**史上最简单的SpringCloud教程 | 第四篇:断路器（Hystrix）**

2017年04月09日 21:14:05

阅读数：278528

转载请标明出处：   
<http://blog.csdn.net/forezp/article/details/69934399>   
本文出自[方志朋的博客](http://blog.csdn.net/forezp)

* 最新版本：
* [史上最简单的SpringCloud教程 | 第四篇:断路器（Hystrix）(Finchley版本)](http://blog.csdn.net/forezp/article/details/81040990)

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

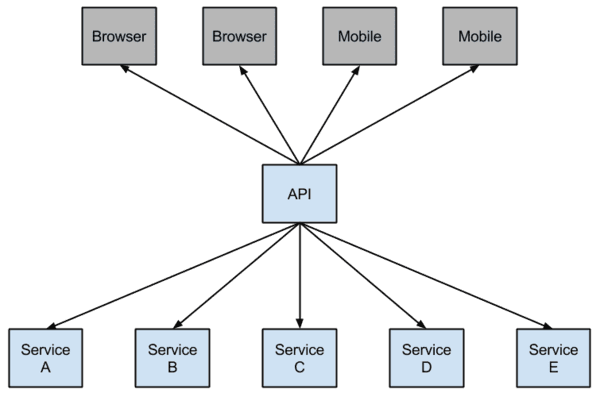
为了解决这个问题，业界提出了断路器模型。

**一、断路器简介**

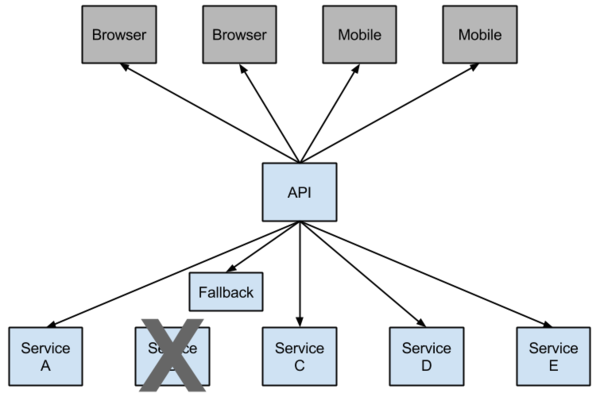
Netflix has created a library called Hystrix that implements the circuit breaker pattern. In a microservice architecture it is common to have multiple layers of service calls.

. —-摘自官网

Netflix开源了Hystrix组件，实现了断路器模式，SpringCloud对这一组件进行了整合。 在微服务架构中，一个请求需要调用多个服务是非常常见的，如下图：



较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystric 是5秒20次） 断路器将会被打开。



断路打开后，可用避免连锁故障，fallback方法可以直接返回一个固定值。

**二、准备工作**

这篇文章基于上一篇文章的工程，首先启动上一篇文章的工程，启动eureka-server 工程；启动service-hi工程，它的端口为8762。

**三、在ribbon使用断路器**

改造serice-ribbon 工程的代码，首先在pox.xml文件中加入spring-cloud-starter-hystrix的起步依赖：

<dependency>

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

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

</dependency>

* 1
* 2
* 3
* 4

在程序的启动类ServiceRibbonApplication 加@EnableHystrix注解开启Hystrix：

@SpringBootApplication

@EnableDiscoveryClient

@EnableHystrix

public class ServiceRibbonApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRibbonApplication.class, args);

}

@Bean

@LoadBalanced

RestTemplate restTemplate() {

return new RestTemplate();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

改造HelloService类，在hiService方法上加上@HystrixCommand注解。该注解对该方法创建了熔断器的功能，并指定了fallbackMethod熔断方法，熔断方法直接返回了一个字符串，字符串为”hi,”+name+”,sorry,error!”，代码如下：

@Service

public class HelloService {

@Autowired

RestTemplate restTemplate;

@HystrixCommand(fallbackMethod = "hiError")

public String hiService(String name) {

return restTemplate.getForObject("http://SERVICE-HI/hi?name="+name,String.class);

}

public String hiError(String name) {

return "hi,"+name+",sorry,error!";

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

启动：service-ribbon 工程，当我们访问<http://localhost:8764/hi?name=forezp>,浏览器显示：

hi forezp,i am from port:8762

此时关闭 service-hi 工程，当我们再访问<http://localhost:8764/hi?name=forezp>，浏览器会显示：

hi ,forezp,orry,error!

这就说明当 service-hi 工程不可用的时候，service-ribbon调用 service-hi的API接口时，会执行快速失败，直接返回一组字符串，而不是等待响应超时，这很好的控制了容器的线程阻塞。

**四、Feign中使用断路器**

Feign是自带断路器的，在D版本的Spring Cloud中，它没有默认打开。需要在配置文件中配置打开它，在配置文件加以下代码：

feign.hystrix.enabled=true

基于service-feign工程进行改造，只需要在FeignClient的SchedualServiceHi接口的注解中加上fallback的指定类就行了：

@FeignClient(value = "service-hi",fallback = SchedualServiceHiHystric.class)

public interface SchedualServiceHi {

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

String sayHiFromClientOne(@RequestParam(value = "name") String name);

}

* 1
* 2
* 3
* 4
* 5
* 6

SchedualServiceHiHystric需要实现SchedualServiceHi 接口，并注入到Ioc容器中，代码如下：

@Component

public class SchedualServiceHiHystric implements SchedualServiceHi {

@Override

public String sayHiFromClientOne(String name) {

return "sorry "+name;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

启动四servcie-feign工程，浏览器打开<http://localhost:8765/hi?name=forezp>,注意此时service-hi工程没有启动，网页显示：

sorry forezp

打开service-hi工程，再次访问，浏览器显示：

>

hi forezp,i am from port:8762

这证明断路器起到作用了。

五、Hystrix Dashboard (断路器：Hystrix 仪表盘)

基于service-ribbon 改造，Feign的改造和这一样。

首选在pom.xml引入spring-cloud-starter-hystrix-dashboard的起步依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>

</dependency>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

在主程序启动类中加入@EnableHystrixDashboard注解，开启hystrixDashboard：

@SpringBootApplication

@EnableDiscoveryClient

@EnableHystrix

@EnableHystrixDashboard

public class ServiceRibbonApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRibbonApplication.class, args);

}

@Bean

@LoadBalanced

RestTemplate restTemplate() {

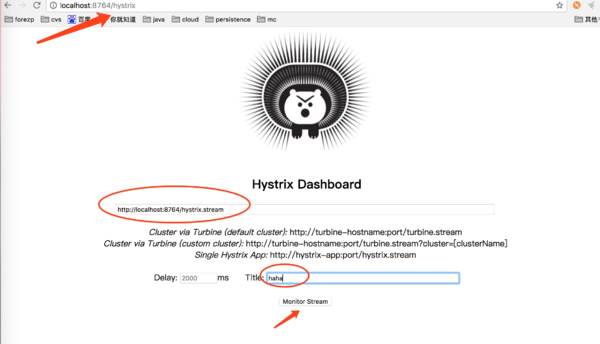
return new RestTemplate();

}

}

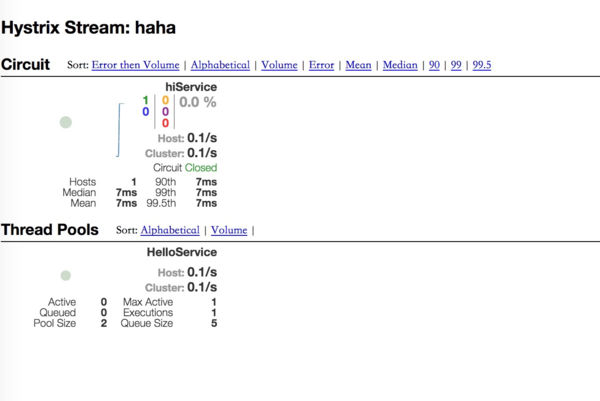
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

打开浏览器：访问<http://localhost:8764/hystrix>,界面如下：



点击monitor stream，进入下一个界面，访问：<http://localhost:8764/hi?name=forezp>

此时会出现监控界面：



本文源码下载：   
<https://github.com/forezp/SpringCloudLearning/tree/master/chapter4>

**六、参考资料**

[circuit\_breaker\_hystrix](http://projects.spring.io/spring-cloud/spring-cloud.html#_circuit_breaker_hystrix_clients)

[feign-hystrix](http://projects.spring.io/spring-cloud/spring-cloud.html#spring-cloud-feign-hystrix)

[hystrix\_dashboard](http://projects.spring.io/spring-cloud/spring-cloud.html#_circuit_breaker_hystrix_dashboard)