Grails Views

Alexis Plantin

Summary

- Attributs dans les GSP
- Passer le Model à la View
- Les directives de Pages
- GStrings dans les GSPs
- Setter des Variables avec les Tags
- Tags logiques
- Tags itératifs
- Filtrer avec les tags
- Grails Dynamic Tags
  - Tags pour les liens
  - Formulaires et attributs
  - Gestion des erreurs
- Afficher les Templates
- Créer une Taglib

Attributs dans les GSP

Attributs dans les GSP

- Attributs disponibles dans les GSP

Attribute	Description
`application`	The `ServletContext` instance
`flash`	The `flash` object for working with flash scope, as discussed in Chapter 7
`out`	The response `Writer` instance
`params`	A map of request parameters
`request`	The `HttpServletRequest` instance
`response`	The `HttpServletResponse` instance
`session`	The `HttpSession` instance

Passer le Model à la View

Passer le Model à la View

- Retourner une Map depuis le Controller
- La Map est bindée comme model de la GSP
- Disponible dans la GSP

Les directives de Pages

Les directives de Pages

- Instruction au début des GSPs
- Utilisées pour les import, content type…

```
<%@ page contentType="text/xml; charset=UTF-8" %>
<%@ page import="java.sql.Time" %>
```

GStrings dans les GSPs

GStrings dans les GSPs

- Utilisation des GStrings
  - Retouner la valeur dans le Controller

```
[product: Product.get(1)]
```

  - Dans la GSP

```
<p>
    ${product.name}
</p>
```

Setter des Variables avec les Tags

Setter des Variables avec les Tags

- Le tag `<g:set>`
  - `var`: le nom de la variable à setter
  - `value`: une expression
  - `scope`: définit le scope de la variable qui est à *`page`* par défaut (bonne pratique)

```
<g:set var="productName" value= "${product.name}"  />
```

Tags logiques

Tags logiques

- 3 tags: `<g:if>`, `<g:elseif>` et `<g:else>`
  - `<g:if>` et `<g:elseif>` prennent un paramètre `test`

```
<g:if test= "${product?.height < 20}">
    Lightweight
</g:if>
<g:elseif test = "${product?.height < 100}">
    Medium
</g:elseif>
<g:else>
    Heavy
</g:else>
```

Tags itératifs

Tags itératifs

- Le `<g:each>` tag équivalent à la méthode each
  - L'attribut `var` (optionnel) pour nommer la variable dans le bloc

```
<div class="locations">
    <g:each var="warehouse" in="${location?.warehouses}">
        <span class="tag">${warehouse.name}</span>
    </g:each>
</div>
```

Tags itératifs

- Le tag <g:while> équivalent à la méthode while

```
<g:set var="i" expr="${location?.warehouses?.size()}" />
<g:while test="${i > 0}">
    <g:set var="i" expr="${i-1}" />
</g:while>
```

- Equivalent au code Groovy

```
def i = 5
while(i > 0)  {
    i = i-1
}
```

Filtrer avec les tags

Filtrer avec les tags

- Itérer sur les objets avec le tag `<g:collect>`

```
<g:collect in="${products}" expr="${it.name}">
    <span class="tag">${it}</span>
</g:collect>
```

- On peut obtenir le même résultat avec la méthode each et GPath

```
<g:each in="${products.name}">
    <span class="tag">${it}</span>
</g:each>
```

Filtrer avec les tags

- Trouver des éléments spécifiques avec le tag `<g:findAll>`

```
<g:findAll in="${products}" expr="${it.weight  > 100}">
    <span class="tag">${it}</span>
</g:findAll>
```

Grails Dynamic Tags

Grails Dynamic Tags

- Fournis au travers de classes appelées tag libraries
- Utilisable dans les autres tags

```
<a href="<g:createLink action="list" />">Product List</a>
```

- Peuvent également être invoqués dans des GStrings

```
<a href="${createLink(action:'list')}">Product List</a>
```

- Plus lisible

Linking Tags

- `<g:link>` crée un simple lien HTML aces les attributs suivants
  - Controller: le nom de Controller vers lequel faire un lien
  - Action: le nom de l'action vers laquelle faire un lien
  - Id: Un identifant à ajouter à la fin de l'URI
  - Params: Tous nes paramètres à passer dans une Map

```
<g:link controller="product" action="list">Products List</
g:link>
<g:link action="show" id="1">Show product with id 1</g:link>
<g:link controller="product" action="list" params= "[max: 5,
sort: 'weight']" >
    Show the 5 lightweight products
</g:link>
<g:link controller="product" action="list" params="${params} ">
    Same parameters
</g:link>
```

Linking Tags

- `<g:createLink>` prend les mêmes paramètres que `<g:link>`
  - Produit seulement le lien

```
<a href="${createLink(controller:'product',
action:'list')}">List Products</a>
```

  - `<g:createLinkTo>` permet de faire des liens vers les ressources statiques de notre application

```
<link rel="stylesheet" href="$
{createLinkTo(dir:'css',file:'main.css')}"></link>
```

  - Généralement utilisés comme des appels de méthodes car leur résultat sont exploités dans d'autres tags

Formulaires & attributs

- Le tag `<form>` pour créer un nouveau formulaire HTML

```
<g:form action="save" name="saveForm" >
    ...
</g:form>
```

- Le résultat HTML est

```
<form action="/wrh/location/save"
      method="post"
      name="saveForm"
      id="saveForm" >
    …
</form>
```

Formulaires & attributs

- Une syntaxe alternative

```
<g:form url="[controller: 'location', action: 'save']">
    ...
</g:form>
```

- Paramètres (tous optionnels):
  - method: GET ou POST
  - controller: le Controller à utiliser dans le lien
  - action: l'action à utiliser dans le lien
  - id: l'id à utiliser dans le lien
  - url: une Map contenant les autres paramètres (action,…)

Formulaires & attributs

- Le `<g:textField>` tag gère les entrées de texte
  - L'attribut name: Le nom du paramètre à envoyer
  - L'attribut value: la valeur de l'élément HTML

```
<g:textField name="country" value="$
{locationInstance?.country}" />
```

  - Le résultat est

```
<input type="text" name="country" value="France" id="country" /
>
```

Formulaires & attributs

- `<g:checkBox>` pour les valeurs booléennes
  - L'attribut value est un booléen
  - La checkBox est affichée dans son état associé

```
<g:checkBox name="aBooleanValue" value="${true}" />
```

Formulaires & attributs

- `<g:radio>` pour les boutons radio
  - Tous les éléments d'un même groupe doivent avoir le même nom
  - Un attribut `checked` avec une valeur de type booléen value

```
<g:radio name="myGroup" value="1" />
<g:radio name="myGroup" value="2" checked="true"/>
```

  - Le résultat est

```
<input type="radio" name="myGroup" value="1" id="myGroup" />
<input type="radio" name="myGroup" checked="checked" value="2"
id="myGroup" />
```

Formulaires & attributs

- `<g:select>` fournit une select list
  - L'attribut `from` avec un liste de valeurs

```
<g:select name="category"
          from="${['Lightweight', 'Medium', 'Heavy']}"
          value="Medium" />
```

  - Le résultat est

```
<select name="category" id="category" >
    <option value="Lightweight" >Lightweight</option>
    <option value="Medium" selected="selected" >Medium</option>
    <option value="Heavy" >Heavy</option>
</select>
```

Formulaires & attributs

- Les attributs de `<g:select>`
  - from: la liste des valeurs possibles
  - value: la valeur sélectionnée
  - optionKey: la valeur de la clé dans la balise option
  - optionValue: la valeur affichée dans la select list
  - noSelection: un couple key/value à afficher lorsqu'aucune sélection n'est faite
  - …

```
<g:select name="product.id"
          from="${wrh.Product.list()}"
          optionKey="id"
          optionValue="name"
          value="${entryInstance?.product?.id}"  />
```

Formulaires & attributs

- `<g:datePicker>` fournit des select lists pour les années, mois, jour, heures, minutes et secondes
  - Possibilité de définir une précision

```
<g:datePicker name="since"
              precision="day"
              value="${staffInstance?.since}"  />
```

Gestion des erreurs

- `<g:hasErrors>` affiche le contenu conditionnellement
- 3 paramètres optionnels
  - bean: Le bean où vérifier la présence d'erreur
  - model: Le nom du Model de référence pour la vérification d'erreur
  - field: Pour vérifier la présence d'erreur sur un attribut particulier

```
<g:hasErrors bean="${product}">
    …
</g:hasErrors>
```

Gestion des erreurs

- `<g:eachError>` affiche le contenu conditionnellement
- Même paramètres que `<g:hasErrors>`

```
<g:eachError bean="${product}">
    <li>${it}</li>
</g:eachError>
```

Gestion des erreurs

- Généralement les 2 tags précédents sont utilisés ensemble

```
<g:hasErrors bean="${product}">
    <ul>
        <g:eachError bean="${product}">
            <li>${it.defaultMessage}</li>
        </g:eachError>
    </ul>
</g:hasErrors>
```

Afficher les Templates

Afficher les Templates

- Un template contient seulement un fragment de page
- Améliore la réutilisabilité du code
- Le nom de fichier doit commencer par _
  - `grails-app/views/product/_productList.gsp`
- Afficher dans une GSP avec le tag render

```
<g:render template="/product/productList"/>
```

Créer une Taglib

Créer une Taglib

- Une simple commande

```
grails create-tagLib Wrh
```

- 2 fichiers créés:
  - Une Taglib dans grails-app/taglib
  - Un Unit Test dans src/test
- 3 types de tags
  - Tags simples
  - Tags logiques
  - Tags itératifs

Créer une Taglib

- Une Closure avec 2 paramètres
  - attrs: Une Map d'attributs
  - body: Une Closure

```
def simpleTag = { attrs, body ->
    …
}
```

```
<g:simpleTag />
```

Tags simples

- Affiche du HTML

```
def sayHello = { attrs, body ->
    def firstName = attrs?.firstName
    def lastName = attrs?.lastName

    def output = "Hello "
    if(firstName != null && lastName != null) {
        output += firstName + ", " + lastName
    }
    else if(firstName != null) {
        output += firstName
    }
    else if(lastName != null) {
        output += lastName
    }
    output += "<br/>"
    out << output
}
```

```
<g:sayHello />
<g:sayHello firstName="Alexis" />
<g:sayHello lastName="Plantin" />
<g:sayHello firstName="Alexis"
            lastName="Plantin" />
```

Tags logiques

- Affiche le contenu conditionnellement

```
def isBrowser = { attrs, body ->
    if(request.getHeader('User-Agent') =~ attrs.userAgent ) {
        out << body()
    }
}
```

```
<g:isBrowser userAgent="MSIE">
    <p>You are currently on Internet Explorer </p>
</g:isBrowser>
<g:isBrowser userAgent="Firefox">
    <p>You are currently on Firefox </p>
</g:isBrowser>
```

Tags itératifs

- Affiche le contenu plusieurs fois

```
def eachEntry = { attrs, body ->
    def entries = attrs.in
    entries?.each { entry ->
        out << body(entry: entry)
    }
}
```

```
<ul>
    <g:eachEntry in="$
{wrh.Entry.findAllByProduct(productInstance)}">
        <li>${entry?.warehouse?.name + ' : ' + entry?.quantity}
</li>
    </g:eachEntry>
</ul>
```

Questions?