



Bootstrapping into Groovy & Grails



Håkan Dahl / Johan Eltes
hakan.dahl@callistaenterprise.se / johan.eltes@callistaenterprise.se
www.callistaenterprise.se



Agenda

- Why Groovy and Grails?
- Groovy highlights
- Grails overview
- Grails application quickstart
- Conclusions
- Questions

Why Groovy and Grails?

- Productivity
- Fun
- Groovy is Java – only so much more!
- Overcome annoyances with the Java language

Neal Ford

<http://memeagora.blogspot.com/2008/02/real-jdk-20.html>

...

Without realizing it, the technical guys didn't think (and apparently still don't) that Java was good enough for a real Version 2. They've been waiting...and now it's here. Groovy is the REAL Java 2.

...

Compare to some expected Java 7 features:

<http://tech.puredanger.com/java7>

GroovyAndGrails, Slide 3
Copyright 2008, Callista Enterprise AB



First look at Groovy

- Groovy is an extension to the JDK
 - Groovy is a language based Java and the JDK- not only the JVM
- Why is Groovy more Java than JRuby, Scala?
 - JRuby and Scala are languages in their own right
 - They have their own type systems and frameworks, but gain interop with Java classes through the JVM
- So, in what way is Groovy different from Java?
 - It is a dynamic language
 - Everything(!) is an object
 - Powerful stuff like Groovy Builders and the Grails framework can be achieved just by using the language (no code gen etc)
- Release 1.5, dec 2007 (*1.0 jan 2007*)

GroovyAndGrails, Slide 4
Copyright 2008, Callista Enterprise AB



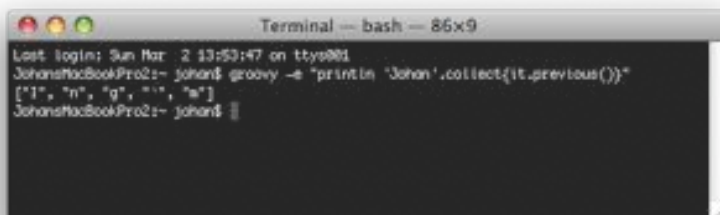
How can we run groovy?

- Dynamically compiled or pre-compiled
- Dynamic options
 - Code on the command line
 - Code in a script file (incrementally creates .class files)
 - .class files stored in memory
 - From within Java, e.g. using Bean Scripting Framework or GroovyShell
- Pre-compiled options
 - Use groovy pre-compiler as part of your build
 - This is what IDE:s do
- A .groovy file may contain “free” script code and classes

GroovyAndGrails, Slide 5
Copyright 2008, Callista Enterprise AB



Running Groovy: From command line

A screenshot of a terminal window titled "Terminal -- bash -- 86x9". The terminal shows the following text:

```
Last login: Sun Mar 2 13:53:47 on ttys001
JohansMacBookPro21~ johan$ groovy -e "println 'Johan'.collect{it.previous()}"
[[!], "n", "g", "", "e"]
JohansMacBookPro21~ johan$
```

GroovyAndGrails, Slide 6
Copyright 2008, Callista Enterprise AB



Running Groovy: Running Groovy from script file

The top window shows a text editor with the following Groovy code:

```
println 'Johan'.collect{it.previous()}
```

The bottom window is a terminal showing the execution of the script:

```
Terminal -- bash -- 101x10
Last login: Sun Mar  2 18:41:00 on ttys002
JohansMacBookPro2:~ johan$ echo 'println 'Johan'.collect{it.previous()}' > MyGroovyScript.groovy
JohansMacBookPro2:~ johan$ groovy MyGroovyScript.groovy
[*I', 'n', 'g', '', 'm']
JohansMacBookPro2:~ johan$
```

GroovyAndGrails, Slide 7
Copyright 2008, Callista Enterprise AB



Running Groovy: Pre-compiling into Java classes

The top window shows the same Groovy code as in the previous slide:

```
println 'Johan'.collect{it.previous()}
```

The bottom window is a terminal showing the pre-compilation process:

```
Terminal -- bash -- 101x12
Last login: Sun Mar  2 19:53:03 on ttys002
JohansMacBookPro2:~ johan$ groovyc MyGroovyScript.groovy
JohansMacBookPro2:~ johan$ ls *.class
MyGroovyScript$_run_closure1.class  MyGroovyScript.class
JohansMacBookPro2:~ johan$ java -cp .:$GROOVY_HOME/embeddable/groovy-all-1.5.4.jar MyGroovyScript
[*I', 'n', 'g', '', 'm']
JohansMacBookPro2:~ johan$
```

GroovyAndGrails, Slide 8
Copyright 2008, Callista Enterprise AB

groovyc supports joint-compilation with Java-code.



Groovy properties – reducing boilerplate code

```
class Customer {
    String name
    String address
}
```

- Access scope
 - properties by default private
 - classes and methods by default public
- Accessor methods (getters/setters) provided by Groovy
 - compare to the typical code generation approach in Java

Demo!

GroovyAndGrails, Slide 9
Copyright 2008, Callista Enterprise AB



Maps

```
def map = [x:1, y:2]

class MyClass {
    def x
    def y

    def sum() {
        x+y
    }
}

def tre = new MyClass(map)

def sju = new MyClass(x:3, y:4)

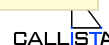
println "tre: ${tre.sum()}, sju: ${sju.sum()}"

println "Property access: ${tre.x} looks like map access: ${map.x}"

print map.getClass()
```

Demo!

GroovyAndGrails, Slide 10
Copyright 2008, Callista Enterprise AB



Closures 1 – basics

```
// define a block of code
def simpleClosure = { println "Hello world" }
// call closure as a method
simpleClosure()

def closureWithNamedParam = { name -> println "Hello ${name}" }
closureWithNamedParam("world")

// a single unnamed parameter to a closure can be
// referred to by the implicit argument "it"
Closure closureWithDefaultParam = { println "Hello ${it}" }
closureWithDefaultParam "world"

// catching the surrounding context
def name = "world"
Closure closureCatchingContext = { println "Hello ${name}" }
closureCatchingContext()
```

GroovyAndGrails, Slide 11
Copyright 2008, Callista Enterprise AB

Demo!



Closures 2 – Closures as arguments

```
[1,2,3].each( { println "I can count ${it}" } )
[1,2,3].each { println "I can count ${it}"}

class ClosureDemo {
  def lastArgIsClosure(name, closure) {
    closure(name)
  }
}

def cd = new ClosureDemo()
cd.lastArgIsClosure("world") { println "Hello ${it}" }
```

When the last argument is a closure, it can be specified after the argument list. The value becomes obvious when there is a lot of code in the closure.

Demo!

GroovyAndGrails, Slide 12
Copyright 2008, Callista Enterprise AB



Closures 3 – resource handling

- Typically the IO-related API:s in Java have coupled methods:

1. create/open resource
2. use resource
3. close resource

- Forgetting to close a resource – or bad exception handling leads to resource leakage
 - running out of file descriptors, sockets, memory, ...
- Groovy has added some sugar in the GDK for this

Demo!

GroovyAndGrails, Slide 13
Copyright 2008, Callista Enterprise AB



Meta-programming: invokeMethod

- Meta-programming: ~ “programming the program”
 - enables extensions to a program at runtime

```
class TheOnlyClassYouWillEverNeed {
    def invokeMethod(String methodName, Object args) {
        "method ${methodName} invoked on instance of class ${this.getClass().name}"
    }

    def theOnlyMethodImNotMissing() {
        "The only method I'm not missing";
    }
}

def cameleont = new TheOnlyClassYouWillEverNeed()

println "${cameleont.theOnlyMethodImNotMissing()}"
println "${cameleont.hiThere()}"
println "${cameleont.doSomUsefulStuff()}"
```

The core mechanism for DSL-ish
meta-programming Grails and
Builders.

Demo!

GroovyAndGrails, Slide 14
Copyright 2008, Callista Enterprise AB



Groovy Builders – putting it all together

```
import groovy.xml.MarkupBuilder

def writer = new StringWriter()

def xml = new MarkupBuilder(writer)

xml.records() {
  car(name:'HSV Maloo', make:'Holden', year:2006) {
    country('Australia')
    record(type:'speed', 'Production Pickup Truck with speed of 271kph')
  }
  car(name:'P50', make:'Peel', year:1962) {
    country('Isle of Man')
    record(type:'size', 'Smallest Street-Legal Car at 99cm wide and 59 kg in weight')
  }
  car(name:'Royale', make:'Bugatti', year:1931) {
    country('France')
    record(type:'price', 'Most Valuable Car at $15 million')
  }
}

println writer
```

GroovyAndGrails, Slide 15
Copyright 2008, Callista Enterprise AB

Demo!



Misc Groovy language features

- Only RuntimeException's
 - Groovy wraps all checked exceptions in RuntimeException

- ?. navigational safety (*against null*)

```
def b = a?.toString()
```

- Groovy boolean evaluation
 - wider range of evaluation criterias than Java

```
Example:
// these all eval to false
if (false)
if (0)
if ([])
if (null)
if ("")
```

GroovyAndGrails, Slide 16
Copyright 2008, Callista Enterprise



Groovy unit-testing

- Unit test framework support
 - `GroovyTestCase` (JUnit 3.8.2)
 - JUnit 4 Demo!
- Mocking
 - ducktyping and meta-programming give super-powers!
 - built in `StubFor` and `MockFor` (*pkg groovy.mock.interceptor*)
 - or use EasyMock (*or any of the other mock-frameworks*)

**Writing tests won't hurt
- but skipping it will!**

GroovyAndGrails, Slide 17
Copyright 2008, Callista Enterprise AB



What about Groovy performance?

Nice dynamic features, but won't that be very slow in runtime... ?

- Be aware: these are micro-benchmarks
<http://shootout.alioth.debian.org/>
 - Groovy (1.6-beta-1) >~ JRuby (1.1)
 - JRuby claims to generally be as fast as Ruby MRI (C-impl)
 - Scala (2.7.0) ~< Java 6 (*~5-15 faster than Groovy/JRuby*)
- JSR-292 (invokedynamic) for Java SE 7
 - expected to bring performance improvements

GroovyAndGrails, Slide 18
Copyright 2008, Callista Enterprise AB



Agenda

- Why Groovy and Grails?
- Groovy highlights
- **Grails overview**
- Grails application quickstart
- Conclusions
- Questions

GroovyAndGrails, Slide 19
Copyright 2008, Callista Enterprise AB



Grails overview

- What is it?
 - a framework for web application development in Groovy
 - CRUD functionality out of the box
 - commands to generate templates for most types of artifacts
 - heavily *inspired* by Ruby-on-Rails
- Release 1.0, feb 2008 (*started mid 2005*)
- Open source
 - <http://grails.codehaus.org/>
 - Apache license (*but packages others*)
- Backed by G2One

GroovyAndGrails, Slide 20
Copyright 2008, Callista Enterprise AB



Grails overview cont'd

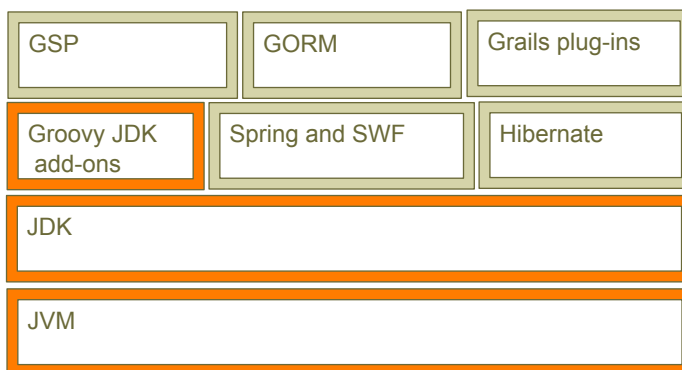
- Characteristics
 - Built on what you already know (*as a Java developer*)
 - Java (via Groovy), Spring, Hibernate
 - Exception: Has it's own HTML template language
 - Convention over configuration
 - Model-driven conventions reduces need for coding and configuration
 - Uses meta-programming over code generation
 - No XML (almost)
 - Don't Repeat Yourself (DRY)

Major diff to Rails: domain classes vs database

GroovyAndGrails, Slide 21
Copyright 2008, Callista Enterprise AB



Major Grails building blocks



GroovyAndGrails, Slide 22
Copyright 2008, Callista Enterprise AB



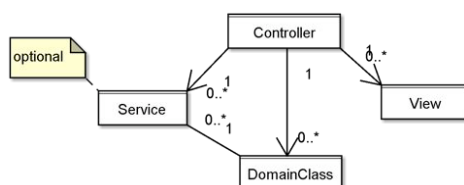
Agenda

- Why Groovy and Grails?
- Groovy highlights
- Grails overview
- **Grails application quickstart**
- Conclusions
- Questions

GroovyAndGrails, Slide 23
Copyright 2008, Callista Enterprise AB



Creating a Grails application



- Start with the domain

```

class Customer {
    String name
    String address
}
  
```

Demo!

GroovyAndGrails, Slide 24
Copyright 2008, Callista Enterprise AB



Understanding Controllers - conventions at work

Lab2 ← <http://localhost:8080/Lab2/customer/create>

Lab2 (/Users/jo)

- grails-app
 - conf
 - controller

```

class CustomerController {
    def scaffold = true
    def create = {
        render 'Create was called!'
    }
}

```

An explicit action in the controller will override the meta-programmed implementation

These actions are generated by default:

edit	index
update	show
create	list
save	delete

GroovyAndGrails, Slide 25
Copyright 2008, Callista Enterprise AB



What's next?

- Add some more Domain Classes
 - Learn some basic GORM
- Write some set-up code for Domain instances
 - To simplify testing
- Generate a View
 - so that we can edit it
- Generate an explicit controller
 - to understand a bit more of Grails
- Create a service class
 - To see some Spring DI and TX in action

GroovyAndGrails, Slide 26
Copyright 2008, Callista Enterprise AB



More GORM

- id, version added dynamically to domain objects
- create relations
- add constraints
- dynamic finder methods

```
class Customer {
    String name
    String address

    static hasMany = [bookings:Booking]
}
```

```
class Booking {
    Date orderDate
    BigDecimal amount

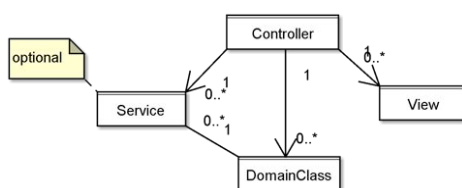
    static belongsTo = [customer:Customer]
}
```

Demo!

GroovyAndGrails, Slide 27
Copyright 2008, Callista Enterprise AB



Services + dependency injection (Spring)



- Services are optional to use
 - use for orchestration logic with declarative transactions
 - can be injected into controllers (and into other services)
- Promotes a rich domain model (*with behaviour*)
- DI available for all beans defined in:
 - `grails-app/conf/spring/resources.groovy` (using *Spring DSL*)
 - `grails-app/conf/spring/resources.xml`

GroovyAndGrails, Slide 28
Copyright 2008, Callista Enterprise AB



Services + dependency injection cont'd

```
class BookingService {
  def transactional = true

  def createBooking(customerId, bookingDetails) {
    println "createBooking"
    /* create booking + details and save them */
  }
}
```

DI

```
class BookingController {
  def bookingService

  def create = {
    bookingService.createBooking(
      params.customerId,
      params.booking)
  }
}
```

GroovyAndGrails, Slide 29
Copyright 2008, Callista Enterprise AB



Spring WebFlow - DSL

- Benefits over WebFlow-XML config
 - convention for binding to view (no need to be explicit)
 - can do anything in Groovy, not constrained to only invoking a service
 - avoids the reflection-like syntax in method calls (*bean-action*)

ResultListController.groovy

GroovyAndGrails, Slide 30
Copyright 2008, Callista Enterprise AB



Agenda

- Why Groovy and Grails?
- Groovy highlights
- Grails overview
- Grails application quickstart
- **Conclusions**
- Questions

GroovyAndGrails, Slide 31
Copyright 2008, Callista Enterprise AB



Wishlist for next Grails release

- Better Maven integration
 - use Maven repos out-of-the-box
- JPA support
 - would enable sharing of existing persistence archives (via Maven) for building parts of a system in Grails (like an admin-UI)

Both Maven and JPA support are mentioned in the Grails 1.1 Roadmap.

GroovyAndGrails, Slide 32
Copyright 2008, Callista Enterprise AB



Conclusions

- Quick start for Java developers
- Speeds up development
- Deploy to the same well-known infrastructure as before
 - no executive descions needed
- Leverages stable and well-proven frameworks
 - Spring for dependency injection
 - Hibernate mapping can support existing databases (doesn't require greenfield development)
 - WebFlow provides a robust mekanism for the common web-problems

GroovyAndGrails, Slide 33
Copyright 2008, Callista Enterprise AB



Callista blir Sverige-partner för G2One

Callista blir Sverige-partner för G2One

Som ett led i vår satsning på agil utveckling för Java-plattformen, har Callista ingått samarbetsavtal med [G2One](#). Vi är "G2One training and consulting partner in Sweden".



Förutom att erbjuda kompetens inom agil utveckling med Groovy och Grails, arrangerar vi kurser i Göteborg, Stockholm och Malmö. Kurserna hålls företrädesvis av lärare från G2One - företaget bakom Groovy och Grails.

Information och anmälningsformulär för våra öppna Grailskurser hittar du [här](#).

<http://www.callistaenterprise.se/nyheter/nyhetsarkiv/callistablirsverigepartnerforg2one>

GroovyAndGrails, Slide 34
Copyright 2008, Callista Enterprise AB



Utbildning: Agil utveckling med Groovy och Grails

Kursen hålls på engelska av företaget från företaget Jaxxon Grails - G2One.

[Show ut »](#)



En personlig licens av Idea Intelliger ingår i kursavgiften. Licensen räcker 12 månader.

Schemalagd: Ja.

Omfattning: 3 dagar.

Pris: 18 500 kr + moms

Förkunskap: Grundläggande web-utveckling i Java.

Övrigt: Övningar sker på deltagarens egen dator. Installationsinstruktioner tillhandshålls i förväg. Kursledaren finns tillgänglig på telefon för installationsutjälp.

This 3 day course covers both Groovy language and Grails framework. After an introduction to the basics of the Groovy language, students are taken through a step-by-step labdriven experience on how to build a solid Grails application learning from the people who created the technology.

[Click for full course description »](#)


Anmälan

Kursutgåva *

- Göteborg 2008-09-08
- Stockholm 2008-09-15
- Malmö 2008-09-22

<http://www.callistaenterprise.se/tjanster/utbildningar/agilutvecklingmedgroovyochgrails>

GroovyAndGrails, Slide 35
Copyright 2008, Callista Enterprise AB



Questions

?