Groovy
Scripting for Java

Ian Darwin, http://www.darwinsys.com/
Java Cookbook site: http://javacook.darwinsys.com/

Notices

This talk was last revised for presentation to the Toronto
Java Users' Group (http://www.jug.org) November 2, 2004

Why is Perl successful?

- Many Java people hate Perl
  Kitchen Sink, Rubbish
  “Purely Eclectic Rubbish Lister
- BUT: Perl is widely used
  - interpreted, no compilation
  - huge API available
  - regex in language
  - flexible, forgiving, unreadable(?)

Can Java Better This?

- Keep Java basic syntax
- Relax some syntax rules
- Add regex support in language
- Add many "missing" convenience methods
- Access to all (std, 3d party) Java API: classpath
- Compile to bytecode if needed
- ...

The Result?

- Groovy, the new Scripting language for Java

Why is Java successful?

- Very few of the ideas in Java 1.0 were new
  - Portable byte code: UCSD P-system
  - OO: C++, Smalltalk
  - Downloading: the web
  - etc.
    - OK, Gosling's bytecode verifier was patentable :-)
- Java was successful because it combined all the right features, in the right place, at the right time.

Groovy keeps most of the best of Java

- Write Once, Groove Anywhere(tm)
- Many rich APIs in JDK, many more available
  - Collections API
  - Networking: Java.net, commons HttpClient, ...

Other Scripting Languages for Java

- In the Java Cookbook I show Java working with Perl, Bean Scripting Framework (BSF), etc
- Perl, Python/Jython, Ruby, etc.
- There is also pNut, Jelly, and other Java scripting languages (not to mention JavaScript)
- Groovy, like Java, combines “all the right features, in the right place, at the right time”
  - Therefore, a good chance at success
- Groovy’s developers acknowledge Java, Ruby, Python and Smalltalk as their main sources

Plan for this talk

Introduction
Differences from Java
Syntax Details
Groovy Collections
Java and Groovy Together
Applications: Persistence and SQL
Applications: Servlets
Applications: XML
More Applications
Summary / More Information

Defining Groovy

- Groovy: “a new agile dynamic language for the JVM combining lots of great features from languages like Python, Ruby and Smalltalk”
- Invented by James Strachan and Bob McWhirter
- Designed to keep the Java momentum while making it *easier to get things done*.
- Interpreted, but can be compiled to bytecode when a speed boost is needed

Standardizing Groovy

- Groovy is being standardized under the Java Community Process - JSR 241.
- Sun voted in favor of this JSR!
- GLS not written yet, just beginning
- Meeting in London UK next week (Nov 11/12) for JSR Expert Group to finalize ambiguities, then GLS writing can begin in earnest

Getting In the Groove

- To get started, download Groovy from http://www.codehaus.org/
  - Groovy needs JDK1.4; will not run with 1.3
  - Groovy provides some 1.5 syntax (foreach)
- Unzip binary distribution, add its bin to your PATH setting
  - UNIX/Linux/MacOS: chmod +x bin/*

How To Run

- Several ways to run Groovy:
  - groovy - run groovy file
  - groovysh - semi-interactive shell
  - groovyConsole - Swing-based runner
  - groovyc - compile groovy to bytecode
  - ant tasks

Other ways of Groovin'

- Plug-ins available for most IDEs
  - Eclipse
    - only in source at present
    - unofficial binary at http://sourceforge.net/projects/nexb
      - Did not work for me!
  - Idea IntelliJ
  - JEdit
  - others
- See the plug-ins list on http://groovy.codehaus.org/

Bottom Line: Why?

- If Groovy is that different from “regular Java”, why clutter your mind with two slightly-different syntaxes?
  - You don’t have to, if Java makes you as productive as you think you can be :-)
  - If you like typing (or watching Eclipse type) boilerplate like

```
public class Foo {
  public static void main(String[] args) {
```

Why?

- In real life, you need every advantage you can get
- Groovy aims to be an “agile language” meaning it should fit well with agile methodologies
  - “Embrace Change”
  - rapid development due to less pickyness
  - great for “glue” between layers of components
- And just because it’s Groovy!

Groovy

Groovy looks really slack...

- To those with Java coding syntax deeply ingrained, Groovy looks seriously *anarchic*:
  - No main method needed to run
  - No “class” keyword needed to run
  - No “method definition” needed to run
  - No parenthesis needed where unambiguous
  - No semicolon required(!)

Groovy Slack?

- These are available, and pretty much the same as Java, but Groovy makes them optional:

println “Hello world”

“That which is not strictly required, is optional” --Ian

So what happens?

- Class with no class statement gets class name from filename
  - minus the .groovy, of course
  - filename must be valid Java class name
- Class with no main() gets a main that just calls run()
- Class with un-contained code gets it compiled into a run() method

Weak Typing

- To make things easier, Groovy does not *require* type declarations
- In the tradition of awk and Perl, a variable is created by assigning a value to it
- A variable *may* be assigned different types during the same script!

-
  ```
  badIdea = "It is not a good idea to mix types"
  println badIdea
  badIdea = Math.PI
  println badIdea
  ```

Everything's an Object

- Groovy is in one sense a more “pure OO language” than Java
  - no separate primitive types
- This sometimes surprises Java newbies:
  ```
  if (”foo”.equals(userInput)) {
      //...
  }
  ```
- This may surprise Groovy newbies:
  ```
  println(1.getClass().getName())
  ```
  prints “java.lang.Integer”

Closures

- A closure is rather like an inner class but (of course) shorter
  - See http://martinfowler.com/bliki/Closures.html for more fine-grained discussion
- Can refer to *non-final* variables from the block that creates the closure
- Optional dummy arguments before `l'
- list.each { item l println item }
- Default is one argument called "it" (Perl $_)
- list.each { println it }

More Closures

- Many GroovyJDK methods take closures
  - like list.each on the previous page
    - method of *all* Groovy objects, like Java's Object.toString()
- Closures simplify:

```
# without closures
x = "Once upon a time...".tokenize()
for (val in x) println val

# with closures
"Once upon a time...".tokenize().each { val | println val }

# with closures, default parameter name
"Once upon a time...".tokenize().each { println it }
```

Closures as Objects

- Everything in Groovy is an Object; even closures
- Can save reference to a Closure:

```
myClosure = { x, y | println "Location=(${x},${y})" }
points = [ 10:10, 20:20, 30:40 ]
points.each myClosure
```

- Can write method that returns a closure
  - Similar effect to Java 1.5 Templates!

Groovy Beans

- JavaBeans (aka POJOs) are useful, but tedious to write
- GroovyBeans are easier to write:

```
public final class MyBean {
	private String name;
	private String address;

	public final String getAddress() {
		return address;
	}
	public final void setAddress(String address) {
		this.address = address;
	}
	public final String getName() {
		return name;
	}
	public final void setName(String name) {
		this.name = name;
	}
}
```

```
class MyBean {
	String name;
	String address;
}

bob = new MyBean(name:"Bobbie Smith",
address:"123 Main St");
println "Hi ${bob.name},
we have your address as ${bob.address}.
Is this correct?"
```

Groovy

Operators

These overloaded operators can be redefined if you choose to (except identity)

Operator	**Method**
a == b	a.equals(b)
a != b	!a.equals(b)
a === b	a == b
a<=>b	a.compareTo(b)
a>b	a.compareTo(b)>0
similarly for >=, <, <=	

RegEx Support

- ~"..." expression creates a Pattern
  - expr = ~”pattern”
  - True if the pattern matches the expr anywhere
  - Similar to Perl
- Also ==~ (matches regex)
  - Requires exact match

```
if ("in a rush" =~ "rush")
    println "Hurry Up"
```

GroovyObject

- All Groovy-provided objects subclass GroovyObject
- Adds support for setProperty/getProperty
- Also support for MetaClass

Air on a GString

- Groovy Strings used to represent string literals
- Automatically flattened to String when passed to methods requiring String
- Provide interpolation of variables
  - println “The name of ${object} is ${object.name}”
- Strings in single quotes are not interpolated
  - println ‘Use ${object} here’
  - actually prints
  - Use ${object} here

Groovy

Collections

- Groovy has language support for lists and maps using [ ]
- Lists (based on ArrayList)
  - [val, val, ...]
- Maps (based on HashMap)
  - [key:val, key:val, ...]
- Additional useful methods in List:
  - sort, count, min, max, reverse, etc

Lists

```
// List (based on ArrayList)
myList = [ "Bush", "Kerry", "Nader", "Badnarik"]
myList.each { pol | println pol }
$ groovy listpol.groovy
Bush
Kerry
Nader
Badnarik
```

Maps

```
// Create and populate
myMap = ["Bush":"Republican", "Kerry":"Democratic",
      "Nader":"Reform", "Badnarik":"Libertarian"]

// Print one
cand = "Kerry"
println "Candidate ${cand} is from the ${myMap[cand]} Party"

// Print all
myMap.each {pol, party |
      println "Candidate for ${party} Party is ${pol}"
}

$ groovy mappol.groovy
Candidate Kerry is from the Democratic Party
Candidate for Republican Party is Bush
Candidate for Democratic Party is Kerry
Candidate for Libertarian Party is Badnarik
Candidate for Reform Party is Nader
```

Ranges

```
// A good idea from Ruby, ranges are n..m
// Range extends List

#validCardYears = 2004..2009
thisYear = Calendar.getInstance().get(Calendar.YEAR);
length   = 5
validCardYears = thisYear..(thisYear+length)

# Print years (e.g., print html <select> for card expiry)
validCardYears.each { println "Valid year: ${it}" }

# In ranges, -1 means last, -2 2nd last, etc.
lastValidCardYear = validCardYears[-1]
println "Last valid year is currently ${lastValidCardYear}"
```

Ranges Details

- Ranges can be inclusive (..) or exclusive (...)
- Exclusive range includes the first value but not the last value
  - 5..8 contains 5,6,7,8
  - 5...8 contains 5,6,7
- Ranges can be use with a foreach

```
for (i in 1..10) {
  println "Hello Number ${i}"
  if (i== 6) {
     println(”I am not a number, I am a Free Man!”)
    }
    }
```

Reminder on the Range

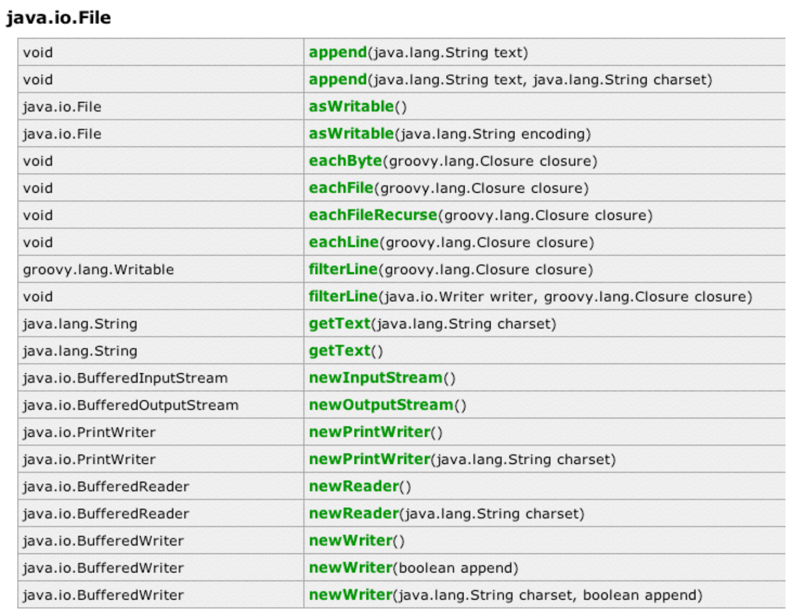
- These examples use integers, but any (sensible) type can be used
  - Integers are objects
    - Everything is an object!

Groovy

"GroovyJDK"

- Collective name for all the additions to the standard API
- Hundreds of convenience routines added to classes in java.lang, java.io, java.util
- Package names:
  - groovy.* is public API, like java.*
  - org.codehaus.groovy.* is private API, like sun.*
- JavaDoc available for GroovyJDK
  - online at http://groovy.codehaus.org/groovy-jdk.html
  - in the distribution ...

Small sample of GroovyJDK - java.io.File



java.io.File

void	**append**(java.lang.String text)
void	**append**(java.lang.String text, java.lang.String charset)
java.io.File	**asWritable**()
java.io.File	**asWritable**(java.lang.String encoding)
void	**eachByte**(groovy.lang.Closure closure)
void	**eachFile**(groovy.lang.Closure closure)
void	**eachFileRecurse**(groovy.lang.Closure closure)
void	**eachLine**(groovy.lang.Closure closure)
groovy.lang.Writable	**filterLine**(groovy.lang.Closure closure)
void	**filterLine**(java.io.Writer writer, groovy.lang.Closure closure)
java.lang.String	**getText**(java.lang.String charset)
java.lang.String	**getText**()
java.io.BufferedInputStream	**newInputStream**()
java.io.BufferedOutputStream	**newOutputStream**()
java.io.PrintWriter	**newPrintWriter**()
java.io.PrintWriter	**newPrintWriter**(java.lang.String charset)
java.io.BufferedReader	**newReader**()
java.io.BufferedReader	**newReader**(java.lang.String charset)
java.io.BufferedWriter	**newWriter**()
java.io.BufferedWriter	**newWriter**(boolean append)
java.io.BufferedWriter	**newWriter**(java.lang.String charset, boolean append)

GroovyJDK Usage

- Still create standard objects, e.g.,
  - new java.io.File(”.”). eachFileRecurse(...)
  - But you get the GroovyJDK behavior.

Calling Groovy From Java

```
import groovy.lang.*;

/** Call Groovy expressions from Java code */
public class JavaGroovy {
        public static void main(String[] args) throws Exception {
                Binding binding = new Binding();
                binding.setVariable("expr", new Integer(42));
                GroovyShell shell = new GroovyShell(binding);

                Object value = shell.evaluate("x = 123; return x + expr");
                System.out.println("Both should print true");
                System.out.println(value.equals(new Integer(165)));
                System.out.println(
                    binding.getVariable("x").equals(new Integer(123)));
        }
}
```

Running a Groovy Script from Java

```
import java.io.File;
import groovy.lang.GroovyObject;
import groovy.lang.GroovyClassLoader;

/** Call Groovy expressions from Java code */
public class JavaGroovyScript {
        public static void main(String[] args) throws Exception {
                new JavaGroovyScript().run();
        }
        void run() throws Exception {
                ClassLoader parent = getClass().getClassLoader();
                GroovyClassLoader loader = new GroovyClassLoader(parent);
                Class groovyClass = loader.parseClass(new File("hello.groovy"));

                // Now call the default method (run) on an instance
                GroovyObject groovyObject =
                   (GroovyObject) groovyClass.newInstance();
                Object[] calledArgs = {};
                groovyObject.invokeMethod("run", calledArgs);
        }
}
```

Calling Java from Groovy

- Compile to Java with groovyc
- Then run the class!
  - either as a main class
  - or via another Java class...

```
$ cat g.groovy
i = 42
println "Hello ${i}"
$ cat j.java
public class j {
        public static void main(String args[]) {
                new g().run();
        }
}
$ groovyc g.groovy
$ javac -classpath $GJAR:. j.java
$ java  -classpath $GJAR:$ASMJAR:. j
Hello 42
```

SwingBuilder

- SwingBuilder is one of the NodeBuilders
  - XML Builders in a few minutes
- SwingBuilder lets you create any J* component
- Use inline Map to specify parameters
- Can use any Swing methods on given components

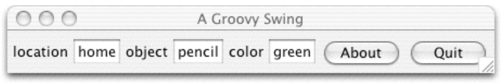
SwingBuilder Demo

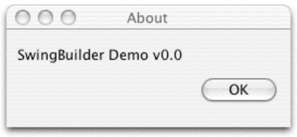
```
import groovy.swing.SwingBuilder;

theMap = ["color":"green", "object":"pencil", "location":"home"];

// create a JFrame with a label and text field for each key in the
Map
swing = new SwingBuilder();
frame = swing.frame(title:'A Groovy Swing', location:[240,240],
        defaultCloseOperation:
javax.swing.WindowConstants.EXIT_ON_CLOSE) {
        panel() {
                for (entry in theMap) {
                        label(text:entry.key)
                        textField(text:entry.value)
                }
```

```
                    button(text:'About', actionPerformed:{
                             pane =
                                 swing.optionPane(
                                    message:'SwingBuilder Demo v0.0')
                             dialog = pane.createDialog(null, 'About')
                             dialog.show()
                     })
                     button(text:'Quit', actionPerformed:{
                           System.exit(0) });
       }
}
frame.pack();
frame.show();
```





SwingBuilder Demo v0.0

Groovy

Groovy & OR Mapping Persistence

- Groovy treats objects as beans
- OR Mapping (JDO, Hibernate, etc) treat beans as persistable data
- An OR Mapping package called pBeans (http://pbeans.sourceforge.net/) is an “extreme” example of ease-of-use
  - Makes tables for you, inserts/updates data on demand - NO SQL mapping needed!
  - Very simple, possibly not as powerful as the others
- Consider a simple `Person` GroovyBean...

pBeans Example

```
ds = new GenericDataSource()
ds.setDriverClassName("org.postgresql.Driver")
ds.setUrl("jdbc:postgresql://localhost/website?
user=test&password=test")
store = new Store(dataSource)

newuser = new Person(name:"Bob Smith",
city:"Toronto", country:"ca")
store.insert(newuser)
foundUserPerson =
store.selectSingle(Person.class, "name", "Bob
Smith")
assert foundPerson.city == "Toronto"
```

GroovySQL

- Groovy provides its own SQL package on top of JDBC
- Class SQL provides connection pooling if a DataSource is used
  - In this example a Connection is provided
- sql.eachRow() takes a query string and a closure
  - Invokes the closure for each row, passing an object with all the columns for this row...

```
import groovy.sql.*;

connection = // get the connection
sql = new Sql(connection);

sql.eachRow("select firstName, lastName from customers") {
cust |
        println "Dear ${cust.firstName} ${cust.lastName}:"
        printRestOfFormLetter()
}

printRestOfFormLetter() {
        // ...
}
```

Groovy

Groovlets: Groovy Servlets

- Lots of Servlet code is mainly “glue” between the Web tier and an EIS back end
  - SQL or JDO or Hibernate or EJB or ...
- Groovy Servlet lets you run Groovy script in response to GET or POST
  - Forms parameters become objects!
  - Standard objects (request, response, out) also available
  - Can use for JSP-like things (”page-centric”)
  - Can do processing and forward to JSP (MVC)

Hello World Groovlet

```
out.println(<<<EOS
<html>
<p>Hello from Groovy
to ${request.remoteHost}
at ${new java.util.Date()}.
EOS
```

So far, not so different from a JSP?

Can do more

- All request parameters available as objects

Hello ${firstName} ${lastName} in ${city}

- Populate a bean, JDO/Hibernate object, DAO

bean = new Person();
bean.firstName = ${firstName}
// etc

Expect additional convenience methods to be added (populateAll, as JSP set property=”*”?)

Configuration

- Add groovy.jar and asm.jar to WEB-INF/lib
- Add entries in WEB-INF/web.xml:

```
<servlet>
   <servlet-name>Groovy</servlet-name>
   <servlet-class>groovy.servlet.GroovyServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>Groovy</servlet-name>
   <url-pattern>*.groovy</url-pattern>
</servlet-mapping>
```

Groovy

GroovyXML

- Several kinds of Builder
  - NodeBuilder is most general
  - SwingBuilder shown under Swing section
  - DomBuilder generates tree from a Document
  - SaxBuilder processes document and iires SAX events
  - MarkupBuilder generates HTML/XML markup...

```
import java.io.PrintWriter;
import groovy.xml.MarkupBuilder;

PrintWriter pw = new PrintWriter(System.out);
html = new MarkupBuilder(new PrintWriter(System.out));
html {
    head {
        title("HTML encoding with Groovy")
    }
    body {
        h1("HTML encoding with Groovy")
        p("This can be used instead of writing HTML by hand")
        p("Some more text")
    }
}
```

Groovy

More Groovy Scripts: fixid

```
# Make sure all the Java files in a collection have a CVS ID String.
# Don't apply to all files because it may damage binary files.
# $Id: fixid.groovy,v 1.2 2004/09/29 13:29:32 ian Exp $

new java.io.File(".").eachFileRecurse({file | if (file.name.endsWith(".java")) {
		old = file.getText()
		if (!old.contains("$Id")) {
			println "Fixing file ${file.name}"
			out = file.newPrintWriter()
			out.println("// Next line added by fixid script; should move into doc comment")
			out.println("// $I"+"d$")   // + to hide from CVS
			out.print(old)
			out.close()
		}
	}
})
```

So many things are simpler in Groovy

```
import java.io.File;

in = new File('foo.txt').readLines();
in.each { println(it) }
```

WordCount: UNIX wc(1) in Groovy

```
# could simplify with File.splitEachLine() but
# this way gets "chars" right.
filename=args[0]
chars=0; lines=0; words=0;
new java.io.File(filename).eachLine {
        chars+=it.length() + 1
        words+=it.tokenize().size();
        lines++;
}
println "\t${lines}\t${words}\t${chars}\t${filename}"
```

Groovy

Summary

- Groovy is an agile scripting language for Java
- Strachan has also called it a “native scripting language for Java” due to its close interaction and bytecode compiler
- Everything in Groovy is there *to make your life easier* in the Java environment

More to Groove On

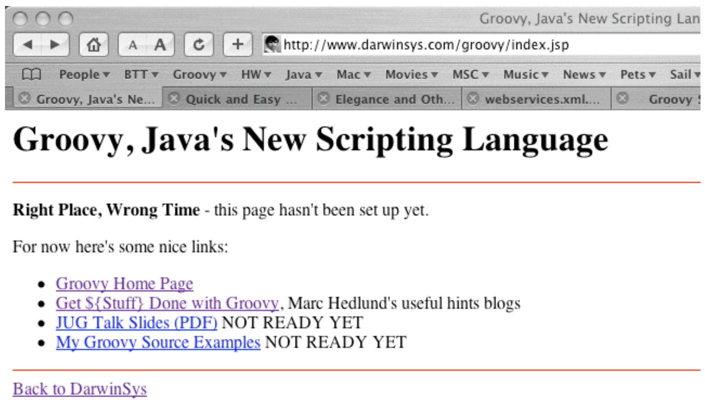
- There is much more to apply Groovy
  - Learn GroovyJDK methods - very powerful
  - GPath (list selection, like XPath, with Groovy's typical conciseness)
  - GroovySWT
  - XML RPC
  - And more -- see the web site.

Information Sources

- Google "Groovy Scripting Java"
- My article: http://www.onjava.com/pub/a/onjava/2004/09/29/groovy.html
- The main web site: http://codehaus.groovy.org/
- No books yet, but several publishers have Groovy books in progress

My Groovy Resource Page

- http://www.darwinsys.com/groovy
  - will contain slides, code examples, links, etc.
  - should be live in a few days.


Groovy, Java's New Scripting Lan
http://www.darwinsys.com/groovy/index.jsp
People BTT Groovy HW Java Mac Movies MSC Music News Pets Sail
Groovy, Java's Ne... Quick and Easy ... Elegance and Oth... webservices.xml... Groovy

Groovy, Java's New Scripting Language

Right Place, Wrong Time - this page hasn't been set up yet.

For now here's some nice links:

- Groovy Home Page
- Get ${Stuff} Done with Groovy, Marc Hedlund's useful hints blogs
- JUG Talk Slides (PDF) NOT READY YET
- My Groovy Source Examples NOT READY YET

Back to DarwinSys

Is Groovy Perfect?

- No!
- Omission of semi-colon makes parser complex
- As does omission of some parenthesis
- Many ambiguities and other issues to be dealt with at the London meeting
- Will Groovy be perfect after the meeting?
- There are also just plain bugs in the implementation

Getting Involved

- Mailing lists: user, developer, JSR
- Open CVS repository
  - language
  - some plug-ins (Eclipse) hosted here
  - all at codehaus.org
- Needed: coders, documenters, polite advocates, etc.
  - use groovy, find bugs, read code, send fixes!

Q & A

?