[Bài đọc] Validation trong Spring MVC

**1. Giới thiệu**

Validation là thao tác cho phép ứng dụng kiểm tra dữ liệu nhập vào bởi người dùng để đảm bảo tính hợp lý và chính xác khi xử lý các thao tác nghiệp vụ. Chẳng hạn, khi cần nhập địa chỉ email, ứng dụng cần đảm bảo email đó là hợp lệ.

Validation có thể được thực hiện ở các tầng khác nhau, chẳng hạn:

* Validate ở tầng giao diện: Sử dụng JavaScript để validate
* Validate ở tầng back-end: Validate ở Controller/Service
* Validate ở tầng CSDL: Sử dụng các ràng buộc trong CSDL (NOT NULL, UNIQUE...)

Spring hỗ trợ một số phương pháp khác nhau để validate ở tầng back-end, chẳng hạn như sử dụng annotation hoặc tự tạo các đối tượng validator.

**2. Validation sử dụng interface Validator của Spring**

Spring hỗ trợ interface Validator để thực hiện các thao tác kiểm tra tính hợp lệ của dữ liệu nhập vào. Interface Validator hoạt động bằng cách sử dụng một đối tượng Errors trong khi tiến hành xác minh, đối tượng xác minh có thể báo cáo cho đối tượng Error những sai phạm trong quá trình xác minh.

Hãy xem xét ví dụ nhỏ sau đây:

public class Person {  
 private String name;  
 private int age;  
 // the usual getters and setters...  
}

Chúng ta sẽ validate dữ liệu của Person bằng cách triển khai hai phương thức sau của interface org.springframework.validation.Validator:

* support(Class) - Validator này được dùng cho các đối tượng thuộc lớp Class?
* validate(Object, org.springframework.validation.Errors) - xác minh đối tượng được truyền vào, trong trường hợp xác minh thấy lỗi, chúng sẽ được đưa vào trong đối tượng Errors.

Triển khai một Validator là khá đơn giản, đặc biệt khi bạn biết lớp trợ giúp ValidationUtil cũng được Spring cung cấp sẵn.

public class PersonValidator implements Validator {  
 */\*\*  
 \* This Validator validates \*just\* Person instances  
 \*/* public boolean supports(Class clazz) {  
 return Person.class.equals(clazz);  
 }  
 public void validate(Object obj, Errors e) {  
 ValidationUtils.rejectIfEmpty(e, "name", "name.empty");  
 Person p = (Person) obj;  
 if (p.getAge() < 0) {  
 e.rejectValue("age", "negativevalue");  
 } else if (p.getAge() > 110) {  
 e.rejectValue("age", "too.darn.old");  
 }  
 }  
}

Như bạn thấy, phương thức static rejectIfEmpty(..)  của lớp ValidationUtil được sử dụng để từ chối thuộc tính ‘name’ nếu nó là null hoặc chuỗi trống. Hãy đọc thêm tài liệu mô tả lớp [ValidationUtils](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/validation/ValidationUtils.html) để thấy những gì mà chức năng này cung cấp bên cạnh những thứ thấy được trong ví dụ bên trên.

Mặc dù chắc chắn có thể triển khai một lớp Validator để xác minh cho mỗi đối tượng lồng trong một đối tượng ‘lớn’, tuy nhiên để tốt hơn nên triển khai Validator cho mỗi đối tượng con của nó. Một ví dụ đơn giản về một đối tượng ‘lớn’  là một đối tượng Customer, nó có hai thuộc tính String (firstName và surname) và một đối tượng Address phức tạp. Các đối tượng Address có thể được sử dụng độc lập với các đối tượng Customer, và vì vậy cũng cần triển khai một lớp AddressValidator. Nếu bạn muốn lớp CustomerValidator sử dụng lại các logic của lớp AddressValidator mà không cần phải copy-and-paste, bạn có thể sử dụng Dependency Injection hoặc khởi tạo một đối tượng AddressValidator bên trong lớp CustomerValidator, và sử dụng nó giống như sau:

public class CustomerValidator implements Validator {  
  
 private final Validator addressValidator;  
  
 public CustomerValidator(Validator addressValidator) {  
 if (addressValidator == null) {  
 throw new IllegalArgumentException("The supplied [Validator] is " +  
 "required and must not be null.");  
 }  
 if (!addressValidator.supports(Address.class)) {  
 throw new IllegalArgumentException("The supplied [Validator] must " +  
 "support the validation of [Address] instances.");  
 }  
 this.addressValidator = addressValidator;  
 }  
 */\*\*  
 \* This Validator validates Customer instances, and any subclasses of Customer too  
 \*/* public boolean supports(Class clazz) {  
 return Customer.class.isAssignableFrom(clazz);  
 }  
 public void validate(Object target, Errors errors) {  
 ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "field.required");  
 ValidationUtils.rejectIfEmptyOrWhitespace(errors, "surname", "field.required");  
 Customer customer = (Customer) target;  
 try {  
 errors.pushNestedPath("address");  
 ValidationUtils.invokeValidator(this.addressValidator, customer.getAddress(), errors);  
 } finally {  
 errors.popNestedPath();  
 }  
 }  
}

Các lỗi về xác minh được báo cho đối tượng Errors. Trong trường hợp Spring Web MVC bạn có thể sử dụng thẻ <spring:bind/> để quan sát các thông báo lỗi, nhưng tất nhiên bạn cũng có thể quan sát đối tượng lỗi của chính mình.

**3. Xử lý mã đối với các thông báo lỗi**

Trong ví dụ trước, chúng ta đã từ chối trường name và age. Nếu chúng ta đưa ra thông báo lỗi bằng cách sử dụng MessageSource, chúng ta sẽ sử dụng mã lỗi được chúng ta đưa ra khi từ chối thuộc tính (trong trường hợp này là ‘name’ và ‘age’). Khi bạn gọi (hoặc là trực tiếp, hoặc gián tiếp, sử dụng ví dụ lớp ValidationUtils) rejectValue hoặc một trong những phương thức reject từ interface Error, triển khai cơ bản sẽ không chỉ đăng ký mã bạn đã truyền vào, mà còn một số mã lỗi bổ sung. Các mã lỗi được sử dụng mà nó đăng ký được xác định bởi MessageCodesResolver ,mặc định DefaultMessageCodesResolver sẽ được sử dụng. Ví dụ, không chỉ đăng ký một thông điệp với mã lỗi bạn đã cung cấp, mà còn các thông điệp chứa thuộc tính name được bạn truyền vào phương thức reject. Do đó trong trường hợp bạn từ chối một thuộc tính bằng cách sử dụng phương thức rejectValue(“age”, “too.darn.old”), ngoài mã too.darn.old, Spring cũng sẽ đăng ký too.darn.old.age và too.darn.old.age.int (cái thứ nhất chứa thuộc tính name và cái thứ hai chứa kiểu của thuộc tính); điều này được thực hiện như một sự tiện lợi để hỗ trợ các nhà phát triển nhắm đến việc đưa ra các thông báo lỗi và những điều tương tự.

Tìm hiểu thêm về MessageCodesResolver và các chiến lược mặc định trong các tài liệu trực tuyến về Java như: [MessageCodesResolver](https://docs.spring.io/spring-framework/docs/5.0.6.RELEASE/javadoc-api/org/springframework/validation/MessageCodesResolver.html) và [DefaultMessageCodesResolver](https://docs.spring.io/spring-framework/docs/5.0.6.RELEASE/javadoc-api/org/springframework/validation/DefaultMessageCodesResolver.html), v.v...

**4. Validation trong Spring**

JSR-303 tiêu chuẩn hóa việc khai báo ràng buộc validation và metadata cho nền tảng Java. Sử dụng API này, bạn sử dụng annotation kết hợp với các ràng buộc validation cho các thuộc tính của model và runtime sẽ thực thi chúng. Có một số ràng buộc tích hợp sẵn mà bạn có thể sử dụng. Bạn cũng có thể định nghĩa các ràng buộc của mình.

Để minh họa cho điều này, hãy xem xét ví dụ về model PersonForm với hai thuộc tính sau:

public class PersonForm {  
 private String name;  
 private int age;  
}

JSR-303 cho phép bạn định nghĩa cách khai báo các ràng buộc validation cùng với các thuộc tính đó:

public class PersonForm {  
  
 @NotNull  
 @Size(max=64)  
 private String name;  
  
 @Min(0)  
 private int age;  
}

Khi một đối tượng của lớp này được xác minh bởi Validator chuẩn JSR-303, các ràng buộc đó sẽ được thực thi.

Xem trang web [Bean Validation](http://beanvalidation.org/) để biết thông tin chung về JSR-303/JSR-349. Thông tin về các khả năng cụ thể của triển khai mặc định xem trong các tài liệu [Hibernate Validator](https://www.hibernate.org/412.html).

Last modified: Thursday, 28 November 2019, 12:01 PM