Experts in delivering business-driven technology solutions.

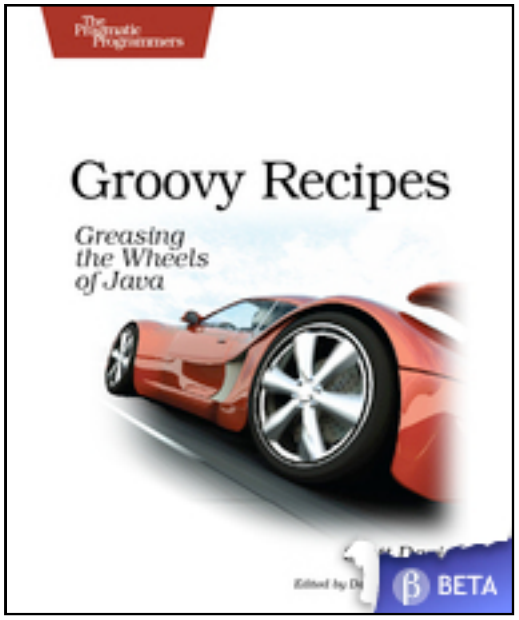
Grails

The most advanced Spring use case

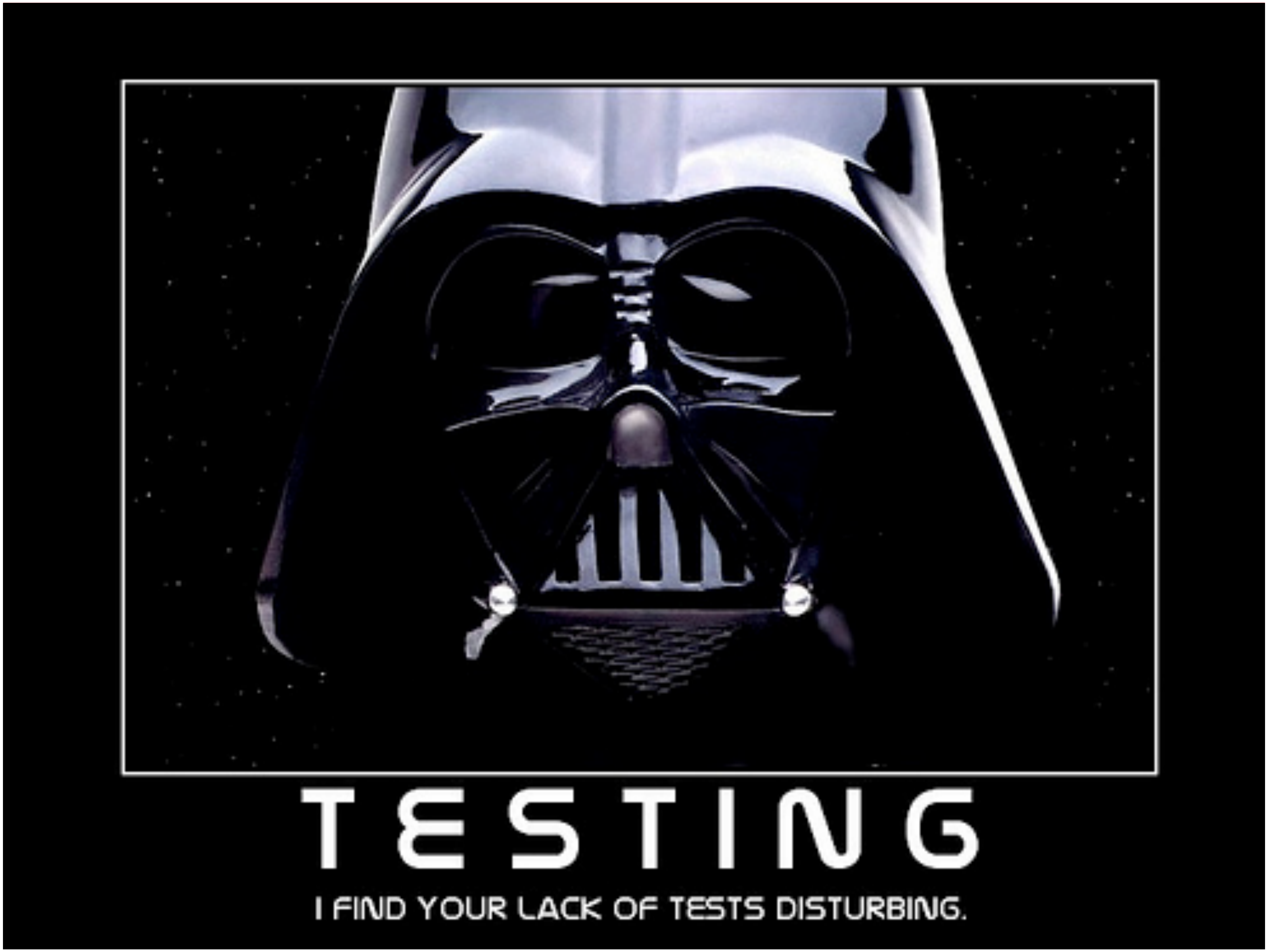
Perficient®

Special Acknowledgement

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



Why do we use Spring?

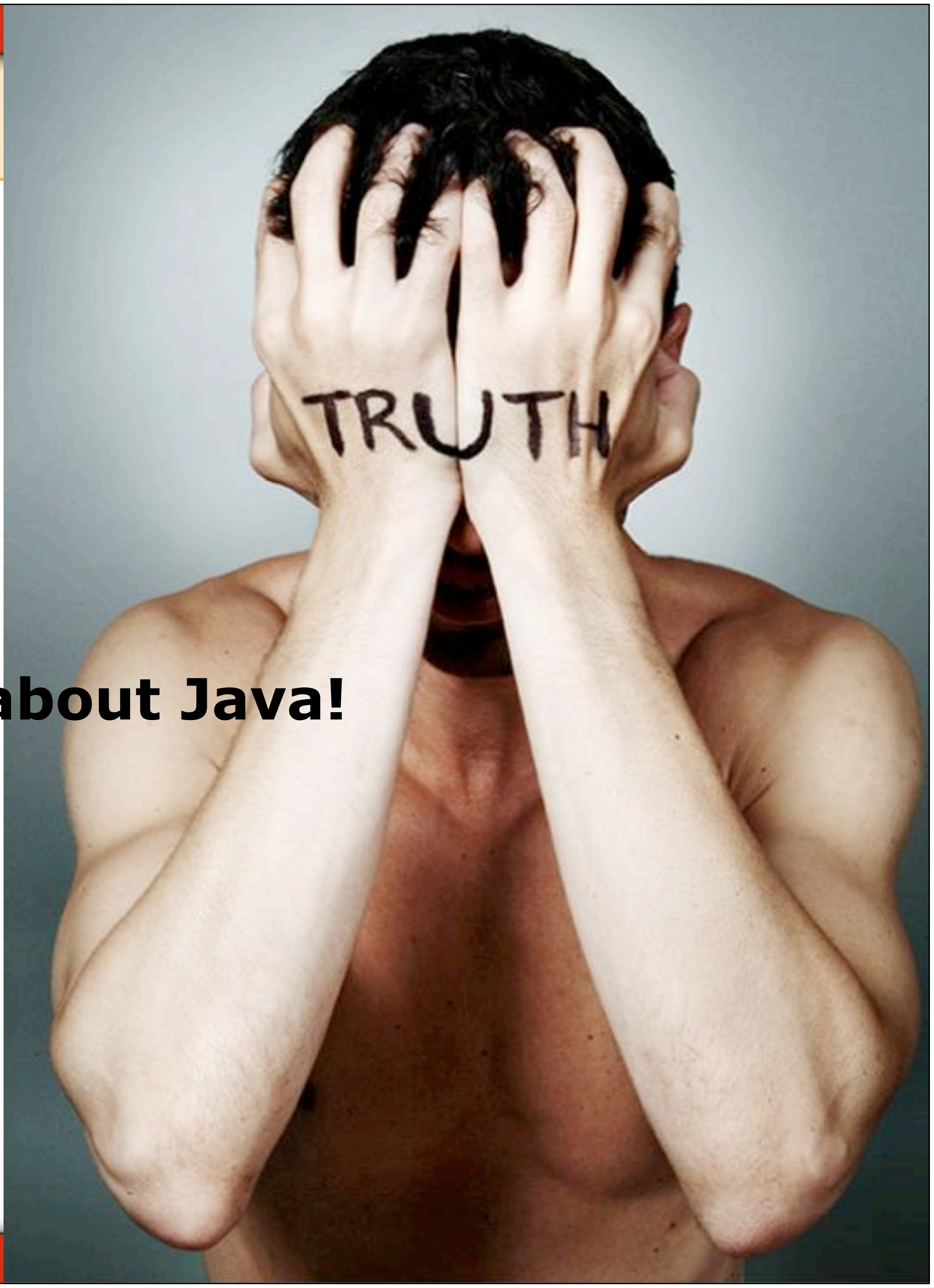

TESTING
I FIND YOUR LACK OF TESTS DISTURBING.

Productivity

easy
STAPLES

Maintainability

Let’s talk about Java!



Let’s talk honestly about Java!

Java (the language) was...

- Great in it's 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.

Solutions

Groovy

Scala

Clojure

What is Groovy

- Dynamic Programming for JVM
- Supports typed and untyped
- Primitives are treated as objects
- List and Hash literals
- Closures
- Operator Overloading

Groovy Case 1

- What is the output of?

```
println( 2.0 - 1.1);
```

Groovy Case #2

- Creating a list in groovy

```
List names = [ “ken”, “craig”, “jay” ]
```

Groovy in the real world - GANT

```
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?

Perficient

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

Spring Framework

LOG4J

HIBERNATE

THE APACHE XML PROJECT

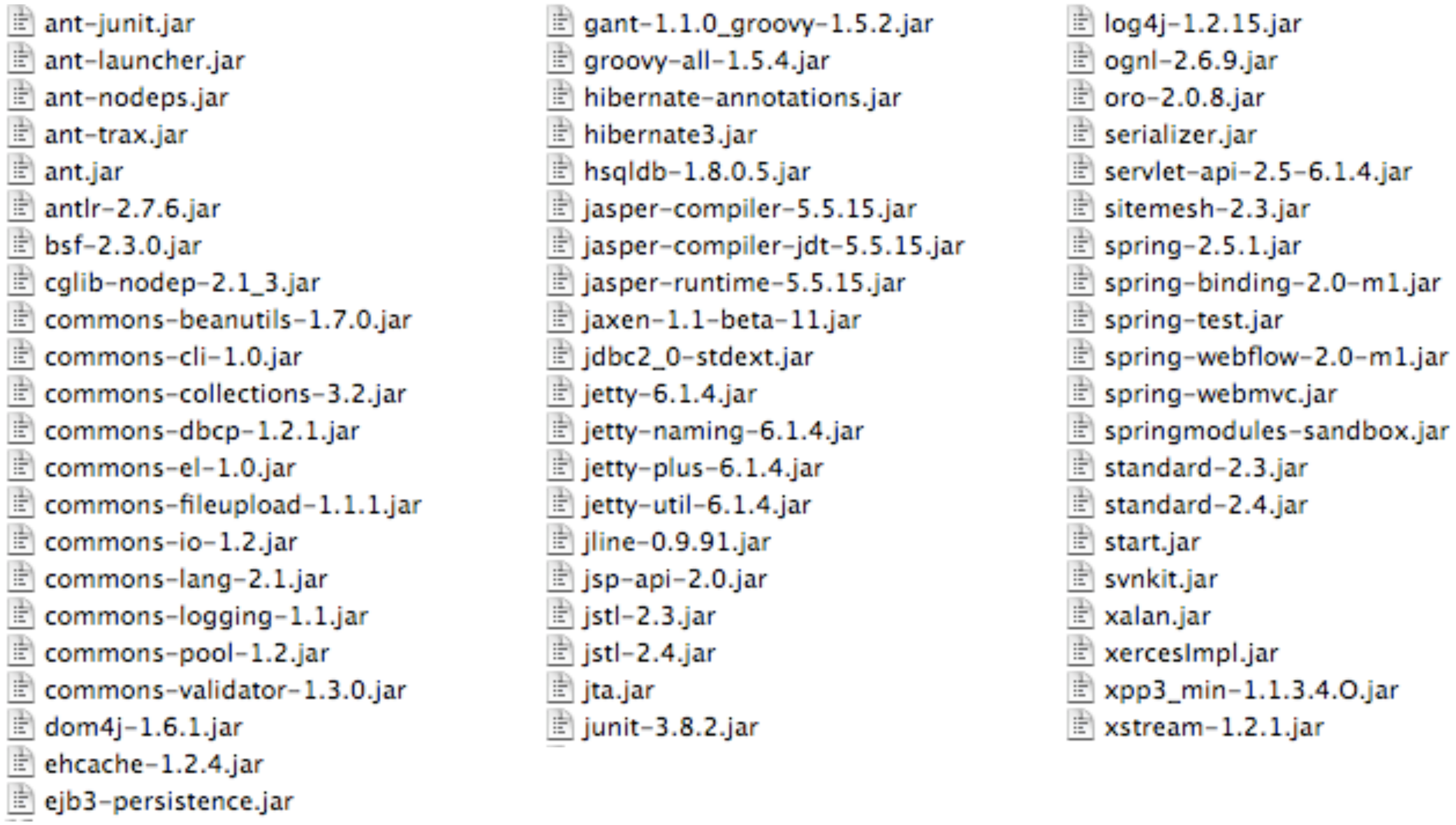
jetty://

HSQL database engine

JUnit.org

THE APACHE ANT PROJECT

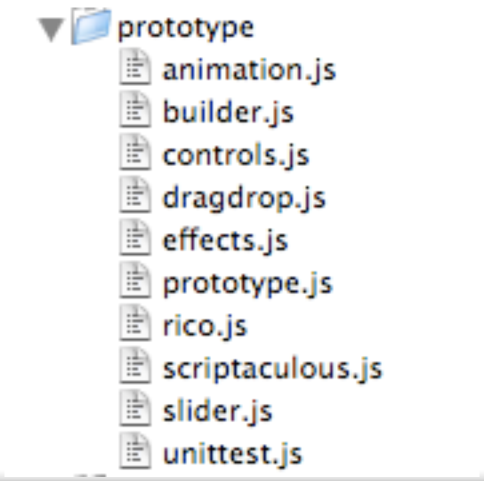
Included JARs



- 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.O.jar
- xstream-1.2.1.jar

Included Ajax Support

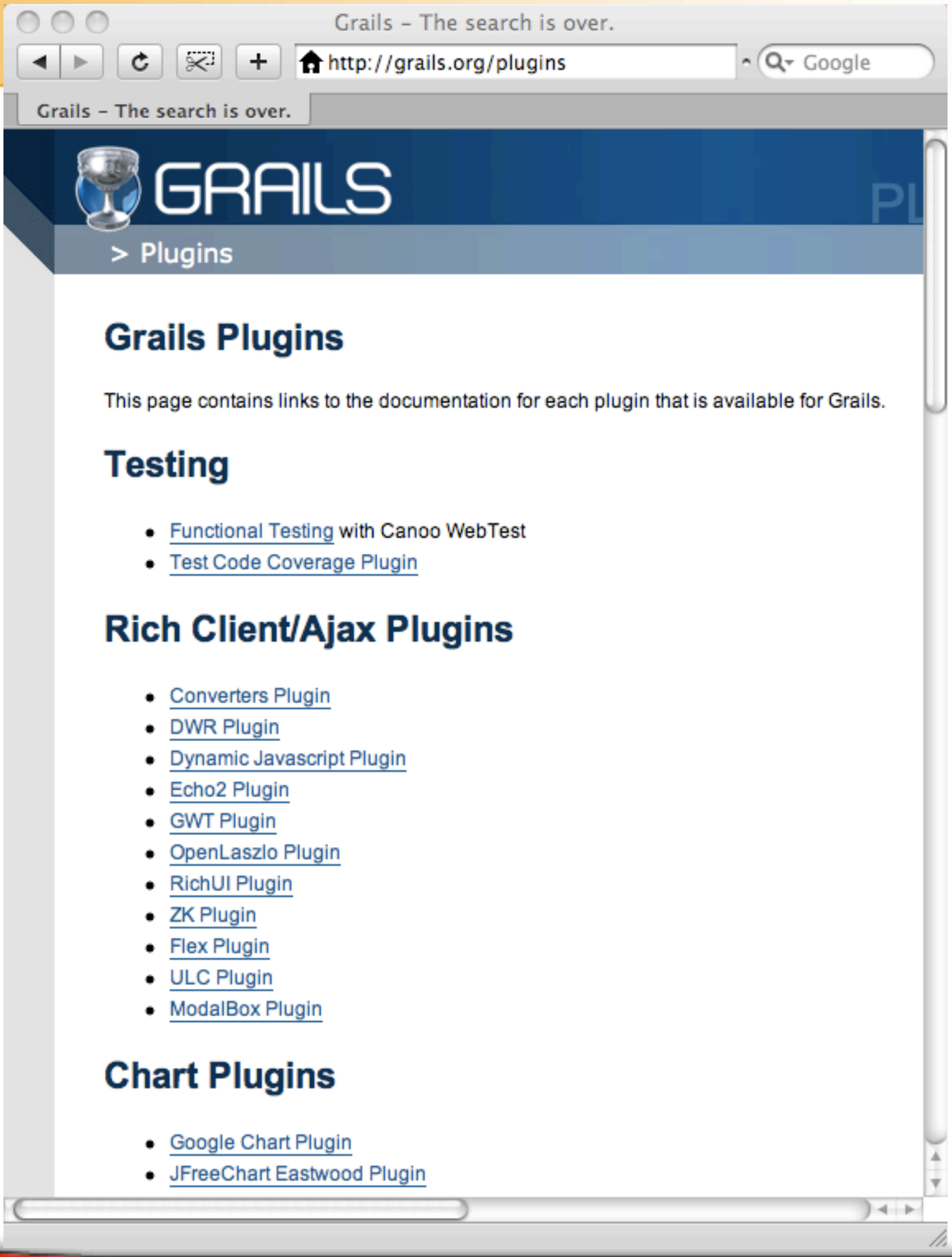
```
/*  Prototype JavaScript framework, version 1.6.0
 *  (c) 2005-2007 Sam Stephenson
 */

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



```
prototype
    animation.js
    builder.js
    controls.js
    dragdrop.js
    effects.js
    prototype.js
    rico.js
    scriptaculous.js
    slider.js
    unittest.js
```

(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 Canoo WebTest
- Test Code Coverage Plugin

Rich Client/Ajax Plugins

- Converters Plugin
- DWR Plugin
- Dynamic Javascript Plugin
- Echo2 Plugin
- GWT Plugin
- OpenLaszlo 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

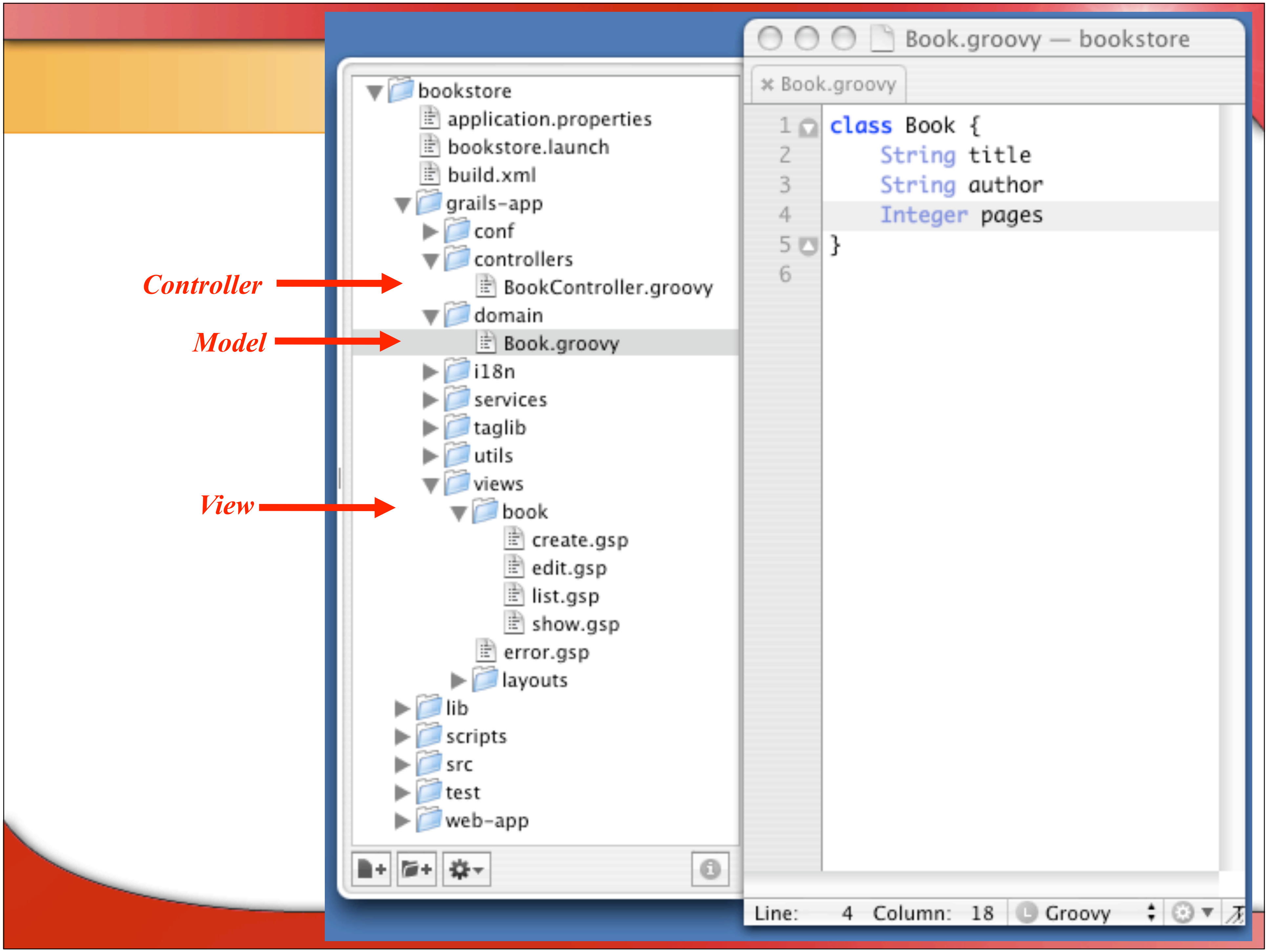
- Download/unzip grails-bin.tar.gz (or zip)
- Create GRAILS_HOME
- Add $GRAILS_HOME/bin to PATH

Your 1-Slide Guide to Grails

Type the following:

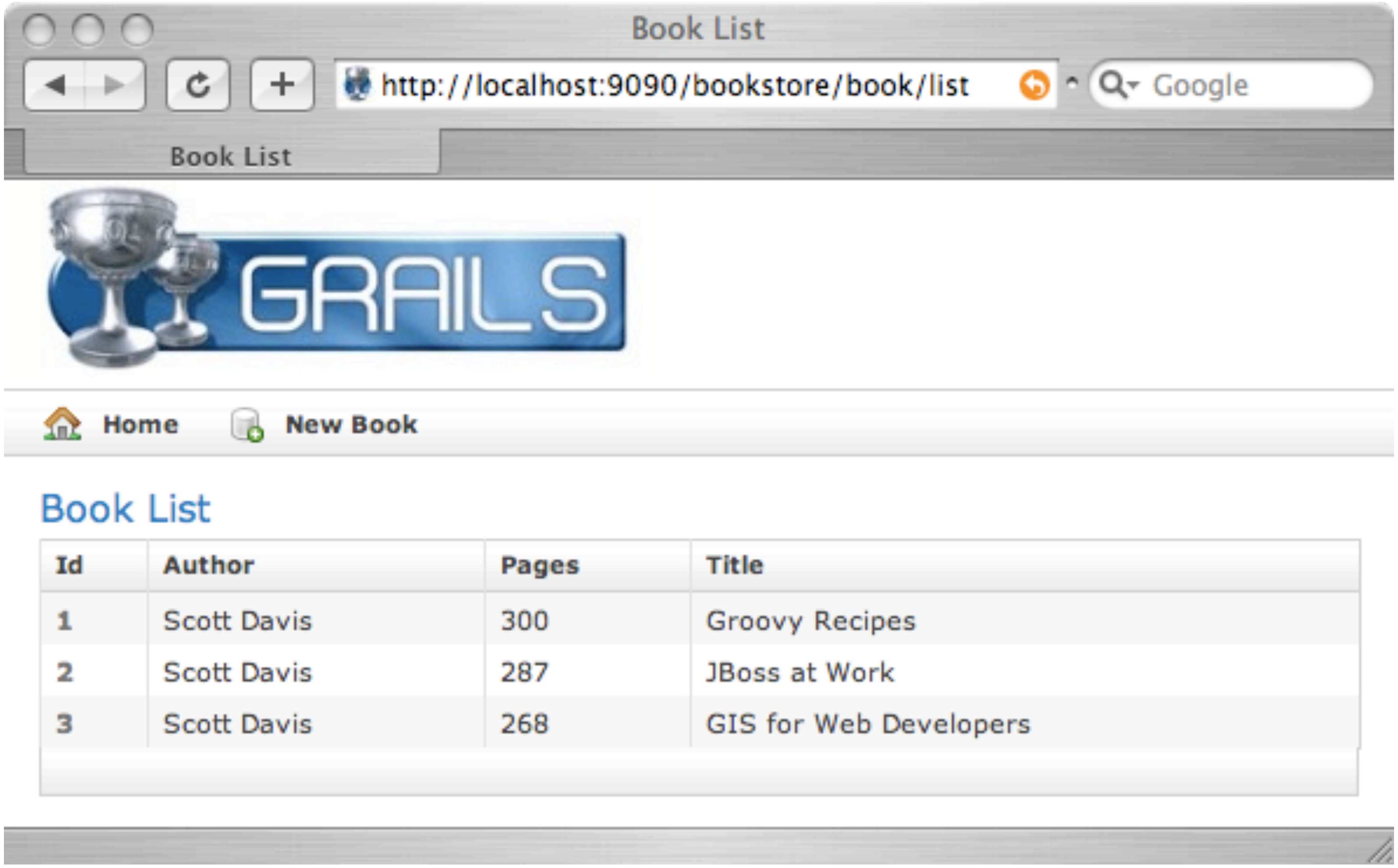
```
$ grails create-app bookstore
$ cd bookstore
$ grails create-domain-class Book
    (Add fields to
     grails-app/domain/Book.groovy)
$ grails generate-all Book
$ grails run-app
```

$ grails help -- shows all available commands



```
class Book {
    String title
    String author
    Integer pages
}
```

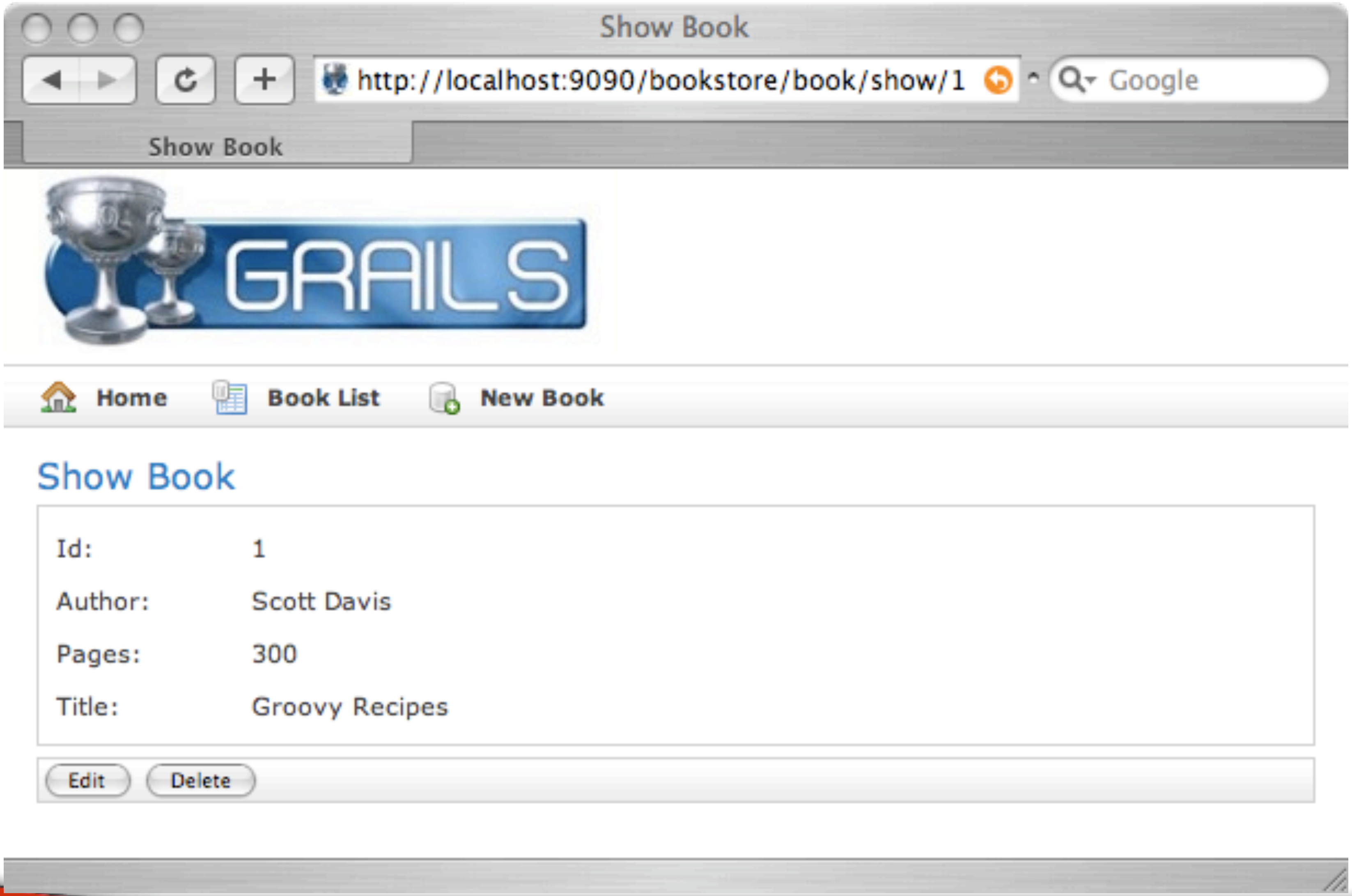
Generated List



Book List

Id	Author	Pages	Title
1	Scott Davis	300	Groovy Recipes
2	Scott Davis	287	JBoss at Work
3	Scott Davis	268	GIS for Web Developers

Generated Show



Show Book

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

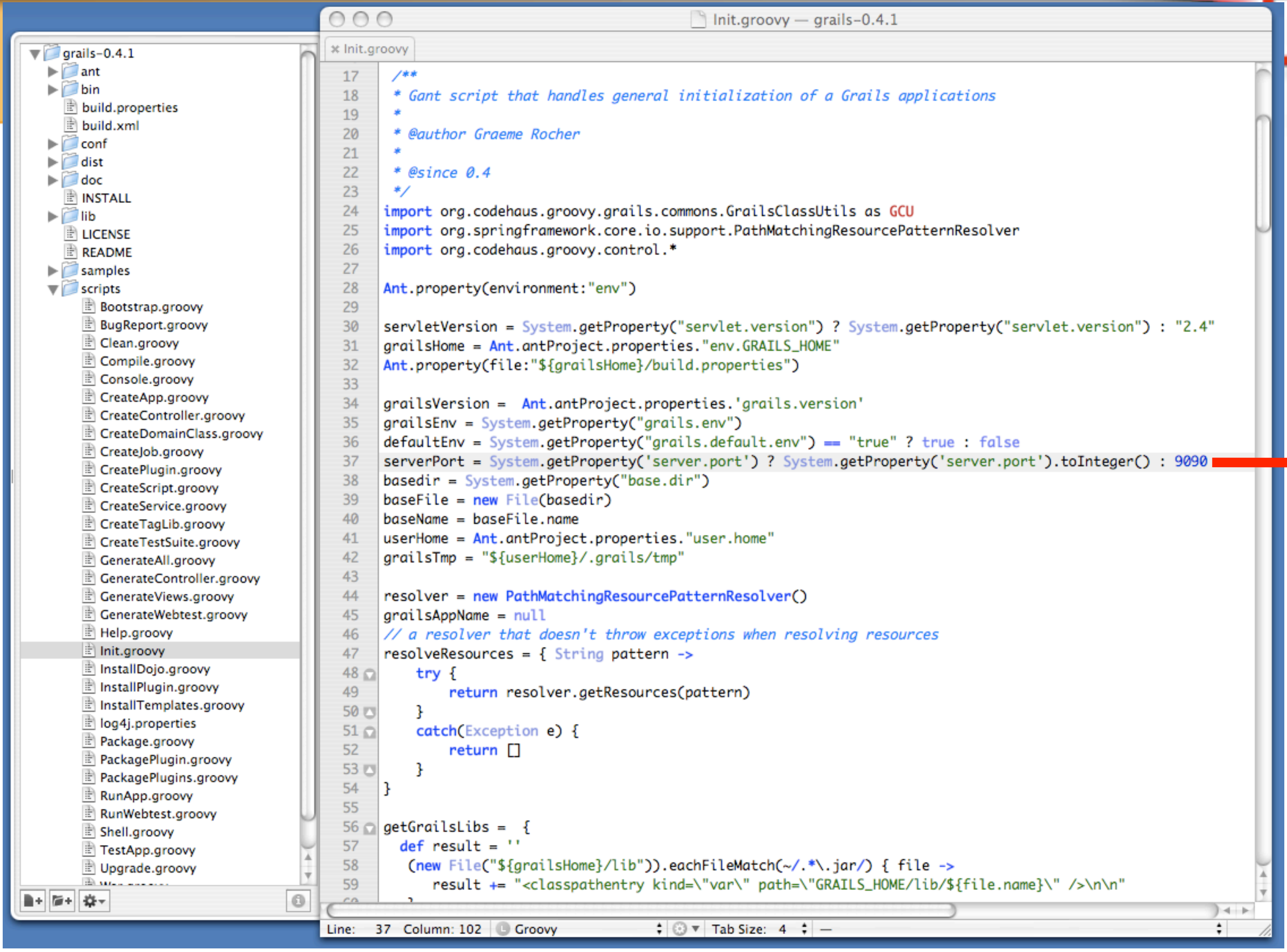
Act 2:
Tweaking the defaults…

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…)



```groovy
 /**
 * Gant script that handles general initialization of a Grails applications
 *
 * @author Graeme Rocher
 *
 * @since 0.4
 */
import org.codehaus.groovy.grails.commons.GrailsClassUtils as GCU
import org.springframework.core.io.support.PathMatchingResourcePatternResolver
import org.codehaus.groovy.control.*

Ant.property(environment:"env")

servletVersion = System.getProperty("servlet.version") ? System.getProperty("servlet.version") : "2.4"
grailsHome = Ant.antProject.properties."env.GRAILS_HOME"
Ant.property(file:"${grailsHome}/build.properties")

grailsVersion =  Ant.antProject.properties.'grails.version'
grailsEnv = System.getProperty("grails.env")
defaultEnv = System.getProperty("grails.default.env") == "true" ? true : false
serverPort = System.getProperty('server.port') ? System.getProperty('server.port').toInteger() : 9090
basedir = System.getProperty("base.dir")
baseFile = new File(basedir)
baseName = baseFile.name
userHome = Ant.antProject.properties."user.home"
grailsTmp = "${userHome}/.grails/tmp"

resolver = new PathMatchingResourcePatternResolver()
grailsAppName = null
// a resolver that doesn't throw exceptions when resolving resources
resolveResources = { String pattern ->
    try {
        return resolver.getResources(pattern)
    }
    catch(Exception e) {
        return []
    }
}

getGrailsLibs =  {
  def result = ''
   (new File("${grailsHome}/lib")).eachFileMatch(~/.*\.jar/) { file ->
      result += "<classpathentry kind=\"var\" path=\"GRAILS_HOME/lib/${file.name}\" />\n\n"
```

```groovy
serverPort = System.getProperty('server.port') ?
                 System.getProperty('server.port').toInteger() :
                 9090
```

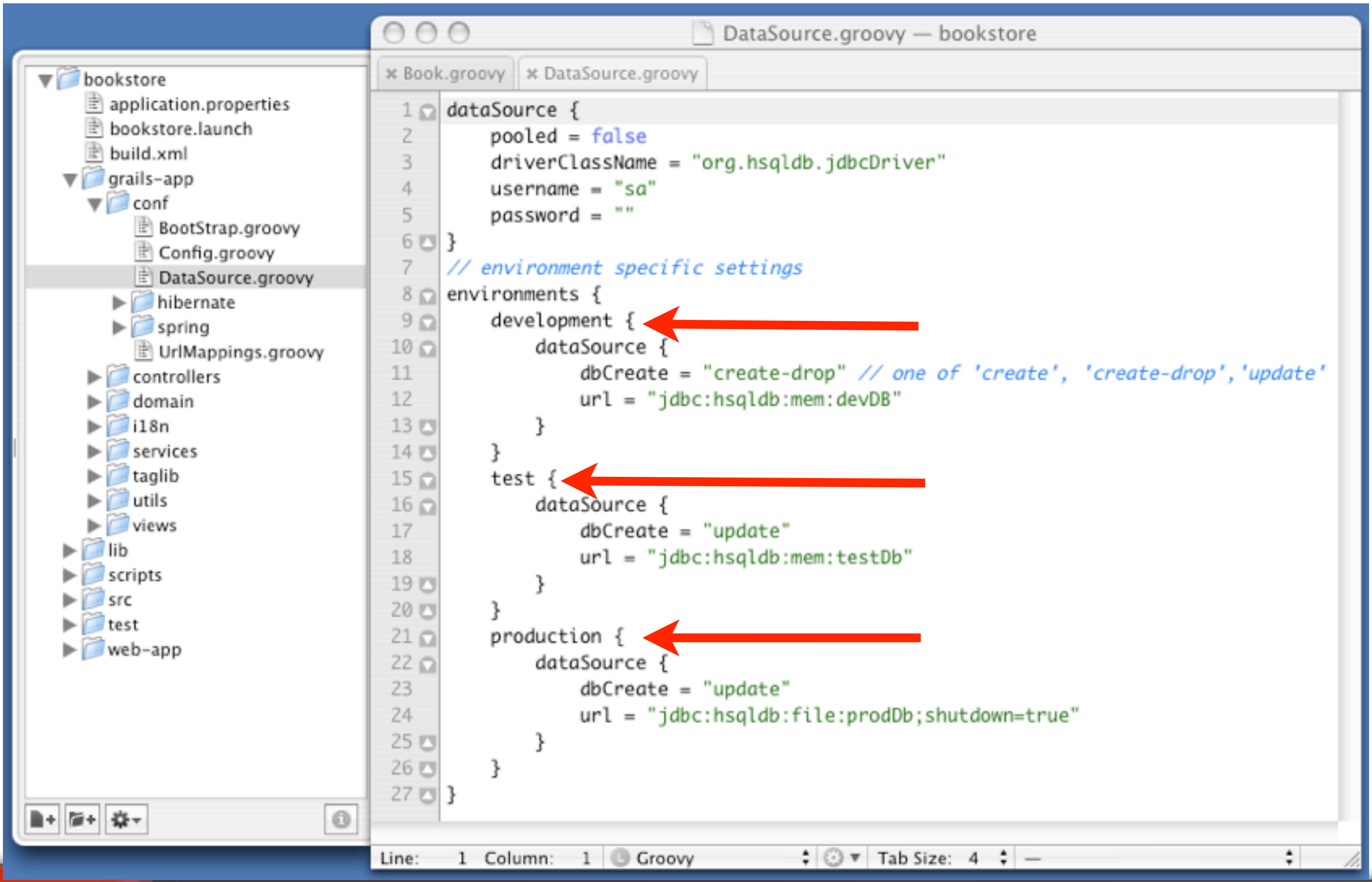
Changing Grails Environments

```
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



```
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

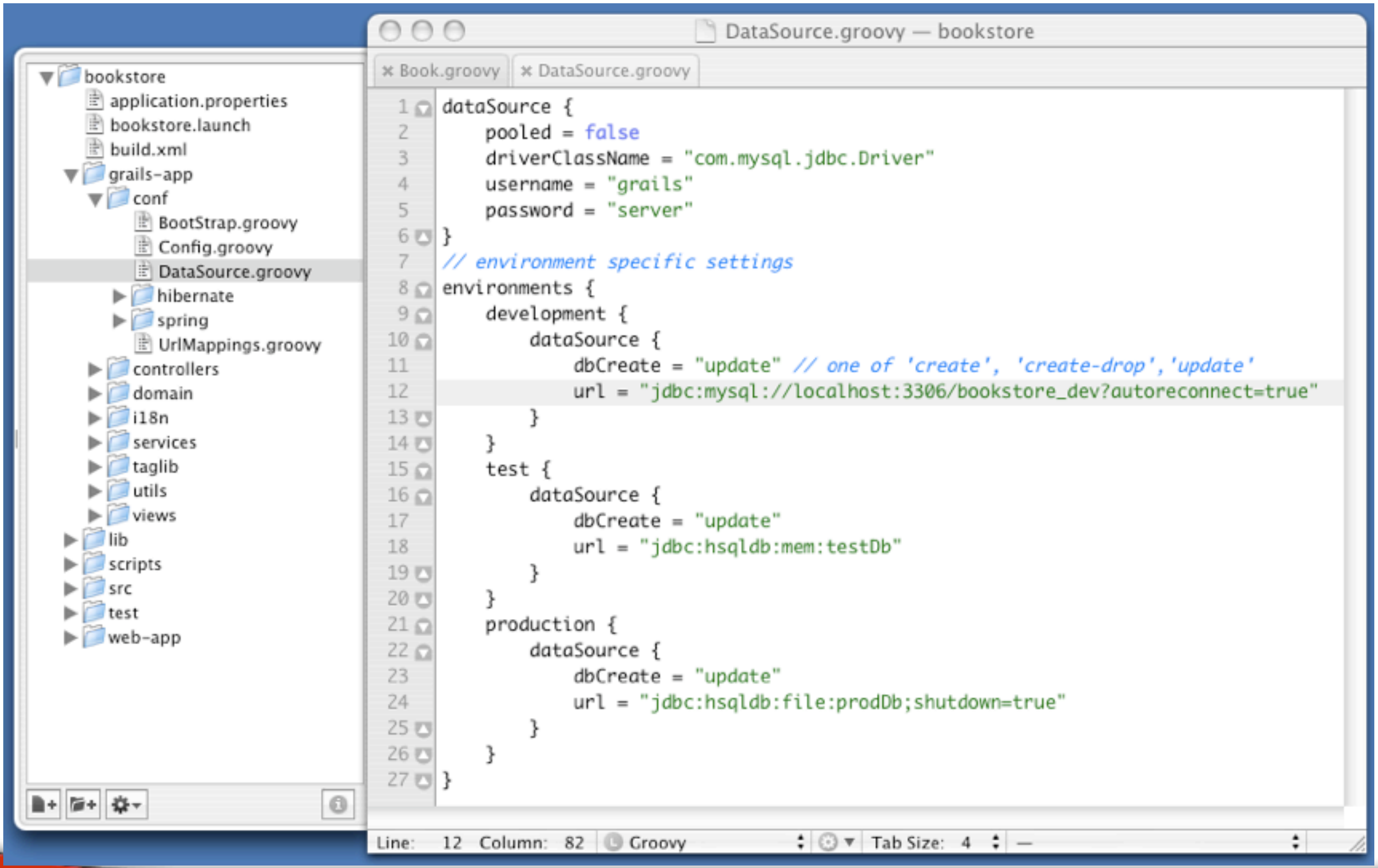
Create the database

```
$ 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



```
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"
        }
    }
}
```

Magic Occurs

```
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/
```

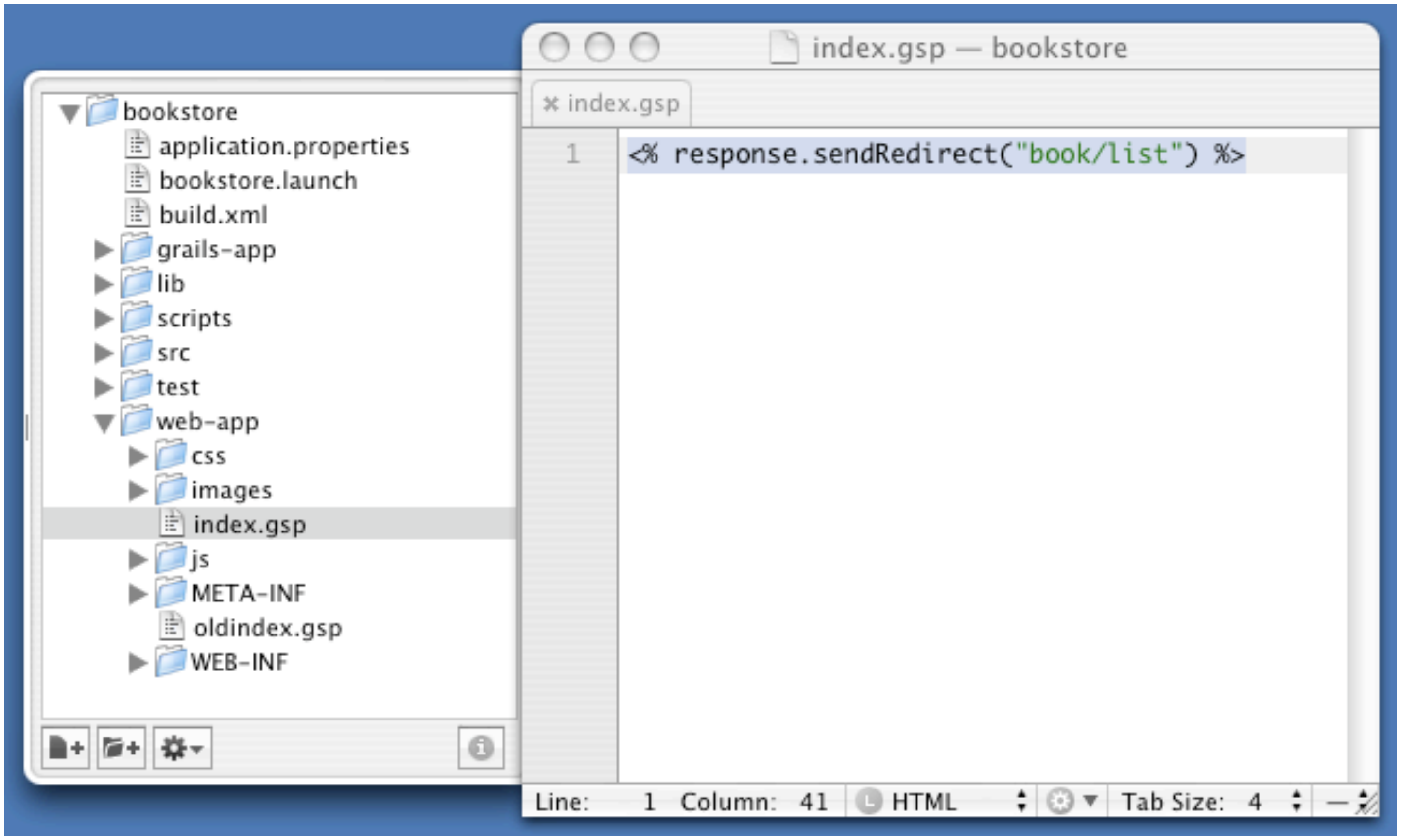
Gotcha: Grails WARs run in PROD by default.

```
$ grails dev war
```

Or run your container with
JAVA_OPTS=-Dgrails.env=development

Changing the Home Page

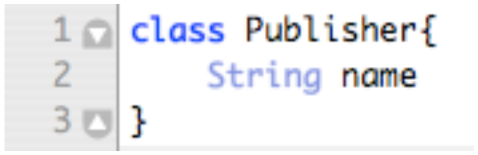
The default homepage is web-app/index.gsp.
You can redirect to any page or controller:



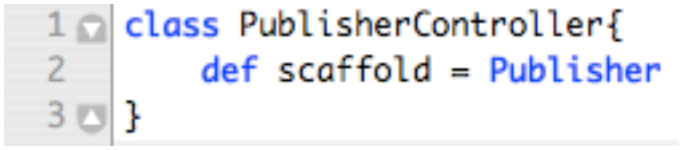
```
<% response.sendRedirect("book/list") %>
```

Act 3:
Understanding Grails Controllers…

Auto-scaffolding



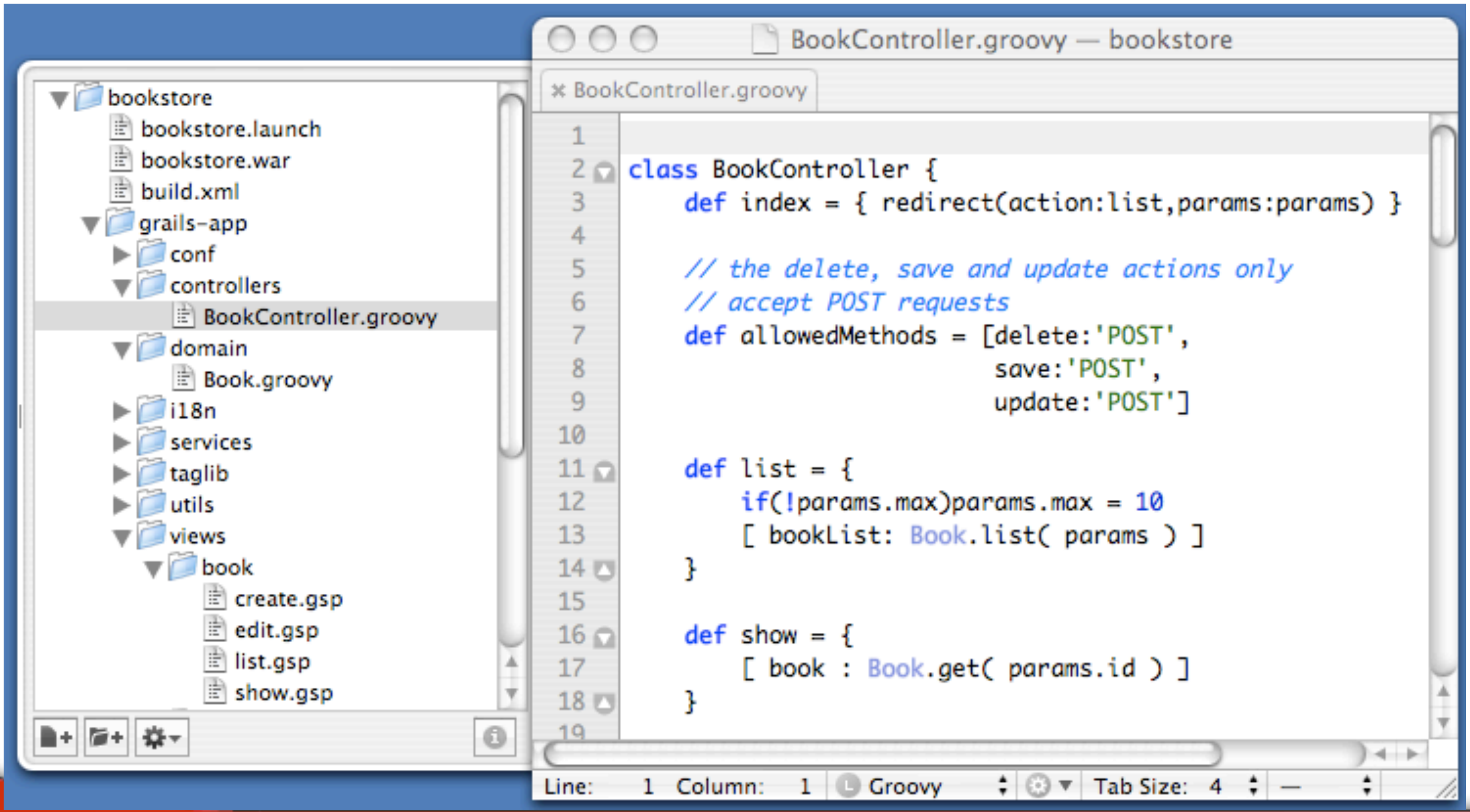
```
class Publisher{
    String name
}
```



```
class PublisherController{
    def scaffold = Publisher
}
```

Generating a Controller

```
$ grails generate-controller
```



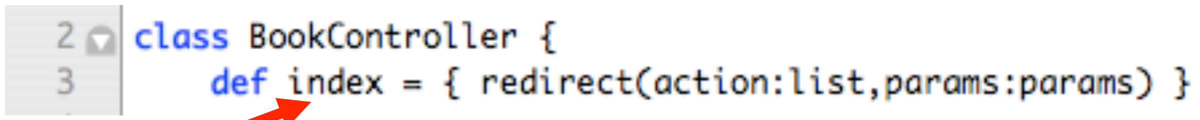
```groovy

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

    // the delete, save and update actions only
    // accept POST requests
    def allowedMethods = [delete:'POST',
                          save:'POST',
                          update:'POST']

    def list = {
        if(!params.max)params.max = 10
        [ bookList: Book.list( params ) ]
    }

    def show = {
        [ book : Book.get( params.id ) ]
    }
```

The Three R's

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

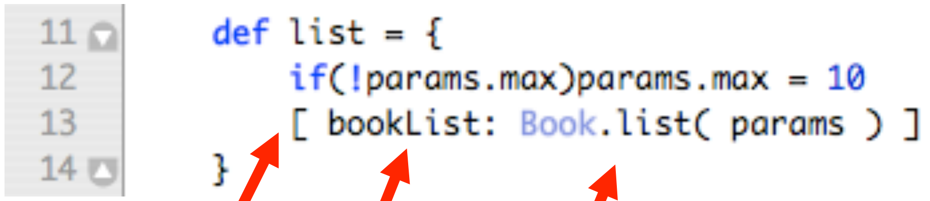

```
class BookController {
    def index = { redirect(action:list,params:params) }
```

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

Controller.list



```
def list = {
    if(!params.max)params.max = 10
    [ bookList: Book.list( params ) ]
}
```

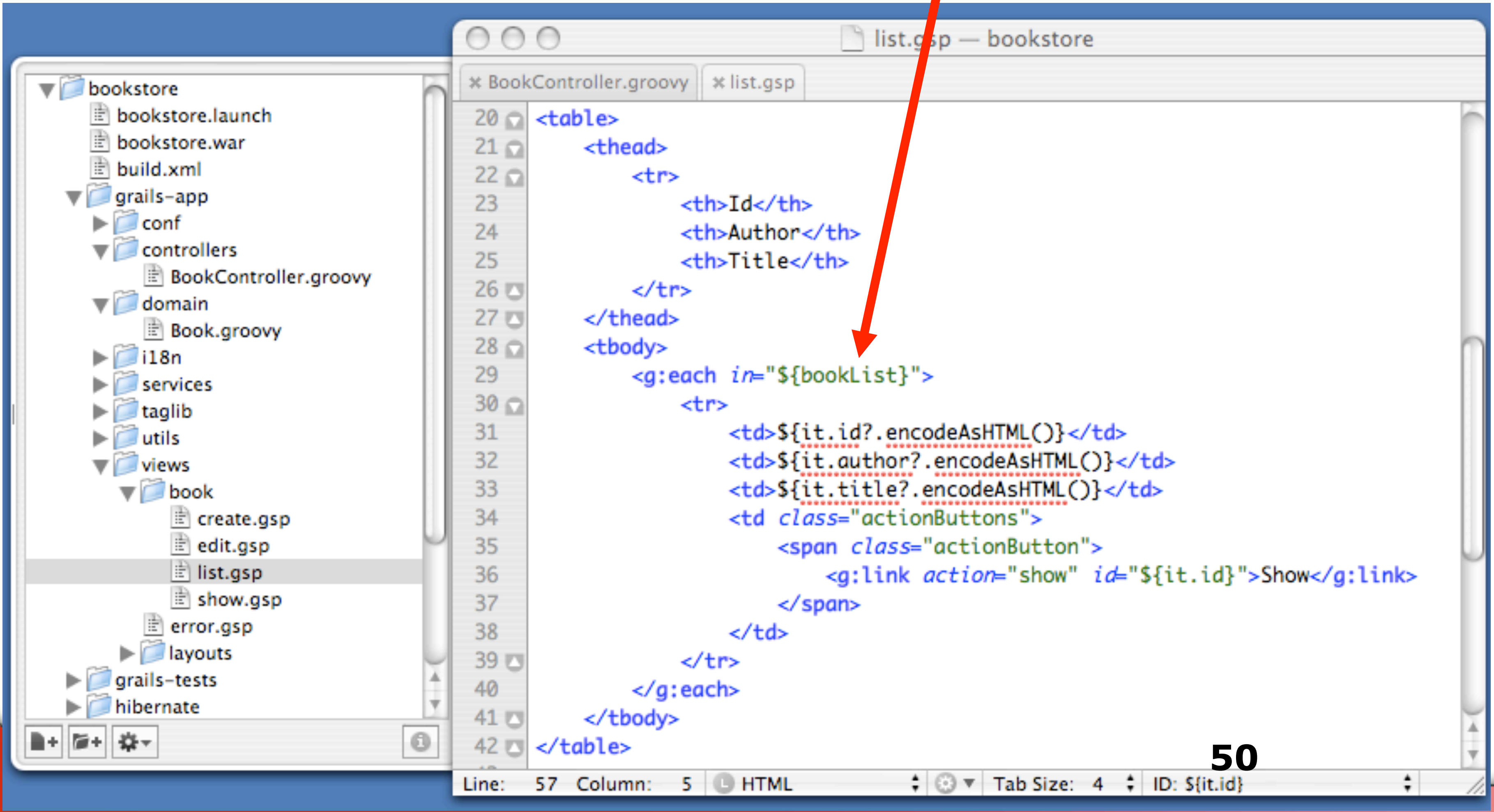
Implicit return statement

GORM
(Grails Object/Relational Mapping)

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

List.gsp

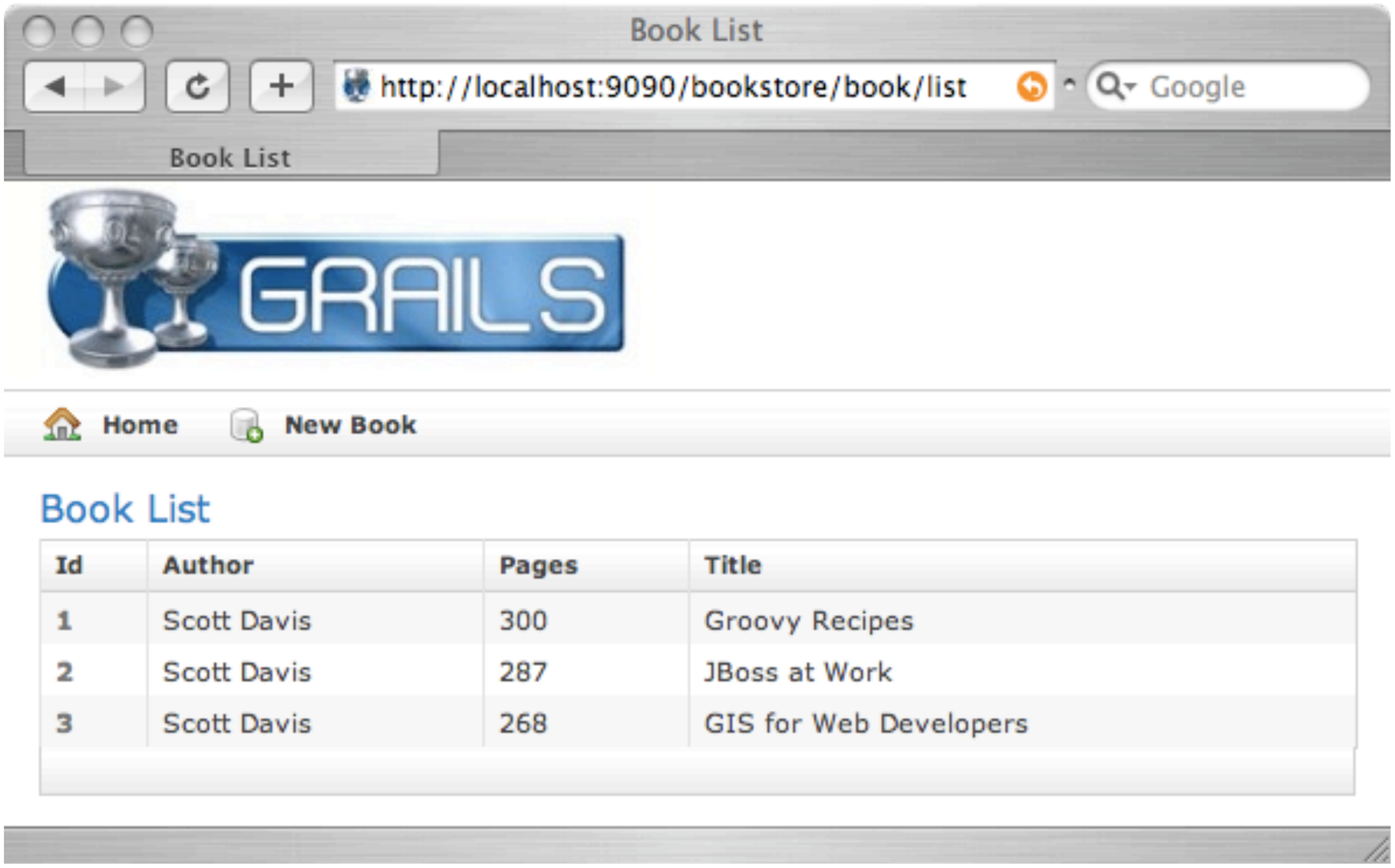
Returned from Controller



```
<table>
    <thead>
        <tr>
            <th>Id</th>
            <th>Author</th>
            <th>Title</th>
        </tr>
    </thead>
    <tbody>
        <g:each in="${bookList}">
            <tr>
                <td>${it.id?.encodeAsHTML()}</td>
                <td>${it.author?.encodeAsHTML()}</td>
                <td>${it.title?.encodeAsHTML()}</td>
                <td class="actionButtons">
                    <span class="actionButton">
                        <g:link action="show" id="${it.id}">Show</g:link>
                    </span>
                </td>
            </tr>
        </g:each>
    </tbody>
</table>
```

Line: 57 Column: 5 HTML Tab Size: 4 ID: ${it.id}

List view



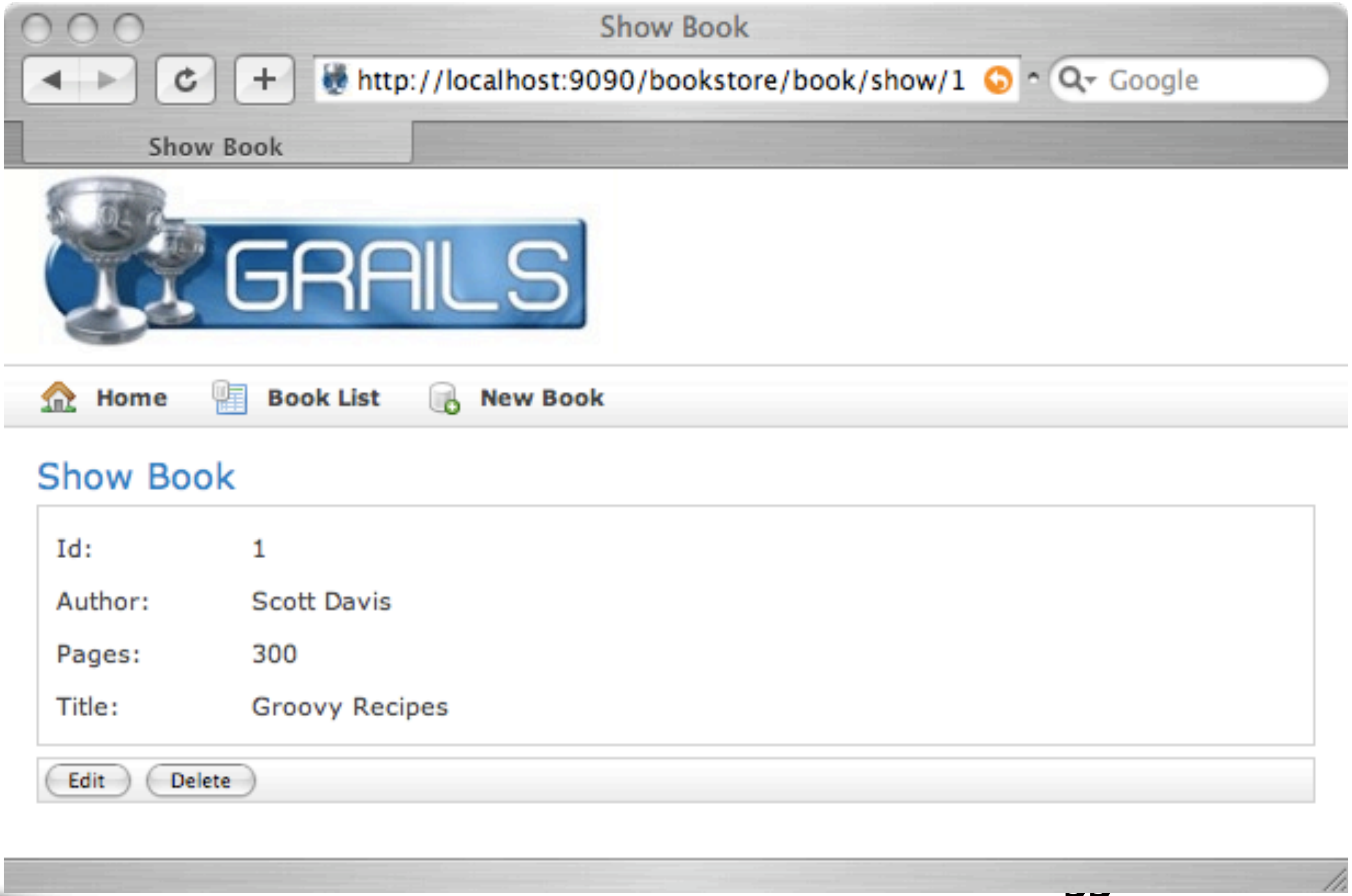
Book List

Id	Author	Pages	Title
1	Scott Davis	300	Groovy Recipes
2	Scott Davis	287	JBoss at Work
3	Scott Davis	268	GIS for Web Developers

Convention over Configuration

- BookController
  - http://localhost:9090/bookstore/book
- BookController.list
  - http://localhost:9090/bookstore/book/list
  - Corresponding list.gsp
- BookController.show(5)
  - http://localhost:9090/bookstore/book/show/5

Show view

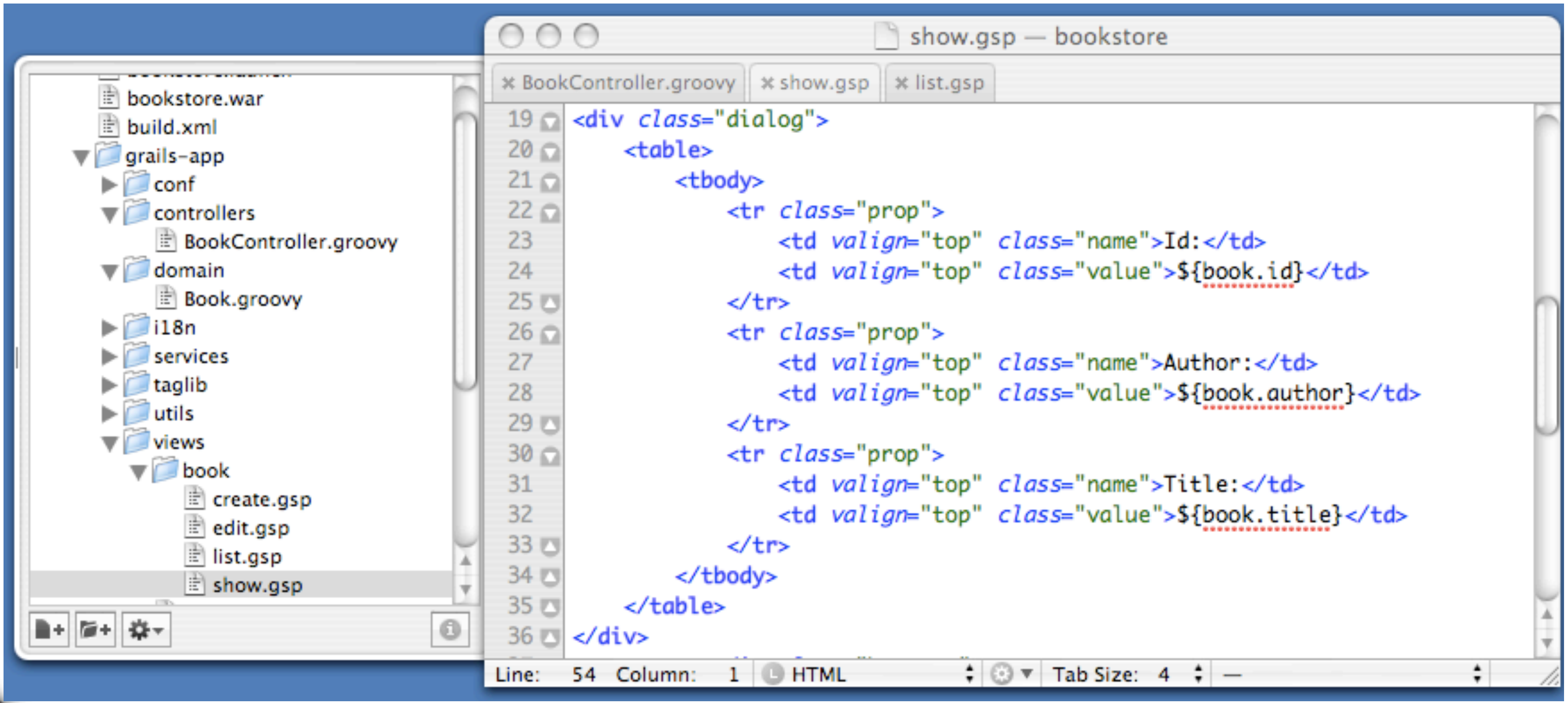


Show Book

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

Controller.show

```
def show = {
    [ book : Book.get( params.id ) ]
}
```

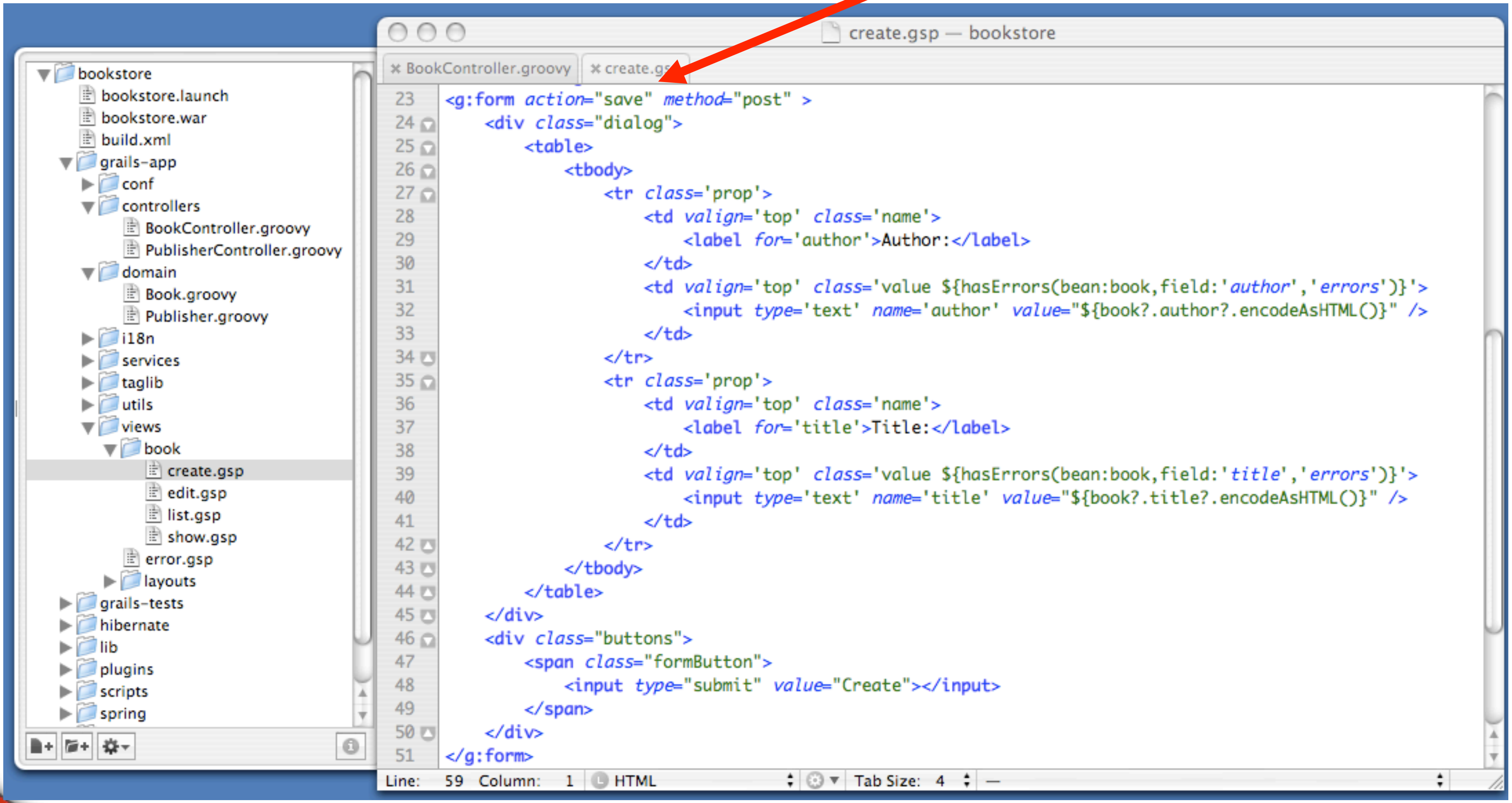


```
<div class="dialog">
    <table>
        <tbody>
            <tr class="prop">
                <td valign="top" class="name">Id:</td>
                <td valign="top" class="value">${book.id}</td>
            </tr>
            <tr class="prop">
                <td valign="top" class="name">Author: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

Line: 54 Column: 1 HTML Tab Size: 4

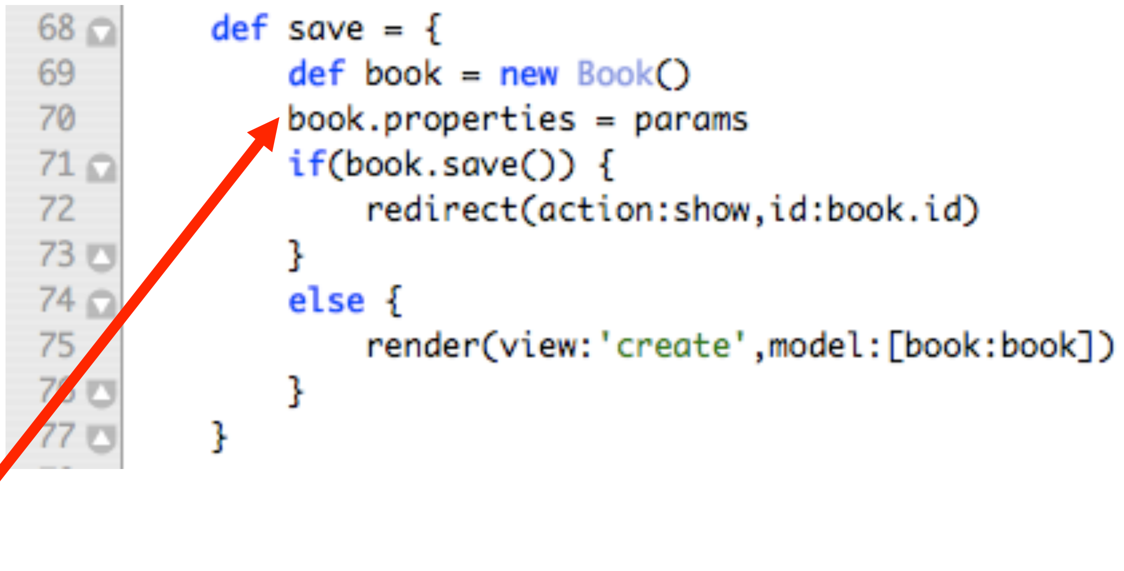
Create.gsp

Controller Method



```
<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>
                    <td valign='top' class='value ${hasErrors(bean:book,field:'author','errors')}'>
                        <input type='text' name='author' value="${book?.author?.encodeAsHTML()}" />
                    </td>
                </tr>
                <tr class='prop'>
                    <td valign='top' class='name'>
                        <label for='title'>Title:</label>
                    </td>
                    <td valign='top' class='value ${hasErrors(bean:book,field:'title','errors')}'>
                        <input type='text' name='title' value="${book?.title?.encodeAsHTML()}" />
                    </td>
                </tr>
            </tbody>
        </table>
    </div>
    <div class="buttons">
        <span class="formButton">
            <input type="submit" value="Create"></input>
        </span>
    </div>
</g:form>
```

Controller.save



```
def save = {
    def book = new Book()
    book.properties = params
    if(book.save()) {
        redirect(action:show,id:book.id)
    }
    else {
        render(view:'create',model:[book:book])
    }
}
```

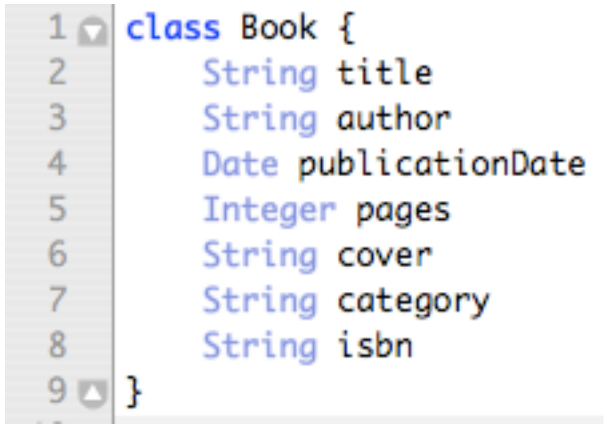
In one line, Param name/value pairs from the form are saved to a POGO (Plain Old Groovy Object).

In the next line, the POGO is saved to the database via GORM.

Act 4:
Understanding Grails Models…
…and Views…
…and GORM…

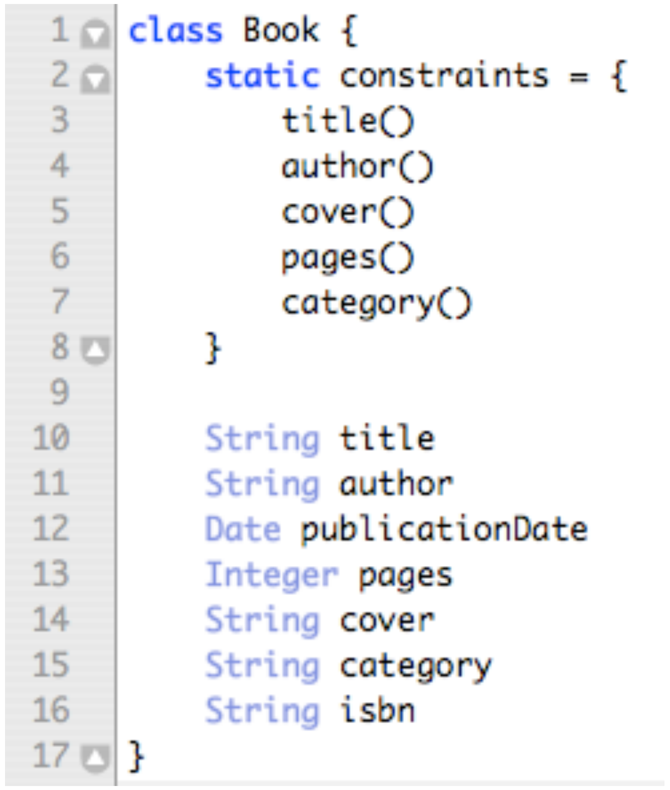
POGOs

- Plain Old Groovy Objects
  - Fields are automatically private
  - Getters and setters are automatically provided
  - Use Wrappers instead of Primitives
    - Integer, Float, Double, Boolean



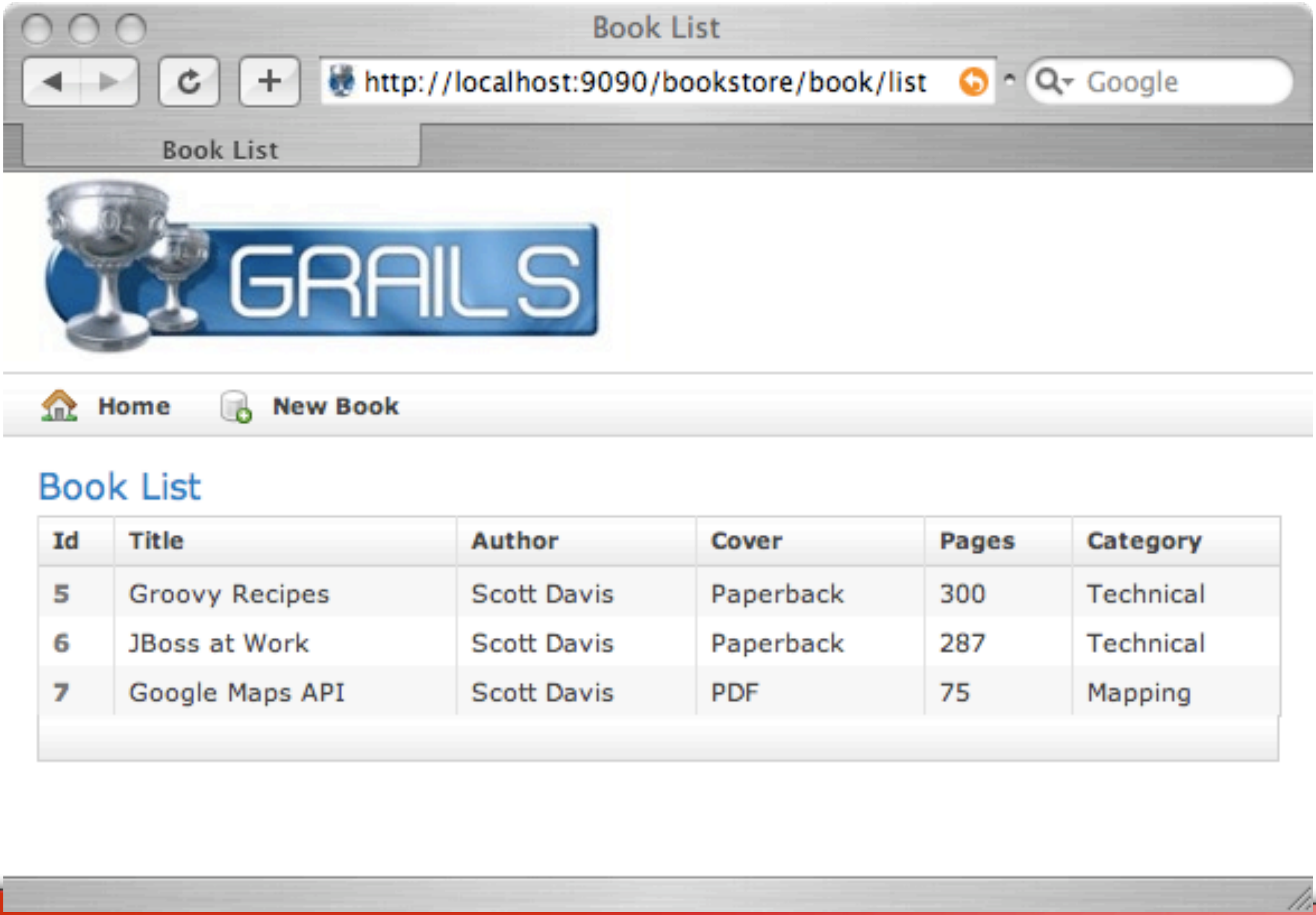
```
class Book {
    String title
    String author
    Date publicationDate
    Integer pages
    String cover
    String category
    String isbn
}
```

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
}
```

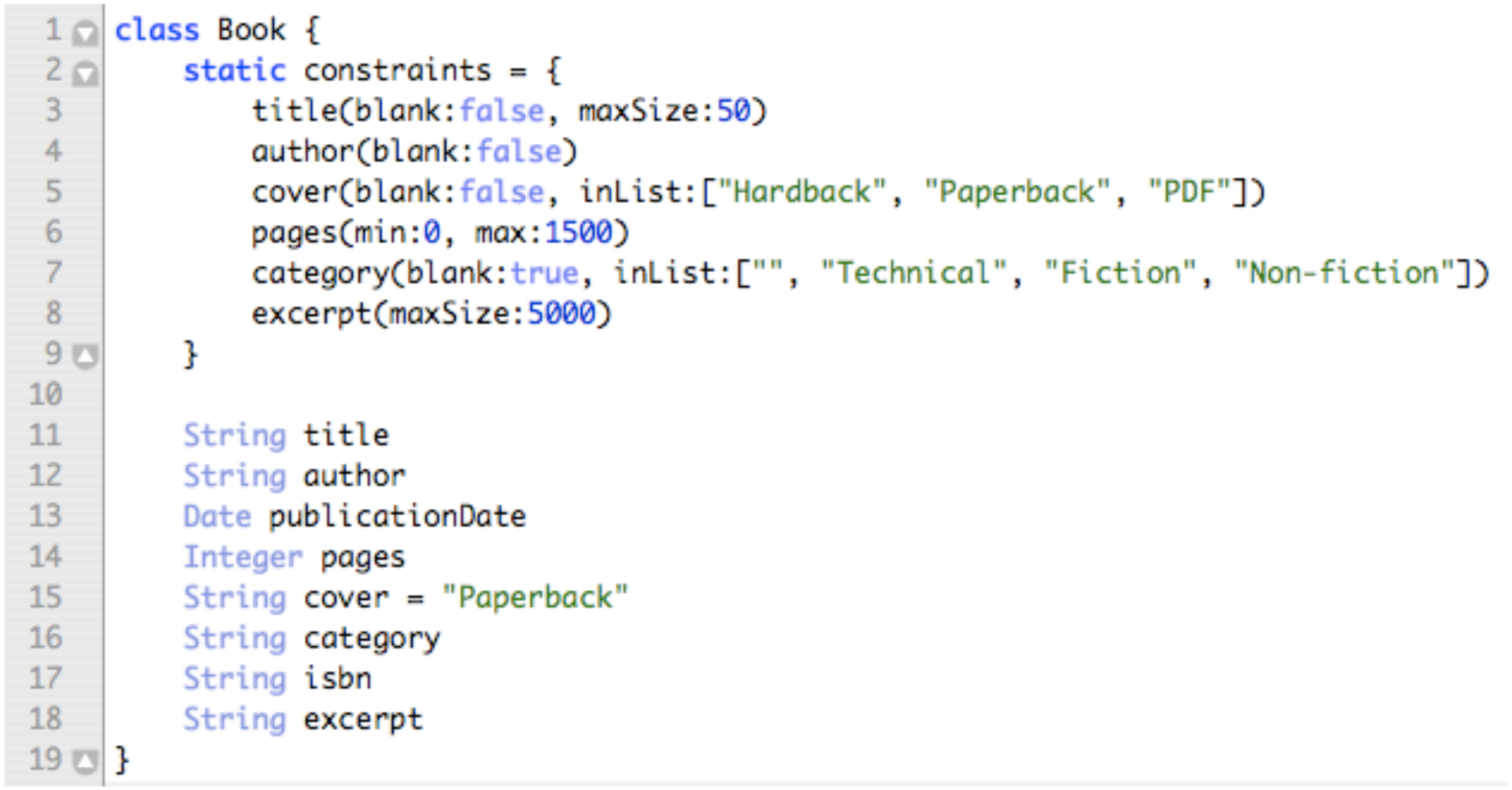
Ordered Fields in List



Book List

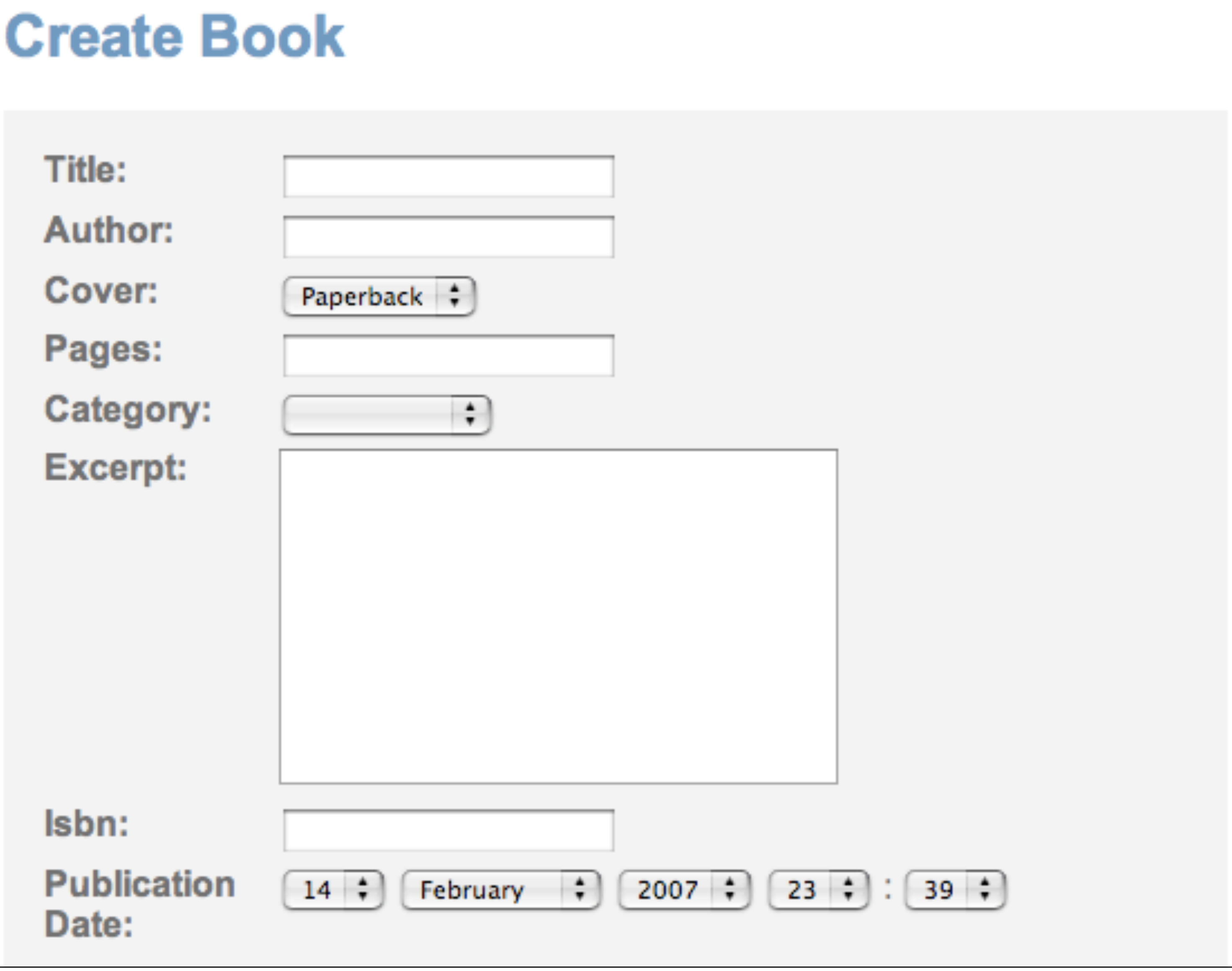
Id	Title	Author	Cover	Pages	Category
5	Groovy Recipes	Scott Davis	Paperback	300	Technical
6	JBoss at Work	Scott Davis	Paperback	287	Technical
7	Google Maps API	Scott Davis	PDF	75	Mapping

Field Validation



```
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
}
```

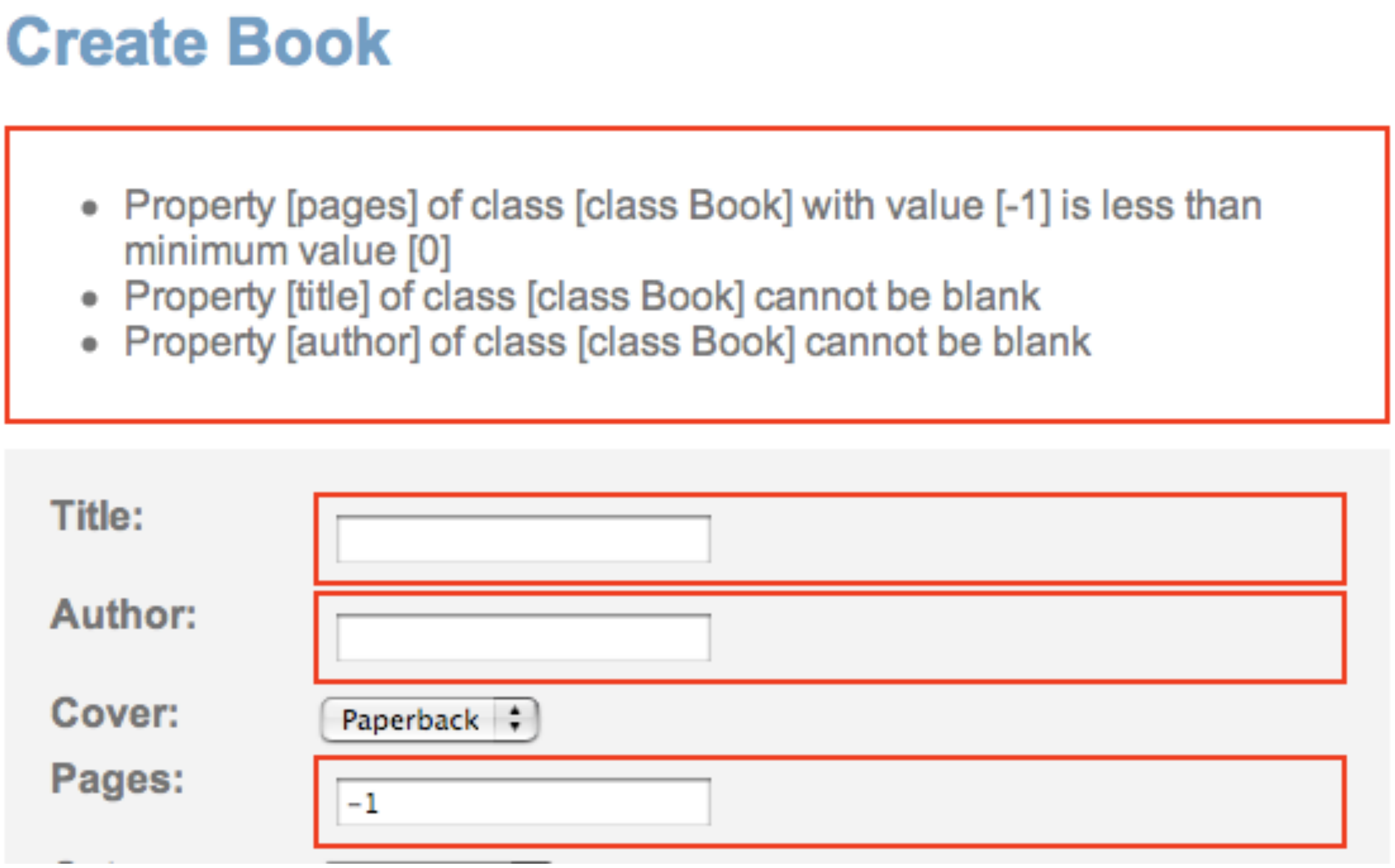
Create Form



Create Book

Title:	
Author:	
Cover:	Paperback
Pages:	
Category:	
Excerpt:	
Isbn:	
Publication Date:	14 February 2007 23 : 39

Validation



Create Book

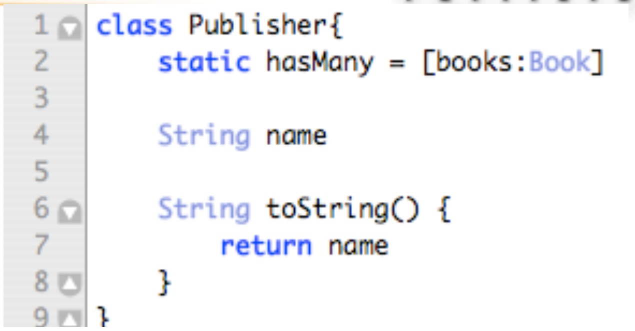
- Property [pages] of class [class Book] with value [-1] is less than minimum value [0]
- Property [title] of class [class Book] cannot be blank
- Property [author] of class [class Book] cannot be blank

Title:	
Author:	
Cover:	Paperback
Pages:	-1

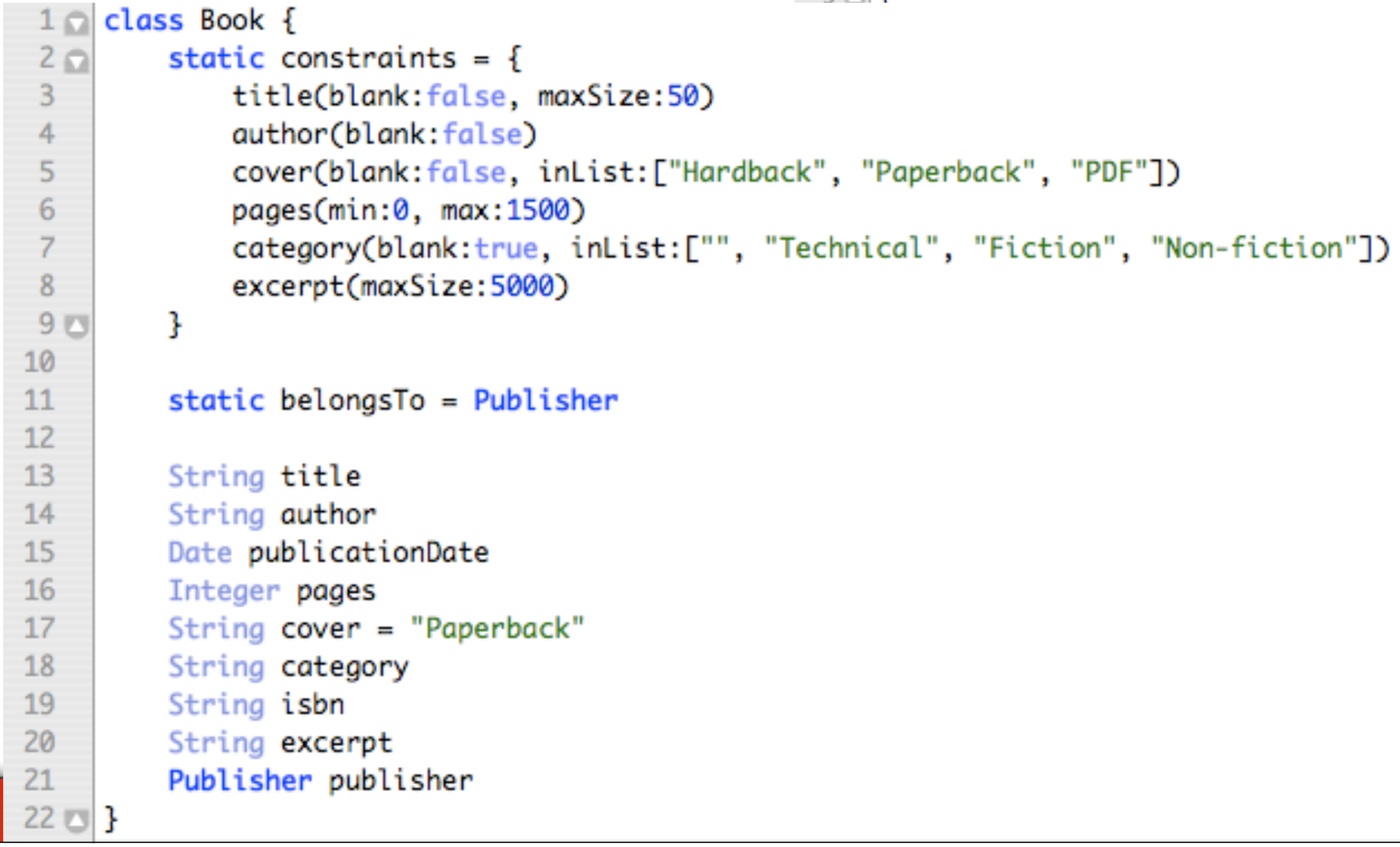
Schema

```
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

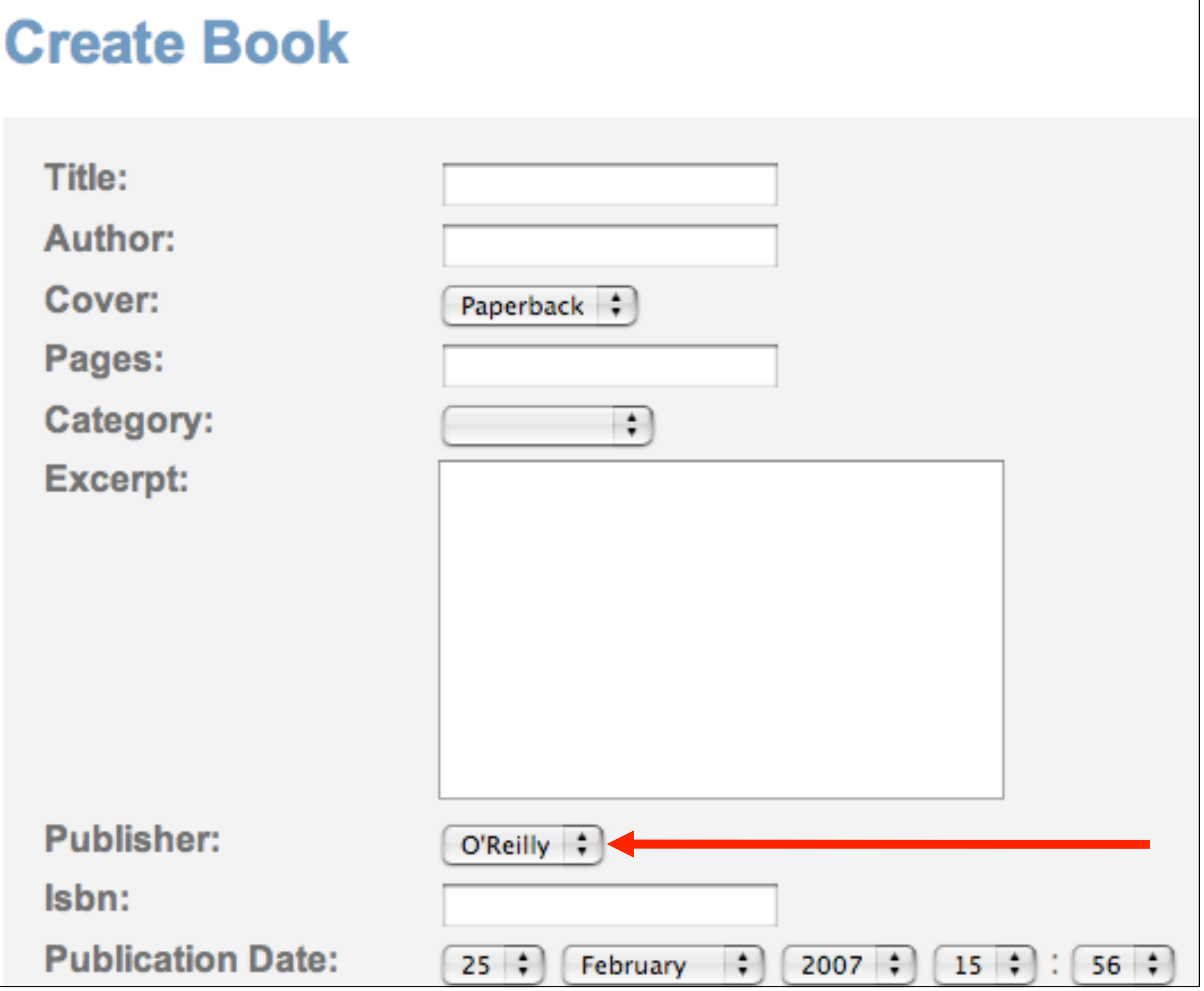


```
class Publisher{
    static hasMany = [books:Book]

    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
}
```

One-to-Many



Create Book

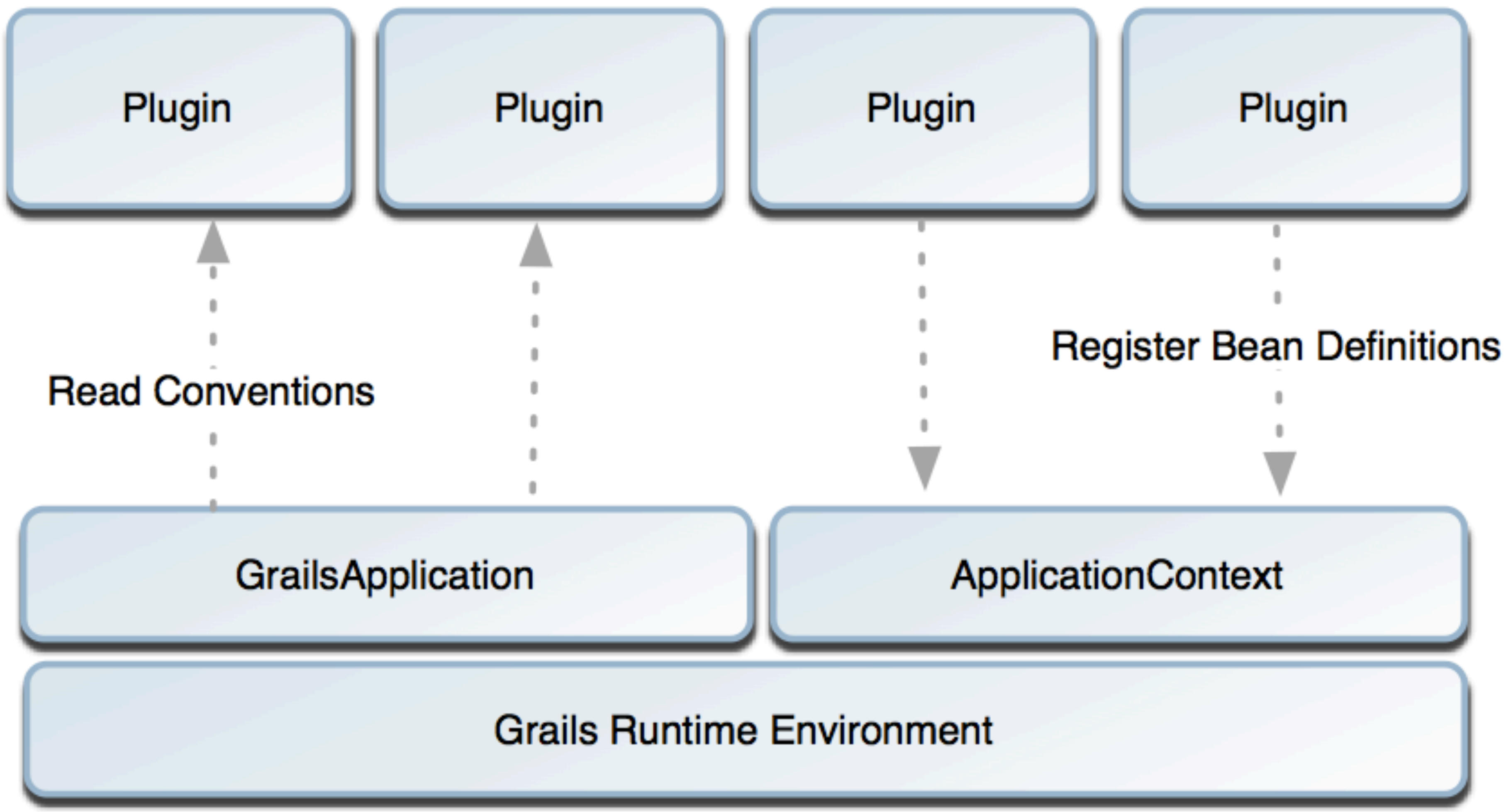
Title:	
Author:	
Cover:	Paperback
Pages:	
Category:	
Excerpt:	
Publisher:	O'Reilly
Isbn:	
Publication Date:	25 February 2007 15 : 56

Act 5:
Understanding Grails Plugin Architecture…

Grails Extension Points

- Spring application context
- Dynamic method registration
- Auto Reloading

Plug-in Architecture



Plug-in Overview

- 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

```
// 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

Example Plug-in with Spring

```
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:

Spring Framework

LOG4J

HIBERNATE

THE APACHE XML PROJECT

jetty://

HSQL database engine

JUnit.org

THE APACHE ANT PROJECT

Summary

- Groovy
- Grails
- Productivity knows no bounds!

Questions

- **Please Fill Out Surveys**

kensipe@gmail.com

twitter: @kensipe

kensipe.blogspot.com