Groovy Primer

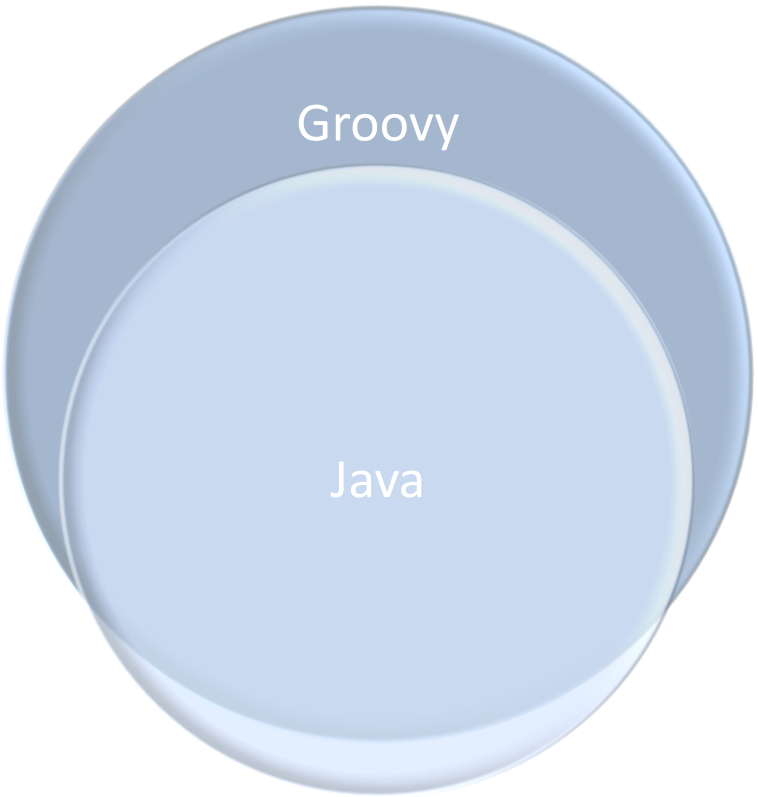
Chris Dail
Senior Software Architect
iWave Software LLC

http://chrisdail.com
Twitter: @chrisdail

What is Groovy?

- An agile dynamic language for the Java Platform
- Both dynamically and statically typed
- Functional programming influence
- Open source and community developed
  - Faster movement than Java
- Rapidly growing and gaining acceptance in Java communities

Groovy …



- Is a superset of Java
  - Makes interop easy
- Uses Java's Type system
- Everything is an Object
- Is Java +
  - syntactic sugar
  - dynamic typing
  - missing features (like closures)

Groovy Compared

- Scala
  - Statically Typed
  - Does not leverage Java's type system
- Clojure (Lisp dialect)
  - Purely functional programming
- Python (Jython) and Ruby (JRuby)
  - Dynamically Typed
  - Does not leverage Java's type system
- Groovy is Unique!

Language Features

- Closures, Categories and Builders
- Syntax for Lists, Maps and Ranges
- Java platform extensions
  - IO, String, Processes
- Regex Syntax
- Property access (Java Beans)
- Operator Overloading
- Metaprogramming and DSLs

Why Groovy?

- Write less, more expressive code
- Write code that is easier to read
- Write code faster
- Easy Java interoperability
  - Less painful context switching

Getting Started

- From Java to Groovy
  - Most Java Classes compile in Groovy simply with an extension change
- Example
  - Adapted from examples by Jim Weirich, Ted Leung and Paul King

Java to Groovy

```java
import java.util.List;
import java.util.ArrayList;
class JavaToGroovy {
    private List<String> filterLongerThan(List<String> strings, int length) {
        List<String> result = new ArrayList<String>();
        for (String s: strings) {
            if (s.length() <= length) {
                result.add(s);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        List<String> names = new ArrayList<String>();
        names.add("Ted");
        names.add("Fred");
        names.add("Jed");
        names.add("Ned");
        System.out.println(names);
        JavaToGroovy e = new JavaToGroovy();
        List<String> shortNames = e.filterLongerThan(names, 3);
        System.out.println(shortNames.size());
        for (String s: shortNames) {
            System.out.println(s);
        }
    }
}
```

JavaToGroovy.java -> JavaToGroovy.groovy

Java to Groovy

```
import java.util.List;
import java.util.ArrayList;
class JavaToGroovy {
    private List<String> filterLongerThan(List<String> strings, int length) {
        List<String> result = new ArrayList<String>();
        for (String s: strings) {
            if (s.length() <= length) {
                result.add(s);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        List<String> names = new ArrayList<String>();
        names.add("Ted");
        names.add("Fred");
        names.add("Jed");
        names.add("Ned");
        System.out.println(names);
        JavaToGroovy e = new JavaToGroovy();
        List<String> shortNames = e.filterLongerThan(names, 3);
        System.out.println(shortNames.size());
        for (String s: shortNames) {
            System.out.println(s);
        }
    }
}
```

Default Core Imports

Optional Semicolons

Alternate for each syntax
Predates Java 5

Java to Groovy (Take 2)

```
class JavaToGroovy {
    private List<String> filterLongerThan(List<String> strings, int length) {
        List<String> result = new ArrayList<String>()
        for (s in strings) {
            if (s.length() <= length) {
                result.add(s)
            }
        }
        return result
    }
    public static void main(String[] args) {
        List<String> names = new ArrayList<String>()
        names.add("Ted")
        names.add("Fred")
        names.add("Jed")
        names.add("Ned")
        System.out.println(names)
        JavaToGroovy e = new JavaToGroovy()
        List<String> shortNames = e.filterLongerThan(names, 3)
        System.out.println(shortNames.size())
        for (s in shortNames) {
            System.out.println(s)
        }
    }
}
```

Java to Groovy (Take 2)

```
class JavaToGroovy {
    private List<String> filterLongerThan(List<String> strings, int length) {
        List<String> result = new ArrayList<String>()
        for (s in strings) {
            if (s.length() <= length) {
                result.add(s)
            }
        }
        return result
    }
    public static void main(String[] args) {
        List<String> names = new ArrayList<String>()
        names.add("Ted")
        names.add("Fred")
        names.add("Jed")
        names.add("Ned")
        System.out.println(names)
        JavaToGroovy e = new JavaToGroovy()
        List<String> shortNames = e.filterLongerThan(names, 3)
        System.out.println(shortNames.size())
        for (s in shortNames) {
            System.out.println(s)
        }
    }
}
```

Dynamic Types

Scripts without class declaration or main

Java to Groovy (Take 3)

```
def filterLongerThan(strings, length) {
    List<String> result = new ArrayList<String>()
    for (s in strings) {
        if (s.length() <= length) {
            result.add(s)
        }
    }
    return result
}
List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
def shortNames = filterLongerThan(names, 3)
System.out.println(shortNames.size())
for (s in shortNames) {
    System.out.println(s)
}
```

Java to Groovy (Take 3)

```
def filterLongerThan(strings, length) {
    List<String> result = new ArrayList<String>()
    for (s in strings) {
        if (s.length() <= length) {
            result.add(s)
        }
    }
    return result
}
List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
def shortNames = filterLongerThan(names, 3)
System.out.println(shortNames.size())
for (s in shortNames) {
    System.out.println(s)
}
```

List finder syntax

Special List Notation

List Processing syntax

Java to Groovy (Take 4)

```
def filterLongerThan(strings, length) {
    return strings.findAll { it.size() <= length }
}

def names = ["Ted", "Fred", "Jed", "Ned"]

System.out.println(names)
def shortNames = filterLongerThan(names, 3)
System.out.println(shortNames.size())
shortNames.each {
    System.out.println(it)
}
```

Java to Groovy (Take 4)

```
def filterLongerThan(strings, length) {
    return strings.findAll { it.size() <= length }
}
def names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
def shortNames = filterLongerThan(names, 3)
System.out.println(shortNames.size())
shortNames.each {
    System.out.println(it)
}
```

Function not required

Optional Brackets

Common IO Syntax

Final Groovy

```
def names = ["Ted", "Fred", "Jed", "Ned"]
println names
def shortNames = names.findAll { it.size() <= 3 }
println shortNames.size()
shortNames.each { println it }
```

```
[Ted, Fred, Jed, Ned]
3
Ted
Fred
Jed
Ned
```

Comparison

```java
import java.util.List;
import java.util.ArrayList;
class JavaToGroovy {
    private List<String> filterLongerThan(List<String> strings, int length) {
        List<String> result = new ArrayList<String>();
        for (String s: strings) {
            if (s.length() <= length) {
                result.add(s);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        List<String> names = new ArrayList<String>();
        names.add("Ted");
        names.add("Fred");
        names.add("Jed");
        names.add("Ned");
        System.out.println(names);
        JavaToGroovy e = new JavaToGroovy();
        List<String> shortNames = e.filterLongerThan(names, 3);
        System.out.println(shortNames.size());
        for (String s: shortNames) {
            System.out.println(s);
        }
    }
}
```

```groovy
def names = ["Ted", "Fred", "Jed", "Ned"]
println names
def shortNames = names.findAll { it.size() <= 3 }
println shortNames.size()
shortNames.each { println it }
```

Java Beans

- Getter and Setter syntax
  - Common dotted notation (`bean.property`)
- Getters and Setters auto-generated
- Constructor syntax
  - Constructs bean then calls setters
  - `property1: value1, property2: value2`
- Last line automatic return

JavaBeans.groovy

```
class Purchase {
    def item
    int quantity
    int cost

    def getTotal() {
        quantity * cost
    }
}
def purchase = new Purchase(item: "software", cost: 5)
purchase.quantity = 4
println "Purchased '$purchase.item' for \$$purchase.total"
```

Read only property

Equivalent to:
Purchase p = new Purchase()
p.setItem("software")
p.setCost(5)

String templates

Closures

- Anonymous Functions, Code Blocks
- Allow Functional programming
- ‘Missing’ Java feature slated for Java 7
- Definition Syntax:

```
{param1, param2 ->
    // Code
}
```

- If no parameters specified, has single default parameter `it`

ClosureBasics.groovy

Closure with 2 arguments

```
def printSum = { a, b -> println "Sum: " + (a + b) }

printSum(5, 7)

def printDiff = { a, b -> println "Difference: " + (a - b) }

def callWithClosure(Closure closure) {
   print "Closure called from method, "
   closure(10, 5)
}

callWithClosure(printSum)
callWithClosure(printDiff)
callWithClosure {a, b ->
    println "Inline closure with $a and $b"
}
```

Calling a Closure

Method with closure parameter

Calling Method with Closure

Calling method with inline closure

NestedClosures.groovy

```
def errorHandling = { Closure c ->
    try {
        c()
    }
    catch (Exception e) {
        println "Closure handled error"
        e.printStackTrace()

        // Can throw as another type or handle some other way
        //throw new WrappedException(e)
    }
    finally {
        // cleanup here, close resources
    }
}

// Code with error handling
errorHandling {
    println new File("does_not_exist.txt").text
}
```

Error handling closure takes closure as a parameter

Call the nested closure

Common error handling

Cleanup resources

Calling the error handling closure

Strings

- Single or double quoted

```
println "with 'single' quotes" + 'with "double" quotes'
```

- GString - variable expansion

```
println "Name: $name"
```

- Multiline String

```
def xml = """<?xml version="1.0"?>
<root>
  <value/>
</root>"""
```

String Methods

Method	Description
"abc".reverse()	Reverses a string producing "cba"
"abc 1 2 3".split()	Splits a string on spaces producing [abc, 1, 2, 3] as an array
"1, 2, 3".tokenize(',')	Tokenizes a string on comma producing [1, 2, 3]
string.eachLine {c}	Calls the closure for each line in string
"abc"[2]	Returns the character at position 2 of the string, 'c'
"testing"[1..3]	Returns the substring between positions 1 and 3, 'est'

Range Operator

Lists and Collections

- List Syntax

```
def list = [1, 2, 3]
def emptyList = []
```

- Collection Methods

```
list.each { println it }
list.collect { it.action }
list.any { it == 2 }
list.every { it != 2 }
list.sort()
list.join(", ")
```

- Overloaded Operators

```
list1 == list2
```

CollectionAny.groovy

```
def list = ["abc", "def", "ghi", "jkl"]

// Java way
boolean found = false;
for (String str: list) {
    if (str.contains("z")) {
        found = true;
        break;
    }
}
if (!found) {
    println "Not Found (Java)"
}

// Groovy way
if (!list.any {it.contains("z")}) {
    println "Not Found (Groovy)"
}
```

Determine if no members of the list contain a 'z'

Maps

- Map Syntax

```
def map = [a: 1, b: 2]
assert map['a'] == 1 && map.b == 2
```

- Methods

```
map.each { k, v -> println "$k=$v" }
map.collect { k, v -> ... }
```

```
map = [1:'a', 2:'b', 3:'c']
def string = map.collect { k, v -> "$k:$v" }.join(', ')
assert string == '1:a, 2:b, 3:c'
```

MapsAsDynamicBeans.groovy

```
class Person {
    String name
    String address
}

def usePerson(p) {
    p.name = "bob"
    p.address = "123 fake street"
    println "Person (${p.getClass()}) '$p.name' lives at: $p.address"
}

usePerson(new Person())
usePerson([:])

def person = [:]
person.name = "Chris"
person.address = "Planet Earth"
person.phone = "555-5326"
println person
```

Standard Java Bean

Code that uses bean

Using Java Bean or Map without modification

Maps are dynamic and extendable

```
Person (class Person) 'bob' lives at: 123 fake street
Person (class java.util.LinkedHashMap) 'bob' lives at: 123 fake street
[name:Chris, address:Planet Earth, phone:555-5326]
```

IO Processing

Methods	Description
file.append(text/bytes/stream)	Appends the text, byte array or inputstream to the file
file.deleteDir()	Recursively deletes all files and subdirectories
dir.eachDir { ... }	Invokes the closure for each subdirectory of this directory
dir.eachFile { ... }	Invokes the closure for each files under the directory
dir.eachFileRecurse { ... }	Invokes the closure for each descendant file in the directory
file.text	Returns the entire contents of the file as a String
file << text	Writes the text to the file
file.withInputStream { ... }	Creates a new InputStream for reading the file and passes it to the closure.
file.eachLine { ... }	Iterates through the file passing each line to the closure.

LineCount.groovy

```
def directories = ["e:\\code\\adapters", "e:\\code\\automator"]
def exclusions = ["com\\chrisdail\\excluded", "src\\test"] as Set
def inclusions = ["com\\chrisdail", "com\\another"] as Set

directories.each {root->
    def lineCount = 0
    def significantCount = 0

    new File(root).eachDirRecurse {dir ->
        // If no inclusion is present, return
        if (!inclusions.any {dir.absolutePath.contains(it)}) {
            return
        }

        // If any exclusion is present, return
        if (exclusions.any {dir.absolutePath.contains(it)}) {
            return
        }

        dir.eachFileMatch(~/.*\.(groovy|java|g|xml|properties)/) {file ->
            file.eachLine {
                def line = it.trim()
                if (line != "") {
                    lineCount++
                    if (line.size() > 2 && !line.startsWith("//") && !line.startsWith("/*")
                            && !line.startsWith("*") && !line.startsWith("import")) {
                        significantCount++
                    }
                }
            }
        }
    }
    println "Lines in '$root': $significantCount / $lineCount"
}
```

Finds each file with a code extension

Count non-empty lines

Significant lines only count 'real' code

Operators

- Elvis Operator (short ternary) ?:

```
a != null ? a : b
a ?: b
```

- Null-Safe traversal ?.

```
a == null ? null : a.someMethod()
a?.someMethod()
```

- Collection Spread *.

```
parent.collect{ child -> child?.action }
parent*.action
```

- **as** operator

```
[1, 2, 3, 4, 4] as Set
```

- Regex Find (=~) and Match (==~)

Operator Overloading

Operator	Method Called	Operator	Method Called
`a + b`	`a.plus(b)`	`a++`	`a.next()`
`a - b`	`a.minus(b)`	`a--`	`a.previous()`
`a * b`	`a.multiply(b)`	`a[b]`	`a.getAt(b)`
`a / b`	`a.div(b)`	`a[b] = c`	`a.putAt(b, c)`
`a % b`	`a.mod(b)`	`a << b`	`a.leftShift(b)`
`a | b`	`a.or(b)`	`a >> b`	`a.rightShift(b)`
`a & b`	`a.and(b)`	`switch (a) { case(b) : }`	`b.isCase(a)`

Operator Overloading

Operator	Method Called
`a == b`	`a.equals(b) or a.compareTo(b) == 0`
`a != b`	`!a.equals(b)`
`a <=> b`	`a.compareTo(b)`
`a > b`	`a.compareTo(b) > 0`
`a >= b`	`a.compareTo(b) >= 0`
`a < b`	`a.compareTo(b) < 0`
`a <= b`	`a.compareTo(b) <= 0`

Dynamic Invocation

- Dynamically calling a method

```
object."$method"(args)
```

- Dynamically getting a property

```
object."$property"
```

Categories

- Add methods to existing classes
- Implementation
  - Class with all static methods (can be Java)
  - First parameter is class to add method to
- Built in categories
  - DOMCategory - Working with Java DOM
  - TimeCategory - Working with Dates
- Mechanism groovy used to extend Java

CategoriesDate.groovy

```
import javax.xml.datatype.DatatypeFactory

class XMLDate {
    static String toXmlFormat(Date date) {
        def c = Calendar.instance
        c.time = date
        DatatypeFactory.newInstance().newXMLGregorianCalendar(c).toXMLFormat()
    }
}

use (XMLDate) {
    println new Date().toXmlFormat()
}
```

Category Definition: Static method

toXmlFormat() will be available to Date

Defines usage scope for the category

Metaprogramming

- What makes Groovy really 'dynamic'
- Meta Class
  - Every Class in Groovy has a MetaClass `class.metaClass`
  - Can have per-instance MetaClass `obj.metaClass`
  - Allows adding, overriding methods and properties
- Mixins
  - At runtime using `class.mixin category`
- Special Methods (interception without AOP)

  `invokeMethod`, `setProperty` and `getProperty`

  `methodMissing` and `propertyMissing`

BuilderIntro.groovy

```
class XmlBuilder {
   def out
   XmlBuilder(out) { this.out = out }
   def invokeMethod(String name, args) {
       out << "<$name>"
       if(args[0] instanceof Closure) {
            args[0].delegate = this
            args[0].call()
       }
       else {
           out << args[0].toString()
       }
       out << "</$name>"
   }
}
def xml = new XmlBuilder(new StringBuffer())
xml.html {
    head {
        title "Hello World"
    }
    body {
        p "Welcome!"
    }
}
```

Builder uses 'invokeMethod' to turn method calls into a DSL

Closures are invoked using this builder

HTML Text node

DSL Builder for XML

```
<html>
  <head><title>Hello World</title></head>
  <body><p>Welcome!</p></body>
</html>
```

Builders

- Common framework for building tree structures
  - `BuilderSupport` class to build your own
- Built-in builders
  - Ant, Swing, XML, Spring, HTTP

Invoking Processes

- Executing processes

  `string.execute()`

- Process Methods
  - `proc.text` - Returns the output of a process
  - `proc1 | proc2` - Pipes the output of proc1 to proc2
  - `waitForOrKill(millis)` - Waits for a period of time. If the process does not terminate by the time, it is terminated.
  - `consumeProcessOutput()` - Consumes the output of a process

Subversion Changlog Example

- Generate a Changelog using Subversion and Groovy
- Subversion Command Line API

```
svn log -r HEAD:1438 -g --xml
```

- Sample XML

```
<?xml version="1.0"?>
<log>
    <logentry revision="1547">
        <author>cdail</author>
        <date>2010-02-16T17:36:19.021297Z</date>
        <msg>Bug 123: Fixed some bug</msg>
        <!-- Merges contain nested log entries -->
        <logentry><!-- ... --></logentry>
    </logentry>
</log>
```

SvnLog.groovy

```
def handleEntry
handleEntry = {entry->
    def message = entry.msg.text()
    if (message.size() >= 3 && message[0..2].equalsIgnoreCase("bug")) {
        println " * $message (${entry.@revision})"
    }
    entry.logentry.each(handleEntry)
}

def proc = "svn log -r HEAD:1438 -g --xml".execute(null,
    new File("/path/to/working/directory"))
new XmlParser().parseText(proc.text).logentry.each(handleEntry)
println "done"
```

Closure is recursive and needs a reference to itself

Refer to data in the actual XML tree

DSL For processing XML

```
* Bug 123: Fixed some bug (1554)
* Bug 126: Some other issue (1588)
* Bug 322: Fixed the thing (1600)
```

SQL - QuartzTriggers.groovy

```
import groovy.sql.Sql

def sql = Sql.newInstance(
    "jdbc:derby://localhost:1527/db",
    "usr", "usr", "org.apache.derby.jdbc.ClientDriver")

def time = System.currentTimeMillis() - 45000l

sql.eachRow("""select * from QRTZ_TRIGGERS where NEXT_FIRE_TIME < ? AND
        TRIGGER_STATE != 'PAUSED'""", [time]) {
    println "Trigger ${it.TRIGGER_NAME} appears to be hung."
    println it
}
println "done"
```

Groovy SQL Wrapper

Query quartz triggers that appear to be hung

Each row returned calls the closure

Property style access for row columns

Tooling Support

- NetBeans, IntelliJ and Eclipse Plugins
  - Dev Environment. Eclipse +
    - SVN, Maven, Spring and Greclipse
    - Doesn’t always play nice with Maven
- Building
  - Maven – Gmaven Plugin
  - Ant – Cross compiler tasks
  - Gradle - Groovy based build system
- Groovy Console

Where to start Grooving?

- Unit testing
- Utility Scripts (line counting, svn changelog, POM update)
- Filtering, scripting, rules language
  - No custom proprietary syntax
- AOP Alternative (intercepting, proxying)
- New Code

Groovy Performance

- Yes, it is slower than Java
  - Slow parts can be writing in Java
- Slow Number Crunching
  - Due to object creation overhead
- Always getting faster
  - Groovy 1.5 vs 1.6 vs 1.7
  - Java 7's InvokeDynamic bytecode (JSR-292)
- Fast Enough

Groovy Projects

- Grails – CoC Web Framework
  - Obviously similar to Rails.
  - Built on proven Java Tech (JPA)
- Griffon – CoC for Swing
- Grape – Dependency Management
  - Think ivy directly from code (@Grab)
- Gradle – Groovy based build system
- Gaelyk – Lightweight toolkit for GAE

Resources

- Groovy – http://groovy.codehaus.org
- Web Console - http://groovyconsole.appspot.com/
- Groovy Documentation – http://groovy.codehaus.org/Documentation
  - GAPI - http://groovy.codehaus.org/gapi/
  - GDK - http://groovy.codehaus.org/groovy-jdk/
- Greclipse - http://groovy.codehaus.org/Eclipse+Plugin
- Groovy in Action (Book)

Questions