FREE eBook

LEARNING grails

Free unaffiliated eBook created from
Stack Overflow contributors.

#grails

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: grails

It is an unofficial and free grails ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official grails.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with grails

Remarks

Grails is a very powerful rapid application development (RAD) framework, for the **Java platform** aimed at multiplying **developers' productivity** thanks to a Convention-over-Configuration, sensible defaults and opinionated APIs. It integrates smoothly with the JVM and the **Groovy language** allowing you to be **immediately productive** whilst providing powerful features, including integrated object-relational-mapping (ORM), Domain-Specific Languages (DSL), runtime and compile-time meta-programming and asynchronous programming.

The Grails homepage is found here

Versions

Version	Remarks	Release Date
2.5.5	latest version in 2.x legacy line	2016-10-27
3.2.2	latest as of 30-Oct-2016	2016-06-24
3.2.3	latest version in 3.x	2016-11-10

Examples

Grails Installation

Note: GRAILS requires a Java JDK installed (a runtime environment JRE is not sufficient) on your system, before setting up Grails. Please refer to, how to install JDK. As of this writing, it is recommended to install the latest JDK.

For Mac OSX, Linux, Cygwin, Solaris and FreeBSD:

The simplest way to manage Grails versions is using sdkman. If `sdkman` is installed, you can then install any version of Grails using

```
sdk install grails [version]
```

This will take care of all steps to get this right. If you skip the `version`, the latest version of grails will be installed. For more about using `sdkman`, refer to sdkman usage page.

For Linux:

```
GRAILS_HOME=$HOME/bin/grails/current
# abbreviating it using "..." for brevity
PATH=$GRAILS_HOME/bin:$JAVA_HOME/bin: ... :$PATH
```

For Windows:

1. Download a Java JDK from Oracle and install on your Windows machine. Take note of the installation folder.
2. Download a version of Grails manually from the Downloads page.
3. Extract the Grails file, anywhere you like.
4. **Important:** You must set up 2 new environment variables `JAVA_HOME` and `GRAILS_HOME` (for Windows 10 found under *Control Panel \ System and Security \ System \ Advanced System settings \ Advanced tab \ Environment Variables) *, pointing to the extracted directories, e.g.

   Name: JAVA_HOME

   Value: C:\Programs\Java\jdk1.8.0_31

   Name: GRAILS_HOME

   Value: c:\grails\grails-3.2.4

5. **Important:** You must extend the Windows `PATH` variable to include both JAVA_HOME and GRAILS_HOME. The path variable is also found in then control panel (see 4), e.g. add the following at the end:

   ;C:\Programs\Java\jdk1.8.0_31\bin;c:\grails\grails-3.2.4;c:\grails\grails-3.2.4\bin

5. To verify your installation is correct, open a Command Prompt and type `GRAILS -VERSION`. You should get something like:

```
| Grails Version: 3.2.4
| Groovy Version: 2.4.6
| JVM Version: 1.8.0_65
```

Creating an application

To create a Grails application, use the `grails create-app` command. The following command creates a Grails application, named `myapp` in the current directory:

```
grails create-app fancy-app
```

Running it, is as simple as visiting the, newly created, application directory:

```
cd fancy-app
```

and then

```
grails run-app
// in order to run the app on a different port, e.g. 8888, use this instead
grails run-app -port 8888
// in order to run the app with a secure communication
grails run-app -https
```

Testing an Application

The create-* commands in Grails automatically create unit or integration tests for you within the src/test/groovy directory. It is of course up to you to populate these tests with valid test logic, information on which can be found in the section on Unit and integration tests.

To execute tests you run the test-app command as follows:

```
grails test-app
```

Creating a Model

A model (see: Model-View-Controller pattern) in Grails is represented by a so-called **Domain Class**. Domain classes can define both the persistence and presentation of information in grails. Domain classes can also contain validations.

To manage a fleet of cars in your Grails application you could define a domain class to describe, store and represent various cars in your fleet.

To create a stub for a domain class execute the following command inside your application folder:

```
grails create-domain-class org.fleetmanager.Car
```

Next, open the generated car.groovy file and edit your domain class as follows:

```
package org.fleetmanager

class Car {
   String      manufacturer
   String      model
   String      color
   Integer     year
   Date        acquisitionDate
   Boolean     isElectric
}
```

Finally, generate a controller for your car domain and a view using the following Grails command:

```
grails generate-all org.fleetmanager.Car
```

Now, you can run your applications, select the car controller and manage your fleet.

Read Getting started with grails online: https://riptutorial.com/grails/topic/1435/getting-started-with-grails

Chapter 2: Deployment

Examples

Executable Jar

One of the easiest ways to deploy Grails 3.x is to build an executable jar file that embeds a servlet container (Tomcat, Undertow, etc) with the application.

Modify `build.gradle`:

```
// Remove or comment out the war plugin:
// apply plugin:"war"

// Enable the executable jar:
springBoot {
    executable = true
}

// Optional: Customize the jar properties:
//  https://docs.gradle.org/current/dsl/org.gradle.api.tasks.bundling.Jar.html
jar {
    archiveName('myapp.jar')
}
```

Build using `./gradlew assemble`

Resulting jar is now a fully executable app you can startup:

```
$ head build/libs/myapp.jar
#!/bin/bash
#
#    .   ____          _            __ _ _
#   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
#  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
#   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
#    '  |____| .__|_| |_|_| |_\__, | / / / /
#   =========|_|==============|___/=/_/_/_/
#   :: Spring Boot Startup Script ::
#
```

You can start it as you would normally for any command line app:

```
$ ./build/libs/myapp.jar
Grails application running at http://localhost:8080 in environment: production
```

It also behaves like an init service:

```
$ ln -s /opt/myapp/myapp.jar /etc/init.d/myapp
$ service myapp [start|stop|status|restart]
```

Detailed documentation is under the spring-boot docs: http://docs.spring.io/spring-boot/docs/current/reference/html/deployment-install.html

War File Creation

When we write an Web application in Grails, to deploy the application we need a "war" file that need's to be put in the servlet container (Tomcat etc).

First goto the project directory :

```
cd to_project_directory
```

1. War file creation from command prompt :

```
grails war
```

2.Its always recommendable that you clean out your application before war creation

Cleaning application from command prompt :

```
grails clean
```

Combining the above two steps in one will result in

```
grails clean && grails war
```

Also you can specify the environment in which you want to create the war file.

```
grails [environment] war
```

Where `[environment]` can take the following values: `dev`, `prod` or `test` for example.

Unlike other commands, the war command runs in the production environment by default instead of development.

Read Deployment online: https://riptutorial.com/grails/topic/3701/deployment

Chapter 3: Domain classes as REST resources

Introduction

The easiest way to create a RESTful API in Grails is to expose a domain class as a REST resource. This can be done by adding the grails.rest.Resource transformation to any domain class.

Examples

Simple REST API with grails

```
import grails.rest.*

@Resource(uri='/books')
class Book {

    String title

    static constraints = {
        title blank:false
    }
}
```

Simply by adding the Resource transformation and specifying a URI, your domain class will automatically be available as a REST resource in either XML or JSON formats. The transformation will automatically register the necessary RESTful URL mapping and create a controller called BookController.

You can try it out by adding some test data to BootStrap.groovy:

```
def init = { servletContext ->
    new Book(title:"The Stand").save()
    new Book(title:"The Shining").save()
}
```

And then hitting the URL `http://localhost:8080/books/1`, which will render the response like:

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1">
    <title>The Stand</title>
</book>
```

If you change the URL to `http://localhost:8080/books/1.json` you will get a JSON response such as:

```
{"id":1,"title":"The Stand"}
```

If you wish to change the default to return JSON instead of XML, you can do this by setting the formats attribute of the Resource transformation:

```
import grails.rest.*

@Resource(uri='/books', formats=['json', 'xml'])
class Book {
    ...
}
```

Mapping to REST resources

If you prefer to keep the declaration of the URL mapping in your UrlMappings.groovy file then simply removing the uri attribute of the Resource transformation and adding the following line to `UrlMappings.groovy` will suffice:

```
"/books"(resources:"book")
```

Extending your API to include more end points then becomes trivial:

```
"/books"(resources:"book") {
    "/publisher"(controller:"publisher", method:"GET")
}
```

The above example will expose the URI `/books/1/publisher`.

Add HTTPS to Grails Server

SSL Certificates use something called public key cryptography.We need to use Https instead of Http because of keeping data secure between servers and improving customer trust. To enable this option in grails, we have to run our app differently. The command below:

```
grails run-app -https
```

Read Domain classes as REST resources online: https://riptutorial.com/grails/topic/8944/domain-classes-as-rest-resources

Chapter 4: GSP

Parameters

Variables and scopes	Details
application	ServletContext instance
applicationContext	Spring ApplicationContext instance
flash	The flash object
grailsApplication	GrailsApplication instance
out	response writer for writing to the output stream
params	params object for retrieving request parameters
request	HttpServletRequest instance
response	HttpServletResponse instance
session	HttpSession instance
webRequest	GrailsWebRequest instance

Examples

Basics

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

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

You can also use the `<%= %>` syntax to output values, like in JSP:

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

GSP also supports JSP-style server-side comments too:

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

Expressions

In GSP the `<%= %>` syntax is rarely used due to the support for **GSP expressions**.

A GSP expression is similar to a **JSP EL** expression or a **Groovy GString** and takes the form `${expr}`:

```
<html>
  <body>
    Hello ${params.name}
  </body>
</html>
```

However, unlike JSP EL you can have any Groovy expression within the `${..}` block.

Any Groovy expression can be interpolated in all string literals, apart from single and triple single quoted strings. Interpolation is the act of replacing a placeholder in the string with its value upon evaluation of the string. The placeholder expressions are surrounded by ${} or prefixed with $ for dotted expressions. The expression value inside the placeholder is evaluated to its string representation when the GString is passed to a method taking a String as argument by calling toString() on that expression.

GSP Tags

There are variety of gsp tags available which can be used to create forms, textfield, radio buttons, check boxes, if-else, for each etc.

<g:if>

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

<g:each>

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

form

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

textField

```
<g:textField name="myField" value="${myValue}" />
```

radio

```
<g:radio name="myGroup" value="1"/>
```

Follow this link for more info - http://docs.grails.org/latest/guide/theWebLayer.html#tags

Read GSP online: https://riptutorial.com/grails/topic/4531/gsp

Credits

S. No	Chapters	Contributors
1	Getting started with grails	Abdelsalam A. Shahlol, Adeel Ansari, Andrea Zago, bronoman, Burt Beckwith, Community, Farshid Zaker, Graeme Rocher, Jason Bourne, Jesús Iglesias, rahul, saw303
2	Deployment	erichelgeson, NachoB, Prakash Thete, saw303
3	Domain classes as REST resources	Jason Bourne
4	GSP	Adeel Ansari, Anshul, audittxl