

Scripting? That's Groovy!

Tom Janofsky
Chariot Solutions

Speaker

- Architect with Chariot Solutions
- Lecturer with Penn State
- Speaker at Java, BEA user groups

Groovy?



Groovy - what is it?

- A scripting language built on top of, and tightly integrates with, Java
- Uses constructs from Ruby and Python for a more compact syntax
- Has a Java-like syntax - easy for Java developers to learn

History

- First beta released December 2003
- Still in beta in 2/04
- Created by James Strachan (also involved with Jelly, dom4j, and Maven) and Bob McWhirter

What is it?

- Runs on top of the JVM
- Can use existing Java classes, can be called from Java code
- Can be compiled to bytecode or run in interpreter

Should we use little languages?

- Why not?
- Expense and learning curve of maintaining a second skill set
- What if support and development dry up?

Little languages

- Why should we use them?
- Take Dave Thomas' advice and learn new languages
- Use languages in effective specialized domains (e.g. unit tests)

Features

- native syntax for maps, lists, regex and autoboxing
- closures, static or dynamic typing, operator overloading, text templating
- GPath (an XPath like language)
- GroovyMarkup - for working with XML, Swing, Ant (anything tree based)
- Groovlets - servlets
- GroovySql - SQL with Groovy

Where to use?

- Unit tests!
- A workflow engine with Groovlets
- Easy web services
- Code generation

Setting it up

- Go to <http://groovy.codehaus.org/>
- Follow the download link
- Download and unzip
- Set `GROOVY_HOME` to path
- Add `GROOVY_HOME/bin` to `PATH`

Running

- `groovysh` - command shell interpreter
- `groovyConsole` - interactive Swing console
- `groovy <filename>` - run script file
- as a Unix script - start with `#!/usr/bin/env groovy`

The Language

Statements

- semicolons are optional, can be used to separate multiple statements on a line
- method calls are made with the dot operator, just like Java
- the return statement is optional, and the return type of a method does not need to be specified (Object in bytecode)

Statements

- Can omit parenthesis if there is at least one param, and it is unambiguous
- Dynamic method dispatch is used by default (i.e. you do not need to supply types) - static typing can be added by adding a type

Statements

- Can avoid `NullPointerException` by using `->` in place of `.`
- JavaBean style properties can be accessed using a dot notation (like `BeanUtils`)

Strings

- Uses both “ and ‘ for strings, making escaping quotes easy
- Can span multiple lines
- Supports Heredocs like PHP - avoid escaping large chunks of HTML, SQL

Heredocs

- Have no quote problems
- Easily assign multiple lines of text in a straightforward fashion

```
foo = <<<XMLDOC
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "...
XMLDOC
```

Expressions

- Can contain arbitrary expressions in `${...}`
- Represented by a `GString` object, substitution done when needed
- Very handy for `toString` methods

```
name = "SomeClass"
classDoc = <<<DOC
public class ${name} {
    //generated at ${new java.util.Date()}
    public ${name}() {

        //default constructor
    }
}
DOC
```

Collections

- native support for collections, lists, maps and arrays
- Can create lists using `[]` syntax
- Can define a Range with `..` notation, Range extends list, `...` excludes last value
- Can use Range with anything Comparable that also has `increment()` and `decrement()`

Maps

- Create maps quickly with special syntax
- Can also access map elements with JavaBean style notation if the keys are Strings

Subscripts

- You can use subscript notation on lists, maps, Strings, arrays, and regexes
- Can be combined with Ranges
- Negative indices can be used to count backwards from the end

Switch statement

- What switch should have been all along
- Match on any type of object
- Check for:
 - Class matches if switch value instance of class
 - regex matches if switch String value matches regex
 - collection matches if the switch value is contained in the collection
 - otherwise, uses equals

for loop

- Works on arrays, collections, maps
- `for (i in 0..9)`
- `for (i in [1,3,5,7])`
- iterate on lists, strings, arrays

Closures

- closures are objects that are blocks of executable code with one method that can be called
- variables declared outside closures can be used inside, and vice-versa
- by default, takes one parameter - called “it”
- additional parameters can be passed in - use comma separated list, followed by |

Classes

- Classes can be defined in Groovy just as in Java
- By default fields and methods are public
- Methods in Groovy classes are available to Java classes
- If types are omitted, they default to Object

Classes

- Scripts without a class declaration can be compiled with groovyc and produce a Java class of the same name
- To declare a method in a script file without writing a class, precede it with the keyword “def”
- These “loose” methods are generated to static methods
- All statements in the script are put in a run() method, a main is generated that calls run

Classes

- Can easily declare beans in Groovy
- If a property is private, then a private Java field is used
- If protected or public, a private field with getter/setter is used
- Can overload auto-created methods

Operator overloading

- Groovy supports operator overloading by mapping methods to operators

Operator	Method
a+b	a.plus(b)
a-b	a.minus(b)
a*b	a.multiply(b)
a/b	a.divide(b)
a++	a.next()
a--	a.previous()
a[b]	a.getAt(b)
a[b]=c	a.putAt(b,c)
a << b	a.leftShift(b)

Operator overloading

- Nulls are safely handled

Operator	Method
<code>a==b</code>	<code>a.equals(b)</code>
<code>a!=b</code>	<code>!a.equals(b)</code>
<code>a===b</code>	instance test
<code>a<=>b</code>	<code>a.compareTo(b)</code>
<code>a>b</code>	<code>a.compareTo(b)>0</code>
<code>a>=b</code>	<code>a.compareTo(b)>=0</code>
<code>a<b</code>	<code>a.compareTo(b)<0</code>
<code>a<=b</code>	<code>a.compareTo(b)<=0</code>

The Features

GroovyMarkup

- Native support for XML, HTML, Ant, Swing
- Uses a *Builder object to allow compact declaration of tree based structures
- Built in support for DOMBuilder, SAXBuilder, AntBuilder, SwingBuilder

GPath

- Use closures to allow XPath type navigation in beans

```
class Person {  
    first  
    last  
    email  
}
```

```
p1 = new Person(first:"tom")  
p2 = new Person(first:"john")  
p3 = new Person(first:"tom")  
l = [p1,p2,p3]  
if (l.any {it.first == "tom" }) {  
    println "Match"  
}
```

Different from Java

- `==` is `.equals()`, `===` is Java `==`
- semicolon and return are optional
- Classes can be referred to by name (e.g. `Driver`, not `Driver.class`)
- things are public by default
- classes in `java.lang`, `groovy.lang`, and `groovy.util` are automatically imported

Added in Groovy

- closures
- list and map syntax
- GPath and regex
- iteration and switch
- static and dynamic typing
- Strings with expressions
- helper methods
- simple bean syntax

Current Drawbacks

- Open bugs (still beta)
- Performance will need work (all reflection based)
- Error messages need work
- Little IDE support at this point
- Tool support for refactoring with Java
- Easy to write very confusing syntax

References

- Groovy home page - <http://groovy.codehaus.org/>
- An Intro to Groovy - <http://ociweb.com/jnb/jnbFeb2004.html>

Contact Info

- Tom Janofsky
- tom@tomjanofsky.com