什么是断路器

断路器模式源于Martin Fowler的Circuit Breaker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 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

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，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: c
```

- `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.c
    }

    public String addServiceFallback() {
        return "error";
    }

}
```

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

```
```

```java
@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，页面显示：error

Feign使用Hystrix

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，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.
```

- 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") In
        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。