Groovy & Grails in Action!

Guillaume Laforge
Groovy Project Manager

Head of Groovy Development

www.devoxx.com
Discover the Groovy dynamic language for the JVM
Understand how you can create DSLs in Groovy
Getting productive with the Grails agile web stack
Guillaume Laforge

- Groovy Project Manager
- JSR-241 Spec Lead
- Head of Groovy Development at SpringSource
- Initiator of the Grails framework
- Co-author of Groovy in Action
- Speaker at numerous conferences
  - JavaOne, QCon, JavaZone, Sun TechDays, JavaPolis, The Spring Experience, JAX, Dynamic Language World...
Back in 2007, Graeme Rocher and Guillaume Laforge created **G2One**, the Groovy/Grails company

**Professional services**

**consulting, expertise, training, support**

Last November, **SpringSource acquired G2One**
Groovy in Action

A bit of history and background
Groovy syntax basics
Interesting Groovy APIs
Domain-Specific Languages
What’s new in Groovy 1.6?
The Groovy Ecosystem
James Strachan to Bob McWhirter in 2003:

Wouldn’t it be “groovy” if we could have native syntax for lists, maps and regexs in Java like in most scripting languages?

Wouldn’t it be “groovy” if we could have duck typing in Java?

Wouldn’t it be “groovy” if we had closures and properties in Java?
Groovy agenda

- A bit of history and background
- Syntax basics
- Interesting Groovy APIs
- Domain-Specific Languages
- What’s new in Groovy 1.6?
- The Groovy ecosystem
Groovy is...

- Groovy is a **dynamic language** for the JVM
- Groovy directly compiles down to bytecode
- Inspired by other languages: Python, Ruby, Smalltalk
- Whose goal is to **simplify the life of developers**
- Apache 2 licensed Open Source project
  - hosted at Codehaus
- Grammar directly deriving from the Java 5 language
  - support for annotations, generics, enums, static imports
Groovy is everywhere

- Integrated in Open Source projects
  - Spring, JBoss SEAM, ServiceMix, Mule, Oracle OC4J

- Integrated in mission-critical applications
  - Mutual of Omaha, US Fortune 500 insurance company
  - US National Cancer Institute

- Integrated in commercial products
  - IBM WebSphere Smash, SAP Composition on Grails
  - Oracle Data Integrator, Oracle Fusion (ADF)
Lots of books to read
Features at a glance

- Totally object-oriented
- Optional typing
- No need to wait for Java 7 / 8 / 9:
  - Closures: reusable / assignable code blocks
  - Properties: auto-generated getters / setters
  - Multimethods: call the right method for the right type
  - BigDecimal arithmetics
  - Useful wrapper APIs: SQL, XML, Swing, templates, JMX...
Java integration

- Groovy generates Java bytecode for the JVM
- Same strings, same regex
- Same APIs — JDK, collections, 3rd party, etc.
- Same security model, same threading model
- Same OO concepts
- Joint-compilation

```
```

```
```

```
```
Let’s get started

- Go to http://groovy.codehaus.org/Download
- Download the latest version
  - Groovy 1.5.7 — maintenance branch
  - Groovy 1.6-beta-2 — soon to be released
- Unzip the archive, set GROOVY_HOME
- Add $GROOVY_HOME/bin to your path

You’re ready!
Tools at your disposal

- Shell and console: groovysh and groovyConsole
- An Ant task, a Maven plugin (GMaven)
- A joint compiler
- Lets you compile both Groovy and Java together!
- IDE support
DEMO

Groovy shell

Groovy console
A bit of history and background

Syntax basics

Interesting Groovy APIs

Domain-Specific Languages

What’s new in Groovy 1.6?

The Groovy ecosystem
public class HelloWorld {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public String greet() {
        return "Hello " + name;
    }

    public static void main(String[] args) {
        HelloWorld helloWorld = new HelloWorld();
        helloWorld.setName("Groovy");
        System.out.println( helloWorld.greet() );
    }
}
public class HelloWorld {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public String greet() {
        return "Hello " + name;
    }

    public static void main(String[] args) {
        HelloWorld helloWorld = new HelloWorld();
        helloWorld.setName("Groovy");
        System.out.println( helloWorld.greet() );
    }
}
From Java to Groovy
class HelloWorld {
    String name
    String greet() { "Hello $name" }
}

def helloWorld = new HelloWorld(name: "Groovy")
println helloWorld.greet()
Native syntax constructs

- Lists
  ```
  def numbers = [1, 2, 3, 4, 5]
  ```

- Maps
  ```
  def map = [FR: "France", BE: "Belgium"]
  ```

- Ranges
  ```
  def allowedAges = 18..65
  ```

- Regular expressions
  ```
  def whitespace = ~/\s+?/
  ```
Groovy Strings

- GStrings: interpolated strings

```groovy
def person = "John"
def letter = """"${new Date()}
    Dear ${person},
    this is a Groovy letter!"
"
println letter
```

- Multiline strings: triple single or double quotes

- Slashy syntax

```groovy
def whitespace = ~/\s+~/
```
Closures

- A reusable / assignable block of code
- delimited by curly braces
- can take parameters
- can be passed as parameters to methods, or inline

```java
def printer = { println it }
def double = { int i -> 2 * i }
new File(“f.txt”).eachLine { line ->
    println line
}
new File(“f.txt”).eachLine( printer )
```
Time savers

The Groovy Truth

```groovy
def blank = ""; assert !blank
def empty = []; assert !empty
```

Safe graph navigation

```groovy
def order = null
assert order?.customer?.address == null
```

Elvis operator

```groovy
def name = customer.name ?: "Unknown"
```
Java 5 features

- Since Groovy 1.5
  - Annotations, generics, static imports, enums, varargs

- You can leverage frameworks supporting annotations
  - Spring, Hibernate, Guice, JPA, EJB3, TestNG, and more

- Sole dynamic language supporting annotations
@Entity
@Name("hotel")
class Hotel implements Serializable {
    @Id @GeneratedValue
    Long id

    @Length(max = 40) @NotNull
    String name

    @Override String toString() {
        "Hotel ${name}"
Groovy agenda

- A bit of history and background
- Syntax basics
- Interesting Groovy APIs
- Domain-Specific Languages
- What’s new in Groovy 1.6?
- The Groovy ecosystem
The GDK — Groovy Dev Kit

- Groovy “decorates” existing JDK APIs
- We can’t extend java.lang.String or java.io.File, can we?

```java
ten File("f.txt").eachLine { println it }
(1..100).findAll { it % 2 == 1 }
speakers.groupBy { it.lastname }

"123".padLeft(5, '0')
"ls -la".execute()
"R3Jvb3Z5IHJvY2tzIQ==".decodeBase64()

Thread.start { /* code to be executed */ }
```
JDBC made easy

With transparent resource handling, thanks to closures

```groovy
def sql = Sql.newInstance(url, usr, pwd, driver)
sql.executeUpdate("insert into table values ($foo, $bar)")
sql.executeUpdate("insert into table values(? ,?)", [a, b])
sql.eachRow("select * from USER") { print it.name }
def list = sql.rows("select * from USER")
```

A poor-man’s dataset notion

```groovy
def set = sql.dataSet("USER")
set.add(name: "Johnny", age: 33)
set.each { user -> println user.name }
set.findAll { it.age > 22 && it.age < 42 }
```
Parsing XML

With this multiline string XML fragment

```java
def xml = ""
    <languages>
        <language name="Groovy">
            <feature coolness="low">SQL</feature>
            <feature coolness="high">Template</feature>
        </language>
        <language name="Perl"/>
    </languages>"
```

Navigate your XML graph as an object graph!

```java
def root = new XmlParser().parseText(xml)
println root.language.feature[1].text() 
root.language.feature .findAll{ it['@coolness'] == "low" } .each{ println it.text() }```
Producing XML too!

The same XML document as before

```java
new MarkupBuilder().languages {
    language(name: "Groovy") {
        feature(coolness: "low", "SQL")
        feature(coolness: "high", "Template")
    }
    language(name: "Perl")
}
```
Following the same “builder pattern”, but for Swing

```groovy
def theMap = [color: "green", object: "pencil"]
def swing = new SwingBuilder()
def frame = swing.frame(
    title: 'A Groovy Swing', location: [240,240],
    defaultCloseOperation: WC.EXIT_ON_CLOSE)
panel {
    for (entry in theMap) {
        label(text: entry.key)
        textField(text: entry.value)
    }
    button(text: 'About', actionPerformed: {
        def pane = swing.optionPane(message: 'SwingBuilder')
        def dialog = pane.createDialog(null, 'About')
        dialog.show()
    })
    button(text: 'Quit', actionPerformed: { System.exit(0) })
}
frame.pack()
frame.show()
```

www.devoxx.com
import javax.swing.WindowConstants as WC
import groovy.swing.SwingBuilder

def theMap = [color: "green", object: "pencil"]
def swing = new SwingBuilder()

def frame = swing.frame(
    title: 'A Groovy Swing', location: [240,240],
    defaultCloseOperation: WC.EXIT_ON_CLOSE)

panel {
    for (entry in theMap) {
        label(text: entry.key)
        textField(text: entry.value)
    }

    button(text: 'About', actionPerformed: {
        def pane = swing.optionPane(message: 'SwingBuilder')
        def dialog = pane.createDialog(null, 'About')
        dialog.show()
    })

    button(text: 'Quit', actionPerformed: { System.exit(0) })
}

frame.pack()
frame.show()
Another builder with Ant

Reuse the wealth of Ant tasks for your build or system scripting needs

```java
// create sequential Ant execution
new AntBuilder().sequential {
    def buildClasses = "build/classes"

    // create classes directory
    mkdir(dir: buildClasses)

    // compile sources
    javac(srcdir: "src",
         destdir: buildClasses)

    // jar everything
    jar(destfile: "my.jar",
         basedir: buildClasses)
}
```
Groovy agenda

- A bit of history and background
- Syntax basics
- Interesting Groovy APIs
- Domain-Specific Languages
- What’s new in Groovy 1.6?
- The Groovy ecosystem
What’s a DSL?

- A Domain-Specific Language is a **small language** designed to be useful for a specific set of tasks, for a given domain.

- Not necessarily Turing-complete.

- Produces some result.
  - configuration, data structures, simulation declarations...

- Can be **internal** or external.

- Has a form: **textual** or visual.
Some examples

- Technical
  - `SELECT * FROM USERS WHERE NAME LIKE 'Guil%'`
  - `^[\w-\.]+@[\w-]+\.[\w-]{2,4}$`

- Notation
  - 1. e4 e5 2. Nf3 Nc6 3. Bb5 a6
  - U R’ U2 R U R’ U R

- Business oriented
  - Risk calculation for insurance policies
  - Human Resources employee skills representation
  - Anti-malaria drug resistance simulation
Towards more readability (1/2)

```java
Compound cq = new Compound();
cq.setName("Chloroquine");
cq.setAbbreviation("cq");
cq.setHalfLife(new Duration(45, Unit.DAY));

Drug drug = new Drug();
drug.setName("Chloroquine");
drug.setAbbreviation("CQ");
drug.includes(new CompoundQuantity(cq, 300, Unit.MILLIGRAM), 1.2);

Regimen regimen = new Regimen();
regimen.add(drug, 2, new Duration(0, Unit.HOUR));
regimen.add(drug, 1, new Duration(6, Unit.HOUR));
regimen.add(drug, 1, new Duration(1, Unit.DAY));
regimen.add(drug, 1, new Duration(1, Unit.DAY));

Effect effect = new Effect();
effect.setName("General");
effect.setFormula(new Formula() {
    BigDecimal calculate(List<BigDecimal> vars) {
        return new BigInteger("3.8").divide(1 + vars.get(1).divide(vars.get(2)));
    }
    void defineParameters() {
        this.params.add("kml", new BigDecimal("68.0"));
    }
});
```
Towards more readability (1/2)

```java
Compound cq = new Compound();
cq.setName("Chloroquine");
cq.setAbbreviation("cq");
cq.setHalfLife(new Duration(45, Unit.DAY));

Drug drug = new Drug();
drug.setName("Chloroquine");
drug.setAbbreviation("CQ");
drug.includes(new CompoundQuantity(cq, 300, Unit.MILLIGRAM), 1.2);

Regimen regimen = new Regimen();
regimen.add(drug, 2, new Duration(0, Unit.HOUR));
regimen.add(drug, 1, new Duration(6, Unit.HOUR));
regimen.add(drug, 1, new Duration(1, Unit.DAY));
regimen.add(drug, 1, new Duration(1, Unit.DAY));

Effect effect = new Effect();
effect.setName("General");
effect.setFormula(new Formula() {
    BigDecimal calculate(List<BigDecimal> vars) {
        return new BigInteger("3.8").divide(1 + vars.get(1).divide(vars.get(2)));
    }
    void defineParameters() {
        this.params.add("kml", new BigDecimal("68.0"));
    }
});
```
Towards more readability (1/2)

```java
Compound cq = new Compound();
cq.setName("Chloroquine");
cq.setAbbreviation("cq");
cq.setHalfLife(new Duration(45, Unit.DAY));

Drug drug = new Drug();
drug.setName("Chloroquine");
drug.setAbbreviation("CQ");
drug INCLUDES new CompoundQuantity(cq, 300, Unit.MILLIGRAM, 1.2);

Regimen regimen = new Regimen();
regimen.add(drug, 2, new Duration(0, Unit.HOUR));
regimen.add(drug, 1, new Duration(6, Unit.HOUR));
regimen.add(drug, 1, new Duration(1, Unit.DAY));
regimen.add(drug, 1, new Duration(1, Unit.DAY));

Effect effect = new Effect();
effect.setName("General");
effect DEFINE Formula(new Formula() {
    BigDecimal calculate(List<BigDecimal> vars) {
        return new BigInteger("3.8").divide(1 + vars.get(1).divide(vars.get(2)));
    }
    void defineParameters() {
        this.params.add("xml", new BigDecimal("68.0"));
    }
});
```
Towards more readability (2/2)

cq = compound(name: "Chloroquine", abbreviation: "cq", halfLife: 45.days)

CQ = drug(name: "Chloroquine", abbreviation: "CQ")
CQ.includes cpd: cq, qty: 300.mg, bioavail: 1.2

regimen = regimen {
    take pills: 2, of: CQ, at: 0.hour
    take pills: 1, of: CQ, at: 6.hours
    take pills: 1, of: CQ, at: 1.day
    take pills: 1, of: CQ, at: 2.days
}

cqEffect = effect(
    name: "General",
    formula: { 3.8 / (1 + kml/cq) },
    parameters: [kml: 68.0]
)
Towards more readability (2/2)

cq = compound(name: "Chloroquine", abbreviation: "cq", halfLife: 45.days)

CQ = drug(name: "Chloroquine", abbreviation: "CQ")
CQ.includes cpd: cq, qty: 300.mg, bioavail: 1.2

regimen = regimen {
    take pills: 2, of: CQ, at: 0.hour
    take pills: 1, of: CQ, at: 6.hours
    take pills: 1, of: CQ, at: 1.day
    take pills: 1, of: CQ, at: 2.days
}

cqEffect = effect(
    name: "General",
    formula: { 3.8 / (1 + kml/cq) },
    parameters: [kml: 68.0] )
Towards more readability (2/2)

cq = compound(name: "Chloroquine", abbreviation: "cq", halfLife: 45.days)

CQ = drug(name: "Chloroquine", abbreviation: "CQ")
CQ.includes cpd: cq, qty: 300.mg, bioavail: 1.2

regimen = regimen {
  take pills: 2, of: CQ, at: 0.hour
  take pills: 1, of: CQ, at: 6.hours
  take pills: 1, of: CQ, at: 1.day
  take pills: 1, of: CQ, at: 2.days
}

cqEffect = effect(
  name: "General",
  formula: { 3.8 / (1 + km1/cq) },
  parameters: [km1: 68.0]
)
Groovy’s MOP to the rescue

- DSLs are possible in Groovy because of
  - Groovy’s **malleable syntax** and APIs
  - Omitting parentheses, semi-columns
  - Named arguments, Closures
  - Everything is an object
  - Groovy’s dynamic nature

- **MOP**: Meta-Object Protocol
  - Nothing hard-wired, **behavior modifiable at runtime!**
Parentheses & named args

- Optional parentheses
  - move left

- Named arguments
  - `compare fund: 'XYZ', withIndicator: 'NIKEI'`
  - `account.debit amount: 30.euros, in: 3.days`

- Plus native syntax elements
  - Lists: `[NIKEI, CAC40, NASDAQ]`
  - Maps: `[CA: 'California', CO: 'Colorado']`
  - Ranges: `Monday..Friday`
Adding properties to numbers

Categories

ExpandoMetaClass
Operator overloading

- Operators are a shortcut for method calls
  
  - $a + b == a.plus(b)$
  - $a - b == a.minus(b)$
  - $a * b == a.multiply(b)$
  - $a / b == a.divide(b)$
  - $a % b == a.modulo(b)$
  - $a | b == a.or(b)$
  - $a & b == a.and(b)$
  - $a[b] == a.getAt(b)$
  - $a << b == a.leftShift(b)$
Operator overloading

- Operators are a shortcut for method calls
  - \( a + b \) == \( a\).plus\( (b) \)
  - \( a - b \) == \( a\).minus\( (b) \)
  - \( a \times b \) == \( a\).multiply\( (b) \)
  - \( a / b \) == \( a\).divide\( (b) \)
  - \( a \% b \) == \( a\).modulo\( (b) \)
  - \( a \mid b \) == \( a\).or\( (b) \)
  - \( a \& b \) == \( a\).and\( (b) \)
  - \( a[b] \) == \( a\).getAt\( (b) \)
  - \( a << b \) == \( a\).leftShift\( (b) \)

- Currency arithmetics
  - \( 30\).euros + 15\).dollars

- Distances
  - \( 12\).kilometers + 3\).meters

- Parallelism or workflow
  - \( \text{taskA} \& \text{taskB} | \text{taskC} \)

- Banking
  - \( \text{account} += 10\).euros
Closures and builders

- Remember the builder pattern? Ant, Swing, Markup...
- You can easily create your own
- Extend BuilderSupport or FactoryBuilderSupport
- Nested method calls with closures as last argument

```java
plan {
    artifact {
        name = 'spring.core'
        type = 'bundle'
        version = '2.5.6'
    }
}
```
Why creating your own DSL?

- Use a **more expressive language** than a general purpose language.
- Share a **common metaphor** between developers and subject matter experts.
- Have **domain experts help** with the design of the business logic of an application.
- Avoid cluttering business code with **too much boilerplate technical code**.
- **Cleanly separate** business logic from application code.
- Let business rules have their **own lifecycle**.
Groovy agenda

- A bit of history and background
- Syntax basics
- Interesting Groovy APIs
- Domain-Specific Languages
- What’s new in Groovy 1.6?
- The Groovy ecosystem
Main focus on performance
to make Groovy the fastest dynamic language

Multiple assignment

```
def (a, b) = [1, 2]
def (int i, String s) = [1, 'Groovy']
(a, b) = functionReturningAList()
```

AST Transformations

@Lazy, @Immutable, @Singleton, @Delegate

Annotation definition
A bit of history and background
Syntax basics
Interesting Groovy APIs
Domain-Specific Languages
What’s new in Groovy 1.6?
The Groovy ecosystem
More than just Groovy...

- First major Groovy child: Grails, but there’s more...

- **Griffon**: a Grails-like MVC framework for building rich Swing applications

- **Gradle**: a new build system learning from the good ideas and mistakes of Ant, Maven and Ivy

- **Easyb**: a Behavior Driven Development toolkit which lets you write user stories with a Groovy DSL

All three will be presented at Devoxx, check your agenda!
The Groovy dynamic language for the JVM simplifies the life of developers.

Groovy opens up some interesting perspectives towards extending and enhancing your applications.

Groovy provides the most seamless Java integration experience.

Groovy lets you create Domain-Specific Languages.

Groovy protects your investment in skills, tools, libraries and application servers.
Grails in Action

A bit of history and background
The Grails stack
Getting started with scaffolding
The layers: persistence, controllers & services, views
The plugin ecosystem
What’s new in Grails 1.1?
Grails agenda

- A bit of history and background
- The Grails stack
- Getting started with scaffolding
- The various layers
- The plugin ecosystem
- What’s new in Grails 1.1?
Guillaume Laforge on the Groovy lists in 2005:

Ruby on Rails is all the rage these days, but you compromise on your past investment, if you’re a Java shop: developer and IT team skills, training, app servers, the great JVM platform, third-party libraries...

Can’t we leverage Groovy and proven technologies like Spring and Hibernate to bring the “Convention over Configuration” paradigm on the Java platform, without compromising on your investment?
Observation: **lack of productivity** in webapp development

We have all the tools at our disposal to change that

Back in 2005, Graeme Rocher, Steven Devijsver and myself launched Grails

Apache licensed OSS project, hosted at Codehaus

Goal similar to Groovy: **simplify the life of developers**

So what’s under the hood?
Grails agenda

- A bit of history and background
- The Grails stack
- Getting started with scaffolding
- The various layers
- The plugin ecosystem
- What’s new in Grails 1.1?
Grails is an MVC action-based framework

Grails follows good principles popularized by Rails

CoC: Convention over Configuration

DRY: Don’t Repeat Yourself

The essence of Rails but with tight integration with the Java platform to protect your investment!
From 1,000 feet

GRAILS

Groovy + Spring 2.5 + Hibernate
Near the ground...

- Grails is built on **rock-solid and proven technologies**
- the JVM, the Java EE specs, existing app servers, and...

  - **Spring**: IoC, DI, Spring MVC, Spring WebFlow
  - **Hibernate**: Object-Relational Mapping
  - **SiteMesh**: page layout and composition
  - Quartz: for job scheduling
  - **Jetty** and **HSQLDB**: for fast development cycles
  - **Productive** out of the box with a **full development stack**
Grails agenda

- A bit of history and background
- The Grails stack
- Getting started with scaffolding
- The various layers
- The plugin ecosystem
- What’s new in Grails 1.1?
First step, download & install

- Go to http://grails.org/Download
- Download the latest archive
- Unzip it somewhere
- Create GRAILS_HOME
- Add $GRAILS_HOME/bin to your PATH

You’re ready to go!

Next step: grails create-app
Getting started with scaffolding

Remember scaffolding is nice for CRUD, admin interfaces, prototyping, getting up and running quickly, but Grails is more than just a CRUD framework!
Layout and naming convention

- grails-app
  - conf
  - controllers
  - domain
  - i18n
  - realms
  - services
  - taglib
  - utils
  - views
  - lib
  - plugins
    - pom.xml
  - scripts
  - src
  - test
  - web-app
Did you notice?

- Where are my configuration files?
- Why haven’t I written any XML file?
- Where are my DAOs?
- Where are my mapping files?
- No database to configure?
- Nor a servlet container or app server to install?
- Or tons of jars to download from Maven repositories?
- There must be some magic under the hood :-)

www.devoxx.com
Grails agenda

- A bit of history and background
- The Grails stack
- Getting started with scaffolding
- The various layers
- The plugin ecosystem
- What’s new in Grails 1.1?
The usual problems

- When you start a project, you have to setup everything
  - The build, wire all components together

- You suffer from impedance mismatch
  - You have to handle the ORM mapping manually

- It’s hard to write clean views
- Taglibs are a pain to develop
Persistence

- Enters **GORM**: Grails Object Relational Mapping
- **Hibernate** under the hood
- Your domain model is a set of POGOs
- Your domain classes are **transparentely mapped**
- Although you can use legacy schemas with GORM DSL
- **No conf**: hibernate.cfg.xml, no *.hbm.xml files!
- But you can bring your own existing ones if needed
- Always a path to configuration when in need
Persistence

- Enters **GORM**: Grails Object Relational Mapping
- **Hibernate** under the hood
- Your domain model is a set of POGOs
- Your domain classes are *transparently* mapped
- Although you can use legacy schemas with GORM DSL
- **No conf**: hibernate.cfg.xml, no *.hbm.xml files!
- But you can bring your own existing ones if needed
- Always a path to configuration when in need

*Not limited by conventions?
All domain classes have got injected methods like

- Book.count(), Book.list(), Book.get(id)
- book.save(), book.delete(), etc...

Dynamic finders

- Book.findByTitle("Harry Potter")
- Book.findAllByAuthorLike("%Rowlins")
- Book.findAllByTitleLikeAndReleaseDateLessThan(
  `%Grails%`, 2007
) // don’t do that :-(
Dynamic finders

- All domain classes have got injected methods like
  - Book.count(), Book.list(), Book.get(id)
  - book.save(), book.delete(), etc...

Dynamic finders

- Book.findByTitle("Harry Potter")
- Book.findAllByAuthorLike("%Rowlins")
- Book.findAllByTitleLikeAndReleaseDateLessThan("%Grails%", 2007)

// don't do that :-)

Where's my DAO?
Queries

- You can use raw HQL queries
  ```java
  Book.find("from Book b where b.title = ?", ['Shining'])
  ```

- Grails wraps the Hibernate Criteria API
  ```java
  def results = Account.list {
    like "holderFirstName", "Fred%"
    and {
      between "balance", 500, 1000
      eq "branch", "London"
    }
    order "holderLastName", "desc"
  }
  ```
Constraints

- Constraints can be added to domain classes through a static constraints field

  ```java
  static constraints = {
    isbn matches: /[0-9]{9}[0-9X]/
  }
  ```

- Many constraints available
  - blank, creditcard, email, nullable, matches, range, unique

- Create your own validator

  ```java
  myIntField = { it % 2 == 0 }
  ```

- You can also register your own constraints
Controllers

- In the controller directory, name ending with Controller
  - \texttt{http://myserver:8080/app/controller/action}
- Handy things at your disposal
  - redirect, chain, render, flash scope
- Controllers return Maps as models for the view to render
  - Possible access to the raw output stream
- Easy databinding from form fields
  - \texttt{book.properties = params}
Controllers

- In the controller directory, name ending with Controller
- http://myserver:8080/app/controller/action
- Handy things at your disposal
  - redirect, chain, render, flash scope
- Controllers return Maps as models for the view to render
- Possible access to the raw output stream
- Easy databinding from form fields
- book.properties = params
Services

- Another naming convention and placement
- Transactional by default
- Just declare your service, it’ll be transparently injected!
- no need to configure it in some XML file
The view tier

- The view tier is essentially composed of two things
  - The Groovy Server Pages
    - Similar to JSPs
    - But with GString interpolation
  - The Grails Tag Libraries
    - Useful existing tag libraries
    - So easy to write your own!
  - Templates, URL mapping, conversations and more
GSPs

Auto-reloading

```html
<html>
  <head>
    <meta name="layout" content="main" />
    <title>Book List</title>
  </head>
  <body>
    <a href="${createLinkTo(dir:'')}">Home</a>
    <g:link action="create">New Book</g:link>
    <g:if test="${flash.message}">
      ${flash.message}
    </g:if>
    <g:each in="${bookList}">${it.title}</g:each>
  </body>
</html>
```
Auto-reloading

<html>
  <head>
    <meta name="layout" content="main" />
    <title>Book List</title>
  </head>
  <body>
    <a href="${createLinkTo(dir:'')}">Home</a>
    <g:link action="create">New Book</g:link>
    <g:if test="${flash.message}>${flash.message}
    </g:if>
    <g:each in="${bookList}">${it.title}</g:each>
  </body>
</html>
Taglibs

- Lots of existing taglibs
- **Logical**: if, else, elseif
- **Iterative**: each, collect, findAll
- **Linking**: createLink, createLinkTo
- **Ajax**: remoteLink, remoteForm, remoteFunction
- **Form**: select, datepicker, currencySelect
- **Validation**: hasError, eachError
- **UI**: rich text editor
Create your own taglib

Yet another Grails convention

```groovy
class MyTaglib {
    def isAdmin = { attrs, body ->
        def user = attrs['user']
        if (user && checkUserPrivs(user)) {
            body()
        }
    }
}
```

Use it in your GSP

```groovy
<g:isAdmin user="${user}">
    Some restricted content
</g:isAdmin>
```
Create your own taglib

Yet another Grails convention

class MyTaglib {
    def isAdmin = { attrs, body ->
        def user = attrs[‘user’]
        if (user && checkUserPrivs(user)) {
            body()
        }
    }
}

Use it in your GSP

<g:isAdmin user="${user}">
    Some restricted content
</g:isAdmin>
Custom URL mappings

Not satisfied with the default URL mapping?

Define your URLs yourself!

class UrlMapping {
  static mappings = {
    "$blog/$yeah?/$month?/$day?" (controller: 'blog',
      action: 'show') {
      constraints {
        year matches: /\d{4}/
      }
    }
  }
}
What have I not shown?

- **Conversation** concept using Spring WebFlow
- **Templates** for reusing view logic and presentation
- Internationalization
- You can reuse JSP tag libraries (in Grails 1.1)
- SiteMesh layouts and page componentization
- Handling of REST style architecture
- Unit and integration tests
- **Reuse legacy schemas** or **Spring configurations**
What have I not shown?

- **Conversation** concept using Spring WebFlow
- **Templates** for reusing view logic and presentation
- Internationalization
- You can reuse JSP tag libraries (in Grails 1.1)
- SiteMesh layouts and page componentization
- Handling of REST style architecture
- Unit and integration tests
- Reuse legacy schemas or Spring configurations

Not limited by conventions?
Grails agenda

- A bit of history and background
- The Grails stack
- Getting started with scaffolding
- The various layers
- The plugin ecosystem
- What’s new in Grails 1.1?
A wealth of plugins

- Grails is built on top of a core plugin system
- **100+ plugins already available**
- **Testing**: Webtest, Easyb, Selenium, jsUnit, Fitnesse
- **Rich client / AJAX**: Yahoo, Ext-JS, GWT, jQuery, iUI
- **Web Services**: XFire, remoting, Axis2, Metro
- **Security**: Spring Security, JSecurity
- **Search**: Compass integration
- Other integration: Drools, Struts, GridGain, Terracotta
A bit of history and background
The Grails stack
Getting started with scaffolding
The various layers
The plugin ecosystem
What’s new in Grails 1.1?
Grails 1.1

- JSP tag library support and reuse
- New GORM events
- Improvements to data binding
- Read-only access to domain objects
- More options for the GORM DSL
- Improvements to dynamic finders
- More support for legacy mappings
- New testing plugin included for easier testing
Grails Summary

- Grails is not just a web framework, but a **full-blown development stack**
- Grails is not a clone of Ruby on Rails
- Grails is **built on proven rock-solid technologies**
- Grails brings back the **productivity to Java developers**
- Grails is a **new weapon for the war on Java complexity**
Thanks for your attention!

http://groovy.codehaus.org
http://grails.org
http://www.springsource.com/g2one
http://glaforge.free.fr/blog/groovy

glaforge@gmail.com
Pictures from the web used in this presentation:

Question marks: http://weblogs.newsday.com
Agenda: http://www.jouy.inra.fr
Mop: http://www.spitjack.com
Harry Potter: http://madhavgopalkrish.files.wordpress.com