AOP 프로그래밍 2

풍부한 서식의 문서를 신속하게 작성해 보세요.

스프링 AOP 구현

- Aspect로 사용할 클래스에 @Aspect 애노테이션을 붙인다
- @Pointcut 애노테이션으로 공통 기능을 적용할 Pointcut을 정의한다
- 공통 기능을 구현한 메서드에 @Around 애노테이션을 적용한다.

Aspect 클래스 구현

```
package basic_5.aspect;

import java.util.Arrays;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.Signature;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect //Aspect로 사용할 클래스
public class ExeTimeAspect {

  //basic_5패키지와 그 하위 패키지에 위치한 타입의 public 메서드를 pointcut으로 설정
  @Pointcut("execution(public * basic_5..*(..))")
  private void publicTarget() {
  }

  //publicTarget() 메서드에 정의한 Pointcut에 공통 기능을 적용
  @Around("publicTarget()")
  //ProceedingJoinPoint 타입 파라미터는 프록시 대상 객체의 메서드를 호출할 때 사용
  public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
    long start = System.nanoTime();
    try {
      //proceed로 대상 객체의 메서드를 호출한다.
      //이 코드를 입력하면 메서드가 실행 되므로 이전과 이후에 공통 기능을 위한 코드를 위치 시키면된다 (여기선 시간 계산 코드)
      Object result = joinPoint.proceed();
      return result;
    } finally {
      long finish = System.nanoTime();
      //getSignature() 메서드는 시그니처를 구할 때
      Signature sig = joinPoint.getSignature();
      //실행시간 출력 코드
      System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",
          //getTarget() 메서드는 대상객체를 구할 때
          joinPoint.getTarget().getClass().getSimpleName(),
          //getName() 메서드는 인자 목록을 구할 때
```

```
          sig.getName(),Arrays.deepToString(joinPoint.getArgs()),
          (finish - start));
    }
  }
}
```

메서드 시그니처

```
//getSignature() 메서드는 시그니처를 구할 때
      Signature sig = joinPoint.getSignature();
```

여기서 시그니처란 **메서드의 이름과 파라미터를 합쳐서** 메서드 시그니처라고 한다.

메서드 이름이 다르거나 파라미터 타입, 개수가 다르면 시그니처가 다르다고 표현한다.

메서드의 **리턴 타입이나 익셉션 타입은 시그니처에 포함되지 않는다**

스프링 설정 클래스

```
package basic_5.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

import basic_5.Calculator;
import basic_5.RecCalculator;
import basic_5.aspect.ExeTimeAspect;

@Configuration
//@Aspect 애노테이션을 붙인 클래스를 공통 기능으로 적용하려면
//@EnableAspectJAutoProxy 애노테이션을 설정 클래스에 붙여야한다
@EnableAspectJAutoProxy
public class AppCtx {

  //ExeTimeAspect 클래스에서 basic_5 패키지 아래 public 메서드를 설정하므로
  //calculator 빈에 ExeTimeAspect 클래스에 정의한 공통기능인 measure()를 적용한다.
  @Bean
  public ExeTimeAspect exeTimeAspect() {
    return new ExeTimeAspect();
  }

  @Bean
  public Calculator calculator() {
    return new RecCalculator();
  }
}
```

@Enable 류 애노테이션

- 스프링은 Enable로 시작하는 다양한 애노테이션을 제공한다.

- @Enable로 시작하는 애노테이션은 관련 기능을 적용하는데 필요한 설정을 대신 처리한다.

ProceedingJoinPoint의 메서드

- Signature getSignature() : 호출되는 메서드에 대한 정보를 구한다.
- Object getTarget() : 대상 객체를 구한다.
- Object[] getArgs() : 파라미터 목록을 구한다.
- org.aspectj.lang.Signature 인터페이스
- String getName() : 호출되는 메서드의 이름을 구한다.
- String toLongString() : 호출되는 메서드를 완전하게 표현한 문장을 구한다.(리턴타입,파라미터 타입 완전 표시)
- String toShortString() : 호출되는 메서드를 축약해서 표현한 문장을 구한다.