Groovy 快速入门

1、集合

（1）List (java.util.List)

```
list = [1, 2, 'hello', new java.util.Date()]

assert list.size() == 4

assert list.get(2) == 'hello'
```

注意：一切都是对象（数字会自动转换）

（2）Map (java.util.Map)

```
map = ['name':'James', 'location':'London']

assert map.size() == 2

assert map.get('name') == 'James'
```

（3）遍历集合

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

for (i in list) { println i }
```

2、闭包（Closures）

闭包类似 Java 的内类，区别是闭包只有单一的方法可以调用，但可以有任意的参数

```
closure = { param | println("hello ${param}") }

closure.call("world!")


closure = { greeting, name | println(greeting + name) }

closure.call("hello ", "world!")
```

● 闭包用“{}”括起，“|”前面是参数，后面是处理语句，使用 call 调用

这里我们使用了一个 Groovy-Beta3 闭包定义，只是为了让您能看懂老版本 Groovy 的一些例子。在新的 Groovy JSR 中对闭包的定义略有改动，因为 | 字符同时也是 Java 中的位操作符；Groovy JSR 建议使用 Nice (另外一种 JRE 语言）样式的 -> 分隔符代替它。

第一个例子演示了在字符串内使用参数的形式：${param}

第二个例子演示了多参数形式：用“，”分隔参数

如果只有一个参数，可以不写，而使用缺省的参数“it”，如下面的例子：

```
closure = { println "hello " + it }

closure.call("world!")
```

3、each

遍历集合，逐个传递给闭包

```
[1, 2, 3].each { item | print "${item}-" }
```

上面例子的输出结果是：1-2-3-

4、collect

遍历集合，逐个传递给闭包，处理后的结果返回给对应的项

```
value = [1, 2, 3].collect { it * 2 }

assert value == [2, 4, 6]
```

5、find

根据闭包断言，返回集合中找到的第一个项目

```
value = [1, 2, 3].find { it > 1 }

assert value == 2
```

6、findAll

根据闭包断言，返回集合中所有找到的项目

```
value = [1, 2, 3].findAll { it > 1 }

assert value == [2, 3]
```

7、inject

遍历集合，第一次将传递的值和集合项目传给闭包，将处理结果作为传递的值，和下一个集合项目传给闭包，依此类推

```
value = [1, 2, 3].inject('counting: ') { str, item | str + item }

assert value == "counting: 123"


value = [1, 2, 3].inject(0) { count, item | count + item }

assert value == 6
```

8、every

如果集合中所有项目都匹配闭包断言，就返回 true，否则返回 false

```
value = [1, 2, 3].every { it < 5 }

assert value



value = [1, 2, 3].every { item | item < 3 }

assert ! value
```

9、any

如果集合中任何项目匹配闭包断言，就返回 true，否则返回 false

```
value = [1, 2, 3].any { it > 2 }

assert value


value = [1, 2, 3].any { item | item > 3 }
```

```
assert value == false
```

10、min/max

返回集合中的最小/最大项目（对象必须可比较）

```
value = [9, 4, 2, 10, 5].max()

assert value == 10

value = [9, 4, 2, 10, 5].min()

assert value == 2

value = ['x', 'y', 'a', 'z'].min()

assert value == 'a'
```

11、join

连接集合中的值成一个字符串

```
value = [1, 2, 3].join('-')

assert value == '1-2-3'
```

12、 yield

在 Python 和 Ruby 中通过 yield 语句创建“yield”风格的 iterators，在 Groovy 同样有效，只是使用的是闭包

```
class Foo{
  static void main(args) {
        foo = new Foo()
        for (x in foo.myGenerator) {
              print("${x}-")
        }
  }
  myGenerator(Closure yield) {
        yield.call("A")
        yield.call("B")
        yield.call("C")
  }
}
```

例子的输出结果是：A-B-C-

Cloures 原型可以省略，call 和括号同样可选，这样更象 Python/Ruby

```
class Foo {
  myGenerator(yield) {
        yield "A"
        yield "B"
        yield "C"
  }

  static void main(args) {
```

```
        foo = new Foo()
        foo.myGenerator { println "Called with ${it}" }
    }
}
```

Groovy 全攻略--嵌入篇

Groovy 被设计得非常轻量级,很容易迁入到任何 Java 应用系统。

你可以使用 BSF 将 Groovy 脚本嵌入任何 Java 代码中.但是 Groovy 提供了一个轻量级的紧密集成.

下面是 3 种主要方法:

1.使用 Shell 调试脚本或表达式

在 Groovy 中你可以使用 GroovyShell 对 Groovy 脚本和表达式进行调试.GroovyShell 允许你通过 Binding 对象传入或传出变量.

```
// 从 Java 代码中调用 Groovy 语句
Binding binding = new Binding();
binding.setVariable("foo", new Integer(2));
GroovyShell shell = new GroovyShell(binding);


Object value = shell.evaluate("println 'Hello World!'; x = 123; return foo * 10");
assert value.equals(new Integer(20));
assert binding.getVariable("x").equals(new Integer(123));
```

2.在 Java 中动态调用运行 Groovy 代码

你可以使用 GroovyClassLoader 将 Groovy 的类动态地载入到 Java 程序中并直接使用或运行它.

下面是一个例子:

```
ClassLoader parent = getClass().getClassLoader();
GroovyClassLoader loader = new GroovyClassLoader(parent);
Class groovyClass = loader.parseClass(new File("src/test/groovy/script/HelloWorld.groovy"));


// 调用实例中的某个方法
GroovyObject groovyObject = (GroovyObject) groovyClass.newInstance();
Object[] args = {};
```

```
groovyObject.invokeMethod("run", args);
```

如果你想使用一个用 Groovy 脚本实现的接口,你可以这么使用它:

```
GroovyClassLoader gcl = new GroovyClassLoader();
Class clazz = gcl.parseClass(myStringwithGroovyClassSource "SomeName.groovy");
Object aScript = clazz.newInstance();
MyInterface myObject = (MyInterface) aScript;
myObject.interfaceMethod();
...
```

如果某个 Groovy 类实现口 MyInterface 接口,那么上面的代码就会很好的工作.myObject 的使用方法与其他实现了 MyInterface 接口的 Java 对象一样.

3.Groovy 脚本引擎

对于那些想将 Groovy 脚本嵌入到服务器并且在每次修改后重新装入的人来说,Groovy 脚本引擎提供了一个彻底的解决方案.你可以设定系列 CLASSPATH 作为根来初始化 Groovy 脚本引擎,这些 GLASSPATH 可以是 URL 也可以是目录名.接着你就可以这些根路径下的任何 Groovy 脚本了.GSE 会跟踪脚本间的依赖关系,因此如果任何有依赖关系的脚本被修改,整颗树将会重新编译和载入.

另外,每次执行脚本时,你都可以传入一个包含脚本可接受属性的 Binding.脚本执行完以后,传入脚本中的那些属性在 Binding 中依然有效.下面是一个例子:

/my/groovy/script/path/hello.groovy:

```
output = "Hello, ${input}!"
```

```
import groovy.lang.Binding;
import groovy.util.GroovyScriptEngine;

String[] roots = new String[] { " /my/groovy/script/path " } ;
GroovyScriptEngine gse = new GroovyScriptEngine(roots);
Binding binding = new Binding();
binding.setVariable( " input " , " world " );
```

```
gse.run( " test.groovy " , binding);
System.out.println(binding.getVariable( " output " ));
```

将打印 "Hello, world!".

4.运行时依赖

和 JDK1.4 一样,Groovy Jar 也依赖与 ASM 库上的运行时,ASM 库包括 4 个 Jar(asm-2.1.jar, asm-util-2.1.jar, asm-attrs-2.1.jar and asm-analysis-2.1). 也就是说,只要将上面的 5 个 Jar 添加到路径中,你就能将轻松地 Groovy 嵌入到你的应用里.

另一种方案可以不用那么多的 Jar.你可以用 GROOVY_HOME/embeddable 目录下的 groovy-all-1.0-beta-x.jar.这个 Jar 包将 Groovy 和 ASM 组合打包成一个方便的 Jar 包.注意:groovy-all-1.0-beta-x.jar 中的 ASM 类使用了不同的命名空间,因此要避免与使用 ASM 的库发生冲突.

Groovy 全攻略--运行篇

Groovy 脚本是一个包含许多语句和类声明的文本文件.Groovy 的用法和其他脚本语言类似.下面是几种运行 Groovy 脚本的方法:

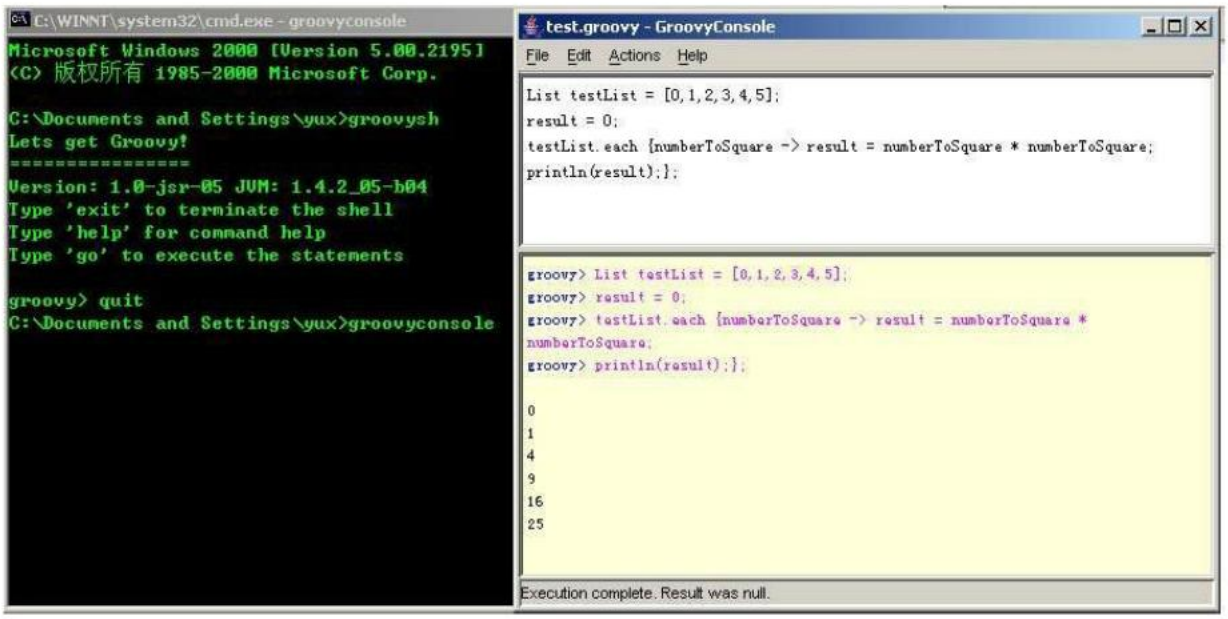
1.使用交互控制台

Groovy 有一个 Swing 交互控制台,允许你像使用 SQL 工具一样输入和执行 Groovy 语句.控制台支持历史记录,因此你可以向前或向后遍历命令.

如果你已经安装了 Groovy 二进制分发包,你可以使用下面的命令启动 Groovy Swing 控制台.

如果像启动命令行控制台,可以输入下面的命令:

```
GroovyConsole
```



如果想启动命令行控制台,可以输入下面的命令

```
Groovysh
```

从源代码分发包中启动 Swing Groovy 控制台

```
maven console
```

2.通过 IDE 运行 Groovy 脚本

有一个叫做 GroovyShell 的类含有 main(String[])方法可以运行任何 Groovy 脚本.你可以用下面的语句执行任何 Groovy 脚本:

```
java  groovy.lang.GroovyShell  foo/MyScript.groovy  [arguments
```

你可以在你的 IDE 中使用上面的 Groovy main()执行或调试任何 Groovy 脚本.

3.用命令行执行 Groovy 脚本

在 GROOVY_HOME\bin 里有个叫'groovy' 或'groovy.bat' 的脚本文件(或者二者都有,根据你的平台不同而不同).这些脚本文件是 Groovy 运行时的一部分.

一旦安装了 Groovy 运行时,你就可以这样运行 Groovy 脚本:

```
groovy foo/MyScript.groovy [arguments]
```

如果你使用从 CVS 构建的 Groovy,那么除了 Groovy 脚本以外,命令行上可能还运行着其他的类.

1.拥有 main()方法的类;

2.和 JUnit 的测试一起运行的继承自 GroovyTestCase 的类,

3.实现 Runnable 接口的类.run()方法被调用.

获得最新最强的 Groovy,输入下面的命令做一个 CVS 检出

```
maven groovy:make-install
```

在 groovy/target/install 里会有一份完整的二进制分发包.你可以将 groovy/target/install/bin 添加到你的路径,这样你就可以轻松地用命令行执行你的 Groovy 脚本了.

4.用 Groovy 创建 Unix 脚本

你可以用 Groovy 编写 Unix 脚本并且像 Unix 脚本一样直接从命令行运行它.倘若你安装的是二进制分发包并且设置好了路径,那么下面的代码将会很好的工作:

将其保存为 helloWorld.groovy.

```
#!/usr/bin/env groovy
println("Hello world")
for (a in this.args) {
  println("Argument: " + a)
}
```

接着从命令行运行这个脚本,

```
chmod +x helloWorld

./helloWorld
```