Basic Data Validators and Error Handling

Basic Data Validators and Error Handling

- JavaServer Faces technology supports a mechanism for validating the data of editable components.
- The SUN's reference implementation of JSF provides some default validation components that can be leveraged to implement validation of any user inputs. The JSF's core library provides tags to validate input. Following are few tags that can be used to validate the input.

Basic Data Validators and Error Handling

- f:validateDoubleRange : This tag checks the value of component within specified range. The value must be convertible to floating-point type or a floating-point itself.
- f:validateLength : This tag checks the length of a value and restrict it within a specified range. The value must be of type java.lang.String.
- f:validateLongRange : Checks is component value is within a specified range. The value must be of numeric type or string convertible to a long.

Basic Data Validators and Error Handling

- Example: restricting the input value between 1-100

```
<h:inputText id="guessValue" required="true">
    <f:validateLongRange minimum="1" maximum="100"></f:validateLongRange>
</h:inputText><br/>
<h:message for="guessValue"></h:message><br/>
```

Customizing Error Message

```
<h:body>
    <h:form>
        <h:inputText id="range" title="Enter value between 1-100"
                     validatorMessage="Value not in range 1-100">
            <f:validateLongRange minimum="1" maximum="100">
            </f:validateLongRange>
        </h:inputText>
        <h:message showDetail="true" for="range"></h:message><br/>
        <h:commandButton value="Try"></h:commandButton>
    </h:form>
</h:body>
```

Custom Validator

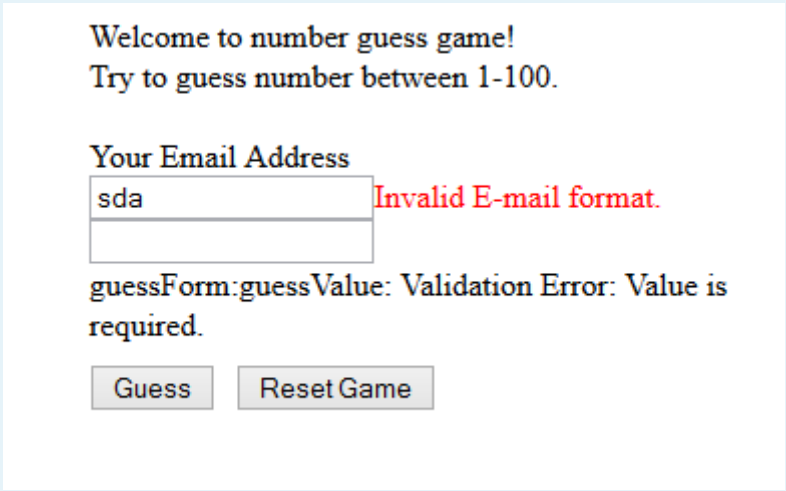
- To create a custom validator you need to follow next steps:
  - Create a validator class by implements javax.faces.validator.Validator interface.
  - Override validate() method.
  - Assign an unique validator ID via @FacesValidator annotation.
  - Reference custom validator class to JSF component via f:validator tag.

Custom Validator Example

```
@FacesValidator("com.opiframe.beans.EmailValidator")
public class EmailValidator implements Validator{

    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-]+(\\." +
                        "[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*" +
                        "(\\.[A-Za-z]{2,})$";
    private Pattern pattern;
    private Matcher matcher;
    @Override
    public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {

        matcher = pattern.matcher(value.toString());
        if(!matcher.matches()){

                FacesMessage msg =
                        new FacesMessage("E-mail validation failed.",
                                "Invalid E-mail format.");
                msg.setSeverity(FacesMessage.SEVERITY_ERROR);
                throw new ValidatorException(msg);

        }
    }

}
```

Using Custom validator

```
<h:outputLabel value="Your Email Address"></h:outputLabel><br/>
<h:inputText id="emailText" required="true" value="">
    <f:validator validatorId="com.opiframe.beans.EmailValidator"></f:validator>
</h:inputText>
<h:message for="emailText" style="color:red" /><br/>
```



ValueChangeListener

- When user make changes in input components, such as h:inputText or h:selectOneMenu, the JSF “value change event” will be fired.
- Two ways to implement it :
- 1. Method binding – In input component, specified a bean’s method directly in the “valueChangeListener” attribute.

Method binding Example

```java
@ManagedBean(name="listener")
@SessionScoped
public class ListenerBean implements Serializable{

    public ListenerBean() {
    }

    public void somethingChanged(ValueChangeEvent e){
        System.out.println(e.getOldValue());
        System.out.println(e.getNewValue());
    }
}
```

```xml
<h:inputText id="text" value="test" onchange="submit()" valueChangeListener="#{listener.somethingChanged}">
    <f:selectItem value="text.value"></f:selectItem>
</h:inputText>
```

ValueChangeListener

- 2. ValueChangeListener interface – In input component, add a “f:valueChangeListener” tag inside, and specified an implementation class of ValueChangeListener interface.

Example

```
@ManagedBean(name= "someBean")
@SessionScoped
public class SomeBean {

    private Map<String,String> countries;
    private String localeCode = "en";
    public SomeBean(){

    countries = new LinkedHashMap<String,String>();
                countries.put("United Kingdom", "en");
                countries.put("French", "fr");
                countries.put("German", "de");
                countries.put("China", "zh_CN");
    }


        public Map<String,String> getCountryInMap() {
                return this.countries;
        }

        public String getLocaleCode() {
                return localeCode;
        }

        public void setLocaleCode(String localeCode) {
                this.localeCode = localeCode;
        }
}
```

Example

```
public class SomeValueListener implements ValueChangeListener{

        @Override
        public void processValueChange(ValueChangeEvent event)
                        throws AbortProcessingException {

                SomeBean temp = (SomeBean) FacesContext.getCurrentInstance().
                        getExternalContext().getSessionMap().get("someBean");

                temp.setLocaleCode(event.getNewValue().toString());

        }

}
```

Example

```
<h:body>
	<h:form>
	<h:panelGrid columns="2">

		Selected country :
		<h:inputText id="country" value="#{someBean.localeCode}" size="20" />

		<h:selectOneMenu value="#{someBean.localeCode}" onchange="submit()">
			<f:valueChangeListener type="com.opiframe.beans.SomeValueListener" />
			<f:selectItems value="#{someBean.countryInMap}" />
		</h:selectOneMenu>

	</h:panelGrid>

	</h:form>
</h:body>
```

HTTP Session and Faces Context

- Faces context contains information about the user session in the application. Also it contains information about the request that was send by client. You can also use the response object from it.
- For example consider you have a login page, where user name and password is required. When user enter that information and press login button he/she will be redirected to new page.
- The problem is that HOW you know the user entered this page via login view?
- Or what if you want to reset some Managed Bean properties during a session?
- This is where session handling comes into picture.

HTTP Session and Faces Context

- Ok, so lets pretender that you have made an Number Guess Game with JSF technology.
- When user has guessed the correct number, we should be able to re-create a managed bean holding the randomed number, just to get a new randomed number.
- This is where you need to use FacesContext object.

HTTP Session and Faces Context

- Invalidating the session and force recreating of session scoped beans.

```
public void restartGame()
{
    FacesContext session = FacesContext.getCurrentInstance();
    session.getExternalContext().invalidateSession();
}
```