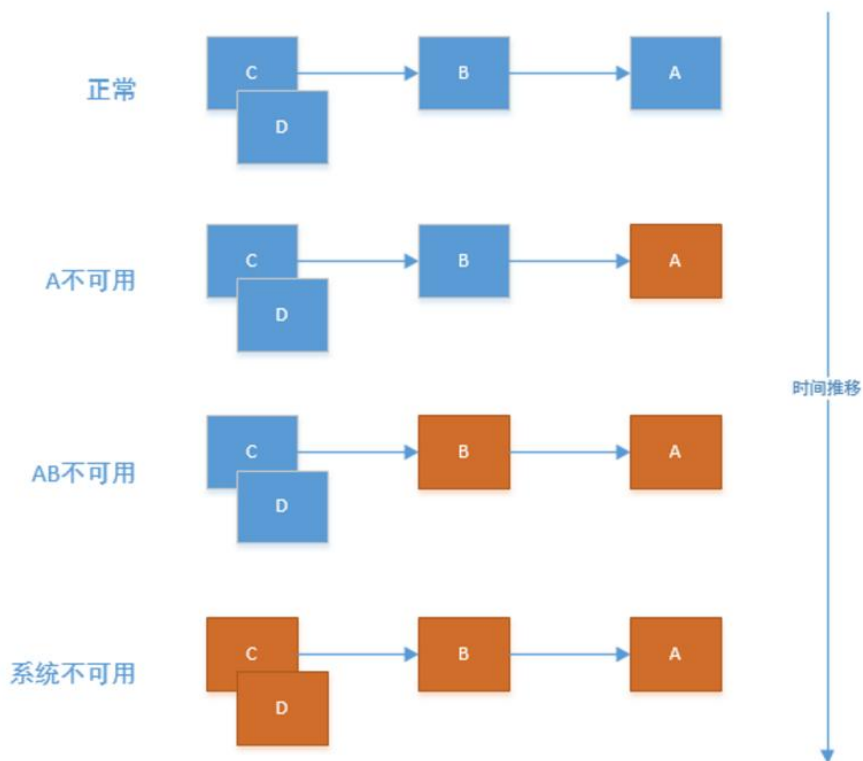


第十章 Spring Cloud熔断器

雪崩效应

- 在微服务架构中通常会有多个服务层调用，基础服务的故障可能会导致级联故障，进而造成整个系统不可用的情况，这种现象被称为服务雪崩效应。服务雪崩效应是一种因“服务提供者”的不可用导致“服务消费者”的不可用，并将不可用逐渐放大的过程。
- 如果下图所示：A作为服务提供者，B为A的服务消费者，C和D是B的服务消费者。A不可用引起了B的不可用，并将不可用像滚雪球一样放大到C和D时，雪崩效应就形成了。



示例

myshop-user是A服务， myshop-web是B服务， 当myshop-user服务暂停，调用B服务的结果：



Whitelabel Error Page

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

Fri Oct 20 15:24:53 CST 2023

There was an unexpected error (type=Internal Server Error, status=500).
connect timed out executing GET http://myshop-user/user/2

后台显示连接错误：

```
2023-10-20 15:24:53.939 ERROR 20544 --- [nio-9002-exec-3] o.a.c.c.C.[.[./].[dispatcherServlet]
```

```
java.net.SocketTimeoutException Create breakpoint : connect timed out
    at java.net.DualStackPlainSocketImpl.waitForConnect(Native Method) ~[na:1.8.0_333]
    at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:81) ~[na:1
```

熔断器-Hystrix

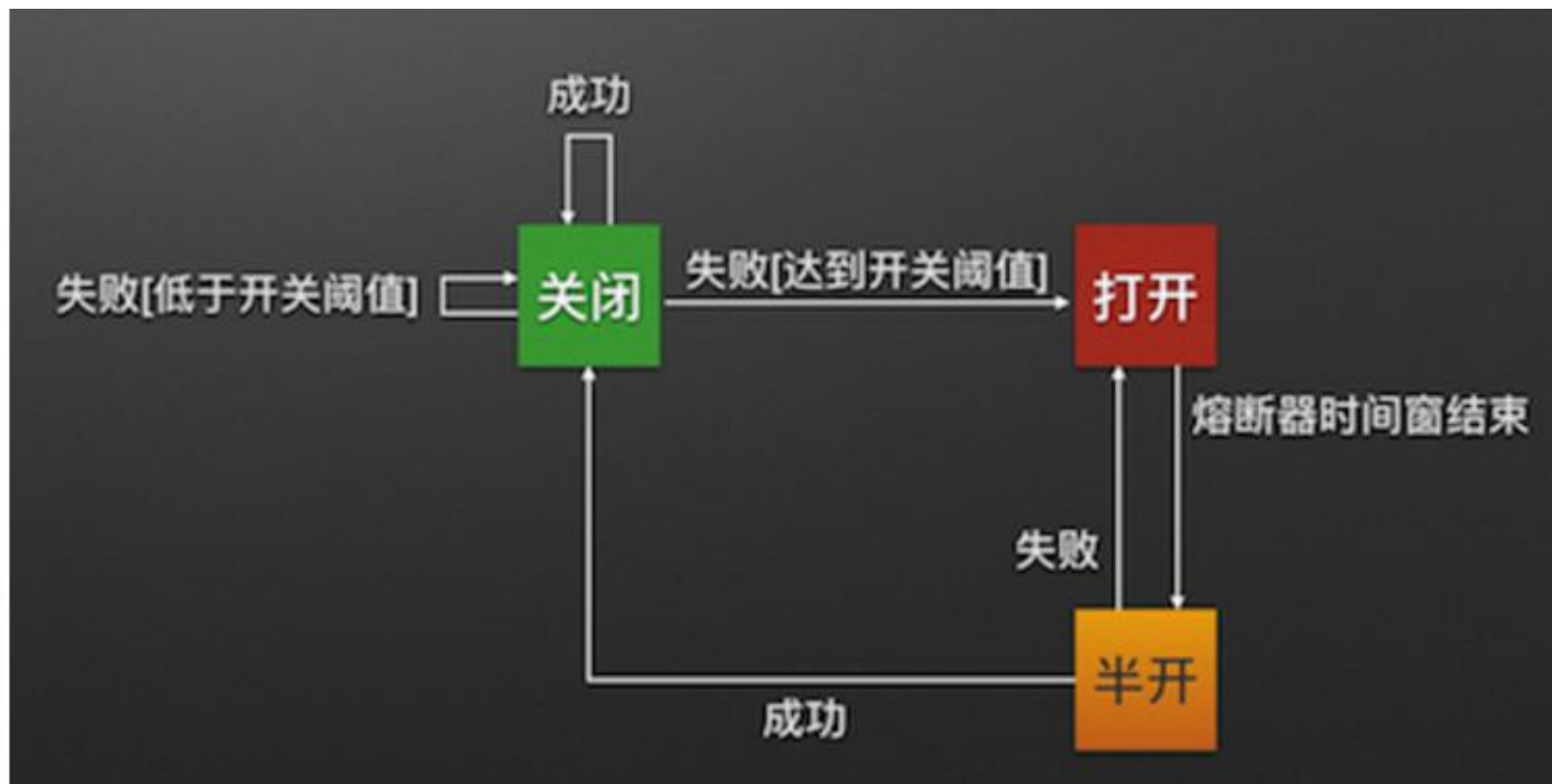
Hystrix是**Netflix**开源的一个延迟和容错库，用于隔离访问远程服务、第三方库，防止出现级联失败（防止雪崩效应）。

熔断机制的原理很简单，像家里的电路熔断器，如果电路发生短路能立刻熔断电路，避免发生灾难。在分布式系统中应用这一模式之后，服务调用方可以自己进行判断某些服务反应慢或者存在大量超时的情况时，能够主动熔断，防止整个系统被拖垮。

不同于电路熔断只能断不能自动重连，**Hystrix**可以实现弹性容错，当情况好转之后，可以自动重连。

通过断路的方式，可以将后续请求直接拒绝掉，一段时间之后允许部分请求通过,如果调用成功则回到电路闭合状态，否则继续断开。

熔断器-Hystrix



Ribbon添加Hystrix熔断器

- 两种解决方案：

1. RestTemplate+ ribbon+Hystrix

2. OpenFeign+Hystrix

- 修改购票微服务（调用方添加熔断器）

1. 导入hystrix的依赖

2. 使用@HystrixCommand注解声明fallback方法

3. 编写fallback方法逻辑

4. 在启动类添加@EnableHystrix注解

```
<dependency>
```

```
  <groupId>org.springframework.cloud</groupId>
```

```
  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
```

```
</dependency>
```

2. 使用@HystrixCommand声明fallback方法;
3. 编写fallback方法逻辑

```
@Autowired
```

```
private RestTemplate restTemplate;
```

```
@RequestMapping(value = "/order", method = RequestMethod.GET)
```

```
@HystrixCommand(fallbackMethod = "fallback")
```

```
public String order() {
```

```
    // 模拟当前用户
```

```
    Integer id = 2;
```

```
    User user = restTemplate.getForObject( url: "http://myshop-user/user/" + id,
```

```
    System.out.println(user + "==正在购票...");
```

```
    return "购票成功";
```

```
}
```

```
/**
```

```
 * 熔断回滚方法
```

```
 */
```

```
public String fallback() { return "服务暂时不可以, 发生熔断...."; }
```


4. 在启动类添加@EnableHystrix注解

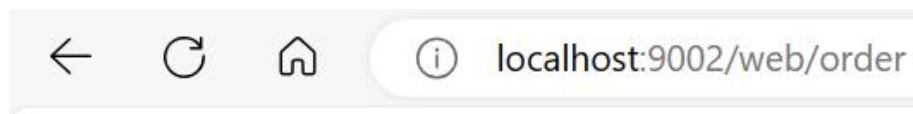
```
@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
@EnableEurekaClient    // 开启Eureka客户端自动配置
@EnableFeignClients    // 开启Feign的自动配置
@EnableHystrix        // 开启熔断功能
public class WebApplication {
    public static void main(String[] args) {
        SpringApplication.run(WebApplication.class, args);
    }
    /** 实例化RestTemplate */
    @Bean
    @LoadBalanced    // 添加Ribbon负载均衡组件
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```


测试

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

当myshop-user不可用时

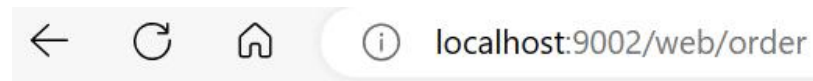


服务暂时不可以，发生熔断....

重启myshop-user后

Instances currently registered with Eureka












Application	AMIs	Availability Zones
MYSHOP-USER	n/a (1)	(1)
MYSHOP-WEB	n/a (1)	(1)



购票成功

OpenFeign开启Hystrix熔断器

- Feign默认也有对Hystix的集成：

```
▼  org.springframework.cloud:spring-cloud-starter-openfeign:2.0.0.M2
   org.springframework.cloud:spring-cloud-starter:2.0.0.M9 (omitted for duplicate)
  >  org.springframework.cloud:spring-cloud-openfeign-core:2.0.0.M2
  >  org.springframework:spring-web:5.0.5.RELEASE
  >  org.springframework.cloud:spring-cloud-commons:2.0.0.M9
   io.github.openfeign:feign-core:9.5.1
  >  io.github.openfeign:feign-slf4j:9.5.1
  >  io.github.openfeign:feign-hystrix:9.5.1
  >  io.github.openfeign:feign-java8:9.5.1
   org.springframework.cloud:spring-cloud-starter-netflix-ribbon:2.0.0.M8 (omitted for duplicate)
   org.springframework.cloud:spring-cloud-starter-netflix-archaius:2.0.0.M8 (omitted for duplicate)
```

- 默认情况下是关闭的。我们需要通过下面的参数来开启：

```
feign:
  hystrix:
    enabled: true # 开启Feign的熔断功能
```

OpenFeign开启Hystrix熔断器

编写熔断器类

```
/**
 * 熔断器类
 */
@Component
public class UserControllerImpl implements UserController
{
    @Override
    public User findById(Integer id) {
        System.out.println("执行了熔断器类...");
        return null;
    }
}
```

UserController接口上增加fallback

```
@FeignClient(value = "myshop-user", fallback = UserControllerImpl.class)
public interface UserController {
    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public User findById(@PathVariable(value = "id") Integer id);
}
```

将WebController改为使用OpenFeign的代码，当所有服务均正常时，测试

← ↻ 🏠 ⓘ localhost:9002/web/order

购票成功

```
public String order(){
    // 模拟当前用户
    Integer id = 2;
    User user = userController.findById(id);
    System.out.println(user+"==正在购票...");
    return "购票成功";
}
```

当myshop-user不可用时，浏览器显示购票成功，但控制台显示执行了熔断器。

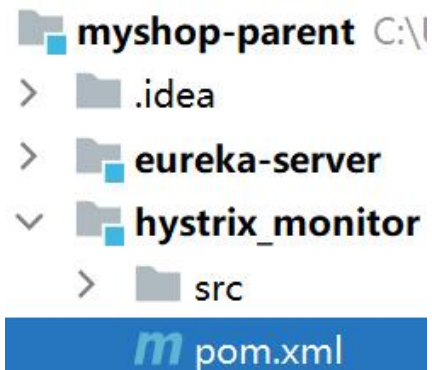
← ↻ 🏠 ⓘ localhost:9002/web/order

购票成功

	控制台	Actuator
Spring Boot	2023-10-20 23:13:46.749	
正在运行	2023-10-20 23:13:46.749	
EurekaApplication	2023-10-20 23:13:46.752	
WebApplication	执行了熔断器类...	
	null==正在购票...	
已完成	执行了熔断器类...	
UserApplication	null==正在购票...	
	执行了熔断器类...	
	null==正在购票...	

Hystrix Dashboard监控面板

- 一、搭建Hystrix Dashboard工程
 - 导入hystrix dashboard的依赖



```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
  </dependency>
</dependencies>
```


Hystrix Dashboard监控面板

- 编写application.yml配置文件

```
server:
  port: 7778
spring:
  application:
    name: hystrix-monitor
```

- 启动类添加@EnableHystrixDashboard注解

```
/**
 * Hystrix Dashboard启动类
 */
@SpringBootApplication
@EnableHystrixDashboard // 开启Hystrix监控面板
public class HystrixApplication {

    public static void main(String[] args) {

        SpringApplication.run(HystrixApplication.class, args);
    }
}
```

- 启动Hystrix Dashboard监控面板微服务



Hystrix Dashboard

Cluster via Turbine (default cluster): `http://turbine-hostname:port/turbine.stream`

Cluster via Turbine (custom cluster): `http://turbine-hostname:port/turbine.stream?cluster=[clusterName]`

Single Hystrix App: `http://hystrix-app:port/hystrix.stream`

Delay: ms Title:

Hystrix Dashboard监控面板

- 二、在服务调用方的启动类，添加监听方法的Servlet
- 启动类添加ServletRegistrationBean

@Bean

```
public ServletRegistrationBean getServlet() {  
    HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();  
    ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);  
    registrationBean.setLoadOnStartup(1);  
    registrationBean.addUrlMappings("/hystrix.stream");  
    registrationBean.setName("HystrixMetricsStreamServlet");  
    return registrationBean;  
}
```

Hystrix Dashboard

http://localhost:9002/hystrix.stream

启动服务，
访问hystrix
dashboard

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

Delay: ms Title:

Example Hystrix App

Monitor Stream

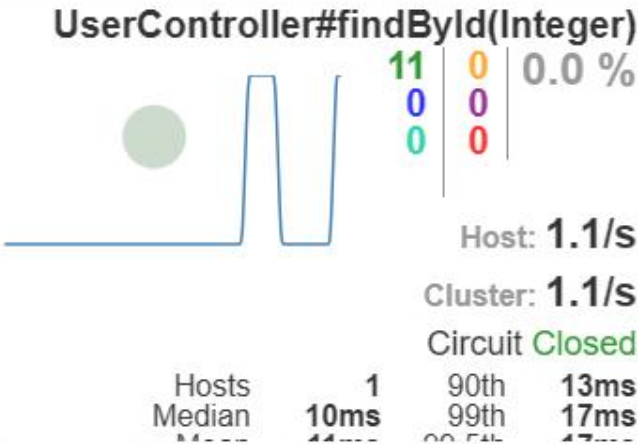
Hystrix Stream: <http://localhost:9002/hystrix.stream>



Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)
[Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)

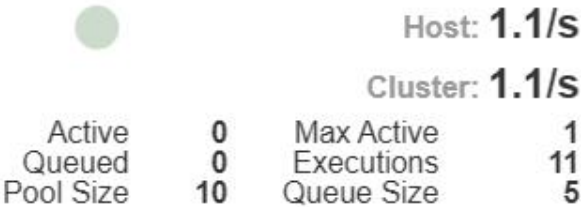
Loading ...

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



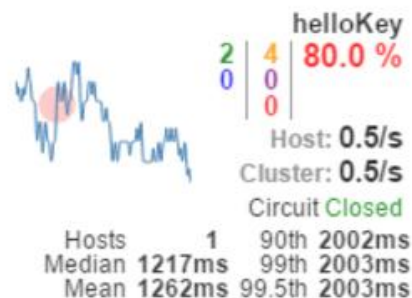
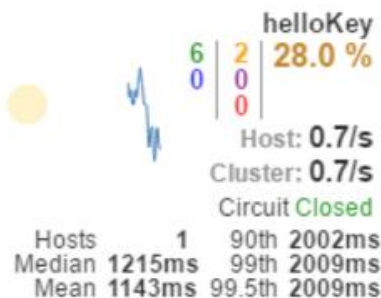
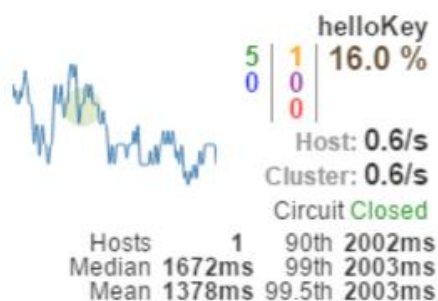
Thread Pools Sort: [Alphabetical](#) | [Volume](#) |

myshop-user



Hystrix Dashboard监控面板

- 我们可以在监控信息的左上部分找到两个重要的图形信息：一个实心圆和一条曲线。
 - 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，如下图所示，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，我们就可以在大量的实例中快速的发现故障实例和高压力实例。



Hystrix Dashboard监控面板

- **曲线**：用来记录2分钟内流量的相对变化，我们可以通过它来观察到流量的上升和下降趋势。
- 其他一些数量指标如下图所示：

