

Kỹ thuật bắt Exception trong PE HSF302 mình sẽ dùng kĩ thuật của **@BindingResult**

**@BindingResult** sẽ giúp chúng ta bắt được các Exception của các Entity và cả các Exception Global mình thêm vào.

Trong đây mình sẽ hoàn toàn lấy ví dụ về bài **ngoctrinh-v3binding**

**Source code chưa chỉnh sửa (Github của thầy Hoàng): Mọi người copy đường dẫn nhé** [**https://github.com/doit-now/spring-craftsman/blob/main/hands-on\_mvc/253\_ngoctrinh-coffee-v3binding.zip**](https://github.com/doit-now/spring-craftsman/blob/main/hands-on_mvc/253_ngoctrinh-coffee-v3binding.zip)

**Source code sau chỉnh sửa:** [**https://github.com/uongthanhtu/SpringbootWithMe/blob/main/HSF/Validate-Exception/ngoctrinh-v3binding-v1.zip**](https://github.com/uongthanhtu/SpringbootWithMe/blob/main/HSF/Validate-Exception/ngoctrinh-v3binding-v1.zip)

**Source code của cả 2 trường hợp (gốc và mới):** [**https://drive.google.com/drive/folders/1Znh9-tElyUJU8pTGcbrPMNaoftEcKRWY?usp=sharing**](https://drive.google.com/drive/folders/1Znh9-tElyUJU8pTGcbrPMNaoftEcKRWY?usp=sharing)

**Lưu ý: Mọi người nên tải source code mới về và theo dõi theo từng giai đoạn để hiểu cái luồng tại sao nó lại chạy như vậy trước rồi hãng code lại để nó có cái nhìn tổng quan nhất nhé.**

# Các lưu ý chính khi dùng kỹ thuật @BindingResult

1. Phương pháp này bắt buộc các bạn phải dùng **binding object** trong lúc xử lý form và gửi form
2. Trong các Controller xử lý các method post của form đó bắt buộc phải có **@Valid** ví dụ: **@Valid @ModelAttribute("selectedOne")**

**@Valid @ModelAttribute("selectedOne")**

1. Luôn đặt **@BindingResult** ngay sau tham số được validate (Dễ hiểu hơn là sau cái **@Valid @ModelAttribute("selectedOne")**. (phía dưới sẽ có ví dụ)

@Valid @ModelAttribute("selectedOne") Product product, BindingResult bindingResult

1. Trong form Thymeleaf luôn phải sử dụng **th:object** trong kĩ thuật **binding object** để nó bắt các exception khi mình khai báo trong Entity của object mình khai báo đó.

# II. Các bước setup để sử dụng được cách bắt Exception này

**(Lưu ý mình sẽ lấy ví dụ bắt các Exception của Product)**

## **Trong pom.xml thêm dependency**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

Dependency này sẽ giúp ta sử dụng được các @Annotation về Exception như: @Size, @Max, @Min,...

## **Trong Product Entity:**

### **Trường hợp 1: Không cho để trống productId khi update:**

@Id

@Column(name = "Id", nullable = false)

@NotBlank(message = "Id is required")

### **Trường hợp 2: Không cho để trống và nhập láo Name** với các ký tự đặc biệt (mọi người có thể lên [Java Regular Expressions](https://www.w3schools.com/java/java_regex.asp) để học các regex căn bản của JAVA)

@Column(name = "Name", columnDefinition = "NVARCHAR(50)", nullable = false)

@Pattern(regexp = "^[\\p{L}\\p{M}0-9 \_-]+$", message = "Do not enter special characters")

@NotBlank(message = "Name is required")

private String name;

### **Trường hợp 3:** Ràng buộc khoảng giá trị của **quantity và price và không cho để trống**

**Ví dụ này mình để:1 <= Quantity <= 100 và 20 000 <= Price <= 10 000 000**

**@Column(name = "Quantity", nullable = false)**

**@NotNull(message = "Quantity is required")**

**@Max(value = 100, message = " The quantity need >= 1 and <= 100")**

**@Min(value = 1, message = " The quantity need >= 1 and <= 100")**

**private int quantity;**

**@Column(name = "Price", nullable = false)**

**@NotNull(message = "Quantity is required")**

**@Max(value = 10000000, message = " The price need >= 20 000 and <= 10 000 000")**

**@Min(value = 20000, message = " The price need >= 20 000 and <= 10 000 000")**

**private double price;**

### Đoạn code đầy đủ của Product Entity:

**@Id**

**@Column(name = "Id", nullable = false)**

**@NotBlank(message = "Id is required")**

**private String id;**

**@Column(name = "Name", columnDefinition = "NVARCHAR(50)", nullable = false)**

**@Pattern(regexp = "^[\\p{L}\\p{M}0-9 \_-]+$", message = "Do not enter special characters")**

**@NotBlank(message = "Name is required")**

**private String name;**

**@Column(name = "Quantity", nullable = false)**

**@NotNull(message = "Quantity is required")**

**@Max(value = 100, message = " The quantity need >= 1 and <= 100")**

**@Min(value = 1, message = " The quantity need >= 1 and <= 100")**

**private int quantity;**

**@Column(name = "Price", nullable = false)**

**@NotNull(message = "Quantity is required")**

**@Max(value = 10000000, message = " The price need >= 20 000 and <= 10 000 000")**

**@Min(value = 20000, message = " The price need >= 20 000 and <= 10 000 000")**

**private double price;**

## Trong ProductController: (Mình sẽ xóa các cmt của thầy Hoàng đi để các bạn theo dõi code dễ dàng hơn nhé)

### **Bước 1:** Để hiểu luồng dữ liệu rõ ràng hơn mọi người thêm giúp mình:

Ở hàm **@GetMapping("/products/edit/{id}")**

Truyền type qua model để mình phân biệt được cái form khi nào xử lý create và khi nào xử lý update:

@GetMapping("/products/edit/{id}")

public String editProduct(@PathVariable("id") String id, Model model) {

**model.addAttribute("type", "edit");**

Product product = productService.getProductById(id);

model.addAttribute("selectedOne", product);

model.addAttribute("cates", categoryService.getAllCategories());

return "product-form";

}

Tương tự như vậy ở **@GetMapping("/products/new")**

**@GetMapping("/products/new")**

**public String newProduct(Model model) {**

**model.addAttribute("type", "create");**

**model.addAttribute("selectedOne", new Product());**

**model.addAttribute("cates", categoryService.getAllCategories());**

**return "product-form"; //.html**

**}**

### **Bước 2:** Xử lý hàm quan trọng nhất là **@PostMapping("/products/save")**

Mọi người nhớ thêm vào hàm save nhé xuống dưới mình sẽ giải thích tại sao có những cái này nhận vào

public String saveProduct**(@Valid @ModelAttribute("selectedOne") Product product, BindingResult bindingResult, Model model,@RequestParam(name = "type") String type)**

#### - Xử lý ở Scope Entity: Ở trong Entity lúc nãy mình có thêm các @Annotation ví dụ như: @Size, @NotBlank, @Max, @Min, @Pattern sẽ được bắt ở cái hàm này và nó sẽ lưu lỗi và message mình truyền vào trong entity: bindingResult.hasErrors() -> Sẽ xử lý các Exception ở Scope Entity

**if(bindingResult.hasErrors()) {**

**model.addAttribute("type", type);**

**model.addAttribute("cates", categoryService.getAllCategories());**

**model.addAttribute("selectedOne", product);**

**return "product-form";**

**}**

Như mọi người thấy mình khi thấy lỗi ở Scope Entity thì mình sẽ cho nó quay lại trang **product-form.html** và truyền lại các thứ như ở các hàm **edit và create** để cho form tiếp tục hiện lại các giá trị cũ và nó không bị reset sau khi bạn ấn nút **save**

#### Xử lý ở Scope Global: Các lỗi logic nếu bạn ràng buộc (Nếu Create 1 Product mà trùng Id)

* Nếu bạn chỉ xử lý **productService.saveProduct(product)** Bạn sẽ không bao giờ thấy lỗi nếu trùng id vì trong cơ chế **save()** thì chỉ cần nó **tìm được id thì nó sẽ update** -> còn **không thì nó sẽ tạo 1 record** mới và **save()**

**=> Sẽ không bao giờ phát hiện ra lỗi nếu ở code cũ của bạn, nên giờ mình sẽ độ thêm để giải quyết vấn đề đó và bạn sẽ thấy type lúc mình truyền vào model ở 2 hàm edit và new hữu dụng (Hữu dụng lần 1)**

**if(type.equalsIgnoreCase("create")) {**

**try {**

**Product productTmp = productService.getProductById(product.getId());**

**if(productTmp != null) {**

**throw new Exception("Product already exists");**

**}**

**}catch (Exception e) {**

**model.addAttribute("type", type);**

**model.addAttribute("cates", categoryService.getAllCategories());**

**model.addAttribute("selectedOne", product);**

**bindingResult.reject("error", "Create Product Failed");**

**bindingResult.reject("error", "Product Id already exists, please try another Product ID");**

**return "product-form";**

**}**

**}**

Trong đoạn code trên mọi người sẽ thấy mình sẽ bắt type ở @RequestParam và nó sẽ biết được form đó đang xử lý **edit** hay **new** (**Tí qua bên product-form.html sẽ hiểu rõ hơn**)

Như mình nói ở trên mình sẽ bắt xem nếu nó create thì mình kiểm tra xem trong database có Product nào có Id đó chưa nếu có rồi thì không cho create và bắt quay lại trang nhập **product-form.html** (Lưu ý các scope của **bindingResult** nó **chỉ tồn tại trong 1 request giống với model** nên nó sẽ chết sau 1 request)

**bindingResult.reject()**

Cái này mục đích chính dùng để bắt các Exception ngoài Scope Entity như trong đây mình bắt Exception về tạo 1 Product mới nhưng trùng ID -> **Scope này được gọi là Global**

### Đoạn code đầy đủ của **saveProduct:**

**@PostMapping("/products/save")**

**public String saveProduct(@Valid @ModelAttribute("selectedOne") Product product, BindingResult bindingResult, Model model,@RequestParam(name = "type") String type) {**

**if(bindingResult.hasErrors()) {**

**model.addAttribute("type", type);**

**model.addAttribute("cates", categoryService.getAllCategories());**

**model.addAttribute("selectedOne", product);**

**return "product-form";**

**}**

**if(type.equalsIgnoreCase("create")) {**

**try {**

**Product productTmp = productService.getProductById(product.getId());**

**if(productTmp != null) {**

**throw new Exception("Product already exists");**

**}**

**}catch (Exception e) {**

**model.addAttribute("type", type);**

**model.addAttribute("cates", categoryService.getAllCategories());**

**model.addAttribute("selectedOne", product);**

**bindingResult.reject("error", "Create Product Failed");**

**bindingResult.reject("error", "Product Id already exists, please try another Product ID");**

**return "product-form";**

**}**

**}**

**productService.saveProduct(product);**

**return "redirect:/products";**

**}**

## Trong product-form.html:

### **Bước 1:** Xử lý với cái type lúc đầu mình nhập vào để nó có ý nghĩa hơn về mặt khi nào **Edit** khi nào **Create**

**<h1 th:if="${type != null}" th:text="${type.equalsIgnoreCase('edit') ? 'Update product' : 'Create' }" >Create | Update product</h1>**

Mình sẽ bắt cái model khi hàm Edit và Create gửi qua để hiện ra cái title nào là Update nào là Create cho tường minh.

### **Bước 2:** Xử lý type để lúc nó Save() mình bắt thông qua **@RequestParam** để biết khi nào là Edit khi nào nó Create thì mình gửi type thông qua input hidden.

Thêm vào ở trong form trước nút button **Save()**

**<input th:if="${type != null}" type="hidden" name="type" th:value="${type}" >**

**<button type="submit">Save</button>**

### Bước 3: Xử lý làm sao cho nó hiện thông báo nếu có lỗi ở từng dòng như Id, Name, Price, Quantity,..

Xử lý với cái ô nhập Id: mọi người thấy nó có 2 cấu trúc hiển thị khác nhau:

**Đối với scope Entity: thì mình sẽ map cái id của th:object xuống là được**

**th:if="${#fields.hasErrors('id')}" th:errors="\*{id}"**

**Đối với scope Global:** Thì nó sẽ lưu vào 1 mảng dữ liệu và nó sẽ phân biệt theo thứ tự bạn để ở trong lúc thả lỗi ở **Controller** và nó lấy vị trí bạn đặt từ trên xuống -> cái này là in lỗi do Id nó trùng mà mình để trong Controller là vị trí thứ 2 **nên nó là [1]**:

**th:if="${#fields.hasGlobalErrors()}" th:text="${#fields.globalErrors()[1]}"**

**Trong Controller:**

**bindingResult.reject("error", "Create Product Failed");**

**bindingResult.reject("error", "Product Id already exists, please try another Product ID"); -> vị trí thứ 2 còn vị trí thứ nhất là mình add vô để thông báo failed cho người ta biết ở đầu form.**

**Trong product-form.html:**

**<p>**

**<label for="id">Product ID</label>**

**<input type="text" th:field="\*{id}" id="id" />**

**<span th:if="${#fields.hasErrors('id')}" th:errors="\*{id}" style="color: red" ></span>**

**<span th:if="${#fields.hasGlobalErrors()}" th:text="${#fields.globalErrors()[1]}" style="color: red" ></span>**

**</p>**

Tương tự như cái báo lỗi ở scope entity trong các cái <p></p> dưới mình sẽ thêm <span></span> thông báo nếu có lỗi ở các field đó với Scope Entity

**<p>**

**<label for="name">Product Name</label>**

**<input type="text" th:field="\*{name}" id="name" />**

**<span th:if="${#fields.hasErrors('name')}" th:errors="\*{name}" style="color: red" ></span>**

**</p>**

**<p>**

**<label for="quantity">Product Quantity</label>**

**<input type="number" th:field="\*{quantity}" id="quantity" />**

**<span th:if="${#fields.hasErrors('quantity')}" th:errors="\*{quantity}" style="color: red" ></span>**

**</p>**

**<p>**

**<label for="price">Product Price</label>**

**<input type="number" th:field="\*{price}" id="price" />**

**<span th:if="${#fields.hasErrors('price')}" th:errors="\*{price}" style="color: red" ></span>**

**</p>**

### **Bước 4:** Thêm vào đầu form thông báo nếu sau khi **create** mà lỗi trùng id thì nó sẽ báo ở đầu form để hiểu rõ báo **Exception ở tầng Global**

**<form th:action="@{'/products/save'}" method="post" th:object="${selectedOne}" >**

**<span th:if="${#fields.hasGlobalErrors()}" th:text="${#fields.globalErrors()[0]}" style="color: red" ></span>**

vì như ở trên mình nói nó mình add nó vào trước nên nó sẽ ở vị trí đầu tiên trong **tầng Global nên nó là [0]**

### Tổng hợp code của product-form.html

**<h1 th:if="${type != null}" th:text="${type.equalsIgnoreCase('edit') ? 'Update product' : 'Create' }" >Create | Update product</h1>**

**<form th:action="@{'/products/save'}" method="post" th:object="${selectedOne}" >**

**<span th:if="${#fields.hasGlobalErrors()}" th:text="${#fields.globalErrors()[0]}" style="color: red" ></span>**

**<p>**

**<label for="id">Product ID</label>**

**<input type="text" th:field="\*{id}" id="id" />**

**<span th:if="${#fields.hasErrors('id')}" th:errors="\*{id}" style="color: red" ></span>**

**<span th:if="${#fields.hasGlobalErrors()}" th:text="${#fields.globalErrors()[1]}" style="color: red" ></span>**

**</p>**

**<p>**

**<label for="name">Product Name</label>**

**<input type="text" th:field="\*{name}" id="name" />**

**<span th:if="${#fields.hasErrors('name')}" th:errors="\*{name}" style="color: red" ></span>**

**</p>**

**<p>**

**<label for="quantity">Product Quantity</label>**

**<input type="number" th:field="\*{quantity}" id="quantity" />**

**<span th:if="${#fields.hasErrors('quantity')}" th:errors="\*{quantity}" style="color: red" ></span>**

**</p>**

**<p>**

**<label for="price">Product Price</label>**

**<input type="number" th:field="\*{price}" id="price" />**

**<span th:if="${#fields.hasErrors('price')}" th:errors="\*{price}" style="color: red" ></span>**

**</p>**

**<p>**

**<label for="cate">Product Category</label>**

**<select th:field="\*{cate}" id="cate">**

**<option th:each="x : ${cates}"**

**th:value="${x.id}"**

**th:text="${x.name}"**

**th:selected="${selectedOne.cate !=null and x.id == selectedOne.cate.id}">???**

**</option>**

**</select>**

**</p>**

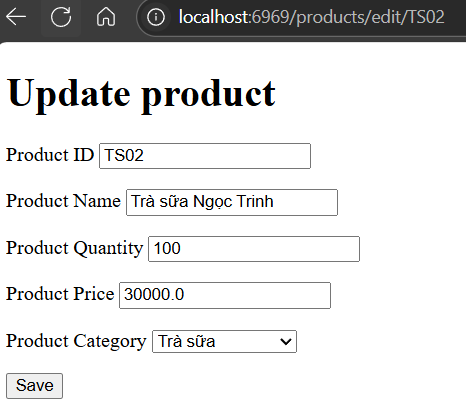
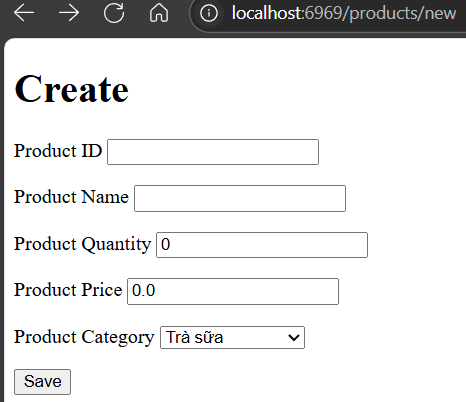
**<input th:if="${type != null}" type="hidden" name="type" th:value="${type}" >**

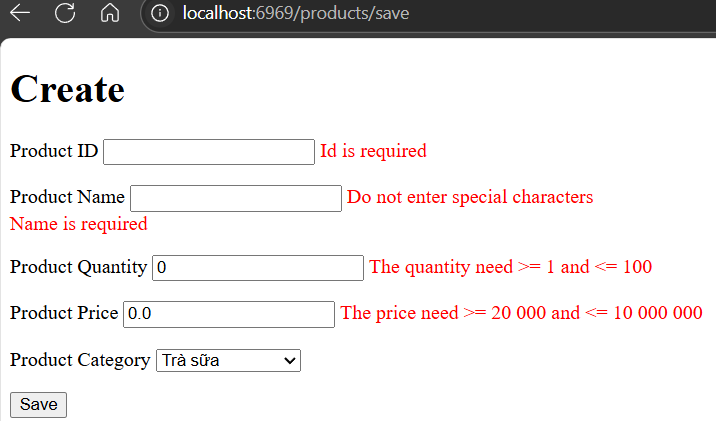
**<button type="submit">Save</button>**

**</form>**

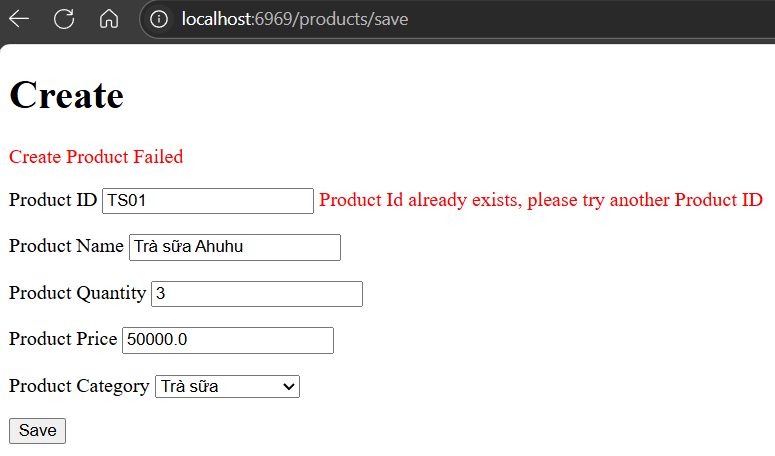
## 

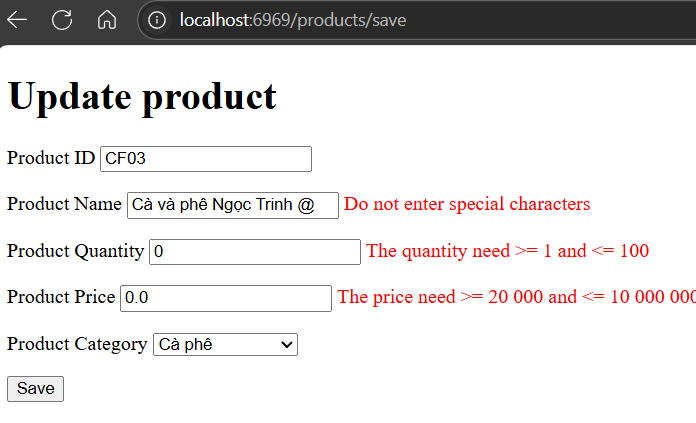
## III . Demo

****

****

****

****

****