Prof. Bernardo Copstein

Prof. Júlio Machado

Roteiro 4: Circuit Breaker

Introdução

No roteiro 3 vimos como acrescentar um "gateway" para, entre outras coisas, servir de ponto de entrada único para todas as requisições a um backend de serviços. Neste roteiro não iremos acrescentar novos serviços em nossa arquitetura e sim configurar um "circuit-breaker" para um dos serviços já existentes.

Um problema com arquiteturas baseadas em micros serviços é falha em sequência. Imagine a situação da figura 1 onde uma solicitação para um micro serviço desencadeia uma série de solicitações para vários micros serviços geograficamente distribuídos.

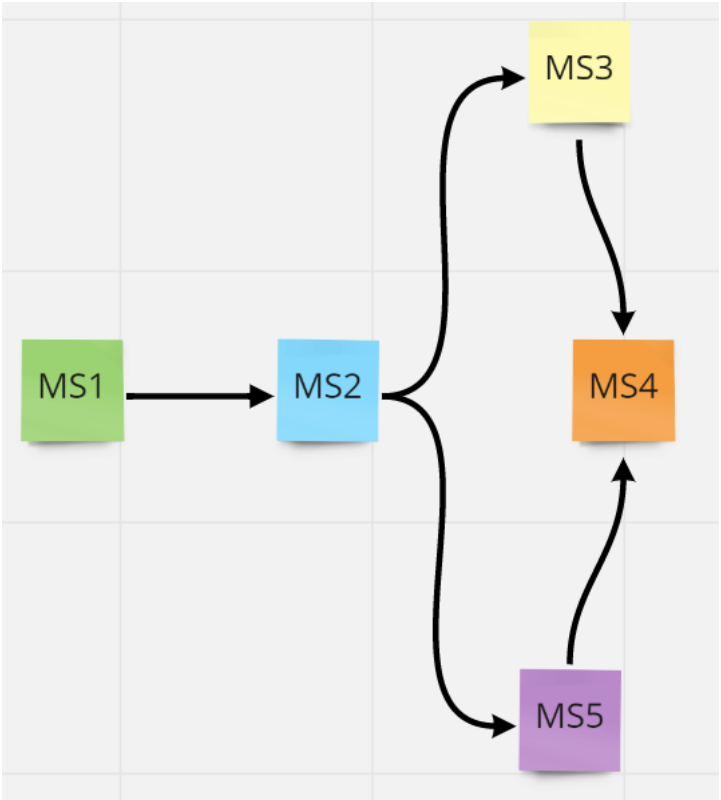

Figura 1 – MS1 faz uma requisição para o MS2. Este por sua vez demanda MS3 e MS5. MS3 como MS5 demandam MS4.

O problema é o que acontece se, por exemplo, MS4 falha? Toda a requisição falha. E pior, tanto MS3 como MS5 tentarão acessar MS4 e vão falhar.

A primeira questão é como um MS sabe que o outro não está respondendo? A maneira mais simples é acrescentar um "timeout" na requisição. Se depois de um certo tempo a resposta não vier assumimos que temos um problema na conexão. Às vezes, porém, ocorrem instabilidades na rede, então é comum que os MS tentem acessar os demais duas ou três vezes antes de desistirem. E em cada uma irão aguardar o timeout para ter certeza de que o outro MS não está lá. Em casos como o da figura a performance do sistema como um todo pode começar a degradar

visto que tanto MS3 como MS5 irão ficar tentando e aguardando os timeouts antes de desistirem. Isso pode fazer ainda com que MS2 desista antes do tempo, também, por timeout visto que MS3 e MS5 demoram demais para responder.

A solução para esse tipo de problema é usar um "circuit-breaker". O "circuit-breaker" se encarrega de monitorar as requisições acompanhando os tempos de resposta. Quando um micro serviço demora mais que o esperado o "circuit-breaker" já sinaliza a falha e/ou responde com uma resposta padrão. Assim o micro serviço que fez a requisição não perde tempo. Adicionalmente o "circuit-breaker" passa a responder todas as demais solicitações com a ação prevista para o caso de falha ao mesmo tempo que continua a monitorar a conexão. Quando esta voltar ao normal ele, automaticamente, passa a direcionar as requisições para o micro serviço que estava anteriormente inacessível.

Passo 1: configurando as dependências

A implementação do "circuit breaker" será feita através do projeto Spring Cloud Circuit Breaker, o qual provê uma camada de abstração sobre a implementação de diferentes "circuit breakers". Neste projeto será utilizada a implementação Resilience4J (https://resilience4j.readme.io/).

O primeiro passo será modificar o POM do projeto contendo o "api gateway" para incluir a dependência necessária. Veja na figura 2 a dependência que deve ser adicionada.

```
<dependency>
       <groupId>org.springframework.cloud</groupId>
           <artifactId>
              spring-cloud-starter-circuitbreaker-reactor-resilience4j
           </artifactId>
</dependency>
```

Figura 2 – configurando as dependências

Passo 2: configurando rotas no "gateway" com "circuit breaker"

O "circuit breaker" será configurado junto ao "API gateway" pois os serviços disponibilizados através do gateway podem potencialmente se comportar de forma ineficiente (aumentando os casos de timeout, por exemplo) ou se tornarem indisponíveis. Nosso objetivo é "empacotar" as rotas no "gateway" com os "circuit breakers". Para tal faremos o uso de "filters" associados às requisições (em especial será utilizado o "CircuitBreaker Gateway Filter"). A figura 3 mostra onde os filters atuam no processo de requisição/resposta a um serviço.

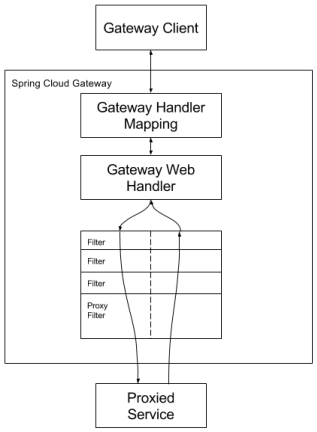

Figura 3 – filters no Spring Cloud Gateway

A figura 4 mostra a configuração de um “circuit breaker” para um dos endpoints do serviço de conversão de moedas. Neste exemplo, uma rota de fallback foi configurada para o caso do circuito estar aberto. A figura 5 mostra a implementação do endpoint de fallback no arquivo “FallbackController.java” que deve ser adicionado ao projeto.

```
import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ApiGatewayConfiguration {
  @Bean
  public RouteLocator gatewayRouter(RouteLocatorBuilder builder) {
    return builder.routes()
                .route(p -> p.path("/currency-exchange/**")
                      .uri("lb://currency-exchange"))
                .route(p -> p.path("/currency-conversion/**")
                      .filters(f -> f.circuitBreaker(c ->
                            c.setName("circuitbreaker_conversion")
                            .setFallbackUri("forward:/fallback/currency-
conversion")))
                      .uri("lb://currency-conversion"))
                .route(p -> p.path("/currency-conversion-feign/**")
                      .uri("lb://currency-conversion"))
                .build();
      }
}
```

Figura 4 – configurando as rotas com “circuit breaker”

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/fallback")
public class FallbackController {
    @GetMapping("/currency-conversion")
    public ResponseEntity<String> currencyConversionFallback() {
        return ResponseEntity
            .status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body("Serviços temporariamente indisponíveis. Tente
novamente.");
            //Implementação mais adequada seria retornar dados em cache de
            //uma requisição anterior com sucesso
    }
}
```

Figura 5 – código do endpoint de fallback

A figura 6 apresenta o diagrama de estados de um “circuit-breaker”. Quando o programa inicia o circuito está fechado (“close”). Nesta situação as mensagens são normalmente encaminhadas para o micro serviço destino. Quando é detectado que o micro serviço destino parou de responder, o circuito abre e passa a responder diretamente usando o “fallback method”. Podemos configurar, entretanto, para que o “circuit-breaker” passe para o estado “half-open”. Neste estado um percentual das requisições é respondido diretamente com o “fallback method” e um percentual segue para tentar atingir o micro serviço alvo. Desta forma quando alguma conseguir o circuito fecha normalmente e volta-se para o estado normal.

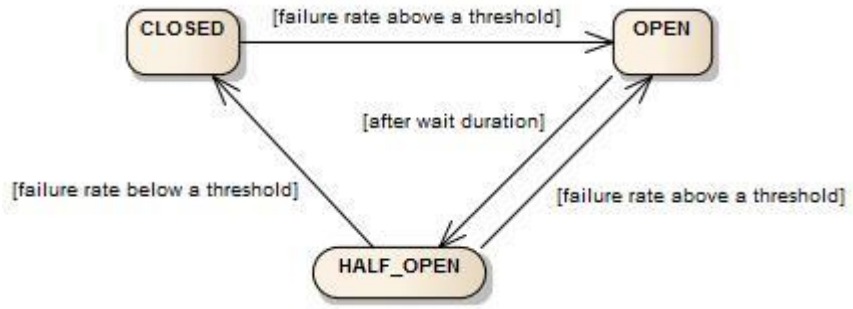

Figura 6 – estados de um “circuit-breaker”

Um “circuit breaker” possui dezenas de configurações (veja mais sobre as propriedades em https://resilience4j.readme.io/docs/circuitbreaker) que podem ser configuradas via código ou via o arquivo de propriedades. Na configuração padrão, por exemplo, o circuito abre após 50% (“failureRateThreshold”) de 100 (“slidingWindowSize”) requisições apresentarem erro. A figura 7 traz um exemplo de configuração via arquivo “application.properties”. Já a figura 8 mostra como seria uma configuração alternativa via código no arquivo “ApiGatewayConfiguration.java”.

```
resilience4j.circuitbreaker.configs.default.slidingWindowSize=5
resilience4j.circuitbreaker.configs.default.failureRateThreshold=50
resilience4j.circuitbreaker.configs.default.permittedNumberOfCallsInHalfOpen
State=3
resilience4j.timelimiter.configs.default.timeoutDuration=200ms
```

Figura 7 - configuração do “circuit-breaker”

```
@Bean
public Customizer<Resilience4JCircuitBreakerFactory> defaulCustomizer() {
	return f -> f.configureDefault(id -> new Resilience4JConfigBuilder(id)
			.circuitBreakerConfig(CircuitBreakerConfig.custom()
				.slidingWindowSize(5)
				.permittedNumberOfCallsInHalfOpenState(3)
				.failureRateThreshold(50.0F)
				.build()
			)
			.timeLimiterConfig(TimeLimiterConfig.custom()
				.timeoutDuration(Duration.ofMillis(200))
				.build()
			)
			.build()
	);
}
```

Figura 8 – configuração do “circuit-breaker”

Passo 3: explorando o “circuit breaker”

Suba todos os micros serviços na ordem: “name server”, câmbio, conversão e “api gateway”.

Para podermos ver o “circuit breaker” em ação, teste com as URLs que seguem:

http://localhost:8765/currency-exchange/from/USD/to/INR

http://localhost:8765/currency-conversion/from/USD/to/INR/quantity/10

A seguir, interrompa o micro serviço de conversão e acesse novamente a URL:

http://localhost:8765/currency-conversion/from/USD/to/INR/quantity/10

Procure observar se o endereço de fallback foi utilizado.