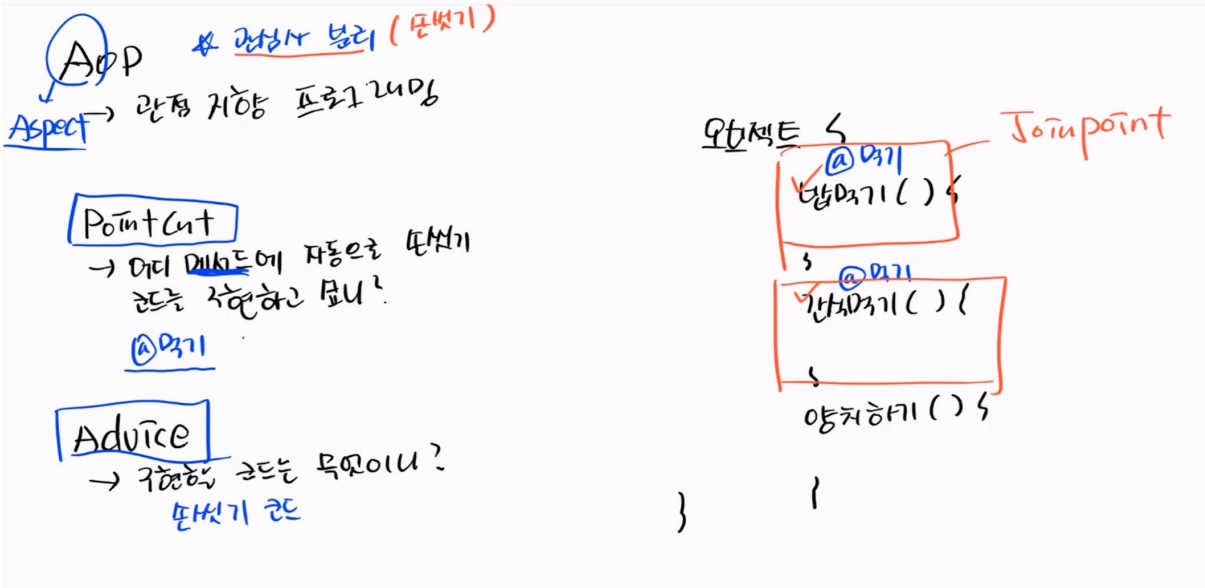
회원가입 컨트롤러 유효성검사 AOP 적용

AOP 개념 설명



`Aspect Oriented Programming`

- 관점 지향 프로그래밍
- **관심사의 분리**

예시) 인간 클래스

- **밥먹기(), 간식먹기(), 양치하기()**라는 메소드가 있는데
- **먹는 행위에는 반드시 손씻기 코드를 먼저 수행해주고 싶음!**

`PointCut`

- 어디 메서드에 자동으로 중복 코드를 구현하고 싶니?
- 타겟 메소드에 어노테이션이 있는 곳!
  - `@먹기`
- 타겟 메소드에 `@먹기` 를 붙여 놓으면 `PointCut`이 적용됨

`Advice`

- 구현할 중복 코드는 무엇이니?
- 원하는 코드를 작성해주면 된다.

정리

- `PointCut`과 `Advice`를 `Aspect`라고 한다.

- 타겟 메소드를 `JoinPoint` 라고 한다.

컨트롤러에 AOP 적용하기

개요

- `Validation`을 `AOP`로 하고 싶다.
- `Validation`은 **요청 바디**만 해주면 된다.
- 우리가 사용할 `HTTP Method` 중에 **요청 바디**가 있는`POST`와 `PUT`만 해주면 된다!

CustomValidationAdvice

```
package shop.mtcoding.bank.handler.aop;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import shop.mtcoding.bank.dto.ResponseDto;
import shop.mtcoding.bank.handler.ex.CustomValidationException;

import java.util.HashMap;
import java.util.Map;

@Component
@Aspect
public class CustomValidationAdvice {
    /**
     * Validation은 body데이터가 있는 POST와 PUT만 해주면 된다.
     * */
    @Pointcut("@annotation(org.springframework.web.bind.annotation.PostMapping)")
    public void postMapping() {

    }
    @Pointcut("@annotation(org.springframework.web.bind.annotation.PutMapping)")
    public void putMapping() {

    }
    // joinPoint(타겟 메소드)의 전후 제어
    @Around("postMapping() || putMapping()")
    public Object validationAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        Object[] args = proceedingJoinPoint.getArgs(); // joinPoint의 매개 변수들
        for (Object arg : args) {
            if (arg instanceof BindingResult) {
                BindingResult bindingResult = (BindingResult) arg;

                if (bindingResult.hasErrors()) {

                    Map<String, String> errorMap = new HashMap<>();

                    for (FieldError error : bindingResult.getFieldErrors()) {
                        errorMap.put(error.getField(), error.getDefaultMessage());
                    }
                    throw new CustomValidationException("유효성 검사 실패", errorMap);
                }
```

```
            }
        }
        return proceedingJoinPoint.proceed(); // 정상적으로 joinPoint(타겟 메소드)를 실행해라
    }
}
```

PointCut

```
/**
 * Validation은 body데이터가 있는 POST와 PUT만 해주면 된다.
 * */
@Pointcut("@annotation(org.springframework.web.bind.annotation.PostMapping)")
public void postMapping() {

}
@Pointcut("@annotation(org.springframework.web.bind.annotation.PutMapping)")
public void putMapping() {
```

- `@PostMapping`, `@PutMapping`이라는 어노테이션이 붙어있는 타겟 메소드에 `Advice`를 적용

Advice

```
// joinPoint(타겟 메소드)의 전후 제어
@Around("postMapping() || putMapping()")
public Object validationAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
    Object[] args = proceedingJoinPoint.getArgs(); // joinPoint의 매개 변수들
    for (Object arg : args) {
        if (arg instanceof BindingResult) {
            BindingResult bindingResult = (BindingResult) arg;

            Map<String, String> errorMap = new HashMap<>();

            for (FieldError error : bindingResult.getFieldErrors()) {
                errorMap.put(error.getField(), error.getDefaultMessage());
            }
            throw new CustomValidationException("유효성 검사 실패", errorMap);
        }
    }
    return proceedingJoinPoint.proceed(); // 정상적으로 joinPoint(타겟 메소드)를 실행해라
}
```

Advice의 다양한 어노테이션

- `@Before` : `joinPoint`(타겟 메소드)전에 실행
- `@After` : `joinPoint` 후에 실행
- `@Around` : `joinPoint` 전후를 제어
    - 파라미터로 `ProceedingJoinPoint`를 받을 수 있다! → 실행중인 타겟 메소드

`@Around("postMapping() || putMapping()")`

- `@PostMapping`, `@PutMapping` 이 붙어 있는 메소드 전후를 제어

로직

- `bindingResult`가 있고, `FiledError`가 있다면 `Map`객체에 담아서 사용자 정의 예외를 터트려라
- 그런 게 없다면 정상적으로 `joinPoint`를 실행해라

CustomValidationException

```
package shop.mtcoding.bank.handler.ex;

import lombok.Getter;

import java.util.Map;
@Getter
public class CustomValidationException extends RuntimeException{

    private Map<String, String> errorMap;

    public CustomValidationException(String message, Map<String, String> errorMap) {
        super(message);
        this.errorMap = errorMap;
    }
}
```

CustomExceptionHandler

- `CustomValidationException` 얘를 처리해줄 메소드를 작성

```
package shop.mtcoding.bank.handler;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import shop.mtcoding.bank.dto.ResponseDto;
import shop.mtcoding.bank.handler.ex.CustomApiException;
import shop.mtcoding.bank.handler.ex.CustomValidationException;

@RestControllerAdvice
public class CustomExceptionHandler {

    private final Logger log = LoggerFactory.getLogger(getClass());

    @ExceptionHandler(CustomApiException.class)
    public ResponseEntity<?> apiException(CustomApiException e) {
        log.error(e.getMessage());
        return new ResponseEntity<>(new ResponseDto<>(-1, e.getMessage(), null),
                                                HttpStatus.BAD_REQUEST);
    }

    // 추가
    @ExceptionHandler(CustomValidationException.class)
    public ResponseEntity<?> validationApiException(CustomValidationException e) {
        log.error(e.getMessage());
        return new ResponseEntity<>(new ResponseDto<>(-1, e.getMessage(), e.getErrorMap()),
                                                HttpStatus.BAD_REQUEST);
    }
}
```

UserController - 수정

```
...

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @PostMapping("/join")
    public ResponseEntity<?> join(@RequestBody @Valid JoinReqDto joinReqDto,
                                  BindingResult bindingResult) {
        // 검증 로직은 AOP가 해결!

        JoinRespDto joinRespDto = userService.join(joinReqDto);
        return new ResponseEntity<>(new ResponseDto<>(1, "회원가입 성공", joinRespDto), HttpStatus.CREATED);
    }
}
```

- 검증 로직이 빠졌다!