

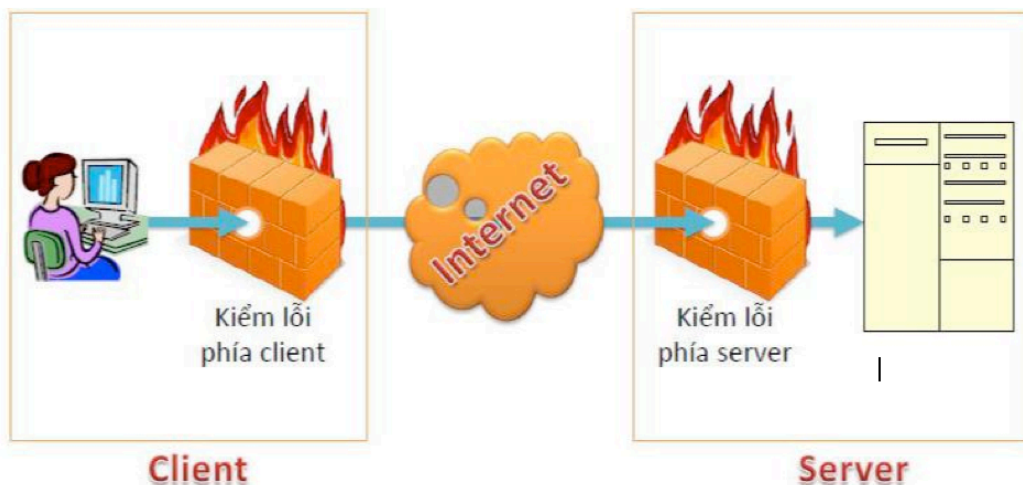
## 5.5 VALIDATION

Kiểm soát tính hợp lệ dữ liệu đầu vào luôn là vấn đề quan trọng hàng đầu của mỗi lập trình viên. Việc làm này cần được tiến hành ở cả 2 phía là client và server.

Kiểm lỗi phía client để có các phản ứng nhanh cho người dùng, làm cho ứng dụng thân thiện hơn với người dùng.

Kiểm lỗi phía server được thực hiện với những điều kiện không thể thực hiện được trên client. Kiểm lỗi phía server có tính bảo mật cao hơn nhưng thực hiện chậm gây phiền hà cho người dùng. Để khắc phục nhược điểm này người ta sử dụng kỹ thuật ajax để kết hợp kiểm lỗi phía server nhưng phản hồi ngay tức thì cho người dùng.

Ngoài ra với các ứng dụng chặt chẽ cần phải kiểm tra cả 2 phía vì hacker có thể vô hiệu việc kiểm lỗi ở client bằng nhiều cách và nếu không có kiểm lỗi phía server thì ứng dụng của chúng ta sẽ có lỗi.



### 5.5.1 Validation phía client

Kiểm duyệt dữ liệu form nhập trước khi chuyển dữ liệu đến server để xử lý là nhiệm vụ cực kỳ quan trọng. Rất nhiều tiêu chí được đặt ra để kiểm duyệt tùy thuộc vào yêu cầu cụ thể của form nhập liệu. Dù sao vẫn có thể liệt kê một số tiêu chí kiểm duyệt chung chung như sau:

- ✓ Không cho để trống ô nhập...
- ✓ Dữ liệu nhập vào phải theo một khuôn dạng nhất định nào đó: email, creditcard, url...
- ✓ Dữ liệu phải nhập vào phải đúng kiểu: số nguyên, số thực, ngày giờ...
- ✓ Dữ liệu nhập vào phải có giá trị tối thiểu, tối đa, trong phạm vi...
- ✓ Dữ liệu nhập phải đúng theo một kết quả tính toán riêng của bạn...

Sau đây là một ví dụ về việc kiểm lỗi form đăng ký thành viên:

**Annotation Validation**

Email  Không để trống email

Password  Mật khẩu ít nhất 6 ký tự

Marks  Điểm phải từ 0 đến 10

Birthday  Ngày sinh phải từ 16 đến 65

Home Page  Địa chỉ website không hợp lệ

ID Card  Số CMND không hợp lệ

Mobile  Số mobile không hợp lệ

Moto Number  Không đúng số xe máy Sài Gòn

Notes  Không chấp nhận mã HTML

### 5.5.2.2 Kiểm lỗi bằng annotation

Chúng ta thực hiện lại bài kiểm lỗi ở trên nhưng với phương pháp sử dụng annotation kiểm lỗi chuyên dụng.

#### ❖ Bước 1: Hiệu chỉnh UserInfo.java

Giữ nguyên toàn bộ mã UserInfo.java, chỉ đính kèm thêm các annotation kiểm lỗi phù hợp với các trường muốn kiểm lỗi.

- ✓ @NotEmpty: không cho để trống chuỗi name
- ✓ @NotNull: không cho phép age null (để trống)
- ✓ @Range: giới hạn giá trị số

```
@NotEmpty(message="Không để trống tên")
String name;

@NotNull
@Range(min=16, max=65)
Integer age;
```

Tất cả các annotation kiểm lỗi đều có thuộc tính message để thiết lập thông báo lỗi. Tuy nhiên thuộc tính này chỉ được sử dụng khi trong file tài nguyên không định nghĩa thông báo lỗi cho thuộc tính kiểm lỗi đó.

Trong ví dụ này nếu bạn không nhập tên thì chưa chắc dòng thông báo “Không để trống tên” đã được xuất ra mà có khi xuất một thông báo khác nếu trong file tài nguyên có định nghĩa thông báo cho luật này (xem phần file tài nguyên).

## ❖ Bước 2: File tài nguyên

Khác với kiểm lỗi bằng tay, ở đây key của file tài nguyên phải đặt đúng qui cách. Tên key đầy đủ gồm 3 phần, mỗi phần có 1 nghĩa khác nhau.

- ✓ Phần 1: tên Annotation kiểm lỗi (NotEmpty)
- ✓ Phần 2: tên command object (user)
- ✓ Phần 3: tên thuộc tính muốn kiểm lỗi (name)

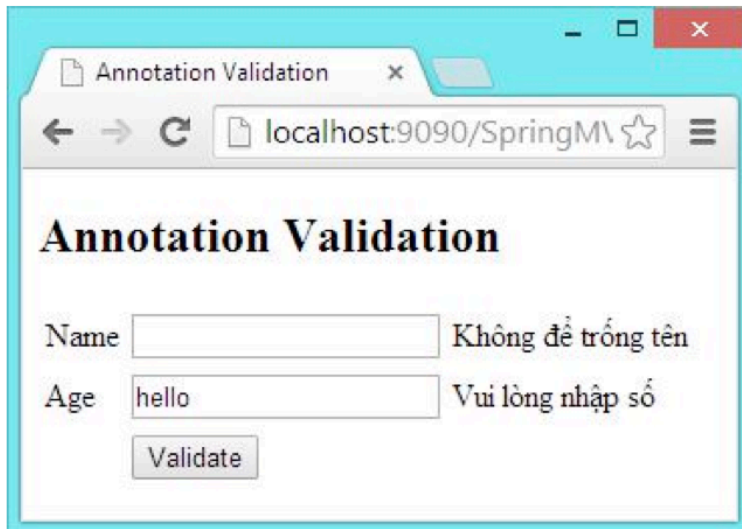
```
NotEmpty.user.name = Không để trống tên  
  
NotNull.user.age = Không để trống tuổi  
Range.user.age = Tuổi phải từ 16 đến 65  
  
#Sai kiểu dữ liệu  
typeMismatch.user.age=Vui lòng nhập số
```

Ví dụ với key NotNull.user.age thì khi thuộc tính age của user bị null thì thông báo “không để trống tuổi” sẽ được truy xuất và hiển thị.

Chú ý:

- ✓ Nếu key là NotNull.user thì tất cả các trường của user bị null thì thông báo trên được hiển thị
- ✓ Nếu key là NotNull thì tất cả các trường của bất kỳ command object nào cũng được thông báo bởi thông báo này.

Key typeMismatch.user.age định nghĩa thông báo cho thuộc tính age của đối tượng user khi nhập dữ liệu sai kiểu. Với khai báo như trên thì khi bạn nhập dữ liệu không đúng kiểu thì nhận được thông báo “Vui lòng nhập số”



### ❖ Bước 3: Hiệu chỉnh SimpleController.java

Bổ sung @Valid vào trước UserInfo để Spring MVC tự động kiểm lỗi user theo các annotation đã đính kèm trong UserInfo. Các lỗi (nếu có) sẽ được bổ sung vào result. Nhiệm vụ của chúng ta là kiểm tra xem trong result có lỗi hay không để lựa chọn view hiển thị cho phù hợp. Toàn bộ mã kiểm lỗi viết bằng tay được xóa sạch.

```
@RequestMapping(method = RequestMethod.POST)
public String simpleValidate(ModelMap map, BindingResult
    result, @ModelAttribute("user") @Valid
    UserInfo user) {
    if(result.hasErrors()){
        return "simple";
    }
    return "success";
}
```

### 5.5.2.3 Annotation kiểm lỗi

Trong ví dụ ở trên chúng ta chỉ mới làm quen vài annotation. String Spring và Java còn có nhiều annotation khác được sử dụng thường xuyên trong kiểm lỗi.

Nhóm các annotation kiểm lỗi đến từ package `javax.validation.constraints` là các annotation gốc được Java định nghĩa sẵn. Nhóm này bao gồm các annotation sau:

Annotation	Mô tả	Ví dụ
@DecimalMax(value)	Giá trị decimal tối đa	@DecimalMax("30.00") Double discount;
@DecimalMin(value)	Giá trị decimal tối thiểu	@DecimalMin("5.00") Double discount;
@Digits(integer,fraction)	Số trong phạm vi. Integer=số chữ số tối đa của phần nguyên, fraction=số chữ số tối đa phần thập phân.	@Digits(integer=6, fraction=2) Double price;
@Max(value)	Số nguyên tối đa	@Max(10) Integer quantity;
@Min(value)	Số nguyên tối thiểu	@Min(5) Integer quantity;
@NotNull	Không cho phép null	@NotNull String username;
@Null	Phải là giá trị null	@Null String unusedString;
@Pattern(regexp)	Phải khớp với biểu thức chính qui	@Pattern(regexp="\\d{9}") String idcard;
@Size(min,max)	Phạm vi kích thước của String, Collection, Map, Array	@Size(min=2, max=240) String description;
@Future	Ngày trong tương lai	@Future Date event;
@Past	Ngày trong quá khứ	@Past Date birthday;
@AssertFalse	Chỉ chấp nhận giá trị false	@AssertFalse Boolean supported;
@AssertTrue	Chỉ chấp nhận giá trị true	@AssertTrue Boolean active;
@ScriptAssert	Chỉ chứa mã script	@ScriptAssert String script;

Bên cạnh các annotation gốc, Spring mở rộng để có thêm một số annotation mới thuộc package `org.hibernate.validator.constraints`. Nhóm này bao gồm:

Annotation	Mô tả	Ví dụ
NotBlank	Chuỗi không rỗng hoặc null	@ NotBlank String name;
Email	Định dạng email	@Email String email;
CreditCardNumber	Định dạng số thẻ tín dụng	@ CreditCardNumber String credicard;
URL	Định dạng url	@URL String home;
Length(min, max)	Giới hạn độ dài chuỗi	@ Length(min=6) String password;
SafeHtml(whitelistType)	NONE, SIMPLE_TEXT, BASIC, BASIC_WITH_IMAGES, RELAXED	@SafeHtml(whitelistType = WhiteListType.NONE) String description;
Range(min, max)	Số trong khoảng	@ Range(min=0, max=10) Double marks;
NotEmpty	string, collection, map hay array không null hoặc rỗng	@NotEmpty String[] hobbies;

#### 5.5.2.4 Định nghĩa và sử dụng annotation kiểm lỗi tùy biến

Trong trường hợp danh sách các annotation kiểm lỗi trên vẫn chưa đáp ứng yêu cầu của bạn, Spring cho phép bạn định nghĩa ra các annotation riêng của mình.

Ví dụ sau chúng ta sẽ định nghĩa 2 ràng buộc kiểm lỗi mới là số chẵn `EvenNumber` và giới hạn tuổi. Mong muốn sẽ được sử dụng như mọi annotation khác:

```
@EvenNumber  
Integer namNhan;
```

```
@AgeRange(min=16, max=65)  
Date birthday;
```

Để định nghĩa ra một luật kiểm lỗi mới, chúng ta phải tạo ra 2 thành phần:

- ✓ Annotation
- ✓ Lớp kiểm lỗi cho annotation

Để hiểu rõ hơn, chúng ta hãy định nghĩa 2 luật kiểm lỗi đã nói ở trên

## ✓ Luật kiểm số chẵn

@EvenNumber Annotation

(EvenNumber.java)

Annotation này phải gắn liền với một lớp kiểm lỗi. Lớp này được chỉ ra bởi @Constraint().

```
@Documented
@Constraint(validatedBy = EvenNumberConstraintValidator.class)
@Target({ ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface EvenNumber {
    String message() default "{EvenNumber} must be an even number";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

Lớp kiểm lỗi cho annotation (EvenNumberConstraintValidator)

Theo qui định, lớp này phải thực thi theo interface ConstraintValidator và cài đặt mã cho 2 phương thức qui định là initialize() và isValid().

✓ Initialize(): nhận annotation muốn kiểm lỗi

isValid(): viết mã kiểm lỗi. Kết quả trả về true là hợp lệ, ngược lại không hợp lệ.

```
public class EvenNumberConstraintValidator
    implements ConstraintValidator<EvenNumber, Integer> {
    @Override
    public void initialize(EvenNumber annotation) { }

    @Override
    public boolean isValid(Integer field,
        ConstraintValidatorContext cxt) { if (field == null) {
        return true;
    }
    return field % 2 == 0;
    }
}
```

## ❖ Luật kiểm giới hạn tuổi @AgeRange(min,max)

Annotation (AgeRange.java)

```
@Documented
@Constraint(validatedBy = AgeRangeNumberConstraintValidator.class)
@Target({ ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface AgeRange {
    int min();

    int max();
}
```

```

String message() default "The age must be between {min} and {max}";

Class<?>[] groups() default {};

Class<? extends Payload>[] payload() default {};
}

```

Lớp kiểm lỗi cho @AgeRange (AgeRangeConstraintValidator.java)

```

public class AgeRangeNumberConstraintValidator
    implements
        ConstraintValidator<AgeRange, Date>
    {
        private AgeRange annotation;

        @Override
        public void initialize(AgeRange annotation) {
            this.annotation = annotation;
        }

        @Override
        public boolean isValid(Date field,
            ConstraintValidatorContext cxt) { if (field == null) {
                return true;
            }
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(field);
            int year = calendar.get(Calendar.YEAR);
            return year >= annotation.min() && year <= annotation.max();
        }
    }
}

```

Sử dụng Annotation tùy biến

Bây giờ chúng ta sẽ sử dụng @AgeRange của mình vừa định nghĩa để kiểm lỗi tuổi đối với ngày sinh. Để tiện chúng ta nâng cấp bài kiểm lỗi ở trên bằng cách thêm vào 1 ô nhập birthday và kiểm lỗi ngày sinh như hình sau:



**Annotation Validation**

Name  Không để trống tên

Age  Tuổi phải từ 16 đến 65

Birthday  Ngày sinh phải từ 16 đến 65

Bước 1: Bổ sung trường nhập ngày sinh vào simple.jsp

```
<tr>
    <td>Birthday</td>
    <td>
        <form:input path="birthday" />
        <form:errors path="birthday"/>
    </td>
</tr>
```

Bước 2: Bổ sung thuộc tính birthday vào UserInfo.java

```
@DateTimeFormat(pattern="dd-MM-yyyy") /*--Định dạng ngày--*/
@AgeRange(min = 16, max = 65)
Date birthday;

public Date getBirthday() {
    return birthday;
}

public void setBirthday(Date birthday) {
    this.birthday = birthday;
}
```

Bước 3: Bổ sung thông báo lỗi vào file tài nguyên SimpleValidation.properties

```
AgeRange.user.birthday = Ngày sinh phải từ 16 đến 65
```

### 5.5.2.5 Hiện thị lỗi

Thông báo lỗi được hiển thị trên viên nhờ `<form:errors path="field"/>`. Thông báo lỗi sẽ được hiển thị riêng cho trường trường khác nhau tùy vào thuộc tính `path`.

Mỗi thông báo lỗi được xuất ra với một thẻ `span` có id riêng có chứa `errors`. Ví dụ trường `name` khi gặp lỗi sẽ xuất thông báo lỗi như sau:

```
<span id="name.errors">Không để trống tên</span>
```

Nếu bạn muốn đổi `span` thành một thẻ khác (ví dụ `<li>`) thì bạn sử dụng thuộc tính `element`.

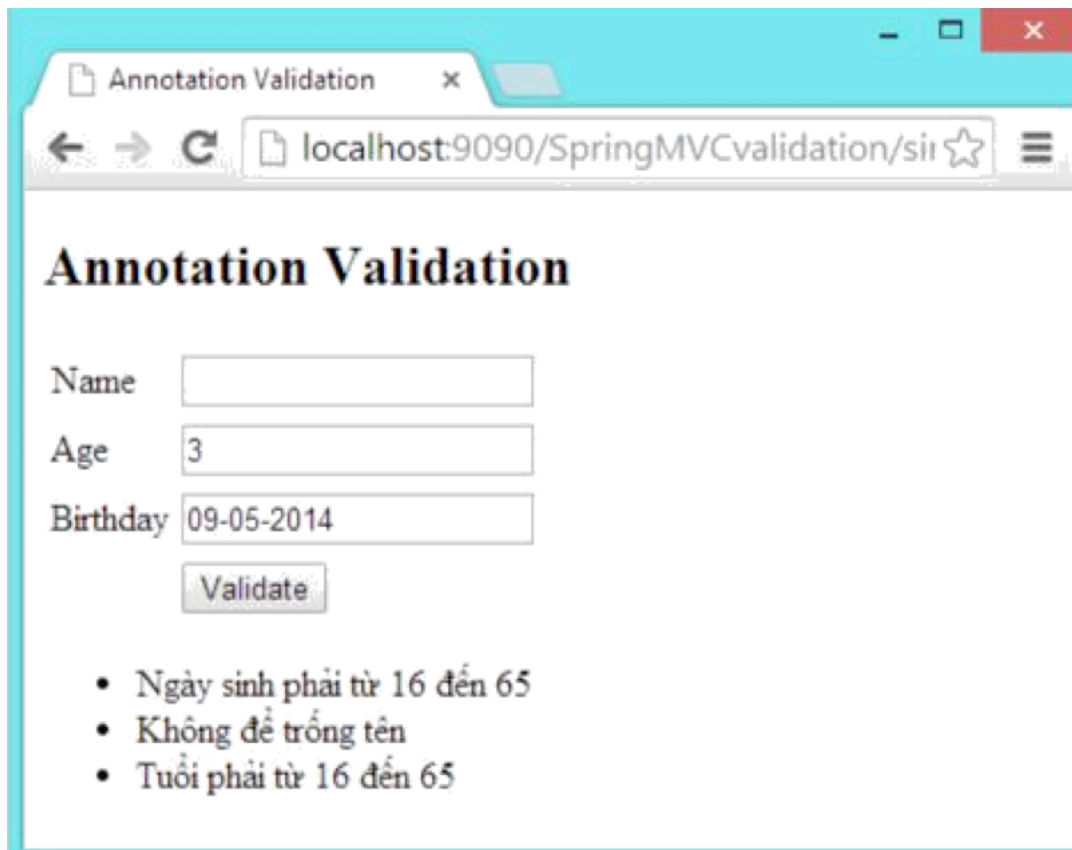
Để hiển thị tất cả các lỗi tại 1 điểm thì thiết lập giá trị cho thuộc tính `path` là `*`, nghĩa là `<form:errors path="*/>`. Thông thường các lỗi cách nhau là `<br>` tức là mỗi thông báo lỗi đặt trên 1 hàng. Nếu bạn muốn thay đổi sự tách biệt này thì điều chỉnh thuộc tính `delimiter`.

Sau đây là định nghĩa và hiển thị lỗi

```
<ul>
```

```
<form:errors path="*" element="li" delimiter="</li><li>" />
```

```
</ul>
```



The screenshot shows a web browser window with the title 'Annotation Validation'. The address bar shows 'localhost:9090/SpringMVCvalidation/sir'. The page content includes a form with three input fields: 'Name' (empty), 'Age' (containing '3'), and 'Birthday' (containing '09-05-2014'). Below the fields is a 'Validate' button. Underneath the button, there is a bulleted list of validation errors:

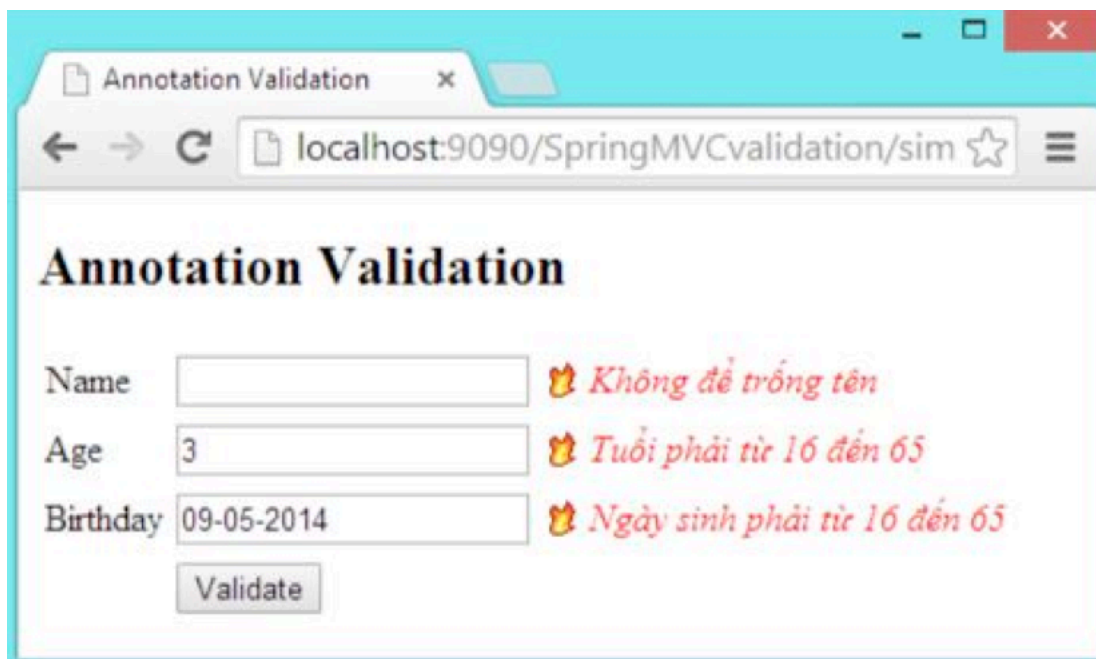
- Ngày sinh phải từ 16 đến 65
- Không để trống tên
- Tuổi phải từ 16 đến 65

Nếu bạn muốn định dạng thông báo lỗi với css thì chỉ cần định nghĩa css cho các span có id kết thúc với errors là được.

CSS này sẽ cho hiển thị lỗi như hình dưới. Tất nhiên bạn phải có ảnh trong thư mục images.

```
<style type="text/css">
span[id$=errors]{
    color:red;
    font-style: italic;
    padding-left: 20px;
    background: url("images/anifire.gif") no-repeat left center;
}
</style>
```

Như vậy để có được thông báo như sau



The screenshot shows a web browser window with the title 'Annotation Validation'. The address bar shows 'localhost:9090/SpringMVCvalidation/sim'. The main content area has a heading 'Annotation Validation'. Below it, there are three input fields with labels 'Name', 'Age', and 'Birthday'. The 'Name' field is empty and has a red error message 'Không để trống tên' (Do not leave name empty). The 'Age' field contains the value '3' and has a red error message 'Tuổi phải từ 16 đến 65' (Age must be from 16 to 65). The 'Birthday' field contains the value '09-05-2014' and has a red error message 'Ngày sinh phải từ 16 đến 65' (Date of birth must be from 16 to 65). Below the input fields is a 'Validate' button.

### 5.5.2.6 Thay đổi thông báo mặc định

Thông thường khi phạm một lỗi thì thông báo phải được đưa ra. Khi không có thông báo nào được định nghĩa thì một thông báo mặc định sẽ được sử dụng.

Bạn có thể thay đổi hoàn toàn thông báo mặc định này bằng cách tạo file ValidationMessages.properties trong thư mục src (hoặc WEB-INF/classes) và định nghĩa lại toàn bộ các thông báo lỗi sau đây.

```
javax.validation.constraints.AssertFalse.message = must be false
javax.validation.constraints.AssertTrue.message = must be true
javax.validation.constraints.DecimalMax.message = must be less than or equal to
{value} javax.validation.constraints.DecimalMin.message = must be greater than or
```

equal to {value} javax.validation.constraints.Digits.message = numeric value out of bounds (<{integer} digits>.<{fraction} digits> expected)

javax.validation.constraints.Future.message = must be in the future

javax.validation.constraints.Max.message = must be less than or equal to {value}

javax.validation.constraints.Min.message = must be greater than or equal to {value}

javax.validation.constraints.NotNull.message = may not be null

javax.validation.constraints.Null.message = must be null

javax.validation.constraints.Past.message = must be in the past

javax.validation.constraints.Pattern.message = must match "{regexp}"

javax.validation.constraints.Size.message = size must be between {min} and {max}

org.hibernate.validator.constraints.CreditCardNumber.message = invalid credit card number

org.hibernate.validator.constraints.Email.message = not a well-formed email address

org.hibernate.validator.constraints.Length.message = length must be between {min} and {max}

org.hibernate.validator.constraints.NotBlank.message = may not be empty

org.hibernate.validator.constraints.NotEmpty.message = may not be empty

org.hibernate.validator.constraints.Range.message = must be between {min} and {max}

org.hibernate.validator.constraints.SafeHtml.message = may have unsafe html content

org.hibernate.validator.constraints.ScriptAssert.message = script expression "{script}" didn't evaluate to true

org.hibernate.validator.constraints.URL.message = must be a valid URL

typeMismatch=type mismatch