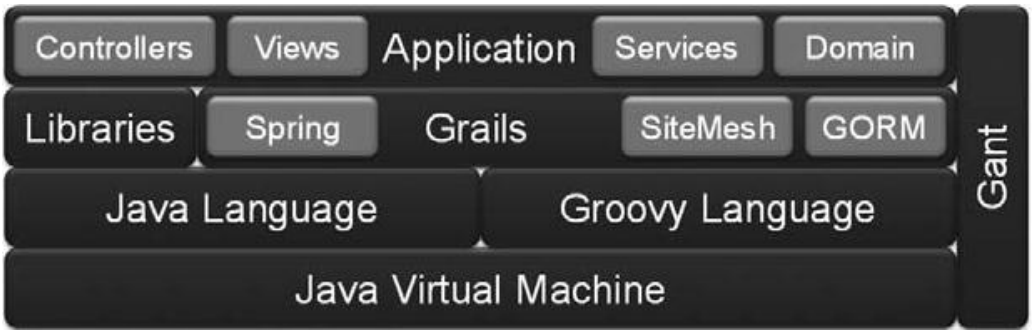
Web Development with Groovy on Grails
IS2150

Assoc Prof Kaushik Dutta
Department of Information Systems
School Computing
National University of Singapore

1. Grails

Grails aims to bring the 'coding by convention' paradigm to Groovy. It's an open-source web application framework that leverages the Groovy language and complements Java Web development. You can use Grails as a standalone development environment that hides all configuration details or integrate your Java business logic. Grails aims to make development as simple as possible and hence should appeal to a wide range of developers not just those from the Java community.

2. Grails Architecture



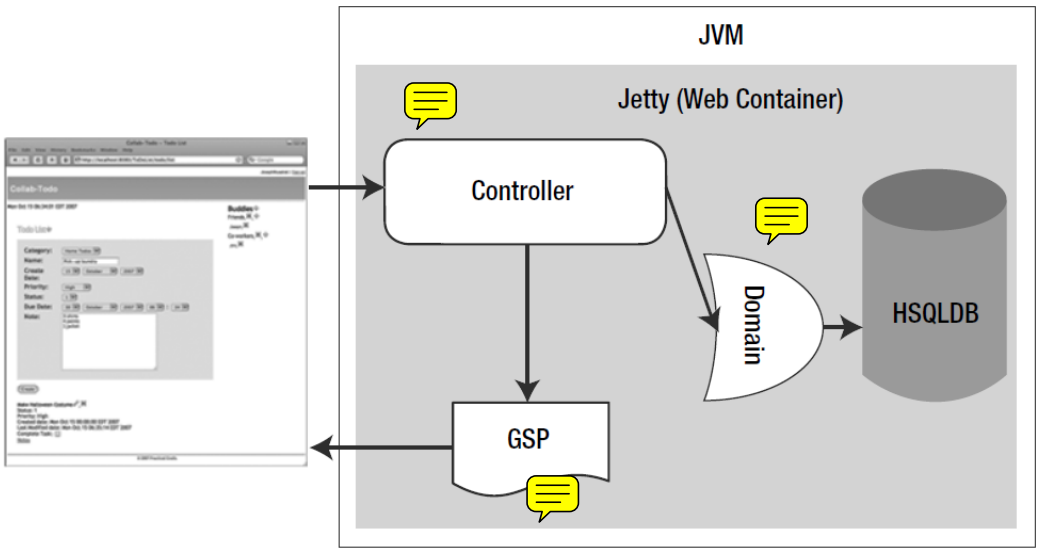
In the above figure, notice that the foundation of Grails is the Java Virtual Machine (JVM). Also, notice the separation in the architecture from the Java language and the JVM. In the past couple of years, the Java community has seen a rash of new and ported languages being run on the JVM. This is particularly important in Grails, because in the next level up from the JVM, you see that both the Java and Groovy languages are being used.

Above the languages you find the Grails framework itself, which is made up of several industry-standard open source projects such as Spring, SiteMesh, and GORM/Hibernate, to name just a few. However, as an application developer, you're not limited to the libraries and frameworks Grails has to offer. Your application can use just about any Java library, whether open source or proprietary. The final layer of the architecture is the applications you will build with Grails. Typically, this layer follows the MVC pattern. Grails also makes it easy to organize your application to make coarse-grained services.

To simplify development, Grails includes a command-line tool for creating many Grails artifacts and managing Grails projects. The Grails command line is built on top

of Gant, a build system that uses the Groovy language to script Apache Ant tasks rather than Ant's XML format.

From a runtime perspective, you can think of Grails out of the box as looking like in the following figure:



The figure shows the Grails default runtime. You see a web browser making a request to a Jetty web container. The container forwards the request on to a controller in a similar fashion to the standard MVC model. The controller may set or use data from a domain class (model). All Grails domain classes are persistable through the Grails Object Relational Mapping (GORM) framework.

You don't need to use a Data Access Object (DAO) pattern or write SQL to persist objects. Out-of-the-box Grails uses an embedded Hibernate SQL Database (HSQLDB) database, which means the database runs in the same JVM as your application and the Jetty web container. When the controller is done, it forwards the request to a GSP, which is the view technology to render the HTML that is returned to the requesting browser.

3. Domains, Controllers and Views

1. Domain Class

A domain class fulfills the M in the Model View Controller (MVC) pattern and represents a persistent entity that is mapped onto an underlying database table. In Grails a domain is a class that lives in the grails-app/domain directory. A domain class can be created with the "create-domain-class" command.

2. Controller Class

Controllers are central to generating your user interface or providing a programmatic REST interface. They typically handle the web requests that come

from a browser or some other client and you'll find that each URL of your application is usually handled by one controller.

Grails provides a feature called scaffolding that automatically creates a user interface for a domain class that allows you create new instances, modify them, and delete them. Enabling scaffolding is very straightforward.

A controller handles requests and creates or prepares the response and is request-scoped. In other words a new instance is created for each request. A controller can generate the response or delegate to a view. To create a controller simply creates a class whose name ends with Controller and places it within the grails-app/controllers directory.

The default URL Mapping setup ensures that the first part of your controller name is mapped to a URI and each action defined within your controller maps to URI within the controller name URI.

3. View Class

Groovy Servers Pages (or GSP for short) is Grails' view technology. It is designed to be familiar for users of technologies such as ASP and JSP, but to be far more flexible and intuitive.

In Grails GSPs live in the grails-app/views directory and are typically rendered automatically (by convention) or via the render method such as:

```
render(view:"index")
```

A GSP is typically a mix of mark-up and GSP tags which aid in view rendering. A GSP typically has a "model" which is a set of variables that are used for view rendering. The model is passed to the GSP view from a controller.

4. Grails Installation

Prerequisites

Before you start using Grails you will need to install a Java SDK (not just a JRE) and set the JAVA_HOME environment variable to the location of that SDK. The minimum required version of the SDK depends on which version of Grails you are using:

- Java SDK 1.4+ for Grails 1.0.x and 1.1.x
- Java SDK 1.5+ for Grails 1.2 or greater

Steps

- Download the latest Grails from http://grails.org/Download
- Extract the archive into an appropriate location; typically C:\grails on Windows or ~/grails on Unix
- Create a GRAILS_HOME environment variable that points to the path where you extracted the archive (eg C:\grails on Windows or ~/grails on Unix)

- If you have not set the JAVA_HOME environment variable yet, create JAVA_HOME environment variable that points to the path where you have installed Java. To see how to set JAVA_HOME variable in windows refer the following link http://confluence.atlassian.com/display/DOC/Setting+the+JAVA_HOME+Variable+in+Windows

- Append a reference to the "bin" directory within the Grails directory to your PATH variable (eg%GRAILS_HOME%\bin on Windows or $GRAILS_HOME/bin on Unix). Note that, for Windows, both PATH and GRAILS_HOME must be defined at the same environment variable level (eg. 'System variables') rather than across environment variable levels (eg. PATH under 'System variables' and GRAILS_HOME under 'User variables')

- Type "grails" at the command line, if a help message is displayed you are ready to start using Grails! If you don't get this verify again your environment variable setting.

5. IDE Integration

Grails development needs nothing more than a text editor, but of course there are many sophisticated tools available to improve productivity, including syntax highlighting, refactoring and grails command execution.

- STS Integration - Using Grails projects in the SpringSource Tool Suite, an Eclipse-based IDE.
- NetBeans IDE Integration - Using NetBeans IDE with Grails projects
- IDEA Integration - Using IntelliJ IDEA IDE with Grails projects
- TextMate Bundle - TextMate Grails bundle
- Gedit - a TextMate like alternative for Linux
- jEdit - A cross platform programmer's text editor written in Java, has excellent Groovy plugin. jEdit is open source software.
- Emacs- GSP support for Emacs. Groovy-Mode, Grails-Minor-Mode

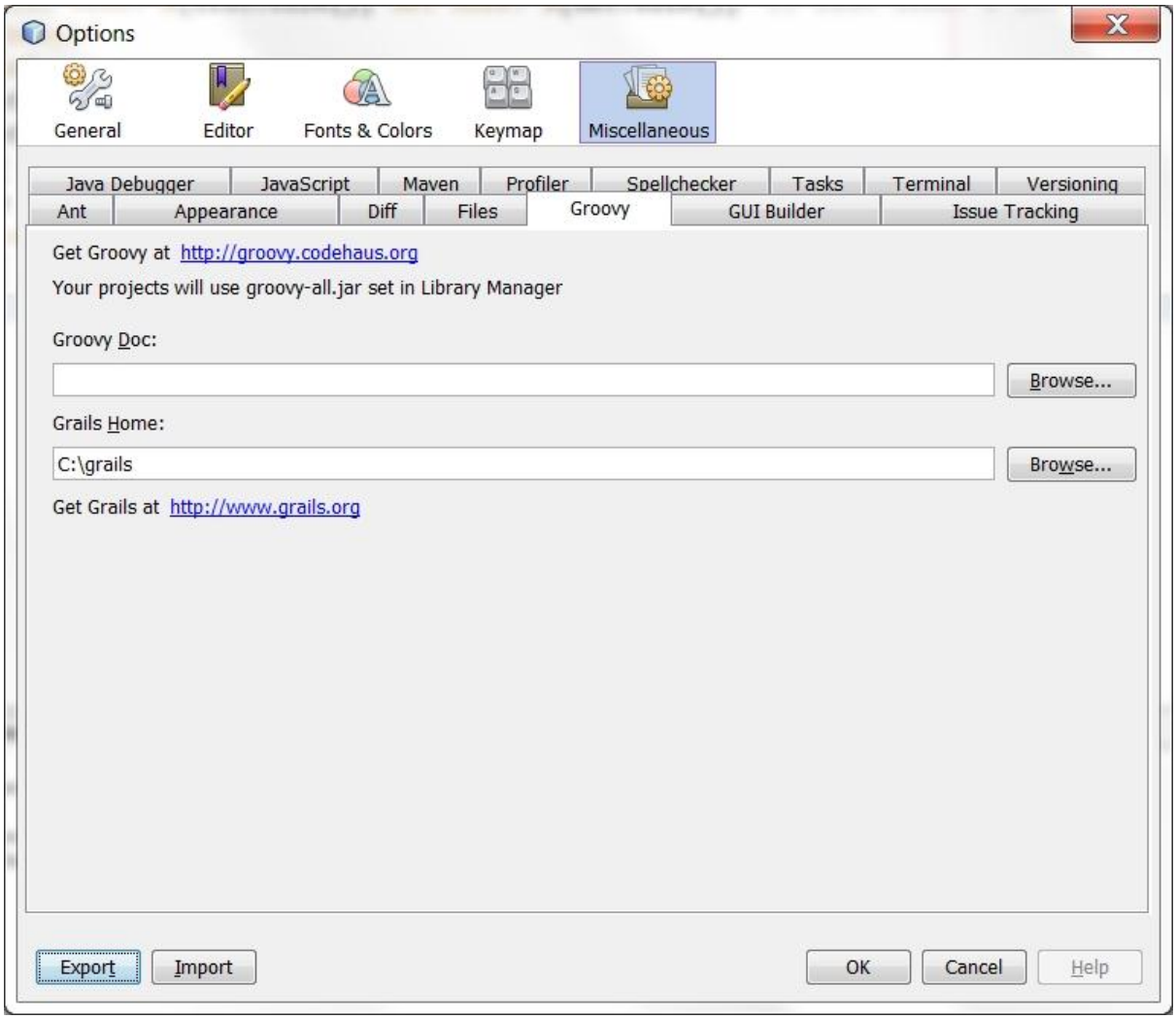
This tutorial will assume Netbeans IDE is used.

6. Writing your first Grails application using Netbeans

Download and install “Netbeans” IDE version 6.5 Java or above to continue with this tutorial.

In Netbeans choose Tools> Plugins > Choose Groovy to install the Grails Netbeans plugin.

Then, go to Tools>Options window and set the location of Grails in the "Groovy" panel.



Once you have installed “Netbeans” and “Grails Framework” you can create new projects easily. Let us create a simple “Employee Manager” web based system for easy management of employees in different “Departments”. The main purpose of this application is to explain persistence using the Grails Object Relational Mapping (GORM) API. This application will focus on how to develop One to Many object relationship in a Grails Project. We will have two domain classes called **Employee** and **Department**. Since a “Department” can have many “Employees”, we will map them by a one-to-many relationship.

1. In the IDE Choose File > New Project and select **Grails Application** from the **Groovy** category. Click next.

2. In the Project Name, type “EmployeeManager”; in Project Location, select the folder where you want the project folder to be created. Click Finish.

Your project is created. Expand the folders and have a look at the source structure created by the IDE via the Grails scripts. Also look at the generated files and notice that many of them have default values filled in.

3. Right click on the “Domain Class” folder > New > Grails Domain Class.
   Name the domain class “Employee” and click Finish.
   The Employee.groovy domain is created in the **Domain Class/employeemanager** node.

   Note: The names of database tables are automatically inferred form the name of the domain class. This can cause problems if the table name matches a keyword in your database. For example, the domain class Group becomes the table ‘group’ by default, but this is a keyword in MySQL. In such cases, try a different name such as “UserGroup”.

4. Open the Employee.groovy domain class and fill in with the following attributes:

```
package employeemanager

class Employee {

    String firstName
    String lastName
    Date birthday
    String description

    static constraints = {
        firstName(blank:false, unique:true)
        lastName(blank:false)
        birthday(nullable:true)
        description(maxSize:1000, nullable:true)
    }

    String toString(){
        "$firstName, $lastName"
    }
}
```

Notice the constraints portion of code. These are nothing but rules governing the values of the properties. They are self-explanatory.

We have also overridden the default toString() method. The default only returns className:objectID. Here we make the toString() method more helpful.

We don't need to worry about creating the getters and setters: Groovy dynamically generates them for us. Employee also gets lots of new and useful **dynamic methods** whose names are handily self-explanatory:

- Employee.**save()** saves the data to the Employee table in the HSQLDB/ Relational database.
- Employee.**delete()** deletes the data from the Employee table.
- Employee.**list()** returns a list of Employees.

- Employee.**get()** returns a single Employee.

All of these methods and more are simply there when you need them. Notice that Employee doesn't extend a parent class or implement a magic interface. Groovy's Meta programming capabilities make these methods simply appear in the proper place on the proper classes. (Only classes in the grails-app/domain directory get these persistence-related methods.)

5. We will now generate the Views and Controllers for our Model (Domain class).
   Right click Employee.groovy > Generate All
   Views are generated in the **Views and Layouts/employee** node.
   Controllers are generated in the **Controllers/employeemanager** node.
   Expand the folders and examine them.

6. Right click the project > Run
   The project will run and you will be directed to
   http://localhost:8080/EmployeeManager/.
   What you are seeing is the index.gsp of the **Views and Layouts** folder.
   You will see a list of available controllers.
   Click on our EmployeeController and try to **create**, **edit** and **delete** some employees.
   All this functionality is generated by Grails automatically for us.

7. Specifying Many-to-One relationship

We will now create the Department domain class and link it to the Employee domain using a Many-to-One relationship.

1. Create a Department domain class and fill in with the following attributes:

```
package employeemanager

class Department {

    String name
    String description

    static constraints = {
        name(blank:false)
    }

    String toString(){
        name
    }
}
```

2. As in the real world, a department has many employees and an employee has a department. We will edit the Department.groovy and Employee.groovy domain classes to reflect this:

```
package employeemanager

class Department {

    static hasMany = [employees: Employee]

    String name
    String description

    static constraints = {
        name(blank:false)
    }

    String toString(){
        name
    }
}
```

```
package employeemanager

class Employee {

    Department department
    static belongsTo = Department

    String firstName
    String lastName
    Date birthday
    String description

    static constraints = {
        firstName(blank:false, unique:true)
        lastName(blank:false)
        birthday(nullable:true)
        description(maxSize:1000, nullable:true)
    }

    String toString(){
        "$firstName, $lastName"
    }
}
```

3. For **both** of the Domain classes, right click > Generate All. Overwrite if prompted.
4. Run the project. You can now add multiple employees into a department.

8. Plugins

The last part of this tutorial will show how to install and use plugins. We will attempt to do so with the Searchable plugin.

1. In command prompt, cd to the EmployeeManager project folder and type the following:
   grails install-plugin searchable
   the Searchable plugin is installed. More information can be found here:
   http://grails.org/plugin/searchable
2. After installing the plugin, its time to use it!
   Edit Department.groovy like so:

```
package employeemanager

class Department {

    static searchable = true

    static hasMany = [employees: Employee]

    String name
    String description

    static constraints = {
        name(blank:false)
    }

    String toString(){
        name
    }
}
```

   Rerun the project and you will see
   grails.plugin.searchable.SearchableController
   in the list of available controllers. Use it to search for your created departments.

3. You can make employees searchable too by adding `static searchable = true` to the domain class.

9. Conclusion

We have learnt how to create and run a simple Grails web application. We also have seen how to link domain classes with the many-to-one relationship. Of course, there are other relationship patterns. Their implementation can be found here: http://grails.org/doc/latest/guide/GORM.html#gormAssociation.

We also touched on plugins and have seen how easy it is to install and use a plugin. There exists many plugins, for auto complete functionality, SSL and login, Facebook, Twitter, Email, just to name a few. All plugins and their documentation can be found here: http://grails.org/plugins/.