Spring MVC2 09 [검증 - Bean Validation]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.22

Bean Validation

Bean Validation ? 특정 필드에 대한 검증 로직은 대부분 빈 값인지, 아니면 범위를 가지는 일반적인 로직이다. → 표준화 시킬 수 있다.

애노테이션과 인터페이스의 모음인 표준 기술이다.

하이버네이트 **Validator** 관련 링크

공식 사이트

http://hibernate.org/validator/

공식 메뉴얼

https://docs.jboss.org/hibernate/validator/6.2/reference/en-US/html_single/

검증 애노테이션 모음

https://docs.jboss.org/hibernate/validator/6.2/reference/en-US/html_single/#validator-defineconstraints-spec

Bean Validation 사용

build.gradle 에 의존 추가

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

기본적인 애노테이션

```
@NotBlank  → 빈칸, 공백 금지
private String itemName;

@NotNull   → null 금지
@Range(min = 1000, max = 1000000)→ 범위
private Integer price;

@NotNull
@Max(9999)
private Integer quantity;
```

Validate 해서 출력해보기

```
@Test
void beanValidation(){
   ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
   Validator validator = factory.getValidator();

   Item item = new Item();
   item.setItemName(" "); //공백
   item.setPrice(0);
   item.setQuantity(10000);

   Set<ConstraintViolation<Item>> validate = validator.validate(item);
   for (ConstraintViolation<Item> itemConstraintViolation : validate) {
       System.out.println("itemConstraintViolation = " + itemConstraintViolation);
       System.out.println("itemConstraintViolation.getMessage() = " +
itemConstraintViolation.getMessage());
   }
}
```

실제 적용

```
@PostMapping("/add")
public String addItem(@Validated @ModelAttribute Item item, BindingResult
bindingResult, RedirectAttributes redirectAttributes, Model model) {

   if(bindingResult.hasErrors()){
```

```
        log.info("bindingResult = {}", bindingResult);
        return "validation/v3/addForm";
    }

    //성공 로직
    Item savedItem = itemRepository.save(item);
    redirectAttributes.addAttribute("itemId", savedItem.getId());
    redirectAttributes.addAttribute("status", true);
    return "redirect:/validation/v3/items/{itemId}";
}
```

@Validated를 적용하면 동작

스프링 부트가 라이브러리에 validation을 확인하면 LocalValidatorFactoryBean 을

글로벌 **Validator**를 등록해서 **@Validated** 애노테이션에 따른 검증을 한다**.**

다른 Validator를 Override를 통해 글로벌로 등록해놓으면 동작하지 않으니 주의한다.

스프링전용 @Validated , 자바 표출 검증 @Valid

검증 순서

순서 **1. @ModelAttribute** 각 필드의 타입 변화 시도

1) 성공하면 다음
2) 실패하면 typeMismatch로 FieldError 추가

순서 **2 Validator** 적용

Bean Validation 에러 코드

애노테이션 이름으로 오류 코드를 만들어 준다.

즉 메시지를 등록하면 변경할 수 있다.

```
#Bean Validation 추가
NotBlank={0} 공백X
Range={0}, {2} ~ {1} 허용
Max={0}, 최대 {1}
```

필드 오류가 아닌 **ObjectError** 일 경우**?**

```
@ScriptAssert(lang = "javascript", script = "_this.price * _this.quantity >=
10000",message = "총 가격이 10000원 이상이어야 합니다.")
```

클래스에 Object에 대한 검증식을 넣어준다.

하지만 이렇게 사용할 경우는 복잡해질 수 잇기 때문에

자바 코드로써 ObjectError를 구현하는 것이 더 편리하게 사용가능

```
//특정 필드가 아닌 복합 룰 검증
if(item.getPrice() != null && item.getQuantity() != null){
   int resultPrice = item.getPrice() * item.getQuantity();
   if(resultPrice < 10000){
       bindingResult.reject("totalPriceMin",new Object[]{10000, resultPrice},
null); → ObjectError 추가
   }
}
```

Bean Validation의 한계점

데이터를 등록할때와 수정할 때의 요구사항이 달라질 수 있다.

만약,

수정 요구사항에 Id가 필수 사항이라고 지정하고 애노테이션을 설정하게된다면?

[HTTP 요청을 악의적으로 임의 수정해서 변경할 수 있기 때문에 id 검증은 수정시에 서버사이드에서 해줘야 한다.]

```
   @NotNull //수정 요구사항 추가된 사항
   private Long id;
```

NotNull조건을 주게된다면 수정페이지가 아닌,

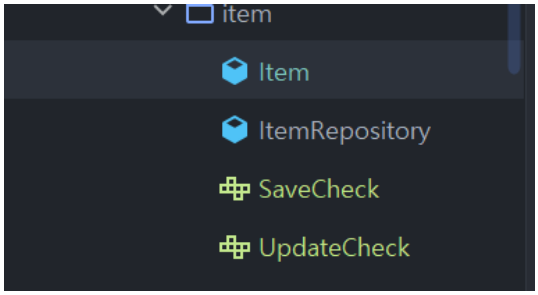
등록시 itemId 값을 직접 입력하지 않으므로 Side Effect로 에러가 발생할 수 있다.

만약 등록조건과 수정조건이 충돌하게 된다. → groups 사용한다.

문제점을 해결 하기 위해 두가지 방법을 사용 할 수 있다.

1. **BeanValidation** 의 **group** 기능사용
2. **Item**을 사용하지 않고 분리 된 **DTO** 사용

BeanValidation - groups



빈 인터페이스 생성

```
public class Item {

   @NotNull(groups = UpdateCheck.class) //수정 요구사항 추가된 사항
   private Long id;

   @NotBlank(message = " 공백 X ", groups = {SaveCheck.class, UpdateCheck.class})
   private String itemName;

   @NotNull( groups = {SaveCheck.class, UpdateCheck.class})
   @Range(min = 1000, max = 1000000,  groups = {SaveCheck.class,
UpdateCheck.class})
   private Integer price;

   @NotNull( groups = {SaveCheck.class, UpdateCheck.class})
   @Max(value = 9999,  groups = SaveCheck.class)
   private Integer quantity;
```

group을 지정해준다.

등록시 - SaveCheck 인터페이스

```
@PostMapping("/add")
public String addItem(@Validated(SaveCheck.class) @ModelAttribute Item item,
BindingResult bindingResult, RedirectAttributes redirectAttributes, Model model) {
```

수정시 - UpdateCheck 인터페이스를 대상으로

```
@PostMapping("/{itemId}/edit")
public String edit2(@PathVariable Long itemId, @Validated(UpdateCheck.class)
@ModelAttribute Item item, BindingResult bindingResult) {
```

→ groups 기능을 사용하기 위해선 @Valid 가아닌 @Validated를 사용해야 한다.

하지만 복잡도가 올라가서 많이 사용하지 않는다.

실무에서는 주로 등록용 객체 수정용 객체를 따로 사용한다. **(DTO 분리)**

– Entity로 관리를 하지만 Entity를 이동시키는것은 좋은 설계가 아니다.

Form 전송 객체 분리

실무에서는 'groups' 를 잘 사용안함.

예를 들면 Item 도메인 객체가 등록과 수정 폼에서 전달하는 데이터가 딱 맞지않기 때문임.

Form→ ItemSaveForm → Controller → Item 생성 **→ Repository**

의 방식으로 동작시키는 것.

ItemSaveForm 을 Item객체를 생성하는 변환 과정이 추가된다.

```
@Data
public class ItemSaveForm {

   @NotBlank
   private String itemName;

   @NotNull
   @Range(min = 1000, max = 1000000)
```

```
    private Integer price;

    @NotNull
    @Max(value = 9999)
    private Integer quantity;
}
```

→ form 전송용 객체를 생성

```
@PostMapping("/add")
public String addItem(@Validated @ModelAttribute("item") ItemSaveForm form,
BindingResult bindingResult, RedirectAttributes redirectAttributes, Model model) {

    //특정 필드가 아닌 복합 룰 검증
    if(form.getPrice() != null && form.getQuantity() != null){
        int resultPrice = form.getPrice() * form.getQuantity();
        if(resultPrice < 10000){
            bindingResult.reject("totalPriceMin",new Object[]{10000, resultPrice},
null);
        }
    }

    if(bindingResult.hasErrors()){
        log.info("bindingResult = {}", bindingResult);
        return "validation/v4/addForm";
    }

    Item item = new Item();
    item.setItemName(form.getItemName());
    item.setPrice(form.getPrice());
    item.setQuantity(form.getQuantity());

    //성공 로직
    Item savedItem = itemRepository.save(item);
```

form 전송을 위한 객체를 사용하고 실제 Repository에 담을때는 Item 객체로 전환을 시켜줘야 한다.

추가로 **Spring**이 **model**에 바인딩 하는 방식을 이용한 방법

→ 추천 안하는 방식 도메인이 파라미터로 들어가 있다면 어떤 사이드 이펙트를 줄지모르기 때문에 또 지저분해지고 명확하지 않다. 다만 이렇게도 동작한다는 것을 기억하기 위해

Spring이 모델에 바인딩 하는 방식은 폼데이터에 name에 맞는 맴버를 찾아서 set 해주는 것이다.

```
<form action="" method="post">

      <div>
          <label for="itemName">상품명</label>
          <input type="text" id="itemName" class="form-control" placeholder="이름을
입력하세요" name="itemName" value="">
     </div>
```

→일 경우 프로퍼티 접근법 setItemName 을 찾아서 넣어주는 형식

그러므로 ItemSaveForm객체를 Item과 동시에 받아도 Item에 해당하는 부분을 채워서 Item 객체를 생성하는 효과.

```
@ModelAttribute("item2") Item item
@Validated @ModelAttribute("item") ItemSaveForm form
```

그리고 Validated를 ItemSaveForm 에 걸어주고 자동 바인딩 될때의 이름이 겹치는 것을 item2로 변경해서 사용

PRG 방식을 사용해서 redirect 할 것이기 때문에 추가로 model에 엔티티가 바인딩 되는 문제는 다시 폼으로 돌아갈 때만 생각하면 된다. – 사실 어떠한 문제가 추가로 발생할지 예상하기 힘들어서 추천은 하지않는다.

```
@PostMapping("/add")
public String addItem2(@ModelAttribute("item2") Item item, @Validated
@ModelAttribute("item") ItemSaveForm form, BindingResult bindingResult,
RedirectAttributes redirectAttributes, Model model) {

   //특정 필드가 아닌 복합 룰 검증
   if(form.getPrice() != null && form.getQuantity() != null){
       int resultPrice = form.getPrice() * form.getQuantity();
       if(resultPrice < 10000){
           bindingResult.reject("totalPriceMin",new Object[]{10000, resultPrice},
null);
       }
   }

   if(bindingResult.hasErrors()){
       log.info("bindingResult = {}", bindingResult);
```

```
        return "validation/v4/addForm";
    }

    //성공 로직
    Item savedItem = itemRepository.save(item);
    redirectAttributes.addAttribute("itemId", savedItem.getId());
    redirectAttributes.addAttribute("status", true);
    return "redirect:/validation/v4/items/{itemId}";
}
```

'Repository에 담을때는 Item 객체로 전환을 시켜줘야 한다'

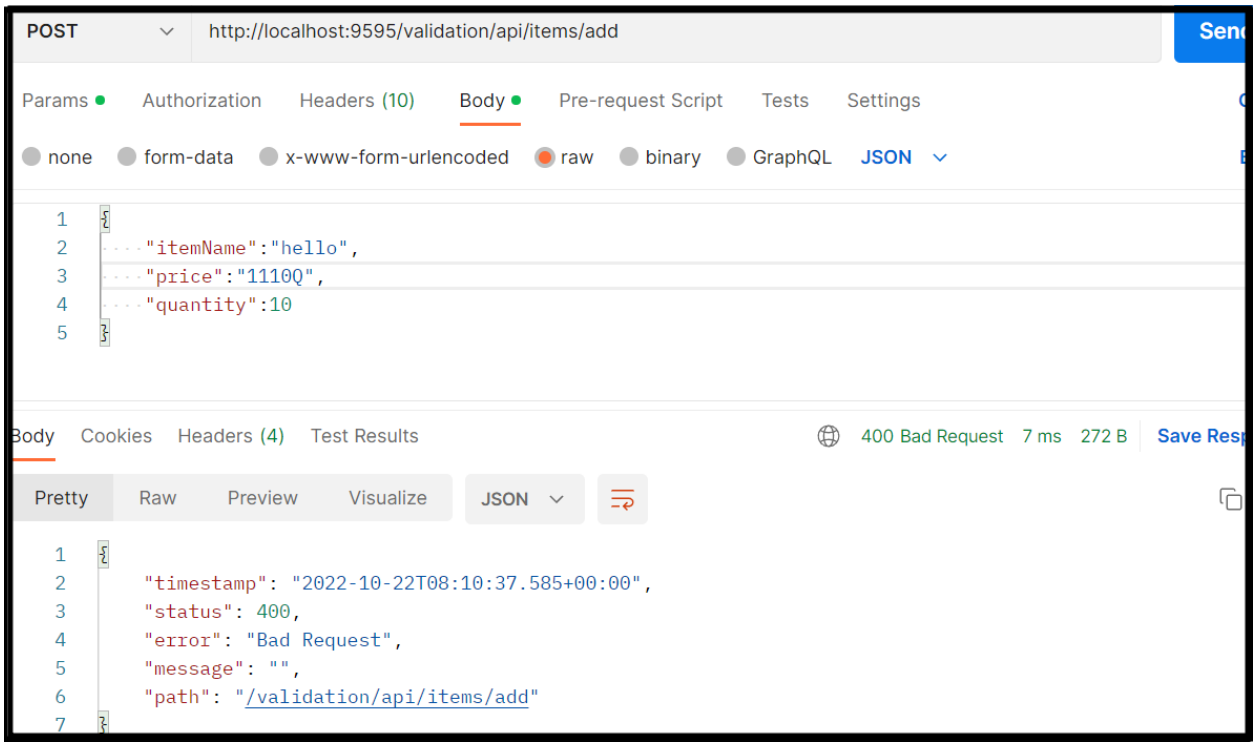
하는 로직 부분을 넣지 않아도 동작하는 것을 알 수 있다.

Bean Validation - HTTP 메시지 컨버터

```
@Slf4j
@RestController
@RequestMapping("/validation/api/items")
public class ValidationItemApiController {

    @PostMapping("/add")
    public Object addItem(@RequestBody @Validated ItemSaveForm form, BindingResult
bindingResult){
        log.info("API 컨트롤러 호출");

        if(bindingResult.hasErrors()){
            log.info("검증 오류 발생 errors = {}", bindingResult);
            return bindingResult.getAllErrors();
        }

        log.info(" 성공 로직 실행");
        return form;
    }
}
```

예외가 발생하는데 컨트롤러 자체가 호출이 안된다

JSON이 객체를 생성하지 못해서 발생한다.



API 사용시 **3**가지의 결과가 나타난다

성공 요청**:** 성공

검증 오류 요청**:** JSON을 객체로 생성하는 것은 성공했고, 검증에서 실패함 → 검증 로직이 작동

실패 요청**:** JSON을 객체로 생성하는 것 자체가 실패함 → 컨트롤러 호출이 실패 validation조차 실행 X

@ModelAttribute vs @RequestBody

요청 파라미터에 **ModelAttribute**는 쿼리스트링이나 form 는 필드 단위로 적용되서 오류가 발생해도 나머지 필드는 정상 처리 할 수있다. 필드 단위로 검증을 거친다

RequestBody는 HttpMessageConverter 단계에서 JSON 데이터를 객체로 변경하는 과정을 거치는데 이때 제대로 객체를 변경하지 못하게 되면 예외가 발생하게 된다.

HttpMessageConverter 대한 예외 처리는 예외 처리 학습 부분을 참고