Spring Cloud（五）Hystrix 断路器

作者：wlgzs-sjl
原文链接：https://ld246.com/article/1607505887218
来源网站：链滴

<h2 id="一-概述">一、概述</h2>
<h3 id="分布式系统面临的问题">分布式系统面临的问题</h3>
复杂分布式体系结构中的应用程序有数十个依赖关系，每个依赖关系在某些时候将不可避免的败！

<h3 id="服务雪崩">服务雪崩</h3>
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 调用其他的微服务，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中内饱。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，以便单个赖关系的失败，不能取消整个应用程序或系统。
<h3 id="什么是Hystrix-">什么是 Hystrix？</h3>
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” 本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类熔断保险丝），向调用方返回一个服务预期的，可处理的备选响应（FallBack），而不是长时间的等或者抛出调用方法无法处理的异常，这样就可以保证了服务调用方的线程不会被长时间，不必要的占，从而避免了故障在分布式系统中的蔓延，乃至雪崩。
<h3 id="Hystrix能干嘛-">Hystrix 能干嘛？</h3>

服务熔断
服务降级
服务限流
接近实时的监控
……

<h2 id="二-服务熔断">二、服务熔断</h2>
熔断机制是对应雪崩效应的一种微服务链路保护机制。
一般是某个服务故障或者异常引起，类似现实世界中的 “保险丝” ， 当某个异常条件被触发直接熔断整个服务，而不是一直等到此服务超时！
当扇出链路的某个微服务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 SpringCloud 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。
<h4 id="-1-新建springcloud-provider-dept-hystrix-8001">（1）新建 springcloud-provider-dept-hystrix-8001</h4>
将之前 8001 的所有东西拷贝一份，修改对应名称
<h4 id="-2-修改pom-添加Hystrix的依赖">（2）修改 pom，添加 Hystrix 的依赖</h4>
<pre><code class="highlight-chroma"><!--Hystrix-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifact</code></pre>

d>spring-cloud-starter-hystrix</artifactId>
            <versio
>1.4.6.RELEASE</version>
        </depend
ncy>
</code></pre>
<h4 id="-3-修改yml-修改eureka实例的id">（3）修改 yml，修改 eureka 实例的 id</h4>

（4）修改 DeptController
@HystrixCommand 报异常后如何处理
<pre><code class="highlight-chroma"><span class="highlight
cl">//一旦调用服务方法失败并抛出了错误信息后
    //会自动调用Hys
rixCommand标注好的fallbackMethod调用类中指定方法
    @HystrixComm
nd(fallbackMethod = "hystrixGet")
</code></pre>
<pre><code class="highlight-chroma"><span class="highlight
cl">@RestController
public class Dept
ontroller {
    @Autowired
    DeptService de
tService;

    @HystrixComm
nd(fallbackMethod = "hystrixGet")
    @GetMapping(
/{id}")
    public Dept get
@PathVariable int id){
        Dept dept=d
ptService.queryById(id);
        if (dept==nul
){
            throw new
RuntimeException(id+"id不存在");
        }
        return dept;
    }

    public Dept hys
rixGet(@PathVariable int id){
        Dept dept=n
w Dept();
        dept.setDept
d(id);
        dept.setDep
Name("该id："+id+"没有对应的信息，null--@HystrixCommand");
        dept.setDbS
urce("no this database in MySQL");
        return dept;

    }
}
</code></pre>
<h4 id="-4-修改主启动类添加新注解--EnableCircuitBreaker">（4）修改主启动类添加新注解 @EnableCircuitBreaker</h4>
<pre><code class="highlight-chroma">@SpringBootApplication
@EnableEurekaClint //本服务启动之后会自动注册进Eureka中
@EnableDiscoverClient //服务发现
@EnableCircuitBraker//添加对熔断的支持
public class DeptProviderHystrix_8001 {
    public static void main(String[] args) {
        SpringApplication.run(DeptProviderHystrix_8001.class,args);
    }
}
</code></pre>
<h4 id="-5-测试">（5）测试</h4>

启动 Eureka 集群 springcloud-eureka-7003
启动主启动类 DeptProviderHystrix_8001
启动客户端 springcloud-consumer-dept-80
访问 http://localhost/111

<h2 id="三-服务降级">三、服务降级</h2>
整体资源快不够了，忍痛将某些服务先关掉，待渡过难关，再开启回来。
所谓降级，就是一般是从整体符合考虑，就是当某个服务熔断之后，服务器将不再被调用，此客户端可以自己准备一个本地的 fallback 回调，返回一个缺省值，这样做，虽然服务水平下降，但好可用，比直接挂掉要强。
服务降级处理是在客户端实现完成的，与服务端没有关系。
<h4 id="-1-修改springcloud-api工程">（1）修改 springcloud-api 工程</h4>

根据已经有的 DeptClientService 接口新建一个实现了 FallbackFactory 接口的类 DeptClientSeviceFallbackFactory，需要加上 @Component 注解。

<pre><code class="highlight-chroma">@Component
public class DeptClientServiceFallbackFactory implements FallbackFactory<DeptClientService> {
    public DeptClietService create(Throwable throwable) {
        return new DptClientService() {
            public Dep getDeptById(int id) {
</code></pre>

```
</span></span><span class="highlight-line"><span class="highlight-cl">            Dept de
t=new Dept();
</span></span><span class="highlight-line"><span class="highlight-cl">            dept.se
DeptId(id);
</span></span><span class="highlight-line"><span class="highlight-cl">            dept.se
DeptName("该id："+id+"没有对应的信息，Consumer客户端提供 的降级信息，此刻服务Provider
经关闭");
</span></span><span class="highlight-line"><span class="highlight-cl">            dept.se
DbSource("no this database in MySQL");
</span></span><span class="highlight-line"><span class="highlight-cl">            return d
pt;
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        public List
lt;Dept&amp;gt; getAll() {
</span></span><span class="highlight-line"><span class="highlight-cl">            return n
ll;
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        public boo
ean addDept(Dept dept) {
</span></span><span class="highlight-line"><span class="highlight-cl">            return fa
se;
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    };
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<ol start="2">
<li>DeptClientService 接口在注解 @FeignClient 中添加 fallbackFactory 属性值。</li>
</ol>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">@FeignClient(value = "SPRINGCLOUD-PROVIDER-DEPT",fallbackFactory = DeptClientServ
ceFallbackFactory.class)
</span></span><span class="highlight-line"><span class="highlight-cl">public interface D
ptClientService {
</span></span></code></pre>
<h4 id="-2-springcloud-consumer-dept-feign-80工程修改yml文件">（2）springcloud-consu
er-dept-feign-80 工程修改 yml 文件</h4>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">feign:
</span></span><span class="highlight-line"><span class="highlight-cl">  hystrix:
</span></span><span class="highlight-line"><span class="highlight-cl">    enabled: true
</span></span></code></pre>
<h4 id="-3-测试">（3）测试</h4>
<ol>
<li>启动 eureka 集群 springcloud-eureka-7003</li>
<li>启动 springcloud-provider-dept-hystrix-8001</li>
<li>启动 springcloud-consumer-dept-feign-80</li>
<li>正常访问测试</li>
</ol>
<p><img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/file/2020/12/image-51d50803.png?imageView2/2/interlace/1/format/jpg"></
>
```

<ol start="5">
故意关闭 springcloud-provider-dept-hystrix-8001

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也会获得提示信息而不会挂起耗死服务器。
<h2 id="四-熔断和降级的异同点">四、熔断和降级的异同点</h2>
<h3 id="相同点-">相同点：</h3>
目的很一致，都是从可用性可靠性着想，为防止系统的整体缓慢甚至崩，采用的技术手段；
最终表现类似，对于两者来说，最终让用户体验到的是某些功能暂时不达或不可用；
粒度一般都是服务级别，当然，业界也有不少更细粒度的做法，比如做数据持久层（允许查询，不允许增删改）；
自治性要求很高，熔断模式一般都是服务基于策略的自动触发，降级虽可人工干预，但在微服务架构下，完全靠人显然不可能，开关预置、配置中心都是必要手段。
<h3 id="不同点-">不同点：</h3>
触发原因不太一样，服务熔断一般是某个服务（下游服务）故障引起，服务降级一般是从整体负荷考虑；
管理目标的层次不太一样，熔断其实是一个框架级的处理，每个微服务需要（无层级之分），而降级一般需要对业务有层级之分（比如降级一般是从最外围服务开始）；
实现方式不太一样。