Функциональное тестирование и сбор покрытия – версия Java

Для выполнения работы необходимы следующие инструменты:

- Java последней версии, можно скачать с http://java.oracle.com
- Apache ant, можно скачать с http://ant.apache.org
- Junit, http://junit.org
- Jcov , https://wiki.openjdk.java.net/display/CodeTools/jcov
- Средства работы с командной строкой (в Windows можно использовать Far Manager или Cygwin, в Unix системах – обычный shell)

1 Первоначальная настройка программной среды.

Необходимо загрузить и установить Java. Для этого нужно скачать с сайта java.oracle.com версию, подходящую к вашей операционной системе.

Затем создайте каталог, в котором будут храниться инструменты. Скачайте и распакуйте туда Apache ant, можно скачать с ant.apache.org (предположим, что это apache-ant-1.9.4)

Скачайте с сайта junit.org файлы hamcrest-core-1.3.jar и junit-4.11.jar (номера версий могут быть новее). Скопируйте эти файлы в каталог apache-ant-1.9.4/lib

Установите переменные среды JAVA_HOME и ANT_HOME, например

SET JAVA_HOME=C:\Progra~1\Java\jdk1.8.0_25

SET ANT_HOME=C:\tools\apache-ant-1.9.4-bin\apache-ant-1.9.4

достаточно удобно делать это непосредственно в файле ant.bat (windows) или ant (linux).

Скачайте и установите JCov с сайта https://wiki.openjdk.java.net/display/CodeTools/jcov

2 Написание простого проекта на ant

Для начала работы необходимо написать код, который мы будем тестировать. Предположим, мы пишем функцию, складывающую два числа.

```
public class Sample {

    public static long mul(int a, int b) {
        return a * b;
    }

    public static void main(String... args) {
        System.out.println(mul(2, 2));
    }

}
```

Скачайте и распакуйте пример с адреса http://stepanoff.info/testing/materials/java_lab2/sample_1_1.zip

Рассмотрим пример подробнее. Он состоит из трех файлов – Sample.java с кодом, а также из файла build.properties , содержащего некоторые параметры и файла build.xml . определяющего инструкции ant для сборки.

Перейдите в каталог с файлом build.xml и запустите в нем ant. Если этап начальной настройки прошел успешно, то соберется jar файл в каталоге dist. Дайте команду **ant run** . Вы должны увидеть

```
run:
     [java] 4
```

Рассмотрим подробнее структуру файла build.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project name="" default="default" basedir="." >
    <property file="build.properties" />
    <target name="init">
        <mkdir dir="${src.dir}" />
        <mkdir dir="${build.dir}" />
        <mkdir dir="${dist.dir}" />
    </target>
    <target name="compile" depends="init">
        <javac srcdir="${src.dir}" destdir="${build.dir}" debug="yes" includeantruntime="yes">
        </javac>
    </target>
    <target depends="compile" name="dist">
        <dirname file="${dist.file}" property="dist.dir"/>
        <mkdir dir="${dist.dir}"/>
        <jar compress="${jar.compress}" jarfile="${dist.dir}/${dist.file}">
            <fileset dir="${build.dir}"/>
            <manifest>
                <attribute name="Main-Class" value="Sample"/>
            </manifest>
        </jar>
    </target>
    <target name="run" depends="dist">
        <java jar="${dist.dir}/${dist.file}" fork="yes"/>
    </target>

    <target name="default" depends="dist"/>
</project>
```

Этот файл содержит инструкции программы ant для сборки проекта. Ant имеет точки входа или цель - <target>, указывающие на конкретные логические действия. Точка входа по-умолчанию называется «dist” и объявлена в свойствах тега «project”. Для каждой цели указывается, от каких других целей она зависит в свойства «depends” . Таким образом, dist зависит от compile, compile от init . Когда мы запускаем ant без параметров, он строит дерево зависимостей и начинает запускать цели в правильлном порядке, в данном случае, сперва init , потом compile и, наконец, dist. Соответственно, ant можно вызвать со следующими командами:

команда	**Выполняемые цели**
ant	init compile run
ant init	init
ant compile	init compile
ant dist	init compile dist
ant run	init compile dist run

Цель init содержит крайне простые инструкции – создание нескольких каталогов, которые будут использоваться для сборки

```
<target name="init">
    <mkdir dir="${src.dir}" />
    <mkdir dir="${build.dir}" />
    <mkdir dir="${dist.dir}" />
</target>
```

Каждая команда mkdir создает каталог с соответствующим именем. Имена здесь являются ссылками на файл build.properties

Цель compile вызывает компилятор java – javac и передает ему на компиляцию каталог src.dir, указанный в build.properties.Результирующие class-файлы будут лежать в build.dir

```
<target name="compile" depends="init">
    <javac srcdir="${src.dir}" destdir="${build.dir}" debug="yes" includeantruntime="yes">
    </javac>
</target>
```

Цель dist вызывает архиватор jar

```
<target name="dist" depends="compile">
    <dirname file="${dist.file}" property="dist.dir"/>
    <mkdir dir="${dist.dir}"/>
    <jar compress="${jar.compress}" jarfile="${dist.dir}/${dist.file}">
        <fileset dir="${build.dir}"/>
        <manifest>
            <attribute name="Main-Class" value="Sample"/>
        </manifest>
```

```
    </jar>
  </target>
```

Эта цель несколько длиннее, чем предыдущие. Она состоит из вызова собственно архиватора jar с указанием каталогов исходных файлов и каталога, в который мы хотим поместить jar-файл, а также из описания манифеста – в данном случае мы указываем, какой класс (Main-Class) содержит в себе точку входа в программу для запуска java –jar

Наконец, цель run тривиально запускает получившийся jar в отдельном процессе (fork=”yes”)

```
  <target name="run" depends="dist">
    <java jar="${dist.dir}/${dist.file}" fork="yes"/>
  </target>
```

2 Написание простого теста jUnit для проекта на ant

Скачайте и распакуйте пример с адреса http://stepanoff.info/testing/materials/java_lab2/sample_1_2.zip

Вы можете увидеть отличия в проекте – добавился каталог tests и увеличился файл buid.xml, который теперь содержит две новые цели – compile-tests и test

Цель compile-tests абсолютно аналогична цели compile – она компилирует файлы из каталога test.dir, заданного в build.properties

```
  <target name="compile-tests" depends="init">
    <javac srcdir="${tests.dir}" destdir="${build.dir}" debug="yes" includeantruntime="yes">
    </javac>
  </target>
```

Зато следующая цель – test – содержит в себе новый тип задачи – junit.

```
  <target name="test" depends="compile,compile-tests">
    <junit printsummary="yes" haltonfailure="yes">
      <classpath>
        <pathelement location="${build.dir}"/>
        <pathelement path="${java.class.path}"/>
      </classpath>
      <formatter type="plain"/>
```

```
    <batchtest fork="yes" todir="${reports.tests}">
      <fileset dir="${tests.dir}">
        <include name="**/*Test*.java"/>
      </fileset>
    </batchtest>
  </junit>
</target>
```

Данная цель берет все тесты, соответствующие шаблону *="**/*Test*.java* в каталоге test.dir и запускает, записывая отчет в каталог reports.tests . В данном случае это будет текстовый файл, но можно добиться и более красивых html отчетов.

Рассмотрим файл SampleTest.java . В нем содержится 9 функций, помеченных аннотацией @Test . По этой аннотации JUnit поймет, что именно эти функции необходимо запускать при тестировании. Тесты заканчиваются командой assertEquals. Эта команда является стандартной для проверки результата теста. Если два последних ее аргумента совпадают, она продолжает исполнение, иначе генерирует исключение AssertionError, которое в дальнейшем перехватывается оболочкой jUnit

Запустить тестирование можно командой ant test . Ant сперва скомпилирует программу, потом тесты и, наконец, эти тесты выполнит.

Запустите тестирование. 5 из 9 тестов упадут. Посмотрите на отчет в каталоге reports. Попробуйте найти ошибки в коде Sample.java и их исправить.

3 Анализ покрытия с помощью JCov

Скачайте и распакуйте пример с адреса http://stepanoff.info/testing/materials/java_lab2/sample_1_3.zip

Воспользуемся методом статической инструментации кода для сбора покрытия. Для этого нам необходимо после этапа компиляции продукта вызвать jcov с параметром Instr.

В файл build.properties добавим атрибут, указывающий путь к jcov, например

jcov.dir=C:\\tools\\jcov_2.0

обратите внимание, что на Windows слеши должны быть двойными

Далее, создадим цель instr

```
<target name="instr" depends="compile">
  <java jar="${jcov.dir}/jcov.jar" fork="yes">
    <arg value="Instr"/>
    <arg value="${build.dir}"/>
```

```
        </java>
      </target>
```

Фактически, эта цель представляет собой вызов java –jar jcov.jar Instr для каталога build.dir . После выполнения этой команды файлы будут специальным образом инструментированы

Следующая цель – собственно тестирование. Она отличается от цели test только наличием дополнительного файла в класспути

```
  <target name="cov" depends="instr,compile-tests">
    <junit printsummary="yes" haltonfailure="no">
      <classpath>
        <pathelement location="${build.dir}"/>
        <pathelement path="${java.class.path}"/>
        <pathelement path="${jcov.dir}/jcov_file_saver.jar"/>
      </classpath>
      <formatter type="plain"/>
      <batchtest fork="yes" todir="${reports.tests}">
        <fileset dir="${tests.dir}">
          <include name="**/*Test*.java"/>
        </fileset>
      </batchtest>
    </junit>
  </target>
```

После выполнения этой цели в текущем каталоге будет создан файл result.xml

Наконец, по окончании тестирования должна запускаться цель cov-report

```
      <target name="cov-report" depends="cov">
        <java jar="${jcov.dir}/jcov.jar" fork="yes">
          <arg value="RepGen"/>
          <arg value="result.xml"/>
        </java>
      </target>
```

Эта цель запускает генератор репортов RepGen.

Выполните команду ant cov-report и убедитесь, что в каталоге report появился отчет о покрытии.