Spring MVC2 08 [스프링의 검증 오류 처리 2]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.21

MessageCodeResolver

-Object 시

```
MessageCodesResolver codesResolver = new DefaultMessageCodesResolver();
String[] messageCodes = codesResolver.resolveMessageCodes("required", "item");
for (String messageCode : messageCodes) {
   System.out.println("messageCode = " + messageCode);
}
```

```
messageCode = required.item
messageCode = required
```

-Field

```
MessageCodesResolver codesResolver = new DefaultMessageCodesResolver();
String[] messageCodes = codesResolver.resolveMessageCodes("required", "item",
"itemName", String.class);
for (String messageCode : messageCodes) {
   System.out.println("messageCode = " + messageCode);
}
```

```
messageCode = required.item.itemName
messageCode = required.itemName
messageCode = required.java.lang.String
messageCode = required
```

오류 메시지 순서가 담긴 배열을 리턴해준다.

오류 코드 관리 전략

구체적인 것에서 덜 구체적인 것으로!

ValidationUtils 사용

```
if (!StringUtils.hasText(item.getItemName())){
   bindingResult.rejectValue("itemName", "required");
}
```

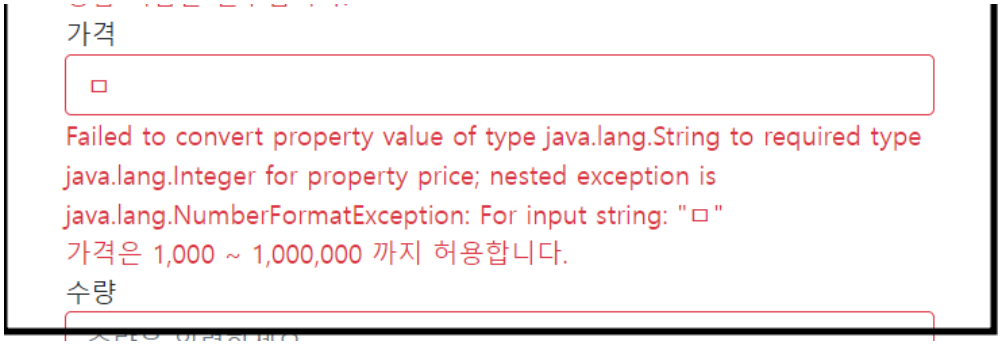
사용전

```
ValidationUtils.rejectIfEmptyOrWhitespace(bindingResult, "itemName", "required");
```

사용후

직접 오류 메시지 설정 (타입 오류등)

스프링이 직접 만든 오류를 처리할 때




```
Field error in object 'item' on field 'price': rejected value [ㅁ]; codes
[typeMismatch.item.price,typeMismatch.price,typeMismatch.java.lang.Integer,
typeMismatch]; arguments
[org.springframework.context.support.DefaultMessageSourceResolvable: codes
[item.price,price]; arguments []; default message [price]]; default message
[Failed to convert property value of type 'java.lang.String' to required
type 'java.lang.Integer' for property 'price'; nested exception is
java.lang.NumberFormatException: For input string: "ㅁ"]
```

default 메시지를 수정하고 싶다?

`typeMismatch.item.price` 를 넣어주면 된다.

errors.properties 를 수정 (추가 해준다)

```
#추가
typeMismatch.java.lang.Integer=숫자를 입력해주세요.
typeMismatch=타입오류입니다.
```

Validator

검증기사용

```
@Component
public class ItemValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return Item.class.isAssignableFrom(clazz);
        //item == clazz 본인과 자식이 지원 되는지 확인
        //item == subItem
    }

    @Override
    public void validate(Object target, Errors errors) {
        //target은 들어오게 되는 Object
        Item item = (Item) target;
        //messageResolver 를 위한 오류 코드를 사용한다.
        if (!StringUtils.hasText(item.getItemName())){
            errors.rejectValue("itemName", "required");
        }
. . . 검증 로직 . . .

    }
```

사용부에서 Component 스캔의 대상이 되게 한 후에

의존 주입 받아서 사용 하면 된다.

Validator 사용이유 **?**

장점**!?** **:** 체계적으로 검증 기능을 도입하기 위해서 사용

WebDataBinder를 통해서 사용한다

```
@InitBinder
public void init(WebDataBinder dataBinder){
   dataBinder.addValidators(itemValidator);
}
```

@InitBinder, @Validated 를 사용해서 편리하게 사용한다.

@Validated : 검증기를 실행 하라는 애노테이션 , @Valid 도 사용가능(그래들 의존추가 필요)

```
@PostMapping("/add")
public String addItemV6(@Validated @ModelAttribute Item item, BindingResult
bindingResult, RedirectAttributes redirectAttributes, Model model) {

   if(bindingResult.hasErrors()){
       log.info("bindingResult = {}", bindingResult);
       return "validation/v2/addForm";
   }
//성공로직
}
```

를 사용하면 `supports 클래스에 맞는 Validator를 찾아준다.`

에 따라서 동작을 한다.

-글로벌 적용도 가능하다.

```
@SpringBootApplication
public class ItemServiceApplication implements WebMvcConfigurer {
  public static void main(String[] args) {
     SpringApplication.run(ItemServiceApplication.class, args);
  }
  @Override
  public Validator getValidator() {
     return new ItemValidator();
  }
}
```