Groovy

An Object Oriented Dynamic Language for the JVM

A Java Program

```java
import java.util.*;
class Erase {
   public static void main(String[] args) {
      List l = new ArrayList();
      l.add("Ted");
      l.add("Fred");
      l.add("Jed");
      l.add("Ned");
      System.out.println(l);
      Erase e = new Erase();
      List r = e.filterLongerThan(l, 3);
      System.out.println(r.size());
      for (Iterator i = r.iterator(); i.hasNext(); ) {
         System.out.println(i.next());
      }
   }
   public List filterLongerThan(List l, int length) {
      List result = new ArrayList();
      for (Iterator i = l.iterator(); i.hasNext(); ) {
           String entry = (String) i.next();
           if (entry.length() < length+1) {
              result.add(entry);
           }
      }
      return result;
   }
```

Groovy 1

```
import java.util.ArrayList
class Erase {
   public static void main(String[] args) {
      List l = new ArrayList()
      l.add("Ted")
      l.add("Fred")
      l.add("Jed")
      l.add("Ned")
      System.out.println(l)
      Erase e = new Erase();
      List r = e.filterLongerThan(l, 3)
      System.out.println(r.size())
      for (i in r) {
          System.out.println(i)
      }
   }
   public List filterLongerThan(List l, int length) {
      List result = new ArrayList()
      for (entry in l) {
           if (entry.length() < length+1) {
              result.add(entry)
           }
      }
      return result
   }
}
```

Groovy 2

```
import java.util.ArrayList
class Erase {
   public static void main(args) {
      l = new ArrayList()
      l.add("Ted")
      l.add("Fred")
      l.add("Jed")
      l.add("Ned")
      System.out.println(l)
      e = new Erase();
      r = e.filterLongerThan(l, 3)
      System.out.println(r.size())
      for (i in r) {
         System.out.println(i)
      }
   }
   public filterLongerThan(l, length) {
      result = new ArrayList()
      for (entry in l) {
           if (entry.length() < length+1) {
              result.add(entry)
           }
      }
      return result
   }
}
```

Groovy 3

```
import java.util.ArrayList
class Erase {
   public static void main(args) {
      l = [ "Ted", "Fred", "Jed", "Ned" ]
      System.out.println(l)

      e = new Erase();
      r = e.filterLongerThan(l, 3)
      System.out.println(r.size())
      for (i in r) {
          System.out.println(i)
      }
   }

   public filterLongerThan(l, length) {
      result = new ArrayList()
      for (entry in l) {
           if (entry.length() < length+1) {
              result.add(entry)
           }
      }
      return result
   }
}
```

Groovy 4

```
import java.util.ArrayList
class Erase {
   public static void main(args) {
      l = [ "Ted", "Fred", "Jed", "Ned" ]
      System.out.println(l)

      e = new Erase();
      r = e.filterLongerThan(l, 3)
      System.out.println(r.size())
      r.each { println it }
   }

   public filterLongerThan(l, length) {
      result = new ArrayList()
      result = l.findAll { entry | entry.length() < length+1 }
      return result
   }
}
```

Groovy 5

```
l = ["Ted", "Fred", "Jed", "Ned"]
println l

length = 3
r = l.findAll { e | e.length() < length+1 }
println r.size()
r.each { println it }
```

Typed Groovy

```
List l = ["Ted", "Fred", "Jed", "Ned"]
println l

Integer length = 3
List r = l.findAll {| String e | e.length() < length+1 }
println r.size()
List r = r.each { println it }
```

Tim Bray 3/15/2004

- In fact I personally believe that Java’s share of enterprise software will decline, but not in favor of anything from Redmond. I think that dynamic languages (Python and friends), particularly in conjunction with Test-Driven Development, are looking more like winners all the time. They generally are cheaper to program in, run just as fast, and have fewer bugs; what’s not to like?

Goals / Applications

- Fluid/Agile application development
  - Optional typing
- Reuse existing Java code
- Unit testing tasks
- Build automation
- Scripting of Java Applications
- Improve efficiency when working with
  - XML
  - SQL

Influences

- Ruby
- Python
- Dylan
- Xen

Language Features

- Optional Typing
- Closures
- Native syntax for lists and maps
- Regex Syntax
- Operator overloading
- GroovyBeans
- Groovy Path Expression language
- Polymorphic iteration and autoboxing
- Compiles direct to Java byte code
- Interoperates cleanly with Java libraries

Environment features

- Groovy Markup
- Ant Scripting
- Groovy SQL
- Groovlets
- UI building - groovy-swt, also a swing builder

Optional Type Declarations

- Typed Groovy = Java + Autoboxing + Syntax

Closures

- Syntax
  - Today
    - { var | block}
  - Tomorrow
    - { | var | block }
- Assign to variables
  - c = { x | return x == “John” }
- Call method
  - c.call(“Fred”)
- Keyword it
  - c = { return it == “John” }

Closures and objects

```
accountFactory = { balance |
   return { op, amount |
     if (op == "deposit") {
        balance = balance + amount
        return balance
     } else if (op == "withdraw") {
        balance = balance - amount
         return balance
     } else if (op == "balance") {
        return balance
     }
   }
}

account = accountFactory.call(5)

println account.call("deposit", 100)
account.call("withdraw", 10)
println account.call("balance", 0)
```

105
95

Timing Closure

```
timer = { closure |
    start = System.currentTimeMillis()
    closure.call()
    println System.currentTimeMillis() - start
}

timer { "sleep 10".execute().waitFor() }
```

Calling closures

- Passing closure after args
  - fn(arg1,…,argn, Closure)
- Can call
  - fn(a,..,n, { x | … })
  - fn(a,..,n) { x | … }

Closures & control structures

- Operations on lists, ranges
  - `l = [1,2,3,4,5,6,7]`
  - `r = 1..7`
- collect
  - Call the closure on every element and return a list of the closure results
  - `l.collect { return it * it }`
  - *[1, 4, 9, 16, 25, 36, 49]*
- each
  - Call the closure on every element of the collection
  - `l.each { print it }`
  - *1234567*

Control structures 2

- find
  - Return the first collection value that causes the closure to evaluate to true
  - `l.find { it == 4 }`
  - *4*
- findAll
  - Return a list of all collection values that cause the closure to evaluate to true
  - `l.findAll { it > 4 }`
  - *[5, 6, 7]*

Control Structure 3

- every
  - return true if every element in collection causes the closure to evaluate to true
  - `r.every { it > 0 }`
    - *true*
  - `r.every { it > 4 }`
    - *false*
- any
  - return true if any element in collection causes the closure to evaluate to true
  - `r.any { it > 4 }`
    - *true*
  - `r.any { it > 100 }`
    - *false*

Control Structures 4

- inject
  - Iterate over the collection passing each succesive closure the result of the previous closure. Arg to inject is an initial value
  - `r.inject(1) { x, y | return x * y }`
    - *5040*

Closures and I/O

- eachLine

  -
    ```
    new File('IO.groovy').eachLine { line |
       println(line)
    }
    ```
- eachByte
- eachFile
- withReader
- withStream
- withWriter

  -
    ```
    new File("groovy-output.txt").withWriter { w |
      new File('IO.groovy').eachLine { line |
         w.write(line)
      }
    }
    ```
- withPrintWriter
- withOutputStream

Easy to use Java code

- Just import code
- Call it

Syntax

- Standalone functions (closures)

  -
    ```
    f = { x, y |
     return x+y
    }
    ```
- Standalone statements
  - `f(1,3)`
  - *4*
- Optional Semicolons
- Optional parentheses

List syntax

- Lists are java.util.List instances
- Lists are enclosed in square brackets
- `l = [ 1, 2 , 3 ]`
- List access via indexing
- `l[0]`
  - 1

List operations

- << is append
  - l << 4
  - *[1,2,3,4]*
- flatten
  - [[1, 2, 3], [4, 5, 6]].flatten()
  - *[1, 2, 3, 4, 5, 6]*
- intersect
  - [1, 2, 4, 6, 8, 10, 12].intersect([1,3,6,9,12])
  - *[1,6,12]*
- minus
  - [1, 2, 4, 6] - [2, 4]
  - *[1, 6]*

List operations 2

- pop
  - [1, 2, 4, 6].pop()
  - 6
- reverse
  - [1, 2, 4, 6].reverse()
  - [6, 4, 2, 1]

Map syntax

- Maps are java.util.Map instances
- Map enclosed with `[]`
- Empty map is written `[:]`
- Key value pairs separated by `,`
- Keys and values separated by `:`
- `m = [ ‘a’:1, ‘b’:2, ‘c’:3 ]`
- Values retrieved by key:
  - `m[‘b’]`
    - *2*

Collection Methods

- l = [1, 2, 4, 6, 2, 3, 5]
- count
  - `l.count(2)`
  - *2*
- join
  - `l.join(“:”)`
  - *“2:4:6:2:3:5”*

Collections 2

- min
  - `l.min()`
  - *1*
- max
  - `l.max()`
  - *1*
- plus
  - `l.plus(“a”)`
  - *[1, 2, 4, 6, 2, 3, 5, a]*
- sort
  - `l.sort()`
  - *[1, 2, 2, 3, 4, 5, 6]*

Ranges

- Ranges implement java.util.List
- Notation allows
  - Inclusive ..
  - Exclusive of top …
- Integer
  - `3..7` contains 3, 4, 5, 6, 7
  - `3…7` contains 3, 4, 5, 6
- Character
  - `“a”..”d”` contains a, b, c, d
  - `“a”…”d”` contains a, b, c

Ranges 2

- Implement groovy.lang.Range
  - getFrom
  - getTo
- Subinterfaces
  - IntRange
    - `contains` method
  - ObjectRange
    - `contains` method

Ranges and slicing

- You can use ranges to access strings and lists
- s = “this is a test”
- s[1..3]
  - *his*
- Reversed ranges give reversed results
- s[3..1]
  - *sih*
- Negative indices start from the end
- s[-4..-1]
  - *test*
- s[-1..-4]
  - *tset*

Methods added to Object

- dump
  - l = [‘a’,’b’,’c’]
  - *"<java.util.ArrayList@1ecc1 elementData=[a, b, c] size=3 modCount=3>"*
- print
- println

Methods added to String

- `s="this is a test"`
- contains
  - `s.contains("is")`
    - *true*
  - `s.contains("ted")`
    - *false*
- count
  - `s.count("is")`
    - 2
- tokenize
  - `s.tokenize()`
    - *["This", "is", "a", "test"]*

String methods 2

- minus
  - s - “a”
    - *“this is test”*
- multiply
  - s * 2
    - *“this is a testthis is a test”*

Regular Expressions

- Based on JDK 1.4 Regex
- ~”pattern”
  - Pattern.compile(“pattern”)
  - `pat = ~".*(\\d{5})"`
- “text” =~ “pattern”
  - Pattern.compile(“pattern”).matcher(“text”)
  - `m = "CA 95014" =~ pat`
- “test” ==~ “pattern”
  - Pattern.compile(“pattern”).matcher(“text”).matches()
  - `"CA 95014" ==~ pat`
  - *true*

Operator overloading

- == (Java equals)
- != (Java !equals)
- === (Java ==)
- <=> (Java compareTo)
- >
- >=
- <
- <=

Operator overloading 2

- +
- -
- *
- /
- ++
- --
- x[y]
- x[y] = z

Switch

- Case on various types
  - String
    - `case ‘string’:`
  - Range
    - `case 1..10:`
  - In a list
    - `case [‘alpha’, ‘beta’, ‘gamma’]:`
  - Class name (instanceof)
    - `case java.util.Date:`
  - Regex
    - `case ~”\\d{5}”:`
- isCase method called for case comparisons
  - Override this to allow your classes to be switched on

Switch 2

```
accountFactory = { balance |
   return { op, amount |
     switch (op) {
       case 'deposit':
        balance = balance + amount
        return balance
       case 'withdraw':
        balance = balance - amount
         return balance
       case 'balance':
        return balance
       default:
        throw IllegalArgumentException
     }
   }
}
```

Looping

- for
  - `for (i in 1..10) { println i }`

  -
    ```
    l = 1..10
    for (i in l) { println i }
    ```
- while

  -
    ```
    i = 0
    while (i < 10 ) {
     println i
     i++
    }
    ```

Looping 2

- each
  - `(1..10).each { println it }`
  - `l.each { println it }`
- times
  - `10.times { println it }`
- upto
  - `1.upto(10) { println it }`
- step
  - `1.step(10,2) { println it }`

Here documents

- Shell style

  -
    ```
    h1= <<<THEEND
    This
    is
    a
    multiline
    string
    THEEND
    ```
- Python style

  -
    ```
    h2 = “””
    This
    is
    a
    Python
    style
    multiline
    string
    “””
    ```

String interpolation

- Use ${expr} to insert the value of expr into a string

-
  ```
  count = 4
  println "The total count is ${count}"
  ```
- *The total count is 4*

Groovy Beans

- Like Java Beans
- Properties
- Auto generate getters and setters
  - for public, protected properties

Groovy Beans 2

```
class Feed {
    String title
    String link
    Person author
    String tagline
    String generator
    String copyright
    String modified
    List entries
}

class Entry {
    String title
    String link
    String id
    String summary
    String content
    Person author
    String created
    String issued
    String modified
}
```

Groovy Beans 3

```
class Person {
    String name
    String url
    String email
}


f = new Feed()

f.author = new Person(
   name:'Ted Leung',url:'http://www.sauria.com/blog',
   email:'twl@sauria.com')

f.entries = [
   new Entry(title:'one',summary:'first post'),
   new Entry(title:'two',summary:'the second post'),
   new Entry(title:'three', summary:'post the third'),
   new Entry(title:'four',summary:'the ponderous fourth post')
]
```

GPath object navigation

- x.y.z = x.getY().getZ()
  - `f.author.name`
    - *Ted Leung*
- x->y->z (avoids nullptr)
  - `f->author->name`
    - *Ted Leung*
- Works over lists
  - `f.entries.name`
    - *[‘one’, ‘two’, ‘three’, ‘four’]*

GPath and closures

-
  ```
  f.entries.any {
     it.author.email == “twl@sauria.com”
  }
  ```
- *true*

Groovy Markup

- Application of closures
- Functions create elements
- Function arguments create either attributes or text content
  - Named arguments create attributes
  - String arguments create text content
  - Maps create mixed content
- Closures create nested content

XML MarkupBuilder

```
xml = new MarkupBuilder()

atom = xml.atom {
  title("Ted Leung off the air")
  link("http://www.sauria.com/noblog")
  author() {
    person() {
      name(f.author.name)
      url(f.author.url)
      email(f.author.email)
    }
  }
  for (e in f.entries) {
    entry() {
      summary(e.summary)
    }
  }
}
```

XML MarkupBuilder Result

```
<atom>
  <title>Ted Leung off the air</title>
  <link>http://www.sauria.com/noblog</link>
  <author>
    <person>
      <name>Ted Leung</name>
      <url>http://www.sauria.com/blog</url>
      <email>twl@sauria.com</email>
    </person>
  </author>
  <entry>
    <title>one</title>
    <summary>first post</summary>
  </entry>
  <entry>
    <title>two</title>
    <summary>the second post</summary>
  </entry>
  <entry>
    <title>three</title>
    <summary>post the third</summary>
  </entry>
  <entry>
    <title>four</title>
    <summary>the ponderous fourth post</summary>
  </entry>
```

Builders

- NodeBuilder
- DOMBuilder
- SAXBuilder
- MarkupBuilder
- AntBuilder
- SwingBuilder
- SWTBuilder

Ant Scripting

```
import groovy.util.AntBuilder
import org.codehaus.groovy.ant.Groovyc

ant = new AntBuilder()

ant.taskdef(name:'groovyc', classname:'org.codehaus.groovy.ant.Groovyc')

ant.sequential {
  echo("copying files")
  myDir = "bin"

  delete(dir:myDir)
  mkdir(dir:myDir)
  copy(todir:myDir) {
    fileset(dir:".") {
      include(name:"**/*.groovy")
      exclude(name:"**/EraseTyped.groovy")
    }
  }

  echo("Compiling Groovy files")
  groovyc(srcdir:myDir, destdir:myDir)

  echo("done")
}
```

GroovySQL

- Use closures to make JDBC easier

```
import groovy.sql.Sql
import java.sql.DriverManager

Class.forName("org.hsqldb.jdbcDriver")
connection =
   DriverManager.getConnection("jdbc:hsqldb:hsql://localhost", "sa", "")

sql = new Sql(connection)

sql.eachRow("SELECT name, price FROM prices") { row |
  println "${row.name} costs ${row.price}"
}
```

Groovlets

- Write servlets using Groovy
- Use the GroovyServlet to process scripts
- Allow implicit access to key servlet objects

Groovlets 2

```
if (session.counter == null) {
    session.counter = 1
}

out.println(<<<EOS
<html>
<head>
<title>Groovy Servlet</title>
</head>
<body>
Hello, ${request.remoteHost}: ${session.counter}! ${new Date()}
<br>src
</body>
</html>
EOS)

session.counter = session.counter + 1
```

Invoking Groovy Scripts

- Interactive Shell
- Interactive Swing Console
- Script compilation

Tool Support

- Eclipse
- IntelliJ
- Ant groovyc task

Embedding Groovy in Java

- Use GroovyShell to execute scripts
- Use GroovyClassLoader to expose Groovy objects to Java
  - Semi inconvenient due to invokeMethod

Implementation

- Each Groovy class is compiled to a Java class
- Java classes callable from Groovy
- Groovy classes callable from Java

Applications

- Template Engines
- Gap (groovy, picocontainer, dynaop)
- Query language like JXPath
- IDE Scripting, IDE/Appserver integration
- OpenEJB Telnet client allows groovy script execution
    - BEA investigating this also

Development Process

- Groovy Team @ Codehaus.org
  - Led by James Strachan
  - Open Source
    - Apache Style License
  - Small but growing community
- JSR-241 (proposed)
  - Apache, BEA, Thoughtworks

Status

- 1.0beta 4
  - 1.0 scheduled for a couple of months
- Eclipse plugin for 2.1.2, 3.0M5

Issues

- Stability
- No static method dispatch
- No eclipse refactoring support
- Syntax still subject to change

Minuses

- Still working out syntax
- Small community for now
- IDE plugins need work
- Not Python or Perl
- Built using Maven

Future features

- Metadata support
- Multiple assignment
- Python style Generators
- Xen style cardinality syntax
- Inner classes
- Mixins
- JDK 1.5 style imports
- JDK 1.5 style varargs
- Syntax extension

Resources

- http://groovy.codehaus.org
- irc://irc.codehaus.org/groovy
- http://www.ociweb.com/jnb/jnbFeb2004
- http://viva.sourceforge.net/talk/jug-mar-2004/slides.html
- http://www.sauria.com/blog