Round 6 - 서킷브레이커

결제 흐름

서킷브레이커 흐름 설정

```
resilience4j:
  circuitbreaker:
    instances:
      paymentService:
        slidingWindowType: TIME
        slidingWindowSize: 30s           # 최근 30초 기준
        minimumNumberOfCalls: 20         # 20건 모일 때부터 실패율 판단
        failureRateThreshold: 60         # 실패율 60% 이상이면 Open
        slowCallDurationThreshold: 2s    # 2초 넘으면 '느린 호출'
        slowCallRateThreshold: 50        # 느린 호출 비율 50% 넘으면 Open
        waitDurationInOpenState: 10s     # Open 유지 10초
        permittedNumberOfCallsInHalfOpenState: 5
        automaticTransitionFromOpenToHalfOpenEnabled: true
        recordExceptions:
          - java.net.SocketTimeoutException
          - org.springframework.web.client.HttpServerErrorException
        ignoreExceptions:
          - org.springframework.web.client.HttpClientErrorException # 
4xx 등은 보통 제외
```

지금 흐름

orderId 로 PG사에 요청

PG사가 > Client 에게 transactionalKey 전달

요청 > 2~3초후 콜백 Url 로 결과 받음.

어디서 받을지 설정해야함.

각 설정에 대한 VS

Getting started

https://resilience4j.readme.io/docs/getting-started-3

- 트래픽이 일정하고 충분히 많은 서비스 → COUNT_BASED 가 안정적이다. (예: 내부 마이크로서비스 간 통신)
- 트래픽이 들쑥날쑥하거나 순간적인 과부하가 중요한 서비스 → TIME_BASED 가 유리하다. (예: 외부 결제 API, 낮에는 호출 많고 밤에는 거의 없음)

---

5. 재시도 조건

- 네트워크 관련 예외 (IOException, SocketTimeoutException)
- Spring RestTemplate 관련 예외 (ResourceAccessException, RestClientException)
- FeignClient에 CircuitBreaker 어노테이션 추가 및 테스트
  https://techblog.woowahan.com/15694/

https://engineering.linecorp.com/ko/blog/circuit-breakers-for-distributed-services

서킷브레이커는 상태에 따라서 서로 다른 동작

서킷브레이커는 3가지의 보통 상태(OPEN, CLOSED, HALF_OPEN)와 2가지의 특별한 상태(DISABLED, FORCED_OPEN)를 갖습니다.

보통 상태로는 정상적으로 호출되고 응답을 주는 CLOSED 상태, 문제 발생이 감지된 OPEN과 HALF_OPEN 상태가 있습니다. 특별한 상태로는 항상 호출을 허용하는 DISABLED 상태와 항상 호출을 거부하는 FORCED_OPEN 상태가 있습니다.

```
io.github.resilience4j.circuitbreaker.CallNotPermittedException:
CircuitBreaker 'pgSimulatorCircuitBreaker' is OPEN and does not
permit further calls

at
io.github.resilience4j.circuitbreaker.CallNotPermittedException.creat
eCallNotPermittedException(CallNotPermittedException.java:48)

at
```

```
io.github.resilience4j.circuitbreaker.internal.CircuitBreakerStateMachine$OpenState.acquirePermission(CircuitBreakerStateMachine.java:752)

at io.github.resilience4j.circuitbreaker.internal.CircuitBreakerStateMachine.acquirePermission(CircuitBreakerStateMachine.java:206)

at io.github.resilience4j.circuitbreaker.CircuitBreaker.lambda$decorateCheckedSupplier$0(CircuitBreaker.java:68)

at io.github.resilience4j.circuitbreaker.CircuitBreaker.executeCheckedSupplier(CircuitBreaker.java:739)

at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.defaultHandling(CircuitBreakerAspect.java:179)

at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.proceed(CircuitBreakerAspect.java:126)

at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.lambda$circuitBreakerAroundAdvice$0(CircuitBreakerAspect.java:108)
```

- 서킷브레이커 설정을 고려할때 주의할점은 각 인스턴스 마다 서킷브레이커가 별개로 동작한다는 점이다. 예를들어 TPS 1000을 처리하는 서버가 인스턴스 10대로 되어있다면 각 인스턴스 서킷브레이커마다 100정도의 TPS를 처리하고 있다고 가정해야한다.
-

1. slidingWindowType (COUNT_BASED vs TIME_BASED)

- COUNT_BASED (기본값): 최근 호출 횟수 기준으로 실패율 계산.

- TIME_BASED: 최근 일정 시간 동안의 호출을 기준으로 실패율 계산.
- 실무 팁:
  - 트래픽이 일정하다면 COUNT_BASED 도 충분하다.
  - 하지만 대부분의 서비스는 피크 타임과 새벽 시간대의 트래픽 차이가 크다. 이런 경우에는 TIME_BASED 가 더 자연스럽다.

2. slidingWindowSize

- 크게 잡으면
  - 장점: 모수가 많아 안정적인 통계
  - 단점: 장애 감지가 느려짐
- 작게 잡으면
  - 장점: 장애를 빠르게 감지 가능
  - 단점: 소수 데이터 때문에 과민 반응(Open/Close 반복)

👉 따라서 아래 2가지를 기준으로 결정하면 된다.

1. 장애를 얼마나 빨리 격리해야 하는가? → 크리티컬하다면 size를 작게
2. 시스템이 얼마나 불안정한가? → 자주 흔들린다면 size를 크게

📌 예시:

- 타임아웃이 3~5초 수준이고, 평균 TPS가 100이라면 → 약 500개 정도 size가 합리적이다.
- TIME_BASED 로 고정 시간(예: 60초) 으로 잡으면 트래픽 변동에도 균형 있게 대응할 수 있다.

failureRateThreshold (실패율 임계치)

- 높게 잡으면: 서킷이 잘 열리지 않아 장애가 계속 전파될 수 있다.
- 낮게 잡으면: 너무 민감하게 반응해 자주 열렸다 닫혔다 할 수 있다.

📌 주의할 점:
• 하나의 서킷브레이커에 여러 API 요청이 섞이면, 특정 API만 불안정해도 평균 실패율이 낮아져서 서킷이 안 열릴 수 있다.
• 예:
• A API: 초당 1000건, 장애 시 실패율 2% (20건 실패)
• B API: 초당 200건, 장애 시 실패율 90% (180건 실패)

• 전체 실패율 = (20+180)/(1200) ≈ 16%
• 만약 failureRateThreshold=20% 로 두면 B API 가 망가져도 서킷이 안 열린다.
• → 이런 경우에는 API별로 분리하거나 threshold를 더 낮춰야 한다.

waitDurationInOpenState

- 서킷이 Open 상태로 유지되는 시간.
- 짧게 잡으면 금방 Half-Open 으로 열어 복구 여부를 확인할 수 있다.
- 보통 서비스 순단(잠깐 끊김)을 고려해서 5초 정도가 기본값으로 적당하다.

📌 팁:

- "더 긴 회복 시간 확보"를 원한다면 이 값을 늘리기보다는 Half-Open 시 허용 호출 수 (permittedNumberOfCallsInHalfOpenState) 를 줄이는 것이 낫다.

permittedNumberOfCallsInHalfOpenState

- Half-Open 상태에서 몇 개의 요청을 통과시켜 볼 것인가?
- 너무 크게 잡으면: 사실상 Closed 와 비슷해져, 아직 회복되지 않은 시스템에 부담을 줄 수 있다.
- 너무 작게 잡으면: 표본이 부족해 "운 좋게 성공한 몇 건" 때문에 Closed 로 돌아가 버릴 수 있다.

📌 고려할 점:

- 대규모 스케일아웃 환경(예: 서버 100대)에서는 각 서버마다 독립적으로 서킷브레이커가 동작한다. 따라서 숫자가 작아도 전체적으로는 꽤 많은 호출이 발생할 수 있다.
- 경험상 TPS의 10~20% 수준이 적당하다.

✅ 요약

1. slidingWindowType: 일반적으로는 TIME_BASED 가 더 적합 (트래픽 변동 대응).
2. slidingWindowSize: 크면 안정적, 작으면 민감. 장애 인지 속도 vs 안정성 균형 고려.
3. failureRateThreshold: 너무 높으면 장애 방치, 너무 낮으면 과민 반응. API별 특성 고려.
4. waitDurationInOpenState: 기본은 5초, 더 필요하면 Half-Open 허용 호출 수로 조정.
5. permittedNumberOfCallsInHalfOpenState: 너무 크면 부담, 너무 작으면 잘못된 판단. 보통 TPS의 10~20% 권장.

windowSize = 10
minum Call = 3
10개를 채우고 10개중에 5개 실패하면 Open 을 한다.

F / F / F <- Open
F / F / F 3개 Call 되기 전까지 평가 노노
F / F / P / P / F <- Open