L'API Bean Validation

Pr. Youssef Saadi

Master Informatique Décisionnelle

Faculté Des Sciences Et Techniques
Université Sultan Moulay Slimane Béni-Mellal

AU: 2019/2020

API Bean Validation

- L'API Bean Validation est issue des travaux de la JSR 303 : https://jcp.org/en/jsr/detail?id=303
- L'intérêt de cette API est de proposer une approche cohérente sous la forme d'un standard pour la validation des données d'un bean.
- Il y a besoin d'un standard pour plusieurs raisons :
  - il existe déjà plusieurs frameworks open source de validation de données
  - les validations se font dans toutes les couches, pas toujours de façon cohérente et fréquemment par duplication de code
  - plusieurs technologies des plate-formes Java ont besoin d'un framework de validation : JPA, JSF, ...

Les annotations de l'API Bean Validation

- **@AssertFalse** : s'applique à un attribut de type booléen et vérifie que sa valeur soit false.
- **@AssertTrue** : s'applique à un attribut de type booléen et vérifie que sa valeur soit true.
- **@DecimalMax**("10.5") : permet de vérifier qu'une valeur flottante est bien inférieure ou égale à la valeur spécifiée dans l'annotation de validation. Attention : la valeur spécifiée doit l'être sous forme d'une chaîne de caractères.
- **@DecimalMin**("10.5") : permet de vérifier qu'une valeur flottante est bien supérieure ou égale à la valeur spécifiée dans l'annotation de validation. Attention : la valeur spécifiée doit l'être sous forme d'une chaîne de caractères.
- **@Digits**(integer=3, fraction=2) : permet de vérifier le nombre de chiffres utilisé par une valeur flottante. Vous devez spécifier deux paramètres entiers dans l'annotation : integer qui indique le nombre de chiffres utilisé pour la partie entière et fraction qui indique le nombre de chiffres utilisé par la partie décimale.

Les annotations de l'API Bean Validation

- **@Email** : et oui, désolé, mais je vous ai fait coder quelque chose qui existait déjà (il faut bien qu'on apprenne les fondamentaux). Ce annotation de validation demande à vérifier si une chaîne de caractères correspond bien à un email. Attention : ce validateur accepte une chaîne vide contrairement à notre validateur.
- **@Future** : permet de vérifier qu'une date est postérieure à la date actuelle.
- **@FutureOrPresent** : permet de vérifier qu'une date est postérieure ou égale à la date actuelle.
- **@Max**(100) : permet de vérifier qu'un entier est inférieur ou égal à la valeur spécifiée dans l'annotation.
- **@Min**(1) : permet de vérifier qu'un entier est supérieur ou égal à la valeur spécifiée dans l'annotation.
- **@Negative** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit strictement inférieure à zéro.
- **@NegativeOrZero** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit inférieure ou égale à zéro.

Les annotations de l'API Bean Validation

- **@NotBlank** : permet de vérifier qu'une chaîne de caractères ne puisse pas être nulle et contienne au moins un caractère qui ne soit pas un séparateur (blanc, tabulation, retour à la ligne, ...).
- **@NotEmpty** : permet de vérifier qu'une chaîne de caractères ne puisse pas être nulle ou vide.
- **@NotNull** : permet de vérifier qu'une valeur ne puisse pas être nulle.
- **@Null** : permet de vérifier qu'une valeur soit nulle.
- **@Past** : permet de vérifier qu'une date est antérieure à la date actuelle.
- **@PastOrPresent** : permet de vérifier qu'une date est antérieure ou égale à la date actuelle.

Les annotations de l'API Bean Validation

- **@Pattern(regExp)** : permet de vérifier qu'une chaîne de caractères correspond à l'expression régulière passée en paramètre de l'annotation.
- **@Positive** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit strictement supérieure à zéro.
- **@PositiveOrZero** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit supérieure ou égale à zéro.
- **@Size(min=3, max=8)** : permet de vérifier qu'une chaîne de caractères ait bien sa taille comprise entre une valeur minimale et une valeur maximale.

Exemple: Validation d'un formulaire JSF

```java
import javax.inject.Named;
import javax.validation.constraints.Email;
import javax.validation.constraints.Size;

@Named("loginBean")
@SessionScoped
public class LoginBean implements Serializable{
    private static final long serialVersionUID = 1L;

    @Email  @Size(min=1)
    private String login = "youssadi@gmail.com";

    @Size(min=3, max=8)
    private String password = "007";


    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String returnAction() {
        return password.equals( "007" ) ? "success" : "failure";
    }

}
```

Exemple: Validation d'un formulaire JSF

```
<f:view>
    <head>
         <title>Login screen</title>
         <link rel="stylesheet" type="text/css" href="../styles.css" />
    </head>
    <body>
        <h1>Login screen</h1>

        <h:form id="form">
                Login:
                <h:inputText id="login" value="#{loginBean.login}"
                required="true"
                requiredMessage="La saisie du login est obligatoire !" />  
                <h:message for="login" styleClass="errorBlock" />
            <br />

                Password:
                <h:inputSecret id="password"
                value="#{loginBean.password}" required="true"
                requiredMessage="La saisie du mot de passe est obligatoire !"
                validatorMessage="Votre mot de passe doit contenir entre 3 à 12 caractères" />  
                <h:message for="password" styleClass="errorBlock" />
            <br />
            <br />

            <h:commandButton action="#{loginBean.returnAction}" value="Connect" />
        </h:form>
    </body>
</f:view>
</html>
```