Grails Controller Part I

Sang Shin
JPassion.com
"Learn with Passion!"

Topics

- Controller and actions
- Scopes
- Models and views
- Rendering
- Controller interceptors
- Redirecting
- Data binding (params)
- XML and JSON responses

Controllers & Actions

Method vs Closure as Actions

- Actions in a controller can be in the form of
  - method or
  - closure
- Methods are preferred (over closure) because they are
  - Memory efficient
  - Allow use of stateless controllers (singleton scope)
  - You can override actions in subclasses
  - Methods can be intercepted with standard proxying mechanisms, something that is complicated to do with closures – this is because, in closure, the actions are in the form of fields

Default URI & Default Action

- A controller has a default URI that maps to the root URI of the controller
  - *BookController* has default URI of */book*
  - *AuthorController* has default URI of */author*
- The default action that is called when the default URI is requested (since no action is specified in the URI) is dictated by the following rules:
  - If there is only one action, it's the default
  - If there is “index” action, it's the default
  - Alternatively, default action can be set with “defaultAction” property
    - *static defaultAction = "myDefaultAction"*

Lab:

Exercise 1: Controllers & Actions
5630_grails_controller1.zip

Scopes

Scopes

- Scopes (sometimes called scope objects) are hash-like objects where you can store data
- Types of Scope objects available to controllers
  - > *servletContext, session, request, params, flash*
- Accessing data in scope objects

```
class StudentController {
    def my_action() {
        def app = servletContext["app"]    // servletContext.app
        def loggedUser = session["logged_user"] // session.logged_user
        def foo = request["foo"]    // request.foo
        def name = params["name"]    // params.name
    }
}
```

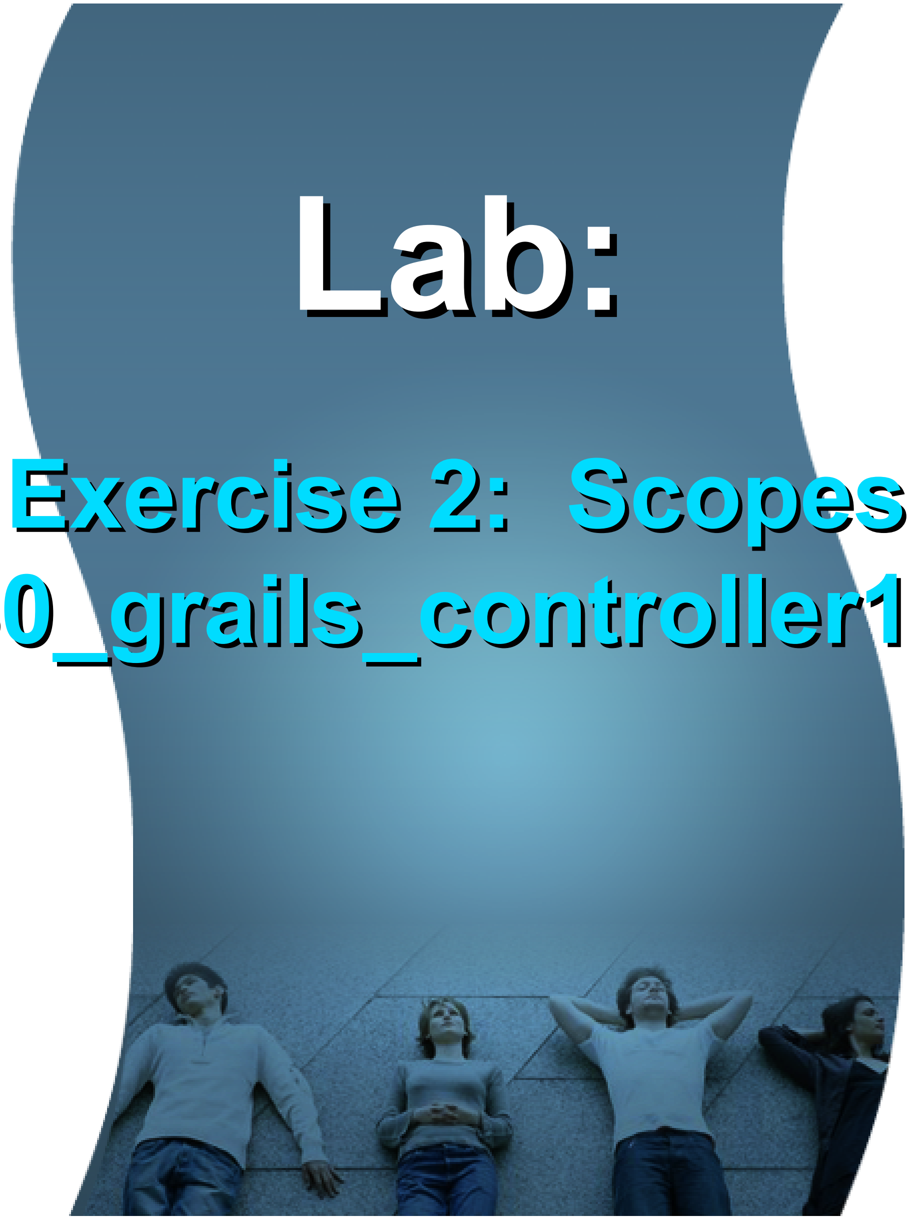
Flash Scope

- Temporary store to make attributes available for this request and the next request only. Afterward, the attributes are cleared
- Useful for setting a message directly before redirecting

```
def delete() {
    def b = Book.get(params.id)
    if (!b) {
        // This flash message is available to the redirected page then gets cleared
        flash.message = "User not found for id ${params.id}"
        redirect(action:'list')
    }
    … // remaining code
}
```

Controllers Have Associated Scopes

- Types of controller scope
  - prototype (default)
    - A new controller will be created for each request
    - It is thread-safe since each request happens on its own controller
  - session
    - One controller for the scope of a user session
    - *static scope = “session”*
  - singleton
    - Only one instance of the controller ever exists (recommended for actions as methods)
    - *static scope = "singleton"*



Lab:

Exercise 2: Scopes

5630_grails_controller1.zip

Models and Views

Returning a Model Object

- A model is a Map object that the view uses when rendering
  - > The keys within that Map correspond to variable names accessible by the view

```
// Return "book" as a key, which can be referenced in the view
def show() {
    [book: Book.findByTitle(params.title)]
}
```

Controller Action

```
<!-- Display information on the book -->
<body>
    Title = ${fieldValue(bean: book, field: "title")},
    Published Date = ${fieldValue(bean: book, field: "publishDate")}
</body
```

View

Returning Model Implicitly

- If no explicit model is returned, the controller's properties will be used as the model implicitly
- Use it only when controller is in “prototype” scope where new instance of a controller gets created per a request
- Not recommended practice – hard to read code

```
// the books and authors properties will be available in the view
class BookController {
    List books
    List authors

    def list() {
        books = Book.list()
        authors = Author.list()
    }
}
```

Returning ModelAndView Object

- You can return an instance of the Spring ModelAndView class
  - *ModelAndView* object can be set with view and model objects

```
import org.springframework.web.servlet.ModelAndView

def index() {
    // get some books just for the index page, perhaps your favorites
    def favoriteBooks = ...

    // forward to the list view to show them
    new ModelAndView("/book/list", [ bookList : favoriteBooks ])
}
```

Selecting a View

- Implicit view selection
  - By default, Grails selects a view with the same name of the action
- Explicit view selection
  - To render a different view, use “render” method with “view” argument

```
def show() {
    def map = [book: Book.get(params.id)]
   // Select grails-app/views/book/display.gsp
    render(view: "display", model: map)
}

def show() {
    def map = [book: Book.get(params.id)]
    // Select grails-app/views/shared/display.gsp
    render(view: "/shared/display", model: map)
}
```

Lab:

Exercise 3: Models & Views
5630_grails_controller1.zip

Rendering

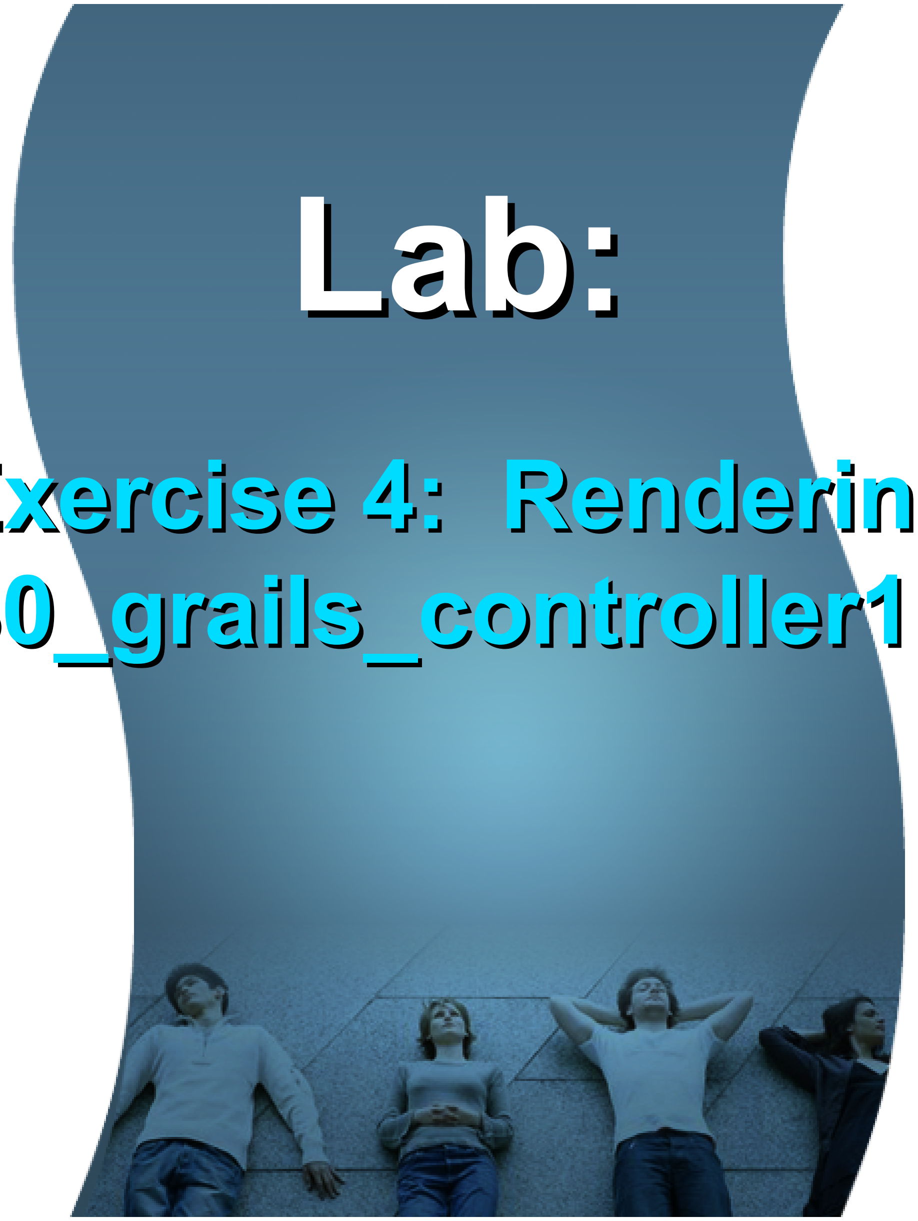
Rendering via “render” method (1)

- Sometimes it's easier (for example with Ajax applications) to render snippets of text or code to the response directly from the controller (instead of selecting a view)

```
// render text
render "Hello World!"

// render some text with encoding and content type
render(text: "<a><b>hello</b></a>", contentType: "text/xml",
                                    encoding: "UTF-8")
```

Rendering via “render” method (2)

```
// render a specific view
render(view: 'show')

// render some markup
render {
   for (b in books) {
      div(id: b.id, b.title)
   }
}

// render a template for each item in a collection
render(template: 'book_template', collection: Book.list())
```



Lab:

Exercise 4: Rendering
5630_grails_controller1.zip

Controller Interceptors

Controller (or Action) Interceptors

- Controller interceptors are used to intercept processing based on either request, session or application state
- There are currently two types of interceptors
  - before
  - after
- If your interceptor is likely to apply to more than one controller, you are almost certainly better off writing a Filter
  - Filters can be applied to multiple controllers or URIs without the need to change the logic of each controller

Before Interceptor Example #1

```
// This interceptor is executed before all actions
def beforeInterceptor = {
    println "Before calling action ${actionUri}"
}

// This interceptor is executed after all actions
def afterInterceptor = {
    println "After calling action ${actionUri}"
}
```

Before Interceptor Example #2

```
// The “beforeInterceptor” defines an interceptor that is used on all actions
// except the “login” action and it executes the “auth” method.
// (In this example, the “auth” method needs to be converted to closure
//  via method closure operator since value of “action” key has to be an object)
def beforeInterceptor = [action: this.&auth, except: 'login']

// defined with private scope, so it's not considered an action
private auth() {
    if (!session.user) {
        redirect(action: 'login')
        return false
    }
}

def login() {
    // display login page
}
```

After Interceptor Examples

```
// The “after” interceptor takes the resulting model as an argument
// and can hence manipulate the model or response.
def afterInterceptor = { model ->
    println "Tracing action ${actionUri}"
}

// An after interceptor may also modify the Spring MVC ModelAndView
// object prior to rendering
def afterInterceptor = { model, modelAndView ->
    println "Current view is ${modelAndView.viewName}"
    if (model.someVar) modelAndView.viewName =
                                        "/mycontroller/someotherview"
    println "View is now ${modelAndView.viewName}"
}
```

Interception Conditions

```
// Executes the interceptor except the specified action(s):
def beforeInterceptor = [action: this.&auth, except: ['login', 'register']]

// Executes the interceptor for only the specified action(s):
def beforeInterceptor = [action: this.&auth, only: ['secure']]
```

Lab:

Exercise 5: Controller Interceptors
5630_grails_controller1.zip

Redirecting

Redirecting

- Actions can be redirected using the *redirect* controller method:

```
class OverviewController {
    def login() {}

    def find() {
        // If a user has not logged in yet, redirect the user to login page
        if (!session.user)
            redirect(action: 'login')
            return
        }
        …
    }
}
```

More Redirecting examples (1)

```
// Parameters can optionally be passed
redirect(action: 'myaction', params: [myparam: "myvalue"])

// Pass request parameters
redirect(action: "next", params: params)

//  Include a fragment in the target URI: "/myapp/test/show#profile"
redirect(controller: "test", action: "show", fragment: "profile")
```

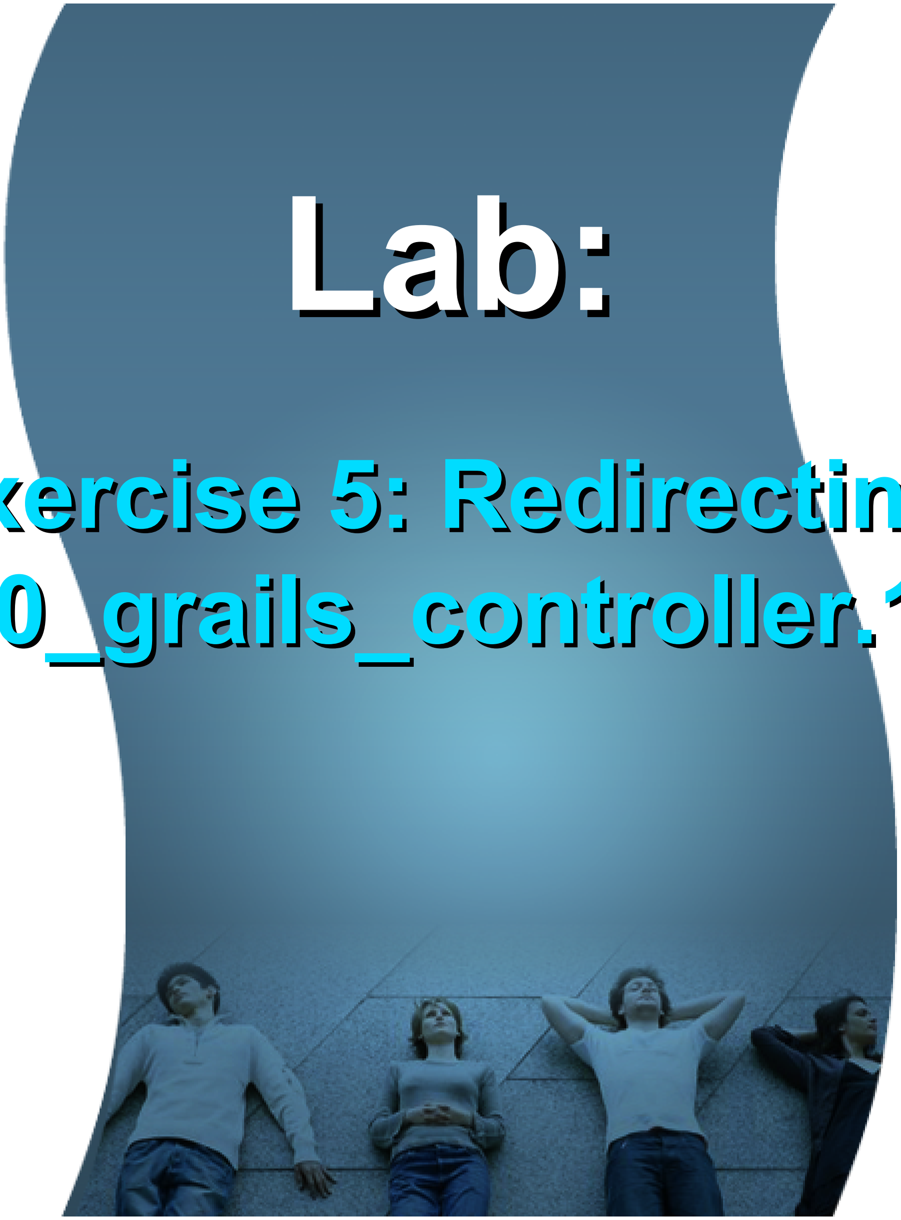
More Redirecting examples (2)

```
// Call the login action within the same class
redirect(action: 'login')

// Also redirects to the index action in the home controller
redirect(controller: 'home', action: 'index')

// Redirect to an explicit URI relative to the application context path
redirect(uri: "/login.html")

// Redirect to a full URL
redirect(url: "http://jpassion.com")
```

Redirecting & Double-submit problem

- Without redirecting, refreshing the “Create” page will cause the same request being sent again – this is “double-submit” problem

```
def save() {
    def teacherInstance = new Teacher(params)
    if (!teacherInstance.save(flush: true)) {
        render(view: "create", model: [teacherInstance: teacherInstance])
        return
    }

    flash.message = message(code: 'default.created.message',
       args: [message(code: 'teacher.label', default: 'Teacher'), teacherInstance.id])
    redirect(action: "show", id: teacherInstance.id)
}
```



Lab:

Exercise 5: Redirecting
5630_grails_controller.1zip

Data Binding (params)

What is Data Binding?

- Data binding is the act of "binding" incoming request parameters onto the properties of an object
- Data binding performs type conversion
  - Request parameters are typically delivered by a form submission and they are always strings while the properties of a Groovy or Java object may well not be
  - Grails perform type conversion during data binding
  - Type conversion errors could occur
- Grails uses Spring's underlying data binding capability to perform data binding

Binding Request Data to Model

```
// The data binding happens within the code of “new Book(params)”.
// By passing the params object to the domain class constructor, Grails
// automatically recognizes that you are trying to bind request
// parameters to Book object.
def save() {
    def book = new Book(params)
    book.save()
}

// Or you can use the properties property to perform data binding onto
// an existing instance
def save() {
    def book = Book.get(params.id)
    book.properties = params
    book.save()
}
```

Mapping Req. Params to Action Args(1)

- Controller action arguments are subject to request parameter data binding as well
- There are 2 categories of controller action arguments
  - Complex types
    - Treated as command objects
  - Basic object types
    - Supported types are the 8 primitives, their corresponding type wrappers and java.lang.String

Mapping Req. Params to Action Args(2)

- The default behavior is to map request parameters to action arguments by name:

```
class AccountingController {
   // accountNumber will be initialized with the value of params.accountNumber
   // accountType will be initialized with params.accountType
   def displayInvoice(String accountNumber, int accountType) {
       // …
   }
}
```

Type Conversion Errors

- Grails will retain type conversion errors inside the errors property of a Grails domain class

```
// Let's say we have Book domain class with URL type field
class Book {
    …
    URL publisherURL
}

// Given the following request coming in
/book/save?publisherURL=a-bad-url

def b = new Book(params)
if (b.hasErrors()) {
    println "The value ${b.errors.getFieldError('publisherURL').rejectedValue}" +
            " is not a valid URL!"
}
```

Lab:

Exercise 7: Data binding (params)
5630_grails_controller1.zip

XML and JSON Responses

Using “render” method to output XML

- The “*render*” method can be passed a block of code to do mark-up building in XML

```
def list() {
    def results = Book.list()

    render(contentType: "text/xml") {
        books {
            for (b in results) {
                book(title: b.title)
            }
        }
    }
}
```

- Generates

```
<books>
    <book title="The Stand" />
    <book title="The Shining" />
</books>
```

Using “render” method to output JSON

- The render method can be passed a block of code to do mark-up building in JSON

```
def list() {
    def results = Book.list()

    render(contentType: "text/json") {
        books = array {
            for (b in results) {
                book title: b.title
            }
        }
    }
}
```

- Generates

```
[
    {title:"The Stand"},
    {title:"The Shining"}
]
```

Automatic XML Marshalling

- Grails also supports automatic marshalling of domain classes to XML

```
render Book.list() as XML

<?xml version="1.0" encoding="ISO-8859-1"?>
<list>
  <book id="1">
    <author>Stephen King</author>
    <title>The Stand</title>
  </book>
  <book id="2">
    <author>Stephen King</author>
    <title>The Shining</title>
  </book>
</list>
```

Automatic JSON Marshalling

- Grails also supports automatic marshalling of domain classes to JSON

```
render Book.list() as JSON

[
    {"id":1,
     "class":"Book",
     "author":"Stephen King",
     "title":"The Stand"},
    {"id":2,
     "class":"Book",
     "author":"Stephen King",
     "releaseDate":new Date(1194127343161),
     "title":"The Shining"}
 ]
```

Lab:

Exercise 8: XML & JSON Responses
5630_grails_controller1.zip

Learn with Passion!
JPassion.com