Grails controllers

Alexis Plantin

Sommaire

- Création d'un Controller
- Action par défaut
- Attributs disponibles
- Scopes disponible
- Afficher du texte
- Afficher une View
- Rediriger une requête
- Retourner un Model
- Data Binding
- Command Objects
- Restrictions sur les méthodes HTTP
- Controller IO

Création d’un Controller

Création d'un Controller

```
grails create-controller wrh.Product
```

- 2 fichiers créés:
  - Un Controller dans `grails-app/controller`
  - Un Unit Test dans `src/test`

Action par défaut

Action par défaut

- Si le Controller ne possède qu'une action, c'est l'action par défaut

```
class SimpleController {
    def hello() {}
}
```

Default Action

- Si le Controller possède une action *index*, elle devient l'action par défaut

```
class SimpleController {
    def hello() {}
    def index() {}
}
```

Default Action

- Si le Controller possède la propriété statique defaultAction, sa valeur correspond au nom de l'action par défaut

```
class SimpleController {
    static defaultAction = 'hello'
    def hello() {}
    def index() {}
}
```

Attributs disponibles

Attributs disponibles

Attribute	Description
`actionName`	The name of the currently executing action
`actionUri`	The relative URI of the executing action
`controllerName`	The name of the currently executing controller
`controllerUri`	The URI of executing controller
`flash`	The object for working with `flash` scope
`log`	An `org.apache.commons.logging.Log` instance
`params`	A map of request parameters
`request`	The `HttpServletRequest` object
`response`	The `HttpServletResponse` object
`session`	The `HttpSession` object
`servletContext`	The `ServletContext` object

Attributs disponibles

- En appelant l'action simple/index
  - actionName: `index`
  - actionUri: `/simple/index`
  - controllerName: `simple`
  - controllerUri: `/simple`
  - flash: `[:]`
  - log: Une instance de log
  - `params: [action:index, controller:simple]`
  - session: une session vide

Attributs disponibles

- Accéder aux paramètres de la requête
  - Avec l'appel suivant: `simple/fooAction?foo=bar`

```
class SimpleController {
    def fooAction() {
        println params.foo
    }
}
```

Scope disponibles

Scope disponibles

- `request:` durée de vie de la requête en cours
- `flash:` durée de vie de la requête en cours et de la suivante
- `session:` jusqu'à l'invalidation de la session utilisateur
- `servletContext:` globale à l'application et disponible jusqu'à ce que l'application se stoppe.

Afficher du texte

Afficher du texte

- En utilisant la puissante méthode render
  - Afficher un simple String

```
render 'Hello World!'
```

  - Specifier le Content Type

```
render text:'<book>The Definitive Guide to Grails</book>',
contentType:'text/xml'
```

Afficher une View

Afficher une View

- Trouver la View par défaut
- Sélectionner une View custom

```
// Selecting a view in the default folder
render(view: 'alternativeShow',
       model: [ productInstance: Product.get(params.id) ])

// Selecting a view in an alternativeFolder
render(view: '/alternativeFolder/alternativeShow',
       model: [ productInstance: Product.get(params.id) ])
```

Afficher une View

- Afficher un template

```
// Selecting a view in the default folder
render(template: 'common/product',
       model: [ productInstance: Product.get(params.id) ])
```

Rediriger une requête

Rediriger une requête

- Redirection vers une autre action du Controller

```
redirect(action:'anotherAction')
```

- Redirection vers une autre action d'un autre Controller

```
redirect(controller: 'otherController', action:'anotherAction')
```

Rediriger une requête

Argument Name	Description
`action`	The name of or a reference to the action to redirect to
`controller`	The name of the controller to redirect to
`id`	The `id` parameter to pass in the redirect
`params`	A map of parameters to pass
`uri`	A relative URI to redirect to
`url`	An absolute URL to redirect to

Retourner un Model

Retourner un Model

- Retourner une Map de données pour être rendue par la View

```
class ProductController {
    def show() {
        [ productInstance: Product.get(params.id) ]
    }
}
```

Data Binding

Data Binding

- Consctruire un Product depuis les paramètres de la requête

```
class ProductController {
    def save() {
        def product = new Product()
        product.name = params.name
        product.weight = params.weight.toFloat()
        product.save()
    }
}
```

Data Binding

- Consctruire un Product depuis les paramètres de la requête

```
class ProductController {
    def save() {
        def product = new Product(params)
        product.save()
    }
}
```

Data Binding

- Mettre à jour un Product depuis les paramètres de la requête

```
class ProductController {
    def update() {
        def product = Product.get(params.id)
        product.properties = params
        product.save()
    }
}
```

Data Binding

- Valider les données

```
class ProductController {
    def update() {
        def product = Product.get(params.id)
        product.properties = params
        if(product.save()) {
            redirect(action: "show", id:product.id)
        }
        else {
            render(view: "edit", model:
[productInstance:product])
        }
    }
}
```

Data Binding

- Plusieurs Domain Objects
  - Code HTML du formulaire

```
<input type="text" name= "product.name" />
<input type="text" name= "product.weight" />
<input type="text" name= "description.name" />
```

  - Code du Controller

```
def product = new Product( params["product"] )
def description = new Description( params["description"] )
```

Data Binding

- La méthode bindData

```
// binds request parameters to a target object
bindData(target, params)
// only use parameters starting with "product."
bindData(target, this.params, "product")
// using inclusive map
bindData(target,
         this.params,
         [include:[‘name', ‘weight']], "product")
// using exclusive map
bindData(target,
         this.params,
         [exclude:[‘weight']], "product")
```

Command Objects

Command Objects

- Définir un Command Object
  - implements grails.validation.Validateable

```
class LocationCreateCommand implements grails.validation.Validateable {
    String name
    String country
    String address
    List warehouses = []

    static constraints = {
        name blank:false
        country blank:false
        address blank: false
        warehouses minSize:1
    }

    Location createLocation() {
        def location = new Location(name: name, country: country, address: address)
        warehouses.each { warehouse ->
            location.addToWarehouses(name:warehouse)
        }
        return location
    }
}
```

Command Objects

- Utiliser les Command Objects

```
def save(LocationCreateCommand cmd) {
    if(cmd.validate()) {
        def location = cmd.createLocation()
        location.save()
        redirect(action:"show", id:location.id)
    }
    else {
        render(view:"create", model:[cmd:cmd])
    }
}
```

- Et n'oubliez pas

```
<g:renderErrors bean="{cmd}" />
```

Restrictions sur les méthodes HTTP

Restrictions sur les méthodes HTTP

- Implémenter la solution

```
class ProductController {
    def delete() {
        if(request.method == "GET") {
            // do not delete in response to a GET request
            // redirect to the list action
            redirect(action: "list")
        } else {
            // carry out the delete here...
        }
    }
}
```

Restrictions sur les méthodes HTTP

- Utiliser la propriété statique

```
static allowedMethods = [delete:['POST', 'DELETE'], action1:
'POST']
```

Controller IO

Controller IO

- File Uploads
  - Basé sur la gestion des uploads de Spring
- Un formulaire d'upload form requiert 2 choses
  - Un tag <form> avec l'attribut enctype ayant pour valeur multipart/form-data
  - Un tag <input> tag don't l'attribut type le nom du fichier
- Bonne nouvelle : Grails fournit un tag pour les formulaires multipart

```
<g:uploadForm action="upload">
    <input type="file" name="myFile" />
    <input type="submit" value="Upload! " />
</g:uploadForm>
```

Controller IO

- Récupérer le fichier uploadé côté Controller

```
def upload() {
    def file = request.getFile('myFile')
    // do something with the file
}
```

- La méthode ne retourne pas un `java.io.File` mais un `org.springframework.web.multipart.MultipartFile`

Controller IO

- L'interface de MultipartFile est

```
interface MultipartFile {
    public byte[] getBytes();
    public String getContentType();
    public java.io.InputStream getInputStream();
    public String getName();
    public String getOriginalFilename();
    public long getSize();
    public boolean isEmpty();
    public void transferTo(java.io.File dest);
}
```

Controller IO

- La plupart de ses méthodes sont utiles
  - `getSize()` pour limiter la taille des fichiers à une certaine valeur
  - Rejecter les fichiers vides `isEmpty()`
  - Autoriser uniquement certains types de fichiers avec `getContentType()`
  - Transférer le fichier sur le serveur avec `transferTo(dest)`

Controller IO

- Uploads et Data Binding
  - Si l'attribut cible est un byte[], les bytes du fichier seront insérés dans l'attribut
  - Si l'attribut cible est un String, le contenu du fichier en tant que String sera inséré dans l'attribut
- Ajouter une propriété photo pour un Product

```
// The Domain Class
class Product {
    byte[] photo
    ...
}

// In the HTML form
<input type="file" name="photo" />

// In the Controller
def user = new Product( params )
```

Questions?