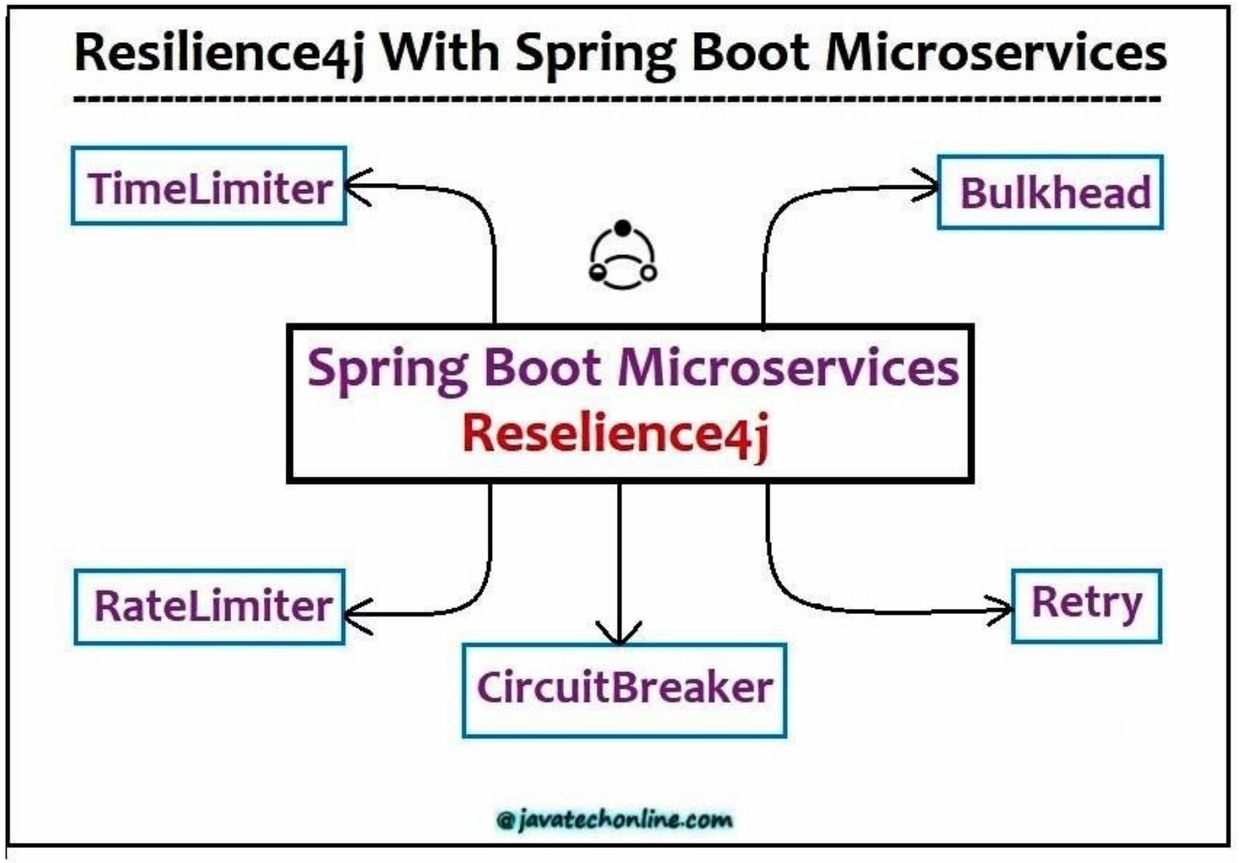
Resilience4J

Norberto Hideaki Enomoto
norberto.enomoto@dxc.com

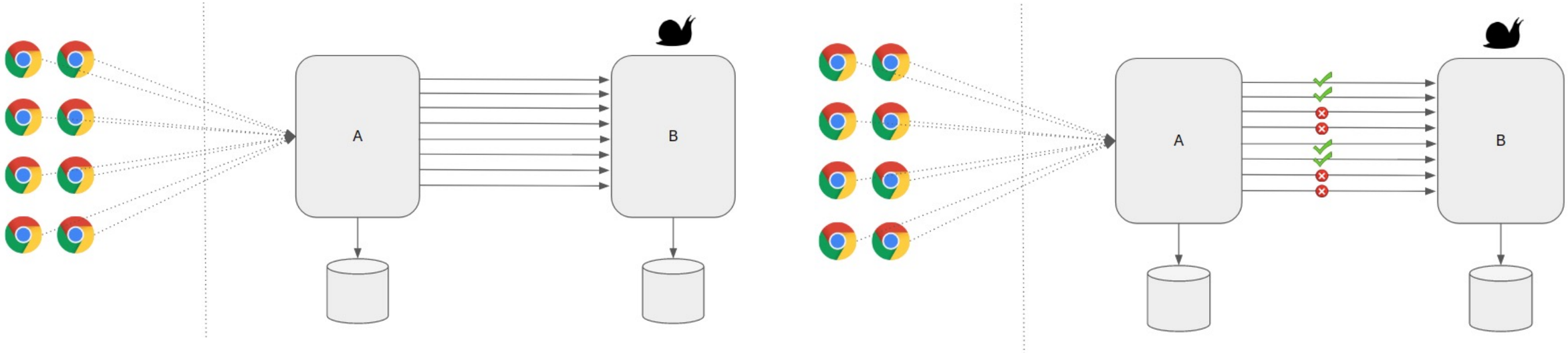
Resilience4J



Rate Limiter

O Rate Limiter limita o número de solicitações em um determinado período. Vamos supor que queremos limitar o número de solicitações em uma API Rest por um determinado período. Existem vários motivos para limitar o número de solicitações de uma API, como proteger os recursos de um ataque de negação de serviço, minimizar a sobrecarga, cumprir um acordo de nível de serviço e muitos outros.

Rate Limiter



Rate Limiter

```
resilience4j.ratelimiter:
  instances:
    getMessageRateLimit:
      limitForPeriod: 2
      limitRefreshPeriod: 5s
```

Somente 2 requests no período de 5 segundos

Bulkhead

No contexto do mecanismo de tolerância a falhas, se quisermos limitar o número de solicitações simultâneas, podemos usar o Bulkhead. Usando o Bulkhead, podemos limitar o número de solicitações simultâneas em um determinado período. Observe a diferença entre Bulkhead e Rate Limiter. O Rate Limiter nunca fala sobre solicitações simultâneas, mas o Bulkhead sim. Portanto, usando o Bulkhead, podemos limitar o número de solicitações simultâneas.

Bulkhead

```
resilience4j.bulkhead:
    instances:
        getMessageBH:
            maxWaitDuration: 0s
            maxConcurrentCalls: 5
```

- indica que se o número de chamadas simultâneas exceder 5, ative o método de fallback.
- indica que não espera nada, mostre a resposta imediatamente com base na configuração.

Time Limiter

Time Limiter é o processo de definir um limite de tempo para a resposta de um microsserviço. Suponha que o microsserviço ‘A’ envie uma solicitação ao microsserviço ‘B’, ele define um limite de tempo para o microsserviço ‘B’ responder. Se o microsserviço ‘B’ não responder dentro desse limite de tempo, será considerado que há alguma falha. O Time limiter é utilizado para chamada assíncronas, utilizando CompletableFuture

Time Limiter

```
resilience4j.timelimiter:
    instances:
        getMessageTL:
            timeoutDuration: 1ms
            cancelRunningFuture: false
```

- Indica que o tempo máximo que uma solicitação pode levar para responder é 1 milissegundo
- Indica que não cancela o Running Completable Future depois do TimeOut.

Retry

Suponha que o microsserviço ‘A’ dependa de outro microsserviço ‘B’. Suponhamos que o microsserviço ‘B’ seja um serviço defeituoso e sua taxa de sucesso seja de apenas 50-60%. No entanto, a falha pode ser devido a qualquer motivo, como serviço não disponível, serviço com bugs que às vezes responde e às vezes não, ou uma falha de rede intermitente etc. No entanto, neste caso, se o microsserviço 'A' tentar enviar novamente a solicitação 2 a 3 vezes, as chances de obter resposta aumentam

Retry

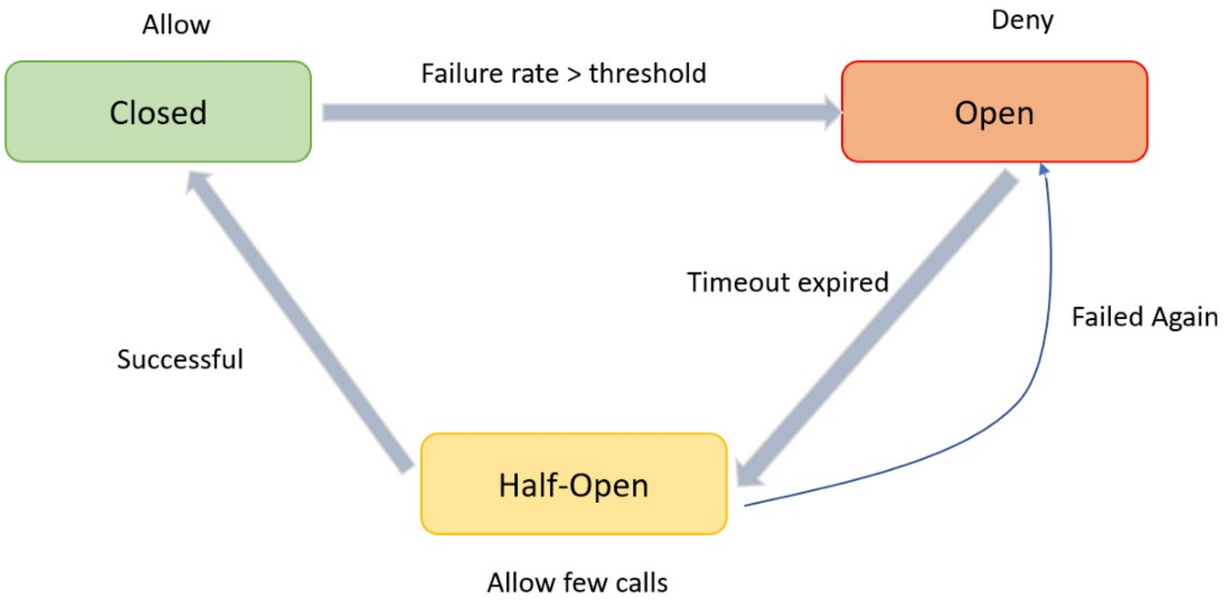
```
resilience4j.retry:
  instances:
    getInvoiceRetry:
      maxAttempts: 5
      waitDuration: 2s
      retryExceptions:
        - org.springframework.web.client.HttpServerErrorException
        - org.springframework.web.client.HttpClientErrorException
        - java.io.IOException
        - org.springframework.web.client.ResourceAccessException
      ignoreExceptions:
        - io.github.robwin.exception.BusinessException
```

5 tentativas em um intervalo de 2 segundos

Circuit Breaker

Por exemplo, se um microsserviço ‘A’ depende do microsserviço ‘B’. Por algum motivo, o microsserviço ‘B’ está apresentando um erro. Em vez de chamar repetidamente o Microsserviço ‘B’, o Microsserviço ‘A’ deve fazer uma pausa (não chamar) até que o Microservice ‘B’ seja completamente ou parcialmente recuperado. Usando o Circuit Breaker, podemos eliminar o fluxo de falhas.

Circuit Breaker



Circuit Breaker

```
resilience4j.circuitbreaker:
    instances:
        getInvoiceCB:
            failureRateThreshold: 80
            slidingWindowSize: 10
            slidingWindowType: COUNT_BASED
            minimumNumberOfCalls: 5
            automaticTransitionFromOpenToHalfOpenEnabled: true
            permittedNumberOfCallsInHalfOpenState: 4
            waitDurationInOpenState: 1s
```

- indica que se 80% das solicitações falharem, abra o circuito, ou seja, defina o estado do disjuntor como aberto.
- indica que se 80% das solicitações em 10 (significa 8) estiverem falhando, abra o circuito.
- indica que estamos usando a janela deslizante COUNT_BASED. Outro tipo é TIME_BASED.
- indica que estamos usando a janela deslizante COUNT_BASED. Outro tipo é TIME_BASED.
- indica que não muda diretamente do estado aberto para o estado fechado, considere também o estado semiaberto.
- indica que, quando estiver no estado semi-aberto, considere o envio de 4 solicitações. Se 80% deles estiverem falhando, mude o disjuntor para o estado aberto.
- indica o intervalo de tempo de espera ao passar do estado aberto para o estado fechado.

Questions and answers