Grails

The most advanced Spring use case
Special Acknowledgement

- Many thanks to Scott Davis!
  - davisworld.org
  - thirstyhead.com

경영

Groovy Recipes
Greasing the Wheels of Java

scott@davisworld.org
Why do we use Spring?
TESTING
I FIND YOUR LACK OF TESTS DISTURBING.
Productivity
Maintainability
Let’s talk about Java!
Let’s talk **honestly** about Java!
Java (the language) was...

- Great in its day
- Is struggling to be...
  - Expressive
  - Concise
  - Productive
Case #1

What is the output of?

System.out.println(2.0-1.1);
Case #2

- Generics are Broken
- Complete the following line:
  `List<String> list = ...`
Case #3

- What does Java 7 Promise for the Language?
  - closures are in
    - no... they are out...
  - They are back in...
    - they’re out...
  - ok... ARM Blocks are in

- Once again closures are fine and good for the language designers, but not for the language users.
What is Groovy

- Dynamic Programming for JVM
- Supports typed and untyped
- Primitives are treated as objects
- List and Hash literals
- Closures
- Operator Overloading
What is the output of?

println( 2.0 - 1.1);
Groovy Case #2

- Creating a list in groovy

```groovy
List names = [ "ken", "craig", "jay" ]
```
task(compile:"The compilation task") {
    depends(clean, init)
    Ant.javac(srcdir:"src/java",
              destdir:"build/classes"
    )
}

task('default':"The default task") {
    depends(compile, jar)
}
Grails
What is Grails?

- MVC action-based web framework inspired by:
  - Convention over configuration
  - Don’t Repeat yourself (DRY)
  - Rails, Django, and TurboGears
What is Grails?

Grails is a fully integrated modern Java web application in a box:
Included JARs

- ant-junit.jar
- ant-launcher.jar
- ant-nodeps.jar
- ant-trax.jar
- ant.jar
- antlr-2.7.6.jar
- bsf-2.3.0.jar
- cglib-nodep-2.1_3.jar
- commons-beanutils-1.7.0.jar
- commons-cli-1.0.jar
- commons-collections-3.2.jar
- commons-dbcp-1.2.1.jar
- commons-el-1.0.jar
- commons-fileupload-1.1.1.jar
- commons-io-1.2.jar
- commons-lang-2.1.jar
- commons-logging-1.1.jar
- commons-pool-1.2.jar
- commons-validator-1.3.0.jar
- dom4j-1.6.1.jar
- ehcache-1.2.4.jar
- ejb3-persistence.jar
- gant-1.1.0_groovy-1.5.2.jar
- groovy-all-1.5.4.jar
- hibernate-annotations.jar
- hibernate3.jar
- hsqldb-1.8.0.5.jar
- jasper-compiler-5.5.15.jar
- jasper-compiler-jdt-5.5.15.jar
- jasper-runtime-5.5.15.jar
- jaxen-1.1-beta-11.jar
- jdbc2_0-stdext.jar
- jetty-6.1.4.jar
- jetty-naming-6.1.4.jar
- jetty-plus-6.1.4.jar
- jetty-util-6.1.4.jar
- jline-0.9.91.jar
- jsp-api-2.0.jar
- jstl-2.3.jar
- jstl-2.4.jar
- jta.jar
- junit-3.8.2.jar
- log4j-1.2.15.jar
- ognl-2.6.9.jar
- oro-2.0.8.jar
- serializer.jar
- servlet-api-2.5-6.1.4.jar
- sitemesh-2.3.jar
- spring-2.5.1.jar
- spring-binding-2.0-m1.jar
- spring-test.jar
- spring-webflow-2.0-m1.jar
- spring-webmvc.jar
- springmodules-sandbox.jar
- standard-2.3.jar
- standard-2.4.jar
- start.jar
- svnkit.jar
- xalan.jar
- xercesImpl.jar
- xpp3_min-1.1.3.4.0.jar
- xstream-1.2.1.jar
Included Ajax Support

/* Prototype JavaScript framework, version 1.6.0
 * (c) 2005-2007 Sam Stephenson
 */

/* script.aculo.us scriptaculous.js v1.8.0,
 * Tue Nov 06 15:01:40 +0300 2007
 * Copyright (c) 2005-2007 Thomas Fuchs
 */
(Almost) Included Ajax Support

$ grails install-dojo
-- Installs the Dojo toolkit.
An advanced Javascript library.
Grails Plugins

This page contains links to the documentation for each plugin that is available for Grails.

Testing

- Functional Testing with Canco WebTest
- Test Code Coverage Plugin

Rich Client/Ajax Plugins

- Converters Plugin
- DWR Plugin
- Dynamic Javascript Plugin
- Echo2 Plugin
- GWT Plugin
- OpenLasdo Plugin
- RichUI Plugin
- ZK Plugin
- Flex Plugin
- ULC Plugin
- ModalBox Plugin

Chart Plugins

- Google Chart Plugin
- JFreeChart Eastwood Plugin
Act 1:
For Those in a Hurry...
Installing Grails

http://grails.org

- Download/unzip grails-bin.tar.gz (or zip)
- Create GRAILS_HOME
- Add $GRAILS_HOME/bin to PATH
Your 1-Slide Guide to Grails

Type the following:

```bash
$ grails create-app bookstore
$ cd bookstore
$ grails create-domain-class Book
   (Add fields to
    grails-app/domain/Book.groovy)
$ grails generate-all Book
$ grails run-app
```

$ grails help -- shows all available commands
<table>
<thead>
<tr>
<th>Id</th>
<th>Author</th>
<th>Pages</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scott Davis</td>
<td>300</td>
<td>Groovy Recipes</td>
</tr>
<tr>
<td>2</td>
<td>Scott Davis</td>
<td>287</td>
<td>JBoss at Work</td>
</tr>
<tr>
<td>3</td>
<td>Scott Davis</td>
<td>268</td>
<td>GIS for Web Developers</td>
</tr>
</tbody>
</table>
Generated Show

Show Book

Id: 1
Author: Scott Davis
Pages: 300
Title: Groovy Recipes

Edit Delete
Act 2: Tweaking the defaults...
Changing the Port

- Grails / Jetty runs on port 8080 by default
  - Option #1: change the port at runtime
    
    ```
    $ grails -Dserver.port=9090 run-app
    ```

  - Option #2: edit `GRAILS_HOME/scripts/Init.groovy`
    (see next page...)
serverPort = System.getProperty('server.port')?
System.getProperty('server.port').toInt() : 9090
Changing Grails Environments

```
grails run-app       // runs with the default "development" data source
grails dev run-app   // runs with the "development" data source
grails prod run-app  // runs with the production data source
grails test run-app  // runs with the test data source
```

- **Dev** (the default) auto-reloads changes to Controllers, Views, and even the Model
  - This is helpful for rapid development

- **Prod** loads all items statically for maximum performance
Changing the Database

```groovy
dataSource {
    pooled = false
    driverClassName = "org.hsqldb.jdbcDriver"
    username = "sa"
    password = ""
}
// environment specific settings
environments {
    development {
        dataSource {
            dbCreate = "create-drop" // one of 'create', 'create-drop','update'
            url = "jdbc:hsqldb:mem:devDB"
        }
    }
    test {
        dataSource {
            dbCreate = "update"
            url = "jdbc:hsqldb:mem:testDb"
        }
    }
    production {
        dataSource {
            dbCreate = "update"
            url = "jdbc:hsqldb:file:prodDb;shutdown=true"
        }
    }
}
```
Why does my data go away?

- `dbCreate == hibernate.hbm2ddl.auto`
  - Create-drop -- creates the tables on startup, drops them on shutdown (DEV)
  - Create -- creates the tables on startup, just deletes the data on shutdown
  - Update -- creates the tables on startup, saves the data between restarts (PROD, TEST)

- Remove the value to manage the schema manually
Changing to MySQL

1) Create the database and user
2) Copy the driver into lib
3) Adjust values in grails-app/conf/DataSource.groovy
Create the database

```
$ mysql --user=root
Welcome to the MySQL monitor.

mysql> create database bookstore_dev;
mysql> use bookstore_dev;
mysql> grant all on bookstore_dev.* to grails@localhost identified by 'server';
mysql> flush privileges;
```

Sanity check the newly created login:

```
$ mysql --user=grails -p
--database=bookstore_dev
```
Point Grails to MySQL

```groovy
dataSource {
    pooled = false
    driverClassName = "com.mysql.jdbc.Driver"
    username = "grails"
    password = "server"
}

// environment specific settings
environments {
    development {
        dataSource {
            dbCreate = "update" // one of 'create', 'create-drop','update'
            url = "jdbc:mysql://localhost:3306/bookstore_dev?autoreconnect=true"
        }
    }
    test {
        dataSource {
            dbCreate = "update"
            url = "jdbc:hsqldb:mem:testDb"
        }
    }
    production {
        dataSource {
            dbCreate = "update"
            url = "jdbc:hsqldb:file:prodDb;shutdown=true"
        }
    }
}
```
```
mysql> show tables;
+-------------------------+
| Tables_in_bookstore_dev |
+-------------------------+
| book                    |
+-------------------------+
mysql> desc book;
+---------+--------------+------+-----+
| Field   | Type         | Null | Key |
+---------+--------------+------+-----+
| id      | bigint(20)   | NO   | PRI |
| version | bigint(20)   | NO   |     |
| title   | varchar(255) | NO   |     |
| author  | varchar(255) | NO   |     |
+---------+--------------+------+-----+
```
Changing the Web server

- To run your app in Tomcat instead of Jetty:

  ```
  $ grails war
  $ cp bookstore.war /opt/tomcat/webapps/
  ```

  Gotcha: Grails WARs run in PROD by default.

  ```
  $ grails dev war
  ```

  Or run your container with
  JAVA_OPTS=-Dgrails.env=development
Changing the Home Page

The default homepage is web-app/index.gsp. You can redirect to any page or controller:
Act 3: Understanding Grails Controllers...
Auto-scaffolding

```java
class Publisher{
  String name
}
```

```java
class PublisherController{
  def scaffold = Publisher
}
```
Generating a Controller

$ grails generate-controller
Each controller closure ends in one of three ways:

- **Redirect**
  - Equivalent to `response.sendRedirect()`
    - `redirect(action:list, params:params)`

- **Return**
  - Calls a GSP named the same as the method
    - `return [ bookList: Book.list(params) ]`

- **Render**
  - Calls a GSP of an arbitrary name
    - `render(view:'edit', model:[book:book])`
Controller.index

```groovy
class BookController {
    def index = { redirect(action:list, params:params) }
}
```

- Index is the default target, just like index.jsp or index.html
- **Params** is a Map of the QueryString name/value pairs
- `redirect()` == `response.sendRedirect()`
- `action:list` == the list closure in this controller
def list = {
    if(!params.max) params.max = 10
    [ bookList: Book.list( params ) ]
}

Implicit return statement

GORM (Grails Object/Relational Mapping)

Map of named objects in the Response
(see list.gsp, next page)
List.gsp

Returned from Controller
List view

Book List

<table>
<thead>
<tr>
<th>Id</th>
<th>Author</th>
<th>Pages</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scott Davis</td>
<td>300</td>
<td>Groovy Recipes</td>
</tr>
<tr>
<td>2</td>
<td>Scott Davis</td>
<td>287</td>
<td>JBoss at Work</td>
</tr>
<tr>
<td>3</td>
<td>Scott Davis</td>
<td>268</td>
<td>GIS for Web Developers</td>
</tr>
</tbody>
</table>
Convention over Configuration

- **BookController**

- **BookController.list**
  - Corresponding list.gsp

- **BookController.show(5)**
Show view
def show = {
    [ book : Book.get( params.id ) ]
}

<div class="dialog">
    <table>
        <tbody>
            <tr class="prop">
                <td valign="top" class="name">Id:</td>
                <td valign="top" class="value">${book.id}</td>
            </tr>
            <tr class="prop">
                <td valign="top" class="name">Author:</td>
                <td valign="top" class="value">${book.author}</td>
            </tr>
            <tr class="prop">
                <td valign="top" class="name">Title:</td>
                <td valign="top" class="value">${book.title}</td>
            </tr>
        </tbody>
    </table>
</div>
Create.gsp

Controller Method

```groovy
<g:form action="save" method="post" >
  <div class="dialog">
    <table>
      <tbody>
        <tr class='prop'>
          <td valign='top' class='name'>
            <label for='author'>Author:</label>
          </td>
        </tr>
        <tr class='prop'>
          <td valign='top' class='name'>
            <label for='title'>Title:</label>
          </td>
        </tr>
      </tbody>
      <div class='buttons'>
        <span class='formButton'>
          <input type='submit' value='Create'/>
        </span>
      </div>
    </table>
  </div>
</g:form>
```
In one line, Param name/value pairs from the form are saved to a POGO (Plain Old Groovy Object).

In the next line, the POGO is saved to the database via GORM.
Act 4:
Understanding Grails Models...
...and Views...
...and GORM...
Plain Old Groovy Objects
- Fields are automatically private
- Getters and setters are automatically provided
- Use Wrappers instead of Primitives
  - Integer, Float, Double, Boolean
Specifying Field Order

class Book {
    static constraints = {
        title()
        author()
        cover()
        pages()
        category()
    }

    String title
    String author
    Date publicationDate
    Integer pages
    String cover
    String category
    String isbn
}
## Ordered Fields in List

![Book List](http://localhost:9090/bookstore/book/list)

**Book List**

<table>
<thead>
<tr>
<th>Id</th>
<th>Title</th>
<th>Author</th>
<th>Cover</th>
<th>Pages</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Groovy Recipes</td>
<td>Scott Davis</td>
<td>Paperback</td>
<td>300</td>
<td>Technical</td>
</tr>
<tr>
<td>6</td>
<td>JBoss at Work</td>
<td>Scott Davis</td>
<td>Paperback</td>
<td>287</td>
<td>Technical</td>
</tr>
<tr>
<td>7</td>
<td>Google Maps API</td>
<td>Scott Davis</td>
<td>PDF</td>
<td>75</td>
<td>Mapping</td>
</tr>
</tbody>
</table>
Field Validation

```java
class Book {
    static constraints = {
        title(blank: false, maxSize: 50)
        author(blank: false)
        cover(blank: false, inList: ["Hardback", "Paperback", "PDF"])
        pages(min: 0, max: 1500)
        category(blank: true, inList: ["", "Technical", "Fiction", "Non-fiction"])
        excerpt(maxSize: 5000)
    }

    String title
    String author
    Date publicationDate
    Integer pages
    String cover = "Paperback"
    String category
    String isbn
    String excerpt
}
```
Create Book

Title: 
Author: 
Cover: Paperback
Pages: 
Category: 
Excerpt: 

ISBN: 
Publication Date: 14 February 2007 23:39
Create Book

- Property [pages] of class [class Book] with value [-1] is less than minimum value [0]
- Property [title] of class [class Book] cannot be blank
- Property [author] of class [class Book] cannot be blank

Title: 

Author: 

Cover: Paperback

Pages: -1
```sql
mysql> desc book;
+------------------+--------------+
| Field            | Type         |
+------------------+--------------+
| id               | bigint(20)   |
| version          | bigint(20)   |
| title            | varchar(50)  |
| pages            | int(11)      |
| category         | varchar(255) |
| isbn             | varchar(255) |
| excerpt          | text         |
| publication_date | datetime     |
| cover            | varchar(255) |
| author           | varchar(255) |
+------------------+--------------+
```
GORM: One-to-many

```java
class Publisher{
    static hasMany = [books:Book]

    String name

    String toString() {
        return name
    }
}

class Book {
    static constraints = {
        title(blank: false, maxSize: 50)
        author(blank: false)
        cover(blank: false, inList: ["Hardback", "Paperback", "PDF"])
        pages(min: 0, max: 1500)
        category(blank: true, inList: ["", "Technical", "Fiction", "Non-fiction"])
        excerpt(maxSize: 5000)
    }

    static belongsTo = Publisher

    String title
    String author
    Date publicationDate
    Integer pages
    String cover = "Paperback"
    String category
    String isbn
    String excerpt
    Publisher publisher
}
Create Book

Title:
Author:
Cover: Paperback
Pages:
Category:
Excerpt:
Publisher: O'Reilly
ISBN:
Publication Date: 25 February 2007
Act 5: Understanding Grails Plugin Architecture...
Grails Extension Points

- Spring application context
- Dynamic method registration
- Auto Reloading
Plug-in Architecture

- Plugin
  - Read Conventions
  - GrailsApplication
    - Grails Runtime Environment

- Plugin
  - Register Bean Definitions
  - ApplicationContext
A Plug-in can:
- doWithSpring
  - participate in Spring config
- doWithApplicationContext
  - post application context initialization activities
- doWithWebDescriptor
  - modify the xml generated for web.xml at runtime
- doWithDynamicMethods
  - add methods
- onChange
  - participate in reload events
Configuring Spring

```java
// Configuring Spring

class JcrGrailsPlugin {
    def version = 0.1
    def dependsOn = [ core:0.4]

    def doWithSpring = {
        jcrRepository(RepositoryFactoryBean) {
            configuration = "classpath:repository.xml"
            homeDir = "/repo"
        }
    }
}
```

Bean name is the method name. First argument is the bean type.

Set properties on the bean.
```java
class I18nGrailsPlugin {
    def version = "0.4.2"
    def watchedResources = "file:..//grails-app/i18n/*\.properties"

    def onChange = { event ->
        def messageSource =
            event.ctx.getBean("messageSource")

        messageSource?.clearCache()
    }
}
```

- Defines a set of files to watch using Spring resource pattern
- When one changes, event is fired and plug-in responds by clearing message cache
Grails Plug-ins

- XFire
  - Exposes grails as a SOAP service
- Searchable
  - Integrates Lucene search
- Remoting
  - Exposes Grails over RMI, HTTP, or burlap
- JMX
  - Exposes Mbeans
- Acegi
  - Adds security support
- JMS
  - Exposes Grails as JMS message driven beans
Conclusion

- Grails is a fully integrated modern Java web application in a box:
Summary

- Groovy
- Grails
- Productivity knows no bounds!
Questions

- Please Fill Out Surveys

  kensipe@gmail.com
  twitter: @kensipe
  kensipe.blogspot.com