

Java Server Faces

Konvertierung und Validierung

- Allgemeines
 - Behandlung von Benutzereingaben
 - Anzeige von Fehlermeldungen
- Konvertierung
 - Standard-Konverter
 - Erstellung eigener Konverter
- Validierung
 - Standard-Validatoren
 - Erstellung eigener Validatoren
 - Backing Beans
 - JSR 303 Bean Validation

Behandlung von Benutzereingaben

Wunschtermin* ... Anzahl der Teilnehmer*



HTTP Request

date = "15.10.2011"
cnt = "1"



dateInput: UIInput (Component)

submittedValue = "15.10.2011"

cntInput: UIInput (Component)

submittedValue = "1"



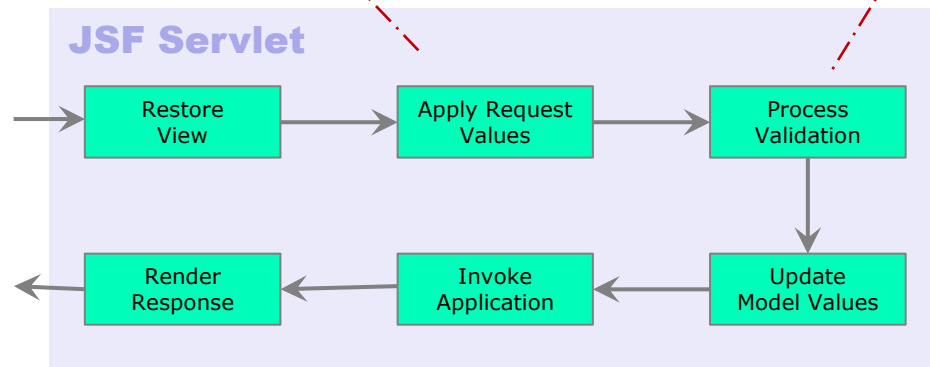
dateInput: UIInput (Component)

localValue = 15.10.2011 (Date)

cntInput: UIInput (Component)

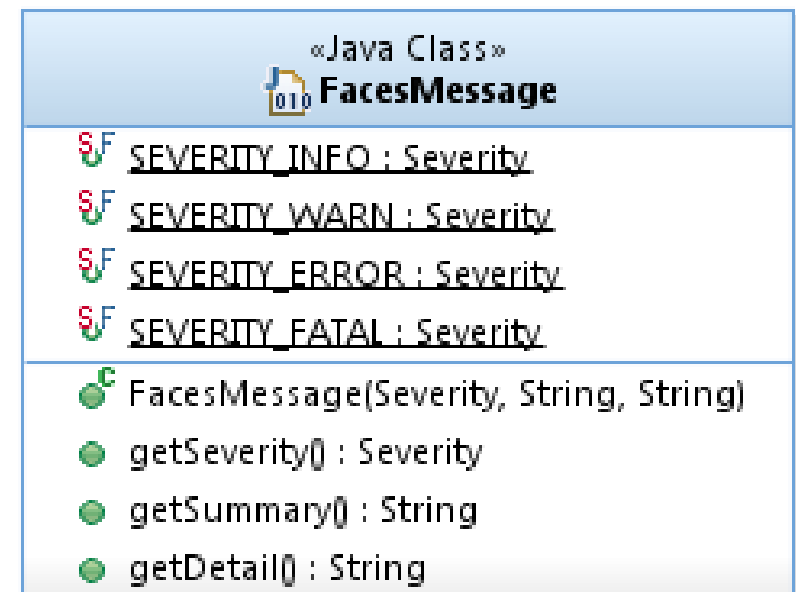
localValue = 1 (int)

+ Validierung



Phase „Process Validation“

- Jede Eingabekomponente der Seite wird aufgerufen
 - Konvertierung
 - Validierung
- Fehlermeldungen werden gesammelt
- Abbruch der Requestverarbeitung bei Auftreten von Fehlermeldungen
 - Übergang zu Render Response
 - Keine Zuweisungen an Modell
 - Keine Aktion
 - Anzeige der Fehlermeldungen in UI



Anzeige von Fehlermeldungen

■ Anzeige in UI

- Tags `<h:message>` und `<h:messages>`
- Anzeige der Kurzfassung und detaillierten Nachricht

```
<h:messages globalOnly="true"  
            showSummary="false"  
            showDetail="true" />
```

Nur globale, nicht
komponentenbezogenen
Meldungen

```
<h:messages globalOnly="false"  
            layout="table" />
```

Tabellarische Darstellung

```
<h:message for="firstday" />
```

komponentenbezogene
Meldung

Built-in Konverter

- Automatische Konvertierung von ...
 - Elementaren Datentypen sowie
 - Objekte vom Typ `BigDecimal` und `BigInteger`
- Für alle anderen Datentypen
 - Verwendung eines Standard-Konverters von JSF
 - Implementierung eines benutzerdefinierten Konverters
- Standard-Konverter von JSF
 - Konvertierung von Zahlen sowie Datumsangaben
 - Konvertierung abhängig von Sprach- und Ländereinstellungen, Muster, Symbole usw.

Konvertierung von Zahlen

- Verwendung des Tags `<f:convertNumber>`

Attribute	Bedeutung
<code>currencyCode</code>	Währungscode nach ISO4217 (z.B. AUD)
<code>currencySymbol</code>	Währungssymbol
<code>groupingUsed</code>	Verwendung des Gruppierungszeichens (z.B. „.“)
<code>integerOnly</code>	Nur Vorkommastellen werden konvertiert
<code>locale</code>	Ländereinstellungen
<code>maxFractionDigits</code>	Maximale Anzahl der Nachkommastellen
<code>maxIntegerDigits</code>	Maximale Anzahl der Vorkommastellen
<code>minFractionDigits</code>	Minimale Anzahl der Nachkommastellen
<code>minIntegerDigits</code>	Minimale Anzahl der Vorkommastellen
<code>pattern</code>	Muster
<code>type</code>	Definiert eine Zahl (<code>number</code>), Währungsangaben (<code>currency</code>) oder Prozentzahl (<code>percent</code>)

Konvertierung von Datumsangaben

■ Tag `<f:convertDateTime>`

Attribute	Bedeutung
dateStyle	Formatierungsart aus <code>java.text.DateFormat</code>
locale	Sprach- und Ländereinstellungen
timeStyle	Formatierungsart für die Zeitangaben
timeZone	Zeitzone
Type	Definiert, ob es sich bei den Angaben um Zeit- bzw. Datumsangaben handelt (<code>date</code> , <code>time</code> , <code>both</code>)

Konvertierung – Standard-Konverter

```
<!-- Anzahl Teilnehmer -->
```

```
<h:inputText id="numberofparticipants"
```

```
    value="#{inhouseTraining.numberOfParticipants}">
```

```
    <f:convertNumber integerOnly="true" type="number"
```

```
        maxIntegerDigits="2" />
```

```
</h:inputText>
```

```
<!-- Erster Schulungstag -->
```

```
<h:inputText id="firstday" value="#{inhouseTraining.firstDay}">
```

```
    <f:convertDateTime pattern="dd.MM.yyyy" />
```

```
</h:inputText>
```

Verwendung von Kennungen (Alternativ)

- Eindeutige Kennung für jede Konverter-Implementierung
- Verwendung bei JSF-Komponenten
 - Konvertierung mit Standardeinstellungen

```
<h:inputText id="firstday"  
             value="#{inhouseTraining.firstDay}"  
             converter="javax.faces.DateTime" />
```

Konverterimplementierungen und Kennungen

Kennung	Implementierungsklasse
<code>javax.faces.BigDecimal</code>	<code>javax.faces.convert.BigDecimalConverter</code>
<code>javax.faces.BigInteger</code>	<code>javax.faces.convert.BigIntegerConverter</code>
<code>javax.faces.Boolean</code>	<code>javax.faces.convert.BooleanConverter</code>
<code>javax.faces.Byte</code>	<code>javax.faces.convert.ByteConverter</code>
<code>javax.faces.Character</code>	<code>javax.faces.convert.CharacterConverter</code>
<code>javax.faces.DateTime</code>	<code>javax.faces.convert.DateTimeConverter</code>
<code>javax.faces.Double</code>	<code>javax.faces.convert.DoubleConverter</code>
<code>javax.faces.Float</code>	<code>javax.faces.convert.FloatConverter</code>
<code>javax.faces.Integer</code>	<code>javax.faces.convert.IntegerConverter</code>
<code>javax.faces.Long</code>	<code>javax.faces.convert.LongConverter</code>
<code>javax.faces.Number</code>	<code>javax.faces.convert.NumberConverter</code>
<code>javax.faces.Short</code>	<code>javax.faces.convert.ShortConverter</code>

Erstellung eigener Konverter

- Interface `javax.faces.convert.Converter` implementieren
- Methoden zum Konvertieren bereitstellen
 - Konvertieren des Strings in ein Objekt mit `getAsObject()`
 - Konvertieren des Objekts in ein String mit `getAsString()`
- Konverter registrieren
 - ID oder
 - Datentyp der Konvertierung
- Konverter mit der JSF-Komponente verknüpfen
 - Tag `f:convert`

Konvertierung – Eigene Konverter

```
public class DebitorConverter implements Converter {

    public Object getAsObject(FacesContext context, UIComponent ui, String value) {
        int index = value.lastIndexOf(' ');
        if (index <= 0) {
            throw new ConverterException("Falsche Angaben!");
        }
        Debitor debtor = new Debitor();
        debtor.setFirstname( value.substring( 0, index ) );
        debtor.setLastname( value.substring( index + 1,value.length() ) );
        return debtor;
    }

    public String getAsString(FacesContext context, UIComponent ui, Object value) {
        try {
            Debitor debtor = (Debitor) value;
            return debtor.getFirstname() + " " + debtor.getLastname();
        } catch (ClassCastException e) {
            throw new ConverterException(e);
        }
    }
}
```

Rück-Konvertierung während Render-Phase

Fehler während der Konvertierung

- Auslösen einer Exception vom Typ

- `javax.faces.convert.ConverterException`

- Unterklasse von `java.lang.RuntimeException`

- Verwendung der Klasse `javax.faces.application.FacesMessage` zum Erstellen einer Meldung

- Definition des Fehlergrads (Konstante aus `FacesMessage`)

- Detaillierte Fehlermeldung

- Kurzfassung

Konverter registrieren

- Eintrag in `faces-config.xml`

```
<converter>
  <converter-id>training.Debitor</converter-id>
  <converter-class>de.ars.training.DebitorConverter</converter-class>
</converter>
```

```
<converter>
  <converter-for-class>de.ars.training.Debitor</converter-for-class>
  <converter-class>de.ars.training.DebitorConverter</converter-class>
</converter>
```

- Annotation

```
@FacesConverter( forClass = Debitor.class )
public class DebitorConverter implements Converter {
  [...]
}
```

Verknüpfung mit einer JSF-Komponente

- Nur bei Registrierung mit ID notwendig bzw. möglich

```
<h:inputText id="firstlastname" value="#{inhouseBooking.debitor}">  
  <f:converter converterId="training.Debitor" />  
</h:inputText>
```

```
<h:inputText id="firstlastname" value="#{inhouseBooking.debitor}"  
  converter="training.Debitor" >  
</h:inputText>
```


Implementierung von JSF

- Validierung von numerischen Werten
- Prüfung der Länge einer Zeichenkette
- Wertebereich durch Attribute `maximum` und `minimum` festgelegt
- Anzeige von Fehlern analog Konvertierung

Kennung	Implementierungsklasse	Tag
<code>javax.faces.DoubleRange</code>	<code>DoubleRangeValidator</code>	<code><f:validateDoubleRange></code>
<code>javax.faces.Length</code>	<code>LengthValidator</code>	<code><f:validateLength></code>
<code>javax.faces.LongRange</code>	<code>LongRangeValidator</code>	<code><f:validateLongRange></code>

```
<h:inputSecret id="password" redisplay="false" required="true">  
    <f:validateLength minimum="4" maximum="13" />  
</h:inputSecret>  
<h:message for="password" />
```

```
<h:inputText id="numberofparticipants"  
    value="#{inhouseTraining.numberOfParticipants}">  
    <f:convertNumber integerOnly="true" type="number" maxIntegerDigits="2" />  
    <f:validateLongRange maximum="12" minimum="3">  
</h:inputText>
```

Fehler während der Validierung

- Fehlermeldung wird von JSF erzeugt
 - ID der Komponente wird für die Zuordnung benutzt
- Anzeige der Fehlermeldungen
 - Tags `<h:message>` und `<h:messages>`

Validierungsklasse erstellen

- Interface `javax.faces.validator.Validator` implementieren
- Validierung in der Methode `validate()` realisieren
 - Auslösen einer Exception vom Typ `javax.faces.validator.ValidatorException` im Fehlerfall
- Validator registrieren
- Verknüpfung zu JSF-Komponente definieren
 - Tag `f:validator`

Konvertierung und Validierung

Validierung – Eigene Validatoren

```
public class TrainingDateValidator implements Validator {

    public void validate(FacesContext context, UIComponent ui,
        Object value) throws ValidatorException {
        try {
            Date trainingDate = (Date) value;
            Calendar calendar = Calendar.getInstance();
            Date currentDate = calendar.getTime();

            if (trainingDate.before(currentDate)) {
                throw new ValidatorException(new FacesMessage(
                    "Das Datum liegt in der Vergangenheit!"));
            }
        } catch (ClassCastException e) {
            throw new ValidatorException(null, e);
        }
    }
}
```

Validierungsklasse registrieren

- Eintrag in `faces-config.xml`

```
<validator>
  <validator-id>training.TrainingDate</validator-id>
  <validator-class>de.ars.training.TrainingDateValidator</validator-class>
</validator>
```

- Annotation

```
@FacesValidator("training.TrainingDate")
public class TrainingDateValidator implements Validator {

    [...]

}
```

Verknüpfung mit einer JSF-Komponente

- Tag `f:validator`
- Attribut `validator` der Komponente
 - Binding Expression mit Verknüpfung zur Validierungsmethode

```
<h:inputText id="firstday" value="#{inhouseTraining.firstDay}">  
  <f:convertDateTime pattern="dd.MM.yyyy" />  
  <f:validator validatorId="training.TrainingDate" />  
</h:inputText>
```

Validierung in einer Methode

- Alternative Möglichkeit zur Validierung von Werten
- Klasse muss das Interface `Validator` nicht implementieren
- Name/Ort der Methode ist frei wählbar
- Voraussetzungen für die Methode
 - Methode muss `public` sein
 - Übergabeparameter vom Typ `FacesContext`, `UIComponent` und `Object` bereit stellen
 - Rückgabewert muss `void` sein
- Auflösung per Method Expression

Backing Beans

- Abhängigkeiten zwischen Benutzereingaben
- Zusammenhang zwischen Werten mehrerer JSF-Komponenten
 - Benutzereingaben als Local Value
- Verwendung einer Managed Bean
 - Attribute mit Referenzen auf die gewünschten JSF-Komponenten
 - Verwendung von Referenzen zum Auslesen der Werte

```
<h:outputText value="Wunschtermin für die Schulung:" />
<!-- Erster Schulungstag -->
<h:inputText id="firstday" value="#{inhouseTraining.firstDay}"
    binding="#{inhouseTrainingBackingBean.firstDayComponent}"
    <f:convertDateTime pattern="dd.MM.yyyy" />
</h:inputText>
<h:message for="firstday" />
```

↖ Injizieren der Komponente(n)


```
<h:outputText value="Spätester Termin für die Schulung:" />
<h:inputText id="latestdate" value="#{inhouseTraining.latestDate}"
    validator="#{inhouseTrainingBackingBean.validate}"
    <f:convertDateTime pattern="dd.MM.yyyy" />
</h:inputText>
<h:message for="latestdate"/>
```

↖ Validierung

Validierung – Backing Beans

```
public class InhouseTrainingBackingBean {  
  
    private UIInput firstDayComponent;  
  
    public void setFirstDayComponent(UIInput firstDayComponent) {  
        this.firstDayComponent = firstDayComponent;  
    }  
  
    public void validate(FacesContext ctx, UIComponent component, Object value) {  
        Date firstTrainingDay = (Date) firstDayComponent.getLocalValue();  
        Date latestTrainingDate = (Date) value;  
        if ( latestTrainingDate.before( firstTrainingDay ) ) {  
            throw new ValidatorException(new FacesMessage(  
                FacesMessage.SEVERITY_ERROR,  
                "Fehler in der Validierung",  
                "Spätester Termin für die Schulung liegt vor dem Wunschtermin!"));  
        }  
    }  
}
```

 Injizieren der Komponente(n)

 Validierung

JSR 303 Bean Validation

- Validierung auf Basis von Annotationen
- Integration in JSF als Standard-Validator
 - Validierung der an Eingabekomponenten gebundenen Properties des Modells vor Zuweisung
- Seit Java EE 6 integriert (manuell Integration für Java EE 5 notwendig)

```
@Named("person")
public class Person {

    @NotNull
    @Size(min=1)
    private String firstName;

    @NotNull
    @Size(min=1, message="{person.firstName.size}")
    private String lastName;

    [...]
}
```

- Wie wird Konvertierung und Validierung während der Requestverarbeitung durch das FacesServlet eingebunden?
- Was ist das Ergebnis der (erfolgreichen) Konvertierung und Validierung?
- Wofür werden Backing Beans verwendet und wie werden Sie technisch umgesetzt?