12. 스프링 AOP - 실전 예제

#0.강의/3.스프링로드맵/6.핵심 원리 - 고급편/강의#

예제 만들기

지금까지 학습한 내용을 활용해서 유용한 스프링 AOP를 만들어보자.

- `@Trace` 애노테이션으로 로그 출력하기
- `@Retry` 애노테이션으로 예외 발생시 재시도 하기

먼저 AOP를 적용할 예제를 만들자.

ExamRepository

```
package hello.aop.exam;

import org.springframework.stereotype.Repository;

@Repository
public class ExamRepository {

    private static int seq = 0;

    /**
     * 5번에 1번 실패하는 요청
     */
    public String save(String itemId) {
        seq++;
        if (seq % 5 == 0) {
            throw new IllegalStateException("예외 발생");
        }
        return "ok";
    }
```

```
}
```

5번에 1번 정도 실패하는 저장소이다. 이렇게 간헐적으로 실패할 경우 재시도 하는 AOP가 있으면 편리하다.

ExamService

```
package hello.aop.exam;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class ExamService {

    private final ExamRepository examRepository;

    public void request(String itemId) {
        examRepository.save(itemId);
    }

}
```

ExamTest

```
package hello.aop.exam;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class ExamTest {

    @Autowired
    ExamService examService;

    @Test
    void test() {
        for (int i = 0; i < 5; i++) {
            examService.request("data" + i);
```

```
        }
    }
}
```

실행해보면 테스트가 5번째 루프를 실행할 때 리포지토리 위치에서 예외가 발생하면서 실패하는 것을 확인할 수 있다.

로그 출력 AOP

먼저 로그 출력용 AOP를 만들어보자.

`@Trace`가 메서드에 붙어 있으면 호출 정보가 출력되는 편리한 기능이다.

@Trace

```
package hello.aop.exam.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Trace {
}
```

TraceAspect

```
package hello.aop.exam.aop;

import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Slf4j
@Aspect
public class TraceAspect {
```

```
    @Before("@annotation(hello.aop.exam.annotation.Trace)")
    public void doTrace(JoinPoint joinPoint) {
        Object[] args = joinPoint.getArgs();
        log.info("[trace] {} args={}", joinPoint.getSignature(), args);
    }
}
```

`@annotation(hello.aop.exam.annotation.Trace)` 포인트컷을 사용해서 `@Trace`가 붙은 메서드에 어드바이스를 적용한다.

ExamService - @Trace 추가

```
package hello.aop.exam;

import hello.aop.exam.annotation.Trace;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class ExamService {

    private final ExamRepository examRepository;

    @Trace
    public void request(String itemId) {
        examRepository.save(itemId);
    }

}
```

`request()`에 `@Trace`를 붙였다. 이제 메서드 호출 정보를 AOP를 사용해서 로그로 남길 수 있다.

ExamRepository - @Trace 추가

```
package hello.aop.exam;

import hello.aop.exam.annotation.Trace;
import org.springframework.stereotype.Repository;

@Repository
```

```
public class ExamRepository {

    @Trace
    public String save(String itemId) {
      //...
    }
}
```

`save()`에 `@Trace`를 붙였다.

ExamTest - 추가

```
@Import(TraceAspect.class)
@SpringBootTest
public class ExamTest {
}
```

`@Import(TraceAspect.class)`를 사용해서 `TraceAspect`를 스프링 빈으로 추가하자. 이제 애스펙트가 적용된다.

실행 결과

```
[trace] void hello.aop.exam.ExamService.request(String) args=[data0]
[trace] String hello.aop.exam.ExamRepository.save(String) args=[data0]
[trace] void hello.aop.exam.ExamService.request(String) args=[data1]
[trace] String hello.aop.exam.ExamRepository.save(String) args=[data1]
...
```

실행해보면 `@Trace`가 붙은 `request()`, `save()` 호출시 로그가 잘 남는 것을 확인할 수 있다.

재시도 AOP

이번에는 좀 더 의미있는 재시도 AOP를 만들어보자.

`@Retry` 애노테이션이 있으면 예외가 발생했을 때 다시 시도해서 문제를 복구한다.

@Retry

```
package hello.aop.exam.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Retry {
    int value() default 3;
}
```

이 애노테이션에는 재시도 횟수로 사용할 값이 있다. 기본값으로 `3` 을 사용한다.

RetryAspect

```
package hello.aop.exam.aop;

import hello.aop.exam.annotation.Retry;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;

@Slf4j
@Aspect
public class RetryAspect {

    @Around("@annotation(retry)")
    public Object doRetry(ProceedingJoinPoint joinPoint, Retry retry) throws
Throwable {

        log.info("[retry] {} retry={}", joinPoint.getSignature(), retry);

        int maxRetry = retry.value();
        Exception exceptionHolder = null;

        for (int retryCount = 1; retryCount <= maxRetry; retryCount++) {
            try {
```

```
                log.info("[retry] try count={}/{}", retryCount, maxRetry);
                return joinPoint.proceed();
            } catch (Exception e) {
                exceptionHolder = e;
            }
        }
        throw exceptionHolder;
    }
}
```

- 재시도 하는 애스펙트이다.
- `@annotation(retry)`, `Retry retry`를 사용해서 어드바이스에 애노테이션을 파라미터로 전달한다.
- `retry.value()`를 통해서 애노테이션에 지정한 값을 가져올 수 있다.
- 예외가 발생해서 결과가 정상 반환되지 않으면 `retry.value()`만큼 재시도한다.

ExamRepository - @Retry 추가

```
package hello.aop.exam;

import hello.aop.exam.annotation.Retry;
import hello.aop.exam.annotation.Trace;
import org.springframework.stereotype.Repository;

@Repository
public class ExamRepository {

    @Trace
    @Retry(value = 4)
    public String save(String itemId) {
      //...
    }
}
```

`ExamRepository.save()` 메서드에 `@Retry(value = 4)`를 적용했다. 이 메서드에서 문제가 발생하면 4번 재시도 한다.

ExamTest - 추가

```
@SpringBootTest
//@Import(TraceAspect.class)
```

```
@Import({TraceAspect.class, RetryAspect.class})
public class ExamTest {
}
```

- `@Import(TraceAspect.class)`는 주석처리하고
- `@Import({TraceAspect.class, RetryAspect.class})`를 스프링 빈으로 추가하자.

실행 결과

```
...
[retry] try count=1/5
[retry] try count=2/5
```

실행 결과를 보면 5번째 문제가 발생했을 때 재시도 덕분에 문제가 복구되고, 정상 응답되는 것을 확인할 수 있다.

참고

스프링이 제공하는 `@Transactional`은 가장 대표적인 AOP이다.

정리