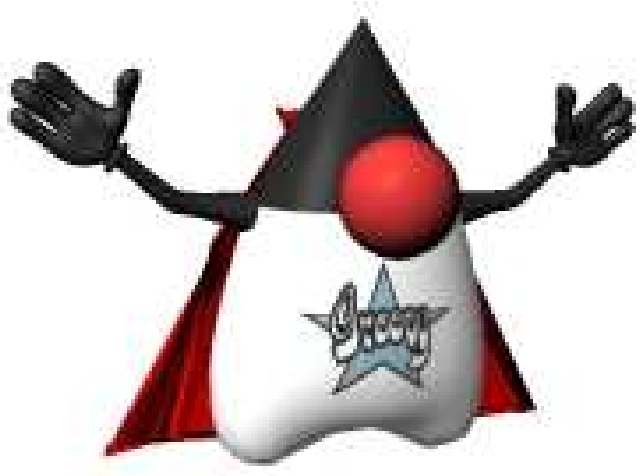
March 26, 2008

Groovy Magic - How Java Got Its Groove On

By Dan Sline



JPMorganChase

Agenda

- What is Groovy?
- Groovy Fundamentals
- Integrating Groovy and Java
- Putting it all together
- About JPMorganChase

What is Groovy

- Groovy is started by James Strachan in early 2003
- JSR-241
- Groovy Project Initiated in Sept. 2003 on the Codehaus website
- Groovy extends many of the Java classes to provide extra functionality that would commonly be in helper classes.

Groovy Fundamentals

- Running Groovy
- Syntax Differences between Groovy and Java
- Standard Input and Output
- Variable Declarations
- Method Declarations
- Closures
- Conditional Logic Processing
- Loop Processing
- File and Directory Handling
- XML
- SQL

Running Groovy

- Groovy code can be run in several ways:
  - Groovy Shell (groovysh.bat)
  - Groovy Console (groovyConsole.bat)
  - Groovy <fileName>
    - You can also run individual commands with groovy –e <statement> (for example groovy –e “println ‘Hello World’”.
  - Groovyc. Note: Groovyc is Groovy’s equivalent of a compiler, it will generate a Java class file to use if you are integrating into Java Code. Also, the Groovyc command also includes an option to compile the Java Classes at the same time as the Groovy Classes.
- Eclipse Plugin
- IntelliJ Plugin

Syntax Differences between Groovy and Java

- Java Statement
  - System.out.println(“I am here”);
- Groovy Statement
  - Println “I am Here”
    - Note System.out is optional
- Semi-colons are optional to complete a line unless you are nesting multiple statements on a line
  - for example: **println** "Hello"; **println** "World"
- Try/Catch Blocks are typically not required in Groovy

Differences (cont)

- Groovy supports, but does not use Java primitives by default. It will use Wrapper Objects instead.
- Groovy provides extra functionality to the Java Types.
- Most of this new functionality removes the need for custom built helper classes (like StringUtils and DateUtils)
- On String we have methods such as: reverse, toLong, toInteger, tokenize
- On Date we have methods such as minus and plus (which each add a number of days) as well as previous and next methods.
- Additionally Groovy has a method on List to reverse the contents of the list.

Differences - Groovy Objects

In the previous slide we mentioned that Groovy uses the Wrapper objects instead of their associated primitives.

In Java to add 1 to a Long, you needed to do the following:

```
long value = Long.longValue();
value += 1;
Long long2 = new Long(value);
```

- In Groovy this is much simpler:

```
def value = 3L
value += 1
println value
value = value.plus(1)
println value
```

The end result will be value = 5

Note: if you look at the class, it will be a java.lang.Long

DifferencesEx
DynamicTypeingEx

Differences – Groovy Objects (cont)

- Groovy Objects such as Longs also provide upto and downto functionality to quickly process a series of numbers
  - For example:
  - **def** upToEx = 3L
  - upToEx.**upto**(5) { **println** it}
    - Prints: 3,4,5 (each on their separate line)
- Date provides functionality such as plus and minus to add days to the current date. This is especially helpful if you are crossing over into time changes.
- One important thing to note, is that BigDecimal will be used unless you force your value to a Float or Double.
- You can specify strings within triple quotes “”” or ‘’’ to include line breaks instead of using the “+” operator to continue the string on the next line in Java.

DateTest

Differences (cont)

Bean like classes do not require the creation of getter and setter methods (provided no access modifiers are added to the variable name). Note: If you declare the variable as private, no getter and setter methods will be created.

```
class BeanExample {

    String name

}
```

In referencing code:

```
def beanEx = new BeanExample()

beanEx.setName("Hello")

println beanEx.getName();
```

GroovyBean

BeanEx

Standard Input and Output

To read information from the console, the following code is needed in Java:

```
BufferedReader inBuff = new BufferedReader( new
    InputStreamReader(System.in));

System.out.println("Input:");

System.out.flush();

try {
        String value = inBuff.readLine();
        System.out.println("value:" + value);
        // -- do something here

} catch (IOException e) {
        e.printStackTrace();

}
```

Groovy only requires a System.in.readLines() call (press Ctrl-z to exit if you are using Windows).

- def value = System.in.readLines()
- Note: System.in.readLine() has been depreciated

ReadLineExample

Standard Input and Output (cont).

- As we discussed a couple slides ago, Java requires the following code to output text:
  - System.out.println(<text>);
- Groovy just requires a println <text>
  - Note: parentheses are not required.
  - Also, as is mentioned before a semi colon is only required when you want to put multiple statements on a line.

Variable Declarations

Java Style:

- String stringVar;

Groovy Style:

- Explicit Typing is not required:
  - E.g def varName = “Value”;
  - Can change type dynamically
  - Like Java, however, you must initialize a variable before you can use it, otherwise you may get a NullPointerException
  - Must either have a “def” keyword or a access modifier (public, protected, private, and default)
  - According to the Groovy website, Groovy processes dynamic typing faster than static typing. (http://docs.codehaus.org/display/GROOVY/Runtime+vs+Compile+time%2C+Static+vs+Dynamic)

Variable Declarations (cont)

Java variable types are supported

- Groovy adds a new type of String call GString
- GString adds support to embed variable names (or any valid Groovy Expression) into the String (like Perl or JSP EL does):
- **def** gStringValue = 101
- GString gs = "Hello $gStringValue"
- **println** gs
- GStrings are also used in SQL
- A GString can be converted back to a String by using the toString() method

Variable Declarations - Collections

Collections Include:

- Lists
- Ranges
- Maps

Collections also provide the following methods:

- any – returns a true if the any of the values in the collection contain the value
- every – returns true if all of the items meet the criteria
- find – finds the first occurrence of the desired information
- findAll – returns all of the occurrences for the desired information
- collect - performs the closure on the collection and returns the adjusted collection.

Variable Declarations – Lists and Ranges

Lists

- Lists are defined like the following:

  def arrayOne = [1,2,3,4]

  Or def arrayOne = [] for an empty array

  The add method can be used to add to an array

  Extra functionality has been added to reverse, flatten, sort, and add up the values in the list

  Note: You still need to think about performance of a List vs. Map if you are processing large amounts of data.

- Ranges

  def range = 1..4

  def range = 10..<15

  def range = ‘e’ .. ‘m’

  ListEx

Variable Declarations (cont)

Maps

- Can create map items on instantiation or use a [:] to create an empty map
- Can add to a map with <mapVar>[<key>] = <value>
- Can mix types inside a map definition
- You can use any data type as a key, however if you have a negative number, it must be in parenthesis
- You can retrieve the value from a map in several ways
  - Closure
  - mapVar[<key]
  - mapVar.getAt(<key>)
  - mapVar.get(<key>)

Maps (cont)

In Java to process through a Map and print out the values the following code is needed:

```
Iterator iter = myMap.keySet().iterator();

Object key;

Object value;

while(iter.hasNext()) {

key = iter.next();

value = myMap.get(key);

System.out.println("key:" + key + ":value:" + value);

}
```

In Groovy, all you need is:

```
mapEx.each{ key, value -> println "$key : $value"}
```

JavaMap

MapEx

Method Declarations

Parameter and Return types are not required

Return statement is not always required. If a return is not explicitly called, the last value is used as the return value. In the example below, “Hello go” is returned

```
public go() {
        def v = "Hello go"
}
```

Closures

"A closure in Groovy is an anonymous chunk of code that may take arguments, return a value, and reference and use variables declared in its surrounding scope. In many ways it resembles anonymous inner classes in Java, and closures are often used in Groovy in the same way that Java developers use anonymous inner classes. However, Groovy closures are much more powerful than anonymous inner classes, and far more convenient to specify and use. "

- From http://groovy.codehaus.org/Closures

Closures (cont)

- If you don't specify an input variable "it" is used.
- They always return a value. The return value is the last value of the closure if an explicit return is not used.
- Closures can be nested, however you need to use different input variables
- However, the only way to return from the method while inside of a closure is to throw an exception.

For example:

- def closure_ex = [1,2,3,4]
- closure_ex.each{println it}

- **def** mapDemo = [1:'a', 2:'b', 3:'c']
- mapDemo.**each**{ key, value -> **println** "${key} ${value}"}

Closures (cont)

Closures can also be defined as a variable:

```
def addOne = {it + 1}

def collectVal = [1,2,3,4].collect(addOne)

collectVal.each{ println "newval ${it}" }
```

- The following output is returned:

```
newval 2
newval 3
newval 4
newval 5
```

ClosureEx

Conditional Logic Processing

Standard Java “If-Else” processing

Groovy has expanded switch statements to handle Strings

```
switch(<var>){
        case <stringvalue> : <do something>
        case 1..3 : <do something>
         case [10,11,12,’yes’] : <do something>
        default : <do something>
        }
```

Note: Don’t forget about “break” statements if you want it to break out of the loop

SwitchEx

Loop processing

Conventional Java while loops are supported

Conventional Java for loops are not supported.

- Use the following syntax instead:
- for(i in 0..9) {<do something>}
- for (i in 0..<9) {<do something}
- def map = [‘a’:1, ‘b’:2, ‘c’:3]
- for (v in map) { <do something}

ForEx

File and Directory Processing

Directory Processing

File Processing

Directory Processing

In Java to get a list of Files you need to the following:

```
import java.io.File;

public class FileList {
        public static void main(String[] args) {
            File file = new File(args[0]);
                if(file.isDirectory()) {
                        File files[] = file.listFiles();
                        for(int i = 0; i < files.length; i++) {
                            System.out.println(files[i].getAbsolutePath());
                        }
                }
        }

}
```

In Groovy:

```
new File(<some directory>).eachFile{ file -> println file.name }
```

- Recursively processing a directory tree is in a single statement
- `new File("C:\\files\\chase\\presentations\\output").eachFileRecurse{ fileInfo -> println fileInfo}`

GroovyFileExample

File Processing

File Processing in Java is worse

```
import java.io.*;
public class FileReading {
        public static void main(String args[]) {
        try {
        FileReader fis = new FileReader("helloworld.txt");
        BufferedReader reader = new BufferedReader(fis);
        String line = reader.readLine();
        while(line != null) {
        System.out.println(line);
        line = reader.readLine();}
        reader.close();
        fis.close();} catch (Exception e) { // do something here }
        } }
```

Groovy is easy

```
new File(<file>). each { myline -> println myline}
```

Writing to a File

Groovy provides a quick and easy way to write to a file

Write method will overwrite any existing file.

- write method will override any existing contents
- append will write to the end of the file, but everytime an append is called (unless it is in a closure), it will open up the file and append to the end of the file which is slow for large files.
- Better approach which keeps the file open while you write to the file (and handles the flush() and close() for you):

```
file.withWriterAppend { writer ->

        array.each{ curLine ->
          writer << curLine + "\r\n“ } }
```

FileEx

XML Processing

MarkupBuilder is used to build XML files

Note: in the example below, custom-xml is in single quotes. If you have a '-' in the name, it must be in single quotes

```
static void main(args) {
        def xml = new MarkupBuilder()
        xml.'custom-xml'() {
            record(att1:'name',  'tag value')
        }

 }
```

Generates:

```
<custom-xml>

  <record att1='name'>tag value</record>

</custom-xml>
```

XML Processing

Reading Files is a snap:

```
<main>
   <level>
      <java>1.5</java>
   </level>
</main>
```

```
class XMLReader {

  static void main(args) {

  def xmlFile = new
     File("C:\\files\\eclipse\\workspace\\groovytest3\\misc\\java.xml")

     def xmlTest = new groovy.util.XmlParser().parse(xmlFile)
              xmlTest.level.each{ level ->
              println(level.java)

     }

  }

}
```

SQL Processing

Java Style

```
Class.forName(<driver>);

Connection conn = null;

Statement stmt = null;

try {
        conn =DriverManager.getConnection(<connect params>);
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("select * from books");
        While(rs.next()) {
            String ISBN = rs.getString("ISBN");
            …
        }

} catch (SqlException e) { <do something here>}

finally{
        try{<Close statement in finally block>} catch (SqlException e) {<catch
            logic>}
        try{<Close connection in finally block>} catch (SqlException e) {<catch
            logic>

}
```

SQL Processing (cont)

Groovy Way

```
def sql = Sql.newInstance("jdbc:mysql://localhost:3306/GroovyDB", "dan",
      "dan", "com.mysql.jdbc.Driver");

 sql.eachRow("Select ISBN, CATEGORY from books") { row ->

  println("${row.ISBN} : ${row.CATEGORY}")

}
```

Which way would you rather do?

Running Shell Commands

- Groovy Gives you an easy way to run Shell Commands:
- For example the following will output the results of the command:

```
println "cmd /c dir
    c:\\files\\chase\\presentations\\output".execute().text
```

Putting It All Together

- Scenario 1:
  - Comparing two CSV Files
- Scenario 2:

Example is a book publisher which has a set of a number of books. We need to create the books and category into the database and have them generate an XML, CSV, and Text File to send to various distributors.

The next step will be to have the three files read by different programs and processed to simulate different distributors getting data.

CompareCSV

Publisher is the main driver for the data

XMLDistributor

ReadTextBase is the main reader that the other two sources read from

ReadTabDelim reads in the tab delim file

ReadDelim does another amount of delimination

References

Main Groovy Website:

- Groovy.codehaus.org
- Groovy GDK can be found at: http://groovy.codehaus.org/groovy-jdk.html

Other Groovy Sites

- About Groovy : http://www.aboutgroovy.com/
- IBM's site has some good Groovy information on it as well: http://www-128.ibm.com/developerworks/views/java/libraryview.jsp?search_by=practically+groovy:

Note: The modified Duke Images were edited by Paul King, one of the co-authors of Groovy In Action.

References (Cont.)

Books

- Groovy In Action (ISBN: 1-932394-84-2)
  - By Dirk Koenig, Andrew Glover, Paul King, and Guillaume Laforge
    - Note: Guillaume Laforge recently mentioned that Groovy In Action had the most formal up to date Groovy Language Specification (source: groovy-user forum 3/16/2008)
- Groovy Recipes (ISBN: 0-9787392-9-9)
  - By Scott Davis
- Programming Groovy (due in print 4/15/2008)
  - By Venkat Subramaniam

Note: The modified Duke Images were edited by Paul King, one of the co-authors of Groovy In Action.

Who We Are - JPMorgan

JP Morgan Chase & Co. is one of the world's largest and most respected financial services firms headquartered in New York.

Mission

- To be the best financial services company in the world.

Clients

- 90 million customers worldwide including some of the most prominent corporations, institutions and governments.

Scope

- Operations in more than 50 countries.

Assets

- More than $1.3 trillion.

Earnings

- For JPMorgan Chase in fiscal 2006, the company reported revenue of $61.4 billion and net income of $13.6 billion.
- $18.3 billion in 2006 in total revenues was produced by the Investment Bank.

Training, Career, and Leadership Opportunities

We seek individuals who will take our firm and our industry to a new level. We need people who are:

- *Intellectually Curious*
- *Innovative*
- *Commercially Focused*
- *Biased Toward Action*
- *Full of Energy*
- *Team Players*
- *Leaders*

JPMorganChase

lbtechnology

Q & A