微服务的代价

---

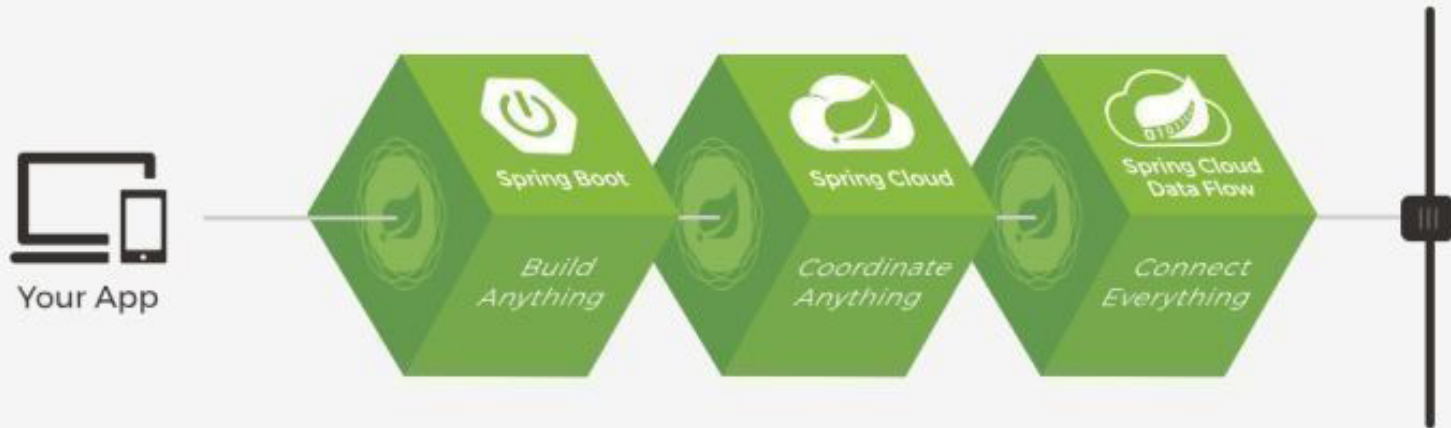
- 分布式系统的复杂性
- 开发、测试等诸多研发过程中的复杂性
- 部署、监控等诸多运维复杂性
- .....

# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# Spring系列

Spring: the source for modern java



## Spring Boot

### BUILD ANYTHING

Spring Boot is designed to get you up and running as quickly as possible, with minimal upfront configuration of Spring. Spring Boot takes an opinionated view of building production-ready applications.

## Spring Cloud

### COORDINATE ANYTHING

Built directly on Spring Boot's innovative approach to enterprise Java, Spring Cloud simplifies distributed, microservice-style architecture by implementing proven patterns to bring resilience, reliability, and coordination to your microservices.

## Spring Cloud Data Flow

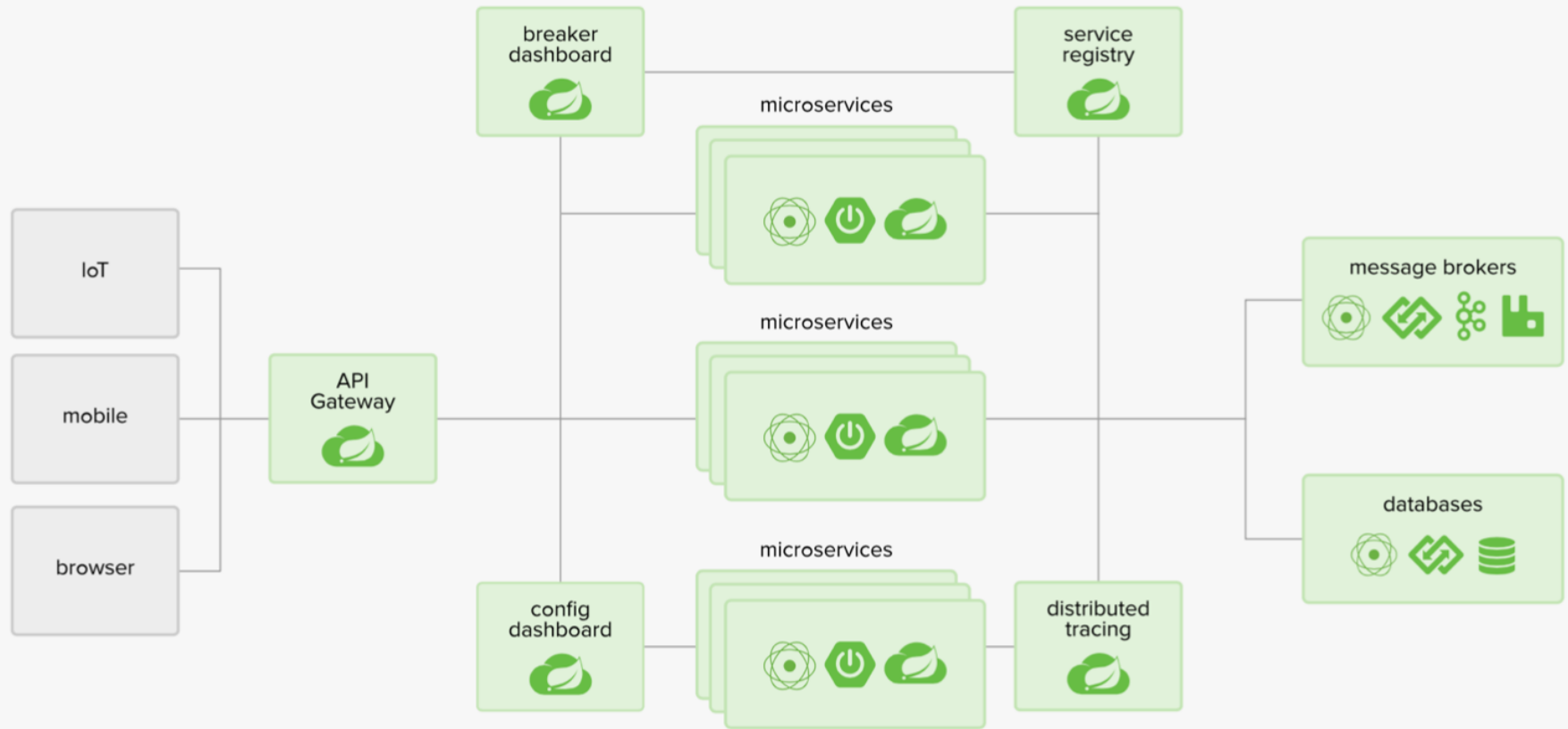
### CONNECT ANYTHING

Connect the Enterprise to the Internet of Anything—mobile devices, sensors, wearables, automobiles, and more.

Spring Cloud Data Flow provides a unified service for creating composable data microservices that address streaming and ETL based data processing patterns.



# Spring Cloud组成



# Spring Cloud 的主要功能

---

- 服务发现
- 服务熔断
- 配置服务
- 服务安全
- 服务网关
- 分布式消息
- 分布式跟踪
- 各种云平台支持

# Spring Cloud 的版本号规则

---

- Spring Cloud 是个大工程，其中包含多个独立项目
- BOM - Release Train
  - London Tube Stations
    - 字母序排列
      - Greenwich, Finchley, Edgware ...
- SR - Service Release

# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# 使用 Eureka 作为服务注册中心

---

- 什么是 Eureka
  - **Eureka** 是在 **AWS** 上定位服务的 **REST** 服务
- Netflix OSS
  - <https://netflix.github.io>
- Spring 对 Netflix 套件的支持
  - **Spring Cloud Netflix**

# 在本地启动一个简单的 Eureka 服务

---

## Starter

eureka-server

- spring-cloud-dependencies
- spring-cloud-starter-netflix-eureka-starter

## 声明

- @EnableEurekaServer

## 注意事项

- 默认端口8761
- Eureka 自己不要注册到 Eureka 了

# 将服务注册到 Eureka Server

---

## Starter

- `spring-cloud-starter-netflix-eureka-client`

## 声明

- `@EnableDiscoveryClient`
- `@EnableEurekaClient`

## 一些配置项

- `eureka.client.service-url.default-zone`
- `eureka.client.instance.prefer-ip-address`

## Bootstrap 属性

- 启动引导阶段加载的属性
- `bootstrap.properties` | `.yml`
- `spring.cloud.bootstrap.name=bootstrap`

## 常用配置

- `spring.application.name=应用名`
- 配置中心相关



# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# 如何获得服务地址

---

## **EurekaClient**

- getNextServerFromEureka()

## **DiscoveryClient**

- getInstances()

## RestTemplate 与 WebClient

- @LoadBalanced
- 实际是通过 ClientHttpRequestInterceptor 实现的
  - LoadBalancerInterceptor
  - LoadBalancerClient
    - RibbonLoadBalancerClient

# 使用 Feign 访问服务

---

## Feign

- 声明式 REST Web 服务客户端
- <https://github.com/OpenFeign/feign>

## Spring Cloud OpenFeign

- spring-cloud-starter-openfeign

# Feign 的简单使用

---

## 开启 Feign 支持

- `@EnableFeignClients`

## 定义 Feign 接口

- `@FeignClient`

## 简单配置

- `FeignClientsConfiguration`
- `Encoder / Decoder / Logger / Contract / Client ...`

# 通过配置定制 Feign

feign-customer-service

```
feign:
  client:
    config:
      feignName:
        connectTimeout: 5000
        readTimeout: 5000
        loggerLevel: full
        errorDecoder: com.example.SimpleErrorDecoder
        retryer: com.example.SimpleRetryer
        requestInterceptors:
          - com.example.FooRequestInterceptor
          - com.example.BarRequestInterceptor
        decode404: false
        encoder: com.example.SimpleEncoder
        decoder: com.example.SimpleDecoder
        contract: com.example.SimpleContract
```

# 使用 Consul 作为服务注册中心

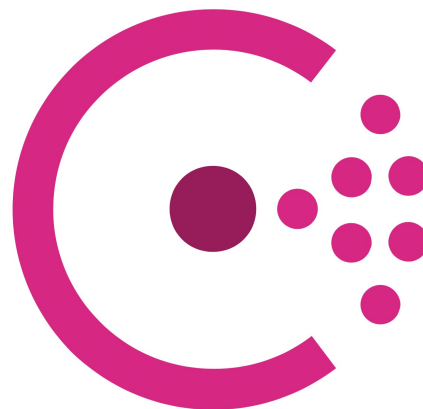
---

- Consul

- <https://www.consul.io>

- 关键特性

- 服务发现
  - 健康检查
  - **KV** 存储
  - 多数据中心支持
  - 安全的服务间通信



# 使用 Consul 提供服务发现能力

---

## Consul 的能力

- Service registry, integrated health checks, and DNS and HTTP interfaces enable any service to discover and be discovered by other services

## 好用的功能

- HTTP API
- DNS ( xxx.service.consul )
- 与 Nginx 联动, 比如 ngx\_http\_consul\_backend\_module



# 如何发现和访问Consul

---

## Spring Cloud Consul

- `spring-cloud-starter-consul-discovery`

### 简单配置

- `spring.cloud.consul.host=localhost`
- `spring.cloud.consul.port=8500`
- `spring.cloud.consul.discovery.prefer-ip-address=true`

# 通过 Docker 启动 Consul

---

- 官方指引

- [https://hub.docker.com/\\_/consul](https://hub.docker.com/_/consul)

- 获取镜像

- **docker pull consul**

- 运行 Consul 镜像

- **docker run --name consul -d -p 8500:8500 -p 8600:8600/udp consul**

- 运行 docker-compose

- **consul.yml**

consul-customer-service

- **docker-compose up**

consul-waiter-service

# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# 断路器模式

---

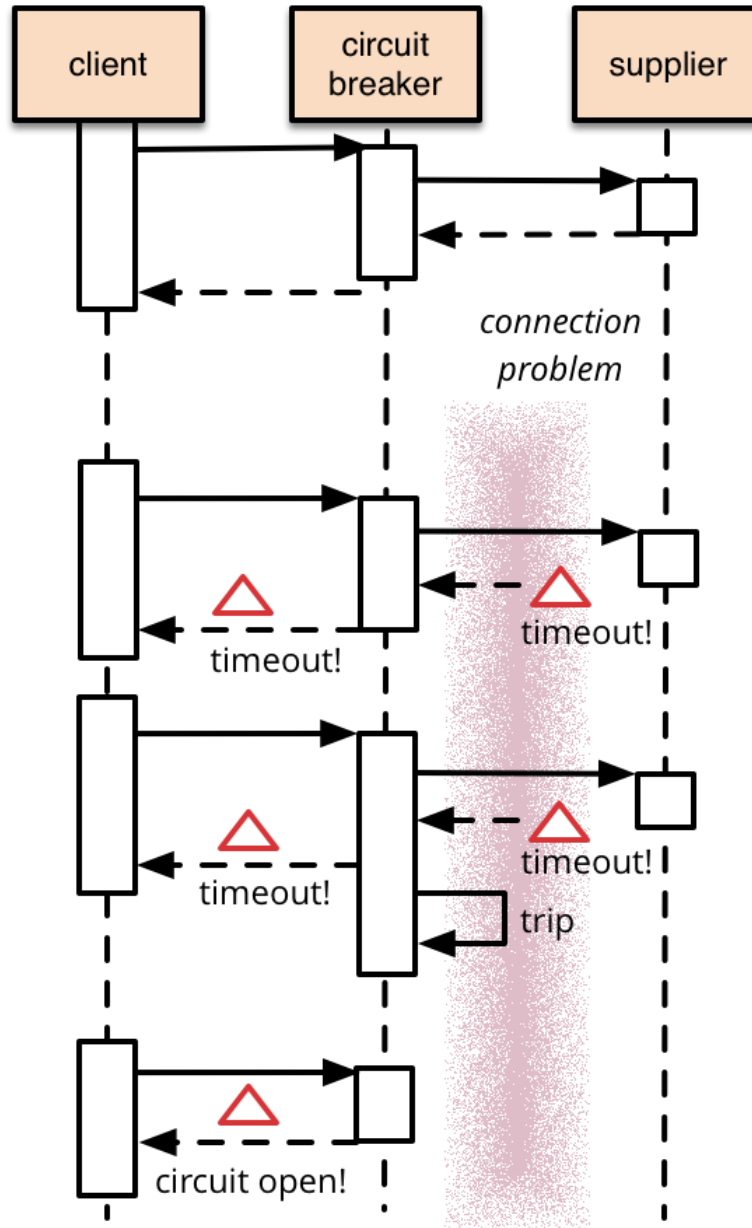
- 断路器

- **Circuit Breaker pattern - Release It, Michael Nygard**
- **CircuitBreaker, Martin Fowler**
  - <https://martinfowler.com/bliki/CircuitBreaker.html>

- 核心思想

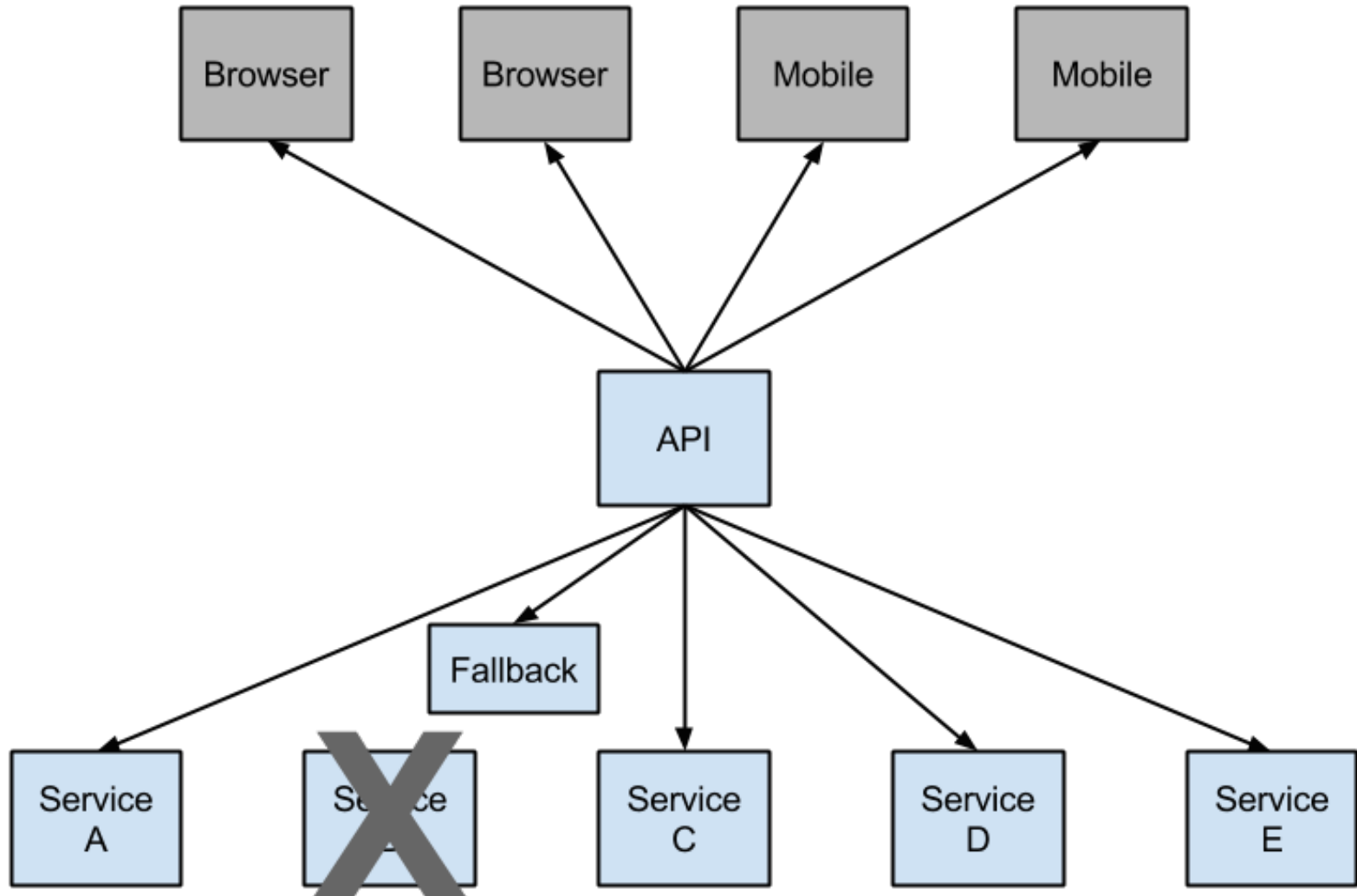
- 在断路器对象中封装受保护的方法调用
- 该对象监控调用和断路情况
- 调用失败触发阈值后，后续调用直接由断路器返回错误，不再执行实际调用

# 断路器示意图



# Netflix Hystrix

---



# Netflix Hystrix

---

- 实现了断路器模式
- @HystrixCommand
  - fallbackMethod / commandProperties
    - @HystrixProperty(name="execution.isolation.strategy", value="SEMAPHORE")
- <https://github.com/Netflix/Hystrix/wiki/Configuration>

### Spring Cloud 支持

- `spring-cloud-starter-netflix-hystrix`
- `@EnableCircuitBreaker`

### Feign 支持

- `feign.hystrix.enabled=true`
- `@FeignClient`
  - `fallback` / `fallbackFactory`



# 如何了解熔断的情况

---

## 打日志

- 在发生熔断时打印特定该日志

## 看监控

- 主动向监控系统埋点，上报熔断情况
- 提供与熔断相关的 Endpoint，让第三方系统来拉取信息

### Spring Cloud 为我们提供了

- Hystrix Metrics Stream
  - spring-boot-starter-actuator
    - /actuator/hystrix.stream
- Hystrix Dashboard
  - spring-cloud-starter-netflix-hystrix-dashboard
    - @EnableHystrixDashboard
    - /hystrix

## Spring Cloud 为我们提供了

- Netflix Turbine
  - spring-cloud-starter-netflix-turbines
  - @EnableTurbine
  - /turbine.stream?cluster=集群名

# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# Spring Cloud Config Server

---

## 目的

- 提供针对外置配置的 HTTP API

## 依赖

- spring-cloud-config-server
- @EnableConfigServer
- 支持 Git / SVN / Vault / JDBC ...

# 使用 Git 作为后端存储

---

## 配置

- MultipleJGitEnvironmentProperties
  - `spring.cloud.config.server.git.uri`

## 配置文件的要素

- `{application}`, 即客户端的 `spring.application.name`
- `{profile}`, 即客户端的 `spring.profiles.active`
- `{label}`, 配置文件的特定标签, 默认 master

## 访问配置内容

- HTTP 请求
  - GET `/ {application} / {profile} [ / {label} ]`
  - GET `/ {application} - {profile} . yml`
  - GET `/ {label} / {application} - {profile} . yml`
  - GET `/ {application} - {profile} . properties`
  - GET `/ {label} / {application} - {profile} . properties`

# Spring Cloud Config Client

---

## 依赖

- `spring-cloud-starter-config`

## 发现配置中心

- `bootstrap.properties | yml`
- `spring.cloud.config.fail-fast=true`
- 通过配置
  - `spring.cloud.config.uri=http://localhost:8888`



# Spring Cloud Config Client

`git-config-waiter-service`

## 发现配置中心

- bootstrap.properties | yml
- 通过服务发现
  - `spring.cloud.config.discovery.enabled=true`
  - `spring.cloud.config.discovery.service-id=configserver`

## 配置刷新

- @RefreshScope
- Endpoint - /actuator/refresh

# 内容提要

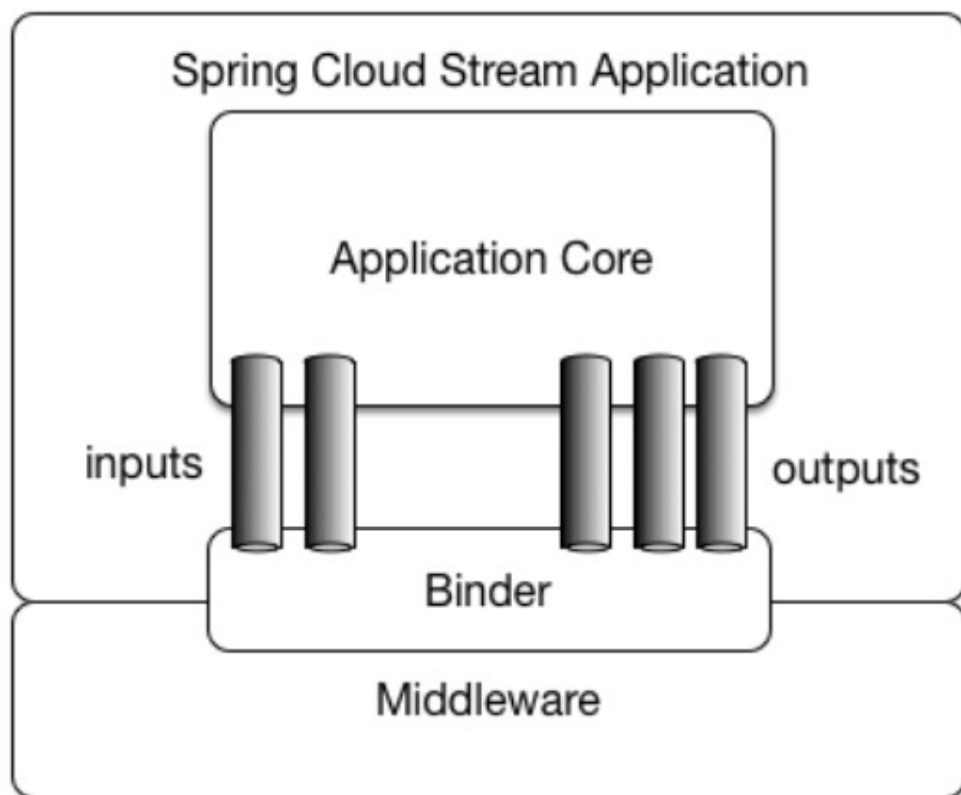
- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# Spring Cloud Stream

---

- Spring Cloud Stream 是什么
  - 一款用于构建消息驱动的微服务应用程序的轻量级框架
- 特性
  - 声明式编程模型
  - 引入多种概念抽象
    - 发布订阅、消费组、分区
  - 支持多种消息中间件
    - **RabbitMQ、Kafka .....**

# Spring Cloud Stream 的一些核心概念



## Binder

- RabbitMQ
- Apache Kafka
- Kafka Streams
- Amazon Kinesis
- RocketMQ
- .....

# Spring Cloud Stream 的一些核心概念

---

## Binding

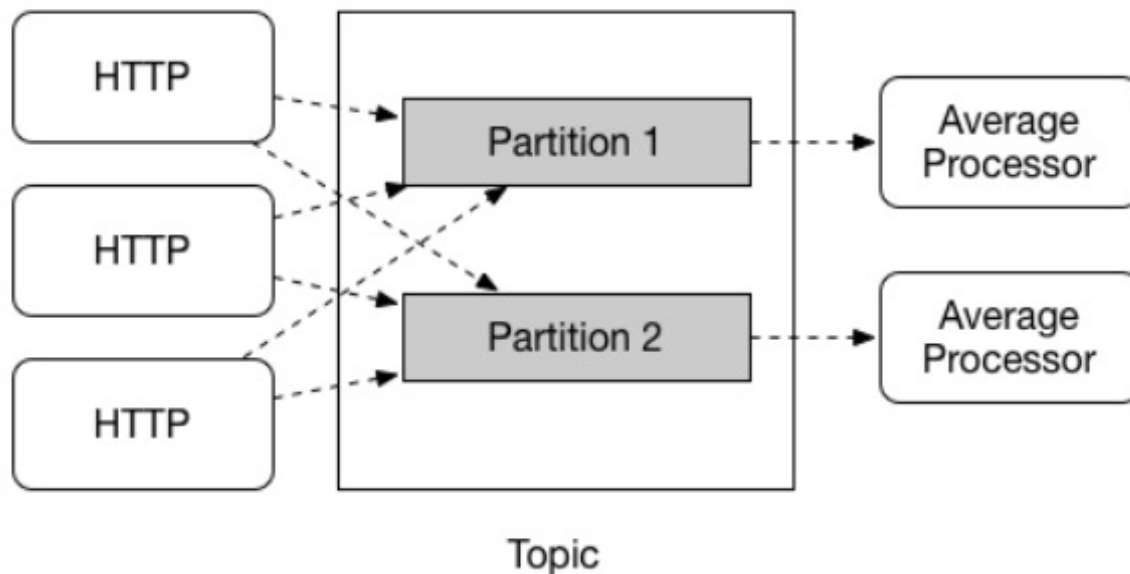
- 应用中生产者、消费者与消息系统之间的桥梁
  - @EnableBinding
  - @Input / SubscribableChannel
  - @Output / MessageChannel

# Spring Cloud Stream 的一些核心概念

## 消费组

- 对同一消息，每个组中都会有一个消费者收到消息

## 分区



# 如何发送与接收消息

---

## 生产消息

- 使用 `MessageChannel` 中的 `send()`
- `@SendTo`

## 消费消息

- `@StreamListener`
  - `@Payload` / `@Headers` / `@Header`

## 其他说明

- 可以使用 `Spring Integration`

# Spring Cloud Stream 对 RabbitMQ 的支持

## 依赖

- Spring Cloud - spring-cloud-starter-stream-rabbit
- Spring Boot - spring-boot-starter-amqp

## 配置

- `spring.cloud.stream.rabbit.binder.*`
- `spring.cloud.stream.rabbit.bindings.<channelName>.consumer.*`
- `spring.rabbitmq.*`





# 通过 Docker 启动 RabbitMQ

---

- 官方指引

- [https://hub.docker.com/\\_/rabbitmq](https://hub.docker.com/_/rabbitmq)

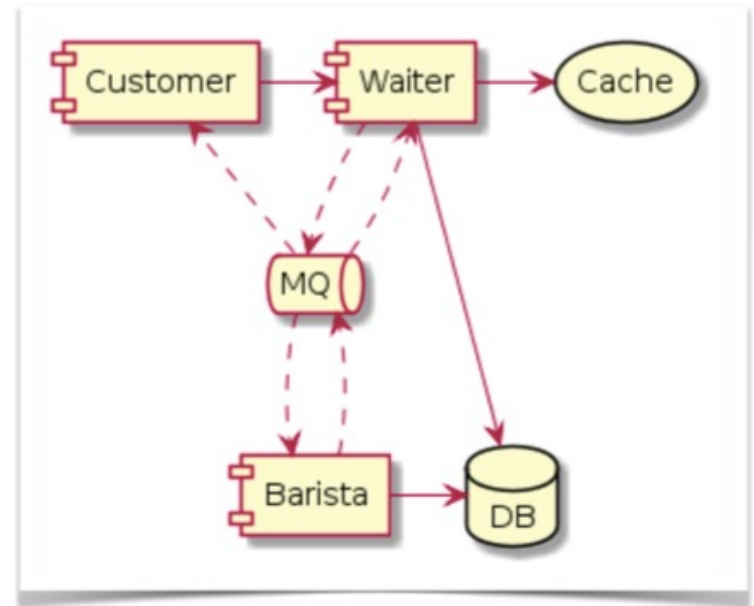
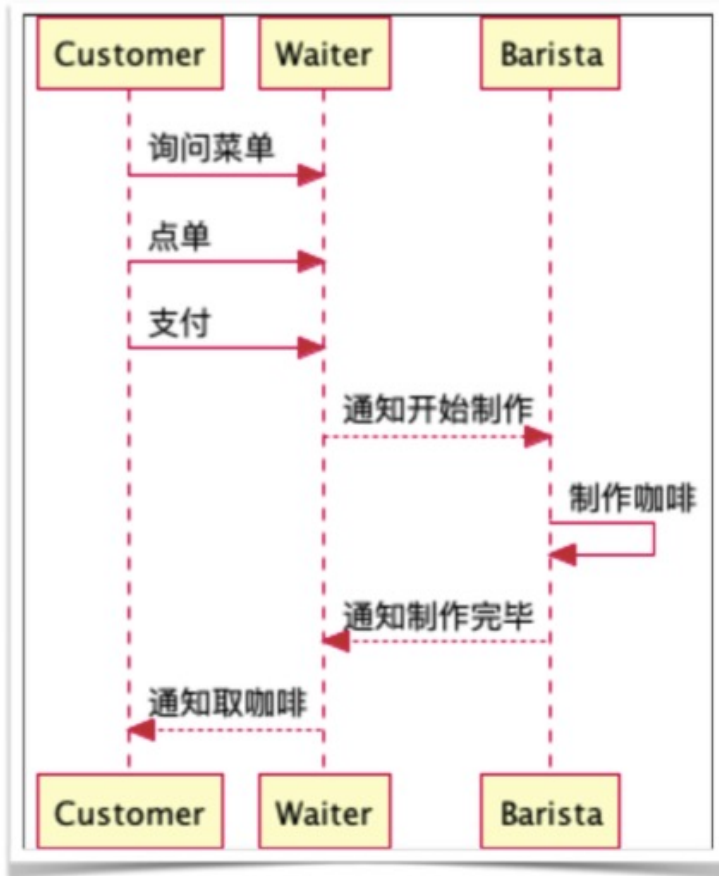
- 获取镜像

- **docker pull rabbitmq:3.7-management**

- 运行 RabbitMQ 镜像

- `docker run --name rabbitmq -d -p 5672:5672 -p 15672:15672 -e RABBITMQ_DEFAULT_USER=spring -e RABBITMQ_DEFAULT_PASS=spring rabbitmq:3.7-management`

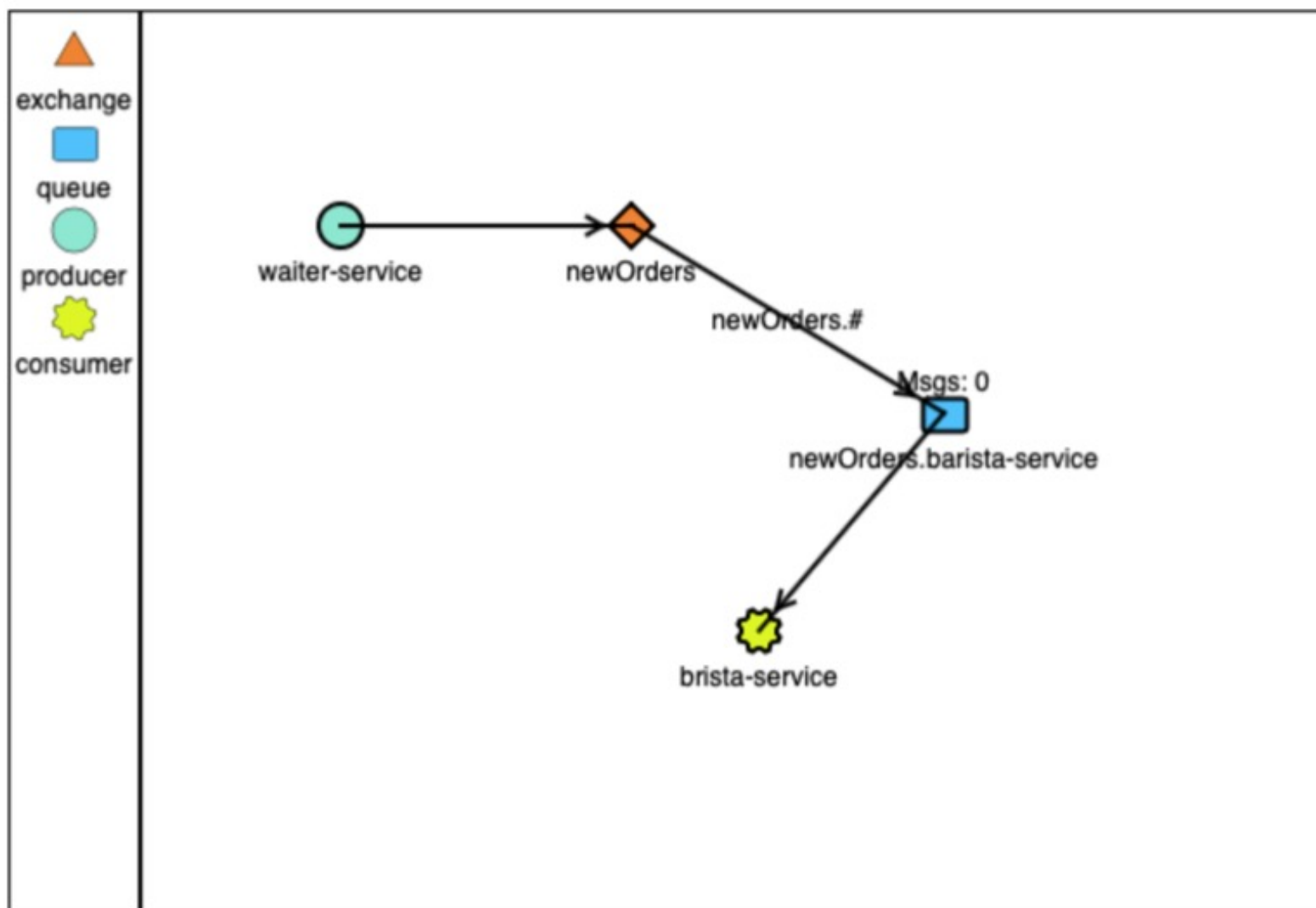
# 回顾 SpringBucks 的目标



`rabbitmq-customer-service`

`rabbitmq-barista-service`

# 消息在 RabbitMQ 的流转



# 内容提要

- 访问Web资源
- Cloud Native概述
- 理解微服务
- Spring Cloud的组成
- 服务注册与发现
- 访问服务
- 服务熔断
- 服务配置中心
- 消息驱动的微服务
- 服务链路治理

# 我们在关注什么？

---

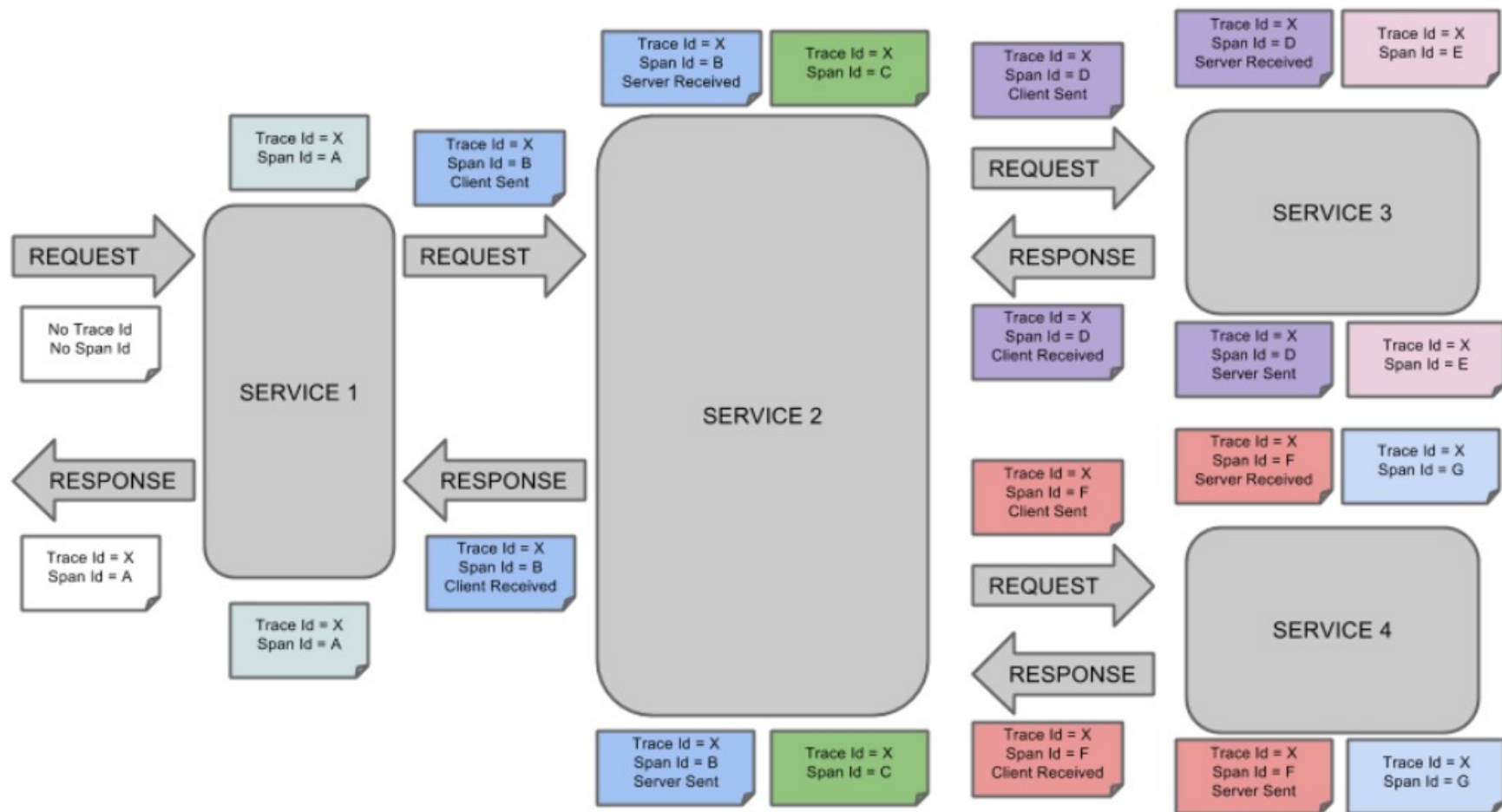
- 系统中都有哪些服务
- 服务之间的依赖关系是什么样的
- 一个常见请求具体的执行路径是什么样的
- 请求每个环节的执行是否正常与耗时情况
- .....

# Google Dapper 的一些术语

---

- Span - 基本的工作单元
- Trace - 由一组 Span 构成的树形结构
- Annotation - 用于及时记录事件
  - cs - Client Sent
  - sr - Server Received
  - ss - Server Sent
  - cr - Client Received

# 通过 Dapper 理解链路治理



# 通过 Spring Cloud Sleuth 实现链路追踪

---

## 依赖

- Spring Cloud Sleuth - spring-cloud-starter-sleuth
- Spring Cloud Sleuth with Zipkin - spring-cloud-starter-zipkin

## 日志输出

- [appName,traceId,spanId,exportable]



# Spring Cloud 提供的服务治理功能

---

## 配置

- `spring.zipkin.base-url=http://localhost:9411/`
  - `spring.zipkin.discovery-client-enabled=false`
- `spring.zipkin.sender.type=web | rabbit | kafka`
- `spring.zipkin.compression.enabled=false`
- `spring.sleuth.sampler.probability=0.1`

# 通过 Docker 启动 Zipkin

---

- 官方指引

- <https://hub.docker.com/r/openzipkin/zipkin>

- 获取镜像

- **docker pull openzipkin/zipkin**

- 运行 Zipkin 镜像

- `docker run --name zipkin -d -p 9411:9411 openzipkin/zipkin`

`zipkin-customer-service`

`zipkin-waiter-service`

# 服务治理关心什么

---

## ■ 宏观上

- 架构设计是否合理
- 哪些链路算是关键链路
- 链路的容量水位趋势
- 对系统变更的管理与审计

## ■ 微观上

- 一个系统都依赖了什么
- 一个系统都有哪些配置
- 一个系统的主观与客观质量