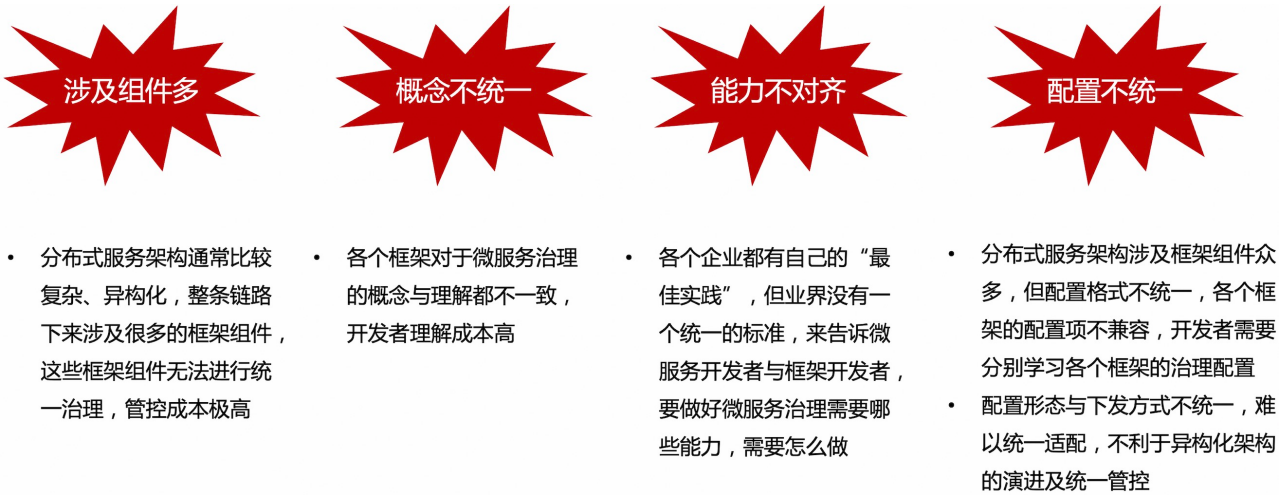
主讲老师：Fox

> 有道笔记地址：https://note.youdao.com/s/QR3szFD8

1.OpenSergo介绍

1.1 为什么需要 OpenSergo

业界微服务治理存在着以下的问题：



因此，社区共同发起了 OpenSergo 项目，旨在提供统一的微服务治理标准与参考实现，以解决上面提到的理解与统一管控复杂度的问题。

1.2 什么是 OpenSergo

> https://opensergo.io/zh-cn/docs/what-is-opensergo/intro/

OpenSergo 是开放通用的，覆盖微服务及上下游关联组件的微服务治理项目。OpenSergo 从微服务的角度出发，涵盖**流量治理、服务容错、服务元信息治理、安全治理**等关键治理领域，提供一系列的治理能力与标准、生态适配与最佳实践，支持 Java, Go, Rust 等多语言生态。OpenSergo 项目由阿里巴巴、bilibili、中国移动、SphereEx 等企业，以及 Kratos、CloudWeGo、ShardingSphere、Database Mesh、Spring Cloud Alibaba、Apache Dubbo 等社区联合发起，共同主导治理标准建设与能力演进。

OpenSergo 的最大特点就是**以统一的一套配置/DSL/协议定义服务治理规则，面向多语言异构化架构，覆盖微服务框架及上下游关联组件。**无论微服务的语言是 Java, Go, Node.js 还是其它语言，无论是标准微服务还是 Mesh 接入，从网关到微服务框架，从数据库到缓存访问，从服务注册发现到配置，开发者都可以通过同一套 OpenSergo CRD 标准配置进行统一的治理管控，而无需关注各框架、语言的差异点，降低异构化、全链路微服务治理管控的复杂度。

1.3 OpenSergo整体架构

OpenSergo 主要包含以下几部分：

- **OpenSergo Spec**：Spec 以统一的一套配置/DSL 定义微服务治理规则与配置，确保开发者可以用同一套标准对不同框架、不同协议、不同语言的微服务架构进行统一治理管控。
- **OpenSergo 控制平面**：OpenSergo 提供 Control Plane 作为统一微服务治理管控组件，承载 OpenSergo CRD 配置转换与下发的职责。
- OpenSergo SDK：OpenSergo 多语言 SDK 提供统一的 OpenSergo 适配层，供各个开源框架/组件接入到 OpenSergo 生态中。目前社区已提供 Java SDK 和 Go SDK。
- 数据面（各框架生态）：各个接入 OpenSergo 生态的微服务框架/组件，都可以通过统一的 OpenSergo CRD 进行服务治理管控。

2. 快速开始

2.1 OpenSergo 控制面安装

OpenSergo 控制平面 (Control Plane) 作为 OpenSergo CRD 的统一管控组件，承载服务治理配置转换与下发的职责。

目前 OpenSergo 控制平面支持针对部署在 Kubernetes 集群中的应用进行统一管控，未来也规划支持非 K8s 环境。

脚本方式一键安装

OpenSergo 提供脚本方式部署控制面：

```
# 注意：采用此种方式进行，会在 $HOME/opensergo/opensergo-control-plane 下载相关资源

# 1. 初始化 OpenSergo 基础资源 (Namespace, CRD, RBAC 等)
wget --no-check-certificate https://raw.githubusercontent.com/opensergo/opensergo-control-plane/main/cmd/init/init.sh && chmod +x init.sh && ./init.sh

# 2. 部署 OpenSergo 控制面 workload
wget --no-check-certificate https://raw.githubusercontent.com/opensergo/opensergo-control-plane/main/cmd/install/k8s/deploy.sh && chmod +x deploy.sh && ./deploy.sh
```

手动安装 YAML

首先需要下载 opensergo-control-plane 项目，然后通过 kubectl 手动在 Kubernetes 集群中安装 OpenSergo 控制面

```
# 创建 opensergo-system 命名空间
kubectl apply -f opensergo-control-plane/k8s/namespace.yaml

# 安装 OpenSergo spec CRD
kubectl apply -f opensergo-control-plane/k8s/crd/bases/fault-tolerance.opensergo.io_circuitbreakerstrategies.yaml
kubectl apply -f opensergo-control-plane/k8s/crd/bases/fault-tolerance.opensergo.io_concurrencylimitstrategies.yaml
kubectl apply -f opensergo-control-plane/k8s/crd/bases/fault-tolerance.opensergo.io_faulttolerancerules.yaml
kubectl apply -f opensergo-control-plane/k8s/crd/bases/fault-tolerance.opensergo.io_ratelimitstrategies.yaml
kubectl apply -f opensergo-control-plane/k8s/crd/bases/fault-tolerance.opensergo.io_throttlingstrategies.yaml
kubectl apply -f opensergo-control-plane/k8s/crd/bases/traffic.opensergo.io_trafficerouters.yaml

# OpenSergo 控制面 RBAC 权限配置
kubectl apply -f opensergo-control-plane/k8s/rbac/rbac.yaml

# 部署 OpenSergo 控制面 workload
kubectl apply -f opensergo-control-plane/k8s/workload/opensergo-control-plane.yaml
```

查看安装信息

注意：Service 默认为 ClusterIP 类型。若希望将 OpenSergo 控制平面对集群外暴露端口（如本地应用测试），则需要修改 Service 类型为 LoadBalancer。

2.2 通过 Sentinel 快速接入 OpenSergo

Sentinel 是阿里巴巴开源的云原生流量治理组件，以流量为切入点，从流量控制、并发控制、熔断降级、热点防护、系统自适应保护等多个维度来帮助保障服务的稳定性，覆盖微服务框架、云原生网关、Service Mesh 等几大场景，支持 Java、Go、C++、Rust 等多种语言的异构微服务架构。

Sentinel 2.0 将原生支持 OpenSergo 流量治理相关 CRD 配置及能力，结合 Sentinel 提供的各框架的适配模块，让 Dubbo, Spring Cloud Alibaba, gRPC, CloudWeGo 等20+框架能够无缝接入到 OpenSergo 生态中，用统一的 CRD 来配置流量路由、流控降级、服务容错等治理规则。无论是 Java 还是 Go 还是 Mesh 服务，无论是 HTTP 请求还是 RPC 调用，还是数据库 SQL 访问，用户都可以用统一的容错治理规则 CRD 来给微服务架构中的每一环配置治理，来保障服务链路的稳定性。

Spring Boot 接入 OpenSergo 数据源

Sentinel 社区提供对接 OpenSergo spec 的动态数据源模块 sentinel-datasource-opensergo，只需要按照 Sentinel 数据源的方式接入即可。

> 注意：在应用启动前，确保 OpenSergo 控制面及 CRD 已经部署在 Kubernetes 集群中。 若您需要在本地启动 demo 应用，则需要将 OpenSergo 控制面的 Service 类型调整为 LoadBalancer 或 NodePort 对外暴露 endpoint。

1.在 pom.xml 中引入 sentinel-datasource-opensergo 数据源模块：

```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-opensergo</artifactId>
    <!-- 对应 Sentinel 1.8.6 版本 -->
    <version>0.1.0-beta</version>
</dependency>
```

2.在项目合适的位置（如 Spring 初始化 hook 或 Sentinel InitFunc 中）中创建并注册 Sentinel OpenSergo 数据源。

在本 demo 中，我们通过 Spring @PostConstruct 来注册数据源：

```
@Configuration
public class DataSourceConfig {

    @PostConstruct
    public void init() throws Exception {
        // 传入 OpenSergo Control Plane 的 endpoint，以及希望监听的应用名.
        // 在我们的例子中，假定应用名为 foo-app， 配置jvm参数 -Dproject.name=foo-app
        OpenSergoDataSourceGroup openSergo = new OpenSergoDataSourceGroup("192.168.65.137", 31896,
            "default", AppNameUtil.getAppName());
        // 初始化 OpenSergo 数据源
        openSergo.start();
        // 订阅 OpenSergo 流控规则，并注册数据源到 Sentinel 流控规则数据源中.
        FlowRuleManager.register2Property(openSergo.subscribeFlowRules());
    }
}
```

3.启动 Spring Boot 应用，通过 -Dproject.name 指定应用名称，如 -Dproject.name=foo-app

在应用启动前，确保 OpenSergo 控制面及 CRD 已经部署在 Kubernetes 集群中。

4.启动应用后，即可编写 FaultToleranceRule、RateLimitStrategy 等 CR YAML 来动态配置流控容错规则，通过 kubectl apply 到集群中即可生效。

示例 CR YAML:

```yaml
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: RateLimitStrategy
metadata:
  name: rate-limit-foo
  labels:
    app: foo-app
spec:
  metricType: RequestAmount
  limitMode: Local
  threshold: 2
  statDurationSeconds: 1
---
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: FaultToleranceRule
metadata:
  name: my-opensergo-rule-1
  labels:
    app: foo-app
spec:
  targets:
    - targetResourceName: 'GET:/foo/{id}'

  strategies:
    - name: rate-limit-foo
      kind: RateLimitStrategy
```

5.持续访问 /foo/{id} 接口，可以观察到每秒钟前两次请求可以正常返回，这一秒后续的请求会返回默认的流控状态码 429。

2.3 流量防护与容错标准 v1alpha1

流量防护与容错是服务流量治理中关键的一环，以流量为切入点，通过流量控制、流量平滑、熔断降级、自适应过载保护等手段来保障服务的稳定性。在 OpenSergo 中，我们期望结合 Sentinel 等框架组件的场景实践对流量防护与容错抽出标准 CRD。一个容错治理规则 (FaultToleranceRule) 由以下三部分组成：

- Target: 针对什么样的请求
- Strategy: 容错或控制策略，如流控、熔断、并发控制、自适应过载保护、离群实例摘除等
- FallbackAction: 触发后的 fallback 行为，如返回某个错误或状态码

流量控制策略

流量控制策略 (RateLimitStrategy)，即控制单位时长内的请求量在一定范围内。多适用于激增流量下保护服务承载能力在容量之内，避免过多流量将服务打垮。RateLimitStrategy 包含以下要素：

字段名	是否必填	类型	描述
metricType	required	string (enum)	指标类型，取值范围 RequestAmount
limitMode	required	string (enum)	控制模式，单机 Local，集群总体 Global，集群按实例数转单机 GlobalToLocal
threshold	required	double	阈值，单位统计时长内最多允许的量
statDurationSeconds	required	int32	统计时长（秒），如 1 代表 1s

以下示例定义了一个集群流控的策略，集群总体维度每秒不超过 2个请求。示例 CR YAML:

```yaml
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: RateLimitStrategy
metadata:
  name: rate-limit-foo
spec:
  metricType: RequestAmount
  limitMode: Global
  threshold: 10
  statDurationSeconds: 1
```

流量平滑

流量平滑策略 (ThrottlingStrategy)，以匀速+排队等待控制效果，对并发请求进行平滑。多适用于异步后台任务（如消息 consumer 批量处理）或对延时不敏感的请求场景。ThrottlingStrategy 包含以下要素：

字段名	是否必填	类型	描述
minIntervalOfRequests	required	string (int+timeUnit)	相邻两个并发请求之间的最短时间间隔
queueTimeout	required	string (int+timeUnit)	最大排队等待时长

以下示例定义了一个匀速排队的策略，相邻两个并发请求的时间间隔不小于 20ms，同时排队平滑的等待时长不超过 500ms。示例 CR YAML:

```yaml
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: ThrottlingStrategy
metadata:
  name: throttling-foo
spec:
  minIntervalOfRequests: '20ms'
  queueTimeout: '500ms'
```

并发控制

并发控制 (ConcurrencyLimitStrategy)，即控制同时并发调用请求的数目。多适用于慢调用场景下的软隔离保护，避免调用端线程池被某些慢调用占满，导致服务不可用甚至链路不可用。

ConcurrencyLimitStrategy 包含以下要素：

字段名	是否必填	类型	描述
maxConcurrency	required	int	最大并发
limitMode	required	string (enum)	控制模式，单机 Local，集群总体 Global

示例 CR YAML:

```yaml
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: ConcurrencyLimitStrategy
```

```
metadata:
  name: concurrency-limit-foo
spec:
  maxConcurrency: 8
  limitMode: 'Local'
```

熔断保护

CircuitBreakerStrategy 对应微服务设计中标准的断路器模式，单机维度生效。CircuitBreakerStrategy 包含以下要素：

- strategy: 熔断策略，目前支持 慢调用比例 SlowRequestRatio、错误比例 ErrorRequestRatio
- triggerRatio: 触发比例
- statDuration: 统计时长，如 1s, 5min；也可考虑 timeUnit 形式
- recoveryTimeout: 进入熔断状态后的等待时长，等待后会进入半开启恢复模式
- minRequestAmount: 单位统计时长内，最小请求数
- slowConditions: 慢调用策略下的条件，若熔断策略为“慢调用比例”则必填
  - maxAllowedRt: 慢调用策略下，超出该响应时长的请求认为是慢调用
- errorConditions: 错误策略下的条件，若熔断策略为“错误比例”则必填
  - errorType: 错误类型

以下示例定义了一个慢调用比例熔断策略（在 30s 内请求超过 500ms 的比例达到 60% 时，且请求数达到5个，则会自动触发熔断，熔断恢复时长为 5s），示例 CR YAML:

```
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: CircuitBreakerStrategy
metadata:
  name: circuit-breaker-slow-foo
spec:
  strategy: SlowRequestRatio
  triggerRatio: '60%'
  statDuration: '30s'
  recoveryTimeout: '5s'
  minRequestAmount: 5
  slowConditions:
    maxAllowedRt: '500ms'
```

自适应过载保护

实例维度自适应过载保护策略 (AdaptiveOverloadProtectionStrategy)，基于某些系统指标与自适应策略结合来对实例维度的稳定性进行整体兜底保护。注意该策略的维度为某个服务的每个 pod 维度，分别生效，不区分具体条件。

AdaptiveOverloadProtectionStrategy 包含以下要素：

- metricType: 过载保护针对的指标类型，如 CPU usage percentage, system load, memory 等
- triggerThreshold: 触发值，超出此值则按条件进行限制
- adaptiveStrategy: 自适应策略，若不支持或不开启则填 NONE；目前 CPU usage 指标支持 BBR 策略

示例 CR YAML:

```yaml
apiVersion: fault-tolerance.opensergo.io/v1alpha1
kind: AdaptiveOverloadProtectionStrategy
metadata:
  name: system-overload-foo
spec:
  metricType: 'CpuPercentage'
  triggerThreshold: '70%'
  adaptiveStrategy: 'BBR'
```

2.4 Spring Cloud Alibaba 快速接入 OpenSergo

Spring Cloud Alibaba 作为一种一站式的微服务解决方案，通过基于 Spring Cloud 微服务标准为用户提供了微服务应用构建过程中的如服务注册与发现、限流降级、分布式事务与分布式消息等在内的完整微服务解决方案。过去几年被国内大量中小企业所采用，帮助大量企业更加方便地拥抱微服务。

Spring Cloud Alibaba 社区提供对接 OpenSergo 的模块 spring-cloud-starter-opensergo-adapter。

1）引入依赖

```xml
<dependency>
   <groupId>com.alibaba.cloud</groupId>
   <artifactId>spring-cloud-starter-alibaba-governance-routing</artifactId>
</dependency>

<dependency>
   <groupId>com.alibaba.cloud</groupId>
   <artifactId>spring-cloud-starter-opensergo-adapter</artifactId>
</dependency>
```

2）在 application.properties 配置文件给消费者配置 OpenSergo 控制面以及 Nacos 注册中心的相关信息：

```
spring.application.name=service-consumer
server.port=18083
management.endpoints.web.exposure.include=*
spring.cloud.nacos.discovery.server-addr=nacos.mall.com:8848
spring.cloud.nacos.discovery.fail-fast=true
spring.cloud.nacos.username=nacos
spring.cloud.nacos.password=nacos
spring.cloud.nacos.discovery.ip=192.168.65.103

#在此处配置 OpenSergo 控制面的地址
spring.cloud.opensergo.endpoint=192.168.65.130:31986
```

在应用启动前，确保 OpenSergo 控制面及 CRD 已经部署在 Kubernetes 集群中。启动应用后，即可编写 TrafficRouter 等 CR YAML 来动态配置流量路由规则，通过 kubectl apply 到集群中即可生效。

流量路由标准

流量路由，顾名思义就是将具有某些属性特征的流量，路由到指定的目标。流量路由是流量治理中重要的一环，多个路由如同流水线一样，形成一条路由链，从所有的地址表中筛选出最终目的地址集合，再通过负载均衡策略选择访问的地址。开发者可以基于流量路由标准来实现各种场景，如灰度发布、金丝雀发布、容灾路由、标签路由等。

流量路由规则 (v1alpha1) 主要分为两部分：

- 流量标签规则 (TrafficRouter)：将特定特征的流量映射至特定特征所对应的 VirtualWorkloads 上。
- Workload 集合的抽象 (VirtualWorkload)：将某一组工作负载（如 Kubernetes Deployment, Statefulset 或者一组 pod，或某个 JVM 进程，甚至是一组 DB 实例）按照一定的特征进行分类。

流量路由规则 (TrafficRouter)

流量路由规则（TrafficRouter）将特定特征的流量映射至特定特征所对应的 VirtualWorkload 上。

TrafficRouter 规则已经在 OpenSergo 控制面以及 Spring Cloud Alibaba 中实现。

假设现在需要将内部测试用户灰度到新版服务，测试用户请求头 tag 为 v2，不符合条件的外部用户流量访问 v1 版本。那么只需要配置如下 CRD 即可：

```
apiVersion: traffic.opensergo.io/v1alpha1
kind: TrafficRouter
metadata:
  name: service-provider
```

```
  namespace: default
  labels:
    app: service-provider
spec:
  hosts:
    - service-provider
  http:
    - match:
        - headers:
            tag:
              exact: v2
      route:
        - destination:
            host: service-provider
            subset: v2
            fallback:
              host: service-provider
              subset: v1
    - route:
        - destination:
            host: service-provider
            subset: v1
```

这条TrafficRouter规则指定了一条最简单的流量路由规则，将请求头 tag 为 v2 的 HTTP 请求路由到 v2 版本，其余的流量都路由到 v1 版本。如果 v2 版本没有对应的节点，则将流量 fallback 至 v1 版本。