# Spring MVC2 06 [ 검증 ( 요청 검증 ) ]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.20

## 검증 Validation

컨트롤러의 중요한 역할 중 하나는 **HTTP** 요청이 정상인지 확인하는 검증하는 것

클라이언트 검증 : 클라이언트 검증은 조작이 가능하므로 보안에 취약하다.

서버 검증 : 서버만으로 검증하면 즉각적인 고객 사용성이 떨어진다. → 입력한게 날아감

즉, 적절하게 두개를 사용하고 최종적으로 서버에서 검증, **API** 시 검증 오류를 응답에 제대로 넘겨줘야 한다.

### 기초적인 검증하기

```
@PostMapping("/add")
public String addItem(@ModelAttribute Item item, RedirectAttributes
redirectAttributes, Model model) {
   //모델에 검증 오류 결과 객체가 필요하다
   Map<String, String> errors = new HashMap<>();
   //검증 로직
   if (!StringUtils.hasText(item.getItemName())){
      errors.put("itemName", "상품 이름은 필수입니다.");
   }
   if(item.getPrice() == null || item.getPrice() < 1000 || item.getPrice() >
1000000 ){
       errors.put("price", "가격은 1,000원에서 1,000,000원 까지 허용합니다.");
   }
   if(item.getQuantity() == null || item.getQuantity() >= 9999){
       errors.put("quantity", "수량은 최대 9,999 까지 허용합니다.");
   }

   //특정 필드가 아닌 복합 룰 검증 해보기
   if(item.getPrice() != null && item.getQuantity() != null){
```

```
        int resultPrice = item.getPrice() * item.getQuantity();
        if(resultPrice < 10000){
            errors.put("globalError", "가격 * 수량의 합은 10,000원 이상이어야 합니다.
현재 값 = " + resultPrice);
        }
    }
    //검증에 실패하면 다시 입력 폼으로
    if(isError(errors)){
        log.info("errors = {}", errors);
        model.addAttribute("errors",errors);
        return "validation/v1/addForm";
    }

    //성공 로직
    Item savedItem = itemRepository.save(item);
    redirectAttributes.addAttribute("itemId", savedItem.getId());
    redirectAttributes.addAttribute("status", true);
    return "redirect:/validation/v1/items/{itemId}";
}
```

간단한 에러 발생시켜 확인

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

클래스 부분도 적용시키는 로직

```
<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>
```

th:class="${errors?.containsKey('itemName')} ? 'form-control field-error' : 'form-control'"

혹은

```
th:classappend="${errors?.containsKey('itemName')} ? 'field-error' : _ "
```

식으로 추가할 수도 있다. ( _ ) No-Operation을 사용.

**errors?** 는 **errors** 값이 **null** 이라면 진행하지 않는다**.** 스프링 **EL**이 제공하는 문법

정리

⇒ 검증오류가 발생시 입력 폼을 다시 보여주는 방식

⇒ 타입오류는 처리가 되지 않았다. 타입 오류가 발생하시면 400 bad request를 발생 시키며 컨트롤러 접근조차 되지 않으므로 검증이 돌지도 않고 죽어버린다.

바인딩이 안되기 때문에 바인딩도 바인딩 조차 사라짐 (어째서 에러가 발생한지 알 수 없다.)

---