Grails Controllers 2

- Amit Jain

Agenda

- ✓ What is Data Binding?
- ✓ Binding Request Data To Model
- ✓ Binding and Association
- ✓ Data binding with Multiple Domain Object
- ✓ Command Objects
- ✓ Error Handling

What is Data Binding?

Data Binding is the act of "Binding" Incoming request parameters onto the properties of an object.

What Data Binding deal with?

- Request Parameters (params) which we typically delivered via form submission are always "strings". Whereas, our groovy or java object properties or we say our domain class model properties "may well not be string"

- So, data binding deals with all necessary type conversions.

Binding Request To Data Model

- Grails uses Spring's underlying data binding capability to perform data binding.

- Object properties can be initialised or assigned directly from corresponding request parameters i.e. with same name/path

- Grails will attempt to bind any incoming request parameters onto the properties of the instance, and perform type conversion if necessary.

Binding Data with Params

params has some utility methods:

- params.int("age")
- params.date("dob","dd-MM-yyyy")
- params.list("student")

To action arguments : action(Integer age){ //... }

Demo

1. Create a domain Person and dynamically bind data with its instance.
   a. Firstly through action arguments
   b. Secondly through param conversion

Binding Request To Data Model

Ways of Binding Data :

- Using Domain Class 'Implicit Constructor'
- Using property 'properties' for existing instance
- Using “bindData(....)” Method

Data Binding: Implicit Constructor

Employee emp = new Employee(params)

By passing the params object to the domain class constructor Grails automatically recognizes that you are trying to bind from request parameters. When incoming request is like :
/employee/save?employeeId=Rx-01

Then the employeeId request parameters would automatically get set on the domain class properties with similar name.

Data Binding: Using “properties”

If you need to perform data binding onto an existing instance then you can use the “properties” property :

```
Book album = Book.get(params.id)

Book.properties = params            //Update Existing Record
```

Note : This has exactly the same effect as using the implicit constructor (Just can used for existing record updation).

Data Binding: bindData method

- Enables inclusion/exclusion of properties to be included while binding the data

- bindData(album, params, [include : ['title' , 'type']]) Include : Specify only 'title' and 'type' property bind to album.

- bindData(album, params, [exclude : "title"])

  Exclude : Specify bind all the property excluding 'title'.

Binding Data: Type conversion errors

- Sometimes when performing data binding it is not possible to convert a particular String into a particular target type. This results in a type conversion error. Grails will retain type conversion errors inside the errors property of a Grails domain class.

- Employee b = new Employee(params)

- if (b.hasErrors()) {
        println "The value ${b.errors.getFieldError('age')}"
  }

- <object>.hasErrors()
- <object>.errors.allErrors()
- <object>.errors.getFieldError()
- <object>.errors.hasFieldErrors()

Demo

1. Consider the Employee Domain class,
   a. bind data using Domain Class 'Implicit Constructor'.
   b. bind data using .properties
   c. using "bindData(....)" Method

1. For Employee Domain, bind all data excluding 'age' property and once again by including firstname and lastname.

Data Binding: Binding and Association

Single-Ended Association : (one-to-one or many-to-one)

- Suppose when incoming request is like :
  /employee/save?dept.id=10

- Grails will automatically detect the ".id" suffix on the request parameter and look-up for the dept instance for the given "id".

Data Binding: Binding and Association

Many-Ended Association : (one-to-many or many-to-many) - If you have a one-to-many or many-to-many association there are different techniques for data binding depending of the association type.

- If you have a Set based association (default for a hasMany) then the simplest way to populate an association is to simply send a list of identifiers.

```
<g:select name="books" from="${Book.list()}"
size="5" multiple="yes" optionKey="id" value="${author?.books}" />
```

- In this case if you submit the form Grails will automatically use the identifiers from the select box to populate the books association.

Data Binding: Multiple Domain Object

- *It is possible to bind data to multiple domain objects from the params object.*

http://localhost:8181/sample/multipleDomainBinding?dept.name=HR &emp.name=Sahi&emp.age=25

- *Above, each parameter has a prefix such as emp. or dept. which is used to isolate which parameters belong to which type.*

Employee e = new Employee(params[emp])

- *This is how we use the prefix before the first dot of the "emp.name" parameter to isolate only parameters below this level to bind.*

Command Objects

- *Command objects are useful in circumstances when you want to populate a subset of the properties needed to update a domain class.*

  Or

  where there is no domain class required for the interaction, but you need features such as data binding and validation.

- *Has all the capabilities of a domain class except persistence*

Command Objects

- Command objects are typically declared in the same source file as a controller directly below the controller class definition
- Or You Can Have Seperate Class named “SampleCommand' that has suffix as “Command”
- class RegisterCommand {

```
   String username
    String password

    static constraints = {
        username(blank: false, minSize: 6)
        password(blank: false, minSize: 6)
    }
}
```

Command Objects

Using Command objects :

- The parameter types must be supplied so that Grails knows what objects to create, populate and validate.

- def register(RegisterCommand cmd) {

```
        //Your Code Here
        }
```

- The form is constructed with inputs' names being that of properties of command object.

Command Objects

- *Before the controller action is executed, Grails will automatically create an instance of the command object class.*
- *Then populate the properties of the command object with request parameters having corresponding names.*
- *and then the command object will be validated.*
- *Validation can be done by cmd.validate()*
- *The errors are contained in an Errors object*

Demo

File Uploads

- Grails supports file uploads using Spring's **MultipartHttpServletRequest** interface.
- In order to transfer files, you need to create a form whose **enctype** attribute is set to “**multipart/form-data**”.
- In addition, an input field of type=”file” must be created within the form.

```
<form name=”upload” enctype=”multipart/form-data”>
        <input type=”file” name=”myFile” />
</form>
```

- This can also be done by using the **g:uploadForm** tag provided by grails.

Programming: File Uploads

- **1st Step :** Create a multipart form like :

```
<g:form action="upload" method="post" enctype="multipart/form-data">
        <input type="file" name="myFile" />
        <input type="submit" />
</g:form>
```

OR

```
<g:uploadForm action="upload" method="post">
        <input type="file" name="myFile" />
        <input type="submit" />
</g:uploadForm>
```

- There are 2 ways to upload file now :
  1. Upload File to System
  2. Through Data Binding, in Database

File Uploads : To File System

- **2nd Step** : You can directly get the file from the params itself

```
def upload() {
        def f = params.myFile
        render f.inputStream.text
}
```

- The file that comes in from the client is an instance of CommonsMultipartFile which contains other information about the file apart from its contents like original file name, size, content type etc.

File Uploads : Through Data Binding

- **2nd Step** : You can bind the incoming file using command objects as well. There are two ways to do this.

  1. Create a **byte array** to hold the contents of the file.

```
class FileCommand {
        byte[] myFile
}
```

  2. Create a **MultipartFile** reference. This allows access to other properties of the file like content type as well.

```
class FileCommand {
        MultipartFile myFile
}
```

File Download : Writing Response

- You can also allow downloading of files by creating an anchor tag that points to the actual location of the file. The browser will automatically start downloading the file.
- However, in certain cases the requirement is that the file be created on the fly, such as an order detail PDF. In such cases, you need to make use of the response object available to controllers:

```
def createOrderPDF() {
            byte[] orderPDF = ... // create the bytes from some source
            response.setHeader("Content-disposition", "attachment; filename=” + fileName)
             response.contentType = contentType  response.contentLength = bytes.length
             response.outputStream << bytes
   }
```

Demo

Questions ?

Thank You

References

- Web Layer
- Command Objects