

1. 概述

1.1. 分布式系统面临的问题

1.2. 是什么

1.3. 能干啥

1.4. 官网资料

1.5. Hystrix官宣, 停更进维

2. Hystrix重要概念

2.1. 服务降级

2.2. 服务熔断

2.3. 服务限流

3. Hystrix案例

3.1. 构建

3.1.1. 新建支付提供者

3.1.2. POM

3.1.3. YML

3.1.4. 主启动

3.1.5. 业务类

3.1.6. 正常场景测试

3.2. 高并发测试

3.2.1. Jmeter压测测试

3.2.1.1. 安装/下载

3.2.1.2. 测试过程

3.2.1.2.1. Jmeter配置

3.2.1.2.2. Jmeter测试

3.2.1.3. 测试结果

3.2.1.4. 原因

3.2.2. Jmeter压测结论

3.2.3. 新建订单服务消费者order80加入其中

3.2.3.1. 新建模块cloud-consumer-feign-hystrix-order80

3.2.3.2. POM

3.2.3.3. YML

3.2.3.4. 主启动

3.2.3.5. 业务类

3.2.3.6. 正常测试

3.2.3.7. 高并发测试

3.2.3.7.1. 20w个线程压8001的timeout接口

3.2.3.7.2. 消费者80微服务再去访问OK服务8001地址

3.2.3.7.3. 测试结果

3.3. 故障和导致现象

3.4. 上述结论

3.5. 如何解决?解决的要求

3.5.1. 要求

3.5.2. 解决

3.6. 服务降级

3.6.1. 降级配置

3.6.2. 8001先从自身找问题

3.6.3. 8001 fallback

3.6.3.1. 业务类启用

3.6.3.2. 主启动类激活

3.6.3.3. 测试

3.6.4. 80 fallback

3.6.4.1. 说明

3.6.4.2. 题外话

3.6.4.3. POM

3.6.4.4. YML

3.6.4.5. 主启动

3.6.4.6. 业务类

3.6.4.7. 测试

3.6.5. 目前问题

3.6.6. 解决方法

3.6.6.1. 每个方法配置一个兜底方法, 代码膨胀

3.6.6.1.1. @DefaultProperties(defaultFallback="")

3.6.6.1.2. controller配置

3.6.6.1.3. 测试

3.6.6.2. 和业务逻辑混在一起, 代码逻辑混乱

3.6.6.2.1. 说明

3.6.6.2.2. 未来我们要面对的异常

3.6.6.2.3. 再看看我们的业务类OrderHyrixController

3.6.6.2.4. 修改cloud-consumer-feign-hystrix-order80

3.6.6.2.5. 统一异常处理

3.6.6.2.6. PaymentFallbackService类实现PaymentFeginService接口

3.6.6.2.7. YML

3.6.6.2.8. PaymentFeignClientService接口

3.6.6.2.9. 测试

3.7. 服务熔断

3.7.1. 断路器

3.7.2. 熔断是什么

3.7.3. 实操

3.7.4.1. PaymentService

3.7.4.2. PaymentController

3.7.4.3. 测试

3.7.4. 原理/小总结

3.7.4.1. 大神结论

3.7.4.2. 熔断类型

3.7.4.3. 官网断路器流程图

3.7.4.3.1. 官网步骤

3.7.4.3.2. 断路器在什么情况下开始起作用

3.7.4.3.3. 断路器开启或者关闭的条件

3.7.4.3.4. 断路器打开之后

3.7.4.3.5. 所有配置

3.8. 服务限流

4. Hystrix工作流程

5. 服务监控HystrixDashboard

5.1. 概述

5.2. 仪表盘9001

5.2.1. POM

5.2.2. YML

5.2.3. HystrixDashboardMain9001 + 新注解@EnableHystrixDashboard

5.2.4. 所有Provider微服务提供类(8001/8002/8003)都需要监控依赖部署

5.2.5. 启动cloud-consumer-hystrix-dashboard9001该微服务后续将监控微服务8001

5.3. 断路器演示(服务监控hystrixDashboard)

5.3.1. 修改cloud-provider-hystrix-payment8001

5.3.2. 监控测试

5.3.2.1. 启动一个eureka或者3个eureka集群均可

5.3.2.2. 观察监控窗口

5.3.2.2.1. 9001监控8001

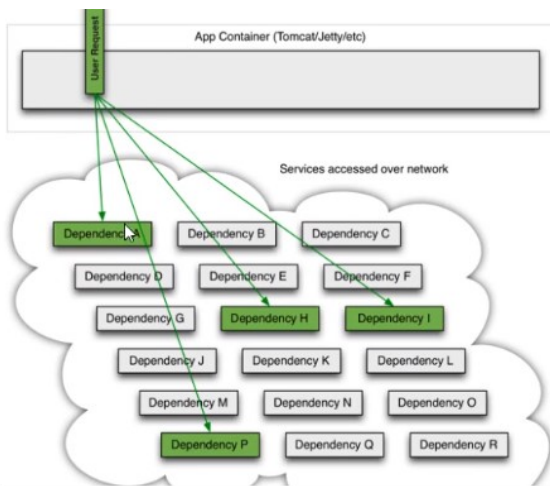
5.3.2.2.2. 测试地址

5.3.2.2.3. 如何看监控窗口

1. 概述

1.1. 分布式系统面临的问题

复杂分布式体系结构中的应用程序有数10个依赖关系,每个依赖关系在某些时候将不可避免地失败



左图中的请求需要调用A, P, H, I四个服务，如果一切顺利则没有什么问题，关键是如果I服务超时会出现什么情况呢？



服务雪崩

多个微服务之间调用的时候，假设微服务A调用微服务B和微服务C，微服务B和微服务C又调用其它的微服务，这就是所谓的“扇出”。如果扇出的链路上某个微服务的调用响应时间过长或者不可用，对微服务A的调用就会占用越来越多的系统资源，进而引起系统崩溃，所谓的“雪崩效应”。

对于高流量的应用来说，单一的后端依赖可能会导致所有服务器上的所有资源都在几秒钟内饱和。比失败更糟糕的是，这些应用程序还可能导致服务之间的延迟增加，备份队列，线程和其他系统资源紧张，导致整个系统发生更多的级联故障。这些都表示需要对故障和延迟进行隔离和管理，以便单个依赖关系的失败，不能取消整个应用程序或系统。

所以通常当你发现一个模块下的某个实例失败后，这时候这个模块依然还会接收流量，然后这个有问题的模块还调用了其他的模块，这样就会发生级联故障，或者叫雪崩。

1.2. 是什么

Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

“断路器”本身是种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

1.3. 能干啥

服务降级

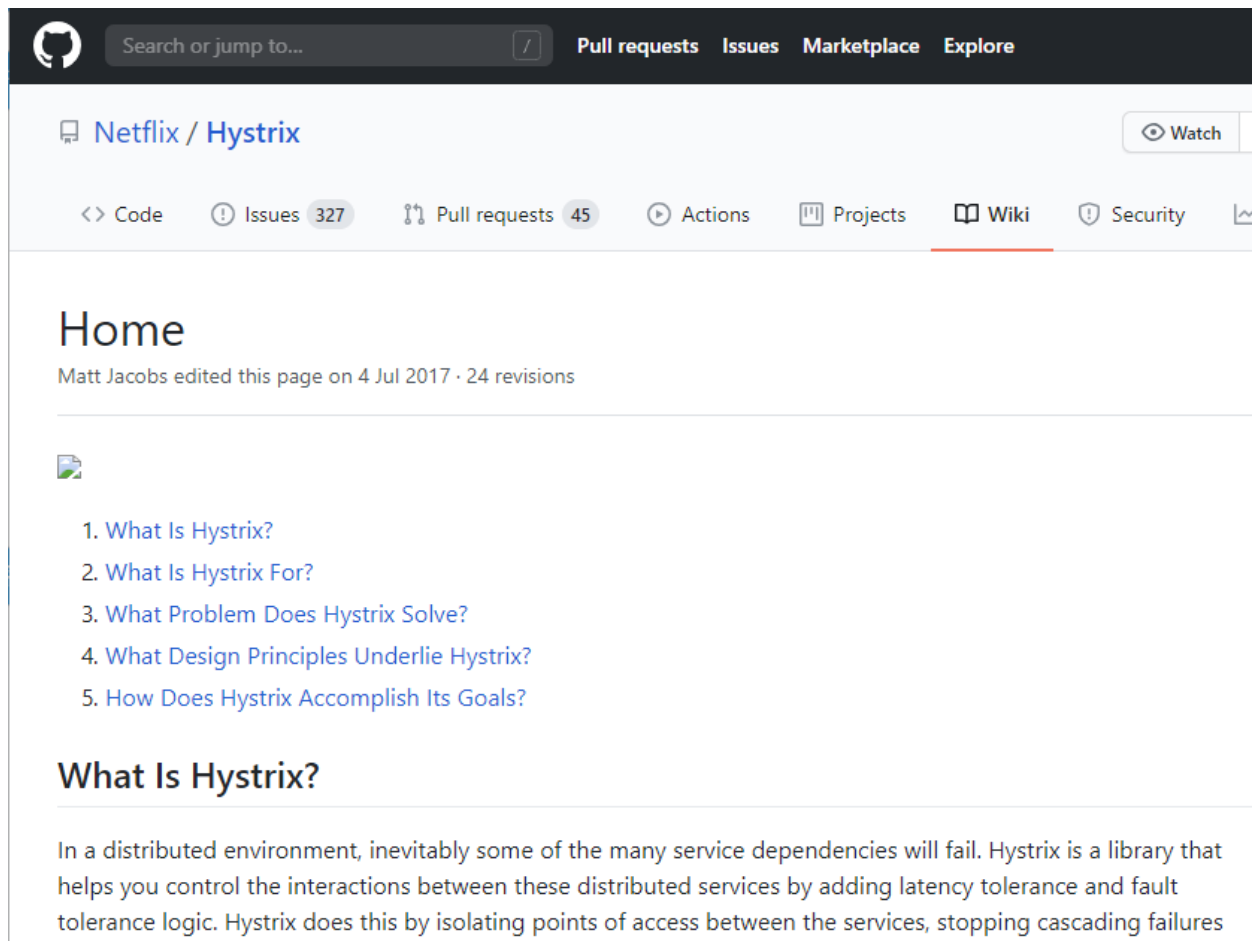
服务熔断

接近实时的监控

...


1.4. 官网资料

<https://github.com/Netflix/hystrix/wiki>



The screenshot shows the GitHub interface for the Netflix Hystrix repository. At the top, there's a dark header with the GitHub logo, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name 'Netflix / Hystrix' is displayed with a 'Watch' button. A secondary navigation bar includes links for Code, Issues (327), Pull requests (45), Actions, Projects, Wiki (which is highlighted with a red underline), and Security. The main content area is titled 'Home' and includes a note that 'Matt Jacobs edited this page on 4 Jul 2017 · 24 revisions'. A small image icon is present, followed by a list of five links: '1. What Is Hystrix?', '2. What Is Hystrix For?', '3. What Problem Does Hystrix Solve?', '4. What Design Principles Underlie Hystrix?', and '5. How Does Hystrix Accomplish Its Goals?'. Below the list, the section 'What Is Hystrix?' is introduced with a paragraph explaining its purpose in a distributed environment.

Home
Matt Jacobs edited this page on 4 Jul 2017 · 24 revisions



1. [What Is Hystrix?](#)
2. [What Is Hystrix For?](#)
3. [What Problem Does Hystrix Solve?](#)
4. [What Design Principles Underlie Hystrix?](#)
5. [How Does Hystrix Accomplish Its Goals?](#)

What Is Hystrix?

In a distributed environment, inevitably some of the many service dependencies will fail. Hystrix is a library that helps you control the interactions between these distributed services by adding latency tolerance and fault tolerance logic. Hystrix does this by isolating points of access between the services, stopping cascading failures

1.5. Hystrix官宣, 停更进维

<https://github.com/Netflix/hystrix>

Hystrix: Latency and Fault Tolerance for Distributed Systems

oss lifecycle maintenance build passing maven central 1.5.18 License Apache 2

Hystrix Status

Hystrix is no longer in active development, and is currently in maintenance mode.

Hystrix (at version 1.5.18) is stable enough to meet the needs of Netflix for our existing applications. Meanwhile, our focus has shifted towards more adaptive implementations that react to an application's real time performance rather than pre-configured settings (for example, through [adaptive concurrency limits](#)). For the cases where something like Hystrix makes sense, we intend to continue using Hystrix for existing applications, and to leverage open and active projects like [resilience4j](#) for new internal projects. We are beginning to recommend others do the same.

Netflix Hystrix is now officially in maintenance mode, with the following expectations to the greater community: Netflix will no longer actively review issues, merge pull-requests, and release new versions of Hystrix. We have made a final release of Hystrix (1.5.18) per [issue 1891](#) so that the latest version in Maven Central is aligned with the last known stable version used internally at Netflix (1.5.11). If members of the community are interested in taking ownership of Hystrix and moving it back into active mode, please reach out to hystrixoss@googlegroups.com.

Hystrix has served Netflix and the community well over the years, and the transition to maintenance mode is in no way an indication that the concepts and ideas from Hystrix are no longer valuable. On the contrary, Hystrix has inspired many great ideas and projects. We thank everyone at Netflix, and in the greater community, for all the contributions made to Hystrix over the years.

被动修复bugs

不再接受合并请求

不再发布新版本

2. Hystrix重要概念

2.1. 服务降级

服务器忙,请稍后再试,不让客户端等待并立刻返回一个友好提示,fallback

哪些情况会发出降级

- 程序运行异常
- 超时
- 服务熔断触发服务降级
- 线程池/信号量也会导致服务降级

2.2. 服务熔断

类比保险丝达到最大服务访问后,直接拒绝访问,拉闸限电,然后调用服务降级的方法并返回友好提示

就是保险丝

服务的降级->进而熔断->恢复调用链路

2.3. 服务限流

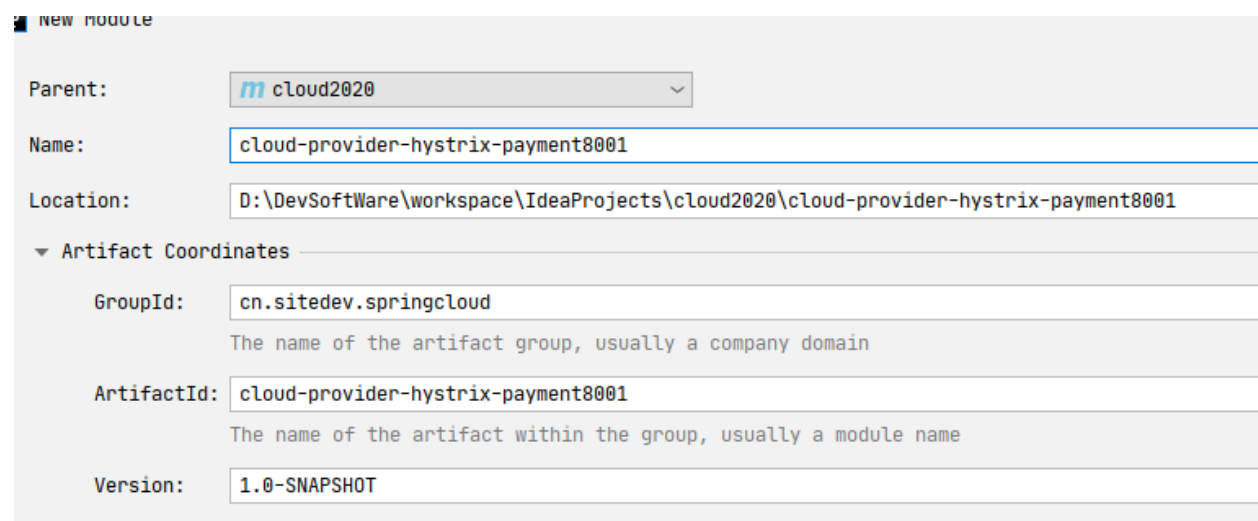
秒杀高并发等操作,严禁一窝蜂的过来拥挤,大家排队,一秒钟N个,有序进行

3. Hystrix案例

3.1. 构建

3.1.1. 新建支付提供者

新建模块cloud-provider-hystrix-payment8001



NEW MODULE

Parent:

Name:

Location:

▼ Artifact Coordinates

GroupId:
The name of the artifact group, usually a company domain

ArtifactId:
The name of the artifact within the group, usually a module name

Version:

3.1.2. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
```



```
12 <artifactId>cloud-provider-hystrix-payment8001</artifactId>
13 <dependencies>
14     <!--hystrix-->
15     <dependency>
16         <groupId>org.springframework.cloud</groupId>
17         <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
18     </dependency>
19     <!--eureka client-->
20     <dependency>
21         <groupId>org.springframework.cloud</groupId>
22         <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
23     </dependency>
24     <dependency>
25         <groupId>cn.sitedev.springcloud</groupId>
26         <artifactId>cloud-api-commons</artifactId>
27         <version>1.0-SNAPSHOT</version>
28     </dependency>
29     <dependency>
30         <groupId>org.springframework.boot</groupId>
31         <artifactId>spring-boot-starter-web</artifactId>
32     </dependency>
33     <!--监控-->
34     <dependency>
35         <groupId>org.springframework.boot</groupId>
36         <artifactId>spring-boot-starter-actuator</artifactId>
37     </dependency>
38     <!--热部署-->
39     <dependency>
40         <groupId>org.springframework.boot</groupId>
41         <artifactId>spring-boot-devtools</artifactId>
42         <scope>runtime</scope>
43         <optional>true</optional>
44     </dependency>
45     <dependency>
46         <groupId>org.projectlombok</groupId>
47         <artifactId>lombok</artifactId>
48         <optional>true</optional>
49     </dependency>
50     <dependency>
51         <groupId>org.springframework.boot</groupId>
52         <artifactId>spring-boot-starter-test</artifactId>
53         <scope>test</scope>
54     </dependency>
```

```
55     </dependencies>
56
57 </project>
```

3.1.3. YML

```
1 server:
2   port: 8001
3 spring:
4   application:
5     name: cloud-provider-hystrix-payment
6 eureka:
7   client:
8     register-with-eureka: true
9     fetch-registry: true
10    service-url:
11      defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
```

3.1.4. 主启动

```
1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6
7 @SpringBootApplication
8 @EnableEurekaClient
9 public class PaymentHystrixMain8001 {
10     public static void main(String[] args) {
11         SpringApplication.run(PaymentHystrixMain8001.class, args);
12     }
13 }
```

3.1.5. 业务类

service:

```

1 package cn.sitedev.springcloud.service;
2
3 import org.springframework.stereotype.Service;
4
5 import java.util.concurrent.TimeUnit;
6
7 @Service
8 public class PaymentService {
9     /**
10      * 正常访问
11      *
12      * @param id
13      * @return
14      */
15     public String paymentInfo_OK(Integer id) {
16         return "线程池:" + Thread.currentThread().getName() + " paymentInfo_OK,id:"
17     }
18
19     /**
20      * 超时访问
21      *
22      * @param id
23      * @return
24      */
25     public String paymentInfo_TimeOut(Integer id) {
26         int timeNumber = 3;
27         try {
28             // 暂停3秒钟
29             TimeUnit.SECONDS.sleep(timeNumber);
30         } catch (InterruptedException e) {
31             e.printStackTrace();
32         }
33         return "线程池:" + Thread.currentThread().getName() + " paymentInfo_TimeOut,
34     }
35 }

```

controller:

```

1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.PaymentService;
4 import lombok.extern.slf4j.Slf4j;

```

```
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PathVariable;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import javax.annotation.Resource;
11
12 @RestController
13 @Slf4j
14 public class PaymentController {
15     @Resource
16     private PaymentService paymentService;
17
18     @Value("${server.port}")
19     private String servicePort;
20
21     /**
22      * 正常访问
23      *
24      * @param id
25      * @return
26      */
27     @GetMapping("/payment/hystrix/ok/{id}")
28     public String paymentInfo_OK(@PathVariable("id") Integer id) {
29         String result = paymentService.paymentInfo_OK(id);
30         log.info("*****result:" + result);
31         return result;
32     }
33
34     /**
35      * 超时访问
36      *
37      * @param id
38      * @return
39      */
40     @GetMapping("/payment/hystrix/timeout/{id}")
41     public String paymentInfo_TimeOut(@PathVariable("id") Integer id) {
42         String result = paymentService.paymentInfo_TimeOut(id);
43         log.info("*****result:" + result);
44         return result;
45     }
46 }
47 }
```

3.1.6. 正常场景测试

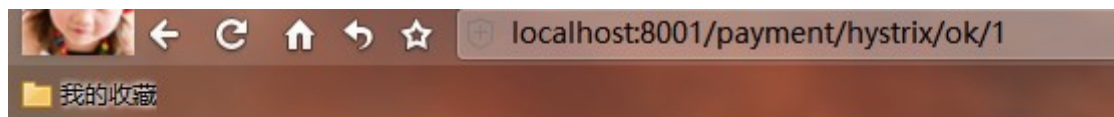
启动eureka7001

Services		Console	Endpoints
Spring Boot		2020-06-27 19:05:44.778 INFO 11144 --- [Thread-64] c	
Running		2020-06-27 19:05:44.790 INFO 11144 --- [Thread-64] e	
EurekaMain7001 [devtools] :7001/		2020-06-27 19:05:44.831 INFO 11144 --- [restartedMain] o	
Not Started		2020-06-27 19:05:44.833 INFO 11144 --- [restartedMain] .!	
EurekaMain7002 [devtools]		2020-06-27 19:05:46.779 INFO 11144 --- [a-EvictionTimer] c	
PaymentMain8002 [devtools]		2020-06-27 19:05:47.539 INFO 11144 --- [restartedMain] o	
PaymentMain8004 [devtools]		2020-06-27 19:05:47.542 INFO 11144 --- [restartedMain] c	
OrderZkMain80 [devtools]		2020-06-27 19:05:47.687 INFO 11144 --- [(1)-192.168.5.1] o	
OrderMain80 [devtools]		2020-06-27 19:05:47.687 INFO 11144 --- [(1)-192.168.5.1] o	

启动cloud-provider-hystrix-payment8001

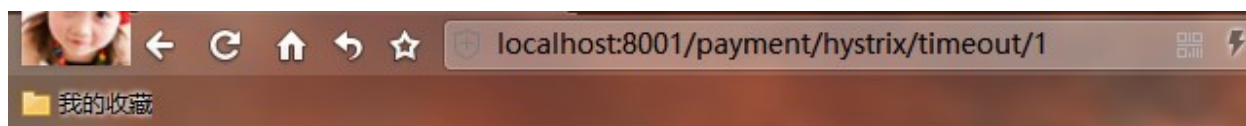
Services		Console	Endpoints
Spring Boot		2020-06-27 19:06:50.616 INFO 7988 --- [restartedMain] o	
Running		2020-06-27 19:06:51.668 INFO 7988 --- [(8)-192.168.5.1] o	
EurekaMain7001 [devtools] :7001/		2020-06-27 19:06:51.669 INFO 7988 --- [(8)-192.168.5.1] o	
PaymentHystrixMain8001 [devtools] :8001/		2020-06-27 19:06:51.681 INFO 7988 --- [(8)-192.168.5.1] o	
Not Started		2020-06-27 19:07:17.797 INFO 7988 --- [freshE] o	
EurekaMain7002 [devtools]		2020-06-27 19:07:17.797 INFO 7988 --- [freshE] o	
PaymentMain8002 [devtools]		2020-06-27 19:07:17.798 INFO 7988 --- [freshE] o	
PaymentMain8004 [devtools]		2020-06-27 19:07:17.798 INFO 7988 --- [freshE] o	
OrderZkMain80 [devtools]		2020-06-27 19:07:17.798 INFO 7988 --- [freshE] o	
OrderMain80 [devtools]		2020-06-27 19:07:17.798 INFO 7988 --- [freshE] o	
OrderConsulMain80 [devtools]		2020-06-27 19:07:17.798 INFO 7988 --- [freshE] o	
PaymentMain8001 [devtools]		2020-06-27 19:07:17.798 INFO 7988 --- [freshE] o	
OrderFeignMain80 [devtools]		2020-06-27 19:07:17.846 INFO 7988 --- [freshE] o	

浏览器访问 <http://localhost:8001/payment/hystrix/ok/1>



线程池:http-nio-8001-exec-2 paymentInfo_OK,id:1 O(∩_∩)O哈哈~

浏览器访问 <http://localhost:8001/payment/hystrix/timeout/1>



线程池:http-nio-8001-exec-8 paymentInfo_TimeOut,id:1 O(∩_∩)O哈哈~ 耗时(秒)3

上述module均通过, 后续会以上述为根基平台, 从正确 -> 错误 -> 降级熔断 -> 恢复 进行演示

3.2. 高并发测试

上述在非高并发情形下,还能勉强满足, 但是...

3.2.1. Jmeter压测测试

3.2.1.1. 安装/下载

下载地址: https://jmeter.apache.org/download_jmeter.cgi

注意事项: 这里使用的Jmeter版本为5.2.1

3.2.1.2. 测试过程

3.2.1.2.1. Jmeter配置

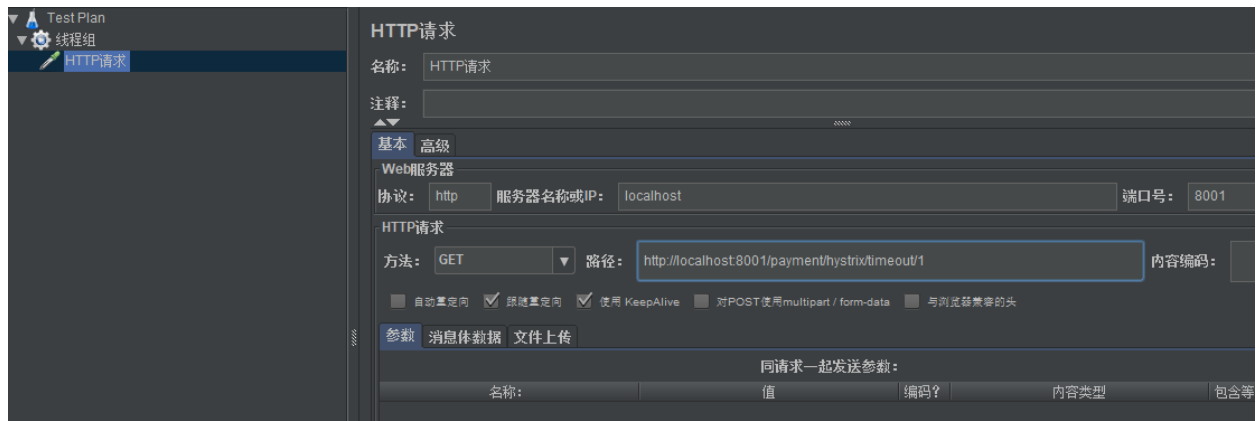
编辑 -> 添加 -> 线程(用户) -> 线程组

- 1 线程数: 2000
- 2 Ramp-Up时间(秒): 0
- 3 循环次数: 100



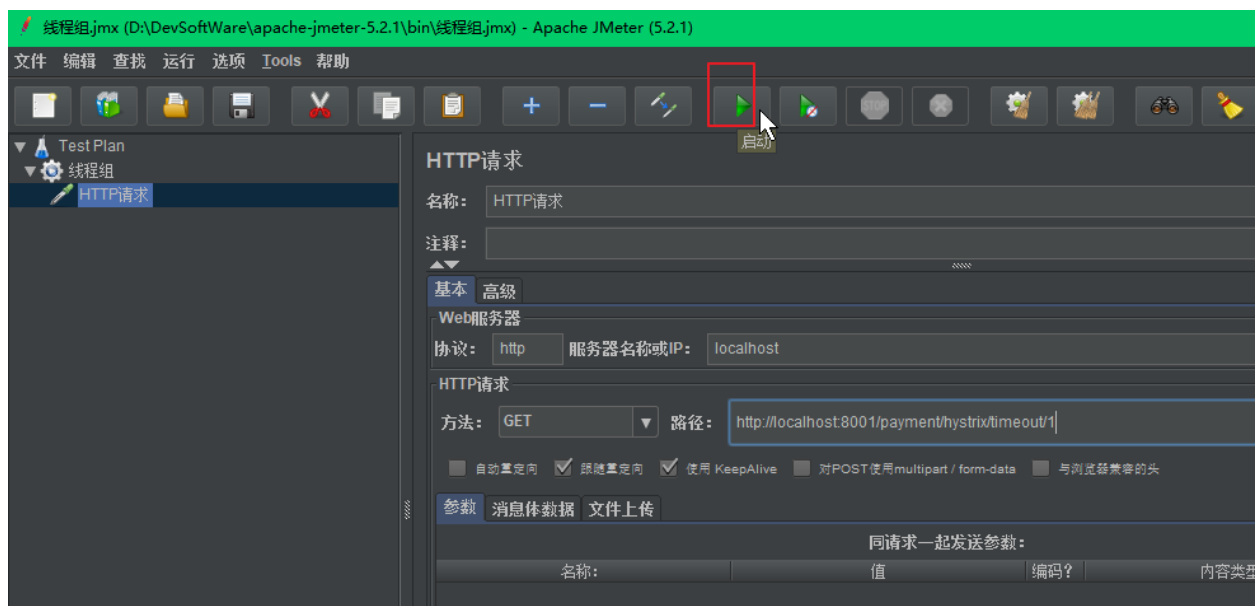
线程组[鼠标右键] -> 添加 -> 取样器 -> HTTP请求

- 1 协议: http
- 2 服务器名称或IP: localhost
- 3 端口号: 8001
- 4 方法: GET
- 5 路径: <http://localhost:8001/payment/hystrix/timeout/1>

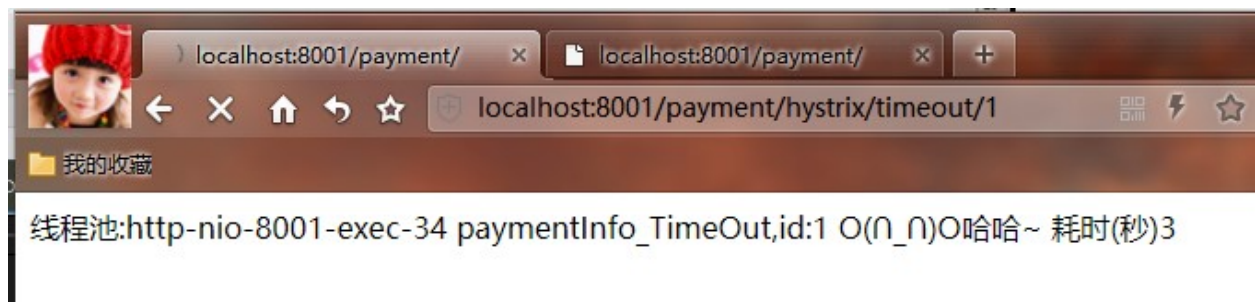


3.2.1.2.2. Jmeter测试

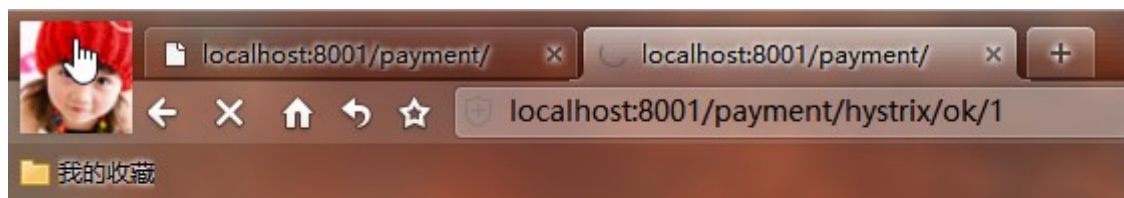
开启Jmeter,来200000个并发压死8001,200000个请求都去访问paymentInfo_TimeOut服务
点击 启动 的按钮图标



浏览器访问 <http://localhost:8001/payment/hystrix/timeout/1>



浏览器访问 <http://localhost:8001/payment/hystrix/ok/1>



线程池:http-nio-8001-exec-34 paymentInfo_OK,id:1 O(n_n)O哈哈~

3.2.1.3. 测试结果

浏览器访问这两个链接时, 两个都在转圈圈, 服务卡死

3.2.1.4. 原因

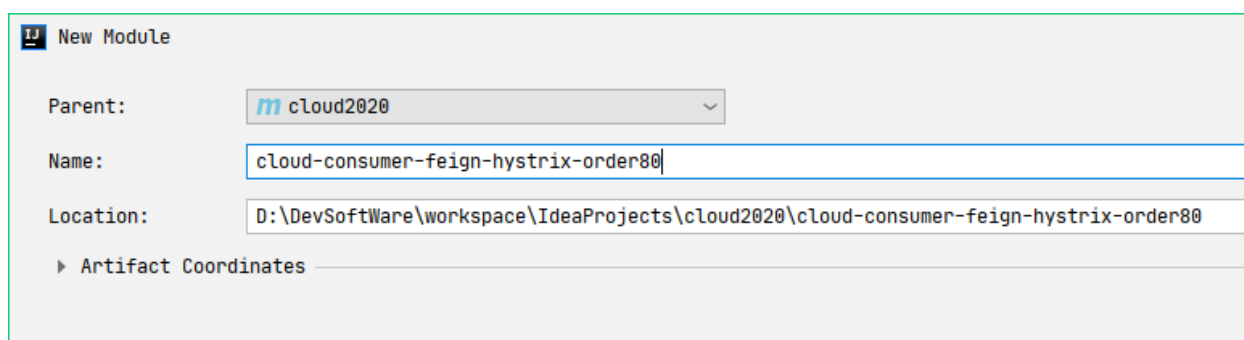
tomcat的默认工作线程数被打满了,没有多余的线程来分解压力和处理

3.2.2. Jmeter压测结论

上面还只是服务提供者8001自己测试,假如此时外部的消费者80也来访问,那消费者只能干等,最终导致消费端80不满意,服务端8001直接被拖死

3.2.3. 新建订单服务消费者order80加入其中

3.2.3.1. 新建模块cloud-consumer-feign-hystrix-order80



3.2.3.2. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0"
5   <parent>
6       <artifactId>cloud2020</artifactId>
7       <groupId>cn.sitedev.springcloud</groupId>
8       <version>1.0-SNAPSHOT</version>
```



```
9     </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>cloud-consumer-feign-hystrix-order80</artifactId>
13     <dependencies>
14         <!--openfeign-->
15         <dependency>
16             <groupId>org.springframework.cloud</groupId>
17             <artifactId>spring-cloud-starter-openfeign</artifactId>
18         </dependency>
19         <!--eureka client-->
20         <dependency>
21             <groupId>org.springframework.cloud</groupId>
22             <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
23         </dependency>
24         <dependency>
25             <groupId>cn.sitedev.springcloud</groupId>
26             <artifactId>cloud-api-commons</artifactId>
27             <version>${project.version}</version>
28         </dependency>
29         <dependency>
30             <groupId>org.springframework.boot</groupId>
31             <artifactId>spring-boot-starter-web</artifactId>
32         </dependency>
33         <!--监控-->
34         <dependency>
35             <groupId>org.springframework.boot</groupId>
36             <artifactId>spring-boot-starter-actuator</artifactId>
37         </dependency>
38         <!--热部署-->
39         <dependency>
40             <groupId>org.springframework.boot</groupId>
41             <artifactId>spring-boot-devtools</artifactId>
42             <scope>runtime</scope>
43             <optional>true</optional>
44         </dependency>
45         <dependency>
46             <groupId>org.projectlombok</groupId>
47             <artifactId>lombok</artifactId>
48             <optional>true</optional>
49         </dependency>
50         <dependency>
51             <groupId>org.springframework.boot</groupId>
```

```

52         <artifactId>spring-boot-starter-test</artifactId>
53         <scope>test</scope>
54     </dependency>
55 </dependencies>
56
57 </project>

```

3.2.3.3. YML

```

1 server:
2   port: 80
3 eureka:
4   client:
5     register-with-eureka: false
6     fetch-registry: true
7     service-url:
8       defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka

```

3.2.3.4. 主启动

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6 import org.springframework.cloud.openfeign.EnableFeignClients;
7
8 @SpringBootApplication
9 @EnableEurekaClient
10 @EnableFeignClients
11 public class OrderHystrixMain80 {
12     public static void main(String[] args) {
13         SpringApplication.run(OrderHystrixMain80.class, args);
14     }
15 }

```

3.2.3.5. 业务类

service:

```
1 package cn.sitedev.springcloud.service;
2
3 import org.springframework.cloud.openfeign.FeignClient;
4 import org.springframework.stereotype.Component;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7
8 @Component
9 @FeignClient(value = "CLOUD-PROVIDER-HYSTRIX-PAYMENT")
10 public interface PaymentHystrixService {
11
12
13     /**
14      * 正常访问
15      *
16      * @param id
17      * @return
18      */
19     @GetMapping("/payment/hystrix/ok/{id}")
20     String paymentInfo_OK(@PathVariable("id") Integer id);
21
22     /**
23      * 超时访问
24      *
25      * @param id
26      * @return
27      */
28     @GetMapping("/payment/hystrix/timeout/{id}")
29     String paymentInfo_TimeOut(@PathVariable("id") Integer id);
30
31 }
```

controller:

```
1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.PaymentHystrixService;
4 import lombok.extern.slf4j.Slf4j;
5 import org.springframework.web.bind.annotation.GetMapping;
```

```

6 import org.springframework.web.bind.annotation.PathVariable;
7 import org.springframework.web.bind.annotation.RestController;
8
9 import javax.annotation.Resource;
10
11 @RestController
12 @Slf4j
13 public class OrderHystrixController {
14     @Resource
15     private PaymentHystrixService paymentHystrixService;
16
17     @GetMapping("/consumer/payment/hystrix/ok/{id}")
18     public String paymentInfo_OK(@PathVariable("id") Integer id) {
19         return paymentHystrixService.paymentInfo_OK(id);
20     }
21
22     @GetMapping("/consumer/payment/hystrix/timeout/{id}")
23     public String paymentInfo_Timeout(@PathVariable("id") Integer id) {
24         return paymentHystrixService.paymentInfo_TimeOut(id);
25     }
26 }

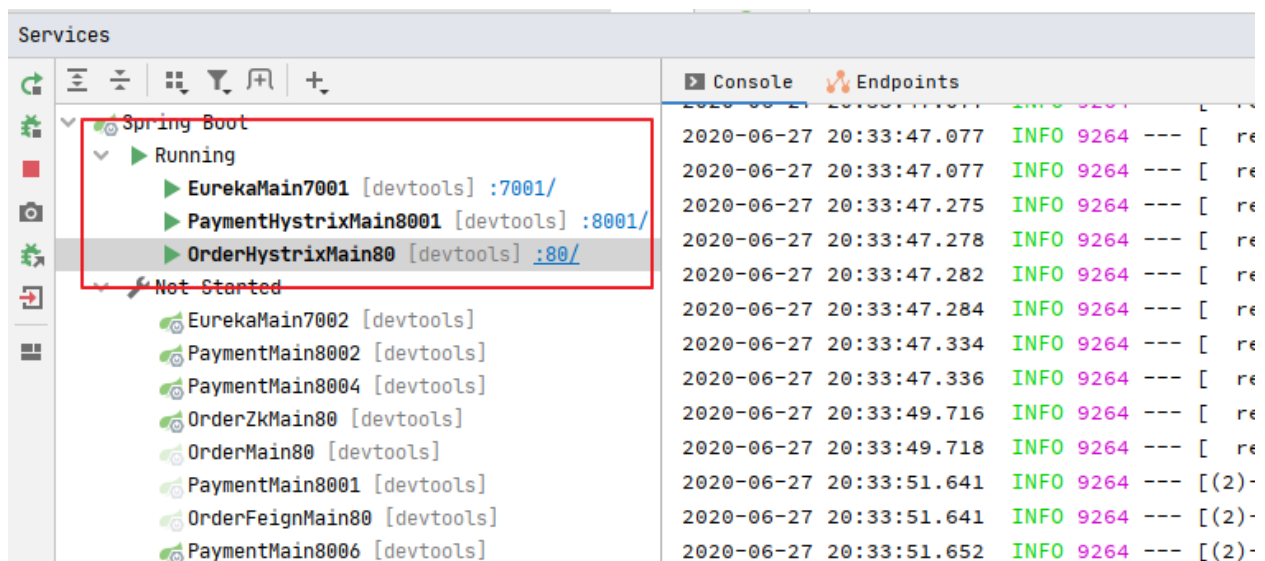
```

3.2.3.6. 正常测试

启动eureka7001

启动payment8001

启动order80



The screenshot shows the Spring Boot IDE interface. On the left, the 'Services' panel lists the running services, with a red box highlighting the following:

- Spring Boot
- Running
 - EurekaMain7001 [devtools] :7001/
 - PaymentHystrixMain8001 [devtools] :8001/
 - OrderHystrixMain80 [devtools] :80/

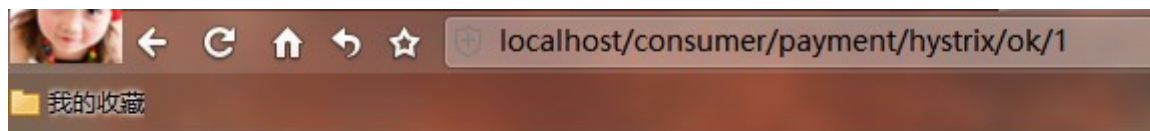
Below the running services, there is a 'Not Started' section with the following services:

- EurekaMain7002 [devtools]
- PaymentMain8002 [devtools]
- PaymentMain8004 [devtools]
- OrderZkMain80 [devtools]
- OrderMain80 [devtools]
- PaymentMain8001 [devtools]
- OrderFeignMain80 [devtools]
- PaymentMain8006 [devtools]

On the right, the 'Console' panel shows log messages for the running services:

Timestamp	Log Level	Message
2020-06-27 20:33:47.077	INFO	9264 --- [re
2020-06-27 20:33:47.077	INFO	9264 --- [re
2020-06-27 20:33:47.275	INFO	9264 --- [re
2020-06-27 20:33:47.278	INFO	9264 --- [re
2020-06-27 20:33:47.282	INFO	9264 --- [re
2020-06-27 20:33:47.284	INFO	9264 --- [re
2020-06-27 20:33:47.334	INFO	9264 --- [re
2020-06-27 20:33:47.336	INFO	9264 --- [re
2020-06-27 20:33:49.716	INFO	9264 --- [re
2020-06-27 20:33:49.718	INFO	9264 --- [re
2020-06-27 20:33:51.641	INFO	9264 --- [(2)-
2020-06-27 20:33:51.641	INFO	9264 --- [(2)-
2020-06-27 20:33:51.652	INFO	9264 --- [(2)-

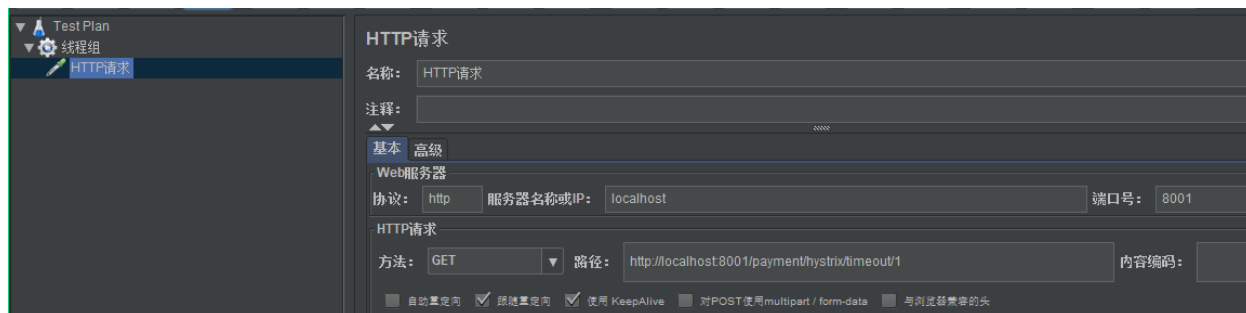
浏览器访问 <http://localhost/consumer/payment/hystrix/ok/1>



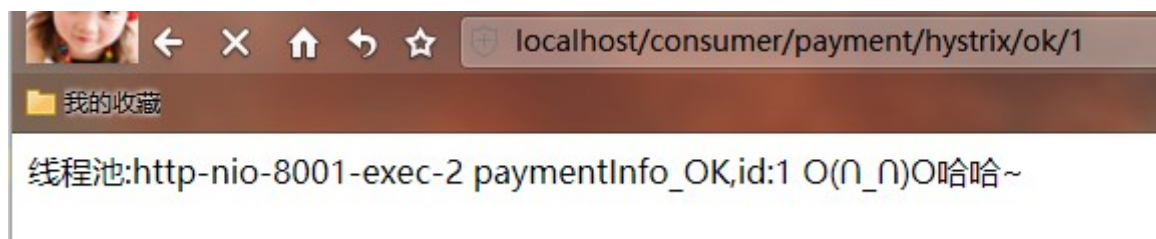
线程池:http-nio-8001-exec-35 paymentInfo_OK,id:1 O(N_N)O哈哈~

3.2.3.7. 高并发测试

3.2.3.7.1. 20w个线程压8001的timeout接口



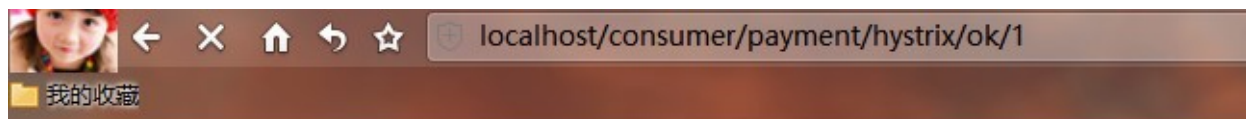
3.2.3.7.2. 消费者80微服务再去访问OK服务8001地址



3.2.3.7.3. 测试结果

浏览器访问: <http://localhost/consumer/payment/hystrix/ok/1>

浏览器要么转圈圈



要么消费端报超时错误



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Jun 27 23:12:01 CST 2020

There was an unexpected error (type=Internal Server Error, status=500).

Read timed out executing GET http://CLOUD-PROVIDER-HYSTRIX-PAYMENT/payment/hystrix/ok/1

```
feign.RetryableException: Read timed out executing GET http://CLOUD-PROVIDER-HYSTRIX-PAYMENT/payment/hystrix/ok/1
    at feign.FeignException.errorExecuting(FeignException.java:213)
    at feign.SynchronousMethodHandler.executeAndDecode(SynchronousMethodHandler.java:115)
    at feign.SynchronousMethodHandler.invoke(SynchronousMethodHandler.java:80)
    at feign.ReflectiveFeign$FeignInvocationHandler.invoke(ReflectiveFeign.java:103)
    at com.sun.proxy.$Proxy114.paymentInfo_OK(Unknown Source)
    at cn.sitedev.springcloud.controller.OrderHyrixController.paymentInfo_OK(OrderHyrixController.java:19)
    at sun.reflect.GeneratedMethodAccessor90.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:190)
    at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:138)
    at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:106)
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:888)
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:793)
    at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)
    at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1040)
```

3.3. 故障和导致现象

8001同一层次的其他接口被困死,因为tomcat线程池里面的工作线程已经被挤占完毕

80此时调用8001,客户端访问响应缓慢,转圈圈

3.4. 上述结论

正因为有上述故障或不佳表现 才有我们的降级/容错/限流等技术诞生

3.5. 如何解决?解决的要求

3.5.1. 要求

超时导致服务器变慢(浏览器请求时转圈) => 超时不再等待

出错(宕机或程序运行出错) => 出错要有兜底

3.5.2. 解决

对方服务(8001)超时了,调用者(80)不能一直卡死等待,必须有服务降级

对方服务(8001)down机了,调用者(80)不能一直卡死等待,必须有服务降级

对方服务(8001)ok,调用者(80)自己有故障或有自我要求(自己的等待时间小于服务提供者)

3.6. 服务降级

3.6.1. 降级配置

@HystrixCommand

3.6.2. 8001先从自身找问题

设置自身调用超时时间的峰值,峰值内可以正常运行,超过了需要有兜底的方法处理,做服务降级 fallback

3.6.3. 8001 fallback

3.6.3.1. 业务类启用

@HystrixCommand报异常后如何处理

一旦调用服务方法失败并抛出了错误信息后,会自动调用@HystrixCommand标注好的 fallbackMethod调用类中的指定方法

图示

```
@HystrixCommand(fallbackMethod = "paymentInfo_TimeOutHandler", commandProperties = {
    @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "3000")
})
public String paymentInfo_TimeOut(Integer id)
{
    int age = 10 / 0;
    int second = 5;
    try { TimeUnit.SECONDS.sleep(second); } catch (InterruptedException e) { e.printStackTrace(); }
    return "线程池:" + Thread.currentThread().getName() + "paymentInfo_TimeOut,id: " + id + "\t" + "O(n_n)O, 耗费秒: " + second;
}
public String paymentInfo_TimeOutHandler(Integer id){
    return "/(T o T)/调用支付接口超时或异常: \t" + "\t当前线程池名字" + Thread.currentThread().getName();
}
```

上图故意制造两个异常:

- 1 int age = 10/0; 计算异常
- 2 我们能接受3秒钟, 它运行5秒钟, 超时异常。

当前服务不可用了, 做服务降级, 兜底的方案都是paymentInfo_TimeOutHandler

```
1 package cn.sitedev.springcloud.service;
2
3 import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
4 import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
5 import org.springframework.stereotype.Service;
6
7 import java.util.concurrent.TimeUnit;
8
9 @Service
10 public class PaymentService {
11     /**
12      * 正常访问
13      *
14      * @param id
15      * @return
16      */
17     public String paymentInfo_OK(Integer id) {
18         return "线程池:" + Thread.currentThread().getName() + " paymentInfo_OK,id:"
19     }
20 }
```

```

20
21  /**
22   * 超时访问
23   *
24   * @param id
25   * @return
26   */
27  @HystrixCommand(fallbackMethod = "paymentInfo_TimeoutHandler", commandProperties =
28      "execution.isolation.thread.timeoutInMilliseconds", value = "3000"))
29  public String paymentInfo_TimeOut(Integer id) {
30      int timeNumber = 5;
31      try {
32          // 暂停3秒钟
33          TimeUnit.SECONDS.sleep(timeNumber);
34      } catch (InterruptedException e) {
35          e.printStackTrace();
36      }
37      return "线程池:" + Thread.currentThread().getName() + " paymentInfo_TimeOut,
38  }
39
40  public String paymentInfo_TimeoutHandler(Integer id) {
41      return "线程池:" + Thread.currentThread().getName() + " paymentInfo_Timeout
42  }
43  }

```

3.6.3.2. 主启动类激活

主启动类添加注解: @EnableCircuitBreaker

```

1  package cn.sitedev.springcloud;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
6  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
7
8  @SpringBootApplication
9  @EnableEurekaClient
10 @EnableCircuitBreaker
11 public class PaymentHystrixMain8001 {
12     public static void main(String[] args) {
13         SpringApplication.run(PaymentHystrixMain8001.class, args);

```



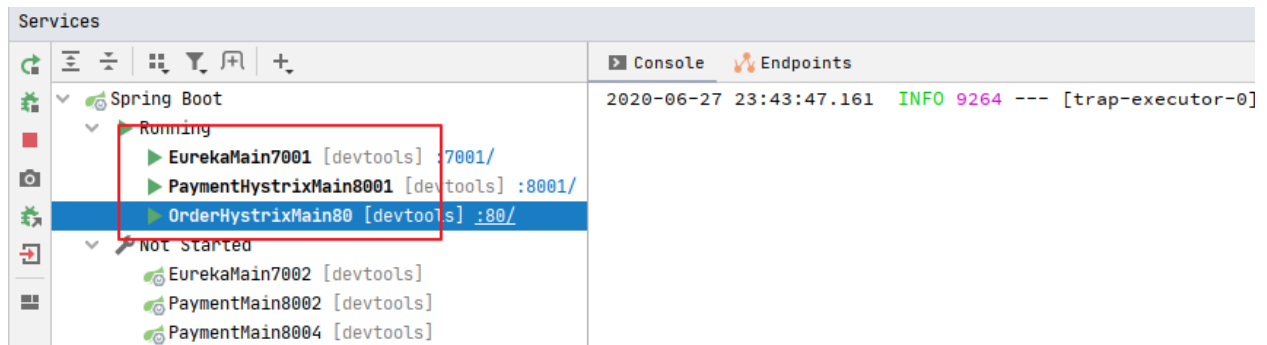
```
14     }
15 }
```

3.6.3.3. 测试

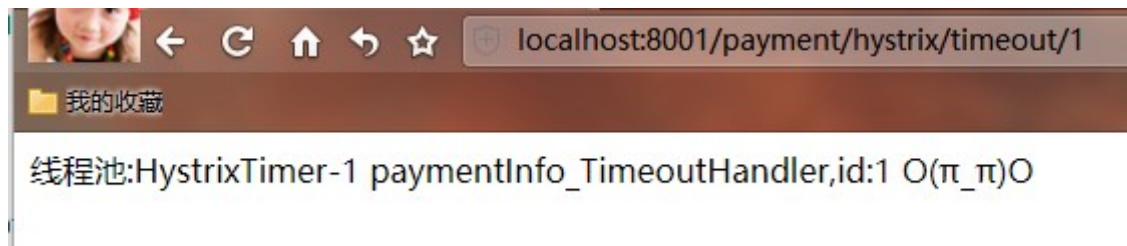
启动eureka7001

启动payment8001

启动order80



浏览器访问 <http://localhost:8001/payment/hystrix/timeout/1>



3.6.4. 80 fallback

3.6.4.1. 说明

80订单微服务,也可以更好的保护自己,自己也依样画葫芦进行客户端降级保护

3.6.4.2. 题外话

我们自己配置过的热部署方式对java代码的改动明显,但对@HystrixCommand内属性的修改建议重启微服务

3.6.4.3. POM

修改内容:

```
1     <!--hystrix-->
2     <dependency>
3         <groupId>org.springframework.cloud</groupId>
4         <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
```

```
5 </dependency>
```

全部内容:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0.xsd">
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>cloud-consumer-feign-hystrix-order80</artifactId>
13    <dependencies>
14        <!--hystrix-->
15        <dependency>
16            <groupId>org.springframework.cloud</groupId>
17            <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
18        </dependency>
19        <!--openfeign-->
20        <dependency>
21            <groupId>org.springframework.cloud</groupId>
22            <artifactId>spring-cloud-starter-openfeign</artifactId>
23        </dependency>
24        <!--eureka client-->
25        <dependency>
26            <groupId>org.springframework.cloud</groupId>
27            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
28        </dependency>
29        <dependency>
30            <groupId>cn.sitedev.springcloud</groupId>
31            <artifactId>cloud-api-commons</artifactId>
32            <version>${project.version}</version>
33        </dependency>
34        <dependency>
35            <groupId>org.springframework.boot</groupId>
36            <artifactId>spring-boot-starter-web</artifactId>
37        </dependency>
38        <!--监控-->
```

```

39     <dependency>
40         <groupId>org.springframework.boot</groupId>
41         <artifactId>spring-boot-starter-actuator</artifactId>
42     </dependency>
43     <!--热部署-->
44     <dependency>
45         <groupId>org.springframework.boot</groupId>
46         <artifactId>spring-boot-devtools</artifactId>
47         <scope>runtime</scope>
48         <optional>true</optional>
49     </dependency>
50     <dependency>
51         <groupId>org.projectlombok</groupId>
52         <artifactId>lombok</artifactId>
53         <optional>true</optional>
54     </dependency>
55     <dependency>
56         <groupId>org.springframework.boot</groupId>
57         <artifactId>spring-boot-starter-test</artifactId>
58         <scope>test</scope>
59     </dependency>
60 </dependencies>
61
62 </project>

```

3.6.4.4. YML

修改内容:

```

1 feign:
2   hystrix:
3     enabled: true

```

全部内容:

```

1 server:
2   port: 80
3 eureka:
4   client:
5     register-with-eureka: false
6     fetch-registry: true

```

```

7     service-url:
8         defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
9
10 feign:
11     hystrix:
12         enabled: true

```

3.6.4.5. 主启动

主启动类添加@EnableHystrix注解

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6 import org.springframework.cloud.netflix.hystrix.EnableHystrix;
7 import org.springframework.cloud.openfeign.EnableFeignClients;
8
9 @SpringBootApplication
10 @EnableEurekaClient
11 @EnableFeignClients
12 @EnableHystrix
13 public class OrderHystrixMain80 {
14     public static void main(String[] args) {
15         SpringApplication.run(OrderHystrixMain80.class, args);
16     }
17 }

```

3.6.4.6. 业务类

controller:

```

1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.PaymentHystrixService;
4 import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
5 import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
6 import lombok.extern.slf4j.Slf4j;
7 import org.springframework.web.bind.annotation.GetMapping;

```

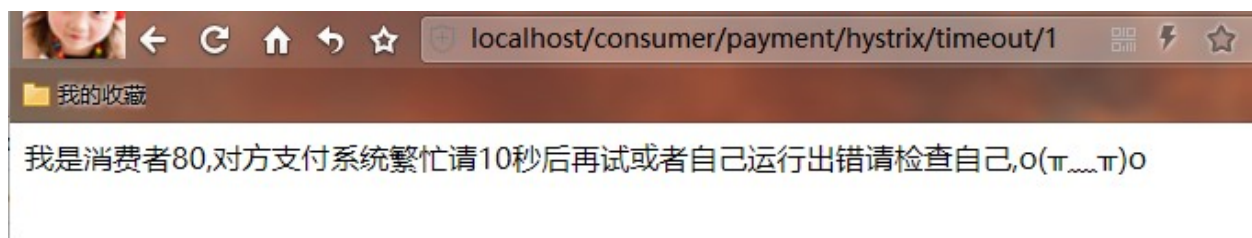
```

8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import javax.annotation.Resource;
12
13 @RestController
14 @Slf4j
15 public class OrderHystrixController {
16     @Resource
17     private PaymentHystrixService paymentHystrixService;
18
19     @GetMapping("/consumer/payment/hystrix/ok/{id}")
20     public String paymentInfo_OK(@PathVariable("id") Integer id) {
21         return paymentHystrixService.paymentInfo_OK(id);
22     }
23
24     @GetMapping("/consumer/payment/hystrix/timeout/{id}")
25     @HystrixCommand(fallbackMethod = "paymentTimeOutFallbackMethod", commandProperties = {
26         @HystrixProperty(value = "execution.isolation.thread.timeoutInMilliseconds", value = "1500")})
27     public String paymentInfo_Timeout(@PathVariable("id") Integer id) {
28         return paymentHystrixService.paymentInfo_TimeOut(id);
29     }
30
31     public String paymentTimeOutFallbackMethod(@PathVariable("id") Integer id) {
32         return "我是消费者80,对方支付系统繁忙请10秒后再试或者自己运行出错请检查自己,o(π_π)o";
33     }
34 }

```

3.6.4.7. 测试

浏览器访问 <http://localhost/consumer/payment/hystrix/timeout/1>



3.6.5. 目前问题

每个业务方法对应一个兜底的方法,代码膨胀

统一和自定义的分开

3.6.6. 解决方法

3.6.6.1. 每个方法配置一个兜底方法, 代码膨胀

3.6.6.1.1. @DefaultProperties(defaultFallback="")

@DefaultProperties(defaultFallback="")

1: 1每个方法配置一个服务降级方法, 技术上可以, 实际上傻X

1: N除了个别重要核心业务有专属, 其它普通的可以通过

@DefaultProperties(defaultFallback="")统一跳转到统一处理结果页面

通用的和独享的各自分开, 避免了代码膨胀, 合理减少了代码量

```
13
14 @RestController
15 @Slf4j
16 @DefaultProperties(defaultFallback = "paymentGlobalFallbackMethod")
17 public class OrderHystrixController {
18     @Resource
19     private PaymentHystrixService paymentHystrixService;
20
21     @GetMapping("/consumer/payment/hystrix/ok/{id}")
22     public String paymentInfo_OK(@PathVariable("id") Integer id) { return paymentHystrixService.paymentInfo_OK(id); }
23
24     @GetMapping("/consumer/payment/hystrix/timeout/{id}")
25     @HystrixCommand(fallbackMethod = "paymentTimeOutFallbackMethod", commandProperties = {@HystrixProperty(name =
26         "execution.isolation.thread.timeoutInMilliseconds", value = "1500")})
27     public String paymentInfo_Timeout(@PathVariable("id") Integer id) {
28         return paymentHystrixService.paymentInfo_TimeOut(id);
29     }
30
31     public String paymentTimeOutFallbackMethod(@PathVariable("id") Integer id) {
32         return "我是消费者80,对方支付系统繁忙请10秒后再试或者自己运行出错请检查自己,ο(Π—Π)ο";
33     }
34
35     @GetMapping("/consumer/payment/hystrix/exception/{id}")
36     // @HystrixCommand注解没有配置FallbackMethod属性, 则使用全局fallback
37     @HystrixCommand
38     public String paymentInfo_Exception() {
39         int i = 10 / 0;
40         return "计算 10 / 0 结果: " + i;
41     }
42
43     // 下面是全局fallback
44     public String paymentGlobalFallbackMethod() {
45         return "Global异常处理信息,请稍后重试.ο(Π—Π)ο";
46     }
47
48 }
```

没有特别指明的就用统一的

3.6.6.1.2. controller配置

```
1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.PaymentHystrixService;
4 import com.netflix.hystrix.contrib.javanica.annotation.DefaultProperties;
5 import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
6 import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
7 import lombok.extern.slf4j.Slf4j;
8 import org.springframework.web.bind.annotation.GetMapping;
```

```

9 import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import javax.annotation.Resource;
13
14 @RestController
15 @Slf4j
16 @DefaultProperties(defaultFallback = "paymentGlobalFallbackMethod")
17 public class OrderHystrixController {
18     @Resource
19     private PaymentHystrixService paymentHystrixService;
20
21     @GetMapping("/consumer/payment/hystrix/ok/{id}")
22     public String paymentInfo_OK(@PathVariable("id") Integer id) {
23         return paymentHystrixService.paymentInfo_OK(id);
24     }
25
26     @GetMapping("/consumer/payment/hystrix/timeout/{id}")
27     @HystrixCommand(fallbackMethod = "paymentTimeOutFallbackMethod", commandProperties(
28         "execution.isolation.thread.timeoutInMilliseconds", value = "1500"))
29     public String paymentInfo_Timeout(@PathVariable("id") Integer id) {
30         return paymentHystrixService.paymentInfo_TimeOut(id);
31     }
32
33     public String paymentTimeOutFallbackMethod(@PathVariable("id") Integer id) {
34         return "我是消费者80,对方支付系统繁忙请10秒后再试或者自己运行出错请检查自己,";
35     }
36
37     @GetMapping("/consumer/payment/hystrix/exception/{id}")
38     // @HystrixCommand注解没有配置fallbackMethod属性, 则使用全局fallback
39     @HystrixCommand
40     public String paymentInfo_Exception() {
41         int i = 10 / 0;
42         return "计算 10 / 0 结果: " + i;
43     }
44
45     // 下面是全局fallback
46     public String paymentGlobalFallbackMethod() {
47         return "Global异常处理信息,请稍后重试.ο(╥﹏╥)ο";
48     }
49 }

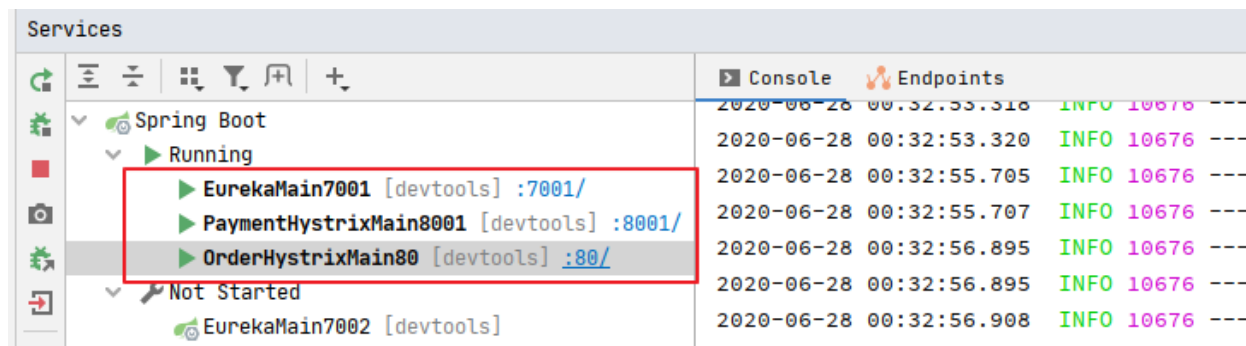
```

3.6.6.1.3. 测试

启动eureka7001

启动payment8001

启动order80



浏览器访问 <http://localhost/consumer/payment/hystrix/exception/1>



3.6.6.2. 和业务逻辑混在一起, 代码逻辑混乱

3.6.6.2.1. 说明

服务降级,客户端去调用服务端,碰上服务端宕机或关闭

本次案例服务降级处理是在客户端80实现完成,与服务端8001没有关系

只需要为Feign客户端定义的接口添加一个服务降级处理的实现类即可实现解耦

3.6.6.2.2. 未来我们要面对异常

运行时异常

超时

宕机

3.6.6.2.3. 再看看我们的业务类OrderHystrixController

```
1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.PaymentHystrixService;
4 import com.netflix.hystrix.contrib.javanica.annotation.DefaultProperties;
5 import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
6 import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
```



```

7 import lombok.extern.slf4j.Slf4j;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import javax.annotation.Resource;
13
14 @RestController
15 @Slf4j
16 @DefaultProperties(defaultFallback = "paymentGlobalFallbackMethod")
17 public class OrderHystrixController {
18     @Resource
19     private PaymentHystrixService paymentHystrixService;
20
21     @GetMapping("/consumer/payment/hystrix/ok/{id}")
22     public String paymentInfo_OK(@PathVariable("id") Integer id) {
23         return paymentHystrixService.paymentInfo_OK(id);
24     }
25
26     @GetMapping("/consumer/payment/hystrix/timeout/{id}")
27     @HystrixCommand(fallbackMethod = "paymentTimeOutFallbackMethod", commandProperties(
28         "execution.isolation.thread.timeoutInMilliseconds", value = "1500"))
29     public String paymentInfo_Timeout(@PathVariable("id") Integer id) {
30         return paymentHystrixService.paymentInfo_TimeOut(id);
31     }
32
33     public String paymentTimeOutFallbackMethod(@PathVariable("id") Integer id) {
34         return "我是消费者80,对方支付系统繁忙请10秒后再试或者自己运行出错请检查自己,";
35     }
36
37     @GetMapping("/consumer/payment/hystrix/exception/{id}")
38     // @HystrixCommand注解没有配置fallbackMethod属性, 则使用全局fallback
39     @HystrixCommand
40     public String paymentInfo_Exception() {
41         int i = 10 / 0;
42         return "计算 10 / 0 结果: " + i;
43     }
44
45     // 下面是全局fallback
46     public String paymentGlobalFallbackMethod() {
47         return "Global异常处理信息,请稍后重试.o(╥﹏╥)o";
48     }
49 }

```

```

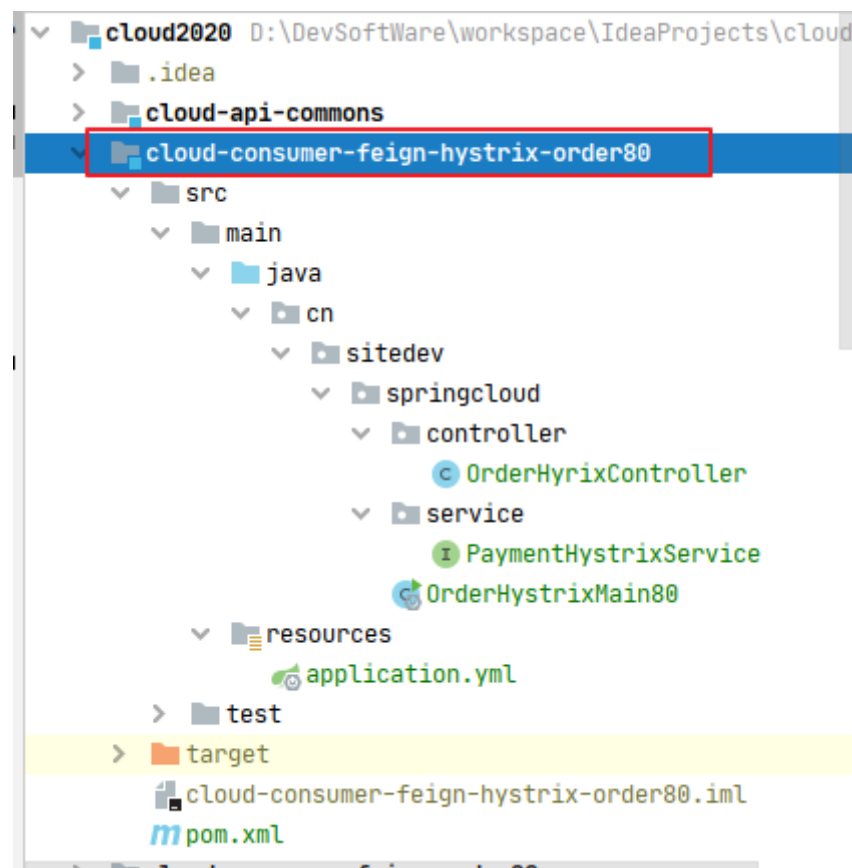
    @GetMapping("/consumer/payment/hystrix/timeout/{id}")
    @HystrixCommand(fallbackMethod = "paymentTimeOutFallbackMethod", commandProperties =
    {
        @HystrixProperty(name =
        "execution.isolation.thread.timeoutInMilliseconds", value = "1500"))
    public String paymentInfo_Timeout(@PathVariable("id") Integer id) {
        return paymentHystrixService.paymentInfo_TimeOut(id);
    }

    public String paymentTimeOutFallbackMethod(@PathVariable("id") Integer id) {
        return "我是消费者80,对方支付系统繁忙请10秒后再试或者自己运行出错请检查自己,ο(╯□╰)ο";
    }

```

混合在一起, 每个业务方法都要提供一个

3.6.6.2.4. 修改cloud-consumer-feign-hystrix-order80



3.6.6.2.5. 统一异常处理

根据cloud-consumer-feign-hystrix-order80已经有的PaymentHystrixService接口,重新新建一个类(PaymentFallbackService)实现接口,统一为接口里面的方法进行异常处理

3.6.6.2.6. PaymentFallbackService类实现PaymentFeginService接口

```

1 package cn.sitedev.springcloud.service;
2

```

```

3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class PaymentFallbackService implements PaymentHystrixService {
7     @Override
8     public String paymentInfo_OK(Integer id) {
9         return "-----PaymentFallbackService fall back-paymentInfo_OK, o(ᵀᵀᵀᵀ)ᵀ";
10    }
11
12    @Override
13    public String paymentInfo_TimeOut(Integer id) {
14        return "-----PaymentFallbackService fall back-paymentInfo_Timeout, o(ᵀᵀᵀᵀ)ᵀ";
15    }
16 }

```

3.6.6.2.7. YML

修改内容:

```

1 feign:
2   hystrix:
3     enabled: true # 在Feign中开启Hystrix

```

完整内容:

```

1 server:
2   port: 80
3 eureka:
4   client:
5     register-with-eureka: false
6     fetch-registry: true
7     service-url:
8       defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
9
10 feign:
11   hystrix:
12     enabled: true # 在Feign中开启Hystrix

```

3.6.6.2.8. PaymentFeignClientService接口

```

1 package cn.sitedev.springcloud.service;
2
3 import org.springframework.cloud.openfeign.FeignClient;
4 import org.springframework.stereotype.Component;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7
8 @Component
9 @FeignClient(value = "CLOUD-PROVIDER-HYSTRIX-PAYMENT", fallback = PaymentFallbackSer
10 public interface PaymentHystrixService {
11     /**
12      * 正常访问
13      *
14      * @param id
15      * @return
16      */
17     @GetMapping("/payment/hystrix/ok/{id}")
18     String paymentInfo_OK(@PathVariable("id") Integer id);
19
20     /**
21      * 超时访问
22      *
23      * @param id
24      * @return
25      */
26     @GetMapping("/payment/hystrix/timeout/{id}")
27     String paymentInfo_TimeOut(@PathVariable("id") Integer id);
28
29 }

```

```

@Component
@FeignClient(value = "CLOUD-PROVIDER-HYSTRIX-PAYMENT", fallback = PaymentFallbackService.class)
public interface PaymentHystrixService {

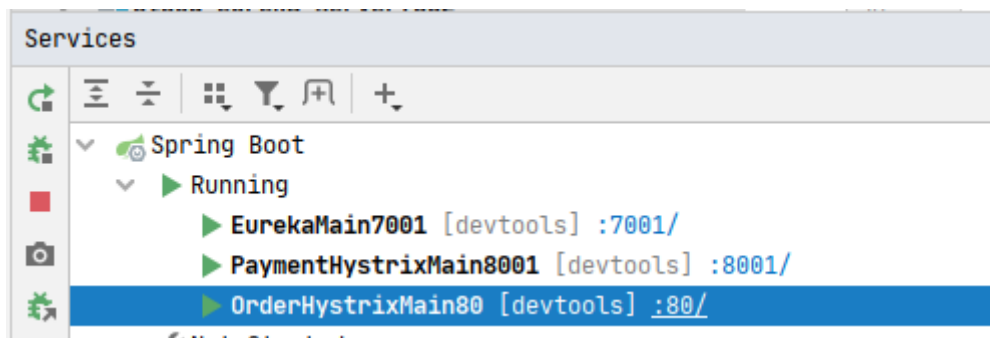
```

3.6.6.2.9. 测试

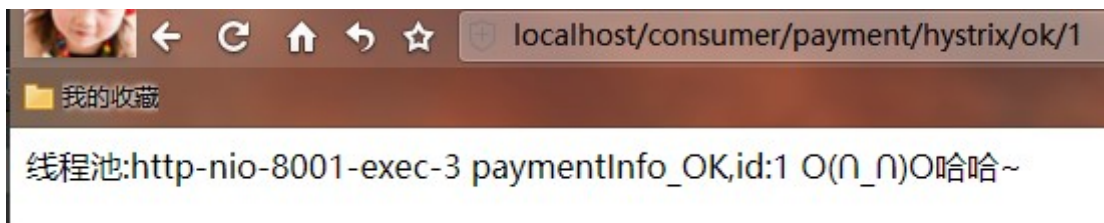
单个eureka先启动7001

PaymentHystrixMain8001启动

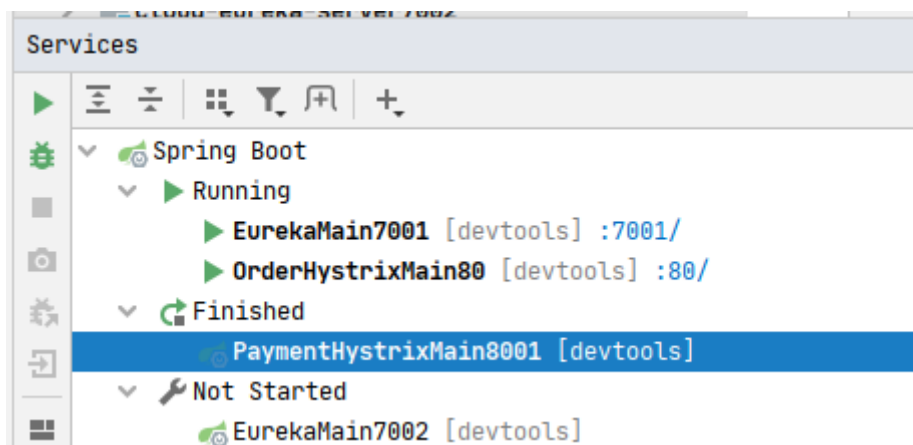
OrderHystrixMain80启动



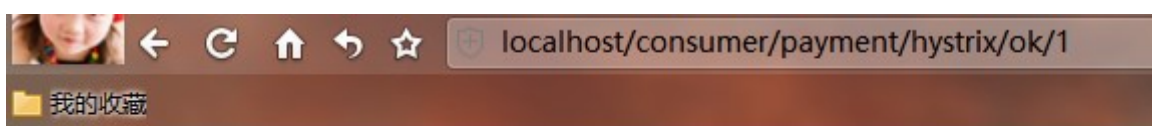
正常访问测试, 浏览器访问: <http://localhost/consumer/payment/hystrix/ok/1>



故意关闭微服务8001



再次浏览器访问: <http://localhost/consumer/payment/hystrix/ok/1> 客户端自己调用提示



-----PaymentFallbackService fall back-paymentInfo_OK, o(∩_∩)O哈哈~

此时服务端provider已经down了,但是我们做了服务降级处理,让客户端在服务端不可用时也会获得提示信息而不会挂起耗死服务器

3.7. 服务熔断

3.7.1. 断路器

一句话就是家里的保险丝

3.7.2. 熔断是什么

熔断机制概述

熔断机制是应对雪崩效应的一种微服务链路保护机制。当扇出链路的某个微服务出错不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速返回错误的响应信息。

当检测到该节点微服务调用响应正常后，恢复调用链路

在 Spring Cloud框架里，熔断机制通过 Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败，就会启动熔断机制。熔断机制的注解是 @HystrixCommand

大神论文

<https://martinfowler.com/bliki/CircuitBreaker.html>



The screenshot shows the top of a web browser displaying the article 'CircuitBreaker' by Martin Fowler on the website martinFowler.com. The page has a dark blue header with the site name and navigation links: Refactoring, Agile, Architecture, About, and ThoughtWorks. Below the header, the article title 'CircuitBreaker' is prominently displayed. The author's name, Martin Fowler, is shown next to a small profile picture. The article is dated 6 March 2014. The main text begins with a paragraph explaining the commonality of remote calls in software systems and the risks of cascading failures. It mentions Michael Nygard's book 'Release It' and the Circuit Breaker pattern. The text is partially visible, showing the beginning of a paragraph about the basic idea of the circuit breaker.

3.7.3. 实操

修改cloud-provider-hystrix-payment8001

3.7.4.1. PaymentService

```
1 package cn.sitedev.springcloud.service;
2
3 import cn.hutool.core.util.IdUtil;
4 import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
5 import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
6 import org.springframework.stereotype.Service;
```

```

7 import org.springframework.web.bind.annotation.PathVariable;
8
9 import java.util.concurrent.TimeUnit;
10
11 @Service
12 public class PaymentService {
13     /**
14      * 正常访问
15      *
16      * @param id
17      * @return
18      */
19     public String paymentInfo_OK(Integer id) {
20         return "线程池:" + Thread.currentThread().getName() + " paymentInfo_OK,id:"
21     }
22
23     /**
24      * 超时访问
25      *
26      * @param id
27      * @return
28      */
29     @HystrixCommand(fallbackMethod = "paymentInfo_TimeoutHandler", commandProperties =
30         "execution.isolation.thread.timeoutInMilliseconds", value = "3000"))
31     public String paymentInfo_TimeOut(Integer id) {
32         int timeNumber = 5;
33         try {
34             // 暂停3秒钟
35             TimeUnit.SECONDS.sleep(timeNumber);
36         } catch (InterruptedException e) {
37             e.printStackTrace();
38         }
39         return "线程池:" + Thread.currentThread().getName() + " paymentInfo_TimeOut,"
40     }
41
42     public String paymentInfo_TimeoutHandler(Integer id) {
43         return "线程池:" + Thread.currentThread().getName() + " paymentInfo_Timeout,"
44     }
45
46     ////////////////服务熔断////////////////////////////////////
47     @HystrixCommand(fallbackMethod = "paymentCircuitBreaker_fallback", commandProperties =
48         // 是否开启断路器
49         @HystrixProperty(name = "circuitBreaker.enabled", value = "true"),

```

```

50         // 请求次数
51         @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value =
52         // 时间窗口期
53         @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value
54         // 失败率达到多少后跳闸
55         @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value
56     public String paymentCircuitBreaker(@PathVariable("id") Integer id) {
57         if (id < 0) {
58             throw new RuntimeException("*****id 不能为负数");
59         }
60         String serialNumber = IdUtil.simpleUUID();
61         return Thread.currentThread().getName() + "\t调用成功, 流水号: " + serialNum
62     }
63
64     public String paymentCircuitBreaker_fallback(@PathVariable("id") Integer id) {
65         return "id 不能为负数, 请稍后再试..., id: " + id;
66     }
67 }

```

```

////////////////////服务熔断////////////////////////////////////
}
@HystrixCommand(fallbackMethod = "paymentCircuitBreaker_fallback", commandProperties = {
    // 是否开启断路器
    @HystrixProperty(name = "circuitBreaker.enabled", value = "true"),
    // 请求次数
    @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "10"),
    // 时间窗口期
    @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "10000"),
    // 失败率达到多少后跳闸
    @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "60"))
}
public String paymentCircuitBreaker(@PathVariable("id") Integer id) {
    if (id < 0) {
        throw new RuntimeException("*****id 不能为负数");
    }
    String serialNumber = IdUtil.simpleUUID();
    return Thread.currentThread().getName() + "\t调用成功, 流水号: " + serialNumber;
}

}

public String paymentCircuitBreaker_fallback(@PathVariable("id") Integer id) {
    return "id 不能为负数, 请稍后再试..., id: " + id;
}

}
}

```

为什么是配置这些参数

详情参见<https://github.com/Netflix/Hystrix/wiki/How-it-Works#circuit-breaker>

The precise way that the circuit opening and closing occurs is as follows:

1. Assuming the volume across a circuit meets a certain threshold
(`HystrixCommandProperties.circuitBreakerRequestVolumeThreshold()`)...
2. And assuming that the error percentage exceeds the threshold error percentage
(`HystrixCommandProperties.circuitBreakerErrorThresholdPercentage()`)...
3. Then the circuit-breaker transitions from `CLOSED` to `OPEN`.
4. While it is open, it short-circuits all requests made against that circuit-breaker.
5. After some amount of time (`HystrixCommandProperties.circuitBreakerSleepWindowInMilliseconds()`), the next single request is let through (this is the `HALF-OPEN` state). If the request fails, the circuit-breaker returns to the `OPEN` state for the duration of the sleep window. If the request succeeds, the circuit-breaker transitions to `CLOSED` and the logic in 1. takes over again.

具体的配置项可以参见HystrixCommandProperties

```
protected HystrixCommandProperties(HystrixCommandKey key, HystrixCommandProperties.Setter builder, String propertyPrefix) {
    this.key = key;
    this.circuitBreakerEnabled = getProperty(propertyPrefix, key, instanceProperty: "circuitBreaker.enabled", builder.getCircuitBreakerEnabled(), def
    this.circuitBreakerRequestVolumeThreshold = getProperty(propertyPrefix, key, instanceProperty: "circuitBreaker.requestVolumeThreshold", builder.g
    this.circuitBreakerSleepWindowInMilliseconds = getProperty(propertyPrefix, key, instanceProperty: "circuitBreaker.sleepWindowInMilliseconds", bui
    this.circuitBreakerErrorThresholdPercentage = getProperty(propertyPrefix, key, instanceProperty: "circuitBreaker.errorThresholdPercentage", build
    this.circuitBreakerForceOpen = getProperty(propertyPrefix, key, instanceProperty: "circuitBreaker.forceOpen", builder.getCircuitBreakerForceOpen(
    this.circuitBreakerForceClosed = getProperty(propertyPrefix, key, instanceProperty: "circuitBreaker.forceClosed", builder.getCircuitBreakerForceC
    this.executionIsolationStrategy = getProperty(propertyPrefix, key, instanceProperty: "execution.isolation.strategy", builder.getExecutionIsolatio
    this.executionTimeoutInMilliseconds = getProperty(propertyPrefix, key, instanceProperty: "execution.isolation.thread.timeoutInMilliseconds", buil
    this.executionTimeoutEnabled = getProperty(propertyPrefix, key, instanceProperty: "execution.timeout.enabled", builder.getExecutionTimeoutEnabled
    this.executionIsolationThreadInterruptOnTimeout = getProperty(propertyPrefix, key, instanceProperty: "execution.isolation.thread.interruptOnTimeo
    this.executionIsolationThreadInterruptOnFutureCancel = getProperty(propertyPrefix, key, instanceProperty: "execution.isolation.thread.interruptOn
    this.executionIsolationSemaphoreMaxConcurrentRequests = getProperty(propertyPrefix, key, instanceProperty: "execution.isolation.semaphore.maxConc
    this.fallbackIsolationSemaphoreMaxConcurrentRequests = getProperty(propertyPrefix, key, instanceProperty: "fallback.isolation.semaphore.maxConcur
    this.fallbackEnabled = getProperty(propertyPrefix, key, instanceProperty: "fallback.enabled", builder.getFallbackEnabled(), default_fallbackEnabl
    this.metricsRollingStatisticalWindowInMilliseconds = getProperty(propertyPrefix, key, instanceProperty: "metrics.rollingStats.timeInMilliseconds"
    this.metricsRollingStatisticalWindowBuckets = getProperty(propertyPrefix, key, instanceProperty: "metrics.rollingStats.numBuckets", builder.getMe
    this.metricsRollingPercentileEnabled = getProperty(propertyPrefix, key, instanceProperty: "metrics.rollingPercentile.enabled", builder.getMetrics
    this.metricsRollingPercentileWindowInMilliseconds = getProperty(propertyPrefix, key, instanceProperty: "metrics.rollingPercentile.timeInMilliseco
    this.metricsRollingPercentileWindowBuckets = getProperty(propertyPrefix, key, instanceProperty: "metrics.rollingPercentile.numBuckets", builder.g
    this.metricsRollingPercentileBucketSize = getProperty(propertyPrefix, key, instanceProperty: "metrics.rollingPercentile.bucketSize", builder.getM
    this.metricsHealthSnapshotIntervalInMilliseconds = getProperty(propertyPrefix, key, instanceProperty: "metrics.healthSnapshot.intervalInMilliseco
    this.requestCacheEnabled = getProperty(propertyPrefix, key, instanceProperty: "requestCache.enabled", builder.getRequestCacheEnabled(), default_r
    this.requestLogEnabled = getProperty(propertyPrefix, key, instanceProperty: "requestLog.enabled", builder.getRequestLogEnabled(), default_request
    this.executionIsolationThreadPoolKeyOverride = HystrixPropertiesChainedProperty.forString().add( name: propertyPrefix + ".command." + key.name()
```

3.7.4.2. PaymentController

```
1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.PaymentService;
4 import lombok.extern.slf4j.Slf4j;
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PathVariable;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import javax.annotation.Resource;
11
12 @RestController
13 @Slf4j
14 public class PaymentController {
15     @Resource
16     private PaymentService paymentService;
17 }
```

```
18     @Value("${server.port}")
19     private String servicePort;
20
21     /**
22      * 正常访问
23      *
24      * @param id
25      * @return
26      */
27     @GetMapping("/payment/hystrix/ok/{id}")
28     public String paymentInfo_OK(@PathVariable("id") Integer id) {
29         String result = paymentService.paymentInfo_OK(id);
30         log.info("*****result:" + result);
31         return result;
32     }
33
34     /**
35      * 超时访问
36      *
37      * @param id
38      * @return
39      */
40     @GetMapping("/payment/hystrix/timeout/{id}")
41     public String paymentInfo_TimeOut(@PathVariable("id") Integer id) {
42         String result = paymentService.paymentInfo_TimeOut(id);
43         log.info("*****result:" + result);
44         return result;
45     }
46
47     @GetMapping("/payment/circuit/{id}")
48     public String paymentCircuitBreaker(@PathVariable("id") Integer id) {
49         String result = paymentService.paymentCircuitBreaker(id);
50         log.info("*****result: " + result);
51         return result;
52     }
53
54 }
```

```

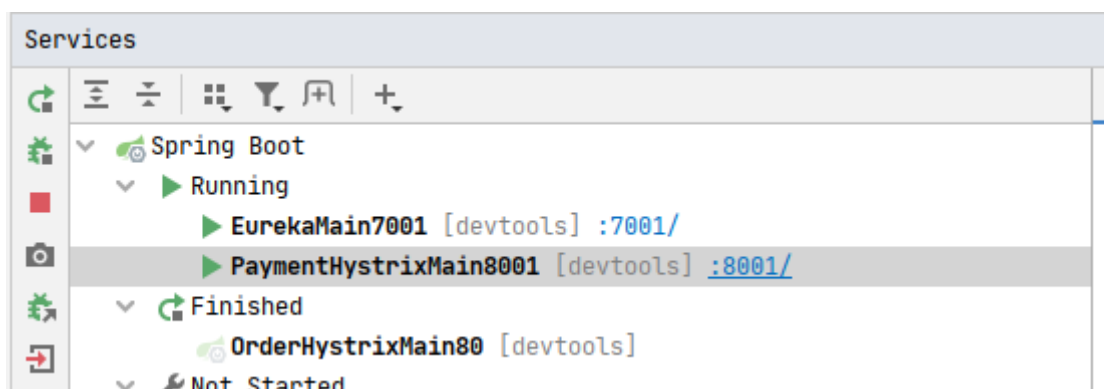
@GetMapping("/payment/circuit/{id}")
public String paymentCircuitBreaker(@PathVariable("id") Integer id) {
    String result = paymentService.paymentCircuitBreaker(id);
    log.info("*****result: " + result);
    return result;
}

```

3.7.4.3. 测试

启动eureka7001

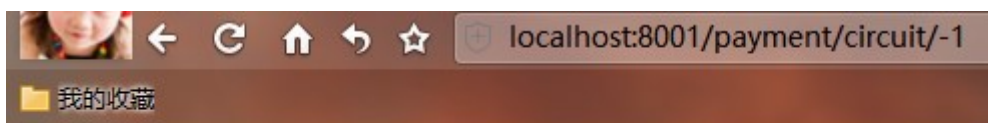
启动payment8001



测试正常情况, 浏览器访问: <http://localhost:8001/payment/circuit/1>

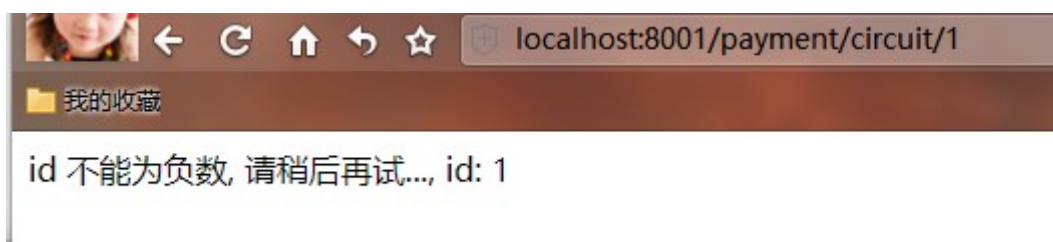


测试异常情况, 浏览器访问: <http://localhost:8001/payment/circuit/-1>

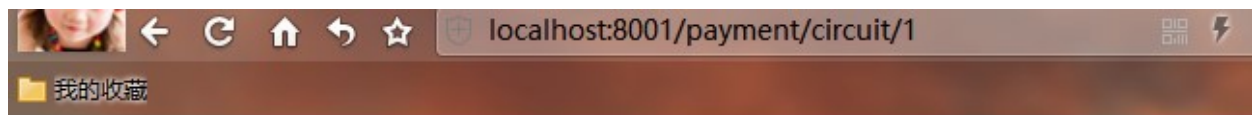


id 不能为负数, 请稍后再试..., id: -1

一次正确一次错误地不停尝试, 直到看到如下图示



重点测试:多次正确,然后慢慢正确,发现刚开始不满足条件,就算是正确的访问也不能进行
继续尝试访问正确地址, 会发现能够正常访问了

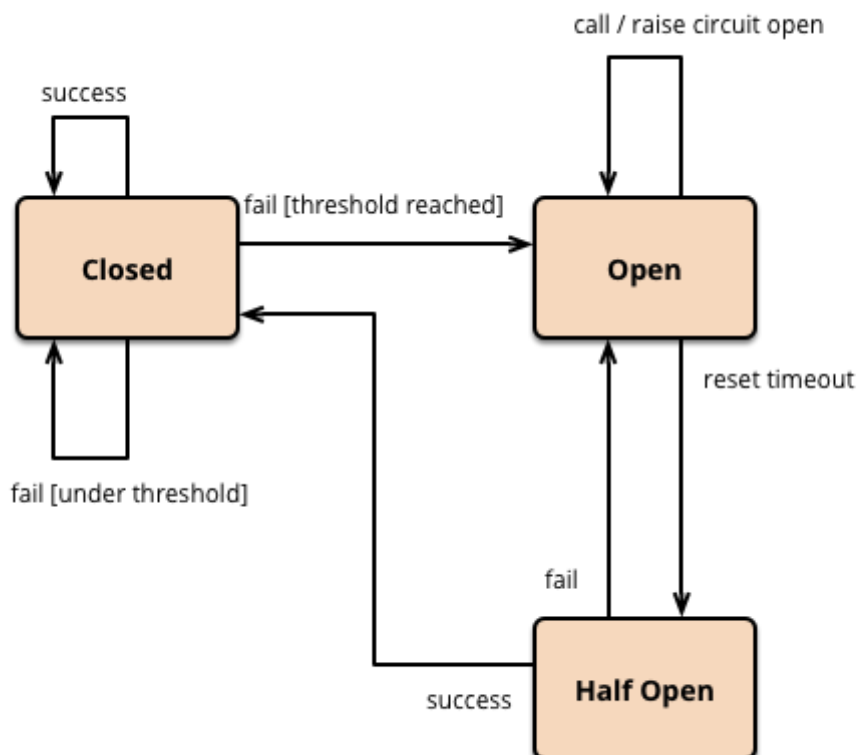


hystrix-PaymentService-10 调用成功, 流水号: f53ddc40fd084d5a86563b2592a1f811

3.7.4. 原理/小总结

3.7.4.1. 大神结论

<https://martinfowler.com/bliki/CircuitBreaker.html>



3.7.4.2. 熔断类型

熔断打开

请求不再调用当前服务,内部设置一般为MTTR(平均故障处理时间),当打开长达到所设时钟则进入半熔断状态

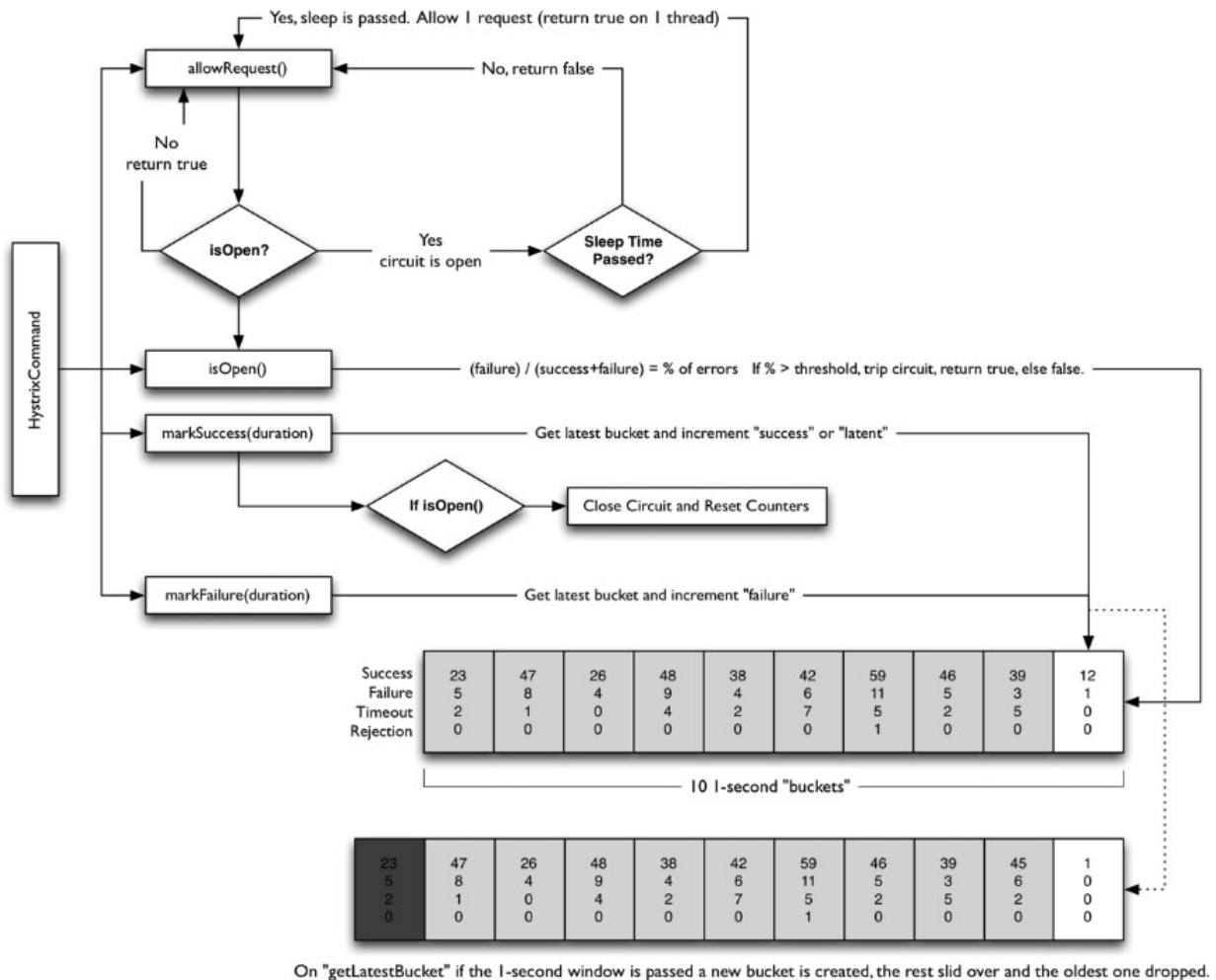
熔断关闭

熔断关闭后不会对服务进行熔断

熔断半开

部分请求根据规则调用当前服务,如果请求成功且符合规则则认为当前服务恢复正常,关闭熔断

3.7.4.3. 官网断路器流程图



3.7.4.3.1. 官网步骤

The precise way that the circuit opening and closing occurs is as follows:

1. Assuming the volume across a circuit meets a certain threshold
(`HystrixCommandProperties.circuitBreakerRequestVolumeThreshold()`)...
2. And assuming that the error percentage exceeds the threshold error percentage
(`HystrixCommandProperties.circuitBreakerErrorThresholdPercentage()`)...
3. Then the circuit-breaker transitions from **CLOSED** to **OPEN** .
4. While it is open, it short-circuits all requests made against that circuit-breaker.
5. After some amount of time
(`HystrixCommandProperties.circuitBreakerSleepWindowInMilliseconds()`), the next single request is let through (this is the **HALF-OPEN** state). If the request fails, the circuit-breaker returns to the **OPEN** state for the duration of the sleep window. If the request succeeds, the circuit-breaker transitions to **CLOSED** and the logic in 1. takes over again.

3.7.4.3.2. 断路器在什么情况下开始起作用

```

//////////////////////////////////// 服务熔断 //////////////////////////////////////
@HystrixCommand(fallbackMethod = "paymentCircuitBreaker_fallback", commandProperties = {
    @HystrixProperty(name = "circuitBreaker.enabled", value = "true"),
    @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "10"),
    @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "10000"),
    @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "60")
})
public String paymentCircuitBreaker(@PathVariable("id") Integer id) {
    if (id < 0) {
        throw new RuntimeException("*****id 不能为负数");
    }
    String serialNumber = IdUtil.simpleUUID();
    return Thread.currentThread().getName() + "\t调用成功, 流水号: " + serialNumber;
}

public String paymentCircuitBreaker_fallback(@PathVariable("id") Integer id) {
    return "id 不能为负数, 请稍后再试..., id: " + id;
}

```

涉及到断路器的三个重要参数：快照时间窗、请求总数阈值、错误百分比阈值。

- 1：快照时间窗：断路器确定是否打开需要统计些请求和错误数据，而统计的时间范围就是快照时间窗，默认为最近的10秒。
- 2：请求总数阈值：在快照时间窗内，必须满足请求总数阈值才有资格熔断。默认为20，意味着在10秒内，如果该 hystrix命令的调用次数不足20次，即使所有的请求都超时或其他原因失败，断路器都不会打开
- 3：错误百分比阈值：当请求总数在快照时间窗内超过了阈值，比如发生了30次调用，如果在这30次调用中，有15次发生了超时异常，也就是超过50%的错误百分比，在默认设定50%阈值情况下，这时候就会将断路器打开。

3.7.4.3.3. 断路器开启或者关闭的条件

1. 当满足一定的阈值的时候(默认10秒钟超过20个请求次数)
2. 当失败率达到一定的时候(默认10秒内超过50%的请求次数)
3. 到达以上阈值,断路器将会开启
4. 当开启的时候,所有请求都不会进行转发
5. 一段时间之后(默认5秒),这个时候断路器是半开状态,会让其他一个请求进行转发. 如果成功,断路器会关闭,若失败,继续开启.重复4和5

3.7.4.3.4. 断路器打开之后

- 1: 再有请求调用的时候，将不会调用主逻辑，而是直接调用降级 fallback。通过断路器，实现了自动地发现错误并将降级逻辑切换为主逻辑，减少响应延迟的效果
- 2: 原来的主逻辑要如何恢复呢？

对于这一问题，hystrix也为我们实现了自动恢复功能

当断路器打开，对主逻辑进行熔断之后，hystrix会启动一个休眠时间窗，在这个时间窗内，降级逻辑是临时的成为主逻辑，当休眠时间窗到期，断路器将进入半开状态，释放一次请求到原来的

主逻辑上，如果此次请求正常返回，那么断路器将继续闭合主逻辑恢复，如果这次请求依然有问题，断路器继续进入打开状态，休眠时间窗重新计时。

3.7.4.3.5. 所有配置

```
//=====ALL
@HystrixCommand(fallbackMethod = "str_fallbackMethod",
    groupKey = "strGroupCommand",
    commandKey = "strCommand",
    threadPoolKey = "strThreadPool",

    commandProperties = {
        // 设置隔离策略，THREAD 表示线程池 SEMAPHORE：信号池隔离
        @HystrixProperty(name = "execution.isolation.strategy", value = "THREAD"),
        // 当隔离策略选择信号池隔离的时候，用来设置信号池的大小（最大并发数）
        @HystrixProperty(name = "execution.isolation.semaphore.maxConcurrentRequests", value = "10"),
        // 配置命令执行的超时时间
        @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "10"),
        // 是否启用超时时间
        @HystrixProperty(name = "execution.timeout.enabled", value = "true"),
        // 执行超时的时候是否中断
        @HystrixProperty(name = "execution.isolation.thread.interruptOnTimeout", value = "true"),
        // 执行被取消的时候是否中断
        @HystrixProperty(name = "execution.isolation.thread.interruptOnCancel", value = "true"),
        // 允许回调方法执行的最大并发数
        @HystrixProperty(name = "fallback.isolation.semaphore.maxConcurrentRequests", value = "10"),
        // 服务降级是否启用，是否执行回调函数
        @HystrixProperty(name = "fallback.enabled", value = "true"),

        // 当隔离策略选择信号池隔离的时候，用来设置信号池的大小（最大并发数）
        @HystrixProperty(name = "execution.isolation.semaphore.maxConcurrentRequests", value = "10"),
        // 配置命令执行的超时时间
        @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "10"),
        // 是否启用超时时间
        @HystrixProperty(name = "execution.timeout.enabled", value = "true"),
        // 执行超时的时候是否中断
        @HystrixProperty(name = "execution.isolation.thread.interruptOnTimeout", value = "true"),
        // 执行被取消的时候是否中断
        @HystrixProperty(name = "execution.isolation.thread.interruptOnCancel", value = "true"),
        // 允许回调方法执行的最大并发数
        @HystrixProperty(name = "fallback.isolation.semaphore.maxConcurrentRequests", value = "10"),
        // 服务降级是否启用，是否执行回调函数
        @HystrixProperty(name = "fallback.enabled", value = "true"),
        // 是否启用断路器
        @HystrixProperty(name = "circuitBreaker.enabled", value = "true"),
        // 该属性用来设置在滚动时间窗中，断路器熔断的最小请求数。例如，默认该值为 20 的时候，
        // 如果滚动时间窗（默认10秒）内仅收到了19个请求，即使这19个请求都失败了，断路器也不会打开。
        @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "20"),
        // 该属性用来设置在滚动时间窗中，表示在滚动时间窗中，在请求数量超过
        // circuitBreaker.requestVolumeThreshold 的情况下，如果错误请求数的百分比超过50，
        // 就把断路器设置为“打开”状态，否则就设置为“关闭”状态。

        // 该属性用来设置当断路器打开之后的休眠时间窗。休眠时间窗结束之后，
        // 会将断路器置为“半开”状态，尝试熔断的请求命令，如果依然失败就将断路器继续设置为“打开”状态，
        // 如果成功就设置为“关闭”状态。
        @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "5000"),
        // 断路器强制打开
        @HystrixProperty(name = "circuitBreaker.forceOpen", value = "false"),
        // 断路器强制关闭
        @HystrixProperty(name = "circuitBreaker.forceClosed", value = "false"),
        // 滚动时间窗设置，该时间用于断路器判断健康度时需要收集信息的持续时间
        @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "10000"),
        // 该属性用来设置滚动时间窗统计指标信息时划分“桶”的数量，断路器在收集指标信息的时候会根据
        // 设置的时间窗长度拆分成多个“桶”来累计各度量值，每个“桶”记录了一段时间内的采集指标。
        // 比如 10 秒内拆分成 10 个“桶”收集这样，所以 timeInMilliseconds 必须能被 numBuckets 整除。否则会抛异常
        @HystrixProperty(name = "metrics.rollingStats.numBuckets", value = "10"),
        // 该属性用来设置对命令执行的延迟是否使用百分位数来跟踪和计算。如果设置为 false，那么所有的概要统计都将返回 -1。
        @HystrixProperty(name = "metrics.rollingPercentile.enabled", value = "false"),
        // 该属性用来设置百分位统计的滚动窗口的持续时间，单位为毫秒。
        @HystrixProperty(name = "metrics.rollingPercentile.timeInMilliseconds", value = "60000"),
        // 该属性用来设置百分位统计滚动窗口中使用“桶”的数量。
        @HystrixProperty(name = "metrics.rollingPercentile.numBuckets", value = "60000"),
        // 该属性用来设置在执行过程中每个“桶”中保留的最大执行次数。如果在滚动时间窗内发生超过该设定值的执行次数，
        // 就从最初的位置开始重写。例如，将该值设置为100，滚动窗口为10秒，若在10秒内一个“桶”中发生了500次执行，
```

```

    @HystrixProperty(name = "metrics.rollingPercentile.bucketSize", value = "100"),
    // 该属性用来设置采集影响断路器状态的健康快照（请求的成功、错误百分比）的间隔等待时间。
    @HystrixProperty(name = "metrics.healthSnapshot.intervalInMilliseconds", value = "500"),
    // 是否开启请求缓存
    @HystrixProperty(name = "requestCache.enabled", value = "true"),
    // HystrixCommand的执行和事件是否打印日志到 HystrixRequestLog 中
    @HystrixProperty(name = "requestLog.enabled", value = "true"),
},
threadPoolProperties = {
    // 该参数用来设置执行命令线程池的核心线程数，该值也就是命令执行的最大并发量
    @HystrixProperty(name = "coreSize", value = "10"),
    // 该参数用来设置线程池的最大队列大小。当设置为 -1 时，线程池将使用 SynchronousQueue 实现的队列，
    // 否则将使用 LinkedBlockingQueue 实现的队列。
    @HystrixProperty(name = "maxQueueSize", value = "-1"),
    // 该参数用来为队列设置拒绝阈值。通过该参数，即使队列没有达到最大值也能拒绝请求。
    // 该参数主要是对 LinkedBlockingQueue 队列的补充，因为 LinkedBlockingQueue
    // 队列不能动态修改它的对象大小，而通过该属性就可以调整拒绝请求的队列大小了。
    @HystrixProperty(name = "queueSizeRejectionThreshold", value = "5"),
}
}
}

public String strConsumer() {
    return "hello 2020";
}
}

```

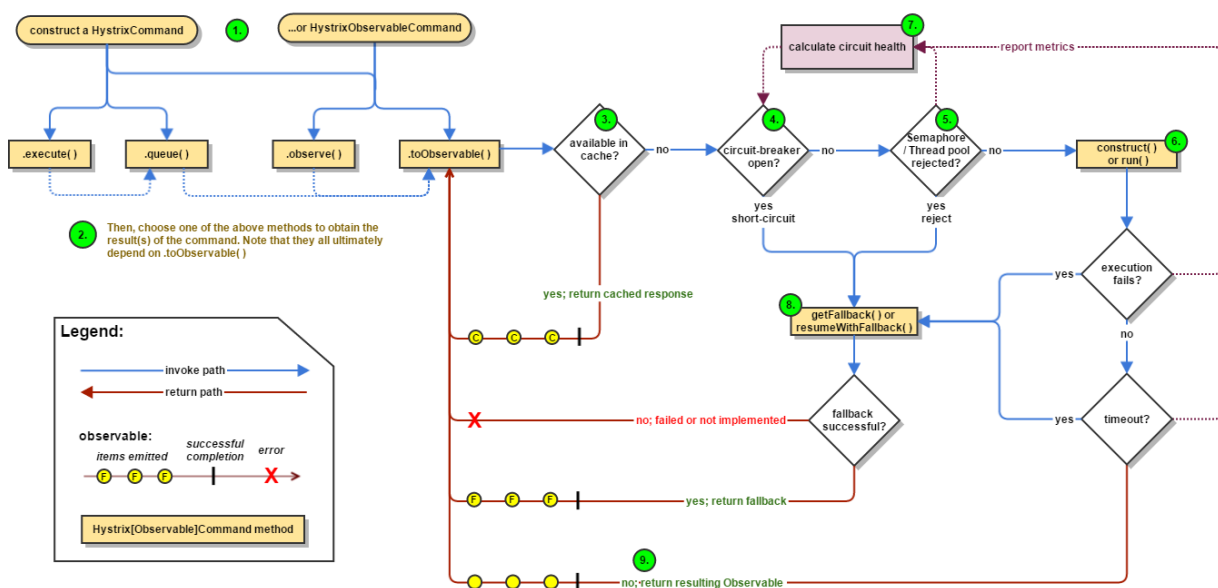
3.8. 服务限流

后面高级篇讲解alibaba的Sentinel说明

4. Hystrix工作流程

官网说明: <https://github.com/Netflix/Hystrix/wiki/How-it-Works>

官网图例:



步骤说明:

1. Construct a `HystrixCommand` or `HystrixObservableCommand` Object
2. Execute the Command
3. Is the Response Cached?
4. Is the Circuit Open?
5. Is the Thread Pool/Queue/Semaphore Full?
6. `HystrixObservableCommand.construct()` or `HystrixCommand.run()`
7. Calculate Circuit Health
8. Get the Fallback
9. Return the Successful Response

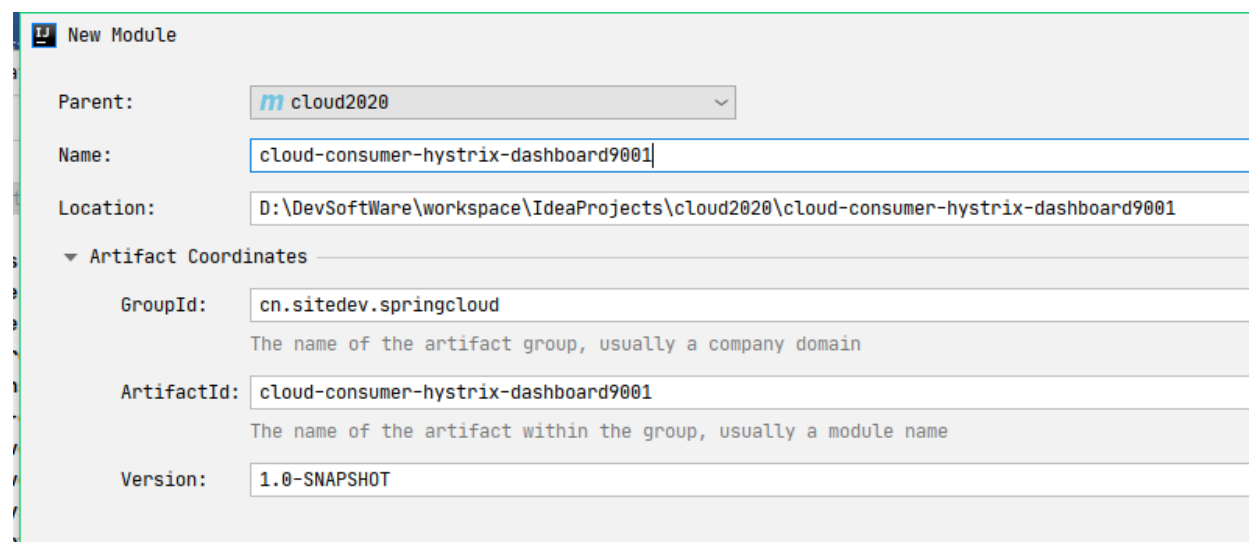
5. 服务监控HystrixDashboard

5.1. 概述

除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。Netflix通过hystrix-metrics-event-stream项目实现了对以上指标的监控。Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面。

5.2. 仪表盘9001

新建cloud-consumer-hystrix-dashboard9001



New Module

Parent: `cloud2020`

Name: `cloud-consumer-hystrix-dashboard9001`

Location: `D:\DevSoftWare\workspace\IdeaProjects\cloud2020\cloud-consumer-hystrix-dashboard9001`

▼ Artifact Coordinates

GroupId: `cn.sitedev.springcloud`
The name of the artifact group, usually a company domain

ArtifactId: `cloud-consumer-hystrix-dashboard9001`
The name of the artifact within the group, usually a module name

Version: `1.0-SNAPSHOT`

5.2.1. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.c
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>cloud-consumer-hystrix-dashboard9001</artifactId>
13    <description>hystrix监控</description>
14
15    <dependencies>
16        <!--hystrix dashboard-->
17        <dependency>
18            <groupId>org.springframework.cloud</groupId>
19            <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
20        </dependency>
21        <!--监控-->
22        <dependency>
23            <groupId>org.springframework.boot</groupId>
24            <artifactId>spring-boot-starter-actuator</artifactId>
25        </dependency>
26        <!--热部署-->
27        <dependency>
28            <groupId>org.springframework.boot</groupId>
29            <artifactId>spring-boot-devtools</artifactId>
30            <scope>runtime</scope>
31            <optional>true</optional>
32        </dependency>
33        <dependency>
34            <groupId>org.projectlombok</groupId>
35            <artifactId>lombok</artifactId>
36            <optional>true</optional>
37        </dependency>
38        <dependency>
39            <groupId>org.springframework.boot</groupId>
40            <artifactId>spring-boot-starter-test</artifactId>
41            <scope>test</scope>
42        </dependency>
43    </dependencies>
44
45
```

```
46 </project>
```

5.2.2. YML

```
1 server:
2   port: 9001
```

5.2.3. HystrixDashboardMain9001+新注解 @EnableHystrixDashboard

```
1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;
6
7 @SpringBootApplication
8 @EnableHystrixDashboard
9 public class HystrixDashboardMain9001 {
10     public static void main(String[] args) {
11         SpringApplication.run(HystrixDashboardMain9001.class, args);
12     }
13 }
```

5.2.4. 所有Provider微服务提供类(8001/8002/8003)都需要监控依赖部署

给服务8001/8002/8003 的pom.xml文件中添加依赖

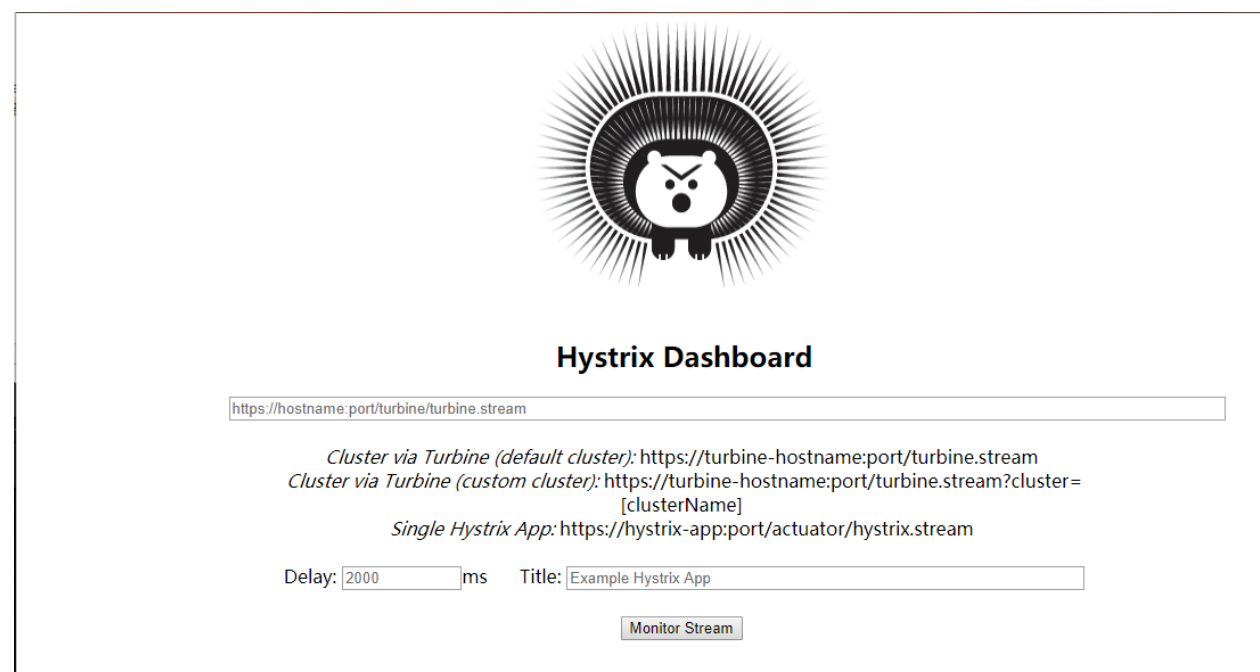
```
1 <!--监控-->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-actuator</artifactId>
5 </dependency>
```

5.2.5. 启动cloud-consumer-hystrix-dashboard9001该微服务后续将监控微服务8001

启动dashboard9001, 查看控制台输出:

services		Console	Endpoints
Spring Boot			
Running			
HystrxDashboardMain9001 [devtools] :9001/		2020-06-28 03:47:33.490 WARN 18864 --- [restartedMain] o.s.c.n.a.ArchaiausA	
Finished		2020-06-28 03:47:33.491 WARN 18864 --- [restartedMain] c.n.c.sources.URLCor	
EurekaMain7001 [devtools]		2020-06-28 03:47:33.492 INFO 18864 --- [restartedMain] c.n.c.sources.URLCor	
PaymentHystrixMain8001 [devtools]		2020-06-28 03:47:33.722 INFO 18864 --- [restartedMain] o.s.s.concurrent.Thr	
OrderHystrixMain80 [devtools]		2020-06-28 03:47:34.175 INFO 18864 --- [restartedMain] o.s.b.d.a.OptionalLi	
		2020-06-28 03:47:36.729 INFO 18864 --- [restartedMain] o.s.cloud.commons.ut	
		2020-06-28 03:47:39.443 INFO 18864 --- [restartedMain] o.s.cloud.commons.ut	
		2020-06-28 03:47:39.474 INFO 18864 --- [restartedMain] o.s.b.a.e.web.Endpoi	
		2020-06-28 03:47:39.592 INFO 18864 --- [restartedMain] o.s.b.w.embedded.tom	
		2020-06-28 03:47:42.006 INFO 18864 --- [restartedMain] o.s.cloud.commons.ut	
		2020-06-28 03:47:42.008 INFO 18864 --- [restartedMain] c.s.s.HystrixDashboa	
		2020-06-28 03:47:43.008 INFO 18864 --- [(4)-192.168.5.1] o.a.c.c.C.[Tomcat].	
		2020-06-28 03:47:43.008 INFO 18864 --- [(4)-192.168.5.1] o.s.web.servlet.Disp	

浏览器访问<http://localhost:9001/hystrix>



5.3. 断路器演示(服务监控hystrixDashboard)

5.3.1. 修改cloud-provider-hystrix-payment8001

注意:新版本Hystrix需要在主启动PaymentHystrixMain8001中指定监控路径

否则会报错: Unable to connect to Command Metric Stream.

修改后的主启动类代码如下:

```
1 package cn.sitedev.springcloud;
2
3 import com.netflix.hystrix.contrib.metrics.eventstream.HystrixMetricsStreamServlet;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```

6 import org.springframework.boot.web.servlet.ServletRegistrationBean;
7 import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
8 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
9 import org.springframework.context.annotation.Bean;
10
11 @SpringBootApplication
12 @EnableEurekaClient
13 @EnableCircuitBreaker
14 public class PaymentHystrixMain8001 {
15     public static void main(String[] args) {
16         SpringApplication.run(PaymentHystrixMain8001.class, args);
17     }
18
19     /**
20      * 此配置是为了服务监控而配置，与服务容错本身无关，springCloud 升级之后的坑
21      * ServletRegistrationBean因为springboot的默认路径不是/hystrix.stream
22      * 只要在自己的项目中配置上下面的servlet即可
23      *
24      * @return
25      */
26     @Bean
27     public ServletRegistrationBean getServlet() {
28         HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();
29         ServletRegistrationBean<HystrixMetricsStreamServlet> registrationBean =
30             new ServletRegistrationBean<>(streamServlet);
31         registrationBean.setLoadOnStartup(1);
32         registrationBean.addUrlMappings("/hystrix.stream");
33         registrationBean.setName("HystrixMetricsStreamServlet");
34         return registrationBean;
35     }
36 }

```

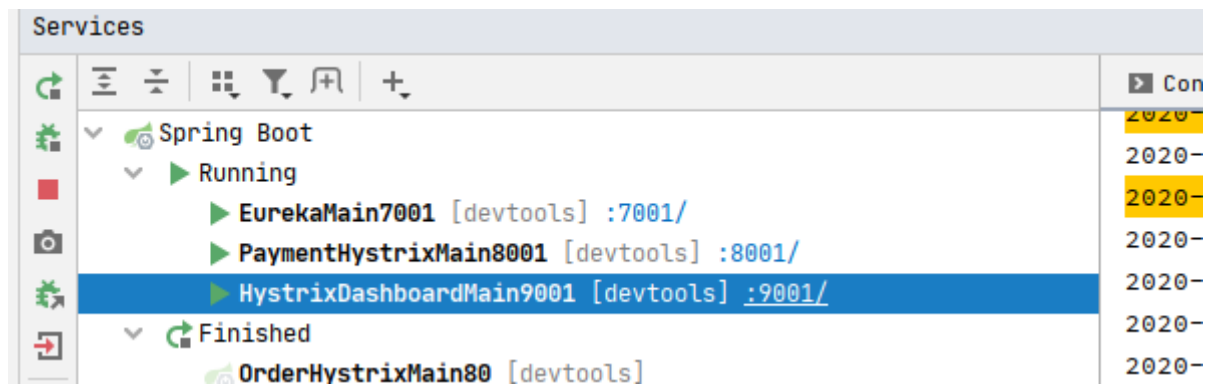
5.3.2. 监控测试

5.3.2.1. 启动一个eureka或者3个eureka集群均可

启动 HystrixDashboardMain9001

启动 EurekaMain7001

启动PaymentHystrixMain8001



5.3.2.2. 观察监控窗口

5.3.2.2.1. 9001监控8001

浏览器打开: <http://localhost:9001/hystrix>



Hystrix Dashboard

Cluster via Turbine (default cluster): <https://turbine-hostname:port/turbine.stream>
Cluster via Turbine (custom cluster): [https://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](https://turbine-hostname:port/turbine.stream?cluster=[clusterName])
Single Hystrix App: <https://hystrix-app:port/actuator/hystrix.stream>

Delay: ms Title:

在该页面上填写以下内容:

- 1 监控地址: `http://localhost:8001/hystrix.stream`
- 2 Delay: `2000`
- 3 Title: `T3`

Hystrix Dashboard

<http://localhost:8001/hystrix.stream>

Cluster via Turbine (default cluster): <https://turbine-hostname:port/turbine.stream>
Cluster via Turbine (custom cluster): [https://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](https://turbine-hostname:port/turbine.stream?cluster=[clusterName])
Single Hystrix App: <https://hystrix-app:port/actuator/hystrix.stream>

Delay: ms Title:

Monitor Stream

点击"Monitor Stream"



5.3.2.2.2. 测试地址

浏览器访问: <http://localhost:8001/payment/circuit/1>



浏览器访问: <http://localhost:8001/payment/circuit/-1>



上述测试通过

先访问正确地址,再访问错误地址,再正确地址,会发现图标断路器都是慢慢放开的.

监控结果,成功

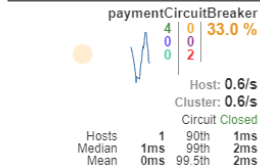
Hystrix Stream: T3



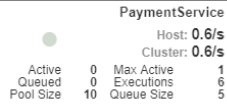
HYSTRIX
DEFEND YOUR APP

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)

[Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



Thread Pools Sort: [Alphabetical](#) | [Volume](#) |



监控结果,失败

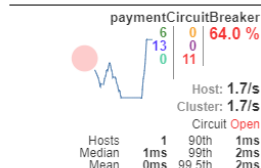
Hystrix Stream: T3



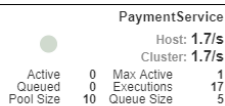
HYSTRIX
DEFEND YOUR APP

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)

[Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



Thread Pools Sort: [Alphabetical](#) | [Volume](#) |



5.3.2.2.3. 如何看监控窗口

7色:



HYSTRIX
DEFEND YOUR APP

[90](#) | [99](#) | [99.5](#)

[Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)

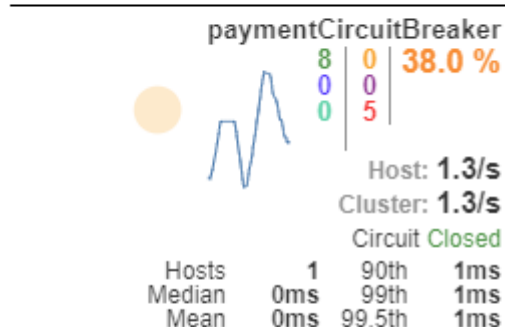
1圈:

实心圆：共有两种含义。它通过颜色的变化代表了实例的健康程度，它的健康度从绿色<黄色<橙色<红色递减。

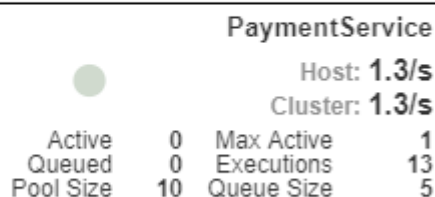
该实心圆除了颜色的变化之外，它的大小也会根据实例的请求流量发生变化，流量越大该实心圆就越大。所以通过该实心圆的展示，就可以在大量的实例中快速的发现故障实例和高压实例。

Hystrix Stream: T3

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [N](#)



Thread Pools Sort: [Alphabetical](#) | [Volume](#) |



1线:

曲线: 用来记录2分钟内流量的相对变化, 可以通过它来观察到流量的上升和下降趋势。

整图说明1:



整图说明2:

