Validation

유효성 검사

Validation이란

```java
@Entity
@Getter @Setter
public class Post {

    @Id @GeneratedValue
    @Column(name = "post_id")
    private Long id;

    private String title;
    private String content;
```

Validation이란

- 올바르지 않은 데이터를 걸러내고, 보안을 유지하기 위해 데이터를 검증하는 것
- Controller의 중요한 역할중 하나는 http 요청이 정상적인지 확인하는 것이다.
- 클라이언트와 서버 둘다 검증을 한다.
- 클라이언트만 할 경우 보안에 취약
- 서버만 할 경우 즉각적인 고객 사용성이 부족해진다.
- 최종적으로 서버에서 검증 구현은 필수

Validation 방법

- Spring 은 데이터 검증에 BindingResult을 사용한다.
- BindingResult는 검증 오류를 보관하는 객체이다.
- BindingResult에 검증 오류를 보관하는 방법은 총 3가지 있다.

1. @ModelAttribute 객체에 타입 오류 등 바인딩이 실패할 경우
2. 개발자가 직접 넣어주는 경우
3. Validatior를 사용하는 경우

Validation 방법

```
@PostMapping("/add")
public String createPostV2(@ModelAttribute Post post, BindingResult bindingResult){

    if (!StringUtils.hasText(post.getTitle())) {
        bindingResult.addError(new FieldError( objectName: "post",  field: "title",  defaultMessage: "제목 입력은 필수 입니
    }
    if (post.getTitle().length() > 20) {
        bindingResult.addError(new FieldError( objectName: "post",  field: "title",  defaultMessage: "제목은 20자까지 허용
    }
    if (bindingResult.hasErrors()){
        return "createPost";
    }
    return "home";
}
```

Validation 방법

- 검증 과정이 많고 길어 질수록 Controller가 지저분해지고 유지보수도 어려워짐.
- 이를 위해 Spring에서 지원하는 것이 Validator Interface이다

```java
@Component
public class PostValidator implements Validator {
    @Override
    public boolean supports(Class<?> clazz) { return Post.class.equals(clazz); }

    @Override
    public void validate(Object target, Errors errors) {

    }
}
```

Validation 방법

```java
@Component
public class PostValidator implements Validator {
    @Override
    public boolean supports(Class<?> clazz) { return Post.class.equals(clazz); }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmpty(errors, field: "title", errorCode: "required");
        Post post = (Post) target;
        if (post.getTitle().length() > 20){
            errors.rejectValue( field: "title", errorCode: "too.long");
        }
        if (post.getContent().length() > 200){
            errors.rejectValue( field: "content", errorCode: "too.long");
        }
    }
```

Validation 방법

```
@PostMapping("/validator")
public String createPostV3(@ModelAttribute Post post, BindingResult bindingResult){
    if (postValidator.supports(post.getClass())){
        postValidator.validate(post, bindingResult);
    }
    if (bindingResult.hasErrors()){
        return "createPost";
    }
    return "home";
}
```

Bean Validation

Bean Validation

- Bean Validation은 클래스 "필드"에 특정 annotation을 적용하여

  필드가 갖는 제약 조건을 정의

- Bean Validation은 특정 구현체가 아닌 기술 표준.
- 구현한 기술중에 일반적으로 사용하는 구현체는 하이버네이트 Validator

Bean Validation

- 사용 예)

```java
@Getter @Setter
@NoArgsConstructor
@AllArgsConstructor
public class PostRequestDto {

    @NotNull
    @Size(max = 20)
    private String title;

    @NotEmpty
    @NotBlank
    private String content;
}
```

Bean Validation

- 글로벌 Validator가 적용되어 있기 때문에, @Valid , @Validated 만 적용하면 된다.

- 검증 오류가 발생하면, FieldError , ObjectError 를 생성해서 BindingResult 에 담아준다.

```
@PostMapping
public ResponseEntity<PostResponseDto> createPost(@Valid @RequestBody PostRequestDto requestDto){
    return postService.createPost(requestDto);
}
```

ControllerAdvice

ControllerAdvice

- 모든 컨트롤러에 대해 전역으로 발생하는 예외를 처리해줄 수 있다.

```java
@RestControllerAdvice
public class ApiControllerAdvice {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>> handleValidationExceptions(MethodArgumentNotValidException ex){
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors()
                .forEach(c -> errors.put(((FieldError) c).getField(), c.getDefaultMessage()));
        return ResponseEntity.badRequest().body(errors);
    }
}
```

ControllerAdvice