Spring MVC2 07 [스프링의 검증 오류 처리 1]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.21

스프링의 검증 오류 처리 방법

스프링의 BindingResult가 오류 처리의 핵심이다.

기본적인 사용 [BindingResult]

-BindingResult 파라미터위치가 중요.

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

   if (!StringUtils.hasText(item.getItemName())){
      bindingResult.addError(new FieldError("item", "itemName", "상품 이름은 필수
입니다."));
   }
   if(item.getPrice() == null || item.getPrice() < 1000 || item.getPrice() >
1000000 ){
       bindingResult.addError(new FieldError("item", "price", " 가격은 1,000원 ~
1,000,000원 사이입니다."));
   }
   if(item.getQuantity() == null || item.getQuantity() >= 9999){
       bindingResult.addError(new FieldError("item", "quantity", "수량은 최대 9,999
까지 허용합니다."));
   }

   //특정 필드가 아닌 복합 룰 검증
   if(item.getPrice() != null && item.getQuantity() != null){
       int resultPrice = item.getPrice() * item.getQuantity();
       if(resultPrice < 10000){
           bindingResult.addError(new ObjectError("item", "가격 * 수량의 합은
10,000원 이상이어야 합니다. 현재 값 = " + resultPrice));
       }
   }
```

```
   //검증에 실패하면 다시 입력 폼으로
   if(bindingResult.hasErrors()){
       log.info("bindingResult = {}", bindingResult);
       return "validation/v2/addForm";

---
. . .
```

주의점

@ModelAttribute Item item, BindingResult bindingResult 처럼

담긴 item에 대한 오류가 바인딩 될 것이기 때문에 바로 뒤에 와야 한다.

타임리프에서 표출시

이전 일반적인 Map으로 파라미터를 넘길때

```
<div th:if="${errors?.containsKey('globalError')}">
   <p class="field-error" th:text="${errors['globalError']}">전체 오류 메시지</p>
</div>

<div>
   <label for="itemName" th:text="#{label.item.itemName}">상품명</label>
   <input type="text" id="itemName" th:field="*{itemName}"
          th:class="${errors?.containsKey('itemName')} ? 'form-control field-error'
: 'form-control'"
          class="form-control" placeholder="이름을 입력하세요">
<div class="field-error" th:if="${errors?.containsKey('itemName')}"
th:text="${errors['itemName']}">
       상품명 오류
   </div>
</div>
```

BindingResult를 사용해서 표현할 때

```
<div th:if="${#fields.hasGlobalErrors()}">
   <p class="field-error" th:each=" err : ${#fields.globalErrors()}"
      th:text="${err}">글로벌 에러 오류 메시지</p>
</div>
```

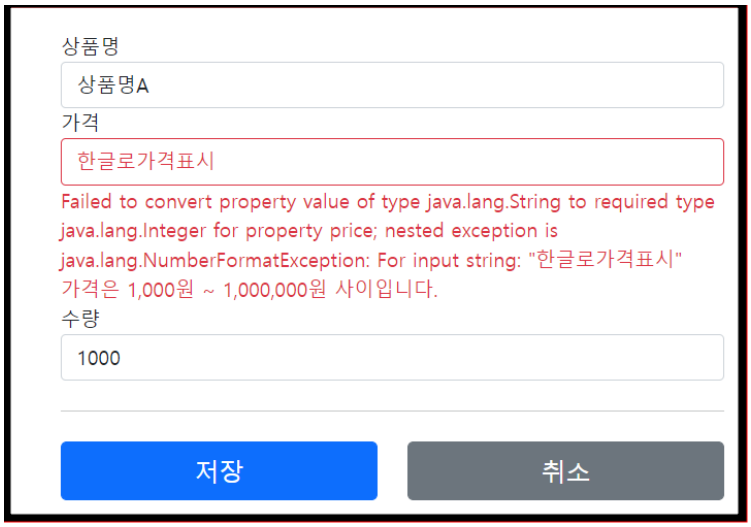
```
<div>
   <label for="itemName" th:text="#{label.item.itemName}">상품명</label>
   <input type="text" id="itemName" th:field="*{itemName}"
th:errorclass="field-error"
          class="form-control" placeholder="이름을 입력하세요">
   <div class="field-error" th:errors="*{itemName}">
       상품명 오류
   </div>
</div>
```

th:field="*{}" 안에 바인딩 되어 있는 에러까지 확인 해서 위와 같은 로직으로 동작한다.

BindingResult 사용

BindingResult를 사용할 시에는 타입이 맞지 않아도 컨트롤러가 호출 된다.

즉, 일단 호출이 된다.



400오류 페이지가 아닌 오류 정보를 담아서 binding한다.

```
bindingResult.addError(new FieldError("item", "price",  ....));
위치에 담아서 보내주는 것.
```

스프링의 오류를 처리하는 법

1. 객체타입 오류 등으로 바인딩이 실패하는 경우 FieldError를 생성한다

2. 사용자가 직접 FieldError등을 생성한다.
3. Validator를 사용한다.

⇒ 즉 바인딩이 실패하는 오류 , 비즈니스 검증 부분에서 실패하는 오류 2가지를 다루는 것.

위와 같은 방식으로 사용했을 시 데이터 유지가 안된다. ⇒ **error**를 더 확인해야한다.

기존 값을 유지시키기

```
bindingResult.addError(new FieldError("item", "price", item.getPrice(), false,
null, null, " 가격은 1,000원 ~ 1,000,000원 사이입니다."));
```

FieldError의 생성자에 추가 파라미터를 넣어서 넘겨준다.

FieldError의 생성자는 2가지가 있다.

```
public FieldError(String objectName, String field, String defaultMessage);
```

```
 public FieldError(String objectName, String field, @Nullable Object
rejectedValue, boolean bindingFailure, @Nullable String[] codes, @Nullable
Object[] arguments, @Nullable String defaultMessage)
```

rejectedValue : 사용자가 입력한 값(거절된 값)

bindingFailure : 타입 오류 같은 바인딩 실패인지, 검증 실패인지 구분 값

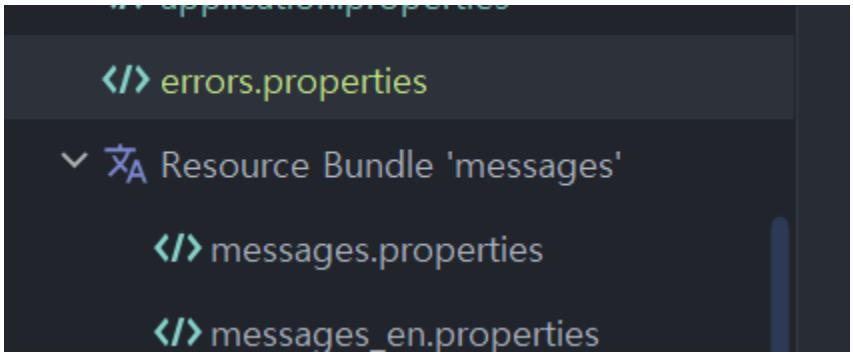
codes : 메시지 코드

arguments : 메시지에서 사용하는 인자

오류 코드와 메시지 처리

errors.properties를 생성하고 application.properties에 errors.properties 메시지를 사용하기 위해 messages옵션에 추가해 준다.

```
spring.messages.basename=messages, errors
```




```
required.item.itemName=상품 이름은 필수입니다.
range.item.price=가격은 {0} ~ {1} 까지 허용합니다.
max.item.quantity=수량은 최대 {0} 까지 허용합니다.
totalPriceMin=가격 * 수량의 합은 {0}원 이상이어야 합니다. 현재 값 = {1}
```

일때,

```
bindingResult.addError(new FieldError("item", "itemName", item.getItemName(),
false, new String[]{"required.item.itemName"}, null ,null));
```

결과 → '상품 이름은 필수 입니다.' 가 바인딩 된다.

```
bindingResult.addError(new FieldError("item", "price", item.getPrice(), false, new
String[]{"range.item.price"}, new Object[]{1000, 1000000}, null));
```

결과 → '가격은 1,000 ~ 1,000,000 까지 허용합니다.' 가 바인딩 된다.

메시지 코드 사용

String 배열 형 으로 순차적으로 찾고 없다면 defaultMessage를 사용.

오류 코드 메시지 자동화 시키기?

FieldError ObjectError가 사용하기 너무 복잡하다.

BindingResult는 target 바로 다음에 오기 때문에 이미 타겟을 알고 있다.

그러므로 rejectValue reject를 사용하면 조금더 편리하게 사용할 수 있다.

```
if (!StringUtils.hasText(item.getItemName())){
   bindingResult.rejectValue("itemName", "required");
}
if(item.getPrice() == null || item.getPrice() < 1000 || item.getPrice() > 1000000
){
   bindingResult.rejectValue("price","range", new Object[]{1000, 1000000},null);
}
if(item.getQuantity() == null || item.getQuantity() >= 9999){
   bindingResult.rejectValue("quantity", "max", new Object[]{9999}, null);
}

//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);
   }
}
```

동작 규칙 → 메시지에서

지정된 메시지 "메시지.타겟.필드" 형식으로 자동 매핑 규칙

축약된 오류 코드

FieldError() 를 직접 다룰 때는 ⇒ range.item.price 와 같이 모두 입력했다

rejectValue() 를 사용 ⇒ range 로 간단하게 입력했다.

이 부분을 이해하려면 **MessageCodesResolver** 를 이해해야 한다**.**

오류 코드와 메시지 상세

오류 코드를 자세히 만들거나 단순히 만들 수 있다.

단순하게 만들면 범용성이 좋고 자세히 만들면 정확한 정보를 전달할 수 있지만 범용성이 떨어진다.

즉**,** 범용성을 사용하다가 자세한 내용이 필요하면 메시지 단계를 두어 단계적으로 사용하는 것이 좋다**.**

error.properties 메시지에서 단계적으로 쓰기 위해 생성을 하고

```
required=필수 값 입니다.
required.item.itemName=상품 이름은 필수입니다.
```

```
bindingResult.rejectValue("itemName", "required");
```

를 rejectValue로 사용하게 되면 더욱 세밀한 required.item.itemName을 호출해준다.

⇒스프링이 **MessageCodesResolver** 로 기능을 지원해 준다**.**