sentinel-core & dashboard

```
sentinel java编码的方式设定规则
控制台的下载使用

https://github.com/fanda521/sentinel-study-project
```

1.官方地址

```
1.官方文档地址
https://sentinelguard.io/zh-cn/docs/basic-implementation.html

2.dashboard地址
https://github.com/alibaba/Sentinel/releases
```

2.sentinele-core学习(编码式)

1.小试牛刀

```
使用流控进行实验
```

1.pom

```
<!-- 1. Spring Web 核心依赖（提供 Web 接口能力） -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- 2. Sentinel 核心依赖（核心限流/熔断逻辑） -->
        <dependency>
            <groupId>com.alibaba.csp</groupId>
            <artifactId>sentinel-core</artifactId>
            <version>${sentinel.version}</version>
        </dependency>

        <!-- 3. Sentinel 注解支持核心依赖（新增！支持 @SentinelResource） -->
        <dependency>
            <groupId>com.alibaba.csp</groupId>
            <artifactId>sentinel-annotation-aspectj</artifactId>
            <version>${sentinel.version}</version>
        </dependency>

        <!-- 4. Sentinel 控制台通信依赖（可选，连接 Sentinel 控制台做可视化配置） -
->
        <dependency>
            <groupId>com.alibaba.csp</groupId>
            <artifactId>sentinel-transport-simple-http</artifactId>
            <version>${sentinel.version}</version>
        </dependency>

```

```
        <!-- 5. Spring Boot 测试依赖（可选，用于接口测试） -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!-- 热点参数限流依赖（新增！解决版本提示问题） -->
        <dependency>
            <groupId>com.alibaba.csp</groupId>
            <artifactId>sentinel-parameter-flow-control</artifactId>
            <version>1.8.6</version>
        </dependency>
```

2.yml

```yaml
spring:
  application:
    name: sentinel-origin # 应用名称（会显示在 Sentinel 控制台）
server:
  port: 8088 # 项目端口（避免与 Dashboard 8080 冲突）

# Sentinel 核心配置
sentinel:
  # 控制台连接配置
  transport:
    dashboard: 127.0.0.1:8080 # Dashboard 地址（若改了端口则填对应端口，如 127.0.0.1:8858）
    port: 8719 # 客户端与控制台通信的端口（默认 8719，若被占用可改，如 8720）
    client-ip: 127.0.0.1 # 客户端 IP（多网卡场景需指定，单机默认即可）
  # 可选：关闭控制台懒加载（默认首次请求后才会在控制台显示应用）
  eager: true
```

3.controller

```java
package com.example.study.sentinelorigin.controller;

import com.alibaba.csp.sentinel.Entry;
import com.alibaba.csp.sentinel.SphU;
import com.alibaba.csp.sentinel.slots.block.BlockException;
import com.alibaba.csp.sentinel.slots.block.RuleConstant;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.List;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 1:22
 */
@RestController
@RequestMapping("/hello")
public class HelloController {
```

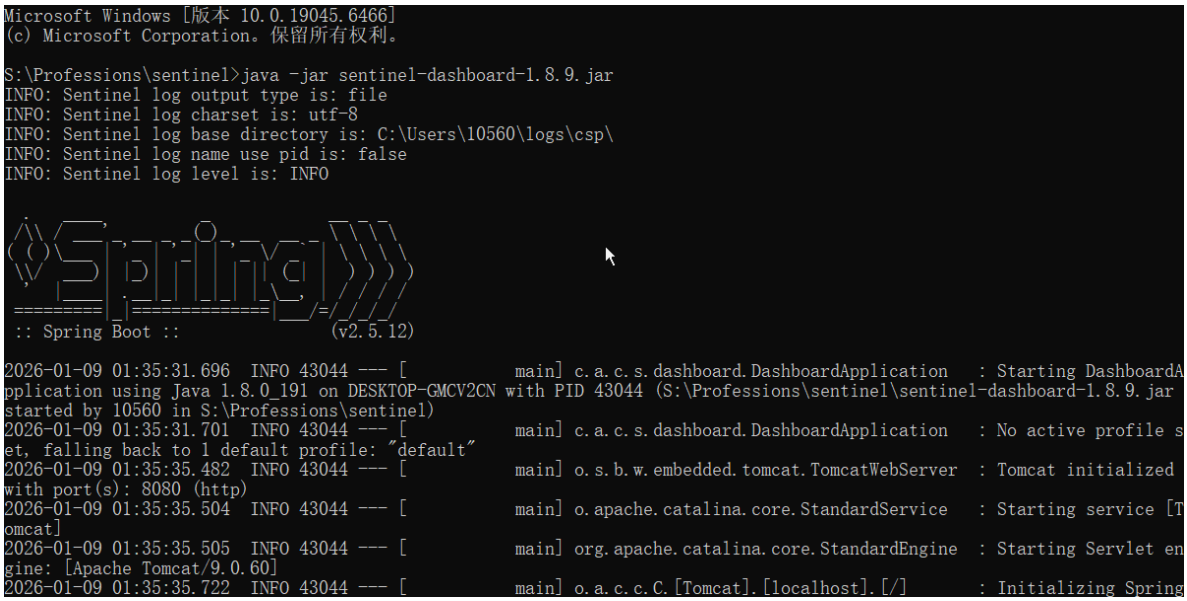
```java

    @RequestMapping("/sayHello")
    public String hello() {

        Entry entry = null;
        // 务必保证finally会被执行
        try {
            // 资源名可使用任意有业务语义的字符串
            entry = SphU.entry("hello");
            // 被保护的业务逻辑
            // do something...
            System.out.println("hello world");
        } catch (BlockException e1) {
            // 资源访问阻止，被限流或被降级
            // 进行相应的处理操作
            System.out.println("限流");
            return "限流";
        } finally {
            if (entry != null) {
                entry.exit();
            }
        }
        return "hello world";
    }

    @PostConstruct
    private static void initFlowQpsRule() {
        List<FlowRule> rules = new ArrayList<>();
        FlowRule rule1 = new FlowRule();
        rule1.setResource("hello");
        // Set max qps to 20
        rule1.setCount(1);
        rule1.setGrade(RuleConstant.FLOW_GRADE_QPS);
        rules.add(rule1);
        FlowRuleManager.loadRules(rules);
    }


}

```

4.测试效果

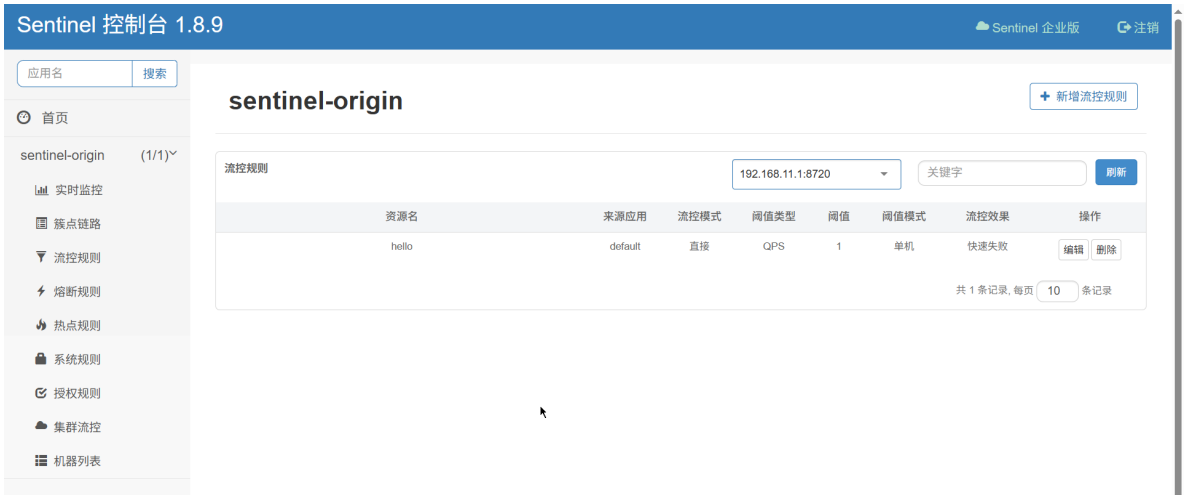
```
一秒内调用多次就触发限流
```

5.dashboard

```
1.下载好jar后
执行 java  -jar sentinel-dashboard-1.8.9.jar

2.启动刚写的服务8088的

3.观察dashboard
http://localhost:8080
默认账号密码
sentinel/sentinel
```




```
如果一直发现sentinel没有下面的应用，就先调用接口触发限流，或者重启应用和sentinel bashboard
```



2.注解的方式

```
1.引入jar
<!-- 3. Sentinel 注解支持核心依赖（新增！支持 @SentinelResource） -->
        <dependency>
            <groupId>com.alibaba.csp</groupId>
```

```
            <artifactId>sentinel-annotation-aspectj</artifactId>
            <version>${sentinel.version}</version>
        </dependency>

2.配置bean
package com.example.study.sentinelorigin.config;

import com.alibaba.csp.sentinel.annotation.aspectj.SentinelResourceAspect;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 2:39
 */
@Configuration
public class SentinelSourceConfig {

    @Bean
    public SentinelResourceAspect sentinelResourceAspect() {
        return new SentinelResourceAspect();
    }
}


3.在目标方法加上注解和定义对应的blockHandler方法
package com.example.study.sentinelorigin.controller;

import com.alibaba.csp.sentinel.Entry;
import com.alibaba.csp.sentinel.SphU;
import com.alibaba.csp.sentinel.annotation.SentinelResource;
import com.alibaba.csp.sentinel.slots.block.BlockException;
import com.alibaba.csp.sentinel.slots.block.RuleConstant;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.List;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 2:29
 */
@RestController
@RequestMapping("/anno")
public class AnnoController {
    @RequestMapping("/sayHello")
    @SentinelResource(value = "anno-hello", blockHandler =
"annoHandleException")
    public String hello() {
        System.out.println("anno world");
        return "anno-hello";
    }
```

```
62
63      public String annoHandleException(BlockException e) {
64          e.printStackTrace();
65          System.out.println("anno-hello-限流");
66          return "anno-hello-限流";
67      }
68
69  }
70
71  4.注册规则，需要再同一个方法中注册否则会不起作用
72  @PostConstruct
73      private static void initFlowQpsRule() {
74          List<FlowRule> rules = new ArrayList<>();
75          FlowRule rule1 = new FlowRule();
76          rule1.setResource("hello");
77          // Set max qps to 20
78          rule1.setCount(1);
79          rule1.setGrade(RuleConstant.FLOW_GRADE_QPS);
80          rules.add(rule1);
81          List<FlowRule> rulesOrigin = FlowRuleManager.getRules();
82
83
84          FlowRule rule2 = new FlowRule();
85          rule2.setResource("anno-hello");
86          // Set max qps to 20
87          rule2.setCount(1);
88          rule2.setGrade(RuleConstant.FLOW_GRADE_QPS);
89          rules.add(rule2);
90          // 多个规则最好写在一个累的方法中，否则可能失效
91
92          FlowRule rule3 = new FlowRule();
93          rule3.setResource("failBlock-test01");
94          // Set max qps to 20
95          rule3.setCount(1);
96          rule3.setGrade(RuleConstant.FLOW_GRADE_QPS);
97          rules.add(rule3);
98
99          rulesOrigin.addAll(rules);
100         FlowRuleManager.loadRules(rulesOrigin);
101     }
```

3.failback和blockhandler

```
1  1.默认
2  需要是public 类在本类中
3
4  2.兜底的方法在其他类中，那就配合对应的xxxClass
```

1.编写controller

```
1  package com.example.study.sentinelorigin.controller;
2
3  import com.alibaba.csp.sentinel.Entry;
4  import com.alibaba.csp.sentinel.SphU;
```

```java
import com.alibaba.csp.sentinel.annotation.SentinelResource;
import com.alibaba.csp.sentinel.slots.block.BlockException;
import com.alibaba.csp.sentinel.slots.block.RuleConstant;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
import com.example.study.sentinelorigin.handle.FailBlockHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.List;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 2:21
 */
@RestController
@RequestMapping("/failBlock")
public class FailBackAndBlockHandlerDefaultClass {


    @RequestMapping("/testO1")
    @SentinelResource(value = "failBlock-testO1", blockHandlerClass = FailBlockHandler.class ,blockHandler = "failBlockTestO1")
    public String testO1() {
        return "failBlock-testO1";
    }
}

```

2.异常处理类

```java
package com.example.study.sentinelorigin.handle;

import com.alibaba.csp.sentinel.slots.block.BlockException;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 2:54
 */
public class FailBlockHandler {

    public static String failBlockTestO1(BlockException e) {
        e.printStackTrace();
        System.out.println("failBlockHandler-限流");
        return "failBlockHandler-限流";
    }
}

```

4.flow

1.thread

```
有两种方式
1.qps
2.线程并发数

这里就是测试线程并发数
```

1.controller

```
@GetMapping("/annoThread")
    @SentinelResource(
            value = CommonConstant.THREAD_RESOURCE_NAME, // 绑定资源名，与流控规则中的资源名一致
            blockHandlerClass = FlowHandler.class,
            blockHandler = "threadFlowBlockHandler" // 指定限流降级方法（局部）
    )
    public String testAnnotationThreadFlow() throws InterruptedException {
        // 模拟耗时业务（睡眠 3 秒，让线程堆积，方便触发并发线程数限流）
        TimeUnit.SECONDS.sleep(3);

        // 正常响应结果
        return String.format("【成功】当前线程：%s，业务执行完成", Thread.currentThread().getName());
    }
```

2.rule

```
package com.example.study.sentinelorigin.rule;

import com.alibaba.csp.sentinel.slots.block.RuleConstant;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
import com.example.study.sentinelorigin.constant.CommonConstant;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.List;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 16:49
 */
@Component
public class FlowRuleConfig {

    @PostConstruct
    public void initAnnotationThreadFlowRules() {
        FlowRule rule = new com.alibaba.csp.sentinel.slots.block.flow.FlowRule();
        rule.setResource(CommonConstant.THREAD_RESOURCE_NAME); // 绑定注解对应的资源名
        rule.setGrade(RuleConstant.FLOW_GRADE_THREAD); // 限流类型：并发线程数（核心）
```

```
        rule.setCount(5); // 最大并发线程数阈值：5
        rule.setLimitApp("default"); // 针对默认应用限流

        // 步骤1：读取现有已加载的规则（转为可修改列表）
        List<FlowRule> existingRules = new ArrayList<>
(FlowRuleManager.getRules());
        System.out.println("追加前，现有规则数：" + existingRules.size());
        // 步骤3：调用loadRules()重新加载（实现追加效果）
        existingRules.add(rule);
        FlowRuleManager.loadRules(existingRules);
        System.out.println("追加后，当前生效规则数：" +
FlowRuleManager.getRules().size());
    }
}

```

3.handle

```
package com.example.study.sentinelorigin.handle;

import com.alibaba.csp.sentinel.slots.block.BlockException;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/9 16:49
 */
public class FlowHandler {

    // 步骤 3：注解指定的降级方法（blockHandler 要求）
    /**
     * 1. 方法权限：public（必须）
     * 2. 返回值：与原方法一致（必须）
     * 3. 参数：与原方法一致 + 末尾追加 BlockException（必须）
     * 4. 若原方法无异常抛出，降级方法可仅追加 BlockException
     */
    public static String threadFlowBlockHandler(BlockException e) {
        return String.format("【降级】当前线程：%s，并发线程数超过阈值 5，拒绝访问",
Thread.currentThread().getName());
    }
}

```

4.client

```
package com.example.study.sentinelorigin.client;

import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.junit.Test;

```

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class FlowTest {
    private static final int THREAD_NUM = 20;
    private static final String TEST_FLOW_THREAD_URL = "http://localhost:8088/flow/annoThread";

    @Test
    public void  testFlowThread() throws InterruptedException {
        ExecutorService executorService = Executors.newFixedThreadPool(THREAD_NUM);
        for (int i = 0; i < THREAD_NUM; i++) {
            executorService.submit(() -> {
                try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
                    HttpGet httpGet = new HttpGet(TEST_FLOW_THREAD_URL);
                    String response = EntityUtils.toString(httpClient.execute(httpGet).getEntity());
                    System.out.println(response);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            });
        }
        Thread.sleep(10000);
        executorService.shutdown();
    }


}

```

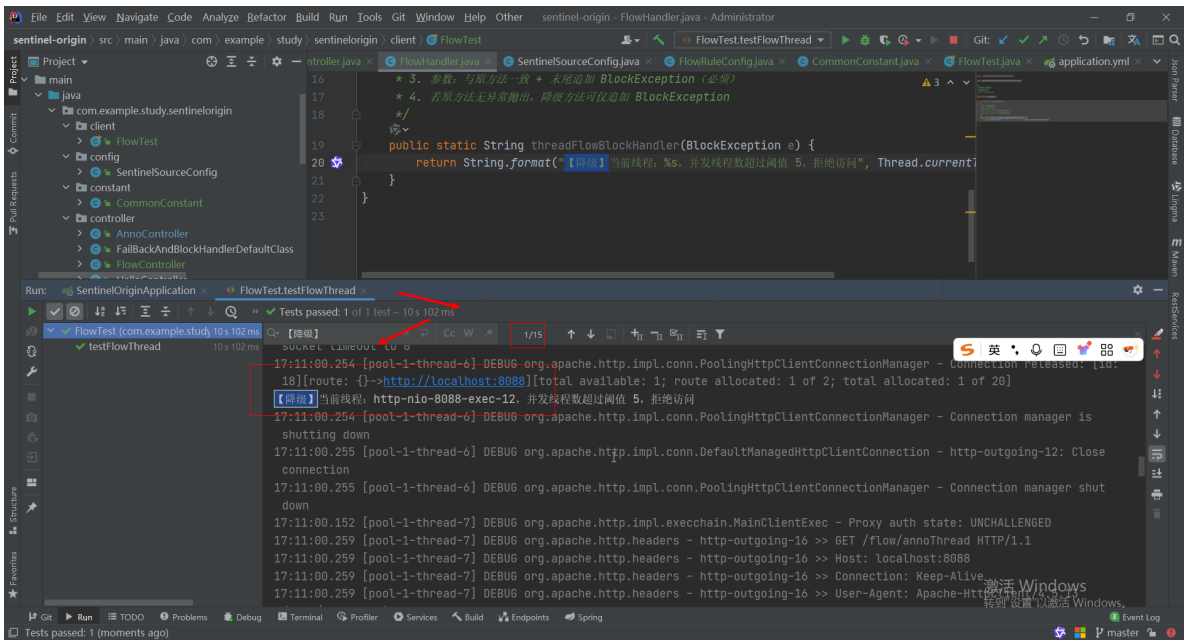
5.constant

```java
// 定义注解绑定的资源名（也可直接在 @SentinelResource 中写死）
    public static final String THREAD_RESOURCE_NAME = "annotationThreadFlowResource";
```

6.pom

```xml
<!-- CloseableHttpClient 核心依赖（新增） -->
        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>4.5.13</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
```

7.效果



2.limitApp

1.controller

```
// 步骤 2：注解式测试接口
    @GetMapping("/limitApp")
    @SentinelResource(
            value = CommonConstant.LIMIT_APP_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "limitAppBlockHandler"
    )
    public String testLimitApp() throws InterruptedException {
        Thread.sleep(2000);
        String format = String.format("【成功】当前线程：%s，请求正常执行",
Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }
```

2.rule

```
// limitApp
        // 规则 1：仅对应用 "appA" 进行 QPS 限流（阈值 2）
        FlowRule appARule = new FlowRule();
        appARule.setResource(CommonConstant.LIMIT_APP_RESOURCE); // 绑定资源
        appARule.setGrade(RuleConstant.FLOW_GRADE_THREAD); // QPS 限流
        appARule.setCount(2); // 每秒最多 2 个请求
        appARule.setLimitApp("appA"); // 仅对 appA 生效（核心配置）

        // 规则 2：对除 appA 之外的所有其他来源（兜底）进行 QPS 限流（阈值 5）
        FlowRule otherRule = new FlowRule();
        otherRule.setResource(CommonConstant.LIMIT_APP_RESOURCE);
```

```
        otherRule.setGrade(RuleConstant.FLOW_GRADE_THREAD);
        otherRule.setCount(5);
        otherRule.setLimitApp("other"); // 对非 appA 来源生效（核心配置）
        existingRules.add(appARule);
        existingRules.add(otherRule);
```

3.handle

```
public static String limitAppBlockHandler(BlockException e) {
        String format = String.format("【降级】当前线程：%s，请求过于频繁，触发 limitApp 限流", Thread.currentThread().getName());
        System.out.println(format);
        return format;

    }
```

4.client

```
@Test
    public void testLimitApp() throws InterruptedException {
        System.out.println("========== 场景 1：测试来源 appA（阈值 2）==========");
        testLimitApp("appA");

        // 间隔 5 秒，让 Sentinel 重置 QPS 统计
        Thread.sleep(5000);

        System.out.println("\n========== 场景 2：测试来源 appB（匹配 other 规则，阈值 5）==========");
        testLimitApp("other");

        // 间隔 5 秒
        Thread.sleep(5000);

        System.out.println("\n========== 场景 3：测试无来源（匹配 other 规则，阈值 5）==========");
        testLimitApp(null);
    }

    /**
     * 模拟指定来源的高并发请求
     * @param appName 应用来源（null 表示无来源）
     */
    private static void testLimitApp(String appName) {
        // 创建固定线程池
        ExecutorService executorService = Executors.newFixedThreadPool(CONCURRENT_THREADS);

        // 提交并发请求任务
        for (int i = 0; i < CONCURRENT_THREADS; i++) {
            executorService.submit(() -> {
                try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
```

```
                    // 构建 GET 请求
                    HttpGet httpGet = new HttpGet(TEST_URL);

                    // 若指定了应用来源，添加请求头 X-Sentinel-App
                    if (appName != null && !appName.isEmpty()) {
                        httpGet.addHeader("X-Sentinel-App", appName);
                    }

                    // 执行请求并获取响应结果
                    String response =
EntityUtils.toString(httpClient.execute(httpGet).getEntity(), "UTF-8");
                    System.out.println(response);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            });
        }

        // 关闭线程池，等待所有任务执行完成
        executorService.shutdown();
        while (!executorService.isTerminated()) {
            // 等待任务结束
        }
    }
```

5.constant

```
// 资源名
    public static final String LIMIT_APP_RESOURCE = "limitAppResource";
```

6.parser

```

package com.example.study.sentinelorigin.config;

import
com.alibaba.csp.sentinel.adapter.servlet.callback.RequestOriginParser;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;

/**
 * 自定义请求来源解析器：从请求头中提取 "X-Sentinel-App" 作为应用来源（limitApp）
 */
@Component
public class CustomRequestOriginParser implements RequestOriginParser {

    public CustomRequestOriginParser() {
        System.out.println("========== CustomRequestOriginParser 被 Spring
实例化了 ==========");
    }

    @Override
```

```
    public String parseOrigin(HttpServletRequest request) {
        // 从请求头中获取应用来源（可改为从请求参数、Cookie 等提取）
        String appName = request.getHeader("X-Sentinel-App");
        // 若请求头中无该字段，默认返回 "unknown"
        System.out.println("请求来源：" + appName);
        return appName == null ? "unknown" : appName;
    }
}
```

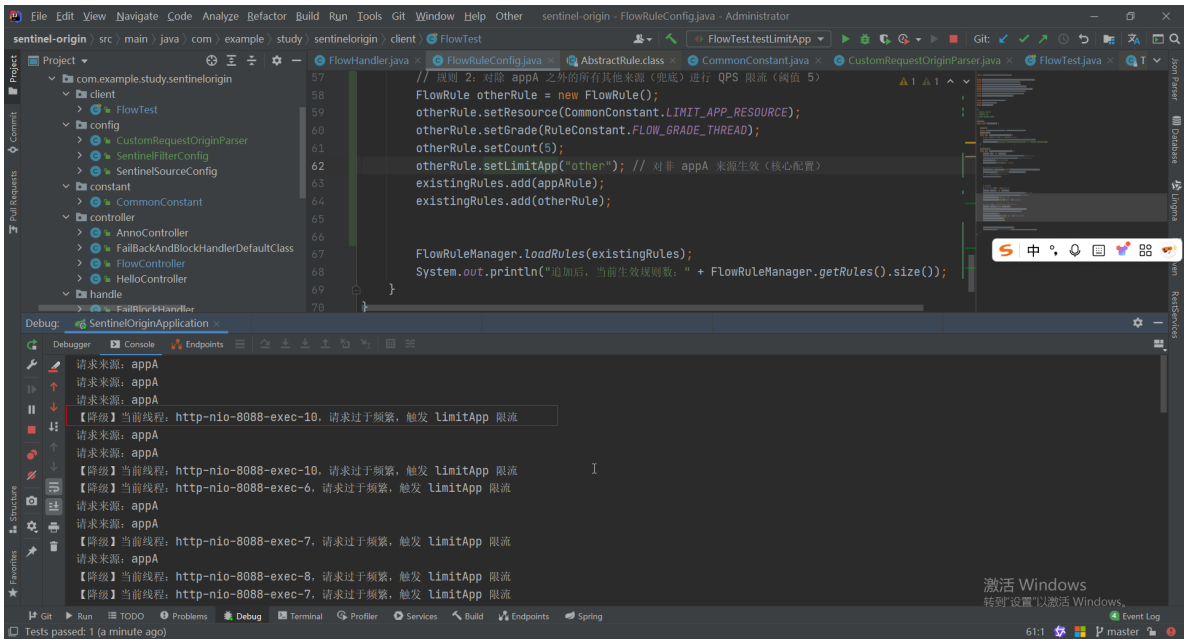
7.commonFilter

```
package com.example.study.sentinelorigin.config;

import com.alibaba.csp.sentinel.adapter.servlet.CommonFilter;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.servlet.DispatcherType;

/**
 * 手动注册 Sentinel CommonFilter，确保 Web 请求被 Sentinel 拦截
 */
@Configuration
public class SentinelFilterConfig {

    @Bean
    public FilterRegistrationBean<CommonFilter>
sentinelCommonFilterRegistration() {
        FilterRegistrationBean<CommonFilter> registrationBean = new
FilterRegistrationBean<>();
        // 注册 Sentinel CommonFilter
        registrationBean.setFilter(new CommonFilter());
        // 拦截所有请求（/* 表示拦截所有路径，确保所有接口都被 Sentinel 处理）
        registrationBean.addUrlPatterns("/*");
        // 设置过滤器顺序（优先级高于其他过滤器，确保先被执行）
        registrationBean.setOrder(1);
        // 匹配所有请求分发类型（包括直接请求、转发、包含等）
        registrationBean.setDispatcherTypes(DispatcherType.REQUEST,
DispatcherType.FORWARD);
        // 启用该过滤器
        registrationBean.setEnabled(true);
        return registrationBean;
    }
}
```

8.pom

```
        <!-- 新增: Sentinel Web Servlet 适配依赖（包含 RequestOriginParser 类） -->
        <dependency>
            <groupId>com.alibaba.csp</groupId>
            <artifactId>sentinel-web-servlet</artifactId>
            <version>1.8.6</version>
        </dependency>
```

9.效果



3.流控效果策略

```
拒绝/排队等待/预热启动
```

1.controller

```
// 接口 1：测试 直接拒绝 效果
    @GetMapping("/strategy/default")
    @SentinelResource(
            value = CommonConstant.DEFAULT_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "defaultBlockHandler"
    )
    public String testDefaultControlBehavior() {
        String format = String.format("【直接拒绝-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;

    }

    // 接口 2：测试 预热/冷启动 效果
    @GetMapping("/strategy/warmup")
```

```
    @SentinelResource(
            value = CommonConstant.WARM_UP_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "warmUpBlockHandler"
    )
    public String testWarmUpControlBehavior() {
        String format = String.format("【预热-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 接口 3：测试 匀速排队 效果
    @GetMapping("/strategy/ratelimiter")
    @SentinelResource(
            value = CommonConstant.RATE_LIMITER_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "rateLimiterBlockHandler"
    )
    public String testRateLimiterControlBehavior() {
        String format = String.format("【匀速排队-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;

    }
```

2.rule

```
// 规则 1：直接拒绝（CONTROL_BEHAVIOR_DEFAULT，默认值）
        FlowRule defaultRule = new FlowRule();
        defaultRule.setResource(CommonConstant.DEFAULT_RESOURCE);
        defaultRule.setGrade(RuleConstant.FLOW_GRADE_QPS); // QPS 限流
        defaultRule.setCount(10); // QPS 阈值 10

 defaultRule.setControlBehavior(RuleConstant.CONTROL_BEHAVIOR_DEFAULT); // 直接拒绝（可省略，默认值）
        existingRules.add(defaultRule);

        // 规则 2：预热/冷启动（CONTROL_BEHAVIOR_WARM_UP）
        FlowRule warmUpRule = new FlowRule();
        warmUpRule.setResource(CommonConstant.WARM_UP_RESOURCE);
        warmUpRule.setGrade(RuleConstant.FLOW_GRADE_QPS);
        warmUpRule.setCount(20); // 最终 QPS 阈值 20

 warmUpRule.setControlBehavior(RuleConstant.CONTROL_BEHAVIOR_WARM_UP); // 预热效果
        warmUpRule.setWarmUpPeriodSec(5); // 预热时间 5 秒（阈值从 10 逐步提升至 20）
        existingRules.add(warmUpRule);

        // 规则 3：匀速排队（CONTROL_BEHAVIOR_RATE_LIMITER）
        FlowRule rateLimiterRule = new FlowRule();
        rateLimiterRule.setResource(CommonConstant.RATE_LIMITER_RESOURCE);
        rateLimiterRule.setGrade(RuleConstant.FLOW_GRADE_QPS);
```

```
        rateLimiterRule.setCount(5); // QPS 阈值 5（每秒允许 5 个请求通过，间隔 200 毫秒/个）

 rateLimiterRule.setControlBehavior(RuleConstant.CONTROL_BEHAVIOR_RATE_LIMITER); // 匀速排队
        rateLimiterRule.setMaxQueueingTimeMs(1000); // 最大排队等待时间 1000 毫秒（1 秒），超过则拒绝
        existingRules.add(rateLimiterRule);
```

3.handler

```
// 降级方法：直接拒绝
    public static String defaultBlockHandler(BlockException e) {
        String format = String.format("【直接拒绝-降级】当前线程：%s，QPS 超过阈值 10，触发限流", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 降级方法：预热
    public static String warmUpBlockHandler(BlockException e) {
        String format = String.format("【预热-降级】当前线程：%s，预热期内 QPS 超过当前阈值，触发限流", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 降级方法：匀速排队
    public static String rateLimiterBlockHandler(BlockException e) {
        String format = String.format("【匀速排队-降级】当前线程：%s，排队时间超过 1 秒，触发限流", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }
```

4.client

```
@Test
    public void testStrategy() throws InterruptedException {
        System.out.println("========== 场景 1：测试 直接拒绝 效果（QPS 阈值 10）==========");
        testControlBehavior(DEFAULT_URL);

        // 间隔 10 秒，让 Sentinel 重置统计
        TimeUnit.SECONDS.sleep(10);

        System.out.println("\n========== 场景 2：测试 预热/冷启动 效果（最终 QPS 阈值 20，预热 5 秒）==========");
        testControlBehavior(WARM_UP_URL);

        // 间隔 10 秒
        TimeUnit.SECONDS.sleep(10);

```

```
        System.out.println("\n========== 场景 3：测试 匀速排队 效果（QPS 阈值 5，最大排队 1 秒）==========");
        testControlBehavior(RATE_LIMITER_URL);
    }

    /**
     * 模拟高并发请求，验证流控效果
     * @param url 测试接口地址
     */
    private static void testControlBehavior(String url) {
        // 创建固定线程池
        ExecutorService executorService = Executors.newFixedThreadPool(STRATEGY_CONCURRENT_THREADS);

        // 记录开始时间
        long startTime = System.currentTimeMillis();

        // 提交并发请求任务
        for (int i = 0; i < STRATEGY_CONCURRENT_THREADS; i++) {
            executorService.submit(() -> {
                try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
                    HttpGet httpGet = new HttpGet(url);
                    String response = EntityUtils.toString(httpClient.execute(httpGet).getEntity(), "UTF-8");
                    System.out.println(response);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            });
        }

        // 关闭线程池，等待所有任务执行完成
        executorService.shutdown();
        while (!executorService.isTerminated()) {}

        // 记录结束时间，打印耗时
        long endTime = System.currentTimeMillis();
        System.out.println("本次请求总耗时：" + (endTime - startTime) + " 毫秒");
    }
```

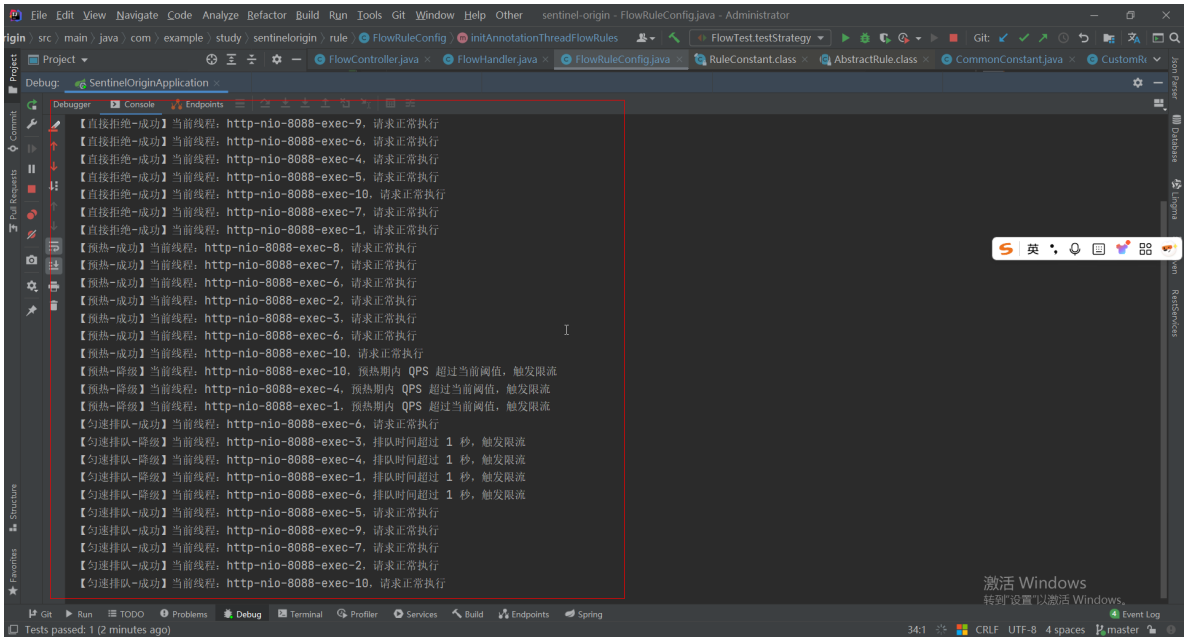
5.constant

```
// 资源名（区分不同流控效果）
    public static final String DEFAULT_RESOURCE = "defaultControlBehaviorResource";
    public static final String WARM_UP_RESOURCE = "warmUpControlBehaviorResource";
    public static final String RATE_LIMITER_RESOURCE = "rateLimiterControlBehaviorResource";
```

6.效果

```
【直接拒绝-成功】当前线程：http-nio-8088-exec-3，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-3，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-4，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-3，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-4，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-3，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-3，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-9，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-9，请求正常执行
【直接拒绝-成功】当前线程：http-nio-8088-exec-9，请求正常执行
【直接拒绝-降级】当前线程：http-nio-8088-exec-9，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-5，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-6，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-10，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-8，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-6，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-5，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-5，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-10，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-4，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-3，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-4，QPS 超过阈值 10，触发限流
【直接拒绝-降级】当前线程：http-nio-8088-exec-3，QPS 超过阈值 10，触发限流
【预热-成功】当前线程：http-nio-8088-exec-9，请求正常执行
【预热-成功】当前线程：http-nio-8088-exec-7，请求正常执行
【预热-成功】当前线程：http-nio-8088-exec-1，请求正常执行
【预热-成功】当前线程：http-nio-8088-exec-10，请求正常执行
【预热-成功】当前线程：http-nio-8088-exec-5，请求正常执行
【预热-成功】当前线程：http-nio-8088-exec-10，请求正常执行
【预热-降级】当前线程：http-nio-8088-exec-9，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-2，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-6，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-1，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-6，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-8，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-9，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-4，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-9，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-1，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-1，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-6，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-8，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-4，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-9，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-3，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-6，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-2，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-2，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-4，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-9，预热期内 QPS 超过当前阈值，触发限流
```

```
【预热-降级】当前线程：http-nio-8088-exec-3，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-3，预热期内 QPS 超过当前阈值，触发限流
【预热-降级】当前线程：http-nio-8088-exec-5，预热期内 QPS 超过当前阈值，触发限流
【匀速排队-成功】当前线程：http-nio-8088-exec-10，请求正常执行
【匀速排队-降级】当前线程：http-nio-8088-exec-3，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-3，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-5，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-1，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-1，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-1，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-3，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-7，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-7，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-10，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-7，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-3，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-5，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-10，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-3，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-3，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-5，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-10，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-5，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-10，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-7，排队时间超过 1 秒，触发限流
【匀速排队-成功】当前线程：http-nio-8088-exec-4，请求正常执行
【匀速排队-降级】当前线程：http-nio-8088-exec-5，排队时间超过 1 秒，触发限流
【匀速排队-降级】当前线程：http-nio-8088-exec-4，排队时间超过 1 秒，触发限流
【匀速排队-成功】当前线程：http-nio-8088-exec-9，请求正常执行
【匀速排队-成功】当前线程：http-nio-8088-exec-8，请求正常执行
【匀速排队-成功】当前线程：http-nio-8088-exec-6，请求正常执行
【匀速排队-成功】当前线程：http-nio-8088-exec-2，请求正常执行
【匀速排队-成功】当前线程：http-nio-8088-exec-1，请求正常执行
```



4.触发模式

```
链路模式尝试了很多方式都失败了
1.设置资源路径合并
2.controller调service
3.不依赖web,本地调
4.手动在UI上添加限流

都不行，暂时跳过
```

1.controller

```
    // ************************ 直接流控 测试接口 ************************
    @GetMapping("/module/direct")
    @SentinelResource(
            value = CommonConstant.DIRECT_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "directBlockHandler"
    )
    public String testDirectStrategy() {
        String format = String.format("【直接流控-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ************************ 关联流控 测试接口 ************************
    // 接口 2.1：当前资源（订单创建）
    @GetMapping("/module/associateCurrent")
    @SentinelResource(
            value = CommonConstant.ASSOCIATE_CURRENT_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "associateCurrentBlockHandler"
    )
    public String testAssociateCurrent() {
        String format = String.format("【关联流控-当前资源（订单创建）-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 接口 2.2：关联资源（库存扣减）
    @GetMapping("/module/associateRef")
    @SentinelResource(
            value = CommonConstant.ASSOCIATE_REF_RESOURCE
    )
    public String testAssociateRef() throws InterruptedException {
        // 模拟库存扣减耗时，便于触发关联限流
        Thread.sleep(100);
        String format = String.format("【关联流控-关联资源（库存扣减）-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ************************ 链路流控 测试接口 ************************
```

```
    // 接口 3.1：入口链路资源（/api/entry）
    @GetMapping("/module/chainEntry")
    @SentinelResource(
            value = CommonConstant.CHAIN_ENTRY_RESOURCE
    )
    public String testChainEntry() throws InterruptedException {
        // 入口链路调用当前资源（用户查询）
        return chainFlowCoreService.doCoreUserQuery();
    }

    // 接口 3.2：当前资源（用户查询）
    @GetMapping("/module/chainCurrent")
    public String testChainCurrent() throws InterruptedException {
        String format = chainFlowCoreService.doCoreUserQuery();
        return format;
    }
```

2.rule

```

    // ************************ 直接流控 测试接口 ************************
    @GetMapping("/module/direct")
    @SentinelResource(
            value = CommonConstant.DIRECT_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "directBlockHandler"
    )
    public String testDirectStrategy() {
        String format = String.format("【直接流控-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ************************ 关联流控 测试接口 ************************
    // 接口 2.1：当前资源（订单创建）
    @GetMapping("/module/associateCurrent")
    @SentinelResource(
            value = CommonConstant.ASSOCIATE_CURRENT_RESOURCE,
            blockHandlerClass = FlowHandler.class,
            blockHandler = "associateCurrentBlockHandler"
    )
    public String testAssociateCurrent() {
        String format = String.format("【关联流控-当前资源（订单创建）-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 接口 2.2：关联资源（库存扣减）
    @GetMapping("/module/associateRef")
    @SentinelResource(
            value = CommonConstant.ASSOCIATE_REF_RESOURCE
    )
    public String testAssociateRef() throws InterruptedException {
```

```
        // 模拟库存扣减耗时，便于触发关联限流
        Thread.sleep(100);
        String format = String.format("【关联流控-关联资源（库存扣减）-成功】当前线程：%s，请求正常执行", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ************************ 链路流控 测试接口 ************************
    // 接口 3.1：入口链路资源（/api/entry）
    @GetMapping("/module/chainEntry")
    @SentinelResource(
            value = CommonConstant.CHAIN_ENTRY_RESOURCE
    )
    public String testChainEntry() throws InterruptedException {
        // 入口链路调用当前资源（用户查询）
        return chainFlowCoreService.doCoreUserQuery();
    }

    // 接口 3.2：当前资源（用户查询）
    @GetMapping("/module/chainCurrent")
    public String testChainCurrent() throws InterruptedException {
        String format = chainFlowCoreService.doCoreUserQuery();
        return format;
    }
```

3.client

```
// 测试接口地址
    private static final String DIRECT_URL = "http://localhost:8088/flow/module/direct";
    private static final String ASSOCIATE_CURRENT_URL = "http://localhost:8088/flow/module/associateCurrent";
    private static final String ASSOCIATE_REF_URL = "http://localhost:8088/flow/module/associateRef";
    private static final String CHAIN_ENTRY_URL = "http://localhost:8088/flow/module/chainEntry";
    private static final String CHAIN_CURRENT_URL = "http://localhost:8088/flow/module/chainCurrent";
    // 并发线程数（用于触发限流）
    private static final int Module_CONCURRENT_THREADS = 20;

    @Test
    public void testModule() throws InterruptedException {
        System.out.println("========== 场景 1：测试 直接流控 效果（当前资源 QPS 阈值 10）==========");
        testStrategy(DIRECT_URL);

        // 间隔 10 秒，让 Sentinel 重置统计
        TimeUnit.SECONDS.sleep(10);

        System.out.println("\n========== 场景 2：测试 关联流控 效果（关联资源 QPS 阈值 5，联动限流当前资源）==========");
        // 第一步：高并发请求关联资源（库存扣减），使其 QPS 超过阈值 5
        System.out.println("--- 第一步：高并发请求关联资源（库存扣减）---");
```

```
        testStrategy(ASSOCIATE_REF_URL);
        // 第二步：立即请求当前资源（订单创建），验证是否被联动限流
        System.out.println("--- 第二步：请求当前资源（订单创建），验证关联限流 ---");
        testStrategy(ASSOCIATE_CURRENT_URL, 5);

        // 间隔 10 秒
        TimeUnit.SECONDS.sleep(10);

        System.out.println("\n========== 场景 3：测试 链路流控 效果（入口链路 QPS 阈值 8）==========");
        // 第一步：高并发请求入口链路（/api/entry），触发链路限流
        System.out.println("--- 第一步：高并发请求入口链路，触发链路限流 ---");
        testStrategy(CHAIN_ENTRY_URL);
        // 第二步：直接请求当前资源，验证是否不受限流影响
        System.out.println("--- 第二步：直接请求当前资源，验证不受链路限流影响 ---");
        testStrategy(CHAIN_CURRENT_URL, 12);
    }

    /**
     * 模拟高并发请求（默认 20 个线程）
     * @param url 测试接口地址
     */
    private static void testStrategy(String url) {
        testStrategy(url, Module_CONCURRENT_THREADS);
    }

    /**
     * 模拟指定线程数的并发请求
     * @param url 测试接口地址
     * @param threadNum 并发线程数
     */
    private static void testStrategy(String url, int threadNum) {
        // 创建固定线程池
        ExecutorService executorService = Executors.newFixedThreadPool(threadNum);

        // 提交并发请求任务
        for (int i = 0; i < threadNum; i++) {
            executorService.submit(() -> {
                for (int j = 0; j < 20; j++) {
                    try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
                        HttpGet httpGet = new HttpGet(url);
                        String response = EntityUtils.toString(httpClient.execute(httpGet).getEntity(), "UTF-8");
                        System.out.println(response);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            });
        }

        // 关闭线程池，等待所有任务执行完成
        executorService.shutdown();
        while (!executorService.isTerminated()) {}
```

```
    }
```

4.handler

```
// ************************ 降级方法 ************************
    // 直接流控降级方法
    public static String directBlockHandler(BlockException e) {
        String format = String.format("【直接流控-降级】当前线程：%s，QPS 超过阈值10，触发限流", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 关联流控-当前资源降级方法
    public static String associateCurrentBlockHandler(BlockException e) {
        String format = String.format("【关联流控-降级】当前线程：%s，关联资源（库存扣减）QPS 超过阈值 5，触发当前资源（订单创建）限流", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // 链路流控-当前资源降级方法
    public static String chainCurrentBlockHandler(BlockException e) {
        String format = String.format("【链路流控-降级】当前线程：%s，入口链路（%s）QPS 超过阈值 8，触发限流", Thread.currentThread().getName(), CommonConstant.CHAIN_ENTRY_RESOURCE);
        System.out.println(format);
        return format;
    }
```

5.constant

```
// 资源名定义
    public static final String DIRECT_RESOURCE = "directStrategyResource"; // 直接流控-当前资源

    public static final String ASSOCIATE_CURRENT_RESOURCE = "associateCurrentResource"; // 关联流控-当前资源（订单创建）
    public static final String ASSOCIATE_REF_RESOURCE = "associateRefResource"; // 关联流控-关联资源（库存扣减）

    public static final String CHAIN_CURRENT_RESOURCE = "chainCurrentResource"; // 链路流控-当前资源（用户查询）
    public static final String CHAIN_ENTRY_RESOURCE = "chainEntryResource"; // 链路流控-入口资源（/api/entry）
```

6.效果

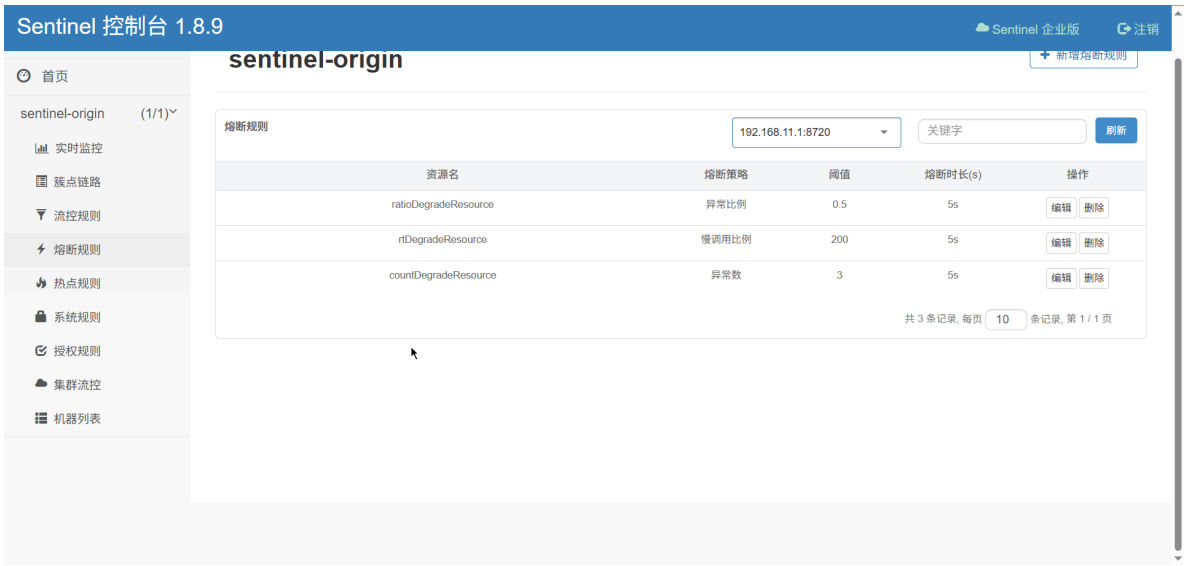
```
【直接流控-成功】当前线程：http-nio-8088-exec-4，请求正常执行
```

```
【直接流控-成功】当前线程：http-nio-8088-exec-13，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-7，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-20，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-1，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-10，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-12，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-16，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-14，请求正常执行
【直接流控-成功】当前线程：http-nio-8088-exec-19，请求正常执行
【直接流控-降级】当前线程：http-nio-8088-exec-5，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-2，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-9，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-11，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-15，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-6，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-3，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-8，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-17，QPS 超过阈值 10，触发限流
【直接流控-降级】当前线程：http-nio-8088-exec-18，QPS 超过阈值 10，触发限流


【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-6，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-9，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-13，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-16，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-14，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-2，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-20，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-15，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-18，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-11，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-17，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-3，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-8，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-4，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-7，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-12，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-5，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-1，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-19，请求正常执行
【关联流控-关联资源（库存扣减）-成功】当前线程：http-nio-8088-exec-10，请求正常执行
【关联流控-降级】当前线程：http-nio-8088-exec-17，关联资源（库存扣减）QPS 超过阈值 5，触发当前资源（订单创建）限流
【关联流控-降级】当前线程：http-nio-8088-exec-6，关联资源（库存扣减）QPS 超过阈值 5，触发当前资源（订单创建）限流
【关联流控-降级】当前线程：http-nio-8088-exec-3，关联资源（库存扣减）QPS 超过阈值 5，触发当前资源（订单创建）限流
【关联流控-降级】当前线程：http-nio-8088-exec-7，关联资源（库存扣减）QPS 超过阈值 5，触发当前资源（订单创建）限流
【关联流控-降级】当前线程：http-nio-8088-exec-12，关联资源（库存扣减）QPS 超过阈值 5，触发当前资源（订单创建）限流

```

5.Degrade

1.降级效果

```
1.rt
2.异常比率
3.异常数

其他的不再重复演示和flow的一样
```



1.controller

```java
package com.example.study.sentinelorigin.controller;

import com.alibaba.csp.sentinel.annotation.SentinelResource;
import com.example.study.sentinelorigin.constant.CommonConstant;
import com.example.study.sentinelorigin.handle.DegradeHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Random;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/10 0:47
 */
@RestController
@RequestMapping("/degrade")
public class DegradeController {

    private static final Random RANDOM = new Random();

    // ========== 业务方法（模拟慢响应，触发 RT 熔断） ==========
    @SentinelResource(
            value = CommonConstant.RT_DEGRADE_RESOURCE,
            blockHandlerClass = DegradeHandler.class,
            fallbackClass = DegradeHandler.class,
            blockHandler = "rtDegradeBlockHandler", // 熔断降级方法
            fallback = "rtDegradeFallback" // 业务异常兜底方法（可选）
    )
    @RequestMapping("/rt")
    public String doSlowBusiness() {
        // 模拟慢响应：睡眠 200 毫秒（超过 RT 阈值 100 毫秒）
```

```
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException("业务执行中断", e);
        }
        String format = String.format("【RT 熔断-成功】线程：%s，业务执行完成（慢响应）", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }


    // ========== 业务方法（模拟随机异常，触发异常比例熔断） ==========
    @SentinelResource(
            value = CommonConstant.RATIO_DEGRADE_RESOURCE,
            blockHandlerClass = DegradeHandler.class,
            fallbackClass = DegradeHandler.class,
            blockHandler = "ratioDegradeBlockHandler",
            fallback = "ratioDegradeFallback"
    )
    @RequestMapping("/exceptionRate")
    public String doRandomExceptionBusiness() {
        // 模拟 60% 概率抛出异常（超过异常比例阈值 0.5）
        if (RANDOM.nextDouble() > 0.4) {
            throw new RuntimeException("业务执行失败（随机异常）");
        }
        String format = String.format("【异常比例-成功】线程：%s，业务执行完成", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }


    // 异常计数器（模拟累计异常）
    private int exceptionCounter = 0;

    // ========== 业务方法（模拟累计异常，触发异常数熔断） ==========
    @SentinelResource(
            value = CommonConstant.COUNT_DEGRADE_RESOURCE,
            blockHandlerClass = DegradeHandler.class,
            fallbackClass = DegradeHandler.class,
            blockHandler = "countDegradeBlockHandler",
            fallback = "countDegradeFallback"
    )
    @RequestMapping("/exceptionCount")
    public String doAccumulateExceptionBusiness() {
        // 模拟每次调用都抛出异常，快速累计异常数（超过阈值 3）
        exceptionCounter++;
        throw new RuntimeException(String.format("业务执行失败（累计异常数：%d）", exceptionCounter));
    }


}

```

2.rule

```java
package com.example.study.sentinelorigin.rule;

import com.alibaba.csp.sentinel.slots.block.RuleConstant;
import com.alibaba.csp.sentinel.slots.block.degrade.DegradeRule;
import com.alibaba.csp.sentinel.slots.block.degrade.DegradeRuleManager;
import com.example.study.sentinelorigin.constant.CommonConstant;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.List;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/10 0:52
 */

@Component
public class DegradeRuleConfig {

    // ========== 配置 RT 熔断规则 ==========
    @PostConstruct
    public void initRtDegradeRule() {
        List<DegradeRule> rules = new ArrayList<>();
        DegradeRule rule = new DegradeRule();

        // 1. 绑定保护资源
        rule.setResource(CommonConstant.RT_DEGRADE_RESOURCE);
        // 2. 指定熔断方式：基于平均响应时间（RT）
        rule.setGrade(RuleConstant.DEGRADE_GRADE_RT);
        // 3. 配置 RT 阈值：100 毫秒（超过该值则计入慢请求）
        rule.setCount(200);
        // 4. 配置熔断窗口时间：5 秒（窗口内所有请求直接被拒绝）
        rule.setTimeWindow(5);
        // 5. 配置最小请求数：5（1 秒内请求数超过 5 才会触发熔断，默认 5）
        rule.setMinRequestAmount(5);
        // 6. 配置慢请求比例阈值：0.5（可选，慢请求占比超过 50% 才触发，默认 0.5）
        rule.setSlowRatioThreshold(0.5);

        rules.add(rule);

        DegradeRuleManager.loadRules(rules);
        System.out.println("===== 基于 RT 的熔断规则加载完成 =====");


        List<DegradeRule> degradeRules = DegradeRuleManager.getRules();
        System.out.println("before add size=" + degradeRules.size());


        DegradeRule rule2 = new DegradeRule();
        // 1. 绑定保护资源
        rule2.setResource(CommonConstant.RATIO_DEGRADE_RESOURCE);
        // 2. 指定熔断方式：基于异常比例
        rule2.setGrade(RuleConstant.DEGRADE_GRADE_EXCEPTION_RATIO);
```

```
        // 3. 配置异常比例阈值：0.5（50%，超过该比例则触发熔断）
        rule2.setCount(0.5);
        // 4. 配置熔断窗口时间：5 秒
        rule2.setTimeWindow(5);
        // 5. 配置最小请求数：5（1 秒内请求数超过 5 才会触发熔断）
        rule2.setMinRequestAmount(5);
        degradeRules.add(rule2);


        DegradeRule rule3 = new DegradeRule();

        // 1. 绑定保护资源
        rule3.setResource(CommonConstant.COUNT_DEGRADE_RESOURCE);
        // 2. 指定熔断方式：基于异常数
        rule3.setGrade(RuleConstant.DEGRADE_GRADE_EXCEPTION_COUNT);
        // 3. 配置异常数阈值：3（统计窗口内异常数超过 3 则触发熔断）
        rule3.setCount(3);
        // 4. 配置熔断窗口时间：5 秒
        rule3.setTimeWindow(5);
        // 5. 配置最小请求数：1（低 QPS 场景，降低最小请求数要求）
        rule3.setMinRequestAmount(1);
        degradeRules.add(rule3);

        DegradeRuleManager.loadRules(degradeRules);
        System.out.println("after add size=" +
DegradeRuleManager.getRules().size());
    }
}

```

3.handler

```
package com.example.study.sentinelorigin.handle;

import com.alibaba.csp.sentinel.slots.block.BlockException;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/10 0:50
 */
public class DegradeHandler {

    // ========== 熔断降级方法（熔断触发时调用） ==========
    public static String rtDegradeBlockHandler(BlockException e) {
        String format = String.format("【RT 熔断-触发】线程：%s，平均响应时间超过
阈值，进入熔断窗口", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ========== 业务异常兜底方法（可选，非熔断触发） ==========
    public static String rtDegradeFallback(Throwable e) {
        String format = String.format("【RT 熔断-兜底】线程：%s，业务执行异常：
%s", Thread.currentThread().getName(), e.getMessage());
```

```
        System.out.println(format);
        return format;
    }



    // ========== 熔断降级方法 ==========
    public static String ratioDegradeBlockHandler(BlockException e) {
        String format = String.format("【异常比例-触发】线程：%s，异常比例超过阈值，进入熔断窗口", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ========== 业务异常兜底方法 ==========
    public static String ratioDegradeFallback(Throwable e) {
        String format = String.format("【异常比例-兜底】线程：%s，业务执行异常：%s", Thread.currentThread().getName(), e.getMessage());
        System.out.println(format);
        return format;
    }



    // ========== 熔断降级方法 ==========
    public static String countDegradeBlockHandler(BlockException e) {
        String format = String.format("【异常数-触发】线程：%s，异常数超过阈值，进入熔断窗口", Thread.currentThread().getName());
        System.out.println(format);
        return format;
    }

    // ========== 业务异常兜底方法 ==========
    public static String countDegradeFallback(Throwable e) {
        String format = String.format("【异常数-兜底】线程：%s，业务执行异常：%s", Thread.currentThread().getName(), e.getMessage());
        System.out.println(format);
        return format;
    }
}

```

4.client

```
package com.example.study.sentinelorigin.client;

import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.junit.Test;

import java.nio.charset.StandardCharsets;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

```java
import java.util.concurrent.TimeUnit;

/**
 * @author lucksoul
 * @version 1.0
 * @date 2026/1/10 1:21
 */
public class DegradeTest {

    // 目标接口地址（替换为你自己的接口地址，对应 Sentinel 保护的接口）
    private static final String RT_TARGET_URL =
"http://localhost:8088/degrade/rt";


    @Test
    public void testRtDegradeResource() {
        // 创建固定线程池
        ExecutorService executorService =
Executors.newFixedThreadPool(20);

        // 记录开始时间
        long startTime = System.currentTimeMillis();
        // 提交并发请求任务
        for (int i = 0; i < 20; i++) {
            executorService.submit(() -> {
                for (int j = 0; j < 20; j++) {
                    // 1. 创建 CloseableHttpClient 实例（推荐使用
HttpClients.createDefault()）
                    try (CloseableHttpClient httpClient =
HttpClients.createDefault()) {
                        // 2. 构建 HTTP GET 请求（若接口是 POST，可使用
HttpPost）
                        HttpGet httpGet = new HttpGet(RT_TARGET_URL);
                        // 可选：设置请求头，模拟浏览器请求
                        httpGet.setHeader("User-Agent", "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36");
                        httpGet.setHeader("Accept", "application/json,
text/plain, */*");

                        // 3. 发送请求，获取响应
                        try (CloseableHttpResponse response =
httpClient.execute(httpGet)) {
                            // 4. 解析响应结果
                            int statusCode =
response.getStatusLine().getStatusCode();
                            String responseBody =
EntityUtils.toString(response.getEntity(), StandardCharsets.UTF_8);

                            // 5. 打印结果
                            System.out.println("=== 接口响应结果 ===");
                            System.out.println("响应状态码：" + statusCode);
                            System.out.println("响应内容：" + responseBody);
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                        System.out.println("请求失败：" + e.getMessage());
                    }
                }
```

```
            });
        }


        // 关闭线程池，等待所有任务执行完成
        executorService.shutdown();
        while (!executorService.isTerminated()) {}

        // 记录结束时间，打印耗时
        long endTime = System.currentTimeMillis();
        System.out.println("本次请求总耗时：" + (endTime - startTime) + " 毫秒");
    }


    // 目标接口地址（替换为你的接口地址，如 RT 熔断、异常比例熔断接口）
    private static final String EXCEPTION_RATE_TARGET_URL = "http://localhost:8088/degrade/exceptionRate";
    // 并发线程数（可调整，建议 10-20 个，确保触发熔断）
    private static final int EXCEPTION_RATE_THREAD_COUNT = 15;
    // 每个线程发送的请求数（可调整，确保 QPS 超过阈值）
    private static final int EXCEPTION_RATE_REQUEST_PER_THREAD = 10;

    @Test
    public void testExceptionRate() {
        // 1. 创建固定大小线程池
        ExecutorService executorService = Executors.newFixedThreadPool(EXCEPTION_RATE_THREAD_COUNT);
        System.out.println("=== 高并发请求开始，线程数：" + EXCEPTION_RATE_THREAD_COUNT + "，每个线程请求数：" + EXCEPTION_COUNT_REQUEST_PER_THREAD + " ===");

        // 2. 提交线程任务
        for (int i = 0; i < EXCEPTION_RATE_THREAD_COUNT; i++) {
            int threadNum = i + 1;
            executorService.submit(() -> {
                // 每个线程创建一个 HttpClient 实例（线程安全，可复用）
                try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
                    for (int j = 0; j < EXCEPTION_RATE_REQUEST_PER_THREAD; j++) {
                        int requestNum = j + 1;
                        try {
                            // 构建 GET 请求
                            HttpGet httpGet = new HttpGet(EXCEPTION_RATE_TARGET_URL);
                            httpGet.setHeader("User-Agent", "HighConcurrentHttpClient/1.0");

                            // 发送请求并获取响应
                            try (CloseableHttpResponse response = httpClient.execute(httpGet)) {
                                // 解析响应
                                String responseBody = EntityUtils.toString(response.getEntity(), StandardCharsets.UTF_8);
                                int statusCode = response.getStatusLine().getStatusCode();

```

```
                                // 打印请求结果（可选，可注释以减少输出干扰）
                                System.out.printf("线程 %d - 请求 %d：状态码 %d，响应内容：%s%n",
                                        threadNum, requestNum, statusCode, responseBody);

                                // 轻微休眠，避免请求过于密集导致接口宕机（可选，根据场景调整）
                                TimeUnit.MILLISECONDS.sleep(50);
                            }
                        } catch (Exception e) {
                            System.out.printf("线程 %d - 请求 %d：失败，原因：%s%n",
                                    threadNum, requestNum, e.getMessage());
                        }
                    }
                } catch (Exception e) {
                    System.out.printf("线程 %d：初始化 HttpClient 失败，原因：%s%n", threadNum, e.getMessage());
                }
            });
        }

        // 3. 关闭线程池，等待所有任务完成
        executorService.shutdown();
        try {
            boolean allTaskCompleted = executorService.awaitTermination(5, TimeUnit.MINUTES);
            if (allTaskCompleted) {
                System.out.println("=== 所有高并发请求执行完成 ===");
            } else {
                System.out.println("=== 部分请求超时未完成 ===");
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
            executorService.shutdownNow();
        }
    }


    // 目标接口地址（替换为你的接口地址，如 RT 熔断、异常比例熔断接口）
    private static final String EXCEPTION_COUNT_TARGET_URL = "http://localhost:8088/degrade/exceptionCount";
    // 并发线程数（可调整，建议 10-20 个，确保触发熔断）
    private static final int EXCEPTION_COUNT_THREAD_COUNT = 15;
    // 每个线程发送的请求数（可调整，确保 QPS 超过阈值）
    private static final int EXCEPTION_COUNT_REQUEST_PER_THREAD = 10;

    @Test
    public void testExceptionCount() {
        // 1. 创建固定大小线程池
        ExecutorService executorService = Executors.newFixedThreadPool(EXCEPTION_COUNT_THREAD_COUNT);
        System.out.println("=== 高并发请求开始，线程数：" + EXCEPTION_COUNT_THREAD_COUNT + "，每个线程请求数：" + EXCEPTION_COUNT_REQUEST_PER_THREAD + " ===");

```

```
        // 2. 提交线程任务
        for (int i = 0; i < EXCEPTION_COUNT_THREAD_COUNT; i++) {
            int threadNum = i + 1;
            executorService.submit(() -> {
                // 每个线程创建一个 HttpClient 实例（线程安全，可复用）
                try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
                    for (int j = 0; j < EXCEPTION_COUNT_REQUEST_PER_THREAD; j++) {
                        int requestNum = j + 1;
                        try {
                            // 构建 GET 请求
                            HttpGet httpGet = new HttpGet(EXCEPTION_COUNT_TARGET_URL);
                            httpGet.setHeader("User-Agent", "HighConcurrentHttpClient/1.0");

                            // 发送请求并获取响应
                            try (CloseableHttpResponse response = httpClient.execute(httpGet)) {
                                // 解析响应
                                String responseBody = EntityUtils.toString(response.getEntity(), StandardCharsets.UTF_8);
                                int statusCode = response.getStatusLine().getStatusCode();

                                // 打印请求结果（可选，可注释以减少输出干扰）
                                System.out.printf("线程 %d - 请求 %d：状态码 %d，响应内容：%s%n",
                                        threadNum, requestNum, statusCode, responseBody);

                                // 轻微休眠，避免请求过于密集导致接口宕机（可选，根据场景调整）
                                TimeUnit.MILLISECONDS.sleep(50);
                            }
                        } catch (Exception e) {
                            System.out.printf("线程 %d - 请求 %d：失败，原因：%s%n",
                                    threadNum, requestNum, e.getMessage());
                        }
                    }
                } catch (Exception e) {
                    System.out.printf("线程 %d：初始化 HttpClient 失败，原因：%s%n", threadNum, e.getMessage());
                }
            });
        }

        // 3. 关闭线程池，等待所有任务完成
        executorService.shutdown();
        try {
            boolean allTaskCompleted = executorService.awaitTermination(5, TimeUnit.MINUTES);
            if (allTaskCompleted) {
                System.out.println("=== 所有高并发请求执行完成 ===");
            } else {
```

```
                System.out.println("=== 部分请求超时未完成 ===");
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
            executorService.shutdownNow();
        }
    }
}

```

5.constant

```
/**
     * 熔断
     */
    // 资源名
    public static final String RT_DEGRADE_RESOURCE = "rtDegradeResource";

    public static final String RATIO_DEGRADE_RESOURCE =
"ratioDegradeResource";

    public static final String COUNT_DEGRADE_RESOURCE =
"countDegradeResource";
```

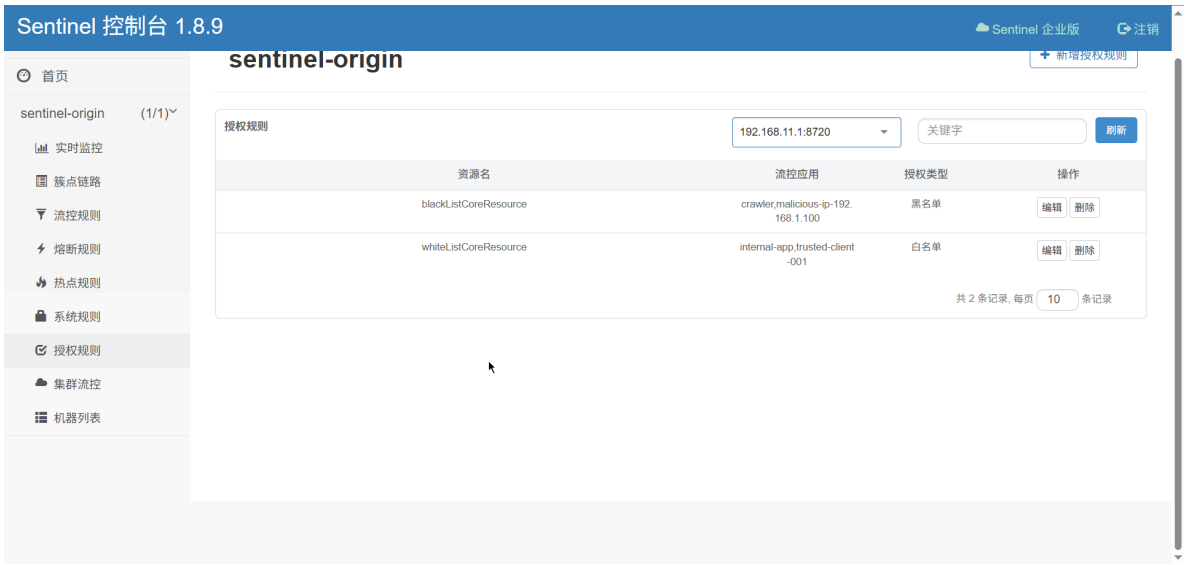
6.效果

```
1.rt
【异常比例-成功】线程：http-nio-8088-exec-5，业务执行完成
【异常比例-成功】线程：http-nio-8088-exec-4，业务执行完成
【异常比例-兜底】线程：http-nio-8088-exec-1，业务执行异常：业务执行失败（随机异常）
【异常比例-兜底】线程：http-nio-8088-exec-2，业务执行异常：业务执行失败（随机异常）
【异常比例-兜底】线程：http-nio-8088-exec-3，业务执行异常：业务执行失败（随机异常）
【异常比例-触发】线程：http-nio-8088-exec-8，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-7，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-6，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-9，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-10，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-2，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-1，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-4，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-3，异常比例超过阈值，进入熔断窗口

2.异常比率
【异常比例-兜底】线程：http-nio-8088-exec-3，业务执行异常：业务执行失败（随机异常）
【异常比例-触发】线程：http-nio-8088-exec-7，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-10，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-6，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-6，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-8，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-8，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-9，异常比例超过阈值，进入熔断窗口
```

```
【异常比例-触发】线程：http-nio-8088-exec-5，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-5，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-3，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-3，异常比例超过阈值，进入熔断窗口
【异常比例-触发】线程：http-nio-8088-exec-3，异常比例超过阈值，进入熔断窗口

3.异常数
【异常数-兜底】线程：http-nio-8088-exec-8，业务执行异常：业务执行失败（累计异常数：2）
【异常数-兜底】线程：http-nio-8088-exec-4，业务执行异常：业务执行失败（累计异常数：3）
【异常数-兜底】线程：http-nio-8088-exec-5，业务执行异常：业务执行失败（累计异常数：1）
【异常数-兜底】线程：http-nio-8088-exec-7，业务执行异常：业务执行失败（累计异常数：4）
【异常数-触发】线程：http-nio-8088-exec-3，异常数超过阈值，进入熔断窗口
【异常数-触发】线程：http-nio-8088-exec-9，异常数超过阈值，进入熔断窗口
【异常数-触发】线程：http-nio-8088-exec-2，异常数超过阈值，进入熔断窗口
【异常数-触发】线程：http-nio-8088-exec-6，异常数超过阈值，进入熔断窗口
【异常数-触发】线程：http-nio-8088-exec-6，异常数超过阈值，进入熔断窗口
【异常数-触发】线程：http-nio-8088-exec-10，异常数超过阈值，进入熔断窗口
【异常数-触发】线程：http-nio-8088-exec-8，异常数超过阈值，进入熔断窗口
```

6.Authority(权限)

```
黑白名单限制
1.自定义RequestOriginParser
一般是在请求头中特殊标记，或者指定特定的域名，ip进行限制
```



1.白名单

```
直接看项目提交记录查看文件变动

```

1.效果

```
【白名单模式-成功】资源：whiteListCoreResource，请求允许访问
【白名单模式-成功】资源：whiteListCoreResource，请求允许访问
【白名单模式-拒绝】资源：whiteListCoreResource，来源不在白名单中，禁止访问
```

2.黑名单

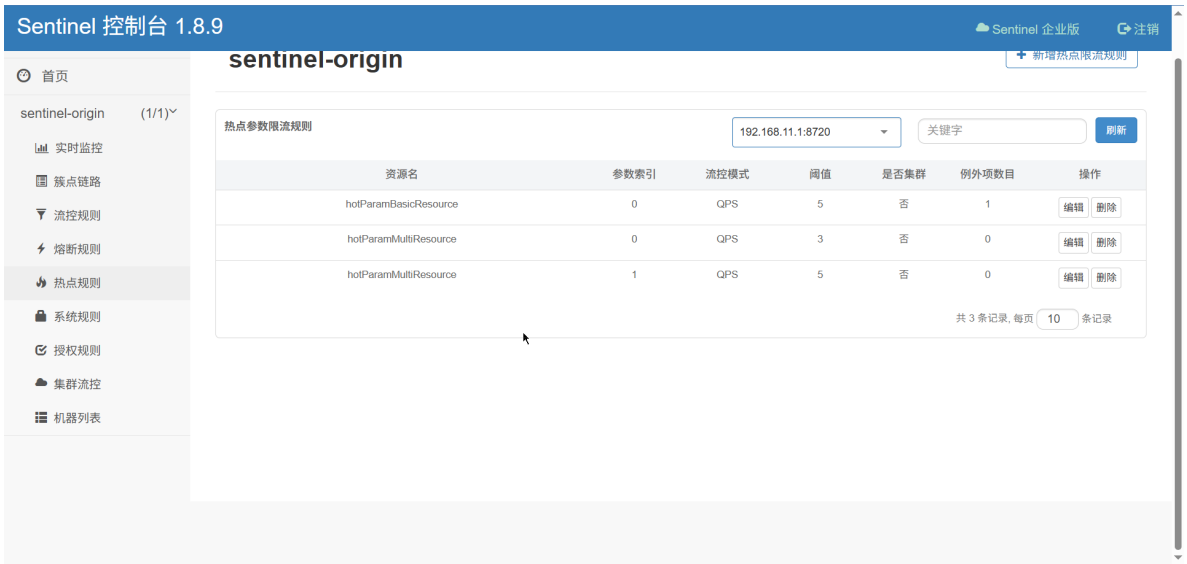
```
直接看项目提交记录查看文件变动
```

1.效果

```
【黑名单模式-拒绝】资源：blackListCoreResource，来源在黑名单中，禁止访问
【黑名单模式-拒绝】资源：blackListCoreResource，来源在黑名单中，禁止访问
【黑名单模式-成功】资源：blackListCoreResource，请求允许访问
```

7.hotparam(热点参数)

```
1.单个参数
2.额外参数
3.多个参数分别限制

直接看项目提交记录查看文件变动
```



1.单个

```
【成功】查询商品，productId：1001，请求正常处理
【成功】查询商品，productId：1001，请求正常处理
【成功】查询商品，productId：1001，请求正常处理
【成功】查询商品，productId：1001，请求正常处理
【成功】查询商品，productId：1001，请求正常处理
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
【限流】商品查询触发热点限流，productId：1001，原因：ParamFlowException
```

2.额外

```
全是成功的
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
```

```
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理
【成功】查询商品，productId：999，请求正常处理

```

3.多个参数

```
2026-01-10 19:13:23.727  INFO 2628 --- [nio-8088-exec-9] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2026-01-10 19:13:23.727  INFO 2628 --- [nio-8088-exec-9] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2026-01-10 19:13:23.728  INFO 2628 --- [nio-8088-exec-9] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
【成功】查询订单，orderId：111，userId：luck
【成功】查询订单，orderId：111，userId：luck
【成功】查询订单，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
```

```
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
```

```
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【限流】订单查询触发热点限流，orderId：111，userId：luck
【成功】查询订单，orderId：111，userId：luck
【成功】查询订单，orderId：111，userId：luck
【成功】查询订单，orderId：111，userId：luck

```