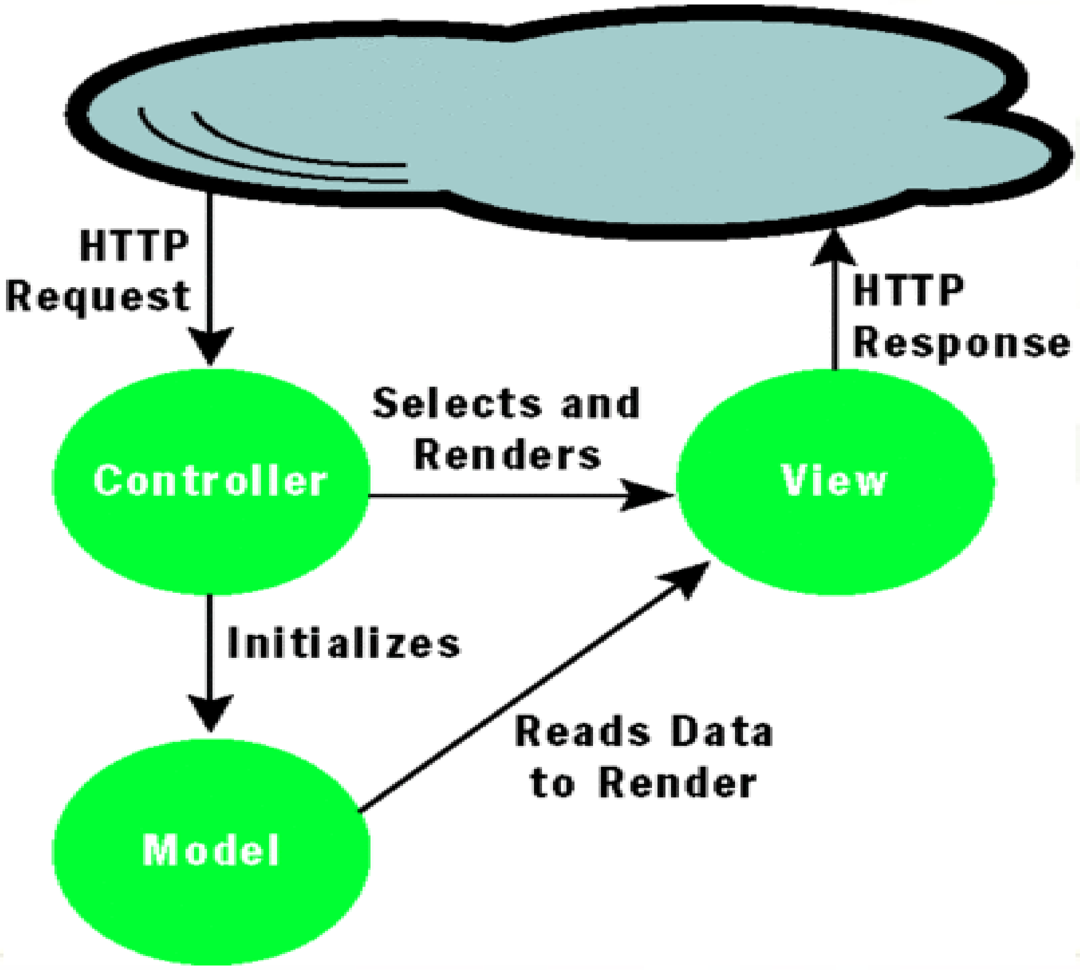
Grails Controllers

- Sachin Verma

Agenda

- ✓ What’s MVC?
- ✓ Controller - Definition and Usage
- ✓ Role played by controller in MVC
- ✓ Action
- ✓ Scaffolding - Static and Dynamic
- ✓ Rendering Views And Models - Basic
- ✓ Redirecting And Forwards
- ✓ Controller Scopes – Params ,session, flash scope
- ✓ Interceptor

MVC Architecture



Defining a Controller

- A class defined under grails-app/controllers directory
- Name ends with Controller as a convention; appended to end of controller name by default

```
grails create-controller sample
```

- Are request-scoped, i.e. a new instance is created for every request

Actions

- Actions are groovy closures, each of these actions maps to a URI
- def myFirstAction() {

render “Sample controller Accessed”

}

- Every controller has a number of actions; the above example maps to <....>/sample/myFirstAction

(SampleController is the name of the controller)

- Actions name starting with “get***” should be avoided.

Scaffolding

- Grails allows you to use dynamically generated code to get you started
- Set the property 'scaffold' of a controller to true to generate code on the fly

```
def scaffold = true
def scaffold = Author // Author is a domain class
```

- Views generated are HTML5 compliant since Grails 2.0.0

Scaffolding by types

- Dynamic Scaffolding

  def scaffold=Author

  It will provide all the artifacts only on run time i.e while application is up. One can't modify the views/gsps, so one has to use only default views.

- Static Scaffolding

  grails generate-all Author

  grails generate-all "*"   //For generating for all domain classes

  It will generate all the artifacts like AuthorController, show.gsp,edit.gsp.. etc

  Here one can modify views/gsps as the all things of code are made static.

Demo

Actions : Setting Default

There are 3 ways to define the default action of a controller

- Create an action named *index*

```
def index () {
        render “The action is right here – at index”
}
```

- Define a property named *defaultAction*

```
static defaultAction = “myFirstAction”
def myFirstAction() {
        render ”WooHoo!!”
        }
```

Restricting access to controller

- By default all controller actions are accessible using any HTTP request method (GET, PUT, POST, etc...).

  request.getMethod()

- To restrict this, define a map by the name of allowedMethods
- Every key of this map specifies the action name and the value of a key specifies the request method that the action should be allowed
- static allowedMethods = [action1: 'POST',

  action2: ['POST', 'GET', 'DELETE']]

Rending the view

- Based on convention, a gsp by the same name as the action is searched for, if no view is specified explicitly

  def myFirstAction = {}

- When trying to access this action, a view with the name of myFirstAction.gsp will be searched. If not found, it will give a 404 error

- To render a custom view, use the render method. render(view:"display")

Creating a model

- A key duty of the controller is gathering data that will be rendered in the view
- The collected data is put in a map called model
- class SampleController {

```
  def show = {
     render(view: 'show', model: [song: Song.get(1) ]
  }
}
```

Demo

Redirecting a request

- Passing control to another action within or outside the same controller
- Redirect method accepts a map as an argument

```
redirect(controller: 'sample', action: 'first')
```

- Arguments : action, controller, id, params, uri, url

Redirect and Forward

Forward:

- Is performed internally by the servlet
- Original URL stays intact

Redirect

Two step process - web application instructs the browser to fetch a second URL, which differs from the original

Slower than a forward, since it requires a

second browser request

Some Examples

```
forward action: "show", id: 4, params: [author: "Stephen King"]
forward (controller: "book", action: "show")


redirect(action:login)
redirect(controller:'home',action:'index')
redirect(uri:"/login.html")
redirect(url:"http://grails.org")
redirect(action:myaction,params:["myparam":"myvalue"])
```

Controller Scopes

- *servletContext* : allows us to share state across the entire web application.
- *session* : allows associating state with a given user.
- *request* : allows the storage of objects for the current request only.
- *params* - map of incoming request parameters
- *flash* : Objects placed in this scope are kept for the duration on the current request and the next request

Params and session

- "params" is a map consisting of incoming request parameters for the controller action

```
def paramsUsage = {
        render "I got the following name: " + params.name
 }
```

- "session" is a map that can be used to store data pertaining to a user; uses cookies to associate a particular state with a user.

```
def sessionsUsage = {
          render "I got the following name: " + session.name
      }
```

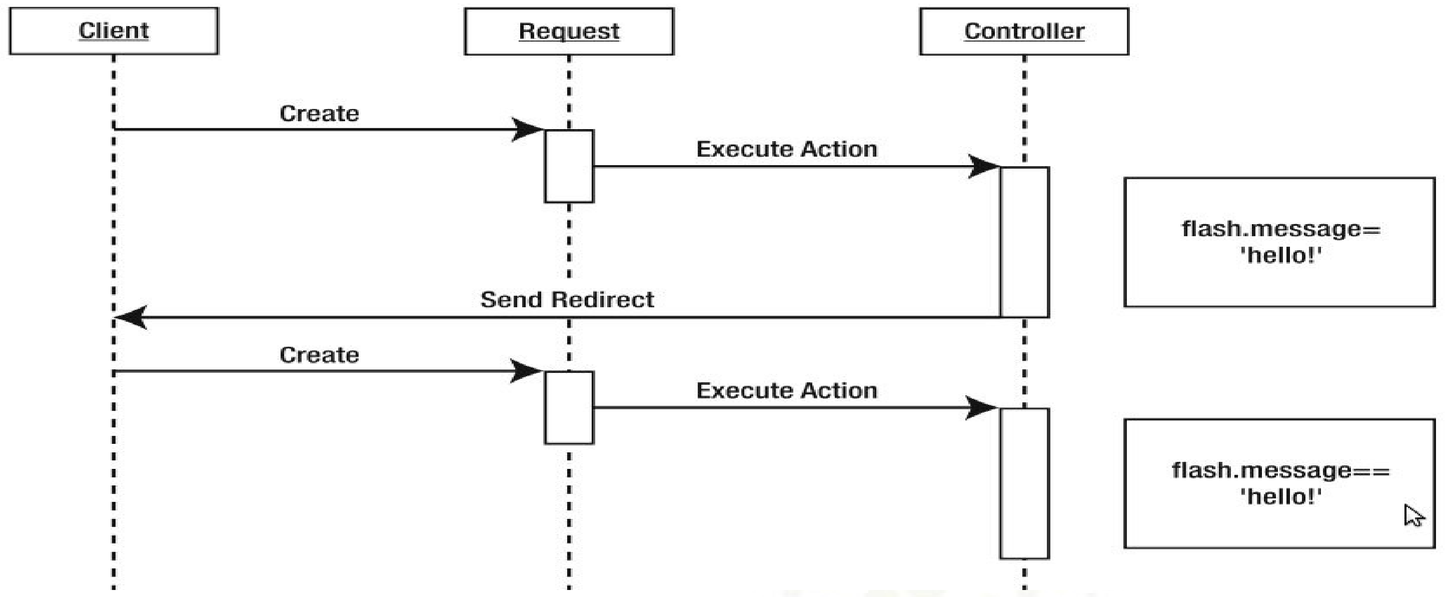
Request Attributes

The following properties are directly accessible in a controller

- actionName : Currently Executing Action
- actionUri : Relative URI of the action
- controllerName : Currently executing controller
- controllerUri : Relative URI of the action
- response : An HttpServletResponse object

Flash scope

- A special scope which is available for two requests
- Useful in the case of a redirect



Demo

Scoped Controllers

By default, a new controller instance is created for each request

You can change this behaviour by placing a controller in a particular scope.

```
static scope = "singleton"
```

- Prototype
- Session
- Singleton

You can define the default strategy under in Config.groovy

```
grails.controllers.defaultScope = "singleton"
```

Before Interceptor

- The 'before' interceptor intercepts processing before the action is executed.
- If it returns 'false' then the following action will not be executed.

```
def beforeInterceptor = {
    println "Tracing action ${actionUri}"
}
```

Before Interceptor

- An action can also be specified to be executed as a before interceptor

```
def beforeInterceptor = [action: this.&myAction]
def myAction() {  println “Hello World!!” }
```

- Also specify that before interceptor be not applied to certain action using the 'except' parameter

- Or specify interceptor to be executed for only one action using the 'only' parameter

After Interceptor

- “afterInterceptor” is executed after an action
- The after interceptor takes the resulting model as an argument and can hence perform post manipulation of the model or response.

```
def afterInterceptor = { model ->
                  println "Tracing action ${actionUri}"
                }
def afterInterceptor = { model, modelAndView ->
      println "Current view is ${modelAndView.viewName}"
      if(model.someVar) modelAndView.viewName = "/mycontroller/someotherview"
    println "View is now ${modelAndView.viewName}"
```

After Interceptor

- Similar to a before interceptor, an action can also be specified to be executed as an after interceptor
- def afterInterceptor = [action: this.&myAction]
- private myAction(model) { println “Ghost!!” }

- The parameters 'except' and 'only' can also be used in after interceptor

Demo

Data Binding: Implicit Constructor

```
Album album = new Album(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 :

/book/save?title=grails

- Then the title 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 :

```
   def 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'**.

Demo

References

http://samet.kilictas.com/what-is-mvc-architecture-model-view-controller/

http://grails.org/doc/latest/guide/theWebLayer.html#controllers

Questions ?

Thank You