

Six Things Groovy Can Do For You

By Alan Green,

Senior Consultant

Cirrus Technologies Pty. Ltd.

Abstract

Groovy is a scripting language for the Java Virtual Machine, due for a 1.0 release in late 2005. This paper introduces Groovy and shows six ways Java programmers can put Groovy to use in their next application: scripting business objects, debugging running applications, simplifying unit tests, building quick web interfaces, mining data from RDBMSs and implementing model objects. For each of these, the paper includes code and discussion.

Introduction

Groovy is a scripting language for the Java Virtual Machine (JVM), due for its 1.0 release in early 2006. Groovy was designed to be used by Java programmers, introducing dynamic typing and convenience features to make an expressive scripting language that is also suitable for many of the tasks for which Java is currently used.

Groovy differentiates itself from other JVM-targeted languages in three key respects. First, it is being standardised via the Java Community Process as JSR-241, which gives it a certain credibility in the enterprise market. Second, its syntax is designed to be familiar to and comfortable for the typical Java programmer. Finally, Groovy programs can use both dynamic and static typing, allowing programmers to choose exactly how much “scriptiness” to add to their Java programs.

With its innate interoperability with Java and emphasis on expressiveness, Groovy is well placed to assist Java developers in their day-to-day tasks. This paper shows six ways that Java developers can put Groovy to Java use, showcasing a small selection of Groovy's features. Where space permits, code examples are provided.

A note on the examples

The examples in this paper draw on a project management application, named Cardboard Schedule. The key classes are Schedule, Person, Task and TaskType. As shown below (in illustration 1), a Schedule contains a list of Person, and each Person has a list of tasks assigned to them. The Schedule also contains a list of Tasks that are not assigned to any Person. Every Task has one TaskType.

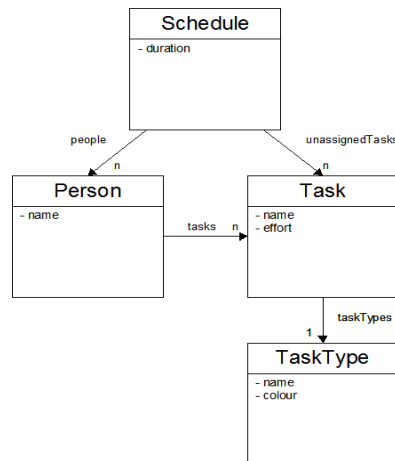


Illustration 1: Cardboard Schedule Model

Thing 1: Script Your Business Objects

Groovy works directly with Java classes and objects. This Groovy script reads a schedule and prints out summary information in a form that is not easily obtainable from the application.

```

01 import avg.cs.command.FileLoadCommand
02
03 def s = new FileLoadCommand("schedules/example3.cs").execute();
04
05 println "Name\t\tEffort"
05 println "----\t\t-----"
07 for (p in s.people) {
08     def effortSum = p.tasks.sum {it.effort}
09     println "${p.name}${effortSum}"
10 }
11 println "Unassigned\t" + s.unassignedTasks.sum {it.effort}
  
```

The output of the script is:

```

Name      Effort
----      -
Andrew    23.0
Jason     18.5
Mark      27.0
Kristen   21.25
Unassigned 4.5
  
```

The script highlights several Groovy features:

- Line 01 imports a Java class, “FileLoadCommand” from the Cardboard Schedule application.
- Line 03 instantiates the FileLoadCommand, and executes it, with a result being a Schedule object. This Schedule object is assigned to a dynamically typed, or “duck-typed” variable, s.
- Incidentally, the return type of execute () is Object, so Groovy avoids a cast at this point. The execute () method is also declared to throw an exception. Groovy does not require exceptions to be declared, and so avoids the overhead of try-catch statements in situations (such as short scripts) where they add little value.
- Line 07 fetches the people property of the Schedule. Since the Schedule class doesn't have a public variable named people, Groovy interprets “s.people” as a JavaBean attribute reference and

calls the Schedule's `getPeople()` method.

- Line 08 sums the `effort` attribute of each Task object in the list `p.tasks`. The curly braces define a closure, a snippet of code that is passed to Groovy's collection `sum()` function.
- Line 09 shows Groovy's string substitution features, which allows Groovy expressions to be embedded in Strings, quoted with “`#{}`” and “`{}`”.

Taken together, these features allow developers to quickly build useful scripts with application classes. By way of contrast, an equivalent program in Java is 33 lines long.

Thing 2: Debug Your Running Application

An interactive Groovy session can be embedded in almost any application. Groovy provides several interactive interfaces, including a Swing Graphical UI, a telnet-style TCP socket session, and a traditional stdin/stdout console. Also, being 100% Java, these interactive interfaces can be added to an application by simply including the Groovy jar files and making a few method calls.

This Groovy script instantiates the main Cardboard Schedule application object, and an interactive GUI session. It then creates a variable in the Groovy session, named `app`, that refers to the running application.

```
01 import avg.cs.gui.CsMainApp
02 import groovy.ui.Console
03
04 println 'Creating application'
05 def app = new CsMainApp(null)
06
07 println 'Starting Groovy'
08 def console = new Console()
09 console.shell.setVariable("app", app)
10 console.run()
11 println 'Showing app'
12 app.show();
```

A programmer can use the Groovy session to interactively inspect and modify the application, while it is running at full speed. What a Groovy way to debug!

Thing 3: Simplify Your Unit Tests

Groovy's expressiveness simplifies common unit testing tasks. For example, to check that a given schedule has three people, with the names, “Alan”, “Tom” and “John”, the Java JUnit code is:

```
List<Person> people = s.getPeople();
assertEquals(3, people.getSize());
assertEquals("Alan", people.get(0).getName());
assertEquals("Tom", people.get(1).getName());
assertEquals("John", people.get(2).getName());
```

With its list manipulation capabilities, Groovy can express the same test in a single line of code:

```
assertEquals(['Alan', 'Tom', 'John'], s.people.collect {it.name})
```

Other Groovy features, such as String manipulation and its less restrictive exception handling also result in smaller, more expressive test cases.

Thing 4: Throw Together A Web Interface

A developer can throw together simple dynamic web pages with Groovy's take on servlets, Groovlets. Groovlets are normal Groovy scripts, with some special variables defined, such as `request` and `response`, which are the script's `HttpServletRequest` and `HttpServletResponse`, respectively.

The following Groovlet takes the name of a schedule file, and lists the tasks for each person on that schedule.

```
01 import avg.cs.command.FileLoadCommand
02
03 def sName = request.getParameter("name")
04 def s = new FileLoadCommand(sName).execute();
05 html.html() {
06     body() {
07         for (p in s.people) {
08             h1(p.name)
09             ol() {
10                 p.tasks.each {li("${it.name}: (${it.effort} days)")}
11             }
12         }
13     }
14 }
```

The output of this is an HTML page <h1> tag and a list of task names for each person.

- Line 03 illustrates the request variable. The script uses the `getParameter()` from the Servlet API to fetch the value of an HTTP request parameter.
- Line 06 introduces another Groovlet variable, `html`, an XML builder. Via a cute (ab)use of Groovy syntax, function calls on the builder object are translated into XML tags. This technique is surprisingly convenient for assembling dynamic HTML pages.

Groovlets are a compact and expressive alternative to Servlets. They have special features that allow developers to build “quick-and-dirty” web interfaces, and are also able to replace Servlets in more traditional Model-View-Controller designs.

Thing 5: Mine Data from Your RDBMS

The majority of enterprise Java applications store their data in RDBMSs. Java's JDBC API is mature and capable, but often results in verbose code. The following script demonstrates Groovy's SQL API, `groovy.sql`.

```
01 import groovy.sql.*
02
03 def sql = Sql.newInstance('jdbc:jtds:sqlserver://localhost:1433/IDS_DB',
04     'net.sourceforge.jtds.jdbc.Driver')
05
06 def queryStr = """select message
07     from AUDIT_RECORD
08     where type = 'DCMD_RESULT' """
09
10 def rows = sql.rows(queryStr)
11
12 def fsCount = 0
13 for (row in rows) {
14     row["message"].eachMatch(/FS\d\d/) { fsCount++ }
15 }
16
17 println "${rows.size()} records, ${fsCount} stations"
18 println "Average = ${fsCount / rows.size()} stations per record"
```

The Groovy SQL API correctly closes connections, statements and result sets, even in the presence of exceptions. The author wrote a Java program to perform the same calculation – at over forty lines long it does not do as good a job with error handling as this simple script.

Thing 6: Implement Model Objects

It is common from Java applications to implement their domain model as a library of JavaBean classes. Groovy has many convenient features for implementing JavaBeans, of which we list two here.

First, Groovy will generate standard JavaBean getters and setters for member variables marked with “@Property”. The following three lines of Groovy:

```
01 class Person {
02     @Property String name
03 }
```

are equivalent to these nine lines of Java:

```
01 public class Person {
02     private String name;
03     public String getName() {
04         return name;
05     }
06     public void setName(String name) {
07         this.name = name;
08     }
09 }
```

Second, Groovy's convenient features for manipulating collections, Strings and JavaBeans allow complex business logic to be expressed in compact and readable forms. As an example, this method to return a schedule's assigned and unassigned tasks as a single list:

```
01 public List<Task> getAllTasks() {
02     List<Task> result = new ArrayList<Task>();
03
04     for (Person p : people) {
05         result.addAll(p.getTasks());
06     }
07
08     result.addAll(unassignedTasks);
09     return result;
10 }
```

can be replaced with just a few lines of Groovy:

```
01 List getAllTasks() {
02     (people.collect {it.tasks}).flatten() + unassignedTasks
03 }
```

Because Groovy domain models require less code to express the same concepts, they are arguably easier to understand and more maintainable.

Conclusion

This paper demonstrated six of the many ways that Groovy is able to assist Java developers perform typical development tasks. The Groovy features that make this possible are:

- familiar syntax – Java-like syntax which is easily learnt and understood by Java programmer,
- extended semantics – including duck-typing, closures, list and string manipulation,
- interoperability with existing Java code, and
- Groovy specific library classes, such as groovy.sql.

Java developers looking to improve their productivity and effectiveness would do well to consider introducing Groovy to their projects.

References and Further Information

1. The Groovy project website <http://groovy.codehaus.org/>. Includes tutorials, reference information and pointers to the Groovy mailing.
2. IBM Developerworks host an excellent set of articles written by Andrew Glover, under the series title, “Practically Groovy.” Articles relevant to this paper include:
 - JDBC Programming with Groovy - <http://www-128.ibm.com/developerworks/java/library/j-pg01115.html>
 - Unit test your Java code faster with Groovy - <http://www-128.ibm.com/developerworks/java/library/j-pg11094/>
 - Stir some Groovy into your Java apps - <http://www-128.ibm.com/developerworks/java/library/j-pg05245/index.html>