**Spring Cloud构建微服务架构（三）断路器**

在微服务架构中，我们将系统拆分成了一个个的服务单元，各单元间通过服务注册与订阅的方式互相依赖。由于每个单元都在不同的进程中运行，依赖通过远程调用的方式执行，这样就有可能因为网络原因或是依赖服务自身问题出现调用故障或延迟，而这些问题会直接导致调用方的对外服务也出现延迟，若此时调用方的请求不断增加，最后就会出现因等待出现故障的依赖方响应而形成任务积压，最终导致自身服务的瘫痪。

举个例子，在一个电商网站中，我们可能会将系统拆分成，用户、订单、库存、积分、评论等一系列的服务单元。用户创建一个订单的时候，在调用订单服务创建订单的时候，会向库存服务来请求出货（判断是否有足够库存来出货）。此时若库存服务因网络原因无法被访问到，导致创建订单服务的线程进入等待库存申请服务的响应，在漫长的等待之后用户会因为请求库存失败而得到创建订单失败的结果。如果在高并发情况之下，因这些等待线程在等待库存服务的响应而未能释放，使得后续到来的创建订单请求被阻塞，最终导致订单服务也不可用。

在微服务架构中，存在着那么多的服务单元，若一个单元出现故障，就会因依赖关系形成故障蔓延，最终导致整个系统的瘫痪，这样的架构相较传统架构就更加的不稳定。为了解决这样的问题，因此产生了断路器模式。

**什么是断路器**

断路器模式源于Martin Fowler的[Circuit Breaker](http://martinfowler.com/bliki/CircuitBreaker.html" \t "_blank)一文。“断路器”本身是一种开关装置，用于在电路上保护线路过载，当线路中有电器发生短路时，“断路器”能够及时的切断故障电路，防止发生过载、发热、甚至起火等严重后果。

在分布式架构中，断路器模式的作用也是类似的，当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

**Netflix Hystrix**

在Spring Cloud中使用了[Hystrix](https://github.com/Netflix/Hystrix" \t "_blank) 来实现断路器的功能。Hystrix是Netflix开源的微服务框架套件之一，该框架目标在于通过控制那些访问远程系统、服务和第三方库的节点，从而对延迟和故障提供更强大的容错能力。Hystrix具备拥有回退机制和断路器功能的线程和信号隔离，请求缓存和请求打包，以及监控和配置等功能。

下面我们来看看如何使用Hystrix。

**准备工作**

在开始加入断路器之前，我们先拿之前构建两个微服务为基础进行下面的操作，主要使用下面几个工程：

* + eureka-server工程：服务注册中心，端口1111
  + compute-service工程：服务单元，端口2222
  + eureka-ribbon：通过ribbon实现的服务单元，依赖compute-service的服务，端口3333
  + eureka-feign：通过feign实现的服务单元，依赖compute-service的服务，端口3333

*若您还没有使用Spring Cloud的经验，可以先阅读[《服务注册与发现》与《服务消费者》](01.Spring%20Cloud构建微服务架构（一）服务注册与发现.docx)，对Spring Cloud构建的微服务有一个初步的认识。*

**Ribbon中引入Hystrix**

* 依次启动eureka-server、compute-service、eureka-ribbon工程
* 访问<http://localhost:1111/>可以看到注册中心的状态
* 访问<http://localhost:3333/add>，调用eureka-ribbon的服务，该服务会去调用compute-service的服务，计算出10+20的值，页面显示30
* 关闭compute-service服务，访问[http://localhost:3333/add](http://localhost:3333/add" \t "_blank)，我们获得了下面的报错信息

|  |
| --- |
| Whitelabel Error Page  This application has no explicit mapping **for** /error, so you are seeing this as a fallback.  Sat Jun 25 21:16:59 CST 2016  There was an unexpected error (type=Internal Server Error, status=500).  I/O error on GET request **for** "http://COMPUTE-SERVICE/add?a=10&b=20": Connection refused: connect; nested exception is java.net.ConnectException: Connection refused: connect |

* pom.xml中引入依赖hystrix依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-hystrix</artifactId>  </dependency> |

* 在eureka-ribbon的主类RibbonApplication中使用@EnableCircuitBreaker注解开启断路器功能：

|  |
| --- |
| @SpringBootApplication  @EnableDiscoveryClient  @EnableCircuitBreaker  **public** **class** **RibbonApplication** {  @Bean  @LoadBalanced  RestTemplate **restTemplate**() {  **return** **new** RestTemplate();  }  **public** **static** **void** **main**(String[] args) {  SpringApplication.run(RibbonApplication.class, args);  }  } |

* 改造原来的服务消费方式，新增ComputeService类，在使用ribbon消费服务的函数上增加@HystrixCommand注解来指定回调方法。

|  |
| --- |
| @Service  **public** **class** **ComputeService** {  @Autowired  RestTemplate restTemplate;  @HystrixCommand(fallbackMethod = "addServiceFallback")  **public** String **addService**() {  **return** restTemplate.getForEntity("http://COMPUTE-SERVICE/add?a=10&b=20", String.class).getBody();  }  **public** String **addServiceFallback**() {  **return** "error";  }  } |

* 提供rest接口的Controller改为调用ComputeService的addService

|  |
| --- |
| @RestController  **public** **class** **ConsumerController** {  @Autowired  **private** ComputeService computeService;  @RequestMapping(value = "/add", method = RequestMethod.GET)  **public** String **add**() {  **return** computeService.addService();  }  } |

* 验证断路器的回调
  + 依次启动eureka-server、compute-service、eureka-ribbon工程
  + 访问<http://localhost:1111/>可以看到注册中心的状态
  + 访问<http://localhost:3333/add>，页面显示：30
  + 关闭compute-service服务后再访问[http://localhost:3333/add](http://localhost:3333/add" \t "_blank)，页面显示：error

更多关于Hystrix的使用可参考[How To Use](https://github.com/Netflix/Hystrix/wiki/How-To-Use" \t "_blank)

**Feign使用Hystrix**

注意这里说的是“使用”，没有错，我们不需要在Feigh工程中引入Hystix，Feign中已经依赖了Hystrix，我们可以在未做任何改造前，尝试下面你的操作：

* 依次启动eureka-server、compute-service、eureka-feign工程
* 访问<http://localhost:1111/>可以看到注册中心的状态
* 访问<http://localhost:3333/add>，调用eureka-feign的服务，该服务会去调用compute-service的服务，计算出10+20的值，页面显示30
* 关闭compute-service服务，访问[http://localhost:3333/add](http://localhost:3333/add" \t "_blank)，我们获得了下面的报错信息

|  |
| --- |
| Whitelabel Error Page  This application has no explicit mapping **for** /error, so you are seeing this as a fallback.  Sat Jun 25 22:10:05 CST 2016  There was an unexpected error (type=Internal Server Error, status=500).  add timed-out and no fallback available. |

如果您够仔细，会发现与在ribbon中的报错是不同的，看到add timed-out and no fallback available这句，或许您已经猜到什么，看看我们的控制台，可以看到报错信息来自hystrix-core-1.5.2.jar，所以在这个工程中，我们要学习的就是如何使用Feign中集成的Hystrix。

* 使用@FeignClient注解中的fallback属性指定回调类

|  |
| --- |
| @FeignClient(value = "compute-service", fallback = ComputeClientHystrix.class)  **public** **interface** **ComputeClient** {  @RequestMapping(method = RequestMethod.GET, value = "/add")  Integer **add**(@RequestParam(value = "a") Integer a, @**RequestParam**(value = "b") Integer b);  } |

* 创建回调类ComputeClientHystrix，实现@FeignClient的接口，此时实现的方法就是对应@FeignClient接口中映射的fallback函数。

|  |
| --- |
| @Component  **public** **class** **ComputeClientHystrix** **implements** **ComputeClient** {  @Override  **public** Integer **add**(@RequestParam(value = "a") Integer a, @**RequestParam**(value = "b") Integer b) {  **return** -9999;  }  } |

* 再用之前的方法验证一下，是否在compute-service服务不可用的情况下，页面返回了-9999。

关于Feign的更多使用方法可参考：[Feign](https://github.com/Netflix/feign)