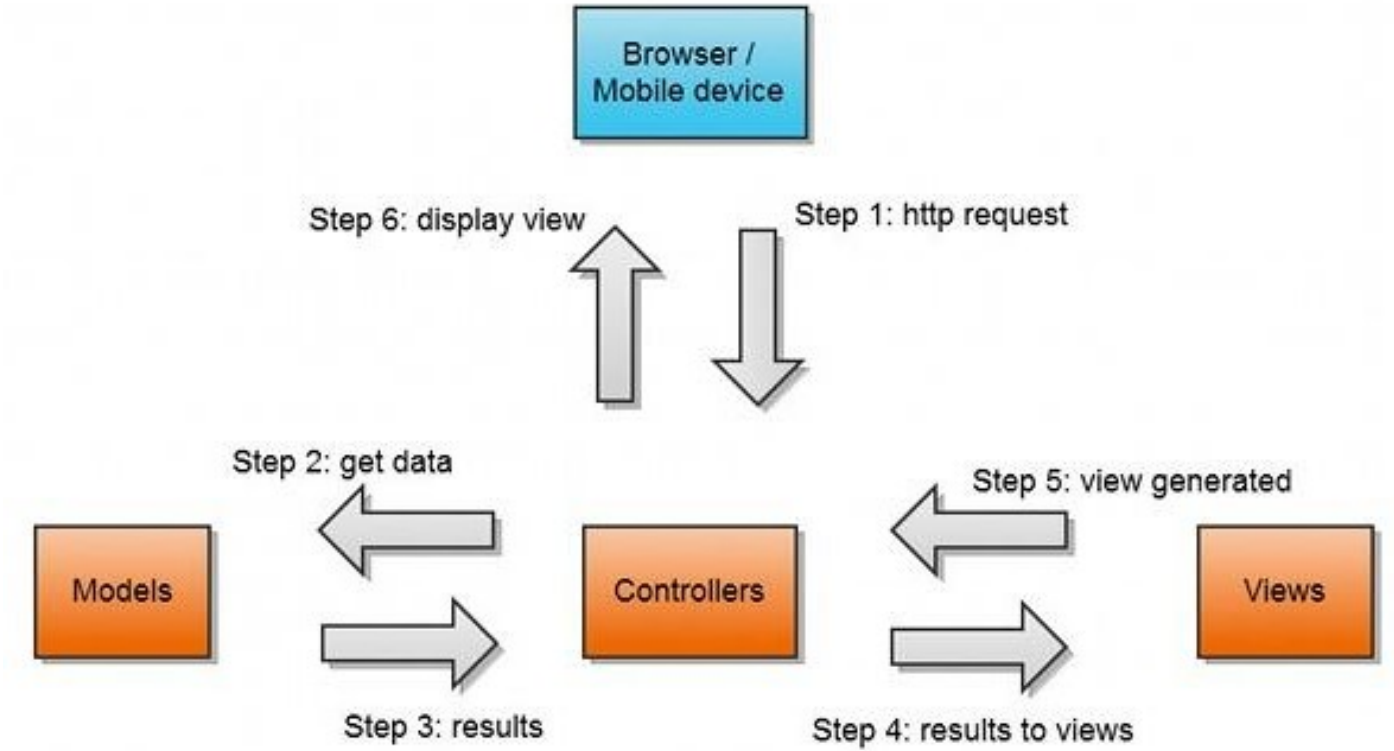
Grails View

Agenda

- Role of views in MVC
- How view is evaluated and rendered by action
- Page import and content type
- Groovy scriptlets ingsp
- Re-use of gsp code using templates
- What is layout, how to apply it on gsp?
- What is asset pipeline plugin and its tags
- How web-app folder is different from assets folder
- Built in grailstags

MVC Architecture



Rendering of Views

- Rendering a simple text:
```
render "Hello World!"
```

- Render a specific view:
```
render(view: 'show')
```

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

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

Page Import and Content Type:

The import directive lets you import classes into the page. However, it is rarely needed due to Groovy's default imports and GSP Tags:

```
<%@ page import="java.awt.*" %>
```

GSP also supports the contentType directive:

```
<%@ page contentType="application/json" %>
```

Groovy Scriptlets

GSP supports the usage of <% %> scriptlet blocks to embed Groovy code (again this is discouraged)

```
<html>
   <body>
     <% out << "Hello GSP!" %>
   </body>
</html>
```

You can also use the <%= %> syntax to output values:

```
<html>
   <body>
     <%="Hello GSP!" %>
   </body>
</html>
```

Groovy Scriptlets (contd….)

GSP also supports JSP-style server-side comments (which are not rendered in the HTML response) as the following example demonstrates:

```
<html>
   <body>
     <%-- This is my comment --%>
     <%="Hello GSP!" %>
   </body>
</html>
```

DEMO

Templates

- Template are the chunks of reusable gsp code.
- They make the views maintainable for developer
- Grails uses the convention of placing an underscore before the name of a view to identify it as a template

For example (a template for showing details of books can be):

```
// grails-app/views/book/_bookTemplate.gsp

<div class="book" id="${book?.id}">
   <div>Title: ${book?.title}</div>
   <div>Author: ${book?.author?.name}</div>
</div>
```

Rendering a template

- Use the render tag to render this template from one of the views in grails-app/views/book:

```
<g:render template="bookTemplate" model="[book: myBook]" />
```

- If you have multiple Book instances you can also render the template for each Book using the render tag with a collection attribute:

```
<g:render template="bookTemplate" var="book" collection="${bookList}" />
```

- For rendering shared template you have to use relative path to the template from inside the views folder: eg. for rendering grails-app/views/shared/_mySharedTemplate.gsp

```
<g:render template="/shared/mySharedTemplate" />
```

tmpl namespace

Since templates are used so frequently there is template namespace, called tmpl, available that makes using templates easier. So,

```
<g:render template="bookTemplate" model="[book:myBook]" />
```

is equivalent to:

```
<tmpl:bookTemplate book="${myBook}" />
```

DEMO

Layouts

Grails leverages Sitemesh, a decorator engine, to support view layouts
Layouts are located in the grails-app/views/layouts directory

```
<html>
    <head>
        <title><g:layoutTitle default="An example decorator" /></title>
        <g:layoutHead />
    </head>
    <body>
        <div class="menu"><!--my common menu goes here--></menu>
            <div class="body">
                <g:layoutBody />
            </div>
        </div>
    </body>
</html>
```

Layout Elements

The key elements are the layoutHead, layoutTitle and layoutBody tag invocations:

- layoutTitle - outputs the target page's title
- layoutHead - outputs the target page's head tag contents
- layoutBody - outputs the target page's body tag contents

To apply layout we do following:

```
<html>
    <head>
        <title>An Example Page</title>
        <meta name="layout" content="main" />
    </head>
    <body>This is my content!</body>
</html>
```

DEMO

Asset Pipeline Plugin

- The Asset-Pipeline is a plugin used for managing and processing static assets in Grails applications.
- Asset-Pipeline functions include processing and minification of both CSS and JavaScript files.
- Asset-Pipeline automatically creates a series of folders within your grails-app directory for maintaining your static assets:
  - grails-app/assets/javascript
  - grails-app/assets/images
  - grails-app/assets/stylesheets
- Its features includes:
  - On the fly processing - No more waiting for your assets to reload after making a change
  - Compiled assets on war create - No more hanging up application boot times while processing files. `grails war`
  - Reduced Dependence - The plugin has compression, minification built in.
  - Easy Debugging - Makes for easy debugging by keeping files separate in development mode.
  - Simpler manifests and taglibs - Read on for more information.

Asset Pipeline continued

- To include these assets in your gsp page:

```
<head>
      <asset:javascript src="application.js"/>
      <asset:stylesheet href="application.css"/>
</head>
```

- To exclude a file from processing you include following configuration:

```
grails.assets.excludes = ["tiny_mce/src/*.js"]
```

- All the assets are automatically minified when grails war is generated. To stop this behaviour we can add following command in Config.groovy

```
grails.assets.minifyJs = false
```

Behaviour of asset pipeline in localhost

```
<!doctype html>
<html><head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=
        <!-- Apple devices fullscreen -->
        <meta name="apple-mobile-web-app-capable" content="yes"/>
        <!-- Apple devices fullscreen -->
        <meta names="apple-mobile-web-app-status-bar-style" content="black-translucent"/>

        <title>     | Admin</title><!-- Login -->


        <link rel="stylesheet" href="/assets/admin/bootstrap.min.css?compile=false"  /><link
compile=false"  /><link rel="stylesheet" href="/assets/admin/plugins/icheck/all.css?compile=
compile=false"  /><link rel="stylesheet" href="/assets/admin/custom-style.css?compile=false"
compile=false"  /><link rel="stylesheet" href="/assets/admin/admin-login.css?compile=false"

        <script src="/assets/admin/jquery.min.js?compile=false" type="text/javascript" ></sc
src="/assets/admin/plugins/nicescroll/jquery.nicescroll.min.js?compile=false" type="text/jav
src="/assets/admin/plugins/validation/jquery.validate.min.js?compile=false" type="text/javas
src="/assets/admin/plugins/icheck/jquery.icheck.min.js?compile=false" type="text/javascript"
type="text/javascript" ></script><script src="/assets/admin/eakroko.js?compile=false" type="
compile=false" type="text/javascript" ></script>
```

Behaviour of assets on production server

```
pe html>
ıead>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalab
 <!-- Apple devices fullscreen -->
 <meta name="apple-mobile-web-app-capable" content="yes"/>
 <!-- Apple devices fullscreen -->
 <meta names="apple-mobile-web-app-status-bar-style" content="black-translucent"/>

 <title>      | Admin</title><!-- Login -->


 <link rel="stylesheet" href="/assets/admin/admin-login-90aee34dba6827e5cbf20bc2f2862c91.css"/>

 <script src="/assets/admin/admin-login-ec3b073fb90a6cfc38bbf2d3e7063202.js" type="text/javascript" >
```

Creating a manifest file

```
/*
 * This is a manifest file that will be compile into application.js,
 * which will include all the files from mockup and its sub-directories related to
 * all front-end.
 *
 * Any js file within this directory can be referenced here using a relative path.
 *
 * We're free to add js to this file and they'll appear at the top
 * of the compile file, but it's generally better to create a new file per js
   scope.
 *
 * */

//=require less-1.7.3.min
//=require spin.min
//=require config
//=require custom
//=require common
//=require_self
```

DEMO

Built-in GSP Tags

- There are many predefined GSP tags available to use in Grails.
- All of them use namespace ‘g’. We will see this..
- They are categorized into following categories:
  - Logic Tags
  - Iteration Tags
  - Link Tags
  - Forms and Fields Tags

Logic Tags

```
//g:if , g:elseif, g:else
<g:if test="${session.role == 'superadmin'}">
   <%-- show super administrative functions --%>
</g:if>
<g:elseif test="${session.role == 'admin'}">
   <%-- show administrative functions --%>
</g:elseif>
<g:else>
   <%-- show basic functions --%>
</g:else>
```

Iteration Tags

```
// g:each
<g:each in="${[1,2,3]}" var="num">
   <p>Number ${num}</p>
</g:each>
```

```
// g:while

<g:set var="num" value="${1}" />
<g:while test="${num < 5 }">
    <p>Number ${num++}</p>
</g:while>
```

Link Tags

GSP also features tags to help you manage linking to controllers and actions

```
<g:link action="show" id="1">Book 1</g:link>

<g:link action="show" id="${currentBook.id}">${currentBook.name}</g:link>

<g:link controller="book">Book Home</g:link>

<g:link controller="book" action="list">Book List</g:link>

<g:link url="[action: 'list', controller: 'book']">Book List</g:link>

<g:link params="[sort: 'title', order: 'asc', author: currentBook.author]"
        action="list">Book List</g:link>
```

Form Tag

- This is a controller/action aware version of the regular HTML form tag.
- The url attribute lets you specify which controller and action to map to.

```
<g:form name="myForm" url="[controller:'book',action:'list']">...</g:form>
```

Form Fields

Following form field tags are allowed :

- textField - For input fields of type 'text'
- field - number, password, uri
- passwordField - For input fields of type 'password'
- checkBox - For input fields of type 'checkbox'
- radio - For input fields of type 'radio'
- hiddenField - For input fields of type 'hidden'
- select - For dealing with HTML select boxes
- submitButton
- actionSubmit

Error & Message Tags

- hasError
- eachError
- message

Resource Tags

```
// generates "/shop/css/main.css"
<g:resource dir="css" file="main.css" />

// generates "http://portal.mygreatsite.com/css/main.css"
<g:resource dir="css" file="main.css" absolute="true" />

// generates "http://admin.mygreatsite.com/css/main.css"
<g:resource dir="css" file="main.css" base="http://admin.mygreatsite.com"/>
```

as method call:

```
<link type="text/css" href="${resource(dir: 'css', file: 'main.css')}" />
```

Image Tag

```
<g:img dir="images" file="logo.png" width="40" height="40"/>
Output: <img src="/shop/images/logo.png" width="40" height="40"/>

<g:img uri="/images/icons/add.png"/>
Output: <img src="/shop/images/logo.png"/>
```

Pagination Tag

TO paginate the large data into multiple steps:

```
<g:paginate next="Forward" prev="Back"
            maxsteps="0" controller="book"
            action="list" total="${bookCount}" />
```

File Upload

Identical to the standard form tag except that it sets the enctype attribute to "multipart/form-data" automatically.

```
<g:uploadForm name="myUpload" action=”actionName” controller=”controllerName”>
    <input type="file" name="myFile" />
</g:uploadForm>
```

DEMO

Coding Conventions

GSP Clean Code

Name should be in camel case.

Bad:

```
Generalinfo.gsp
```

Good:

generalInformation.gsp

generalInfo.gsp

GSP Clean Code

Make extensive use of taglib rather than doing calculations in gsp.

Bad:

```
<g:if test="${endDate < startDate}">
     0
</g:if>
<g:else>
     ${endDate - startDate} days left
</g:else>
```

Good:

```
<taglib:daysDiff startDate="${startDate}"
endDate="${endDate}">
```

GSP Clean Code

DO NOT write any GORM query in GSP

Bad:

```
<g:each in="${Person.list()}">
     //SOMETHING
</g:each>
```

Good:

All the data is sent from the action and gsp is used just for showing it.

```
//in controller
render view:'list',model:[persons:Person.list()]
//in gsp
<g:each in="${persons}">
     //SOMETHING
</g:each>
```

GSP Clean Code

- DO NOT make database calls in layouts (Try to put such data which is used in layout either in session or in application context)
- Typecast VO in gsp, so that you have all properties readily available for

```
<g:set var="vo" value="${vo as AccountVO}"/>
```

- Make templates rather than having single big gsp file.
- Write all the javascript in js files. It helps in caching, code is reusable, can be maintained easily.
- Application level js file should be named as all.js / common.js / application.js
- Avoid in-line styling

GSP Clean Code

All the css statements should be clubbed into one block similarly all JS statement:

Bad:

```
<link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css', file: 'all.
css')}"/>
<script type="text/javascript" src="${resource(dir: 'js/lempa', file:
'all.js')}"></script>
<link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css', file: 'tip.
css')}"/>
<script type="text/javascript" src="${resource(dir: 'js/jquery', file: 'tip-1.3.1.js')}"
></script>
```

Good:

```
<link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css', file: 'all.
css')}"/>
<link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css', file: 'tip.
css')}"/>
<script type="text/javascript" src="${resource(dir: 'js/lempa', file:
'all.js')}"></script>
<script type="text/javascript" src="${resource(dir: 'js/jquery', file: 'tip-1.3.1.js')}"
></script>
```

Resources

- http://grails.github.io/grails-doc/2.5.3/guide/theWebLayer.html
- http://www.oodlestechnologies.com/blogs/Importance-of-Grails-asset-pipeline-plugin
- https://grails.org/plugin/asset-pipeline

QUESTIONS?

Thank You!!!