

sentinel-core & dashboard

```
1 sentinel java编码的方式设定规则  
2 控制台的下载使用  
3  
4 https://github.com/fanda521/sentinel-study-project
```

1.官方地址

```
1 1. 官方文档地址  
2 https://sentinelguard.io/zh-cn/docs/basic-implementation.html  
3  
4 2.dashboard地址  
5 https://github.com/alibaba/sentinel/releases
```

2.sentinel-core学习(编码式)

1.小试牛刀

```
1 | 使用流控进行实验
```

1.pom

```
1 <!-- 1. Spring Web 核心依赖（提供 web 接口能力） -->  
2     <dependency>  
3         <groupId>org.springframework.boot</groupId>  
4         <artifactId>spring-boot-starter-web</artifactId>  
5     </dependency>  
6  
7     <!-- 2. Sentinel 核心依赖（核心限流/熔断逻辑） -->  
8     <dependency>  
9         <groupId>com.alibaba.csp</groupId>  
10        <artifactId>sentinel-core</artifactId>  
11        <version>${sentinel.version}</version>  
12    </dependency>  
13  
14    <!-- 3. Sentinel 注解支持核心依赖（新增！支持 @SentinelResource） -->  
15    <dependency>  
16        <groupId>com.alibaba.csp</groupId>  
17        <artifactId>sentinel-annotation-aspectj</artifactId>  
18        <version>${sentinel.version}</version>  
19    </dependency>  
20  
21    <!-- 4. Sentinel 控制台通信依赖（可选，连接 Sentinel 控制台做可视化配置） -  
->  
22    <dependency>  
23        <groupId>com.alibaba.csp</groupId>  
24        <artifactId>sentinel-transport-simple-http</artifactId>  
25        <version>${sentinel.version}</version>  
26    </dependency>  
27
```

```

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

```

2.yml

```

1  spring:
2      application:
3          name: sentinel-origin # 应用名称（会显示在 Sentinel 控制台）
4  server:
5      port: 8088 # 项目端口（避免与 Dashboard 8080 冲突）
6
7  # Sentinel 核心配置
8  sentinel:
9      # 控制台连接配置
10     transport:
11         dashboard: 127.0.0.1:8080 # Dashboard 地址（若改了端口则填对应端口，如
12             127.0.0.1:8858）
13         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）
14         client-ip: 127.0.0.1 # 客户端 IP（多网卡场景需指定，单机默认即可）
15         # 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）
16         eager: true

```

3.controller

```

1 package com.example.study.sentinelorigin.controller;
2
3 import com.alibaba.csp.sentinel.Entry;
4 import com.alibaba.csp.sentinel.SphU;
5 import com.alibaba.csp.sentinel.slots.block.BlockException;
6 import com.alibaba.csp.sentinel.slots.block.RuleConstant;
7 import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
8 import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import javax.annotation.PostConstruct;
13 import java.util.ArrayList;
14 import java.util.List;
15
16 /**
17 * @author lucksoul
18 * @version 1.0
19 * @date 2026/1/9 1:22
20 */
21 @RestController
22 @RequestMapping("/hello")
23 public class HelloController {

```

```

24
25
26     @RequestMapping("/sayHello")
27     public String hello() {
28
29         Entry entry = null;
30         // 务必保证finally会被执行
31         try {
32             // 资源名可使用任意有业务语义的字符串
33             entry = SphU.entry("hello");
34             // 被保护的业务逻辑
35             // do something...
36             System.out.println("hello world");
37         } catch (BlockException e1) {
38             // 资源访问阻止，被限流或被降级
39             // 进行相应的处理操作
40             System.out.println("限流");
41             return "限流";
42         } finally {
43             if (entry != null) {
44                 entry.exit();
45             }
46         }
47         return "hello world";
48     }
49
50     @PostConstruct
51     private static void initFlowQpsRule() {
52         List<FlowRule> rules = new ArrayList<>();
53         FlowRule rule1 = new FlowRule();
54         rule1.setResource("hello");
55         // Set max qps to 20
56         rule1.setCount(1);
57         rule1.setGrade(RuleConstant.FLOW_GRADE_QPS);
58         rules.add(rule1);
59         FlowRuleManager.loadRules(rules);
60     }
61
62
63 }
64

```

4. 测试效果

1 | 一秒内调用多次就触发限流



5. dashboard

```

1 | 1. 下载好jar后
2 | 执行 java -jar sentinel-dashboard-1.8.9.jar
3 |
4 | 2. 启动刚写的服务8088的
5 |
6 | 3. 观察dashboard
7 | http://localhost:8080
8 | 默认账号密码
9 | sentinel/sentinel

```

```

Microsoft Windows [版本 10.0.19045.6466]
(c) Microsoft Corporation。保留所有权利。

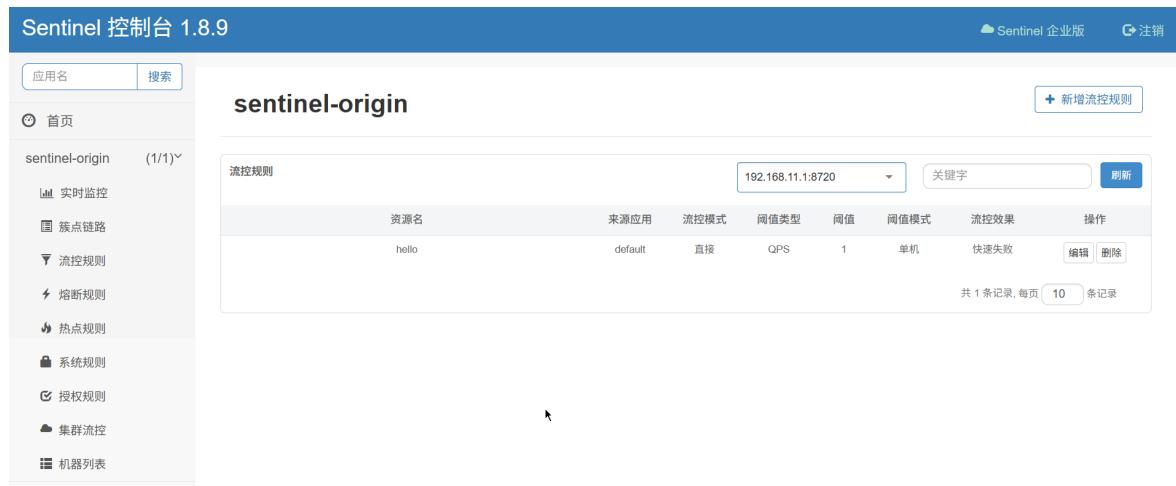
S:\Professions\sentinel>java -jar sentinel-dashboard-1.8.9.jar
INFO: Sentinel log output type is: file
INFO: Sentinel log charset is: utf-8
INFO: Sentinel log base directory is: C:\Users\10560\logs\csp\
INFO: Sentinel log name use pid is: false
INFO: Sentinel log level is: INFO


:: Spring Boot ::      (v2.5.12)

2026-01-09 01:35:31.696  INFO 43044 --- [           main] c.a.c.s.dashboard.DashboardApplication : Starting DashboardA
pplication using Java 1.8.0_191 on DESKTOP-GMCV2CN with PID 43044 (S:\Professions\sentinel\sentinel-dashboard-1.8.9.jar
started by 10560 in S:\Professions\sentinel)
2026-01-09 01:35:31.701  INFO 43044 --- [           main] c.a.c.s.dashboard.DashboardApplication : No active profile s
et, falling back to 1 default profile: "default"
2026-01-09 01:35:35.482  INFO 43044 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized
with port(s): 8080 (http)
2026-01-09 01:35:35.504  INFO 43044 --- [           main] o.apache.catalina.core.StandardService : Starting service [T
omcat]
2026-01-09 01:35:35.505  INFO 43044 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet en
gine: [Apache Tomcat/9.0.60]
2026-01-09 01:35:35.722  INFO 43044 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]          : Initializing Spring

```

1 | 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



流控规则							
	资源名	来源应用	流控模式	阈值类型	阈值	间值模式	流控效果
	hello	default	直接	QPS	1	单机	快速失败

2.注解的方式

```

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

```

```
5             <artifactId>sentinel-annotation-aspectj</artifactId>
6             <version>${sentinel.version}</version>
7         </dependency>
8
9     2.配置bean
10    package com.example.study.sentinelorigin.config;
11
12    import com.alibaba.csp.sentinel.annotation.aspectj.SentinelResourceAspect;
13    import org.springframework.context.annotation.Bean;
14    import org.springframework.context.annotation.Configuration;
15
16    /**
17     * @author lucksoul
18     * @version 1.0
19     * @date 2026/1/9 2:39
20     */
21    @Configuration
22    public class SentinelSourceConfig {
23
24        @Bean
25        public SentinelResourceAspect sentinelResourceAspect() {
26            return new SentinelResourceAspect();
27        }
28    }
29
30
31    3.在目标方法加上注解和定义对应的blockHandler方法
32    package com.example.study.sentinelorigin.controller;
33
34    import com.alibaba.csp.sentinel.Entry;
35    import com.alibaba.csp.sentinel.SphU;
36    import com.alibaba.csp.sentinel.annotation.SentinelResource;
37    import com.alibaba.csp.sentinel.slots.block.BlockException;
38    import com.alibaba.csp.sentinel.slots.block.RuleConstant;
39    import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
40    import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
41    import org.springframework.web.bind.annotation.RequestMapping;
42    import org.springframework.web.bind.annotation.RestController;
43
44    import javax.annotation.PostConstruct;
45    import java.util.ArrayList;
46    import java.util.List;
47
48    /**
49     * @author lucksoul
50     * @version 1.0
51     * @date 2026/1/9 2:29
52     */
53    @RestController
54    @RequestMapping("/anno")
55    public class AnnoController {
56        @RequestMapping("/sayHello")
57        @SentinelResource(value = "anno-hello", blockHandler =
58        "annoHandleException")
59        public String hello() {
60            System.out.println("anno world");
61            return "anno-hello";
62        }
63    }
```

```

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.fallback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;

```

```

5 import com.alibaba.csp.sentinel.annotation.SentinelResource;
6 import com.alibaba.csp.sentinel.slots.block.BlockException;
7 import com.alibaba.csp.sentinel.slots.block.RuleConstant;
8 import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
9 import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
10 import com.example.study.sentinelorigin.handle.FailBlockHandler;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 import javax.annotation.PostConstruct;
15 import java.util.ArrayList;
16 import java.util.List;
17
18 /**
19 * @author lucksoul
20 * @version 1.0
21 * @date 2026/1/9 2:21
22 */
23 @RestController
24 @RequestMapping("/failBlock")
25 public class FailBackAndBlockHandlerDefaultClass {
26
27
28     @RequestMapping("/test01")
29     @SentinelResource(value = "failBlock-test01", blockHandlerClass =
FailBlockHandler.class ,blockHandler = "failBlockTest01")
30     public String test01() {
31         return "failBlock-test01";
32     }
33 }
34

```

2.异常处理类

```

1 package com.example.study.sentinelorigin.handle;
2
3 import com.alibaba.csp.sentinel.slots.block.BlockException;
4
5 /**
6 * @author lucksoul
7 * @version 1.0
8 * @date 2026/1/9 2:54
9 */
10 public class FailBlockHandler {
11
12     public static String failBlockTest01(BlockException e) {
13         e.printStackTrace();
14         System.out.println("failBlockHandler-限流");
15         return "failBlockHandler-限流";
16     }
17 }
18

```

4.flow

1.thread

```
1 | 有两种方式
2 | 1.qps
3 | 2.线程并发数
4 |
5 | 这里就是测试线程并发数
```

1.controller

```
1 | @GetMapping("/annoThread")
2 |     @SentinelResource(
3 |         value = CommonConstant.THREAD_RESOURCE_NAME, // 绑定资源名, 与流控
规则中的资源名一致
4 |         blockHandlerClass = FlowHandler.class,
5 |         blockHandler = "threadFlowBlockHandler" // 指定限流降级方法 (局部)
6 |     )
7 |     public String testAnnotationThreadFlow() throws InterruptedException {
8 |         // 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)
9 |         TimeUnit.SECONDS.sleep(3);
10 |
11 |         // 正常响应结果
12 |         return String.format("【成功】当前线程: %s, 业务执行完成",
13 |             Thread.currentThread().getName());
14 |     }
15 | }
```

2.rule

```
1 | package com.example.study.sentinelorigin.rule;
2 |
3 | import com.alibaba.csp.sentinel.slots.block.RuleConstant;
4 | import com.alibaba.csp.sentinel.slots.block.flow.FlowRule;
5 | import com.alibaba.csp.sentinel.slots.block.flow.FlowRuleManager;
6 | import com.example.study.sentinelorigin.constant.CommonConstant;
7 | import org.springframework.stereotype.Component;
8 |
9 | import javax.annotation.PostConstruct;
10 | import java.util.ArrayList;
11 | import java.util.List;
12 |
13 | /**
14 | * @author lucksoul
15 | * @version 1.0
16 | * @date 2026/1/9 16:49
17 | */
18 | @Component
19 | public class FlowRuleConfig {
20 |
21 |     @PostConstruct
22 |     public void initAnnotationThreadFlowRules() {
23 |         FlowRule rule = new
com.alibaba.csp.sentinel.slots.block.flow.FlowRule();
24 |         rule.setResource(CommonConstant.THREAD_RESOURCE_NAME); // 绑定注解对
应的资源名
25 |         rule.setGrade(RuleConstant.FLOW_GRADE_THREAD); // 限流类型: 并发线程数
(核心)
```

```

26     rule.setCount(5); // 最大并发线程数阈值: 5
27     rule.setLimitApp("default"); // 针对默认应用限流
28
29     // 步骤1: 读取现有已加载的规则 (转为可修改列表)
30     List<FlowRule> existingRules = new ArrayList<>
31         (FlowRuleManager.getRules());
32     System.out.println("追加前, 现有规则数: " + existingRules.size());
33     // 步骤3: 调用loadRules()重新加载 (实现追加效果)
34     existingRules.add(rule);
35     FlowRuleManager.loadRules(existingRules);
36     System.out.println("追加后, 当前生效规则数: " +
37     FlowRuleManager.getRules().size());
38 }
```

3.handle

```

1 package com.example.study.sentinelorigin.handle;
2
3 import com.alibaba.csp.sentinel.slots.block.BlockException;
4
5 /**
6 * @author lucksoul
7 * @version 1.0
8 * @date 2026/1/9 16:49
9 */
10 public class FlowHandler {
11
12     // 步骤 3: 注解指定的降级方法 (blockHandler 要求)
13     /**
14      * 1. 方法权限: public (必须)
15      * 2. 返回值: 与原方法一致 (必须)
16      * 3. 参数: 与原方法一致 + 末尾追加 BlockException (必须)
17      * 4. 若原方法无异常抛出, 降级方法可仅追加 BlockException
18      */
19     public static String threadFlowBlockHandler(BlockException e) {
20         return String.format("【降级】当前线程: %s, 并发线程数超过阈值 5, 拒绝访问",
21             Thread.currentThread().getName());
22     }
23 }
```

4.client

```

1 package com.example.study.sentinelorigin.client;
2
3 import org.apache.http.client.methods.HttpGet;
4 import org.apache.http.impl.client.CloseableHttpClient;
5 import org.apache.http.impl.client.HttpClients;
6 import org.apache.http.util.EntityUtils;
7 import org.junit.Test;
8
```

```

9 import java.util.concurrent.ExecutorService;
10 import java.util.concurrent.Executors;
11 import java.util.concurrent.TimeUnit;
12
13 public class FlowTest {
14     private static final int THREAD_NUM = 20;
15     private static final String TEST_FLOW_THREAD_URL =
16         "http://localhost:8088/flow/annoThread";
17
18     @Test
19     public void testFlowThread() throws InterruptedException {
20         ExecutorService executorService =
21             Executors.newFixedThreadPool(THREAD_NUM);
22         for (int i = 0; i < THREAD_NUM; i++) {
23             executorService.submit(() -> {
24                 try (CloseableHttpClient httpClient =
25                     HttpClients.createDefault()) {
26                    HttpGet httpGet = new HttpGet(TEST_FLOW_THREAD_URL);
27                     String response =
28                         EntityUtils.toString(httpClient.execute(httpGet).getEntity());
29                     System.out.println(response);
30                 } catch (Exception e) {
31                     e.printStackTrace();
32                 }
33             });
34             Thread.sleep(10000);
35             executorService.shutdown();
36         }
37     }

```

5.constant

```

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

```

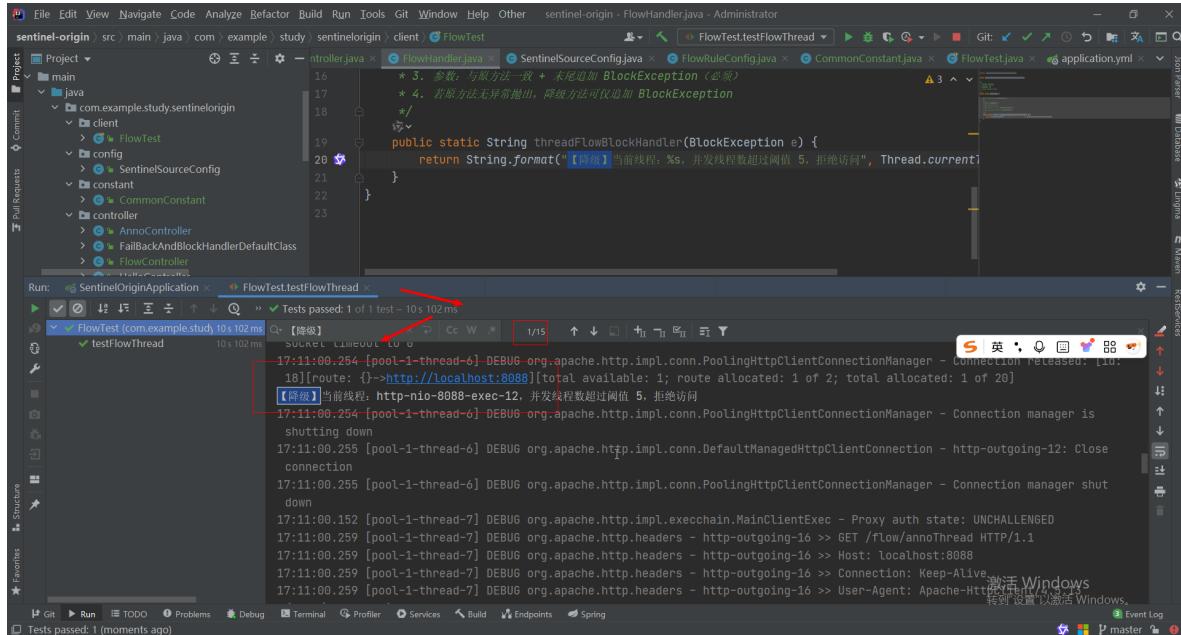
6.pom

```

1 <!-- CloseableHttpClient 核心依赖（新增） -->
2     <dependency>
3         <groupId>org.apache.httpcomponents</groupId>
4         <artifactId>httpclient</artifactId>
5         <version>4.5.13</version>
6     </dependency>
7     <dependency>
8         <groupId>junit</groupId>
9         <artifactId>junit</artifactId>
10    </dependency>

```

7.效果



2.limitApp

1.controller

```
1 // 步骤 2: 注解式测试接口
2 @GetMapping("/limitApp")
3 @SentinelResource(
4     value = CommonConstant.LIMIT_APP_RESOURCE,
5     blockHandlerClass = FlowHandler.class,
6     blockHandler = "limitAppBlockHandler"
7 )
8 public String testLimitApp() throws InterruptedException {
9     Thread.sleep(2000);
10    String format = String.format("【成功】当前线程: %s, 请求正常执行",
11        Thread.currentThread().getName());
12    System.out.println(format);
13    return format;
}
```

2.rule

```
1 // limitApp
2 // 规则 1: 仅对应用 "appA" 进行 QPS 限流 (阈值 2)
3 FlowRule appARule = new FlowRule();
4 appARule.setResource(CommonConstant.LIMIT_APP_RESOURCE); // 绑定资源
5 appARule.setGrade(RuleConstant.FLOW_GRADE_THREAD); // QPS 限流
6 appARule.setCount(2); // 每秒最多 2 个请求
7 appARule.setLimitApp("appA"); // 仅对 appA 生效 (核心配置)
8
9 // 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)
10 FlowRule otherRule = new FlowRule();
11 otherRule.setResource(CommonConstant.LIMIT_APP_RESOURCE);
```

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

3. handle

```
1 public static String limitAppBlockHandler(BlockException e) {
2     String format = String.format("【降级】当前线程: %s, 请求过于频繁, 触发
3     limitApp 限流", Thread.currentThread().getName());
4     System.out.println(format);
5     return format;
6 }
```

4.client

```
1  @Test
2      public void testLimitApp() throws InterruptedException {
3          System.out.println("===== 场景 1: 测试来源 appA (阈值 2)
4          =====");
5          testLimitApp("appA");
6
7          // 间隔 5 秒, 让 Sentinel 重置 QPS 统计
8          Thread.sleep(5000);
9
10         System.out.println("\n===== 场景 2: 测试来源 appB (匹配 other 规
11        则, 阈值 5) =====");
12         testLimitApp("other");
13
14         // 间隔 5 秒
15         Thread.sleep(5000);
16
17         System.out.println("\n===== 场景 3: 测试无来源 (匹配 other 规则, 阈
18        值 5) =====");
19         testLimitApp(null);
20     }
21
22     /**
23      * 模拟指定来源的高并发请求
24      * @param appName 应用来源 (null 表示无来源)
25      */
26     private static void testLimitApp(String appName) {
27         // 创建固定线程池
28         ExecutorService executorService =
29             Executors.newFixedThreadPool(CONCURRENT_THREADS);
30
31         // 提交并发请求任务
32         for (int i = 0; i < CONCURRENT_THREADS; i++) {
33             executorService.submit(() -> {
34                 try (CloseableHttpClient httpClient =
35                     HttpClientClients.createDefault()) {
36
37                     // 提交请求逻辑
38
39                 }
40             });
41         }
42     }
43 }
```

```

31         // 构建 GET 请求
32         HttpGet httpGet = new HttpGet(TEST_URL);
33
34         // 若指定了应用来源，添加请求头 X-Sentinel-App
35         if (appName != null && !appName.isEmpty()) {
36             httpGet.addHeader("X-Sentinel-App", appName);
37         }
38
39         // 执行请求并获取响应结果
40         String response =
41             EntityUtils.toString(httpClient.execute(httpGet).getEntity(), "UTF-8");
42         System.out.println(response);
43     } catch (Exception e) {
44         e.printStackTrace();
45     }
46 }
47
48 // 关闭线程池，等待所有任务执行完成
49 executorService.shutdown();
50 while (!executorService.isTerminated()) {
51     // 等待任务结束
52 }
53 }
```

5.constant

```

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

6.parser

```

1
2 package com.example.study.sentinelorigin.config;
3
4 import
5 com.alibaba.csp.sentinel.adapter.servlet.callback.RequestOriginParser;
6 import org.springframework.stereotype.Component;
7
8 import javax.servlet.http.HttpServletRequest;
9
10 /**
11 * 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)
12 */
13 @Component
14 public class CustomRequestOriginParser implements RequestOriginParser {
15
16     public CustomRequestOriginParser() {
17         System.out.println("===== CustomRequestOriginParser 被 Spring
实例化了 =====");
18     }
19
20     @Override
```

```

20     public String parseOrigin(HttpServletRequest request) {
21         // 从请求头中获取应用来源（可改为从请求参数、Cookie 等提取）
22         String appName = request.getHeader("X-Sentinel-App");
23         // 若请求头中无该字段，默认返回 "unknown"
24         System.out.println("请求来源: " + appName);
25         return appName == null ? "unknown" : appName;
26     }
27 }
```

7.commonFilter

```

1 package com.example.study.sentinelorigin.config;
2
3 import com.alibaba.csp.sentinel.adapter.servlet.CommonFilter;
4 import org.springframework.boot.web.servlet.FilterRegistrationBean;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 import javax.servlet.DispatcherType;
9
10 /**
11 * 手动注册 Sentinel CommonFilter，确保 web 请求被 Sentinel 拦截
12 */
13 @Configuration
14 public class SentinelFilterConfig {
15
16     @Bean
17     public FilterRegistrationBean<CommonFilter>
sentinelCommonFilterRegistration() {
18         FilterRegistrationBean<CommonFilter> registrationBean = new
FilterRegistrationBean<>();
19         // 注册 Sentinel CommonFilter
20         registrationBean.setFilter(new CommonFilter());
21         // 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）
22         registrationBean.addUrlPatterns("/*");
23         // 设置过滤器顺序（优先级高于其他过滤器，确保先被执行）
24         registrationBean.setOrder(1);
25         // 匹配所有请求分发类型（包括直接请求、转发、包含等）
26         registrationBean.setDispatcherTypes(DispatcherType.REQUEST,
DispatcherType.FORWARD);
27         // 启用该过滤器
28         registrationBean.setEnabled(true);
29         return registrationBean;
30     }
31 }
```

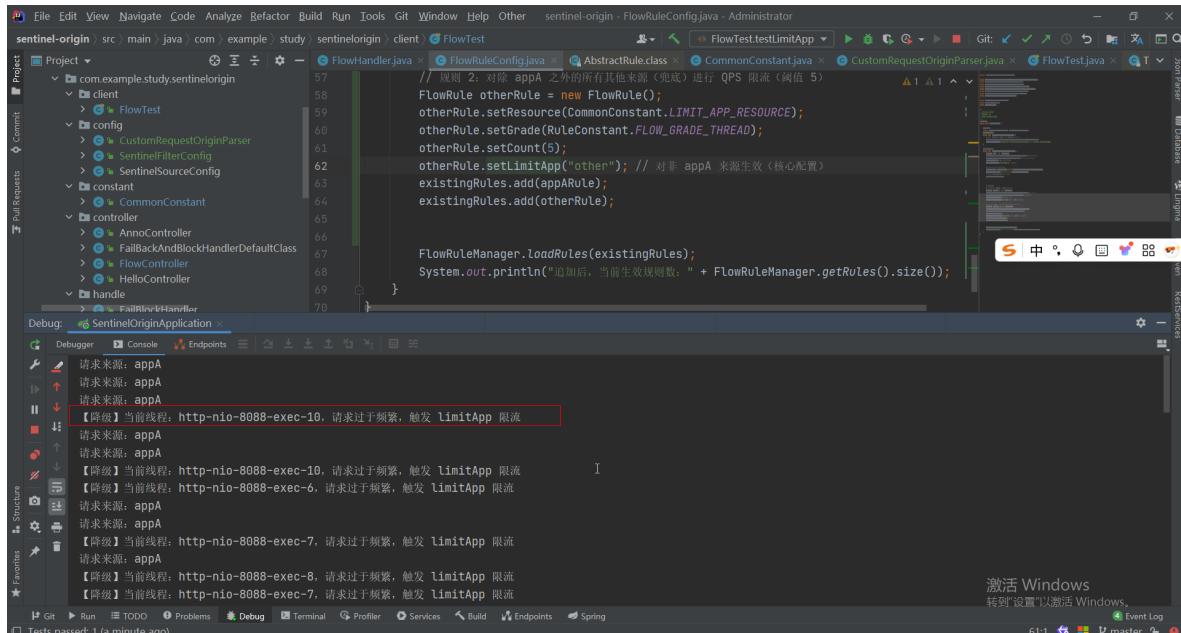
8.pom

```

1      <!-- 新增: Sentinel web servlet 适配依赖 (包含 RequestOriginParser 类) --
2
3      <dependency>
4          <groupId>com.alibaba.csp</groupId>
5          <artifactId>sentinel-web-servlet</artifactId>
6          <version>1.8.6</version>
7      </dependency>

```

9.效果



3.流控效果策略

1 | 拒绝/排队等待/预热启动

1.controller

```

1 // 接口 1: 测试 直接拒绝 效果
2     @GetMapping("/strategy/default")
3     @SentinelResource(
4         value = CommonConstant.DEFAULT_RESOURCE,
5         blockHandlerClass = FlowHandler.class,
6         blockHandler = "defaultBlockHandler"
7     )
8     public String testDefaultControlBehavior() {
9         String format = String.format("【直接拒绝-成功】当前线程: %s, 请求正常执
10        行", Thread.currentThread().getName());
11         System.out.println(format);
12         return format;
13     }
14
15 // 接口 2: 测试 预热/冷启动 效果
16     @GetMapping("/strategy/warmup")

```

```

17     @SentinelResource(
18         value = CommonConstant.WARM_UP_RESOURCE,
19         blockHandlerClass = FlowHandler.class,
20         blockHandler = "warmUpBlockHandler"
21     )
22     public String testWarmUpControlBehavior() {
23         String format = String.format("【预热-成功】当前线程: %s, 请求正常执行",
24             Thread.currentThread().getName());
25         System.out.println(format);
26         return format;
27     }
28
29     // 接口 3: 测试 匀速排队 效果
30     @GetMapping("/strategy/ratelimiter")
31     @SentinelResource(
32         value = CommonConstant.RATE_LIMITER_RESOURCE,
33         blockHandlerClass = FlowHandler.class,
34         blockHandler = "rateLimiterBlockHandler"
35     )
36     public String testRateLimiterControlBehavior() {
37         String format = String.format("【匀速排队-成功】当前线程: %s, 请求正常执
38         行", Thread.currentThread().getName());
39         System.out.println(format);
40         return format;
41     }

```

2.rule

```

1 // 规则 1: 直接拒绝 (CONTROL_BEHAVIOR_DEFAULT, 默认值)
2     FlowRule defaultRule = new FlowRule();
3     defaultRule.setResource(CommonConstant.DEFAULT_RESOURCE);
4     defaultRule.setGrade(RuleConstant.FLOW_GRADE_QPS); // QPS 限流
5     defaultRule.setCount(10); // QPS 阈值 10
6
7     defaultRule.setControlBehavior(RuleConstant.CONTROL_BEHAVIOR_DEFAULT); // 直接拒绝 (可省略, 默认值)
8     existingRules.add(defaultRule);
9
10    // 规则 2: 预热/冷启动 (CONTROL_BEHAVIOR_WARM_UP)
11    FlowRule warmUpRule = new FlowRule();
12    warmUpRule.setResource(CommonConstant.WARM_UP_RESOURCE);
13    warmUpRule.setGrade(RuleConstant.FLOW_GRADE_QPS);
14    warmUpRule.setCount(20); // 最终 QPS 阈值 20
15
16    warmUpRule.setControlBehavior(RuleConstant.CONTROL_BEHAVIOR_WARM_UP); // 预热效果
17    warmUpRule.setWarmUpPeriodSec(5); // 预热时间 5 秒 (阈值从 10 逐步提升至 20)
18    existingRules.add(warmUpRule);
19
20    // 规则 3: 匀速排队 (CONTROL_BEHAVIOR_RATE_LIMITER)
21    FlowRule rateLimiterRule = new FlowRule();
22    rateLimiterRule.setResource(CommonConstant.RATE_LIMITER_RESOURCE);
23    rateLimiterRule.setGrade(RuleConstant.FLOW_GRADE_QPS);

```

```

22         rateLimiterRule.setCount(5); // QPS 阈值 5 (每秒允许 5 个请求通过, 间隔
23 200 毫秒/个)
24
25     rateLimiterRule.setControlBehavior(RuleConstant.CONTROL_BEHAVIOR_RATE_LIMI-
TER); // 匀速排队
26     rateLimiterRule.setMaxQueueingTimeMs(1000); // 最大排队等待时间 1000
毫秒 (1 秒), 超过则拒绝
27     existingRules.add(rateLimiterRule);

```

3.handler

```

1 // 降级方法: 直接拒绝
2     public static String defaultBlockHandler(BlockException e) {
3         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());
4         System.out.println(format);
5         return format;
6     }
7
8 // 降级方法: 预热
9     public static String warmupBlockHandler(BlockException e) {
10        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());
11        System.out.println(format);
12        return format;
13    }
14
15 // 降级方法: 匀速排队
16     public static String rateLimiterBlockHandler(BlockException e) {
17         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());
18         System.out.println(format);
19         return format;
20     }

```

4.client

```

1 @Test
2     public void testStrategy() throws InterruptedException {
3         System.out.println("===== 场景 1: 测试 直接拒绝 效果 (QPS 阈值 10)
=====");
4         testControlBehavior(DEFAULT_URL);
5
6         // 间隔 10 秒, 让 Sentinel 重置统计
7         TimeUnit.SECONDS.sleep(10);
8
9         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) =====");
10        testControlBehavior(WARM_UP_URL);
11
12        // 间隔 10 秒
13        TimeUnit.SECONDS.sleep(10);
14

```

```

15     System.out.println("\n===== 场景 3: 测试 匀速排队 效果 (QPS 阈值
16 5, 最大排队 1 秒) =====");
17     testControlBehavior(RATE_LIMITER_URL);
18 }
19 /**
20 * 模拟高并发请求, 验证流控效果
21 * @param url 测试接口地址
22 */
23 private static void testControlBehavior(String url) {
24     // 创建固定线程池
25     ExecutorService executorService =
26 Executors.newFixedThreadPool(STRATEGY_CONCURRENT_THREADS);
27
28     // 记录开始时间
29     long startTime = System.currentTimeMillis();
30
31     // 提交并发请求任务
32     for (int i = 0; i < STRATEGY_CONCURRENT_THREADS; i++) {
33         executorService.submit(() -> {
34             try (CloseableHttpClient httpClient =
35 HttpClients.createDefault()) {
36                 HttpGet httpGet = new HttpGet(url);
37                 String response =
38 EntityUtils.toString(httpClient.execute(httpGet).getEntity(), "UTF-8");
39                 System.out.println(response);
40             } catch (Exception e) {
41                 e.printStackTrace();
42             }
43         });
44     }
45
46     // 关闭线程池, 等待所有任务执行完成
47     executorService.shutdown();
48     while (!executorService.isTerminated()) {}
49
50     // 记录结束时间, 打印耗时
51     long endTime = System.currentTimeMillis();
52     System.out.println("本次请求总耗时: " + (endTime - startTime) + " 毫
53 秒");
54 }

```

5.constant

```

1 // 资源名 (区分不同流控效果)
2     public static final String DEFAULT_RESOURCE =
3 "defaultControlBehaviorResource";
4     public static final String WARM_UP_RESOURCE =
5 "warmUpControlBehaviorResource";
6     public static final String RATE_LIMITER_RESOURCE =
7 "rateLimiterControlBehaviorResource";

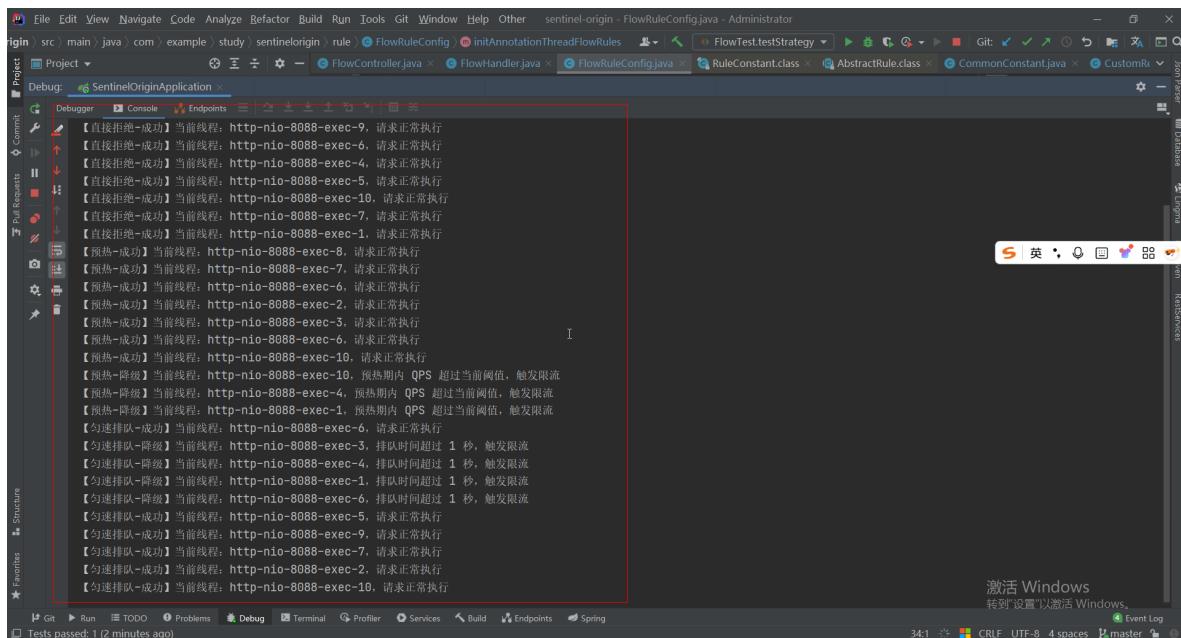
```

6.效果


```

58 【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
59 【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
60 【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
61 【匀速排队-成功】当前线程: http-nio-8088-exec-10, 请求正常执行
62 【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
63 【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
64 【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
65 【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
66 【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
67 【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
68 【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
69 【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
70 【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
71 【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
72 【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
73 【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
74 【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
75 【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
76 【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
77 【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
78 【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
79 【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
80 【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
81 【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
82 【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
83 【匀速排队-成功】当前线程: http-nio-8088-exec-4, 请求正常执行
84 【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
85 【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
86 【匀速排队-成功】当前线程: http-nio-8088-exec-9, 请求正常执行
87 【匀速排队-成功】当前线程: http-nio-8088-exec-8, 请求正常执行
88 【匀速排队-成功】当前线程: http-nio-8088-exec-6, 请求正常执行
89 【匀速排队-成功】当前线程: http-nio-8088-exec-2, 请求正常执行
90 【匀速排队-成功】当前线程: http-nio-8088-exec-1, 请求正常执行
91

```



4. 触发模式

```
1 链路模式尝试了很多方式都失败了
2 1.设置资源路径合并
3 2.controller调service
4 3.不依赖web,本地调
5 4.手动在UI上添加限流
6
7 都不行, 暂时跳过
```

1.controller

```
1 // **** 直接流控 测试接口 ****
2 @GetMapping("/module/direct")
3 @SentinelResource(
4     value = CommonConstant.DIRECT_RESOURCE,
5     blockHandlerClass = FlowHandler.class,
6     blockHandler = "directBlockHandler"
7 )
8 public String testDirectStrategy() {
9     String format = String.format("【直接流控-成功】当前线程: %s, 请求正常执
10 行", Thread.currentThread().getName());
11     System.out.println(format);
12     return format;
13 }
14
15 // **** 关联流控 测试接口 ****
16 // 接口 2.1: 当前资源（订单创建）
17 @GetMapping("/module/associateCurrent")
18 @SentinelResource(
19     value = CommonConstant.ASSOCIATE_CURRENT_RESOURCE,
20     blockHandlerClass = FlowHandler.class,
21     blockHandler = "associateCurrentBlockHandler"
22 )
23 public String testAssociateCurrent() {
24     String format = String.format("【关联流控-当前资源（订单创建）-成功】当前线
25 程: %s, 请求正常执行", Thread.currentThread().getName());
26     System.out.println(format);
27     return format;
28 }
29
30 // 接口 2.2: 关联资源（库存扣减）
31 @GetMapping("/module/associateRef")
32 @SentinelResource(
33     value = CommonConstant.ASSOCIATE_REF_RESOURCE
34 )
35 public String testAssociateRef() throws InterruptedException {
36     // 模拟库存扣减耗时, 便于触发关联限流
37     Thread.sleep(100);
38     String format = String.format("【关联流控-关联资源（库存扣减）-成功】当前线
39 程: %s, 请求正常执行", Thread.currentThread().getName());
40     System.out.println(format);
41     return format;
42 }
43 // **** 链路流控 测试接口 ****
```

```

43     // 接口 3.1: 入口链路资源 (/api/entry)
44     @GetMapping("/module/chainEntry")
45     @SentinelResource(
46         value = CommonConstant.CHAIN_ENTRY_RESOURCE
47     )
48     public String testChainEntry() throws InterruptedException {
49         // 入口链路调用当前资源（用户查询）
50         return chainFlowCoreService.doCoreUserQuery();
51     }
52
53     // 接口 3.2: 当前资源（用户查询）
54     @GetMapping("/module/chainCurrent")
55     public String testChainCurrent() throws InterruptedException {
56         String format = chainFlowCoreService.doCoreUserQuery();
57         return format;
58     }

```

2.rule

```

1     // **** 直接流控 测试接口 ****
2     @GetMapping("/module/direct")
3     @SentinelResource(
4         value = CommonConstant.DIRECT_RESOURCE,
5         blockHandlerClass = FlowHandler.class,
6         blockHandler = "directBlockHandler"
7     )
8     public String testDirectStrategy() {
9         String format = String.format("【直接流控-成功】当前线程: %s, 请求正常执
10        行", Thread.currentThread().getName());
11        System.out.println(format);
12        return format;
13    }
14
15     // **** 关联流控 测试接口 ****
16     // 接口 2.1: 当前资源（订单创建）
17     @GetMapping("/module/associateCurrent")
18     @SentinelResource(
19         value = CommonConstant.ASSOCIATE_CURRENT_RESOURCE,
20         blockHandlerClass = FlowHandler.class,
21         blockHandler = "associateCurrentBlockHandler"
22     )
23     public String testAssociateCurrent() {
24         String format = String.format("【关联流控-当前资源（订单创建）-成功】当前线
25        程: %s, 请求正常执行", Thread.currentThread().getName());
26         System.out.println(format);
27         return format;
28     }
29
30     // 接口 2.2: 关联资源（库存扣减）
31     @GetMapping("/module/associateRef")
32     @SentinelResource(
33         value = CommonConstant.ASSOCIATE_REF_RESOURCE
34     )
35     public String testAssociateRef() throws InterruptedException {

```

```

35     // 模拟库存扣减耗时，便于触发关联限流
36     Thread.sleep(100);
37     String format = String.format("【关联流控-关联资源（库存扣减）-成功】当前线
38 程: %s, 请求正常执行", Thread.currentThread().getName());
39     System.out.println(format);
40     return format;
41 }
42 // ***** 链路流控 测试接口 *****
43 // 接口 3.1: 入口链路资源 (/api/entry)
44 @GetMapping("/module/chainEntry")
45 @SentinelResource(
46     value = CommonConstant.CHAIN_ENTRY_RESOURCE
47 )
48 public String testChainEntry() throws InterruptedException {
49     // 入口链路调用当前资源（用户查询）
50     return chainFlowCoreService.doCoreUserQuery();
51 }
52
53 // 接口 3.2: 当前资源（用户查询）
54 @GetMapping("/module/chainCurrent")
55 public String testChainCurrent() throws InterruptedException {
56     String format = chainFlowCoreService.doCoreUserQuery();
57     return format;
58 }
```

3.client

```

1 // 测试接口地址
2     private static final String DIRECT_URL =
3         "http://localhost:8088/flow/module/direct";
4     private static final String ASSOCIATE_CURRENT_URL =
5         "http://localhost:8088/flow/module/associateCurrent";
6     private static final String ASSOCIATE_REF_URL =
7         "http://localhost:8088/flow/module/associateRef";
8     private static final String CHAIN_ENTRY_URL =
9         "http://localhost:8088/flow/module/chainEntry";
10    private static final String CHAIN_CURRENT_URL =
11        "http://localhost:8088/flow/module/chainCurrent";
12        // 并发线程数（用于触发限流）
13    private static final int Module_CONCURRENT_THREADS = 20;
14
15    @Test
16    public void testModule() throws InterruptedException {
17        System.out.println("===== 场景 1: 测试 直接流控 效果（当前资源 QPS
18 阈值 10）=====");
19        testStrategy(DIRECT_URL);
20
21        // 间隔 10 秒，让 Sentinel 重置统计
22        TimeUnit.SECONDS.sleep(10);
23
24        System.out.println("\n===== 场景 2: 测试 关联流控 效果（关联资源 QPS
25 阈值 5，联动限流当前资源）=====");
26        // 第一步：高并发请求关联资源（库存扣减），使其 QPS 超过阈值 5
27        System.out.println("--- 第一步：高并发请求关联资源（库存扣减）---");
```

```
21     testStrategy(ASSOCIATE_REF_URL);
22     // 第二步：立即请求当前资源（订单创建），验证是否被联动限流
23     System.out.println("--- 第二步：请求当前资源（订单创建），验证关联限流 ---");
24 );
25
26     // 间隔 10 秒
27     TimeUnit.SECONDS.sleep(10);
28
29     System.out.println("\n===== 场景 3：测试 链路流控 效果（入口链路 QPS
30 阈值 8）=====");
31     // 第一步：高并发请求入口链路（/api/entry），触发链路限流
32     System.out.println("--- 第一步：高并发请求入口链路，触发链路限流 ---");
33     testStrategy(CHAIN_ENTRY_URL);
34     // 第二步：直接请求当前资源，验证是否不受限流影响
35     System.out.println("--- 第二步：直接请求当前资源，验证不受链路限流影响 ---");
36 );
37
38 /**
39 * 模拟高并发请求（默认 20 个线程）
40 * @param url 测试接口地址
41 */
42 private static void testStrategy(String url) {
43     testStrategy(url, Module_CONCURRENT_THREADS);
44 }
45
46 /**
47 * 模拟指定线程数的并发请求
48 * @param url 测试接口地址
49 * @param threadNum 并发线程数
50 */
51 private static void testStrategy(String url, int threadNum) {
52     // 创建固定线程池
53     ExecutorService executorService =
54         Executors.newFixedThreadPool(threadNum);
55
56     // 提交并发请求任务
57     for (int i = 0; i < threadNum; i++) {
58         executorService.submit(() -> {
59             for (int j = 0; j < 20; j++) {
60                 try (CloseableHttpClient httpClient =
61                     HttpClients.createDefault()) {
62                     HttpGet httpGet = new HttpGet(url);
63                     String response =
64                         EntityUtils.toString(httpClient.execute(httpGet).getEntity(), "UTF-8");
65                     System.out.println(response);
66                 } catch (Exception e) {
67                     e.printStackTrace();
68                 }
69             }
70         });
71     }
72
73     // 关闭线程池，等待所有任务执行完成
74     executorService.shutdown();
75     while (!executorService.isTerminated()) {}
```

4.handler

```

1 // ****降级方法*****
2 // 直接流控降级方法
3 public static String directBlockHandler(BlockException e) {
4     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());
5     System.out.println(format);
6     return format;
7 }
8
9 // 关联流控-当前资源降级方法
10 public static String associateCurrentBlockHandler(BlockException e) {
11     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",
12     Thread.currentThread().getName());
13     System.out.println(format);
14     return format;
15 }
16
17 // 链路流控-当前资源降级方法
18 public static String chainCurrentBlockHandler(BlockException e) {
19     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);
20     System.out.println(format);
21     return format;
22 }

```

5.constant

```

1 // 资源名定义
2     public static final String DIRECT_RESOURCE = "directStrategyResource"; // 
直接流控-当前资源
3
4     public static final String ASSOCIATE_CURRENT_RESOURCE =
"associateCurrentResource"; // 关联流控-当前资源(订单创建)
5     public static final String ASSOCIATE_REF_RESOURCE =
"associateRefResource"; // 关联流控-关联资源(库存扣减)
6
7     public static final String CHAIN_CURRENT_RESOURCE =
"chainCurrentResource"; // 链路流控-当前资源(用户查询)
8     public static final String CHAIN_ENTRY_RESOURCE = "chainEntryResource";
// 链路流控-入口资源(/api/entry)

```

6.效果

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

```
2 【直接流控-成功】当前线程: http-nio-8088-exec-13, 请求正常执行
3 【直接流控-成功】当前线程: http-nio-8088-exec-7, 请求正常执行
4 【直接流控-成功】当前线程: http-nio-8088-exec-20, 请求正常执行
5 【直接流控-成功】当前线程: http-nio-8088-exec-1, 请求正常执行
6 【直接流控-成功】当前线程: http-nio-8088-exec-10, 请求正常执行
7 【直接流控-成功】当前线程: http-nio-8088-exec-12, 请求正常执行
8 【直接流控-成功】当前线程: http-nio-8088-exec-16, 请求正常执行
9 【直接流控-成功】当前线程: http-nio-8088-exec-14, 请求正常执行
10 【直接流控-成功】当前线程: http-nio-8088-exec-19, 请求正常执行
11 【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
12 【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
13 【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
14 【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
15 【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
16 【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
17 【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
18 【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
19 【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
20 【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
21
22
23 【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
24 【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
25 【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
26 【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
27 【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
28 【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
29 【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
30 【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
31 【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
32 【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
33 【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
34 【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
35 【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
36 【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
37 【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
38 【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
39 【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
40 【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
41 【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
42 【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
43 【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
44 【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
45 【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
46 【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
47 【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
48
```

5.Degrade

1.降级效果

- ```

1 1.rt
2 2.异常比率
3 3.异常数
4
5 其他的不再重复演示和flow的一样

```

| 资源名                  | 熔断策略  | 阀值  | 熔断时长(s) | 操作                                      |
|----------------------|-------|-----|---------|-----------------------------------------|
| ratioDegradeResource | 异常比例  | 0.5 | 5s      | <button>编辑</button> <button>删除</button> |
| rtDegradeResource    | 慢调用比例 | 200 | 5s      | <button>编辑</button> <button>删除</button> |
| countDegradeResource | 异常数   | 3   | 5s      | <button>编辑</button> <button>删除</button> |

## 1.controller

```

1 package com.example.study.sentinelorigin.controller;
2
3 import com.alibaba.csp.sentinel.annotation.SentinelResource;
4 import com.example.study.sentinelorigin.constant.CommonConstant;
5 import com.example.study.sentinelorigin.handle.DegradeHandler;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 import java.util.Random;
10
11 /**
12 * @author lucksoul
13 * @version 1.0
14 * @date 2026/1/10 0:47
15 */
16 @RestController
17 @RequestMapping("/degrade")
18 public class DegradeController {
19
20 private static final Random RANDOM = new Random();
21
22 // ===== 业务方法（模拟慢响应，触发 RT 熔断） =====
23 @SentinelResource(
24 value = CommonConstant.RT_DEGRADE_RESOURCE,
25 blockHandlerClass = DegradeHandler.class,
26 fallbackClass = DegradeHandler.class,
27 blockHandler = "rtDegradeBlockHandler", // 熔断降级方法
28 fallback = "rtDegradeFallback" // 业务异常兜底方法（可选）
29)
30 @RequestMapping("/rt")
31 public String doSlowBusiness() {
32 // 模拟慢响应：睡眠 200 毫秒（超过 RT 阀值 100 毫秒）
33 }
34 }

```

```
33 try {
34 Thread.sleep(1000);
35 } catch (InterruptedException e) {
36 throw new RuntimeException("业务执行中断", e);
37 }
38 String format = String.format("【RT 熔断-成功】线程: %s, 业务执行完成 (慢
39 响应)", Thread.currentThread().getName());
40 System.out.println(format);
41 return format;
42 }
43
44 // ===== 业务方法 (模拟随机异常, 触发异常比例熔断) =====
45 @SentinelResource(
46 value = CommonConstant.RATIO_DEGRADE_RESOURCE,
47 blockHandlerClass = DegradeHandler.class,
48 fallbackClass = DegradeHandler.class,
49 blockHandler = "ratioDegradeBlockHandler",
50 fallback = "ratioDegradeFallback"
51)
52 @RequestMapping("/exceptionRate")
53 public String doRandomExceptionBusiness() {
54 // 模拟 60% 概率抛出异常 (超过异常比例阈值 0.5)
55 if (RANDOM.nextDouble() > 0.4) {
56 throw new RuntimeException("业务执行失败 (随机异常)");
57 }
58 String format = String.format("【异常比例-成功】线程: %s, 业务执行完成",
59 Thread.currentThread().getName());
60 System.out.println(format);
61 return format;
62 }
63
64 // 异常计数器 (模拟累计异常)
65 private int exceptionCounter = 0;
66
67 // ===== 业务方法 (模拟累计异常, 触发异常数熔断) =====
68 @SentinelResource(
69 value = CommonConstant.COUNT_DEGRADE_RESOURCE,
70 blockHandlerClass = DegradeHandler.class,
71 fallbackClass = DegradeHandler.class,
72 blockHandler = "countDegradeBlockHandler",
73 fallback = "countDegradeFallback"
74)
75 @RequestMapping("/exceptionCount")
76 public String doAccumulateExceptionBusiness() {
77 // 模拟每次调用都抛出异常, 快速累计异常数 (超过阈值 3)
78 exceptionCounter++;
79 throw new RuntimeException(String.format("业务执行失败 (累计异常数:
80 %d)", exceptionCounter));
81 }
82
83 }
84 }
```

## 2.rule

```
1 package com.example.study.sentinelorigin.rule;
2
3 import com.alibaba.csp.sentinel.slots.block.RuleConstant;
4 import com.alibaba.csp.sentinel.slots.block.degrade.DegradeRule;
5 import com.alibaba.csp.sentinel.slots.block.degrade.DegradeRuleManager;
6 import com.example.study.sentinelorigin.constant.CommonConstant;
7 import org.springframework.stereotype.Component;
8
9 import javax.annotation.PostConstruct;
10 import java.util.ArrayList;
11 import java.util.List;
12
13 /**
14 * @author lucksoul
15 * @version 1.0
16 * @date 2026/1/10 0:52
17 */
18
19 @Component
20 public class DegradeRuleConfig {
21
22 // ===== 配置 RT 熔断规则 =====
23 @PostConstruct
24 public void initRtDegradeRule() {
25 List<DegradeRule> rules = new ArrayList<>();
26 DegradeRule rule = new DegradeRule();
27
28 // 1. 绑定保护资源
29 rule.setResource(CommonConstant.RT_DEGRADE_RESOURCE);
30 // 2. 指定熔断方式: 基于平均响应时间 (RT)
31 rule.setGrade(RuleConstant.DEGRADE_GRADE_RT);
32 // 3. 配置 RT 阈值: 100 毫秒 (超过该值则计入慢请求)
33 rule.setCount(200);
34 // 4. 配置熔断窗口时间: 5 秒 (窗口内所有请求直接被拒绝)
35 rule.setTimeWindow(5);
36 // 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)
37 rule.setMinRequestAmount(5);
38 // 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)
39 rule.setSlowRatioThreshold(0.5);
40
41 rules.add(rule);
42
43 DegradeRuleManager.loadRules(rules);
44 System.out.println("===== 基于 RT 的熔断规则加载完成 =====");
45
46
47 List<DegradeRule> degradeRules = DegradeRuleManager.getRules();
48 System.out.println("before add size=" + degradeRules.size());
49
50
51 DegradeRule rule2 = new DegradeRule();
52 // 1. 绑定保护资源
53 rule2.setResource(CommonConstant.RATIO_DEGRADE_RESOURCE);
54 // 2. 指定熔断方式: 基于异常比例
55 rule2.setGrade(RuleConstant.DEGRADE_GRADE_EXCEPTION_RATIO);
```

```

56 // 3. 配置异常比例阈值: 0.5 (50%, 超过该比例则触发熔断)
57 rule2.setCount(0.5);
58 // 4. 配置熔断窗口时间: 5 秒
59 rule2.setTimewindow(5);
60 // 5. 配置最小请求数: 5 (1 秒内请求数超过 5 才会触发熔断)
61 rule2.setMinRequestAmount(5);
62 degradeRules.add(rule2);
63
64
65 DegradeRule rule3 = new DegradeRule();
66
67 // 1. 绑定保护资源
68 rule3.setResource(CommonConstant.COUNT_DEGRADE_RESOURCE);
69 // 2. 指定熔断方式: 基于异常数
70 rule3.setGrade(RuleConstant.DEGRADE_GRADE_EXCEPTION_COUNT);
71 // 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)
72 rule3.setCount(3);
73 // 4. 配置熔断窗口时间: 5 秒
74 rule3.setTimewindow(5);
75 // 5. 配置最小请求数: 1 (低 QPS 场景, 降低最小请求数要求)
76 rule3.setMinRequestAmount(1);
77 degradeRules.add(rule3);
78
79 DegradeRuleManager.loadRules(degradeRules);
80 System.out.println("after add size=" +
81 DegradeRuleManager.getRules().size());
82 }
83

```

### 3.handler

```

1 package com.example.study.sentinelorigin.handle;
2
3 import com.alibaba.csp.sentinel.slots.block.BlockException;
4
5 /**
6 * @author LuckSoul
7 * @version 1.0
8 * @date 2026/1/10 0:50
9 */
10 public class DegradeHandler {
11
12 // ===== 熔断降级方法 (熔断触发时调用) =====
13 public static String rtDegradeBlockHandler(BlockException e) {
14 String format = String.format("【RT 熔断-触发】线程: %s, 平均响应时间超过
15 阈值, 进入熔断窗口", Thread.currentThread().getName());
16 System.out.println(format);
17 return format;
18 }
19
20 // ===== 业务异常兜底方法 (可选, 非熔断触发) =====
21 public static String rtDegradeFallback(Throwable e) {
22 String format = String.format("【RT 熔断-兜底】线程: %s, 业务执行异常:
23 %s", Thread.currentThread().getName(), e.getMessage());
24 }
25
26 }

```

```

22 System.out.println(format);
23 return format;
24 }
25
26
27
28 // ===== 熔断降级方法 =====
29 public static String ratioDegradeBlockHandler(BlockException e) {
30 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());
31 System.out.println(format);
32 return format;
33 }
34
35 // ===== 业务异常兜底方法 =====
36 public static String ratioDegradeFallback(Throwable e) {
37 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());
38 System.out.println(format);
39 return format;
40 }
41
42
43 // ===== 熔断降级方法 =====
44 public static String countDegradeBlockHandler(BlockException e) {
45 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());
46 System.out.println(format);
47 return format;
48 }
49
50 // ===== 业务异常兜底方法 =====
51 public static String countDegradeFallback(Throwable e) {
52 String format = String.format("【异常数-兜底】线程: %s, 业务执行异常:
%s", Thread.currentThread().getName(), e.getMessage());
53 System.out.println(format);
54 return format;
55 }
56 }
57

```

#### 4.client

```

1 package com.example.study.sentinelorigin.client;
2
3 import org.apache.http.client.methods.CloseableHttpResponse;
4 import org.apache.http.client.methods.HttpGet;
5 import org.apache.http.impl.client.CloseableHttpClient;
6 import org.apache.http.impl.client.HttpClients;
7 import org.apache.http.util.EntityUtils;
8 import org.junit.Test;
9
10 import java.nio.charset.StandardCharsets;
11 import java.util.concurrent.ExecutorService;
12 import java.util.concurrent.Executors;

```

```
13 import java.util.concurrent.TimeUnit;
14
15 /**
16 * @author lucksoul
17 * @version 1.0
18 * @date 2026/1/10 1:21
19 */
20 public class DegradeTest {
21
22 // 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）
23 private static final String RT_TARGET_URL =
24 "http://localhost:8088/degrade/rt";
25
26 @Test
27 public void testRtDegradeResource() {
28 // 创建固定线程池
29 ExecutorService executorService =
30 Executors.newFixedThreadPool(20);
31
32 // 记录开始时间
33 long startTime = System.currentTimeMillis();
34 // 提交并发请求任务
35 for (int i = 0; i < 20; i++) {
36 executorService.submit(() -> {
37 for (int j = 0; j < 20; j++) {
38 // 1. 创建 CloseableHttpClient 实例（推荐使用
39 HttpClients.createDefault())
40 try (CloseableHttpClient httpClient =
41 HttpClients.createDefault()) {
42 // 2. 构建 HTTP GET 请求（若接口是 POST，可使用
43 HttpPost)
44 HttpGet httpGet = new HttpGet(RT_TARGET_URL);
45 // 可选：设置请求头，模拟浏览器请求
46 httpGet.setHeader("User-Agent", "Mozilla/5.0
47 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36");
48 httpGet.setHeader("Accept", "application/json,
49 text/plain, */*");
50
51 // 3. 发送请求，获取响应
52 try (CloseableHttpResponse response =
53 httpClient.execute(httpGet)) {
54 // 4. 解析响应结果
55 int statusCode =
56 response.getStatusLine().getStatusCode();
57 String responseBody =
58 EntityUtils.toString(response.getEntity(),
59 StandardCharsets.UTF_8);
60
61 // 5. 打印结果
62 System.out.println("==> 接口响应结果 ==>");
63 System.out.println("响应状态码: " + statusCode);
64 System.out.println("响应内容: " + responseBody);
65 }
66 } catch (Exception e) {
67 e.printStackTrace();
68 System.out.println("请求失败: " + e.getMessage());
69 }
70 }
71 }
72 }
73 }
74 }
75 }
```

```
61 });
62 }
63
64
65 // 关闭线程池，等待所有任务执行完成
66 executorService.shutdown();
67 while (!executorService.isTerminated()) {}
68
69 // 记录结束时间，打印耗时
70 long endTime = System.currentTimeMillis();
71 System.out.println("本次请求总耗时: " + (endTime - startTime) + " 毫
72 秒");
73 }
74
75 // 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）
76 private static final String EXCEPTION_RATE_TARGET_URL =
77 "http://localhost:8088/degrade/exceptionRate";
78 // 并发线程数（可调整，建议 10-20 个，确保触发熔断）
79 private static final int EXCEPTION_RATE_THREAD_COUNT = 15;
80 // 每个线程发送的请求数（可调整，确保 QPS 超过阈值）
81 private static final int EXCEPTION_RATE_REQUEST_PER_THREAD = 10;
82
83 @Test
84 public void testExceptionRate() {
85 // 1. 创建固定大小线程池
86 ExecutorService executorService =
87 Executors.newFixedThreadPool(EXCEPTION_RATE_THREAD_COUNT);
88 System.out.println("==> 高并发请求开始，线程数: " +
89 EXCEPTION_RATE_THREAD_COUNT + "，每个线程请求数: " +
90 EXCEPTION_RATE_REQUEST_PER_THREAD + " ==>");
91
92 // 2. 提交线程任务
93 for (int i = 0; i < EXCEPTION_RATE_THREAD_COUNT; i++) {
94 int threadNum = i + 1;
95 executorService.submit(() -> {
96 // 每个线程创建一个 HttpClient 实例（线程安全，可复用）
97 try (CloseableHttpClient httpClient =
98 HttpClients.createDefault()) {
99 for (int j = 0; j < EXCEPTION_RATE_REQUEST_PER_THREAD;
j++) {
100 int requestNum = j + 1;
101 try {
102 // 构建 GET 请求
103 HttpGet httpGet = new
104 HttpGet(EXCEPTION_RATE_TARGET_URL);
105 httpGet.setHeader("User-Agent",
106 "HighConcurrentHttpClient/1.0");
107
108 // 发送请求并获取响应
109 try (CloseableHttpResponse response =
110 httpClient.execute(httpGet)) {
111 // 解析响应
112 String responseBody =
113 EntityUtils.toString(response.getEntity(),
114 StandardCharsets.UTF_8);
115 int statusCode =
116 response.getStatusLine().getStatusCode();
117 }
118 }
119 }
120 }
121 }
122 }
123 }
```

```
107 // 打印请求结果（可选，可注释以减少输出干扰）
108 System.out.printf("线程 %d - 请求 %d: 状态码
109 %d, 响应内容: %s%n",
110 // 轻微休眠，避免请求过于密集导致接口宕机（可选,
111 // 根据场景调整）
112 TimeUnit.MILLISECONDS.sleep(50);
113 }
114 } catch (Exception e) {
115 System.out.printf("线程 %d - 请求 %d: 失败，原因:
116 %s%n",
117 threadNum, requestNum,
118 e.getMessage());
119 }
120 } catch (Exception e) {
121 System.out.printf("线程 %d: 初始化 HttpClient 失败，原因:
122 %s%n", threadNum, e.getMessage());
123 }
124
125 // 3. 关闭线程池，等待所有任务完成
126 executorService.shutdown();
127 try {
128 boolean allTaskCompleted = executorService.awaitTermination(5,
129 TimeUnit.MINUTES);
130 if (allTaskCompleted) {
131 System.out.println("==> 所有高并发请求执行完成 ==>");
132 } else {
133 System.out.println("==> 部分请求超时未完成 ==>");
134 }
135 } catch (InterruptedException e) {
136 e.printStackTrace();
137 executorService.shutdownNow();
138 }
139
140
141 // 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）
142 private static final String EXCEPTION_COUNT_TARGET_URL =
143 "http://localhost:8088/degrade/exceptionCount";
144 // 并发线程数（可调整，建议 10-20 个，确保触发熔断）
145 private static final int EXCEPTION_COUNT_THREAD_COUNT = 15;
146 // 每个线程发送的请求数（可调整，确保 QPS 超过阈值）
147 private static final int EXCEPTION_COUNT_REQUEST_PER_THREAD = 10;
148
149 @Test
150 public void testExceptionCount() {
151 // 1. 创建固定大小线程池
152 ExecutorService executorService =
153 Executors.newFixedThreadPool(EXCEPTION_COUNT_THREAD_COUNT);
154 System.out.println("==> 高并发请求开始，线程数: " +
155 EXCEPTION_COUNT_THREAD_COUNT + "，每个线程请求数: " +
156 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 {

```

```

198 System.out.println("==> 部分请求超时未完成 ==<");
199 }
200 } catch (InterruptedException e) {
201 e.printStackTrace();
202 executorService.shutdownNow();
203 }
204 }
205 }
206

```

## 5.constant

```

1 /**
2 * 熔断
3 */
4 // 资源名
5 public static final String RT_DEGRADE_RESOURCE = "rtDegradeResource";
6
7 public static final String RATIO_DEGRADE_RESOURCE =
8 "ratioDegradeResource";
9
10 public static final String COUNT_DEGRADE_RESOURCE =
11 "countDegradeResource";

```

## 6.效果

```

1 1.rt
2 【异常比例-成功】线程: http-nio-8088-exec-5, 业务执行完成
3 【异常比例-成功】线程: http-nio-8088-exec-4, 业务执行完成
4 【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)
5 【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)
6 【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)
7 【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
8 【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
9 【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
10 【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
11 【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
12 【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
13 【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
14 【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
15 【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
16
17 2. 异常比率
18 【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)
19 【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
20 【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
21 【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
22 【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
23 【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
24 【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
25 【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

```

```

26 【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
27 【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
28 【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
29 【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
30 【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
31
32 3. 异常数
33 【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)
34 【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)
35 【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)
36 【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)
37 【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
38 【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
39 【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
40 【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
41 【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
42 【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
43 【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)

- 1 黑白名单限制
- 2 1. 自定义 RequestOriginParser
- 3 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

| 资源名                   | 流控应用                               | 授权类型 | 操作                                      |
|-----------------------|------------------------------------|------|-----------------------------------------|
| blackListCoreResource | crawler/malicious-ip-192.168.1.100 | 黑名单  | <button>编辑</button> <button>删除</button> |
| whiteListCoreResource | internal-app/trusted-client-001    | 白名单  | <button>编辑</button> <button>删除</button> |

### 1. 白名单

- 1 直接看项目提交记录查看文件变动
- 2

### 1. 效果

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

## 2.黑名单

- 1 | 直接看项目提交记录查看文件变动

### 1.效果

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

## 7.hotparam(热点参数)

- 1 | 1. 单个参数
- 2 | 2. 额外参数
- 3 | 3. 多个参数分别限制
- 4 |
- 5 | 直接看项目提交记录查看文件变动

The screenshot shows the Sentinel Control Panel interface. On the left, there's a sidebar with navigation links: 首页, sentinel-origin (1/1), 实时监控, 簇点链路, 流控规则, 熔断规则, 热点规则 (highlighted in grey), 系统规则, 授权规则, 集群流控, and 机器列表. The main content area has a title 'sentinel-origin'. Below it, there's a search bar with '192.168.11.1:8720' and a '刷新' button. A table lists '热点参数限流规则' (Hot Parameter Flow Control Rules) with three entries:

| 资源名                   | 参数索引 | 流控模式 | 阈值 | 是否集群 | 例外项数目 | 操作        |
|-----------------------|------|------|----|------|-------|-----------|
| hotParamBasicResource | 0    | QPS  | 5  | 否    | 1     | [编辑] [删除] |
| hotParamMultiResource | 0    | QPS  | 3  | 否    | 0     | [编辑] [删除] |
| hotParamMultiResource | 1    | QPS  | 5  | 否    | 0     | [编辑] [删除] |

At the bottom right of the table, it says '共 3 条记录, 每页 10 条记录'.

### 1.单个

- 1 | 【成功】查询商品, `productId: 1001`, 请求正常处理
- 2 | 【成功】查询商品, `productId: 1001`, 请求正常处理
- 3 | 【成功】查询商品, `productId: 1001`, 请求正常处理
- 4 | 【成功】查询商品, `productId: 1001`, 请求正常处理
- 5 | 【成功】查询商品, `productId: 1001`, 请求正常处理
- 6 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`
- 7 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`
- 8 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`
- 9 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`
- 10 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`
- 11 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`
- 12 | 【限流】商品查询触发热点限流, `productId: 1001`, 原因: `ParamFlowException`

### 2.额外



```
58 【成功】查询商品, productId: 999, 请求正常处理
59 【成功】查询商品, productId: 999, 请求正常处理
60 【成功】查询商品, productId: 999, 请求正常处理
61 【成功】查询商品, productId: 999, 请求正常处理
62 【成功】查询商品, productId: 999, 请求正常处理
63 【成功】查询商品, productId: 999, 请求正常处理
64 【成功】查询商品, productId: 999, 请求正常处理
65 【成功】查询商品, productId: 999, 请求正常处理
66 【成功】查询商品, productId: 999, 请求正常处理
67 【成功】查询商品, productId: 999, 请求正常处理
68 【成功】查询商品, productId: 999, 请求正常处理
69 【成功】查询商品, productId: 999, 请求正常处理
70 【成功】查询商品, productId: 999, 请求正常处理
71 【成功】查询商品, productId: 999, 请求正常处理
72 【成功】查询商品, productId: 999, 请求正常处理
73 【成功】查询商品, productId: 999, 请求正常处理
74 【成功】查询商品, productId: 999, 请求正常处理
75 【成功】查询商品, productId: 999, 请求正常处理
76 【成功】查询商品, productId: 999, 请求正常处理
77 【成功】查询商品, productId: 999, 请求正常处理
78 【成功】查询商品, productId: 999, 请求正常处理
79 【成功】查询商品, productId: 999, 请求正常处理
80 【成功】查询商品, productId: 999, 请求正常处理
81 【成功】查询商品, productId: 999, 请求正常处理
82 【成功】查询商品, productId: 999, 请求正常处理
83 【成功】查询商品, productId: 999, 请求正常处理
84 【成功】查询商品, productId: 999, 请求正常处理
85 【成功】查询商品, productId: 999, 请求正常处理
86 【成功】查询商品, productId: 999, 请求正常处理
87 【成功】查询商品, productId: 999, 请求正常处理
88 【成功】查询商品, productId: 999, 请求正常处理
89 【成功】查询商品, productId: 999, 请求正常处理
90 【成功】查询商品, productId: 999, 请求正常处理
91 【成功】查询商品, productId: 999, 请求正常处理
92
```

### 3.多个参数

```
1 2026-01-10 19:13:23.727 INFO 2628 --- [nio-8088-exec-9] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2 2026-01-10 19:13:23.727 INFO 2628 --- [nio-8088-exec-9] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
3 2026-01-10 19:13:23.728 INFO 2628 --- [nio-8088-exec-9]
o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
4 【成功】查询订单, orderId: 111, userId: luck
5 【成功】查询订单, orderId: 111, userId: luck
6 【成功】查询订单, orderId: 111, userId: luck
7 【限流】订单查询触发热点限流, orderId: 111, userId: luck
8 【限流】订单查询触发热点限流, orderId: 111, userId: luck
9 【限流】订单查询触发热点限流, orderId: 111, userId: luck
10 【限流】订单查询触发热点限流, orderId: 111, userId: luck
11 【限流】订单查询触发热点限流, orderId: 111, userId: luck
12 【限流】订单查询触发热点限流, orderId: 111, userId: luck
13 【限流】订单查询触发热点限流, orderId: 111, userId: luck
14 【限流】订单查询触发热点限流, orderId: 111, userId: luck
```



```
73 【限流】订单查询触发热点限流, orderId: 111, userId: luck
74 【限流】订单查询触发热点限流, orderId: 111, userId: luck
75 【限流】订单查询触发热点限流, orderId: 111, userId: luck
76 【限流】订单查询触发热点限流, orderId: 111, userId: luck
77 【限流】订单查询触发热点限流, orderId: 111, userId: luck
78 【限流】订单查询触发热点限流, orderId: 111, userId: luck
79 【限流】订单查询触发热点限流, orderId: 111, userId: luck
80 【限流】订单查询触发热点限流, orderId: 111, userId: luck
81 【限流】订单查询触发热点限流, orderId: 111, userId: luck
82 【限流】订单查询触发热点限流, orderId: 111, userId: luck
83 【限流】订单查询触发热点限流, orderId: 111, userId: luck
84 【限流】订单查询触发热点限流, orderId: 111, userId: luck
85 【限流】订单查询触发热点限流, orderId: 111, userId: luck
86 【限流】订单查询触发热点限流, orderId: 111, userId: luck
87 【限流】订单查询触发热点限流, orderId: 111, userId: luck
88 【限流】订单查询触发热点限流, orderId: 111, userId: luck
89 【限流】订单查询触发热点限流, orderId: 111, userId: luck
90 【限流】订单查询触发热点限流, orderId: 111, userId: luck
91 【成功】查询订单, orderId: 111, userId: luck
92 【成功】查询订单, orderId: 111, userId: luck
93 【成功】查询订单, orderId: 111, userId: luck
94
```