Apache ShenYu document

Apache ShenYu

2022 年 07 月 13 日

Contents

1

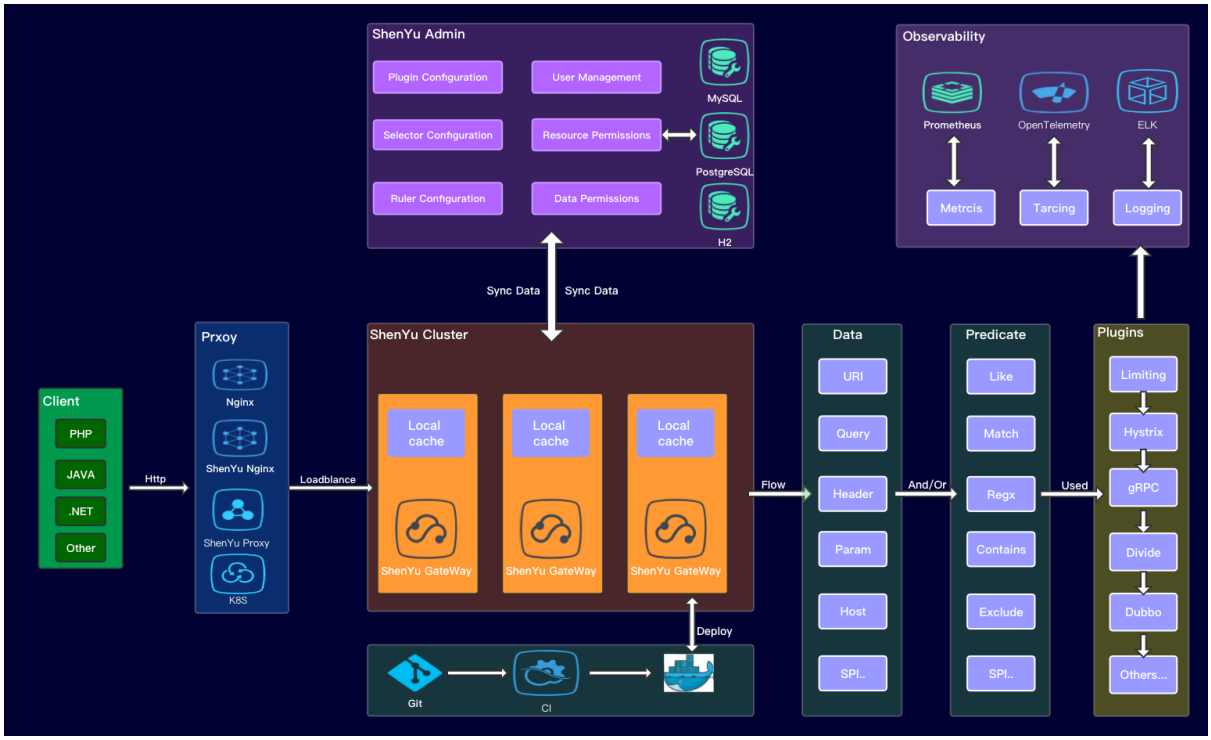
什么是 Apache ShenYu

这是一个异步的，高性能的，跨语言的，响应式的 API 网关。

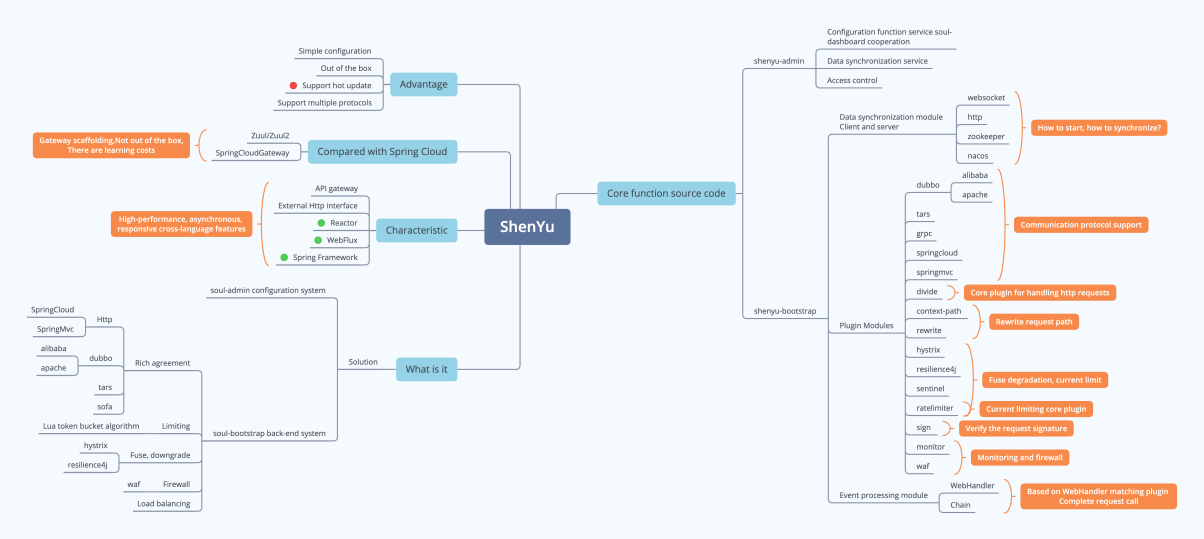
2 功能

- 支持各种语言 (http 协议)，支持 Dubbo、Spring Cloud、gRPC、Motan、Sofa、Tars 等协议。
- 插件化设计思想，插件热插拔，易扩展。
- 灵活的流量筛选，能满足各种流量控制。
- 内置丰富的插件支持，鉴权，限流，熔断，防火墙等等。
- 流量配置动态化，性能极高。
- 支持集群部署，支持 A/B Test，蓝绿发布。

3 架构图


ShenYu Admin
Plugin Configuration
User Management
MySQL
Selector Configuration
Resource Permissions
PostgreSQL
Ruler Configuration
Data Permissions
H2
Observability
Prometheus
OpenTelemetry
ELK
Metrcis
Tarcing
Logging
Sync Data
Sync Data
Client
PHP
JAVA
.NET
Other
Http
Prxoy
Nginx
ShenYu Nginx
ShenYu Proxy
K8S
Loadblance
ShenYu Cluster
Local cache
Local cache
Local cache
ShenYu GateWay
ShenYu GateWay
ShenYu GateWay
Flow
Data
URI
Query
Header
Param
Host
SPI..
And/Or
Predicate
Like
Match
Regx
Contains
Exclude
SPI..
Used
Plugins
Limiting
Hystrix
gRPC
Divide
Dubbo
Others...
Deploy
Git
CI

4 脑图


ShenYu
What is it
Solution
soul-admin configuration system
soul-bootstrap back-end system
Rich agreement
Http
SpringCloud
SpringMvc
dubbo
alibaba
apache
tars
sofa
Limiting
Lua token bucket algorithm
Fuse, downgrade
hystrix
resilience4j
Firewall
waf
Load balancing
Advantage
Simple configuration
Out of the box
Support hot update
Support multiple protocols
Compared with Spring Cloud
Zuul/Zuul2
SpringCloudGateway
Gateway scaffolding,Not out of the box, There are learning costs
Characteristic
API gateway
External Http interface
Reactor
WebFlux
Spring Framework
High-performance, asynchronous, responsive cross-language features
Core function source code
shenyu-admin
Configuration function service soul-dashboard cooperation
Data synchronization service
Access control
shenyu-bootstrap
Data synchronization module Client and server
websocket
http
zookeeper
nacos
How to start, how to synchronize?
Plugin Modules
dubbo
alibaba
apache
tars
grpc
springcloud
springmvc
Communication protocol support
divide
Core plugin for handling http requests
context-path
rewrite
Rewrite request path
hystrix
resilience4j
sentinel
ratelimiter
Fuse degradation, current limit
Current limiting core plugin
sign
Verify the request signature
monitor
waf
Monitoring and firewall
Event processing module
WebHandler
Chain
Based on WebHandler matching plugin Complete request call

5 模块

- shenyu-admin : 插件和其他信息配置的管理后台
- shenyu-bootstrap : 用于启动项目，用户可以参考
- shenyu-client : 用户可以使用 Spring MVC，Dubbo，Spring Cloud 快速访问
- shenyu-disruptor : 基于 disruptor 的封装
- shenyu-register-center : shenyu-client 提供各种 rpc 接入注册中心的支持
- shenyu-common : 框架的通用类
- shenyu-dist : 构建项目
- shenyu-metrics : prometheus（普罗米修斯）实现的 metrics
- shenyu-plugin : ShenYu 支持的插件集合
- shenyu-spi : 定义 ShenYu spi
- shenyu-spring-boot-starter : 支持 spring boot starter
- shenyu-sync-data-center : 提供 ZooKeeper，HTTP，WebSocket，Nacos 的方式同步数据
- shenyu-examples : RPC 示例项目
- shenyu-web : 包括插件、请求路由和转发等的核心处理包

6 关于

Apache ShenYu 已经被很多公司广泛使用在越来越多的业务系统，它能以高性能和灵活性让我们方便快捷的集成自己的服务和 API 。

在中国的双 11 购物狂欢节中，Apache ShenYu 集群成功支撑了海量的互联网业务。

7 Design

Apache ShenYu 网关通过插件、选择器和规则完成流量控制。相关数据结构可以参考之前的 ShenYu Admin 数据结构 。

7.1 插件

- 在 shenyu-admin 后台，每个插件都用 handle（json 格式）字段来表示不同的处理，而插件处理是就是用来管理编辑 json 里面的自定义处理字段。
- 该功能主要是用来支持插件处理模板化配置的。

7.2 选择器和规则

选择器和规则是 Apache ShenYu 网关中最灵魂的东西。掌握好它，你可以对任何流量进行管理。

一个插件有多个选择器，一个选择器对应多种规则。选择器相当于是对流量的一级筛选，规则就是最终的筛选。对一个插件而言，我们希望根据我们的配置，达到满足条件的流量，插件才会被执行。选择器和规则就是为了让流量在满足特定的条件下，才去执行我们想要的，这种规则首先要明白。

插件、选择器和规则执行逻辑如下，当流量进入到 Apache ShenYu 网关之后，会先判断是否有对应的插件，该插件是否开启；然后判断流量是否匹配该插件的选择器；然后再判断流量是否匹配该选择器的规则。如果请求流量能满足匹配条件才会执行该插件，否则插件不会被执行，处理下一个。Apache ShenYu 网关就是这样通过层层筛选完成流量控制。

7.3 流量筛选

流量筛选，是选择器和规则的灵魂，对应为选择器与规则里面的匹配条件 (conditions)，根据不同的流量筛选规则，我们可以处理各种复杂的场景。流量筛选可以从 Header, URI, Query, Cookie 等等 Http 请求获取数据，

然后可以采用 Match，=，SpEL，Regex，Groovy，Exclude 等匹配方式，匹配出你所预想的数据。多组匹配添加可以使用 And/Or 的匹配策略。

具体的介绍与使用请看: 选择器与规则管理 。

本篇主要讲解数据同步原理，数据同步是指在 shenyu-admin 后台操作数据以后，使用何种策略将数据同步到 Apache ShenYu 网关。Apache ShenYu 网关当前支持 ZooKeeper、WebSocket、Http 长轮询、Nacos 、Etcd 和 Consul 进行数据同步。

数据同步的相关配置请参考用户文档中的 数据同步配置 。

7.4 背景

网关是流量请求的入口，在微服务架构中承担了非常重要的角色，网关高可用的重要性不言而喻。在使用网关的过程中，为了满足业务诉求，经常需要变更配置，比如流控规则、路由规则等等。因此，网关动态配置是保障网关高可用的重要因素。

在实际使用 Apache ShenYu 网关过程中，用户也反馈了一些问题:

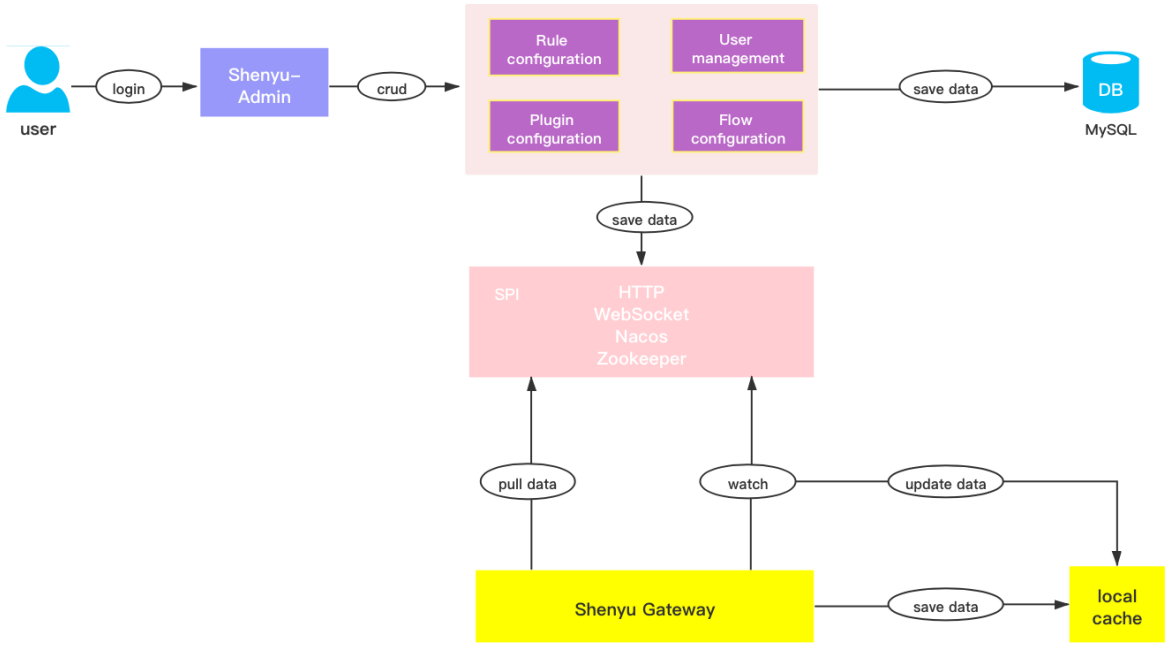
- 依赖 Zookeeper，怎么使用 Etcd、Consul、Nacos 等其他注册中心?
- 依赖 Redis、influxdb，没有使用限流插件、监控插件，为什么需要这些?
- 配置同步为什么不使用配置中心?
- 为什么不能动态配置更新?
- 每次都要查询数据库，使用 Redis 不就行了吗?

根据用户的反馈信息，我们对 Apache ShenYu 也进行了部分的重构，当前数据同步特性如下:

- 所有的配置都缓存在 Apache ShenYu 网关内存中，每次请求都使用本地缓存，速度非常快。
- 用户可以在 shenyu-admin 后台任意修改数据，并马上同步到网关内存。
- 支持 Apache ShenYu 的插件、选择器、规则数据、元数据、签名数据等数据同步。
- 所有插件的选择器，规则都是动态配置，立即生效，不需要重启服务。
- 数据同步方式支持 Zookeeper、Http 长轮询、Websocket、Nacos、Etcd 和 Consul。

7.5 原理分析

下图展示了 Apache ShenYu 数据同步的流程, Apache ShenYu 网关在启动时, 会从配置服务同步配置数据,并且支持推拉模式获取配置变更信息,然后更新本地缓存。管理员可以在管理后台(shenyu-admin),变更用户权限、规则、插件、流量配置，通过推拉模式将变更信息同步给 Apache ShenYu 网关，具体是 push 模式，还是 pull 模式取决于使用哪种同步方式。



在最初的版本中，配置服务依赖 Zookeeper 实现，管理后台将变更信息 push 给网关。而现在可以支持 WebSocket、Http 长轮询、Zookeeper、Nacos、Etcd 和 Consul，通过在配置文件中设置 shenyu.sync.${strategy} 指定对应的同步策略，默认使用 webosocket 同步策略，可以做到秒级数据同步。但是，有一点需要注意的是，Apache ShenYu 网关和 shenyu-admin 必须使用相同的同步策略。

如下图所示，shenyu-admin 在用户发生配置变更之后，会通过 EventPublisher 发出配置变更通知，由 EventDispatcher 处理该变更通知，然后根据配置的同步策略 (http、weboscket、zookeeper、naocs、etcd、consul)，将配置发送给对应的事件处理器。

- 如果是 websocket 同步策略，则将变更后的数据主动推送给 shenyu-web，并且在网关层，会有对应的 WebsocketDataHandler 处理器来处理 shenyu-admin 的数据推送。
- 如果是 zookeeper 同步策略，将变更数据更新到 zookeeper，而 ZookeeperSyncCache 会监听到 zookeeper 的数据变更，并予以处理。
- 如果是 http 同步策略，由网关主动发起长轮询请求，默认有 90s 超时时间，如果 shenyu-admin 没有数据变更，则会阻塞 http 请求，如果有数据发生变更则响应变更的数据信息，如果超过 60s 仍然没有数据变更则响应空数据，网关层接到响应后，继续发起 http 请求，反复同样的请求。

7.6 Zookeeper 同步原理

基于 zookeeper 的同步原理很简单，主要是依赖 zookeeper 的 watch 机制。Apache ShenYu 网关会监听配置的节点，shenyu-admin 在启动的时候，会将数据全量写入 zookeeper，后续数据发生变更时，会增量更新 zookeeper 的节点，与此同时，Apache ShenYu 网关会监听配置信息的节点，一旦有信息变更时，会更新本地缓存。

Apache ShenYu 将配置信息写到 zookeeper 节点，是通过精心设计的，如果您想深入了解代码实现，请参考源码 ZookeeperSyncDataService。

7.7 WebSocket 同步原理

websocket 和 zookeeper 机制有点类似，将网关与 shenyu-admin 建立好 websocket 连接时，shenyu-admin 会推送一次全量数据，后续如果配置数据发生变更，则以增量形式将变更数据通过 websocket 主动推送给 Apache ShenYu 网关。

使用 websocket 同步的时候，特别要注意断线重连，也就是要保持心跳。Apache ShenYu 使用 java-websocket 这个第三方库来进行 websocket 连接。如果您想深入了解代码实现，请参考源码 WebsocketSyncDataService。

7.8 Http 长轮询同步原理

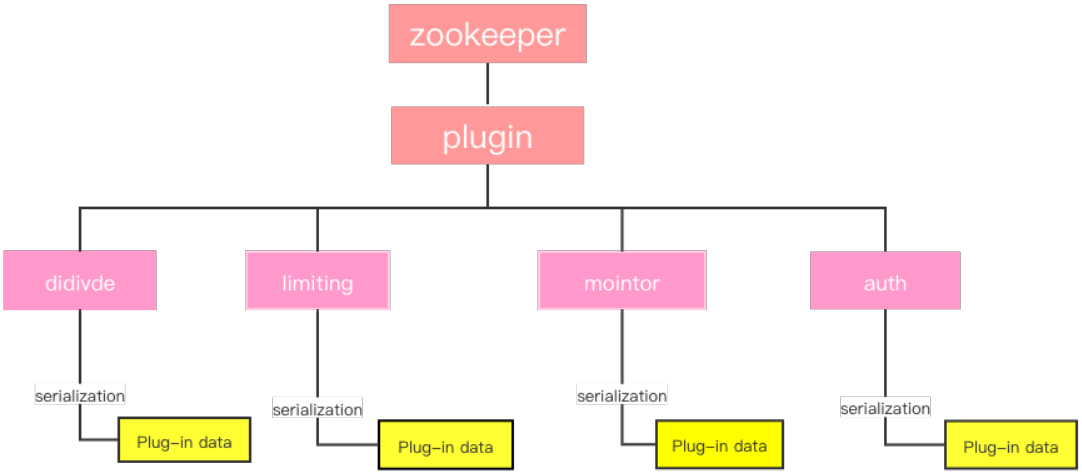
Zookeeper 和 WebSocket 数据同步的机制比较简单，而 Http 长轮询则比较复杂。Apache ShenYu 借鉴了 Apollo、Nacos 的设计思想，取其精华，自己实现了 Http 长轮询数据同步功能。注意，这里并非传统的 ajax 长轮询！

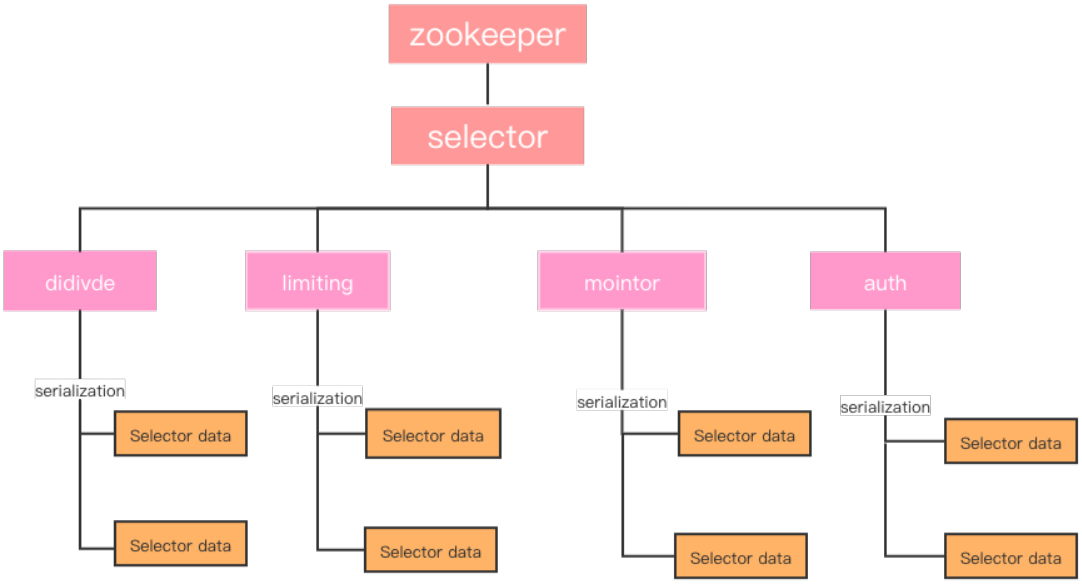
Http 长轮询机制如上所示，Apache ShenYu 网关主动请求 shenyu-admin 的配置服务，读取超时时间为 90s，意味着网关层请求配置服务最多会等待 90s，这样便于 shenyu-admin 配置服务及时响应变更数据，从而实现准实时推送。

http 请求到达 shenyu-admin 之后，并非立马响应数据，而是利用 Servlet3.0 的异步机制，异步响应数据。首先，将长轮询请求任务 LongPollingClient 扔到 BlockingQueue 中，并且开启调度任务，60s 后执行，这样做的目的是 60s 后将该长轮询请求移除队列。因为即便是没有配置变更，也需要让网关知道，不能一直等待。而且网关请求配置服务时，也有 90s 的超时时间。

如果这段时间内，管理员在 shenyu-admin 变更了配置数据，此时，会挨个移除队列中的长轮询请求，并响应数据，告知是哪个 Group 的数据发生了变更（我们将插件、规则、流量配置、用户配置数据分成不同的组）。网关收到响应信息之后，只知道是哪个 Group 发生了配置变更，还需要再次请求该 Group 的配置数据。这里可能会存在一个疑问：为什么不是直接将变更的数据写出？我们在开发的时候，也深入讨论过该问题，因为 http 长轮询机制只能保证准实时，如果在网关层处理不及时，或者管理员频繁更新配置，很有可能便错过了某个配置变更的推送，安全起见，我们只告知某个 Group 信息发生了变更。

当 shenyu-web 网关层接收到 http 响应信息之后，拉取变更信息（如果有变更的话），然后再次请求 shenyu-admin 的配置服务，如此反复循环。如果您想深入了解代码实现，请参考源码 HttpSyncDataService。





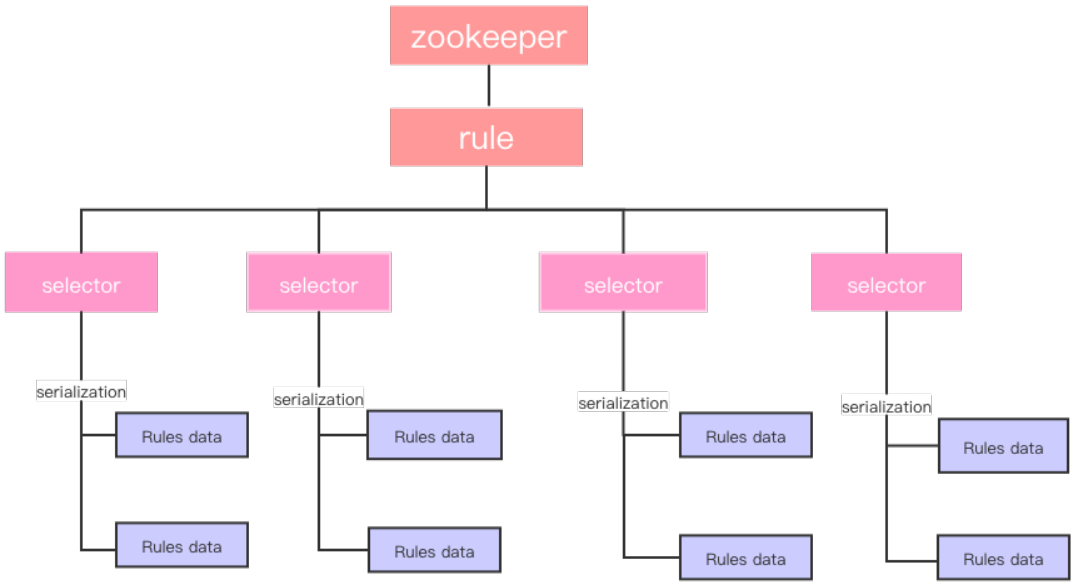


图 1: zookeeper 节点设计

7.9 Nacos 同步原理

Nacos 的同步原理与 Zookeeper 基本类似，主要依赖于 Nacos 的配置管理，各个配置节点的路径与 Zookeeper 类似。

Apache ShenYu 网关会监听配置的节点，启动时，如果 Nacos 中不存在配置节点，将同步全量的数据写入 Nacos 中，后序数据发送变更时，全量更新 Nacos 中的配置节点，与此同时，Apache ShenYu 网关会监听配置信息的节点，一旦有信息变更时，会更新本地缓存。

如果您想深入了解代码实现，请参考源码 NacosSyncDataService 和 Nacos 的官方文档。

7.10 Etcd 同步原理

Etcd 数据同步原理与 Zookeeper 类似，主要依赖于 Etcd 的 watch 机制，各个配置节点路径与 Zookeeper 相同。

Etcd 的原生 API 使用稍有点复杂，所有对其进行了一定的封装。

Apache ShenYu 网关会监听配置的节点，启动时，如果 Etcd 中不存在配置节点，将同步全量的数据写入 Etcd 中，后序数据发送变更时，增量更新 Etcd 中的配置节点，与此同时，Apache ShenYu 网关会监听配置信息的节点，一旦有信息变更时，会更新本地缓存。

如果您想深入了解代码实现，请参考源码 EtcdSyncDataService。

7.11 Consul 同步原理

Consul 数据同步原理是网关定时轮询 Consul 的配置中心，获取配置版本号与本地进行比对。

Apache ShenYu 网关会定时轮询配置的节点，默认间隔时间为 1s。启动时，如果 Consul 中不存在配置节点，将同步全量的数据写入 Consul 中，后续数据发送变更时，增量更新 Consul 中的配置节点，与此同时，Apache ShenYu 网关会定时轮询配置信息的节点，拉取配置版本号与本地进行比对，若发现版本号变更时，会更新本地缓存。

如果您想深入了解代码实现，请参考源码 ConsulSyncDataService。

Apache ShenYu Admin 是网关的后台管理系统，能够可视化管理所有插件、选择器和规则，设置用户、角色，控制资源。

7.12 插件、选择器和规则

- 插件：Apache ShenYu 使用插件化设计思想，实现插件的热插拔，极易扩展。内置丰富的插件，包括 RPC 代理、熔断和限流、权限认证、监控等等。
- 选择器：每个插件可设置多个选择器，对流量进行初步筛选。
- 规则：每个选择器可设置多个规则，对流量进行更细粒度的控制。
- 数据库 UML 类图:

- 设计详解:
  - 一个插件对应多个选择器，一个选择器对应多个规则。
  - 一个选择器对应多个匹配条件，一个规则对应多个匹配条件。
  - 每个规则在对应插件下，有不同的处理能力。

7.13 资源权限

- 资源代表的是 shenyu-admin 用户后台中的菜单或者按钮。
- 资源权限数据表用来存储用户名称、角色、资源数据以及对应关系。
- 数据库 UML 类图:
- 设计详解:
  - 一个用户对应多个角色，一个角色对应多个资源。

7.14 数据权限

- 数据权限表用来存储用户，选择器、规则对应关系。
- 数据库 UML 类图:
- 设计详解:
  - 数据权限的表为：data_permission，一个用户对应多条数据权限。
  - 数据权限表中字段 data_type 区分不同的类型数据，具体对应关系如下：0 -> 选择器，1 -> 规则。
  - 数据权限表中字段 data_id 存放相应类型的主键 id。

7.15 元数据

- 元数据主要是用于网关的泛化调用。
- 每个接口方法，对应一条元数据。
- 数据库 UML 类图:
- 设计详解:
  - path：在请求网关的时候，会根据 path 来匹配到一条数据，然后进行后续的流程。
  - rpc_ext：用于保存 RPC 代理中的扩展信息。

7.16 字典管理

- 字典管理主要用来维护和管理公用数据字典。
- 数据库 UML 类图:

SPI 全称为 Service Provider Interface, 是 JDK 内置的一种服务提供发现功能, 一种动态替换发现的机制。

shenyu-spi 是 Apache ShenYu 网关自定义的 SPI 扩展实现，设计和实现原理参考了 Dubbo 的 SPI 扩展实现 。

7.17 注册中心扩展

通过哪种方式实现服务的注册，当前支持 Consul、Etcd、Http、Nacos 和 Zookeeper。注册中心的扩展包括客户端和服务端，接口分别为 ShenyuServerRegisterRepository 和 ShenyuClientRegisterRepository 。

7.18 监控中心扩展

负责服务的监控,通过 SPI 加载具体实现,当前支持 Prometheus ,服务接口是 MetricsBootService 。

7.19 负载均衡扩展

从多个服务提供方中选择一个进行调用，当前支持的算法有 Hash、Random 和 RoundRobin，扩展接口是 LoadBalance 。

7.20 RateLimiter 扩展

在 RateLimiter 插件中，使用何种限流算法，当前支持 Concurrent、LeakyBucket、SlidingWindow 和 TokenBucket，扩展接口是 RateLimiterAlgorithm 。

7.21 匹配方式扩展

在添加选择器和规则时，使用哪种匹配方式，当前支持 And、Or，扩展接口是 MatchStrategy 。

7.22 条件参数扩展

在添加选择器和规则时，使用哪种条件参数，当前支持 URI、RequestMethod、Query、Post、IP、Host、Cookie 和 Header，扩展接口是 ParameterData 。

7.23 条件策略扩展

在添加选择器和规则时，使用哪种条件策略，当前支持 Match、Contains、Equals、Groovy、Regex、SpEL、TimerAfter、TimerBefore 和 Exclude，扩展接口是 PredicateJudge 。

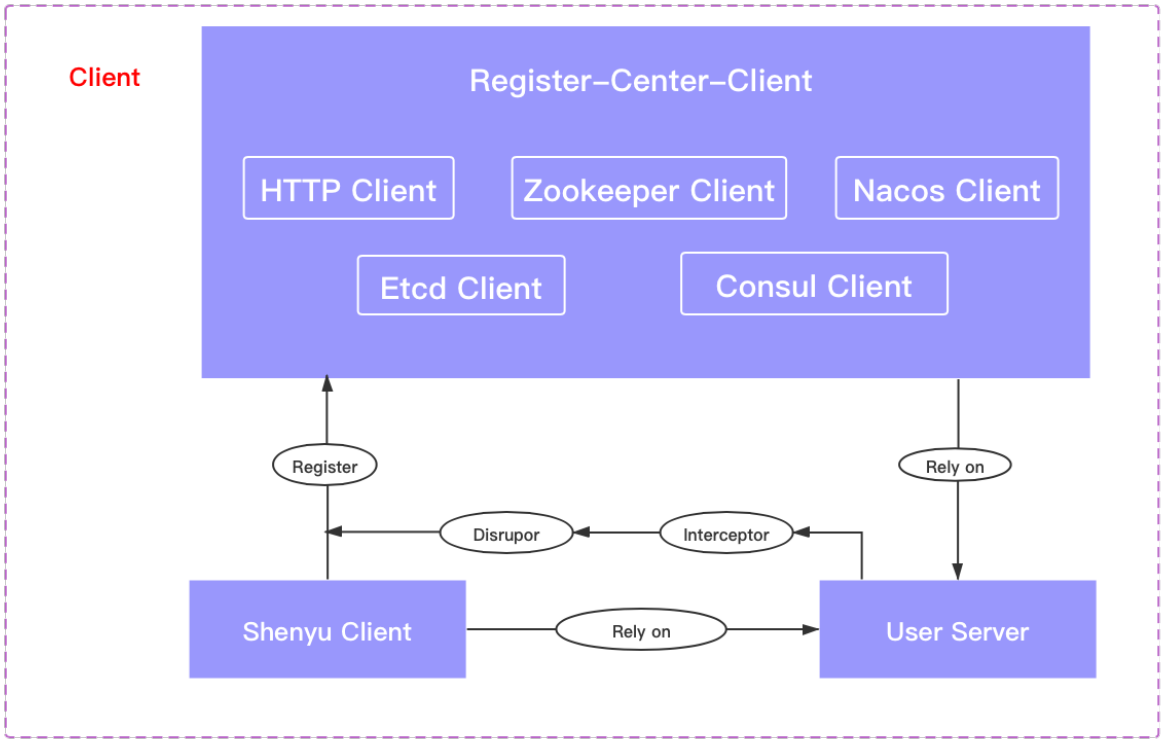
应用客户端接入是指将你的微服务接入到 Apache ShenYu 网关，当前支持 Http、Dubbo、Spring Cloud、gRPC、Motan、Sofa、Tars 等协议的接入。

将应用客户端接入到 Apache ShenYu 网关是通过注册中心来实现的，涉及到客户端注册和服务端同步数据。注册中心支持 Http、Zookeeper、Etcd、Consul 和 Nacos。

客户端接入的相关配置请参考用户文档中的 客户端接入配置 。

7.24 设计原理

7.24.1 注册中心客户端

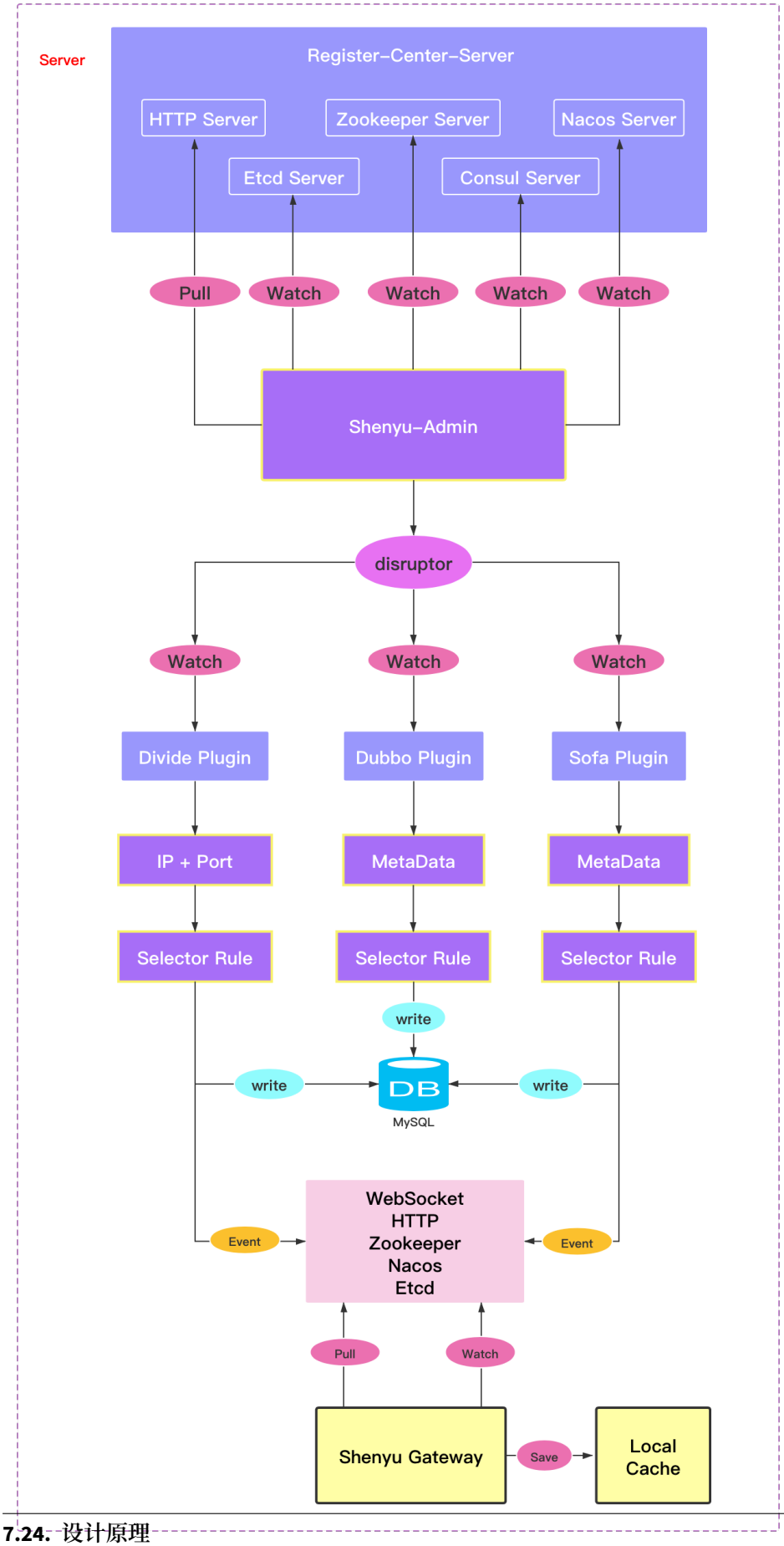


在你的微服务配置中声明注册中心客户端类型，如 Http 或 Zookeeper。应用程序启动时使用 SPI 方式加载并初始化对应注册中心客户端，通过实现 Spring Bean 相关的后置处理器接口，在其中获取需

要进行注册的服务接口信息，将获取的信息放入 Disruptor 中。

注册中心客户端从 Disruptor 中读取数据，并将接口信息注册到 shenyu-admin，Disruptor 在其中起数据与操作解耦的作用，利于扩展。

7.24.2 注册中心服务端



在 shenyu-admin 配置中声明注册中心服务端类型，如 Http 或 Zookeeper。当 shenyu-admin 启动时，读取配置类型，加载并初始化对应的注册中心服务端，注册中心服务端收到 shenyu-client 注册的接口信息后，将其放入 Disruptor 中，然后会触发注册处理逻辑，将服务接口信息更新并发布同步事件。

Disruptor 在其中起到数据与操作解耦，利于扩展。如果注册请求过多，导致注册异常，也有数据缓冲作用。

7.24.3 Http 注册原理

Http 服务注册原理较为简单，在 shenyu-client 启动后，会调用 shenyu-admin 的相关服务注册接口，上传数据进行注册。

shenyu-admin 收到请求后进行数据更新和数据同步事件发布，将接口信息同步到 Apache ShenYu 网关。

7.24.4 Zookeeper 注册原理

Zookeeper 存储结构如下:

```
shenyu
   ├──regsiter
   ├    ├──metadata
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├     ├               ├──${ruleName} : save metadata data of
MetaDataRegisterDTO
   ├    ├──uri
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├     ├               ├──${ip:prot} : save uri data of URIRegisterDTO
   ├    ├     ├               ├──${ip:prot}
```

shenyu-client 启动时，将服务接口信息（MetaDataRegisterDTO/URIRegisterDTO）写到如上的 zookeeper 节点中。

shenyu-admin 使用 Zookeeper 的 Watch 机制，对数据的更新和删除等事件进行监听，数据变更后触发对应的注册处理逻辑。在收到 MetaDataRegisterDTO 节点变更后, 触发 selector 和 rule 的数据变更和数据同步事件发布。收到 URIRegisterDTO 节点变更后，触发 selector 的 upstream 的更新和数据同步事件发布。

7.25 Etcd 注册原理

Etcd 的键值存储结构如下:

```
shenyu
   ├──regsiter
   ├    ├──metadata
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├     ├               ├──${ruleName} : save metadata data of
MetaDataRegisterDTO
   ├    ├──uri
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├     ├               ├──${ip:prot} : save uri data of URIRegisterDTO
   ├    ├     ├               ├──${ip:prot}
```

shenyu-client 启动时，将服务接口信息（MetaDataRegisterDTO/URIRegisterDTO）以 Ephemeral 方式写到如上的 Etcd 节点中。

shenyu-admin 使用 Etcd 的 Watch 机制，对数据的更新和删除等事件进行监听，数据变更后触发对应的注册处理逻辑。在收到 MetaDataRegisterDTO 节点变更后，触发 selector 和 rule 的数据变更和数据同步事件发布。收到 URIRegisterDTO 节点变更后，触发 selector 的 upstream 的更新和数据同步事件发布。

7.26 Consul 注册原理

Consul 的 Metadata 和 URI 分两部分存储，URIRegisterDTO 随着服务注册记录在服务的 metadata 里，服务下线时随着服务节点一起消失。

Consul 的 MetaDataRegisterDTO 存在 Key/Value 里，键值存储结构如下:

```
shenyu
   ├──regsiter
   ├    ├──metadata
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├     ├               ├──${ruleName} : save metadata data of
MetaDataRegisterDTO
```

shenyu-client 启动时，将服务接口信息（MetaDataRegisterDTO/URIRegisterDTO）分别放在 ServiceInstance 的 Metadata（URIRegisterDTO）和 KeyValue（MetaDataRegisterDTO），按照上述方式进行存储。

shenyu-admin 通过监听 Catalog 和 KeyValue 的 index 的变化，来感知数据的更新和删除，数据变更后触发对应的注册处理逻辑。在收到 MetaDataRegisterDTO 节点变更后，触发 selector 和 rule 的数据变更和数据同步事件发布。收到 URIRegisterDTO 节点变更后，触发 selector 的 upstream 的更新和数据同步事件发布。

7.27 Nacos 注册原理

Nacos 注册分为两部分：URI 和 Metadata。URI 使用实例注册方式，在服务异常的情况下，相关 URI 数据节点会自动进行删除，并发送事件到订阅端，订阅端进行相关的下线处理。Metadata 使用配置注册方式，没有相关上下线操作，当有 URI 实例注册时，会相应的发布 Metadata 配置，订阅端监听数据变化，进行更新处理。

URI 实例注册命令规则如下：

```
shenyu.register.service.${rpcType}
```

初始监听所有的 RpcType 节点, 其下的 ${contextPath} 实例会对应注册到其下, 根据 IP 和 Port 进行区分, 并携带其对应的 contextPath 信息。URI 实例上下线之后, 触发 selector 的 upstream 的更新和数据同步事件发布。

URI 实例上线时，会发布对应的 Metadata 数据，其节点名称命令规则如下：

```
shenyu.register.service.${rpcType}.${contextPath}
```

订阅端会对所有的 Metadata 配置继续监听，当初次订阅和配置更新后，触发 selector 和 rule 的数据变更和数据同步事件发布。

7.27.1 SPI 扩展

SPI 名称	详细说明
ShenyuClientRegisterRepository	ShenYu 网关客户端接入注册服务资源

已知实现类	详细说明
HttpClientRegisterRepository	基于 Http 请求的实现
ZookeeperClientRegisterRepository	基于 Zookeeper 注册的实现
EtcdClientRegisterRepository	基于 Etcd 注册的实现
ConsulClientRegisterRepository	基于 Consul 注册的实现
NacosClientRegisterRepository	基于 Nacos 注册的实现

SPI 名称	详细说明
ShenyuServerRegisterRepository	ShenYu 网关客户端注册的后台服务资源

已知实现类	详细说明
ShenyuHttpRegistryController	使用 Http 服务接口来处理客户端注册请求
ZookeeperServerRegisterRepository	使用 Zookeeper 来处理客户端注册节点
EtcdServerRegisterRepository	使用 Etcd 来处理客户端注册节点
ConsulServerRegisterRepository	使用 Consul 来处理客户端注册节点
NacosServerRegisterRepository	使用 Nacos 来处理客户端注册节点

8 Deployment

本文介绍使用二进制包部署 Apache ShenYu 网关。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

8.1 启动 Apache ShenYu Admin

- 下载 apache-shenyu-incubating-${current.version}-admin-bin.tar.gz
- 解压缩 apache-shenyu-incubating-${current.version}-admin-bin.tar.gz。进入 bin 目录。
- 使用 h2 来存储后台数据:

```
> windows: start.bat --spring.profiles.active = h2

> linux: ./start.sh --spring.profiles.active = h2
```

- 使用 MySQL 来存储后台数据，需按照 指引文档 初始化数据库，将 mysql-connector.jar 拷贝到 /${your_work_dir}/ext-lib, 进入 /conf 目录修改 application-mysql.yaml 中 jdbc 的配置。

```
> windows: start.bat --spring.profiles.active = mysql

> linux: ./start.sh --spring.profiles.active = mysql
```

- 使用 PostgreSql 来存储后台数据，需按照 指引文档 初始化数据库，进入 /conf 目录修改 application-pg.yaml 中 jdbc 的配置。

```
> windows: start.bat --spring.profiles.active = pg

> linux: ./start.sh --spring.profiles.active = pg
```

- 使用 Oracle 来存储后台数据，需按照 指引文档 初始化数据库，进入 /conf 目录修改 application-oracle.yaml 中 jdbc 的配置。

```
> windows: start.bat --spring.profiles.active = oracle

> linux: ./start.sh --spring.profiles.active = oracle
```

8.2 启动 Apache ShenYu Bootstrap

- 下载`apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz <https://archive.apache.org/dist/incubator/shenyu/2.4.3/apache-shenyu-incubating-2.4.3-bootstrap-bin.tar.gz>‘__
- 解压缩 apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz。进入 bin 目录。

```
> windwos : start.bat

> linux : ./start.sh
```

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

本文是介绍在集群环境中快速部署 ShenYu 网关。

在阅读本文档时，你可以先阅读二进制包部署。

8.3 环境准备

- 至少准备两台已经安装了 JDK1.8+ 的服务器用于部署网关启动器。
- 准备一台已经安装了 mysql、pgsql、h2 和 JDK1.8+ 的服务器用于部署网关管理端。
- 准备一台服务器用于部署 Nginx。

8.4 启动 Apache ShenYu Admin

- 在你的网关管理端服务器下载并解压apache-shenyu-incubating-${current.version}-admin-bin.tar.gz 。
- 配置你的数据库，进入/conf 目录，在 application.yaml 文件中修改 spring.profiles.active 节点为 mysql, pg or h2。
- 配置你的数据同步方式，进入/conf 目录，在 application.yaml 文件中修改 shenyu.sync 节点为 websocket, http, zookeeper, etcd, consul 或者 nacos。
- 进入 bin 目录，启动 ShenYu Bootstrap。

```
> windows: start.bat

> linux: ./start.sh
```

8.5 启动 Apache ShenYu Boostrap

- 在你的网关启动器服务器下载并解压apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz 。
- 配置你的数据同步方式，进入/conf 目录，在 application.yaml 文件中修改 shenyu.sync 节点为 websocket, http, zookeeper, etcd, consul 或者 nacos，这个配置必须与 ShenyYu Admin 的配置保持相同。
- 进入 bin 目录，启动 ShenYu Admin。

```
> windwos : start.bat

> linux : ./start.sh
```

在完成这些操作后，你将成功部署 ShenYu Boostrap 集群。

假如你 10.1.1.1 和 10.1.1.2 两台服务器在将部署 ShenYu Bootstrap, 并且在 10.1.1.3 部署 nginx。

8.6 启动 Nginx

- 下载并安装 nginx。
- 在 nginx.conf 文件中修改 upstream 和 server 节点的配置。

```
upstream shenyu_gateway_cluster {
  ip_hash;
  server 10.1.1.1:9195 max_fails=3 fail_timeout=10s weight=50;
  server 10.1.1.2:9195 max_fails=3 fail_timeout=10s weight=50;
}

server {
  listen 9195;
  location / {
        proxy_pass http://shenyu_gateway_cluster;
        proxy_set_header HOST $host;
        proxy_read_timeout 10s;
        proxy_connect_timeout 10s;
  }
}
```

- 启动 nginx.

```
> windows: ./nginx.exe

> linux: /usr/local/nginx/sbin/nginx
```

- 验证 nginx 配置是否生效, 在 ShenYu Bootstrap 或者 Nginx 的日志文件中查看请求被分发到那台服务器上。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

本文介绍使用 k8s 来部署 Apache ShenYu 网关。

目录

一. 使用 h2 作为数据库

1. 创建 nameSpace 和 configMap
2. 部署 shenyu-admin
3. 部署 shenyu-bootstrap 二. 使用 mysql 作为数据库

和 h2 过程类似，需要注意的两个地方

1. 需要加载 mysql-connector.jar，所以需要一个文件存储的地方
2. 需要指定外部 mysql 数据库配置，通过 endpoint 来代理外部 mysql 数据库

具体流程如下:

1. 创建 nameSpace 和 configMap
2. 创建 endpoint 代理外部 mysql
3. 创建 pv 存储 mysql-connector.jar
4. 部署 shenyu-admin
5. 部署 shenyu-bootstrap

8.7 一. 使用 h2 作为数据库

8.7.1 1. 创建 nameSpace 和 configMap

- 创建文件 shenyu-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: shenyu
  labels:
    name: shenyu
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: shenyu-cm
  namespace: shenyu
data:
  application-local.yml: |
```

```
server:
    port: 9195
    address: 0.0.0.0
spring:
    main:
        allow-bean-definition-overriding: true
    application:
        name: shenyu-bootstrap
management:
    health:
        defaults:
        enabled: false
shenyu:
    local:
        enabled: true
    file:
        enabled: true
    cross:
        enabled: true
    dubbo:
        parameter: multi
    sync:
        websocket:
        urls: ws://shenyu-admin-svc.shenyu.svc.cluster.local:9095/websocket
    exclude:
        enabled: false
        paths:
        - /favicon.ico
    extPlugin:
        enabled: true
        threads: 1
        scheduleTime: 300
        scheduleDelay: 30
    scheduler:
        enabled: false
        type: fixed
        threads: 16
logging:
    level:
        root: info
        org.springframework.boot: info
        org.apache.ibatis: info
        org.apache.shenyu.bonuspoint: info
        org.apache.shenyu.lottery: info
        org.apache.shenyu: info
```

- 执行 `kubectl apply -f shenyu-ns.yaml`

8.7.2 2. 部署 shenyu-admin

- 创建文件 shenyu-admin.yaml

```
# 示例使用 nodeport 方式暴露端口
apiVersion: v1
kind: Service
metadata:
  namespace: shenyu
  name: shenyu-admin-svc
spec:
  selector:
    app: shenyu-admin
  type: NodePort
  ports:
  - protocol: TCP
    port: 9095
    targetPort: 9095
    nodePort: 31095
---
# shenyu-admin
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: shenyu
  name: shenyu-admin
spec:
  selector:
    matchLabels:
      app: shenyu-admin
  replicas: 1
  template:
    metadata:
      labels:
        app: shenyu-admin
    spec:
      containers:
      - name: shenyu-admin
        image: apache/shenyu-admin:${current.version}
        imagePullPolicy: Always
        ports:
        - containerPort: 9095
        env:
        - name: 'TZ'
          value: 'Asia/Beijing'
```

- 执行 kubectl apply -f shenyu-admin.yaml

8.7.3 3. 部署 shenyu-bootstrap

- 创建文件 shenyu-bootstrap.yaml

```
# 示例使用 nodeport 方式暴露端口
apiVersion: v1
kind: Service
metadata:
  namespace: shenyu
  name: shenyu-bootstrap-svc
spec:
  selector:
    app: shenyu-bootstrap
  type: NodePort
  ports:
  - protocol: TCP
    port: 9195
    targetPort: 9195
    nodePort: 31195
---
# shenyu-bootstrap
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: shenyu
  name: shenyu-bootstrap
spec:
  selector:
    matchLabels:
      app: shenyu-bootstrap
  replicas: 1
  template:
    metadata:
      labels:
        app: shenyu-bootstrap
    spec:
      volumes:
      - name: shenyu-bootstrap-config
        configMap:
          name: shenyu-cm
          items:
          - key: application-local.yml
            path: application-local.yml
      containers:
      - name: shenyu-bootstrap
        image: apache/shenyu-bootstrap:${current.version}
        ports:
        - containerPort: 9195
        env:
```

```
        - name: TZ
          value: Asia/Beijing
        volumeMounts:
        - name: shenyu-bootstrap-config
          mountPath: /opt/shenyu-bootstrap/conf/application-local.yml
          subPath: application-local.yml
```

- 执行 kubectl apply -f shenyu-bootstrap.yaml

8.8 二. 使用 mysql 作为数据库

8.8.1 1. 创建 nameSpace 和 configMap

- 创建文件 shenyu-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: shenyu
  labels:
    name: shenyu
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: shenyu-cm
  namespace: shenyu
data:
  application-local.yml: |
    server:
        port: 9195
        address: 0.0.0.0
    spring:
        main:
            allow-bean-definition-overriding: true
        application:
            name: shenyu-bootstrap
    management:
        health:
            defaults:
            enabled: false
    shenyu:
        local:
            enabled: true
        file:
            enabled: true
        cross:
```

```
            enabled: true
        dubbo:
            parameter: multi
        sync:
            websocket:
            urls: ws://shenyu-admin-svc.shenyu.svc.cluster.local:9095/websocket
        exclude:
            enabled: false
            paths:
            - /favicon.ico
        extPlugin:
            enabled: true
            threads: 1
            scheduleTime: 300
            scheduleDelay: 30
        scheduler:
            enabled: false
            type: fixed
            threads: 16
    logging:
        level:
            root: info
            org.springframework.boot: info
            org.apache.ibatis: info
            org.apache.shenyu.bonuspoint: info
            org.apache.shenyu.lottery: info
            org.apache.shenyu: info
  application-mysql.yml: |
    spring.datasource.url: jdbc:mysql://mysql.shenyu.svc.cluster.local:3306/shenyu?
useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=Asia/Shanghai&
zeroDateTimeBehavior=convertToNull
    spring.datasource.username: {your_mysql_user}
    spring.datasource.password: {your_mysql_password}
```

- 执行 `kubectl apply -f shenyu-ns.yaml`

8.8.2 2. 创建 endpoint 代理外部 mysql

- 创建文件 shenyu-ep.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: mysql
  namespace: shenyu
spec:
  ports:
  - port: 3306
```

```
    name: mysql
    targetPort: {your_mysql_port}
---
kind: Endpoints
apiVersion: v1
metadata:
  name: mysql
  namespace: shenyu
subsets:
- addresses:
  - ip: {your_mysql_ip}
  ports:
  - port: {your_mysql_port}
    name: mysql
```

- 执行 kubectl apply -f shenyu-ep.yaml

8.8.3 3. 创建 pv 存储 mysql-connector.jar

- 创建文件 shenyu-store.yaml

```
# 示例使用 pvc、pv、storageClass 来存储文件
apiVersion: v1
kind: PersistentVolume
metadata:
  name: shenyu-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /home/shenyu/shenyu-admin/k8s-pv  # 指定节点上的目录，该目录下面需要包含 mysql-
connector.jar
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - {your_node_name} # 指定节点
---
kind: PersistentVolumeClaim
apiVersion: v1
```

```
metadata:
  name: shenyu-pvc
  namespace: shenyu
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: local-storage
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

- 执行 kubectl apply -f shenyu-store.yaml
- pv 挂载目录下上传 mysql-connector.jar

8.8.4 4. 部署 shenyu-admin

- 创建文件 shenyu-admin.yaml

```
# 示例使用 nodeport 方式暴露端口
apiVersion: v1
kind: Service
metadata:
  namespace: shenyu
  name: shenyu-admin-svc
spec:
  selector:
    app: shenyu-admin
  type: NodePort
  ports:
  - protocol: TCP
    port: 9095
    targetPort: 9095
    nodePort: 31095
---
# shenyu-admin
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: shenyu
  name: shenyu-admin
```

```
spec:
  selector:
    matchLabels:
      app: shenyu-admin
  replicas: 1
  template:
    metadata:
      labels:
        app: shenyu-admin
    spec:
      volumes:
      - name: mysql-connector-volume
        persistentVolumeClaim:
          claimName: shenyu-pvc
      - name: shenyu-admin-config
        configMap:
          name: shenyu-cm
          items:
          - key: application-mysql.yml
            path: application-mysql.yml
      containers:
      - name: shenyu-admin
        image: apache/shenyu-admin:${current.version}
        imagePullPolicy: Always
        ports:
        - containerPort: 9095
        env:
        - name: 'TZ'
          value: 'Asia/Beijing'
        - name: SPRING_PROFILES_ACTIVE
          value: mysql
        volumeMounts:
        - name: shenyu-admin-config
          mountPath: /opt/shenyu-admin/config/application-mysql.yml
          subPath: application-mysql.yml
        - mountPath: /opt/shenyu-admin/ext-lib
          name: mysql-connector-volume
```

- 执行 kubectl apply -f shenyu-admin.yaml

8.8.5 3. 部署 shenyu-bootstrap

- 创建文件 shenyu-bootstrap.yaml

```
# 示例使用 nodeport 方式暴露端口
apiVersion: v1
kind: Service
metadata:
  namespace: shenyu
  name: shenyu-bootstrap-svc
spec:
  selector:
    app: shenyu-bootstrap
  type: NodePort
  ports:
  - protocol: TCP
    port: 9195
    targetPort: 9195
    nodePort: 31195
---
# shenyu-bootstrap
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: shenyu
  name: shenyu-bootstrap
spec:
  selector:
    matchLabels:
      app: shenyu-bootstrap
  replicas: 1
  template:
    metadata:
      labels:
        app: shenyu-bootstrap
    spec:
      volumes:
      - name: shenyu-bootstrap-config
        configMap:
          name: shenyu-cm
          items:
          - key: application-local.yml
            path: application-local.yml
      containers:
      - name: shenyu-bootstrap
        image: apache/shenyu-bootstrap:${current.version}
        ports:
        - containerPort: 9195
        env:
```

```
        - name: TZ
          value: Asia/Beijing
        volumeMounts:
        - name: shenyu-bootstrap-config
          mountPath: /opt/shenyu-bootstrap/conf/application-local.yml
          subPath: application-local.yml
```

- 执行 `kubectl apply -f shenyu-bootstrap.yaml`

本文介绍使用 helm 来部署 Apache ShenYu 网关。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

本文介绍在部署 Apache ShenYu 网关前, 所需要准备的一些先决条件。

8.9 数据库环境准备

在部署 shenyu-admin 项目前, 需初始化其所使用的数据库（数据库目前支持: Mysql、PostgreSql、Oracle）, 其中所用到的脚本文件都存放在 项目根目录下的 db 目录 中, 以下介绍了各数据库的初始步骤.

8.9.1 Mysql

在项目 mysql 初始化脚本目录 中找到初始化脚本 schema.sql, 使用客户端连接工具连接您的 Mysql 服务并执行, 由此您会得到一个名为 shenyu 的数据库, 它之后可作为 shenyu-admin 项目的数据库使用.

8.9.2 PostgreSql

在项目 pg 初始化脚本目录 中找到初始化脚本 create-database.sql、create-table.sql, 并使用客户端连接工具连接您的 PostgreSql 服务依次执行，由此您会得到一个名为 shenyu 的数据库, 它之后可作为 shenyu-admin 项目的数据库使用.

8.9.3 Oracle

在项目 oracle 初始化脚本目录 中找到初始化脚本 schema.sql, 使用客户端连接工具连接您的 Oracle 服务创建一个数据库, 在此数据库上执行 schema.sql 脚本, 由此您便初始化了 shenyu-admin 的数据库, 之后可在项目配置文件 中调整您的 oracle 环境配置.

本文介绍单机环境快速启动 Apache ShenYu 网关。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

8.10 环境准备

- 本地正确安装 JDK1.8+

8.11 启动 Apache ShenYu Bootstrap

- 下载 apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz
- 解压缩 `apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz`。进入 bin 目录。

```
> windwos : start.bat

> linux : ./start.sh
```

8.12 选择器及规则配置

参考本地模式进行选择器及规则的配置。

示例:

- 如服务地址是 `http://127.0.0.1:8080/helloworld`, 直接访问将返回如下

```
{
  "name" : "Shenyu",
  "data" : "hello world"
}
```

- 按照如下进行选择器和规则配置

8.13 使用 postman

Headers 中添加 `localKey: 123456`。如果需要自定义 localKey，可以使用 sha512 工具根据明文生成 key，并更新 `shenyu.local.sha512Key` 属性。

请求方式 POST，地址 `http://localhost:9195/shenyu/plugin/selectorAndRules`，body 选择 raw json，内容如下:

```
Headers

localKey: 123456
```

```
{
    "pluginName": "divide",
    "selectorHandler": "[{\"upstreamUrl\":\"127.0.0.1:8080\"}]",
```

```
    "conditionDataList": [{
        "paramType": "uri",
        "operator": "match",
        "paramValue": "/**"
    }],
    "ruleDataList": [{
        "ruleHandler": "{\"loadBalance\":\"random\"}",
        "conditionDataList": [{
            "paramType": "uri",
            "operator": "match",
            "paramValue": "/**"
        }]
    }]
}
```

8.14 使用 curl

```
curl --location --request POST 'http://localhost:9195/shenyu/plugin/
selectorAndRules' \
--header 'Content-Type: application/json' \
--header 'localKey: 123456' \
--data-raw '{
    "pluginName": "divide",
    "selectorHandler": "[{\"upstreamUrl\":\"127.0.0.1:8080\"}]",
    "conditionDataList": [{
        "paramType": "uri",
        "operator": "match",
        "paramValue": "/**"
    }],
    "ruleDataList": [{
        "ruleHandler": "{\"loadBalance\":\"random\"}",
        "conditionDataList": [{
            "paramType": "uri",
            "operator": "match",
            "paramValue": "/**"
        }]
    }]
}'
```

- 通过 `http://localhost:9195/helloworld` 请求服务，返回如下：

```
{
  "name" : "Shenyu",
  "data" : "hello world"
}
```

本文介绍如何基于 Apache ShenYu 搭建属于你自己的网关。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

8.15 启动 Apache ShenYu Admin

- docker 用户参考 docker 部署 Apache ShenYu Admin
- liunx/windows 用户参考二进制包部署 Apache ShenYu Admin

8.16 搭建自己的网关（推荐）

- 首先新建一个空的 springboot 项目，可以参考 shenyu-bootstrap，也可以在 spring 官网 创建。
- 引入如下 jar 包:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
        <version>2.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
        <version>2.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.apache.shenyu</groupId>
        <artifactId>shenyu-spring-boot-starter-gateway</artifactId>
        <version>${current.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.shenyu</groupId>
        <artifactId>shenyu-spring-boot-starter-sync-data-websocket</artifactId>
        <version>${current.version}</version>
    </dependency>
</dependencies>
```

其中，${project.version} 请使用当前最新版本。

- 在你的 application.yaml 文件中加上如下配置:

```
spring:
  main:
    allow-bean-definition-overriding: true
management:
```

```
  health:
    defaults:
      enabled: false
shenyu:
  sync:
    websocket:
      urls: ws://localhost:9095/websocket  //设置成你的 shenyu-admin 地址
```

本文介绍本地环境启动 Apache ShenYu 网关。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

8.17 环境准备

- 本地正确安装 JDK1.8+
- 本地正确安装 Git
- 本地正确安装 Maven
- 选择一款开发工具，比如 IDEA

8.18 下载编译代码

- 下载代码

```
> git clone https://github.com/apache/incubator-shenyu.git
> cd incubator-shenyu
> mvn clean install -Dmaven.javadoc.skip=true -B -Drat.skip=true -Djacoco.skip=true
-DskipITs -DskipTests
```

- 使用开发工具启动 org.apache.shenyu.admin.ShenyuAdminBootstrap，访问 http://localhost:9095，默认用户名和密码分别为: admin 和 123456。
  - 如果使用 h2 来存储，设置变量 --spring.profiles.active = h2 启动服务。
  - 如果使用 MySQL 来存储, 需按照 指引文档 初始化数据库和修改 application-mysql.yml 中的 jdbc 相关配置，再设置变量 --spring.profiles.active = mysql 启动服务。
  - 如果使用 PostgreSql 来存储，需按照 指引文档 初始化数据库和修改 application-pg.yml 中的 jdbc 相关配置，再设置变量 --spring.profiles.active = pg 启动服务。
  - 如果使用 Oracle 来存储，需按照 指引文档 初始化数据库和修改 application-oracle.yml 中的 jdbc 相关配置，再设置变量 --spring.profiles.active = oracle 启动服务。
- 使用开发工具启动 org.apache.shenyu.bootstrap.ShenyuBootstrapApplication。

本文介绍使用 docker-compose 来部署 Apache ShenYu 网关。

在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

8.19 下载 shell 脚本

```
curl -O https://raw.githubusercontent.com/apache/incubator-shenyu/master/shenyu-dist/shenyu-docker-compose-dist/src/main/resources/install.sh
```

8.20 执行脚本

这个脚本会下载需要的配置文件、mysql-connector，如果发现下载失败可以重复执行。

```
sh ./install.sh # 默认拉取最新配置，如果需要部署已发布版本，可增加一个参数表示版本号，比如：v2.4.2 或 latest
```

8.21 初始化 shenyu-admin 存储数据源

参考数据库初始文档 初始化数据库环境。

8.22 修改配置文件

修改脚本下载的配置文件来设置 JDBC 等配置。

8.23 执行 docker-compose

```
docker-compose -f ./shenyu-${VERSION}/docker-compose.yaml up -d
```

本文介绍使用 docker 来部署 Apache ShenYu 网关。

> 在阅读本文档前，你需要先阅读部署先决条件文档来完成部署 shenyu 前的环境准备工作.

8.24 启动 Apache ShenYu Admin

```
> docker pull apache/shenyu-admin:${current.version}
> docker network create shenyu
```

- 使用 h2 来存储后台数据:

```
> docker run -d -p 9095:9095 --net shenyu apache/shenyu-admin:${current.version}
```

- 使用 MySQL 来存储后台数据，按照 指引文档 初始化数据库，将 mysql-connector.jar 拷贝到 /${your_work_dir}/ext-lib:

```
docker run -v /${your_work_dir}/ext-lib:/opt/shenyu-admin/ext-lib -e "SPRING_
PROFILES_ACTIVE=mysql" -e "spring.datasource.url=jdbc:mysql://${your_ip_port}/
shenyu?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=Asia/
Shanghai&zeroDateTimeBehavior=convertToNull" -e "spring.datasource.username=${your_
username}" -e "spring.datasource.password=${your_password}" -d -p 9095:9095 --net
shenyu apache/shenyu-admin:${current.version}
```

另外一种方式，从 配置文件地址 中把 application.yml、application-mysql.yml、application-pg.yml、application-oracle.yml 配置放到 ${your_work_dir}/conf ，调整 application.yml 中的配置 spring.profiles.active = mysql，然后执行以下语句:

```
docker run -v ${your_work_dir}/conf:/opt/shenyu-admin/conf -v /${your_work_dir}/
ext-lib:/opt/shenyu-admin/ext-lib -d -p 9095:9095 --net shenyu apache/shenyu-admin:
${current.version}
```

- 使用 PostgreSql 来存储后台数据, 按照 指引文档 初始化数据库, 执行以下语句:

```
docker run -e "SPRING_PROFILES_ACTIVE=pg" -e "spring.datasource.
url=jdbc:postgresql://${your_ip_port}/shenyu?useUnicode=true&characterEncoding=utf-
8&useSSL=false" -e "spring.datasource.username=${your_username}" -e "spring.
datasource.password=${your_password}" -d -p 9095:9095 --net shenyu apache/shenyu-
admin:${current.version}
```

另外一种方式，从 配置文件地址 中把 application.yml、application-mysql.yml、application-pg.yml、application-oracle.yml 配置放到 ${your_work_dir}/conf，调整 application.yml 中的配置 spring.profiles.active = pg，然后执行以下语句:

```
docker run -v ${your_work_dir}/conf:/opt/shenyu-admin/conf -d -p 9095:9095 --net
shenyu apache/shenyu-admin:${current.version}
```

- 使用 Oracle 来存储后台数据, 按照 指引文档 初始化数据库, 执行以下语句:

```
docker run -e "SPRING_PROFILES_ACTIVE=oracle" -e "spring.datasource.
url=jdbc:oracle:thin:@localhost:1521/shenyu" -e "spring.datasource.username=${your_
username}" -e "spring.datasource.password=${your_password}" -d -p 9095:9095 --net
shenyu apache/shenyu-admin:${current.version}
```

另外一种方式，从 配置文件地址 中把 application.yml、application-mysql.yml、application-pg.yml、application-oracle.yml 配置放到 ${your_work_dir}/conf，调整 application.yml 中的配置 spring.profiles.active = oracle，然后执行以下语句:

```
docker run -v ${your_work_dir}/conf:/opt/shenyu-admin/conf -d -p 9095:9095 --net
shenyu apache/shenyu-admin:${current.version}
```

8.25 启动 Apache ShenYu Bootstrap

宿主机中，bootstrap 的配置文件所在目录记为 $BOOTSTRAP_CONF。

```
> docker network create shenyu
> docker pull apache/shenyu-bootstrap:${current.version}
> docker run -d \
  -p 9195:9195 \
  -v $BOOTSTRAP_CONF:/opt/shenyu-bootstrap/conf \
  apache/shenyu-bootstrap:${current.version}
```

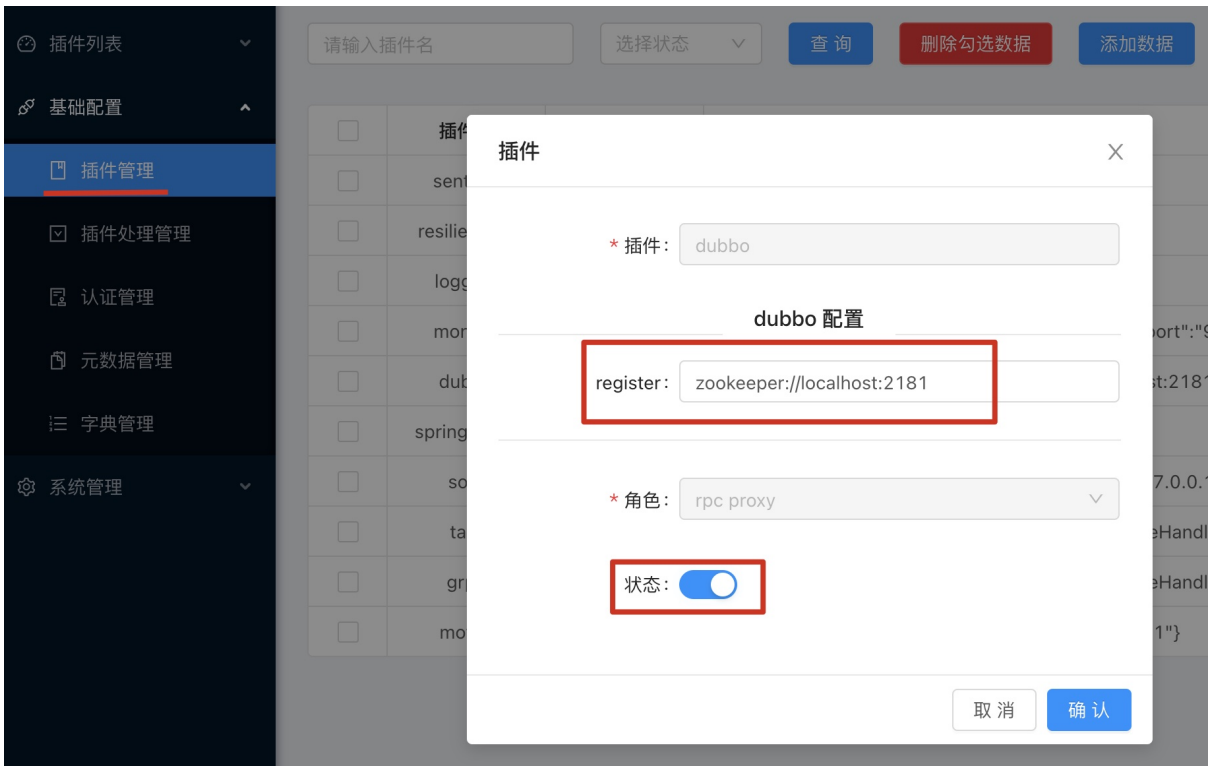
9 Quick Start

本文档演示如何将 Dubbo 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的示例代码 。

9.1 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 dubbo 插件设置为开启，并设置你的注册地址，请确保注册中心在你本地已经开启。



启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。

如果客户端是 apache dubbo，注册中心使用 zookeeper，请参考如下配置：

```
<!-- apache shenyu  apache dubbo plugin start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-apache-dubbo</artifactId>
    <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.7.5</version>
</dependency>
<!-- Dubbo zookeeper registry dependency start -->
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-client</artifactId>
    <version>4.0.1</version>
    <exclusions>
        <exclusion>
            <artifactId>log4j</artifactId>
            <groupId>log4j</groupId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>4.0.1</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>4.0.1</version>
</dependency>
<!-- Dubbo zookeeper registry dependency end -->
<!-- apache shenyu  apache dubbo plugin end-->
```

如果客户端是 alibaba dubbo，注册中心使用 zookeeper，请参考如下配置：

```
<!-- apache shenyu alibaba dubbo plugin start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-alibaba-dubbo</artifactId>
    <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
```

```
    <artifactId>dubbo</artifactId>
    <version>${alibaba.dubbo.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-client</artifactId>
    <version>${curator.version}</version>
    <exclusions>
        <exclusion>
            <artifactId>log4j</artifactId>
            <groupId>log4j</groupId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>${curator.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>${curator.version}</version>
</dependency>
<!-- apache shenyu  alibaba dubbo plugin end-->
```

9.2 运行 shenyu-examples-dubbo 项目

下载 shenyu-examples-dubbo .

修改 spring-dubbo.xml 中的注册地址为你本地（注意区分 dubbo 的版本是 apache dubbo 还是 alibaba dubbo），如：

```
<dubbo:registry address="zookeeper://localhost:2181"/>
```

运行相应的 main 方法启动项目， （注意区分 dubbo 的版本是 apache dubbo 还是 alibaba dubbo）。

成功启动会有如下日志：

```
2021-02-06 20:58:01.807  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/insert","pathDesc":"Insert a row of data","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboTestService",
"methodName":"insert","ruleName":"/dubbo/insert","parameterTypes":"org.dromara.
shenyu.examples.dubbo.api.entity.DubboTest","rpcExt":"{\"group\":\"\",\"version\":\
"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\":\"\"}",
"enabled":true}
```

```
2021-02-06 20:58:01.821  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findAll","pathDesc":"Get all data","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboTestService",
"methodName":"findAll","ruleName":"/dubbo/findAll","parameterTypes":"","rpcExt":"{\
"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\
":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.833  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findById","pathDesc":"Query by Id","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboTestService",
"methodName":"findById","ruleName":"/dubbo/findById","parameterTypes":"java.lang.
String","rpcExt":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\
"retries\":2,\"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.844  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByListId","pathDesc":"","rpcType":"dubbo","serviceName":
"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService","methodName
":"findByListId","ruleName":"/dubbo/findByListId","parameterTypes":"java.util.List
","rpcExt":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\
":2,\"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.855  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByIdsAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","methodName":"findByIdsAndName","ruleName":"/dubbo/findByIdsAndName",
"parameterTypes":"java.util.List,java.lang.String","rpcExt":"{\"group\":\"\",\
"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\
":\"\"}","enabled":true}
2021-02-06 20:58:01.866  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/batchSave","pathDesc":"","rpcType":"dubbo","serviceName":
"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService","methodName
":"batchSave","ruleName":"/dubbo/batchSave","parameterTypes":"java.util.List",
"rpcExt":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\
":2,\"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.876  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByArrayIdsAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","methodName":"findByArrayIdsAndName","ruleName":"/dubbo/findByArrayIdsAndName",
"parameterTypes":"[Ljava.lang.Integer;,java.lang.String","rpcExt":"{\"group\":\"\",
\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\
":\"\"}","enabled":true}
2021-02-06 20:58:01.889  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/saveComplexBeanTestAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","methodName":"saveComplexBeanTestAndName","ruleName":"/dubbo/
saveComplexBeanTestAndName","parameterTypes":"org.dromara.shenyu.examples.dubbo.
api.entity.ComplexBeanTest,java.lang.String","rpcExt":"{\"group\":\"\",\"version\
":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\":\"\"}",
"enabled":true}
```

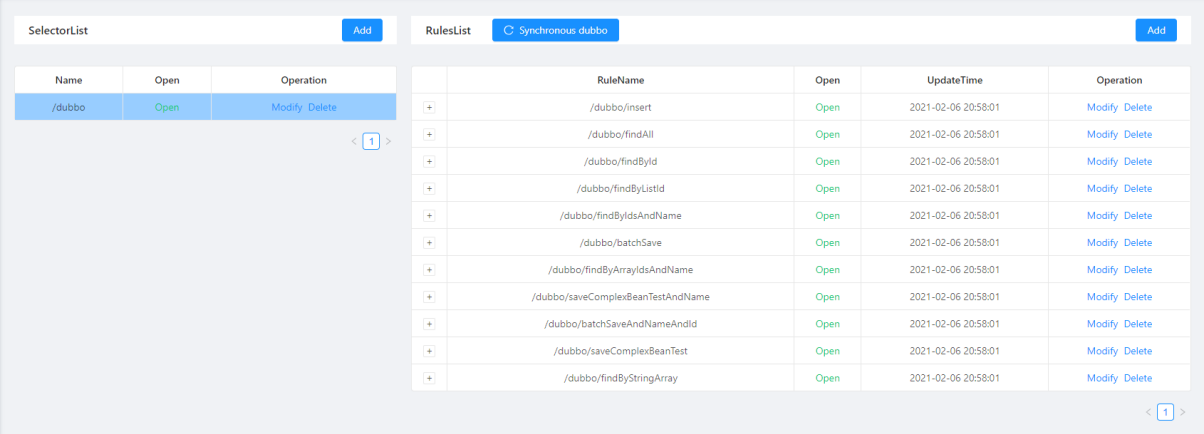
```
2021-02-06 20:58:01.901  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/batchSaveAndNameAndId","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","methodName":"batchSaveAndNameAndId","ruleName":"/dubbo/batchSaveAndNameAndId",
"parameterTypes":"java.util.List,java.lang.String,java.lang.String","rpcExt":"{\
"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\
":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.911  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/saveComplexBeanTest","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","methodName":"saveComplexBeanTest","ruleName":"/dubbo/saveComplexBeanTest",
"parameterTypes":"org.dromara.shenyu.examples.dubbo.api.entity.ComplexBeanTest",
"rpcExt":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\
":2,\"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.922  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByStringArray","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","methodName":"findByStringArray","ruleName":"/dubbo/findByStringArray",
"parameterTypes":"[Ljava.lang.String;","rpcExt":"{\"group\":\"\",\"version\":\"\",\
"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\":\"\"}","enabled
":true}
```

注意：当您需要同时暴露多个协议时，请不要配置 `shenyu.client.dubbo.props.port`。

9.3 测试

`shenyu-examples-dubbo` 项目成功启动之后会自动把加 `@ShenyuDubboClient` 注解的接口方法注册到网关。

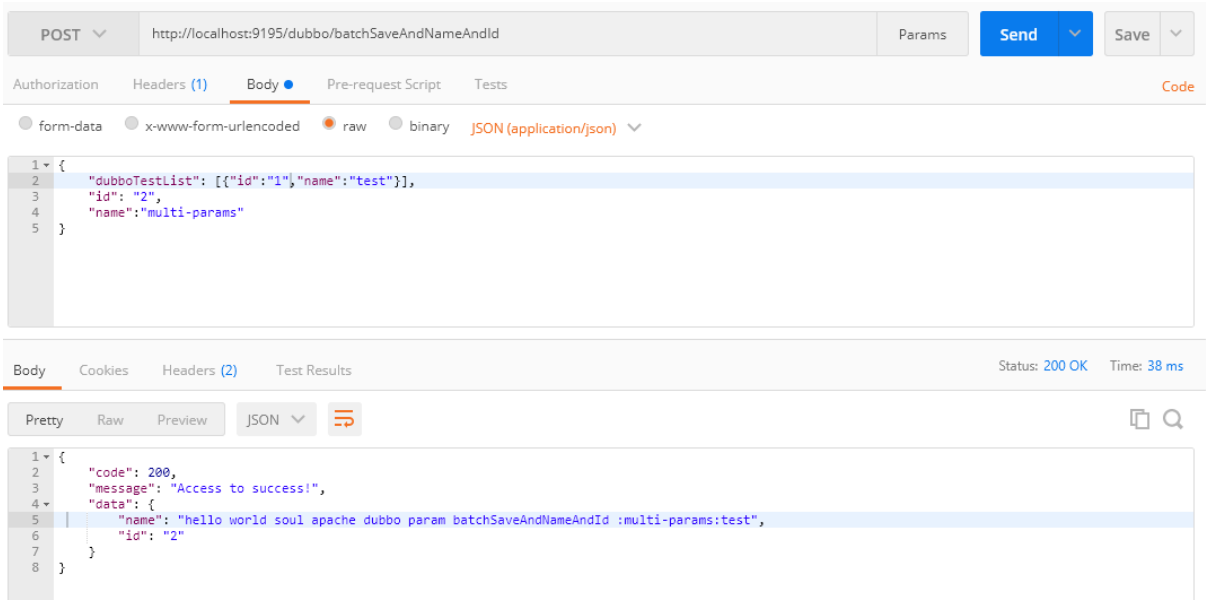
打开插件列表 -> `rpc proxy` -> `dubbo` 可以看到插件规则配置列表：



下面使用 `postman` 模拟 `http` 的方式来请求你的 `dubbo` 服务：

复杂多参数示例：对应接口实现类为 org.apache.shenyu.examples.alibaba.dubbo.service.impl.DubboMultiParamServiceImpl#batchSaveAndNameAndId

```
@Override
@ShenyuDubboClient(path = "/batchSaveAndNameAndId")
public DubboTest batchSaveAndNameAndId(List<DubboTest> dubboTestList, String id,
String name) {
    DubboTest test = new DubboTest();
    test.setId(id);
    test.setName("hello world shenyu alibaba dubbo param batchSaveAndNameAndId :" +
name + ":" + dubboTestList.stream().map(DubboTest::getName).collect(Collectors.
joining("-")));
    return test;
}
```



当你的参数不匹配时会报如下异常：

```
2021-02-07 22:24:04.015 ERROR 14860 --- [:20888-thread-3] o.d.shenyu.web.handler.
GlobalErrorHandler  : [e47b2a2a] Resolved [ShenyuException: org.apache.dubbo.
remoting.RemotingException: java.lang.IllegalArgumentException: args.length !=
types.length
```

```
        java.lang.IllegalArgumentException: args.length != types.length
        at org.apache.dubbo.common.utils.PojoUtils.realize(PojoUtils.java:91)
        at org.apache.dubbo.rpc.filter.GenericFilter.invoke(GenericFilter.java:82)
        at org.apache.dubbo.rpc.protocol.ProtocolFilterWrapper$1.
invoke(ProtocolFilterWrapper.java:81)
        at org.apache.dubbo.rpc.filter.ClassLoaderFilter.invoke(ClassLoaderFilter.
java:38)
        at org.apache.dubbo.rpc.protocol.ProtocolFilterWrapper$1.
invoke(ProtocolFilterWrapper.java:81)
        at org.apache.dubbo.rpc.filter.EchoFilter.invoke(EchoFilter.java:41)
        at org.apache.dubbo.rpc.protocol.ProtocolFilterWrapper$1.
invoke(ProtocolFilterWrapper.java:81)
        at org.apache.dubbo.rpc.protocol.dubbo.DubboProtocol$1.reply(DubboProtocol.
java:150)
        at org.apache.dubbo.remoting.exchange.support.header.HeaderExchangeHandler.
handleRequest(HeaderExchangeHandler.java:100)
        at org.apache.dubbo.remoting.exchange.support.header.HeaderExchangeHandler.
received(HeaderExchangeHandler.java:175)
        at org.apache.dubbo.remoting.transport.DecodeHandler.
received(DecodeHandler.java:51)
        at org.apache.dubbo.remoting.transport.dispatcher.ChannelEventRunnable.
run(ChannelEventRunnable.java:57)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.
java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.
java:624)
        at java.lang.Thread.run(Thread.java:748)
        ] for HTTP POST /dubbo/batchSaveAndNameAndId
```

本文档演示如何将 Spring Cloud 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的示例代码 。

9.4 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 springCloud 插件设置为开启。

启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。

引入网关对 Spring Cloud 的代理插件，并添加相关注册中心依赖:

```
<!-- apache shenyu springCloud plugin start-->
                <dependency>
                     <groupId>org.apache.shenyu</groupId>
```

```
                    <artifactId>shenyu-spring-boot-starter-plugin-springcloud</
artifactId>
                    <version>${project.version}</version>
                </dependency>

                <dependency>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-commons</artifactId>
                    <version>2.2.0.RELEASE</version>
                </dependency>

                <dependency>
                    <groupId>org.apache.shenyu</groupId>
                    <artifactId>shenyu-spring-boot-starter-plugin-httpclient</
artifactId>
                    <version>${project.version}</version>
                </dependency>
        <!-- springCloud if you config register center is eureka please dependency
end-->
                <dependency>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-starter-netflix-eureka-client</
artifactId>
                    <version>2.2.0.RELEASE</version>
                </dependency>
        <!-- apache shenyu springCloud plugin end-->
```

eureka 配置信息如下:

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
```

特别注意: 请保证 springCloud 注册中心服务发现配置为开启

- 配置方式

```
spring:
  cloud:
    discovery:
      enabled: true
```

- 代码方式

```
@SpringBootApplication
@EnableDiscoveryClient
```

```
public class ShenyuBootstrapApplication {

    /**
     * Main Entrance.
     *
     * @param args startup arguments
     */
    public static void main(final String[] args) {
        SpringApplication.run(ShenyuBootstrapApplication.class, args);
    }
}
```

启动 shenyu-bootstrap 项目。

9.5 运行 shenyu-examples-springcloud

示例项目中我们使用 eureka 作为 Spring Cloud 的注册中心。你可以使用本地的 eureka，也可以使用示例中提供的应用。

下载 shenyu-examples-eureka 、shenyu-examples-springcloud .

启动 eureka 服务，运行 org.apache.shenyu.examples.eureka.EurekaServerApplicationmain 方法启动项目。

启动 spring cloud 服务，运行 org.apache.shenyu.examples.springcloud.ShenyuTestSpringCloudApplicationmain 方法启动项目。

从 2.4.3 开始，用户可以不配置 shenyu.client.springCloud.props.port。

成功启动会有如下日志:

```
2021-02-10 14:03:51.301  INFO 2860 --- [           main] o.s.s.concurrent.
ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2021-02-10 14:03:51.669  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/save","pathDesc":"","rpcType
":"springCloud","ruleName":"/springcloud/order/save","enabled":true}
2021-02-10 14:03:51.676  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/path/**","pathDesc":"",
"rpcType":"springCloud","ruleName":"/springcloud/order/path/**","enabled":true}
2021-02-10 14:03:51.682  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/findById","pathDesc":"",
"rpcType":"springCloud","ruleName":"/springcloud/order/findById","enabled":true}
2021-02-10 14:03:51.688  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/path/**/name","pathDesc":"",
"rpcType":"springCloud","ruleName":"/springcloud/order/path/**/name","enabled
":true}
```

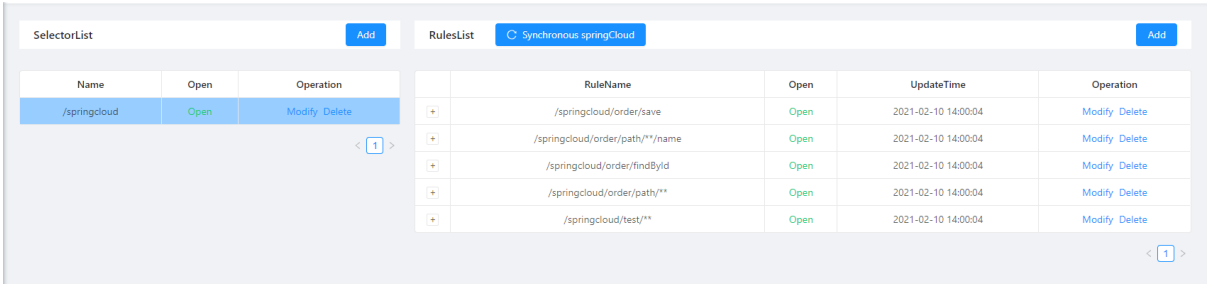
```
2021-02-10 14:03:51.692  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/test/**","pathDesc":"","rpcType":
"springCloud","ruleName":"/springcloud/test/**","enabled":true}
2021-02-10 14:03:52.806  WARN 2860 --- [           main]
ockingLoadBalancerClientRibbonWarnLogger : You already have
RibbonLoadBalancerClient on your classpath. It will be used by default. As Spring
Cloud Ribbon is in maintenance mode. We recommend switching to
BlockingLoadBalancerClient instead. In order to use it, set the value of `spring.
cloud.loadbalancer.ribbon.enabled` to `false` or remove spring-cloud-starter-
netflix-ribbon from your project.
2021-02-10 14:03:52.848  WARN 2860 --- [           main] iguration
$LoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently working
with default default cache. You can switch to using Caffeine cache, by adding it to
the classpath.
2021-02-10 14:03:52.921  INFO 2860 --- [           main] o.s.c.n.eureka.
InstanceInfoFactory       : Setting initial instance status as: STARTING
2021-02-10 14:03:52.949  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Initializing Eureka in region us-east-1
2021-02-10 14:03:53.006  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using JSON encoding codec LegacyJacksonJson
2021-02-10 14:03:53.006  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using JSON decoding codec LegacyJacksonJson
2021-02-10 14:03:53.110  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using XML encoding codec XStreamXml
2021-02-10 14:03:53.110  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using XML decoding codec XStreamXml
2021-02-10 14:03:53.263  INFO 2860 --- [           main] c.n.d.s.r.aws.
ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-02-10 14:03:53.546  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Disable delta property : false
2021-02-10 14:03:53.546  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Single vip registry refresh property : null
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Force full registry fetch : false
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Application is null : false
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Registered Applications size is zero : true
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Application version is -1: true
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Getting all instance registry info from the eureka server
2021-02-10 14:03:53.754  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : The response status is 200
2021-02-10 14:03:53.756  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Starting heartbeat executor: renew interval is: 30
2021-02-10 14:03:53.758  INFO 2860 --- [           main] c.n.discovery.
InstanceInfoReplicator     : InstanceInfoReplicator onDemand update allowed rate
per min is 4
```

```
2021-02-10 14:03:53.761  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Discovery Client initialized at timestamp 1612937033760 with
initial instances count: 0
2021-02-10 14:03:53.762  INFO 2860 --- [           main] o.s.c.n.e.s.
EurekaServiceRegistry        : Registering application SPRINGCLOUD-TEST with eureka
with status UP
2021-02-10 14:03:53.763  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Saw local status change event StatusChangeEvent
[timestamp=1612937033763, current=UP, previous=STARTING]
2021-02-10 14:03:53.765  INFO 2860 --- [nfoReplicator-0] com.netflix.discovery.
DiscoveryClient    : DiscoveryClient_SPRINGCLOUD-TEST/host.docker.
internal:springCloud-test:8884: registering service...
2021-02-10 14:03:53.805  INFO 2860 --- [           main] o.s.b.w.embedded.tomcat.
TomcatWebServer  : Tomcat started on port(s): 8884 (http) with context path ''
2021-02-10 14:03:53.807  INFO 2860 --- [           main] .s.c.n.e.s.
EurekaAutoServiceRegistration : Updating port to 8884
2021-02-10 14:03:53.837  INFO 2860 --- [nfoReplicator-0] com.netflix.discovery.
DiscoveryClient    : DiscoveryClient_SPRINGCLOUD-TEST/host.docker.
internal:springCloud-test:8884 - registration status: 204
2021-02-10 14:03:54.231  INFO 2860 --- [           main] o.d.s.e.s.
ShenyuTestSpringCloudApplication : Started ShenyuTestSpringCloudApplication in 6.
338 seconds (JVM running for 7.361)
```

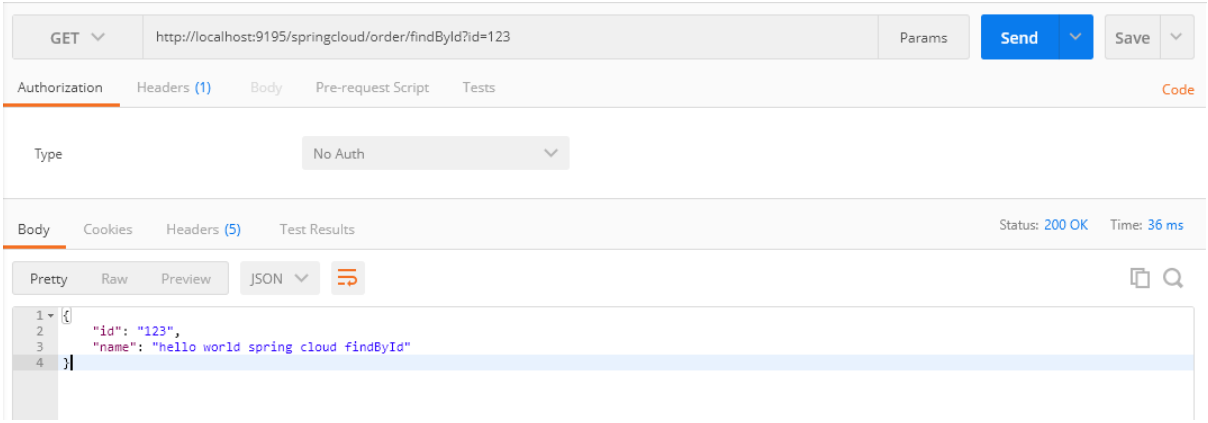
9.6 测试 Http 请求

shenyu-examples-springcloud 项目成功启动之后会自动把加 @ShenyuSpringCloudClient 注解的接口方法注册到网关。

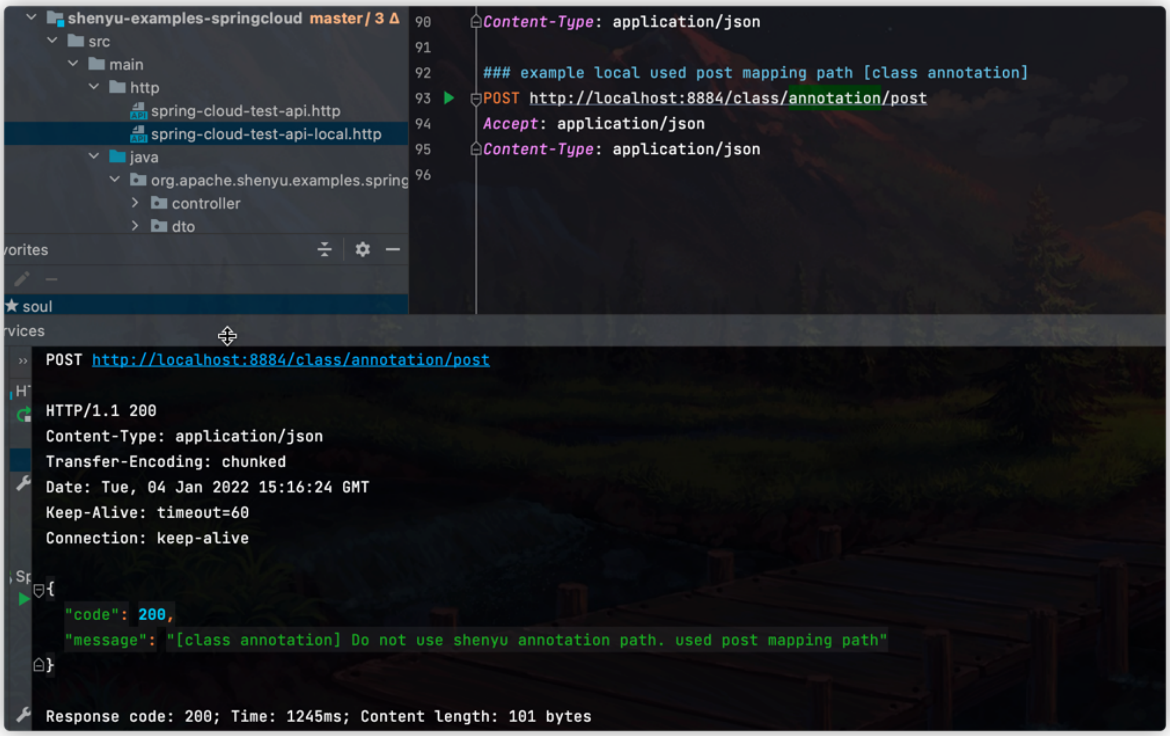
打开插件列表 -> rpc proxy -> springCloud 可以看到插件规则配置列表:



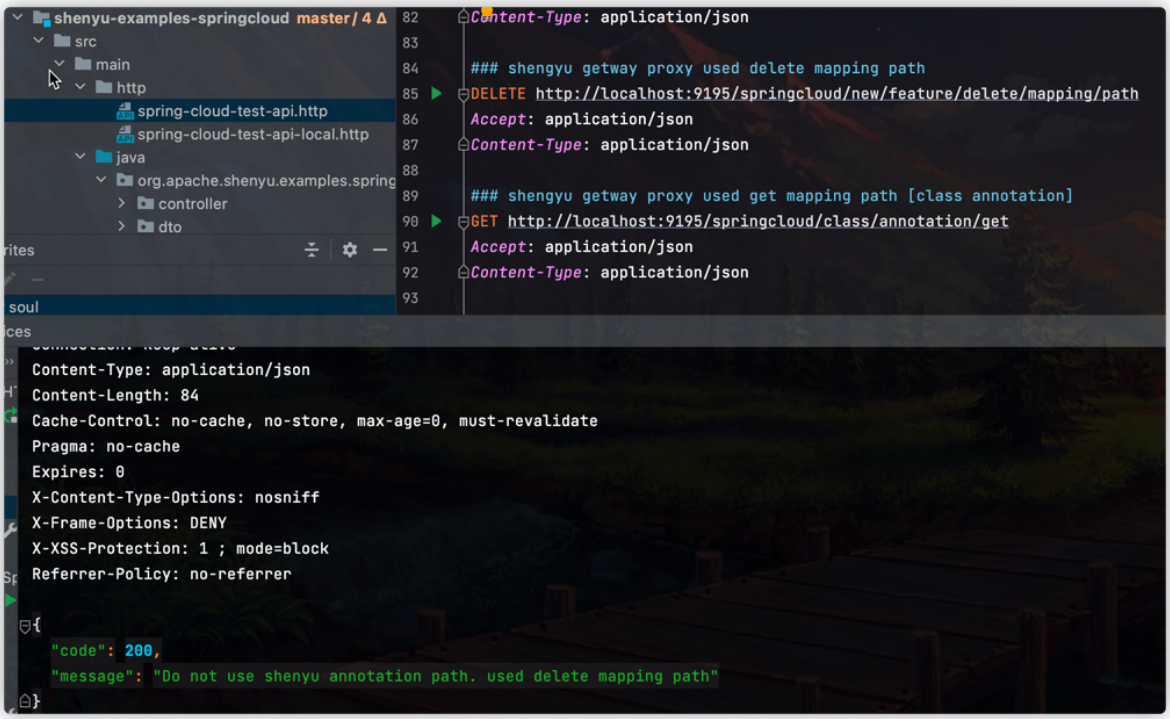
下面使用 postman 模拟 http 的方式来请求你的 SpringCloud 服务:



使用 IDEA HTTP Client 插件模拟 http 的方式来请求你的 SpringCloud 服务 [本地访问，不使用 shenyu 代理]:



使用 IDEA HTTP Client 插件模拟 http 的方式来请求你的 SpringCloud 服务 [使用 shenyu 代理]:



本文档演示如何将 Sofa 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的示例代码 。

9.7 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 sofa 插件设置为开启，并设置你的注册地址，请确保注册中心在你本地已经开启。

启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。

如果客户端是 sofa，注册中心使用 zookeeper，请参考如下配置：

```
<!-- apache shenyu sofa plugin start-->
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-rpc-all</artifactId>
    <version>5.7.6</version>
</dependency>
<dependency>
       <groupId>org.apache.curator</groupId>
       <artifactId>curator-client</artifactId>
       <version>4.0.1</version>
```

```
   </dependency>
   <dependency>
       <groupId>org.apache.curator</groupId>
       <artifactId>curator-framework</artifactId>
       <version>4.0.1</version>
   </dependency>
   <dependency>
       <groupId>org.apache.curator</groupId>
       <artifactId>curator-recipes</artifactId>
       <version>4.0.1</version>
   </dependency>

<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-sofa</artifactId>
    <version>${project.version}</version>
</dependency>
<!-- apache shenyu sofa plugin end-->
```

9.8 运行 shenyu-examples-sofa 项目

下载 shenyu-examples-sofa

设置 `application.yml` 的 `zk` 注册地址，如：

```
com:
  alipay:
    sofa:
      rpc:
        registry-address: zookeeper://127.0.0.1:2181
```

运行 `org.apache.shenyu.examples.sofa.service.TestSofaApplicationmain` 方法启动 sofa 服务。

成功启动会有如下日志：

```
2021-02-10 02:31:45.599  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/insert","pathDesc":"Insert a row of data","rpcType":"sofa",
"serviceName":"org.dromara.shenyu.examples.sofa.api.service.SofaSingleParamService
","methodName":"insert","ruleName":"/sofa/insert","parameterTypes":"org.dromara.
shenyu.examples.sofa.api.entity.SofaSimpleTypeBean","rpcExt":"{\"loadbalance\":\
"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.605  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/findById","pathDesc":"Find by Id","rpcType":"sofa","serviceName
":"org.dromara.shenyu.examples.sofa.api.service.SofaSingleParamService","methodName
":"findById","ruleName":"/sofa/findById","parameterTypes":"java.lang.String",
"rpcExt":"{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
```

```
2021-02-10 02:31:45.611  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/findAll","pathDesc":"Get all data","rpcType":"sofa",
"serviceName":"org.dromara.shenyu.examples.sofa.api.service.SofaSingleParamService
","methodName":"findAll","ruleName":"/sofa/findAll","parameterTypes":"","rpcExt":"
{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.616  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/batchSaveNameAndId","pathDesc":"","rpcType":"sofa","serviceName
":"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName
":"batchSaveNameAndId","ruleName":"/sofa/batchSaveNameAndId","parameterTypes":
"java.util.List,java.lang.String,java.lang.String#org.dromara.shenyu.examples.sofa.
api.entity.SofaSimpleTypeBean","rpcExt":"{\"loadbalance\":\"hash\",\"retries\":3,\
"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.621  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/saveComplexBeanAndName","pathDesc":"","rpcType":"sofa",
"serviceName":"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService",
"methodName":"saveComplexBeanAndName","ruleName":"/sofa/saveComplexBeanAndName",
"parameterTypes":"org.dromara.shenyu.examples.sofa.api.entity.SofaComplexTypeBean,
java.lang.String","rpcExt":"{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\":-1}
","enabled":true}
2021-02-10 02:31:45.627  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/findByArrayIdsAndName","pathDesc":"","rpcType":"sofa",
"serviceName":"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService",
"methodName":"findByArrayIdsAndName","ruleName":"/sofa/findByArrayIdsAndName",
"parameterTypes":"[Ljava.lang.Integer;,java.lang.String","rpcExt":"{\"loadbalance\
":\"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.632  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/findByStringArray","pathDesc":"","rpcType":"sofa","serviceName
":"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName
":"findByStringArray","ruleName":"/sofa/findByStringArray","parameterTypes":
"[Ljava.lang.String;","rpcExt":"{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\
":-1}","enabled":true}
2021-02-10 02:31:45.637  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/saveTwoList","pathDesc":"","rpcType":"sofa","serviceName":"org.
dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName":
"saveTwoList","ruleName":"/sofa/saveTwoList","parameterTypes":"java.util.List,java.
util.Map#org.dromara.shenyu.examples.sofa.api.entity.SofaComplexTypeBean","rpcExt":
"{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.642  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/batchSave","pathDesc":"","rpcType":"sofa","serviceName":"org.
dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName":
"batchSave","ruleName":"/sofa/batchSave","parameterTypes":"java.util.List#org.
dromara.shenyu.examples.sofa.api.entity.SofaSimpleTypeBean","rpcExt":"{\
"loadbalance\":\"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
```

```
2021-02-10 02:31:45.647  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/findByListId","pathDesc":"","rpcType":"sofa","serviceName":
"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName":
"findByListId","ruleName":"/sofa/findByListId","parameterTypes":"java.util.List",
"rpcExt":"{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.653  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/saveComplexBean","pathDesc":"","rpcType":"sofa","serviceName":
"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName":
"saveComplexBean","ruleName":"/sofa/saveComplexBean","parameterTypes":"org.dromara.
shenyu.examples.sofa.api.entity.SofaComplexTypeBean","rpcExt":"{\"loadbalance\":\
"hash\",\"retries\":3,\"timeout\":-1}","enabled":true}
2021-02-10 02:31:45.660  INFO 2156 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : sofa client register success: {"appName":"sofa","contextPath":"/
sofa","path":"/sofa/findByIdsAndName","pathDesc":"","rpcType":"sofa","serviceName":
"org.dromara.shenyu.examples.sofa.api.service.SofaMultiParamService","methodName":
"findByIdsAndName","ruleName":"/sofa/findByIdsAndName","parameterTypes":"java.util.
List,java.lang.String","rpcExt":"{\"loadbalance\":\"hash\",\"retries\":3,\"timeout\
":-1}","enabled":true}
2021-02-10 02:31:46.055  INFO 2156 --- [           main] o.a.c.f.imps.
CuratorFrameworkImpl        : Starting
2021-02-10 02:31:46.059  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:zookeeper.version=3.4.6-1569965, built on
02/20/2014 09:09 GMT
2021-02-10 02:31:46.059  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:host.name=host.docker.internal
2021-02-10 02:31:46.059  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.version=1.8.0_211
2021-02-10 02:31:46.059  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.vendor=Oracle Corporation
2021-02-10 02:31:46.059  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.home=C:\Program Files\Java\jdk1.8.0_
211\jre
2021-02-10 02:31:46.059  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.class.path=C:\Program Files\Java\
jdk1.8.0_211\jre\lib\charsets.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\
deploy.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\ext\access-bridge-64.jar;C:\
Program Files\Java\jdk1.8.0_211\jre\lib\ext\cldrdata.jar;C:\Program Files\Java\
jdk1.8.0_211\jre\lib\ext\dnsns.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\ext\
jaccess.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\ext\jfxrt.jar;C:\Program
Files\Java\jdk1.8.0_211\jre\lib\ext\localedata.jar;C:\Program Files\Java\jdk1.8.0_
211\jre\lib\ext\nashorn.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\ext\sunec.
jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\ext\sunjce_provider.jar;C:\Program
Files\Java\jdk1.8.0_211\jre\lib\ext\sunmscapi.jar;C:\Program Files\Java\jdk1.8.0_
211\jre\lib\ext\sunpkcs11.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\ext\zipfs.
jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\javaws.jar;C:\Program Files\Java\
jdk1.8.0_211\jre\lib\jce.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\jfr.jar;C:\
Program Files\Java\jdk1.8.0_211\jre\lib\jfxswt.jar;C:\Program Files\Java\jdk1.8.0_
211\jre\lib\jsse.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\management-agent.
jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\plugin.jar;C:\Program Files\Java\
jdk1.8.0_211\jre\lib\resources.jar;C:\Program Files\Java\jdk1.8.0_211\jre\lib\rt.
jar;D:\X\dlm_github\shenyu\shenyu-examples\shenyu-examples-sofa\shenyu-examples-
sofa-service\target\classes;D:\SOFT\m2\repository\com\alipay\sofa\rpc-sofa-boot-
```

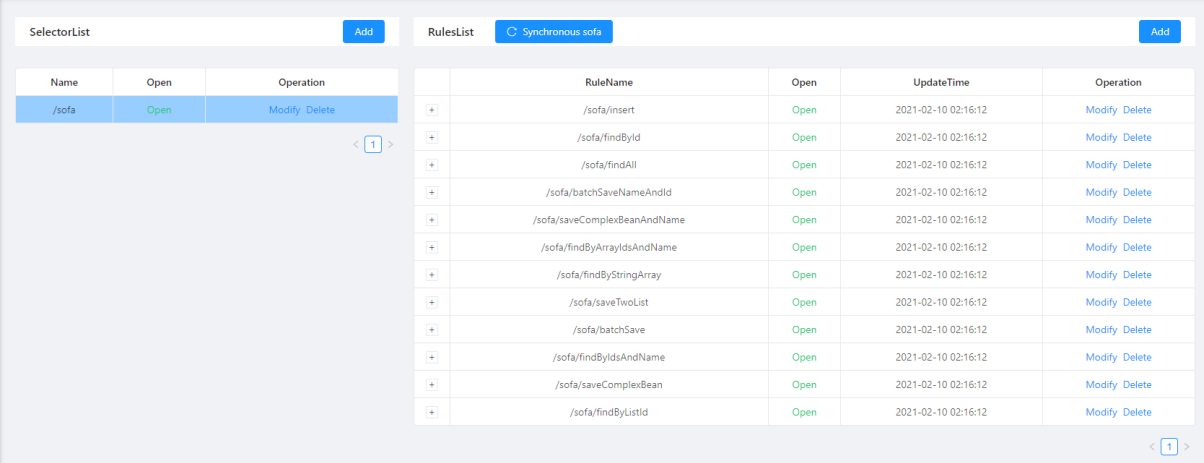
```
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.library.path=C:\Program Files\Java\
jdk1.8.0_211\bin;C:\Windows\Sun\Java\bin;C:\Windows\system32;C:\Windows;C:\Program
Files\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\
Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Windows\system32;C:\
Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\
Windows\System32\OpenSSH\;C:\Program Files\Java\jdk1.8.0_211\bin;C:\Program Files\
Java\jdk1.8.0_211\jre\bin;D:\SOFT\apache-maven-3.5.0\bin;C:\Program Files\Go\bin;
C:\Program Files\nodejs\;C:\Program Files\Python\Python38\;C:\Program Files\
OpenSSL-Win64\bin;C:\Program Files\Git\bin;D:\SOFT\protobuf-2.5.0\src;D:\SOFT\zlib-
1.2.8;c:\Program Files (x86)\Microsoft SQL Server\100\Tools\Binn\;c:\Program Files\
Microsoft SQL Server\100\Tools\Binn\;c:\Program Files\Microsoft SQL Server\100\DTS\
Binn\;C:\Program Files\Docker\Docker\resources\bin;C:\ProgramData\DockerDesktop\
version-bin;D:\SOFT\gradle-6.0-all\gradle-6.0\bin;C:\Program Files\mingw-w64\x86_
64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin;D:\SOFT\hugo_extended_0.55.5_Windows-
64bit;C:\Users\DLM\AppData\Local\Microsoft\WindowsApps;C:\Users\DLM\go\bin;C:\
Users\DLM\AppData\Roaming\npm;;C:\Program Files\Microsoft VS Code\bin;C:\Program
Files\nimbella-cli\bin;.
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.io.tmpdir=C:\Users\DLM\AppData\Local\
Temp\
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.compiler=<NA>
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:os.name=Windows 10
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:os.arch=amd64
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:os.version=10.0
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:user.name=DLM
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:user.home=C:\Users\DLM
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:user.dir=D:\X\dlm_github\shenyu
2021-02-10 02:31:46.061  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Initiating client connection, connectString=127.0.0.1:21810
sessionTimeout=60000 watcher=org.apache.curator.ConnectionState@3e850122
2021-02-10 02:31:46.069  INFO 2156 --- [27.0.0.1:21810)] org.apache.zookeeper.
ClientCnxn          : Opening socket connection to server 127.0.0.1/127.0.0.
1:21810. Will not attempt to authenticate using SASL (unknown error)
2021-02-10 02:31:46.071  INFO 2156 --- [27.0.0.1:21810)] org.apache.zookeeper.
ClientCnxn          : Socket connection established to 127.0.0.1/127.0.0.1:21810,
initiating session
2021-02-10 02:31:46.078  INFO 2156 --- [27.0.0.1:21810)] org.apache.zookeeper.
ClientCnxn          : Session establishment complete on server 127.0.0.1/127.0.0.
1:21810, sessionid = 0x10005b0d05e0001, negotiated timeout = 40000
2021-02-10 02:31:46.081  INFO 2156 --- [ain-EventThread] o.a.c.f.state.
ConnectionStateManager     : State change: CONNECTED
```

```
2021-02-10 02:31:46.093  WARN 2156 --- [           main] org.apache.curator.utils.
ZKPaths          : The version of ZooKeeper being used doesn't support Container
nodes. CreateMode.PERSISTENT will be used instead.
2021-02-10 02:31:46.141  INFO 2156 --- [           main] o.d.s.e.s.service.
TestSofaApplication     : Started TestSofaApplication in 3.41 seconds (JVM running
for 4.423)
```

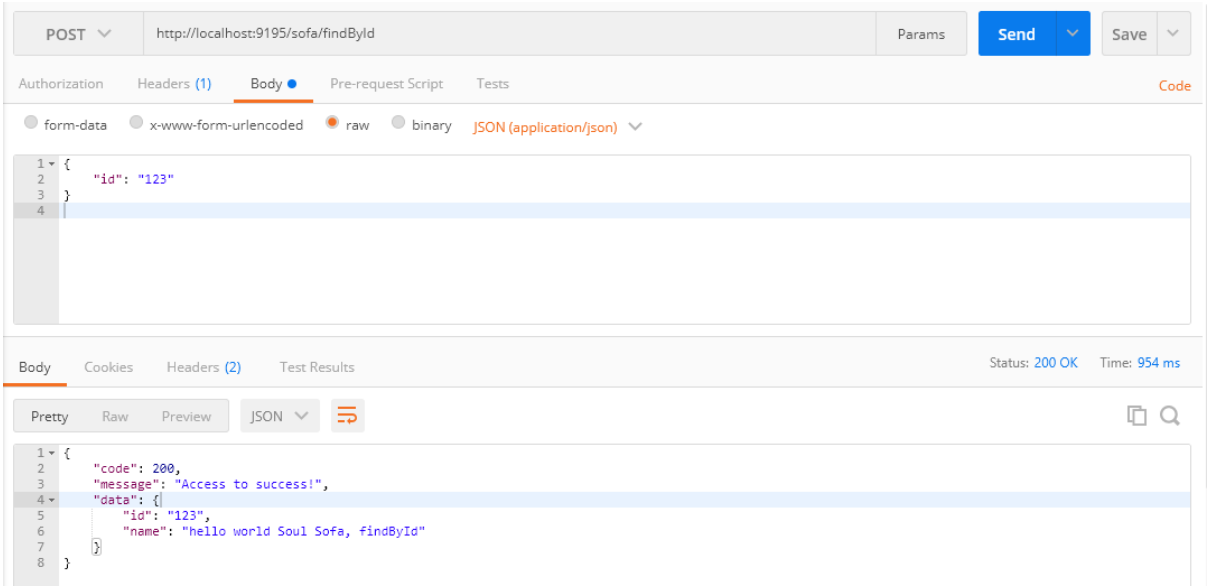
9.9 测试

shenyu-examples-sofa 项目成功启动之后会自动把加 @ShenyuSofaClient 注解的接口方法注册到网关。

打开插件列表 -> rpc proxy -> sofa 可以看到插件规则配置列表:

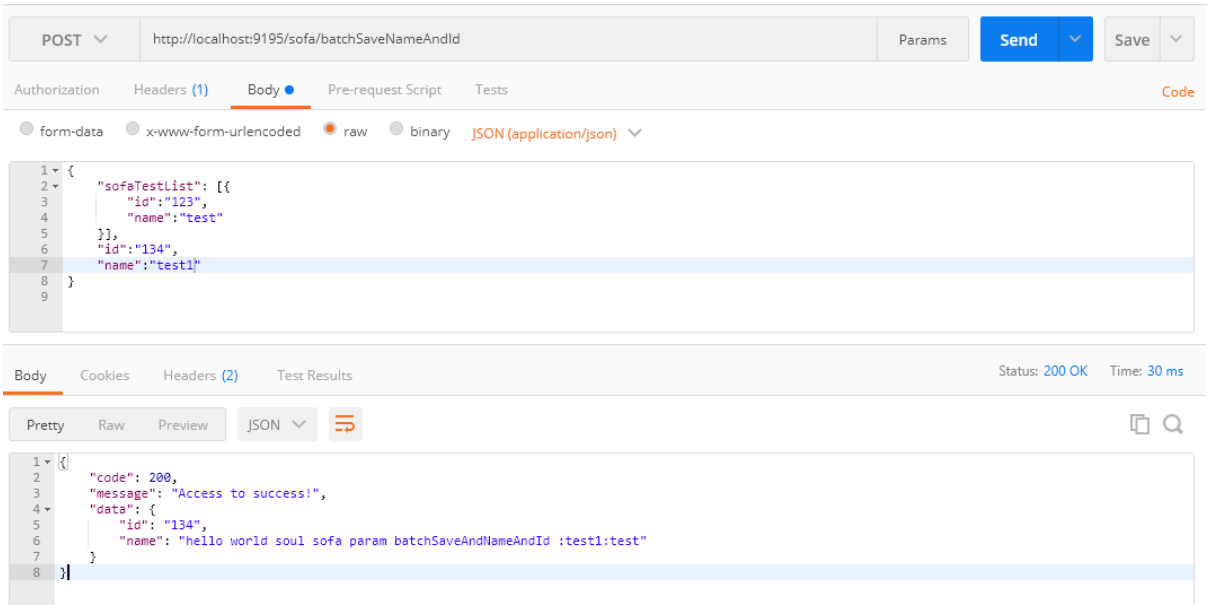


下面使用 postman 模拟 http 的方式来请求你的 sofa 服务:



复杂多参数示例: 对应接口实现类为 org.apache.shenyu.examples.sofa.service.impl.SofaMultiParamServiceImpl#batchSaveNameAndId

```
@Override
@ShenyuSofaClient(path = "/batchSaveNameAndId")
public SofaSimpleTypeBean batchSaveNameAndId(final List<SofaSimpleTypeBean>
sofaTestList, final String id, final String name) {
        SofaSimpleTypeBean simpleTypeBean = new SofaSimpleTypeBean();
        simpleTypeBean.setId(id);
        simpleTypeBean.setName("hello world shenyu sofa param batchSaveAndNameAndId
:" + name + ":" + sofaTestList.stream().map(SofaSimpleTypeBean::getName).
collect(Collectors.joining("-")));
        return simpleTypeBean;
        }
```



本文档演示如何将 gRPC 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的 示例代码 。

9.10 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 gRPC 插件设置为开启。

启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。

引入网关对 gRPC 的代理插件，在网关的 pom.xml 文件中增加如下依赖:

```
<!-- apache shenyu grpc plugin start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
```

```
    <artifactId>shenyu-spring-boot-starter-plugin-grpc</artifactId>
    <version>${project.version}</version>
</dependency>
<!-- apache shenyu grpc plugin end-->
```

9.11 运行 shenyu-examples-grpc 项目

下载 shenyu-examples-grpc

在 shenyu-examples-grpc 下执行以下命令生成 java 代码:

```
mvn protobuf:compile //编译消息对象
mvn protobuf:compile-custom //依赖消息对象，生成接口服务
```

或者, 如果你是通过 IntelliJ IDEA 打开 Apache ShenYu 工程, 首先在 maven root 下 install 整个项目然后在 Maven 工具栏中选中 protobuf:compile 和 protobuf:compile-custom, 然后右键 Run Maven Build 一键生成 proto 文件对应的 java 代码。然后让 idea 识别生成的 target 文件夹

运行 org.apache.shenyu.examples.grpc.ShenyuTestGrpcApplication 中的 main 方法启动项目。

成功启动会有如下日志，表示将 gRPC 服务成功注册到 shenyu-admin 中。

```
2021-06-18 19:33:32.866  INFO 11004 --- [or_consumer_-19] o.a.s.r.client.http.
utils.RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/clientStreamingFun","pathDesc":
"clientStreamingFun","rpcType":"grpc","serviceName":"stream.StreamService",
"methodName":"clientStreamingFun","ruleName":"/grpc/clientStreamingFun",
"parameterTypes":"io.grpc.stub.StreamObserver","rpcExt":"{\"timeout\":5000,\
"methodType\":\"CLIENT_STREAMING\"}","enabled":true,"host":"172.20.10.6","port
":8080,"registerMetaData":false}
2021-06-18 19:33:32.866  INFO 11004 --- [or_consumer_-17] o.a.s.r.client.http.
utils.RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/echo","pathDesc":"echo","rpcType":"grpc",
"serviceName":"echo.EchoService","methodName":"echo","ruleName":"/grpc/echo",
"parameterTypes":"echo.EchoRequest,io.grpc.stub.StreamObserver","rpcExt":"{\
"timeout\":5000,\"methodType\":\"UNARY\"}","enabled":true,"host":"172.20.10.6",
"port":8080,"registerMetaData":false}
2021-06-18 19:33:32.866  INFO 11004 --- [or_consumer_-20] o.a.s.r.client.http.
utils.RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/bidiStreamingFun","pathDesc":"bidiStreamingFun
","rpcType":"grpc","serviceName":"stream.StreamService","methodName":
"bidiStreamingFun","ruleName":"/grpc/bidiStreamingFun","parameterTypes":"io.grpc.
stub.StreamObserver","rpcExt":"{\"timeout\":5000,\"methodType\":\"BIDI_STREAMING\"}
","enabled":true,"host":"172.20.10.6","port":8080,"registerMetaData":false}
2021-06-18 19:33:32.866  INFO 11004 --- [or_consumer_-21] o.a.s.r.client.http.
utils.RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/unaryFun","pathDesc":"unaryFun","rpcType":"grpc
","serviceName":"stream.StreamService","methodName":"unaryFun","ruleName":"/grpc/
unaryFun","parameterTypes":"stream.RequestData,io.grpc.stub.StreamObserver","rpcExt
":"{\"timeout\":5000,\"methodType\":\"UNARY\"}","enabled":true,"host":"172.20.10.6
","port":8080,"registerMetaData":false}
```

```
2021-06-18 19:33:32.866  INFO 11004 --- [or_consumer_-18] o.a.s.r.client.http.
utils.RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/serverStreamingFun","pathDesc":
"serverStreamingFun","rpcType":"grpc","serviceName":"stream.StreamService",
"methodName":"serverStreamingFun","ruleName":"/grpc/serverStreamingFun",
"parameterTypes":"stream.RequestData,io.grpc.stub.StreamObserver","rpcExt":"{\
"timeout\":5000,\"methodType\":\"SERVER_STREAMING\"}","enabled":true,"host":"172.
20.10.6","port":8080,"registerMetaData":false}
```


9.12 简单测试

shenyu-examples-grpc 项目成功启动之后会自动把加 @ShenyuGrpcClient 注解的接口方法注册到网关。

打开 插件列表 -> rpc proxy -> grpc 可以看到插件规则配置列表。

下面使用 postman 模拟 http 的方式来请求你的 gRPC 服务。请求参数如下:

```
{
    "data": [
        {
            "message": "hello grpc"
        }
    ]
}
```

当前是以 json 的格式传递参数，key 的名称默认是 data，你可以在 GrpcConstants.JSON_DESCRIPTOR_PROTO_FIELD_NAME 中进行重置；value 的传入则根据你定义的 proto 文件。

9.13 流式调用

Apache ShenYu 可以支持 gRPC 的流式调用，下面展示的是 gRPC 四种方法类型的调用。在流式调用中，你可以通过数组的形式传递多个参数。

- UNARY

请求参数如下:

```
{
    "data": [
        {
            "text": "hello grpc"
        }
    ]
}
```

通过 postman 模拟 http 请求，发起 UNARY 调用。

- CLIENT_STREAMING

请求参数如下:

```
{
    "data": [
        {
            "text": "hello grpc"
        },
        {
            "text": "hello grpc"
        },
        {
            "text": "hello grpc"
        }
    ]
}
```

通过 postman 模拟 http 请求，发起 CLIENT_STREAMING 调用。

- SERVER_STREAMING

请求参数如下:

```
{
    "data": [
        {
            "text": "hello grpc"
        }
    ]
}
```

通过 postman 模拟 http 请求，发起 SERVER_STREAMING 调用。

- BIDI_STREAMING

请求参数如下:

```
{
    "data": [
        {
            "text": "hello grpc"
        },
        {
            "text": "hello grpc"
        },
        {
            "text": "hello grpc"
        }
    ]
```

```
}
```

通过 postman 模拟 http 请求，发起 BIDI_STREAMING 调用。

本文档演示如何将 Tars 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的示例代码 。

9.14 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 tars 插件设置为开启。

启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。

引入网关对 tars 的依赖:

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-tars</artifactId>
    <version>${project.version}</version>
</dependency>

<dependency>
    <groupId>com.tencent.tars</groupId>
    <artifactId>tars-client</artifactId>
    <version>1.7.2</version>
</dependency>
```

9.15 运行 shenyu-examples-tars 项目

下载shenyu-examples-tars

修改 application.yml 中的 host 为你本地 ip。

修改配置 src/main/resources/ShenyuExampleServer.ShenyuExampleApp.config.conf:

- 建议弄清楚 config 的主要配置项含义, 参考开发指南。
- config 中的 ip 要注意提供成本机的。
- local=..., 表示开放的本机给 tarsnode 连接的端口, 如果没有 tarsnode, 可以去掉这项配置。
- locator: registry 服务的地址，必须是有 ip 和 port 的，如果不需要 registry 来定位服务，则不需要配置。

- node=tars.tarsnode.ServerObj@xxxx，表示连接的 tarsnode 的地址，如果本地没有 tarsnode，这项配置可以去掉。

更多 config 配置说明请参考 Tars 官方文档

运行 org.apache.shenyu.examples.tars.ShenyuTestTarsApplicationmain 方法启动项目。

注:服务启动时需要在启动命令中指定配置文件地址 **-Dconfig=xxx/ShenyuExampleServer.ShenyuExampleApp.config.**

如果不加-Dconfig 参数配置会可能会如下抛异常:

```
com.qq.tars.server.config.ConfigurationException: error occurred on load server
config
    at com.qq.tars.server.config.ConfigurationManager.
loadServerConfig(ConfigurationManager.java:113)
    at com.qq.tars.server.config.ConfigurationManager.init(ConfigurationManager.
java:57)
    at com.qq.tars.server.core.Server.loadServerConfig(Server.java:90)
    at com.qq.tars.server.core.Server.<init>(Server.java:42)
    at com.qq.tars.server.core.Server.<clinit>(Server.java:38)
    at com.qq.tars.spring.bean.PropertiesListener.
onApplicationEvent(PropertiesListener.java:37)
    at com.qq.tars.spring.bean.PropertiesListener.
onApplicationEvent(PropertiesListener.java:31)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
doInvokeListener(SimpleApplicationEventMulticaster.java:172)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
invokeListener(SimpleApplicationEventMulticaster.java:165)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
multicastEvent(SimpleApplicationEventMulticaster.java:139)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
multicastEvent(SimpleApplicationEventMulticaster.java:127)
    at org.springframework.boot.context.event.EventPublishingRunListener.
environmentPrepared(EventPublishingRunListener.java:76)
    at org.springframework.boot.SpringApplicationRunListeners.
environmentPrepared(SpringApplicationRunListeners.java:53)
    at org.springframework.boot.SpringApplication.
prepareEnvironment(SpringApplication.java:345)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:308)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1226)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1215)
    at org.apache.shenyu.examples.tars.ShenyuTestTarsApplication.
main(ShenyuTestTarsApplication.java:38)
Caused by: java.lang.NullPointerException
    at java.io.FileInputStream.<init>(FileInputStream.java:130)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at com.qq.tars.common.util.Config.parseFile(Config.java:211)
    at com.qq.tars.server.config.ConfigurationManager.
loadServerConfig(ConfigurationManager.java:63)
    ... 17 more
```

```
The exception occurred at load server config
```

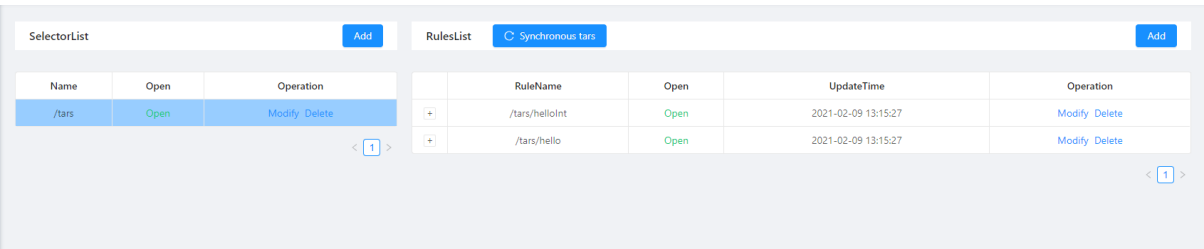
成功启动会有如下日志:

```
[SERVER] server starting at tcp -h 127.0.0.1 -p 21715 -t 60000...
[SERVER] server started at tcp -h 127.0.0.1 -p 21715 -t 60000...
[SERVER] server starting at tcp -h 127.0.0.1 -p 21714 -t 3000...
[SERVER] server started at tcp -h 127.0.0.1 -p 21714 -t 3000...
[SERVER] The application started successfully.
The session manager service started...
[SERVER] server is ready...
2021-02-09 13:28:24.643  INFO 16016 --- [           main] o.s.b.w.embedded.tomcat.
TomcatWebServer  : Tomcat started on port(s): 55290 (http) with context path ''
2021-02-09 13:28:24.645  INFO 16016 --- [           main] o.d.s.e.tars.
ShenyuTestTarsApplication     : Started ShenyuTestTarsApplication in 4.232 seconds
(JVM running for 5.1)
2021-02-09 13:28:24.828  INFO 16016 --- [pool-2-thread-1] o.d.s.client.common.
utils.RegisterUtils  : tars client register success: {"appName":"127.0.0.1:21715",
"contextPath":"/tars","path":"/tars/helloInt","pathDesc":"","rpcType":"tars",
"serviceName":"ShenyuExampleServer.ShenyuExampleApp.HelloObj","methodName":
"helloInt","ruleName":"/tars/helloInt","parameterTypes":"int,java.lang.String",
"rpcExt":"{\"methodInfo\":[{\"methodName\":\"helloInt\",\"params\":[{},{}],\
"returnType\":\"java.lang.Integer\"},{\"methodName\":\"hello\",\"params\":[{},{}],\
"returnType\":\"java.lang.String\"}]}","enabled":true}
2021-02-09 13:28:24.837  INFO 16016 --- [pool-2-thread-1] o.d.s.client.common.
utils.RegisterUtils  : tars client register success: {"appName":"127.0.0.1:21715",
"contextPath":"/tars","path":"/tars/hello","pathDesc":"","rpcType":"tars",
"serviceName":"ShenyuExampleServer.ShenyuExampleApp.HelloObj","methodName":"hello",
"ruleName":"/tars/hello","parameterTypes":"int,java.lang.String","rpcExt":"{\
"methodInfo\":[{\"methodName\":\"helloInt\",\"params\":[{},{}],\"returnType\":\
"java.lang.Integer\"},{\"methodName\":\"hello\",\"params\":[{},{}],\"returnType\":\
"java.lang.String\"}]}","enabled":true}
```

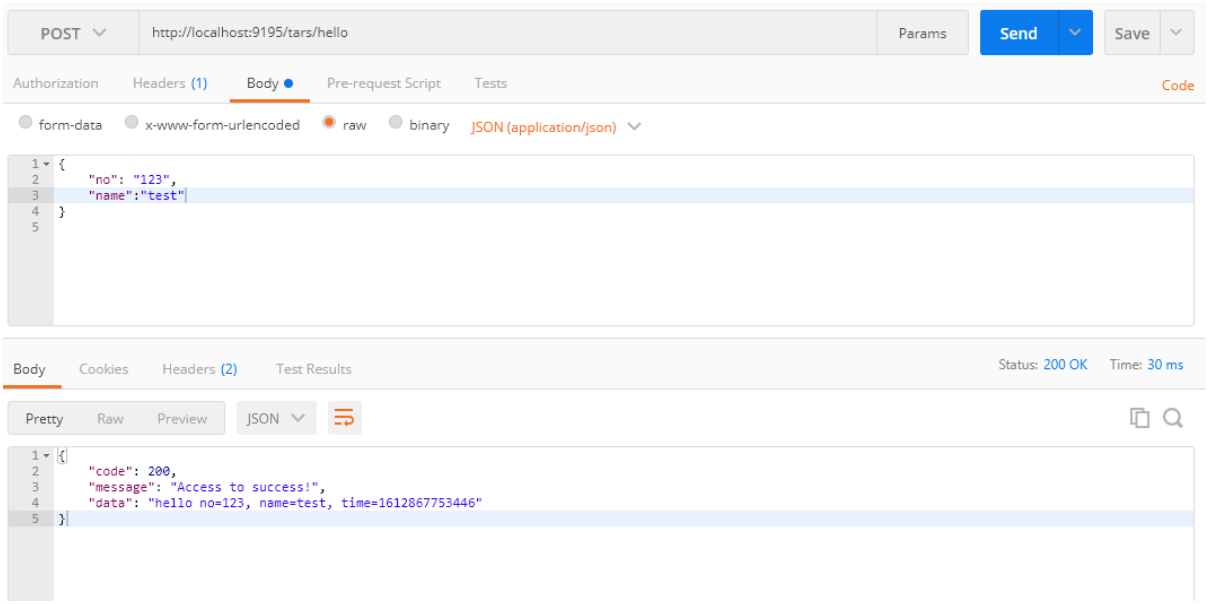
9.16 测试

shenyu-examples-tars 项目成功启动之后会自动把加 @ShenyuTarsClient 注解的接口方法注册到网关。

打开插件列表 -> rpc proxy -> tars 可以看到插件规则配置列表:



下面使用 postman 模拟 http 的方式来请求你的 tars 服务:



本文档演示如何将 Http 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的示例代码 。

9.17 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 divide 插件设置为开启。在 Apache ShenYu 网关中，Http 请求是由 divide 插件进行处理。

启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。

引入网关对 Http 的代理插件，在网关的 pom.xml 文件中增加如下依赖:

```
<!--if you use http proxy start this-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-divide</artifactId>
    <version>${project.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-httpclient</artifactId>
    <version>${project.version}</version>
</dependency>
```

9.18 运行 shenyu-examples-http 项目

下载 shenyu-examples-http

运行 `org.apache.shenyu.examples.http.ShenyuTestHttpApplicationmain` 方法启动项目。

从 `2.4.3` 开始，用户可以不配置 `shenyu.client.http.props.port`。

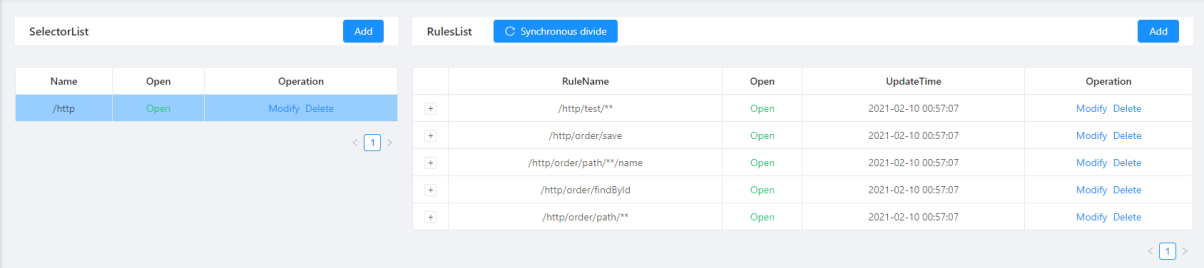
成功启动会有如下日志:

```
2021-02-10 00:57:07.561  INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : http client register success: {"appName":"http","context":"/http",
"path":"/http/test/**","pathDesc":"","rpcType":"http","host":"192.168.50.13","port
":8188,"ruleName":"/http/test/**","enabled":true,"registerMetaData":false}
2021-02-10 00:57:07.577  INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : http client register success: {"appName":"http","context":"/http",
"path":"/http/order/save","pathDesc":"Save order","rpcType":"http","host":"192.168.
50.13","port":8188,"ruleName":"/http/order/save","enabled":true,"registerMetaData
":false}
2021-02-10 00:57:07.587  INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : http client register success: {"appName":"http","context":"/http",
"path":"/http/order/path/**/name","pathDesc":"","rpcType":"http","host":"192.168.
50.13","port":8188,"ruleName":"/http/order/path/**/name","enabled":true,
"registerMetaData":false}
2021-02-10 00:57:07.596  INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : http client register success: {"appName":"http","context":"/http",
"path":"/http/order/findById","pathDesc":"Find by id","rpcType":"http","host":"192.
168.50.13","port":8188,"ruleName":"/http/order/findById","enabled":true,
"registerMetaData":false}
2021-02-10 00:57:07.606  INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : http client register success: {"appName":"http","context":"/http",
"path":"/http/order/path/**","pathDesc":"","rpcType":"http","host":"192.168.50.13",
"port":8188,"ruleName":"/http/order/path/**","enabled":true,"registerMetaData
":false}
2021-02-10 00:57:08.023  INFO 3700 --- [           main] o.s.b.web.embedded.netty.
NettyWebServer  : Netty started on port(s): 8188
2021-02-10 00:57:08.026  INFO 3700 --- [           main] o.d.s.e.http.
ShenyuTestHttpApplication     : Started ShenyuTestHttpApplication in 2.555 seconds
(JVM running for 3.411)
```

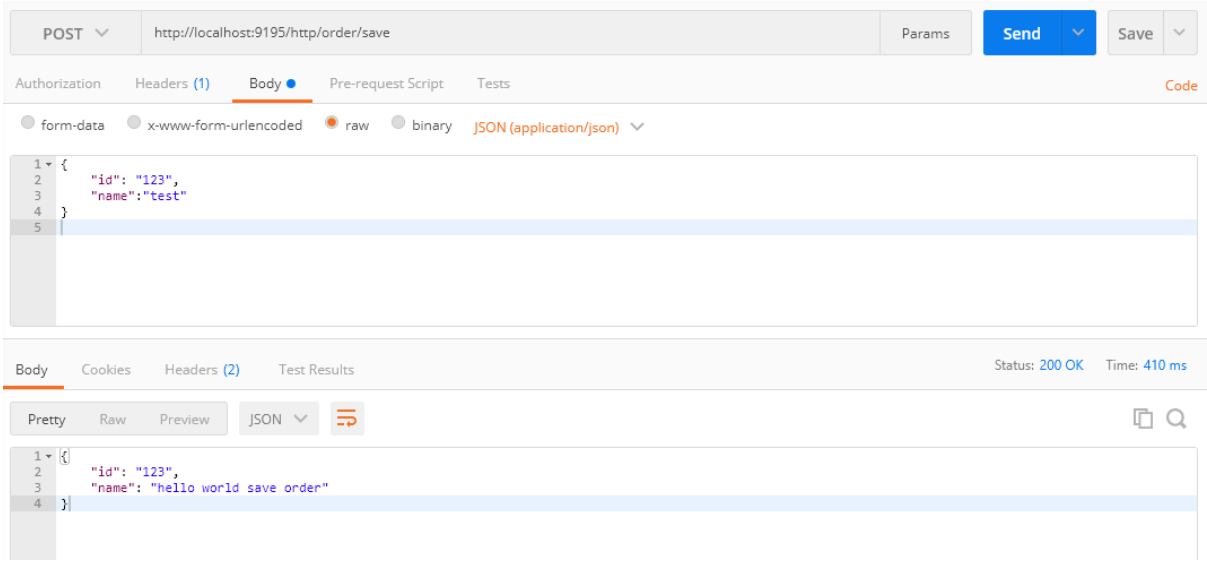
9.19 测试 Http 请求

shenyu-examples-http 项目成功启动之后会自动把加 @ShenyuSpringMvcClient 注解的接口方法注册到网关。

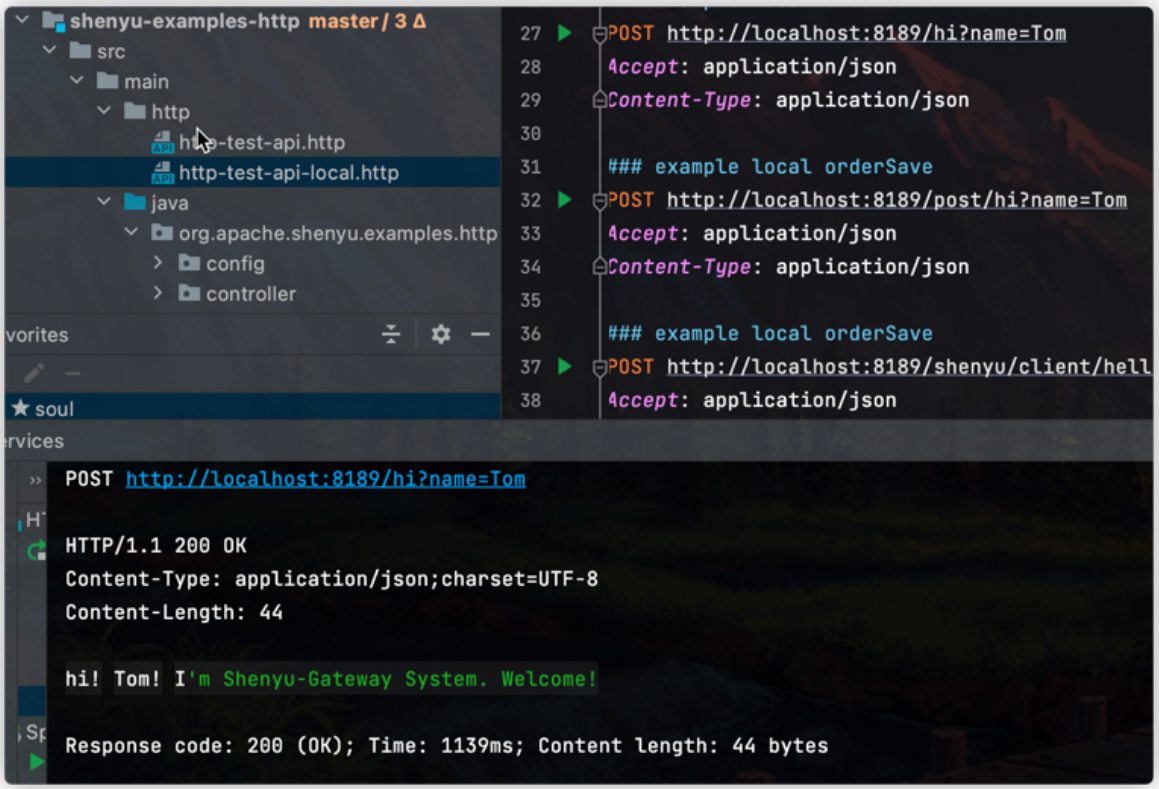
打开插件列表 -> Proxy -> divide 可以看到插件规则配置列表:



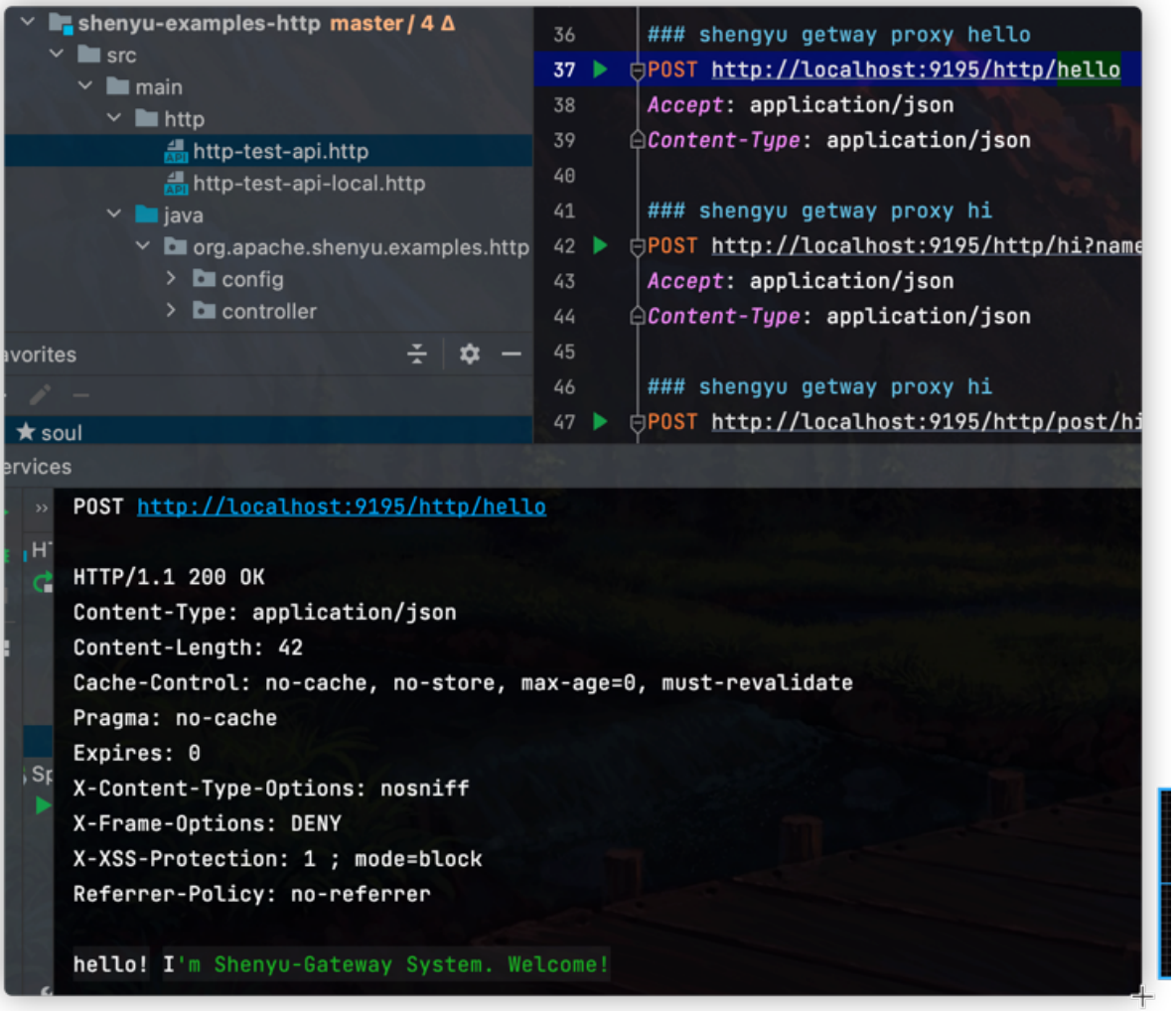
下面使用 postman 模拟 http 的方式来请求你的 http 服务:



下面使用 IDEA HTTP Client Plugin 模拟 http 的方式来请求你的 http 服务 [本地访问, 不使用 shenyu 代理]:



下面使用 IDEA HTTP Client Plugin 模拟 http 的方式来请求你的 http 服务 [使用 shenyu 代理]:



本文档演示如何将 Motan 服务接入到 Apache ShenYu 网关。您可以直接在工程下找到本文档的示例代码 。

9.20 环境准备

请参考运维部署的内容，选择一种方式启动 shenyu-admin。比如，通过 本地部署 启动 Apache ShenYu 后台管理系统。

启动成功后，需要在基础配置-> 插件管理中，把 motan 插件设置为开启。

启动网关，如果是通过源码的方式，直接运行 shenyu-bootstrap 中的 ShenyuBootstrapApplication。

注意，在启动前，请确保网关已经引入相关依赖。本地已经成功启动 zookeeper。

引入网关对 Motan 的代理插件，在网关的 pom.xml 文件中增加如下依赖:

```
<!-- apache shenyu motan plugin -->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-motan</artifactId>
```

```
    <version>${project.version}</version>
</dependency>

<dependency>
    <groupId>com.weibo</groupId>
    <artifactId>motan-core</artifactId>
    <version>1.1.9</version>
</dependency>

<dependency>
    <groupId>com.weibo</groupId>
    <artifactId>motan-registry-zookeeper</artifactId>
    <version>1.1.9</version>
</dependency>

<dependency>
    <groupId>com.weibo</groupId>
    <artifactId>motan-transport-netty4</artifactId>
    <version>1.1.9</version>
</dependency>

<dependency>
    <groupId>com.weibo</groupId>
    <artifactId>motan-springsupport</artifactId>
    <version>1.1.9</version>
</dependency>
```

9.21 运行 shenyu-examples-motan 项目

下载 shenyu-examples-motan 。

运行 `org.apache.shenyu.examples.motan.service.TestMotanApplicationmain` 方法启动项目。

成功启动会有如下日志:

```
2021-07-18 16:46:25.388  INFO 96 --- [           main] o.s.b.w.embedded.tomcat.
TomcatWebServer  : Tomcat started on port(s): 8081 (http) with context path ''
2021-07-18 16:46:25.393  INFO 96 --- [           main] o.a.s.e.m.service.
TestMotanApplication   : Started TestMotanApplication in 3.89 seconds (JVM running
for 4.514)
2021-07-18 16:46:25.396  INFO 96 --- [           main] info
            : [ZookeeperRegistry] Url (null) will set to available to Registry
[zookeeper://localhost:2181/default_rpc/com.weibo.api.motan.registry.
RegistryService/1.0/service]
2021-07-18 16:46:25.399  INFO 96 --- [       Thread-6] o.a.s.c.c.s.
ShenyuClientShutdownHook     : hook Thread-0 will sleep 3000ms when it start
```

```
2021-07-18 16:46:25.399  INFO 96 --- [       Thread-6] o.a.s.c.c.s.
ShenyuClientShutdownHook     : hook SpringContextShutdownHook will sleep 3000ms
when it start
2021-07-18 16:46:25.445  INFO 96 --- [ntLoopGroup-3-2] info
            : NettyChannelHandler channelActive: remote=/192.168.1.8:49740 local=/
192.168.1.8:8002
2021-07-18 16:46:25.445  INFO 96 --- [ntLoopGroup-3-1] info
            : NettyChannelHandler channelActive: remote=/192.168.1.8:49739 local=/
192.168.1.8:8002
2021-07-18 16:46:25.925  INFO 96 --- [or_consumer_-17] o.a.s.r.client.http.utils.
RegisterUtils  : motan client register success: {"appName":"motan","contextPath":"/
motan","path":"/motan/hello","pathDesc":"","rpcType":"motan","serviceName":"org.
apache.shenyu.examples.motan.service.MotanDemoService","methodName":"hello",
"ruleName":"/motan/hello","parameterTypes":"java.lang.String","rpcExt":"{\
"methodInfo\":[{\"methodName\":\"hello\",\"params\":[{\"left\":\"java.lang.String\
",\"right\":\"name\"}]}],\"group\":\"motan-shenyu-rpc\"}","enabled":true,"host":
"192.168.220.1","port":8081,"registerMetaData":false}
```

9.22 测试 Http 请求

shenyu-examples-motan 项目成功启动之后会自动把加 @ShenyuMotanClient 注解的接口方法注册到网关，并添加选择器和规则，如果没有，可以手动添加。

打开插件列表 -> rpc proxy -> motan 可以看到插件规则配置列表:

下面使用 postman 模拟 http 的方式来请求你的 motan 服务:

10

User Guide

本文档旨在帮助 http 服务接入到 Apache ShenYu 网关。Apache ShenYu 网关使用 divide 插件来处理 http 请求。

接入前，请正确启动 shenyu-admin，并开启 divide 插件，在网关端和 Http 服务端引入相关依赖。可以参考前面的 Http 快速开始。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.1 在网关中引入 divide 插件

- 在网关的 pom.xml 文件中增加如下依赖：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-divide</artifactId>
    <version>${project.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-httpclient</artifactId>
    <version>${project.version}</version>
</dependency>
```

10.2 Http 请求接入网关（springMvc 体系用户）

- SpringBoot 用户

  可以参考：shenyu-examples-http

  在你的 http 服务中的 pom.xml 文件新增如下依赖:

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-client-springmvc</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

- SpringMvc 用户

  可以参考：shenyu-examples-springmvc

  在你的 http 服务中的 pom.xml 文件新增如下依赖:

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-client-springmvc</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

  并在你的 bean 定义的 xml 文件中新增如下:

```
<bean id="springMvcClientBeanPostProcessor" class="org.apache.shenyu.client.
springmvc.init.SpringMvcClientBeanPostProcessor">
    <constructor-arg ref="clientPropertiesConfig"/>
    <constructor-arg ref="clientRegisterRepository"/>
</bean>

<!-- 根据实际的注册类型配置注册中心 -->
<bean id="shenyuRegisterCenterConfig" class="org.apache.shenyu.register.common.
config.ShenyuRegisterCenterConfig">
    <property name="registerType" value="http"/>
    <property name="serverLists" value="http://localhost:9095"/>
</bean>

<!-- 客户端属性配置 -->
<bean id="clientPropertiesConfig"
      class="org.apache.shenyu.register.common.config.ShenyuClientConfig.
ClientPropertiesConfig">
  <property name="props">
      <map>
          <entry key="contextPath" value="/你的 contextPath"/>
          <entry key="appName" value=" 你的 app 名字"/>
          <entry key="port" value=" 你的端口"/>
          <entry key="isFull" value="false"/>
```

```
      </map>
    </property>
</bean>

<!-- 根据实际的注册类型配置客户端注册仓库 -->
<bean id="clientRegisterRepository" class="org.apache.shenyu.register.client.
http.HttpClientRegisterRepository">
    <constructor-arg ref="shenyuRegisterCenterConfig"/>
</bean>

<bean id="shenyuClientShutdownHook" class="org.apache.shenyu.client.core.
shutdown.ShenyuClientShutdownHook">
    <constructor-arg ref="shenyuRegisterCenterConfig"/>
    <constructor-arg ref="clientRegisterRepository"/>
</bean>

<bean id="contextRegisterListener" class="org.apache.shenyu.client.springmvc.
init.ContextRegisterListener">
    <constructor-arg ref="clientPropertiesConfig"/>
</bean>
```

在你的 controller 的接口上加上 @ShenyuSpringMvcClient 注解。

你可以把注解加到 Controller 类上面，里面的 path 属性则为前缀，如果含有 /** 代表你的整个接口需要被网关代理。

示例一

下面表示的是 /test/payment，/test/findByUserId 都会被网关代理。

```
@RestController
@RequestMapping("/test")
@ShenyuSpringMvcClient(path = "/test/**")
public class HttpTestController {

    @PostMapping("/payment")
    public UserDTO post(@RequestBody final UserDTO userDTO) {
        return userDTO;
    }

    @GetMapping("/findByUserId")
    public UserDTO findByUserId(@RequestParam("userId") final String userId) {
        UserDTO userDTO = new UserDTO();
        userDTO.setUserId(userId);
        userDTO.setUserName("hello world");
        return userDTO;
    }
}
```

示例二

下面表示的是：/order/save 会被网关代理，而 /order/findById 则不会。

```
@RestController
@RequestMapping("/order")
@ShenyuSpringMvcClient(path = "/order")
public class OrderController {

    @PostMapping("/save")
    @ShenyuSpringMvcClient(path = "/save")
    public OrderDTO save(@RequestBody final OrderDTO orderDTO) {
        orderDTO.setName("hello world save order");
        return orderDTO;
    }

    @GetMapping("/findById")
    public OrderDTO findById(@RequestParam("id") final String id) {
        OrderDTO orderDTO = new OrderDTO();
        orderDTO.setId(id);
        orderDTO.setName("hello world findById");
        return orderDTO;
    }
}
```

示例三：这是一种简化的使用方式，只需要一个简单的注释即可使用元数据注册到网关. 特别说明：目前只支持 @RequestMapping、@GetMapping、@PostMapping、@DeleteMapping、@PutMapping 注解，并且只对 @XXXMapping 中的第一个路径有效

```
@RestController
@RequestMapping("new/feature")
public class NewFeatureController {

  /**
   * no support gateway access api.
   *
   * @return result
   */
  @RequestMapping("/gateway/not")
  public EntityResult noSupportGateway() {
    return new EntityResult(200, "no support gateway access");
  }

  /**
   * Do not use shenyu annotation path. used request mapping path.
   *
   * @return result
   */
  @RequestMapping("/requst/mapping/path")
  @ShenyuSpringCloudClient
  public EntityResult requestMappingUrl() {
```

```
    return new EntityResult(200, "Do not use shenyu annotation path. used request
mapping path");
  }

  /**
   * Do not use shenyu annotation path. used post mapping path.
   *
   * @return result
   */
  @PostMapping("/post/mapping/path")
  @ShenyuSpringCloudClient
  public EntityResult postMappingUrl() {
    return new EntityResult(200, "Do not use shenyu annotation path. used post
mapping path");
  }

  /**
   * Do not use shenyu annotation path. used post mapping path.
   *
   * @return result
   */
  @GetMapping("/get/mapping/path")
  @ShenyuSpringCloudClient
  public EntityResult getMappingUrl() {
    return new EntityResult(200, "Do not use shenyu annotation path. used get
mapping path");
  }
}
```

- 启动你的项目，你的服务接口接入到了网关，进入 shenyu-admin 后台管理系统的插件列表 -> http process -> divide，看到自动创建的选择器和规则。

10.3 Http 请求接入网关（其他语言，非 springMvc 体系）

- 首先在 shenyu-admin 找到 divide 插件，进行选择器，和规则的添加，进行流量的匹配筛选。
- 如果不懂怎么配置，请参考 选择器和规则管理。
- 您也可以自定义开发属于你的 http-client，参考 多语言 Http 客户端开发。

10.4 用户请求

当你的 Http 服务接入到 Apache ShenYu 网关后，请求方式没有很大的变动，小的改动有两点。

- 第一点，你之前请求的域名是你自己的服务，现在要换成网关的域名。
- 第二点，Apache ShenYu 网关需要有一个路由前缀，这个路由前缀就是你接入项目进行配置 contextPath，如果熟的话，可以在 shenyu-admin 中的 divide 插件进行自由更改。
  - 比如你有一个 order 服务它有一个接口，请求路径 http://localhost:8080/test/save。
  - 现在就需要换成：http://localhost:9195/order/test/save。
  - 其中 localhost:9195 为网关的 ip 端口，默认端口是 9195，/order 是你接入网关配置的 contextPath。
  - 其他参数，请求方式不变。

然后你就可以进行访问了，如此的方便与简单。

本篇将介绍如何将网关实例注册到注册中心。Apache ShenYu 网关目前支持注册到 zookeeper、etcd。

10.5 添加 Maven 依赖

首先，在网关的 pom.xml 文件中引入如下依赖。

```
<!--shenyu instance start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-instance</artifactId>
    <version>${project.version}</version>
</dependency>
<!--shenyu instance end-->
```

10.6 使用 zookeeper

> 请注意，从 ShenYu 2.5.0 起将不再支持 Zookeeper 3.4.x 或更低版本。如果您已经使用了 Zookeeper 3.4.x 或更低的版本，您需要使用更高的 Zookeeper 版本并重新初始化数据。

在网关的 yml 文件中添加如下配置：

```
instance:
    enabled: true
    registerType: zookeeper
    serverLists: localhost:2181 # 配置成你的 zookeeper 地址，集群环境请使用（,）分隔
    props:
      sessionTimeout: 3000 # 可选，默认 3000
      connectionTimeout: 3000 # 可选，默认 3000
```

10.7 使用 etcd

在网关的 yml 文件中添加如下配置:

```
instance:
    enabled: true
    registerType: etcd
    serverLists: http://localhost:2379 # 配置成你的 etcd 地址，集群环境请使用（,）分隔。
    props:
      etcdTimeout: 3000 # 可选，默认 3000
      etcdTTL: 5 # 可选，默认 5
```

10.8 使用 consul

在网关的 yml 文件中添加如下配置:

```
instance:
    enabled: true
    registerType: consul
    serverLists: localhost:8848 # 配置成你的 consul 地址，集群环境请使用（,）分隔。
    props:
      consulTimeout: 3000 # 可选，默认 3000
      consulTTL: 3000 # 可选，默认 3000
```

配置完成后，启动网关，就会成功注册到相应注册中心。

本篇主要讲解如何配置数据同步策略，数据同步是指在 shenyu-admin 后台操作数据以后，使用何种策略将数据同步到 Apache ShenYu 网关。Apache ShenYu 网关当前支持 ZooKeeper、WebSocket、Http 长轮询、Nacos、Etcd 和 Consul 进行数据同步。

数据同步原理请参考设计文档中的 数据同步原理 。

10.9 WebSocket 同步配置（默认方式，推荐）

- Apache ShenYu 网关配置

  首先在 pom.xml 文件中引入以下依赖:

```
<!-- apache shenyu data sync start use websocket-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-sync-data-websocket</artifactId>
    <version>${project.version}</version>
</dependency>
```

然后在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    websocket :
      urls: ws://localhost:9095/websocket
      # urls: 是指 shenyu-admin 的地址，如果有多个，请使用（,）分割。
```

- shenyu-admin 配置

  在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    websocket:
      enabled: true
```

当建立连接以后会全量获取一次数据，以后的数据都是增量的更新与新增，性能好。而且也支持断线重连（默认 30 秒）。推荐使用此方式进行数据同步，也是 Apache ShenYu 默认的数据同步策略。

10.10 Zookeeper 同步配置

请注意，从 ShenYu 2.5.0 起将不再支持 Zookeeper 3.4.x 或更低版本。如果您已经使用了 Zookeeper 3.4.x 或更低的版本，您需要使用更高的 Zookeeper 版本并重新初始化数据。

- Apache ShenYu 网关配置

  首先在 pom.xml 文件中引入以下依赖:

```
<!-- apache shenyu data sync start use zookeeper-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-sync-data-zookeeper</artifactId>
    <version>${project.version}</version>
</dependency>
```

然后在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    zookeeper:
      url: localhost:2181
      # url: 配置成你的 zookeeper 地址，集群环境请使用（,）分隔
      sessionTimeout: 5000
      connectionTimeout: 2000
```

- shenyu-admin 配置

在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    zookeeper:
      url: localhost:2181
      # url: 配置成你的 zookeeper 地址，集群环境请使用（,）分隔
      sessionTimeout: 5000
      connectionTimeout: 2000
```

使用 zookeeper 同步机制也是非常好的，时效性也高，但是要处理 zookeeper 环境不稳定，集群脑裂等问题。

10.11 Http 长轮询同步配置

- Apache ShenYu 网关配置

首先在 pom.xml 文件中引入以下依赖:

```
<!-- apache shenyu data sync start use http-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-sync-data-http</artifactId>
    <version>${project.version}</version>
</dependency>
```

然后在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    http:
      url: http://localhost:9095
      # url: 配置成你的 shenyu-admin 的 ip 与端口地址，多个 admin 集群环境请使用（,）分隔。
```

- shenyu-admin 配置

在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    http:
      enabled: true
```

使用 Http 长轮询进行数据同步，会让网关很轻量，但时效性略低。它是根据分组 key 来拉取，如果数据量过大，过多，会有一定的影响。原因是一个组下面的一个小地方更改，都会拉取整个组的数据。

10.12 Nacos 同步配置

- Apache ShenYu 网关配置

首先在 pom.xml 文件中引入以下依赖:

```
<!-- apache shenyu data sync start use nacos-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-sync-data-nacos</artifactId>
    <version>${project.version}</version>
</dependency>
```

然后在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    nacos:
      url: localhost:8848
      # url: 配置成你的 nacos 地址，集群环境请使用（,）分隔。
      namespace: 1c10d748-af86-43b9-8265-75f487d20c6c
      username:
      password:
      acm:
        enabled: false
        endpoint: acm.aliyun.com
        namespace:
        accessKey:
        secretKey:
      # 其他参数配置，请参考 naocs 官网。
```

- shenyu-admin 配置

在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    nacos:
      url: localhost:8848
      # url: 配置成你的 nacos 地址，集群环境请使用（,）分隔。
      namespace: 1c10d748-af86-43b9-8265-75f487d20c6c
      username:
      password:
      acm:
        enabled: false
        endpoint: acm.aliyun.com
        namespace:
        accessKey:
        secretKey:
      # 其他参数配置，请参考 naocs 官网。
```

10.13 Etcd 同步配置

- Apache ShenYu 网关配置

首先在 pom.xml 文件中引入以下依赖:

```
<!-- apache shenyu data sync start use etcd-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-sync-data-etcd</artifactId>
    <version>${project.version}</version>
    <exclusions>
        <exclusion>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-grpclb</artifactId>
        </exclusion>
        <exclusion>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-netty</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

然后在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    etcd:
      url: http://localhost:2379
      # url: 配置成你的 etcd, 集群环境请使用（,）分隔。
```

- shenyu-admin 配置

在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    etcd:
      url: http://localhost:2379
      # url: 配置成你的 etcd, 集群环境请使用（,）分隔。
```

10.14 Consul 同步配置

- Apache ShenYu 网关配置

首先在 pom.xml 文件中引入以下依赖:

```
<!-- apache shenyu data sync start use consul-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-sync-data-consul</artifactId>
    <version>${project.version}</version>
</dependency>
```

然后在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    consul:
      url: http://localhost:8500
      waitTime: 1000    # 查询等待时间
      watchDelay: 1000  # 数据同步间隔时间
```

- shenyu-admin 配置

在 yml 文件中进行如下配置:

```
shenyu:
  sync:
    consul:
      url: http://localhost:8500

在\ ``Apache ShenYu``\ 网关和\ ``shenyu-admin`` 重新配置数据同步策略后，需要重启服务。

``Apache ShenYu``\ 网关 和 ``shenyu-admin`` 必须使用相同的同步策略。
```

此篇文章是介绍 sofa 服务接入到 Apache ShenYu 网关，Apache ShenYu 网关使用 sofa 插件来接入 sofa 服务。

接入前，请正确启动 shenyu-admin，并开启 sofa 插件，在网关端和 sofa 服务端引入相关依赖。可以参考前面的 Sofa 快速开始。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.15 在网关中引入 sofa 插件

- 在网关的 pom.xml 文件中增加如下依赖:
- sofa 版本换成你的，引入你需要的注册中心依赖，以下是参考。

```
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-rpc-all</artifactId>
    <version>5.7.6</version>
    <exclusions>
        <exclusion>
            <groupId>net.jcip</groupId>
            <artifactId>jcip-annotations</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-client</artifactId>
    <version>4.0.1</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>4.0.1</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>4.0.1</version>
</dependency>
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-sofa</artifactId>
    <version>${project.version}</version>
</dependency>
```

- 重启网关服务。

10.16 sofa 服务接入网关

可以参考：shenyu-examples-sofa

如果是 springboot 构建，引入以下依赖：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-client-sofa</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

如果是 spring 构建，引入以下依赖：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-client-sofa</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

并在你的 bean 定义的 xml 文件中新增如下：

```
<bean id ="sofaServiceBeanPostProcessor" class ="org.apache.shenyu.client.sofa.
SofaServiceBeanPostProcessor">
     <constructor-arg  ref="shenyuRegisterCenterConfig"/>
</bean>
<bean id="shenyuRegisterCenterConfig" class="org.apache.shenyu.register.common.
config.ShenyuRegisterCenterConfig">
       <property name="registerType" value="http"/>
       <property name="serverList" value="http://localhost:9095"/>
       <property name="props">
          <map>
            <entry key="contextPath" value="/你的 contextPath"/>
            <entry key="appName" value=" 你的名字"/>
            <entry key="ifFull" value="false"/>
          </map>
        </property>
</bean>
```

10.17 sofa 插件设置

- 首先在 shenyu-admin 插件管理中，把 sofa 插件设置为开启。
- 其次在 sofa 插件中配置你的注册地址或者其他注册中心的地址.

```
{"protocol":"zookeeper","register":"127.0.0.1:2181"}
```

10.18 接口注册到网关

- 在 sofa 服务的类或者方法上加上 @ShenyuSofaClient 注解，表示该类或接口方法注册到网关。
- 启动 sofa 服务提供者，成功注册后，进入后台管理系统的 插件列表 -> rpc proxy -> sofa，会看到自动注册的选择器和规则信息。

10.19 sofa 用户请求及参数说明

可以通过 http 的方式来请求你的 sofa 服务。Apache ShenYu 网关需要有一个路由前缀，这个路由前缀就是接入网关配置的 contextPath。

比如你有一个 order 服务它有一个接口，它的注册路径 /order/test/save

现在就是通过 post 方式请求网关：http://localhost:9195/order/test/save

其中 localhost:9195 为网关的 ip 端口，默认端口是 9195，/order 是你 sofa 接入网关配置的 contextPath

- 参数传递：
  - 通过 http 协议，post 方式访问网关，通过在 http body 中传入 json 类型参数。
  - 更多参数类型传递，可以参考 shenyu-examples-sofa 中的接口定义，以及参数传递方式。
- 单个 java bean 参数类型（默认）
- 自定义实现多参数支持：
  - 在你搭建的网关项目中，新增一个类 MySofaParamResolveService，实现 org.apache.shenyu.plugin.api.sofa.SofaParamResolveService 接口。

```
public interface SofaParamResolveService {

    /**
     * Build parameter pair.
     * this is Resolve http body to get sofa param.
     *
     * @param body           the body
     * @param parameterTypes the parameter types
     * @return the pair
     */
    Pair<String[], Object[]> buildParameter(String body, String parameterTypes);
}
```

- body 为 http 中 body 传的 json 字符串。
- parameterTypes: 匹配到的方法参数类型列表，如果有多个，则使用，分割。
- Pair 中，left 为参数类型，right 为参数值，这是 sofa 泛化调用的标准。
- 把你的类注册成 Spring 的 bean，覆盖默认的实现。

```
@Bean
 public SofaParamResolveService mySofaParamResolveService() {
         return new MySofaParamResolveService();
 }
```

此篇文介绍如何将 Tars 服务接入到 Apache ShenYu 网关，Apache ShenYu 网关使用 tars 插件来接入 Tars 服务。

接入前，请正确启动 shenyu-admin，并开启 tars 插件，在网关端和 tars 服务端引入相关依赖。可以参考前面的 Tars 快速开始 。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.20 在网关中引入 tars 插件

引入网关对 Tars 的代理插件，在网关的 pom.xml 文件中增加如下依赖：

```
<!-- apache shenyu tars plugin start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-tars</artifactId>
    <version>${project.version}</version>
</dependency>

<dependency>
    <groupId>com.tencent.tars</groupId>
    <artifactId>tars-client</artifactId>
    <version>1.7.2</version>
</dependency>
<!-- apache shenyu tars plugin end-->
```

- 重启你的网关服务。

10.21 Tars 服务接入网关

可以参考：shenyu-examples-tars

- 在由 Tars 构建的微服务中，引入如下依赖：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-client-tars</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

在 Tasr 服务接口实现类上加上 @ShenyuTarsService 注解，在方法上加上注解 @ShenyuTarsClient，启动你的服务提供者，成功注册后，在后台管理系统进入插件列表 -> rpc proxy -> tars，会看到自动注册的选择器和规则信息。

示例：

```
@TarsServant("HelloObj")
@ShenyuTarsService(serviceName = "ShenyuExampleServer.ShenyuExampleApp.HelloObj")
public class HelloServantImpl implements HelloServant {
    @Override
    @ShenyuTarsClient(path = "/hello", desc = "hello")
    public String hello(int no, String name) {
        return String.format("hello no=%s, name=%s, time=%s", no, name, System.
currentTimeMillis());
    }

    @Override
    @ShenyuTarsClient(path = "/helloInt", desc = "helloInt")
    public int helloInt(int no, String name) {
        return 1;
    }
}
```

10.22 用户请求

可以通过 http 的方式来请求你的 tars 服务。Apache ShenYu 网关需要有一个路由前缀，这个路由前缀就是接入网关配置的 contextPath。比如：http://localhost:9195/tars/hello 。

应用客户端接入是指将你的微服务接入到 Apache ShenYu 网关，当前支持 Http、Dubbo、Spring Cloud、gRPC、Motan、Sofa、Tars 等协议的接入。

将应用客户端接入到 Apache ShenYu 网关是通过注册中心来实现的，涉及到客户端注册和服务端同步数据。注册中心支持 Http、Zookeeper、Etcd、Consul 和 Nacos。

本篇文章介绍将应用客户端接入到 Apache ShenYu 网关，应该如何配置。相关原理请参考设计文档中的 客户端接入原理 。

10.23 Http 方式注册配置

10.23.1 shenyu-admin 配置

在 yml 文件中配置注册类型为 http，配置信息如下：

```
shenyu:
  register:
    registerType: http
    props:
```

```
      checked: true  # 是否开启检测
      zombieCheckTimes: 5 # 失败几次后剔除服务
      scheduledTime: 10 # 定时检测间隔时间 （秒）
```

10.23.2 shenyu-client 配置

下面展示的是 http 服务作为客户端接入到 Apache ShenYu 网关时，通过 Http 方式注册配置信息。其他客户端接入时（Dubbo、Spring Cloud 等），配置方式同理。

在微服务中的 yml 文件配置注册方式设置为 http，并填写 shenyu-admin 服务地址列表，配置信息如下：

```
shenyu:
  client:
    registerType: http
    serverLists: http://localhost:9095
    props:
      contextPath: /http
      appName: http
      port: 8188
      isFull: false
# registerType : 服务注册类型，填写 http
# serverList: 为 http 注册类型时，填写 Shenyu-Admin 项目的地址，注意加上 http://，多个地址用英文逗号分隔
# port: 你本项目的启动端口，目前 springmvc/tars/grpc 需要进行填写
# contextPath: 为你的这个 mvc 项目在 shenyu 网关的路由前缀， 比如/order ，/product 等等，网关会根据你的这个前缀来进行路由.
# appName: 你的应用名称，不配置的话，会默认取 `spring.application.name` 的值
# isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller；目前适用于 springmvc/springcloud
```

10.24 Zookeeper 方式注册配置

> 请注意，从 ShenYu 2.5.0 起将不再支持 Zookeeper 3.4.x 或更低版本。如果您已经使用了 Zookeeper 3.4.x 或更低的版本，您需要使用更高的 Zookeeper 版本并重新初始化数据。

10.24.1 shenyu-admin 配置

- 首先在 pom 文件中加入相关的依赖（默认已经引入）：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-server-zookeeper</artifactId>
    <version>${project.version}</version>
</dependency>
```

- 然后在 yml 文件中配置注册类型为 zookeeper，填写 zookeeper 服务地址和参数，配置信息如下：

```
shenyu:
  register:
    registerType: zookeeper
    serverLists: localhost:2181
    props:
      sessionTimeout: 5000
      connectionTimeout: 2000
```

10.24.2 shenyu-client 配置

下面展示的是 http 服务作为客户端接入到 Apache ShenYu 网关时，通过 Zookeeper 方式注册配置信息。其他客户端接入时（Dubbo、Spring Cloud 等），配置方式同理。

- 首先在 pom 文件中加入相关的依赖：

```
<!-- apache shenyu zookeeper register center -->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-client-zookeeper</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

- 然后在 yml 中配置注册类型为 zookeeper，并填写 Zookeeper 服务地址和相关参数，如下：

```
shenyu:
  client:
    registerType: zookeeper
    serverLists: localhost:2181
    props:
      contextPath: /http
      appName: http
      port: 8189
      isFull: false
# registerType : 服务注册类型，填写 zookeeper
# serverList: 为 zookeeper 注册类型时，填写 zookeeper 地址，多个地址用英文逗号分隔
# port: 你本项目的启动端口，目前 springmvc/tars/grpc 需要进行填写
# contextPath: 为你的这个 mvc 项目在 shenyu 网关的路由前缀， 比如/order , /product 等等，网关会根据你的这个前缀来进行路由.
# appName: 你的应用名称，不配置的话，会默认取 `spring.application.name` 的值
# isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller；目前适用于 springmvc/springcloud
```

10.25 Etcd 方式注册配置

10.25.1 shenyu-admin 配置

- 首先在 pom 文件中加入相关的依赖（默认已经引入）：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-server-etcd</artifactId>
    <version>${project.version}</version>
</dependency>
```

- 然后在 yml 配置注册类型为 etcd, 填写 etcd 服务地址和参数，配置信息如下：

```
shenyu:
  register:
    registerType: etcd
    serverLists : http://localhost:2379
```

10.25.2 shenyu-client 配置

下面展示的是 http 服务作为客户端接入到 Apache ShenYu 网关时，通过 Etcd 方式注册配置信息。其他客户端接入时（Dubbo、Spring Cloud 等），配置方式同理。

- 首先在 pom 文件中加入相关的依赖：

```
<!-- apache shenyu etcd register center -->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-client-etcd</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

- 然后在 yml 中配置注册类型为 etcd, 并填写 etcd 服务地址和相关参数，如下：

```
shenyu:
  client:
    registerType: etcd
    serverLists: http://localhost:2379
    props:
      contextPath: /http
      appName: http
      port: 8189
      isFull: false
# registerType : 服务注册类型，填写 etcd
# serverList: 为 etcd 注册类型时，填写 etcd 地址，多个地址用英文逗号分隔
# port: 你本项目的启动端口，目前 springmvc/tars/grpc 需要进行填写
# contextPath: 为你的这个 mvc 项目在 shenyu 网关的路由前缀， 比如/order , /product 等等，网关会根据你的这个前缀来进行路由.
```

```
# appName: 你的应用名称，不配置的话，会默认取 `spring.application.name` 的值
# isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller；目前适用于
springmvc/springcloud
```

10.26 Consul 方式注册配置

10.26.1 shenyu-admin 配置

- 首先在 pom.xml 文件中加入相关的依赖:

```
<!-- apache shenyu consul register start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-server-consul</artifactId>
    <version>${project.version}</version>
</dependency>

<!-- apache shenyu consul register start -->
<dependency>
    <groupId>com.ecwid.consul</groupId>
    <artifactId>consul-api</artifactId>
    <version>${consul.api.version}</version>
</dependency>
<!-- apache shenyu consul register end-->
```

- 在 yml 文件配置注册中心为 consul, consul 的特有配置在 props 节点下进行配置, 配置信息如下:

```
shenyu:
  register:
    registerType: consul
    serverLists: localhost:8500
    props:
      delay: 1
      wait-time: 55
      name: shenyuAdmin
      instanceId: shenyuAdmin
      hostName: localhost
      port: 8500
      tags: test1,test2
      preferAgentAddress: false
      enableTagOverride: false

# registerType : 服务注册类型，填写 consul
# serverLists: consul client agent 地址 (sidecar 模式部署 (单机或者集群)，也可以是 consul
server agent 的地址 (只能连接一个 consul server agent 节点，如果是集群，那么会存在单点故障问
题))
```

```
# delay: 对 Metadata 的监控每次轮询的间隔时长，单位为秒，默认 1 秒
# wait-time: 对 Metadata 的监控单次请求的等待时间（长轮询机制），单位为秒，默认 55 秒
# instanceId: consul 服务必填，consul 需要通过 instance-id 找到具体服务
# name 服务注册到 consul 时所在的组名
# hostName: 为 consul 注册类型时，填写 注册服务实例的 地址， 该注册中心注册的服务实例地址，并不
会用于客户端的调用，所以该配置可以不填，port，preferAgentAddress 同理
# port: 为 consul 注册类型时，填写 注册服务实例的 端口
# tags: 对应 consul 配置中的 tags 配置
# preferAgentAddress: 使用 consul 客户端侧的 agent 对应的 address 作为注册服务实例的
address，会覆盖 hostName 的手动配置
# enableTagOverride: 对应 consul 配置中的 enableTagOverride 配置
```

10.26.2 shenyu-client 配置

下面展示的是 springCloud 服务作为客户端接入到 Apache ShenYu 网关时，通过 Consul 方式注册配置信息（springCloud 服务本身的注册中心可以随意选择，与 shenyu 所选择的注册中心并不会存在冲突，example 中使用的是 eureka）。其他客户端接入时（Dubbo、Spring Cloud 等），配置方式同理。

- 首先在 pom 文件中加入相关的依赖：

```
<!-- apache shenyu consul register center -->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-client-consul</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

- 然后在 yml 文件中配置注册方式为 consul，额外还需要配置 shenyu.register.props，配置信息如下：

```
shenyu:
  register:
    registerType: consul
    serverLists: localhost:8500
    props:
      name: shenyuSpringCloudExample
      instanceId: shenyuSpringCloudExample
      hostName: localhost
      port: 8500
      tags: test1,test2
      preferAgentAddress: false
      enableTagOverride: false
  client:
    springCloud:
      props:
        contextPath: /springcloud
        port: 8884
```

```
# registerType : 服务注册类型，填写 consul
# serverLists: consul client agent 地址 (sidecar 模式部署 (单机或者集群)，也可以是 consul
server agent 的地址 (只能连接一个 consul server agent 节点，如果是集群，那么会存在单点故障问
题))
# shenyu.client.props.port: 你本项目的启动端口，目前 springmvc/tars/grpc 需要进行填写
# contextPath: 为你的这个 mvc 项目在 shenyu 网关的路由前缀， 比如/order ，/product 等等，网
关会根据你的这个前缀来进行路由.
# appName: 你的应用名称，不配置的话，会默认取 `spring.application.name` 的值
# isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller；目前适用于
springmvc
# instanceId: consul 服务必填，consul 需要通过 instance-id 找到具体服务
# name 服务注册到 consul 时所在的组名
# hostName: 为 consul 注册类型时，填写 注册服务实例的 地址， 该注册中心注册的服务实例地址，并不
会用于客户端的调用，所以该配置可以不填，port，preferAgentAddress 同理
# port: 为 consul 注册类型时，填写 注册服务实例的 端口
# tags: 对应 consul 配置中的 tags 配置
# preferAgentAddress: 使用 consul 客户端侧的 agent 对应的 address 作为注册服务实例的
address，会覆盖 hostName 的手动配置
# enableTagOverride: 对应 consul 配置中的 enableTagOverride 配置
```

10.27 Nacos 方式注册配置

10.27.1 shenyu-admin 配置

- 首先在 pom 文件中加入相关的依赖（默认已经引入）：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-server-nacos</artifactId>
    <version>${project.version}</version>
</dependency>
```

- 然后在 yml 文件中配置注册中心为 nacos，填写相关 nacos 服务地址和参数，还有 nacos 的命名空间（需要和 shenyu-client 保持一致），配置信息如下：

```
shenyu:
  register:
    registerType: nacos
    serverLists : localhost:8848
    props:
      nacosNameSpace: ShenyuRegisterCenter
```

10.27.2 shenyu-client 配置

下面展示的是 http 服务作为客户端接入到 Apache ShenYu 网关时，通过 Nacos 方式注册配置信息。其他客户端接入时（Dubbo、Spring Cloud 等），配置方式同理。

- 首先在 pom 文件中加入相关的依赖：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-client-nacos</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

- 然后在 yml 中配置注册方式为 nacos，并填写 nacos 服务地址和相关参数，还需要 Nacos 命名空间（需要和 shenyu-admin 端保持一致），IP（可不填，则自动获取本机 ip）和端口，配置信息如下：

```
shenyu:
  client:
    registerType: nacos
    serverLists: localhost:8848
    props:
      contextPath: /http
      appName: http
      port: 8188
      isFull: false
      nacosNameSpace: ShenyuRegisterCenter
# registerType : 服务注册类型，填写 nacos
# serverList: 为 nacos 注册类型时，填写 nacos 地址，多个地址用英文逗号分隔
# port: 你本项目的启动端口，目前 springmvc/tars/grpc 需要进行填写
# contextPath: 为你的这个 mvc 项目在 shenyu 网关的路由前缀，比如/order ，/product 等等，网关会根据你的这个前缀来进行路由.
# appName: 你的应用名称，不配置的话，会默认取 `spring.application.name` 的值
# isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller；目前适用于 springmvc/springcloud
# nacosNameSpace: nacos 的命名空间
```

10.28 同时注册多种服务类型

以同时注册 http 和 dubbo 服务举例。在 yml 参考如下配置即可：

```
shenyu:
  register:
    registerType: nacos
    serverLists: localhost:8848
  client:
    http:
```

```
        props:
            contextPath: /http
            appName: http
            port: 8188
            isFull: false
    dubbo:
        props:
            contextPath: /dubbo
            appName: dubbo
            port: 28080
    props:
      nacosNameSpace: ShenyuRegisterCenter
# registerType : 服务注册类型，填写 nacos
# serverList: 为 nacos 注册类型时，填写 nacos 地址，多个地址用英文逗号分隔
# http.port: 你本项目的启动 Http 端口，目前 springmvc/SpringCloud 需要进行填写
# http.contextPath: 为你的这个 mvc 项目在 shenyu 网关的路由前缀，比如/order ，/product 等等，网关会根据你的这个前缀来进行路由.
# http.appName: 你的应用名称，不配置的话，会默认取 `spring.application.name` 的值
# http.isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller；目前适用于 springmvc/springcloud
# dubbo.contextPath: 为你的项目中对应 dubbo 接口的 contextPath
# dubbo.port: dubbo 服务端口
# dubbo.appName: dubbo 应用名称
# nacosNameSpace: nacos 的命名空间
```

总结，本文主要介绍了如何将你的微服务（当前支持 Http、Dubbo、Spring Cloud、gRPC、Motan、Sofa、Tars 等协议）接入到 Apache ShenYu 网关。介绍了注册中心的原理，Apache ShenYu 网关支持的注册中心有 Http、Zookeeper、Etcd、Consul、Nacos 等方式。介绍了以 http 服务作为客户端接入到 Apache ShenYu 网关时，使用不同方式注册配置信息。

10.29 说明

- 此篇文章是介绍 dubbo 服务接入到 Apache ShenYu 网关，Apache ShenYu 网关使用 dubbo 插件来接入 Dubbo 服务。
- 当前支持 alibaba dubbo（< 2.7.x）以及 apache dubbo (>=2.7.x)。
- 接入前，请正确启动 shenyu-admin，并开启 dubbo 插件，在网关端和 Dubbo 服务端引入相关依赖。可以参考前面的 Dubbo 快速开始。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.30 在网关中引入 dubbo 插件

- 在网关的 pom.xml 文件中增加如下依赖:
  - alibaba dubbo 用户, dubbo 版本换成你的，引入你需要的注册中心依赖，以下是参考。

```
<!-- apache shenyu alibaba dubbo plugin start-->
<dependency>
  <groupId>org.apache.shenyu</groupId>
  <artifactId>shenyu-spring-boot-starter-plugin-alibaba-dubbo</artifactId>
   <version>${project.version}</version>
</dependency>
<!-- apache shenyu  alibaba dubbo plugin end-->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>dubbo</artifactId>
  <version>2.6.5</version>
</dependency>
<dependency>
  <groupId>org.apache.curator</groupId>
  <artifactId>curator-client</artifactId>
  <version>4.0.1</version>
</dependency>
<dependency>
  <groupId>org.apache.curator</groupId>
  <artifactId>curator-framework</artifactId>
  <version>4.0.1</version>
</dependency>
<dependency>
  <groupId>org.apache.curator</groupId>
  <artifactId>curator-recipes</artifactId>
  <version>4.0.1</version>
</dependency>
```

  - apache dubbo 用户，dubbo 版本换成你的，引入你需要的注册中心依赖，如下是参考。

```
<!-- apache shenyu apache dubbo plugin start-->
<dependency>
   <groupId>org.apache.shenyu</groupId>
   <artifactId>shenyu-spring-boot-starter-plugin-apache-dubbo</artifactId>
   <version>${project.version}</version>
</dependency>
<!-- apache shenyu apache dubbo plugin end-->

<dependency>
   <groupId>org.apache.dubbo</groupId>
   <artifactId>dubbo</artifactId>
   <version>2.7.5</version>
</dependency>
```

```
<!-- Dubbo Nacos registry dependency start -->
<dependency>
   <groupId>org.apache.dubbo</groupId>
   <artifactId>dubbo-registry-nacos</artifactId>
   <version>2.7.5</version>
</dependency>
<dependency>
   <groupId>com.alibaba.nacos</groupId>
   <artifactId>nacos-client</artifactId>
   <version>1.1.4</version>
</dependency>
<!-- Dubbo Nacos registry dependency  end-->

<!-- Dubbo zookeeper registry dependency start-->
<dependency>
   <groupId>org.apache.curator</groupId>
   <artifactId>curator-client</artifactId>
   <version>4.0.1</version>
</dependency>
<dependency>
   <groupId>org.apache.curator</groupId>
   <artifactId>curator-framework</artifactId>
   <version>4.0.1</version>
</dependency>
<dependency>
   <groupId>org.apache.curator</groupId>
   <artifactId>curator-recipes</artifactId>
   <version>4.0.1</version>
</dependency>
<!-- Dubbo zookeeper registry dependency end -->
```

- 重启网关服务。

10.31 dubbo 服务接入网关

可以参考：shenyu-examples-dubbo

- alibaba dubbo 用户

如果是 springboot 构建，引入以下依赖：

```
<dependency>
     <groupId>org.apache.shenyu</groupId>
     <artifactId>shenyu-spring-boot-starter-client-alibaba-dubbo</artifactId>
     <version>${shenyu.version}</version>
</dependency>
```

如果是 spring 构建，引入以下依赖：

```
<dependency>
   <groupId>org.apache.shenyu</groupId>
   <artifactId>shenyu-client-alibaba-dubbo</artifactId>
   <version>${shenyu.version}</version>
</dependency>
```

并在你的 bean 定义的 xml 文件中新增如下:

```
<bean id="clientConfig" class="org.apache.shenyu.register.common.config.
PropertiesConfig">
    <property name="props">
      <map>
        <entry key="contextPath" value="/你的 contextPath"/>
        <entry key="appName" value=" 你的名字"/>
      </map>
    </property>
</bean>

<bean id="shenyuRegisterCenterConfig" class="org.apache.shenyu.register.common.
config.ShenyuRegisterCenterConfig">
   <property name="registerType" value="http"/>
   <property name="serverList" value="http://localhost:9095"/>
</bean>

<bean id="shenyuClientRegisterRepository" class="org.apache.shenyu.client.core.
register.ShenyuClientRegisterRepositoryFactory" factory-method="newInstance">
       <property name="shenyuRegisterCenterConfig" ref="shenyuRegisterCenterConfig
"/>
 </bean>

<bean id ="alibabaDubboServiceBeanListener" class ="org.apache.shenyu.client.
alibaba.dubbo.AlibabaDubboServiceBeanListener">
   <constructor-arg name="clientConfig" ref="clientConfig"/>
   <constructor-arg name="shenyuClientRegisterRepository" ref=
"shenyuClientRegisterRepository"/>
</bean>
```

- apache dubbo 用户

如果是 springboot 构建，引入以下依赖:

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-client-apache-dubbo</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

需要在你的 客户端项目定义的 application.yml 文件中新增如下:

```
dubbo:
  registry:
    address: dubbo 注册中心地址

shenyu:
  register:
    registerType: shenyu 服务注册类型 #http #zookeeper #etcd #nacos #consul
    serverLists: shenyu 服务注册地址 #http://localhost:9095 #localhost:2181 #http://
localhost:2379 #localhost:8848
  client:
    dubbo:
      props:
        contextPath: /你的 contextPath
        appName: 你的应用名称
```

如果是 spring 构建，引入以下依赖：

```
<dependency>
   <groupId>org.apache.shenyu</groupId>
   <artifactId>shenyu-client-apache-dubbo</artifactId>
   <version>${shenyu.version}</version>
</dependency>
```

需要在你的 bean 定义的 xml 文件中新增如下：

```
<bean id = "apacheDubboServiceBeanListener" class="org.apache.shenyu.client.apache.
dubbo.ApacheDubboServiceBeanListener">
    <constructor-arg ref="clientPropertiesConfig"/>
    <constructor-arg ref="clientRegisterRepository"/>
</bean>

<!-- 根据实际的注册类型配置注册中心 -->
<bean id="shenyuRegisterCenterConfig" class="org.apache.shenyu.register.common.
config.ShenyuRegisterCenterConfig">
    <property name="registerType" value=" 你的服务注册类型"/>
    <property name="serverLists" value=" 你的服务注册地址"/>
</bean>

<!-- 客户端属性配置 -->
<bean id="clientPropertiesConfig"
      class="org.apache.shenyu.register.common.config.ShenyuClientConfig.
ClientPropertiesConfig">
<property name="props">
  <map>
    <entry key="contextPath" value="/你的 contextPath"/>
    <entry key="appName" value=" 你的应用名字"/>
  </map>
</property>
</bean>
```

```
<!-- 根据实际的注册类型配置客户端注册仓库 -->
<bean id="clientRegisterRepository" class="org.apache.shenyu.register.client.http.
HttpClientRegisterRepository">
    <constructor-arg ref="shenyuRegisterCenterConfig"/>
</bean>

<bean id="shenyuClientShutdownHook" class="org.apache.shenyu.client.core.shutdown.
ShenyuClientShutdownHook">
    <constructor-arg ref="shenyuRegisterCenterConfig"/>
    <constructor-arg ref="clientRegisterRepository"/>
</bean>
```

需要在你的 客户端项目定义的 application.yml 文件中新增如下:

```
dubbo:
  registry:
    address: dubbo 注册中心地址
    port: dubbo 服务端口号
```

10.32 dubbo 插件设置

- 首先在 shenyu-admin 插件管理中，把 dubbo 插件设置为开启。
- 其次在 dubbo 插件中配置你的注册地址，或者其他注册中心的地址。

```
{"register":"zookeeper://localhost:2181"}   or {"register":"nacos://localhost:8848
"}
```

10.33 接口注册到网关

- 在 dubbo 服务实现类的方法上加上 @ShenyuDubboClient 注解，表示该接口方法注册到网关。
- 启动你的提供者，成功启动后，进入后台管理系统的插件列表 -> rpc proxy -> dubbo，会看到自动注册的选择器和规则信息。

10.34 dubbo 用户请求及参数说明

可以通过 http 的方式来请求你的 dubbo 服务。Apache ShenYu 网关需要有一个路由前缀，这个路由前缀就是你接入项目进行配置 contextPath

> 比如你有一个 order 服务它有一个接口，它的注册路径 /order/test/save 现在就是通过 post 方式请求网关：http://localhost:9195/order/test/save 其中 localhost:9195 为网关的 ip 端口，默认端口是 9195 ， /order 是你 dubbo 接入网关配置的 contextPath

- 参数传递:
  - 通过 http 协议，post 方式访问网关，通过在 http body 中传入 json 类型参数。
  - 更多参数类型传递，可以参考 shenyu-examples-dubbo 中的接口定义，以及参数传递方式。
- 单个 java bean 参数类型（默认）
- 多参数类型支持，在网关的 yaml 配置中新增如下配置:

```
shenyu:
    dubbo:
      parameter: multi
```

- 自定义实现多参数支持:
  - 在你搭建的网关项目中, 新增一个类 MyDubboParamResolveService, 实现 org.apache.shenyu.web.dubbo.DubboParamResolveService 接口。

```
public interface DubboParamResolveService {

   /**
    * Build parameter pair.
    * this is Resolve http body to get dubbo param.
    *
    * @param body           the body
    * @param parameterTypes the parameter types
    * @return the pair
    */
   Pair<String[], Object[]> buildParameter(String body, String
parameterTypes);
}
```

  - body 为 http 中 body 传的 json 字符串。
  - parameterTypes: 匹配到的方法参数类型列表，如果有多个，则使用 , 分割。
  - Pair 中，left 为参数类型，right 为参数值，这是 dubbo 泛化调用的标准
  - 把你的类注册成 Spring 的 bean，覆盖默认的实现。

```
@Bean
public DubboParamResolveService myDubboParamResolveService() {
        return new MyDubboParamResolveService();
}
```

10.35 服务治理

- 标签路由
  - 请求时在 header 中添加 Dubbo_Tag_Route，并设置对应的值，之后当前请求就会路由到指定 tag 的 provider，只对当前请求有效。
- 服务提供者直连
  - 设置 @ShenyuDubboClient 注解中的 url 属性。
  - 修改 Admin 控制台修改元数据内的 url 属性。
  - 对所有请求有效。
- 参数验证和自定义异常
  - 指定 validation = "shenyuValidation"。
  - 在接口中抛出 ShenyuException 时，异常信息会返回，需要注意的是显式抛出 ShenyuException。

```
@Service(validation = "shenyuValidation")
public class TestServiceImpl implements TestService {

    @Override
    @ShenyuDubboClient(path = "/test", desc = "test method")
    public String test(@Valid HelloServiceRequest name) throws
ShenyuException {
        if (true){
            throw new ShenyuException("Param binding error.");
        }
        return "Hello " + name.getName();
    }
}
```

  - 请求参数

```
public class HelloServiceRequest implements Serializable {

    private static final long serialVersionUID = -5968745817846710197L;

    @NotEmpty(message = "name cannot be empty")
    private String name;

    @NotNull(message = "age cannot be null")
    private Integer age;

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

- 发送请求

```
{
    "name": ""
}
```

- 返回

```
{
    "code": 500,
    "message": "Internal Server Error",
    "data": "name cannot be empty,age cannot be null"
}
```

- 当按照要求传递请求参数时，会返回自定义异常的信息

```
{
    "code": 500,
    "message": "Internal Server Error",
    "data": "Param binding error."
}
```

10.36 Http –> 网关–> Dubbo Provider

实际上就是把 http 请求，转成 dubbo 协议，内部使用 dubbo 泛化来进行调用。dubbo 服务在接入网关的时候，加上了 @ShenyuDubboClient 注解，并设置了 path 字段来指定请求路径。然后在 yml 中配置了 contextPath。

假如有一个这样的方法, contextPath 配置的是 /dubbo。

```
@Override
@ShenyuDubboClient(path = "/insert", desc = "插入一条数据")
public DubboTest insert(final DubboTest dubboTest) {
```

```
    return dubboTest;
}
```

那么请求的路径为：http://localhost:9195/dubbo/insert，localhost:9195 是网关的地址，如果你更改了，这里也要改。

请求参数：DubboTest 是一个 javabean 对象，有 2 个字段，id 与 name，那么我们通过 body 中传递这个对象的 json 数据就好。

```
{"id": "1234", "name": "XIAO5y"}
```

如果接口中，没有参数，那么 body 传值为：

```
{}
```

如果接口有很多个参数，请参考上面介绍过的多参数类型支持。

此篇文章是介绍 springCloud 服务接入到 Apache ShenYu 网关，Apache ShenYu 网关使用 springCloud 插件来接入 Spring Cloud 服务。

接入前，请正确启动 shenyu-admin，并开启 springCloud 插件，在网关端和 springCloud 服务端引入相关依赖。可以参考前面的 Spring Cloud 快速开始。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.37 在网关中引入 springCloud 插件

- 在网关的 pom.xml 文件中引入如下依赖。

```
<!-- apache shenyu springCloud plugin start-->
<dependency>
     <groupId>org.apache.shenyu</groupId>
     <artifactId>shenyu-spring-boot-starter-plugin-springcloud</artifactId>
      <version>${project.version}</version>
</dependency>

<dependency>
     <groupId>org.apache.shenyu</groupId>
     <artifactId>shenyu-spring-boot-starter-plugin-httpclient</artifactId>
     <version>${project.version}</version>
 </dependency>
 <!-- apache shenyu springCloud plugin end-->

 <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-commons</artifactId>
```

```
      <version>2.2.0.RELEASE</version>
 </dependency>
```

- 如果你使用 eureka 作为 springCloud 的注册中心
  - 在网关的 pom.xml 文件中，新增如下依赖：

```
<dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
     <version>2.2.0.RELEASE</version>
</dependency>
```

- 在网关的 yml 文件中，新增如下配置：

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/ # 你的 eureka 地址
  instance:
    prefer-ip-address: true
```

- 如果你使用 nacos 作为 springCloud 的注册中心
  - 在网关的 pom.xml 文件中，新增如下依赖：

```
<dependency>
      <groupId>com.alibaba.cloud</groupId>
      <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
      <version>2.1.0.RELEASE</version>
</dependency>
```

- 在网关的 yml 文件中新增如下配置：

```
spring:
   cloud:
     nacos:
       discovery:
          server-addr: 127.0.0.1:8848 # 你的 nacos 地址
```

特别提示：请确保 spring Cloud 注册中心服务发现配置开启

- 配置方式

```
spring:
  cloud:
    discovery:
      enabled: true
```

- 代码方式

```
@SpringBootApplication
@EnableDiscoveryClient
public class ShenyuBootstrapApplication {

    /**
     * Main Entrance.
     *
     * @param args startup arguments
     */
    public static void main(final String[] args) {
        SpringApplication.run(ShenyuBootstrapApplication.class, args);
    }
}
```

- 重启你的网关服务。

10.38 SpringCloud 服务接入网关

可以参考：shenyu-examples-springcloud

- 在由 SpringCloud 构建的微服务中，引入如下依赖:

```
<dependency>
     <groupId>org.apache.shenyu</groupId>
     <artifactId>shenyu-spring-boot-starter-client-springcloud</artifactId>
     <version>${shenyu.version}</version>
</dependency>
```

- 在 controller 接口上加上 @ShenyuSpringCloudClient 注解。注解可以加到类或方法上面，path 属性为前缀，如果含有 /** 代表你的整个接口需要被网关代理。
- 示例一：代表 /test/payment, /test/findByUserId 都会被网关代理。

```
@RestController
@RequestMapping("/test")
@ShenyuSpringCloudClient(path = "/test/**")
public class HttpTestController {

    @PostMapping("/payment")
    public UserDTO post(@RequestBody final UserDTO userDTO) {
        return userDTO;
    }

    @GetMapping("/findByUserId")
    public UserDTO findByUserId(@RequestParam("userId") final String userId) {
        UserDTO userDTO = new UserDTO();
        userDTO.setUserId(userId);
        userDTO.setUserName("hello world");
```

```
        return userDTO;
    }
}
```

- 示例二：代表 /order/save，会被网关代理，而/order/findById 则不会。

```
@RestController
@RequestMapping("/order")
@ShenyuSpringCloudClient(path = "/order")
public class OrderController {

    @PostMapping("/save")
    @ShenyuSpringMvcClient(path = "/save")
    public OrderDTO save(@RequestBody final OrderDTO orderDTO) {
        orderDTO.setName("hello world save order");
        return orderDTO;
    }

    @GetMapping("/findById")
    public OrderDTO findById(@RequestParam("id") final String id) {
        OrderDTO orderDTO = new OrderDTO();
        orderDTO.setId(id);
        orderDTO.setName("hello world findById");
        return orderDTO;
    }
}
```

- 示例三：isFull：true 代表整个服务都会被网关代理。

```
shenyu:
  client:
    registerType: http
    serverLists: http://localhost:9095
    props:
      contextPath: /http
      appName: http
      isFull: true
# registerType : 服务注册类型，请参考应用客户端接入文档
# serverList: 服务列表，请参考应用客户端接入文档
# contextPath: 为你的项目在 shenyu 网关的路由前缀。 比如/order ， /product 等等，网关会根据你的这个前缀来进行路由。
# appName: 你的应用名称，不配置的话，会默认取 application 中的名称
# isFull: 设置 true 代表代理你的整个服务，false 表示代理你其中某几个 controller
```

```
@RestController
@RequestMapping("/order")
public class OrderController {
```

```
    @PostMapping("/save")
    @ShenyuSpringMvcClient(path = "/save")
    public OrderDTO save(@RequestBody final OrderDTO orderDTO) {
        orderDTO.setName("hello world save order");
        return orderDTO;
    }

    @GetMapping("/findById")
    public OrderDTO findById(@RequestParam("id") final String id) {
        OrderDTO orderDTO = new OrderDTO();
        orderDTO.setId(id);
        orderDTO.setName("hello world findById");
        return orderDTO;
    }
}
```

示例四：这是一种简化的使用方式，只需要一个简单的注解，使用元数据注册到网关。特别说明：目前只支持 @RequestMapping、@GetMapping、@PostMapping、@DeleteMapping、@PutMapping 注解，并且只对 @XXXMapping 中的第一个路径有效。

```
@RestController
@RequestMapping("new/feature")
public class NewFeatureController {

  /**
   * no support gateway access api.
   *
   * @return result
   */
  @RequestMapping("/gateway/not")
  public EntityResult noSupportGateway() {
    return new EntityResult(200, "no support gateway access");
  }

  /**
   * Do not use shenyu annotation path. used request mapping path.
   *
   * @return result
   */
  @RequestMapping("/requst/mapping/path")
  @ShenyuSpringCloudClient
  public EntityResult requestMappingUrl() {
    return new EntityResult(200, "Do not use shenyu annotation path. used request
mapping path");
  }

  /**
   * Do not use shenyu annotation path. used post mapping path.
```

```
     *
     * @return result
     */
    @PostMapping("/post/mapping/path")
    @ShenyuSpringCloudClient
    public EntityResult postMappingUrl() {
        return new EntityResult(200, "Do not use shenyu annotation path. used post
mapping path");
    }

    /**
     * Do not use shenyu annotation path. used post mapping path.
     *
     * @return result
     */
    @GetMapping("/get/mapping/path")
    @ShenyuSpringCloudClient
    public EntityResult getMappingUrl() {
        return new EntityResult(200, "Do not use shenyu annotation path. used get
mapping path");
    }
}
```

- 启动你的服务成功注册后, 进入后台管理系统的插件列表 -> rpc proxy -> springCloud, 会看到自动注册的选择器和规则信息。

10.39 用户请求

和之前的访问方式没有大的改变，需要注意的是:

- 你之前请求的域名是你自己的服务，现在要换成网关的域名。
- 网关需要有一个路由前缀，这个路由前缀就是你接入项目进行配置 contextPath，可以在 shenyu-admin 中的 springCloud 插件进行更改。

  比如你有一个 order 服务它有一个接口，请求路径 http://localhost:8080/test/save

  现在就需要换成：http://localhost:9195/order/test/save

  其中 localhost:9195 为网关的 ip 端口，默认端口是 9195，/order 是你接入网关配置的 contextPath

  其他参数，请求方式不变。然后你就可以进行访问了，如此的方便与简单。

此篇文章是介绍 gRPC 服务接入到 Apache ShenYu 网关，Apache ShenYu 网关使用 grpc 插件来接入 gRPC 服务。

接入前，请正确启动 shenyu-admin，并开启 grpc 插件，在网关端和 grpc 服务端引入相关依赖。可以参考前面的 gRPC 快速开始。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.40 在网关中引入 grpc 插件

引入网关对 gRPC 的代理插件，在网关的 pom.xml 文件中增加如下依赖：

```
<!-- apache shenyu grpc plugin start-->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-grpc</artifactId>
    <version>${project.version}</version>
</dependency>
<!-- apache shenyu grpc plugin end-->
```

- 重启你的网关服务。

10.41 gRPC 服务接入网关

可以参考：shenyu-examples-grpc

- 在由 gRPC 构建的微服务中，引入如下依赖：

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-client-grpc</artifactId>
    <version>${shenyu.version}</version>
    <exclusions>
        <exclusion>
            <artifactId>guava</artifactId>
            <groupId>com.google.guava</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

在 shenyu-examples-grpc 下执行以下命令生成 java 代码。

```
mvn protobuf:compile //编译消息对象
mvn protobuf:compile-custom //依赖消息对象，生成接口服务
```

在 gRPC 服务接口实现类上加上 @ShenyuGrpcClient 注解。启动你的服务提供者，成功注册后，在后台管理系统进入插件列表 -> rpc proxy -> grpc，会看到自动注册的选择器和规则信息。

示例：

```
@Override
@ShenyuGrpcClient(path = "/echo", desc = "echo")
```

```
public void echo(EchoRequest request, StreamObserver<EchoResponse>
responseObserver) {
    System.out.println("Received: " + request.getMessage());
    EchoResponse.Builder response = EchoResponse.newBuilder()
            .setMessage("ReceivedHELLO")
            .addTraces(Trace.newBuilder().setHost(getHostname()).build());
    responseObserver.onNext(response.build());
    responseObserver.onCompleted();
}
```

10.42 用户请求

可以通过 http 的方式来请求你的 grpc 服务。Apache ShenYu 网关需要有一个路由前缀，这个路由前缀就是你接入项目进行配置 contextPath。

如果你的 proto 文件定义如下:

```
message EchoRequest {
  string message = 1;
}
```

那么请求参数如下所示:

```
{
    "data": [
        {
            "message": "hello grpc"
        }
    ]
}
```

当前是以 json 的格式传递参数，key 的名称默认是 data，你可以在 GrpcConstants.JSON_DESCRIPTOR_PROTO_FIELD_NAME 中进行重置；value 的传入则根据你定义的 proto 文件。

Apache ShenYu 可以支持 gRPC 的流式调用，通过数组的形式传递多个参数。

如果你的 proto 文件定义如下:

```
message RequestData {
  string text = 1;
}
```

对应的方法调用请求参数如下:

- UNARY

```
{
  "data": [
    {
      "text": "hello grpc"
    }
  ]
}
```

- CLIENT_STREAMING

```
{
  "data": [
    {
      "text": "hello grpc"
    },
    {
      "text": "hello grpc"
    },
    {
      "text": "hello grpc"
    }
  ]
}
```

- SERVER_STREAMING

```
{
  "data": [
    {
      "text": "hello grpc"
    }
  ]
}
```

- BIDI_STREAMING

```
{
  "data": [
    {
      "text": "hello grpc"
    },
    {
      "text": "hello grpc"
    },
    {
      "text": "hello grpc"
    }
  ]
}
```

此篇文介绍如何将 Motan 服务接入到 Apache ShenYu 网关，Apache ShenYu 网关使用 motan 插件来接入 Motan 服务。

接入前，请正确启动 shenyu-admin，并开启 motan 插件，在网关端和 motan 服务端引入相关依赖。可以参考前面的 Motan 快速开始 。

应用客户端接入的相关配置请参考：客户端接入配置。

数据同步的相关配置请参考：数据同步配置。

10.43 在网关中引入 motan 插件

引入网关对 Motan 的代理插件，在网关的 pom.xml 文件中增加如下依赖：

```
  <!-- apache shenyu motan plugin -->
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-plugin-motan</artifactId>
    <version>${project.version}</version>
</dependency>

<dependency>
<groupId>com.weibo</groupId>
<artifactId>motan-core</artifactId>
<version>1.1.9</version>
</dependency>

<dependency>
<groupId>com.weibo</groupId>
<artifactId>motan-registry-zookeeper</artifactId>
<version>1.1.9</version>
</dependency>

<dependency>
<groupId>com.weibo</groupId>
<artifactId>motan-transport-netty4</artifactId>
<version>1.1.9</version>
</dependency>

<dependency>
<groupId>com.weibo</groupId>
<artifactId>motan-springsupport</artifactId>
<version>1.1.9</version>
</dependency>
```

- 重启你的网关服务。

10.44 Motan 服务接入网关

可以参考：shenyu-examples-motan

- 在由 Motan 构建的微服务中，引入如下依赖：

```
        <dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-spring-boot-starter-client-motan</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```

在 Motan 服务接口实现类的方法上加上注解 @ShenyuMotanClient，启动你的服务提供者，成功注册后，在后台管理系统进入插件列表 -> rpc proxy -> motan，会看到自动注册的选择器和规则信息。

示例：

```
    @MotanService(export = "demoMotan:8002")
public class MotanDemoServiceImpl implements MotanDemoService {
    @Override
    @ShenyuMotanClient(path = "/hello")
    public String hello(String name) {
        return "hello " + name;
    }
}
```

10.45 用户请求

可以通过 http 的方式来请求你的 motan 服务。Apache ShenYu 网关需要有一个路由前缀，这个路由前缀就是接入网关配置的 contextPath。比如：http://localhost:9195/motan/hello 。

11

Plugin Center

12 Developer Documentation

12.1 准备

1. 克隆 Apache ShenYu 的代码.
2. 安装并启动 `docker` .

12.2 在本地开启集成测试

1. 用 Maven 构建

```
./mvnw -B clean install -Prelease,docker -Dmaven.javadoc.skip=true -Dmaven.test.
skip=true
```

2. 构建 `shenyu-integrated-test`

```
./mvnw -B clean install -Pit -DskipTests -f ./shenyu-integrated-test/pom.xml
```

3. docker-compose 运行

```
docker-compose -f ./shenyu-integrated-test/${{ matrix.case }}/docker-compose.yml up
-d
```

   你需要把 `${{ matrix.case }}` 替换成具体的目录, 比如 `shenyu-integrated-test-http`.

4. 运行测试

```
./mvnw test -Pit -f ./shenyu-integrated-test/${{ matrix.case }}/pom.xml
```

12.3 说明

- 本文是说明，如果网关前面有一层 nginx 的时候，如何获取正确的 ip 与端口。
- 获取正确的之后，在插件以及选择器中，可以根据 ip，与 host 来进行匹配。

12.4 默认实现

- 在 Apache ShenYu 网关自带实现为: org.apache.shenyu.web.forward.ForwardedRemoteAddressResolver。
- 它需要你在 nginx 设置 X-Forwarded-For，以便来获取正确的 ip 与 host。

12.5 扩展实现

- 新增一个类 CustomRemoteAddressResolver，实现 org.apache.shenyu.plugin.api.RemoteAddressResolver

```
public interface RemoteAddressResolver {

    /**
     * Resolve inet socket address.
     *
     * @param exchange the exchange
     * @return the inet socket address
     */
    default InetSocketAddress resolve(ServerWebExchange exchange) {
        return exchange.getRequest().getRemoteAddress();
    }

}
```

- 把你新增的实现类注册成为 spring 的 bean，如下

```
   @Bean
public RemoteAddressResolver customRemoteAddressResolver() {
        return new CustomRemoteAddressResolver();
        }
```

12.6 说明

- 本文主要讲解其他语言的 http 服务如何接入网关。
- 如何自定义开发 shenyu-http-client。

12.7 自定义开发

- 请求方式：POST
- 请求路径：http://shenyu-admin/shenyu-client/springmvc-register ，其中 shenyu-admin 表示为 admin 后台管理系统的 ip + port 。
- 请求参数：Apache ShenYu 网关默认的需要参数，通过 body 里面传入，json 类型。

```
{
    "appName": "xxx", //应用名称 必填
    "context": "/xxx", //请求前缀 必填
    "path": "xxx", //路径需要唯一 必填
    "pathDesc": "xxx", //路径描述
    "rpcType": "http", //rpc 类型  必填
    "host": "xxx", //服务 host 必填
    "port": xxx, //服务端口 必填
    "ruleName": "xxx", //可以同 path 一样  必填
    "enabled": "true", //是否开启
    "registerMetaData": "true" //是否需要注册元数据
}
```

12.8 说明

- 本文主要介绍 Apache ShenYu 的文件上传下载的支持。

12.9 文件上传

- 默认限制文件大小为 10M。
- 如果想修改，在启动服务的时候，使用--file.size = 30，为 int 类型。
- 你之前怎么上传文件，还是怎么上传。

12.10 文件下载

- Apache ShenYu 支持流的方式进行下载，之前的接口怎么写的，现在还是怎么写，根本不需要变。

12.11 说明

- 插件是 Apache ShenYu 网关的核心执行者，每个插件在开启的情况下，都会对匹配的流量，进行自己的处理。
- 在 Apache ShenYu 网关里面，插件分为两类。
  - 一类是单一职责的调用链，不能对流量进行自定义的筛选。
  - 一类是能对匹配的流量，执行自己的职责调用链。
- 用户可以参考 shenyu-plugin 模块，新增自己的插件处理，如果有好的公用插件，可以向官网提交 pr。

12.12 单一职责插件

- 引入如下依赖:

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-plugin-api</artifactId>
    <version>${project.version}</version>
</dependency>
```

- 用户新增一个类 MyShenyuPlugin，直接实现 org.apache.shenyu.plugin.api.ShenyuPlugin

```
public interface ShenyuPlugin {

    /**
     * Process the Web request and (optionally) delegate to the next
     * {@code WebFilter} through the given {@link ShenyuPluginChain}.
     *
     * @param exchange the current server exchange
     * @param chain    provides a way to delegate to the next filter
     * @return {@code Mono<Void>} to indicate when request processing is complete
     */
    Mono<Void> execute(ServerWebExchange exchange, ShenyuPluginChain chain);

    /**
     * return plugin order .
     * This attribute To determine the plugin execution order in the same type
plugin.
```

```
     *
     * @return int order
     */
    int getOrder();

    /**
     * acquire plugin name.
     * this is plugin name define you must Provide the right name.
     * if you impl AbstractShenyuPlugin this attribute not use.
     *
     * @return plugin name.
     */
    default String named() {
        return "";
    }

    /**
     * plugin is execute.
     * if return true this plugin can not execute.
     *
     * @param exchange the current server exchange
     * @return default false.
     */
    default Boolean skip(ServerWebExchange exchange) {
        return false;
    }
}
```

- 接口方法详细说明
  - execute() 方法为核心的执行方法，用户可以在里面自由的实现自己想要的功能。
  - getOrder() 指定插件的排序。
  - named() 指定插件的名称，命名采用 Camel Case，如：dubbo、springCloud。
  - skip() 在特定的条件下，该插件是否被跳过。
- 注册成 Spring 的 bean，参考如下，或者直接在实现类上加 @Component 注解。

```
@Bean
public ShenyuPlugin myShenyuPlugin() {
    return new MyShenyuPlugin();
}
```

12.13 匹配流量处理插件

- 引入如下依赖:

```
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-plugin-base</artifactId>
    <version>${project.version}</version>
</dependency>
```

- 新增一个类 CustomPlugin, 继承 org.apache.shenyu.plugin.base.AbstractShenyuPlugin
- 以下是参考:

```
/**
 * This is your custom plugin.
 * He is running in after before plugin, implement your own functionality.
 * extends AbstractShenyuPlugin so you must user shenyu-admin And add related plug-
in development.
 *
 * @author xiaoyu(Myth)
 */
public class CustomPlugin extends AbstractShenyuPlugin {

    /**
     * return plugin order .
     * The same plugin he executes in the same order.
     *
     * @return int
     */
    @Override
    public int getOrder() {
        return 0;
    }

    /**
     * acquire plugin name.
     * return you custom plugin name.
     * It must be the same name as the plug-in you added in the admin background.
     *
     * @return plugin name.
     */
    @Override
    public String named() {
        return "shenYu";
    }

    /**
```

```
     * plugin is execute.
     * Do I need to skip.
     * if you need skip return true.
     *
     * @param exchange the current server exchange
     * @return default false.
     */
    @Override
    public Boolean skip(final ServerWebExchange exchange) {
        return false;
    }

    /**
     * this is Template Method child has Implement your own logic.
     *
     * @param exchange exchange the current server exchange
     * @param chain    chain the current chain
     * @param selector selector
     * @param rule     rule
     * @return {@code Mono<Void>} to indicate when request handling is complete
     */
    @Override
    protected abstract Mono<Void> doExecute(ServerWebExchange exchange,
ShenyuPluginChain chain, SelectorData selector, RuleData rule) {
        LOGGER.debug(".......... function plugin start.............");
        /*
         * Processing after your selector matches the rule.
         * rule.getHandle() is you Customize the json string to be processed.
         * for this example.
         * Convert your custom json string pass to an entity class.
         */
        final String ruleHandle = rule.getHandle();

        final Test test = GsonUtils.getInstance().fromJson(ruleHandle, Test.class);

        /*
         * Then do your own business processing.
         * The last execution  chain.execute(exchange).
         * Let it continue on the chain until the end.
         */
        System.out.println(test.toString());
        return chain.execute(exchange);
    }
}
```

- 详细讲解：
  - 继承该类的插件，插件会进行选择器规则匹配。

  - 首先在 shenyu-admin 后台管理系统-> 基础配置-> 插件管理中，新增一个插件，注意名称与你自定义插件的 named() 方法要一致。
  - 重新登陆 shenyu-admin 后台，可以看见刚新增的插件，然后就可以进行选择器规则匹配。
  - 在规则中，有个 handler 字段，是自定义处理数据，在 doExecute() 方法中，通过 final String ruleHandle = rule.getHandle(); 获取，然后进行你的操作。

- 注册成 Spring 的 bean，参考如下或者直接在实现类上加 @Component 注解。

```
@Bean
public ShenyuPlugin customPlugin() {
    return new CustomPlugin();
}
```

12.14 订阅你的插件数据，进行自定义的处理

- 新增一个类 PluginDataHandler，实现 org.apache.shenyu.plugin.base.handler.PluginDataHandler

```
public interface PluginDataHandler {

    /**
     * Handler plugin.
     *
     * @param pluginData the plugin data
     */
    default void handlerPlugin(PluginData pluginData) {
    }

    /**
     * Remove plugin.
     *
     * @param pluginData the plugin data
     */
    default void removePlugin(PluginData pluginData) {
    }

    /**
     * Handler selector.
     *
     * @param selectorData the selector data
     */
    default void handlerSelector(SelectorData selectorData) {
    }

    /**
     * Remove selector.
```

```
     *
     * @param selectorData the selector data
     */
    default void removeSelector(SelectorData selectorData) {
    }

    /**
     * Handler rule.
     *
     * @param ruleData the rule data
     */
    default void handlerRule(RuleData ruleData) {
    }

    /**
     * Remove rule.
     *
     * @param ruleData the rule data
     */
    default void removeRule(RuleData ruleData) {
    }

    /**
     * Plugin named string.
     *
     * @return the string
     */
    String pluginNamed();

}
```

- 注意 pluginNamed() 要和你自定义的插件名称相同。
- 注册成 Spring 的 bean，参考如下或者直接在实现类上加 @Component 注解。

```
@Bean
public PluginDataHandler pluginDataHandler() {
    return new PluginDataHandler();
}
```

12.15 动态加载自定义插件

- 当使用此功能时候，上述扩展 ShenyuPlugin, PluginDataHandler, 不用成为 spring bean。只需要构建出扩展项目的 jar 包即可。
- 使用以下配置:

```
shenyu:
  extPlugin:
    path:  //加载扩展插件 jar 包路径
    enabled: true //是否开启
    threads: 1  //加载插件线程数量
    scheduleTime: 300 //间隔时间（单位：秒）
    scheduleDelay: 30 //网关启动后延迟多久加载（单位：秒）
```

12.15.1 插件加载路径详解

- 此路径是为存放扩展插件 jar 包的目录。
- 可以使用 -Dplugin-ext=xxxx 指定, 也可以使用 shenyu.extPlugin.path 配置文件指定, 如果都没配置，默认会加载网关启动路径下的 ext-lib 目录。
- 优先级：-Dplugin-ext=xxxx > shenyu.extPlugin.path > ext-lib(default)

12.16 说明

- 本文介绍如何对 org.springframework.web.server.WebFliter 进行扩展。

12.17 跨域支持

- 新增 org.apache.shenyu.web.filter.CrossFilter 实现 WebFilter。

```
public class CrossFilter implements WebFilter {

    private static final String ALLOWED_HEADERS = "x-requested-with, authorization,
Content-Type, Authorization, credential, X-XSRF-TOKEN,token,username,client";

    private static final String ALLOWED_METHODS = "*";

    private static final String ALLOWED_ORIGIN = "*";

    private static final String ALLOWED_EXPOSE = "*";

    private static final String MAX_AGE = "18000";
```

```
    @Override
    @SuppressWarnings("all")
    public Mono<Void> filter(final ServerWebExchange exchange, final WebFilterChain
chain) {
        ServerHttpRequest request = exchange.getRequest();
        if (CorsUtils.isCorsRequest(request)) {
            ServerHttpResponse response = exchange.getResponse();
            HttpHeaders headers = response.getHeaders();
            headers.add("Access-Control-Allow-Origin", ALLOWED_ORIGIN);
            headers.add("Access-Control-Allow-Methods", ALLOWED_METHODS);
            headers.add("Access-Control-Max-Age", MAX_AGE);
            headers.add("Access-Control-Allow-Headers", ALLOWED_HEADERS);
            headers.add("Access-Control-Expose-Headers", ALLOWED_EXPOSE);
            headers.add("Access-Control-Allow-Credentials", "true");
            if (request.getMethod() == HttpMethod.OPTIONS) {
                response.setStatusCode(HttpStatus.OK);
                return Mono.empty();
            }
        }
        return chain.filter(exchange);
    }
}
```

- 将 CrossFilter 注册成为 Spring 的 bean。

12.18 网关过滤 springboot 健康检查

- 注意顺序，使用 @Order 注解

```
@Component
@Order(-99)
public final class HealthFilter implements WebFilter {

    private static final String[] FILTER_TAG = {"/actuator/health", "/health_check
"};

    @Override
    public Mono<Void> filter(@Nullable final ServerWebExchange exchange, @Nullable
final WebFilterChain chain) {
        ServerHttpRequest request = Objects.requireNonNull(exchange).getRequest();
        String urlPath = request.getURI().getPath();
        for (String check : FILTER_TAG) {
            if (check.equals(urlPath)) {
                String result = JsonUtils.toJson(new Health.Builder().up().
build());
                DataBuffer dataBuffer = exchange.getResponse().bufferFactory().
wrap(result.getBytes());
```

```
                return exchange.getResponse().writeWith(Mono.just(dataBuffer));
            }
        }
        return Objects.requireNonNull(chain).filter(exchange);
    }
}
```

12.19 继承 org.apache.shenyu.web.filter.AbstractWebFilter

- 新增一个类继承 AbstractWebFilter，并实现它的两个方法。

```
/**
  * this is Template Method ,children Implement your own filtering logic.
  *
  * @param exchange the current server exchange
  * @param chain    provides a way to delegate to the next filter
  * @return {@code Mono<Boolean>} result: TRUE (is pass), and flow next filter; FALSE
(is not pass) execute doDenyResponse(ServerWebExchange exchange)
  */
 protected abstract Mono<Boolean> doFilter(ServerWebExchange exchange,
WebFilterChain chain);

 /**
  * this is Template Method ,children Implement your own And response client.
  *
  * @param exchange the current server exchange.
  * @return {@code Mono<Void>} response msg.
  */
 protected abstract Mono<Void> doDenyResponse(ServerWebExchange exchange);
```

- doFilter 方法返回 Mono<true> 表示通过，反之则不通过，不通过的时候，会调用 doDenyResponse 输出相关信息到前端。

12.20 说明

- 主要介绍在单机环境下，然后使用本地 API 更新网关数据。
- 统一返回结果:

```
success
```

- 统一请求前缀: localhost:9095/shenyu
- 统一请求头: localKey: 123456

12.21 插件数据

12.21.1 新增或者更新插件

新增或者更新插件

请求方式

POST

请求路径

/plugin/saveOrUpdate

请求参数

名称	类型	是否必需	默认值	描述
PluginData	*PluginData*	True		插件对象（Body 里面传 Json 对象）

PluginData

名称	类型	是否必需	默认值	描述
id	String	False		插件 ID
name	String	True		插件名称
config	String	False		插件配置（Json 格式）
role	String	False		插件角色
enabled	Boolean	False		是否开启

请求示例

POST body

```
{"id":3,"name":"divide","enabled":"true"}
```

12.21.2 清空所有数据

清空所有插件，选择器，规则数据

请求方式

GET

请求路径

/cleanAll

12.21.3 清空插件数据

清空单个插件，选择器，规则数据

请求方式

GET

请求路径

/cleanPlugin?name = xxxx

Request 参数

名称	类型	是否必需	默认值	描述
name	String	true		插件名称

12.21.4 删除插件

删除单个插件 (不包含，插件里面的选择器与规则)

请求方式

GET

请求路径

/plugin/delete?name = xxxx

Request 参数

名称	类型	是否必需	默认值	描述
name	String	true		插件名称

12.21.5 删除所有插件

删除所有插件 (不包含，插件里面的选择器与规则)

请求方式

GET

请求路径

/plugin/deleteAll

12.21.6 获取插件

根据名称获取插件数据

请求方式

GET

请求路径

/plugin/findByName?name=xxxx

Request 参数

名称	类型	是否必需	默认值	描述
name	String	true		插件名称

12.21.7 新增或更新选择器

新增或者更新插件

请求方式

POST

请求路径

/plugin/selector/saveOrUpdate

请求参数

名称	类型	是否必需	默认值	描述
SelectorData	*SelectorData*	True		选择器对象（Body 里面传 Json 对象）

SelectorData

名称	类型	是否必需	默认值	描述
id	String	False		选择器 ID
pluginName	String	True		插件名称
name	String	False		选择器名称（不填则默认生成 plugin:selector+ 随机数字）
matchMode	Integer	False		匹配模式（0：and;1：or），不填默认生成 And 模式
type	Integer	False		流量类型 0：全流量;1：自定义流量）不填默认生成全流量
sort	Integer	False		排序，不填默认生成 10
enabled	Boolean	False		是否开启，不填默认生成 true
logged	Boolean	False		是否打印日志，不填默认生成为 false
handle	String	False		选择器处理（Json 对象，根据每个插件不同，传的对象不同）
conditionList	*Condition*	False		条件集合，自定义流量需要传，全流量不用传（Json List 对象）

Condition

名称	类型	是否必需	默认值	描述
paramType	String	True		参数类型（post, uri, query, host, header, cookie, req_method, domain）
operator	String	True		匹配方式（match, =, regex, >, <, contains, SpEL, Groovy, TimeBefore, TimeAfter）
paramName	String	False		参数名称（uri 参数类型时候，可以不传）
paramValue	Integer	False		匹配值

请求示例

POST body

```
{
    "pluginName": "divide",
    "type": 1,
    "handle": "[{\"upstreamUrl\":\"127.0.0.1:8089\"}]",
    "conditionDataList": [{
        "paramType": "uri",
        "operator": "match",
        "paramName": null,
        "paramValue": "/**"
    }]
}
```

返回数据

选择器 ID

```
xxxxx
```

12.21.8 新增选择器与规则

新增一条选择器与多条规则

请求方式

POST

请求路径

/plugin/selectorAndRules

请求参数

名称	类型	是否必需	默认值	描述
SelectorRulesData	*SelectorRulesData*	True		选择器规则对象（Body 里面传 Json 对象）

SelectorRulesData

名称	类型	是否必需	默认值	描述
pluginName	String	True		插件名称
selectorName	String	False		选择器名称（不填则默认生成 plugin:selector+ 随机数字）
matchMode	Integer	False		匹配模式（0：and;1：or），不填默认生成 And 模式
selectorHandler	String	False		选择器处理（Json 对象，根据每个插件不同，传的对象不同）
conditionList	*ConditionData*	True		选择器条件集合（Json List 对象）
ruleDataList	*RuleLocalData*	True		规则对象集合（Json List 对象）

RuleLocalData

名称	类型	是否必需	默认值	描述
ruleName	String	False		规则名称
ruleHandler	String	True		规则处理（不同的插件传不同的值）
matchMode	Integer	False		匹配模式（0：and;1：or)
conditionList	*ConditionData*	True		规则条件集合（Json List 对象）

ConditionData

名称	类型	是否必需	默认值	描述
paramType	String	True		参数类型（post, uri, query, host, header, cookie, req_method, domain）
operator	String	True		匹配方式（match, =, regex, >, <, contains, SpEL, Groovy, TimeBefore, TimeAfter）
paramName	String	False		参数名称（uri 参数类型时候，可以不传）
paramValue	Integer	False		匹配值

请求示例

POST body

```
{
    "pluginName": "divide",
    "selectorHandler": "[{\"upstreamUrl\":\"127.0.0.1:8089\"}]",
    "conditionDataList": [{
        "paramType": "uri",
        "operator": "match",
        "paramValue": "/http/**"
    }],
    "ruleDataList": [{
        "ruleHandler": "{\"loadBalance\":\"random\"}",
        "conditionDataList": [{
            "paramType": "uri",
            "operator": "=",
            "paramValue": "/http/test/payment"
        }]
    }, {
        "ruleHandler": "{\"loadBalance\":\"random\"}",
        "conditionDataList": [{
            "paramType": "uri",
            "operator": "=",
            "paramValue": "/http/order/save"
        }]
    }]
}
```

12.21.9 删除选择器

根据选择器 id 与插件名称删除选择器

请求方式

GET

请求路径

/plugin/selector/delete?pluginName=xxxx&&id=xxxx

Request 参数

名称	类型	是否必需	默认值	描述
pluginName	String	true		插件名称
id	String	true		选择器 id

12.21.10 获取插件下的所有选择器

根据插件名称获取所有选择器

请求方式

GET

请求路径

/plugin/selector/findList?pluginName=xxxx

Request 参数

名称	类型	是否必需	默认值	描述
pluginName	String	true		插件名称

12.21.11 新增或更新规则

新增或者更新规则数据

请求方式

POST

请求路径

/plugin/rule/saveOrUpdate

请求参数

名称	类型	是否必需	默认值	描述
RuleData	*RuleData*	True		规则对象（Body 里面传 Json 对象）

RuleData

名称	类型	是否必需	默认值	描述
id	String	False		规则 ID
pluginName	String	True		插件名称
name	String	False		规则名称（不填则默认生成 plugin:rule+ 随机数字）
selectorId	String	True		选择器 ID（不填则默认生成 plugin:rule+ 随机数字）
matchMode	Integer	False		匹配模式（0：and;1：or），不填默认生成 And 模式
sort	Integer	False		排序，不填默认生成 10
enabled	Boolean	False		是否开启，不填默认生成 true
logged	Boolean	False		是否打印日志，不填默认生成为 false
handle	String	False		规则处理（Json 对象，根据每个插件不同，传的对象不同）
conditionList	*ConditionData*	False		条件集合（Json List 对象）

---

conditionList

名称	类型	是否必需	默认值	描述
paramType	String	True		参数类型（post, uri, query, host, header, cookie, req_method, domain）
operator	String	True		匹配方式（match, =, regex, >, <, contains, SpEL, Groovy, TimeBefore, TimeAfter）
paramName	String	False		参数名称（uri 参数类型时候，可以不传）
paramValue	Integer	False		匹配值

请求示例

POST body

```
{
    "pluginName": "divide",
    "selectorId": 123456,
    "handle": "{\"loadBalance\":\"random\"}",
    "conditionDataList": [{
        "paramType": "uri",
        "operator": "=",
        "paramValue": "/test"
    }]
}
```

返回数据

规则 ID

```
xxxxx
```

12.21.12 删除规则

根据选择器 id 与规则 id 删除规则

请求方式

GET

请求路径

/plugin/rule/delete?selectorId=xxxx&&id=xxxx

Request 参数

名称	类型	是否必需	默认值	描述
selectorId	String	true		选择器 ID
id	String	true		规则 ID

12.21.13 获取规则集合

根据选择器 ID 获取所有规则

请求方式

GET

请求路径

/plugin/rule/findList?selectorId=xxxx

Request 参数

名称	类型	是否必需	默认值	描述
selectorId	String	true		选择器 ID

12.22 元数据

12.22.1 新增或者更新元数据

新增或者更新元数据

请求方式

POST

请求路径

/meta/saveOrUpdate

请求参数

名称	类型	是否必需	默认值	描述
MetaData	*MetaData*	True		元数据对象（Body 里面传 Json 对象）

MetaData

名称	类型	是否必需	默认值	描述
id	String	False		元数据 ID
appName	String	True		应用名称
contextPath	String	True		contextPath
path	String	True		请求路径
rpcType	String	True		rpc 类型（dubbo，sofa，tars，springCloud，motan，grpc）
serviceName	String	True		接口名称
methodName	String	True		方法名称
parameterTypes	String	True		参数类型
rpcExt	String	False		rpc 扩展参数（json 对象）
enabled	Boolean	False		是否开启

12.22.2 删除元数据

删除元数据

请求方式

GET

请求路径

/meta/delete?rpcType=xxxx&&path=xxx

Request 参数

名称	类型	是否必需	默认值	描述
rpcType	String	true		rpc 类型（dubbo，sofa，tars，springCloud，motan，grpc）
path	String	true		路径

12.23 签名数据

12.23.1 新增或者更新

新增或者更新签名数据

请求方式

POST

请求路径

/auth/saveOrUpdate

请求参数

名称	类型	是否必需	默认值	描述
AppAuthData	*AppAuthData*	True		签名对象（Body 里面传 Json 对象）

AppAuthData

名称	类型	是否必需	默认值	描述
appKey	String	True		app key
appSecret	String	True		app secret
enabled	Boolean	False		是否开启
open	Boolean	False		是否是开放平台
paramDataList	*AuthParamData*	false		参数集合，open 为 true 时候需要传（Json list 对象）
AuthPathData	*AuthPathData*	false		路径集合，open 为 true 时候需要传（Json list 对象）

AuthParamData

名称	类型	是否必需	默认值	描述
appName	String	True		应用名称
appParam	String	True		应用参数

AuthPathData

名称	类型	是否必需	默认值	描述
appName	String	True		应用名称
path	String	True		路径
enabled	Boolean	False		是否开启

12.23.2 删除

删除签名数据

请求方式

GET

请求路径

/auth/delete?appKey=xxxx

Request 参数

名称	类型	是否必需	默认值	描述
appKey	String	true		app key

12.24 说明

- 本文主要介绍如何对 Apache ShenYu 进行优化。

12.25 本身消耗

- Apache ShenYu 本身所有的操作，都是基于 JVM 内存来匹配，本身消耗时间大概在 1-3ms 左右。

12.26 底层 Netty 调优

- Apache ShenYu 内置依赖 spring-webflux 而其底层是使用的 netty 。
- 我们可以自定义 netty 的相关参数来对 Apache ShenYu 进行优化，以下是示例:

```
@Bean
 public NettyReactiveWebServerFactory nettyReactiveWebServerFactory() {
     NettyReactiveWebServerFactory webServerFactory = new
NettyReactiveWebServerFactory();
     webServerFactory.addServerCustomizers(new EventLoopNettyCustomizer());
     return webServerFactory;
 }

 private static class EventLoopNettyCustomizer implements NettyServerCustomizer {

     @Override
     public HttpServer apply(final HttpServer httpServer) {
         return httpServer
                 .tcpConfiguration(tcpServer -> tcpServer
                         .runOn(LoopResources.create("shenyu-netty", 1, DEFAULT_IO_
WORKER_COUNT, true), false)
```

```
                        .selectorOption(ChannelOption.SO_REUSEADDR, true)
                        .selectorOption(ChannelOption.ALLOCATOR,
PooledByteBufAllocator.DEFAULT)
                        .option(ChannelOption.TCP_NODELAY, true)
                        .option(ChannelOption.ALLOCATOR, PooledByteBufAllocator.
DEFAULT));
     }
 }
```

- 这个类在 shenyu-bootstrap 中已经内置，在压测的时候，可以根据自己的需求来进行优化设置。
- 业务线程模型，请参考：线程模型 。

12.27 说明

- 本文介绍基于 Apache ShenYu 网关返回自定义的数据格式。
- 网关需要统一的返回格式，如果需要自己定义格式，可以进行扩展。

12.28 默认实现

- 默认的实现为 org.apache.shenyu.plugin.api.result.DefaultShenyuResult
- 返回的数据格式如下：

```
public class DefaultShenyuEntity implements Serializable {

    private static final long serialVersionUID = -2792556188993845048L;

    private Integer code;

    private String message;

    private Object data;

}
```

- 返回的 json 格式如下：

```
{
    "code": -100, //返回码,
    "message": " 您的参数错误，请检查相关文档!", //提示字段
    "data": null  // 具体的数据
}
```

12.29 扩展

- 新增一个类 CustomShenyuResult 实现 org.apache.shenyu.plugin.api.result.ShenyuResult

```
/**
 * The interface shenyu result.
 */
public interface ShenyuResult<T> {

    /**
     * The response result.
     *
     * @param exchange the exchange
     * @param formatted the formatted object
     * @return the result object
     */
    default Object result(ServerWebExchange exchange, Object formatted) {
        return formatted;
    }

    /**
     * format the origin, default is json format.
     *
     * @param exchange the exchange
     * @param origin the origin
     * @return format origin
     */
    default Object format(ServerWebExchange exchange, Object origin) {
        // basic data
        if (ObjectTypeUtils.isBasicType(origin)) {
            return origin;
        }
        // error result or rpc origin result.
        return JsonUtils.toJson(origin);
    }

    /**
     * the response context type, default is application/json.
     *
     * @param exchange the exchange
     * @param formatted the formatted data that is origin data or byte[] convert
string
     * @return the context type
     */
    default MediaType contentType(ServerWebExchange exchange, Object formatted) {
        return MediaType.APPLICATION_JSON;
    }
```

```
    /**
     * Error t.
     *
     * @param code    the code
     * @param message the message
     * @param object  the object
     * @return the t
     */
    T error(int code, String message, Object object);
}
```

处理顺序：format->“contextType“->“result“。format 方法进行数据的格式化，若数据为基本类型返回自身，其他类型转换为 JSON，参数 origin 为原始数据，可根据情况执行格式化处理。contextType，若是基本类型，使用 text/plain，默认为 application/json，参数 formatted 为 format 方法处理之后的数据，可结合 format 的返回结果进行数据类型的自定义处理。result 的参数 formatted 为 format 方法处理之后的数据，默认返回自身，可结合 format 的返回结果进行数据类型的自定义处理。

- 其中泛型 T 为自定义的数据格式，返回它就好。
- 把你新增的实现类注册成为 Spring 的 bean，如下：

```
@Bean
public ShenyuResult<?> customShenyuResult() {
    return new CustomShenyuResult();
}
```

12.30 说明

- 本文主要介绍 Apache ShenYu 的线程模型，以及各种场景的使用。

12.31 IO 与 Work 线程

- Apache ShenYu 内置依赖 spring-webflux，而其底层是使用的是 netty，这一块主要是使用的 netty 线程模型。

12.32 业务线程

- 默认使用调度线程来执行。
- 默认使用固定的线程池来执行，其线程数为 cpu * 2 + 1。

12.33 切换类型

- reactor.core.scheduler.Schedulers。
- 可以使用 -Dshenyu.scheduler.type=fixed 这个是默认。设置其他的值就会使用弹性线程池来执行 Schedulers.elastic()。
- 可以使用 -Dshenyu.work.threads = xx 来指定线程数量, 默认为 cpu * 2 + 1, 最小为 16 个线程。

12.34 说明

- 用户可以自定义签名认证算法来实现验证。

12.35 扩展

- 默认的实现为 org.apache.shenyu.plugin.sign.service.DefaultSignService。
- 新增一个类 CustomSignService 实现 org.apache.shenyu.plugin.sign.api.SignService。

```
public interface SignService {

    /**
     * Sign verify pair.
     *
     * @param exchange   the exchange
     * @return the pair
     */
    Pair<Boolean, String> signVerify(ServerWebExchange exchange);
}
```

- Pair 中返回 true，表示验证通过，为 false 的时候，会把 String 中的信息输出到前端。

- 把新增的实现类注册成为 Spring 的 bean，如下

```
@Bean
public SignService customSignService() {
    return new CustomSignService();
}
```

12.36 其他扩展

当你只希望修改签名算法时可以参考如下。

- 签名算法，默认使用的 org.apache.shenyu.common.utils.SignUtils#generateSign，还可以新增一个类 CustomSignProvider 实现 org.apache.shenyu.plugin.sign.api.SignProvider.

```
/**
 * The Sign plugin sign provider.
 */
public interface SignProvider {

    /**
     * acquired sign.
     *
     * @param signKey sign key
     * @param params  params
     * @return sign
     */
    String generateSign(String signKey, Map<String, String> params);
}
```

- 把新增的实现类 CustomSignProvider 注入到 Spring IoC 即可，如下

```
@Bean
public SignProvider customSignProvider() {
    return new CustomSignProvider();
}
```