



Spring mvc – Formulaire

Règles de validation personnalisées



Règle de Validation personnalisée

- Nous pouvons valider les champs du formulaire selon des règles métiers ou de notre choix
- Par exemple nous pouvons définir comme règle qu'**un serial number d'un personnage , doit commencer par les lettres "xyz"**
- Spring va appeler notre règle
- Cette règle retournera un booléen pour indiquer si elle est respectée ou non.

Il n'y a pas d'annotation prédéfinie pour cela

- Nous allons créer notre propre annotation
- Pour notre validation de serialNumber , nous allons créer une annotation personnalisée ...
disons **@MaSuperAnnotation** ou **@SerialNumber**

Avancé

Personnage.java

```
@SerialNumber(value="xyz", message="serial number incorrect")  
private String serialNumber;
```



TODO list

- On doit créer la règle de validation
- Ajouter la règle à l'objet Personnage
- Afficher le message d'erreur dans le formulaire html
- Mettre à jour la page de confirmation

Créer la règle de validation

- Il faut créer l'annotation personnalisée `@SerialNumber`
- Créer une classe `SerialNumberConstraintValidator` qui va contenir la logique métier concernant la validation et déterminer si oui ou non une valeur soumise, remplit cette règle.

Créer l'annotation @SerialNumber

Usage exemple :

Personnage.java

```
@SerialNumber(value="xyz", message="serial number incorrect")  
private String serialNumber;
```

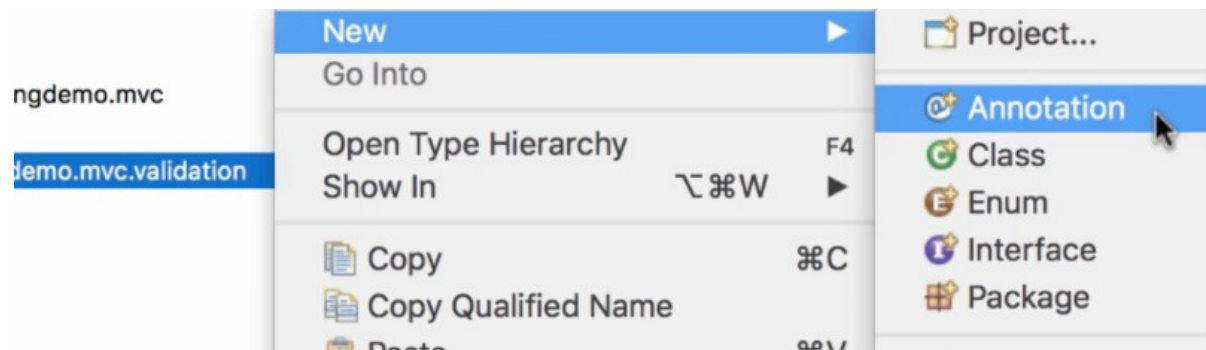
un serial number d'un personnage ,
doit commencer par les lettres "xyz"

Le champ dans notre
entité Personnage.java

Message d'erreur

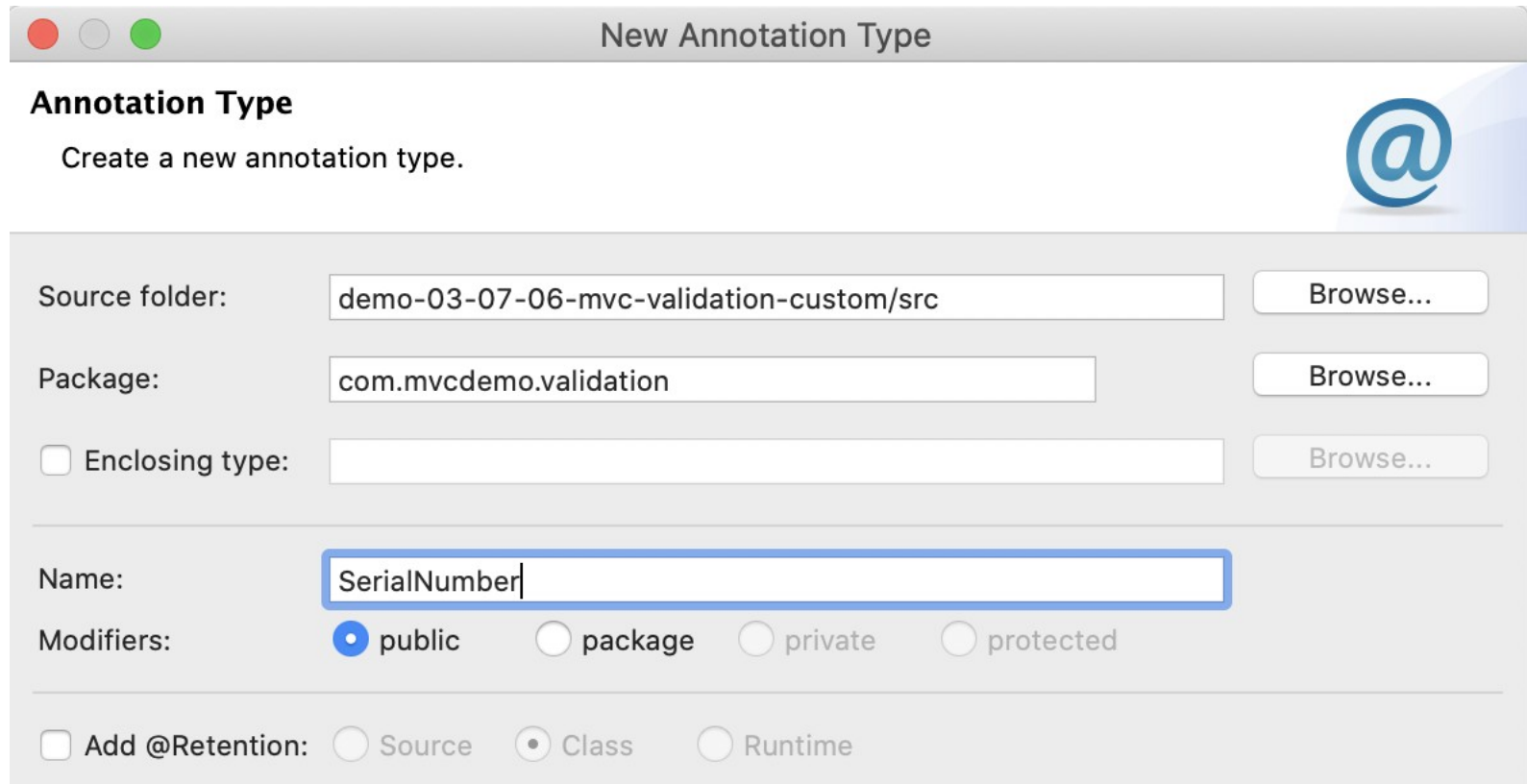
Créer l'annotation @SerializedName

- Dans le repertoire projet/src/
- Créer un package `com.mvcdemo.validation`
- Y créer un fichier @Annotation comme suit :



Package validation

Create>New>@Annotation



New Annotation Type

Annotation Type
Create a new annotation type.

Source folder: demo-03-07-06-mvc-validation-custom/src Browse...

Package: com.mvcdemo.validation Browse...

☐ Enclosing type: Browse...

Name: **SerialNumber**

Modifiers: ☒ public ☐ package ☐ private ☐ protected

☐ Add @Retention: ☐ Source ☒ Class ☐ Runtime

>Finish

=> resultat `SerialNumber.java`

```
1 package com.mvcdemo.validation;  
2  
3 public @interface SerialNumber {  
4  
5 }  
6
```

Ensuite on ajoute le validateur

```
package com.mvcdemo.validation;  
import javax.validation.Constraint;  
  
@Constraint(validatedBy = SerialNumberConstraintValidator.class)  
public @interface SerialNumber {  
  
}
```

Ce composant n'existe pas pour le moment.
Il contiendra le code de validation

SerialNumber.java (suite I)

```
package com.mvcdemo.validation;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import javax.validation.Constraint;

@Constraint(validatedBy = SerializableConstraintValidator.class)
@Target( {ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Serializable {

}
```

Durée maintien
de l'annotation

Préciser les types
d'éléments sur lesquels cette
Annotation sera appliquée

Définir les attributs d'une annotation

- Nous nous sommes permis d'écrire une annotation qui n'existe pas , avec des attributs qui n'existent pas encore non plus

Personnage.java

```
@SerialNumber(value="xyz", message="serial number incorrect")  
private String serialNumber;
```

- L'annotation existe désormais , mais pas ses attributs value et message. Il nous faut les créer dans SerialNumber.java

@SerialNumber (suite II)

```
package com.mvcdemo.validation;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import javax.validation.Constraint;

@Constraint(validatedBy = SerializableConstraintValidator.class)
@Target( {ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Serializable {
    // définir le serialNumber par défaut

    //définir le message par défaut

    //group par défaut (on peut rassembler des contraintes de validation)

    //payload par défaut (des infos additionnelles concernant l'erreur de validation
    //comme le niveau de sécurité, un code d'erreur ...etc)

}
```

@SerialNumber (suite III)

```
package com.mvcdemo.validation;
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;

@Constraint(validatedBy = SerializableConstraintValidator.class)
@Target( {ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Serializable {
    // définir le serialNumber par défaut
    public String value() default "xyz";
    //définir le message par défaut
    public String message() default "serialNumber incorrect";
    //groupe par défaut
    public Class<?>[] groups() default {}; //collection vide
    //payload par défaut
    public Class<? extends Payload >[] payload() default {};
}
```

ConstraintValidator

- Dans le package validation créer une nouvelle classe **SerialNumberConstraintValidator.java**

```
package com.mvcdemo.validation;
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class SerialNumberConstraintValidator implements
ConstraintValidator<SerialNumber , String>{

    private String serialNumberPrefix;

    @Override
    public void initialize (SerialNumber arg0) {

    }

    @Override
    public boolean isValid(Object arg0, ConstraintValidatorContext arg1) {
        // TODO Auto-generated method stub
        return false;
    }

}
```

ConstraintValidator (suite I)

```
public class SerialNumberConstraintValidator
    implements ConstraintValidator<SerialNumber, String>{

    private String serialNumberPrefix;

    @Override
    public void initialize (SerialNumber unSerialNumber) {
        serialNumberPrefix = unSerialNumber.value();
    }

    @Override
    public boolean isValid(String arg0, ConstraintValidatorContext arg1) {
        // TODO Auto-generated method stub
        return false;
    }
}
```



```
@SerialNumber(value="xyz", message="serial number incorrect")
private String serialNumber;
```

Spring MVC pour la validation appelle la méthode isValid(...)

Données du formulaire HTML
Saisie par l'utilisateur

On peut prévoir des messages
d'erreur supplémentaires ici

```
@Override
public boolean isValid(String leSerialNumber,
    unConstraintValidatorConstraint) {
    boolean result ;
    result = leSerialNumber.startsWith(serialNumberPrefix);
    return result;
}
```

Logique métier pour retourner
Vrai ou faux

ConstraintValidator

```
public class SerialNumberConstraintValidator implements ConstraintValidator<SerialNumber, String>{

    private String serialNumberPrefix;

    @Override
    public void initialize (SerialNumber unSerialNumber) {
        serialNumberPrefix = unSerialNumber.value();
    }

    @Override
    public boolean isValid(String leSerialNumber,
        ConstraintValidatorContext unConstraintValidatorContext) {

        @Override
        public boolean isValid(String leSerialNumber,
            ConstraintValidatorContext unConstraintValidatorContext) {
            boolean result;
            if(leSerialNumber !=null)
                result= leSerialNumber.startsWith(serialNumberPrefix);
            else
                result = true;;
            return result;
        }
    }
}
```

Vous pouvez ajouter votre propre
logique de validation
d'un champ de formulaire

Ajout de la validation à l'entité et au formulaire

TODO list récapitulatif :

- Créer la règle de validation (annotation & validator)
- Ajout de la règle à la classe Personnage.java
- Ajout du message d'erreur dans le formulaire html
- Mettre à jour la page de confirmation

Entité et formulaire (c'est du déjà vu...)

```
public class Personnage {  
  
    private String prenom;  
  
    @NotNull( message="...")  
    @Size(min=1, message="...")  
    private String nom;  
  
    @NotNull(message="...s")  
    @Min(value=0, message="...")  
    @Max(value=10,message="...")  
    private Integer pointsDeVie;  
  
    @Pattern(regexp=". . . ",  
    Message="....")  
    private String email;  
  
    @SerializedName  
    private String serialNumber;  
  
    //getters & setters
```

value et
message par défaut

```
<form:form action="processForm" modelAttribute="personnage">  
  
    Prénom: <form:input path="prenom" />  
    <br><br>  
  
    Nom (*): <form:input path="nom" />  
    <form:errors path="nom" cssClass="error"/>  
  
    <br><br>  
    points de vie : <form:input path="pointsDeVie"/>  
    <form:errors path="pointsDeVie" cssClass="error" />  
    <br><br>  
  
    email : <form:input path="email"/>  
    <form:errors path="email" cssClass="error" />  
    <br><br>  
  
    serial number : <form:input path="serialNumber"/>  
    <form:errors path="serialNumber" cssClass="error" />  
    <br>  
    <br>  
    <input type="submit" value="Valider" />  
    <br>  
</form:form>
```

Maj de la page de confirmation

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<title>personnage valide - vue</title>
</head>
<body>
Le personnage soumis est valide : ${personnage.nom}
                                     ${personnage.prenom}

<br><br>
Points de vie valide : ${personnage.pointsDeVie}
<br><br>

email valide : ${personnage.email}

<br><br>
serialNumber valide : ${personnage.serialNumber}
</body>
</html>
```

Tester , executer l'application

Remplir le formulaire SVP.

Prénom:

Nom (*):

points de vie :

email :

serial number :

() signifie que le champ est obligatoire.*

Remplir le formulaire SVP.

Prénom:

Nom (*):

points de vie :

email :

serial number : **serial number incorrect**

() signifie que le champ est obligatoire.*

Tout est Ok !! ...