Apache ShenYu document

Apache ShenYu

Jul 01, 2022

Contents

1

What is the Apache ShenYu?

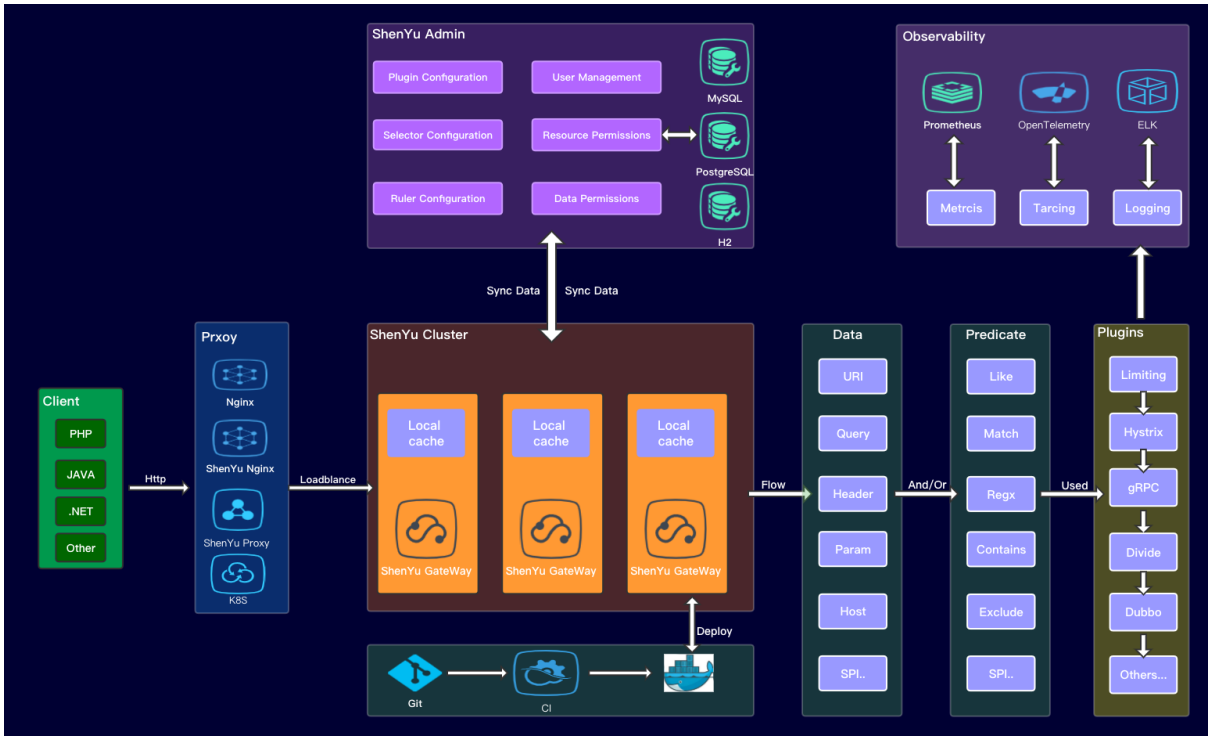
This is an asynchronous, high-performance, cross-language, responsive API gateway.

2 Features

- Support various languages (http protocol), support Dubbo, Spring Cloud, gRPC, Motan, Sofa, Tars and other protocols.
- Plugin design idea, plugin hot swap, easy to expand.
- Flexible flow filtering to meet various flow control.
- Built-in rich plugin support, authentication, limiting, fuse, firewall, etc.
- Dynamic flow configuration, high performance.
- Support cluster deployment, A/B Test, blue-green release.

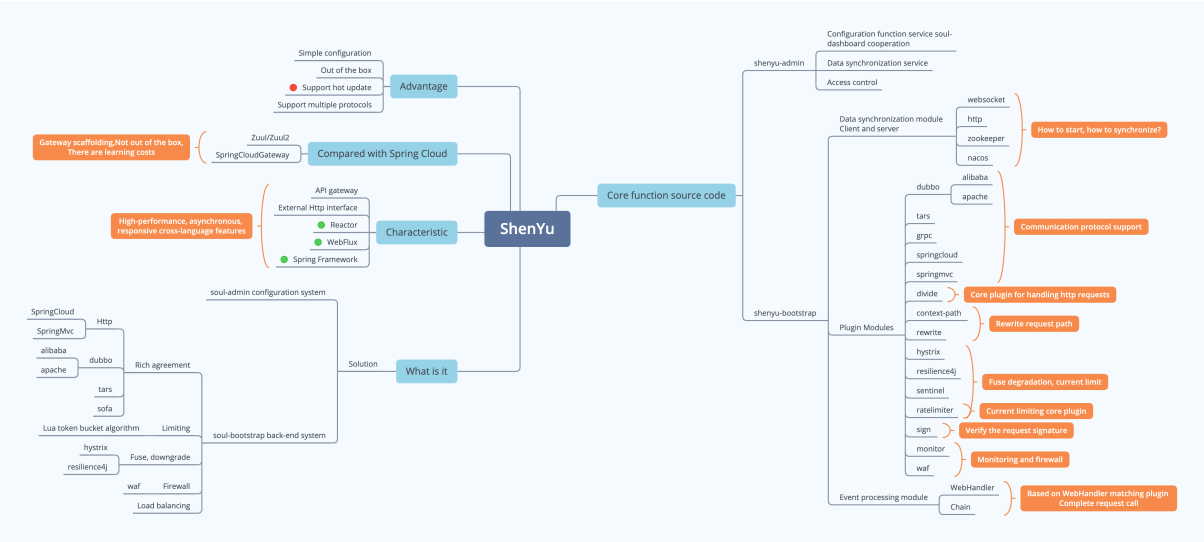
3

Architecture Diagram



4

Mind maps


ShenYu
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
What is it
Solution
soul-admin configuration system
soul-bootstrap back-end system
Rich agreement
SpringCloud
SpringMvc
Http
alibaba
apache
dubbo
tars
sofa
Limiting
Lua token bucket algorithm
Fuse, downgrade
hystrix
resilience4j
Firewall
waf
Load balancing
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
Fuse degradation, current limit
ratelimiter
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 Modules

- shenyu-admin : plugins and other configuration information management background
- shenyu-bootstrap : with the startup project, users can refer to
- shenyu-client : user fast access with Spring MVC, Dubbo, Spring Cloud.
- shenyu-common : framework common class
- shenyu-disruptor : based on disruptor Enclosure
- shenyu-register-center : rpc type register for shenyu-client
- shenyu-dist : build project
- shenyu-metrics : metrics impl by prometheus.
- shenyu-plugin : ShenYu provider plugin collection.
- shenyu-spi : ShenYu spi define.
- shenyu-spring-boot-starter : support for the spring boot starter
- shenyu-sync-data-center : provider ZooKeeper, HTTP, WebSocket, Nacos to sync data
- shenyu-examples : the RPC examples project
- shenyu-web : core processing packages including plugins, request routing and forwarding, and so on

6 About

Apache ShenYu has been used widely in more and more systems in many companies, and it’ s simple and convenient to integrate Services/APIs with the high performance and flexibility.

In double eleven online shopping carnival of China, ShenYu clusters successfully supported a large volume of internet business.

7 Design

This document explains the principle of data synchronization. Data synchronization refers to the strategy used to synchronize data to ShenYu gateway after shenyu-admin background operation data. ShenYu gateway currently supports ZooKeeper, WebSocket, HTTP Long Polling, Nacos, Etcd and Consul for data synchronization.

See Data Synchronization Configuration for configuration information about data synchronization.

7.1 Preface

Gateway is the entrance of request and it is a very important part in micro service architecture, therefore the importance of gateway high availability is self-evident. When we use gateway, we have to change configuration such as flow rule, route rule for satisfying business requirement. Therefore, the dynamic configuration of the gateway is an important factor to ensure the high availability of the gateway.

In the actual use of Apache ShenYu Gateway, users also feedback some problems:

- Apache ShenYu depends on ZooKeeper, how to use Etcd, Consul, Nacos and other registry center?
- Apache ShenYu depends on Redis and InfluxDB, and do not use limiting plugins or monitoring plugins. Why need these?
- Why not use configuration center for configuration synchronization?
- Why can’t updates be configured dynamically?
- Every time you want to query the database, Redis is a better way.

According to the feedback of users, we have also partially reconstructed ShenYu. The current data synchronization features are as follows:

- All configuration is cached in ShenYu gateway memory, each request uses local cache, which is very fast.
- Users can modify any data in the background of shenyu-admin, and immediately synchronize to the gateway memory.

- Support ShenYu plugin, selector, rule data, metadata, signature data and other data synchronization.
- All plugin selectors and rules are configured dynamically and take effect immediately, no service restart required.
- Data synchronization mode supports Zookeeper, HTTP long polling, Websocket, Nacos, Etcd and Consul.

7.2 Principle Analysis

The following figure shows the process of data synchronization of ShenYu. ShenYu Gateway will synchronize configuration data from configuration service at startup, and support push-pull mode to get configuration change information, and then update local cache. The administrator can change the user permissions, rules, plugins and traffic configuration in the admin system(shenyu-admin), and synchronize the change information to ShenYu Gateway through the push-pull mode. Whether the mode is push or pull depends on the synchronization mode used.

In the original version, the configuration service relied on the Zookeeper implementation to manage the back-end push of changes to the gateway. Now, WebSocket, HTTP long polling, ZooKeeper, Nacos, Etcd, and Consul can now be supported by specifying the corresponding synchronization policy by setting `shenyu.sync.${strategy}` in the configuration file. The default WeboSocket synchronization policy can be used to achieve second level data synchronization. However, it is important to note that Apache ShenYu Gateway and shenyu-admin must use the same synchronization policy.

As showing picture below,`shenyu-admin` will issue a configuration change notification through `EventPublisher` after users change configuration,`EventDispatcher` will handle this modification and send configuration to corresponding event handler according to configured synchronization strategy.

- If it is a `websocket` synchronization strategy,it will push modified data to `shenyu-web`,and corresponding `WebsocketDataHandler` handler will handle `shenyu-admin` data push at the gateway layer
- If it is a `zookeeper` synchronization strategy,it will push modified data to `zookeeper`,and the `ZookeeperSyncCache` will monitor the data changes of `zookeeper` and process them
- If it is a `http` synchronization strategy,`shenyu-web` proactively initiates long polling requests,90 seconds timeout by default,if there is no modified data in `shenyu-admin`,http request will be blocked,if there is a data change, it will respond to the changed data information,if there is no data change after 60 seconds,then respond with empty data,gateway continue to make http request after getting response,this kind of request will repeat.

7.2.1 Zookeeper Synchronization

The zookeeper-based synchronization principle is very simple,it mainly depends on `zookeeper` watch mechanism,`shenyu-web` will monitor the configured node,when `shenyu-admin` starts,all the data will be written to `zookeeper`,it will incrementally update the nodes of `zookeeper` when data changes,at the same time, `shenyu-web` will monitor the node for configuration information, and update the local cache once the information changes

`Apache ShenYu` writes the configuration information to the zookeeper node,and it is meticulously designed. If you want to learn more about the code implementation, refer to the source code `ZookeeperSyncDataService`.

7.2.2 WebSocket Synchronization

The mechanism of `websocket` and `zookeeper` is similar,when the gateway and the `shenyu-admin` establish a `websocket` connection,`shenyu-admin` will push all data at once,it will automatically push incremental data to `shenyu-web` through `websocket` when configured data changes

When we use websocket synchronization,pay attention to reconnect after disconnection,which also called keep heartbeat.`Apache ShenYu` uses `java-websocket` ,a third-party library,to connect to `websocket`. If you want to learn more about the code implementation, refer to the source code `WebsocketSyncDataService`.

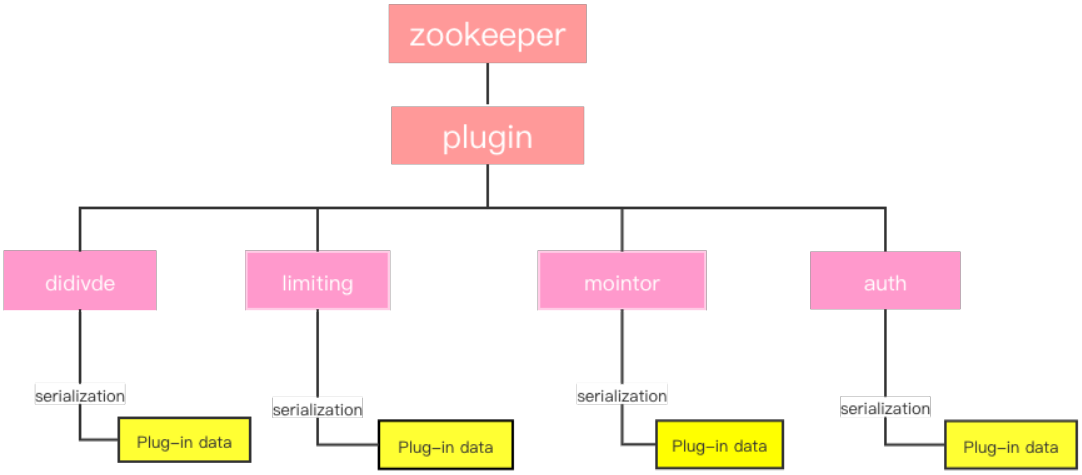
7.2.3 Http Long Polling

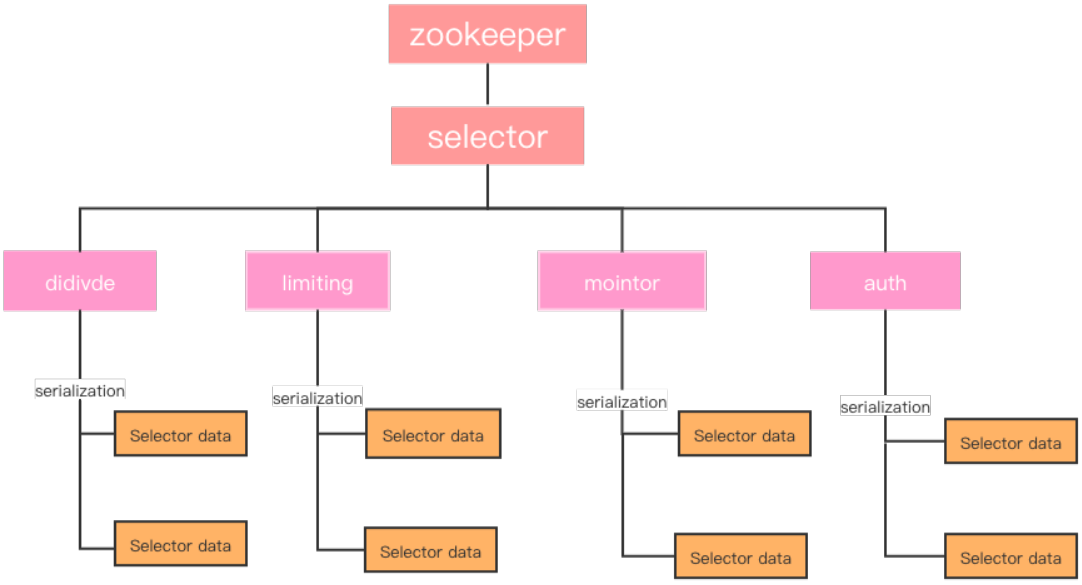
The mechanism of zookeeper and websocket data synchronization is relatively simple,but http synchronization will be relatively complicated.ShenYu borrows the design ideas of `Apollo` and `Nacos` and realizes `http` long polling data synchronization using their advantages.Note that this is not traditional ajax long polling.

http long polling mechanism as above,shenyu-web gateway requests shenyu-admin configuration services,timeout is 90 seconds,it means gateway layer request configuration service will wait at most 90 seconds,this is convenient for shenyu-admin configuration service to respond modified data in time,and therefore we realize near real-time push.

After the http request reaches shenyu-admin, it does not respond immediately,but uses the asynchronous mechanism of Servlet3.0 to asynchronously respond to the data.First of all,put long polling request task `LongPollingClient` into `BlocingQueue`,and then start scheduling task,execute after 60 seconds,this aims to remove the long polling request from the queue after 60 seconds,even there is no configured data change.Because even if there is no configuration change,gateway also need to know,otherwise it will wait,and there is a 90 seconds timeout when the gateway requests configuration services.

If the administrator changes the configuration data during this period,the long polling requests in the queue will be removed one by one, and respond which group’ s data has changed(we distribute plugins, rules, flow configuration , user configuration data into different groups).After gateway receives response,it only knows which Group has changed its configuration,it need to request again to get group





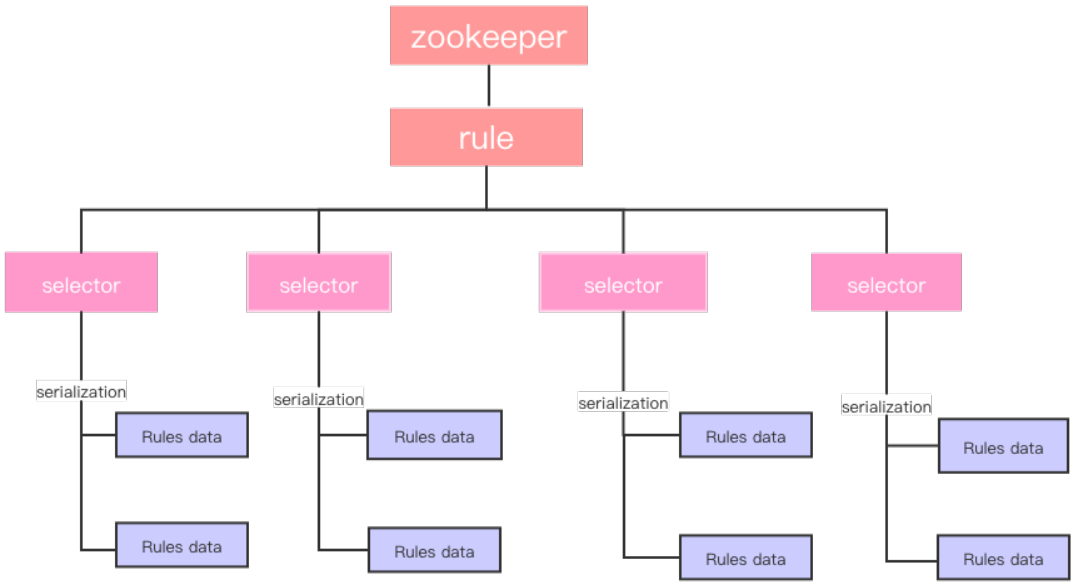


Figure1: Zookeeper Node Design

configuration data.Someone may ask,why don't you write out the changed data directly?We also discussed this issue deeply during development, because the http long polling mechanism can only guarantee quasi real-time,if gateway layer does not handle it in time,or administrator updates configuration frequently,we probably missed some configuration change push.For security, we only inform that a certain Group information has changed.

When `shenyu-web` gateway layer receives the http response information,pull modified information(if exists),and then request `shenyu-admin` configuration service again,this will repeatedly execute. If you want to learn more about the code implementation, refer to the source code `HttpSyncDataService`.

7.2.4 Nacos Synchronization

The synchronization principle of Nacos is basically similar to that of ZooKeeper, and it mainly depends on the configuration management of Nacos. The path of each configuration node is similar to that of ZooKeeper.

ShenYu gateway will monitor the configured node. At startup, if there is no configuration node in Nacos, it will write the synchronous full amount of data into Nacos. When the sequential data send changes, it will update the configuration node in Nacos in full amount. The local cache is updated.

If you want to learn more about the code implementation, please refer to the source code `NacosSyncDataService` and the official documentation for Nacos .

7.2.5 Etcd Synchronization

Etcd data synchronization principle is similar to Zookeeper, mainly relying on Etcd's watch mechanism, and each configuration node path is the same as that of Zookeeper.

The native API for Etcd is a bit more complicated to use, so it's somewhat encapsulated.

ShenYu gateway will listen to the configured node. When startup, if there is no configuration node in Etcd, it will write the synchronous full amount of data into Etcd. When the sequential data send changes, it will update the configuration node in Etcd incrementally.

If you want to learn more about the code implementation, refer to the source `EtcdSyncDataService`.

7.2.6 Consul Synchronization

Consul data synchronization principle is that the gateway regularly polls Consul's configuration center to get the configuration version number for local comparison.

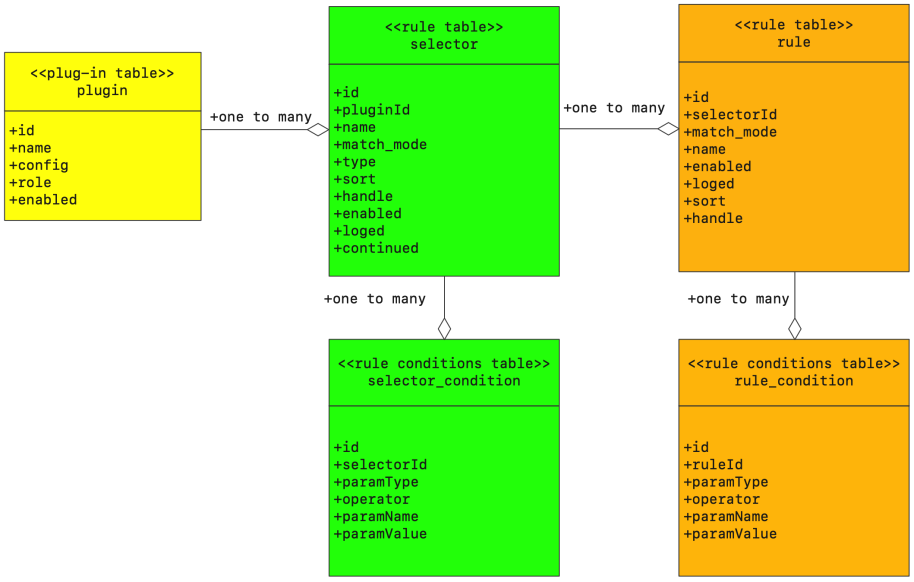
ShenYu gateway will poll the configured nodes regularly, and the default interval is 1s. When startup, if there is no configuration node in Consul, write the synchronous full amount of data into Consul, then incrementally update the configuration node in Consul when the subsequent data is sent to change. At the same time, Apache ShenYu Gateway will regularly polls the node of configuration information and pull the configuration version number for comparison with the local one. The local cache is updated when the version number is changed.

If you want to learn more about the code implementation, refer to the source `ConsulsyncDataService`.

Apache Shenyu Admin is the management system of the gateway, which can manage all plugins, selectors and rules visually, set users, roles and resources.

7.3 Plugin, Selector And Rule

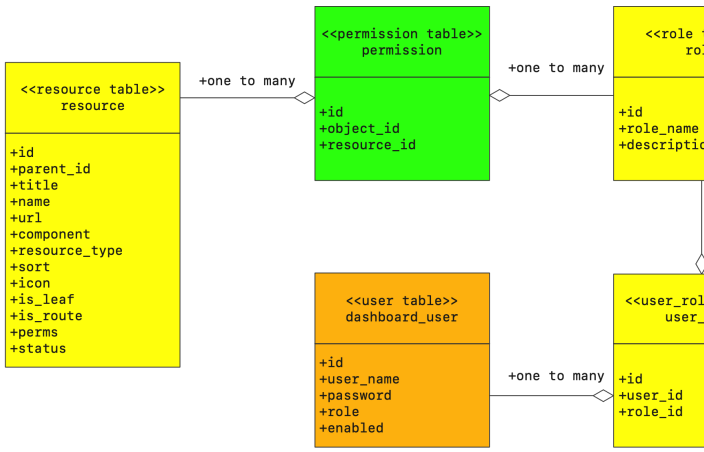
- Plugin: ShenYu uses the plugin design idea to realize the hot plug of the plugin, which is easy to expand. Built-in rich plugins, including RPC proxy, circuit breaker and current limiting, authority and certification, monitoring, and more.
- Selector: Each plugin can set multiple selectors to carry out preliminary filtering of traffic.
- Rule: Multiple rules can be set per selector for more fine-grained control of flow.
- The Database Table UML Diagram:



- Detailed design:
  - One plugin corresponds to multiple selectors,one selector corresponds to multiple rules.
  - One selector corresponds to multiple match conditions,one rule corresponds to multiple match conditions.
  - Each rule handles differently in corresponding plugin according to field handler,field handler is a kind of data of JSON string type.You can view detail during the use of shenyu-admin.

7.4 Resource Permission

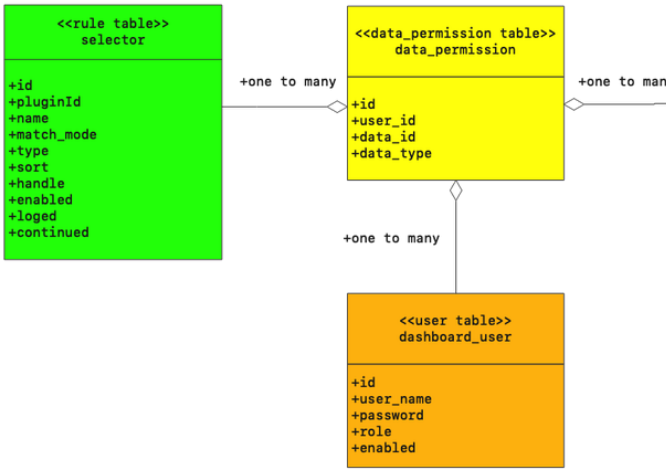
- The resource are the menus and buttons in the shenyu-admin console.
- Resource Permission use database to store user name,role,resource data and relationship.



- The Resource Permission Table UML Diagram:
- Detailed design:
  - one user corresponds to multiple role,one role corresponds to multiple resources.

7.5 Data Permissin

- Data Permission use database to store the relationship between users, selectors and rules.



- The Data Permission Table UML Diagram:
- Detailed design:
  - The most important table is `data_permission`, where a user corresponds to multiple data permissions.
  - The field `data_type` distinguishes between different types of data, which corresponds to the following: 0 -> selector, 1 -> rule.
  - The field `data_id` holds the primary key id of the corresponding type.

7.6 Meta Data

- Metadata is used for generic invoke by gateway.
- For each interface method, there is one piece of metadata.
- The Database Table UML Diagram:
- Detailed design:
  - `path`: When the gateway is requested, a piece of data will be matched according to `path`, and then the subsequent process will be carried out.
  - `rpc_ext`: Used to hold extended information for the RPC proxy.。

7.7 Dictionary Management

- Dictionary management is used to maintain and manage public data dictionaries.
- The Database Table UML Diagram:

ShenYu gateway realizes flow control through plugins, selectors and rules. For related data structure, please refer to the previous Apache ShenYu Admin Database Design .

7.8 Plugin

In Apache ShenYu Admin System, each plugin uses Handle (JSON format) fields to represent different processing, and the plugin processing is used to manage and edit the custom processing fields in the JSON.

The main purpose of this feature is to enable plugins to handle templated configurations.

7.9 Selector And Rule

Selector and rule are the most soul of Apache ShenYu Gateway. Master it and you can manage any traffic.

A plugin has multiple selectors, and one selector corresponds to multiple rules. The selector is the first level filter of traffic, and the rule is the final filter. For a plugin, we want to meet the traffic criteria based on our configuration before the plugin will be executed. Selectors and rules are designed to allow traffic to perform what we want under certain conditions. The rules need to be understood first.

The execution logic of plugin, selector and rule is as follows. When the traffic enters into ShenYu gateway, it will first judge whether there is a corresponding plugin and whether the plugin is turned on. Then determine whether the traffic matches the selector of the plugin. It then determines whether the traffic matches the rules of the selector. If the request traffic meets the matching criteria, the plugin will be executed. Otherwise, the plugin will not be executed. Process the next one. ShenYu gateway is so through layers of screening to complete the flow control.

7.10 Traffic filtering

Traffic filtering is the soul of the selector and the rule, corresponding to the matching conditions in the selector and the rule. According to different traffic filtering rules, we can deal with various complex scenes. Traffic filtering can fetch data from Http requests such as `Header`, `URI`, `Query`, `Cookie`, etc.

You can then use `Match`, `=`, `SpEL`, `Regex`, `Groovy`, `Exclude`, etc, to Match the desired data. Multi-group matching Adds matching policies that can use And/Or.

please refre to Selector And Rule Config for details.

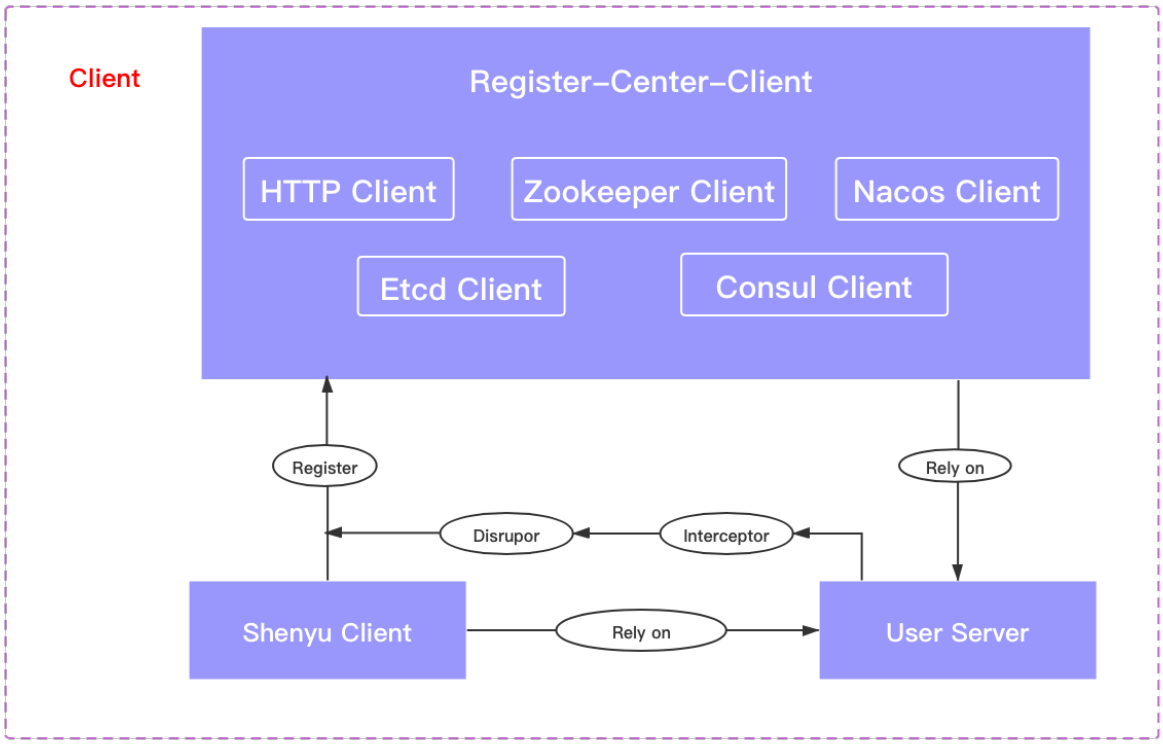
Application client access means to access your microservice to ShenYu gateway, currently supports HTTP, Dubbo, Spring Cloud, gRPC, Motan, Sofa, Tars and other protocols access.

Connecting the application client to ShenYu gateway is realized through the registration center, which involves the registration of the client and the synchronization of the server data. The registry supports HTTP, ZooKeeper, Etcd, Consul, and Nacos.

Refer to the client access configuration in the user documentation for Application Client Access Config
.

7.11 Design principle

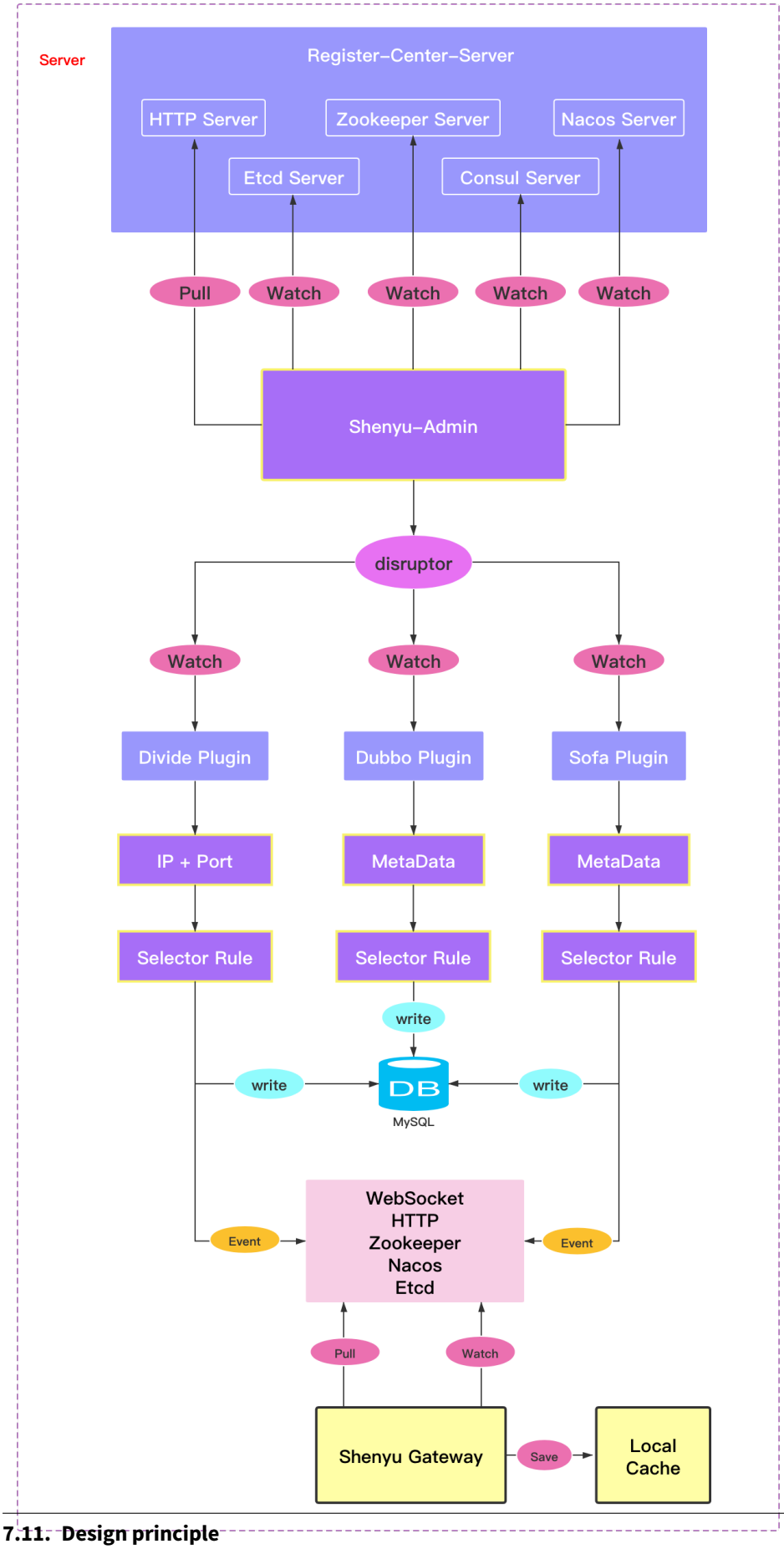
7.11.1 Client



Declare the registry client type, such as HTTP or ZooKeeper, in your microservice configuration. Use SPI to load and initialize the corresponding registry client when the application starts, implement the post-processor interface associated with the Spring Bean, get the service interface information to register in it, and place the obtained information into Disruptor.

The Registry client reads data from the Disruptor and registers the interface information with shenyu-admin, where the Disruptor decouples data from operations for scaling.

7.11.2 Server


Server
Register-Center-Server
HTTP Server
Zookeeper Server
Nacos Server
Etcd Server
Consul Server
Pull
Watch
Watch
Watch
Watch
Shenyu-Admin
disruptor
Watch
Watch
Watch
Divide Plugin
Dubbo Plugin
Sofa Plugin
IP + Port
MetaData
MetaData
Selector Rule
Selector Rule
Selector Rule
write
write
DB
write
MySQL
WebSocket
HTTP
Zookeeper
Nacos
Etcd
Event
Event
Pull
Watch
Shenyu Gateway
Save
Local
Cache

Declare the registry server type, such as HTTP or ZooKeeper, in the Shenyu-Admin configuration. When shenyu-admin is started, it will read the configuration type, load and initialize the corresponding registry server, and when the registry server receives the interface information registered by shenyu-client, it will put it into Disruptor, which will trigger the registration processing logic to update the interface information and publish a synchronous event.

Disruptor provides data and operations decoupling for expansion. If there are too many registration requests, resulting in abnormal registration, there is also a data buffer role.

7.12 Http Registry

The principle of HTTP service registration is relatively simple. After Shenyu-Client is started, the relevant service registration interface of Shenyu-Admin will be called to upload data for registration.

After receiving the request, shenyu-admin will update the data and publish the data synchronization event to synchronize the interface information to ShenYu Gateway.

7.13 Zookeeper Registry

Zookeeper storage struct is:

```
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 starts up, the service interface information (MetaDataRegisterDTO/URIRegisterDTO) wrote above the Zookeeper nodes.

shenyu-admin uses the Watch mechanism of Zookeeper to monitor events such as data update and deletion, and triggers the corresponding registration processing logic after data changes. Upon receipt of a change to the MetadataregisterDTO node, the data change and data synchronization event publication of the selector and rule is triggered. Upon receipt of a UriRegisterDTO node change, the upstream of the selector is triggered to publish an update and data synchronization event.

7.14 Etcd Registry

Etcd storage struct is:

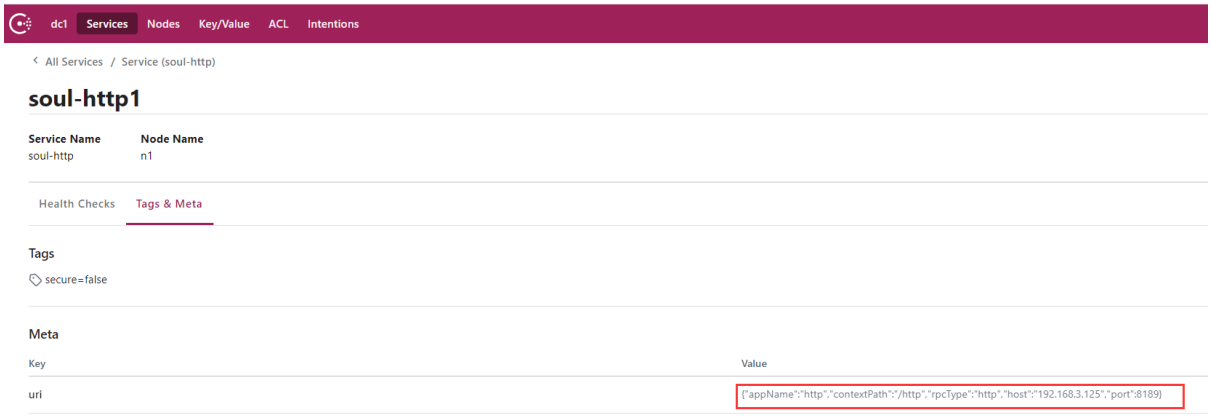
```
shenyu
   ├──regsiter
   ├    ├──metadata
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├     ├              ├──${ruleName} : save metadata data of
MetaDataRegisterDTO
   ├    ├──uri
   ├    ├     ├──${rpcType}
   ├    ├     ├      ├────${contextPath}
   ├    ├    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

shenyu-client starts up, the service interface information (MetaDataRegisterDTO/URIRegisterDTO) wrote in Ephemeral way above Etcd of the node.

shenyu-admin uses Etcd' s Watch mechanism to monitor events such as data update and deletion, and triggers the corresponding registration processing logic after data changes. Upon receipt of a change to the MetadataregisterDTO node, the data change and data synchronization event publication of the selector and rule is triggered. Upon receipt of a UriRegisterDTO node change, the upstream of the selector is triggered to publish an update and data synchronization event.

7.15 Consul Registry

Consul register client will save URIRegisterDTO to service instance metadata, and URIRegisterDTO will disappear with service unregister.



And Consul register client will save MetaDataRegisterDTO to Key/Value store, storage struct is:

```
shenyu
   ├──regsiter
   ├    ├──metadata
```

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

When shenyu-client is started, The service interface information (MetaDataRegisterDTO/URIRegisterDTO) on the Metadata of the ServiceInstance (URIRegisterDTO) and Key-Value (MetaDataRegisterDTO), Store as described above.

shenyu-admin senses the update and deletion of data by monitoring the change of index of Catalog and KeyValue, and triggers the corresponding registration processing logic after the change of data. Upon receipt of a change to the MetadataregisterDTO node, the data change and data synchronization event publication of the selector and rule is triggered. Upon receipt of a UriRegisterDTO node change, the upstream of the selector is triggered to publish an update and data synchronization event.

7.16 Nacos Register

Nacos registration is divided into two parts: URI and Metadata. URI is registered by instance. In case of service exception, the relevant URI data node will be deleted automatically and send events to the subscriber, and the subscriber will carry out relevant offline processing. Metadata is registered by configuration without any related up-down operation. When a URI instance is registered, the Metadata configuration will be published accordingly. The subscriber monitors data changes and carries out update processing.

The URI instance registration command rules are as follows:

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

Listens on all RpcType nodes initially, and the `${contextPath}` instances registered under them are distinguished by IP and Port, and carry their corresponding contextPath information. After the URI instance is offline, it triggers the update and data synchronization event publication of the selector’s upstream.

When the URI instance goes online, the corresponding Metadata data will be published. The node name command rules are as follows:

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

The subscriber side continues to listen for all Metadata configurations, triggering selector and rule data changes and data synchronization events after the initial subscription and configuration update.

7.17 SPI

SPI Name	*Description*
ShenyuClientRegisterRepository	ShenYu client register SPI

Implementation Class	*Description*
HttpClientRegisterRepository	Http client register repository
ZookeeperClientRegisterRepository	Zookeeper client register repository
EtcdClientRegisterRepository	Etcd client register repository
ConsulClientRegisterRepository	Consul client register repository
NacosClientRegisterRepository	Nacos client register repository

SPI Name	*Description*
ShenyuServerRegisterRepository	ShenYu server register SPI

Implementation Class	*Description*
ShenyuHttpRegistryController	Http server repository
ZookeeperServerRegisterRepository	Zookeeper server registry repository
EtcdServerRegisterRepository	Etcd server registry repository
ConsulServerRegisterRepository	Consul server registry repository
NacosServerRegisterRepository	Nacos server registry repository

SPI, called Service Provider Interface, is a built-in JDK Service that provides discovery function and a dynamic replacement discovery mechanism.

shenyu-spi is a custom SPI extension implementation for Apache Shenyu gateway. The design and implementation principles refer to SPI Extension Implementations .

7.18 Registry Center

`Consul`, `Etcd`, `Http`, `Nacos` and `Zookeeper` are supported. The expansion of the registry including client and server, interface respectively `ShenyuServerRegisterRepository` and `ShenyuClientRegisterRepository`.

7.19 Metrics Center

Responsible for service monitoring, loading concrete implementation through SPI, currently support Prometheus, service interface is MetricsBootService.

7.20 Load Balance

Select one of the service providers to call. Currently, the supported algorithms are Has, Random, and RoundRobin, and the extended interface is LoadBalance.

7.21 RateLimiter

In the RateLimiter plugin, which stream limiting algorithm to use, currently supporting Concurren, LeakyBucke, SlidingWindow and TokenBucket, the extension interface is RateLimiterAlgorithm.

7.22 Match Strategy

Which matching method to use when adding selectors And rules, currently supports And, Or, And the extension interface is MatchStrategy.

7.23 Parameter Data

Currently, URI,RequestMethod, Query, Post, IP, Host, Cookie, and Header are supported. The extended interface is ParameterData.

7.24 Predicate Judge

Which conditional policy to use when adding selectors and rules currently supports Match, Contains,Equals, Groovy, Regex, SpEL, TimerAfter, TimerBefore and Exclude. The extension interface is PredicateJudge.

8 Deployment

Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

This article introduces the use of `k8s` to deploy the `Apache ShenYu` gateway.

Catalog

I. Using h2 as a database

1. create nameSpace and configMap
2. deploying shenyu-admin
3. deploy shenyu-bootstrap

II. Use mysql as the database

Similar to the h2 process, there are two points to note

1. you need to load mysql-connector.jar, so you need a place to store the file
2. you need to specify an external mysql database configuration to proxy the external mysql database via endpoint

The process is as follows.

1. create nameSpace and configMap
2. create endpoint to proxy external mysql
3. create pv store mysql-connector.jar
4. deploy shenyu-admin
5. deploy shenyu-bootstrap

8.1 I. Using h2 as a database

8.1.1 1. Create nameSpace and configMap

- create 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
```

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

8.1.2 2. Create shenyu-admin

- create shenyu-admin.yaml

```
# Example of using the nodeport type to expose ports
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'
```

- executekubectl apply -f shenyu-admin.yaml

8.1.3 3. Create shenyu-bootstrap

- create shenyu-bootstrap.yaml

```
# Example of using the nodeport type to expose ports
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
```

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

8.2 II. Use mysql as the database

8.2.1 1. Create nameSpace and configMap

- create 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}
```

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

8.2.2 2. Create endpoint to represent mysql

- create 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
```

- execute `kubectl apply -f shenyu-ep.yaml`

8.2.3 3. Create pv to store mysql-connector.jar

- create shenyu-store.yaml

```
# Example of using pvc、pv、storageClass to store jar file
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  # Specify the directory on the node,
which should contain `mysql-connector.jar`
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - {your_node_name} # Specify node
---
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
```

- execute `kubectl apply -f shenyu-pv.yaml`
- pv mounted directory upload `mysql-connector.jar`

8.2.4 4. Create shenyu-admin

- create shenyu-admin.yaml

```
# Example of using the nodeport type to expose ports
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
```

- execute`kubectl apply -f shenyu-admin.yaml`

8.2.5 3. Create shenyu-bootstrap

- create shenyu-bootstrap.yaml

```
# Example of using the nodeport type to expose ports
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
```

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

This article describes how to build your own gateway based on `Apache ShenYu`.

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

8.3 Start Apache ShenYu Admin

- docker reference docker deployment Apache ShenYu Admin
- liunx/windows reference binary packages deployment Apache ShenYu Admin

8.4 Build your own gateway (recommended)

- first create an empty `springboot` project, you can refer to `shenyu-bootstrap`, or you can create it on spring official website.
- introduce the following `jar` package:

```
<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>
```

among them, `${project.version}` please use the current latest version.

- add the following configuration to your `application.yaml` file:

```
spring:
  main:
    allow-bean-definition-overriding: true
management:
  health:
    defaults:
      enabled: false
shenyu:
  sync:
    websocket:
      urls: ws://localhost:9095/websocket  //set to your shenyu-admin address
```

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

This aritcle introduces how to delopy the Shenyu gateway in cluster enviroment.

> In this part, you can see ShenYu Binary Packages Deployment before deploying.

8.5 Enviromental Preparation

- Two or more Gateway Boostrap servers, these servers must install JDK1.8+.
- A server for Gateway Admin, this server must install mysql/pgsql/h2 and JDK1.8+.
- A server for nginx.

8.6 Start Apache ShenYu Admin

- download and unzip apache-shenyu-incubating-${current.version}-admin-bin.tar.gz in your Gateway Admin server.
- config your database, go to the `/conf` directory, and modify `spring.profiles.active` of the configuration in `application.yaml` to `mysql`, `pg` or `h2`.
- config your way of synchronization, go to the `/conf` directory, and modify `shenyu.sync` of configuration in `application.yaml` to `websocket`, `http`, `zookeeper`, `etcd`, `consul` or `nacos`.
- start Apache ShenYu Admin in `bin` directory.

```
> windows: start.bat

> linux: ./start.sh
```

8.7 Start Apache ShenYu Boostrap

- download and unzip apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz in your Gateway Boostrap server.
- config your synchronization, go to the `/conf` directory, and modify `shenyu.sync` of configuration in `application.yaml` to `websocket`, `http`, `zookeeper`, `etcd`, `consul` or `nacos`, this configuaration must remain the same of `ShenyYu Admin`.
- repeat above-mentioned operations in each `ShenYu Bootstrap` server.
- start Apache ShenYu Bootstrap in `bin` directory.

```
> windwos : start.bat

> linux : ./start.sh
```

After completing these operations, you will deploy `ShenYu Boostrap` Cluster.

For example. you will deploy `ShenYu Bootstrap` in `10.1.1.1` and `10.1.1.2` and deploy nginx in `10.1.1.3`.

8.8 Start Nginx

- download and install `nginx`.
- modify `upstream` and `server` of configuration in `nginx.conf`.

```
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;
  }
}
```

- start nginx.

```
> windows: ./nginx.exe

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

- verify nginx, looking at your `ShenYu Bootstrap` log or `Nginx` log, Where will the verification request go.

This article introduces the use of `helm` to deploy the `Apache ShenYu` gateway.

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

This article describes some of the prerequisites you need to prepare before deploying the Apache ShenYu gateway.

8.9 Database Initialize

Before deploying the `Shenyu-admin` project, initialize the database it uses (databases currently support: Mysql, PostgreSql, Oracle), which used the script files are stored in db directory project root directory, The following describes the initial steps for each database.

8.9.1 Mysql

In the mysql initialization scripts directory found in the initialization script `schema.sql`, Use the client connection tool to connect to your Mysql service and execute, so you get a database named `shenyu`, which can later be used as the database for the `Shenyu-admin` project.

8.9.2 PostgreSql

In the pg initialization scripts directory found in the initialization script `create-database.sql`、`create-table.sql`, and use the client connection tool to connect to your PostgreSql service. so you get a database named `shenyu`, which can later be used as a database for the `Shenyu-admin` project.

8.9.3 Oracle

In the oracle initialization scripts directory found in the initialization script `schema.sql`, Use the client connection tool to connect to your Oracle service to create a database, execute the `schema.sql` script on this database, and initialize the `Shenyu-admin` database. After can be project configuration file to adjust your Oracle environment configuration.

This article introduces the use of `docker-compose` to deploy the `Apache ShenYu` gateway.

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

8.10 Download shell script

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

8.11 execute script

This script will download the required configuration files and mysql-connector, and can be executed repeatedly if the download fails.

```
sh ./install.sh #The latest configuration is pulled by default. If you need to deploy the released version, you can add a parameter to indicate the version number, such as: v2.4.2 or latest
```

8.12 Initialize the shenyu-admin database

Refer to the database initialization documentation to initialize the database.

8.13 Modify the configuration file

Modify the configuration file downloaded by the script to set up configurations such as JDBC.

8.14 Execute docker-compose

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

This article introduces how to start the Apache ShenYu gateway in the local environment.

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

8.15 Environmental preparation

- Install JDK1.8+ locally
- Install Git locally
- Install Maven locally
- Choose a development tool, such as IDEA

8.16 Download the compiled code

- Download

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

- use the development tool to start org.apache.shenyu.admin.ShenyuAdminBootstrap, Visit http://localhost:9095, the default username and password are: admin and 123456 respectively.
  - If you use h2 for storage, set the variable --spring.profiles.active = h2 and start the server.

  - If you use MySQL for storage, follow the guide document to initialize the database and modify the JDBC configuration in application-mysql.yml, set the variable --spring.profiles.active = mysql and start the server.
  - If you use PostgreSql for storage, follow the guide document to initialize the database and modify the JDBC configuration in application-pg.yml, set the variable --spring.profiles.active = pg and start the server.
  - If you use Oracle for storage, follow the guide document to initialize the database and modify the JDBC configuration in application-oracle.yml, set the variable --spring.profiles.active = oracle.

- use the development tool to start org.apache.shenyu.bootstrap.ShenyuBootstrapApplication.

This article introduces the use of docker to deploy the Apache ShenYu gateway.

Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

8.17 Start Apache ShenYu Admin

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

- use h2 to store data:

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

- use MySQL to store data, follow the guide document to initialize the database, copy mysql-connector.jar to /$(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}
```

another way is to put the application.yml、application-mysql.yml、application-pg.yml、application-oracle.yml configuration in ${your_work_dir}/conf from Configure address , modify the configuration spring.profiles.active = mysql in application.yml, and then execute the following statement:

```
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}
```

- use PostgreSql to store data, follow the guide document to initialize the database, execute the following statement:

```
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}
```

another way is to put the application.yml、application-mysql.yml、application-pg.yml、application-oracle.yml configuration in ${your_work_dir}/conf, modify the configuration spring.profiles.active = pg in application.yml,and then execute the following statement:

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

- use Oracle to store data, follow the guide document to initialize the database, execute the following statement:

```
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}
```

another way is to put the application.yml、application-mysql.yml、application-pg.yml、application-oracle.yml configuration in ${your_work_dir}/conf, modify the configuration spring.profiles.active = oracle in application.yml, and then execute the following statement:

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

8.18 Start Apache ShenYu Bootstrap

In the host, the directory where the bootstrap configuration file is located is recorded as $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}
```

This article introduces how to quick start the Apache ShenYu gateway in the standalone environment.

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

8.19 Environmental preparation

- Install JDK1.8+ locally

8.20 Start Apache ShenYu Bootstrap

- download apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz
- unzip `apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz`。go to the `bin` directory.

```
> windwos : start.bat

> linux : ./start.sh
```

8.21 Selector and rule configuration

please refer to Developer Local Model add the selector and rule.

example:

- your service address is`http://127.0.0.1:8080/helloworld` and the response like follow:

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

- use the follow data to add selector and rule

8.22 by postman

Add `localKey: 123456` to Headers. If you need to customize the localKey, you can use the sha512 tool to generate the key based on plaintext and update the `shenyu.local.sha512Key` property.

`POST` method, address`http://localhost:9195/shenyu/plugin/selectorAndRules`, body use `raw` `json` content:

```
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": "/**"
        }]
    }]
}
```

8.23 by 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": "/**"
        }]
    }]
}'
```

- open `http://localhost:9195/helloworld`:

```
{
  "name" : "Shenyu",
```

```
  "data" : "hello world"
}
```

This article introduces the deployment of the Apache ShenYu gateway using the binary packages.

> Before you read this document, you need to complete some preparations before deploying Shenyu according to the Deployment Prerequisites document.

8.24 Start Apache ShenYu Admin

- download apache-shenyu-incubating-${current.version}-admin-bin.tar.gz
- unzip apache-shenyu-incubating-${current.version}-admin-bin.tar.gz。go to the bin directory.
- use h2 to store data：

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

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

- use MySQL to store data, follow the guide document to initialize the database, copy mysql-connector.jar to /$(your_work_dir)/ext-lib, go to the /conf directory, and modify the JDBC configuration in application-mysql.yml.

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

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

- use PostgreSql to store data, follow the guide document to initialize the database, go to the /conf directory, and modify the JDBC configuration in application-pg.yml.

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

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

- use Oracle to store data, follow the guide document to initialize the database, go to the /conf directory, and modify the JDBC configuration in application-oracle.yml.

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

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

8.25 Start Apache ShenYu Bootstrap

- download apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz
- unzip `apache-shenyu-incubating-${current.version}-bootstrap-bin.tar.gz`。go to the `bin` directory.

```
> windwos : start.bat

> linux : ./start.sh
```

9 Quick Start

This document introduces how to quickly access the Apache ShenYu gateway using Spring Cloud. You can get the code example of this document by clicking here .

9.1 Environment to prepare

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the springCloud plugin on in the BasicConfig -> Plugin.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

Note: Before starting, make sure the gateway has added dependencies.

Add the gateway proxy plugin for `Spring Cloud` and add the your registry center dependencies:

```
<!-- 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 config information:

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

Note: Please ensure that the spring Cloud registry service discovery configuration is enabled

- Configuration method

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

- code method

```
@SpringBootApplication
@EnableDiscoveryClient
public class ShenyuBootstrapApplication {

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

Restart the shenyu-bootstrap project.

9.2 Run the shenyu-examples-springcloud project

In the example project we used Eureka as the registry for Spring Cloud. You can use the local Eureka or the application provided in the example.

Download shenyu-examples-eureka 、 shenyu-examples-springcloud .

Startup the Eureka service: Execute the org.apache.shenyu.examples.eureka.EurekaServerApplication main method to start project.

Startup the Spring Cloud service: Execute the org.apache.shenyu.examples.springcloud.ShenyuTestSpringCloudApplication main method to start project.

Since 2.4.3, shenyu.client.springCloud.props.port can be non-configured if you like.

The following log appears when the startup is successful:

```
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.
```

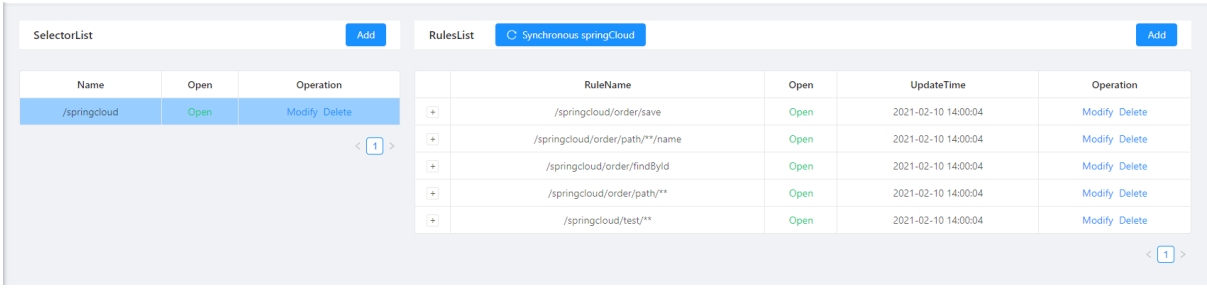
```
2021-02-10 14:03:52.921  INFO 2860 --- [           main] o.s.c.n.eureka.
InstanceInfoFactory       : Setting initial instance status as: STARTING
2021-02-10 14:03:52.949  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Initializing Eureka in region us-east-1
2021-02-10 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)
```

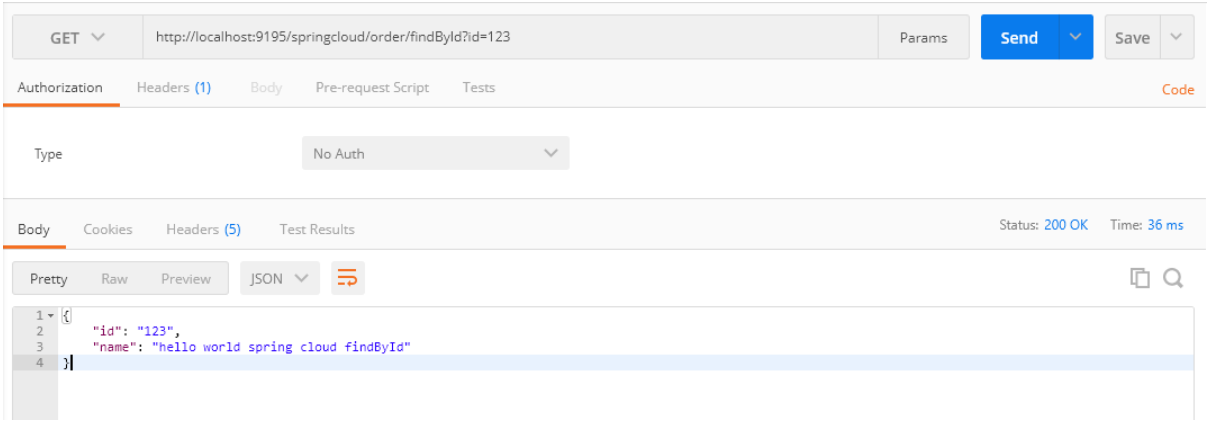
9.3 Test

The `shenyu-examples-springcloud` project will automatically register interface methods annotated with `@ShenyuSpringCloudClient` in the Apache ShenYu gateway after successful startup.

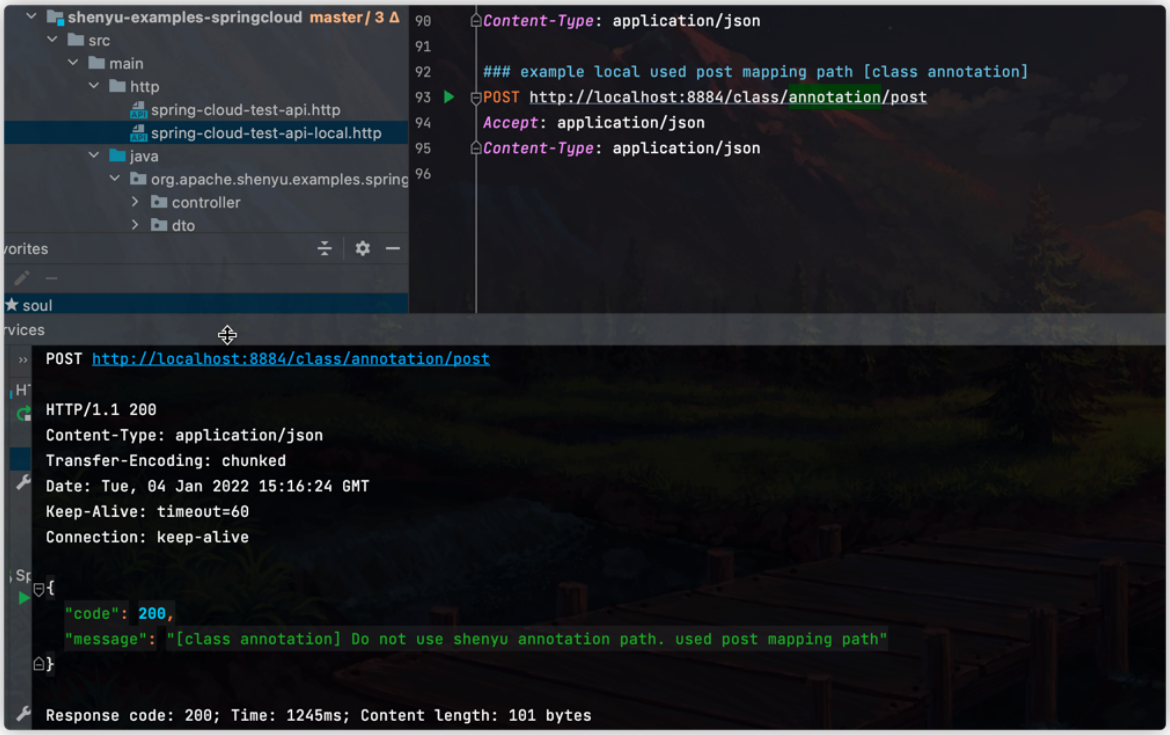
Open PluginList -> rpc proxy -> springCloud to see the list of plugin rule configurations:



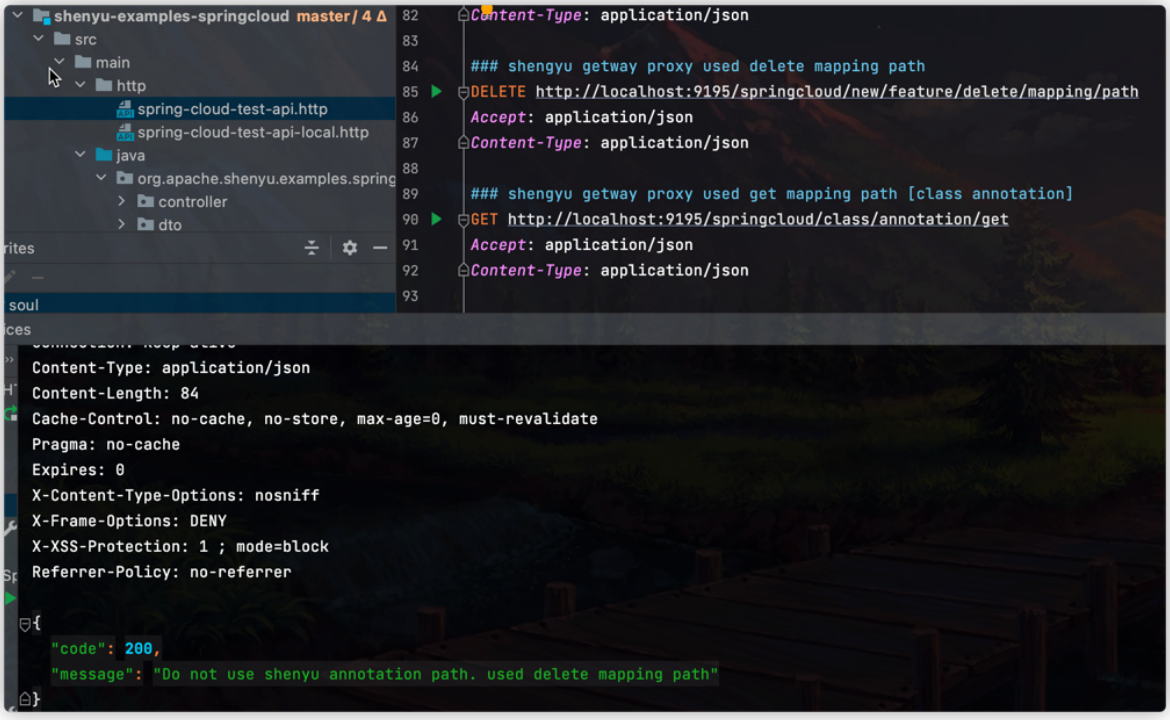
Use PostMan to simulate HTTP to request your SpringCloud service:



Use IDEA HTTP Client Plugin to simulate HTTP to request your SpringCloud service[local:no Shenyu proxy]:



Use IDEA HTTP Client Plugin to simulate HTTP to request your SpringCloud service[Shenyu proxy]:



This document introduces how to quickly access the Apache ShenYu gateway using Sofa RPC. You can get the code example of this document by clicking here.

9.4 Environment to prepare

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the Sofa plugin on in the BasicConfig -> Plugin, and set your registry address. Please make sure the registry center is open locally.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

Note: Before starting, make sure the gateway has added dependencies.

If client is `sofa`, registry center is `Zookeeper`, please refer to the following configuration:

```
<!-- 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-->
```

9.5 Run the shenyu-examples-sofa project

Download shenyu-examples-sofa, replace the register address in `spring-dubbo.xml` with your local zk address, such as:

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

Execute the `org.apache.shenyu.examples.sofa.service.TestSofaApplication` main method to start sofa service.

The following log appears when the startup is successful:

```
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-
starter\6.0.4\rpc-sofa-boot-starter-6.0.4.jar;D:\SOFT\m2\repository\com\alipay\
sofa\rpc-sofa-boot-core\6.0.4\rpc-sofa-boot-core-6.0.4.jar;D:\SOFT\m2\repository\
com\alipay\sofa\sofa-rpc-all\5.5.7\sofa-rpc-all-5.5.7.jar;D:\SOFT\m2\repository\
com\alipay\sofa\bolt\1.4.6\bolt-1.4.6.jar;D:\SOFT\m2\repository\org\javassist\
javassist\3.20.0-GA\javassist-3.20.0-GA.jar;D:\SOFT\m2\repository\io\netty\netty-
all\4.1.43.Final\netty-all-4.1.43.Final.jar;D:\SOFT\m2\repository\com\alipay\sofa\
hessian\3.3.6\hessian-3.3.6.jar;D:\SOFT\m2\repository\com\alipay\sofa\tracer-core\
2.1.2\tracer-core-2.1.2.jar;D:\SOFT\m2\repository\io\opentracing\opentracing-api\0.
22.0\opentracing-api-0.22.0.jar;D:\SOFT\m2\repository\io\opentracing\opentracing-
noop\0.22.0\opentracing-noop-0.22.0.jar;D:\SOFT\m2\repository\io\opentracing\
opentracing-mock\0.22.0\opentracing-mock-0.22.0.jar;D:\SOFT\m2\repository\io\
opentracing\opentracing-util\0.22.0\opentracing-util-0.22.0.jar;D:\SOFT\m2\
repository\com\alipay\sofa\lookout\lookout-api\1.4.1\lookout-api-1.4.1.jar;D:\SOFT\
m2\repository\com\alipay\sofa\runtime-sofa-boot-starter\3.1.4\runtime-sofa-boot-
starter-3.1.4.jar;D:\SOFT\m2\repository\org\apache\curator\curator-client\2.9.1\
curator-client-2.9.1.jar;D:\SOFT\m2\repository\org\apache\zookeeper\zookeeper\3.4.
6\zookeeper-3.4.6.jar;D:\SOFT\m2\repository\log4j\log4j\1.2.16\log4j-1.2.16.jar;D:\
SOFT\m2\repository\jline\jline\0.9.94\jline-0.9.94.jar;D:\SOFT\m2\repository\io\
netty\netty\3.7.0.Final\netty-3.7.0.Final.jar;D:\SOFT\m2\repository\com\google\
guava\guava\16.0.1\guava-16.0.1.jar;D:\SOFT\m2\repository\org\apache\curator\
```

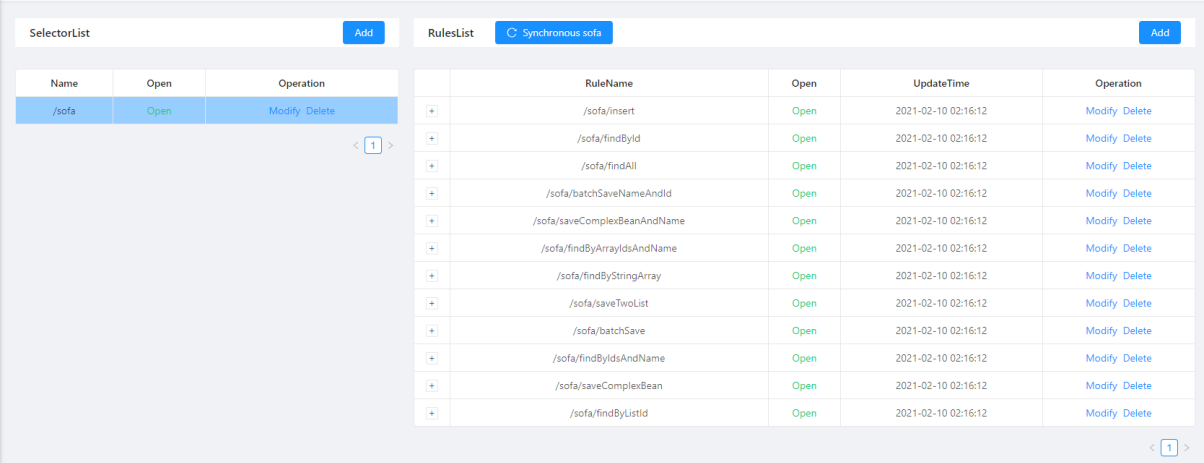
```
2021-02-10 02:31:46.060  INFO 2156 --- [           main] org.apache.zookeeper.
ZooKeeper           : Client environment:java.library.path=C:\Program Files\Java\
jdk1.8.0_211\bin;C:\Windows\Sun\Java\bin;C:\Windows\system32;C:\Windows;C:\Program
Files\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\
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)
```

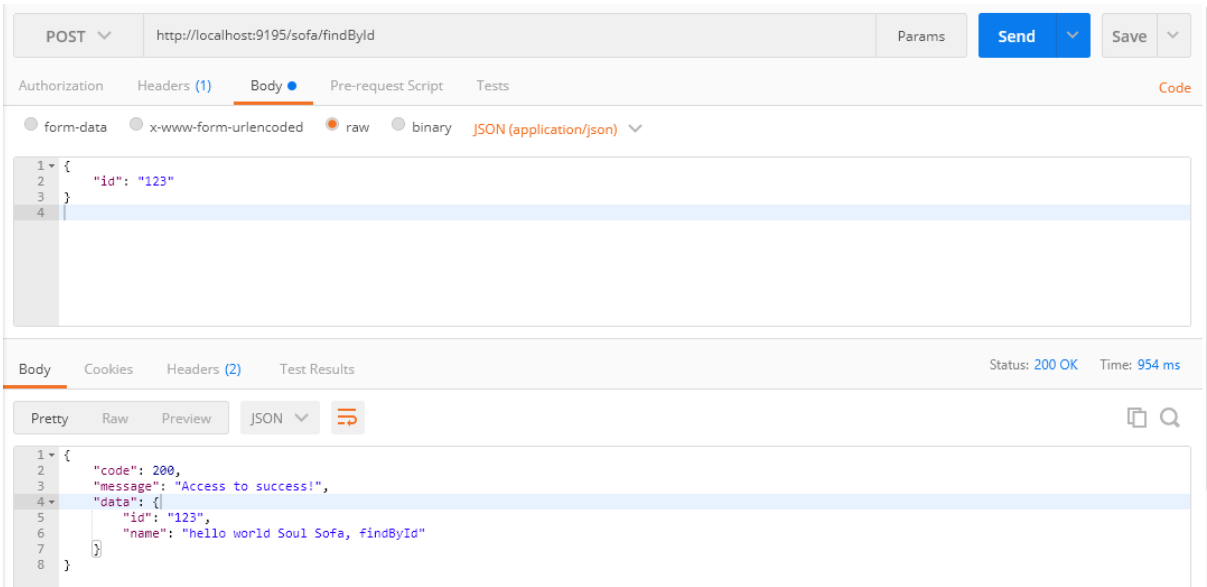
9.6 Test

The `shenyu-examples-sofa` project will automatically register interface methods annotated with `@ShenyuSofaClient` in the Apache ShenYu gateway after successful startup.

Open PluginList -> rpc proxy -> sofa to see the list of plugin rule configurations:

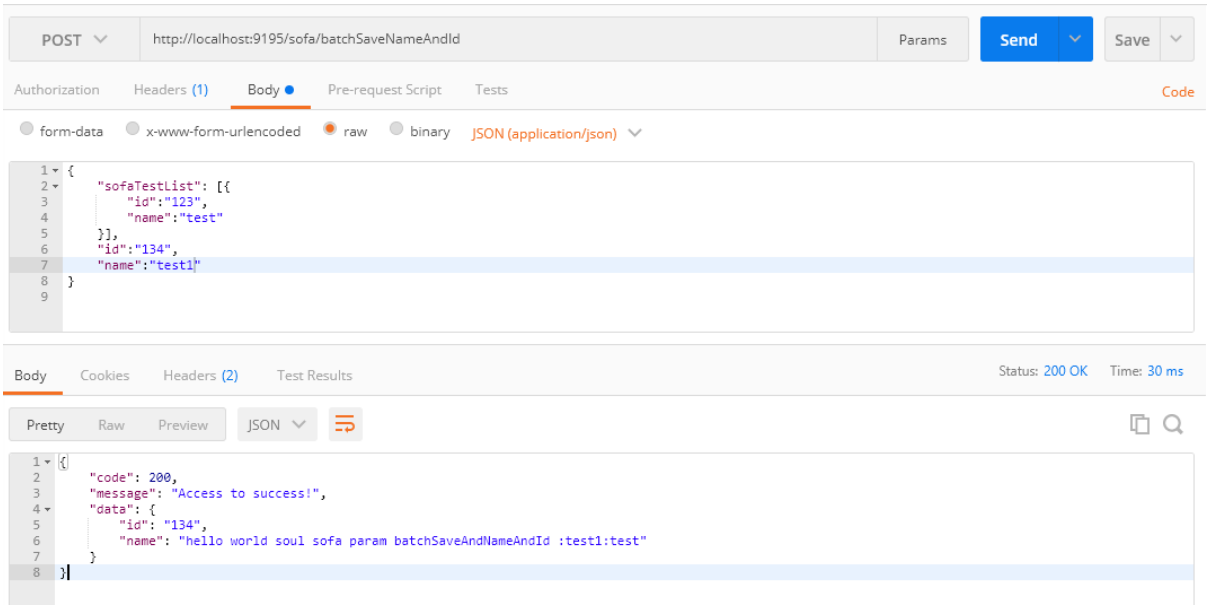


Use PostMan to simulate HTTP to request your Sofa service:



Complex multi-parameter example: The related interface implementation class is `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;
}
```



This document introduces how to quickly access the Apache ShenYu gateway using Http. You can get the code example of this document by clicking here.

9.7 Environment to prepare

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the Divide plugin on in the BasicConfig -> Plugin. In the Apache ShenYu gateway, the HTTP request is handled by the Divide plugin.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

Note: Before starting, make sure the gateway has added dependencies.

Add the following dependencies to the gateway's `pom.xml` file:

```
<!--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>
```

9.8 Run the shenyu-examples-http project

Download shenyu-examples-http

Execute the `org.apache.shenyu.examples.http.ShenyuTestHttpApplication` main method to start project.

Since `2.4.3`, `shenyu.client.http.props.port` can be non-configured if you like.

The following log appears when the startup is successful:

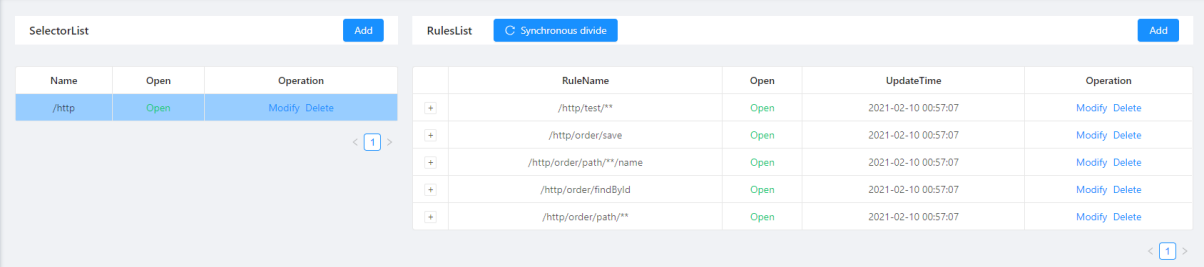
```
2021-02-10 00:57:07.561  INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : http client register success: {"appName":"http","context":"/http",
"path":"/http/test/**","pathDesc":"","rpcType":"http","host":"192.168.50.13","port
":8188,"ruleName":"/http/test/**","enabled":true,"registerMetaData":false}
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)
```

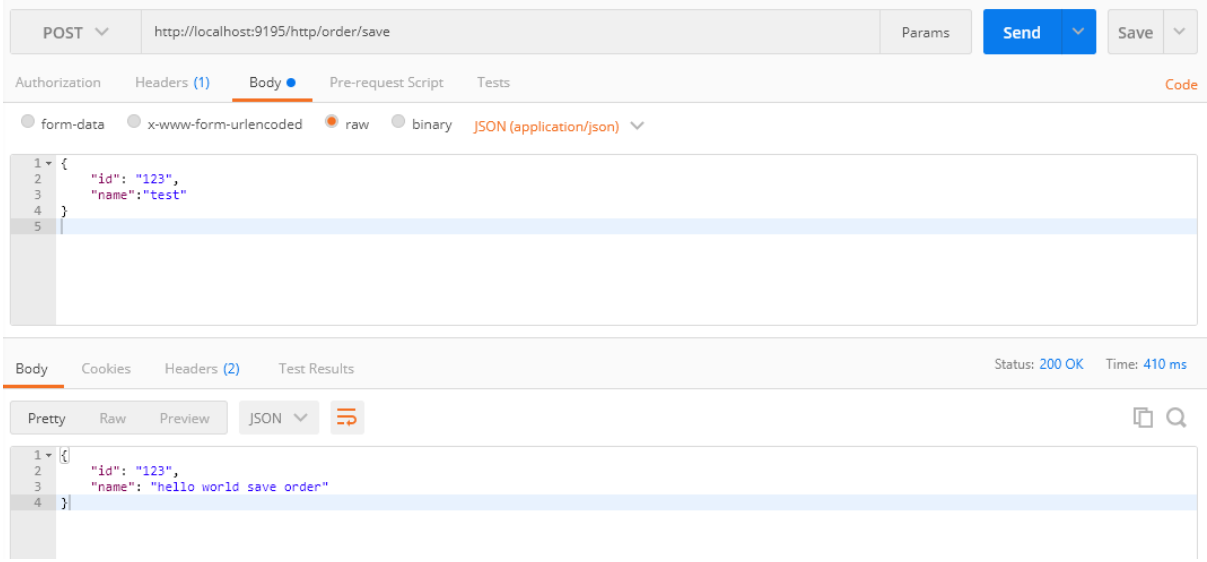
9.9 Test

The `shenyu-examples-http` project will automatically register interface methods annotated with `@ShenyuSpringMvcClient` in the Apache ShenYu gateway after successful startup.

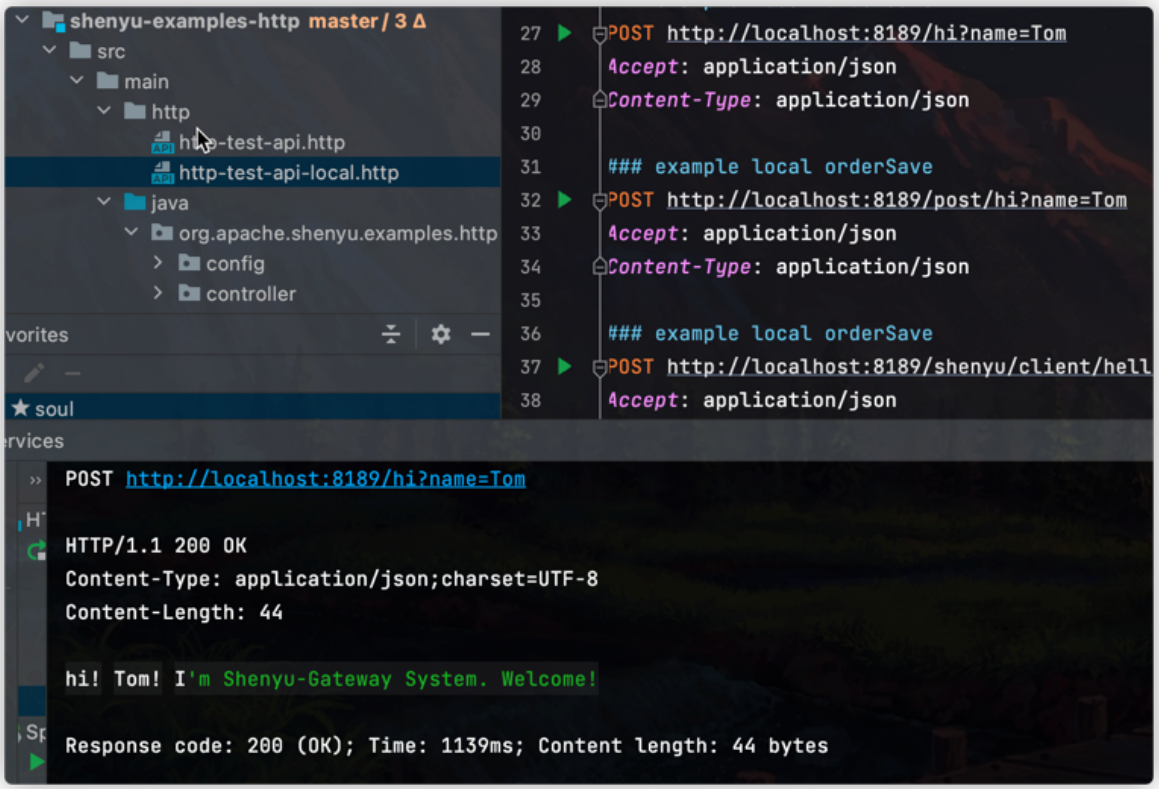
Open PluginList -> Proxy -> divide to see the list of plugin rule configurations:



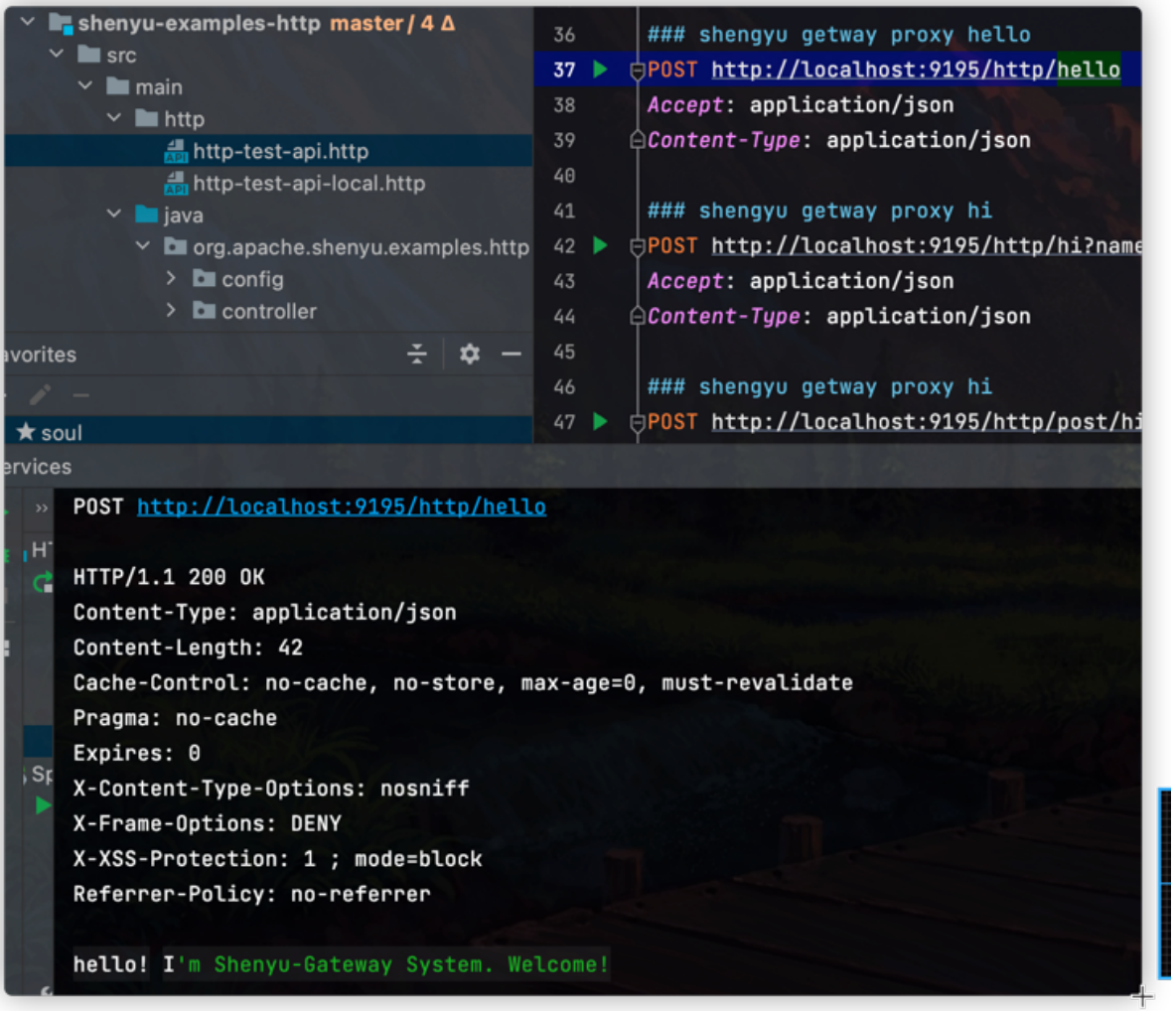
Use PostMan to simulate HTTP to request your http service:



Use IDEA HTTP Client Plugin to simulate HTTP to request your http service[local:no Shenyu proxy]:



Use IDEA HTTP Client Plugin to simulate HTTP to request your http service[Shenyu proxy]:



This document introduces how to quickly access the Apache ShenYu Gateway using Tars. You can get the code example of this document by clicking here .

9.10 Environment to prepare

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the Sofa plugin on in the BasicConfig -> Plugin.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

Note: Before starting, make sure the gateway has added dependencies.

`shenyu-bootstrap` need to import tars dependencies:

```
<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>
```

9.11 Run the shenyu-examples-tars project

Download shenyu-examples-tars .

Modify `host` in `application.yml` to be your local IP

Modify config `src/main/resources/ShenyuExampleServer.ShenyuExampleApp.config.conf`:

- It is recommended to make clear the meaning of the main configuration items of config, refer to the development guide
- bind IP in config should pay attention to providing cost machine
- local=⋯, Indicates the open port that the native machine connects to the tarsnode. If there is no tarsnode, this configuration can be dropped
- `locator`: Indicates the address (frame address) of the main control center, which is used to obtain the IP list according to the service name, If Registry is not required to locate the service, this configuration can be dropped
- `node=tars.tarsnode.ServerObj@xxxx`, Indicates the address of the connected tarsnode. If there is no tarsnode locally, this configuration can be removed

More config configuration instructions, Please refer to TARS Official Documentation

Execute the `org.apache.shenyu.examples.tars.ShenyuTestTarsApplication` main method to start project.

Note: The `configuration file address` needs to be specified in the startup command when the service starts **-Dconfig=xxx/ShenyuExampleServer.ShenyuExampleApp.config.conf**

If the `-Dconfig` parameter is not added, the configuration may throw the following exceptions:

```
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
```

The following log appears when the startup is successful:

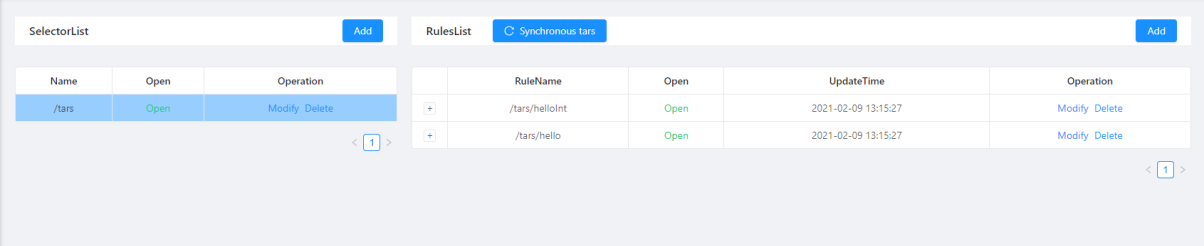
```
[SERVER] server starting at tcp -h 127.0.0.1 -p 21715 -t 60000...
[SERVER] server started at tcp -h 127.0.0.1 -p 21715 -t 60000...
[SERVER] server starting at tcp -h 127.0.0.1 -p 21714 -t 3000...
[SERVER] server started at tcp -h 127.0.0.1 -p 21714 -t 3000...
[SERVER] The application started successfully.
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}
```

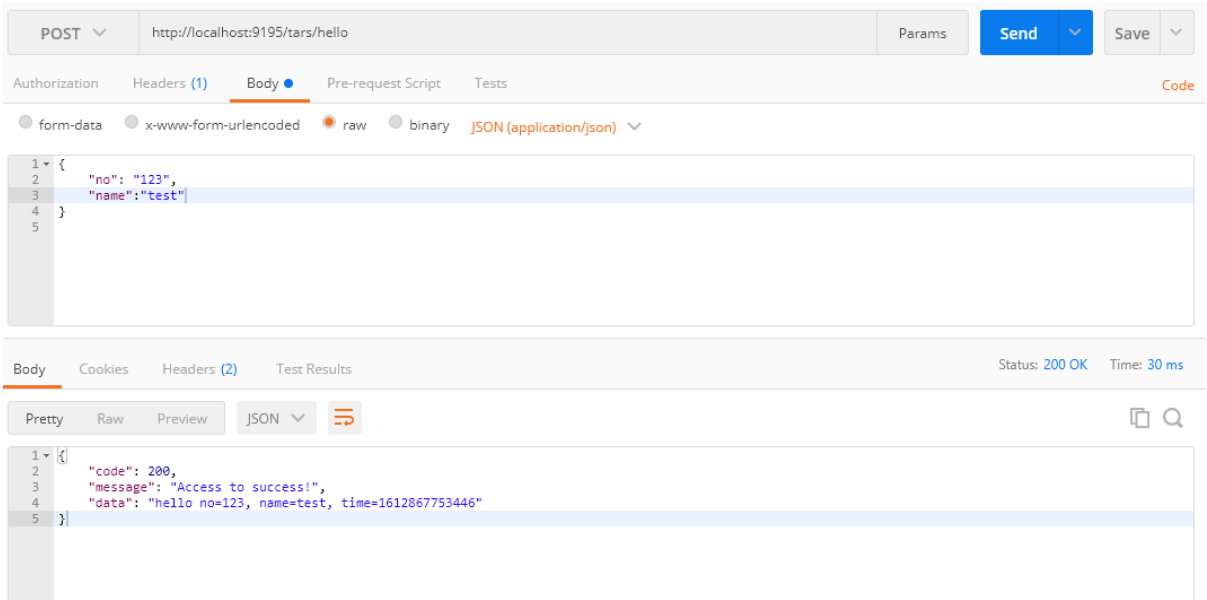

9.12 Test

The `shenyu-examples-tars` project will automatically register interface methods annotated with `@ShenyuTarsClient` in the Apache ShenYu gateway after successful startup.

Open PluginList -> rpc proxy -> tars to see the list of plugin rule configurations:



Use PostMan to simulate HTTP to request your tars service:



This document introduces how to quickly access the Apache ShenYu gateway using Dubbo. You can get the code example of this document by clicking here.

9.13 Environment to prepare

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the Dubbo plugin on in the BasicConfig -> Plugin, and set your registry address. Please make sure the registry center is open locally.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

Note: Before starting, make sure the gateway has added dependencies.

If client is `apache dubbo`, registry center is `Zookeeper`, please refer to the following configuration:

```
<!-- 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-->
```

If client is `alibaba dubbo`, registry center is `Zookeeper`, please refer to the following configuration:

```
<!-- 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-->
```

9.14 Run the shenyu-examples-dubbo project

Download shenyu-examples-dubbo .

replace the register address in `shenyu-examples-alibaba-dubbo-service/src/main/resources/spring-dubbo.xml` with your local zk address, such as:

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

Execute the `org.apache.shenyu.examples.alibaba.dubbo.service.TestAlibabaDubboApplication` main method to start dubbo project.

The following log appears when the startup is successful:


```
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}
```

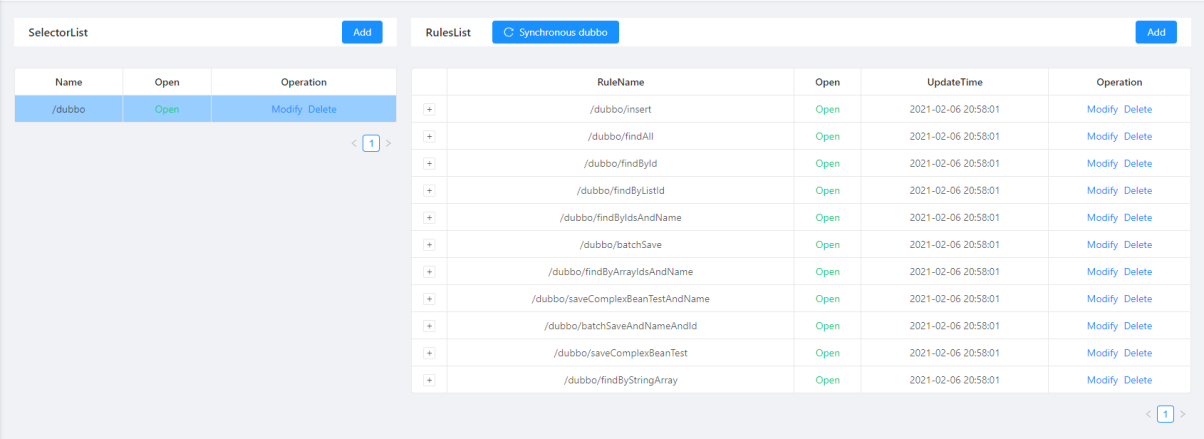
```
2021-02-06 20:58:01.889  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/saveComplexBeanTestAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.shenyu.examples.dubbo.api.service.DubboMultiParamService
","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}
```


Note: When you need to expose `multiple protocols` at the same time, please do not configure `shenyu.client.dubbo.props.port`.

9.15 Test

The `shenyu-examples-dubbo` project will automatically register interface methods annotated with `@ShenyuDubboClient` in the Apache ShenYu gateway after successful startup.

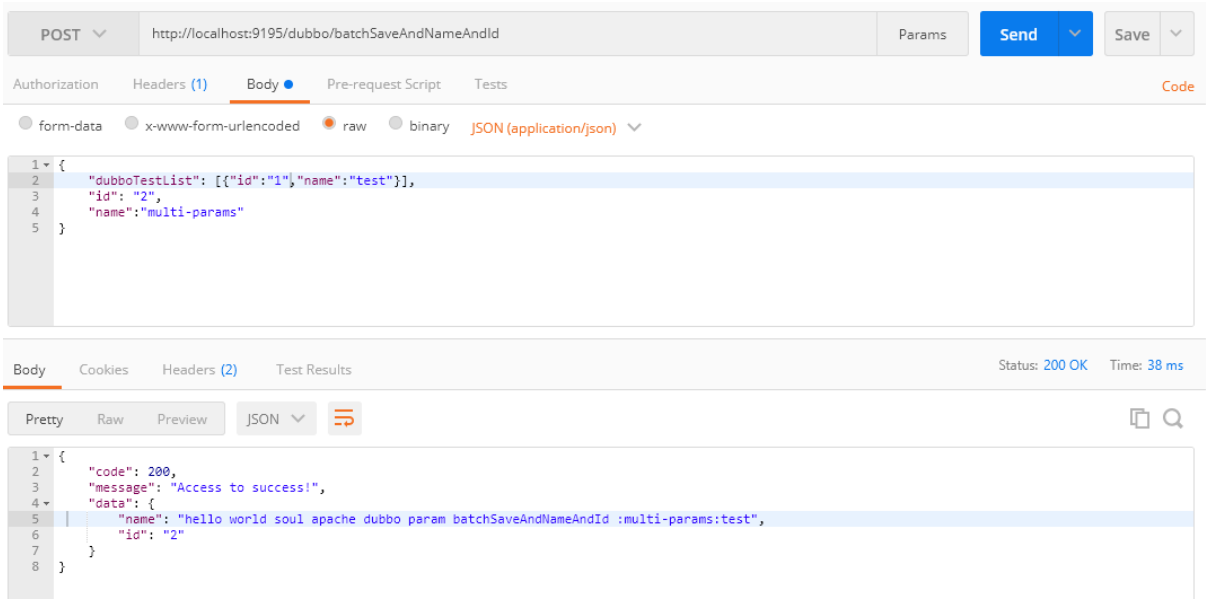
Open PluginList -> rpc proxy -> dubbo to see the list of plugin rule configurations:



Use `PostMan` to simulate `HTTP` to request your `Dubbo` service:

Complex multi-parameter example: The related interface implementation class is `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;
}
```



When your arguments do not match, the following exception will occur:

```
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
```

This document introduces how to quickly access the Apache ShenYu gateway using gRPC. You can get the code example of this document by clicking here .

9.16 Prepare For Environment

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the gRPC plugin on in the BasicConfig -> Plugin.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

Note: Before starting, make sure the gateway has added dependencies.

Add the following dependencies to the gateway' s `pom.xml` file:

```
<!-- 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-->
```

9.17 Run the shenyu-examples-grpc project

Download shenyu-examples-grpc

Run the following command under `shenyu-examples-grpc` to generate Java code:

```
mvn protobuf:compile
mvn protobuf:compile-custom
```

Execute the `org.apache.shenyu.examples.grpc.ShenyuTestGrpcApplication` main method to start project.

The following log appears when the startup is successful:

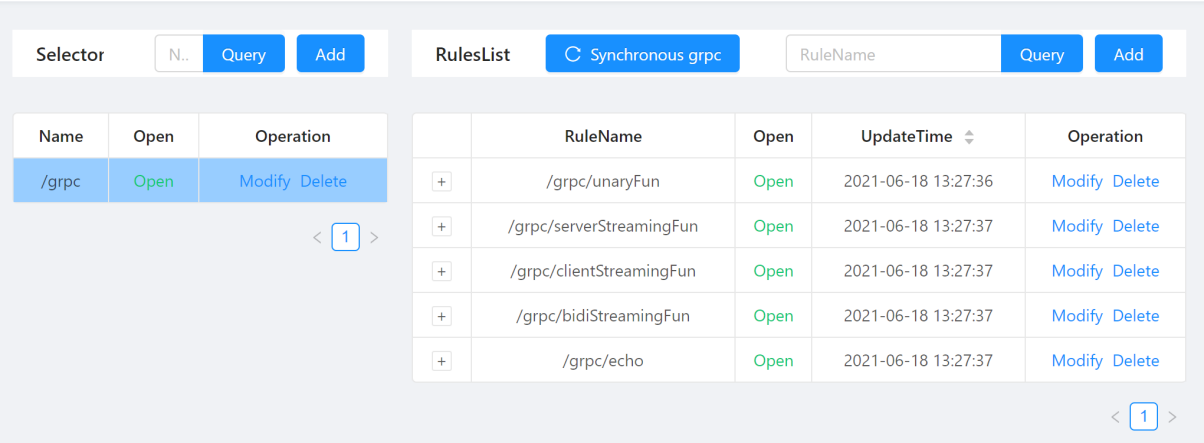
```
2021-06-18 19:33:32.866  INFO 11004 --- [or_consumer_-19] o.a.s.r.client.http.
utils.RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/clientStreamingFun","pathDesc":
"clientStreamingFun","rpcType":"grpc","serviceName":"stream.StreamService",
"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}
```


9.18 Test

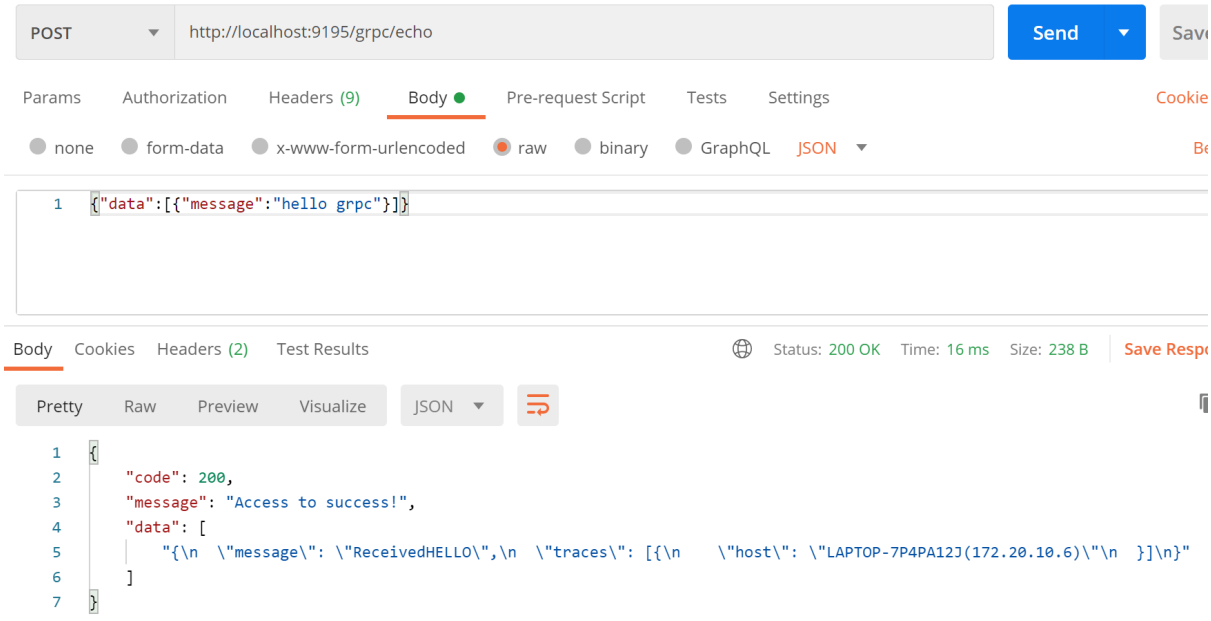
The `shenyu-examples-grpc` project will automatically register interface methods annotated with `@ShenyuGrpcClient` in the Apache ShenYu gateway after successful startup.

Open PluginList -> rpc proxy -> gRPC to see the list of plugin rule configurations:



Use `postman` to simulate `http` to request your gRPC service. The following is the request body.

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



The parameters are passed in json format. The name of the key is `data` by default, and you can reset it in `GrpcConstants.JSON_DESCRIPTOR_PROTO_FIELD_NAME`. The input of value is based on the proto file defined by you.

9.19 Streaming

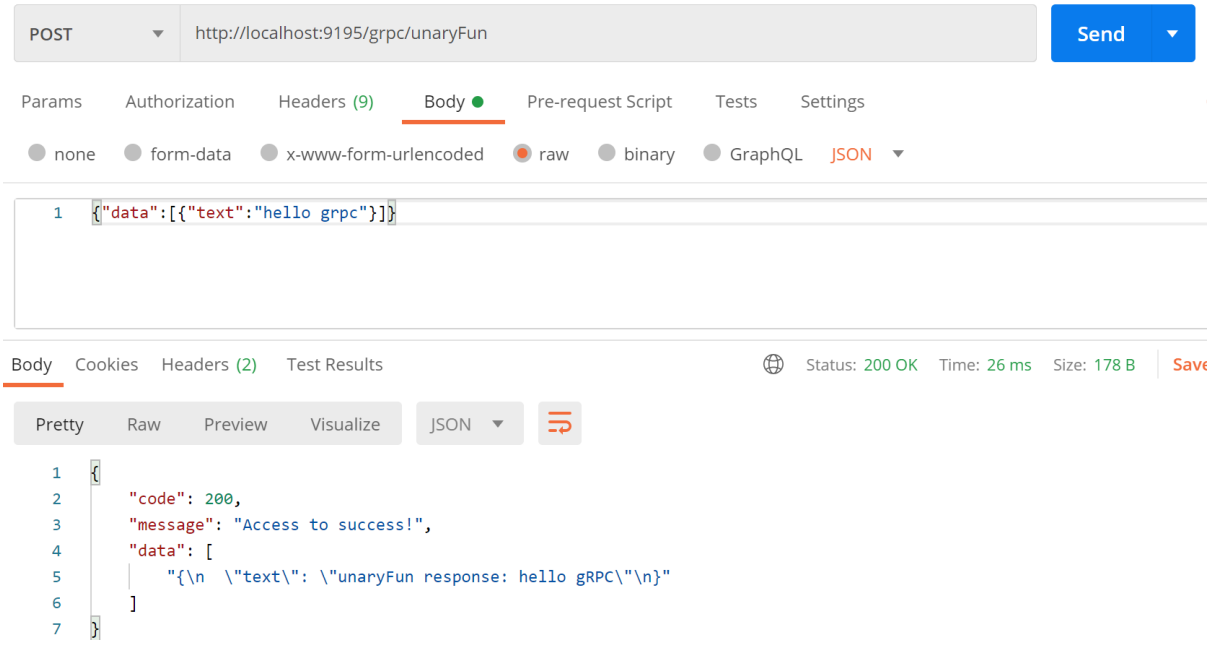
the Apache ShenYu can support streaming of gRPC. The following shows the calls of the four method types of gRPC. In streaming, you can pass multiple parameters in the form of an array.

- UNARY

The request body like this.

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

Then, call gRPC service by UNARY method type.



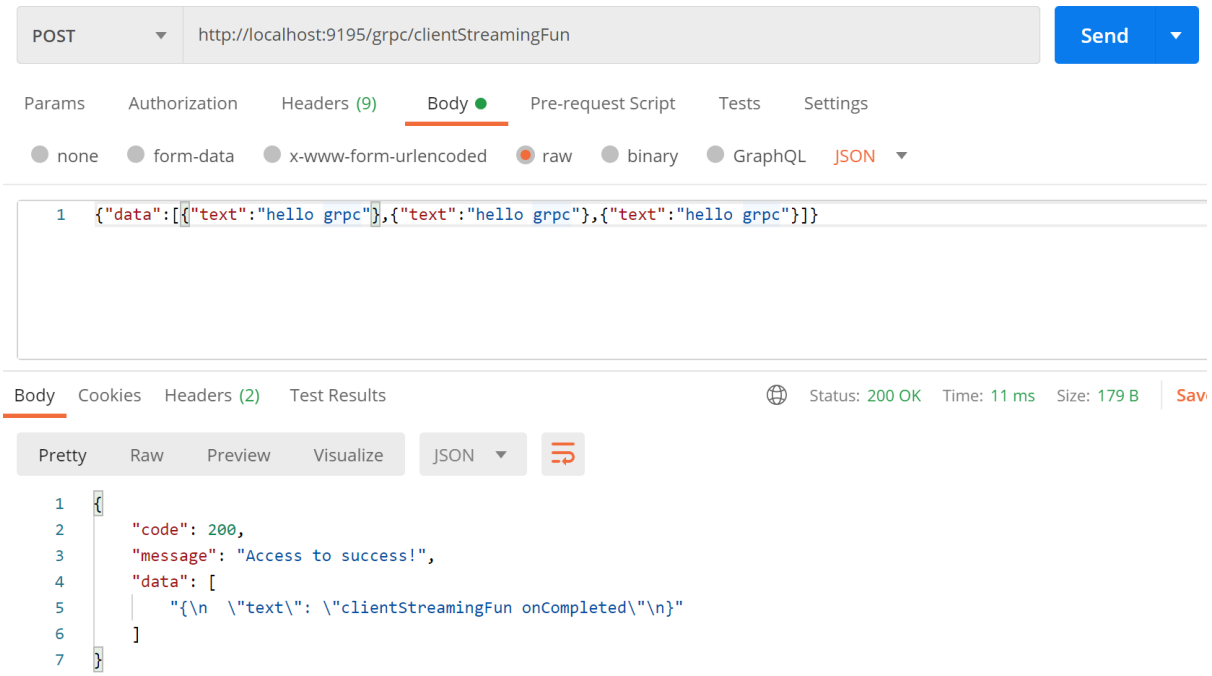
- CLIENT_STREAMING

The request body like this.

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

```
            "text": "hello grpc"
        }
    ]
}
```

Then, call gRPC service by `CLIENT_STREAMING` method type.

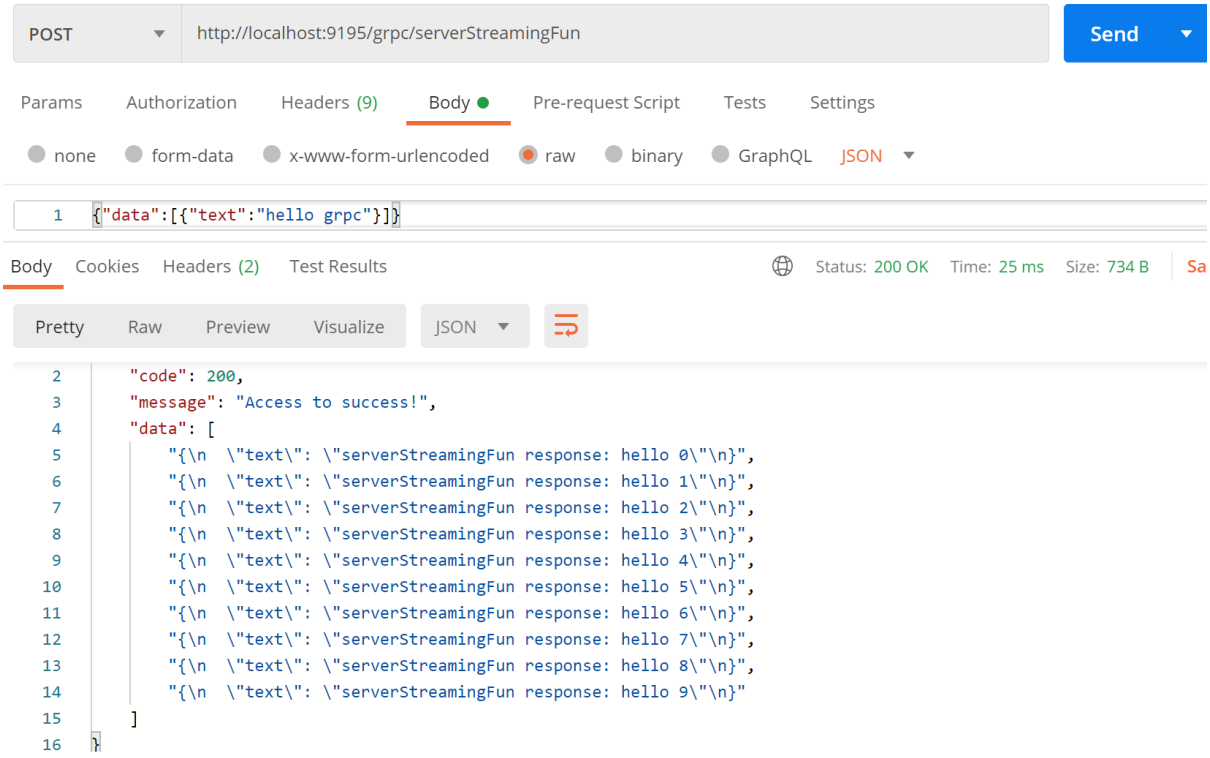


- `SERVER_STREAMING`

The request body like this.

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

Then, call gRPC service by `SERVER_STREAMING` method type.

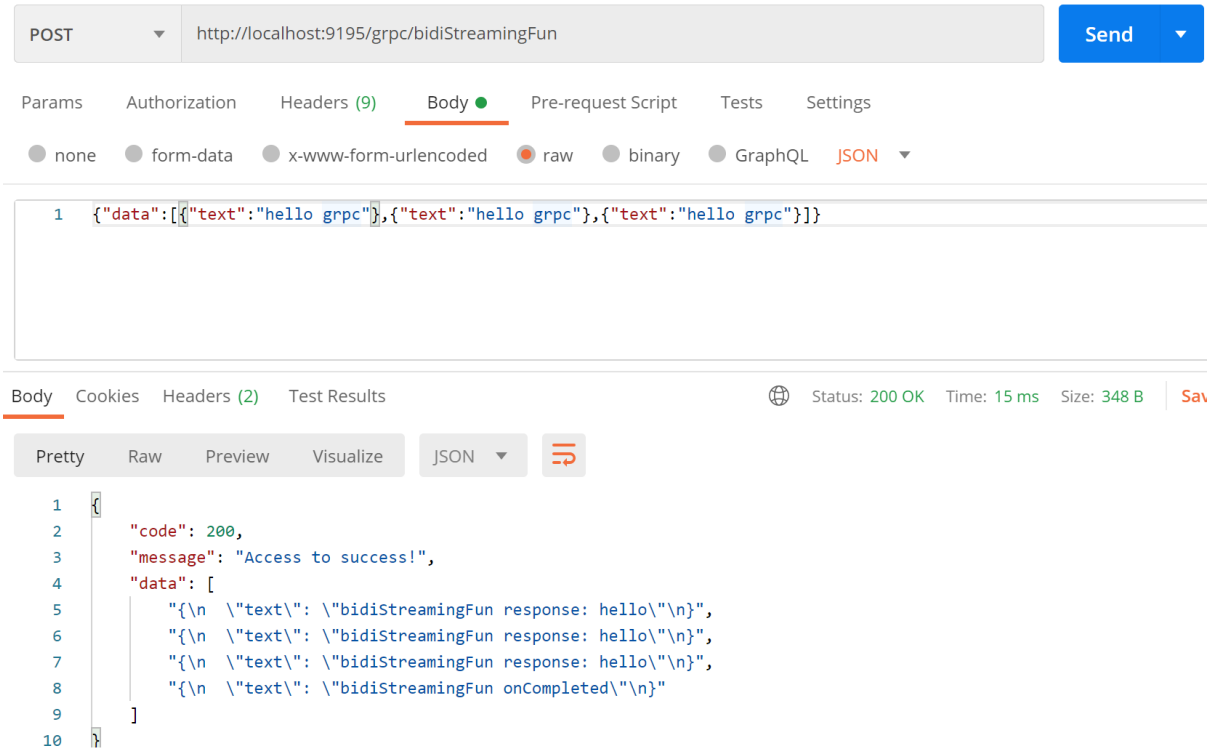


- `BIDI_STREAMING`

The request body like this.

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

Then, call gRPC service by `BIDI_STREAMING` method type.



This document introduces how to quickly access the Apache ShenYu gateway using Motan RPC. You can get the code example of this document by clicking here.

9.20 Environment to prepare

Please refer to the deployment to select a way to start shenyu-admin. For example, start the Apache ShenYu gateway management system through local deployment .

After successful startup, you need to open the Sofa plugin on in the BasicConfig -> Plugin.

If you are a startup gateway by means of source, can be directly run the ShenyuBootstrapApplication of shenyu-bootstrap module.

> Note: Before starting, make sure the gateway has added dependencies. Start up zookeeper in local.

Import the gateway proxy plugin for `Motan` and add the following dependencies to the gateway's `pom.xml` file:

```
<!-- 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 Run the shenyu-examples-motan project

Download shenyu-examples-motan .

Run main method of `org.apache.shenyu.examples.motan.service.TestMotanApplication` to start this project.

log info as follows after starting:

```
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 Test

The `shenyu-examples-motan` project will automatically register the `@ShenyuMotanClient` annotated interface methods with the gateway and add selectors and rules. If not, you can manually add them.

Open PluginList -> rpc proxy -> motan to see the list of plugin rule configurations:

Use PostMan to simulate HTTP to request your Motan service:

10 User Guide

This document is intended to help the Dubbo service access the Apache ShenYu gateway. The Apache ShenYu gateway uses the Dubbo plugin to handle dubbo service.

Support Alibaba Dubbo(< 2.7.x) and Apache Dubbo (>=2.7.x).

Before the connection, start `shenyu-admin` correctly, start Dubbo plugin, and add related dependencies on the gateway and Dubbo application client. Refer to the previous Quick start with Dubbo .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config .

10.1 Add dubbo plugin in gateway

Add these dependencies in gateway's `pom.xml`.

Alibaba dubbo user, configure the dubbo version and registry center with yours.

```
<!-- 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 user, configure the dubbo version and registry center with yours.

```
<!-- 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 -->
```

- restart gateway service.

10.2 Dubbo service access gateway

Dubbo integration with gateway, please refer to : shenyu-examples-dubbo .

- Alibaba Dubbo User
  - SpringBoot

    Add these dependencies:

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

  - Spring

    Add these dependencies:

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

    Inject these properties into your Spring beans XML file:

```
<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 User
  - SpringBoot

    Add these dependencies:

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

    Add these in your client project' s application.yml:

```
dubbo:
  registry:
    address: dubbo register address
    port: dubbo service port

shenyu:
  register:
    registerType: shenyu service register type #http #zookeeper #etcd
#nacos #consul
    serverLists: shenyu service register address #http://localhost:9095
#localhost:2181 #http://localhost:2379 #localhost:8848
  client:
    dubbo:
      props:
        contextPath: /your contextPath
```

```
        appName: your app name
```

- Spring

  Add these dependencies:

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

  Injecct these properties into your Spring beans XML file:

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

 <!-- Config register repository according to your register type -->
 <bean id="shenyuRegisterCenterConfig" class="org.apache.shenyu.register.
common.config.ShenyuRegisterCenterConfig">
     <property name="registerType" value="your service registerType"/>
     <property name="serverLists" value="your service register serverLists
"/>
 </bean>

 <!-- Client properties config -->
 <bean id="clientPropertiesConfig"
    class="org.apache.shenyu.register.common.config.ShenyuClientConfig.
ClientPropertiesConfig">
    <property name="props">
        <map>
            <entry key="contextPath" value="/your contextPath"/>
            <entry key="appName" value="your appName"/>
        </map>
    </property>
 </bean>

<!-- Config register repository according to your register type -->
<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>
```

Add these in your client project' s application.yml:

```
dubbo:
  registry:
    address: dubbo register address
    port: dubbo service port
```

10.3 Dubbo configuration

- Enable dubbo option in `shenyu-admin`.
- Configure your registry address in dubbo.

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

10.3.1 Configure the interface with gateway

- you can add the annotation `@ShenyuDubboClient` to your dubbo service implementation class, so that the interface method will be configured with gateway.
- Start your provider. After successful startup, go to PluginList -> rpc Proxy -> dubbo in the backend management system. You will see auto-registered selectors and rules information.

10.3.2 Dubbo user request and parameter explanation.

- Communicate with dubbo service through Http transport protocol.
- Apache ShenYu gateway need a route prefix which configured when accessing the project.

```
# for example: you have an order service and it has a interface, registry address:
/order/test/save

# now we can communicate with gateway through POST request http://localhost:9195/
order/test/save

# localhost:9195 is gateway's ip port, default port is 9195 , /order is the
contextPath you set through gateway.
```

- parameter deliver:
  - communicate with gateway through body or json of http post request.

  - more parameter types, please refer to the interface definition in shenyu-examples-dubbo and parameter passing method.

- Single java bean parameter type (`default`).
- Multi-parameter type support, add this config value in gateway's yaml file:

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

- Support for customized multi-parameter type
- Create a new implementation class `MyDubboParamResolveService` in your gateway project of `org.apache.shenyu.web.dubbo.DubboParamResolveService`.

```
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 is the json string in http request.
- `parameterTypes`: the list of method parameter types that are matched, split with `,`.
- in Pair, left is parmeter type, right is parameter value, it's the standard of dubbo generalization calls.
- Inject your class into Spring bean, cover the default implementation.

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

10.4 Service governance

- Tag route
  - Add `Dubbo_Tag_Route` when send request, the current request will be routed to the provider of the specified tag, which is only valid for the current request.
- Explicit Target
  - Set the `url` property in the annotation `@ShenyuDubboClient`.
  - Update the configuration in Admin.
  - It’ s valid for all request.
- Param valid and ShenyuException
  - Set `validation="shenyuValidation"`.
  - When `ShenyuException` is thrown in the interface, exception information will be returned. It should be noted that `ShenyuException` is thrown explicitly.

```
@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();
        }
}
```

  - Request param

```
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;
    }
}
```

- Send request

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

- Response

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

- Error message

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

10.5 Http –> Gateway –> Dubbo Provider

It basically switches from HTTP request to Dubbo protocol, then invoke Dubbo service and return to the result. Two things need to notice after intgeration with gateway, one is the added annoation `@ShenyuDubboClient`, another is a path used to speicify the request path. And you added a config value of `contextPath`.

If you have a function like this, the config value in contextPath is `/dubbo`

```
@Override
@ShenyuDubboClient(path = "/insert", desc = "insert data")
public DubboTest insert(final DubboTest dubboTest) {
```

```
    return dubboTest;
}
```

So our request path is: http://localhost:9195/dubbo/insert, localhost:9195 is the gateway's domain name,if you changed before,so does with yours here..

`DubboTest` is a java bean object, has 2 parameters, id and name, so we can transfer the value's json type through request body.

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

If your interface has no parameter, then the value is:

```
{}
```

If the interface has multiple parameters, refer to the multi-parameter type support described above.

This document focuses on how to use different data synchronization strategies. Data synchronization refers to the strategy used to synchronize data to ShenYu gateway after shenyu-admin background operation data. ShenYu gateway currently supports ZooKeeper, WebSocket, HTTP Long Polling, Nacos, Etcd and Consul for data synchronization.

For details about the data synchronization principles, see Data Synchronization Design in the design document.

10.6 WebSocket Synchronization Config（default strategy, recommend）

- `Apache ShenYu` gateway config

  Add these dependencies in `pom.xml`:

```
<!-- 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>
```

Add these config values in yaml file:

```
shenyu:
  sync:
    websocket :
      urls: ws://localhost:9095/websocket
      #urls: address of shenyu-admin, multi-address will be separated with (,).
```

- `shenyu-admin` config

Add these config values in yaml file:

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

After the connection is established, the data will be fully obtained once, and the subsequent data will be updated and added increments, with good performance. It also supports disconnection (default: `30` seconds). This mode is recommended for data synchronization and is the default data synchronization strategy of ShenYu.

10.7 Zookeeper Synchronization Config

Please pay attention! From ShenYu 2.5.0, ShenYu will no longer support Zookeeper 3.4.x or below version. If you're already using Zookeeper, You need to use Zookeeper with a higher version and initialize the data.

- `Apache ShenYu` gateway config

Add these dependencies in `pom.xml`:

```
<!-- 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>
```

Add these config values in yaml file:

```
shenyu:
  sync:
    zookeeper:
      url: localhost:2181
       #url: config with your zk address, used by the cluster environment,
separated with (,).
      sessionTimeout: 5000
      connectionTimeout: 2000
```

- `shenyu-admin` config

Add these config values in yaml file:

```
shenyu:
  sync:
    zookeeper:
      url: localhost:2181
       #url: config with your zk address, used by the cluster environment,
separated with (,).
```

```
      sessionTimeout: 5000
      connectionTimeout: 2000
```

It is a good idea to use ZooKeeper synchronization mechanism with high timeliness, but we also have to deal with the unstable environment of ZK, cluster brain splitting and other problems.

10.8 HTTP Long Polling Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in pom.xml:

```
<!-- 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>
```

Add these config values in yaml file:

```
shenyu:
    sync:
        http:
             url: http://localhost:9095
        #url: config your shenyu-admin  ip and port, cluster IP by split by (,)
```

- shenyu-admin config

  Add these config values in yaml file:

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

HTTP long-polling makes the gateway lightweight, but less time-sensitive. It pulls according to the group key, if the data is too large, it will have some influences, a small change under a group will pull the entire group.

10.9 Nacos Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in pom.xml:

```
<!-- 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>
```

Add these config values in yaml file:

```
shenyu:
  sync:
    nacos:
      url: localhost:8848
         # url: config with your nacos address, please use (,) to split your
cluster environment.
      namespace: 1c10d748-af86-43b9-8265-75f487d20c6c
      username:
      password:
      acm:
        enabled: false
        endpoint: acm.aliyun.com
        namespace:
        accessKey:
        secretKey:
     # other configure, please refer to the naocs website.
```

- shenyu-admin config

  Add these config values in yaml file:

```
shenyu:
  sync:
      nacos:
        url: localhost:8848
        namespace: 1c10d748-af86-43b9-8265-75f487d20c6c
        username:
        password:
        acm:
          enabled: false
          endpoint: acm.aliyun.com
          namespace:
          accessKey:
          secretKey:
```

```
        # url: config with your nacos address, pls use (,) to split your cluster
environment.
        # other configure, pls refer to the naocs website.
```

10.10 Etcd Synchronization Config

- Apache ShenYu gateway config

  Add these dependencies in `pom.xml`:

```
<!-- 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>
```

Add these config values in yaml file:

```
shenyu:
    sync:
       etcd:
         url: http://localhost:2379
       #url: config with your etcd address, used by the cluster environment,
separated with (,).
```

- `shenyu-admin` config

  Add these config values in yaml file:

```
shenyu:
  sync:
    etcd:
      url: http://localhost:2379
       #url: config with your etcd address, used by the cluster environment,
separated with (,).
```

10.11 Consul Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in pom.xml:

```
<!-- 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>
```

Add these config values in yaml file:

```
shenyu:
    sync:
      consul:
                url: http://localhost:8500
        waitTime: 1000  # query wait time
        watchDelay: 1000    # Data synchronization interval
```

- shenyu-admin config

  Add these config values in yaml file:

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

After the data synchronization strategy of Apache ShenYu gateway and shenyu-admin
is reconfigured, the microservice needs to be restarted.

the Apache ShenYu gateway and shenyu-admin must use the same synchronization
strategy.
```

Application client access means to access your microservice to ShenYu gateway, currently supports HTTP, Dubbo, Spring Cloud, gRPC, Motan, Sofa, Tars and other protocols access.

Connecting the application client to ShenYu gateway is realized through the registration center, which involves the registration of the client and the synchronization of the server data. The registry supports HTTP, ZooKeeper, Etcd, Consul, and Nacos.

This article describes how to configure the application client to access the Apache ShenYu gateway. For related principles, see Application Client Access in the design document .

10.12 Http Registry Config

10.12.1 shenyu-admin config

Set the register type to ' Http in the yml file. The configuration information is as follows:

```
shenyu:
  register:
    registerType: http
    props:
        checked: true  # is checked
        zombieCheckTimes: 5 # how many times does it fail to detect the service
        scheduledTime: 10 # timed detection interval time
```

10.12.2 shenyu-client config

The following shows the configuration information registered through Http when the Http service accesses the Apache ShenYu gateway as a client. Other clients (such as Dubbo and Spring Cloud) can be configured in the same way.

```
shenyu:
  register:
    registerType: http
    serverLists: http://localhost:9095
  client:
    http:
        props:
            contextPath: /http
            appName: http
            port: 8188
            isFull: false
# registerType : register type, set http
# serverList: when register type is http, set shenyu-admin address list, pls note
'http://' is necessary.
# port: your project port number; apply to springmvc/tars/grpc
# contextPath: your project's route prefix through shenyu gateway, such as /order ,
/product etc, gateway will route based on it.
# appName: your project name,the default value is`spring.application.name`.
# isFull: set true means providing proxy for your entire service, or only a few
controller. apply to springmvc/springcloud
```

10.13 Zookeeper Registry Config

Please pay attention! From ShenYu 2.5.0, ShenYu will no longer support Zookeeper 3.4.x or below version. If you're already using Zookeeper, You need to use Zookeeper with a higher version and initialize the data.

10.13.1 shenyu-admin config

First add the related dependencies to the pom file (already added by default) :

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

- In the yml file, set the register type to zookeeper and enter the service address and parameters of zookeeper. The configuration information is as follows:

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

10.13.2 shenyu-client config

The following shows the configuration information registered by zookeeper when the Http service accesses the Apache ShenYu gateway as a client. Other clients (such as Dubbo and Spring Cloud) can be configured in the same way.

- First add dependencies to the pom file:

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

- Then set the register type to zookeeper in yml and enter the service address and related parameters as follows:

```
shenyu:
  register:
    registerType: zookeeper
    serverLists: localhost:2181
  client:
    http:
        props:
            contextPath: /http
            appName: http
            port: 8189
            isFull: false
# registerType : register type, set zookeeper
# serverList: when register type is zookeeper, set zookeeper address list
# port: your project port number; apply to springmvc/tars/grpc
# contextPath: your project's route prefix through shenyu gateway, such as /order ,
/product etc, gateway will route based on it.
# appName: your project name,the default value is`spring.application.name`.
# isFull: set true means providing proxy for your entire service, or only a few
controller. apply to springmvc/springcloud
```

10.14 Etcd Registry Config

10.14.1 shenyu-admin config

First add the related dependencies to the pom file (already added by default) :

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

- Then set register type to `etcd` in `yml` and enter `etcd` service address and parameters. The configuration information is as follows:

```
shenyu:
  register:
    registerType: etcd
    serverLists : http://localhost:2379
    props:
      etcdTimeout: 5000
      etcdTTL: 5
```

10.14.2 shenyu-client config

The following shows the configuration information registered by Etcd when the Http service accesses the Apache ShenYu gateway as a client. Other clients (such as Dubbo and Spring Cloud) can be configured in the same way.

- First add dependencies to the pom file:

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

- Then set the register type to etcd in yml and enter the etcd service address and related parameters as follows:

```
shenyu:
  register:
    registerType: etcd
    serverLists: http://localhost:2379
  client:
    http:
        props:
            contextPath: /http
            appName: http
            port: 8189
            isFull: false
# registerType : register type, set etcd
# serverList: when register type is etcd, add etcd address list
# port: your project port number; apply to springmvc/tars/grpc
# contextPath: your project's route prefix through shenyu gateway, such as /order ,
/product etc, gateway will route based on it.
# appName: your project name,the default value is`spring.application.name`.
# isFull: set true means providing proxy for your entire service, or only a few
controller. apply to springmvc/springcloud
```

10.15 Consul Registry Config

10.15.1 shenyu-admin config

First add the related dependencies to the pom file :

```
<!-- 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-->
```

- In the `yml` file to configure the registry as `consul`, the unique configuration of consul is configured under the props node, the configuration information is as follows:

```
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 : register type, set consul.
# serverLists: consul client agent address (sidecar deployment (single machine or
cluster), or the address of consul server agent (only one node of consul server
agent can be connected, if it is a cluster, then there will be a single point of
failure))
# delay: The interval of each polling of monitoring metadata, in seconds, the
default value is 1 second.
# wait-time: The waiting time for each polling of metadata monitoring, in seconds,
the default value is 55 second.
# instanceId: Required, Consul needs to find specific services through instanceId.
# name: The name where the service is registered to consul.
# hostName: When registering the type for consul, fill in the address of the
registered service instance. The service instance address registered in the
registry will not be used for client calls, so this configuration does not need to
be filled in. Port and preferAgentAddress are the same.
# port: When registering the type for consul, fill in the port of the registered
service instance.
# tags: Corresponding to the tags configuration in the consul configuration
```

```
# preferAgentAddress: Using the address corresponding to the agent on the consul
client side as the address of the registered service instance will override the
manual configuration of hostName
# enableTagOverride: Corresponding to the enableTagOverride configuration in the
consul configuration
```

10.15.2 shenyu-client config

Register configuration information through the `Consul` method (the registry of the springCloud service itself can be selected at will, and there will be no conflict with the registry selected by shenyu, eureka is used in the example).

- First add dependencies to the pom file:

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

- Then set the register type to `consul` in `yml` and config `shenyu.register.props`, and related parameters as follows:

```
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 : register type, set consul.
# serverLists: consul client agent address (sidecar deployment (single machine or
cluster), or the address of consul server agent (only one node of consul server
agent can be connected, if it is a cluster, then there will be a single point of
failure))
# delay: The interval of each polling of monitoring metadata, in seconds, the
default value is 1 second.
```

```
# wait-time: The waiting time for each polling of metadata monitoring, in seconds,
the default value is 55 second.
# instanceId: Required, Consul needs to find specific services through instanceId.
# name: The name where the service is registered to consul.
# hostName: When registering the type for consul, fill in the address of the
registered service instance. The service instance address registered in the
registry will not be used for client calls, so this configuration does not need to
be filled in. Port and preferAgentAddress are the same.
# port: When registering the type for consul, fill in the port of the registered
service instance.
# tags: Corresponding to the tags configuration in the consul configuration
# preferAgentAddress: Using the address corresponding to the agent on the consul
client side as the address of the registered service instance will override the
manual configuration of hostName
# enableTagOverride: Corresponding to the enableTagOverride configuration in the
consul configuration
```

10.16 Nacos Registry Config

10.16.1 shenyu-admin config

First add the related dependencies to the pom file (already added by default) :

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

- Then in the yml file, configure the registry as nacos, fill in the related nacos service address and parameters, and nacos namespace (need to be consistent with shenyu-client), the configuration information is as follows:

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

10.16.2 shenyu-client config

The following shows the configuration information registered by Nacos when the Http service accesses the Apache ShenYu gateway as a client. Other clients (such as Dubbo and Spring Cloud) can be configured in the same way.

- First add dependencies to the pom file:

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

- Then in yml configure registration mode as nacos, and fill in nacos service address and related parameters, also need nacos namespace (need to be consistent with shenyu-admin), IP (optional, then automatically obtain the local IP address) and port, configuration information is as follows:

```
shenyu:
  register:
    registerType: nacos
    serverLists: localhost:8848
    props:
      nacosNameSpace: ShenyuRegisterCenter
  client:
    http:
        props:
            contextPath: /http
            appName: http
            port: 8188
            isFull: false
# registerType : register type, set nacos
# serverList: when register type is nacos, add nacos address list
# port: your project port number; apply to springmvc/tars/grpc
# contextPath: your project's route prefix through shenyu gateway, such as /order ,
/product etc, gateway will route based on it.
# appName: your project name,the default value is`spring.application.name`.
# isFull: set true means providing proxy for your entire service, or only a few
controller. apply to springmvc/springcloud
# nacosNameSpace: nacos namespace
```

10.17 Register different type API at same time

follow example use the http and dubbo.

the yml configuration like follow:

```
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 : register type, set nacos
# serverList: when register type is nacos, add nacos address list
# http.port: your project port number; apply to springmvc
# http.contextPath: your project's route prefix through shenyu gateway, such as /
order , /product etc, gateway will route based on it.
# http.appName: your project name,the default value is`spring.application.name`.
# http.isFull: set true means providing proxy for your entire service, or only a
few controller. apply to springmvc/springcloud
# dubbo.contextPath: your project dubbo service's context path
# dubbo.port: your project dubbo rpc port
# dubbo.appName: your project dubbo appliation name
# nacosNameSpace: nacos namespace
```

In conclusion, this paper mainly describes how to connect your microservices (currently supporting Http, Dubbo, Spring Cloud, gRPC, Motan, Sofa, Tars and other protocols) to the Apache ShenYu gateway. the Apache ShenYu gateway support registry has Http, Zookeeper, Etcd, Consul, Nacos and so on. This paper introduces the different ways to register configuration information when Http service is used as the client to access Apache ShenYu gateway.

This document is intended to help the Http service access the Apache ShenYu gateway. The Apache ShenYu gateway uses the Divide plugin to handle Http requests.

Before the connection, start shenyu-admin correctly, start Divide plugin, and add related dependencies on the gateway and Http application client. Refer to the previous Quick start with Http .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config .

10.18 Add divide plugin in gateway

- Add the following dependencies to the gateway' s `pom.xml` file:

```
<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>
```

10.19 Http request access gateway (for springMvc)

- SpringBoot

  Please refer this: shenyu-examples-http

  Add the following dependencies to the `pom.xml` file in your `Http` service:

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

- SpringMvc

  Please refer this: shenyu-examples-springmvc

  Add the following dependencies to the `pom.xml` file in your `Http` service:

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

  Add the following to the `XML` file defined by your `bean` :

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

<!-- Config register center according to your register type-->
<bean id="shenyuRegisterCenterConfig" class="org.apache.shenyu.register.common.
config.ShenyuRegisterCenterConfig">
    <property name="registerType" value="http"/>
    <property name="serverLists" value="http://localhost:9095"/>
</bean>

<!-- Client properties config -->
<bean id="clientPropertiesConfig"
      class="org.apache.shenyu.register.common.config.ShenyuClientConfig.
ClientPropertiesConfig">
  <property name="props">
      <map>
          <entry key="contextPath" value="/your contextPath"/>
          <entry key="appName" value="your appName"/>
          <entry key="port" value="your port"/>
          <entry key="isFull" value="false"/>
      </map>
  </property>
</bean>

<!-- Config register repository according to your register type -->
<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>
```

Add this annotation `@ShenyuSpringMvcClient` in your `controller` interface.

You can apply the annotation to class-level in a controller. The name of the `path` variable is prefix and `/**` will apply proxy for entire interfaces.

Example(1)

The following indicates that `/test/payment`, `/test/findByUserId` will be proxy by the gateway.

```
@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;
    }
 }
```

Example(2)

The following indicates that `/order/save` is proxied by the gateway, while `/order/findById` is not.

```
@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;
    }
}
```

example (3): This is a simplified way to use it, just need a simple annotation to register to the gateway using metadata. Special note: currently only supports `@RequestMapping`, `@GetMapping`,

`@PostMapping`, `@DeleteMapping`, `@PutMapping` annotations, and only valid for the first path in `@XXXMapping`

```
@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");
  }
}
```

- Start your project, your service interface is connected to the gateway, go to the shenyu-admin management system plugin list -> HTTP process -> Divide, see automatically created selectors and rules.

10.20 Http request access gateway(other framework)

- First, find divide plugin in shenyu-admin, add selector, and rules, and filter traffic matching.
- If you don't know how to configure, please refer to Selector Detailed Explanation.
- You can also develop your customized http-client, refer to multi-language Http client development。

10.21 User request

- Send the request as before, only two points need to notice.
- Firstly, the domain name that requested before in your service, now need to replace with gateway's domain name.
- Secondly, Apache ShenYu Gateway needs a route prefix which comes from contextPath, it configured during the integration with gateway, you can change it freely in divide plugin of shenyu-admin, if you are familiar with it.
  - for example, if you have an order service, and it has an interface, the request url: http://localhost:8080/test/save
  - Now need to change to: http://localhost:9195/order/test/save
  - We can see localhost:9195 is your gateway's ip port, default port number is 9195 , /order is your contextPath which you configured with gateway.
  - Other parameters doesn't change in request method.
- Then you can visit, very easy and simple.

This document is intended to help the Sofa service access the Apache ShenYu gateway. The Apache ShenYu gateway uses the Sofa plugin to handle sofa service.

Before the connection, start shenyu-admin correctly, start Sofa plugin, and add related dependencies on the gateway and Sofa application client. Refer to the previous Quick start with Sofa .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config .

10.22 Add sofa plugin in gateway

- Add the following dependencies in the gateway's `pom.xml` file:
- Replace the sofa version with yours, and replace the jar package in the registry with yours, The following is a reference。

```
<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>
```

- Restart the gateway service.

10.23 Sofa service access gateway

you can refer to: shenyu-examples-sofa

- SpringBoot

  Add the following dependencies :

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

- Spring

  Add the following dependencies:

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

Add the following in the xml file of your bean definition:

```
    <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="/your contextPath"/>
             <entry key="appName" value="your name"/>
             <entry key="isFull" value="false"/>
        </map>
   </property>
</bean>
```

10.24 Plugin Settings

- First in the shenyu-admin plugin management, set the sofa plugin to open.
- Secondly, configure your registered address in the sofa plugin, or the address of other registry.

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

10.25 Interface registered to the gateway

- For your sofa service implementation class, add @ShenyuSofaClient annotation to the method, Indicates that the interface method is registered to the gateway.
- Start the sofa service provider, after successful registration, enter the pluginList -> rpc proxy -> sofa in the background management system, you will see the automatic registration of selectors and rules information.

10.26 User request and parameter description

ShenYu gateway needs to have a routing prefix, this routing prefix is for you to access the project for configuration contextPath .

> For example, if you have an order service, it has an interface and its registration path /order/test/save
>
> Now it's to request the gateway via post: http://localhost:9195/order/test/save
>
> Where localhost:9195 is the IP port of the gateway, default port is 9195, /order is the contextPath of your sofa access gateway configuration

- Parameter passing:
  - Access the gateway through http post, and pass through body and json.
  - For more parameter type transfer, please refer to the interface definition in shenyu-examples-sofa and the parameter transfer method.
- Single java bean parameter type (default)
- Customize multi-parameter support:
- In the gateway project you built, add a new class MySofaParamResolveService, implements org.apache.shenyu.plugin.api.sofa.SofaParamResolveService .

```
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 is the json string passed by body in http.
- parameterTypes: list of matched method parameter types, If there are multiple, use , to separate.
- In Pair, left is the parameter type, and right is the parameter value. This is the standard for sofa generalization calls.
- Register your class as a String bean and override the default implementation.

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

This document is intended to help the Motan service access the Apache ShenYu gateway. The Apache ShenYu gateway uses the Motan plugin to handle motan service.

Before the connection, start shenyu-admin correctly, start Motan plugin, and add related dependencies on the gateway and Motan application client. Refer to the previous Quick start with Motan .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config .

10.27 Add motan plugin in gateway

Add the following dependencies to the gateway's pom.xml file:

```
<!-- 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>
```

- Restart your gateway service.

10.28 Motan service access gateway

Please refer to: shenyu-examples-motan

- In the microservice built by `Motan`, add the following dependencies:

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

Add `@ShenyuMotanClient` annotation to the method of `Motan` service interface implementation class, start your service provider, after successful registration, go to PluginList -> rpc proxy -> motan in the background management system, you will see automatic registration of selectors and rules information.

Example:

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

10.29 User Request

You can request your motan service by Http. The Apache ShenYu gateway needs to have a route prefix which is the contextPath configured by the access gateway. For example: http://localhost:9195/motan/hello .

This document is intended to help the gRPC service access the Apache ShenYu gateway. The Apache ShenYu gateway uses the gRPC plugin to handle gRPC service.

Before the connection, start shenyu-admin correctly, start gRPC plugin, and add related dependencies on the gateway and gRPC application client. Refer to the previous Quick start with gRPC .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config .

10.30 Add gRPC plugin in gateway

Add the following dependencies in the gateway's pom.xml file:

```
<!-- 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-->
```

- Restart the gateway service.

10.31 gRPC service access gateway

You can refer to: shenyu-examples-grpc .

- In the microservice built by gRPC, add the following dependencies:

```
<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>
```

Execute command to generate java code in `shenyu-examples-grpc` project.

```
mvn protobuf:compile
mvn protobuf:compile-custom
```

Add `@ShenyuGrpcClient` Annotation on the gRPC service interface implementation class. Start your service provider, after successful registration, in the background management system go to PluginList -> rpc proxy -> gRPC, you will see automatic registration of selectors and rules information.

Example:

```
@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();
}
```

10.32 User Request

You can request your gRPC service by Http. The `Apache ShenYu` gateway needs to have a route prefix that you access to configure `contextPath`.

If your `proto` file is defined as follows:

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

So the request parameters look like this:

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

The parameters are currently passed in `json` format, and the name of `key` defaults to `data`, which you can reset in `GrpcConstants.JSON_DESCRIPTOR_PROTO_FIELD_NAME`; The `value` is passed in according to the `proto` file you define.

the Apache ShenYu can support streaming calls to gRPC service, passing multiple arguments in the form of an array.

If your `proto` file is defined as follows:

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

The corresponding method call request parameters are as follows:

- 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"
        }
    ]
}
```

This document will introduce how to register the gateway instance to the registry. The Apache ShenYu gateway currently supports registration to zookeeper , etcd and consul.

10.33 Add Maven dependency

First, introduce the following dependencies in the gateway's pom.xml file.

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

10.34 Use zookeeper

> Please pay attention! From ShenYu 2.5.0, ShenYu will no longer support Zookeeper 3.4.x or below version. If you're already using Zookeeper, You need to use Zookeeper with a higher version and initialize the data.

Add the following configuration to the gateway's yml file:

```
instance:
    enabled: true
    registerType: zookeeper
    serverLists: localhost:2181 #config with your zk address, used by the cluster
environment, separated with (,).
    props:
      sessionTimeout: 3000 #Optional, default 3000
      connectionTimeout: 3000 #Optional, default 3000
```

10.35 Use etcd

Add the following configuration to the gateway' s yml file:

```
instance:
    enabled: true
    registerType: etcd
    serverLists: http://localhost:2379 #config with your etcd address, used by the
cluster environment, separated with (,).
    props:
      etcdTimeout: 3000 #Optional, default 3000
      etcdTTL: 5 #Optional, default 5
```

10.36 Use consul

Add the following configuration to the gateway' s yml file:

```
instance:
    enabled: true
    registerType: consul
    serverLists: localhost:8848 #config with your consul address, used by the
cluster environment, separated with (,).
    props:
      consulTimeout: 3000 #Optional, default 3000
      consulTTL: 3000 #Optional, default 3000
```

> After the configuration is complete, start the gateway and it will successfully register to the corresponding registration center.

This document is intended to help the Spring Cloud service access the Apache ShenYu gateway. The Apache ShenYu gateway uses the springCloud plugin to handle Spring Cloud service.

Before the connection, start shenyu-admin correctly, start springCloud plugin, and add related dependencies on the gateway and springCloud application client. Refer to the previous Quick start with Spring Cloud .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config .

10.37 Add springcloud plugin in gateway

- add these dependencies in gateway' s pom.xml:

```
<!-- 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>
```

- If you use `eureka` as SpringCloud registry center.

  add these dependencies:

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

add these config values in gateway' s yaml file:

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

- if you use `nacos` as Spring Cloud registry center.

  add these dependencies:

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

```
      <version>2.1.0.RELEASE</version>
</dependency>
```

add these config values in gateway' s yaml file:

```
spring:
   cloud:
     nacos:
       discovery:
          server-addr: 127.0.0.1:8848 # your nacos address
```

Special note: Please ensure that the spring Cloud registry service discovery configuration is enabled

- Configuration method

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

- code method

```
@SpringBootApplication
@EnableDiscoveryClient
public class ShenyuBootstrapApplication {

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

- restart your gateway service.

10.38 SpringCloud service access gateway

Please refer to shenyu-examples-springcloud

- Add the following dependencies to your `Spring Cloud` microservice :

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

- Add the annotation `@ShenyuSpringCloudClient` in your `controller` interface. you can apply the annotation to class-level in a controller.the name of the path variable is prefix and '/**' will apply proxy for entire interfaces.
- example (1): both `/test/payment` and `/test/findByUserId` will be handled by gateway.

```
@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;
    }
 }
```

example (2): `/order/save` will be handled by gateway, and `/order/findById` won't.

```
@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;
    }
}
```

example (3): `isFull: true` represents that all service will be represented by the gateway.

```
shenyu:
  client:
    registerType: http
    serverLists: http://localhost:9095
    props:
      contextPath: /http
      appName: http
      isFull: true
# registerType : service registre type, see the application client access document
# serverList: server list, see the application client access document
# contextPath: route prefix for your project in ShenYu gateway.
# appName: your application name
# isFull: set true to proxy your all service and false to proxy some of your
controllers
```

```
@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;
    }
}
```

example (4): This is a simplified way to use it, just need a simple annotation to register to the gateway using metadata. Special note: currently only supports `@RequestMapping`, `@GetMapping`, `@PostMapping`, `@DeleteMapping`, `@PutMapping` annotations, and only valid for the first path in `@XXXMapping`.

```
@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");
    }
}
```

- After successfully registering your service, go to the backend management system PluginList -> rpc proxy -> springCloud ' , you will see the automatic registration of selectors and rules information.

10.39 User Request

- Send the request as before, only two points need to notice.
- firstly,the domain name that requested before in your service, now need to replace with gateway's domain name.
- secondly, Apache ShenYu gateway needs a route prefix which comes from `contextPath`, it configured during the integration with gateway, you can change it freely in divide plugin of `shenyu-admin`, if your familiar with it.

  For example, your have an `order` service and it has a interface, the request url: `http://localhost:8080/test/save`.

  Now need to change to: `http://localhost:9195/order/test/save`.

  We can see `localhost:9195` is the gateway's ip port, default port number is `9195`, `/order` is the `contextPath` in your config yaml file.

  The request of other parameters doesn't change. Then you can visit, very easy and simple.

This document is intended to help the `Tars` service access the `Apache ShenYu` gateway. The `Apache ShenYu` gateway uses the `tars` plugin to handle `tars` service.

Before the connection, start `shenyu-admin` correctly, start `tars` plugin, and add related dependencies on the gateway and `tars` application client. Refer to the previous Quick start with Tars .

For details about client access configuration, see Application Client Access Config .

For details about data synchronization configurations, see Data Synchronization Config).

10.40 Add tars plugin in gateway

Add the following dependencies to the gateway's `pom.xml` file:

```
<!-- 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-->
```

- Restart your gateway service.

10.41 Tars service access gateway

Please refer to: shenyu-examples-tars

- In the microservice built by Tars, add the following dependencies:

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

Add @ShenyuTarsService Annotation on the tars service interface implementation class and @ShenyuTarsClient on the method, start your service provider, and register successfully. In the background management system, enter PluginList -> rpc proxy -> tars, you will see the automatic registration of selectors and rules information.

Example:

```
@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;
    }
}
```

10.42 User Request

You can request your tars service by Http. The Apache ShenYu gateway needs to have a route prefix which is the contextPath configured by the access gateway. For example: http://localhost:9195/tars/hello.

11

Plugin Center

12 Developer Documentation

12.1 Description

- Users can customize the signature authentication algorithm to achieve verification.

12.2 Extension

- The default implementation is `org.apache.shenyu.plugin.sign.service.DefaultSignService`.
- Declare a new class named `CustomSignService` and implements `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);
}
```

- When returning true in Pair, the sign verification passes. If there's false, the String in Pair will be return to the frontend to show.
- Register defined class as a Spring Bean.

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

12.2.1 Others

If you only want to modify the signature algorithm, refer to the following.

- The default implementation of the signature algorithm is `org.apache.shenyu.common.utils.SignUtils#generateSign`.
- Declare a new class named `CustomSignProvider` and implements `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);
}
```

- Put `CustomSignProvider` to `Spring IoC`

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

12.3 Description

- This doc shows a demo for how to extend `org.springframework.web.server.WebFliter`.

12.4 CORS Support

- `org.apache.shenyu.web.filter.CrossFilter` is designed for `WebFilter` implementation.

```
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);
    }
}
```

- Registering `CrossFilter` as a `Spring Bean`.

12.5 Filtering Spring Boot health check

- You can control the order by applying `@Order` to the implementation class .

```
@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);
    }
}
```

12.6 Extending org.apache.shenyu.web.filter.AbstractWebFilter

- Add a new class and inherit from `org.apache.shenyu.web.filter.AbstractWebFilter`.
- Implement abstract methods of parent class.

```
/**
  * 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);
```

- if method `doFilter` returns `Mono<true>`, this filter is passing, While rejecting, it will call method `doDenyResponse` and sending infos in response body to frontend.

12.7 Description

- This document focuses on how to access gateways for HTTP services in other languages.
- How to customize the development of shenyu-http-client.

12.8 Customize Http Client

- Request Method: POST
- Request Path: http://soul-admin/soul-client/springmvc-register, shenyu-admin represents IP + Port of admin
- Request Params： passing JSON type parameters through the body.

```
{
    "appName": "xxx", //required
    "context": "/xxx", //required
    "path": "xxx", //required
    "pathDesc": "xxx",
    "rpcType": "http", //required
    "host": "xxx", //required
    "port": xxx, //required
    "ruleName": "xxx", //required
    "enabled": "true", //required
    "registerMetaData": "true" //required
}
```

12.9 Preparation

1. Clone the code of Apache ShenYu.
2. Install and start docker.

12.10 Start integration test locally

1. Build with Maven

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

2. Build integrated tests

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

3. Start docker compose

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

You need to replace `${{ matrix.case }}` with the exact directory, such as `shenyu-integrated-test-http`.

4. Run test

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

12.11 Description

- This doc offers examples for customising response structure in Apache ShenYu gateway.
- The response body structure in gateways should be unified, it is recommended for specify yours.

12.12 Default Implementation

- The default implementation class is `org.apache.shenyu.plugin.api.result.DefaultShenyuResult`.
- Following is the response structure:

```
public class ShenyuDefaultEntity implements Serializable {

    private static final long serialVersionUID = -2792556188993845048L;

    private Integer code;

    private String message;

    private Object data;

}
```

- The returned `json` as follows:

```
{
    "code": -100, //response code,
    "message": "Your parameter error, please check the relevant documentation!", //
hint messages
    "data": null  // business data
}
```

12.13 Extensions

- Declare a new class named `CustomShenyuResult` and implements `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);
}
```

Processing sequence: `format->``contextType``->``result``. The `format` method performs data formatting. If the data is a basic type and returns itself, other types are converted to `JSON`, and the parameter `origin` is the original data. Formatting can be performed according to the situation. `contextType`, if it is a basic type, use `text/plain`, the default is `application/json`, the parameter `formatted` is the data processed by the `format` method, and can be combined with the return result of `format` for data type Define processing. The parameter `formatted` of `result` is the data processed by the `format` method, which returns to itself by default, and can be combined with the return result of `format` for custom processing of the data type.

- `T` is a generic parameter for your response data.
- Register defined class as a `Spring Bean`.

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

12.14 Description

- Plugins are core executors of `Apache ShenYu` gateway. Every plugin handles matched requests when enabled.
- There are two kinds of plugins in the `Apache ShenYu` gateway.
  - The first type is a chain with single responsibility, and can not custom filtering of traffic.
  - The other one can do its own chain of responsibility for matched traffic.
- You could reference from shenyu-plugin module and develop plugins by yourself. Please fire pull requests of your wonderful plugins without hesitate.

12.15 Single Responsibility Plugins

- Add following dependency:

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

- Declare a new class named `MyShenyuPlugin` and implements `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 offer 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;
    }
}
```

Detailed instruction of interface methods:

- `execute()` core method, you can do any task here freely.
- `getOrder()` get the order of current plugin.
- `named()` acquire the name of specific plugin that uses the `Camel Case`, eg: `dubbo`, `springCloud`.
- `skip()` determines whether this plugin should be skipped under certain conditions.
- Register plugin in `Spring` as a `Bean`, or simply apply `@Component` in implementation class.

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

12.16 Matching Traffic Processing Plugin

- Introduce the following dependency:

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

- Add a new class `CustomPlugin`, inherit from `org.apache.shenyu.plugin.base.AbstractShenyuPlugin`
- examples down below:

```
/**
 * 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);
    }
}
```

- Detailed explanation:
  - Plugins will match the selector rule for customized plugins inherit from this abstract class.
  - Firstly define a new plugin in `shenyu-admin` -> BasicConfig -> Plugin, please mind that your plugin name should match the `named()` method overridden in your class.
  - Re-login `shenyu-admin`, the plugin you added now showing on plugin-list page, you can choose selectors for matching.
  - There is a field named `handler` in rules, it is customized json string to be processed. You can process data after acquiring a ruleHandle (`final String ruleHandle = rule.getHandle();`) in `doExecute()` method.
- Register plugin in `Spring` as a `Bean`, or simply apply `@Component` in implementation class.

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

12.17 Subscribe your plugin data and do customized jobs

- Declare a new class named `PluginDataHandler` and implements `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();

}
```

- Ensure `pluginNamed()` is same as the plugin name you defined.
- Register defined class as a `Spring Bean`, or simply apply `@Component` in implementation class.

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

12.18 Dynamic loading

- When using this feature, the above extensions `ShenyuPlugin`, `PluginDataHandler`, do not need to be `spring bean`. You just need to build the jar package of the extension project.
- Config in Yaml:

```
shenyu:
  extPlugin:
    path:  //Load the extension plugin jar package path
    enabled: true //Whether to turn on
    threads: 1  //Number of loading plug-in threads
    scheduleTime: 300  //Cycle time (in seconds)
    scheduleDelay: 30 //How long the shenyu gateway is delayed to load after it
starts (in seconds)
```

12.18.1 Plugin loading path details

- This path is for the directory where the extended plugin jar package is stored。
- Used `-Dplugin-ext=xxxx`, Also used `shenyu.extPlugin.path` in yaml, If neither is configured, the `ext-lib` directory in the apache shenyu gateway boot path will be loaded by default.
- Priority： `-Dplugin-ext=xxxx` > `shenyu.extPlugin.path` > `ext-lib(default)`

12.19 Description

- This doc demonstrates how to get correct IP address and host when Apache ShenYu serves behind nginx reverse proxy.
- After fetched real IP and host, you can match them with plugins and selectors.

12.20 Default Implementation

- The embedded implementation in Apache ShenYu is :`org.apache.shenyu.web.forward.ForwardedRemoteAddressResolver`.
- You need to config `X-Forwarded-For` in nginx first to get correct IP address and host.

12.21 Implement through a Plugin

- Declare a new class named `CustomRemoteAddressResolver` and implements `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();
    }

}
```

- Register defined class as a Spring Bean.

```
@Bean
public SignService customRemoteAddressResolver() {
```

```
        return new CustomRemoteAddressResolver();
}
```

12.22 Description

- This doc shows how to do performance optimization for Apache ShenYu.

12.23 Time Consumption

- Apache ShenYu is JVM driven and processing time for a single request is nearly between 1-3 ms.

12.24 Netty Optimization

- `spring-webflux` is one of dependencies of ShenYu, and it uses Netty in lower layer.
- The demo down below demonstrates tuning ShenYu by customizing params in Netty.

```
@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));
    }
}
```

- The `shenyu-bootstrap` module offers this class, you may modify it when benchmarking your app if necessary.

- You can get references of business thread model from thread model

12.25 Description

- Standalone environment, then use the local `API` to update the apache shenyu gateway data。
- Common result:

```
success
```

- Common preFix: `localhost:9095/shenyu`
- Common Header: `localKey: 123456`

12.26 Plugin

12.26.1 saveOrUpdate

save or update plugin data

Request Method

POST

Path

/plugin/saveOrUpdate

Request Parameters

Name	Type	Required	Default	Description
PluginData	*PluginData*	True		Plugin data object (pass Json object inside Body)

PluginData

Name	Type	Required	Default	Description
id	String	False		plugin id
name	String	True		plugin name
config	String	False		plugin configuration (Json format)
role	String	False		plugin role
enabled	Boolean	False		whether to turn on

Example

POST body

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

12.26.2 CleanAll

Clear all data (plugins, selectors, rules)

Request Method

GET

Path

/cleanAll

12.26.3 Clean Plugin

Clear plugin data（selector, rule）

Request Method

GET

Path

/cleanPlugin?name = xxxx

RequestParam

Name	Type	Required	Default	Description
name	String	true		plugin name

12.26.4 Delete plugin

Remove plugin data (not included, the selectors and rules data)

Request Method

GET

Path

/plugin/delete?name = xxxx

RequestParam

Name	Type	Required	Default	Description
name	String	true		plugin name

12.26.5 Delete All Plugin

Remove all plugin data (not included, the selectors and rules data)

Request Method

GET

Path

/plugin/deleteAll

12.26.6 Find plugin by name

Find plugin by name

Request Method

GET

Path

/plugin/findByName?name=xxxx

RequestParam

Name	Type	Required	Default	Description
name	String	true		plugin name

12.26.7 Save or Update Selector

Save or Update Selector

Request Method

POST

Path

/plugin/selector/saveOrUpdate

RequestParam

Name	Type	Required	Default	Description
SelectorData	*SelectorData*	True		Selector object (pass Json object inside Body)

SelectorData

Name	Type	Required	Default	Description
id	String	False		selector id
pluginName	String	True		plugin name
name	String	False		Selector name (default is plugin:selector+random number if not filled)
matchMode	Integer	False		Matching mode (0:and;1:or), not filled with the default generation And mode
type	Integer	False		Traffic type(0: full traffic; 1: custom traffic) do not fill in the default generation of full traffic
sort	Integer	False		Sort by, not filled by default generate 10
enabled	Boolean	False		Whether to turn on, not fill in the default generation true
logged	Boolean	False		Whether or not to print the log, do not fill in the default generated into false
handle	String	False		Selector handler (Json objects, depending on each plug-in, different objects are passed)
conditionList	*Condition*	False		Conditional collection, custom traffic needs to be passed, full traffic does not need to be passed (Json List object)

Condition

Name	Type	Required	Default	Description
paramType	String	True		param type（post, uri, query, host, header, cookie, req_method, domain）
operator	String	True		operator （match, =, regex, >, <, contains, SpEL, Groovy, TimeBefore，TimeAfter）
paramName	String	False		param mame（The uri parameter type can be passed without）
paramValue	Integer	False		param value

Example

POST body

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

Result

Is selector id

```
xxxxx
```

12.26.8 Add Selector And Rules

Add a selector with multiple rules

Request Method

POST

Path

/plugin/selectorAndRules

RequestParam

Name	Type	Required	Default	Description
SelectorRulesData	*SelectorRulesData*	True		Selector rule object (Body inside pass Json object)

SelectorRulesData

Name	Type	Required	Default	Description
pluginName	String	True		plugin name
selectorName	String	False		Selector name (if not filled in, it is generated by default plugin:selector+random number)
matchMode	Integer	False		Matching mode (0:and;1:or), not filled with the default generation And mode
selectorHandler	String	False		Selector handler (Json objects, depending on each plug-in, different objects are passed)
conditionList	*ConditionData*	True		Selector condition collection (Json List object)
ruleDataList	*RuleLocalData*	True		Rule condition collection (Json List object)

RuleLocalData

Name	Type	Required	Default	Description
ruleName	String	False		rule name
ruleHandler	String	True		Rule handler (different plugins pass different values))
matchMode	Integer	False		Matching pattern (0:and;1:or)
conditionList	*ConditionData*	True		Rule condition collection (Json List object)

ConditionData

Name	Type	Required	Default	Description
paramType	String	True		param type（post, uri, query, host, header, cookie, req_method, domain）
operator	String	True		operator （match, =, regex, >, <, contains, SpEL, Groovy, TimeBefore，TimeAfter）
paramName	String	False		param mame（The uri parameter type can be passed without）
paramValue	Integer	False		param value

Example

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"
        }]
    }]
}
```

12.26.9 Delete Selector

Delete selectors based on selector id and plugin name

Request Method

GET

Path

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

RequestParam

Name	Type	Required	Default	Description
pluginName	String	true		plugin name
id	String	true		selector id

12.26.10 Find All Selector

Get all selectors by plugin name

Request Method

GET

Path

/plugin/selector/findList?pluginName=xxxx

RequestParam

Name	Type	Required	Default	Description
pluginName	String	true		plugin name

12.26.11 Save or Update Rule Data

Save or Update Rule Data

Request Method

POST

Path

/plugin/rule/saveOrUpdate

RequestParam

Name	Type	Required	Default	Description
RuleData	*RuleData*	True		Rule object (pass Json object inside Body)

RuleData

Name	Type	Required	Default	Description
id	String	False		rule id
pluginName	String	True		plugin name
name	String	False		Rule name (default generation if not filled plugin:rule+random number)
selectorId	String	True		Selector id
matchMode	Integer	False		Matching mode (0:and;1:or), not filled with the default generation And mode
sort	Integer	False		Sort by , not filled by default generate 10
enabled	Boolean	False		Whether to turn on, not fill in the default generation true
logged	Boolean	False		Whether or not to print the log, do not fill in the default generated into false
handle	String	False		Rule handler (Json objects, depending on each plug-in, different objects are passed)
conditionList	*ConditionData*	False		Conditional collections (Json List objects)

---

conditionList

Name	Type	Required	Default	Description
paramType	String	True		param type（post, uri, query, host, header, cookie, req_method, domain）
operator	String	True		operator （match, =, regex, >, <, contains, SpEL, Groovy, TimeBefore，TimeAfter）
paramName	String	False		param mame（The uri parameter type can be passed without）
paramValue	Integer	False		param value

Example

POST body

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

Result

Is rule id

```
xxxxx
```

12.26.12 Delete rule data

Delete rules based on selector id and rule id

Request Method

GET

Path

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

RequestParam

Name	Type	Required	Default	Description
selectorId	String	true		selector ID
id	String	true		rule ID

12.26.13 Find Rule data List

Get all rules by selector ID

Request Method

GET

Path

/plugin/rule/findList?selectorId=xxxx

RequestParam

Name	Type	Required	Default	Description
selectorId	String	true		selector id

12.27 Meta data

12.27.1 Save Or Update

Save Or Update Meta data

Request Method

POST

Path

/meta/saveOrUpdate

RequestParam

Name	Type	Required	Default	Description
MetaData	*MetaData*	True		Metadata object (pass Json object inside Body)

MetaData

Name	Type	Required	Default	Description
id	String	False		ID
appName	String	True		app name
contextPath	String	True		contextPath
path	String	True		path
rpcType	String	True		rpc type（dubbo, sofa, tars, springCloud, motan, grpc）
serviceName	String	True		service name
methodName	String	True		method name
parameterTypes	String	True		parameter types
rpcExt	String	False		rpc extension parameters (json objects)
enabled	Boolean	False		Whether to turn on

12.27.2 Delete

Delete Meta data

Request Method

GET

Path

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

RequestParam

Name	Type	Required	Default	Description
rpcType	String	true		rpc type（dubbo, sofa, tars, springCloud, motan, grpc）
path	String	true		path

12.28 App Sign Data

12.28.1 Save Or Update

Save Or Update App Sign Data

Request Method

POST

Path

/auth/saveOrUpdate

RequestParam

Name	Type	Required	Default	Description
AppAuthData	*AppAuthData*	True		Signature object (Json object passed inside the Body)

AppAuthData

Name	Type	Required	Default	Description
appKey	String	True		app key
appSecret	String	True		app secret
enabled	Boolean	False		Whether to turn on
open	Boolean	False		is open
paramDataList	*AuthParamData*	false		Parameter set, open is true when you need to pass (Json list object)
AuthPathData	*AuthPathData*	false		Path collection, open is true when you need to pass (Json list object)

AuthParamData

Name	Type	Required	Default	Description
appName	String	True		app name
appParam	String	True		app param

AuthPathData

Name	Type	Required	Default	Description
appName	String	True		app name
path	String	True		path
enabled	Boolean	False		Whether to turn on

12.28.2 Delete

Delete App Sign Data

Request Method

GET

Path

/auth/delete?appKey=xxxx

RequestParam

Name	Type	Required	Default	Description
appKey	String	true		app key

12.29 Description

- This article gives an introduction to thread models in ShenYu and usage in various scenarios.

12.30 IO And Work Thread

- `spring-webflux` is one of dependencies of ShenYu, and it uses Netty thread model in lower layer.

12.31 Business Thread

- Use scheduling thread to execute by default.
- A fixed thread pool manages business threads, the number of threads is count in this formula: cpu * 2 + 1.

12.32 Type Switching

- `reactor.core.scheduler.Schedulers`.
- `-Dshenyu.scheduler.type=fixed` is a default config. If set to other value, a flexible thread pool will take place it.`Schedulers.elastic()`.
- `-Dshenyu.work.threads = xx` is for configuring number of threads, the default value calculates in following formula `cpu * 2 + 1` with a minimum of 16 threads.

12.33 description

- This doc gives a brief description for upload and download files using `Apache ShenYu`.

12.34 File Upload

- The default file size limit is `10M`.
- For custom limitation, use`--file.size` with an integer variable. e.g.`--file.size = 30`
- Upload your files just as way you did before

12.35 File Download

- `Apache ShenYu` supports download files in stream. There is no need to change anything.