链滴

Spring-Cloud 系列第 4 篇:spring-cloud-Hystrix

作者：xjtushilei

原文链接：https://ld246.com/article/1497579906780

来源网站：链滴

自学spring-cloud系列，越来越感觉spring-cloud很强大！

主要分为以下几篇：

1. spring-cloud-config: 分布式配置管理
2. spring-cloud-eureka: 服务注册与发现
3. spring-cloud-eureka-consumer: 远程服务调用和及其负载均衡
4. spring-cloud-Hystrix: 熔断器保证服务高可用
5. spring-cloud-config-eureka-ribbon: 分布式配置管理的高可用
6. spring-cloud-bus: 配置信息的实时更新
7. spring-cloud-zuul: 网关基础服务

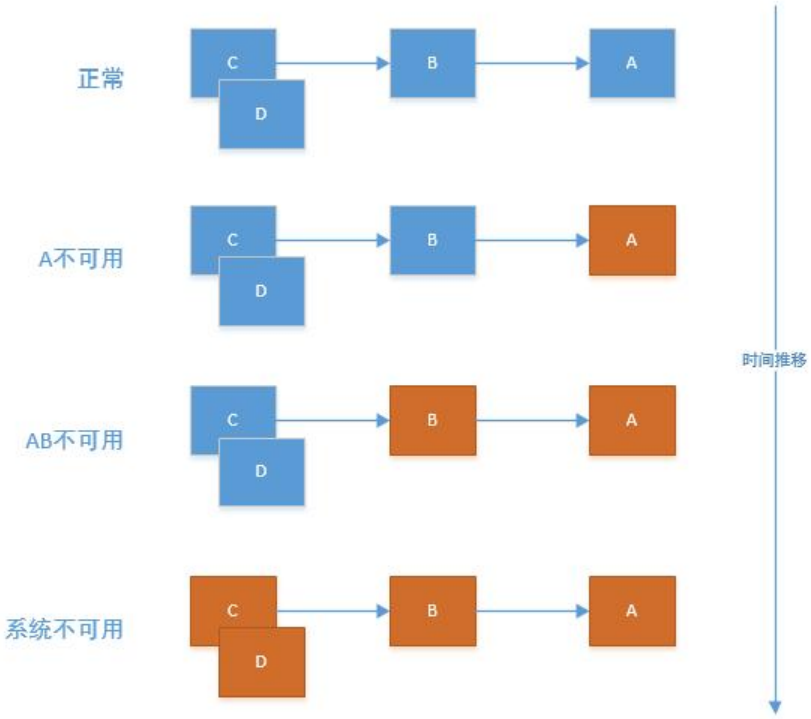
熔断器介绍

说起springcloud熔断让我想起了去年股市中的熔断，多次痛的领悟，随意实施的熔断对整个系统的响是灾难性的，好了接下来我们还是说正事。

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“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。

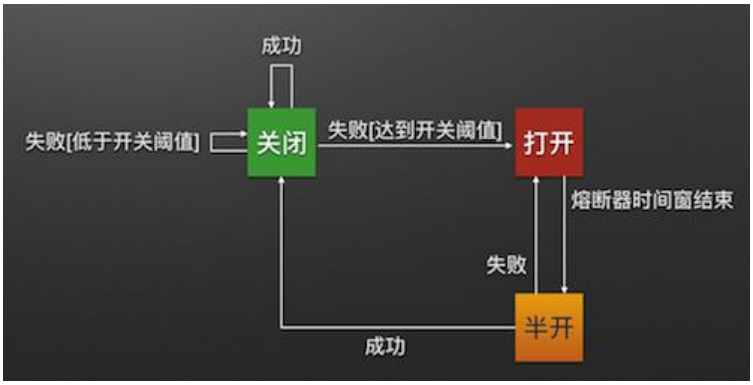


熔断器（CircuitBreaker）

原文链接：Spring-Cloud 系列第 4 篇:spring-cloud-Hystrix

熔断器的原理很简单，如同电力过载保护器。它可以实现快速失败，如果它在一段时间内侦测到许多似的错误，会强迫其以后的多个调用快速失败，不再访问远程服务器，从而防止应用程序不断地尝试行可能会失败的操作，使得应用程序继续执行而不用等待修正错误，或者浪费CPU时间去等到长时间超时产生。熔断器也可以使应用程序能够诊断错误是否已经修正，如果已经修正，应用程序会再次尝调用操作。

熔断器模式就像是那些容易导致错误的操作的一种代理。这种代理能够记录最近调用发生错误的次数然后决定使用允许操作继续，或者立即返回错误。 熔断器开关相互转换的逻辑如下图：



Hystrix特性

1. 断路器机制

断路器很好理解, 当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时后(默认5秒), 自动切换到半开路状态(HALF-OPEN). 这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN). Hystrix的断路器就像我们家庭电路的保险丝, 一旦后端服务不可用, 断路器会直接切断请求链, 避免发送大量无效请求影响系统吞吐量, 并断路器有自我检测并恢复的能力.

2. Fallback

Fallback相当于是降级操作. 对于查询操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常时候, 可以使用fallback方法返回的值. fallback方法的返回值一般是设置的默认值或者来自缓存.

3. 资源隔离

在Hystrix中, 主要通过线程池来实现资源隔离. 通常在使用的时候我们会根据调用的远程服务划分出多线程池. 例如调用产品服务的Command放入A线程池, 调用账户服务的Command放入B线程池. 这样的主要优点是运行环境被隔离开了. 这样就算调用服务的代码存在bug或者由于其他原因导致自己所线程池被耗尽时, 不会对系统的其他服务造成影响. 但是带来的代价就是维护多个线程池会对系统带来外的性能开销. 如果是对性能有严格要求而且确信自己调用服务的客户端代码不会出问题的话, 可以使Hystrix的信号模式(Semaphores)来隔离资源.

Feign Hystrix

为了使用 Hystrix 的 dashboard 面板，我们暂时使用了原生的的熔断器，没有使用和feign的结合。

如何使用

1. 添加依赖

```
<!--远程调用-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
<!--熔断器-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
<!--hystrix-dashboard 监控-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
</dependency>
```

2. 创建一个模拟远程调用，并模拟调用失败

```
@Service
public class CallDependencyService {

    private Random random = new Random();

    /**
     * 模拟获取用户信息，并模拟调用失败！
     */
    @HystrixCommand(fallbackMethod = "fallback")
    public String getRandomNumber() {
        int randomInt = random.nextInt(10);
        if (randomInt < 8) {  //模拟调用失败情况
            throw new RuntimeException("调用服务失败！");
        } else {
            return "调用服务成功！随机数是:" + randomInt;
        }
    }

    public String fallback() {
        return "随机数小小于8，表示将要熔断！这是熔断器的降级方法。";
    }
}
```

这里不是真正的远程调用，但是，如果想真的远程调用的话，可以使用我上一篇文章的三个服务，我里也集成了，如下：

```
@Component
@FeignClient(name = "eureka-client-1")
public interface HelloRemoteInterface {
```

```
    @RequestMapping(value = "/hi")
    public String hi();
}
```

熔断过程如下：

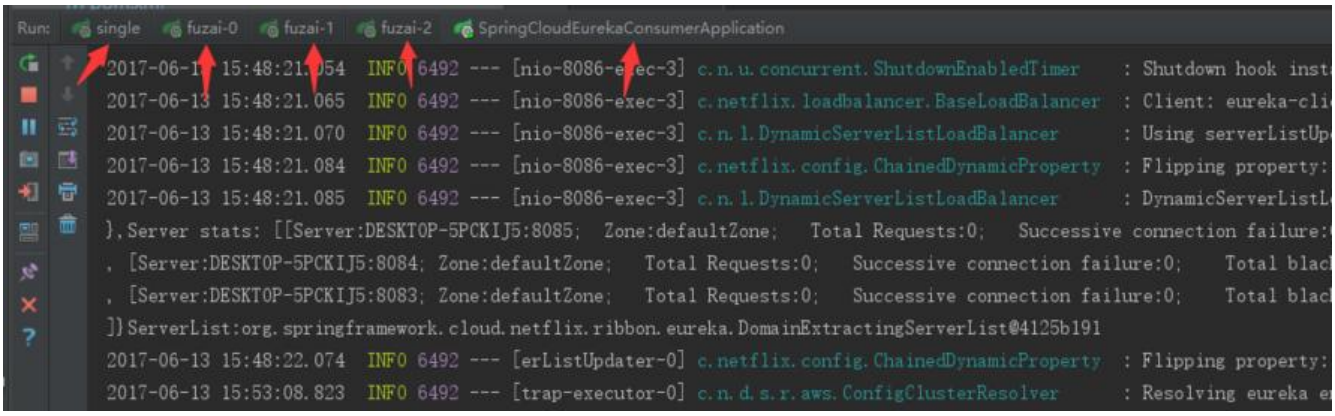
```
@Service
public class HelloRemoteService {

    @Autowired
    HelloRemoteInterface helloRemoteInterface;

    @HystrixCommand(fallbackMethod = "fallback")
    public String hi() {
        return helloRemoteInterface.hi();
    }

    public String fallback() {
        return "这是熔断器的降级方法。";
    }
}
```

当然了，你需要启动我之前的那些服务，如图：



其中，如果是简单测试的话，不建议把feign拿过来测试，因为还需要测试之前的好多功能。如果你有深刻理解的话，还是有一定难度的。所以，建议删除这里的代码，直接测试随机数那个熔断器，你很方便理解。

当然，我们需要一个controller：

```
@RestController
public class ConsumerController {

    @Autowired
    HelloRemoteService helloRemoteService;

    @Autowired
    CallDependencyService callDependencyService;
```

```
    @RequestMapping("/hello")
    public String hello() {
        return helloRemoteService.hi();
    }

    @RequestMapping("/random")
    public String hi() {
        return callDependencyService.getRandomNumber();
    }
}
```

配置文件：

```
server:
  port: 8086

spring:
  application:
    name: eureka-consumer

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

当然了，你如果不远程调用，那个eureka没有鸟用。

3. 启动

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
@EnableHystrix
@EnableHystrixDashboard
public class SpringCloudEurekaConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringCloudEurekaConsumerApplication.class, args);
    }
}
```

使用 EnableCircuitBreaker 或者 EnableHystrix 表明Spring boot工程启用hystrix,两个注解是等价。

其中 EnableHystrixDashboard 注解表示启动对hystrix的监控，可以查看web面板。

4. 测试熔断效果

我们访问 http://localhost:8086/random ,可以看到结果。

并打开 http://localhost:8086/hystrix 来访问我们的控制面板，输入 http://localhost:8086/hystrix.stream 来监控我们的访问。

熔断效果还是很好的。发现这个真的强大！其实自己造一个轮子也不难，但是有点麻烦咯，关键是设没有人家精巧和有经验。

示例源码

所有源码在我的github仓库里，传送门：https://github.com/xjtushilei/spring-cloud-simples.git

支持

如果你喜欢~ 给个星