Integrallis

Ideas, Implemented.

Test Driven Development with Grails

By Joseph Faisal Nusairat

Java Development Education Series

www.integrallis.com

Saturday, July 17, 2010

A publication of Integrallis Software, LLC.
www.integrallis.com/en/training
info@integrallis.com

OVERVIEW & OBJECTIVES

- Today we will how to create a Grails application using Spock and TDD to help drive the testing and learning process.

- By the end of the class we should have a simple Grails application developed a small Grails application that is also fully tested.

CAVEAT

- Due to this being a one day class we won't be able to dive into great depths on every Grails topic but we will cover a breadth of coverage and the ability to go out and find more answers.

- We will test the complete Grails system from front to back.

AREAS

- Domains
- Controllers
- Groovy Pages
- Tag Libraries
- Services
- Functional Testing
- URL Mappings

MATERIAL CONVENTIONS

- Each training day consists of a mix of lecture time, discussion time and Q & A, guided exercises and labs

- For the guided exercises you will see a green "follow along" sign on the slides

- For the labs you'll see an orange sign with the lab number

DOWNLOADS

- Let's download and configure the items needed for running Groovy and Grails

- This will be the tools necessary for runniny and using groovy and grails.

DOWNLOADS

JAVA AND GRAILS

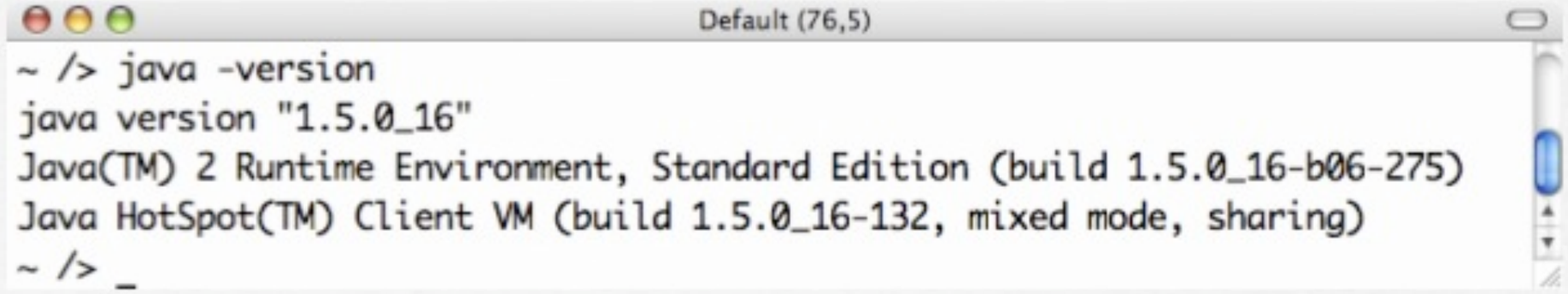
- Java JDK 5 or 6 => http://java.sun.com/javase/
- Grails 1.3.3 => http://grails.org/

If the required software has not been previously installed, the instructor will work with the class to complete the installation

DOWNLOADS

CHECKING JAVA INSTALLATION

- On a command shell, check the version of Java installed in your system.



Any version equal to or higher than 1.5 should suffice

DOWNLOADS

CHECKING GRAILS INSTALLATION

- Once downloaded, uncompress to a directory of your choice.
- Set the GRAILS_HOME env variable.
- Add GRAILS_HOME/bin to your system path

WHAT IS SPOCK?

SPOCK

BASICS OF SPOCK

- Spock is not your average vulcan, it is in fact the newest testing framework out there for Groovy developers.
- This mocking framework was written specifically for Groovy development and not based on just an altered version of an existing framework.
- http://code.google.com/p/spock/

SPOCK

HOW SPOCK DIFERS

- Many things separate Spock from the pack of testing frameworks out there.

- Spock creates assertions that come in a specific expecting block and are not as rudimentary.

- Also data driven tests are more easily written by allowing you an ability to easily test various pieces of data in a test.

- In the end tests are easier to read and create.

SPOCK

NEW ASSERTIONS

- With Spock assertions are created differently. It is written in a more easy to read manner than in other frameworks. This is much more of an easyB like reading than your normal mock testing.

- There are basically two ways of creating tests.

SPOCK

BASIC BLOCKS

- The block usages are created to create tests that are more easily readable and usable

- The basic block usages:

```
• [given: <initialization>] when: <stimulus> then: <condition>
• [given: <initialization>] expect: <condition>
• [given: <initialization>] expect: <condition> where:
  <parameterization>
```


Saturday, July 17, 2010

The when / then model changes a bit when we test domain objects

SPOCK

WHEN - THEN

- With the when / then model, we define what is going to define what we are testing.

```
def testIndex() {
  given :
    controller.todoService = todoServiceMock
  when:
    controller.index()

  then:
   redirectArgs == [action : “list”]
}
```


Saturday, July 17, 2010

The when / then model changes a bit hwne we teset domain objects

SPOCK

DATA DRIVEN TESTS

- With the expect and where we can also use them in a way to make them more data driven tests.
- Data driven tests are an easy way to test a method multiple times while just altering the data slightly.
- This can be very good if its a method that has multiple outcomes. That way instead of multiple methods we are going to just the the one method multiple times.

SPOCK

WHEN - THEN - WHERE - EXPECT

- The special thing about the where clause is it is an ease in testing multiple values which could then have multiple outcomes.

```
def "test isNeeded"() {
  expect:
   todoService.isNeededPriority(a) == b

  where:
    a << [Priority.LOW, Priority.MEDIUM, Priority.HIGH]
    b << [false, true, true]
}
```


Saturday, July 17, 2010

The when / then model changes a bit hwne we teset domain objects

SPOCK

SPOCK AND GRAILS

- The way will use Spock with Grails is via a handy plugin made by the Spock community.

- To install :
  grails install-plugin spock 0.4-groovy-1.7

WHAT IS GRAILS?

GRAILS

FOUNDING OF GRAILS

- Grails is a best of breed framework technology specifically written for the Java community.
- It began work in July of 2005 with an initial release in March of the following year originally called "Groovy on Rails". HOwver the name was dropped due to a request by DHH.
- In November of 2008 it was acquired by Spring Source which was later acquired by VMWare

GRAILS

WHY GRAILS

- Development was just taking too long for web applications. Java applications vs Rails development time was ridiculously bad.

- We had the tools in place but the glue that put them together was not there ... Grails solved that problem.

GRAILS

WHAT MAKES UP GRAILS

- Spring
- Hibernate
- Jetty
- Ant
- SiteMesh
- HSQ DB
- JUnit

GRAILS

BASICS OF GRAILS

- Grails is not just a front end MVC framework, but a complete web application framework that includes the web front end, domain tier, services tier, and whatever else we need that Spring provides.

- Grails uses Spring to drive much of its web and back end with Hibernate handling the persistent tier.

- Grails web tier is still based on the MVC concept though.

LOOKING AROUND GRAILS

GRAILS

CREATING A BASIC GRAILS APPLICATION

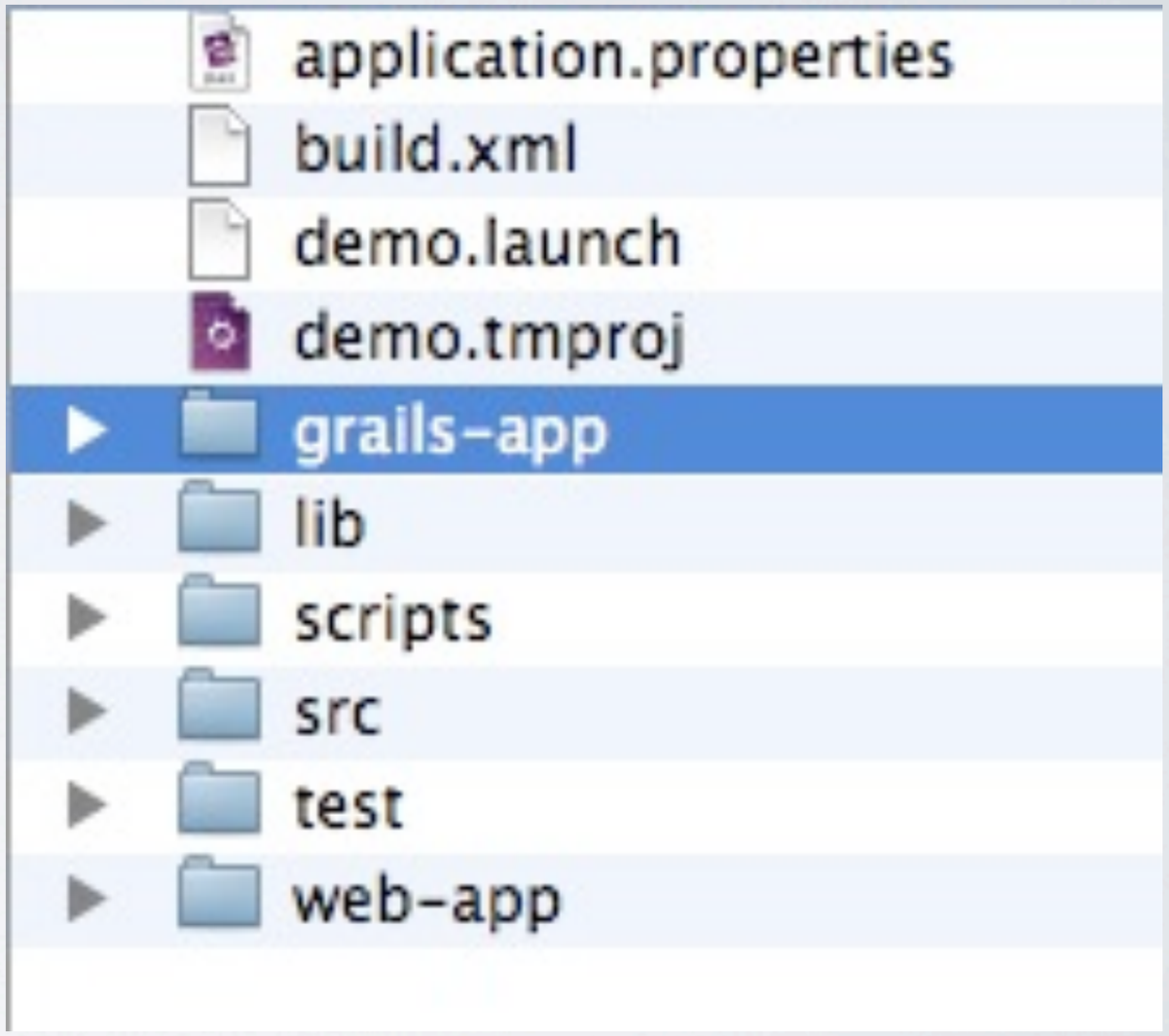
- Creating a basic Grails application is relatively easy once installed, just type:
  grails create-app <app-name>

- Now you will have created a directory of items.

GRAILS

CONSEQUENCES OF THE CREATION

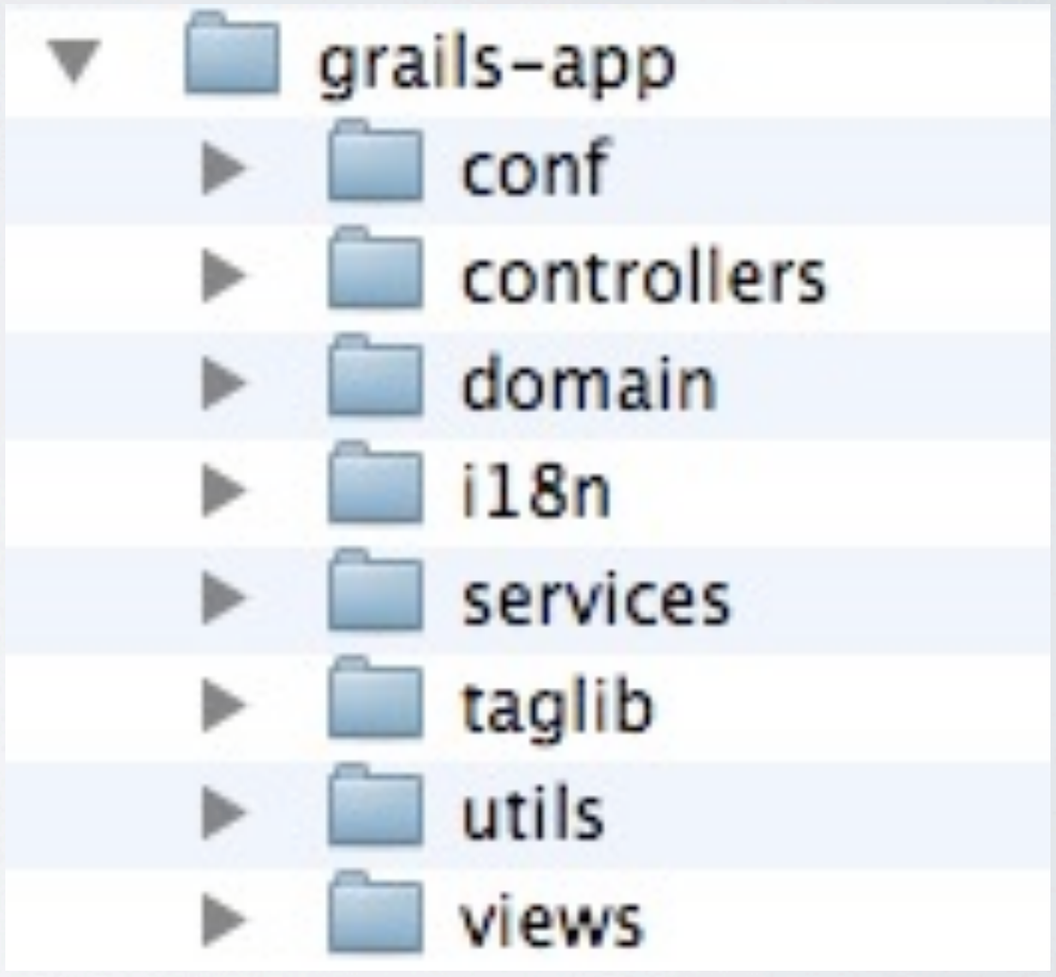
- grails-app is where most of our coding will occur.
- lib is for the jars but not used as much anymore
- test is for our test cases
- web-app stores css, images, javascript



GRAILS

THE GRAILS-APP DIRECTORY

- The app directory contains all the important items.
- This will contain configurations as well as areas that are often auto redeployed (except for domain)



Saturday, July 17, 2010

We will go over what each directory will contain.
We will examine each directory as needed.

DOMAINS

DOMAINS

GRAILS OBJECT RELATIONAL MAPPER

- Domain tier in Grails is based on Hibernate.
- The GORM tier extends, enhances, and creates shortcuts to interact with the database. However at the core all these items are eventually converted into GORM classes.
- GORM provides us the functionality and expandability in Hibernate but made easier to use thanks to Groovy meta-programming.

DOMAINS

CONFIGURING THE DATASOURCE

- By default Grails can only be defined to have one specific datasource. This of course can be over written with more specific items written in other places. However with GORM the default is one datasource.

- We can however define different datasources for development / test / production.

- Configuration changes are made in Datasource.groovy

Saturday, July 17, 2010

There are plugins as well to allow different domains to be defined by different datasources.

```
dataSource {
    pooled = true
    driverClassName = "org.hsqldb.jdbcDriver"
    username = "sa"
    password = ""
}
hibernate {
    cache.use_second_level_cache = true
    cache.use_query_cache = true
    cache.provider_class = 'net.sf.ehcache.hibernate.EhCacheProvider'
}
// environment specific settings
environments {
    development {
        dataSource {
            dbCreate = "create-drop" // one of 'create', 'create-drop','update'
            url = "jdbc:hsqldb:mem:devDB"
        }
    }
    test {
        dataSource {
        }
    }
    production {
        dataSource {
        }
    }
}
```


There are plugins as well to allow different domains to be defined by different datasources.

DOMAINS

CREATING A DOMAIN OBJECT

- Domain objects are created by adding a POGO to the domain class under grails-app.
- Making domain objects is very simple, and its just like writing a regular class.
- The name of the class is the name of the table name, and then each item on the class is a column on that table. And the type also corresponds to that type.

DOMAINS

DOMAIN EXAMPLE

- Here is an example of a domain object "Todo" with 5 columns on it.

```
class Todo {
  String name
  String description
  Priority priority
  Date dueDate
  Category category
}
```

DOMAINS

CRUD OPERATIONS

- Operating on a domain object is very simple and works via meta-programming. Since the class lives under the domain directory there are many methods already wired into it, thus it won't need to have its own DAO to manage it like in traditional Java apps.

```
def todo = new Todo(name : 'my name')
// to save
todo.save()

// or to delete
todo.delete()

// or get by the pk
Todo.get(0)
```

DOMAINS

CONSTRAINTS

- The ability to create constraints allows us to add validation before saving the object and also to be accessed in the controller / services which allows us to pass meaningful messages back to the front end on the validation.

- Grails has many built in constraints and the ability to even add more constraints to there via specific validators.

DOMAINS

DEFINING A CONSTRAINT

- Constraints are defined on the domain class along with the rest of the domain information.
- Please note the default if you don't overwrite any items on the domain is that each item is nullable = false

```
class Todo {
  // ...
  static constraints = {
    name(blank: false)
    dueDate(nullable: true)
}
```

DOMAINS

ERROR MESSAGES

- The error messages for these items are stored in our message properties file. By default a blank message for example will correspond to the key:
  default.blank.message={0} is required.

- However in case we want to customize the message first we can put the name of the class and field ahead of it. So we can get:
  todo.name.blank=Name is required

Saturday, July 17, 2010

To really look at the name possibilities look at
todo.errors.allErrors.each {}

DOMAINS

QUERYING

- Besides the default findBy queries querying can be done by HQL and Criteria queries.

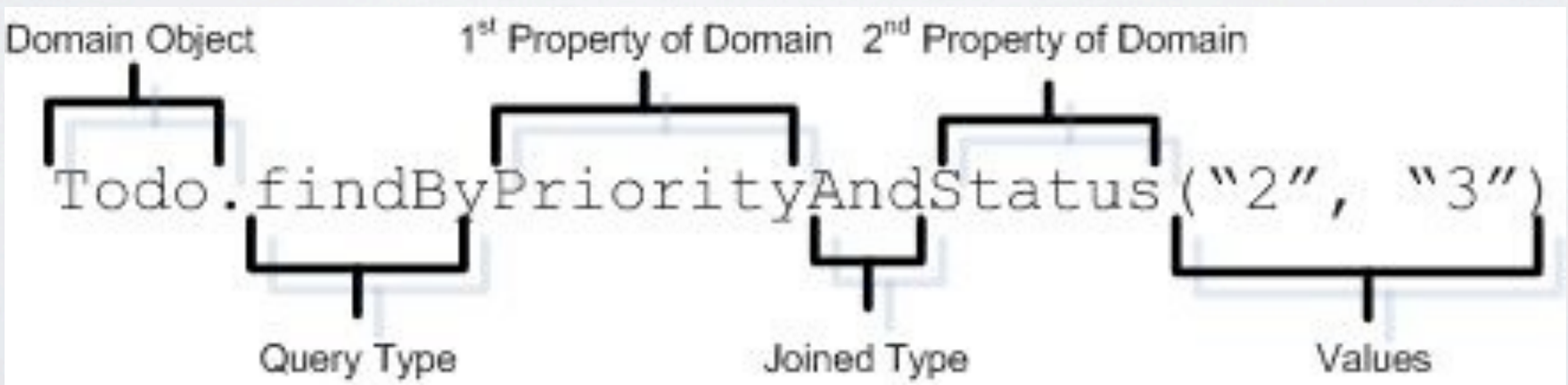
- Querying in Grails can go from the purely meta programming model, to regular HQL querying, to criteria queries.

DOMAINS

DYNAMIC FINDERS

- Grails inherited from Rails the ability to create generic finders. These allow the ability to write a finder thats not actually there.
  Todo.findByNameAndPriority("joseph", Priority.LOW)

- The method does not exist in the class or any item, but exists for us.



DOMAINS

ISSUES WITH DYNAMIC FINDERS

- These are fun to look at from a class room perspective but from a practical perspective it can start to get a bit complicated.

- For one thing you can use MAX of two join criteria. So for generic queries it works but in a bigger set it doesn't.

DOMAINS

HQL QUERIES

- HQL queries are just like the HQL queries we find in Hibernate. You can query the table in a syntax that looks much like SQL but is HQL.

- The big advantage here is that while it supports most of the items most SQL supports its written generically enough that switching database then requires no code changes.

- Hibernate defines dialects for different databases so it knows how to change the code.

Saturday, July 17, 2010

This works well for most SQL databases, but will have an issue if you were to try any noSQL system like mongo.

DOMAINS

HQL QUERY EXAMPLE

- Calling it is easy enough you simply use the domain name followed by findAll or executeQuery. I tend to keep in the habit of using executeQuery since i can also write updates with it.

```
Todo.findAll("from Todo t where t.name = 'joseph'")

Todo.executeQuery("Select t.name from Todo t where
t.priority = :pri", [pri : Priority.Low])

Todo.executeQuery("Select t.name from Todo t where
t.priority = ?", [Priority.Low])
```


Saturday, July 17, 2010

This works well for most SQL databases, but will have an issue if you were to try any noSQL system like mongo.
Please note the "from" must be lower case

DOMAINS

CRITERIA QUERYING

- Criteria queries are a very unique concept to Hibernate and a very powerful robust way. They allow you to express your queries in code.

- With regular hibernate criteria queries can be painful and messing but with GORM it adds a DSL layer around them. This is probably one of the BEST parts of the Gorm querying.

DOMAINS

CRITERIA CONSTRUCTION IN SPRING HIBERNATE

- Even with Spring helping with Criteria construction it is still rather complicated.

```
  public List find(final String name, final Strng desc) {
     List list = getHibernateTemplate().findByCriteria(
DetachedCriteria.forClass(Todo.class)
.add(Expression.eq("name", name)
.add(Expression.eq(“desc”, desc) ); Integer
(pedigreeToteItemDetailId)))
  }
```

DOMAINS

GORM CRITERIA QUERIES

- With GORM we create prettier Criteria queries

```
Todo.createCriteria().list() {
  eq(“name”, “joseph”)
}

Todo.createCriteria().list {
 and {
   like(“name”, “%joseph%”)
   between(priority, Priority.LOW, Priority.HIGH)
 }
 order(‘name’, ‘asc’)
}
```

DOMAINS

MAPPING

- While having the columns and table name reflect the names of our class be a useful thing. For many database and DBAs this isn't a practical thing.

- Many instances we want to change the table name, change the join types, etc. This will allow us to create our code without altering the database.

DOMAINS

CREATING THE TODO

- Lets go over how we create the Todo, Keyword, and Category class

- In addition we will show how to create a unit and mock integration test to make sure it works.

DOMAINS

HOMEWORK

- Create domain classes for our team.
- Write tests cases for at least player to make sure that we can query from it and retrieve data.
- Refer to the next slide for the data that should be on each.

- Player
  name / pos / dateOfBirth / Team / eligible (t/f) / GameType
  (enum .. soccer, football, etc)

- Team
  name / active (t/f)

- PlayerPreviousTeam
  player / team

- Game
  homeTeam / awayTeam / date / homeScore / awayScore

CONTROLLERS

CONTROLLERS

WHAT ARE CONTROLLERS?

- Controllers are the main first area of interaction with the front end and the server. They are the items that will interact with the parameters, request, and session scope directly with the service tier.

- The controllers are based on Spring controllers, each interaction with them is via a closure and not a method.

CONTROLLERS

WHAT GOES IN A CONTROLLER

- The controller should only maintain the logic for formatting the parameters. The processing and the logic should be sent to the server.

- The controllers should also be in charge of where and how to redirect the data to.

CONTROLLERS

CREATING A CONTROLLER

- Creating a controller is simple, one can create it with the command: grails create-controller <controllerName>
- This will create a controller in the grails-app/controller directory as well as a corresponding view directory with the same name under view.
- The only format that must be enforced if my creating by hand is the controller end with the name "Controller"

CONTROLLERS

DEFAULT ATTRIBUTES IN THE CLOSURES

- log - to output log statements
- request - the HttpServletRequest object
- session - the HttpSession instance
- servletContext - the ServletContext instance
- flash - map for the current request and next request only
- params - map of available request / controller parameters

Saturday, July 17, 2010

There are a few more than these, but they are not commonly used
Ones not used: grailsApplication, grailsAttributes, actionUri, controllerUri, actionName ,controllerName

CONTROLLERS

RENDERING AND REDIRECT

- Rendering and redirecting is the core ability to send data to a view or for displaying the data directly to the output or to redirect to another area.

- The nice thing about the rendering / redirect code is its ease in ability to use and its straight forward look.

- Some frameworks like Seam tend to abstract to a layer that makes it even hard to read.

CONTROLLERS

REDIRECETS

- redirect(uri:“/book/list”)
- redirect(url:”http://www.google.com”)
- redirect(action:“show”)
- redirect(controller:‘book’, action :‘list’)
- redirect(action:“show”, id: 4, params: [author :‘King’])

CONTROLLERS

RENDER

- render “some text”
- render (text:“<xml>Some xml</xml>”, contentType :“text/xml”, encoding:“UTF-8”)
- render(template:‘book’, model:[book : my bookVar])
- render(template:‘book’, collection : myCollectionVar)
- render(template:‘book’, bean : myBeanVar)

CONTROLLERS

RENDER

- render (view: "viewName", model:[book : my bookVar])
- render(view: 'myView')
- render {div(id:'myDiv',"some text inside") }

CONTROLLERS

COMMAND PATTERN

- Normally one can access any variables set in classes quite easily. We can grab the objects set in the parameters, then pass them to the services, even formatting when possible.

```
def save = {
  log.info “save this name ${params.name}”
  if (params.name != null) {
   nameService.save(params.name)
  }
  else {
   flash.message “Name cannot be null”
  }
}
```

CONTROLLERS

COMMAND PATTERN

- As you can see this can be very messy the more variables we add. Not only that but the validation can get complicated.

- Especially if what you pass in is different than what you are going to save. This is where the formatting of data is much different than the saving of items.

- For example like needing a password and password confirmation.

CONTROLLERS

COMMAND PATTERN

```
class UpdateUserCommand {
  String name
  String password
  String confirmPassword

  static constraints = {
   name(blank:false)
   password(blank:false, validator : {
     obj.properties['confirmPassword'] == val
   }
  }
}
```

CONTROLLERS

COMMAND PATTERN

```
def update = { UpdateUserCommand cmd ->
  if (cmd.hasErrors()) {
    flash.errors("Has errors")
  }
 else {
   nameService.save(cmd)
  }
}
```

CONTROLLERS

CREATING THE TODO CONTROLLER

- Using some generic scaffolding we will use them to create the controller mocks. Not the best way but fastest way of doing it.

- We shall then create ControllerSpec tests to test these items out.

CONTROLLERS

HOMEWORK

- Controllers for each of the items generically.
- Implement the command pattern we went over for the game.
- Write a spec test for at least one of the controllers

- Create for the following controllers
  - Player
  - Team
  - Game

SERVICES

SERVICES

PURPOSE OF SERVICES

- The purpose of a Service is to provide the business logic portion of our application.

- We want to keep logic out of the controller and domains, this allows logical separation of items.

SERVICES

WHAT ARE THE SERVICES?

- At the core the services are Spring beans. They will be stored and can be accessed from the Spring application context. This makes them no different than any other Spring bean you have used before except now it requires less configuration to create.

SERVICES

CREATING A SERVICE

- Services are any items created inside of the "services" folder and are any groovy classes ending with the word "Service"
- If the name of the services is "SearchService" in the package "com.tekspike", you will be able to look up the service
- In addition by default all services are transactional.

SERVICES

CALLING A SERVICE

- Services are extremly easy to call because they are stored in the spring scope and thus can either be looked up or injected anywhere they are needed.

- For most purposes we will just inject them into our objects. They can be injected into the controllers, tag libraries, or other services.

SERVICES

INJECTING THE SERVICE

- Injecting the service is based off of the name.

```
class TodoService {
  def save() {
   // save
  }
}
```

```
class TodoController {
  def todoService

  def saveUpdate = {
   todoService.save()
  }
}
```

SERVICES

INJECTING THE SERVICE

- Retrieving the service directly via code.

```
import org.codehaus.groovy.grails.commons.ApplicationHolder
…
ApplicationContext ctx = (ApplicationContext)
ApplicationHolder.getApplication().getMainContext();
CountryServiceInt service = (CountryServiceInt) ctx.getBean
("countryService");
String str = service.sayHello(request.getParameter.("name"));
//do something with str
```

SERVICES

TRANSACTIONING

- Running a transaction can be an expensive operation and should only be done when necessary.
- With previous versions of Grails transactioning was not as verbose as it was in Spring. This changed with Grails 1.2.0
- Detailed transactioning can only be controlled at the per method level

SERVICES

ENABLING / DISABLING TRANSACTIONS

- Transactions can either be enabled class wide or per a method.
- The default when nothing is marked is to make the transaction class wide.
- Class level transactioning can be enabled / disabled via:

```
class TodoService {
  static transactional = false

}
```

SERVICES

ENABLING / DISABLING TRANSACTIONS

- Transactioning at a per method level has many more options that makes use of transaction

```
import org.springframework.transaction.annotation.*

class TodoService {

  @Transactional(readOnly = true)
  def save() { }
}
```

SERVICES

OPTIONS ON TRANSACTIONING

- There are a variety of options you can set on the transactioning
- propagation : be it requires new, supported, etc
- isolation
- readOnly
- rollbackFor / rollbackForClassname
- noRollbackFor / noRollbackForClassname

http://static.springsource.org/spring/docs/1.2.x/reference/transaction.html

SERVICES

SERVICE SCOPES

- In addition to transactioning one can set the scope the service is going to be in.
- By default all services will be a singleton
- However setting the scope is as easy as adjusting a static variable on the class.

```
class TodoService {

  static scope = “flash”

  def save() { }
}
```

SERVICES

POSSIBLE SCOPES

- The only thing to really be careful with scopes and injections is the request scope. If you try to auto inject a request scope into an item that doesn't necessarily have it you may get unexpected results.

- The allowed scopes are as follows:
  - flash / singleton / prototype / flow / conversation / session / request

SERVICES

CALLING TAGS AND SESSIONS FROM THE SERVICE

- The one thing nice about Grails is its ability to access anything from everywhere. So we can in theory access request data from the service and even the tag library.

- This should be done CAUTIOUSLY though since accessing request / session data should be done optimistically in your code.

- This is more useful when creating Services that aren't really "services" in the traditional sense.

Saturday, July 17, 2010

This is more of a bonus material.
And this in fact is a bit dangerous to do as well.

SERVICES

CALLING TAGS AND SESSIONS FROM THE SERVICE

```
import org.springframework.web.context.request.RequestContextHolder
import static
org.springframework.web.context.request.RequestAttributes.SCOPE_REQU
EST
import
org.codehaus.groovy.grails.plugins.web.taglib.ApplicationTagLib


class TodoService {
  def todoService

  def saveUpdate = {
    def pub =  RequestContextHolder.currentRequestAttributes()
     .getAttribute('publisher',SCOPE_REQUEST)
    ApplicationTagLib g = new ApplicationTagLib()
    def name = g.message(code: ‘default.new’)
  }
}
```


Its also good to put the scope=”request”
Please note this is only good for accessing grails tag libs

SERVICES

CREATING THE TODO SERVICE

- Here we will show how to use the TodoService to create a generic service class.

SERVICES

HOMEWORK

- Create a service to schedule a game with.
- The service is a bit complicated in that it needs to call out the domain methods to retrieve items.
- Certain rules must be in place to create a game:
  - 5 eligible players
  - Both teams active
- Use some mock testing to create this with.

TAG LIBRARIES

TAG LIBRARIES

WHAT ARE TAG LIBRARIES

- Tag libraries have been around since the early days of JSP and heavily used in Struts; however, making use of them required quite a bit of pain.

- One had to have one class per each tag, the tag had to extend base classes, you had to create an XML definition for it, and then define that XML in the web.xml.

- This was quite the mess, Grails has simplified tag development.

TAG LIBRARIES

WHAT ARE TAG LIBRARIES

- Tags are used in GSP pages where you have code that does not make sense to have in the Controller but you do not want to put into the page as a scriptlet.

- This allows the code to be accessed easily and be reused for the GSP.

- Often times this is used to control the formatting of data, be careful not to put too much business logic in there.

TAG LIBRARIES

CREATING A TAG LIBRARY

- Tag libraries are easy to create and are merely classes endign with TagLib inside of the taglib directory.

- In the tag library we can identify the name space to store it in, if not it will go to the default grails namespace.

- When creating tag libraries remember the default behavior is to output a streaming char buffer.

TAG LIBRARIES

CREATING A TAG LIBRARY

```
class SecurityTagLib {
  static namespace = “s”
  def name = {attrs ->
    out << attrs.user
  }
}

<s:name user=”joseph”/>
```

TAG LIBRARIES

CREATING A TAG LIBRARY

```
class SecurityTagLib {
  static namespace = “s”
  def name = {attrs, body ->
    if (attrs.user == “true”) {
      out << body()
    }
  }
}

<s:name user=”true”>
  Joseph
</s:name>
```

TAG LIBRARIES

CREATING A TAG LIBRARY

```
class SecurityTagLib {
  static namespace = “s”
  static returnObjectForTags = ['user']

  def userService

  def user = {attrs
   userService.get(attrs.id)
  }
}

${s.user(3)}
```


Saturday, July 17, 2010

In this case we have the actual user object beign returned
THis is very good when setting a bean or needing aclass to look up items.

TAG LIBRARIES

HOMEWORK

- A fairly simple tag library, we will display a player in a more readable format since player info can be repeated often.

- ShowPlayersPreviousTeam

  - Process a lit of there previous teams to be returned to the page as a list object.

GROOVY PAGES

GROOVY PAGES

WHAT IS A GSP?

- GSP are groovy versions are converted into JSP pages upon complication.
- GSP pages provide features not normally found in JSP pages. The most obvious ones are :
  - Site Mesh
  - Express Groovy Items
  - Groovy Tag Libs

GROOVY PAGES

HOMEWORK

- At this point we shall put it all together in creating a uniformed page that works.

URL MAPPINGS

URL MAPPINGS

WHAT ARE TAG LIBRARIES

- In an age of RESTful URLs becoming all the norm and also a necessity for a successful web site. They can also be used to help hide what controllers and actions you are using.

- With Grails creating custom mappings is extremely simple and very powerful to use.

- The mapping are located in UrlMappings.groovy

URL MAPPINGS

CREATING A BASIC MAP

- The basic mappings basically just create a different URL and forward it to a different controller and action
- We can even just forward in a more generic basis where we define the new controller name then let it pass through whatever action.

```
class UrlMappings {
  static mappings = {
    "/api/charge/"(controller: 'charge', action: 'index')
    "/api/custom/$action" (controller: 'custom')
}
```

URL MAPPINGS

CREATING A COMPLEX MAPPING

- However this is not the more interesting mappings.
- More interesting mappings allow us to forward based on more dynamic variables.
- The way the mappings are they will try to map the most specific and work there way to the least specific.

URL MAPPINGS

CREATING A COMPLEX MAPPING

```
"/$state/$city?/$year?/$month?"
  (controller : "camp", action : "find") {
            constraints {
                //city(inList:["phoenix","columbus"])
                state(validator : {
                  if (Camp.findByState(it) != null
                        || CampRequest.findByState(it) != null) {
                        return true
                    } else {
                        return false
                    }
                })
            }
    }
```


Saturday, July 17, 2010

This is an example where we can use constraints to decide where to go to

URL MAPPINGS

HOMEWORK

- Start with hard coding that /football /soccer forward to URLs
- We are going to want to create an easy mapping so that when going to a URL you can go to /<team> and get the team name, or go to /<player> and get the player information.