Apache ShenYu document

Apache ShenYu

Jul 31, 2021
1 What is the Apache ShenYu? 1
2 Features 2
   2.1 Architecture Diagram ........................................... 3
3 Design 4
   3.1 Apache ShenYu Admin Database Design ............................ 4
       3.1.1 Plugin, Selector And Rule .................................... 4
       3.1.2 Resource Permission ........................................ 5
       3.1.3 Data Permission ............................................ 6
       3.1.4 Meta Data .................................................. 7
       3.1.5 Dictionary Management ...................................... 8
   3.2 Data Synchronization Design ...................................... 8
       3.2.1 Preface ..................................................... 8
       3.2.2 Principle Analysis .......................................... 9
           Zookeeper Synchronization .................................. 9
           WebSocket Synchronization ................................ 11
           Http Long Polling .......................................... 11
           Nacos Synchronization ...................................... 12
           Etcd Synchronization ....................................... 12
           Consul Synchronization .................................... 12
   3.3 Application Client Access ........................................ 13
       3.3.1 Design principle ........................................... 13
           Client ....................................................... 13
           Server ..................................................... 15
       3.3.2 Http Registry .............................................. 16
       3.3.3 Zookeeper Registry ......................................... 16
       3.3.4 Etcd Registry ............................................ 17
       3.3.5 Consul Registry .......................................... 17
       3.3.6 Nacos Register ........................................... 18
       3.3.7 SPI ...................................................... 19
### 5.7 Quick start with Motan

- **5.7.1 Environment to prepare**
- **5.7.2 Run the shenyu-examples-motan project**
- **5.7.3 Test**

## 6 User Guide

### 6.1 Data Synchronization Config

- **6.1.1 WebSocket Synchronization Config** (default strategy, recommend)
- **6.1.2 Zookeeper Synchronization Config**
- **6.1.3 HTTP Long Polling Synchronization Config**
- **6.1.4 Nacos Synchronization Config**
- **6.1.5 Etcd Synchronization Config**
- **6.1.6 Consul Synchronization Config**

### 6.2 Application Client Access Config

- **6.2.1 Http Registry Config**
- **6.2.2 Zookeeper Registry Config**
- **6.2.3 Etcd Registry Config**
- **6.2.4 Consul Registry Config**
- **6.2.5 Nacos Registry Config**

### 6.3 Http Proxy

- **6.3.1 Add divide plugin in gateway**
- **6.3.2 Http request access gateway (for springMvc)**
- **6.3.3 Http request access gateway (other framework)**
- **6.3.4 User request**

### 6.4 Dubbo Proxy

- **6.4.1 Add dubbo plugin in gateway**
- **6.4.2 Dubbo service access gateway**
- **6.4.3 Dubbo configuration**
  - Configure the interface with gateway
  - Dubbo user request and parameter explanation.
- **6.4.4 Service governance**
- **6.4.5 Http -> Gateway -> Dubbo Provider**

### 6.5 Spring Cloud Proxy

- **6.5.1 Add springcloud plugin in gateway**
What is the Apache ShenYu?

This is an asynchronous, high-performance, cross-language, responsive API gateway.
• 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.
2.1 Architecture Diagram
3.1 Apache ShenYu Admin Database Design

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

3.1.1 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.

3.1.2 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.

3.1.3 Data Permission

• 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.

3.1.4 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.
3.1.5 Dictionary Management

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

3.2 Data Synchronization 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, Etcld and Consul for data synchronization.

See Data Synchronization Configuration for configuration information about data synchronization.

3.2.1 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 Etcld, 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, Etcld and Consul.
3.2.2 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, Etcld, and Consul can now be supported by specifying the corresponding synchronization policy by setting `shenyu.sync.${strategy}` in the configuration file. The default WebSocket 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 shown in the 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 the corresponding event handler according to the configured synchronization strategy.

- If it is a websocket synchronization strategy, it will push modified data to shenyu-web, and the corresponding WebsocketCacheHandler 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.

Zookeeper Synchronization

The zookeeper-based synchronization principle is very simple, it mainly depends on the 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.
3.2. Data Synchronization Design

Figure 1: Zookeeper Node Design
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.

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 BlockingQueue, 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 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.

3.2. Data Synchronization Design
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.

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.

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.
3.3 Application Client Access

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, Etc, Consul, and Nacos.

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

3.3.1 Design principle

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.
3.3. Application Client Access
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.

### 3.3.2 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.

### 3.3.3 Zookeeper Registry

Zookeeper storage struct is:

```plaintext
shenyu
  └──register
      └──metadata
          └─$rpcType
              └─${contextPath}
                  └─${ruleName} : save metadata data of MetaDataRegisterDTO

shenyu
  └──uri
      └─$rpcType
          └─${contextPath}
              └─${ip:prot} : save uri data of URIRegisterDTO
```

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.
### 3.3.4 Etcd Registry

Etcd storage struct is:

```
├──register
│   ├──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.

### 3.3.5 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:

```
├──register
│   ├──metadata
│       └─${rpcType}
```
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.

3.3.6 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.
3.3.7 SPI

<table>
<thead>
<tr>
<th>SPI Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShenyuClientRegisterRepository</td>
<td>ShenYu client register SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpClientRegisterRepository</td>
<td>Http client register repository</td>
</tr>
<tr>
<td>ZookeeperClientRegisterRepository</td>
<td>Zookeeper client register repository</td>
</tr>
<tr>
<td>EtcdClientRegisterRepository</td>
<td>Etcd client register repository</td>
</tr>
<tr>
<td>ConsulClientRegisterRepository</td>
<td>Consul client register repository</td>
</tr>
<tr>
<td>NacosClientRegisterRepository</td>
<td>Nacos client register repository</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPI Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShenyuServerRegisterRepository</td>
<td>ShenYu server register SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShenyuHttpRegistryController</td>
<td>Http server repository</td>
</tr>
<tr>
<td>ZookeeperServerRegisterRepository</td>
<td>Zookeeper server registry repository</td>
</tr>
<tr>
<td>EtcdServerRegisterRepository</td>
<td>Etcd server registry repository</td>
</tr>
<tr>
<td>ConsulServerRegisterRepository</td>
<td>Consul server registry repository</td>
</tr>
<tr>
<td>NacosServerRegisterRepository</td>
<td>Nacos server registry repository</td>
</tr>
</tbody>
</table>

3.4 Flow Control

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

3.4.1 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.
3.4.2 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.
4.1 Local Deployment

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

4.1.1 Environmental preparation

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

4.1.2 Download the compiled code

- Download

```bash
> 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](http://localhost:9095), the default username and password are: admin and 123456 respectively.
  - If you use h2 to store, set the variable `--spring.profiles.active = h2`.
  - If you use MySQL for storage, modify the mysql configuration in application.yaml.

- use the development tool to start org.apache.shenyu.bootstrap.ShenyuBootstrapApplication.
4.2 Binary Packages Deployment

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

4.2.1 Start Apache ShenYu Admin

- download 2.4.0 download apache-shenyu-admin-bin-2.4.0-RELEASE.tar.gz
- unzip apache-shenyu-admin-bin-2.4.0-RELEASE.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, go to the /conf directory, and modify the configuration of mysql in application.yaml.
  > windows: start.bat
  > linux: ./start.sh

4.2.2 Start Apache ShenYu Bootstrap

- download 2.4.0 download apache-shenyu-bootstrap-bin-2.4.0-RELEASE.tar.gz
- unzip apache-shenyu-bootstrap-bin-2.4.0-RELEASE.tar.gz, go to the bin directory.
  > windwos : start.bat
  > linux : ./start.sh

4.3 Docker Deployment

This article introduces the use of docker to deploy the Apache ShenYu gateway.
### 4.3.1 Start Apache ShenYu Admin

```bash
> docker pull apache/shenyu-admin
> docker network create shenyu
```

- use h2 to store data:

```bash
> docker run -d -p 9095:9095 --net shenyu apache/shenyu-admin
```

- use MySQL to store data, copy mysql-connector.jar to /$(your_work_dir)/ext-lib:

```bash
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" -e "spring.datasource.user=${your_username}" -e "spring.datasource.password=${your_password}" -d -p 9095:9095 --net shenyu apache/shenyu-admin
```

Another way is to put the application.yml configuration in ${your_work_dir}/conf, and then execute the following statement:

```bash
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
```

### 4.3.2 Start Apache ShenYu Bootstrap

```bash
> docker network create shenyu
> docker pull apache/shenyu-bootstrap
> docker run -d -p 9195:9195 --net shenyu apache/shenyu-bootstrap
```

### 4.4 k8s Deployment

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

### 4.5 Helm Deployment

This article introduces the use of helm to deploy the Apache ShenYu gateway.
This article describes how to build your own gateway based on Apache ShenYu.

### 4.6.1 Start Apache ShenYu Admin

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

### 4.6.2 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:

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

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

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

```yaml
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
5.1 Quick start with Http

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.

5.1.1 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 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:

```xml
<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>
```
5.1.2 Run the shenyu-examples-http project

Download shenyu-examples-http

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

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}
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 00:57:07.606 INFO 3700 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils : http client register success: {
"appName": "http", "context": "/http",
"path": "/http/order/findById", "pathDesc": "Find by id", "rpcType": "http", "host": "192.168.50.13", "port": 8188,
"ruleName": "/http/order/findById", "enabled": true, "registerMetaData": false}
```

5.1.3 Test

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

Open PluginList –> rpc proxy –> divide to see the list of plugin rule configurations:
5.2 Quick start with Dubbo

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.

5.2.1 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 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:

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

5.2. Quick start with Dubbo
If client is alibaba dubbo, registry center is Zookeeper, please refer to the following configuration:

```xml
<!-- 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>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 alibaba dubbo plugin end-->
```

5.2. Quick start with Dubbo
5.2.2 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:

```
  "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}
```

```
  "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":null,"rpcExt":{
  "group":"","version":"","loadbalance":"random","retries":2,"timeout":10000,"url":null},
  "enabled":true}
```

5.2. Quick start with Dubbo
5.2. Quickstart with Dubbo

```java
@BatchSaveAndNameAndId
public class BatchSaveAndNameAndIdTest {
    @Test
    public void processBatchSaveAndNameAndId() {
        // Test code...
    }
}
```

Apache ShenYu document
5.2.3 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:

```
GET http://localhost:9105/dubbo/findByStringArray
```

```
{
  "name": 200,
  "message": "Access to success!",
  "data": {
    "name": "Hello world Soul Apache, findByStringArray",
    "s": "s"
  }
}
```
Complex multi-parameter example: The related interface implementation class is `org.apache.shenyu.examples.alibaba.dubbo.service.impl.DubboMultiParamServiceImpl#batchSaveAndNameAndId`.

```java
@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:

```
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)
```

5.2. Quick start with Dubbo
5.3 Quick start with Spring Cloud

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.

5.3.1 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:

```xml
<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>
</dependency>
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</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>

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

Restart the shenyu-bootstrap project.

### 5.3.2 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.

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'
  "appName":"springCloud-test",
  "context":/springcloud",
  "path":"/springcloud/order/save",
  "pathDesc":"",
  "rpcType":"springCloud",
  "ruleName":/springcloud/order/save",
  "enabled":true}
  "appName":"springCloud-test",
  "context":/springcloud",
  "path":/springcloud/order/path/**,
  "pathDesc":",
  "rpcType":"springCloud",
  "ruleName":/springcloud/order/path/**",
  "enabled":true}
  "appName":"springCloud-test",
  "context":/springcloud",
  "path":/springcloud/order/findById",
  "pathDesc":",
  "rpcType":"springCloud",
  "ruleName":/springcloud/order/findById",
  "enabled":true}
  "appName":"springCloud-test",
  "context":/springcloud",
  "path":/springcloud/order/path/**/name",
  "pathDesc":",
  "rpcType":"springCloud",
  "ruleName":/springcloud/order/path/**/name",
  "enabled":true}
  "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] oockingLoadBalancerClientRibbonWarnLogger: 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] $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] InstanceInfoFactory: Setting initial instance status as: STARTING
2021-02-10 14:03:53.263 INFO 2860 --- [main] c.n.d.s.r.aws.ConfigClusterResolver: Resolving eureka endpoints via configuration

5.3. Quick start with Spring Cloud
DiscoveryClient : Disable delta property : false
DiscoveryClient : Single vip registry refresh property : null
DiscoveryClient : Force full registry fetch : false
DiscoveryClient : Application is null : false
DiscoveryClient : Registered Applications size is zero : true
DiscoveryClient : Application version is -1: true
DiscoveryClient : Getting all instance registry info from the eureka server
DiscoveryClient : The response status is 200
DiscoveryClient : Starting heartbeat executor: renew interval is: 30
2021-02-10 14:03:53.761 INFO 2860 --- [main] c.n.discovery.
InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min is 4
DiscoveryClient : Discovery Client initialized at timestamp 1612937033760 with initial instances count: 0
2021-02-10 14:03:53.763 INFO 2860 --- [main] o.s.c.n.e.s.
EurekaServiceRegistry : Registering application SPRINGCLOUD-TEST with eureka with status UP
DiscoveryClient : Discovery Client_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
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)

5.3. Quick start with Spring Cloud

37
5.3.3 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:

![Plugin List](image)

Use PostMan to simulate HTTP to request your SpringCloud service:

![PostMan Request](image)

5.4 Quick start with Sofa

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.

5.4.1 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 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:
5.4.2 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:

```java
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}
```
5.4. Quick start with Sofa
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\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\re\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\Microsoft SQL Server\100\Tools\Binn\;C:\Program Files\Microsoft SQL Server\100\DTS\Binn\;C:\Program Files\Docker\Docker\resources\bin;C:\ProgramData\Docker\Desktop\version-bin;D:\SOFT\gradle-6.0-all\gradle-6.0\bin;C:\Program Files\mingw-w64\x64_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\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\n
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:\dlm_github\shenyu

2021-02-10 02:31:46.061 INFO 2156 --- [main] org.apache.zookeeper.ZooKeeper: Initiating client connection, connectionString=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: 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: 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: 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

5.4. Quick start with Sofa 42
5.4.3 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

5.4. Quick start with Sofa
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;
}

### 5.5 Quick start with gRPC

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.

#### 5.5.1 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:
5.5.2 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:

"appName":"127.0.0.1:8080",
"contextPath":"/grpc",
"path":"/grpc/clientStreamingFun",
"pathDesc":null,
"serviceName":"stream.StreamService",
"methodName":"clientStreamingFun",
"ruleName":null,
"parameterTypes":null,
"rpcExt":null,
"host":"127.0.0.1:8080",
"port":8080,
"registerMetaData":false
}

"appName":"127.0.0.1:8080",
"contextPath":"/grpc",
"path":"/grpc/echo",
"pathDesc":null,
"serviceName":null,
"methodName":null,
"ruleName":null,
"parameterTypes":null,
"rpcExt":null,
"host":"127.0.0.1:8080",
"port":8080,
"registerMetaData":false
}

"appName":"127.0.0.1:8080",
"contextPath":"/grpc",
"path":"/grpc/bidiStreamingFun",
"pathDesc":null,
"serviceName":null,
"methodName":null,
"ruleName":null,
"parameterTypes":null,
"rpcExt":null,
"host":"127.0.0.1:8080",
"port":8080,
"registerMetaData":false
}

"appName":"127.0.0.1:8080",
"contextPath":"/grpc",
"path":"/grpc/unaryFun",
"pathDesc":null,
"serviceName":null,
"methodName":null,
"ruleName":null,
"parameterTypes":null,
"rpcExt":null,
"host":"127.0.0.1:8080",
"port":8080,
"registerMetaData":false
}
5.5.3 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:

![Plugin List]

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

```json
{
    "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 protofile defined by you.

### 5.5.4 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.
The request body like this.

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

Then, call gRPC service by CLIENT_STREAMING method type.
5.5. Quick start with gRPC

The request body like this.

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

Then, call gRPC service by SERVER_STREAMING method type.
The request body like this.

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

Then, call gRPC service by BIDI_STREAMING method type.
5.6 Quick start with Tars

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.

5.6.1 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 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:

```xml
<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>
</dependency>
```
5.6.2 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:

```java
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)
```
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:

TomcatWebServer : Tomcat started on port(s): 55290 (http) with context path ''
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}
5.6.3 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:

```
POST http://localhost:9193/tars/hello 

Body: 
{
  "test": "test"
}
```

Response:
```
{
  "code": 200,
  "message": "access to success!",
  "data": "hello mb-123, name=gott, time=2021-03-07T15:46:06"
}
```
5.7 Quick start with Motan

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.

5.7.1 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 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:

```xml
<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>
```
5.7.2 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:

```
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
RegistryService/1.0/service]
ShenyuClientShutdownHook : hook Thread-0 will sleep 3000ms when it start
ShenyuClientShutdownHook : hook SpringContextShutdownHook will sleep 3000ms when it start
: NettyChannelHandler channelActive: remote=/192.168.1.8:49740 local=/192.168.1.8:8002
: NettyChannelHandler channelActive: remote=/192.168.1.8:49739 local=/192.168.1.8:8002
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":null,"params":null,"right":null,"name":null,"group":null,"motan-shenyu-rpc":null","enabled":true,"host":"
"192.168.220.1","port":8081,registerMetaData":false\}
```

5.7. Quick start with Motan
5.7.3 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:
6.1 Data Synchronization Config

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.

6.1.1 WebSocket Synchronization Config (default strategy, recommend)

- Apache ShenYu gateway config
  
  Add these dependencies in pom.xml:

  ```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>
  </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.

### 6.1.2 Zookeeper Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in pom.xml:

```xml
<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.
6.1.3 HTTP Long Polling Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in pom.xml:

```xml
<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:

```yaml
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:

```yaml
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.

6.1.4 Nacos Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in pom.xml:

```xml
<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 nacos 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 nacos website.

6.1.5 Etcd Synchronization Config

• Apache ShenYu gateway config

  Add these dependencies in pom.xml:

```xml
<!-- apache shenyu data sync start use etcd-->
<dependency>
  <groupId>org.apache.shenyu</groupId>
  <artifactId>shenyu-spring-boot-starter-sync-data-etcd</artifactId>
</dependency>
```
Add these config values in yaml file:

```yaml
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:

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

### 6.1.6 Consul Synchronization Config

- Apache ShenYu gateway config

Add these dependencies in `pom.xml`:

```xml
<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.

6.2 Application Client Access Config

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, Etcld, 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.

6.2.1 Http Registry Config

shenyu-admin config

Set the register type to 'Http in the yaml 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
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.

```yaml
shenyu:
  client:
    registerType: http
    serverLists: http://localhost:9095
    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
```

6.2.2 Zookeeper Registry Config

shenyu-admin config

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

```xml
<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:

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

6.2. Application Client Access Config
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:

```xml
<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:

```yaml
shenyu:
  client:
    registerType: zookeeper
    serverLists: localhost:2181
    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
```

6.2.3 Etcd Registry Config

shenyu-admin config

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

```xml
<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

**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:

```xml
<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:
  client:
    registerType: etcd
    serverLists: http://localhost:2379
    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
6.2.4 Consul Registry Config

shenyu-admin config

First add the related dependencies to the pom file:

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

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-consul-discovery</artifactId>
    <version>2.2.6.RELEASE</version>
</dependency>
```

- In the yml file to configure the registry as consul, you also need to configure spring.cloud.consul, the configuration information is as follows:

```
shenyu:
  register:
    registerType: consul
    props:
      delay: 1
      wait-time: 55

spring:
  cloud:
    consul:
      discovery:
        instance-id: shenyu-admin-1
        service-name: shenyu-admin
        tags-as-metadata: false
        host: localhost
        port: 8500
```

# registerType : register type, set consul.
# 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.
# instance-id: Required, Consul needs to find specific services through instance-id.
# 6.2. Application Client Access Config

## shenyu-client config

Note that the consul registry is currently incompatible with the Spring Cloud service and will conflict with the Eureka/Nacos registry.

The following shows the configuration information registered by Consul 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:

  ```xml
  <dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-consul-discovery</artifactId>
  <version>2.2.6.RELEASE</version>
  </dependency>
  ```

- Then set the register type to consul in yml and config spring.cloud.consul, and related parameters as follows:

  ```yaml
  shenyu:
  client:
    registerType: consul
    props:
      contextPath: /http
      appName: http
      port: 8188
      isFull: false

  spring:
  cloud:
    consul:
      discovery:
        instance-id: shenyu-http-1
        service-name: shenyu-http
        host: localhost
        port: 8500
  # registerType : register type, set consul.
  # 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
# instance-id: Required, Consul needs to find specific services through instance-id.
# service-name: The name where the service is registered to consul. If not configured, the value of `spring.application.name` will be taken by default.
# host: Consul server host, the default value is localhost.
# port: Consul server port, the default value is 8500.

## 6.2.5 Nacos Registry Config

### shenyu-admin config

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

```xml
<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
```

### 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:

```xml
<dependency>
    <groupId>org.apache.shenyu</groupId>
    <artifactId>shenyu-register-client-nacos</artifactId>
    <version>${shenyu.version}</version>
</dependency>
```
• Then in `yml` configure registration mode as `naco`, 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:

```yaml
shenyu:
  client:
    registerType: nacos
    serverLists: localhost:8848
    props:
      contextPath: /http
      appName: http
      port: 8188
      isFull: false
      nacosNameSpace: ShenyuRegisterCenter
```

# 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

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.

### 6.3 Http Proxy

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.
6.3.1 Add divide plugin in gateway

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

```xml
<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>
```

6.3.2 Http request access gateway (for springMvc)

Please refer this: shenyu-examples-http

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

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

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

```xml
<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:

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

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

```java
@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.

```java
@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;
}

• 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.

6.3.3 Http request access gateway (other framework)

• First of all, 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.

6.3.4 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 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 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.
6.4 Dubbo Proxy

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.

6.4.1 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.

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

6.4. Dubbo Proxy
Apache dubbo user, configure the dubbo version and registry center with yours.

```xml
<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>

<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>

<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>
```

- restart gateway service.
6.4.2 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="alibabaDubboServiceBeanPostProcessor" class="org.apache.shenyu.client.alibaba.dubbo.AlibabaDubboServiceBeanPostProcessor">
      <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>
    ```

- Apache Dubbo User
  - SpringBoot
    
    Add these dependencies:
- Spring

Add these dependencies:

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

Inject these properties into your Spring beans XML file:

```xml
@Bean id ="apacheDubboServiceBeanPostProcessor" class ="org.apache.shenyu.
client.apache.dubbo.ApacheDubboServiceBeanPostProcessor">
  <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>
```

### 6.4.3 Dubbo configuration

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

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

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.

```markdown
# 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:

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

- Support for customized multi-parameter type

```java
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
 */

```

6.4. Dubbo Proxy
/*
   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 parameter 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();
}

6.4.4 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();
    }
}
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;
    }
}

- Request param

{  
    "name": ""
}

- Response

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

- Error message

6.4. Dubbo Proxy
6.4.5 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 integration with gateway, one is the added annotation @ShenyuDubboClient, another is a path used to specify 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

```java
@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.

6.5 Spring Cloud Proxy

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.
### 6.5.1 Add springcloud plugin in gateway

- add these dependencies in gateway’s `pom.xml`:

```xml
<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>
```

If you use eureka as SpringCloud registry center.

add these dependencies:

```xml
<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:

```yaml
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:

```xml
<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:

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

- restart your gateway service.

### 6.5.2 SpringCloud service access gateway

Please refer to [shenyu-examples-springcloud](#)

- Add the following dependencies to your Spring Cloud microservice:

```xml
<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.

```java
@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);
```

6.5. Spring Cloud Proxy 83
example (2): /order/save will be handled by gateway, and /order/findById won’t.

```java
@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.

```yaml
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

6.5. Spring Cloud OrderController {

```java
@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;
}

- 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.

6.5.3 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 don't change. Then you can visit, very easy and simple.

6.5. Spring Cloud Proxy
6.6 Sofa Proxy

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.

6.6.1 Add sofa plugin in gateway

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

```xml
<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>
</dependency>
```
• Restart the gateway service.

### 6.6.2 Sofa service access gateway

you can refer to: [shenyu-examples-sofa](#)

• SpringBoot

Add the following dependencies: xml

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

• Spring

Add the following dependencies: xml

```java
<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:

```xml
<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>
```
6.6.3 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.

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

6.6.4 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.

6.6.5 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.

```java
public interface SofaParamResolveService {

    /**
     * Build parameter pair.
     * this is Resolve http body to get sofa param.
     *
```
```java
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.

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

### 6.7 gRPC Proxy

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.

#### 6.7.1 Add gRPC plugin in gateway

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

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

- Restart the gateway service.
6.7.2 gRPC service access gateway

You can refer to: shenyu-examples-grpc.

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

```xml
<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:

```java
@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();
}
```
6.7.3 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:

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

So the request parameters look like this:

```json
{
  "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:

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

The corresponding method call request parameters are as follows:

- UNARY

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

- CLIENT_STREAMING

```json
{
  "data": [
    {
      "text": "hello grpc"
    }
  ]
}
```
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.)

6.8 Tars Proxy
6.8.1 Add tars plugin in gateway

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

```xml
<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>
```

• Restart your gateway service.

6.8.2 Tars service access gateway

Please refer to: shenyu-examples-tars

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

```xml
<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:

```java
@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;
}
}

6.8.3 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.

6.9 Motan Proxy

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.

6.9.1 Add motan plugin in gateway

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

```xml
<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>
```
• Restart your gateway service.

6.9.2 Motan service access gateway

Please refer to: shenyu-examples-motan

• In the microservice built by Motan, add the following dependencies:

```xml
<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:

```java
@MotanService(export = "demoMotan:8002")
public class MotanDemoServiceImpl implements MotanDemoService {
    @Override
    @ShenyuMotanClient(path = "/hello")
    public String hello(String name) {
        return "hello " + name;
    }
}
```
6.9.3 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.
7.1 2.3.0

7.1.1 soul-admin

- Add open field to allow app path authentication or not in sign plugin.
- Optimize divide plugin to use common plugin template in soul-dashboard.
- Add default values and rule checks in plugin handler.
- Add resource management to allow user to add plugin, adjust menu and button resource and so on in soul-dashboard and soul-admin.
- Add menu and data permission in soul-admin.
- Add H2 stroe for soul-admin

7.1.2 soul-bootstrap

- Add tars plugin
- Add sentinel plugin – Add sofa plugin
- Add Resilience4j plugin for soul-plugin.
- Add Context path mapping plugin for soul-plugin.
- Add Grpc plugin supports grpc protocol.
- Support form submission for dubbo plugin.
- feat(plugin handle):
- Add dist package module
- Add test cases for soul-admin
- Add register center for consul
- Add register center for etcd
• Add register center for nacos
• Add register center for zookeeper

7.2 2.2.0

• Complete plug-in architecture design, plug-in hot-swappable.
• Fully supports all versions of dubbo, alibaba-dubbo, apache-dubbo.
• Support dubbo generalization call, multi-parameter, complex parameter interface.
• Enhance the monitoring plug-in, remove influxdb support, increase memory, CPU, QPS, TPS, response delay and other indicators, and support access to Prometheus.
• The springCloud plug-in supports two registration centers, eureka and nacos.
• Waf plug-in enhancements, black and white albums, and mixed modes.
• Removed the Hystrix fuse function, independent as a plug-in support.
• Modify the data synchronization method bug in Zookeeper, and add the nacos synchronization method.
• Diversified customer support, providing traditional and springboot access to spring.
• Optimize the soul-background control interface.
• Load balancing algorithm bug repair.
• Fix bugs when uploading large files.
8.1 Latest Releases

Apache ShenYu is released as source code tarballs with corresponding binary tarballs for convenience.

Apache ShenYu - Version: 2.3.0 (Release Date: Apr 2, 2020)
- Source Codes: zip, tar
- ShenYu-admin Binary Distribution: tar
- ShenYu-bootstrap Binary Distribution: tar

Apache ShenYu Dashboard - Version: 2.3.0 (Release Date: Apr 2, 2020)
- Source Codes: zip, tar
- ShenYu-dashboard Binary Distribution: tar

8.2 PDF

Apache ShenYu provides a packaged and downloaded PDF of the blog for users and developers to use.
- English