Groovyを使って、より高速にJavaコードをユニット・テストする

Andrew Glover

2004年 11月 09日

まだあまり古い話ではありませんが、developerWorksの寄稿者であるAndrew Gloverがalt.lang.jreシリーズの一部として、Javaプラットフォーム用の標準言語として新しく提案されたGroovyの紹介記事を書きました。読者からの反応が非常に大きかったので、この新しい技術を使うための実際的なガイドとして、コラムを始めることになりました。この最初の記事では、GroovyとJUnitを使ってJavaコードをユニット・テストするための単純な戦略を紹介します。

告白から始めましょう。私はユニット・テストの中毒患者です。実を言うと、いくらユニット・テストを書いても書き足りません。ずっと長い間、何かに対応するユニット・テストを書かずにいると、震えが来てしまいます。ユニット・テストによって、自分のコードが動作すること、壊す心配をせず瞬時に変更可能だと安心できるのです。

さらに、中毒患者なるが故に、私はテスト・ケースも書きすぎてしまいがちです。ただし私がハイ（興奮状態）になるのは、テスト・ケースを書くことから来るのではなく、その結果を見ることから来ています。ですから、もし私が非常に速くテスト・ケースを書くことができれば、もっと早く結果を見ることができます。そうすれば私はもっと気持ちが良くなります。そう、もっと早く。

最近私は自分のユニット・テスト中毒を抑えるためにGroovyを見ているのですが、少なくとも今までのところ、正に驚嘆しています。この新しい言語がユニット・テストにもたらすアジリティ（agility）には興奮すべきところがあり、真剣に調べるだけの価値があります。Groovyの実用的な面を紹介する新しいシリーズの第1回として、今回はGroovyで行うユニット・テストがいかに楽しいものであるかを説明します。最初はGroovy がいかにJavaプラットフォームでの開発に貢献するのかから始め、JUnitの`TestCase`クラスのGroovy版エクステンションを特に強調しながら、GroovyとJUnitによるユニット・テストの詳細に入ります。そして最後に、そうした素敵（groovy）な特徴を生かして、EclipseやMavenと統合するにはどうすべきかを実例を示しながらまとめることにします。

さらば、純Java主義よ

Groovyによるユニット・テストの実際面に入る前に、より一般的な問題、つまりGroovyが開発ツールボックスの中で占める位置を考えることが重要だと思います。実際のところGroovy

は、Java Runtime Environment (JRE) 上で実行する唯一のスクリプト言語というわけではなく、Javaプラットフォーム用の標準言語として提案された唯一のものだ、ということです。alt.lang.jreシリーズ（参考文献）を見たことがある人もいると思いますが、Javaプラットフォームのスクリプト化には山ほどオプションがあり、その大部分は高速なアプリケーション開発のための、高度にアジャイルな環境を提供するものです。

こうした豊富な選択肢がありながら、開発者の多くは自分たちが好きな、そして一番慣れているもの、つまりJava言語を選択してしまっています。大部分の場合、Javaプログラミングは妥当な選択なのですが、Javaのみに凝り固まってしまうことには一つ、大きな問題があるのです。ある賢人が、次のように言っています。「手元にある道具が金槌しか無い場合には、どんな問題も釘に見えてしまうものである。」これはソフトウェア開発にも大いに当てはまる真実ではないかと思います。

このシリーズで皆さんにお伝えしようとしているのは、アプリケーション開発における選択肢はJava言語のみではなく、Java言語のみに限定すべきものでもない、ということです。さらにまた、スクリプト言語は、他の場合には丸で意味をなさなくても、ある場合には意味があるのだということもお伝えしようと思っています。プロフェッショナルが初心者と違うのは、どういう場合にスクリプトの力を借りるべきか、逆にどういう場合にスクリプト化を避けるべきかを知っている、という点なのです。

このシリーズについて

どのようなツールであれ、開発作業の中に採り入れるためには、どういう場合に使うべきか、または使うべきではないかをよく知る必要があります。スクリプト言語は非常に強力なツールですが、その強力さは、適切なシナリオで適切な使い方をした場合にのみ発揮されます。実用的なGroovyシリーズはそういう点を念頭に置き、Groovyの実用的な使い方に焦点を絞って、どういう場合に、どのように使うのかを解説して行きます。

例えばハイ・パフォーマンスでトランザクションが集中する、企業全体に渡るようなアプリケーションであれば、スクリプトには向かないものです。こうした場合には通常のJ2EEスタックが最適かもしれません。一方スクリプトは、特にGroovyでのスクリプトは、コンフィギュレーション・システムやビルド・システムなど、あまりパフォーマンス重視ではない、小さな、非常に特別なアプリケーションでは、大いに意味をなすものです。またレポート・アプリケーションや、そしてなによりも重要な、ユニット・テストなどにも、ほとんど完璧と言えるほどに適しています。

なぜGroovyでユニット・テストなのか

他のスクリプト・プラットフォームと比べてGroovyが特に魅力的なのは、Javaプラットフォームとスムーズに統合できることです。JRE用の他の代替言語はJava以前のものに基づいたものが多いのですが、Groovyはそれらと異なりJava言語に基づいているので、Java開発者にとっては驚くほど短期間で習得できます。そして一旦習得できれば、他とは比較できないほど高速開発が可能なプラットフォームとなるのです。

Groovyの持つ強力さの秘密は、その構文にあります。Java構文ではあるのですが、はるかに規則が少ないのです。例えばGroovyではセミコロンが必要なく、また変数タイプとアクセス修飾子はオプションとなっています。さらにGroovyは`Collections`や`File/IO`を含めて、皆さんが既にお馴

染みの標準Javaライブラリーを利用しています。そして最後に、JUnitを含めて任意のJavaライブラリーを、Groovy内部から利用できます。

規則を緩めたJava風の構文を採用していることや標準Javaライブラリーを再利用していること、またビルドから実行までのサイクルが短いことなどから、Groovyはユニット・テストを素早く開発するための選択肢として最適なのです。ただし私の言葉だけから信用しないでください。コードで実際に見てみましょう。

JUnitとGroovy

GroovyでJavaコードをユニット・テストするほど楽なことはありません。そして、取りかかるための選択肢もたくさんあります。最も素直な選択肢は、業界標準であるJUnitで行くことです。JUnitの単純さと強力さには並ぶものがありませんし、Java開発ルールとしての使いやすさも他に匹敵するものはありません。またJUnitとGroovyを組み合わせて悪い理由は何もないので、そうしない理由はありません。実際、JUnitとGroovyを一緒に使ったところを一度見れば、皆さんも後戻りできなくなることは確実です。ここで重要なのは、Java言語でできるのと同じことがGroovyを使ったJUnitでもでき、しかもキーを押す回数はずっと少なくてすむ、ということです。

はじめに

JUnitとGroovyをダウンロードしたら（参考文献）2つの選択肢があります。一つは通常のJUnitテスト・ケースを書く方法です。これは皆さんが今までずっとしてきたのと同じで、ご推薦の、JUnitの`TestCase`を拡張します。もう一つはGroovyの持つ素敵な、`GroovyTestCase`エクステンションを使うことです。（`GroovyTestCase`エクステンションは結果的にJUnitの`TestCase`を拡張します。）最初の選択肢は、最もJavaらしく成功に至るための近道です。一方2番目の選択肢では、アジャイル性を充分に生かしたGroovyの世界に入り込むことができます。

まず手始めにJavaオブジェクトとして、与えられた`string`にフィルターを適用し、一致の有無に応じて`Boolean`値を戻すものを考えてみてください。このフィルターは、`indexOf()`のような単純な`string`操作であったり、あるいはもっと強力に正規表現であったりします。必要なフィルターは`setFilter()`メソッドによってランタイム時に設定され、`apply()` メソッドはフィルターの対象として`string`を取ります。リスト1は、単純なJavaコードを使った`Filter`インターフェースの例を示しています。

リスト1. 単純なJavaフィルター・インターフェース

```
public interface Filter {
  void setFilter(String fltr);  boolean applyFilter(String value);
}
```

この考え方としては、この機能を使って、大きなリストから必要なパッケージ名や必要ないパッケージ名をフィルターしようということです。結果的に私は`RegexPackageFilter`と`SimplePackageFilter`という、2つの実装を作りました。

GroovyとJUnitの強力さと単純さとを組み合わせると、リスト2に示すような優雅なテスト・スイートができあがります。

リスト2. JUnitを使ったGroovyのRegexFilterTest

```
import junit.framework.TestCase
import com.vanward.sedona.frmwrk.filter.impl.RegexPackageFilter
class RegexFilterTest extends TestCase {  void testSimpleRegex() {
    fltr = new RegexPackageFilter()
    fltr.setFilter("java.*")
    val = fltr.applyFilter("java.lang.String")
    assertEquals("value should be true", true, val)
  }
}
```

リスト2のコードは、Groovyに慣れているか否かに関わらず、おなじみのものに見えるはずです。これは単にJavaコードであって、セミコロンやアクセス修飾子、変数タイプが無いだけなので、当然なことです。上のJUnitテストには、`testSimpleRegex()` という一つのテスト・ケースがあります。これは、`RegexPackageFilter`が正規表現`#java.*#`を使って、`#java.lang.String#` との一致を正しく検出した、と言おうとしています。

GroovyがJUnitを拡張する

JUnitの`TestCase`クラスを拡張して機能を追加するのは、大部分のJUnitエクステンションで利用している一般的な手法です。例えばDbUnitフレームワーク（ 参考文献 ）には、データベースの状態管理が非常に簡単になる、手軽な`DatabaseTestCase`があります。そして必要不可欠な`MockStrutsTestCase`（ StrutsTestCaseフレームワークより、参考文献 ）は、strutsコード実行用の仮想的`servlet`コンテナーを生成します。こうした強力なフレームワークはどちらもJUnitを優雅に拡張し、そのコア・コードには無いような機能を提供します。そして今やGroovyもそこにまで達し、それを実現したのです。

StrutsTestCaseやDbUnitと同様、JUnitの`TestCase`のGroovy版エクステンションには、いくつか重要な機能があります。この特別なエクステンションを使うと`groovy`コマンドでテスト・スイートを実行でき、また数多くの新しい`assert`メソッドも提供しています。こうしたメソッドは、スクリプトが正しく実行していることや様々な配列タイプの長さを伝えたり、また様々な配列タイプの内容を伝えたりするには便利です。

GroovyTestCaseで遊ぶ

`GroovyTestCase`に何ができるかを学ぶためには、実際に動作しているところを見るのが一番です。私はリスト3で新しい`SimpleFilterTest`を書きましたが、今回は`GroovyTestCase`を拡張して行っています。

リスト3. 実際のGroovyTestCase

```
import groovy.util.GroovyTestCase
import com.vanward.sedona.frmwrk.filter.impl.SimplePackageFilter
class SimpleFilterTest extends GroovyTestCase {

  void testSimpleJavaPackage() {
    fltr = new SimplePackageFilter()
    fltr.setFilter("java.")
    val = fltr.applyFilter("java.lang.String")
    assertEquals("value should be true", true, val)
  }
}
```

このテスト・スイートは、JavaベースのJUnitテスト・スイートを実行するために必要な`main()`メソッド無しに、コマンドラインから実行できることに注意してください。実際、上の`SimpleFilterTest`をJavaコードで書いたとしたら、リスト4に示すようなものになるでしょう。

リスト4. 同じテスト・ケースをJavaコードで書いたもの

```
import junit.framework.TestCase;
import com.vanward.sedona.frmwrk.filter.Filter;
import com.vanward.sedona.frmwrk.filter.impl.SimplePackageFilter;

public class SimplePackageFilterTest extends TestCase {

   public void testSimpleRegex() {
 Filter fltr = new SimplePackageFilter();
 fltr.setFilter("java.");
 boolean val = fltr.applyFilter("java.lang.String");
 assertEquals("value should be true", true, val);
   }

   public static void main(String[] args) {
  junit.textui.TestRunner.run(SimplePackageFilterTest.class);
   }
}
```

assertでテストする

`GroovyTestCase`を使うとコマンドラインでテストを実行できるようになるばかりではなく、非常に便利な`assert`メソッドも使えるようになります。例えば`assertArrayEquals`は2つの配列を、その長さと個々の値をチェックすることによって、等しいのだ、と断言します。Groovyのassertの実際例はリスト5で見ることができます。これは`strings`を配列に分割する、Javaベースの素晴らしいメソッドです。（ 下記のクラス例は、Java 1.4で追加された`string`機能を使っても書けることに注意してください。Java 1.3との後方互換性を確実にするために、私はJakarta CommonsのStringUtilsクラスを使っています。 ）

リスト5. JavaのStringSplitterクラスを定義する

```
import org.apache.commons.lang.StringUtils;

public class StringSplitter {
  public static String[] split(final String input, final String separator){
   return StringUtils.split(input, separator);
  }
}
```

リスト6は、Groovyテスト・スイートとそれに対応した`assertArrayEquals`メソッドを使うことで、このクラスのテストがいかに簡単になるかを示しています。

リスト6. GroovyTestCaseにあるassertArrayEqualsを使う

```
import groovy.util.GroovyTestCase
import com.vanward.resource.string.StringSplitter

class StringSplitTest extends GroovyTestCase {

  void testFullSplit() {
    splitAr = StringSplitter.split("groovy.util.GroovyTestCase", ".")
    expect = ["groovy", "util", "GroovyTestCase"].toArray()
    assertArrayEquals(expect, splitAr)
  }
}
```

他にも遊び方が

Groovyでは単独でも、一まとめでもテストを実行することができます。`GroovyTestCase`エクステンションを使えば、単独テストの実行には何の苦労も要りません。単純にgroovyコマンドを実行してから、必要なテスト・スイートを実行すればよいのです。これをリスト7に示します。

リスト7. groovy コマンドでGroovyTestCaseを実行する

```
$./groovy test/com/vanward/sedona/frmwrk/filter/impl/SimpleFilterTest.groovy
.
Time: 0.047

OK (1 test)
```

Groovyではまた、`GroovyTestSuite`と呼ばれる、標準のJUnitテスト・スイートも提供しています。単純にこのテスト・スイートを実行し、パスの中でスクリプトに渡します。このテスト・スイートは、`groovy`コマンドとほとんど同じようにスクリプトを実行します。この手法の良いところは、IDEの中でスクリプトが実行できることです。例えばEclipseでは（「メイン・クラスを検索する時には外部jarsをインクルードする」ことを確実にチェックして）、単純にサンプル・プロジェクトに対する新しい実行の構成(run configuration)を作り、次にメイン・クラス`groovy.util.GroovyTestSuite`を探します（図1）。

図1. Eclipseを使ってGroovyTestSuiteを実行する

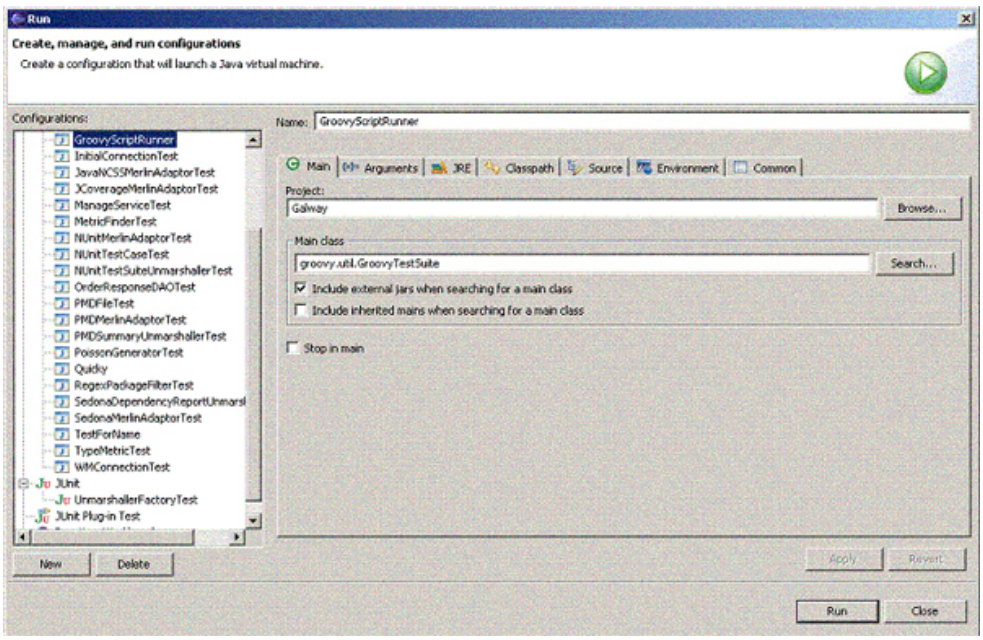
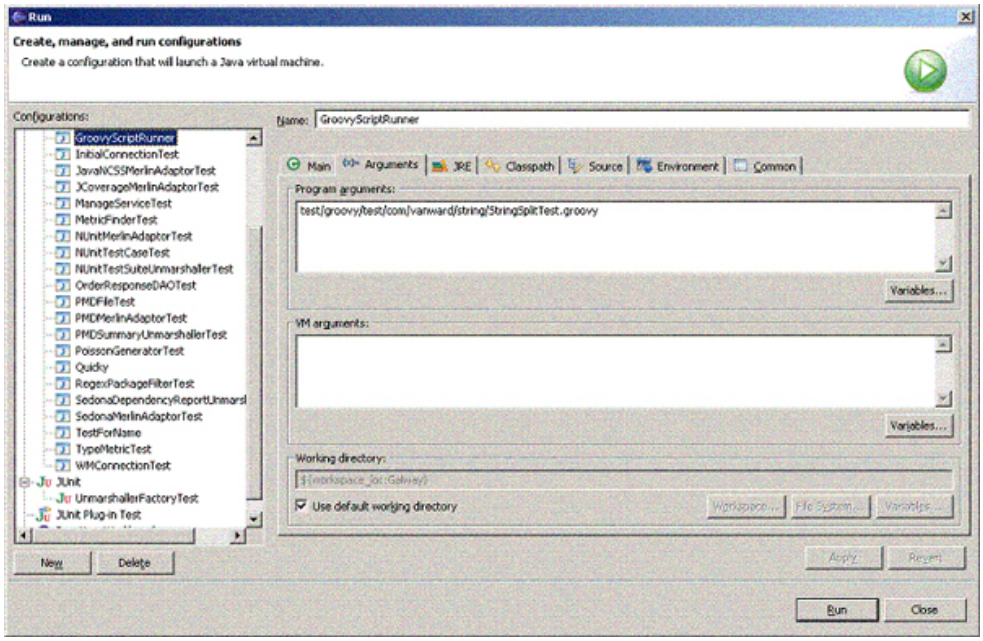


図2を見ると、Argumentsタブをクリックしてスクリプトへのパスを書き込む時に何が起きるかが分かるでしょう。

図2. Eclipseでスクリプトへのパスを規定する



好みのJUnitのGroovyスクリプトを実行するのは、対応する実行の構成をEclipse内で見つけるのと同じくらい、非常に簡単です。

AntとMavenでテストする

JUnitのようなフレームワークの良いところは、人の介在なしにテスト・スイート全体を、ビルドの一部として実行できることです。より多くの人がコード・ベースにテスト・ケースを追加するようになると、テスト・スイート全体は大きくなり、回帰プラットフォーム(regression platform)として最適なものになります。さらに、AntやMavenなどのビルド・フレームワークは、JUnitのバッチ実行を要約するレポート機能を追加しています。

大量のGroovyテスト・ケースをビルドに組み込むための方法として最も単純なのは、通常のJavaバイトコードの中にコンパイルし、AntやMavenが提供する標準のJUnitバッチ・コマンドの中に含めてしまうことです。幸いGroovyには、Groovyスクリプトからバイトコードへのコンパイルを扱えるAntタグがあります。ですから実際に機能するバイトコードにスクリプトを変換するプロセスは、これ以上ないほど単純です。例えばビルドにMavenを利用している場合には、maven.xmlファイルに2つの新しいゴール(goal)を追加し、project.xmlファイルに2つの依存関係を追加し、build.propertiesファイルに1つ単純なフラグを追加すれば、それで全て完了です。

サンプルのスクリプトをコンパイルする、という新しいゴールでmaven.xmlファイルを更新することから始めることにしましょう（ リスト8 ）。

リスト8. Groovycのゴールを定義する、maven.xml fileファイルの例

```
<goal name="run-groovyc" prereqs="java:compile,test:compile">

  <path id="groovy.classpath">
    <pathelement path="${maven.build.dest}"/>
    <pathelement path="target/classes"/>
    <pathelement path="target/test-classes"/>
    <path refid="maven.dependency.classpath"/>
  </path>

<taskdef name="groovyc" classname="org.codehaus.groovy.ant.Groovyc">
   <classpath refid="groovy.classpath"/>
</taskdef>

<groovyc destdir="${basedir}/target/test-classes" srcdir="${basedir}/test/groovy"
         listfiles="true">
<classpath refid="groovy.classpath"/>
</groovyc>

</goal>
```

上記のコードでは幾つかのことが行われています。まず、`run-groovyc`という名前の新しいゴールを定義していますが、このゴールには2つの前提条件があります。サンプルのソース・コードをコンパイルする`java:compile`と、通常のJava-JUnitクラスであれば何でもコンパイルする`test:compile`という2つです。次に`<path>`タグでクラスパスを作っています。この場合、クラスパスはビルド・ディレクトリー（コンパイルされたコードはここにあります）と、それに関連した全ての依存関係（つまりJARファイル）を含んでいます。その次に、`<taskdef>` Antタグで`groovyc`タスクを定義しています。

また私がMavenに対して、クラスパスのどこでクラス`org.codehaus.groovy.ant.Groovyc`を見つけるようにしているか、その方法にも注意してください。この例の最後の行では、`<groovyc>`タグを定義しています。このタグは`test/groovy`ディレクトリーで見つかる全てのGroovyスクリプトをコンパイルし、その結果の.classファイルを`target/test-classes`ディレクトリーに置きます。

重要な詳細の幾つか

Groovyスクリプトをコンパイルして、その結果できるバイトコードを実行するためには、project.xmlファイルから新しい依存関係を2つ（groovyとasm）定義する必要があります（リスト9）。

リスト9. project.xmlファイル用の新しい依存関係

```
 <dependency>
   <groupId>groovy</groupId>
   <id>groovy</id>
   <version>1.0-beta-6</version>
 </dependency>

 <dependency>
   <groupId>asm</groupId>
   <id>asm</id>
   <version>1.4.1</version>
 </dependency>
```

このスクリプトをコンパイルして通常のJavaバイトコードにすれば、任意の標準JUnitランナーで実行することができます。AntとMavenにはランナー・タグがあるので、次のステップとしては、

新しくコンパイルされたGroovyスクリプトをJUnitが拾い上げるようにすることです。MavenのJUnitランナーはパターン一致を使って実行すべきテスト・スイートを見つけるので、特別なフラグをbuild.propertiesファイルに追加して、.javaファイルではなくクラスを検索するようにMavenに対して伝えます（リスト10）。

リスト10. Maven プロジェクトのbuild.propertiesファイル

```
maven.test.search.classdir = true
```

最後に、リスト11に示すように、maven.xmlファイルにテストの`goal`を定義します。こうすることによって、どのユニット・テストが実行するよりも前に、Groovyスクリプトが新しい`run-groovyc`ゴールでコンパイルされるようになります。

リスト11. maven.xmlに対する新しいゴール

```
<goal name="test">
    <attainGoal name="run-groovyc"/>
    <attainGoal name="test:test"/>
  </goal>
```

最後に忘れてならないのは・・・

2つの新しいゴール（一つはスクリプトをコンパイルし、もう一つはJavaとGroovy JUnitの組み合わせテストを実行します。）が定義できたので、あとはこれらを実行して、全てが問題なく動作しているかどうかを確認するだけです。

リスト12を見ると、Mavenを実行して`test`ゴールにパスする時に何が起きるかが分かります。最初に`run-groovyc`ゴールを実行して（これがまた`java:compile`ゴールと`test:compile`ゴールを実行します）から、次にMavenに付属している標準の`test:test`ゴールを実行するのです。`test:test`ゴールが、新しく作られたGroovyスクリプト（この場合では新しくコンパイルされたGroovyスクリプト）と、通常のJava JUnitテストの両方を取り上げることに注意してください。

リスト12. 新しいゴールを実行する

```
$ ./maven test

test:
java:compile:
    [echo] Compiling to /home/aglover/dev/target/classes
    [javac] Compiling 15 source files to /home/aglover/dev/target/classes

test:compile:
    [javac] Compiling 4 source files to /home/aglover/dev/target/test-classes

run-groovyc:
    [groovyc] Compiling 2 source files to /home/aglover/dev/target/test-classes
    [groovyc] /home/aglover/dev/test/groovy/test/RegexFilterTest.groovy
    [groovyc] /home/aglover/dev/test/groovy/test/SimpleFilterTest.groovy

test:test:
    [junit] Running test.RegexFilterTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.656 sec
    [junit] Running test.SimpleFilterTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.609 sec
```

```
    [junit] Running test.SimplePackageFilterTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.578 sec
BUILD SUCCESSFUL
Total time: 42 seconds
Finished at: Tue Sep 21 17:37:08 EDT 2004
```

まとめ

実用的なGroovyの第1回として、この新しいスクリプト言語を使った、最も実用的なアプリケーションの一つについて学んできました。多くの開発者にとってユニット・テストは開発プロセスの重要な一部になりつつありますが、GroovyとJUnitによって、Javaコードのユニット・テストが非常に楽になるのです。

Groovyはアジャイルであることと構文が単純なことから、効果的なJUnitテストを素早く書くための、そしてJUnitテストを自動化ビルドに組み込むための素晴らしいプラットフォームになります。私のようにコード品質に執念を持つ人間には、この組み合わせによって苛立ちが大きく抑えられ、私が最も望むこと、つまり堅牢なソフトウェアを、しかも素早く書く、と言う目標に近づくことができるのです。

次回はGroovyでのAntスクリプト記述に関して解説します。ご期待ください！

関連トピック

- Andy Gloverによる実用的なGroovyの全記事のリストを見てください。
- この記事の著者がGroovy紹介記事の最初としてalt.lang.jreシリーズに寄せた記事「Feeling Groovy」（developerWorks, 2004年8月）をお見逃しなく。
- The JUnit home pageからJUnitをダウンロードしてください。
- The DbUnit home pageからDbUnitをダウンロードしてください。
- The StrutsTestCase home pageからStrutsTestCaseをダウンロードしてください。
- JUnitによるJavaの（そしてつまりGroovyの）アプリケーションのユニット・テストに関して、Malcolm Davisによる「AntとJUnitを用いた漸進的開発」（developerWorks, 2000年11月）で学んでみてください。
- Erik Hatcherが「Automating the build and test process」（developerWorks, 2001年8月）の中で、XP涅槃の境地に一歩近づくためにはAntとJUnitをどのように組み合わせるべきかを解説しています。
- MavenはAntに代わるものとして、特にプロジェクト管理で効果を発揮します。Mavenについてさらに詳しく知るために、Charles Chanによる「プロジェクト管理: Mavenでもっと簡単に」（developerWorks, 2003年4月）を読んでください。
- DbUnitについてさらに詳しく知るために、Philippe Girolamによる「DbUnitとAnthillによるテスト環境の制御」（developerWorks, 2004年4月）を読んでください。br>
- DdUnitについてはそれ以外にも、Andrew Gloverによる「Effective Unit Testing with DbUnit」（OnJava, 2004年2月刊）があります。
- Bitter Java andBetter, Faster, Lighter Javaの著者Bruce Tateはアジャイル性の主張者であり、いつ見ても興味深いBlogがあります。
- Groovyの最も強力な機能の一つがアジャイル性です。アジャイル開発（またはXP）の下にある原理についてさらに詳しく知るために、Roy Millerによるエクストリーム・プログラミングの神秘を解くシリーズを読んでください（developerWorks, 2002年8月）。
- Richard HightowerとNicholas LesieckiによるJava tools for extreme programmingは、Javaプラットフォームでのアジャイル開発に関する技術者用ガイドです。
- Just GroovyはGroovyに特化したWebサイトで、正にGroovyのことばかりを集めています。
- developerWorksのJava technologyゾーンには、Javaプログラミングのあらゆる面に関する記事が豊富に用意されています。
- Developer BookstoreにはJava関連の技術書をはじめ、広範な話題に関する書籍が豊富に用意されていますので、ご覧ください。