创建父工程

Spring Cloud Alibaba 的环境在父工程中创建，微服务的各个组件作为子工程，继承父工程的环境。

Spring Boot ---》Spring Cloud ---》Spring Cloud Alibaba

pom.xml 中添加。

```xml
<dependencyManagement>
    <dependencies>
        <!-- Spring Cloud Hoxton -->
        <dependency>

 <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-
dependencies</artifactId>
            <version>Hoxton.SR3</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!-- Spring Cloud Alibaba -->
        <dependency>

 <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-
dependencies</artifactId>
            <version>2.2.1.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
```

```
</dependencyManagement>
```

1 Nacos 服务注册

解压，启动服务。

Nacos 搭建成功，接下来注册服务。

在父工程路径下创建子工程，让子工程继承父工程的环境依赖，pom.xml 中添加 nacos 发现组件。

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

application.yml 中配置

```
spring:
  cloud:
    nacos:
      discovery:
        # 指定nacos server地址
        server-addr: localhost:8848
  application:
    name: my-nacos
```

2 Nacos 服务发现与调用

pom.xml 添加 discovery，完成服务发现。

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

通过 discoveryClient 发现注册到 nacos 中的 provider 服务。

```
@RestController
public class ConsumerController {

    @Autowired
    private DiscoveryClient discoveryClient;

    @GetMapping("/instances")
    public List<ServiceInstance> instances(){
        List<ServiceInstance> provider =
discoveryClient.getInstances("provider");
        return provider;
    }

}
```

```java
@Configuration
public class ConsumerConfig {

    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

}
```

```java
@RestController
public class ConsumerController {

    @Autowired
    private DiscoveryClient discoveryClient;
    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/index")
    public String index(){
        List<ServiceInstance> provider =
discoveryClient.getInstances("provider");
        int index =
ThreadLocalRandom.current().nextInt(provider.si
ze());
        String url =
provider.get(index).getUri()+"/index";
        return "consumer随机远程调用
provier："+this.restTemplate.getForObject(url,
String.class);
    }

}
```

3 Ribbon 负载均衡

```
@Configuration
public class ConsumerConfig {

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

}
```

```
@RestController
public class ConsumerController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/index")
    public String index(){
        return "consumer远程调用
provier: "+this.restTemplate.getForObject("http:
//provider/index", String.class);
    }

}
```

> 随机

```
server:
  port: 8180
provider:
  ribbon:
    NFLoadBalancerRuleClassName:
com.netflix.loadbalancer.RandomRule
```

> Nacos 权重

```
@Slf4j
public class NacosWeightedRule extends
AbstractLoadBalancerRule {

    @Autowired
    private NacosDiscoveryProperties
nacosDiscoveryProperties;

    @Override
    public void
initWithNiwsConfig(IClientConfig iClientConfig)
{
        //读取配置文件
    }

    @Override
    public Server choose(Object o) {
        ILoadBalancer loadBalancer =
this.getLoadBalancer();
        BaseLoadBalancer baseLoadBalancer =
(BaseLoadBalancer) loadBalancer;
        //获取要请求的微服务名称
        String name =
baseLoadBalancer.getName();
```

```
        //获取服务发现的相关API
        NamingService namingService =
nacosDiscoveryProperties.namingServiceInstance(
);
        try {
            Instance instance =
namingService.selectOneHealthyInstance(name);
            log.info("选择的实例是port=
{},instance={}",instance.getPort(),instance);
            return new NacosServer(instance);
        } catch (NacosException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

```
server:
  port: 8180
provider:
  ribbon:
    NFLoadBalancerRuleClassName:
com.southwind.configuration.NacosWeightedRule
```

4 Sentinel 服务限流降级

雪崩效应

解决方案

1、设置线程超时

2、设置限流

3、熔断器 Sentinel、Hystrix

1、pom.xml 引入依赖

```xml
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-
sentinel</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
actuator</artifactId>
</dependency>
```

2、application 配置

```yaml
management:
  endpoints:
    web:
      exposure:
        include: '*'
spring:
  cloud:
    sentinel:
      transport:
        dashboard: localhost:8080
```

3、下载 Sentinel 控制台，解压，启动。

4.1 流控规则

> 直接限流

> 关联限流

> 链路限流

1、pom.xml 添加依赖

```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-core</artifactId>
    <version>1.7.1</version>
</dependency>

<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-web-
servlet</artifactId>
    <version>1.7.1</version>
</dependency>
```

2、application.yml

```
spring:
    cloud:
        sentinel:
          filter:
            enabled: false
```

3、写配置类

```
package com.southwind.configuration;
```

```
import
com.alibaba.csp.sentinel.adapter.servlet.Common
Filter;
import
org.springframework.boot.web.servlet.FilterRegi
strationBean;
import
org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configur
ation;

@Configuration
public class FilterConfiguration {

    @Bean
    public FilterRegistrationBean
registrationBean(){
        FilterRegistrationBean registrationBean
= new FilterRegistrationBean();
        registrationBean.setFilter(new
CommonFilter());
        registrationBean.addUrlPatterns("/*");

 registrationBean.addInitParameter(CommonFilter
.WEB_CONTEXT_UNIFY,"false");

 registrationBean.setName("sentinelFilter");
        return registrationBean;
    }
}
```

4、Service

```java
@Service
public class HelloService {

    @SentinelResource("test")
    public void test(){
        System.out.println("test");
    }
}
```

5、Controller

```java
@GetMapping("/test1")
public String test1(){
    this.helloService.test();
    return "test1";
}

@GetMapping("/test2")
public String test2(){
    this.helloService.test();
    return "test2";
}
```

4.2 流控效果

> 快速失败

直接抛出异常

> Warm UP

给系统一个预热的时间，预热时间段内单机阈值较低，预热时间过后单机阈值增加，预热时间内当前的单机阈值是设置的阈值的三分之一，预热时间过后单机阈值恢复设置的值。

> 排队等待

当请求调用失败之后，不会立即抛出异常，等待下一次调用，时间范围是超时时间，在时间范围内如果能请求成功则不抛出异常，如果请求则抛出异常。

4.3 降级规则

> RT

单个请求的响应时间超过阈值，则进入准降级状态，接下来 1 S 内连续 5 个请求响应时间均超过阈值，就进行降级，持续时间为时间窗口的值。

> 异常比例

每秒异常数量占通过量的比例大于阈值，就进行降级处理，持续时间为时间窗口的值。

> 异常数

1 分钟内的异常数超过阈值就进行降级处理，时间窗口的值要大于 60S，否则刚结束熔断又进入下一次熔断了。

4.4 热点规则

热点规则是流控规则的更细粒度操作，可以具体到对某个热点参数的限流，设置限流之后，如果带着限流参数的请求量超过阈值，则进行限流，时间为统计窗口时长。

必须要添加 @SentinelResource，即对资源进行流控。

```
@GetMapping("/hot")
@SentinelResource("hot")
public String hot(
        @RequestParam(value = "num1",required =
false) Integer num1,
        @RequestParam(value = "num2",required =
false) Integer num2){
    return num1+"-"+num2;
}
```

4.5 授权规则

给指定的资源设置流控应用（追加参数），可以对流控应用进行访问权限的设置，具体就是添加白名单和黑名单。

如何给请求指定流控应用，通过实现 RequestOriginParser 接口来完成，代码如下所示。

```
package com.southwind.configuration;

import
com.alibaba.csp.sentinel.adapter.servlet.callba
ck.RequestOriginParser;
import org.springframework.util.StringUtils;
import javax.servlet.http.HttpServletRequest;

public class RequestOriginParserDefinition
implements RequestOriginParser {
    @Override
    public String
parseOrigin(HttpServletRequest
httpServletRequest) {
        String name =
httpServletRequest.getParameter("name");
        if(StringUtils.isEmpty(name)){
            throw new RuntimeException("name is
null");
        }
        return name;
    }
}
```

要让 RequestOriginParserDefinition 生效，需要在配置类中进行配置。

```
package com.southwind.configuration;

import
com.alibaba.csp.sentinel.adapter.servlet.callba
ck.WebCallbackManager;
import
org.springframework.context.annotation.Configur
ation;

import javax.annotation.PostConstruct;

@Configuration
```

```
public class SentinelConfiguration {

    @PostConstruct
    public void init(){

 WebCallbackManager.setRequestOriginParser(new 
RequestOriginParserDefinition());
    }
}
```

4.6 自定义规则异常返回

创建异常处理类

```
package com.southwind.handler;

import 
com.alibaba.csp.sentinel.adapter.servlet.callba
ck.UrlBlockHandler;
import 
com.alibaba.csp.sentinel.slots.block.BlockExcep
tion;
import 
com.alibaba.csp.sentinel.slots.block.degrade.De
gradeException;
import 
com.alibaba.csp.sentinel.slots.block.flow.FlowE
xception;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class ExceptionHandler implements
UrlBlockHandler {
    @Override
    public void blocked(HttpServletRequest
httpServletRequest, HttpServletResponse
httpServletResponse, BlockException e) throws
IOException {

 httpServletResponse.setContentType("text/html;
charset=utf-8");
        String msg = null;
        if(e instanceof FlowException){
            msg = "限流";
        }else if(e instanceof DegradeException)
{
            msg = "降级";
        }

 httpServletResponse.getWriter().write(msg);
    }
}
```

进行配置。

```java
@Configuration
public class SentinelConfiguration {

    @PostConstruct
    public void init(){

 WebCallbackManager.setUrlBlockHandler(new
ExceptionHandler());
    }
}
```

5 整合 RocketMQ

5.1 安装 RocketMQ

1、传入 Linux 服务器

2、解压缩

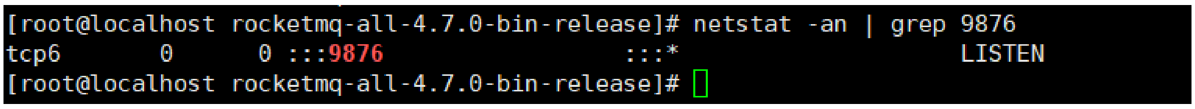
```
unzip rocketmq-all-4.7.1-bin-release.zip
```

3、启动 NameServer

```
nohup ./bin/mqnamesrv &
```

4、检查是否启动成功

```
netstat -an | grep 9876
```

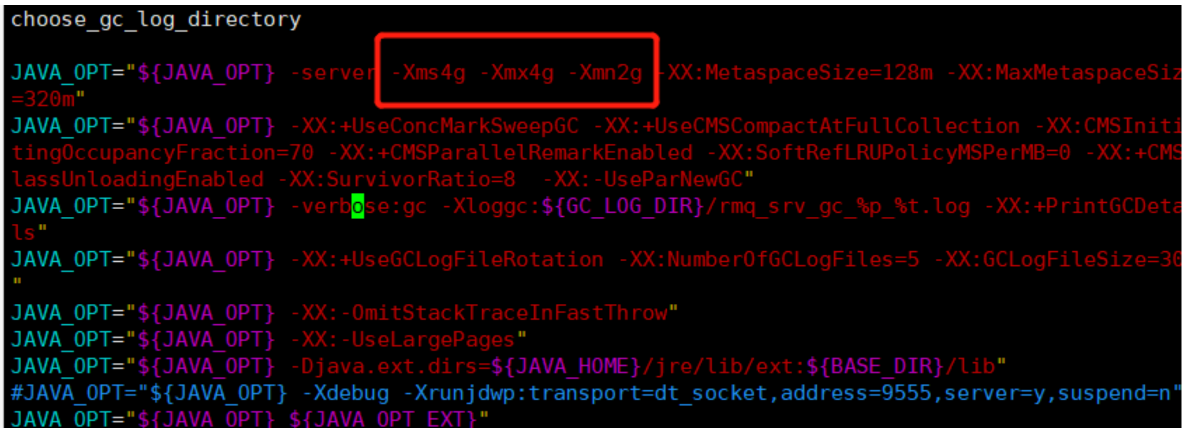


```
[root@localhost rocketmq-all-4.7.0-bin-release]# netstat -an | grep 9876
tcp6       0      0 :::9876                 :::*                    LISTEN
[root@localhost rocketmq-all-4.7.0-bin-release]#
```

5、启动 Broker

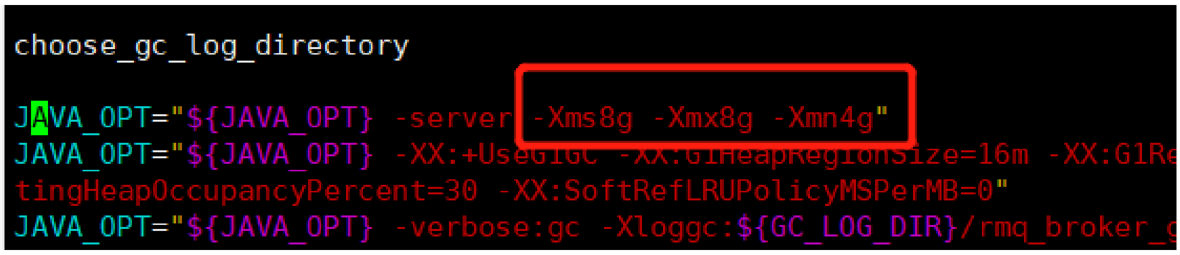
启动之前需要编辑配置文件，修改 JVM 内存设置，默认给的内存 4 GB，超过我们的 JVM 了。

```
cd bin
vim runserver.sh
```



```
choose_gc_log_directory

JAVA_OPT="${JAVA_OPT} -server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSiz
=320m"
JAVA_OPT="${JAVA_OPT} -XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -XX:CMSIniti
tingOccupancyFraction=70 -XX:+CMSParallelRemarkEnabled -XX:SoftRefLRUPolicyMSPerMB=0 -XX:+CMS
lassUnloadingEnabled -XX:SurvivorRatio=8  -XX:-UseParNewGC"
JAVA_OPT="${JAVA_OPT} -verbose:gc -Xloggc:${GC_LOG_DIR}/rmq_srv_gc_%p_%t.log -XX:+PrintGCDeta
ls"
JAVA_OPT="${JAVA_OPT} -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=30
"
JAVA_OPT="${JAVA_OPT} -XX:-OmitStackTraceInFastThrow"
JAVA_OPT="${JAVA_OPT} -XX:-UseLargePages"
JAVA_OPT="${JAVA_OPT} -Djava.ext.dirs=${JAVA_HOME}/jre/lib/ext:${BASE_DIR}/lib"
#JAVA_OPT="${JAVA_OPT} -Xdebug -Xrunjdwp:transport=dt_socket,address=9555,server=y,suspend=n"
JAVA_OPT="${JAVA_OPT} ${JAVA_OPT_EXT}"
```

```
vim runbroker.sh
```



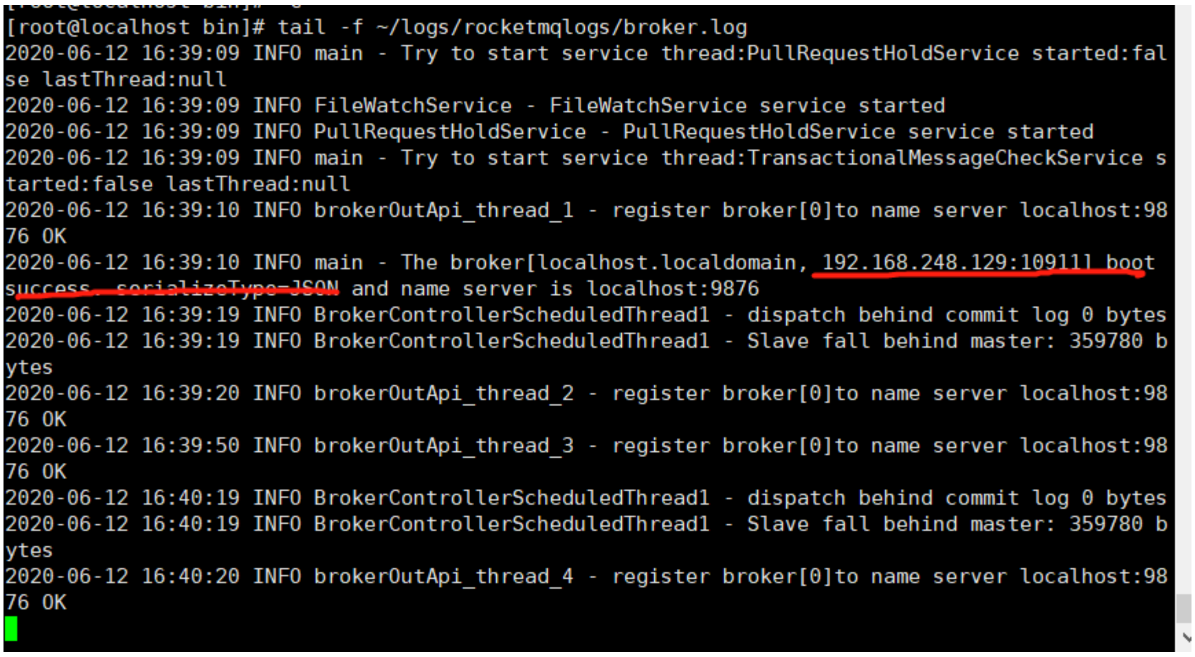
```
choose_gc_log_directory

JAVA_OPT="${JAVA_OPT} -server -Xms8g -Xmx8g -Xmn4g"
JAVA_OPT="${JAVA_OPT} -XX:+UseG1GC -XX:G1HeapRegionSize=16m -XX:G1Re
tingHeapOccupancyPercent=30 -XX:SoftRefLRUPolicyMSPerMB=0"
JAVA_OPT="${JAVA_OPT} -verbose:gc -Xloggc:${GC_LOG_DIR}/rmq_broker_g
```

启动 Broker

```
nohup ./mqbroker -n localhost:9876 &
```

可以查看日志

```
tail -f ~/logs/rocketmqlogs/broker.log
```



```
[root@localhost bin]# tail -f ~/logs/rocketmqlogs/broker.log
2020-06-12 16:39:09 INFO main - Try to start service thread:PullRequestHoldService started:fal
se lastThread:null
2020-06-12 16:39:09 INFO FileWatchService - FileWatchService service started
2020-06-12 16:39:09 INFO PullRequestHoldService - PullRequestHoldService service started
2020-06-12 16:39:09 INFO main - Try to start service thread:TransactionalMessageCheckService s
tarted:false lastThread:null
2020-06-12 16:39:10 INFO brokerOutApi_thread_1 - register broker[0]to name server localhost:98
76 OK
2020-06-12 16:39:10 INFO main - The broker[localhost.localdomain, 192.168.248.129:10911] boot
success. serializeType=JSON and name server is localhost:9876
2020-06-12 16:39:19 INFO BrokerControllerScheduledThread1 - dispatch behind commit log 0 bytes
2020-06-12 16:39:19 INFO BrokerControllerScheduledThread1 - Slave fall behind master: 359780 b
ytes
2020-06-12 16:39:20 INFO brokerOutApi_thread_2 - register broker[0]to name server localhost:98
76 OK
2020-06-12 16:39:50 INFO brokerOutApi_thread_3 - register broker[0]to name server localhost:98
76 OK
2020-06-12 16:40:19 INFO BrokerControllerScheduledThread1 - dispatch behind commit log 0 bytes
2020-06-12 16:40:19 INFO BrokerControllerScheduledThread1 - Slave fall behind master: 359780 b
ytes
2020-06-12 16:40:20 INFO brokerOutApi_thread_4 - register broker[0]to name server localhost:98
76 OK
```

启动成功

6、测试 RocketMQ

消息发送

```
cd bin
export NAMESRV_ADDR=localhost:9876
./tools.sh
org.apache.rocketmq.example.quickstart.Producer
```

消息接收

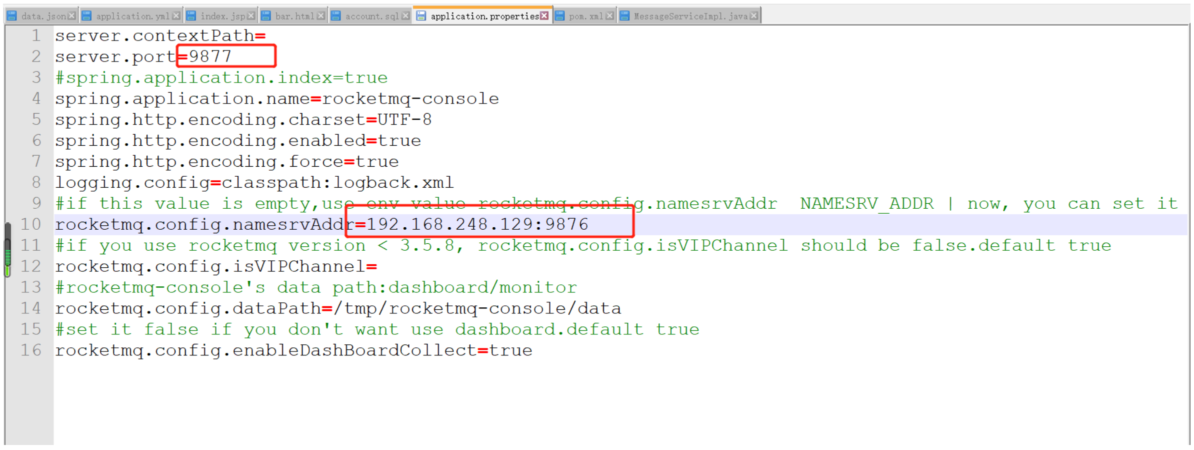
```
cd bin
export NAMESRV_ADDR=localhost:9876
./tools.sh
org.apache.rocketmq.example.quickstart.Consumer
```

7、关闭 RocketMQ

```
cd bin
./mqshutdown broker
./mqshutdown namesrv
```

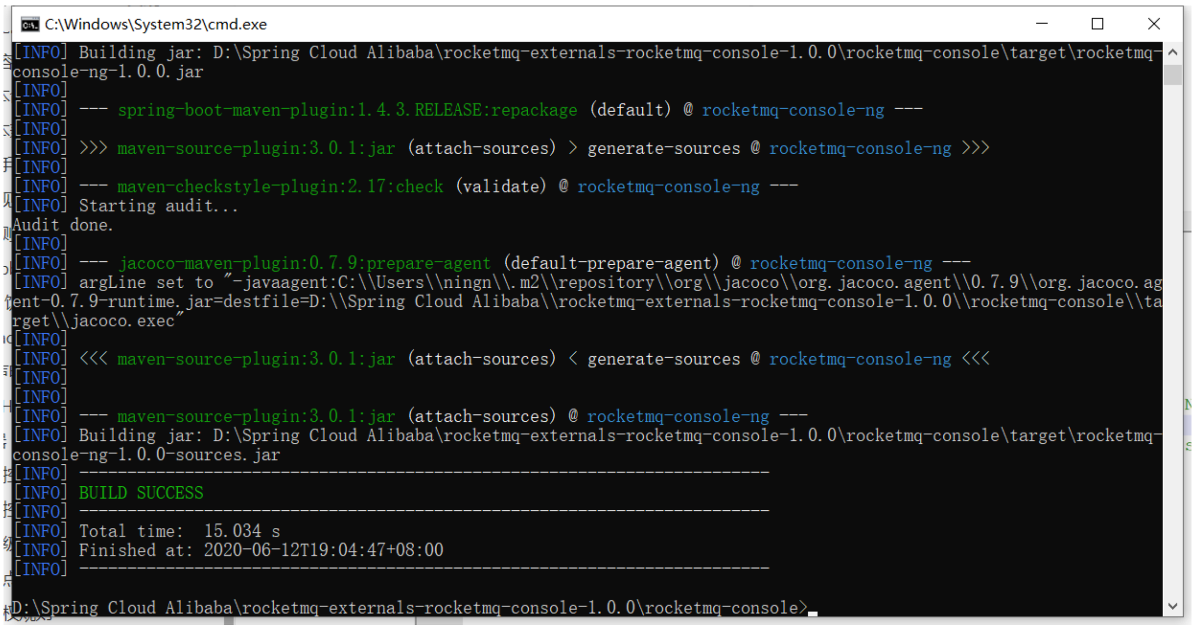
5.2 安装 RocketMQ 控制台

1、解压缩，修改配置，打包



```
server.contextPath=
server.port=9877
#spring.application.index=true
spring.application.name=rocketmq-console
spring.http.encoding.charset=UTF-8
spring.http.encoding.enabled=true
spring.http.encoding.force=true
logging.config=classpath:logback.xml
#if this value is empty,use env value rocketmq.config.namesrvAddr  NAMESRV_ADDR | now, you can set it
rocketmq.config.namesrvAddr=192.168.248.129:9876
#if you use rocketmq version < 3.5.8, rocketmq.config.isVIPChannel should be false.default true
rocketmq.config.isVIPChannel=
#rocketmq-console's data path:dashboard/monitor
rocketmq.config.dataPath=/tmp/rocketmq-console/data
#set it false if you don't want use dashboard.default true
rocketmq.config.enableDashBoardCollect=true
```

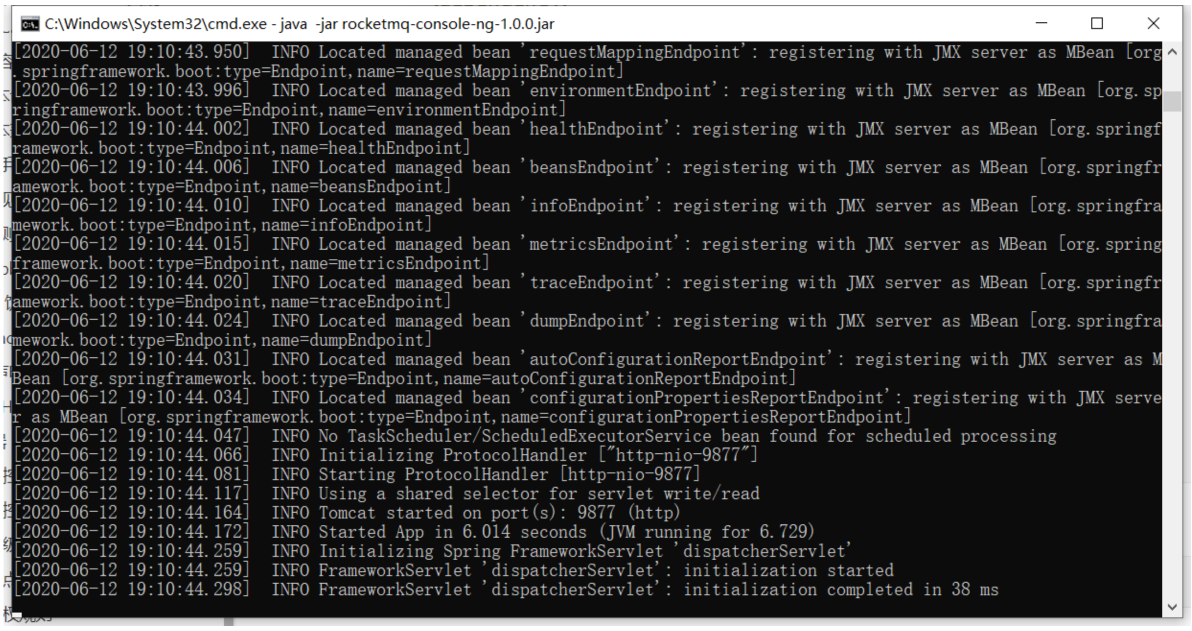
```
mvn clean package -Dmaven.test.skip=true
```



```
C:\Windows\System32\cmd.exe
[INFO] Building jar: D:\Spring Cloud Alibaba\rocketmq-externals-rocketmq-console-1.0.0\rocketmq-console\target\rocketmq-
console-ng-1.0.0.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:1.4.3.RELEASE:repackage (default) @ rocketmq-console-ng ---
[INFO]
[INFO] >>> maven-source-plugin:3.0.1:jar (attach-sources) > generate-sources @ rocketmq-console-ng >>>
[INFO]
[INFO] --- maven-checkstyle-plugin:2.17:check (validate) @ rocketmq-console-ng ---
[INFO] Starting audit...
Audit done.
[INFO]
[INFO] --- jacoco-maven-plugin:0.7.9:prepare-agent (default-prepare-agent) @ rocketmq-console-ng ---
[INFO] argLine set to "-javaagent:C:\\Users\\ningn\\.m2\\repository\\org\\jacoco\\org.jacoco.agent\\0.7.9\\org.jacoco.ag
ent-0.7.9-runtime.jar=destfile=D:\\Spring Cloud Alibaba\\rocketmq-externals-rocketmq-console-1.0.0\\rocketmq-console\\ta
rget\\jacoco.exec"
[INFO]
[INFO] <<< maven-source-plugin:3.0.1:jar (attach-sources) < generate-sources @ rocketmq-console-ng <<<
[INFO]
[INFO]
[INFO] --- maven-source-plugin:3.0.1:jar (attach-sources) @ rocketmq-console-ng ---
[INFO] Building jar: D:\Spring Cloud Alibaba\rocketmq-externals-rocketmq-console-1.0.0\rocketmq-console\target\rocketmq-
console-ng-1.0.0-sources.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  15.034 s
[INFO] Finished at: 2020-06-12T19:04:47+08:00
[INFO] ------------------------------------------------------------------------

D:\Spring Cloud Alibaba\rocketmq-externals-rocketmq-console-1.0.0\rocketmq-console>
```

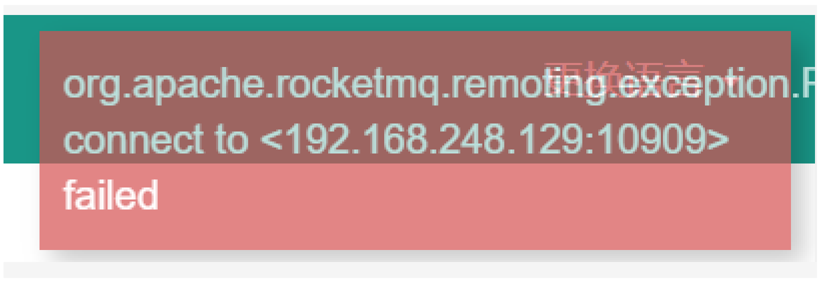
2、进入 target 启动 jar

```
java -jar rocketmq-console-ng-1.0.0.jar
```



```
C:\Windows\System32\cmd.exe - java  -jar rocketmq-console-ng-1.0.0.jar
[2020-06-12 19:10:43.950]  INFO Located managed bean 'requestMappingEndpoint': registering with JMX server as MBean [org
.springframework.boot:type=Endpoint,name=requestMappingEndpoint]
[2020-06-12 19:10:43.996]  INFO Located managed bean 'environmentEndpoint': registering with JMX server as MBean [org.sp
ringframework.boot:type=Endpoint,name=environmentEndpoint]
[2020-06-12 19:10:44.002]  INFO Located managed bean 'healthEndpoint': registering with JMX server as MBean [org.springf
ramework.boot:type=Endpoint,name=healthEndpoint]
[2020-06-12 19:10:44.006]  INFO Located managed bean 'beansEndpoint': registering with JMX server as MBean [org.springfr
amework.boot:type=Endpoint,name=beansEndpoint]
[2020-06-12 19:10:44.010]  INFO Located managed bean 'infoEndpoint': registering with JMX server as MBean [org.springfra
mework.boot:type=Endpoint,name=infoEndpoint]
[2020-06-12 19:10:44.015]  INFO Located managed bean 'metricsEndpoint': registering with JMX server as MBean [org.spring
framework.boot:type=Endpoint,name=metricsEndpoint]
[2020-06-12 19:10:44.020]  INFO Located managed bean 'traceEndpoint': registering with JMX server as MBean [org.springfr
amework.boot:type=Endpoint,name=traceEndpoint]
[2020-06-12 19:10:44.024]  INFO Located managed bean 'dumpEndpoint': registering with JMX server as MBean [org.springfra
mework.boot:type=Endpoint,name=dumpEndpoint]
[2020-06-12 19:10:44.031]  INFO Located managed bean 'autoConfigurationReportEndpoint': registering with JMX server as M
Bean [org.springframework.boot:type=Endpoint,name=autoConfigurationReportEndpoint]
[2020-06-12 19:10:44.034]  INFO Located managed bean 'configurationPropertiesReportEndpoint': registering with JMX serve
r as MBean [org.springframework.boot:type=Endpoint,name=configurationPropertiesReportEndpoint]
[2020-06-12 19:10:44.047]  INFO No TaskScheduler/ScheduledExecutorService bean found for scheduled processing
[2020-06-12 19:10:44.066]  INFO Initializing ProtocolHandler ["http-nio-9877"]
[2020-06-12 19:10:44.081]  INFO Starting ProtocolHandler [http-nio-9877]
[2020-06-12 19:10:44.117]  INFO Using a shared selector for servlet write/read
[2020-06-12 19:10:44.164]  INFO Tomcat started on port(s): 9877 (http)
[2020-06-12 19:10:44.172]  INFO Started App in 6.014 seconds (JVM running for 6.729)
[2020-06-12 19:10:44.259]  INFO Initializing Spring FrameworkServlet 'dispatcherServlet'
[2020-06-12 19:10:44.259]  INFO FrameworkServlet 'dispatcherServlet': initialization started
[2020-06-12 19:10:44.298]  INFO FrameworkServlet 'dispatcherServlet': initialization completed in 38 ms
```

打开浏览器访问 localhost:9877，如果报错

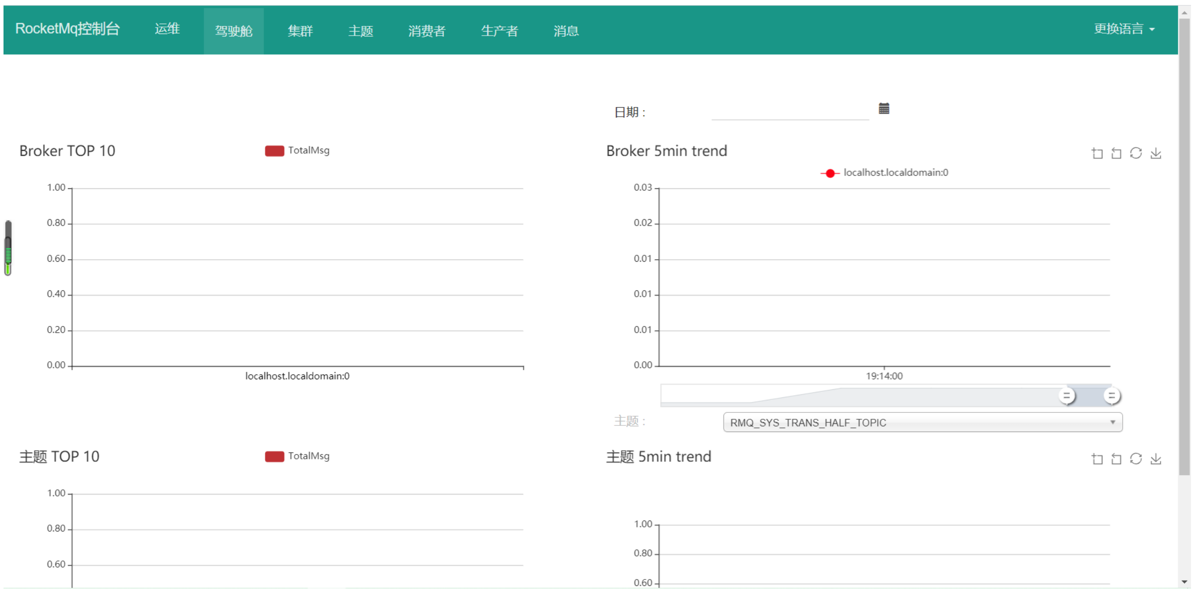


```
org.apache.rocketmq.remoting.exception.R
connect to <192.168.248.129:10909>
failed
```

这是因为我们的 RocketMQ 安装在 Linux 中，控制台在 windows，Linux 需要开放端口才能访问，开放 10909 和 9876 端口

```
firewall-cmd --zone=public --add-port=10909/tcp
--permanent
firewall-cmd --zone=public --add-port=9876/tcp
--permanent
systemctl restart firewalld.service
firewall-cmd --reload
```

重新启动控制台项目



5.3 Java 实现消息发送

1、pom.xml 中引入依赖

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-
starter</artifactId>
    <version>2.1.0</version>
</dependency>
```

2、生产消息

```
package com.southwind;

import
org.apache.rocketmq.client.producer.DefaultMQPr
oducer;
import
org.apache.rocketmq.client.producer.SendResult;
```

```
import
org.apache.rocketmq.common.message.Message;

public class Test {
    public static void main(String[] args)
throws Exception {
        //创建消息生产者
        DefaultMQProducer producer = new
DefaultMQProducer("myproducer-group");
        //设置NameServer

 producer.setNamesrvAddr("192.168.248.129:9876"
);
        //启动生产者
        producer.start();
        //构建消息对象
        Message message = new
Message("myTopic","myTag",("Test
MQ").getBytes());
        //发送消息
        SendResult result =
producer.send(message, 1000);
        System.out.println(result);
        //关闭生产者
        producer.shutdown();
    }
}
```

3、直接运行，如果报错 sendDefaultImpl call timeout，可以开放 10911 端口

```
firewall-cmd --zone=public --add-port=10911/tcp
--permanent
systemctl restart firewalld.service
firewall-cmd --reload
```

打开 RocketMQ 控制台，可查看消息。

5.4 Java 实现消息消费

```
package com.southwind.service;

import lombok.extern.slf4j.Slf4j;
import
org.apache.rocketmq.client.consumer.DefaultMQPu
shConsumer;
import
org.apache.rocketmq.client.consumer.listener.Co
nsumeConcurrentlyContext;
import
org.apache.rocketmq.client.consumer.listener.Co
nsumeConcurrentlyStatus;
import
org.apache.rocketmq.client.consumer.listener.Me
ssageListenerConcurrently;
import
org.apache.rocketmq.client.exception.MQClientEx
ception;
import
org.apache.rocketmq.common.message.MessageExt;

import java.util.List;

@Slf4j
```

```
public class ConsumerTest {
    public static void main(String[] args)
throws MQClientException {
        //创建消息消费者
        DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("myconsumer-group");
        //设置NameServer

 consumer.setNamesrvAddr("192.168.248.129:9876"
);
        //指定订阅的主题和标签
        consumer.subscribe("myTopic","*");
        //回调函数
        consumer.registerMessageListener(new
MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus
consumeMessage(List<MessageExt> list,
ConsumeConcurrentlyContext
consumeConcurrentlyContext) {
                log.info("Message=>{}",list);
                return
ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        //启动消费者
        consumer.start();
    }
}
```

5.5 Spring Boot 整合 RocketMQ

> provider

1、pom.xml

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-
starter</artifactId>
    <version>2.1.0</version>
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.7.0</version>
</dependency>
```

2、application.yml

```
rocketmq:
  name-server: 192.168.248.129:9876
  producer:
    group: myprovider
```

3、Order

```
package com.southwind.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Data
```

```
@AllArgsConstructor
@NoArgsConstructor
public class Order {
    private Integer id;
    private String buyerName;
    private String buyerTel;
    private String address;
    private Date createDate;
}
```

4、Controller

```
@Autowired
private RocketMQTemplate rocketMQTemplate;

@GetMapping("/create")
public Order create(){
    Order order = new Order(
        1,
        "张三",
        "123123",
        "软件园",
        new Date()
    );

 this.rocketMQTemplate.convertAndSend("myTopic"
,order);
    return order;
}
```

> consumer

1、pom.xml

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-
starter</artifactId>
    <version>2.1.0</version>
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.7.0</version>
</dependency>
```

2、application.yml

```
rocketmq:
  name-server: 192.168.248.129:9876
```

3、Service

```
@Slf4j
@Service
@RocketMQMessageListener(consumerGroup =
"myConsumer",topic = "myTopic")
public class SmsService implements
RocketMQListener<Order> {
    @Override
    public void onMessage(Order order) {
        log.info("新订单{},发短信",order);
    }
}
```

6 服务网关

Spring Cloud Gateway 是基于 Netty，跟 Servlet 不兼容，所以你的工程中不能出现 Servlet 的组件 。

1、pom.xml

注意，一定不能出现 spring web 的依赖，因为 Gateway 与 Servlet 不兼容。

```
<dependency>

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

2、application.yml

```
server:
  port: 8010
spring:
  application:
    name: gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
      routes:
        - id: provider_route
          uri: http://localhost:8081
          predicates:
            - Path=/provider/**
          filters:
            - StripPrefix=1
```

上面这种做法其实没有用到 nacos ，现在我们让 gateway 直接去 nacos 中发现服务，配置更加简单了。

1、pom.xml 引入 nacos

```
<dependency>


 <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-
gateway</artifactId>
</dependency>


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

2、application.yml

```
server:
  port: 8010
spring:
  application:
    name: gateway
  cloud:
      gateway:
        discovery:
          locator:
            enabled: true
```

6.1 Gateway 限流

基于路由限流

1、pom.xml

```
<dependency>

 <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-
gateway</artifactId>
</dependency>

<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-spring-cloud-gateway-
adapter</artifactId>
</dependency>
```

2、配置类

```
package com.southwind.configuration;

import
com.alibaba.csp.sentinel.adapter.gateway.common
.rule.GatewayFlowRule;
import
com.alibaba.csp.sentinel.adapter.gateway.common
.rule.GatewayRuleManager;
import
com.alibaba.csp.sentinel.adapter.gateway.sc.Sen
tinelGatewayFilter;
```

```java
import com.alibaba.csp.sentinel.adapter.gateway.sc.callback.BlockRequestHandler;
import com.alibaba.csp.sentinel.adapter.gateway.sc.callback.GatewayCallbackManager;
import com.alibaba.csp.sentinel.adapter.gateway.sc.exception.SentinelGatewayBlockExceptionHandler;
import org.springframework.beans.factory.ObjectProvider;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.codec.ServerCodecConfigurer;
import org.springframework.web.reactive.function.BodyInserters;
```

```
import
org.springframework.web.reactive.function.serve
r.ServerResponse;
import
org.springframework.web.reactive.result.view.Vi
ewResolver;
import
org.springframework.web.server.ServerWebExchang
e;
import reactor.core.publisher.Mono;

import javax.annotation.PostConstruct;
import java.util.*;

@Configuration
public class GatewayConfiguration {
    private final List<ViewResolver>
viewResolvers;
    private final ServerCodecConfigurer
serverCodecConfigurer;



    public
GatewayConfiguration(ObjectProvider<List<ViewRe
solver>> viewResolversProvider,

 ServerCodecConfigurer serverCodecConfigurer) {
        this.viewResolvers =
viewResolversProvider.getIfAvailable(Collection
s::emptyList);
        this.serverCodecConfigurer =
serverCodecConfigurer;
    }
    //配置限流的异常处理
    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public SentinelGatewayBlockExceptionHandler
sentinelGatewayBlockExceptionHandler() {
        return new
SentinelGatewayBlockExceptionHandler(viewResolv
ers, serverCodecConfigurer);
    }

    //配置初始化的限流参数
    @PostConstruct
    public void initGatewayRules(){
        Set<GatewayFlowRule> rules = new
HashSet<>();
        rules.add(
                new
GatewayFlowRule("provider_route")
                .setCount(1)
                .setIntervalSec(1)
        );
        GatewayRuleManager.loadRules(rules);
    }

    //初始化限流过滤器
    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public GlobalFilter sentinelGatewayFilter()
{
        return new SentinelGatewayFilter();
    }
    //自定义限流异常页面
    @PostConstruct
    public void initBlockHandlers(){
        BlockRequestHandler blockRequestHandler
= new BlockRequestHandler() {
            @Override
            public Mono<ServerResponse>
handleRequest(ServerWebExchange
serverWebExchange, Throwable throwable) {
                Map map = new HashMap();
                map.put("code",0);
                map.put("msg","被限流了");
                return
ServerResponse.status(HttpStatus.OK)

.contentType(MediaType.APPLICATION_JSON)

.body(BodyInserters.fromObject(map));
            }
        };

 GatewayCallbackManager.setBlockHandler(blockRe
questHandler);
    }
}
```

3、application.yml

```
server:
  port: 8010
spring:
  application:
    name: gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
      routes:
        - id: provider_route
          uri: http://localhost:8081
          predicates:
            - Path=/provider/**
          filters:
            - StripPrefix=1
```

基于 API 分组限流

1、修改配置类，添加基于 API 分组限流的方法，修改初始化的限流参数

```
package com.southwind.configuration;

import
com.alibaba.csp.sentinel.adapter.gateway.common
.SentinelGatewayConstants;
import
com.alibaba.csp.sentinel.adapter.gateway.common
.api.ApiDefinition;
import
com.alibaba.csp.sentinel.adapter.gateway.common
.api.ApiPathPredicateItem;
import
com.alibaba.csp.sentinel.adapter.gateway.common
.api.ApiPredicateItem;
```

```
import com.alibaba.csp.sentinel.adapter.gateway.common.api.GatewayApiDefinitionManager;
import com.alibaba.csp.sentinel.adapter.gateway.common.rule.GatewayFlowRule;
import com.alibaba.csp.sentinel.adapter.gateway.common.rule.GatewayRuleManager;
import com.alibaba.csp.sentinel.adapter.gateway.sc.SentinelGatewayFilter;
import com.alibaba.csp.sentinel.adapter.gateway.sc.callback.BlockRequestHandler;
import com.alibaba.csp.sentinel.adapter.gateway.sc.callback.GatewayCallbackManager;
import com.alibaba.csp.sentinel.adapter.gateway.sc.exception.SentinelGatewayBlockExceptionHandler;
import org.springframework.beans.factory.ObjectProvider;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
```

```java
import
org.springframework.core.annotation.Order;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import
org.springframework.http.codec.ServerCodecConfi
gurer;
import
org.springframework.web.reactive.function.BodyI
nserters;
import
org.springframework.web.reactive.function.serve
r.ServerResponse;
import
org.springframework.web.reactive.result.view.Vi
ewResolver;
import
org.springframework.web.server.ServerWebExchang
e;
import reactor.core.publisher.Mono;

import javax.annotation.PostConstruct;
import java.util.*;

@Configuration
public class GatewayConfiguration {

    private final List<ViewResolver>
viewResolvers;
    private final ServerCodecConfigurer
serverCodecConfigurer;
```

```
    public
GatewayConfiguration(ObjectProvider<List<ViewResolver>> viewResolversProvider,

 ServerCodecConfigurer serverCodecConfigurer) {
        this.viewResolvers =
viewResolversProvider.getIfAvailable(Collections::emptyList);
        this.serverCodecConfigurer =
serverCodecConfigurer;
    }

    //配置限流的异常处理
    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public SentinelGatewayBlockExceptionHandler
sentinelGatewayBlockExceptionHandler() {
        return new
SentinelGatewayBlockExceptionHandler(viewResolvers, serverCodecConfigurer);
    }

    //配置初始化的限流参数
    @PostConstruct
    public void initGatewayRules(){
        Set<GatewayFlowRule> rules = new
HashSet<>();
        rules.add(new
GatewayFlowRule("provider_api1").setCount(1).setIntervalSec(1));
        rules.add(new
GatewayFlowRule("provider_api2").setCount(1).setIntervalSec(1));
```

```
        GatewayRuleManager.loadRules(rules);
    }

    //初始化限流过滤器
    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public GlobalFilter sentinelGatewayFilter()
{
        return new SentinelGatewayFilter();
    }

    //自定义限流异常页面
    @PostConstruct
    public void initBlockHandlers(){
        BlockRequestHandler blockRequestHandler
= new BlockRequestHandler() {
            @Override
            public Mono<ServerResponse>
handleRequest(ServerWebExchange
serverWebExchange, Throwable throwable) {
                Map map = new HashMap();
                map.put("code",0);
                map.put("msg","被限流了");
                return
ServerResponse.status(HttpStatus.OK)

.contentType(MediaType.APPLICATION_JSON)

.body(BodyInserters.fromObject(map));
            }
        };
```

```
 GatewayCallbackManager.setBlockHandler(blockRe
questHandler);
    }

    //自定义API分组
    @PostConstruct
    private void initCustomizedApis(){
        Set<ApiDefinition> definitions = new
HashSet<>();
        ApiDefinition api1 = new
ApiDefinition("provider_api1")
                .setPredicateItems(new
HashSet<ApiPredicateItem>(){{
                    add(new
ApiPathPredicateItem().setPattern("/provider/ap
i1/**")

.setMatchStrategy(SentinelGatewayConstants.URL_
MATCH_STRATEGY_PREFIX));
                }});
        ApiDefinition api2 = new
ApiDefinition("provider_api2")
                .setPredicateItems(new
HashSet<ApiPredicateItem>(){{
                    add(new
ApiPathPredicateItem().setPattern("/provider/ap
i2/demo1"));
                }});
        definitions.add(api1);
        definitions.add(api2);
```

```
  GatewayApiDefinitionManager.loadApiDefinitions
(definitions);
    }
}
```

2、Controller 添加方法

```
@GetMapping("/api1/demo1")
public String demo1(){
    return "demo";
}

@GetMapping("/api1/demo2")
public String demo2(){
    return "demo";
}

@GetMapping("/api2/demo1")
public String demo3(){
    return "demo";
}

@GetMapping("/api2/demo2")
public String demo4(){
    return "demo";
}
```

也可以基于 Nacos 服务发现组件进行限流

```yaml
server:
  port: 8010
spring:
  application:
    name: gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
```

API 分组代码修改，改为 discovery 中的服务名。

```java
ApiDefinition api2 = new
ApiDefinition("provider_api2")
        .setPredicateItems(new
HashSet<ApiPredicateItem>(){{
            add(new
ApiPathPredicateItem().setPattern("/p1/api2/dem
o1"));
        }});
```

7 分布式事务

7.1 模拟分布式事务异常

1、创建两个工程 order、pay，pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

2、建两个数据库 order、pay，两个微服务分别访问。

3、分别写两个服务的 application.yml

```yaml
server:
  port: 8010
spring:
  application:
    name: order
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: 123456
    url: jdbc:mysql://localhost:3306/order
```

```yaml
server:
  port: 8020
spring:
  application:
    name: pay
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: 123456
    url: jdbc:mysql://localhost:3306/pay
```

4、分别写两个 Service

```java
package com.southwind.service;

import
org.springframework.beans.factory.annotation.Au
towired;
import
org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;
```

```java
@Service
public class OrderService {
    @Autowired
    private JdbcTemplate jdbcTemplate;

    public void save(){
        this.jdbcTemplate.update("insert into 
orders(username) values ('张三')");
    }
}
```

```java
package com.southwind.service;

import 
org.springframework.beans.factory.annotation.Au
towired;
import 
org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;

@Service
public class PayService {
    @Autowired
    private JdbcTemplate jdbcTemplate;

    public void save(){
        this.jdbcTemplate.update("insert into 
pay(username) values ('张三')");
    }
}
```

5、控制器 Order 通过 RestTemplate 调用 Pay 的服务

```java
package com.southwind.controller;

import com.southwind.service.OrderService;
import
org.springframework.beans.factory.annotation.Au
towired;
import
org.springframework.web.bind.annotation.GetMapp
ing;
import
org.springframework.web.bind.annotation.RestCon
troller;
import
org.springframework.web.client.RestTemplate;

@RestController
public class OrderController {

    @Autowired
    private OrderService orderService;
    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/save")
    public String save(){
        //订单
        this.orderService.save();
        int i = 10/0;
        //支付

 this.restTemplate.getForObject("http://localho
st:8020/save",String.class);
        return "success";
    }
}
```

```
package com.southwind.controller;

import com.southwind.service.PayService;
import
org.springframework.beans.factory.annotation.Au
towired;
import
org.springframework.web.bind.annotation.GetMapp
ing;
import
org.springframework.web.bind.annotation.RestCon
troller;

@RestController
public class PayController {
    @Autowired
    private PayService payService;

    @GetMapping("/save")
    public String save(){
        this.payService.save();
        return "success";
    }
}
```

6、启动类

```
package com.southwind;
```

```
import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBo
otApplication;
import
org.springframework.context.annotation.Bean;
import
org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class OrderApplication {

    public static void main(String[] args) {

 SpringApplication.run(OrderApplication.class,
args);
    }

    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```

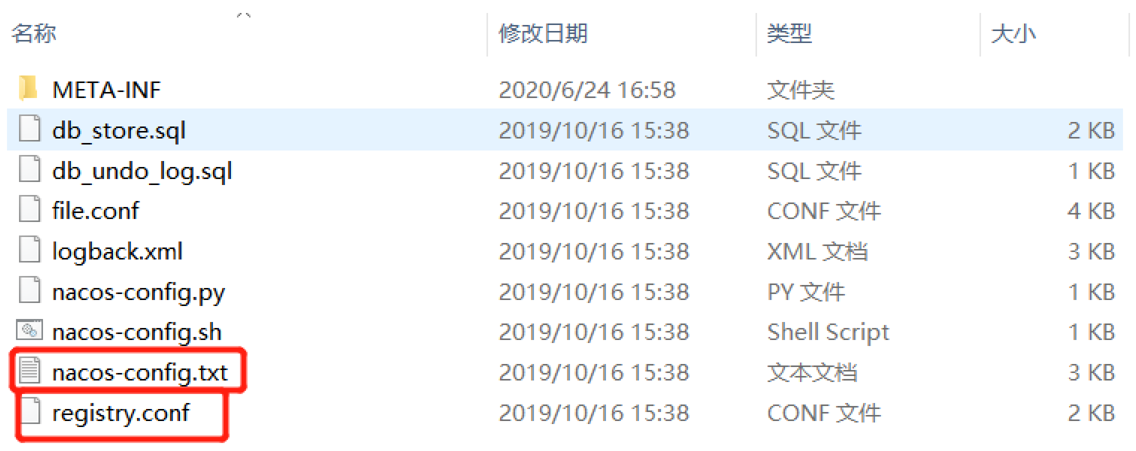
```
package com.southwind;

import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBo
otApplication;

@SpringBootApplication
public class PayApplication {

    public static void main(String[] args) {

 SpringApplication.run(PayApplication.class,
args);
    }

}
```

分布式异常模拟结束，Order 存储完成之后，出现异常，会导致 Pay 无法存储，但是 Order 数据库不会进行回滚。

7.2 Seata 解决

1、下载

2、解压，修改两个文件

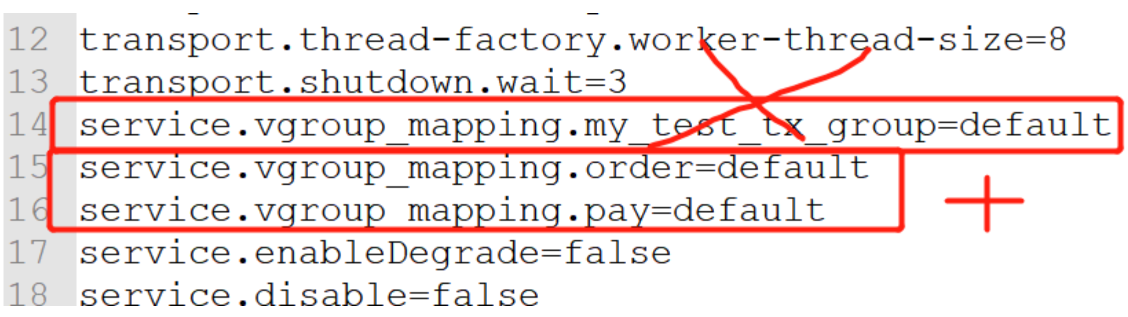


名称	修改日期	类型	大小
META-INF	2020/6/24 16:58	文件夹	
db_store.sql	2019/10/16 15:38	SQL 文件	2 KB
db_undo_log.sql	2019/10/16 15:38	SQL 文件	1 KB
file.conf	2019/10/16 15:38	CONF 文件	4 KB
logback.xml	2019/10/16 15:38	XML 文档	3 KB
nacos-config.py	2019/10/16 15:38	PY 文件	1 KB
nacos-config.sh	2019/10/16 15:38	Shell Script	1 KB
nacos-config.txt	2019/10/16 15:38	文本文档	3 KB
registry.conf	2019/10/16 15:38	CONF 文件	2 KB

regisry.conf

```
registry {
  type = "nacos"
  nacos {
    serverAddr = "localhost"
    namespace = "public"
    cluster = "default"
  }
}

config {
  type = "nacos"
  nacos {
    serverAddr = "localhost"
    namespace = "public"
    cluster = "default"
  }
}
```

nacos-config.txt



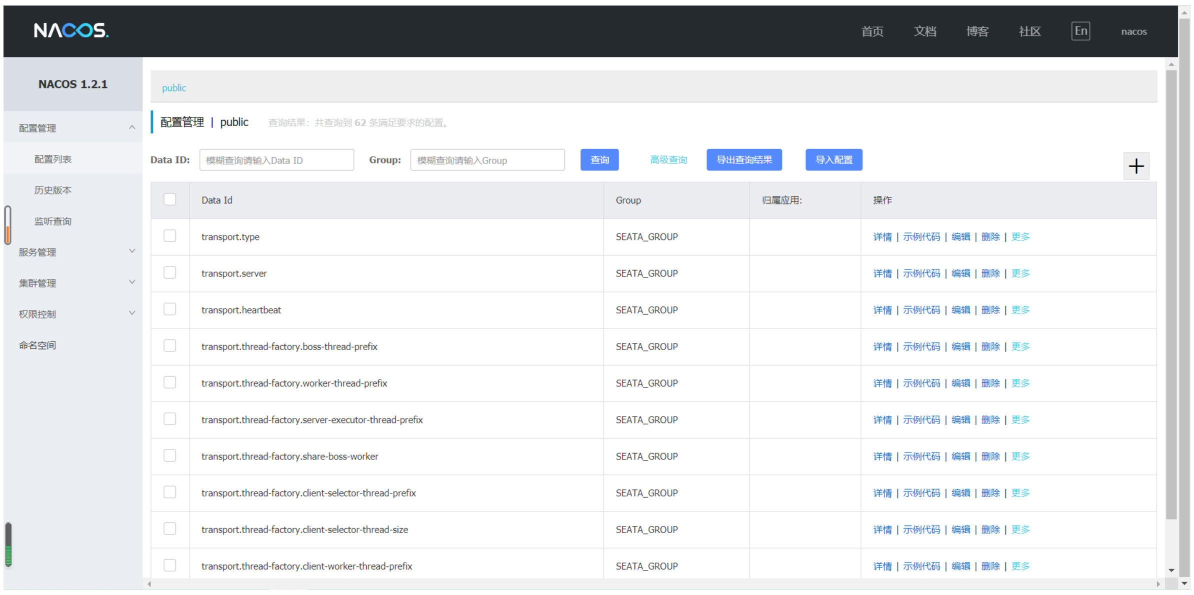
```
transport.thread-factory.worker-thread-size=8
transport.shutdown.wait=3
service.vgroup_mapping.my_test_tx_group=default
service.vgroup_mapping.order=default
service.vgroup_mapping.pay=default
service.enableDegrade=false
service.disable=false
```

3、启动 Nacos，运行 nacos-config.sh 将 Seata 配置导入 Nacos

进入 conf，右键 Git Bash Here

```
cd conf
sh nacos-config.sh 127.0.0.1
```

执行成功，刷新 Nacos，配置加入

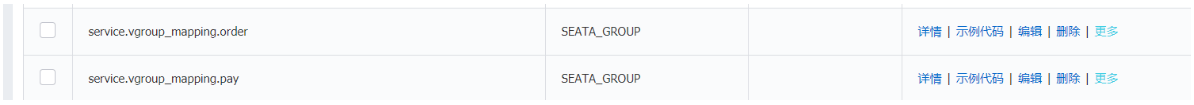


配置管理 | public 查询结果：共查询到 62 条满足要求的配置。

Data ID: 模糊查询请输入Data ID Group: 模糊查询请输入Group

	Data Id	Group	归属应用:	操作
☐	transport.type	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.server	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.heartbeat	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.boss-thread-prefix	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.worker-thread-prefix	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.server-executor-thread-prefix	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.share-boss-worker	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.client-selector-thread-prefix	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.client-selector-thread-size	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	transport.thread-factory.client-worker-thread-prefix	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多

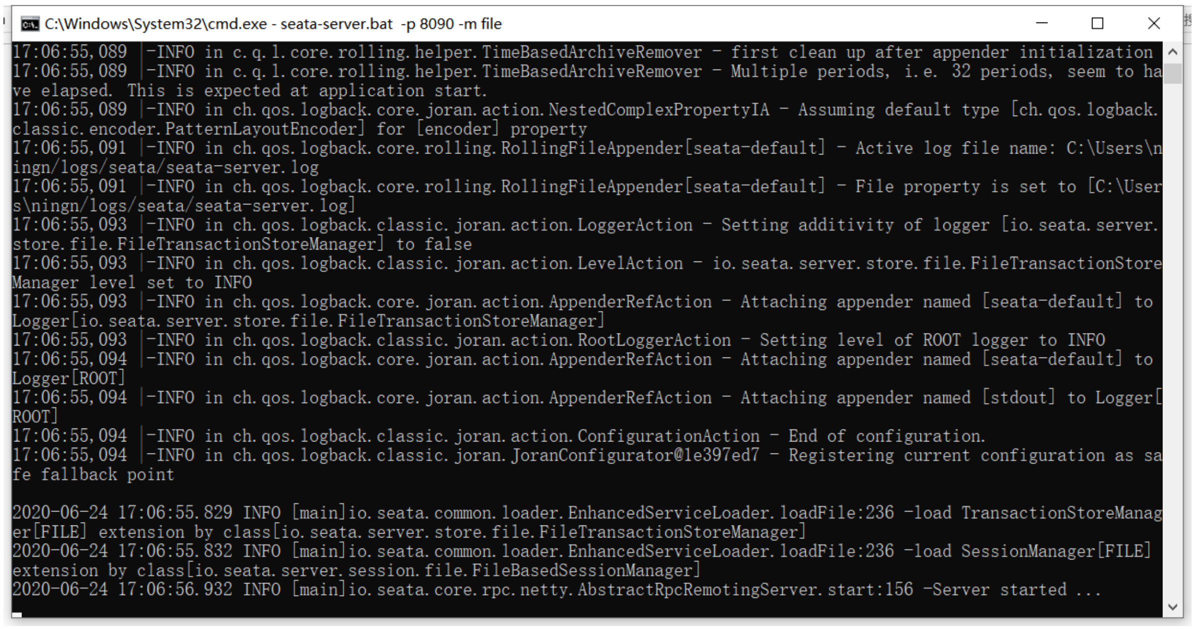
nacos-config.txt 配置已生效



☐	service.vgroup_mapping.order	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多
☐	service.vgroup_mapping.pay	SEATA_GROUP		详情 | 示例代码 | 编辑 | 删除 | 更多

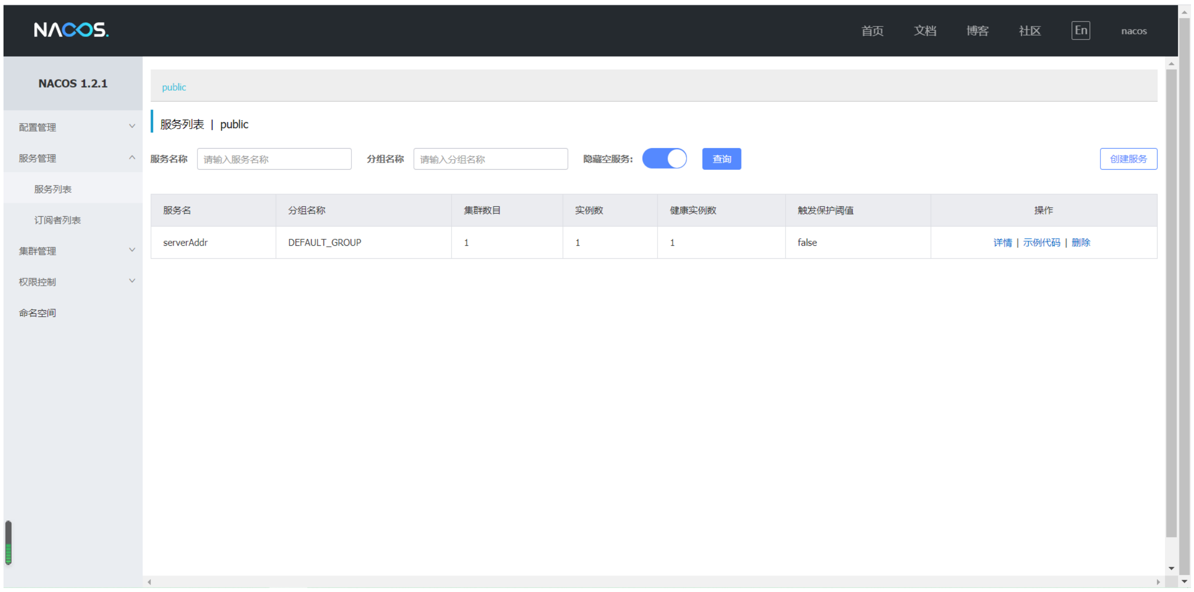
4、启动 Seata Server， **JDK 8 以上环境无法启动**

```
cd bin
seata-server.bat -p 8090 -m file
```



```
C:\Windows\System32\cmd.exe - seata-server.bat  -p 8090 -m file
17:06:55,089 |-INFO in c.q.l.core.rolling.helper.TimeBasedArchiveRemover - first clean up after appender initialization
17:06:55,089 |-INFO in c.q.l.core.rolling.helper.TimeBasedArchiveRemover - Multiple periods, i.e. 32 periods, seem to ha
ve elapsed. This is expected at application start.
17:06:55,089 |-INFO in ch.qos.logback.core.joran.action.NestedComplexPropertyIA - Assuming default type [ch.qos.logback.
classic.encoder.PatternLayoutEncoder] for [encoder] property
17:06:55,091 |-INFO in ch.qos.logback.core.rolling.RollingFileAppender[seata-default] - Active log file name: C:\Users\n
ingn/logs/seata/seata-server.log
17:06:55,091 |-INFO in ch.qos.logback.core.rolling.RollingFileAppender[seata-default] - File property is set to [C:\User
s\ningn/logs/seata/seata-server.log]
17:06:55,093 |-INFO in ch.qos.logback.classic.joran.action.LoggerAction - Setting additivity of logger [io.seata.server.
store.file.FileTransactionStoreManager] to false
17:06:55,093 |-INFO in ch.qos.logback.classic.joran.action.LevelAction - io.seata.server.store.file.FileTransactionStore
Manager level set to INFO
17:06:55,093 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appender named [seata-default] to
Logger[io.seata.server.store.file.FileTransactionStoreManager]
17:06:55,093 |-INFO in ch.qos.logback.classic.joran.action.RootLoggerAction - Setting level of ROOT logger to INFO
17:06:55,094 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appender named [seata-default] to
Logger[ROOT]
17:06:55,094 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appender named [stdout] to Logger[
ROOT]
17:06:55,094 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - End of configuration.
17:06:55,094 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator@1e397ed7 - Registering current configuration as sa
fe fallback point

2020-06-24 17:06:55.829 INFO [main]io.seata.common.loader.EnhancedServiceLoader.loadFile:236 -load TransactionStoreManag
er[FILE] extension by class[io.seata.server.store.file.FileTransactionStoreManager]
2020-06-24 17:06:55.832 INFO [main]io.seata.common.loader.EnhancedServiceLoader.loadFile:236 -load SessionManager[FILE]
extension by class[io.seata.server.session.file.FileBasedSessionManager]
2020-06-24 17:06:56.932 INFO [main]io.seata.core.rpc.netty.AbstractRpcRemotingServer.start:156 -Server started ...
```

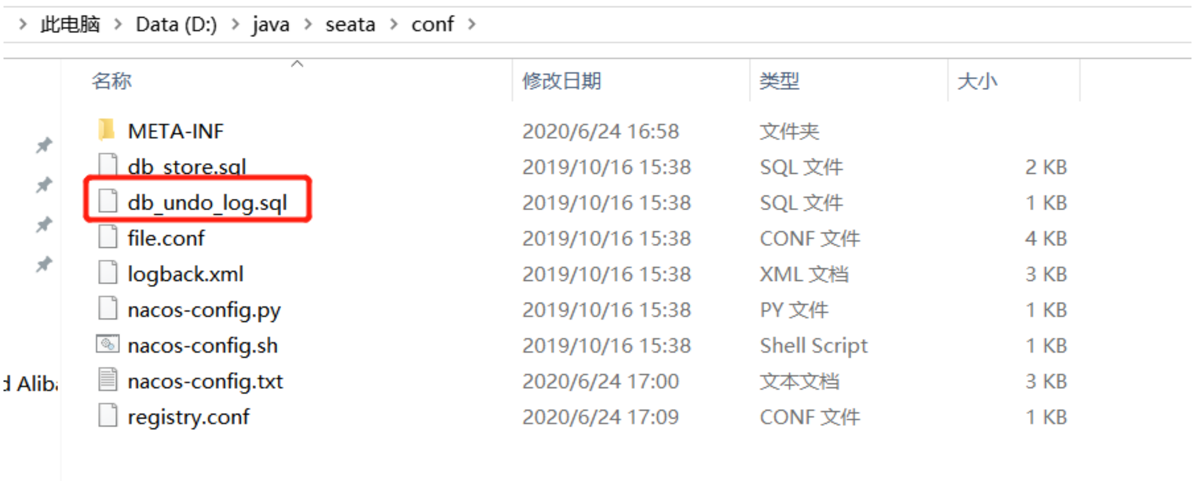
启动成功，Nacos 注册成功。



服务名	分组名称	集群数目	实例数	健康实例数	触发保护阈值	操作
serverAddr	DEFAULT_GROUP	1	1	1	false	详情 | 示例代码 | 删除

Seata 服务环境搭建完毕，接下来去应用中添加。

1、初始化数据库，在两个数据库中添加事务日志记录表，SQL Seata 已经提供。



名称	修改日期	类型	大小
META-INF	2020/6/24 16:58	文件夹	
db_store.sql	2019/10/16 15:38	SQL 文件	2 KB
db_undo_log.sql	2019/10/16 15:38	SQL 文件	1 KB
file.conf	2019/10/16 15:38	CONF 文件	4 KB
logback.xml	2019/10/16 15:38	XML 文档	3 KB
nacos-config.py	2019/10/16 15:38	PY 文件	1 KB
nacos-config.sh	2019/10/16 15:38	Shell Script	1 KB
nacos-config.txt	2020/6/24 17:00	文本文档	3 KB
registry.conf	2020/6/24 17:09	CONF 文件	1 KB

2、直接在两个数据库运行脚本。

```
CREATE TABLE `undo_log` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `branch_id` bigint(20) NOT NULL,
  `xid` varchar(100) NOT NULL,
  `context` varchar(128) NOT NULL,
  `rollback_info` longblob NOT NULL,
  `log_status` int(11) NOT NULL,
  `log_created` datetime NOT NULL,
  `log_modified` datetime NOT NULL,
  `ext` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT
CHARSET=utf8;
```

3、两个工程的 pom.xml 添加 Seata 组件和 Nacos Config 组件。

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-
seata</artifactId>
    <version>2.1.1.RELEASE</version>
</dependency>

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

4、给 JDBCTemplate 添加代理数据源

```
package com.southwind;

import io.seata.rm.datasource.DataSourceProxy;
import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBo
otApplication;
import
org.springframework.context.annotation.Bean;
import
org.springframework.jdbc.core.JdbcTemplate;
import
org.springframework.web.client.RestTemplate;

import javax.sql.DataSource;

@SpringBootApplication
```

```java
public class OrderApplication {

    public static void main(String[] args) {

 SpringApplication.run(OrderApplication.class,
args);
    }

    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

    @Bean
    public JdbcTemplate jdbcTemplate(DataSource
dataSource){
        return new JdbcTemplate(new
DataSourceProxy(dataSource));
    }
}
```

```java
package com.southwind;

import io.seata.rm.datasource.DataSourceProxy;
import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBo
otApplication;
import
org.springframework.context.annotation.Bean;
import
org.springframework.jdbc.core.JdbcTemplate;
```

```
import javax.sql.DataSource;

@SpringBootApplication
public class PayApplication {

    public static void main(String[] args) {

 SpringApplication.run(PayApplication.class,
args);
    }

    @Bean
    public JdbcTemplate jdbcTemplate(DataSource
dataSource){
        return new JdbcTemplate(new
DataSourceProxy(dataSource));
    }

}
```

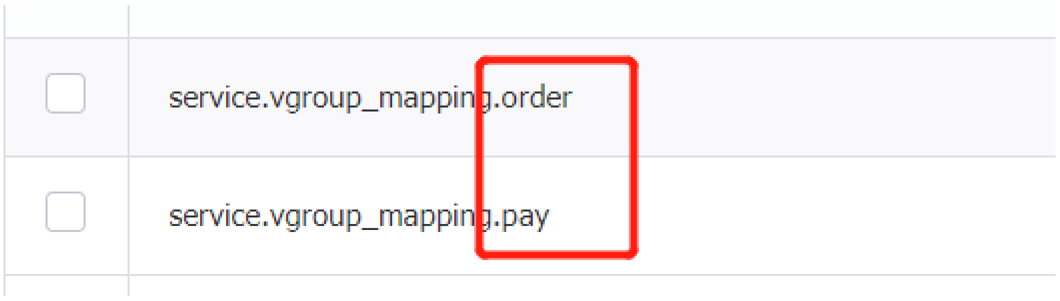
5、将 registry.conf 复制到两个工程的 resources 下。

6、给两个工程添加 bootstrap.yml 读取 Nacos 配置。

```
spring:
  application:
    name: order
  cloud:
    nacos:
      config:
        server-addr: localhost:8848
        namespace: public
        group: SEATA_GROUP
    alibaba:
      seata:
        tx-service-group:
${spring.application.name}
```

```
spring:
  application:
    name: pay
  cloud:
    nacos:
      config:
        server-addr: localhost:8848
        namespace: public
        group: SEATA_GROUP
    alibaba:
      seata:
        tx-service-group:
${spring.application.name}
```

tx-service-group 需要和 Nacos 配置中的名称一致。


service.vgroup_mapping.order

service.vgroup_mapping.pay

7、在 Order 调用 Pay 处添加注解 @GlobalTransactional

```
package com.southwind.controller;

import com.southwind.service.OrderService;
import
io.seata.spring.annotation.GlobalTransactional;
import
org.springframework.beans.factory.annotation.Au
towired;
import
org.springframework.web.bind.annotation.GetMapp
ing;
import
org.springframework.web.bind.annotation.RestCon
troller;
import
org.springframework.web.client.RestTemplate;

@RestController
public class OrderController {

    @Autowired
    private OrderService orderService;
    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/save")
    @GlobalTransactional
    public String save(){
        //订单
        this.orderService.save();
        int i = 10/0;
        //支付

 this.restTemplate.getForObject("http://localho
st:8020/save",String.class);
        return "success";
    }
}
```