Project Unit Resources and Development Model

Reference Guide

Liu Jianhong <flowas@gmail.com>

Project Unit Resources and Development Model: Reference Guide

by Liu Jianhong

1.0.0-SNAPSHOT

Table of Contents

List of Figures

List of Examples

Introduction

The goal of Project Unit is to provide a Self-growth, portable and modular development advocacy,so that duplication, conflict, sticky and cumbersome to be diluted or eliminated. Coupled with standardized structure and process, is indeed a wonderful direction.

Chapter 1. Installation

To use the Project Unit, you need to put the JAR on the classpath of your project. Most of the features of Project Unit are enabled automatically when it's added to the classpath. Some extra configuration, covered below.

Maven dependency configuration

If you are using Maven [http://maven.apache.org/] as your build tool, you can add the following single dependency to your pom.xml file to include Project Unit:

```
<dependency>
   <groupId>net.flowas.codegen</groupId>
   <artifactId>project-unit</artifactId>
   <version>${projectunit.version}</version>
</dependency>
```

Tip

Substitute the expression ${projectunit.version} with the most recent or appropriate version of Project Unit.

Distribution Downloads

You can download zipped archive from github [https://github.com/includeeasy/projectUnit/blob/master/forge-mock/lib/ant-1.8.1.jar] ,and extract it to a directory,copy all jars in 'forge-1.0.0-SNAPSHOT\lib' to your project's classpath.

Run the Tool

Change directory to "bin/" and run "forge.bat" on Windows "forge.sh" on Linux.When execute succeed,a window like following will appear:

Figure 1.1. Seam forge screenshots

In java project,you can run the flowing code to start the software:

```
    public class Main
```

```
{
   public static void main(final String[] args)
   {
   System.setProperty("jline.terminal", "jline.UnsupportedTerminal");
      StartMain.main(args);
   }

}
```

Chapter 2. Building block style development

You can put some wars,jars,zips,code fragment and regular process together to build a project.

Figure 2.1. Has logon module

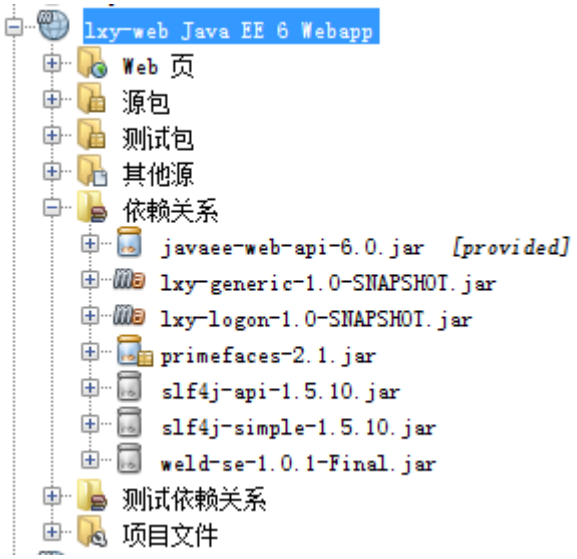


Figure 2.2. No logon module

Consider to use CDI+EL,Metawidget,Multimodal Architecture and Interfaces,XProc,After-Processing Pattern,s-ramp

Chapter 3. Mock plugin for seam forge

The plugin is designed to let programmers write only logic code, which automatically generates code

Scope

Frameworks are used : JUnit,EasyMock,Mockito,PowerMock.

Witch class can be mocked : normal class,static method,singleton class.

Furthermore, The plugin can generate code for mock java.lang.Runtime,javax.persistence.Persistence,and so on.when mock those class the plugin generate all code for execute command ,operate database,use ftp and so on.

commands

Insert mock code into test class.

```
mock fromXML|editeXML --file --class
```

fromXML	Specify system insert mock code according to a XML file,and such you must specify a XML file
editXML	Popup a swing window that allow you to specify which class to be mocked,lastly,ti generate an XML file.
--file	Specify a XML file that list some classes to be mocked
--class	Full qualified class name to be mocked,generally,produce a demo mock files for the class

Demonstrate

Example 3.1. Insert code to an exist test class

We mock java.io.File,java.lang.Runtime and javax.persistence.Persistence(That is mock an database),suppose we has an xml file named "demo.xml" and it's context is the following:

```
<root>
 <test-class>net.flowas.demo.MyTest</test-class>
 <system-under-test>
 </system-under-test>
 <depend-on-component>
  <class name="java.io.File">
   <method>exists()</method>
  </class>
  <class name="java.lang.Runtime">
      <method>exec(java.lang.String)</method>
  </class>
  <class name="javax.persistence.Persistence">
      <method>createEntityManagerFactory(java.lang.String)</method>
  </class>-->
 </depend-on-component>
</root>
```

Figure 3.1. We create a project that named "mockapp"

```
Windows? Really? Okay...
[org.springframework.roo.root] bin $ new-project --named mockapp --topLevelPacka
ge net.flowas.mockapp
Use [/webtools/spring/roo/git/roo/src/site/docbook/forge-1.0.0-SNAPSHOT/bin/mock
app] as project directory? [Y/n] Y
Wrote /webtools/spring/roo/git/roo/src/site/docbook/forge-1.0.0-SNAPSHOT/bin/moc
kapp/src/main/java/net/flowas/mockapp/HelloWorld.java
Wrote /webtools/spring/roo/git/roo/src/site/docbook/forge-1.0.0-SNAPSHOT/bin/moc
kapp/src/main/resources/META-INF/forge.xml
***SUCCESS*** Created project [mockapp] in new working directory [mockapp]
[mockapp] mockapp $ mock fromXML --file demo.xml
```

Figure 3.2. That produces some files as following

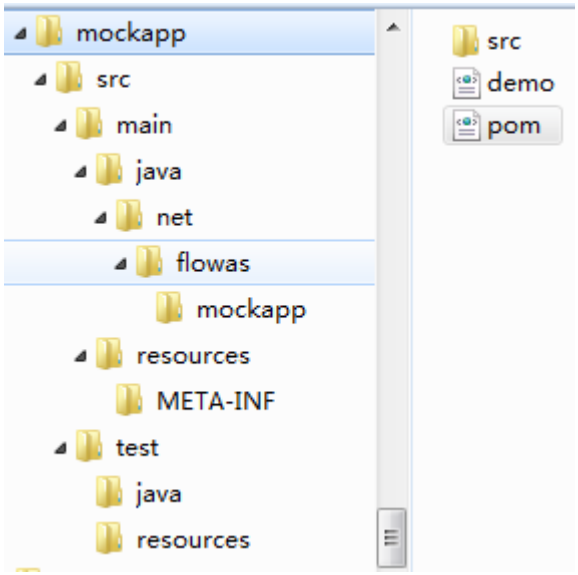
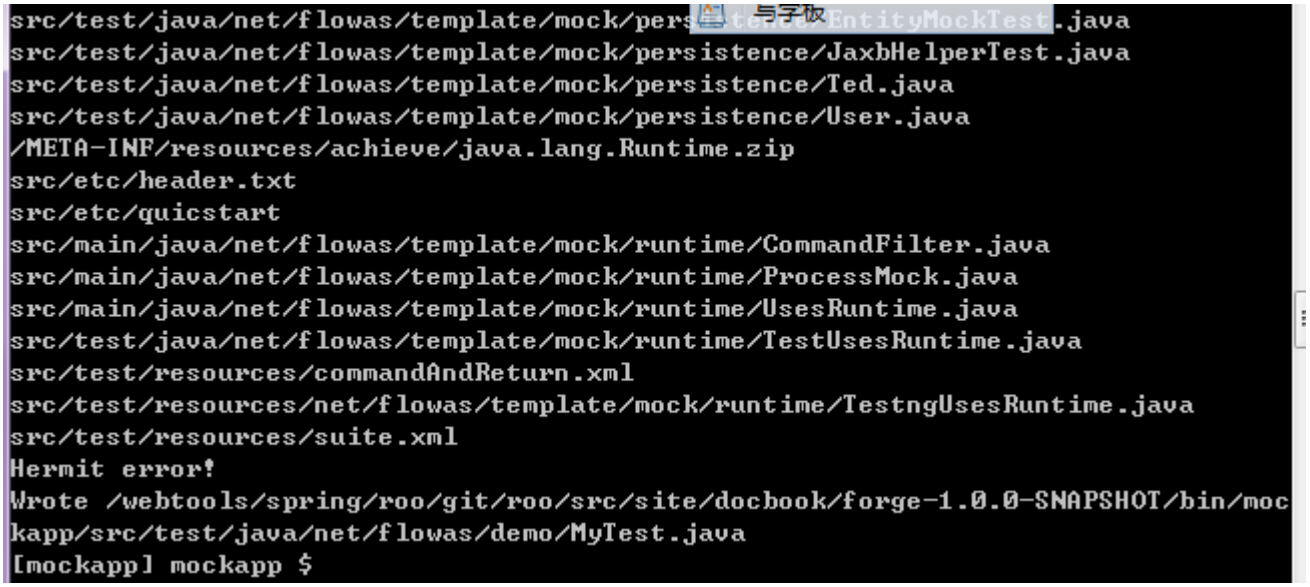


Figure 3.3. Execute "mock fromXML --file demo.xml" and now produces many files



Finally,three test class are generated:

net.flowas.demo.MyTest
net.flowas.template.mock.persistence.EntityMockTest
net.flowas.template.mock.runtime.TestUsesRuntime

Use code generation in IDE

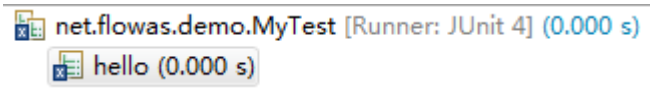
Run launcher.Main ,and import generated project as maven project

Create a test class named "net.flowas.demo.MyTest",

```
import java.io.File;
import java.io.InputStream;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import junit.framework.Assert;
import org.junit.Before;
import org.junit.Test;
public class MyTest {
 @Before
 public void setUp() throws Exception {
 }
 @Test
 public void hello() throws Exception {
  File file = new File(".");
  Assert.assertFalse(file.exists());
  Process process = Runtime.getRuntime().exec("ant --verrsion");
  InputStream in = process.getInputStream();
  byte[] b = new byte[in.available()];
```

```
  in.read(b);
  Assert.assertEquals("1.8.1", new String(b));
  EntityManagerFactory emf = Persistence
    .createEntityManagerFactory("ggfortest");
  EntityManager em = emf.createEntityManager();
  List list = em.createQuery("select u from User as u").getResultList();
  Assert.assertTrue(list.size()>0);
 }
}
```

Figure 3.4. Now you run "MyTest" as JUnit bound to fail



Execute "mock fromXML --file demo.xml" and reopen "net.flowas.demo.MyTest"

```
import java.io.File;
import java.io.InputStream;
import java.lang.reflect.Method;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import junit.framework.Assert;
import net.flowas.template.mock.persistence.DataInitializer;
import net.flowas.template.mock.runtime.CommandFilter;
import net.flowas.template.mock.runtime.ProcessMock;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.mockito.invocation.InvocationOnMock;
import org.mockito.stubbing.Answer;
import org.powermock.api.easymock.PowerMock;
import org.powermock.api.mockito.PowerMockito;
import org.powermock.core.classloader.annotations.PrepareForTest;
import org.powermock.modules.junit4.PowerMockRunner;

@RunWith(PowerMockRunner.class)
@PrepareForTest({ File.class,Persistence.class })
public class MyTest {
 @Before
 public void setUp() throws Exception {
  Method method = PowerMock.method(Persistence.class,
    "createEntityManagerFactory", String.class);
  Method expectedMethod = PowerMock.method(DataInitializer.class,
    "createEntityManagerFactory", String.class);
  PowerMock.replace(method).with(expectedMethod);
  PowerMock.replayAll();
  InputStream in = this.getClass().getResourceAsStream("/data.xml");
  DataInitializer.insert(in);
  PowerMockito.mockStatic(Runtime.class);
  Runtime mockedRuntime = PowerMockito.mock(Runtime.class);
```

```
  PowerMockito.when(Runtime.getRuntime()).thenReturn(mockedRuntime);
  final CommandFilter commandFilter = new CommandFilter() {
   @Override
   public String[] exec(String command) {
    if (command.equals("ant --verrsion")) {
     return new String[] { "1.8.1", "" };
    }
    return null;
   }
  };
  Answer<Process> answer = new Answer<Process>() {
   @Override
   public Process answer(InvocationOnMock invocation) throws Throwable {
    return new ProcessMock((String) invocation.getArguments()[0],
      commandFilter);
   }
  };
  PowerMockito.when(mockedRuntime.exec(Mockito.anyString())).thenAnswer(
    answer);
  Answer<File> answerFile = new Answer<File>() {
   @Override
   public File answer(InvocationOnMock invocation) throws Throwable {
    Object[] arguments = invocation.getArguments();
    File mockFile = new File((java.lang.String) arguments[0]);
    File spy = PowerMockito.spy(mockFile);
    PowerMockito.when(spy.exists()).thenReturn(false);
    return spy;
   }
  };
  PowerMockito.whenNew(File.class).withArguments(Mockito.anyString())
    .thenAnswer(answerFile);
 }

 @Test
 public void hello() throws Exception {
  File file = new File(".");
  Assert.assertFalse(file.exists());
  Process process = Runtime.getRuntime().exec("ant --verrsion");
  InputStream in = process.getInputStream();
  byte[] b = new byte[in.available()];
  in.read(b);
  Assert.assertEquals("1.8.1", new String(b));
  EntityManagerFactory emf = Persistence
    .createEntityManagerFactory("ggfortest");
  EntityManager em = emf.createEntityManager();
  List list = em.createQuery("select u from User as u").getResultList();
  Assert.assertTrue(list.size()>0);
 }
}
```

Figure 3.5. Now you run "MyTest" as JUnit likely to be successful