[Bài đọc] So sánh annotaion @Valid và @Validated

1. Tổng quan

Trong bài đọc này, chúng ta sẽ tìm hiểu sự khác biệt giữa  [@Valid](https://translate.google.com/website?sl=en&tl=vi&ajax=1&elem=1&se=1&u=https://docs.oracle.com/javaee/7/api/javax/validation/Valid.html) và [@Validated](https://translate.google.com/website?sl=en&tl=vi&ajax=1&elem=1&se=1&u=https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/validation/annotation/Validated.html) trong Spring.

Kiểm tra thông tin đầu vào của người dùng là một tính năng phổ biến trong hầu hết các ứng dụng. Trong Hệ sinh thái Java, chúng ta đặc biệt sử dụng [API validate Bean tiêu chuẩn của Java](https://www-baeldung-com.translate.goog/javax-validation?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=ajax,elem) để hỗ trợ điều này. Hơn nữa, điều này cũng được tích hợp tốt với Spring từ phiên bản 4.0 trở đi. Các @Valid và @Validated  annotation xuất phát từ API Chuẩn Bean này .

Trong các phần tiếp theo, chúng ta hãy xem xét chúng một cách chi tiết.

2. @Valid và @Validated  annotation

Trong Spring, chúng ta sử dụng annotation @Valid của JSR-303 để validate cấp phương thức . Hơn nữa, chúng ta cũng sử dụng nó để đánh dấu một thuộc tính thành viên để validate . Tuy nhiên, annotation này không hỗ trợ validate nhóm.

Nhóm giúp giới hạn các ràng buộc được áp dụng trong quá trình validate. Một trường hợp sử dụng cụ thể là trình hướng dẫn giao diện người dùng (UI wizards). Ở đây, trong bước đầu tiên, chúng ta có thể có một nhóm các thông tin nhất định. Trong bước tiếp theo, có thể có một nhóm khác thuộc cùng loại bean. Do đó, chúng ta cần áp dụng các ràng buộc đối với các trường giới hạn này trong mỗi bước, nhưng @Valid không hỗ trợ việc này.

Trong trường hợp này, đối với cấp nhóm, chúng ta phải sử dụng @Validated của Spring , đây là một biến thể của @Valid trong JSR-303. Annotation này được sử dụng ở mức phương thức. Và để đánh dấu thuộc tính thành viên, chúng ta vẫn sử dụng annotation @Valid.

Bây giờ, hãy xem xét cách sử dụng các annotation này qua ví dụ sau.

3. Ví dụ

Xét một biểu mẫu đăng ký người dùng đơn giản được phát triển với Spring Boot/Spring MVC. Chúng ta sẽ chỉ có tên và thuộc tính mật khẩu :

public class UserAccount {

    @NotNull

    @Size(min = 4, max = 15)

    private String password;

    @NotBlank

    private String name;

*// standard constructors / setters / getters / toString*

}

Tiếp theo, trong mã Controller, chúng ta có phương thức saveBasicInfo với annotation @Valid để validate đầu vào của người dùng:

@RequestMapping(value = "/saveBasicInfo", method = RequestMethod.POST)

public String saveBasicInfo(

  @Valid @ModelAttribute("useraccount") UserAccount useraccount,

  BindingResult result,

  ModelMap model) {

    if (result.hasErrors()) {

        return "error";

    }

    return "success";

}

Bây giờ hãy chạy test phương thức này:

@Test

public void givenSaveBasicInfo\_whenCorrectInput\_thenSuccess() throws Exception {

    this.mockMvc.perform(MockMvcRequestBuilders.post("/saveBasicInfo")

      .accept(MediaType.TEXT\_HTML)

      .param("name", "test123")

      .param("password", "pass"))

      .andExpect(view().name("success"))

      .andExpect(status().isOk())

      .andDo(print());

}

Sau khi validate rằng test chạy thành công, bây giờ chúng ta mở rộng chức năng. Bước hợp lý tiếp theo là chuyển đổi biểu mẫu này thành một biểu mẫu đăng ký nhiều bước. Bước đầu tiên với tên và mật khẩu vẫn không thay đổi. Trong bước thứ hai, chúng ta sẽ nhập thêm thông tin như tuổi và số điện thoại. Do đó, chúng ta sẽ cập nhật UserAccount với các trường bổ sung sau:

public class UserAccount {

    @NotNull

    @Size(min = 4, max = 15)

    private String password;

    @NotBlank

    private String name;

    @Min(value = 18, message = "Age should not be less than 18")

    private int age;

    @NotBlank

    private String phone;

*// standard constructors / setters / getters / toString*

}

Tuy nhiên, lần này chúng ta sẽ nhận thấy rằng test trước không thành công. Điều này là do chúng ta không vượt qua các trường tuổi và điện thoại , vẫn chưa có trong hình ảnh trên giao diện người dùng . Để hỗ trợ hành vi này, chúng ta sẽ cần validate nhóm và annotation @Validated .

Đối với điều này, chúng ta cần phải nhóm các trường tạo thành hai nhóm riêng biệt. Đầu tiên, chúng ta sẽ cần tạo hai giao diện đánh dấu. Một cái riêng cho mỗi nhóm hoặc mỗi bước. Chúng ta có thể tham khảo bài viết của chúng ta về [validate nhóm](https://www-baeldung-com.translate.goog/javax-validation-groups?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=ajax,elem) để thực hiện chính xác việc này. Ở đây, chúng ta hãy tập trung vào sự khác biệt trong các annotation.

Chúng ta sẽ có giao diện BasicInfo cho bước đầu tiên và  AdvanceInfo  cho bước thứ hai. Hơn nữa, chúng ta sẽ cập nhật lớp UserAccount để sử dụng các giao diện đánh dấu này như sau:

public class UserAccount {

    @NotNull(groups = BasicInfo.class)

    @Size(min = 4, max = 15, groups = BasicInfo.class)

    private String password;

    @NotBlank(groups = BasicInfo.class)

    private String name;

    @Min(value = 18, message = "Age should not be less than 18", groups = AdvanceInfo.class)

    private int age;

    @NotBlank(groups = AdvanceInfo.class)

    private String phone;

*// standard constructors / setters / getters / toString*

}

Ngoài ra, bây giờ chúng ta sẽ cập nhật bộ điều khiển của mình để sử dụng annotation @Validated thay vì @Valid :

@RequestMapping(value = "/saveBasicInfoStep1", method = RequestMethod.POST)

public String saveBasicInfoStep1(

  @Validated(BasicInfo.class)

  @ModelAttribute("useraccount") UserAccount useraccount,

  BindingResult result, ModelMap model) {

    if (result.hasErrors()) {

        return "error";

    }

    return "success";

}

Kết quả của bản cập nhật này, test của chúng ta đã chạy thành công. Bây giờ chúng ta sẽ test phương thức mới:

@Test

public void givenSaveBasicInfoStep1\_whenCorrectInput\_thenSuccess() throws Exception {

    this.mockMvc.perform(MockMvcRequestBuilders.post("/saveBasicInfoStep1")

      .accept(MediaType.TEXT\_HTML)

      .param("name", "test123")

      .param("password", "pass"))

      .andExpect(view().name("success"))

      .andExpect(status().isOk())

      .andDo(print());

}

Thao tác này cũng chạy thành công. Do đó, chúng ta có thể thấy việc sử dụng @Validated cần thiết cho việc validate nhóm như thế nào .

Tiếp theo, hãy xem @Valid cần thiết như thế nào để validate các thuộc tính lồng nhau.

4. Sử dụng annotation @Valid để đánh dấu các đối tượng lồng nhau

Các annotation @Valid được sử dụng để đánh dấu các thuộc tính lồng nhau. Ví dụ, trong kịch bản trên, hãy tạo một  đối tượng UserAddress  :

public class UserAddress {

    @NotBlank

    private String countryCode;

*// standard constructors / setters / getters / toString*

}

Để đảm bảo validate đối tượng lồng nhau này, chúng ta sẽ đánh dấu thuộc tính bằng annotation @Valid :

public class UserAccount {

*//...*

    @Valid

    @NotNull(groups = AdvanceInfo.class)

    private UserAddress useraddress;

*// standard constructors / setters / getters / toString*

}

5. Ưu và nhược điểm

Hãy xem xét một số các ưu và nhược điểm của việc sử dụng annotation @Valid và @Validated trong Spring.

Annotation @Valid sẽ đảm bảo validate toàn bộ đối tượng. Tuy nhiên, sẽ không phù hợp khi cần validate một phần của đối tượng.

Mặt khác, chúng ta có thể sử dụng @Validated  để validate nhóm, bao gồm cả validate từng phần ở trên.  Tuy nhiên, trong trường hợp này, các thực thể được validate phải biết quy tắc validate cho tất cả các nhóm hoặc trường hợp sử dụng mà nó được sử dụng. Ở đây chúng ta đang trộn lẫn các mối quan tâm, do đó, điều này có thể dẫn đến anti-pattern.

6. Kết luận

Trong bài đọc này, chúng ta đã tìm hiểu những điểm khác biệt chính giữa @Valid và @Validated.

Tóm lại, đối với những thao tác validate cơ bản, chúng ta sẽ sử dụng annotation @Valid trong các lệnh gọi phương thức. Mặt khác, đối với thao tác validate theo nhóm, chúng ta sẽ cần sử dụng annotation @Validated của Spring. Các @Valid  annotation cũng cần thiết để validate các thuộc tính lồng bên trong (nested properties).

Last modified: Saturday, 14 August 2021, 8:19 AM

[Previous Activity](https://james.codegym.vn/mod/page/view.php?id=12414&forceview=1)

Top of Form