Spring Boot 유효성 검사 v2 - AOP로 자동화

> valid.md의 두 번째 버전. AOP를 적용해서 컨트롤러마다 반복되는 `errors.hasErrors()` 체크를 자동화한다.

v1의 문제점

valid.md 방식은 **모든 컨트롤러 메서드**에 동일한 코드가 반복된다:

```
// BoardController.save()
@PostMapping("/boards/save")
public String save(@Valid BoardRequest.SaveOrUpdateDTO reqDTO, Errors errors) {
    if (errors.hasErrors()) {
        throw new Exception400(errors.getAllErrors().get(0).getDefaultMessage());
    }
    // ...
}

// BoardController.update()
@PostMapping("/boards/{id}/update")
public String update(@PathVariable("id") int id, @Valid
BoardRequest.SaveOrUpdateDTO reqDTO, Errors errors) {
    if (errors.hasErrors()) {
        throw new Exception400(errors.getAllErrors().get(0).getDefaultMessage());
    }
    // ...
}

// UserController.login()
@PostMapping("/login")
public String login(@Valid UserRequest.LoginDTO reqDTO, Errors errors) {
    if (errors.hasErrors()) {
        throw new Exception400(errors.getAllErrors().get(0).getDefaultMessage());
    }
    // ...
}

// UserController.join()
@PostMapping("/join")
public String join(@Valid UserRequest.JoinDTO reqDTO, Errors errors) {
    if (errors.hasErrors()) {
        throw new Exception400(errors.getAllErrors().get(0).getDefaultMessage());
    }
    // ...
}
```

4군데에 똑같은 3줄이 반복된다. 메서드가 늘어나면 계속 복붙해야 한다.

v2 해결: AOP로 횡단 관심사 분리

AOP란?

```
AOP (Aspect-Oriented Programming) = 관점 지향 프로그래밍

횡단 관심사(cross-cutting concern)를 분리하는 기법.
여러 메서드에 공통으로 적용되는 로직을 한 곳에서 관리한다.
```

유효성 검사 에러 처리는 전형적인 **횡단 관심사**다:

- 모든 컨트롤러에서 동일한 패턴
- 비즈니스 로직이 아닌 부가 기능
- 한 곳에서 관리하면 유지보수가 편함

구현

의존성 (build.gradle)

```
implementation 'org.springframework.boot:spring-boot-starter-validation'
implementation 'org.springframework.boot:spring-boot-starter-aop'
```

Step 1. DTO에 어노테이션 (v1과 동일)

```
// BoardRequest.java
@Data
public static class SaveOrUpdateDTO {
    @NotBlank(message = "제목을 입력해주세요")
    private String title;

    @NotBlank(message = "내용을 입력해주세요")
    private String content;
}

// UserRequest.java
@Data
public static class LoginDTO {
    @NotBlank(message = "유저네임을 입력해주세요")
    private String username;

    @NotBlank(message = "비밀번호를 입력해주세요")
    private String password;
}

@Data
public static class JoinDTO {
```

```
    @NotBlank(message = "유저네임을 입력해주세요")
    @Size(min = 3, max = 20, message = "유저네임은 3~20자로 입력해주세요")
    private String username;

    @NotBlank(message = "비밀번호를 입력해주세요")
    @Size(min = 4, max = 20, message = "비밀번호는 4~20자로 입력해주세요")
    private String password;

    @Email(message = "이메일 형식이 올바르지 않습니다")
    private String email;
}

// ReplyRequest.java
@Data
public static class SaveDTO {
    @NotNull(message = "게시글 ID가 필요합니다")
    private Integer boardId;

    @NotBlank(message = "댓글을 입력해주세요")
    private String comment;
}
```

Step 2. AOP Aspect 클래스 생성

새 파일: `_core/aop/ValidationAspect.java`

```
package com.example.boardv1._core.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;

import com.example.boardv1._core.errors.ex.Exception400;

@Aspect
@Component
public class ValidationAspect {

    // 포인트컷: 모든 컨트롤러의 모든 메서드
    @Pointcut("execution(* com.example.boardv1..*Controller.*(..))")
    public void controllerMethods() {}

    // @Before: 컨트롤러 메서드 실행 전에 가로채기
    @Before("controllerMethods()")
    public void validationCheck(JoinPoint jp) {
        // 메서드의 모든 파라미터를 순회
        for (Object arg : jp.getArgs()) {
            // Errors 타입 파라미터를 찾으면
```

```
            if (arg instanceof Errors errors) {
                // 에러가 있으면 Exception400 throw
                if (errors.hasErrors()) {
                    throw new Exception400(
                        errors.getAllErrors().get(0).getDefaultMessage()
                    );
                }
            }
        }
    }
}
```

Step 3. Controller에서 Errors만 받기 (체크 로직 삭제)

AOP가 자동으로 체크하므로 **Controller에는 `@Valid` + `Errors`만 선언**하면 된다:

```
// BoardController.java
@PostMapping("/boards/save")
public String save(@Valid BoardRequest.SaveOrUpdateDTO reqDTO, Errors errors)
throws IOException {
    // ← if (errors.hasErrors()) 체크 코드가 필요 없다! AOP가 자동 처리

    User sessionUser = (User) session.getAttribute("sessionUser");
    if (sessionUser == null)
        throw new Exception401("인증되지 않았습니다.");

    boardService.게시글쓰기(reqDTO.getTitle(), reqDTO.getContent());
    return "redirect:/";
}

@PostMapping("/boards/{id}/update")
public String update(@PathVariable("id") int id, @Valid
BoardRequest.SaveOrUpdateDTO reqDTO, Errors errors) {
    // ← AOP가 자동 처리

    User sessionUser = (User) session.getAttribute("sessionUser");
    if (sessionUser == null)
        throw new Exception401("인증되지 않았습니다.");

    boardService.게시글수정(id, reqDTO.getTitle(), reqDTO.getContent(),
sessionUser.getId());
    return "redirect:/boards/" + id;
}
```

```
// UserController.java
@PostMapping("/login")
public String login(@Valid UserRequest.LoginDTO reqDTO, Errors errors,
HttpServletResponse resp) {
    // ← AOP가 자동 처리
```

```
    User sessionUser = userService.로그인(reqDTO.getUsername(),
reqDTO.getPassword());
    session.setAttribute("sessionUser", sessionUser);

    Cookie cookie = new Cookie("username", sessionUser.getUsername());
    cookie.setHttpOnly(false);
    resp.addCookie(cookie);

    return "redirect:/";
}

@PostMapping("/join")
public String join(@Valid UserRequest.JoinDTO reqDTO, Errors errors) {
    // ← AOP가 자동 처리

    userService.회원가입(reqDTO.getUsername(), reqDTO.getPassword(),
reqDTO.getEmail());
    return "redirect:/login-form";
}
```

```
// ReplyController.java
@PostMapping("/replies/save")
public String save(@Valid ReplyRequest.SaveDTO reqDTO, Errors errors) {
    // ← AOP가 자동 처리

    User sessionUser = (User) session.getAttribute("sessionUser");
    if (sessionUser == null)
        throw new RuntimeException("인증되지 않았습니다.");

    replyService.댓글등록(reqDTO.getBoardId(), reqDTO.getComment(),
sessionUser.getId());
    return "redirect:/boards/" + reqDTO.getBoardId();
}
```

AOP 동작 원리

실행 흐름

```
[요청] POST /boards/save (title="", content="내용")
    ↓
[Spring MVC] @Valid → 유효성 검사 실행 → Errors 객체에 에러 저장
    ↓
[AOP - @Before] ValidationAspect.validationCheck() 자동 실행
    ├── jp.getArgs()로 파라미터 순회
    ├── Errors 타입 발견
    ├── errors.hasErrors() == true
    └── throw new Exception400("제목을 입력해주세요")
```

```
    ↓
[GlobalExceptionHandler] ex400() → alert + history.back()
    ↓
[화면] 알림창 → 이전 페이지
```

> **핵심**: Controller 메서드가 실행되기 **전에** AOP가 먼저 Errors를 체크한다.

AOP 핵심 어노테이션

어노테이션	설명
`@Aspect`	이 클래스가 AOP 관점(Aspect)임을 선언
`@Component`	스프링 빈으로 등록
`@Pointcut`	어디에 적용할지 (대상 메서드 지정)
`@Before`	대상 메서드 실행 **전에** 실행

Pointcut 표현식 해석

```
@Pointcut("execution(* com.example.boardv1..*Controller.*(..))")
```

```
execution(                          // 메서드 실행 시점에
  *                                 // 리턴 타입 상관없이
  com.example.boardv1..             // 이 패키지 하위 어디든
  *Controller                       // Controller로 끝나는 클래스의
  .*                                // 모든 메서드
  (..)                              // 파라미터 상관없이
)
```

JoinPoint 파라미터

```
@Before("controllerMethods()")
public void validationCheck(JoinPoint jp) {
    jp.getArgs();           // 메서드에 전달된 모든 파라미터 배열
    jp.getSignature();      // 메서드 시그니처 (이름, 리턴타입 등)
    jp.getTarget();         // 실제 대상 객체 (Controller 인스턴스)
}
```

---

v1 vs v2 비교

v1 (수동 체크)

```
// 매 메서드마다 반복
@PostMapping("/boards/save")
public String save(@Valid BoardRequest.SaveOrUpdateDTO reqDTO, Errors errors) {
    if (errors.hasErrors()) {
        throw new Exception400(errors.getAllErrors().get(0).getDefaultMessage());
    }
    // 비즈니스 로직...
}
```

v2 (AOP 자동화)

```
// 깔끔하게 비즈니스 로직만 존재
@PostMapping("/boards/save")
public String save(@Valid BoardRequest.SaveOrUpdateDTO reqDTO, Errors errors) {
    // 비즈니스 로직...
}
```

항목	v1 (수동)	v2 (AOP)
errors 체크 코드	매 메서드마다 3줄 반복	**0줄** (AOP가 자동)
새 메서드 추가 시	체크 코드 복붙 필요	`@Valid` + `Errors`만 선언
체크 로직 변경 시	모든 메서드 수정	**ValidationAspect 1곳만 수정**
실수로 체크 빼먹을 가능성	있음	**없음** (자동 적용)

변경 파일 요약

파일	변경 내용
`build.gradle`	`spring-boot-starter-validation` + `spring-boot-starter-aop` 추가
`BoardRequest.java`	DTO 필드에 `@NotBlank` 추가
`UserRequest.java`	DTO 필드에 `@NotBlank`, `@Size`, `@Email` 추가
`ReplyRequest.java`	DTO 필드에 `@NotNull`, `@NotBlank` 추가
`_core/aop/ValidationAspect.java`	**새 파일** - AOP 유효성 검사 자동화
`BoardController.java`	`@Valid` + `Errors` 파라미터 추가 (체크 코드 없음)
`UserController.java`	`@Valid` + `Errors` 파라미터 추가 (체크 코드 없음)
`ReplyController.java`	`@Valid` + `Errors` 파라미터 추가 (체크 코드 없음)

프로젝트 구조 변경

```
_core/
├── errors/
│   ├── GlobalExceptionHandler.java   # 기존 유지
│   └── ex/
│       ├── Exception400.java         # 기존 유지
│       ├── Exception401.java
│       ├── Exception403.java
│       ├── Exception404.java
│       └── Exception500.java
└── aop/
    └── ValidationAspect.java         # ← 새로 추가
```

주의사항

1. **`Errors` 파라미터 필수**: `@Valid`만 쓰고 `Errors`를 안 받으면 Spring이 바로 `MethodArgumentNotValidException`을 던진다. AOP가 가로챌 수 없으므로 반드시 `Errors`를 파라미터에 포함해야 한다.
2. **파라미터 순서**: `@Valid DTO reqDTO, Errors errors` - Errors는 @Valid 파라미터 바로 다음에 위치해야 한다.
3. **AOP 의존성**: `spring-boot-starter-aop`를 build.gradle에 추가해야 `@Aspect`가 동작한다.