28.4.2006

Jussi Riihelä

jussi.riihela@nokia.com

Content

- Basic facts and motivation
- Groovy features
- IDE support and runtime dependencies
- Criticism

Groovy -- Basic facts

- New dynamic, object oriented scripting language for JVM
- Language features from Ruby, Python and SmallTalk
- Java like syntax – easy to pickup by Java programmers
- What comes in the package?
  - Groovy shell interpreter
  - Groovyc compiler to compile groovy into .class files
  - Groovysh, interactive groovy shell
  - Class libraries
- Developed by James Strachan and Bob McWhirter in 2003 as an open source project
- Groovy is currently undergoing standardization through the Java Community Process under JSR 241
- Related work
  - JPython
  - JRuby
  - BeanShell

```
C:\Temp\java\groovy>type hello.groovy
println "Hello World!"

C:\Temp\java\groovy>groovy hello.groovy
Hello World!
C:\Temp\java\groovy>
```

Need for scripting languages in general

- We use far more code than we write
- More and more time is used to wire components together
- We write more test code than real code
- No need to compile class files before execution
- CPU cycles keep getting cheaper, developers largely don’t
- More productive
- It’s fun

Why yet another dynamic language?

- Dynamic language designed especially for Java Platform
- Effortless transition from Java to Groovy
  - Groovy is based on Java's syntax and libraries, which are known to Java programmers.
  - Groovy feels like Java with fewer restrictions.
    - Freeing developers from compilation
    - Permitting dynamic types
    - Easing syntactical constructs
    - Allowing its scripts to be used inside normal Java applications
    - Providing a shell interpreter
- Lots of reusable Java software, components and tools
- Re-using existing JVMs
  - Maturity
  - Security
  - interoperability

Transition from Java to Groovy

- Sample Groovy code
- Introduce Groovy language features
- Modify correct Java program to take advantage of Groovy's features

Grooving

Original Java Code

```
import java.util.*;
class Erase {
  public static void main(String[] args) {
          List names = new ArrayList();
          names.add("Austin");
          names.add("Vanessa");
          names.add("Alotta");
          names.add("Scott");
          System.out.println(names);
          Erase e = new Erase();
          List short_names = e.filterLongerThan(names, 5);
          System.out.println (short_names.size());
          Iterator i = short_names.iterator();
          while (i.hasNext()) {
            String s = (String) i.next();
            System.out.println(s);
          }
  }
  public List filterLongerThan (List strings, int length) {
          List result = new ArrayList();
          Iterator i = strings.iterator();
          while ( i.hasNext() ) {
            String s = (String) i.next();
            if (s.length() < length+1) {
                result.add(s);
            }
          }
          return result;
  }
}
```

Output:

```
[Austin, Vanessa, Alotta, Scott]
1
Scott
```

Grooving

Remove semicolons

- Bad luck for compiler builders

```
names.add("Vanessa");
```



```
names.add("Vanessa")
```

```
import java.util.*
class Erase {
  public static void main(String[] args) {
          List names = new ArrayList()
          names.add("Austin")
          names.add("Vanessa")
          names.add("Alotta")
          names.add("Scott")
          System.out.println(names)
          Erase e = new Erase()
          List short_names = e.filterLongerThan(names, 5)
          System.out.println (short_names.size())
          Iterator i = short_names.iterator()
          while (i.hasNext()) {
            String s = (String) i.next()
            System.out.println(s)
          }
  }
  public List filterLongerThan (List strings, int length) {
          List result = new ArrayList()
          Iterator i = strings.iterator()
          while ( i.hasNext() ) {
            String s = (String) i.next()
            if (s.length() < length+1) {
                result.add(s)
            }
          }
          return result
  }
}
```

Grooving

Remove Java (1.4) iterators

- New collection helper methods

```
Iterator i =
short_names.iterator()
while (i.hasNext()) {
 String s=(String)i.next()
 System.out.println(s)
}
```

```
for(s in short_names) {
  System.out.println(s)
}
```

Note: Java 1.5 has improved iterators using templates

```
import java.util.*
class Erase {
  public static void main(String[] args) {
          List names = new ArrayList()
          names.add("Austin")
          names.add("Vanessa")
          names.add("Alotta")
          names.add("Scott")
          System.out.println(names)
          Erase e = new Erase()
          List short_names = e.filterLongerThan(names, 5)
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
  public List filterLongerThan (List strings, int length) {
          List result = new ArrayList()
          for( s in strings) {
            if (s.length() < length+1) {
                result.add(s)
            }
          }
          return result
  }
}
```

Dynamic Typing

```
class Song {
  String name
}

class Book {
  String name
}

def printName(thing){
  println "The name is " + thing.name
}
mySong = new Song()
mySong.name = "Foxy Lady"
myBook = new Book()
myBook.name = "The Da Vinci Code"

printName(mySong)
printName(myBook)
```

```
The name is Foxy Lady
The name is The Da Vinci Code
```

- Power of polymorphism without interfaces
- printName takes one parameter and attempts to print parameter's name property

Grooving

Dynamic typing (diff)

- Object's type is discovered dynamically at runtime

```
import java.util.*
class Erase {
  public static void main(String[] args) {
          List names = new ArrayList()
          names.add("Austin")
          names.add("Vanessa")
          names.add("Alotta")
          names.add("Scott")
          System.out.println(names)
          Erase e = new Erase()
          List short_names = e.filterLongerThan(names, 5)
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
  public List filterLongerThan (List strings, int length) {
          List result = new ArrayList()
          for( s in strings) {
            if (s.length() < length+1) {
                result.add(s)
            }
          }
          return result
  }
}
```

Grooving

Dynamic typing

What about these?

```
import java.util.*
class Erase {
  public static main(args) {
          List names = new ArrayList()
          names.add("Austin")
          names.add("Vanessa")
          names.add("Alotta")
          names.add("Scott")
          System.out.println(names)
          Erase e = new Erase()
          List short_names = e.filterLongerThan(names, 5)
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
  public filterLongerThan(strings, length) {
          List result = new ArrayList()
          for( s in strings) {
            if (s.length() < length+1) {
                result.add(s)
            }
          }
          return result
  }
}
```

Grooving

Convenient List initialization

```
List names=
  new ArrayList();
names.add("Austin")
names.add("Vanessa")
names.add("Alotta")
names.add("Scott")
```

```
List names = ["Austin",
              "Vanessa",
              "Alotta",
              "Scott"]
```

```
import java.util.*
class Erase {
  public static main(args) {
          List names = ["Austin", "Vanessa",
                        "Alotta", "Scott"]
          System.out.println(names)
          Erase e = new Erase()
          List short_names = e.filterLongerThan(names, 5)
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
  public filterLongerThan(strings, length) {
          List result = new ArrayList()
          for( s in strings) {
            if (s.length() < length+1) {
                result.add(s)
            }
          }
          return result
  }
}
```

Closures

- A closure in Groovy is an anonymous chunk of code that may
  - take arguments
  - return a value
  - reference and use variables declared in its surrounding scope
- Closures are first class objects that are similar to anonymous inner classes found in the Java language.
- Closures can have names
- Closures can be assigned
- Looks like ordinary code blocks, but are not executed until the call() is made on the closure

```
class Dog{
  def action

  def train() {
    action.call()
  }
}

sit = { println "Sit, Sit! Sit! Good dog"}
down = { println "Down! DOWN!" }

myDog = new Dog()
myDog.action = sit;
myDog.train()

mollie = new Dog()
mollie.action = down
mollie.train()
```

```
Sit, Sit! Sit! Good dog
Down! DOWN!
```

Grooving

Closure

```
List result =
  new ArrayList()
for( s in strings) {
 if(s.length()<length+1)
 {
   result.add(s)
 }
}
```

```
List result =
 strings.findAll
 {s->s.length()<length+1}
```

```
import java.util.*
class Erase {
  public static main(args) {
          List names = ["Austin", "Vanessa",
                        "Alotta", "Scott"]
          System.out.println(names)
          Erase e = new Erase()
          List short_names = e.filterLongerThan(names, 5)
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
  public filterLongerThan(strings, length) {
          List result = strings.findAll
             { s -> s.length() < length + 1 }
          return result
  }
}
```

Grooving

Refactoring

- Removing method call

```
Erase e = new Erase()
List short_names =
 e.filterLongerThan(names,
                    5)
...
public
filterLongerThan(strings,
                 length){
 List result =
   strings.findAll
  {s->s.length()<length+1}
 return result
}
```



```
int length = 5
List result =
names.findAll
{s->s.length()<length+1}
```

```
import java.util.*
class Erase {
  public static main(args) {
          List names = ["Austin", "Vanessa",
                        "Alotta", "Scott"]
          System.out.println(names)
          int length = 5
          List short_names = names.findAll
                          { s -> s.length() < length + 1 }
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
}
```

Grooving

Refactoring (diff)

- Class and main structures do not contribute much

```
System.out.println(names)
```

```
println names
```

```
for(s in short_names) {
   System.out.println(s)
}
```

```
short_names.each{s->println(s)}
```

```
import java.util.*
class Erase {
  public static main(args) {
          List names = ["Austin", "Vanessa",
                        "Alotta", "Scott"]
          System.out.println(names)
          int length = 5
          List short_names = names.findAll
                          { s -> s.length() < length + 1 }
          System.out.println (short_names.size())
          for(s in short_names) {
            System.out.println(s)
          }
  }
}
```

Grooving

Final version

- 7 lines of code
- Original: 30 lines of code

```
List names = ["Austin", "Vanessa", "Alotta", "Scott"]
println names
int length = 5
List short_names = names.findAll
                          { s -> s.length() < length + 1 }
println short_names.size()
short_names.each { s -> println(s) }
```

XML Generation

- Support for hierarchical structures in code
- Builders
  - XML
  - DOM
  - Swing
  - Etc.

```
import groovy.xml.*
data = ['Rod': ['Misha':9, 'Bowie':3],
        'Eric': ['Poe':5, 'Doc':4] ]
def xml = new MarkupBuilder()
doc = xml.people() {
  for( s in data) {
    person(name: s.key) {
      for(d in s.value) {
         pet(name:d.key, age:d.value)
      }
    }
  }
}
println doc
```

```
<people>
  <person name='Rod'>
    <pet name='Bowie' age='3' />
    <pet name='Misha' age='9' />
  </person>
  <person name='Eric'>
    <pet name='Poe' age='5' />
    <pet name='Doc' age='4' />
  </person>
</people>
```

XML Parsing

- Uses GPath to access elements
  - XPath like syntax

```
xml2 = """<people>
  <person name='Rod'>
    <pet name='Bowie' age='3' />
    <pet name='Misha' age='9' />
  </person>
  <person name='Eric'>
    <pet name='Poe' age='5' />
    <pet name='Doc' age='4' />
  </person>
</people>"""

people = new groovy.util.XmlParser().parseText(xml2)
println people.person.pet['@name']
```

```
[Bowie, Misha, Poe, Doc]
```

Some other features added to Groovy not available in Java

- Native syntax for maps and arrays
  - `def myMap = ["Austin":35, "Scott":20]`
- Native support for regular expressions
  - `if ("name" ==~ "na.*" ) { println "wow" }`
- Extended switch statement
- Embedded expressions inside strings
  - `def name="jussi"; println "hello $name"`
- New helper methods added to the JDK
  - For example String contains methods `count(), tokenize(), each()`
- You can add your own methods to existing classes
- Operator overloading
  - a + b maps to a.plus(b)

Groovy modules

- Grails == “Groovy on Rails”
- GORM == Grails object relation mapping
- Groovy SOAP
- XMLRPC
- Jabber-RPC
- GroovyServer Pages

Runtime dependencies

- Java 1.4
- Groovy jars
- ASM is a Java bytecode manipulation framework. http://asm.objectweb.org/
- ANTLR, ANother Tool for Language Recognition. http://www.antlr.org/

IDE Support

- Plug-ins for example for:
  - Eclipse, IntelliJ IDEA, NetBeans, JEdit, XCode, Emacs, UltraEdit
- Case Eclipse plugin:
  - “The Groovy Eclipse Plugin allows you to edit, compile and run groovy scripts and classes.”
  - Syntax highlight and intending.
  - Supports Eclipse 3.1 and higher
  - “Plugin is in no shape to be released.” Current version 0.9.2.

Criticism

- Design Issues
  - Is Java code legal Groovy code
    - Java++ or complete new language
    - Keep up with changing Java
  - Optional features difficult to implement
    - Semicolons
    - Return statement
    - Parenthesis on method invocation
  - Underspecified
    - Lack of documentation
    - Huge number of features
    - Feature interactions could lead to very confusing errors
- Development Process issues
  - Lack of organized development – no meeting notes etc.
  - Language development time spent on “wrong things”
  - Lack of vision
- If you write Groovy code now, prepare to change it later on (Mike Spille, Jan 2005)

Summary

- Groovy is an agile dynamic language for the Java 2 Platform
- Java-like syntax
- Features from Ruby, Python and SmallTalk
- Can use Java libraries
- Standardization under JSR 241

Resources

- Rod Cope: JavaOne 2005 Groovy Presentation
- Groovy homepage, http://groovy.codehaus.org
- Andrew Glover: alt.lang.jre: Feeling Groovy, http://www-128.ibm.com/developerworks/java/library/j-alj08034.html
- http://www.nl-jug.org/pages/events/content/jspring_2004/sessions/triveratech/slides.pdf/
- http://onestepback.org/articles/groovy/groovyfeatures.html
- http://www.pyrasun.com/mike/mt/archives/2005/01/09/20.57.06/
- http://www.cabochon.com/~stevey/sokoban/docs/article-groovy.html