Dromara soul document

Dromara

Apr 21, 2021

Contents

1

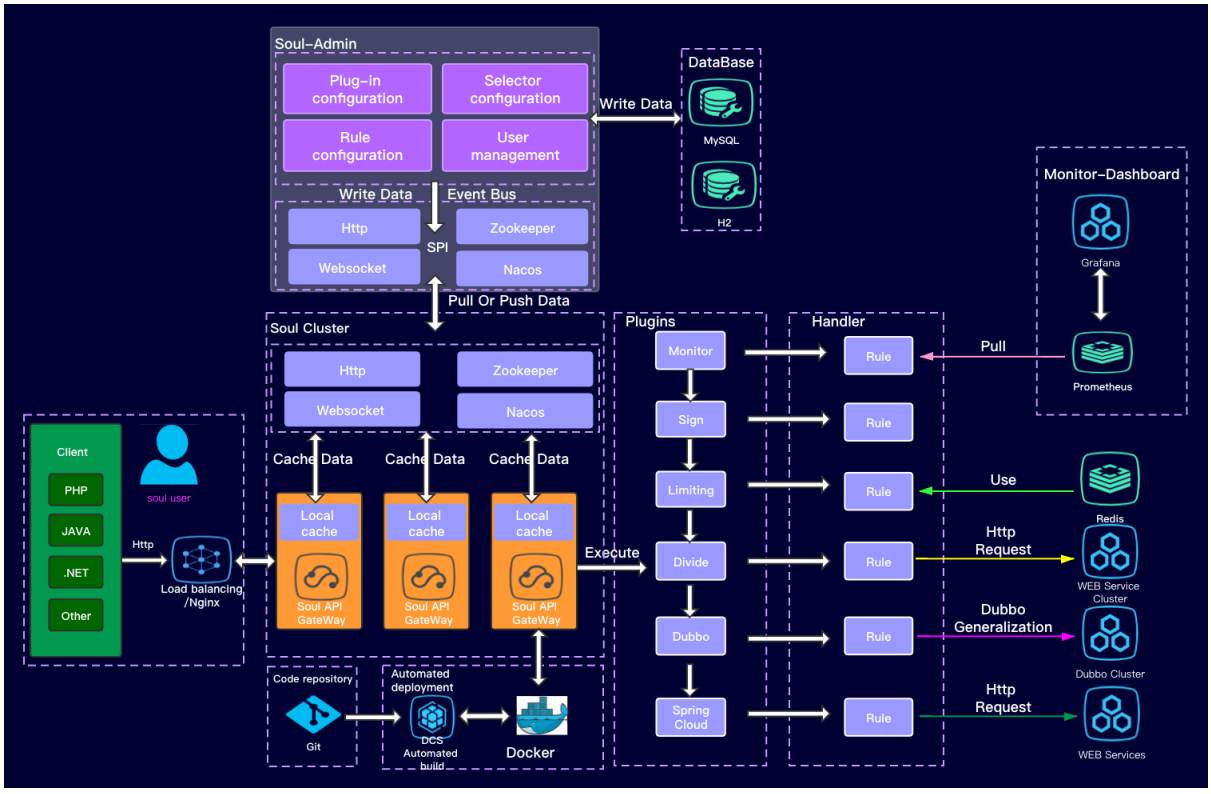
What is the Soul?

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

2 Features

- Support various languages (http protocol), support dubbo, spring cloud protocol.
- 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, gateway consumption is 1~2ms.
- Support cluster deployment, A/B Test, blue-green release.

2.1 Architecture Diagram



3 Team Introduction

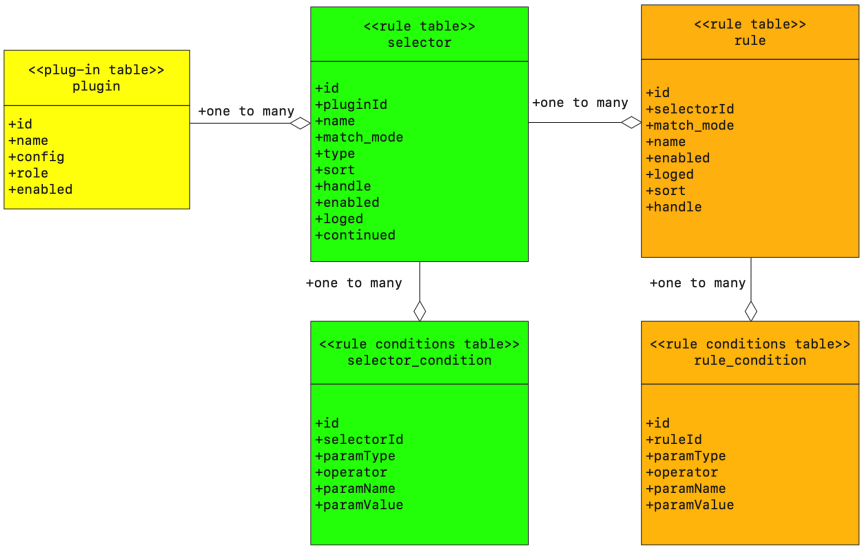
3.1 Team Member（In no particular order）

Name	Github	Role	Company
Xiao Yu	yu199195	VP	jd.com
Zhang Yonglun	tuohai666	PMC	jd.com
Deng Liming	dengliming	PMC	a start-up company
Tang Yudong	tydhot	PMC	perfma
Zhang lei	SaberSola	PMC	helloglobal
Huang Xiaofeng	huangxfchn	committer	shein
Ding jianming	nuo-promise	committer	a start-up company
Feng Zhenbing	fengzhenbing	committer	a start-up company
yangze	HoldDie	committer	IBM

4 Design

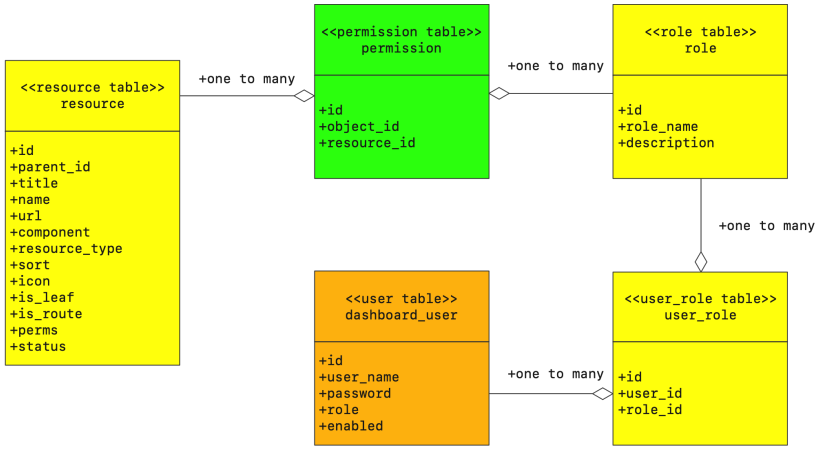
4.1 Database Design

- Plugin use database to store plugin, selector, rule configuration data and relationship.



- 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 admin.

  - Plugin use database to store user name,role,resource data and relationship.



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

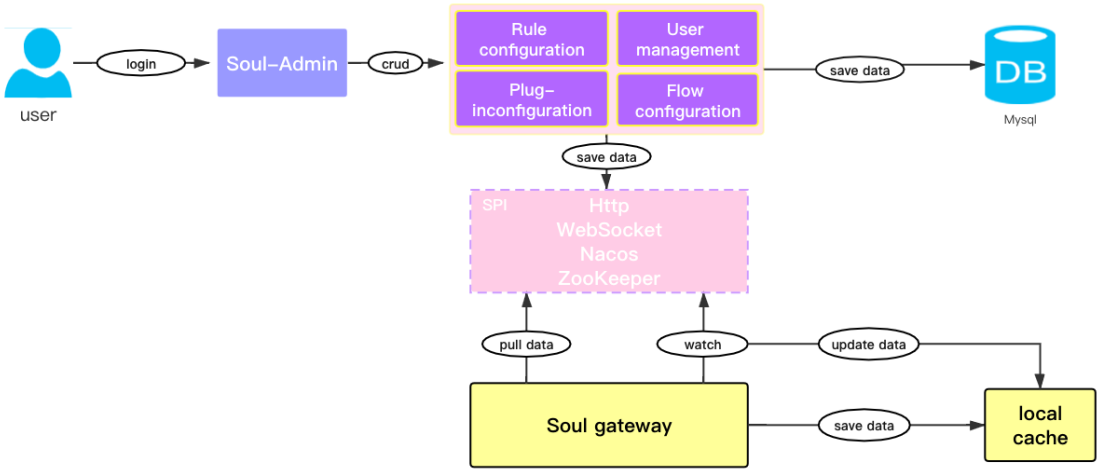
4.2 Configuration Flow Introduction

4.2.1 Description

- This article introduces the flow of synchronizing to the gateway after the data operation of admin backend system.

4.2.2 Usage

- User can arbitrary modify data in soul-admin backend and this will immediately synchronize to the jvm memory of the gateway.
- Synchronize the plugin data of soul,selector,rule data, metadata, signature data, etc.
- All the rules of plugin selectors are dynamically configured and take effect immediately without restarting the service.



- Data Flow Chart:

4.2.3 Feature

- All the configurations of user can be dynamically updated, there is no need to restart the service for any modification.
- Local cache is used to provide efficient performance during high concurrency.

4.3 Data Synchronization Design

4.3.1 Description

This article mainly explains three ways of database synchronization and their principles.

4.3.2 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. Then, how does `Soul` support dynamic configuration?

Anyone who has used `Soul` knows, `Soul` plugin are hot swap,and the selector, rule of all plugins are dynamic configured, they take effect immediately without restarting service.But during using `Soul` gateway, users also report many problems

- Rely on `zookeeper`, this troubles users who use `etcd` `consul` and `nacos` registry
- Rely on `redis`,`influxdb`, I have not used the limiting plugin, monitoring plugin, why do I need these

Therefore,we have done a partial reconstruction of `Soul`,after two months of version iteration,we released version `2.0`

- Data Synchronization removes the strong dependence on zookeeper,and we add http long polling and websocket
- Limiting plugin and monitoring plugin realize real dynamic configuration,we use admin backend for dynamic configuration instead of yml configuration before

Q: Someone may ask me,why don't you use configuration center for synchronization?

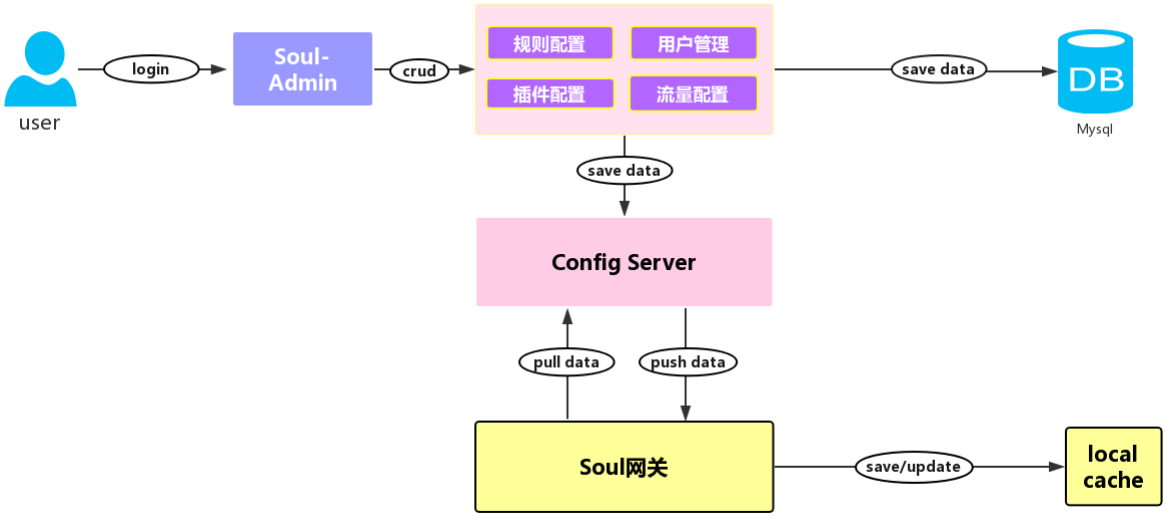
First of all, it will add extra costs, not only for maintenance, but also make Soul heavy; In addition, using configuration center, data format is uncontrollable and it is not convenient for soul-admin to do configuration management.

Q: Someone may also ask,dynamic configuration update?Every time I can get latest data from database or redis,why are you making it complicated?

As a gateway, soul cached all the configuration in the HashMap of JVM in order to provide higher response speed and we use local cache for every request, It's very fast. So this article can also be understood as three ways of memory synchronization in a distributed environment.

4.3.3 Principle Analysis

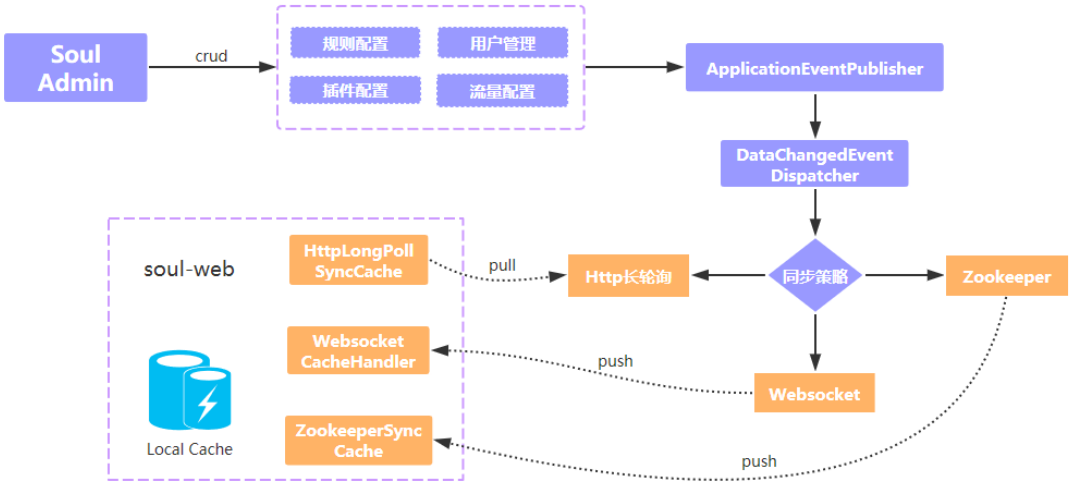
This is a HD uncoded image, it shows the flow of Soul data synchronization, when Soul gateway starts, it will synchronize configuration data from the configuration service and support push-pull mode to obtain configuration change information, and update the local cache.When administrator changes user,rule,plugin,flow configuration in the backend, modified information will synchronize to the Soul gateway through the push-pull mode,whether it is the push mode or the pull mode depends on the configuration.About the configuration synchronization module, it is actually a simplified configuration center.



At version 1.x ,configuration service depends on zookeeper,management backend push the modified information to gateway.But version 2.x supports webosocket,http,zookeeper,it can specify the corresponding synchronization strategy through soul.sync.strategy and use webosocket synchronization strategy by default which can achieve second-level data synchronization.But,note that soul-web and soul-admin must use the same synchronization mechanism.

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

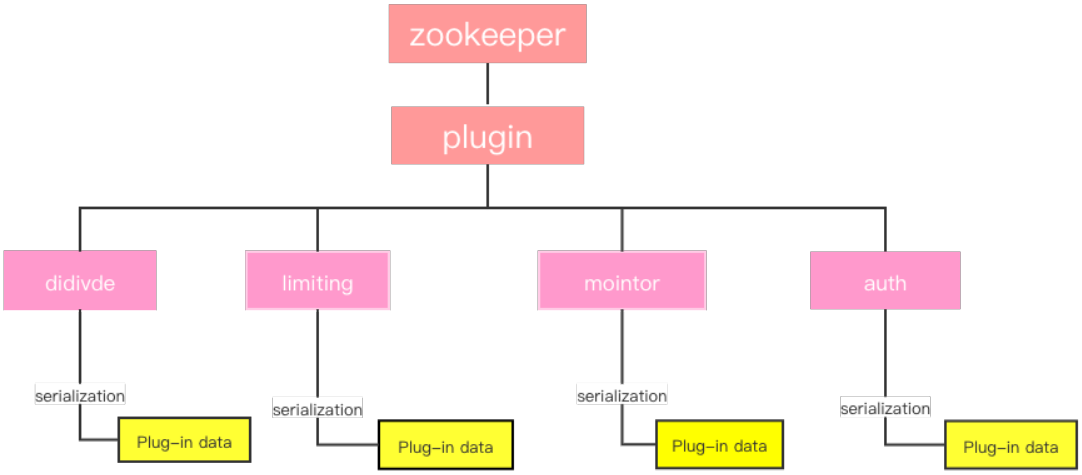
- If it is a `websocket` synchronization strategy,it will push modified data to `soul-web`,and corresponding `WebsocketCacheHandler` handler will handle `admin` data push at the gateway layer
- If it is a `zookeeper` synchronization strategy,it will push modified data to `zookeeper`,and the `ZookeeperSyncCache` will monitor the data changes of `zookeeper` and process them
- If it is a `http` synchronization strategy,`soul-web` proactively initiates long polling requests,90 seconds timeout by default,if there is no modified data in `soul-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

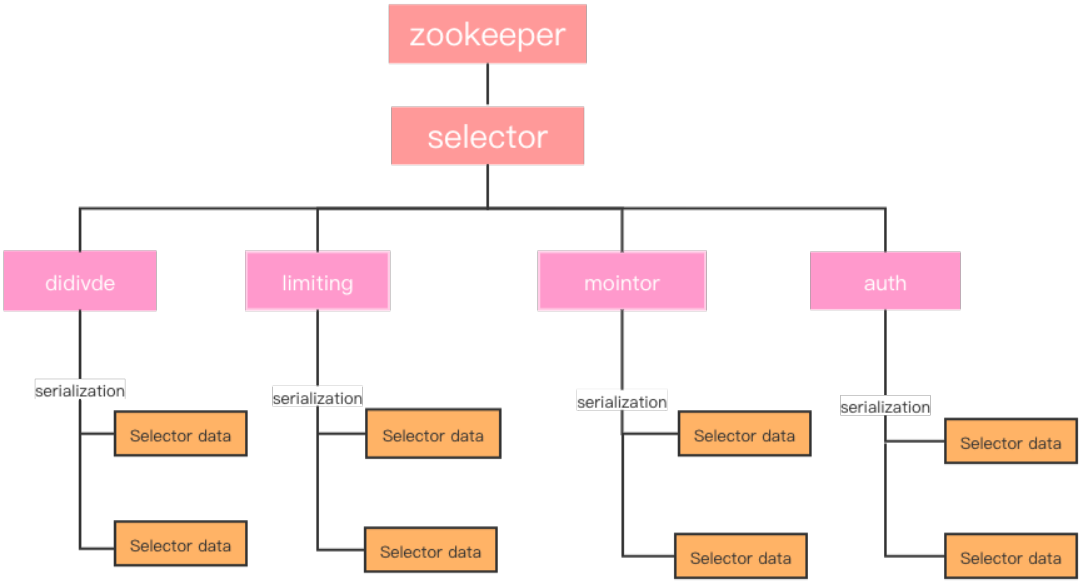


4.3.4 Zookeeper Synchronization

The zookeeper-based synchronization principle is very simple,it mainly depends on `zookeeper` watch mechanism,`soul-web` will monitor the configured node,when `soul-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, `soul-web` will monitor the node for configuration information, and update the local cache once the information changes

`soul` writes the configuration information to the zookeeper node,and it is meticulously designed.





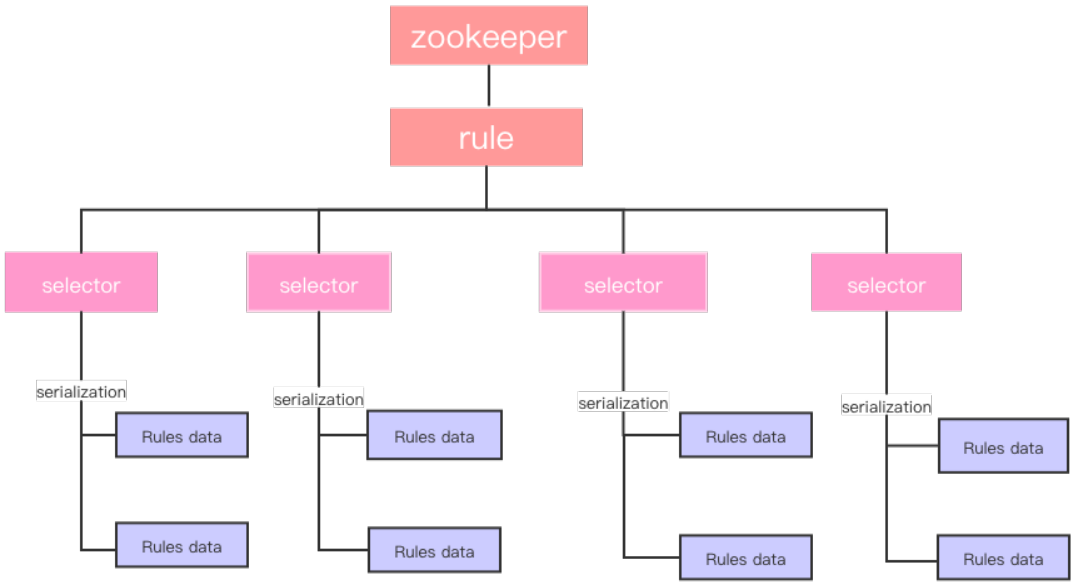


Figure1: Zookeeper Node Design

4.3.5 WebSocket Synchronization

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

When we use websocket synchronization,pay attention to reconnect after disconnection,which also called keep heartbeat.`soul` uses `java-websocket` ,a third-party library,to connect to `websocket`.

```
public class WebsocketSyncCache extends WebsocketCacheHandler {
    /**
     * The Client.
     */
    private WebSocketClient client;

    public WebsocketSyncCache(final SoulConfig.WebsocketConfig websocketConfig) {
        ScheduledThreadPoolExecutor executor = new ScheduledThreadPoolExecutor(1,
                SoulThreadFactory.create("websocket-connect", true));
         client = new WebSocketClient(new URI(websocketConfig.getUrl())) {
                @Override
                public void onOpen(final ServerHandshake serverHandshake) {
                  //....
                }
                @Override
                public void onMessage(final String result) {
                  //....
                }
            };
        //connect
        client.connectBlocking();
        //reconnect after disconnection,using scheduling thread pool,execute every
30 seconds
        executor.scheduleAtFixedRate(() -> {
            if (client != null && client.isClosed()) {
                    client.reconnectBlocking();
            }
        }, 10, 30, TimeUnit.SECONDS);
    }
```

4.3.6 Http Long Polling

The mechanism of zookeeper and websocket data synchronization is relatively simple,but http synchronization will be relatively complicated.Soul 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,soul-web gateway requests admin configuration services,timeout is 90 seconds,it means gateway layer request configuration service will wait at most 90 seconds,this is

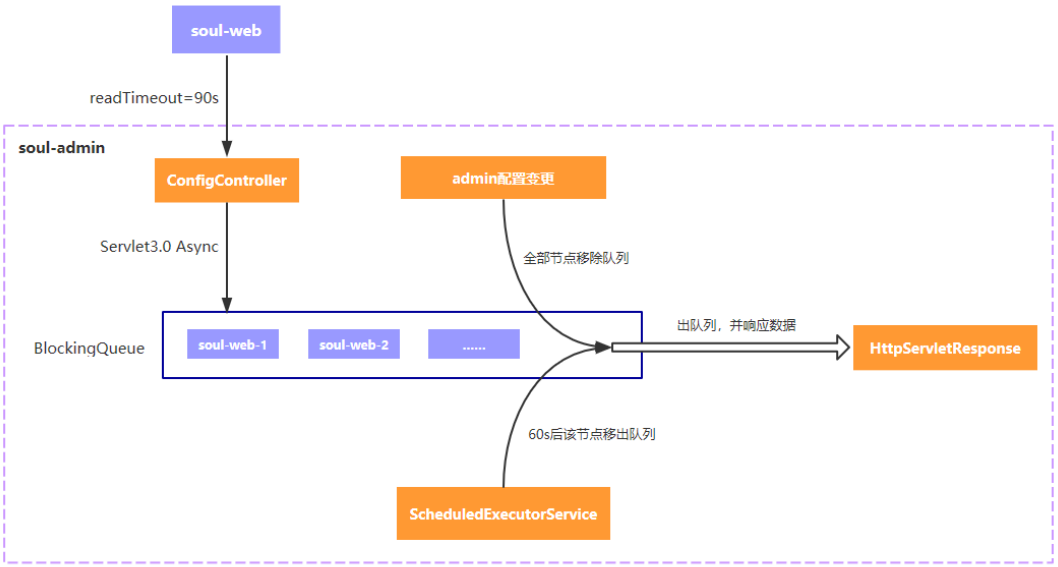


Figure2: http long polling

convenient for admin configuration service to respond modified data in time,and therefore we realize near real-time push.

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

```
public void doLongPolling(final HttpServletRequest request, final
HttpServletResponse response) {
    // since soul-web may not receive notification of a configuration change, MD5
value may be different,so respond immediately
    List<ConfigGroupEnum> changedGroup = compareMD5(request);
    String clientIp = getRemoteIp(request);
    if (CollectionUtils.isNotEmpty(changedGroup)) {
        this.generateResponse(response, changedGroup);
        return;
    }

    // Servlet3.0 asynchronously responds to http request
    final AsyncContext asyncContext = request.startAsync();
    asyncContext.setTimeout(0L);
    scheduler.execute(new LongPollingClient(asyncContext, clientIp, 60));
}

class LongPollingClient implements Runnable {
```

```
    LongPollingClient(final AsyncContext ac, final String ip, final long
timeoutTime) {
        // omit......
    }
    @Override
    public void run() {
        // join a scheduled task, if there is no configuration change within 60
seconds, it will be executed after 60 seconds and respond to http requests
        this.asyncTimeoutFuture = scheduler.schedule(() -> {
            // clients are blocked queue,saved the request from soul-web
            clients.remove(LongPollingClient.this);
            List<ConfigGroupEnum> changedGroups =
HttpLongPollingDataChangedListener.compareMD5((HttpServletRequest) asyncContext.
getRequest());
            sendResponse(changedGroups);
        }, timeoutTime, TimeUnit.MILLISECONDS);
        //
        clients.add(this);
    }
}
```

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.

```
// soul-admin configuration changed,remove the requests from the queue one by one
and respond to them
class DataChangeTask implements Runnable {
    DataChangeTask(final ConfigGroupEnum groupKey) {
        this.groupKey = groupKey;
    }
    @Override
    public void run() {
        try {
            for (Iterator<LongPollingClient> iter = clients.iterator(); iter.
hasNext(); ) {
                LongPollingClient client = iter.next();
                iter.remove();
                client.sendResponse(Collections.singletonList(groupKey));
            }
        } catch (Throwable e) {
            LOGGER.error("data change error.", e);
```

```
        }
    }
}
```

When `soul-web` gateway layer receives the http response information,pull modified information(if exists),and then request `soul-admin` configuration service again,this will repeatedly execute.

4.3.7 Storage Address

github: https://github.com/dromara/soul

gitee: https://gitee.com/dromara/soul

There also have video tutorials on the project homepage,you can go to watch it if needed.

4.3.8 At Last

This article introduces that,in order to optimize the response speed, `soul` as a highly available micro service gateway, its three ways to cache the configuration rule selector data locally.After learning this article,I believe you have a certain understanding of the popular configuration center,it may be easier to learn their codes,I believe you can also write a distributed configuration center.Version 3.0 is already under planning,and I believe it will definitely surprise you.

4.4 MetaData Concept Design

4.4.1 Description

- This article mainly explains the concept,design of metadata and how to connect in the soul gateway.

4.4.2 Technical Solutions

- Add a new table in the database,and data can synchronize to the JVM memory of gateway according to the data synchronization scheme.
- Table Structure:

```
CREATE TABLE  IF NOT EXISTS `meta_data` (
  `id` varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL
COMMENT 'id',
  `app_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL
COMMENT 'application name',
  `path` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL
COMMENT 'path,not repeatable',
  `path_desc` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL COMMENT 'path description',
```

```
  `rpc_type` varchar(64) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL
COMMENT 'rpc type
  `service_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NULL
DEFAULT NULL COMMENT 'service name',
  `method_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NULL
DEFAULT NULL COMMENT 'method name',
  `parameter_types` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NULL DEFAULT NULL COMMENT 'multiple parameter types, split by comma',
  `rpc_ext` varchar(1024) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NULL
DEFAULT NULL COMMENT 'rpc extension information,json format',
  `date_created` datetime(0) NOT NULL COMMENT 'create date',
  `date_updated` datetime(0) NOT NULL ON UPDATE CURRENT_TIMESTAMP(0) COMMENT
'update date',
  `enabled` tinyint(4) NOT NULL DEFAULT 0 COMMENT 'enable status',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci ROW_FORMAT =
Dynamic;
```

- Metadata design as below,the most important is using it in dubbo’ s generalization call.
- Pay attention to the field `path`,we will match specific data according to your field path during requesting gateway,and then carry out the follow-up process.
- Pay attention to the field `rpc_ext`,if it is a dubbo service interface and service interface has group and version field,this field exists.
- dubbo field structure as below,then we store json format string.

```
public static class RpcExt {
    private String group;
    private String version;
    private String loadbalance;
    private Integer retries;
    private Integer timeout;
}
```

4.4.3 MetaData Storage

- A dubbo interface corresponds to a meta data.
- SpringCloud protocol, only store one record, path: `/contextPath/**`.
- Http service, no data.

5 Admin

5.1 Dict Management

5.1.1 Explanation

- Dictionary management is primarily used to maintain and manage common data dictionaries.

Table design

- sql

```
CREATE TABLE IF NOT EXISTS `soul_dict` (
   `id` varchar(128) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'primary key id',
   `type` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'type',
   `dict_code` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'dictionary
encoding',
   `dict_name` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'dictionary
name',
   `dict_value` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT
'dictionary value',
   `desc` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT 'dictionary
description or remarks',
   `sort` int(4) NOT NULL COMMENT 'sort',
   `enabled` tinyint(4) DEFAULT NULL COMMENT 'whether it is enabled',
   `date_created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'create time
',
   `date_updated` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
TIMESTAMP COMMENT 'update time',
   PRIMARY KEY (`id`)
 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- The current usage scenario is when the plugin `handle` configuring the `data_type=3` (`select box`)

eg. `degradeRuleGrade` is one of fields of sentinel’s `handle` json

When it adds rules, it automatically looks up all the general dictionaries of `type='degradeRuleGrade'` in the `soul_dict` table as a select-box when you edit the General rules field

5.2 Plugin Handle Explanation

5.2.1 Explanation

- In our Soul-Admin background, each plugin uses the Handle field to represent a different processing, and plugin processing is used to manage and edit custom processing fields in JSON.
- This feature is mainly used to support the plug-in handling template configuration

5.2.2 Table Design

- sql

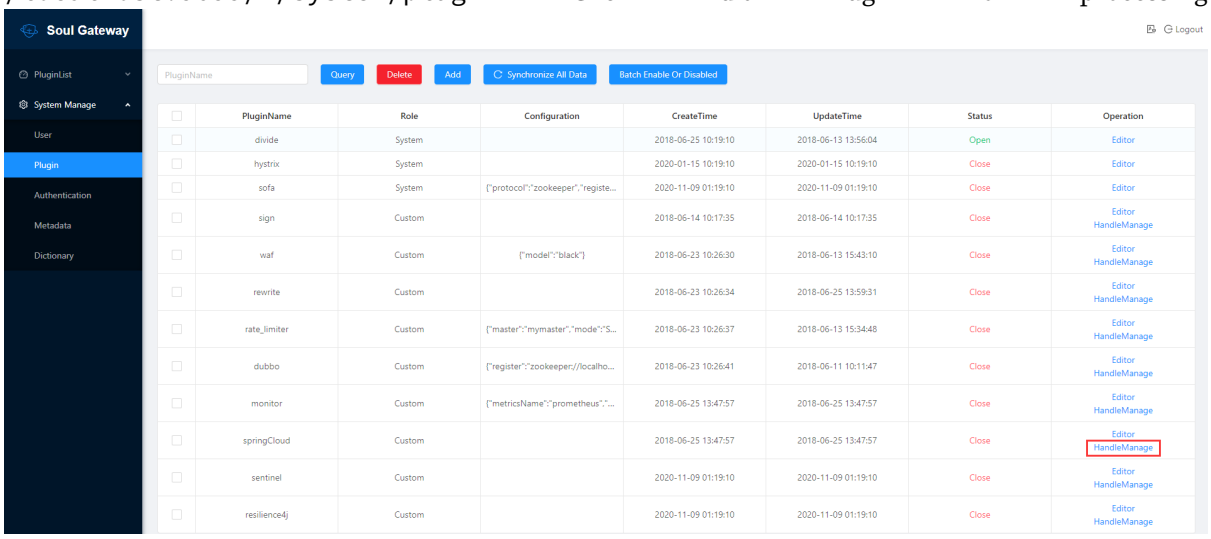
```
CREATE TABLE IF NOT EXISTS `plugin_handle` (
  `id` varchar(128) NOT NULL,
  `plugin_id` varchar(128) NOT NULL COMMENT 'plugin id',
  `field` varchar(100) NOT NULL COMMENT 'field',
  `label` varchar(100) DEFAULT NULL COMMENT 'label',
  `data_type` smallint(6) NOT NULL DEFAULT '1' COMMENT 'data type 1 number 2 string
3 select box',
  `type` smallint(6) NULL COMMENT 'type, 1 means selector, 2 means rule',
  `sort` int(4)  NULL COMMENT 'sort',
  `ext_obj` varchar(1024) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT 'extra
configuration (json format data)',
  `date_created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'create time
',
  `date_updated` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
TIMESTAMP COMMENT 'update time',
  PRIMARY KEY (`id`),
  UNIQUE KEY `plugin_id_field_type` (`plugin_id`,`field`,`type`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE = utf8mb4_unicode_ci ROW_FORMAT =
Dynamic;
```

5.2.3 Tutorial

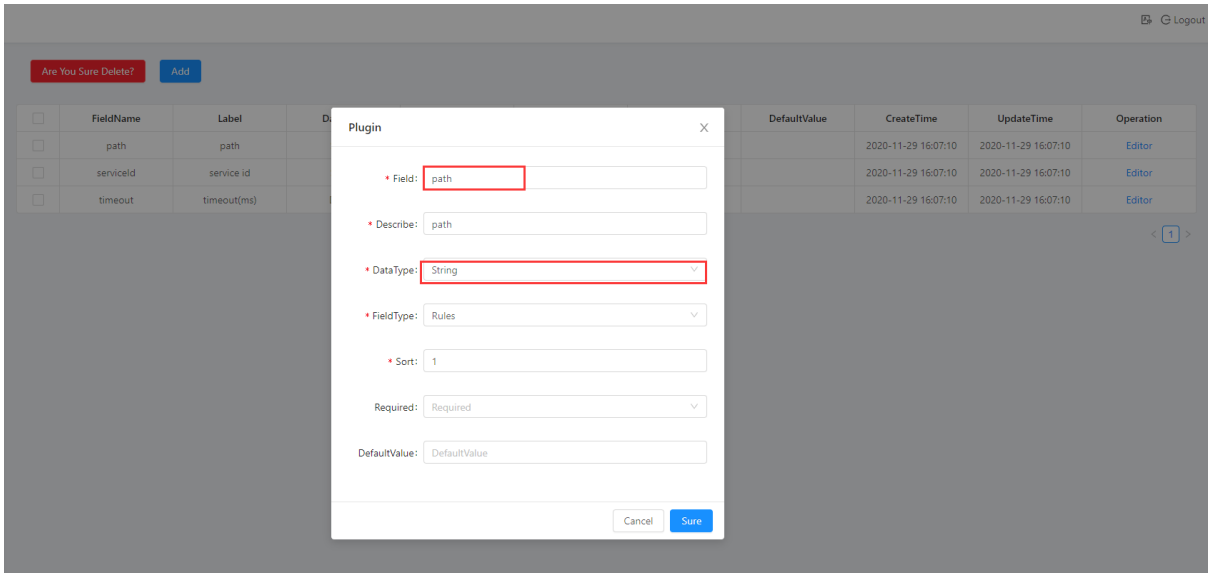
eg. When we developed the `springCloud` plugin, the rule table needed to store some configuration into the handle field, Configure the corresponding entity class as follows:

```
public class SpringCloudRuleHandle implements RuleHandle {

    /**
     * this remote uri path.
     */
    private String path;

    /**
     * timeout is required.
     */
    private long timeout = Constants.TIME_OUT;
}
```

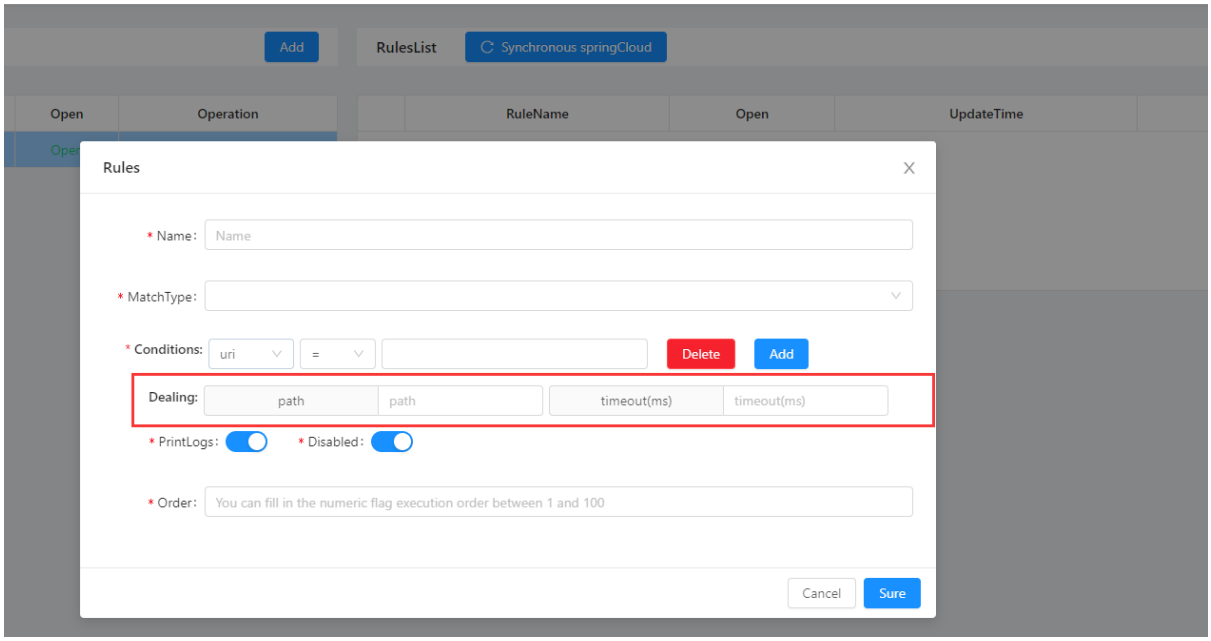
step1. We can go directly to the plug-in management link `http://localhost:9095/#/system/plugin` Click Edit Plugin for processing



step2. Add a string type field path and a numeric type TIMEOUT

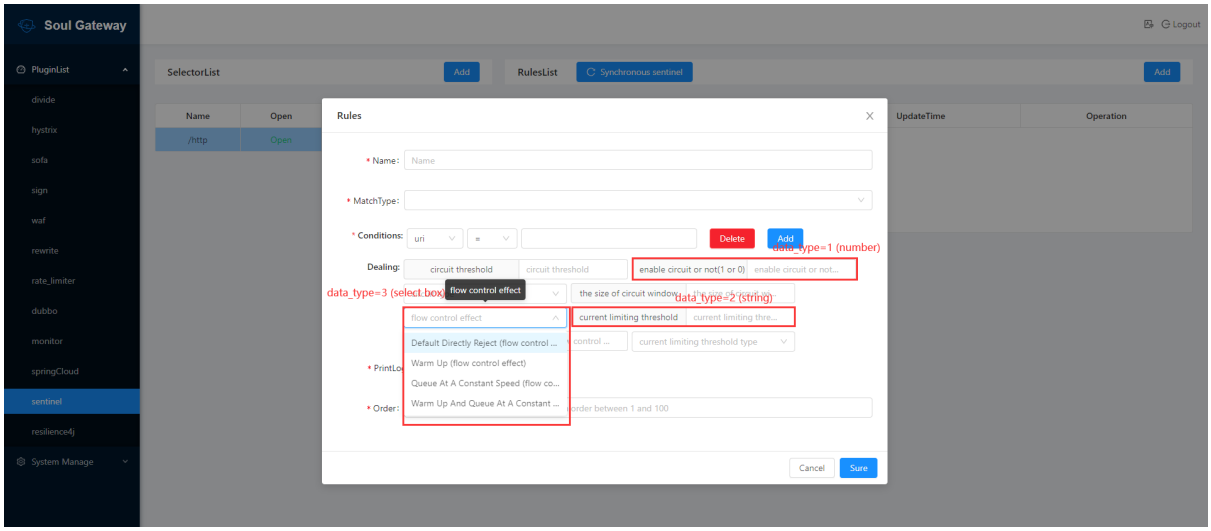


step3. Finally, you can enter path, TIMEOUT and commit to the handle field when you add a rule in the plugin rule configuration page



*Note: If data_type is configured to be ``3`` ``selection box``, the input field drop-down selection on the new rule page is displayed by going to the*soul_dict） *table to find all the options available*

- The Sentinel plug-in, for example, is shown below:



5.3 Selector Detailed Explanation

5.3.1 Features

- Selector and rule are the key point of soul gateway, you can manage any request with it.
- This chapter is mainly focus on the concepts of selector and rule and how to use it.

5.3.2 Overview

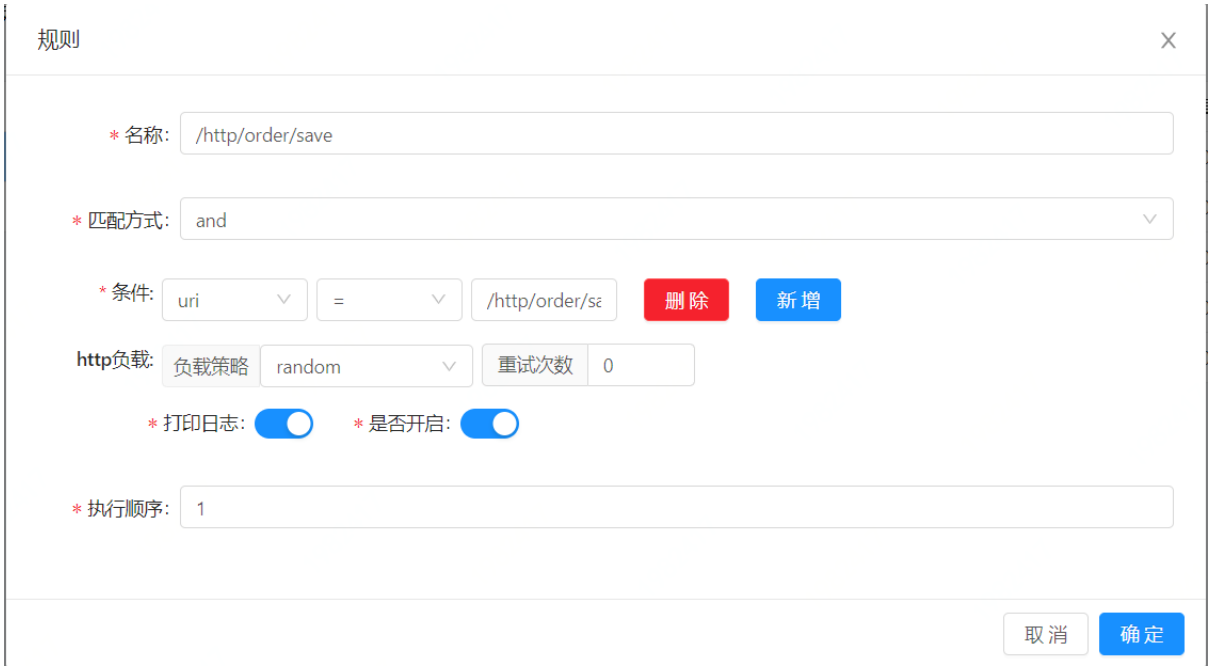
- One plugin has many selector and a selector has many rules, selector is the first filter of request, and the rule is the final filter.
- Please consider this, it would be perfect when the plugin executes the request until it reached the config value.
- Selector and rule are designed to execute only when the request meet the specific condition.
- Please refer to the previous data structure database design.

5.3.3 Selector

- selector detailed explanation:
  - name: create your selector with a distinguish name.
  - type: custom flow is customized request, full flow is full request. customized request will be handled by the conditions as below, while full request won't.
  - match method: you can combine these conditions with 'and', 'or' operators.
  - condition:
    - uri: filter request with uri method and support fuzzy matching (/**).
    - header: filter request with request header.
    - query: filter request with query string.
    - ip: filter request with your real ip.
    - host: filter request with your real host.
    - post: not recommend to use.
    - condition match:

        - match：fuzzy string matching，recommend to combine with uri，support restful matching.（/test/**).
        - =：if the values are the same, then they match.
        - regEx：regex matching，match characters in regex expression.
        - like：string fuzzy matching.
    - open option：only work with enabled.
    - print log：it will print the matching log with the open option enabled.
    - execution order：the smaller will have high priorty to execute among multi-selectors.
- the above picture means: when the prefix of the request uri is `/test` and the value of `module` in`header` is`test`, it will redirect to this service `1.1.1.1:8080`.
- selector advice：combine `uri` conditon and `match` prefix（/contextPath）as the first request filter.

5.3.4 Rule



- when the request was passed by the seletor, then it will be processed by the rule, the final filter.
- rule is the final confirmation about how to execute request logically.
- rule detailed explanation：
    - name：create your rule with a distinguish name.
    - match method: you can combine these conditions with ‘and’，‘or’ operators.
    - condition：
        * uri: filter request with uri method and support fuzzy matching (/**).
        * header: filter request with request header.

    * query: filter request with query string.
    * ip: filter request with your real ip.
    * host: filter request with your real host.
    * post: not recommend to use.
    * condition match:
      · match：fuzzy string matching，recommend to combine with uri，support restful matching.（/test/**）
      · =：if the values are the same, then they match.
      · regEx：regex matching，match characters in regex expression.
      · like：string fuzzy matching.
  - open option：only work with enabled.
  - print log：it will print the matching log with the open option enabled.
  - execution order：the smaller will have high priorty to execute among multi-rules.
  - handle: different plugin has different execution method, pls refer to the specific one.
- above picture means: when the request `uri` equals to `/http/order/save`, it will execute based on this rule，load strategy is `random`.
- combine selector means：when the request `uri` is `/http/order/save`, it will be redicted to `1.1.1.1:8080` by `random` method.
- rule advice: combine `uri` condition with `match` the real `uri path` condition as the final filter.

5.3.5 Condition Explanation

- uri matching （recommend）
  - uri matching is based on your request uri, the frontend won't change anything before accessing the gateway.
  - the `match` filter method is the same with`springmvc` fuzzy matching.
  - in selector，we recommend to match with the prefix of uri, and use the specific path in rule.
  - when changing the match method, the matching field name can be filled randomly, but make sure the match value must be correct.
- header matching
  - header matches with your `http` request header value.
- query matching
  - it matches the query string in your uri, such as: /test?a=1&&b=2.
  - so you can add a new condition, choose query method: a = 1.
- ip matching

  - it matches the ip of the http caller.
  - especially in the waf plugin, if you find some ip is unsafe, you can add a match condition with this ip, then it can’ t access any more.
  - if you use nginx proxy before soul, you can get the right ip with refering to parsing-ip-and-host

- host matching
  - it matches the host of http caller.
  - especially in waf plugin, if you find some host is unsafe, you can add a match condition with this host, then it can’ t access any more.
  - if you use nginx proxy before soul, you can get the right ip with refering to parsing-ip-and-host

- post matching
  - not recommend to use.

6 Users Guide

6.1 Environment Setup

6.1.1 Features

- Soul is an open source plugin framework, which is flexibility and extensibility since 2.2.0 version. With soul you can easily create application with your own gateway.
- System Requirement: JDK 1.8+, Mysql 5.5.20 +.

6.1.2 Soul-Admin

remote download

- 2.3.0 download `soul-admin-bin-2.3.0-RELEASE.tar.gz`
- tar `soul-admin-bin-2.3.0-RELEASE.tar.gz`. cd `/bin`.
- use h2 store.

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

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

- use `mysql` store. cd `/conf` and then modify `mysql` config in `application.yaml`.

```
> windwos : start.bat

> linux : ./start.sh
```

docker

```
> docker pull dromara/soul-admin
> docker network create soul
```

- use h2 store

```
> docker run -d -p 9095:9095 --net soul dromara/soul-admin
```

- use `mysql` store.

```
> docker run -e "SPRING_PROFILES_ACTIVE=mysql" -d -p 9095:9095 --net soul dromara/
soul-admin
```

If you want to override environment variables, you can do like this.

```
docker run -e "SPRING_PROFILES_ACTIVE=mysql" -e "spring.datasource.url=jdbc:mysql:/
/192.168.1.9:3306/soul?useUnicode=true&characterEncoding=utf-8&useSSL=false" -e
"spring.datasource.password=123456" -d -p 9095:9095 --net soul dromara/soul-admin
```

Another way, bind volume and mounts

Put your `application.yml` in xxx directory, then run like this.

```
docker run -v D:\tmp\conf:/opt/soul-admin/conf/ -d -p 9095:9095 --net soul dromara/
soul-admin
```

local

- download

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

- setup for `SoulAdminBootstrap`.
  - if use h2 store please set env `--spring.profiles.active = h2`
  - if use mysql store, please modify `mysql` config in `application.yaml`.

Visit `http://localhost:9095/index.html` default username： admin password: 123456.

6.1.3 Soul-Bootstrap

remote download

- 2.3.0 download `soul-bootstrap-bin-2.3.0-RELEASE.tar.gz`
- tar `soul-bootstrap-bin-2.3.0-RELEASE.tar.gz`, ant then cd `/bin`.

```
> windwos : start.bat

> linux : ./start.sh
```

docker

```
> docker network create soul
> docker pull dromara/soul-bootstrap
> docker run -d -p 9195:9195 --net soul dromara/soul-bootstrap
```

local

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

- setup for `SoulBootstrap`.

6.1.4 Build your own gateway（recommend）

- First of all, create a new Spring Boot project. You can refer to the way how you start the soul-bootstrap, or visit Spring Initializer
- Add these JAR into your Maven pom.xml:

```
<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>
```

```
<!--soul gateway start-->
<dependency>
      <groupId>org.dromara</groupId>
      <artifactId>soul-spring-boot-starter-gateway</artifactId>
      <version>${last.version}</version>
</dependency>

 <!--soul data sync start use websocket-->
 <dependency>
      <groupId>org.dromara</groupId>
      <artifactId>soul-spring-boot-starter-sync-data-websocket</artifactId>
      <version>${last.version}</version>
 </dependency>
```

- Add these config values into your `application.yaml`:

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

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

- Environment Setup has finished, now your project is launched.

6.2 Integrate Http with soul gateway

6.2.1 Features

- This chapter is a guide about integrating Http service with soul gateway.
- Soul gateway uses divide plugin handling http request, pls enable it in soul-admin background.
- Please start `soul-admin` successfully before integrating and Environment Setup is Ok.

6.2.2 Configure soul gateway as Http proxy.

- Add these dependencies in gateway’ s pom.xml:

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

 <dependency>
     <groupId>org.dromara</groupId>
     <artifactId>soul-spring-boot-starter-plugin-httpclient</artifactId>
     <version>${last.version}</version>
 </dependency>
```

- pls restart the gateway.

6.2.3 Http request via soul gateway（springMVC user）

- pls make sure divide plugin has enabled in soul-admin background.

add Soul-Client methods（available for SpringMVC,SpringBoot user）

- SpringBoot User
  - Add these dependencies in your local maven repository pom.xml:

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-client-springmvc</artifactId>
    <version>${last.version}</version>
</dependency>
```

  - Backend server register center config, please look:register center access.

- SpringMVC User
  - Add these dependencies in your local maven repository pom.xml:

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-client-springmvc</artifactId>
    <version>${last.version}</version>
</dependency>
```

  - Inject these properties into your Spring beans XML file:

```
<bean id ="springMvcClientBeanPostProcessor" class ="org.dromara.soul.
client.springmvc.init.SpringMvcClientBeanPostProcessor">
     <constructor-arg  ref="soulRegisterCenterConfig"/>
</bean>

<bean id="soulRegisterCenterConfig" class="org.dromara.soul.register.
common.config.SoulRegisterCenterConfig;">
     <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 server name"/>
            <entry key="port" value="your server port"/>
            <entry key="isFull" value="false"/>
          </map>
     </property>
</bean>
```

- Add this annotation `@SoulSpringMvcClient` 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.
  - Example1: both `/test/payment` and `/test/findByUserId` will be handled by proxy service.

```
@RestController
@RequestMapping("/test")
@SoulSpringMvcClient(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;
    }
}
```

  - Example2: `/order/save` will be handled by proxy service, but `/order/findById` won’t.

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

    @PostMapping("/save")
    @SoulSpringMvcClient(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;
    }
}
```

- Kick off your project with your interface, which is integrated with soul gateway.

6.2.4 Configure soul gateway as an Http proxy（other framework）

- first of all, enable the divide plugin in `soul-admin`, then add selector and rule which will filter the request.
- if you don’ t know how to configure, pls refer to selector guide.
- you can also develop your cutomized http-client, refer to multi-language Http client development。

6.2.5 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, soul 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 `soul-admin`, if your 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.

# Any questions, pls join the group and we can talk about it.
```

- Then you can visit, very easy and simple.

6.3 Integrate dubbo with soul gateway

6.3.1 Features

- This chapter is a guide about integrating dubbo service with soul gateway.
- Support Alibaba Dubbo(< 2.7.x) and Apache Dubbo (>=2.7.x).
- Please start `soul-admin` successfully before integrating, and Environment Setup is Ok.

6.3.2 Configure soul gateway as Dubbo proxy

- Add these dependencies in gateway's `pom.xml`.
- Alibaba dubbo user, configure the dubbo version and registry center with yours.

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

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

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

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

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

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

```
   <version>4.0.1</version>
</dependency>
<!-- Dubbo zookeeper registry dependency end -->
```

- restart gateway service.

  Dubbo integration with gateway,pls refer to : soul-examples-dubbo

- Alibaba Dubbo User
  - SpringBoot
    - Add these dependencies:

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

    - backend server register center config, please look:register center access.
  - Spring
    - Add these dependencies:

```
 <dependency>
   <groupId>org.dromara</groupId>
   <artifactId>soul-client-alibaba-dubbo</artifactId>
   <version>${last.version}</version>
</dependency>
```

    - Inject these properties into your Sping beans XML file:

```
<bean id ="alibabaDubboServiceBeanPostProcessor" class ="org.dromara.soul.
client.alibaba.dubbo.AlibabaDubboServiceBeanPostProcessor">
     <constructor-arg  ref="soulRegisterCenterConfig"/>
</bean>

<bean id="soulRegisterCenterConfig" class="org.dromara.soul.register.
common.config.SoulRegisterCenterConfig">
   <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:

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

    * backend server register center config, please look:register center_access:

  - Spring
    * Add these dependencies:

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

    * Injecct these properties into your Spring beans XML file:

```
<bean id ="apacheDubboServiceBeanPostProcessor" class ="org.dromara.soul.
client.apache.dubbo.ApacheDubboServiceBeanPostProcessor">
   <constructor-arg  ref="soulRegisterCenterConfig"/>
</bean>

<bean id="soulRegisterCenterConfig" class="org.dromara.soul.register.
common.config.SoulRegisterCenterConfig">
   <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>
```

6.3.3 Dubbo configuration

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

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

Configure the interface with gateway

- you can add the annotation `@SoulDubboClient` to your dubbo service implementation class, so that the interface method will be configured with gateway.
- start your provider and get the log `dubbo client register success`, then your dubbo interface has been added with soul gateway successfully.Pls refer to `soul-test-dubbo` project.

Dubbo user request and parameter explanation.

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

```
# for example: you have an order service and it has a interface, his 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, pls refer to the interface definition in soul-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:

```
soul:
  dubbo:
    parameter: multi
```

- Support for customized multi-parameter type
- Create a new implementation class A in your gateway project of `org.dromara.soul.web.dubbo.DubboParamResolveService`.

```
public interface DubboParamResolveService {

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

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

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

6.3.4 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 @SoulDubboClient.
  - Update the configuration in Admin.
  - It’s valid for all request.
- Param valid and SoulException
  - Set validation="soulValidation".
  - When SoulException is thrown in the interface, exception information will be returned. It should be noted that SoulException is thrown explicitly.

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

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

- Request param

```
public class HelloServiceRequest implements Serializable {

    private static final long serialVersionUID = -5968745817846710197L;

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

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

    public String getName() {
        return name;
    }

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

    public Integer getAge() {
        return age;
    }

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

- Send request

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

- Response

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

- Error message

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

Let's break down this process: http –> gateway –> dubbo provider

- It basically switches from HTTP request to Dubbo protocol, then invoke Dubbo service and return to the result.
- Two things need to notice after intgeration with gateway, one is the added annoation `@SoulDubboClient`, another is a path used to speicify the request path.
- And you added a config value of `contextPath`.
- If you still remember, then we can start.
- If you have a function like this, the config value in contextPath is `/dubbo`

```
@Override
@SoulDubboClient(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..

How about the request parameter? `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 your interface has multi-parameter, pls refer to the guide above.

6.4 SpringCloud Proxy

6.4.1 Features

- This article is a guide about how to integrate Spring Cloud with soul gateway quickly.
- Please enable `springCloud` plug-in in soul-admin background.
- Please start `soul-admin` successfully before integrating and Environment Setup is Ok.

6.4.2 Configure soul gateway as Spring Cloud proxy

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

```
<!--soul springCloud plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-springcloud</artifactId>
    <version>${last.version}</version>
</dependency>
 <!--soul springCloud plugin end-->

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

- If you use `eureka` as SpringCloud registry center.
  - add these dependencies:

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

- add these config values in gateway' s yaml file:

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/ #your eureka address
```

```
  instance:
    prefer-ip-address: true
```

- if you use nacos as Spring Cloud registry center.
  - add these dependencies:

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

- add these config values in gateway's yaml file:

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

- restart your gateway service.

6.4.3 SpringCloud integration with gateway

- add these dependencies in your project:

```
<dependency>
     <groupId>org.dromara</groupId>
     <artifactId>soul-spring-boot-starter-client-springcloud</artifactId>
     <version>${last.version}</version>
</dependency>
```

- backend server register center config, please look:register center access.
- add the annotation `@SoulSpringCloudClient` in your `controller` interface.
- you can apply the annotation to class-level in a controller.the name of the path variable is prefix and '/**' will apply proxy for entire interfaces.
  - example (1):both `/test/payment` and `/test/findByUserId` will be handled by gateway.

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

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

```
    }

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

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

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

    @PostMapping("/save")
    @SoulSpringMvcClient(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 service, get the log `dubbo client register success`, then your interface has been added with soul gateway successfully.

6.4.4 Plugin Setting

- enable Spring Cloud plugin in `soul-admin`.

6.4.5 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, soul 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 `soul-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.

# Any questions, pls join the group and we can talk about it.
```

- Then you can visit, very easy and simple.

6.5 Sofa RPC Proxy

6.5.1 Description

- This article is about sofa users using sofa plug-in support,and the tutorial of connecting your own sofa service to the soul gateway.
- Before connecting, please start `soul-admin` correctly and Setup Environment Ok。

6.5.2 Introduce the plug-in that the gateway supports for sofa

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

```
<dependency>
   <groupId>com.alipay.sofa</groupId>
   <artifactId>sofa-rpc-all</artifactId>
   <version>5.7.6</version>
</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.dromara</groupId>
   <artifactId>soul-spring-boot-starter-plugin-sofa</artifactId>
   <version>${last.version}</version>
</dependency>
```

- Restart the gateway service.

6.5.3 sofa service access gateway, you can refer to： soul-examples-sofa

- Springboot
  - Introduce the following dependencies :

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

  - backend server register center config, please look:register center access.
- Spring

- Introduce the following dependencies:

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-client-sofa</artifactId>
    <version>${project.version}</version>
</dependency>
```

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

```
    <bean id ="sofaServiceBeanPostProcessor" class ="org.dromara.soul.client.
sofa.SofaServiceBeanPostProcessor">
         <constructor-arg  ref="soulRegisterCenterConfig"/>
    </bean>

 <bean id="soulRegisterCenterConfig" class="org.dromara.soul.register.common.
config.SoulRegisterCenterConfig">
   <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>
```

6.5.4 Plugin Settings

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

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

6.5.5 Interface registered to the gateway

- For your sofa service implementation class, add @SoulSofaClient annotation to the method， Indicates that the interface method is registered to the gateway.
- Start your provider and output the log `sofa client register success`. You’re done. Your sofa interface has been published to the soul gateway. If you still don’t understand, you can refer to the `soul-test-sofa` project.

6.5.6 sofa user request and parameter description

- To put it bluntly, it is to request your sofa service through http
- Soul 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 soul-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 A, implements org.dromara.soul.plugin.api.sofa.SofaParamResolveService。

```
public interface SofaParamResolveService {

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

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

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

6.6 Use Different Data-Sync Strategy

6.6.1 Features

- Data synchronization is the key of gateway high performance, which is to sync 'soul-admin' config data into the JVM memory of soul cluster.
- Implementation principles, pls refer to: dataSync。
- In the article, the gateway is the environment you setup. please refer to: Environment Setup.

6.6.2 Websocket sync（default method， recommend）

- gateway setting（note:restart）
  - Add these dependencies in `pom.xml`:

```
<!--soul data sync start use websocket-->
<dependency>
  <groupId>org.dromara</groupId>
  <artifactId>soul-spring-boot-starter-sync-data-websocket</artifactId>
  <version>${last.version}</version>
</dependency>
```

  - add these config values in springboot yaml file:

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

  - soul-admin config, enable this parameter `--soul.sync.websocket=''` in soul admin, then restart service.

```
soul:
  sync:
     websocket:
```

- When the connection is established, getting the full data once,then adding and upating data subsequently, which is a good performance.
- Support disconnection and reconnection (default 30 sec).

6.6.3 Zookeeper Sync

- gateway setting（note: restart）
  - Add these dependencies in pom.xml:

```
<!--soul data sync start use zookeeper-->
  <dependency>
       <groupId>org.dromara</groupId>
        <artifactId>soul-spring-boot-starter-sync-data-zookeeper</artifactId>
        <version>${last.version}</version>
  </dependency>
```

  - Add these dependencies in springboot yaml file:

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

  - soul-admin config: configure the soul-admin's starting parameter with `--soul.sync.zookeeper.url='your address'`,then restart the service.

```
soul:
  sync:
    zookeeper:
        url: localhost:2181
        sessionTimeout: 5000
        connectionTimeout: 2000
```

  - It is good 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.

6.6.4 Http long-polling sync

- gateway setting（note:restart）
  - Add these dependencies in pom.xml:

```
<!--soul data sync start use http-->
  <dependency>
       <groupId>org.dromara</groupId>
        <artifactId>soul-spring-boot-starter-sync-data-http</artifactId>
        <version>${last.version}</version>
  </dependency>
```

  - add these config values in your springboot yaml file:

```
soul :
    sync:
        http:
             url: http://localhost:9095
#url: config with your soul-admin's ip and port url, pls use (,) to split
multi-admin cluster environment.
```

  - soul-admin config, configure the soul-admin's starting parameter with `--soul.sync.http=''`, then restart service.

```
soul:
  sync:
     http:
```

- 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.
- it may hit bug in soul-admin cluster.

6.6.5 Nacos sync

- gateway setting（note:restart）
  - Add these dependencies in your `pom.xml`:

```
<!--soul data sync start use nacos-->
  <dependency>
       <groupId>org.dromara</groupId>
        <artifactId>soul-spring-boot-starter-sync-data-nacos</artifactId>
        <version>${last.version}</version>
  </dependency>
```

  - add these config values in the springboot yaml file:

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

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

* soul-admin config: passing values one by one with '–' operator in the soul-Admin startup parameter.

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

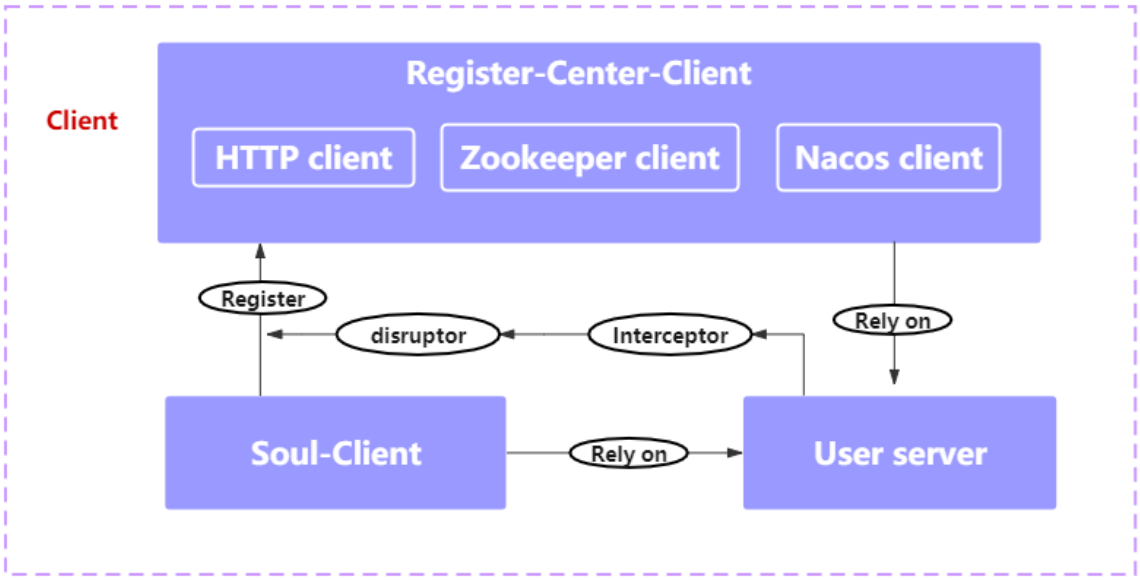
7 Register Center

7.1 Register Center Design

7.1.1 Description

- This article mainly explains three ways of register center and their principles.

Client



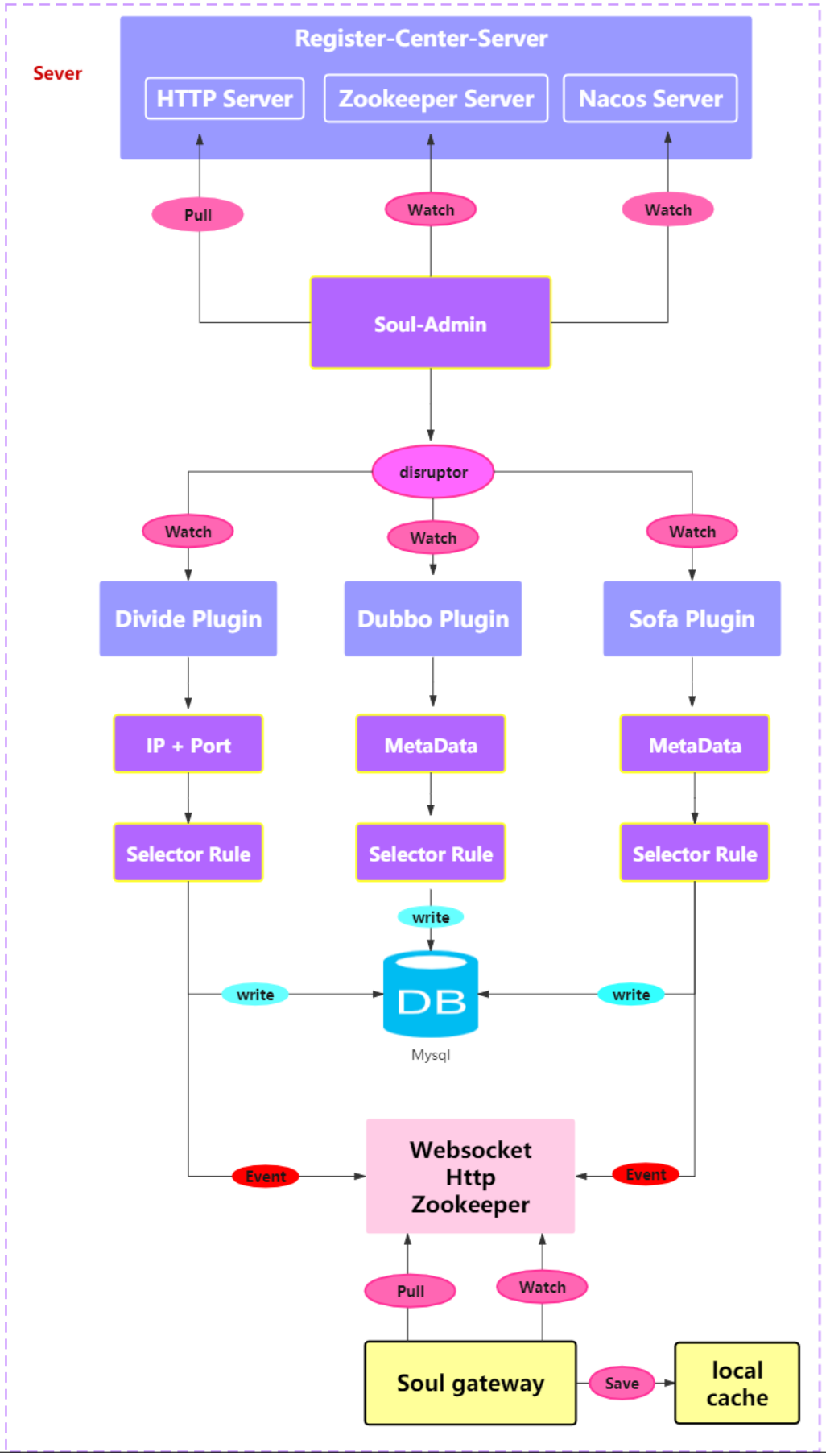
When client server start, the register center client will be loaded by spi.

Put data to Disruptor when spring bean load.

Soul register client get data from Disruptor, and it will send request to register server.

Disruptor can decouple data from operation and facilitate expansion.

Server


Sever
Register-Center-Server
HTTP Server
Zookeeper Server
Nacos Server
Pull
Watch
Watch
Soul-Admin
disruptor
Watch
Watch
Watch
Divide Plugin
Dubbo Plugin
Sofa Plugin
IP + Port
MetaData
MetaData
Selector Rule
Selector Rule
Selector Rule
write
write
DB
write
Mysql
Event
Websocket
Http
Zookeeper
Event
Pull
Watch
Soul gateway
Save
local
cache

When Soul-Admin server start, register center server will be loaded by spi. Meanwile Disruptor will be inited too.

Soul register server get data from register client, and then put then to Disruptor.

Soul-Admin Disruptor consumer get data from register server by Disruptor queue, then save them to database and publish data synchronize event.

Disruptor can decouple data from operation and buffering.

7.1.2 Http Registry

Principle of http register center is simple

Call interface of register server when Soul-Client start.

Soul-Admin accept request, then save to database and publish data synchronize event.

7.1.3 Zookeeper Registry

Zookeeper storage struct is:

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

Zookeeper register client will save data to zookeeper when soul client is started.

Zookeeper register server will keep watching the change of data node.

Trigger selector and rule data update and event will be published, when metadata data node update.

Trigger selector and upstream update and event will be published, when uri data node update.

7.1.4 Etcd Registry

Etcd storage struct is:

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

Etcd register client will save data to etcd when soul client is started.

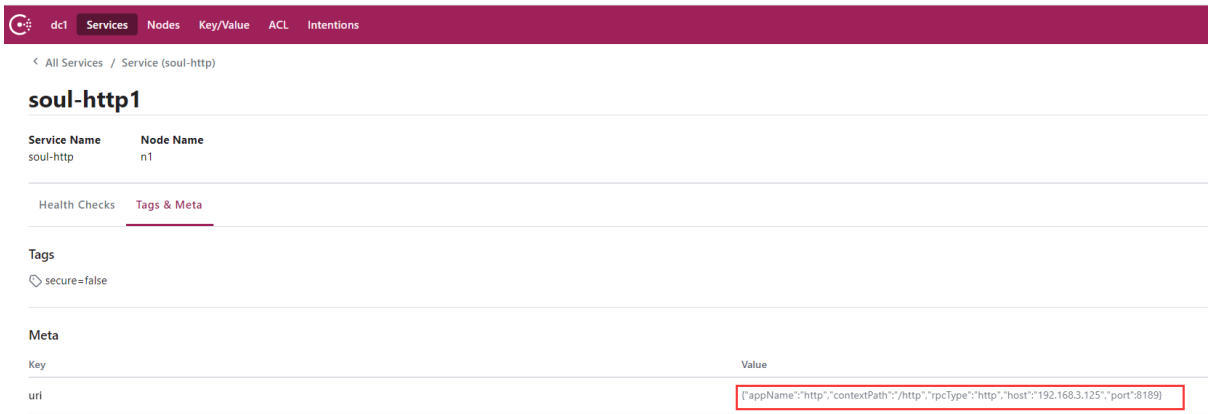
Etcd register server will keep watching the change of data node.

Trigger selector and rule data update and event will be published, when metadata data node update.

Trigger selector and upstream update and event will be published, when uri data node update.

7.1.5 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:

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

Consul register client will save data to consul when soul client is started.

Consul register server will keep watching the change of data node.

Trigger selector and rule data update and event will be published, when metadata data node update.

Trigger selector and upstream update and event will be published, when uri data node update.

7.1.6 Nacos Register

Nacos register have two parts： URI and Metadata。

URI is instance register. URI instance node will be deleted when server is down.

URI service’s instance name will be named like below. Every URI instance has ip, port and contextPath as identifiers.

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

When URI instance up, it will publish metadata config. It’s name like below.

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

Trigger selector and upstream update and event will be published, when URI service up or down.

Trigger selector and rule data update and event will be published, when metadata config update.

7.1.7 SPI

SPI Name	*Description*
SoulClientRegisterRepository	Soul client register SPI

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

SPI Name	*Description*
SoulServerRegisterRepository	Soul server register SPI

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

7.2 Register Center Access

7.2.1 Explain

Explain register center access config

7.2.2 HTTP Registry

Soul-Admin

- Set the config in application.yml

```
soul:
  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
```

Soul-Client

- Set the config in application.yml

```
soul:
  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 Soul-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 soul 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
```

7.2.3 Zookeeper Registry

Soul-Admin

- Add dependency in pom.xml (Default has been added):

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

- Set the config in application.yml

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

Soul-Client

- Add dependency in pom.xml (Default has been added):

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-register-client-zookeeper</artifactId>
    <version>${project.version}</version>
</dependency>
```

- Set the config in application.yml

```
soul:
  client:
    registerType: zookeeper
    serverLists: localhost:2181
    props:
      contextPath: /http
```

```
      appName: http
      port: 8188
      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 soul 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
```

7.2.4 Etcd Registry

Soul-Admin

- Add dependency in pom.xml (Default has been added):

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

- Set the config in application.yml

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

Soul-Client

- Add dependency in pom.xml (Default has been added):

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-register-client-etcd</artifactId>
    <version>${project.version}</version>
</dependency>
```

- Set the config in application.yml

```
soul:
  client:
    registerType: etcd
    serverLists: http://localhost:2379
    props:
      contextPath: /http
      appName: http
      port: 8188
      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 soul 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
```

7.2.5 Consul Registry

Soul-Admin

- Add dependency in pom.xml :

```
<!--soul-register-server-consul (Default has been added)-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-register-server-consul</artifactId>
    <version>${project.version}</version>
</dependency>

<!--spring-cloud-starter-consul-discovery need add by yourself, suggest use 2.2.6.
RELEASE version, other version maybe can't work-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-consul-discovery</artifactId>
    <version>2.2.6.RELEASE</version>
</dependency>
```

- Set the config in application.yml, additional need add spring.cloud.consul:

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

```
spring:
  cloud:
    consul:
      discovery:
        instance-id: soul-admin-1
        service-name: soul-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.
# 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.
# tags-as-metadata: false, Required, This option must be set to false, otherwise
the URI information will not be found, will cause to selector and upstream cache
unable to update.
```

Soul-Client

Note, consul registry is not compatible with current and SpringCloud will and Eureka / Nacos registry conflicts

- Add dependency in pom.xml (need add by yourself, suggest use 2.2.6.RELEASE version, other version maybe can't work):

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

- Set the config in application.yml, additional need add spring.cloud.consul:

```
soul:
  client:
    registerType: consul
    props:
      contextPath: /http
      appName: http
```

```
      port: 8188
      isFull: false

spring:
  cloud:
    consul:
      discovery:
        instance-id: soul-http-1
        service-name: soul-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 soul 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.
```

7.2.6 Nacos Registry

Soul-Admin

- Add dependency in pom.xml (Default has been added):

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

- Set the config in application.yml

```
soul:
  register:
    registerType: nacos
    serverLists : localhost:8848
    props:
      nacosNameSpace: SoulRegisterCenter
```

Soul-Client

- Add dependency in pom.xml (Default has been added):

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-register-client-nacos</artifactId>
    <version>${project.version}</version>
</dependency>
```

- Set the config in application.yml

```
soul:
  client:
    registerType: nacos
    serverLists: localhost:8848
    props:
      contextPath: /http
      appName: http
      port: 8188
      isFull: false
      nacosNameSpace: SoulRegisterCenter
# 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 soul 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
```

8 Quick Start

8.1 Quick start with Dubbo

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

8.1.1 Environment to prepare

Please refer to the setup and launch `soul-admin` and `soul-bootstrap`, In addition, if you use ZooKeeper for Dubbo, you need to download it in advance.

8.1.2 Run the soul-examples-dubbo project

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

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

Execute the `TestApacheDubboApplication` main method to start dubbo project.

The following log appears when the startup is successful:

```
2021-02-06 20:58:01.807  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/insert","pathDesc":"Insert a row of data","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboTestService",
"methodName":"insert","ruleName":"/dubbo/insert","parameterTypes":"org.dromara.
soul.examples.dubbo.api.entity.DubboTest","rpcExt":"{\"group\":\"\",\"version\":\"\
",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\":\"\"}",
"enabled":true}
2021-02-06 20:58:01.821  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findAll","pathDesc":"Get all data","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboTestService",
"methodName":"findAll","ruleName":"/dubbo/findAll","parameterTypes":"","rpcExt":"{\
"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\
":10000,\"url\":\"\"}","enabled":true}
```

```
2021-02-06 20:58:01.833  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findById","pathDesc":"Query by Id","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboTestService",
"methodName":"findById","ruleName":"/dubbo/findById","parameterTypes":"java.lang.
String","rpcExt":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\
"retries\":2,\"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.844  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByListId","pathDesc":"","rpcType":"dubbo","serviceName":
"org.dromara.soul.examples.dubbo.api.service.DubboMultiParamService","methodName":
"findByListId","ruleName":"/dubbo/findByListId","parameterTypes":"java.util.List",
"rpcExt":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\
":2,\"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.855  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByIdsAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboMultiParamService",
"methodName":"findByIdsAndName","ruleName":"/dubbo/findByIdsAndName",
"parameterTypes":"java.util.List,java.lang.String","rpcExt":"{\"group\":\"\",\
"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\
":\"\"}","enabled":true}
2021-02-06 20:58:01.866  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/batchSave","pathDesc":"","rpcType":"dubbo","serviceName":
"org.dromara.soul.examples.dubbo.api.service.DubboMultiParamService","methodName":
"batchSave","ruleName":"/dubbo/batchSave","parameterTypes":"java.util.List","rpcExt
":"{\"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\
"timeout\":10000,\"url\":\"\"}","enabled":true}
2021-02-06 20:58:01.876  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/findByArrayIdsAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboMultiParamService",
"methodName":"findByArrayIdsAndName","ruleName":"/dubbo/findByArrayIdsAndName",
"parameterTypes":"[Ljava.lang.Integer;,java.lang.String","rpcExt":"{\"group\":\"\",
\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\
":\"\"}","enabled":true}
2021-02-06 20:58:01.889  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/saveComplexBeanTestAndName","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboMultiParamService",
"methodName":"saveComplexBeanTestAndName","ruleName":"/dubbo/
saveComplexBeanTestAndName","parameterTypes":"org.dromara.soul.examples.dubbo.api.
entity.ComplexBeanTest,java.lang.String","rpcExt":"{\"group\":\"\",\"version\":\"\
",\"loadbalance\":\"random\",\"retries\":2,\"timeout\":10000,\"url\":\"\"}",
"enabled":true}
2021-02-06 20:58:01.901  INFO 3724 --- [pool-2-thread-1] o.d.s.client.common.utils.
RegisterUtils  : dubbo client register success: {"appName":"dubbo","contextPath":"/
dubbo","path":"/dubbo/batchSaveAndNameAndId","pathDesc":"","rpcType":"dubbo",
"serviceName":"org.dromara.soul.examples.dubbo.api.service.DubboMultiParamService",
"methodName":"batchSaveAndNameAndId","ruleName":"/dubbo/batchSaveAndNameAndId",
"parameterTypes":"java.util.List,java.lang.String,java.lang.String","rpcExt":"{\
"group\":\"\",\"version\":\"\",\"loadbalance\":\"random\",\"retries\":2,\"timeout\
":10000,\"url\":\"\"}","enabled":true}
```

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

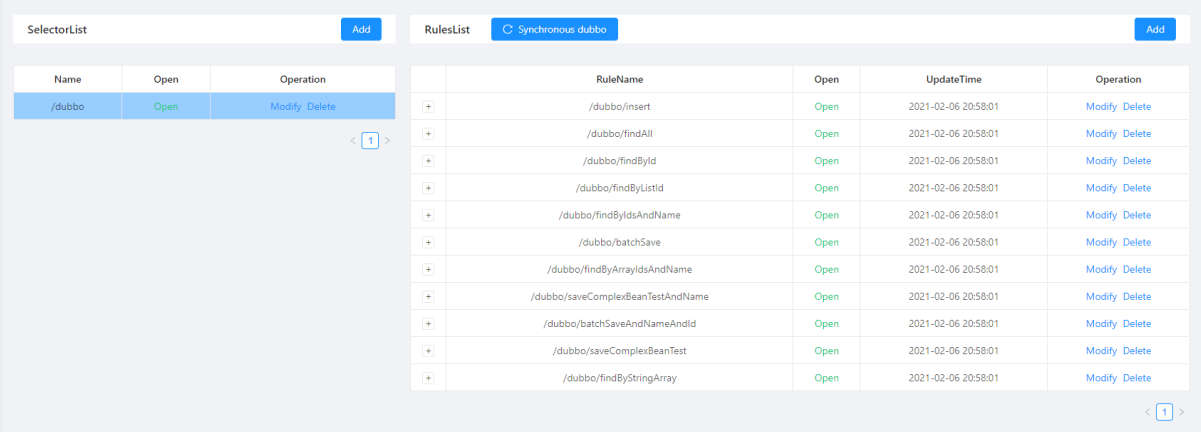

8.1.3 Dubbo plugin settings

- first enabled the dubbo plugin in the `soul-admin` plugin management.
- then configure your registry address in dubbo.

8.1.4 Testing

The `soul-examples-dubbo` project will automatically register interface methods annotated with `@SoulDubboClient` in the soul gateway after successful startup.

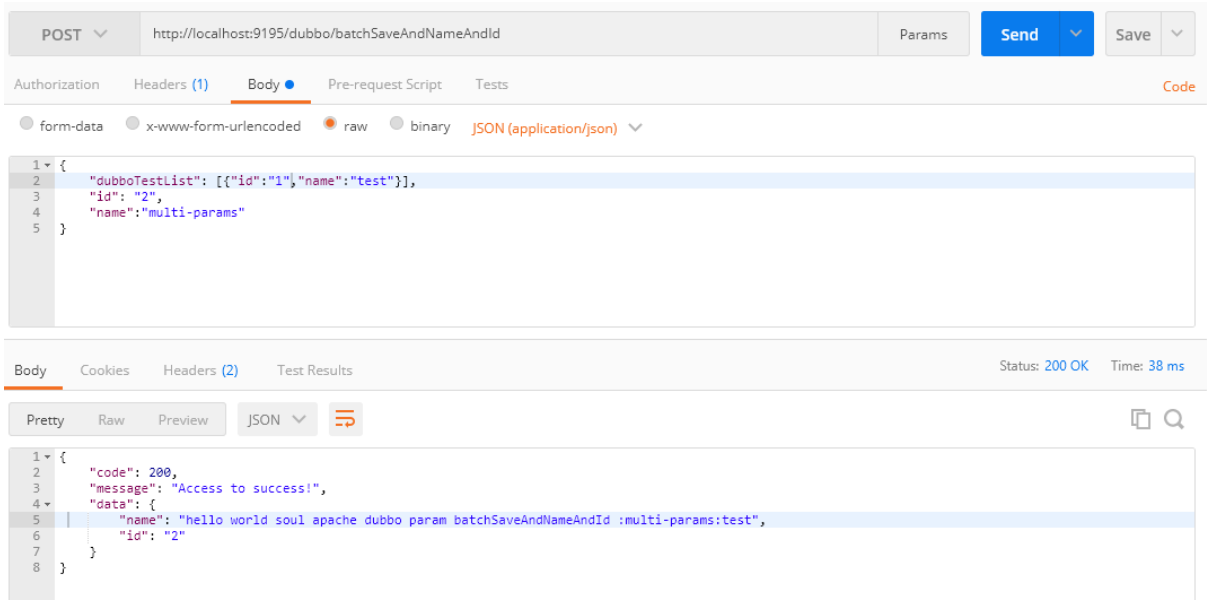
Open Plugin Management -> dubbo to see the list of plugin rule configurations



Use PostMan to simulate HTTP to request your Dubbo service

Complex multi-parameter example: The related interface implementation class is `org.dromara.soul.examples.apache.dubbo.service.impl.DubboMultiParamServiceImpl#batchSaveAndNameAndId`.

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



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

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

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

8.2 Quick start with http

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

8.2.1 Environment to prepare

Please refer to the setup and launch `soul-admin` and `soul-bootstrap`.

Introducing gateway to HTTP proxy plugin

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

```
<!--if you use http proxy start this-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-divide</artifactId>
```

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

<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-httpclient</artifactId>
    <version>${last.version}</version>
</dependency>
```

8.2.2 Run the soul-examples-http project

Downloadsoul-examples-http

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

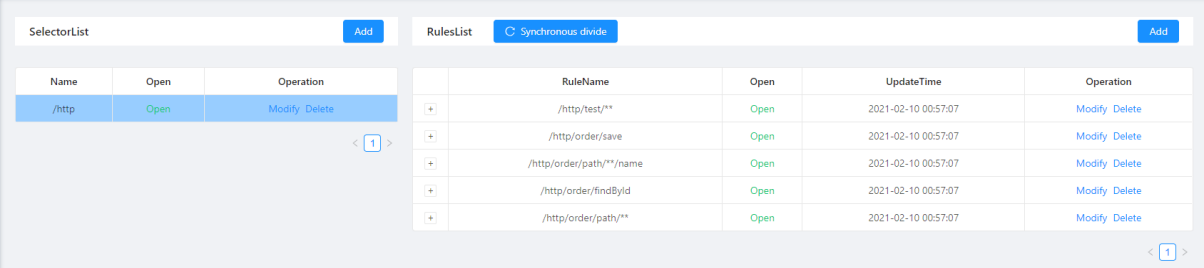
8.2.3 Enable the Divide plugin to handle HTTP requests

- enabled the `divide` plugin in the `soul-admin` plugin management.

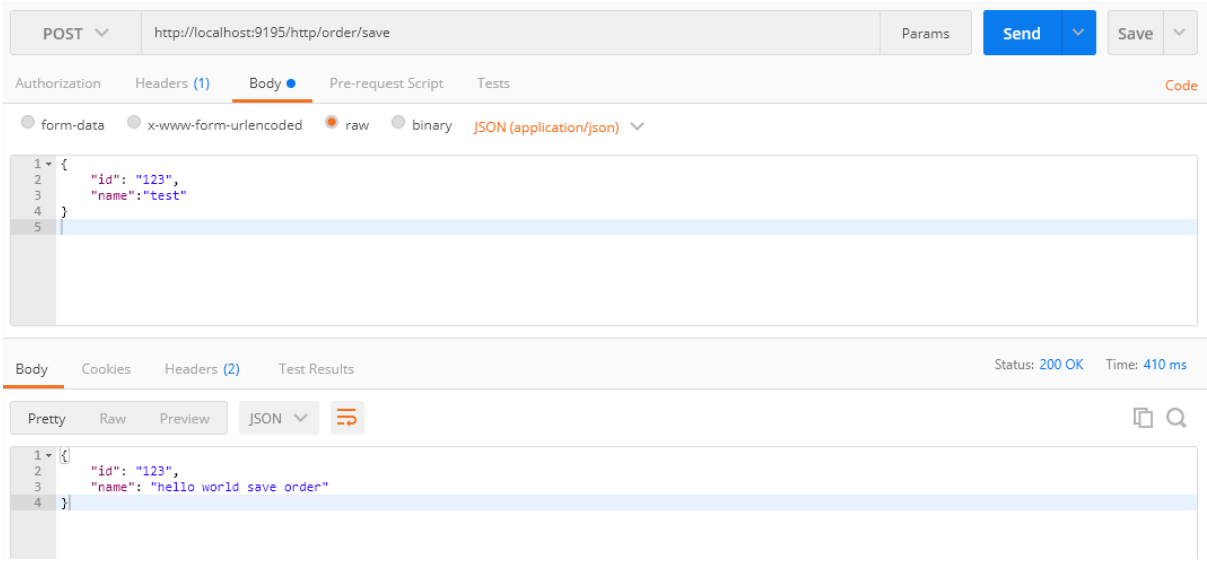
8.2.4 Testing http request

The `soul-examples-http` project will automatically register interface methods annotated with `@SoulSpringMvcClient` in the soul gateway after successful startup.

Open Plugin Management -> divide to see the list of plugin rule configurations



Use PostMan to simulate HTTP to request your http service



8.3 Quick start with grpc

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

8.3.1 Environment to prepare

Please refer to the setup and launch `soul-admin` and `soul-bootstrap`.

Note: `soul-bootstrap` need to import grpc dependencies

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

8.3.2 Run the soul-examples-grpc project

Download soul-examples-grpc

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

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

Execute the `org.dromara.soul.examples.grpc.SoulTestGrpcApplication` main method to start project.

The following log appears when the startup is successful:

```
2021-02-10 01:57:02.154  INFO 76 --- [           main] o.d.s.e.grpc.
SoulTestGrpcApplication     : Started SoulTestGrpcApplication in 2.088 seconds (JVM
running for 3.232)
2021-02-10 01:57:02.380  INFO 76 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : grpc client register success: {"appName":"127.0.0.1:8080",
"contextPath":"/grpc","path":"/grpc/echo","pathDesc":"","rpcType":"grpc",
"serviceName":"echo.EchoService","methodName":"echo","ruleName":"/grpc/echo",
"parameterTypes":"echo.EchoRequest,io.grpc.stub.StreamObserver","rpcExt":"{\
"timeout\":-1}","enabled":true}
```

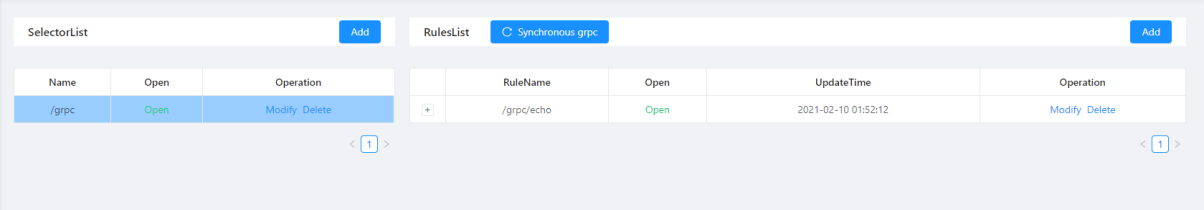
8.3.3 Grpc plugin settings

- enabled the `grpc` plugin in the `soul-admin` plugin management.

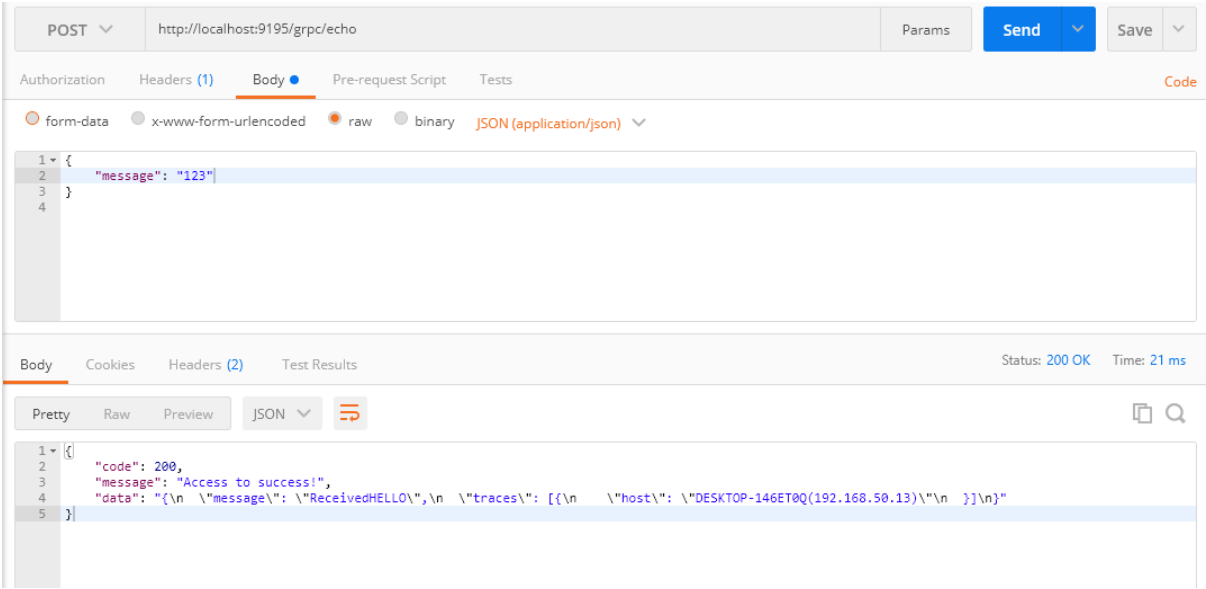
8.3.4 Testing

The `soul-examples-grpc` project will automatically register interface methods annotated with `@SoulGrpcClient` in the soul gateway after successful startup.

Open Plugin Management -> grpc to see the list of plugin rule configurations



Use PostMan to simulate HTTP to request your Grpc service



8.4 Quick start with SpringCloud

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

8.4.1 Environment to prepare

Please refer to the setup and launch `soul-admin` and `soul-bootstrap`.

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

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

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

<!-- If using Eureka as a registry needs to be introduced -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    <version>2.2.0.RELEASE</version>
</dependency>

<!--soul springCloud plugin start end-->
```

Startup the `soul-bootstrap` project

8.4.2 Run the soul-examples-springcloud and soul-examples-eureka project

In the example project we use `eureka` as the SpringCloud registry

Download soul-examples-eureka and soul-examples-springcloud

1. Startup the Eureka service

Execute the `org.dromara.soul.examples.eureka.EurekaServerApplication` main method to start project.

2. Startup the Spring Cloud service

Execute the `org.dromara.soul.examples.springcloud.SoulTestSpringCloudApplication` 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'
2021-02-10 14:03:51.669  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/save","pathDesc":"","rpcType
":"springCloud","ruleName":"/springcloud/order/save","enabled":true}
2021-02-10 14:03:51.676  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/path/**","pathDesc":"",
"rpcType":"springCloud","ruleName":"/springcloud/order/path/**","enabled":true}
```

```
2021-02-10 14:03:51.682  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/findById","pathDesc":"",
"rpcType":"springCloud","ruleName":"/springcloud/order/findById","enabled":true}
2021-02-10 14:03:51.688  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/order/path/**/name","pathDesc":"",
"rpcType":"springCloud","ruleName":"/springcloud/order/path/**/name","enabled
":true}
2021-02-10 14:03:51.692  INFO 2860 --- [pool-1-thread-1] o.d.s.client.common.utils.
RegisterUtils  : springCloud client register success: {"appName":"springCloud-test
","context":"/springcloud","path":"/springcloud/test/**","pathDesc":"","rpcType":
"springCloud","ruleName":"/springcloud/test/**","enabled":true}
2021-02-10 14:03:52.806  WARN 2860 --- [           main]
ockingLoadBalancerClientRibbonWarnLogger : You already have
RibbonLoadBalancerClient on your classpath. It will be used by default. As Spring
Cloud Ribbon is in maintenance mode. We recommend switching to
BlockingLoadBalancerClient instead. In order to use it, set the value of `spring.
cloud.loadbalancer.ribbon.enabled` to `false` or remove spring-cloud-starter-
netflix-ribbon from your project.
2021-02-10 14:03:52.848  WARN 2860 --- [           main] iguration
$LoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently working
with default default cache. You can switch to using Caffeine cache, by adding it to
the classpath.
2021-02-10 14:03:52.921  INFO 2860 --- [           main] o.s.c.n.eureka.
InstanceInfoFactory       : Setting initial instance status as: STARTING
2021-02-10 14:03:52.949  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Initializing Eureka in region us-east-1
2021-02-10 14:03:53.006  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using JSON encoding codec LegacyJacksonJson
2021-02-10 14:03:53.006  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using JSON decoding codec LegacyJacksonJson
2021-02-10 14:03:53.110  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using XML encoding codec XStreamXml
2021-02-10 14:03:53.110  INFO 2860 --- [           main] c.n.d.provider.
DiscoveryJerseyProvider   : Using XML decoding codec XStreamXml
2021-02-10 14:03:53.263  INFO 2860 --- [           main] c.n.d.s.r.aws.
ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-02-10 14:03:53.546  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Disable delta property : false
2021-02-10 14:03:53.546  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Single vip registry refresh property : null
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Force full registry fetch : false
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Application is null : false
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Registered Applications size is zero : true
```

```
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Application version is -1: true
2021-02-10 14:03:53.547  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Getting all instance registry info from the eureka server
2021-02-10 14:03:53.754  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : The response status is 200
2021-02-10 14:03:53.756  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Starting heartbeat executor: renew interval is: 30
2021-02-10 14:03:53.758  INFO 2860 --- [           main] c.n.discovery.
InstanceInfoReplicator     : InstanceInfoReplicator onDemand update allowed rate
per min is 4
2021-02-10 14:03:53.761  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Discovery Client initialized at timestamp 1612937033760 with
initial instances count: 0
2021-02-10 14:03:53.762  INFO 2860 --- [           main] o.s.c.n.e.s.
EurekaServiceRegistry        : Registering application SPRINGCLOUD-TEST with eureka
with status UP
2021-02-10 14:03:53.763  INFO 2860 --- [           main] com.netflix.discovery.
DiscoveryClient    : Saw local status change event StatusChangeEvent
[timestamp=1612937033763, current=UP, previous=STARTING]
2021-02-10 14:03:53.765  INFO 2860 --- [nfoReplicator-0] com.netflix.discovery.
DiscoveryClient    : DiscoveryClient_SPRINGCLOUD-TEST/host.docker.
internal:springCloud-test:8884: registering service...
2021-02-10 14:03:53.805  INFO 2860 --- [           main] o.s.b.w.embedded.tomcat.
TomcatWebServer  : Tomcat started on port(s): 8884 (http) with context path ''
2021-02-10 14:03:53.807  INFO 2860 --- [           main] .s.c.n.e.s.
EurekaAutoServiceRegistration : Updating port to 8884
2021-02-10 14:03:53.837  INFO 2860 --- [nfoReplicator-0] com.netflix.discovery.
DiscoveryClient    : DiscoveryClient_SPRINGCLOUD-TEST/host.docker.
internal:springCloud-test:8884 - registration status: 204
2021-02-10 14:03:54.231  INFO 2860 --- [           main] o.d.s.e.s.
SoulTestSpringCloudApplication : Started SoulTestSpringCloudApplication in 6.338
seconds (JVM running for 7.361)
```

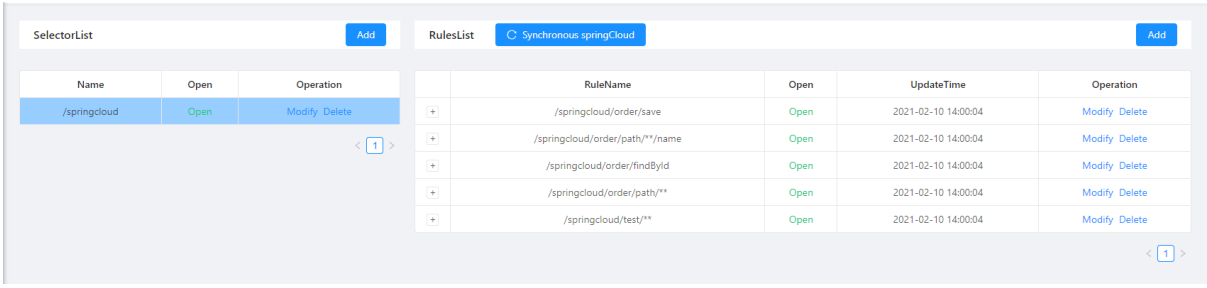
8.4.3 Enable the springCloud plugin

- enabled the `springCloud` plugin in the `soul-admin` plugin management.

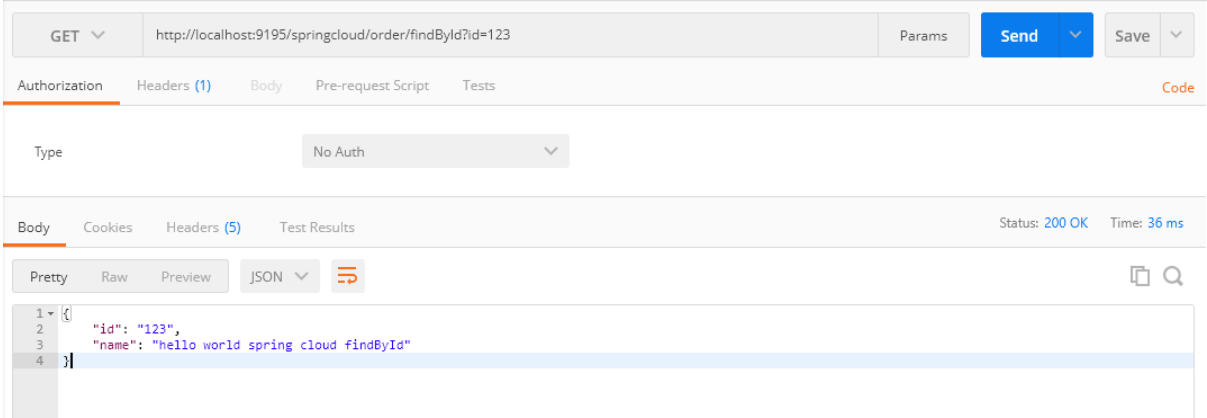
8.4.4 Testing http request

The `soul-examples-springcloud` project will automatically register interface methods annotated with `@SoulSpringCloudClient` in the soul gateway after successful startup.

Open Plugin Management -> springcloud to see the list of plugin rule configurations



Use PostMan to simulate HTTP to request your SpringCloud service



8.5 Quick start with sofa

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

8.5.1 Environment to prepare

Please refer to the setup and launch `soul-admin` and `soul-bootstrap`, In addition, if you use ZooKeeper for Sofa, you need to download it in advance.

Note: `soul-bootstrap` need to import `sofa` dependencies

```
<!-- soul sofa plugin starter-->
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-rpc-all</artifactId>
    <version>${sofa.rpc.version}</version>
</dependency>
<dependency>
```

```
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-sofa</artifactId>
    <version>${project.version}</version>
</dependency>

<!-- zookeeper -->
<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>
<!-- soul sofa plugin end-->
```

8.5.2 Run the soul-examples-sofa project

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

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

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

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

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

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

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

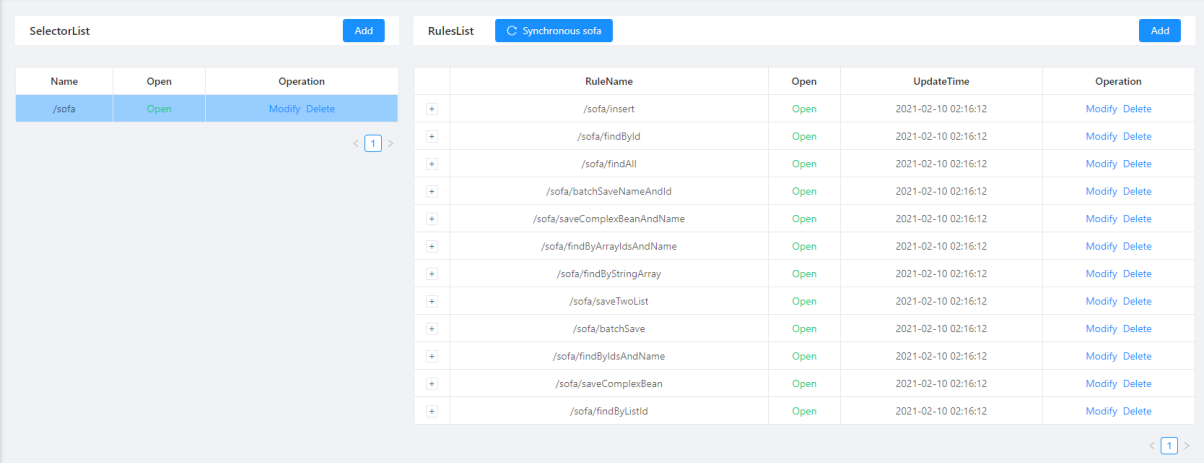
8.5.3 Sofa plugin settings

- first enabled the `sofa` plugin in the `soul-admin` plugin management.
- then configure your registry address in `sofa`.

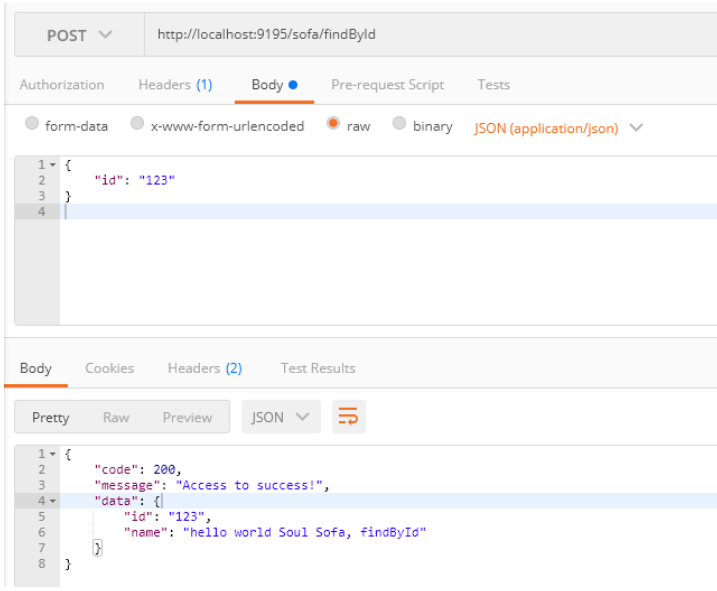
8.5.4 Testing

The `soul-examples-sofa` project will automatically register interface methods annotated with `@SoulSofaClient` in the soul gateway after successful startup.

Open Plugin Management -> sofa to see the list of plugin rule configurations

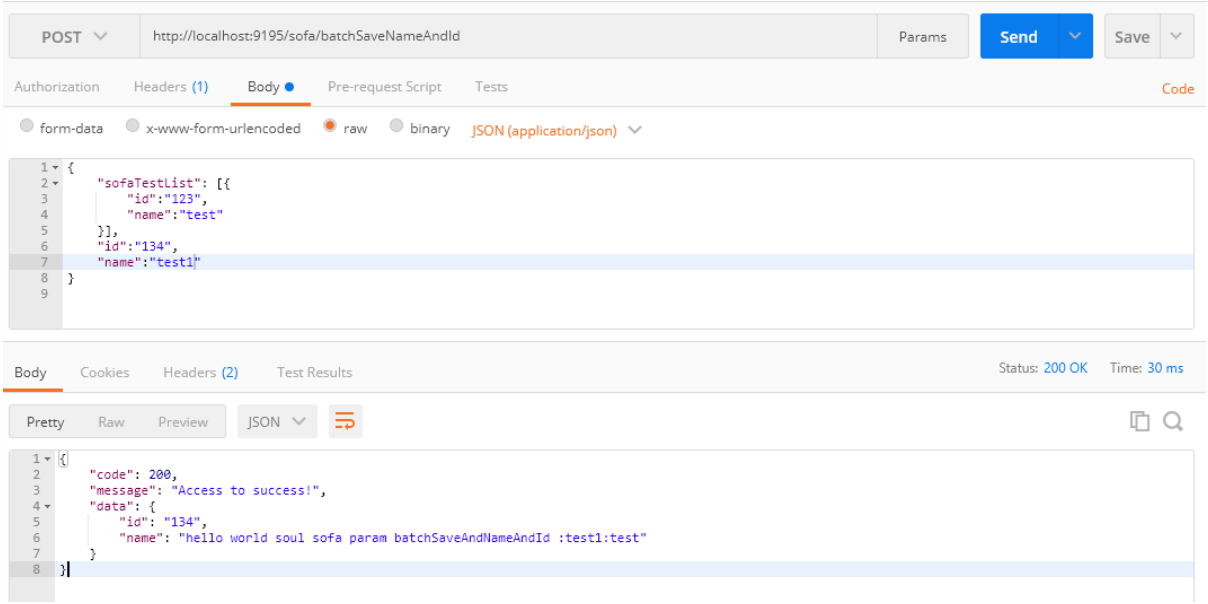


Use PostMan to simulate HTTP to request your Sofa service



Complex multi-parameter example: The related interface implementation class is `org.dromara.soul.examples.sofa.service.impl.SofaMultiParamServiceImpl#batchSaveNameAndId`

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



8.6 Quick start with Tars

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

8.6.1 Environment to prepare

Please refer to the setup and launch `soul-admin` and `soul-bootstrap`.

Note: `soul-bootstrap` need to import tars dependencies

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-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>
```

8.6.2 Run the soul-examples-tars project

Download soul-examples-tars

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

Modify config `src/main/resources/SoulExampleServer.SoulExampleApp.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 toTARS Official Documentation

Execute the `org.dromara.soul.examples.tars.SoulTestTarsApplication` main method to start project.

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

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

```
com.qq.tars.server.config.ConfigurationException: error occurred on load server
config
    at com.qq.tars.server.config.ConfigurationManager.
loadServerConfig(ConfigurationManager.java:113)
    at com.qq.tars.server.config.ConfigurationManager.init(ConfigurationManager.
java:57)
    at com.qq.tars.server.core.Server.loadServerConfig(Server.java:90)
    at com.qq.tars.server.core.Server.<init>(Server.java:42)
    at com.qq.tars.server.core.Server.<clinit>(Server.java:38)
    at com.qq.tars.spring.bean.PropertiesListener.
onApplicationEvent(PropertiesListener.java:37)
```

```
    at com.qq.tars.spring.bean.PropertiesListener.
onApplicationEvent(PropertiesListener.java:31)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
doInvokeListener(SimpleApplicationEventMulticaster.java:172)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
invokeListener(SimpleApplicationEventMulticaster.java:165)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
multicastEvent(SimpleApplicationEventMulticaster.java:139)
    at org.springframework.context.event.SimpleApplicationEventMulticaster.
multicastEvent(SimpleApplicationEventMulticaster.java:127)
    at org.springframework.boot.context.event.EventPublishingRunListener.
environmentPrepared(EventPublishingRunListener.java:76)
    at org.springframework.boot.SpringApplicationRunListeners.
environmentPrepared(SpringApplicationRunListeners.java:53)
    at org.springframework.boot.SpringApplication.
prepareEnvironment(SpringApplication.java:345)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:308)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1226)
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:1215)
    at org.dromara.soul.examples.tars.SoulTestTarsApplication.
main(SoulTestTarsApplication.java:38)
Caused by: java.lang.NullPointerException
    at java.io.FileInputStream.<init>(FileInputStream.java:130)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at com.qq.tars.common.util.Config.parseFile(Config.java:211)
    at com.qq.tars.server.config.ConfigurationManager.
loadServerConfig(ConfigurationManager.java:63)
    ... 17 more
The exception occurred at load server config
```

The following log appears when the startup is successful:

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

```
2021-02-09 13:28:24.837  INFO 16016 --- [pool-2-thread-1] o.d.s.client.common.
utils.RegisterUtils  : tars client register success: {"appName":"127.0.0.1:21715",
"contextPath":"/tars","path":"/tars/hello","pathDesc":"","rpcType":"tars",
"serviceName":"SoulExampleServer.SoulExampleApp.HelloObj","methodName":"hello",
"ruleName":"/tars/hello","parameterTypes":"int,java.lang.String","rpcExt":"{\
"methodInfo\":[{\"methodName\":\"helloInt\",\"params\":[{},{}],\"returnType\":\
"java.lang.Integer\"},{\"methodName\":\"hello\",\"params\":[{},{}],\"returnType\":\
"java.lang.String\"}]}","enabled":true}
```

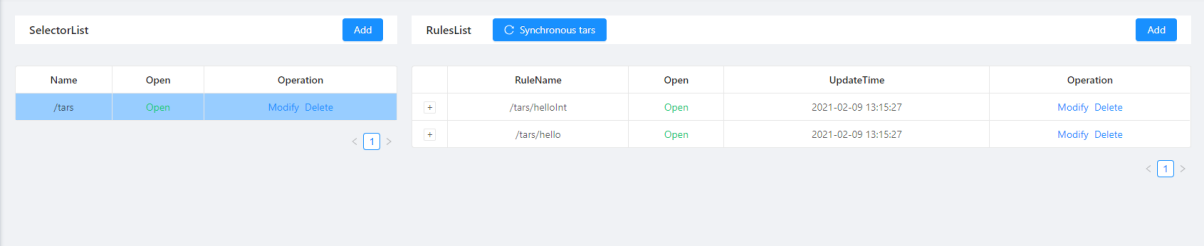

8.6.3 Tars plugin settings

- enabled the `tars` plugin in the `soul-admin` plugin management.

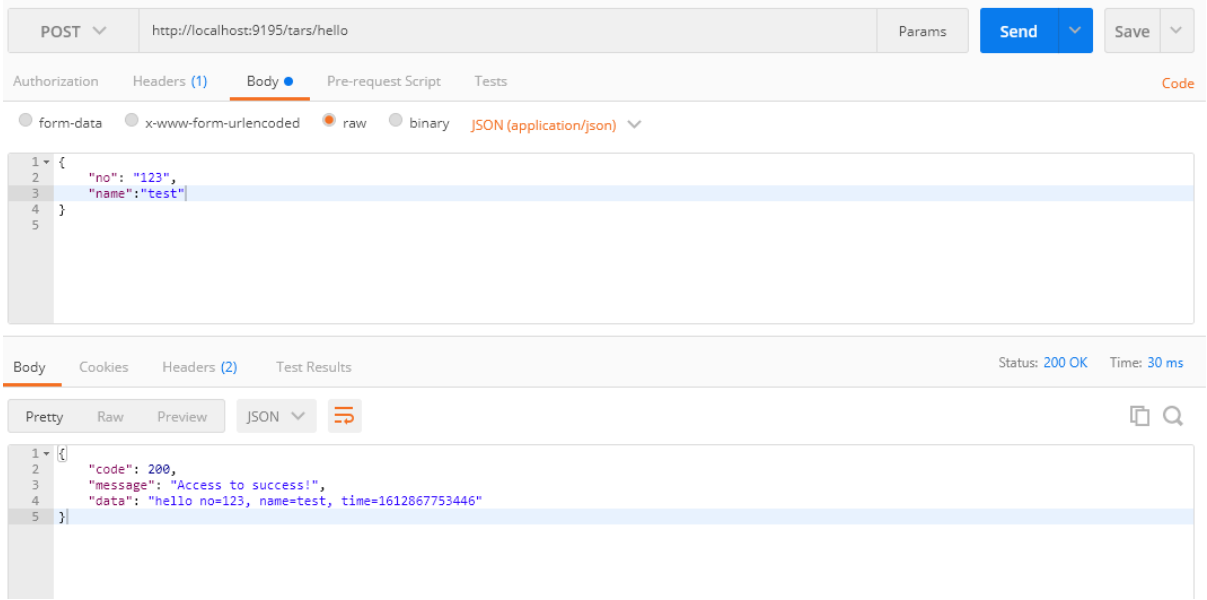
8.6.4 Testing

The `soul-examples-tars` project will automatically register interface methods annotated with `@SoulTarsClient` in the soul gateway after successful startup.

Open Plugin Management -> tars to see the list of plugin rule configurations



Use PostMan to simulate HTTP to request your tars service



9 Plugins

9.1 Divide Plugin

9.1.1 Explanation

- `Divide` is the core processing plugin for gateway to process `http` requests.

9.1.2 Plugin Setting

- Enable plugin, `soul-admin` –> plugin management–> `divide` set to enable.
- Divide plugin， cooperate with `starter` to take effect， please refer to：user-http。

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

 <dependency>
     <groupId>org.dromara</groupId>
     <artifactId>soul-spring-boot-starter-plugin-httpclient</artifactId>
     <version>${last.version}</version>
 </dependency>
```

9.1.3 Plugin Detail

- Divide is a plugin for http forward proxy, and all http requests are called by this plugin in load balancing.
- Selectors and rules, please refer to: selector.
- Http configuration is the real invoked configuration after the gateway matches the traffic; You can set multiple configurations and concrete load balancing weights in the rules.
  - Configuration Detail:
    * The first box: hostName, generally fill in `localhost`, which is temporarily unused.
    * The second box: http protocol, usually fill in `http://` or `https://`, if not, the default is: `http://`.
    * The third box: ip and port, where you fill in the ip+port of your real service.
    * The fourth box: load balancing weight.
  - Ip + Port Detection
    * In soul-admin, there is a scheduled task to scan the configured ip and port. If it is found that the ip and port is offline, it will be removed.
    * It can be configured as follows:

```
soul.upstream.check: true  //Default is ture, if setting false, program will not
detect.
soul.upstream.scheduledTime: 10  //Timing detection interval, default 10 seconds.
```

9.2 Dubbo Plugin

9.2.1 Explanation

- Dubbo is a plugin that converts `http protocol` into `Dubbo protocol` and it is also the key for gateway to realize dubbo generic service.
- Dubbo plugin needs to cooperate with metadata to realize dubbo calls, please refer to: metaData.
- Apache dubbo and alibaba dubbo users both use the same plugin.

```
<!--if you use dubbo start this-->
 <dependency>
     <groupId>org.dromara</groupId>
     <artifactId>soul-spring-boot-starter-plugin-alibab-dubbo</artifactId>
     <version>${last.version}</version>
 </dependency>

 <dependency>
     <groupId>org.dromara</groupId>
```

```
    <artifactId>soul-spring-boot-starter-plugin-apache-dubbo</artifactId>
    <version>${last.version}</version>
</dependency>
```

9.2.2 Plugin Setting

- In `soul-admin` –> plugin management-> dubbo setting enable.
- In the configuration of dubbo plugin, the configuration is as follows: Configure the registration center of dubbo.

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

- Plugin needs to cooperate with `starter` to take effect, please refer to: user-dubbo.
- Selectors and rules, please refer to: selector.

9.2.3 Metadata

- Every dubbo interface method corresponds to a piece of metadata, which can be found in soul-admin –>metadata management.
- Path: your http request.
- RPC extension parameters, corresponding to some configurations of dubbo interface; If you want to adjust, please modify here, which support json format like the following fields:

```
{"timeout":10000,"group":"",version":"","loadbalance":"","retries":1}
```

9.3 SpringCloud Plugin

9.3.1 Explanation

- This plugin is the core of transforming `http protocol` into `springCloud protocol`.

9.3.2 Introducing Plugin Support of SpringCould Gateway

- Introducing those dependencies in the pom.xml file of the gateway.

```
<!--soul springCloud plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-springcloud</artifactId>
    <version>${last.version}</version>
</dependency>
```

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-httpclient</artifactId>
    <version>${last.version}</version>
</dependency>
<!--soul springCloud plugin end-->

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

9.3.3 Plugin Setting

- In `soul-admin` –> plugin management-> springCloud, set to enable.
- This plugin needs to cooperate with `starter` dependency, please refer to:user-spring.
- Selectors and rules, please refer to: selector.

9.3.4 Detail

- Application name: it is your specific application name that needs to be invoked after the conditions are matched.
- Soul will obtain the real IP of the corresponding service and initiate http proxy calls from registration center of springCloud.

9.4 Sofa Plugin

9.4.1 Description

- The sofa plug-in is a plug-in that converts the HTTP protocol into the sofa protocol, and it is also the key to the gateway to realize the sofa generalization call.
- sThe sofa plug-in needs to cooperate with metadata to realize the call of Dubbo. Please refer to: Metadata。

```
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-sofa</artifactId>
    <version>${last.version}</version>
</dependency>
```

9.4.2 Plugin Settings

- First, go to `soul-admin` –> plug-in management-> setting `sofa` is open.
- Then, in the configuration of sofa plug-in, config sofa’ s register center like this:

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

- The plug-in needs to be used with a dependent `starter`. For details, please see: user-sofa.
- Selector’ s rules， see : selector。

9.4.3 Plugin Metadata

- Each sofa interface method corresponds to a piece of metadata, which can be viewed in the soul-admin > metadata management.
- `url`: It’ s your http urls.
- RPC extension parameter, corresponding to some configuration of sofa interface.If you wanna be modify it, please modify it here. Support JSON format. The following fields:

```
{"loadbalance":"hash","retries":3,"timeout":-1}
```

9.5 RateLimiter Plugin

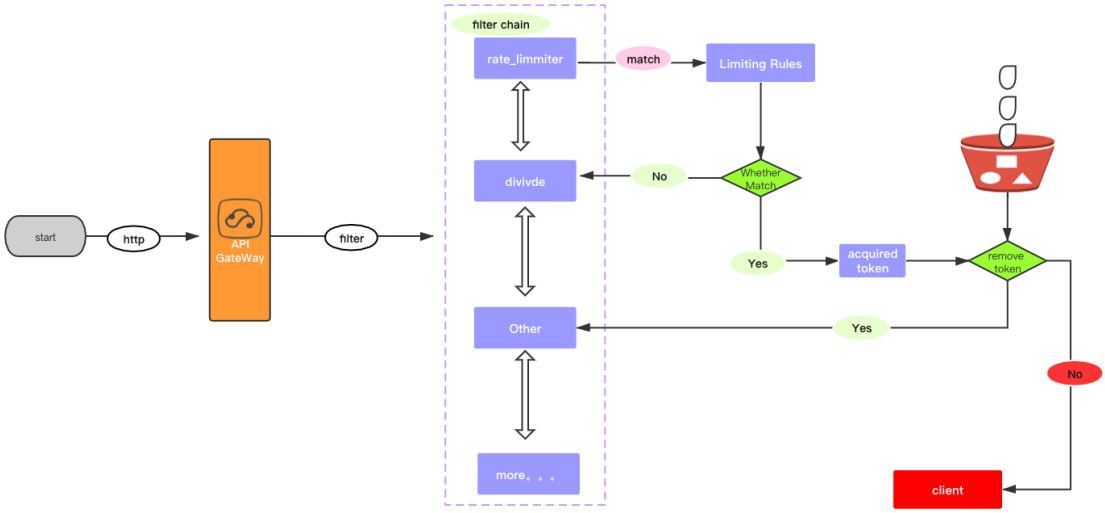
9.5.1 Explanation

- rateLimiter is core implementation of gateway restrictions on network traffic.
- The soul gateway provides a variety of current limiting algorithms, including `token bucket algorithm`, `concurrent token bucket algorithm`, `leaky bucket algorithm` and `sliding time window algorithm`.
- The implementation of current limiting algorithm of soul gateway is based on `redis`.
- You can set to the interface level or the parameter level. How to use it depends on your traffic configuration.

9.5.2 Technical Solution

Using redis token bucket algorithm to limit traffic.

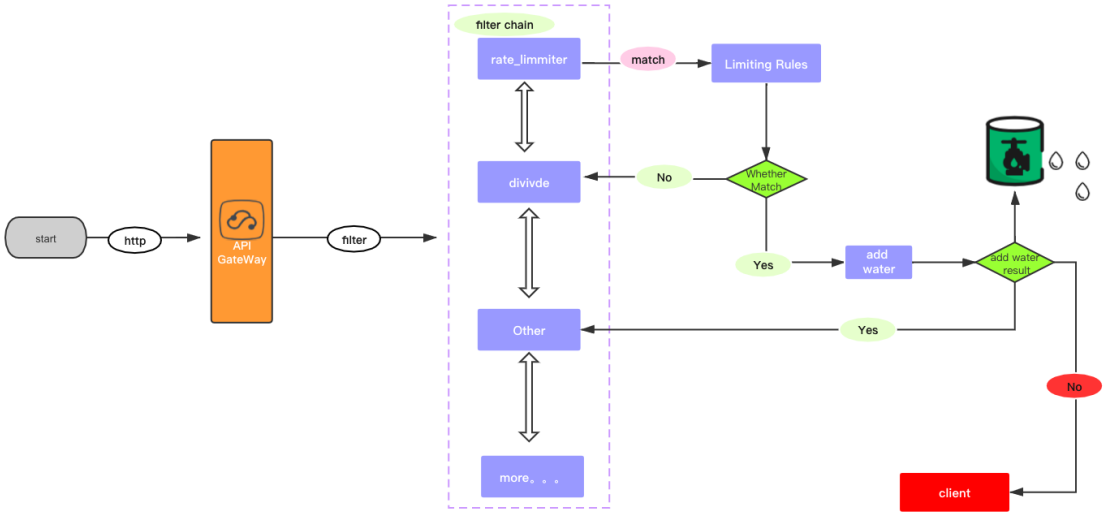
- The system generates the token at a constant rate, and then puts the token into the token bucket.
- The token bucket’ s capacity. When the bucket is full, the token put into it will be discarded.
- Each time requests come, you need to obtain a token from the token bucket. If there are tokens, the service will be provided; if there are no tokens, the service will be rejected.



- Flow Diagram:

Using redis leaky bucket algorithm to limit traffic.

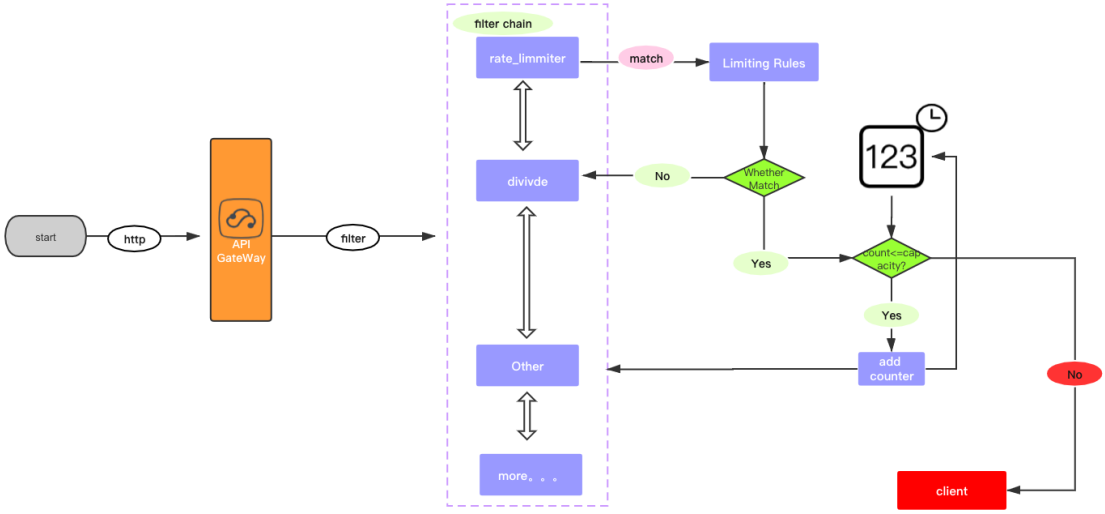
- water (request) go to the leaky bucket first. The leaky bucket goes out at a fixed speed. When the flow speed is too fast, it will overflow directly (reject service)



- Flow Diagram:

Using redis sliding time window algorithm to limit traffic.

- The sliding time window maintains the count value of unit time. Whenever a requests pass, the count value will be increased by 1. When the count value exceeds the preset threshold, other requests in unit time will be rejected. If the unit time has ended, clear the counter to zero and start the next round counting.



- Flow Diagram:

9.5.3 Plugin Setting

- In `soul-admin`–> plugin management–> `rate_limiter` set to enable.
- Configure redis in the plugin.
- Currently, supporting redis patterns of single, sentinel, and cluster.
- If it is a sentinel, cluster and other multi-node configuration in URL, please use ; for each instance; Division. For example, 192.168.1.1:6379; 192.168.1.2:6379。
- If the user don’ t use, please disable the plugin in the background.

9.5.4 Plugin Detail

- Introduce `rateLimiter` dependency in pom.xml file of the gateway.

```
<!-- soul ratelimiter plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-ratelimiter</artifactId>
    <version>${last.version}</version>
</dependency>
<!-- soul ratelimiter plugin end-->
```

- Selectors and rules, please refer to: selector。
- Detailed description of the rules

- – Token bucket algorithm/Concurrent token bucket algorithm

lgorithmName：tocketBucket/concurrent

replenishRate：It is how many requests you allow users to execute per second, while not discarding any requests. This is the filling rate of token bucket.

burstCapacity：it is the maximum number of requests that users are allowed to execute in one second. This is token bucket can save the number of token.

- – Leaky bucket algorithm

algorithmName：leakyBucket

replenishRate：The rate at which requests are executed per unit time, and the rate at which water droplets leak out of the leaky bucket.

burstCapacity：The maximum number of requests that users are allowed to execute in one second. This is the amount of water in the bucket.

- – Sliding time window algorithm

algorithmName：sildingWindow

replenishRate：The rate of requests per unit time, used to calculate the size of the time window.

burstCapacity：The maximum number of requests in the time window (per unit time).

9.6 Hystrix Plugin

9.6.1 Explanation

- Hystrix plugin is the core implementation used by gateway to fuse traffic.
- Use semaphores to process requests.

9.6.2 Plugin Setting

- In `soul-admin` –> plugin management –> `hystrix`, set to enable.
- If the user don’t use, please disable the plugin in the background.

9.6.3 Plugin Instruction

- Introduce `hystrix` dependency in the pom.xml file of the gateway.

```
<!-- soul hystrix plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-hystrix</artifactId>
     <version>${last.version}</version>
```

```
</dependency>
<!-- soul hystrix plugin end-->
```

- Selectors and rules, please refer to: selector.
- Hystrix processing details:
  - Trip minimum request quantity: the minimum request quantity, which must be reached at least before the fuse is triggered
  - Error half-score threshold: the percentage of exceptions in this period of time.
  - Maximum concurrency: the maximum concurrency
  - Trip sleep time (ms): the recovery time after fusing.
  - Grouping Key: generally set as: contextPath
  - Command Key: generally set to specific path interface.
  - CallBackUrl: default url: /fallback/hystrix.

9.7 Sentinel Plugin

9.7.1 Explanation

- Sentinel is one of the options that supports flow control and circuit breaking.
- Sentinel supports flow control and circuit breaking functions for gateway.

9.7.2 Plugin Setting

- In `soul-admin` –> plugin management –> `sentinel` set to enable.
- If you don’ t want to use it, please close the plugin in `soul-admin`.

9.7.3 Plugin Usage

- Introducing the follow supports to the pom.xml file of soul project.

```
<!-- soul sentinel plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-sentinel</artifactId>
     <version>${last.version}</version>
</dependency>
<!-- soul sentinel plugin end-->
```

- Selectors and rules, please refer to: selector

- Sentinel Processing Details
  - `flowRuleEnable (1 or 0)`: whether enable sentinel flow control function.
  - `flowRuleControlBehavior`: effect(reject directly/ queue/ slow start up), it do not support flow control by invocation relation.
  - `flowRuleGrade`: type of current limit threshold(QPS or Thread Count)。
  - `degradeRuleEnable (1 or 0)`: whether enable circuit breaking function of `sentinel`.
  - `degradeRuleGrade`: circuit breaker strategy, support RT of seconds level/ Error Ratio of seconds level/ Error Count of minutes level strategy.
  - `degradeRuleCount`: threshold.
  - `degradeRuleTimeWindow`: time of degrading(unit: second).
  - `fallbackUri`: degraded uri after circuit breaking.

9.8 Resilience4j Plugin

9.8.1 Explanation

- Resilience4j is one of the options that supports flow control and circuit breaking.
- Resilience4j supports flow control and circuit breaking functions for gateway.

9.8.2 Plugin Setting

- In `soul-admin` –> plugin management –> `resilience4j` set to enable.
- If you don’t want to use it, please close the plugin in `soul-admin`.

9.8.3 Plugin Usage

- Introducing the follow supports to the pom.xml file of soul project.

```
<!-- soul resilience4j plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-resilience4j</artifactId>
     <version>${last.version}</version>
</dependency>
<!-- soul resilience4j plugin end-->
```

- Selectors and rules, please refer to: selector
- Resilience4j Processing Details

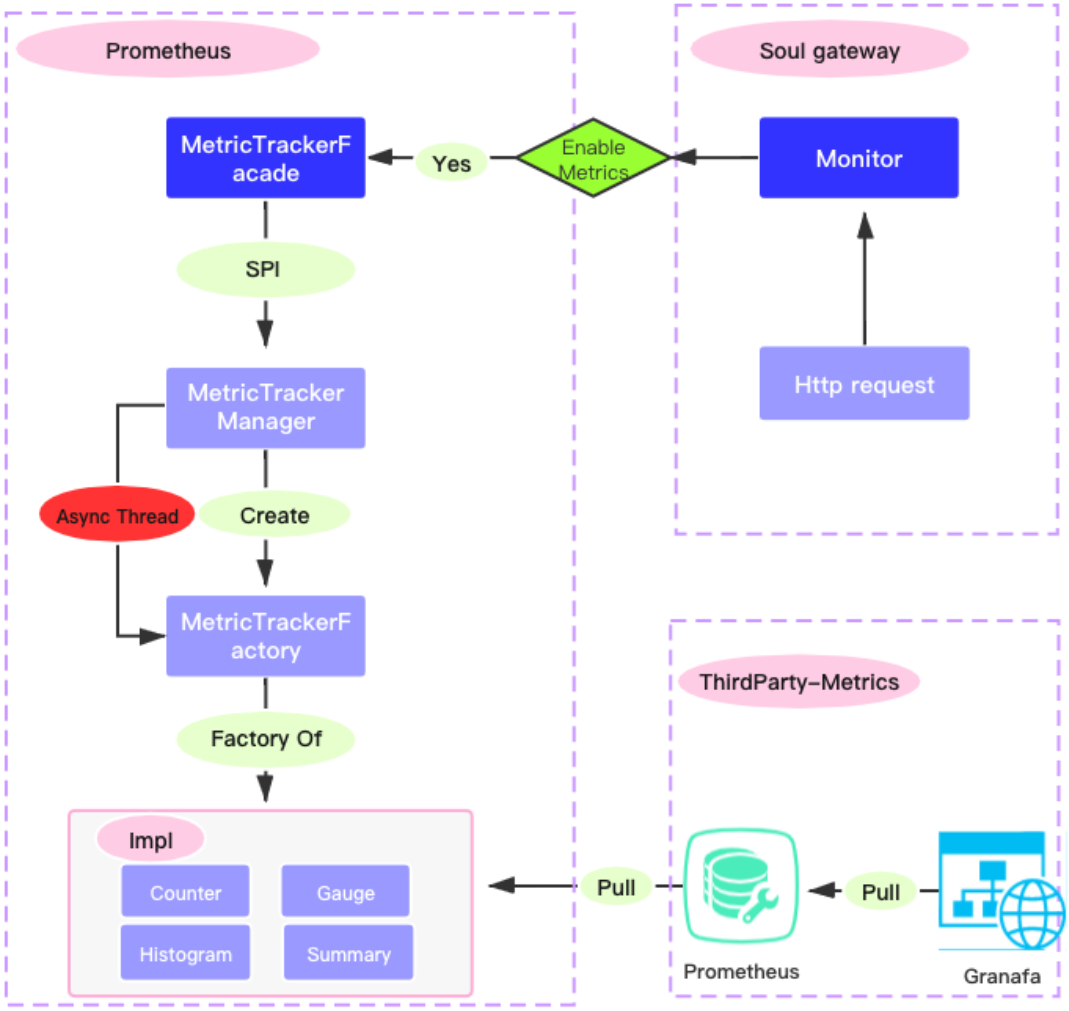
- timeoutDurationRate ： Configures wait time(ms) a thread waits for a permission,default value:5000。
- limitRefreshPeriod ： Configures the period of a limit refresh. After each period the rate limiter sets its permissions count back to the limitForPeriod value,default value:500。
- limitForPeriod ： Configures the number of permissions available during one limit refresh period,default value:50。
- circuitEnable ： Configures circuitBreaker enable,0:OFF,1:ON ,default value:0。
- timeoutDuration ： Configures request CircuitBreaker timeout(ms),default value:30000。
- fallbackUri ： Configures the fall back uri。
- slidingWindowSize ： Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed,default value:100。
- slidingWindowType ： Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed, Sliding window can either be 0:count-based or 1:time-based.,default value:0。
- minimumNumberOfCalls ： Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate or slow call rate,default value:100。
- waitIntervalFunctionInOpenState ： Configures the circuitBreaker time(ms) of duration,default value:10。
- permittedNumberOfCallsInHalfOpenState：Configures the number of permitted calls when the CircuitBreaker is half open,default value:10。
- failureRateThreshold： Configures the failure rate threshold in percentage,When the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls,default value:50。
- automaticTransitionFromOpenToHalfOpenEnabled ： Configures automatically transition from open state to half open state,true:ON,false:OFF,default value:false。

9.9 Monitor Plugin

9.9.1 Explanation

- Monitor plugin is used to monitor its own running status(JVM-related) by gateway, include request response delay, QPS, TPS, and other related metrics.

9.9.2 Technical Solutions



- Flow Diagram
- Make even tracking in soul gateway by asynchronous or synchronous mode.
- The `prometheus` server pulls metrics' through http request, and then displays it by `Grafana`.

9.9.3 Plugin Setting

- In `soul-admin`-> plugin management-> monitor, set to enable.
- Add the following configuration in the monitor plugin.

```
{"metricsName":"prometheus","host":"localhost","port":"9191","async":"true"}

# port : Pulled ports for exposing to prometheus service.
# host : If not filled in, it is the host of soul Gateway.
# async :"true" is asynchronous event tracking,  false is synchronous event
tracking.
```

- If the user don't use, please disable the plugin in the background.

- Introduce `monitor` dependency in the pom.xml file of the gateway.

```
<!-- soul monitor plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-monitor</artifactId>
    <version>${last.version}</version>
</dependency>
<!-- soul monitor plugin end-->
```

- Selectors and rules, please refer to: selector.
- Only when the url is matched, the url will request event tracking.

9.9.4 Metrics Detail

- All JVM, thread, memory, and other related infomation will be made event tracking, you can add a JVM module in the Granfana' panel, and it will be fully displayed, please refer to: https://github.com/prometheus/jmx_exporter
- There are also the following custom `metrics`

Name	type	target	description
request_total	Counter	none	collecting all requests of Soul Gateway
http_request_total	Counter	path,type	collecting all matched requests of monitor

9.9.5 Collect metrics

Users need to install `Prometheus` service to collect

- Choose the corresponding environment download address to install
- Modify configuration file: `prometheus.yml`

```
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped
from this config.
  - job_name: 'prometheus'
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
    - targets: ['localhost:9190']
```

Note: The `job_name` corresponds to the `metricsName` of the `monitor` plug-in configuration

- After the configuration is completed, you can directly double-click `prometheus.exe` in the window to start. The default boot port is `9090`, Success can be verified at http://localhost:9090/

9.9.6 Panel Display

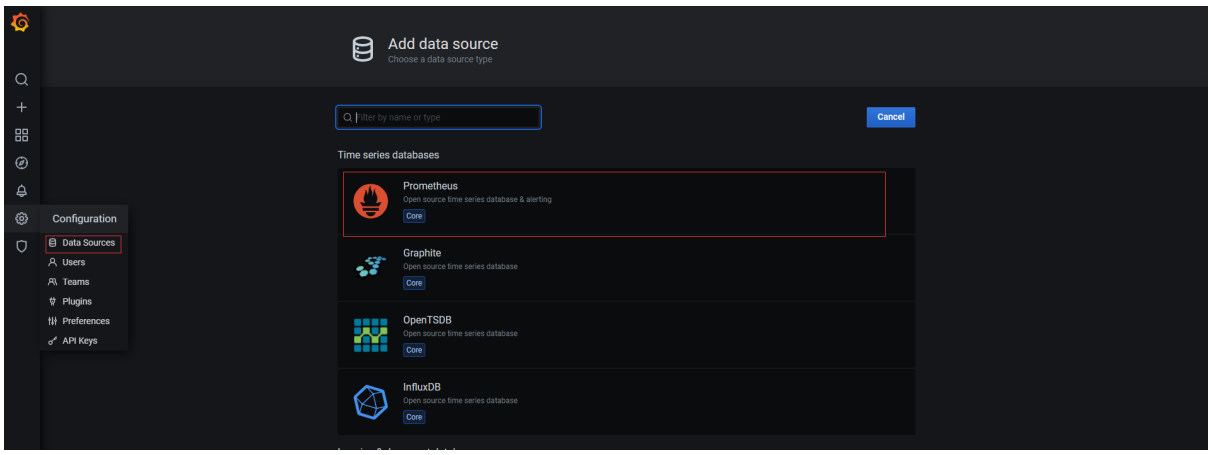
It is recommended to use `Grafana`, Users can customize the query to personalize the display panel.

Here's how to install and deploy `Grafana for Windows`

- Install Grafana

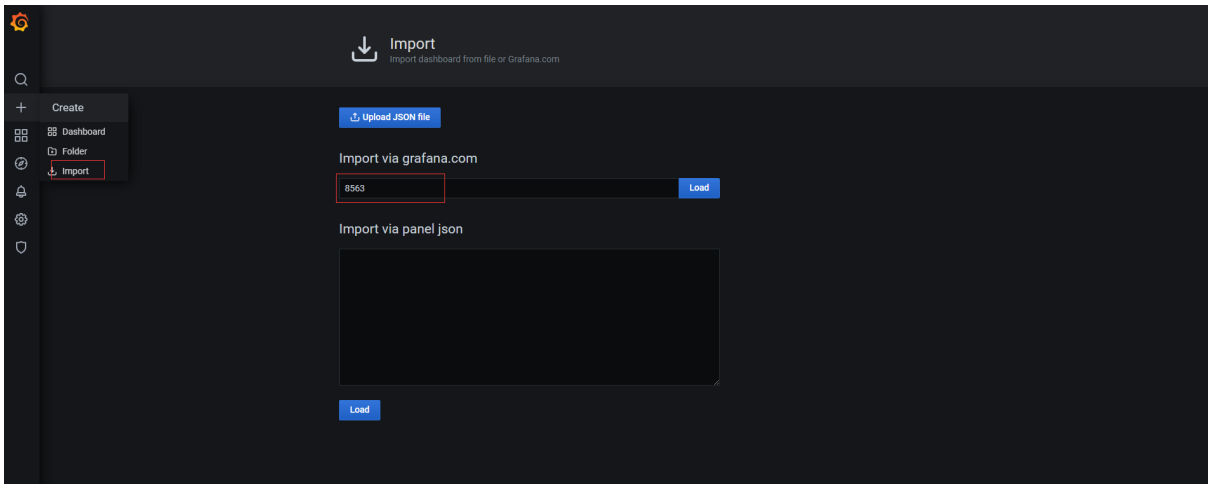
download Unzip it and enter the `bin` directory and `double-click` `grafana-server.exe` to run it. Go to http://localhost:3000/?orgId=1 `admin/admin` to verify the success

- Config Prometheus DataSource

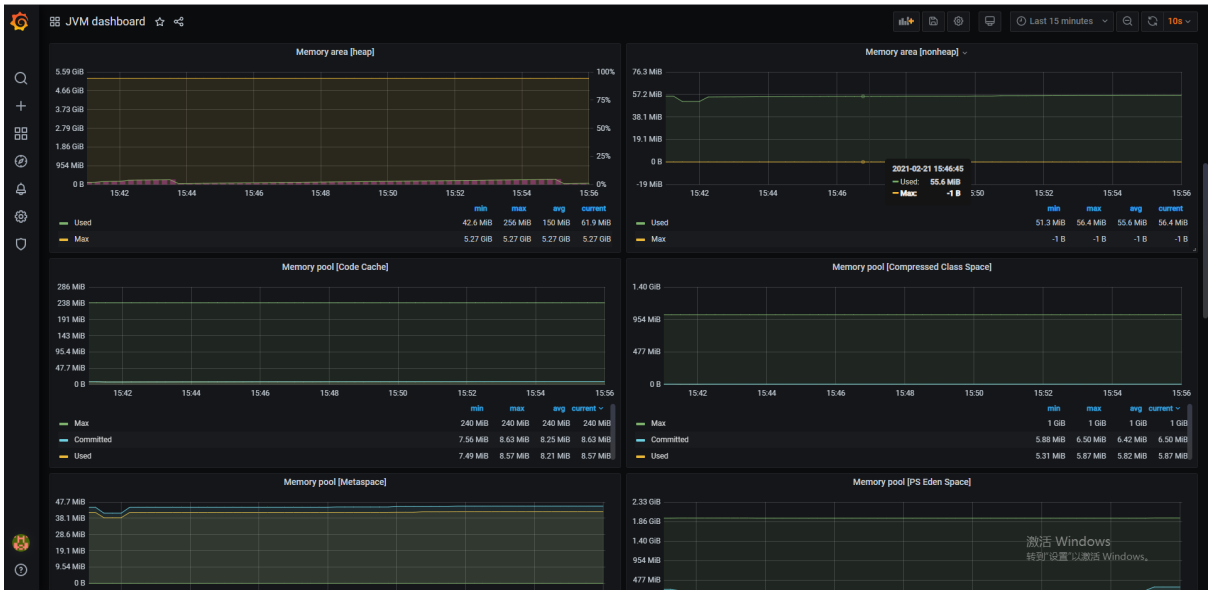


- Config JVM Dashboard

Click `Create` - `Import` and enter the dashboards ID (8563 recommended).



The final JVM monitor panel looks like this:



- Config Custom Metric Dashboard `request_total`、`http_request_total`

Click `Create` - `Import` and enter the panel config json

The final custom HTTP request monitoring panel looks like this:

9.10 Waf Plugin

9.10.1 Explanation

- Waf is the core implementation of gateway to realize firewall function for network traffic.

9.10.2 Plugin Setting

- In `soul-admin` –> plugin management-> `waf` set to enable.
- If the user don’ t use, please disable the plugin in the background.
- Add configuration mode in plugin editing.

```
{"model":"black"}
# The default mode is blacklist mode; If setting is mixed, it will be mixed mode.
We will explain it specifically below.
```

9.10.3 Plugin Setting

- Introducing `waf` dependency in the pom.xml of the gateway.

```
<!-- soul waf plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-waf</artifactId>
    <version>${last.version}</version>
</dependency>
<!-- soul waf plugin end-->
```

- Selectors and rules, please refer to : selector
- When `model` is set to `black` mode, only the matched traffic will execute the rejection policy, and the unmatched traffic will be skipped directly.
- When `model` is set to `mixed` mode, all traffic will pass through waf plugin. For different matching traffic, users can set whether to reject or pass.

9.10.4 Situation

- Waf is also the pre-plugin of soul, which is mainly used to intercept illegal requests or exception requests and give relevant rejection policies.
- When faced with replay attacks, you can intercept illegal ip and host, and set reject strategy according to matched ip or host.
- How to determine ip and host, please refer to: parsing-ip-and-host

9.11 Sign Plugin

9.11.1 Explanation

- Sign is a native plugin of soul Gateway and is used to process signature authentication of requests.

9.11.2 Plugin Setting

- In `soul-admin` -> plugin management –> `sign` set to enable.

9.11.3 Plugin Usage

- Introducing `sign` dependency in the pom.xml file of the gateway

```
<!-- soul sign plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-sign</artifactId>
   <version>${last.version}</version>
</dependency>
<!-- soul sign plugin end-->
```

- Selectors and rules, please refer to: selector.
- Only those matched requests can be authenticated by signature.

9.11.4 Add AK/SK

- In soul-admin –> In authentication management, click Add to add a new AK/SK。

9.11.5 Implementation of Gateway Technology

- Adopt Ak/SK authentication technical scheme.
- Adopt authentication plug-in and Chain of Responsibility Pattern to realize.
- Take effect when the authentication plugin is enabled and all interfaces are configured for authentication.

Authentication Guide

- Step 1: AK/SK is assigned by the gateway. For example, the AK assigned to you is: `1test123456781` SK is: ` 506eeb535cf740d7a755cb49f4a1536'
- Step 2: Decide the gateway path you want to access, such as ` /api/service/abc'
- Step 3: Construct parameters (the following are general parameters)

Field	Value	Description
timestamp	current timestamp(String)	The number of milliseconds of the current time（gateway will filter requests the before 5 minutes）
path	/api/service/abc	The path that you want to request(Modify by yourself according to your configuration of gateway)
version	1.0.0	`1.0.0` is a fixed string value

Sort the above two field natually according to the key, then splice fields and fields, finally splice SK. The following is a code example.

Step 1: First, construct a Map.

```
Map<String, String> map = Maps.newHashMapWithExpectedSize(2);
//timestamp is string format of millisecond. String.valueOf(LocalDateTime.now().
toInstant(ZoneOffset.of("+8")).toEpochMilli())
map.put("timestamp","1571711067186");  // Value should be string format of
milliseconds
map.put("path", "/api/service/abc");
map.put("version", "1.0.0");
```

Step 2: Sort the Keys naturally, then splice the key and values, and finally splice the SK assigned to you.

```
List<String> storedKeys = Arrays.stream(map.keySet()
                .toArray(new String[]{}))
                .sorted(Comparator.naturalOrder())
                .collect(Collectors.toList());
final String sign = storedKeys.stream()
                .map(key -> String.join("", key, params.get(key)))
                .collect(Collectors.joining()).trim()
                .concat("506EEB535CF740D7A755CB4B9F4A1536");
```

- The returned sign value should be:path/api/service/abctimestamp1571711067186version1.0.0506EEB535CF740D7A

Step 3: Md5 encryption and then capitalization.

```
DigestUtils.md5DigestAsHex(sign.getBytes()).toUpperCase()
```

- The final returned value is:A021BF82BE342668B78CD9ADE593D683

9.11.6 Request GateWay

- If your visited path is:/api/service/abc。
- Address: http: domain name of gateway /api/service/abc.
- Set `header`， `header` Parameter:

Field	Value	Description
timestamp	1571711067186	Timestamp when signing
appKey	1TEST123456781	The AK value assigned to you
sign	A90E66763793BDBC817CF3B52AAAC041	The signature obtained above
version	1.0.0	1.0.0 is a fixed value.

- The signature plugin will filter requests after 5 minutes by default
- If the authentication fails, will return code 401, message may change.

```
"code":401,"message":"sign is not pass,Please check you sign algorithm!","data
":null}
```

9.11.7 Extension

- Please refer to: dev-sign.

9.12 Rewrite Plugin

9.12.1 Explanation

- When making proxy invokes to the target service, soul Gateway also allows users to rewrite the request path using the `rewrite` plugin.

9.12.2 Plugin Setting

- In `soul-admin` –> plugin management –> `rewrite` ,set to enable.
- Introduce `rewrite` support in the pox.xml file of the gateway.
- If the user don’ t use, please disable the plugin in the background.

```
<!-- soul rewrite plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-rewrite</artifactId>
   <version>${last.version}</version>
</dependency>
<!-- soul rewrite plugin end-->
```

- Selectors and rules, please refer to: selector.
- Only those matched request will be rewritten.

9.12.3 Situation

- As the name suggests, rewrite is a redefinition of URI.
- When the request is matched, set the user-defined path, and the user-defined path will overwrite the previous real path.
- When invoking, the user-defined path will be used.

9.13 Websocket Plugin

9.13.1 Explanation

- Soul gateway also support proxy of websocket.
- In websocket support, divide plugin is used in it.

9.13.2 Plugin Setting

- In `soul-admin` –> plugin management –> `divide`, set to enable.
- Introducing dependencies in the pom.xml file of the gateway

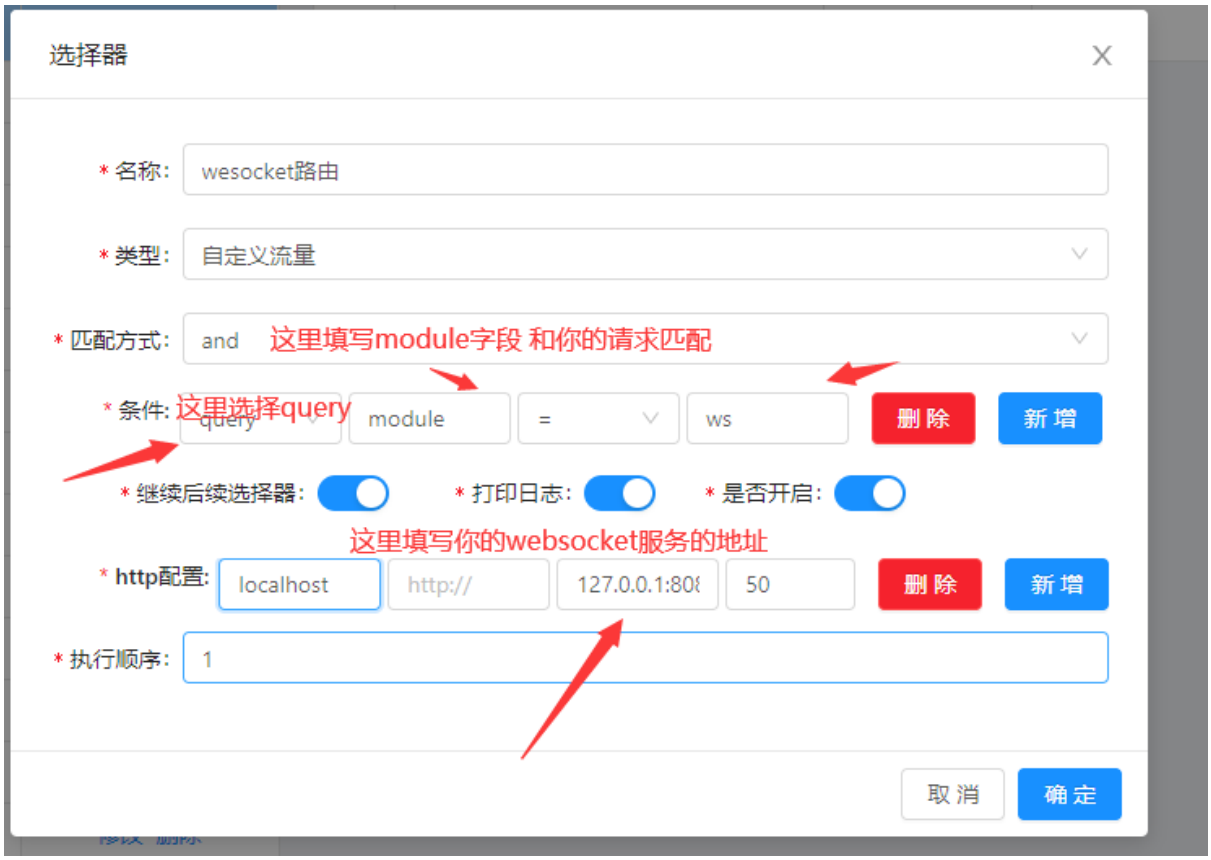
```
<!--if you use http proxy start this-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-divide</artifactId>
    <version>${last.version}</version>
</dependency>

<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-httpclient</artifactId>
    <version>${last.version}</version>
</dependency>
```

9.13.3 Request Path

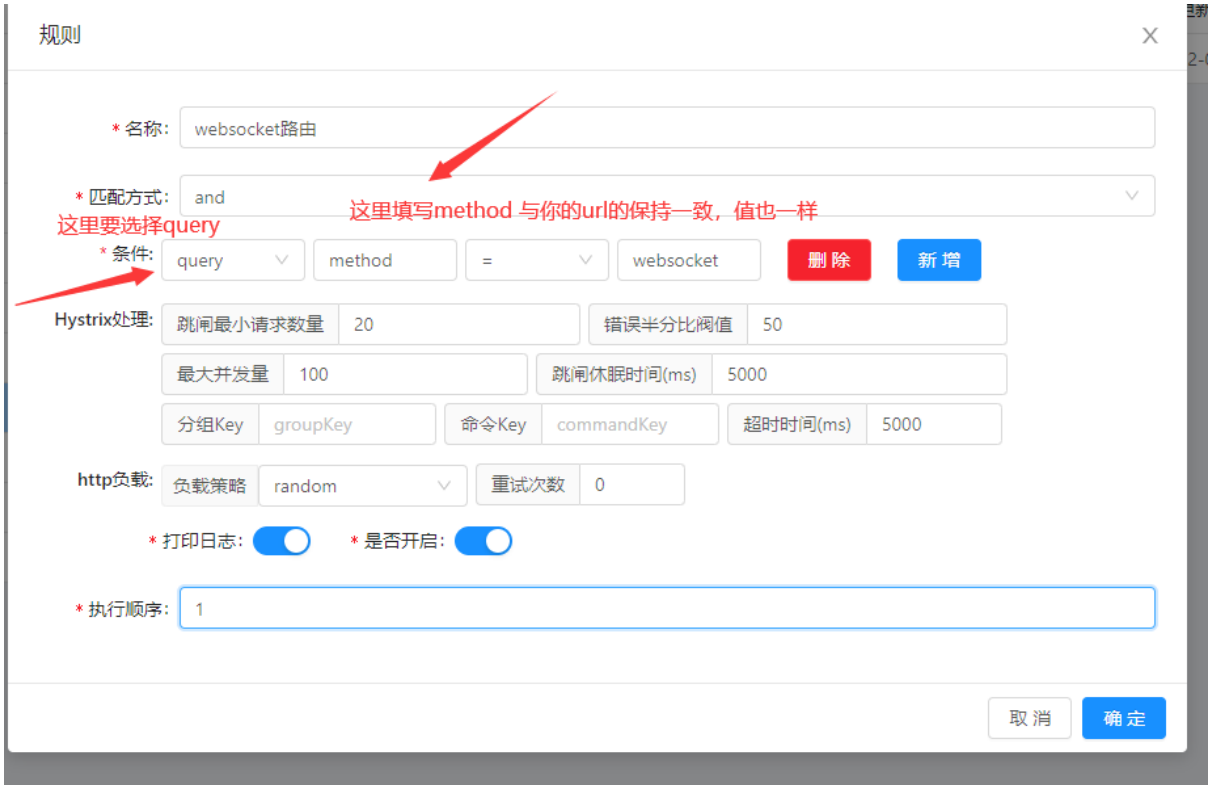
- When using soul proxy websocket, its request path is (example):`ws://localhost:9195/?module=ws&method=/websocket&rpcType=websocket`.

```
Detail:
1.localhost:8080 Is the IP and port started by soul.
2.module (Required) :Value is the key that matching selector.
3.method  (Parameter) : Your websocket path is also used as a matching rule.
4.rpcType : websocket must be filled in, and must be websocket
```

- Add a new configuration to the selector in the `divide` plugin, as follows



- Add a new rule in this selector:



- In summary, pay attention to your path at this time `ws://localhost:9195/?module=ws&method=/websocket&rpcType=websocket`.

It will be matched by your new selector rule, and then the real websocket address of the proxy is:`127.0.0.1:8080/websocket`,so that soul can proxy websocket.

You can communicate with websocket service, it is actually very simple.

- I would like to add just one last word that you can decide the name and value of module and method by yourself as long as the selector and the rule can match, this is just an example,

9.14 Plugin Context Path Mapping

9.14.1 Explanation

- When making invokes to the target service, soul Gateway also allows users to customize the context using the `context_path` plugin.

9.14.2 Plugin Setting

- In `soul-admin` –> plugin management –> `context_path`, set to enable.
- Introduce `context_path` support in the pox.xml file of the gateway.
- If the user don’ t use, please disable the plugin in the background.

```
<!-- soul context_path plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-context-path</artifactId>
   <version>${last.version}</version>
</dependency>
<!-- soul context_path plugin end-->
```

- Selectors and rules, please refer to: selector。

9.14.3 Situation

- As the name suggests,the context_path plugin redefines the contextPath of URI.
- When the request is matched, the custom `contextPath` is set, then the custom `contextPath` will be intercepted according to the requested Url to obtain the real Url, for example, the request path is `/soul/http/order`, The configured contextPath is `/soul/http`, then the requested url is `/order`.

9.15 Redirect Plugin

9.15.1 Explanation

When the soul gateway makes proxy calls to the target service, it also allows users to use the `redirect` plugin to redirect requests.

9.15.2 Plugin Setting

- In `soul-admin` –> plugin management –> `redirect` ,set to enable。
- Introduce `redirect` support in the pox.xml file of the gateway.
- If the user don’ t use, please disable the plugin in the background.
- Selectors and rules, only matching requests will be forwarded and redirected, please see: Selector rules。

9.15.3 Maven Dependency

Add the plugin dependency in the pom.xml file of the soul-bootstrap project.

```
<!-- soul redirect plugin start-->
<dependency>
    <groupId>org.dromara</groupId>
    <artifactId>soul-spring-boot-starter-plugin-redirect</artifactId>
   <version>${last.version}</version>
</dependency>
<!-- soul redirect plugin end-->
```

9.15.4 Situation

As the name suggests, the `redirect` plugin is to re-forward and redirect `uri`.

Redirect

- When we configure a custom path in `Rule`, it should be a reachable service path.
- When the request is matched, the `Soul Gateway` will perform the `308` service jump according to the customized path.

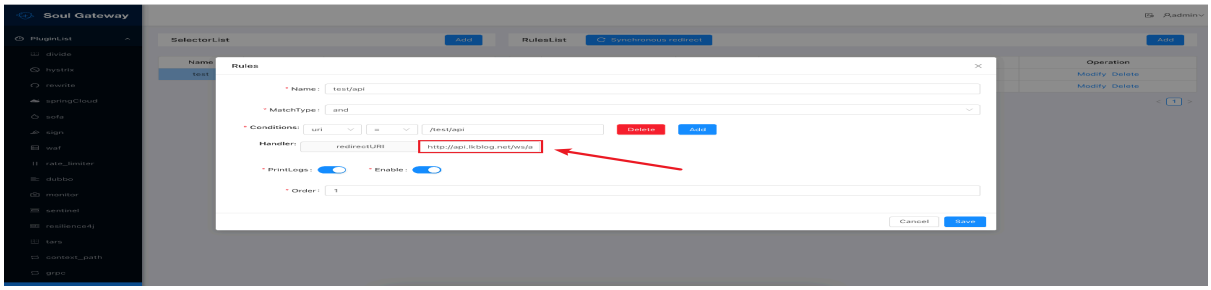


Figure1: Redirect

Gateway’ s own interface forwarding

- When the matching rules are met, the service will use the `DispatcherHandler` internal interface for forwarding.
- To implement the gateway’ s own interface forwarding, we need to use / as the prefix in the configuration path. The specific configuration is as shown in the figure below.

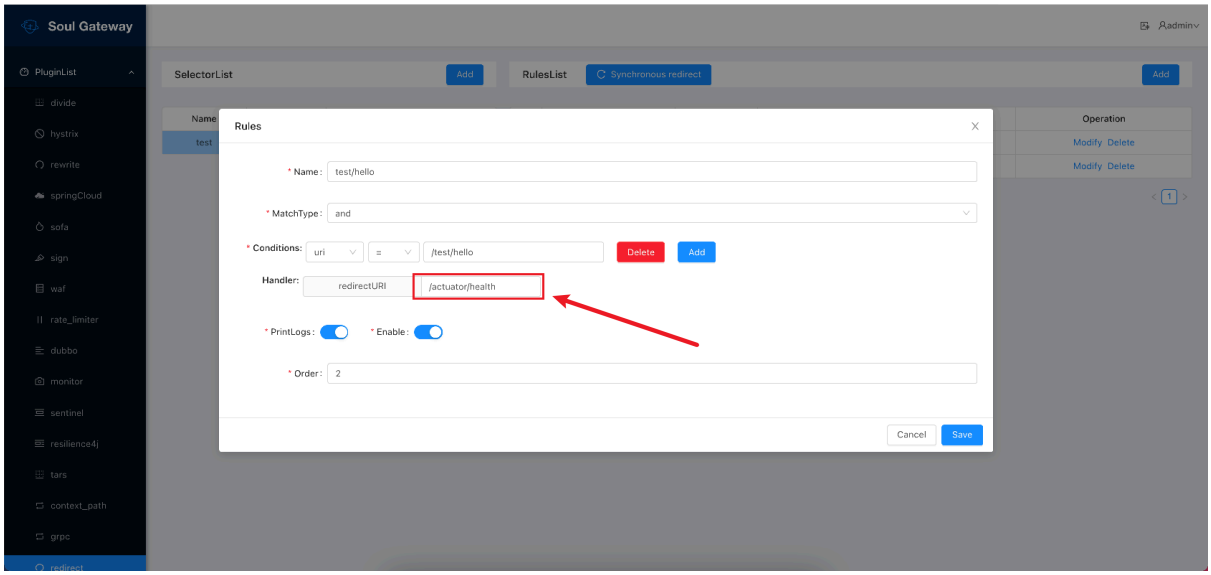


Figure2: Forwarding

10 Developer Guide

10.1 Filter Extension

10.1.1 Description

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

10.1.2 CORS Support

- `org.dromara.soul.bootstrap.cors.CrossFilter` is designed for WebFilter implementation.

```
public class CrossFilter implements WebFilter {

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

    private static final String ALLOWED_METHODS = "*";

    private static final String ALLOWED_ORIGIN = "*";

    private static final String ALLOWED_EXPOSE = "*";

    private static final String MAX_AGE = "18000";

    @Override
    @SuppressWarnings("all")
    public Mono<Void> filter(final ServerWebExchange exchange, final WebFilterChain
chain) {
        ServerHttpRequest request = exchange.getRequest();
        if (CorsUtils.isCorsRequest(request)) {
            ServerHttpResponse response = exchange.getResponse();
            HttpHeaders headers = response.getHeaders();
```

```
            headers.add("Access-Control-Allow-Origin", ALLOWED_ORIGIN);
            headers.add("Access-Control-Allow-Methods", ALLOWED_METHODS);
            headers.add("Access-Control-Max-Age", MAX_AGE);
            headers.add("Access-Control-Allow-Headers", ALLOWED_HEADERS);
            headers.add("Access-Control-Expose-Headers", ALLOWED_EXPOSE);
            headers.add("Access-Control-Allow-Credentials", "true");
            if (request.getMethod() == HttpMethod.OPTIONS) {
                response.setStatusCode(HttpStatus.OK);
                return Mono.empty();
            }
        }
        return chain.filter(exchange);
    }
}
```

- Registering CrossFilter as a Spring Bean and you are ready to go.

10.1.3 Filtering Spring Boot health check

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

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

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

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

10.1.4 Extending org.dromara.soul.web.filter.AbstractWebFilter

- Add a new class and inherit from org.dromara.soul.web.filter.AbstractWebFilter.
- Implement abstract methods of parent class.

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

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

- if method doFilter returns Mono, this filter is passing, but not vice versa. While rejecting, it will call method doDenyResponse and sending infos in response body to frontend.

10.2 Custom Plugin

10.2.1 Description

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

10.2.2 Single Responsibility Plugins

- Add following dependency:

```
<dependency>
       <groupId>org.dromara</groupId>
       <artifactId>soul-plugin-api</artifactId>
       <version>${last.version}</version>
 </dependency>
```

- Declare a new class named "A" and implements `org.dromara.soul.plugin.api.SoulPlugin`

```
public interface SoulPlugin {

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

    /**
     * return plugin order .
     * This attribute To determine the plugin execution order in the same type
plugin.
     *
     * @return int order
     */
    int getOrder();

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

    /**
     * plugin is execute.
     * if return true this plugin can not execute.
     *
```

```
     * @param exchange the current server exchange
     * @return default false.
     */
    default Boolean skip(ServerWebExchange exchange) {
        return false;
    }
}
```

Detailed instruction of interface methods:

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

```
@Bean
public SoulPlugin a() {
    return new A();
}
```

10.2.3 Matching Traffic Processing Plugin

- Introduce the following dependency:

```
<dependency>
       <groupId>org.dromara</groupId>
       <artifactId>soul-plugin-base</artifactId>
       <version>${last.version}</version>
 </dependency>
```

- Add a new class A, inherit from `org.dromara.soul.plugin.base.AbstractSoulPlugin`
- examples down below:

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

    /**
```

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

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

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

    @Override
    protected Mono<Void> doExecute(ServerWebExchange exchange, SoulPluginChain
chain, SelectorZkDTO selector, RuleZkDTO rule) {
        LOGGER.debug("..... function plugin start.............");
        /*
         * Processing after your selector matches the rule.
         * rule.getHandle() is you Customize the json string to be processed.
         * for this example.
         * Convert your custom json string pass to an entity class.
         */
        final String ruleHandle = rule.getHandle();

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

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

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

```
@Bean
public SoulPlugin a() {
    return new A();
}
```

10.2.4 Subscribe your plugin data and do customized jobs

- Declare a new class named "A" and implements `org.dromara.soul.plugin.base.handler.PluginDataHandler`

```
public interface PluginDataHandler {

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

    /**
```

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

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

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

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

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

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

}
```

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

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

10.3 File Uploading And Downloading

10.3.1 description

- This doc gives a brief description for uploading and downloading files using soul.

10.3.2 File Uploading

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

10.3.3 File Downloading

- Soul supports downloading files in streams. There is no need to change anything.

10.4 Fetching Correct IP Address And Host

10.4.1 Description

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

10.4.2 Default Implementation

- The embedded implementation in soul is :`org.dromara.soul.web.forwarde.ForwardedRemoteAddressResolver`。
- You need to config `X-Forwarded-For` in `nginx` first to get correct IP address and host.

10.4.3 Implement through a Plugin

- Declare a new class named "A" and implements `org.dromara.soul.plugin.api.RemoteAddressResolver`.

```
public interface RemoteAddressResolver {

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

}
```

- Register defined class as a Spring Bean.

```
@Bean
public SignService a() {
      return new A
}
```

10.5 Custom Response

10.5.1 Description

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

10.5.2 Default Implementation

- The default implementation class is `org.dromara.soul.plugin.api.result.DefaultSoulResult`.
- Following is the response structure.

```
public class SoulDefaultEntity implements Serializable {

    private static final long serialVersionUID = -2792556188993845048L;

    private Integer code;

    private String message;

    private Object data;

}
```

- The returned json as follows:

```
{
    "code": -100, //response code,
    "message": " 您的参数错误，请检查相关文档!", //hint messages
    "data": null  // business data
}
```

10.5.3 Extensions

- Declare a new class named “A” and implements `org.dromara.soul.plugin.api.result.SoulResult`

```
public interface SoulResult<T> {

    /**
     * Success t.
     *
     * @param code    the code
     * @param message the message
     * @param object  the object
     * @return the t
     */
    T success(int code, String message, Object object);

    /**
     * Error t.
     *
     * @param code    the code
```

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

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

```
@Bean
public SoulResult a() {
      return new A();
}
```

10.6 Custom Sign Algorithm

10.6.1 Description

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

10.6.2 Extension

- The default implementation is `org.dromara.soul.plugin.sign.service.DefaultSignService`.
- Declare a new class named "A" and implements `org.dromara.soul.plugin.api.SignService`.

```
public interface SignService {

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

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

```
@Bean
public SignService a() {
      return new A
}
```

10.7 A multilingual HTTP client

10.7.1 Description

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

10.7.2 Customize Http Client

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

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

10.8 Thread Model

10.8.1 Description

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

10.8.2 IO And Work Thread

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

10.8.3 Business Thread

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

10.8.4 Type Switching

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

10.9 Soul Optimize

10.9.1 Description

- This doc shows how to do performance optimization for soul.

10.9.2 Time Consumption

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

10.9.3 Netty Optimization

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

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

```
private static class EventLoopNettyCustomizer implements NettyServerCustomizer {

    @Override
    public HttpServer apply(final HttpServer httpServer) {
        return httpServer
                .tcpConfiguration(tcpServer -> tcpServer
                        .runOn(LoopResources.create("soul-netty", 1, DEFAULT_IO_
WORKER_COUNT, true), false)
                        .selectorOption(ChannelOption.SO_REUSEADDR, true)
                        .selectorOption(ChannelOption.ALLOCATOR,
PooledByteBufAllocator.DEFAULT)
                        .option(ChannelOption.TCP_NODELAY, true)
                        .option(ChannelOption.ALLOCATOR, PooledByteBufAllocator.
DEFAULT));
    }
}
```

- Soul-bootstrap offers this class, you may modify it when benchmarking your app if necessary.
- You can get references of business thread model from thread model

11 Community

11.1 Soul Contributor

You can report a bug, submit a new function enhancement suggestion, or submit a pull request directly.

11.1.1 Submit an Issue

- Before submitting an issue, please go through a comprehensive search to make sure the problem cannot be solved just by searching.
- Check the Issue List to make sure the problem is not repeated.
- Create a new issue and choose the type of issue.
- Define the issue with a clear and descriptive title.
- Fill in necessary information according to the template.
- Choose a label after issue created, for example: bug, enhancement, discussion.
- Please pay attention for your issue, you may need provide more information during discussion.

11.1.2 Developer Flow

Fork Soul repo

- Fork a `Soul` repo to your own repo to work, then setting upstream.

```
git remote add upstream https://github.com/dromara/soul.git
```

Choose Issue

- Please choose the issue to be edited. If it is a new issue discovered or a new function enhancement to offer, please create an issue and set the right label for it.
- After choosing the relevant issue, please reply with a deadline to indicate that you are working on it.

Create Branch

- Switch to forked master branch, pull codes from upstream, then create a new branch.

```
git checkout master
git pull upstream master
git checkout -b issueNo
```

Notice ： We will merge PR using squash, commit log will be different form upstream if you use old branch.

Coding

- Please obey the Code of Conduct during the process of development and finish the check before submitting the pull request.
- push code to your fork repo.

```
git add modified-file-names
git commit -m 'commit log'
git push origin issueNo
```

Submit Pull Request

- Send a pull request to the master branch.
- The mentor will do code review before discussing some details (including the design, the implementation and the performance) with you. The request will be merged into the branch of current development version after the edit is well enough.
- At last, congratulate to be an official contributor of Soul.

Delete Branch

- You can delete the remote branch (origin/issueNo) and the local branch (issueNo) associated with the remote branch (origin/issueNo) after the mentor merged the pull request into the master branch of Soul.

```
git checkout master
git branch -d issueNo
git push origin --delete issueNo
```

Notice: Please note that in order to show your id in the contributor list, don’t forget the configurations below:

```
git config --global user.name "username"
git config --global user.email "username@mail.com"
```

FAQ

- After each Pull Request (PR), you need to execute the following operations, otherwise, the previous PR commit records will be mixed with this PR commit records. The specific operation process is as follows:

```
git checkout master
git fetch upstream
git reset --hard upstream/master
git push -f
```

11.2 Soul Committer

11.2.1 Committer Promotion

After you have made a lot of contributions, the community will nominate. Become a committer you will have

- Permissions written by Soul repository
- Idea is used legally

11.2.2 Committer Responsibilities

- Develop new features;
- Refactor codes;
- Review pull requests reliably and in time;
- Consider and accept feature requests;
- Answer questions;
- Update documentation and example;
- Improve processes and tools;
- Guide new contributors join community.

11.2.3 Committer Routine

1. A committer needs to review every day the Pull Request and issue list to be processed by the community, and assign a suitable committer, that is, Assignee.
2. After a committer is assigned with an issue, the following work is required:

- Estimate whether it is a long-term issue. If it is, please label it as pending;
- Add issue labels, such as bug, enhancement, discussion, etc;
- Add milestone.

Notice: Whether it is a community issue or not, there must be Assignee until the issue is completed.

11.3 Soul Code Conduct

11.3.1 Development Guidelines

- Write codes with heart. Pursue clean, simplified and extremely elegant codes. Agree with concepts in <Refactoring: Improving the Design of Existing Code> and <Clean Code: A Handbook of Agile Software Craftsmanship>.
- Be familiar with codes already had, to keep consistent with the style and use.
- Highly reusable, no duplicated codes or configurations.
- Delete codes out of use in time.

11.3.2 Contributor Covenant Submitting of Conduct

- Make sure all the test cases are passed, Make sure `./mvnw clean install` can be compiled and tested successfully.
- Make sure the test coverage rate is not lower than the master branch.
- Make sure to check codes with Checkstyle. codes that violate check rules should have special reasons. Find checkstyle template from `https://github.com/dromara/soul/blob/master/script/soul_checkstyle.xml`, please use checkstyle `8.8` to run the rules.
- Careful consideration for each `pull request`; Small and frequent `pull request` with complete unit function is welcomed.
- Conform to `Contributor Covenant Code of Conduct` below.

11.3.3 Contributor Covenant Code of Conduct

- Use linux line separators.
- Keep indents (including blank lines) consistent with the previous one.
- Keep one blank line after class definition.
- No meaningless blank lines. Please extract private methods to instead of blank lines if too long method body or different logic code fragments.
- Use meaningful class, method and variable names, avoid to use abbreviate.
- Return values are named with `result`; Variables in the loop structure are named with `each`; Replace `each` with `entry` in map.
- Name property files with `Spinal Case`(a variant of `Snake Case` which uses hyphens – to separate words).
- Split codes that need to add notes with it into small methods, which are explained with method names.

- Have constants on the left and variable on the right in = and `equals` conditional expressions; Have variable on the left and constants on the right in `greater than` and `less than` conditional expressions.
- Beside using same names as input parameters and global fields in assign statement, avoid using `this` modifier.
- Design class as `final` class except abstract class for extend.
- Make nested loop structures a new method.
- Order of members definition and parameters should be consistent during classes and methods.
- Use guard clauses in priority.
- Minimize the access permission for classes and methods.
- Private method should be just next to the method in which it is used; writing private methods should be in the same as the appearance order of private methods.
- No `null` parameters or return values.
- Replace if else return and assign statement with ternary operator in priority.
- Use `LinkedList` in priority. Use `ArrayList` for use index to get element only.
- Use capacity based `Collection` such as `ArrayList`, `HashMap` must indicate initial capacity to avoid recalculate capacity.
- Use English in all the logs and javadoc.
- Include Javadoc, todo and fixme only in the comments.
- Only `public` classes and methods need javadoc, other methods, classes and override methods do not need javadoc.

11.3.4 Contributor Covenant Unit Test of Conduct

- Test codes and production codes should follow the same kind of code of conduct.
- Unit test should follow AIR (Automatic, Independent, Repeatable) principle.
  - Automatic: Unit test should run automatically, not interactively. Check test result manually and `System.out`, `log` are prohibited, use assert to check test results.
  - Independent: Call each other and sequence dependency during unit test cases are prohibited. Every test case should run independent.
  - Repeatable: Unit test case should not dependency external environment, they can run repeatable.
- Unit test should follow BCDE (Border, Correct, Design, Error) design principle.
  - Border: Border value test, test for loop border, special value and value sequence to get expect result.
  - Correct: Correct value test, test for correct value to get expect result.

  - Design: Design with production codes.
  - Error: Error value test, test for error input, exception to get expect result.
- Without particular reasons, test cases should be fully covered.
- Every test case need precised assertion.
- Environment preparation codes should be separate from test codes.
- Only those that relate to junit `Assert`, hamcrest `CoreMatchers` and `Mockito` can use static import.
- For single parameter asserts, `assertTrue`, `assertFalse`, `assertNull` and `assertNotNull` should be used.
- For multiple parameter asserts, `assertThat` should be used.
- For accurate asserts, try not to use `not`, `containsString` to make assertions.
- Actual values of test cases should be named `actualXXX`, expected values `expectedXXX`.
- Class for test case and `@Test` annotation do not need javadoc.

12 Doc Download

12.1 PDF

- English
- 中文