Validacions amb Java Bean Validation (JSR-303)

A qui va dirigit

Aquest how-to va dirigit a tots aquells usuaris que tinguin la necessitat de definir validacions pròpies a servidor amb Canigó 3.

Versió de Canigó

Els passos descrits en aquest document apliquen a les versions de Canigó 3 que utilitzen JSF2 / Richfaces 4.0.0.

Introducció

Què és l'especificació JSR-303?

JSR-303 (Bean Validation) és una especificació que defineix l'API per poder validar JavaBeans, tant per Java EE com per Java SE. L'objectiu d'aquesta especificació es proveir els mecanismes necessaris per declarar i validar Beans.

Hibernate Validator

La implementació de referència de l'especificació JSR-303 és Hibernate Validator però n'existeixen altres com Apache Bean Validator. Per a aquest how-to s'ha fet servir Hibernate Validator, concretament la versió 4.1.

Validacions amb Java Bean Validation (JSR-303)

Exemple d'ús validacions JSR- 303

Configuració prèvia

Al **pom.xml** cal afegir la següent dependència:

```
...
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>4.1.0.Final</version>
</dependency>
...
```

Validadors que ja es proporcionen amb Hibernate Validator

Hibernate Validator ja proporciona una sèrie de validadors dels quals es pot fer ús. Alguns d'ells són:

- `@NotNull`: Valida que un valor no sigui *null*
- `@Pattern`: Valida que una cadena de text acompleixi l'expressió regular indicada a l'anotació
- `@Past`: Valida que una data sigui anterior a la data del servidor
- `@Min`: Valida que un número sigui major o igual a l'especificat a l'anotació
- `@Max`: Valida que un número sigui menor o igual a l'especificat a l'anotació

Crear validadors propis

Per a aquest how-To s'ha elaborat un validador que verificarà que un objecte no sigui *null* i si aquest objecte fos una cadena de text, validarà també que no arriba buida (el validador `@NotNull` esmentat anteriorment només verifica que un objecte no sigui *null*).
Per implementar aquest validador cal seguir els següents passos:

Definir la Interfície de validació

En primer lloc caldrà crear la interfície del validador, fet que servirà per poder cridar-la mitjançant anotacions:

```
package cat.gencat.canigo.howTo.validators;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import cat.gencat.canigo.howTo.validators.impl.CustomNotNullImpl;

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy=CustomNotNullImpl.class)
public @interface CustomNotNull {
        String message() default "{value.is.null}";
        Class<?>[] groups() default {};
        Class<? extends Payload>[] payload() default {};
}
```

Validacions amb Java Bean Validation (JSR-303)

Respecte a les anotacions de la `@interface`:

- Amb `@Target` declarem els tipus d'elements als que es podrà aplicar la anotació. En aquest cas es podrà declarar en mètodes i camps.
- Amb `@Retention` s'especifica la vida útil de l'anotació. En aquest cas s'ha escollit RUNTIME per tal que es mantingui accessible en temps d'execució.
- Amb `@Constraint` indiquem la classe que conté la implementació de la validació a realitzar.
-

Respecte al contingut d'aquesta `@interface`:

- `message`: Definim la clau de internacionalització del missatge d'error. N'indiquem un valor per defecte que caldrà que l'informem a un fitxer de internacionalització de missatges de validacions concret que esmentarem més endavant.
- `groups`: Permet agrupar validacions. No el farem servir a l'exemple però cal definir-ho donat que és un camp obligatori.
- `payload`: Permet que els clients de l'API puguin inicialitzar la validació amb els seus propis objectes. No el farem servir a l'exemple però cal definir-ho donat que és un camp obligatori.

Es podrien definir altres mètodes i propietats a aquesta `@interface` si es necessitessin fer servir com a part de la validació. Per a l'exemple d'aquest how-to no es precisen més.

Definir la implementació de la validació

Caldrà definir una classe encarregada de fer la validació:

```
package cat.gencat.canigo.howTo.validators.impl;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
import cat.gencat.canigo.howTo.validators.CustomNotNull;

public class CustomNotNullImpl implements ConstraintValidator<CustomNotNull, Object> {

  @Override
  public void initialize(CustomNotNull arg0) { }

  @Override
  public boolean isValid(Object value, ConstraintValidatorContext ctx) {
    return checkNull(codeValue);  }

  private boolean checkNull(Object value) {
    boolean isValid=true;
    if (value==null){
      isValid = false;
    } else if ((value instanceof String) && ((String)value).length()==0){
      isValid = false;
    }
    return isValid;
  }
}
```

El mètode més important d'aquesta classe és el mètode *isValid*, que retornarà un booleà indicant si ha passat o no la validació. La signatura d'aquest mètode variarà en funció de la signatura de la pròpia classe i com faci la implementació del *ConstraintValidator*. En aquest exemple s'ha especificat que es validarà un *Object*, donat que es voldrà aplicar la validació tant a camps *String* com *Integer.*

Validacions amb Java Bean Validation (JSR-303)

Definir els missatges del validador

Cal definir els codis i missatges que farà servir el validador. Aquests missatges s'han de incloure dins un fitxer amb un nom concret que cal crear a l'arrel del classpath (src/main/resources) :

ValidationMessages.properties

```
value.is.null=Missatge per defecte del validador CustomNotNull
edat.is.null=L'edat ha d'estar informada
data.past=La data indicada és anterior a la d'avui
```

Aquests missatges es poden internacionalitzar, de igual manera que es fa amb la resta de missatges a Canigó, creant nous fitxers amb el sufix de l'idioma:

- ValidationMessages_ca_ES.properties
- ValidationMessages_es_ES.properties
- ....

Utilitzar els validadors

La manera de fer servir aquest tipus de validacions és mitjançant anotacions. Aquestes anotacions es poden aplicar als tipus d'elements especificats per el `@Target` de la interfície del validador. També es poden fer servir en diferents capes de l'aplicació (Beans, Pojos, ...). A l'exemple d'aquest how-to s'han afegit al Bean:

```
package cat.gencat.canigo.howTo.bean;

import java.util.Date;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.validation.constraints.Past;
import org.springframework.stereotype.Component;
import cat.gencat.canigo.howTo.validators.CustomNotNull;

@Component("howToValidacioBean")
public class HowToValidacioBean {

 private Integer edat=20;

 @Past ( message="{data.past}")
 private Date data;

 @CustomNotNull
 private String code;

 @CustomNotNull(message="{edat.is.null}")
 public Integer getEdat() {return edat; }

 public void setEdat(Integer edat) {this.edat = edat;}

 public Date getData() {return data;}

 public void setData(Date data) {this.data = data;}

 public String getCode() {return code;}

 public void setCode(String code) {this.code = code;}

//Continua a la següent pàgina...
```

Validacions amb Java Bean Validation (JSR-303)

```
// ... Continua des de la pàgina anterior

  public void doSubmit() {
    try {
      FacesContext.getCurrentInstance().addMessage("validacionsForm",
       new FacesMessage(FacesMessage.SEVERITY_INFO,"Formulari sense errors",null));
    } catch (Exception e) {
      FacesContext.getCurrentInstance().addMessage("validacionsForm",
       new FacesMessage(FacesMessage.SEVERITY_ERROR,e.getCause().getMessage(),null));
    }
  }
}
```

Segons es pot veure a aquest exemple, s'han aplicat validacions als camps:

- **data**: S'ha aplicat un validador propi d'Hibernate Validator, `@Past`. S'ha internacionalitzat el seu missatge de validació.
- **code**: S'ha aplicat el validador creat a aquest camp. No s'ha informat cap codi de missatge d'internacionalització, per tant n'agafarà el definit per defecte a la interfície: *value.is.null*.
- **getEdat()**: S'ha aplicat el validador creat a aquest mètode. S'ha informat un codi de missatge a la pròpia anotació, per tant n'agafarà aquest codi en comptes del definit per defecte al a interfície.

La vista associada a aquest Bean és molt simple, només conté un formulari amb aquests camps:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:rich="http://richfaces.org/rich"
  xmlns:c="http://java.sun.com/jstl/core"
  xmlns:a4j="http://richfaces.org/a4j"
  template="/views/layouts/template.jsf">
  <ui:define name="body">
    <h:form id="validacionsForm">
      <h:panelGrid columns="3">

        <h:outputLabel value="Edat"/>
        <h:inputText value="#{howToValidacioBean.edat}" id="edat" />
        <h:message for="edat" />

        <h:outputLabel value="#{msg.dateLabel}"/>
        <rich:calendar id="data" value="#{howToValidacioBean.data}"/>
        <h:message for="data" />

        <h:outputLabel value="Codi"/>
        <h:inputText id="code" value="#{howToValidacioBean.customCode}"/>
        <h:message for="code" />

        <h:commandButton value="Enviar" action="#{howToValidacioBean.doSubmit}"/>
        <h:message for="validacionsForm" />

      </h:panelGrid>
    </h:form>
  </ui:define>
</ui:composition>
```