

Struts Validazione : Validation.xml e metodo validate()

Questo tips & trick illustra come gestire, in maniera semplice, la validazione di un form Struts.

1. Plugin Validator
2. Metodo Validate() all'interno dei FormBean
3. Validazione, di logica semantica, all'interno delle Action

1. PLUGIN VALIDATOR

Quando possibile, è buona cosa utilizzare il plugin Validator, basato su criteri tramite file Xml e regular expression.

1.1. Installazione

Nel file Struts-config è necessario “attivare” il Validator:

```
<message-resources parameter="resource.ApplicationResources" />
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
    value="/WEB-INF/validatorrules.xml,/WEB-INF/validation.xml" />
</plug-in>
```

Aggiungere i file in allegato (validator-rules.xml e validation.xml) al path /WEB-INF

Aggiungere i messaggi di errori standard al file resource.ApplicationResources.

1.2. Esempio Pratico

Struts-config.xml

```
<form-beans >
  <form-bean name="CercaCartaForm"
    type="it.demo.form.CercaCartaForm" />
</form-beans>

<action
  input="/cercacarta.jsp"
  name="CercaCartaForm"
  path="/CercaCarta"
  scope="request"
  validate="true"
  type="it.demo.gestioneterzalineacredito.action.CercaCartaAction" />
</action-mappings>
```

L'attributo *input* definisce la pagina in cui ritornare nel caso in cui venga generato qualche errore durante la validazione.

Pagina JSP: cercacarta.jsp

// Definizione del campo testo

```
<html text name="CercaCartaForm" property="numcarta" />
```

//Generazione dell'errore da visualizzare, in questo punto del codice verrà visualizzato

//l'eventuale messaggio di errore

```
<html:errors property="numcarta"/>
```

Relativo Form Bean : CercaCartaForm **extends ValidatorForm**
variabili contenute : String numcarta

Per validare i dati è sufficiente specificare all'interno del file di configurazione "validation.xml" i diversi criteri, di ogni Form.

```
<form-validation>
  <formset>
    (1) <form name="CercaCartaForm">
      (2) <field property="numcarta" depends="required">
        (3) <arg0 key="validator.numcarta" />
      </form>
    </formset>
  </form-validation>
```

L'esempio riportato qui di sopra definire un nuovo criterio di validazione per un determinato Form. Al punto (1) viene definito tramite l'attributo *name* il nome del FormBean definito all'interno dello Struts-Config.xml.

Al punto (2) viene definito il criterio (*depends*) per un determinato campo, determinato dall'attributo *property*.

In questo caso il campo "numcarta" viene impostato come obbligatorio (*required*).

Infine, al punto (3) viene settato il messaggio di errore da stampare, memorizzato all'interno del file di ApplicationResource, nella seguente maniera:

```
validator.numcarta = Il numero di carta di credito
```

Il messaggio di errore visualizzato all'utente finale, nel punto in cui è definito il tag `<html:errors property="numcarta" />`

sarà il risultato del concatenamento tra la stringa del messaggio da noi richiesto più quello del criterio impostato (errors.TIPOCRITERIO)

In pratica in questo caso :

```
validator.numcarta + errors.required
```

in quanto errors.required è così definito:

```
errors.required={0} è richiesto.
```

e al posto del parametro {0} viene inserito il messaggio validator.numcarta.

Qui di seguito vengono riportati alcuni esempi di criteri più complessi, visti durante il workshop:

1) Campo numcarta obbligatorio, minimo 16 caratteri e solo numeri

Nota:

^ = inizio stringa,

\$ = fine stringa,

* = per tutti i caratteri

[..] = caratteri possibili

```
<field property="numcarta" depends="required,minlength,mask">
  <arg0 key="validator.numcarta" />
  <arg1 key="{var:minlength}" name="minlength" resource="false"/>
  <var><var-name>minlength</var-name><var-value>16</var-value></var>
  <var><var-name>mask</var-name><var-value>^[0-9]*$</var-value></var>
</field>
```

HOWTO : ^[A-Za-Z]*\$ lettere maiuscole e minuscole
 ^[A][0-8]*\$ iniziare con lettera A poi con numeri tra 0 e 8

2) Campo *importo*, valido solo se è maggiore di 1000 e se il numero di carta di credito è '1234567890123456'

Nota:

la funzione *validwhen*, vuole che venga definito il criterio all'interno della variabile "test".

Il messaggio di errore stampato sarà *errors.invalid*.

```
<field property="importo" depends="validwhen">  
  <arg0 key="validator.importo"/>  
  <var><var-name>test</var-name><var-value>((*this* >= 1000) and  
  (numcarta == '1234567890123456'))</var-value></var>  
</field>
```

3) Campo *email* deve essere validato secondo la sintassi di un indirizzo:

```
<field property="email" depends="email">  
  <arg0 key="validator.email"/>  
</field>
```

4) Il campo *datascadenza* deve contenere una data valida, secondo il formato giorno/mese/anno.

Nota:

il formato deve essere impostato all'interno della variabile "datePattern".

Utilizzando invece "datePatternStrict", NON vengono accettate date abbreviate del tipo 1/1/2006, ma solo 01/01/2006.

```
<field property="datascadenza" depends="date">  
  <arg0 key="validator.data"/>  
  <var><var-name>datePattern</var-name>  
  <var-value>dd/MM/yyyy</var-value></var>  
</field>
```

5) Il campo *importo* deve essere obbligatorio, intero e contenuto nel Range 10 e 30.

```
<field property="importo" depends="required,intRange">  
  <arg1 name="intRange" key="{var:min}" resource="false"/>  
  <arg2 name="intRange" key="{var:max}" resource="false"/>  
  <var><var-name>min</var-name><var-value>10</var-value></var>  
  <var><var-name>max</var-name><var-value>30</var-value></var>  
</field>
```

Altri esempi, e una spiegazione dettagliata di tutti i Tag validation, è possibile reperirla al seguente indirizzo:

http://struts.apache.org/1.2.8/userGuide/dev_validator.html

2. PLUGIN VALIDATOR + VALIDATE()

Utilizzando il plugin di validazione sopra descritto, non è necessario definire un metodo Validate() all'interno dell'oggetto Form, in quanto viene già implementato dalla classe ValidatorForm.

Tuttavia alcuni controlli possono risultare lunghi o poco chiari utilizzando i file xml di validazione. Proprio per questo motivo, viene presentato qui di seguito un sistema per sfruttare entrambe le validazioni offerte dal framework Struts (Validate() e validation.xml)

1.1. Esempio Pratico

Riprendendo l'esempio proposto in precedenza, all'interno dell'oggetto CercaCartaForm, viene effettuato un Overriding del metodo validate(), richiamando però anche il plugin validator, tramite il meccanismo di *super*.

```
public ActionErrors validate (ActionMapping mapping,
                             HttpServletRequest request){
    ActionErrors errors = new ActionErrors();
    errors = super.validate(mapping,request);
    //ESEMPIO BANALE DI UNA NUOVA VALIDAZIONE FATTA A MANO
    if (numcarta.equals("1234567890123456"))
        errors.add ("numcarta",
                    new ActionMessage("errore.numcarta.sequenzaerrata"));
}
```

Ovviamente l'errore stampato a video sarà quello definito con la label "errore.numcarta.sequenzaerrata", memorizzata sempre nel file di ApplicationResources.

NOTA:

Prima vengono effettuati i controlli tramite i criteri impostati nel file validation.xml, e successivamente quelli aggiunti tramite codice.

Non utilizzando il meccanismo di super.validate, verranno effettuati solo i controlli definiti nel codice Java.

RIEPILOGO:

MECCANISMO DI VALIDAZIONE	OGGETTO FORM
Validation.xml	- extends ValidatorForm - non definito validate()
Validation.xml + Validazione tramite codice	- extends ValidatorForm - definire metodo validate - richiamare super.validate
Validazione tramite codice	- extends ValidatorForm - definire metodo validate - non usare super.validate

3. VALIDAZIONE SEMANTICA TRAMITE ACTION

Spesso alcune applicazioni effettuano controlli sintattici sui dati inseriti dall'utente, ma anche controlli semantici tramite accessi al database.

1.1 CASI D'USO (Esempi)

- Prima di effettuare transazioni di movimenti bancari, viene verificato che la carta non sia stata bloccata o che possieda la disponibilità necessaria.

In caso di esito negativo, l'utente viene rimandato alla pagina iniziale in cui deve compilare il form, visualizzando il relativo messaggio di errore.

1.2. Esempio Pratico

```
//CONTROLLO CHE LA CARTA SIA BLOCCATA e ABBIA LA DISPONIBILITA'  
//TRAMITE CHIAMATA SERVIZIO  
// esito = true se bloccata  
// disp = true se possiedi la disponibilità minima  
ActionMessages errors = new ActionMessages();  
If (esito){  
    errors.add("errore", new  
        ActionMessage("validator.bloccocarta"));  
if (!disp){  
    errors.add("errore ", new  
        ActionMessage("validator.dispcarta"));  
if (!errors.isEmpty()){  
    saveErrors(request, errors);  
    return mapping.findForward("Errore");  
}  
return mapping.findForward("Accesso");
```

In caso ci siano degli errori, si viene rimandati al Forward "Errore", stampando i messaggi d'errore dove è stato definito il tag: `<html:errors proprerty="errore" />`

Nota:

Utilizzare ActionMessages, e non ActionError, in quanto è stato deprecato.