Converter và Validation

Người viết: Phạm Văn Nam

**Mục lục**

[1. JSF Life-cycle 3](#_Toc293997671)

[2. Tổng quan 3](#_Toc293997672)

[3. Sử dụng Standard Converters và Standard Validators 3](#_Toc293997673)

[3.1. Standar Converter 4](#_Toc293997674)

[3.1.1 Converter cơ bản JSF hỗ trợ 4](#_Toc293997675)

[3.1.2 Sử dụng Standar Converter 5](#_Toc293997676)

[3.1.3 Convertor Error 5](#_Toc293997677)

[3.1.4 Hiển thị thông báo lỗi 5](#_Toc293997678)

[3.1.5 Thay đổi message thông báo lỗi 6](#_Toc293997679)

[3.2. Standar Validator: 6](#_Toc293997680)

[3.2.1. Validator cơ bản JSF hỗ trợ. 6](#_Toc293997681)

[3.2.2. Sử dụng Standar Validator 6](#_Toc293997682)

[3.2.3. Hiển thị lỗi validation 7](#_Toc293997683)

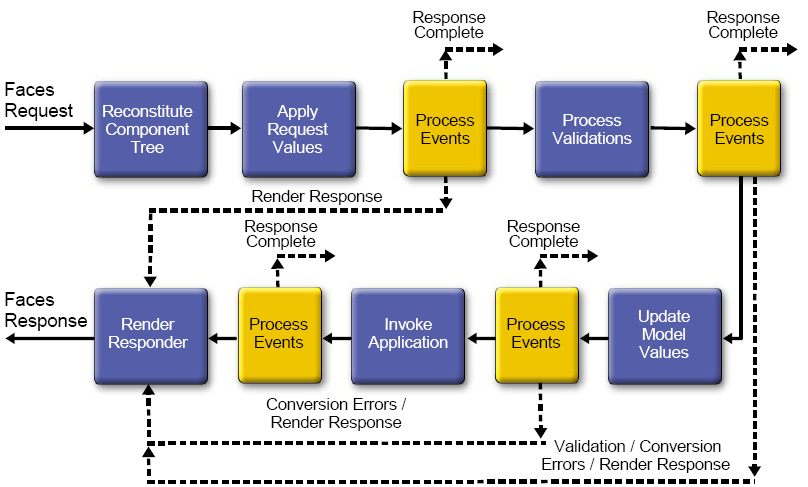
[Bạn có thể override các validator message mặc định với các key như sau. 7](#_Toc293997684)

[4. Custom Convertors và Custom Validate 7](#_Toc293997685)

[4.1 CustomConvertors 7](#_Toc293997686)

[4.2Custom Validate **9**](#_Toc293997687)

JSF Life-cycle



Tổng quan

* Khi người dùng click vào nút submit, browser sẽ gửi dữ liệu của form đến server, chúng ta gọi giá trị này là request value.
* Giá trị của request được lưu trong các đối tượng component gọi là submitted value.
* Tất cả giá trị mà người dùng nhập trên form là kiểu dữ liệu String, sau đó browser sẽ gửi đi chuỗi mà người dùng cung cấp đến server. Trái lại các ứng dụng web giải quyết nhiều loại dữ liệu như: int, Date...Tiến trình conversion xử lý việc chuyển đổi string sang các loại đó.
* Giá trị được chuyển đổi không được chuyển trực tiếp đến bean. Mà chúng được lưu bên trong các đối tượng component như các giá trị cục bộ (local values).  Sau khi conversion, các giá trị cục bộ được kiểm tra (validated). Người thiết kế sẽ quyết định điều kiện validation, ví dụ: trường dữ liệu nhập vào nên có chiều dài minimum và maximun là bao nhiêu. Sau khi các biến cục bộ được kiểm tra, "Update Model Values" phase được khởi động và các giá trị cục bộ được lưu trong các bean.
* JSF sử dụng phương pháp two-step để dễ dàng bảo trì tính toàn vẹn của mô hình. Giả sử một vài giá trị được cập nhật trước khi lỗi của người dùng được phát hiện. Mô hình có thể ở trạng thái mâu thuẩn.
* Vì lý do đó mà việc đầu tiên JSF là convert và validate tất cả input của người dùng. Nếu lỗi được tìm thấy, sẽ thông báo cho người dùng. "Update Model Values" phase sẽ bắt đầu chỉ khi tất cả các validation thành công.

Sử dụng Standard Converters và Standard Validators

Standar Converter

* + 1. Converter cơ bản JSF hỗ trợ
* convertNumber

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Value |
| type | String | number (default), currency, or percent |
| pattern | String | Formatting pattern, as defined in java.text.DecimalFormat |
| maxFractionDigits | int | Maximum number of digits in the fractional part |
| minFractionDigits | int | Minimum number of digits in the fractional part |
| maxIntegerDigits | int | Maximum number of digits in the integer part |
| minIntegerDigits | int | Minimum number of digits in the integer part |
| integerOnly | boolean | true if only the integer part is parsed (default: false) |
| groupingUsed | boolean | True if grouping separators are used (default: true) |
| locale | java.util.Locale | Locale whose preferences are to be used for parsing and formatting |
| currencyCode | String | ISO 4217 currency code to use when converting currency values |
| currencySymbol | String | Currency symbol to use when converting currency values |

* convertDateTime

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Value |
| type | String | Date (default), time, or both |
| dateStyle | String | default, short, medium, long, or full |
| timeStyle | String | default, short, medium, long, or full |
| pattern | String | Formatting pattern, as defined in java.text.SimpleDateFormat |
| locale | java.util.Locale | Locale whose preferences are to be used for parsing and formatting |
| timeZone | java.util.TimeZone | Time zone to use for parsing and formatting |
|  |  |  |
|  |  |  |

* + 1. Sử dụng Standar Converter
* Để sử dụng chúng ta sẽ gắn một converter vào một textfield
* Quy định định dạng giá trị hiện hành với ít nhất hai chữ số sau dấu chấm.

<h:inputText id=*"amount"* value=*"*#{payment.amount}*"* >

<f:convertNumber minFractionDigits=*"2"* />

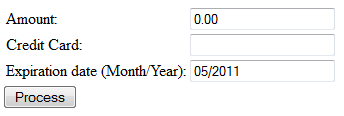
</h:inputText>

* Thiết lập pattern là MM/yyyy cho trường có kiểu dữ liệu là datetime.

<h:inputText id=*"date"* value=*"*#{payment.date}*"* >

<f:convertDateTime pattern=*"MM/yyyy"* />

</h:inputText>

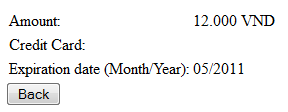


* Thiết lập pattern *#,##0.000 VND* hiển thị giá trị amount

<h:outputText value=*"*#{payment.amount}*"*>

<f:convertNumber pattern=*"#,##0.000 VND"*/>

</h:outputText>



* + 1. Convertor Error
* Khi dữ liệu nhập vào testfield không đúng với các converter quy định thì nó sẽ thông báo message khai báo giá trị không hợp lệ.
* JSF hiển thị lại trang hiện hành ngay lập tức sau khi phase "Process Validations" hoàn thành.
* Nếu người dùng cung cấp một giá trị input không hợp lệ ứng dụng web sẽ không sử dụng các giá trị input không hợp lệ đó.
  + 1. ****Hiển thị thông báo lỗi****
* Điều quan trọng là hiển thị cho người dùng nguyên nhân dẫn đến lỗi. Bạn nên thêm thẻ h:message mỗi khi sử dụng converter và validator.

<h:inputText id=*"amount"* value=*"*#{payment.amount}*"* />

<h:message for=*"amount"* styleClass=*"errorMessage"* />

* Message có hai phần: chi tiết và tổng thể. Mặc định, thẻ h:message thể hiện chi tiết và dấu phần tổng thể. Nếu bạn muốn thể hiện tổng thể message hãy sử dụng các thuộc tính sau.

<h:message for=*"amount"* showSummary=*"true"* showDetail=*"false"*/>

* Bạn có thể sử dụng styleClass hoặc thuộc tính style để thay đổi kiểu hiển thị của lỗi.

<h:messages styleClass=*"errorMessage"*/>

hoặc <h:message for=*"amount"* style=*"color:red"*/>

* Hiển thị tất cả các message lỗi từ tất cả component với thẻ h:messages.

<h:messages layout=*"table"*/>

* + 1. Thay đổi message thông báo lỗi

Thiết lập message bundle và thêm message thay thế bằng cách sử dụng key: javax.faces.component.UIInput.CONVERSION.  
javax.faces.component.UIInput.CONVERSION=Please correct your input.

Standar Validator**:**

1. Validator cơ bản JSF hỗ trợ.

|  |  |  |  |
| --- | --- | --- | --- |
| JSP Tag | Validator Class | Attributes | Validates |
| f:validateDoubleRange | DoubleRangeValidator | minimum, maximum | a double value within an optional range |
| f:validateLongRange | LongRangeValidator | minimum, maximum | a long value within an optional range |
| f:validateLength | LengthValidator | minimum, maximum | a String with a minimum and maximum number of characters |
| f:validateRequired | RequiredValidator |  | The presence of a value |
| f:validateRegex | RegexValidator | pattern | A String against a regular expression |
| f:validateBean | BeanValidator | validation-  Groups | Specifies validation groups for bean validators (see the JSR 303 specification for  details) |

1. Sử dụng Standar Validator

* Để sử dụng chúng ta sẽ gắn một converter vào một textfield
* Kiểm tra chiều dài của chuỗi thêm thẻ validator vào bên trong thẻ component.

<h:inputText id=*"card"* value=*"*#{payment.card}*"* >

<f:validateLength minimum=*"13"* />

</h:inputText>

Khi form được submit, validator sẽ đảm bảo rằng các giá trị chứa ít nhất 13 ký tự. Ngược lại message lỗi sẽ phát sinh bên cạnh component nếu có lỗi. Sử dụng thẻ h:message hoặc h:messages để thông báo lỗi

* Kiểm tra giới hạn giá trị của một số >0 và <=100.

<h:inputText id=*"amount"* value=*"*#{payment.amount}*"*>

<f:validateDoubleRange minimum=*"10"* maximum=*"10000"* />

</h:inputText>

* Kiểm tra giá trị được yêu cầu (not null) - cung cấp thuộc tính required="true".

<h:inputText id=*"date"* value=*"*#{payment.date}*"* required=*"true"* />

1. Hiển thị lỗi validation

Bạn có thể override các validator message mặc định với các key như sau.

javax.faces.component.UIInput.REQUIRED  
javax.faces.validator.NOT\_IN\_RANGE  
javax.faces.validator.DoubleRangeValidator.MAXIMUM  
javax.faces.validator.LongRangeValidator.MAXIMUM  
javax.faces.validator.DoubleRangeValidator.MINIMUM  
javax.faces.validator.LongRangeValidator.MINIMUM  
javax.faces.validator.DoubleRangeValidator.TYPE  
javax.faces.validator.LongRangeValidator.TYPE  
javax.faces.validator.LengthValidator.MAXIMUM  
javax.faces.validator.LengthValidator.MINIMUM

Custom Convertors và Custom Validate

* JSF hỗ trợ các kiểu converter và validate cơ bản nhưng trong thực tế việc convertor và validate phải dựa trên rất nhiều cơ sở khác Custom Convertors và Custom Validate sẽ giúp bạn thực hiên việc đó.
* Dưới đây là ví dụ cơ bản về custom convertor và custom validator

**Custom Convertors**

* Trong phần này sẽ hướng dẫn bạn định dạng kiểu dữ liệu để kiểm tra số của thẻ tín dụng. Ví dụ người dùng có thể nhập vào chuỗi ký tự dài 13, 14, 15, 16, 22 chúng ta sẽ dịnh dạng lại chuỗi ký tự này theo mong muốn

length 13: xxxx xxx xxx xxx

length 14: xxxxx xxxx xxxxx

length 15: xxxx xxxxxx xxxxx

length 16: xxxx xxxx xxxx xxxx

length 22: xxxxxx xxxxxxxx xxxxxxxx

* Các bước tạo Custom Convertors
* Bước 1: Implementing Custom Converter Classes

Một custom converters phải implement Converter interface trong đo có hai phương thức sau

* getAsObject(FacesContext, UIComponent, String)

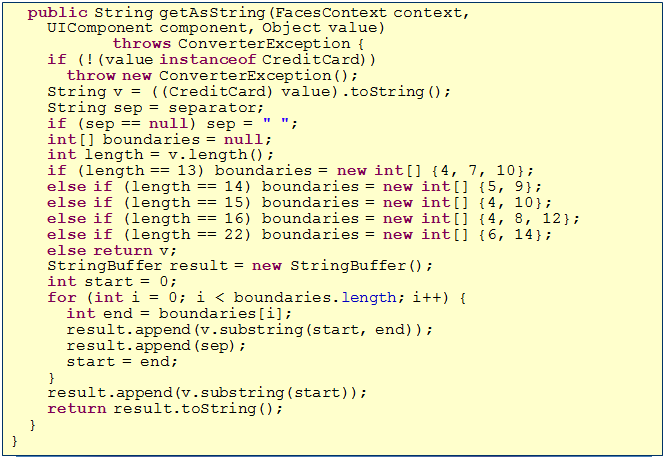
Phương thức này chuyển đổi một chuối ký tự thành một đối tượng mà bạn mong muốn và tung ngoại lệ ConverterException khi không thể chuyển đổi được. Đầu vào của phương thức là chuỗi ký tự được nhập từ người dùng.

* getAsString(FacesContext, UIComponent, Object)

Được sử dụng khi hiển thị đối tượng sau khi chuyển đổi lên giao diện.

Chuỗi ký tự mà người dùng nhâp vào không thể chuyển đổi được trong trường hợp đó sẽ tung ngoại lệ ConverterException bằng cách sử dụng FacesMessage

Implement Converter Interface và hiên thực hai phương thức getAsObject(), getAsString().



* Bước 2: Register Converter

Để Register Converter bạn khai báo trong file faces-config.xml

<converter>

<converter-id>creditCardConverter</converter-id>

<converter-class>com.corejsf.CreditCardConverter</converter-class>

</converter>

* Bước 3: Use the Converter in the Page

Sử dụng converter có 2 cách.

Cách 1: sử dụng thẻ f:converter

<h:inputText id=*"card"* value=*"*#{payment.card}*"*>

<f:converter converterId=*"creditCardConverter"* />

</h:inputText>

Cách 2: sử dụng trực tiếp trong các thẻ component

<h:inputText id=*"card"* value=*"*#{payment.card}*"*

converter=*"creditCardConverter"*>

</h:inputText>

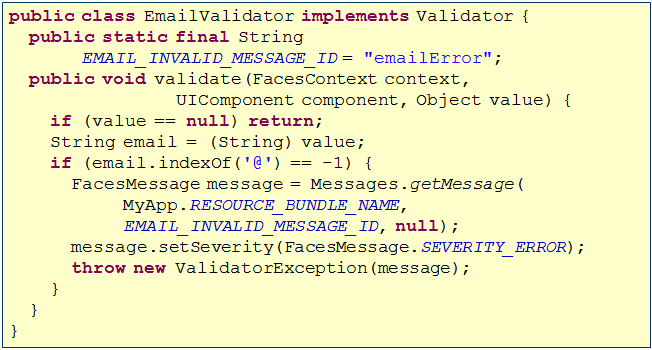
**Custom Validate**

Các bước tạo Custom Validate. Trong phần này sẽ hướng dấn bạn validate email

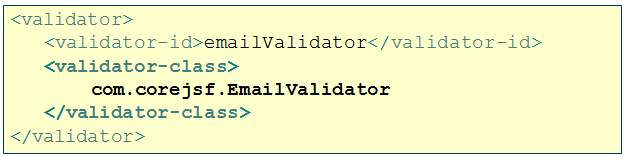
* Bước 1: Implement Validator interface

Override validate()

Validate errors: tung ngoại lệ ValidationException sử dụng FacesMessage



* Bước 2: Đăng ký validator trong file face-config.xml



* Bước 3: Sử dụng custom validators

