**开始**

**后端代码调整**

#### 1. 修改pom.xml

<!-- for Nacos rule publisher sample -->

<dependency>

<groupId>com.alibaba.csp</groupId>

<artifactId>sentinel-datasource-nacos</artifactId>

<!-- <scope>test</scope>-->

</dependency>

找到以上依赖，将scope删除或者注释掉

#### 2. 修改配置文件

找到配置文件 application.properties 文件，在末尾添加一下配置：

sentinel.nacos.enable=true

sentinel.nacos.server-addr=localhost:8848

sentinel.nacos.namespace=

sentinel.nacos.username=nacos

sentinel.nacos.password=nacos

sentinel.nacos.group-id=SENTINEL-GROUP

#### 3. 调整包结构

在com.alibaba.csp.sentinel.dashboard.rule包下创建nacos包，并在该包下根据以下结构新建包

├─auth #存放授权规则相关类

├─degrade #存放降级规则相关类

├─flow #存放限流规则相关类

├─gateway #存放网关限流规则相关类

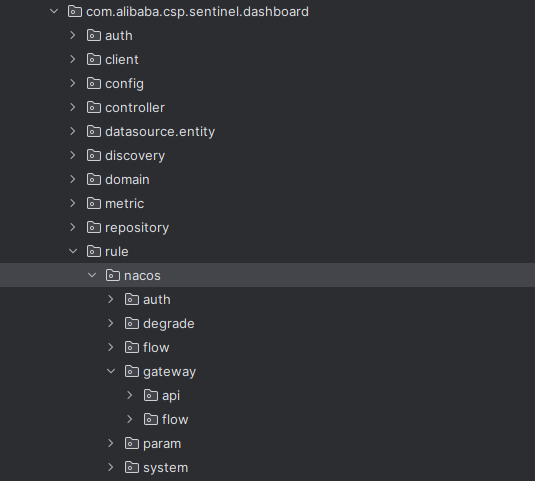
│ ├─api

│ └─flow

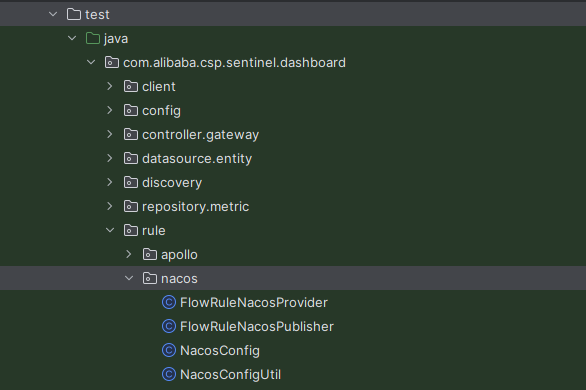
├─param #存放热点规则相关类

└─system #存放系统规则相关类

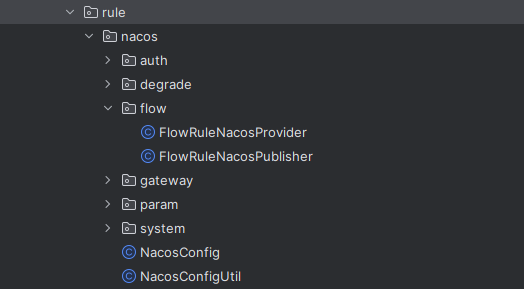
如下图：



#### 4. 增加配置类



如上图所示，将test包下的NacosConfigUtil、NacosConfig、FlowRuleNacosPublisher、FlowRuleNacosProvider复制到刚刚新建的nacos包下，如下图所示：



#### 5. 增加SentinelNacosProperties文件

在com.alibaba.csp.sentinel.dashboard.rule.nacos包下创建SentinelNacosProperties.java文件，代码如下：

package com.alibaba.csp.sentinel.dashboard.rule.nacos;

import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

@Component

@ConfigurationProperties(prefix = "sentinel.nacos")

@ConditionalOnProperty(name = "sentinel.nacos.enable", havingValue = "true")

public class SentinelNacosProperties {

private String serverAddr;

private String namespace;

private String username;

private String password;

private String groupId = "SENTINEL\_GROUP";

public String getServerAddr() {

return serverAddr;

}

public void setServerAddr(String serverAddr) {

this.serverAddr = serverAddr;

}

public String getNamespace() {

return namespace;

}

public void setNamespace(String namespace) {

this.namespace = namespace;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getGroupId() {

return groupId;

}

public void setGroupId(String groupId) {

this.groupId = groupId;

}

}

#### 6. 修改NacosConfig文件

修改后的代码如下：

/\*

\* Copyright 1999-2018 Alibaba Group Holding Ltd.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.alibaba.csp.sentinel.dashboard.rule.nacos;

import com.alibaba.csp.sentinel.dashboard.datasource.entity.gateway.ApiDefinitionEntity;

import com.alibaba.csp.sentinel.dashboard.datasource.entity.gateway.GatewayFlowRuleEntity;

import com.alibaba.csp.sentinel.dashboard.datasource.entity.rule.\*;

import com.alibaba.csp.sentinel.datasource.Converter;

import com.alibaba.fastjson.JSON;

import com.alibaba.nacos.api.PropertyKeyConst;

import com.alibaba.nacos.api.config.ConfigFactory;

import com.alibaba.nacos.api.config.ConfigService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.autoconfigure.AutoConfigureAfter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.util.List;

import java.util.Properties;

/\*\*

\* @author Eric Zhao

\* @since 1.4.0

\*/

@Configuration

public class NacosConfig {

@Bean

public Converter<List<FlowRuleEntity>, String> flowRuleEntityEncoder() {

return JSON::toJSONString;

}

@Bean

public Converter<String, List<FlowRuleEntity>> flowRuleEntityDecoder() {

return s -> JSON.parseArray(s, FlowRuleEntity.class);

}

@Bean

public Converter<List<DegradeRuleEntity>, String> degradeRuleEntityEncoder() {

return JSON::toJSONString;

}

@Bean

public Converter<String, List<DegradeRuleEntity>> degradeRuleEntityDecoder() {

return s -> JSON.parseArray(s, DegradeRuleEntity.class);

}

@Bean

public Converter<List<ParamFlowRuleEntity>, String> paramsRuleEntityEncoder() {

return JSON::toJSONString;

}

@Bean

public Converter<String, List<ParamFlowRuleEntity>> paramsRuleEntityDecoder() {

return s -> JSON.parseArray(s, ParamFlowRuleEntity.class);

}

@Bean

public Converter<List<SystemRuleEntity>, String> systemRuleEntityEncoder() {

return JSON::toJSONString;

}

@Bean

public Converter<String, List<SystemRuleEntity>> systemRuleEntityDecoder() {

return s -> JSON.parseArray(s, SystemRuleEntity.class);

}

@Bean

public Converter<List<AuthorityRuleEntity>, String> authRuleEntityEncoder() {

return JSON::toJSONString;

}

@Bean

Converter<String, List<AuthorityRuleEntity>> authRuleEntityDecoder() {

return s -> JSON.parseArray(s, AuthorityRuleEntity.class);

}

@Bean

public Converter<List<ApiDefinitionEntity>, String> apiDefinitionEntityEncoder() {

return JSON::toJSONString;

}

@Bean

public Converter<String, List<ApiDefinitionEntity>> apiDefinitionEntityDecoder() {

return s -> JSON.parseArray(s, ApiDefinitionEntity.class);

}

@Bean

public Converter<List<GatewayFlowRuleEntity>, String> gatewayFlowRuleEntityEncoder() {

return JSON::toJSONString;

}

@Bean

public Converter<String, List<GatewayFlowRuleEntity>> gatewayFlowRuleEntityDecoder() {

return s -> JSON.parseArray(s, GatewayFlowRuleEntity.class);

}

@Bean

public ConfigService nacosConfigService(SentinelNacosProperties sentinelNacosProperties) throws Exception {

Properties properties = new Properties();

properties.put(PropertyKeyConst.SERVER\_ADDR, sentinelNacosProperties.getServerAddr());

properties.put(PropertyKeyConst.NAMESPACE, sentinelNacosProperties.getNamespace());

properties.put(PropertyKeyConst.USERNAME, sentinelNacosProperties.getUsername());

properties.put(PropertyKeyConst.PASSWORD, sentinelNacosProperties.getPassword());

return ConfigFactory.createConfigService(properties);

}

}

#### 7. 修改NacosConfigUtil文件

修改后的代码如下：

package com.alibaba.csp.sentinel.dashboard.rule.nacos;

/\*\*

\* @author Eric Zhao

\* @since 1.4.0

\*/

public final class NacosConfigUtil {

public static final String GROUP\_ID = "SENTINEL\_GROUP";

public static final String FLOW\_DATA\_ID\_POSTFIX = "-flow-rules";

public static final String PARAM\_FLOW\_DATA\_ID\_POSTFIX = "-param-rules";

public static final String DEGRADE\_DATA\_ID\_POSTFIX = "-degrade-rules";

public static final String SYS\_DATA\_ID\_POSTFIX = "-system-rules";

public static final String AUTH\_DATA\_ID\_POSTFIX = "-auth-rules";

public static final String CLUSTER\_MAP\_DATA\_ID\_POSTFIX = "-cluster-map";

/\*\*

\* cc for `cluster-client`

\*/

public static final String CLIENT\_CONFIG\_DATA\_ID\_POSTFIX = "-cc-config";

/\*\*

\* cs for `cluster-server`

\*/

public static final String SERVER\_TRANSPORT\_CONFIG\_DATA\_ID\_POSTFIX = "-cs-transport-config";

public static final String SERVER\_FLOW\_CONFIG\_DATA\_ID\_POSTFIX = "-cs-flow-config";

public static final String SERVER\_NAMESPACE\_SET\_DATA\_ID\_POSTFIX = "-cs-namespace-set";

public static final String GATEWAY\_FLOW\_DATA\_ID\_POSTFIX = "-gateway-flow-rules";

public static final String GATEWAY\_API\_DATA\_ID\_POSTFIX = "-gateway-api-rules";

private NacosConfigUtil() {}

}

#### 8. 创建拉取和推送配置的类

在刚创建的auth包下创建以下两个类：

* AuthorityRuleNacosProvider 用于从nacos拉取配置
* AuthorityRuleNacosPublisher 用于控制台向nacos推送配置

参照上面拷贝的FlowRuleNacosProvider和FlowRuleNacosPublisher文件中的代码创建以上两个文件，完整的代码如下：

@Component("authRuleNacosProvider")

public class AuthorityRuleNacosProvider implements DynamicRuleProvider<List<AuthorityRuleEntity>> {

@Autowired

private ConfigService configService;

@Autowired

private Converter<String, List<AuthorityRuleEntity>> converter;

@Override

public List<AuthorityRuleEntity> getRules(String appName) throws Exception {

String rules = configService.getConfig(appName + NacosConfigUtil.AUTH\_DATA\_ID\_POSTFIX,

NacosConfigUtil.GROUP\_ID, 3000);

if (StringUtil.isEmpty(rules)) {

return new ArrayList<>();

}

return converter.convert(rules);

}

}

@Component("authRuleNacosPublisher")

public class AuthorityRuleNacosPublisher implements DynamicRulePublisher<List<AuthorityRuleEntity>> {

@Autowired

private ConfigService configService;

@Autowired

private Converter<List<AuthorityRuleEntity>, String> converter;

@Override

public void publish(String app, List<AuthorityRuleEntity> rules) throws Exception {

AssertUtil.notEmpty(app, "app name cannot be empty");

if (rules == null) {

return;

}

String convertedRule = converter.convert(rules);

configService.publishConfig(app + NacosConfigUtil.AUTH\_DATA\_ID\_POSTFIX,

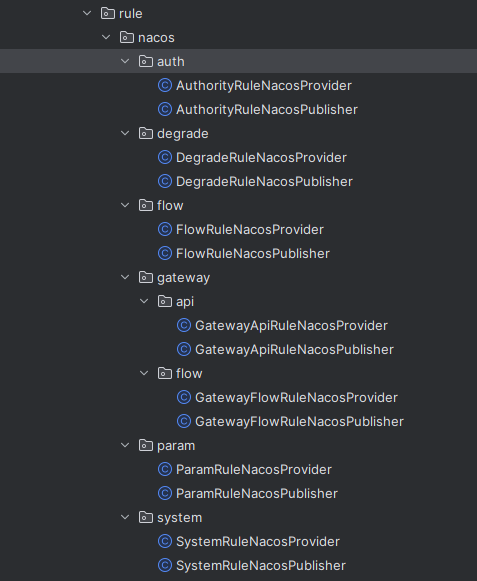
NacosConfigUtil.GROUP\_ID, convertedRule, ConfigType.JSON.getType());

}

}

依次创建降级规则、网关规则、热点规则、系统规则等文件，代码大同小异，不同点就是Converter<List<AuthorityRuleEntity>, String> converter中的entity不同，具体的可以回看NacosConfig中的配置，另一个就是data-id不同。这里就不一一罗列代码了。

最终的代码结构如下图：



#### 9. 修改FlowControllerV2

修改 com.alibaba.csp.sentinel.dashboard.controller.v2包下的FlowControllerV2，内容如下：

// @Autowired

// @Qualifier("flowRuleDefaultProvider")

// private DynamicRuleProvider<List<FlowRuleEntity>> ruleProvider;

// @Autowired

// @Qualifier("flowRuleDefaultPublisher")

// private DynamicRulePublisher<List<FlowRuleEntity>> rulePublisher;

@Autowired

@Qualifier("flowRuleNacosProvider")

private DynamicRuleProvider<List<FlowRuleEntity>> ruleProvider;

@Autowired

@Qualifier("flowRuleNacosPublisher")

private DynamicRulePublisher<List<FlowRuleEntity>> rulePublisher;

注释掉原来的ruleProvider和rulePublisher，引入我们自己创建的。

#### 10. 修改AuthorityRuleController

修改 com.alibaba.csp.sentinel.dashboard.controller包下的AuthorityRuleController，内容如下：

// 增加以下代码

@Autowired

@Qualifier("authRuleNacosProvider")

private DynamicRuleProvider<List<AuthorityRuleEntity>> ruleProvider;

@Autowired

@Qualifier("authRuleNacosPublisher")

private DynamicRulePublisher<List<AuthorityRuleEntity>> rulePublisher;

@GetMapping("/rules")

@AuthAction(PrivilegeType.READ\_RULE)

public Result<List<AuthorityRuleEntity>> apiQueryAllRulesForMachine(@RequestParam String app,

@RequestParam String ip,

@RequestParam Integer port) {

if (StringUtil.isEmpty(app)) {

return Result.ofFail(-1, "app cannot be null or empty");

}

if (StringUtil.isEmpty(ip)) {

return Result.ofFail(-1, "ip cannot be null or empty");

}

if (port == null || port <= 0) {

return Result.ofFail(-1, "Invalid parameter: port");

}

if (!appManagement.isValidMachineOfApp(app, ip)) {

return Result.ofFail(-1, "given ip does not belong to given app");

}

try {

// 注释掉下面一行代码

// List<AuthorityRuleEntity> rules = sentinelApiClient.fetchAuthorityRulesOfMachine(app, ip, port);

// 增加以下代码

List<AuthorityRuleEntity> rules = ruleProvider.getRules(app);

if (rules != null && !rules.isEmpty()) {

for (AuthorityRuleEntity entity : rules) {

entity.setApp(app);

}

}

rules = repository.saveAll(rules);

return Result.ofSuccess(rules);

} catch (Throwable throwable) {

logger.error("Error when querying authority rules", throwable);

return Result.ofFail(-1, throwable.getMessage());

}

}

private boolean publishRules(String app, String ip, Integer port) {

List<AuthorityRuleEntity> rules = repository.findAllByMachine(MachineInfo.of(app, ip, port));

// 注释掉这一行

// return sentinelApiClient.setAuthorityRuleOfMachine(app, ip, port, rules);

// 增加以下代码

try {

rulePublisher.publish(app, rules);

return true;

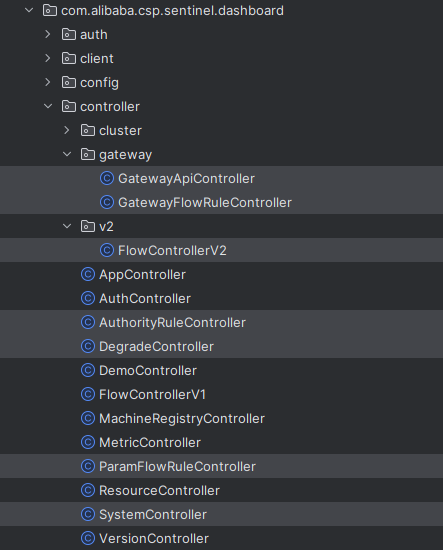
} catch (Exception e) {

return false;

}

}

需要增加以及改动的代码，已经在代码上加了注释，请仔细阅读。同理修改其他几个controller，如下图所示，改动的位置大致相同，分别注入对应的provider和publisher，请自行修改。



**前端代码修改**

#### sidebar.html

找到webapp/resources/app/scripts/directives/sidebar/sidebar.html文件，找到以下代码：

<li ui-sref-active="active" ng-if="!entry.isGateway">

<a ui-sref="dashboard.flowV1({app: entry.app})">

<i class="glyphicon glyphicon-filter"></i>&nbsp;&nbsp;流控规则</a>

</li>

修改为：

<li ui-sref-active="active" ng-if="!entry.isGateway">

<a ui-sref="dashboard.flow({app: entry.app})">

<i class="glyphicon glyphicon-filter"></i>&nbsp;&nbsp;流控规则</a>

</li>

#### identity.js

找到webapp/resources/app/scripts/controllers/identity.js文件，找到以下代码：

function saveFlowRule() {

if (!FlowService.checkRuleValid(flowRuleDialogScope.currentRule)) {

return;

}

FlowService.newRule(flowRuleDialogScope.currentRule).success(function (data) {

if (data.code === 0) {

flowRuleDialog.close();

let url = '/dashboard/flow/' + $scope.app;

$location.path(url);

} else {

alert('失败：' + data.msg);

}

}).error((data, header, config, status) => {

alert('未知错误');

});

}

修改为：

function saveFlowRule() {

if (!FlowService.checkRuleValid(flowRuleDialogScope.currentRule)) {

return;

}

FlowService.newRule(flowRuleDialogScope.currentRule).success(function (data) {

if (data.code === 0) {

flowRuleDialog.close();

let url = '/dashboard/v2/flow/' + $scope.app;

$location.path(url);

} else {

alert('失败：' + data.msg);

}

}).error((data, header, config, status) => {

alert('未知错误');

});

}

#### flow\_v2.html

找到webapp/resources/app/views/flow\_v2.html文件，找到以下代码：

<a class="btn btn-default-inverse" style="float: right; margin-right: 10px;" ui-sref="dashboard.flowV1({app: app})">

回到单机页面

</a>

将其注释掉或删掉。

至此sentinel dashboard持久化规则到nacos改造完毕，实现了双向同步。改造的过程中，省略了一部分代码，是因为几乎类似