Maven3 实战笔记 10 使用 Maven 进行测试

刘岩

Email:suhuanzheng7784877@163.com

1. 测试简介

这里所说的测试主要是指单元测试，一般说 Java 的单元测试都知道有个 Junit。确实目前很多公司，无论什么行业，基本都是使用 junit 进行单元测试，一般银行、电信、股票项目每个功能类都必须有相应的单元测试类，而且测试用例也是极其苛刻的。而且每一行代码都需要有 log 追踪。生成的测试报告也要成为相应的成果物交与甲方。稍微差一点的，单元测试也就是一个形式，因为很多情况（由以国内政务项目为主）都是业务逻辑集成 UI 功能都做完了，之后再回去补单元测试，其实是为了应付甲方合同中的测试报告。话说多了，偏离了主题。Maven 构建项目的时候可以执行我们的单元测试，底层是调用了 Maven 的插件 maven-surefire-plugin 执行单元测试用例。

2. 新模块的需求与开发

再展示 Maven 的测试功能之前，我们先来开发一个新的模块，随机验证码的功能模块。随机验证码我们在网上都用过，在此不再详细说了，直接看代码即可。此处用到了一个 google 开源的组件。这个在笔者的另一篇 blog

http://suhuanzheng7784877.iteye.com/blog/1076066 中有介绍，在此不再赘述。

先来看生成随机数业务代码

AccountImageUtil

```
package com.liuyan.account.mail;

import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.Random;
```

```
import com.google.code.kaptcha.impl.DefaultKaptcha;
import com.google.code.kaptcha.util.Config;

public class AccountImageUtil {

    static DefaultKaptcha defaultKaptcha = new DefaultKaptcha();
    static Map<Integer, String> map;
    static int key = 0;
    static Random random;
    static {
        defaultKaptcha.setConfig(new Config(new Properties()));
        map = new HashMap<Integer, String>();
        random = new Random();
    }

    public static String generateText() {
        StringBuffer sb = new StringBuffer();

        for (int i = 0; i < 4; i++) {
            int a = random.nextInt(10);
            sb.append(a);
        }
        map.put(key, sb.toString());
        key++;
        return sb.toString();
    }

}
```

以上业务辅助类初始化了图片生成组件，之后调用随机函数类生成四位的随机数。

再来看看生成图片的类 AccountImageServiceImpl，接口不再给出了

```
package com.liuyan.account.mail;

import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.imageio.ImageIO;
```

```java
public class AccountImageServiceImpl implements AccountImageService
{

    @Override
    public byte[] generateText() {

        BufferedImage bufferedImage = AccountImageUtil.defaultKaptcha
                .createImage(AccountImageUtil.generateText());

        ByteArrayOutputStream out = new ByteArrayOutputStream();

        try {
            ImageIO.write(bufferedImage, "jpg", out);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        File file = new File("c:/1.jpg");
        try {
            FileOutputStream fileOut = new FileOutputStream(file);
            fileOut.write(out.toByteArray());
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        return out.toByteArray();

    }

    public String getText(Integer key) {
        String value = AccountImageUtil.map.get(key);
        return value;
    }

    @Override
    public boolean validateText(Integer key, String text) {

        String value = AccountImageUtil.map.get(key);
```

```
        return value.equals(text);
    }
}
```

这个类就是调用辅助类生成图片的。下面我们编写单元测试。

3. 新模块的测试

AccountImageUtilTest 代码如下

```
package com.liuyan.account.mail;
import static org.junit.Assert.assertFalse;
import org.junit.Test;

public class AccountImageUtilTest {

    @Test
    public void testGenerateText() {
        String value = AccountImageUtil.generateText();
        assertFalse("1111".equals(value));
    }

}
```

AccountImageServiceImplTest 如下

```
package com.liuyan.account.mail;
import static org.junit.Assert.*;
import org.junit.Test;
public class AccountImageServiceImplTest {

    AccountImageServiceImpl accountImageServiceImpl = new
AccountImageServiceImpl();

    @Test
    public void testGenerateText() {

        byte[] b = accountImageServiceImpl.generateText();

        assertTrue(b != null);

    }

    @Test
    public void testValidateText() {
        String text = accountImageServiceImpl.getText(0);
```

```
        System.out.println("--------------------------------" +
text);
        assertTrue(!text.equals("0000"));
    }

}
```

之后在控制台运行如下指令

```
mvn test
```

之后效果如下

```
-------------------------------------------------------

 T E S T S

-------------------------------------------------------

Running com.liuyan.account.mail.AccountImageServiceImplTest

--------------------------------1990

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.926 sec

Running com.liuyan.account.mail.AccountImageUtilTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec



Results :



Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
```

运行报告是 junit 自己的报告输出，和咱们在 Eclipse 运行的报告差不多。以上代表运行了 3 个用例，和预期效果不符的是 0 个，失败的用例是 0 个，忽略的用例数是 0 个。

如果需要跳过单元测试，则可以运行如下命令

```
mvn package -DskipTests
```

大家可能要问，为何 Maven 能够自己寻找我们编写的测试类呢？其实还是那句约定大于配置。Maven 自动去寻找 src/test/java 下面的类，当此文件夹下面的类符合以下规范，那么 Maven 默认认为他们是单元测试用例类。

Test*.java:任何目录下以 Test 为开始的类

*Test.java: 任何目录下以 Test 为结尾的类

*TestCase.java: 任何目录下以 TestCase 为结尾的类。

如果想在一段时间内节省项目构建时间，暂时全部忽略单元测试。那么可以在 pom.xml 中配置如下

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.5</version>
            <configuration>
                <skipTests>true</skipTests>
            </configuration>
        </plugin>
    </plugins>
</build>
```

等到项目完全开发完了，需要测试用例的时候将其注释掉即可。

本个模块有两个测试用例类，如果仅仅想运行一个测试用例该怎么办。运行下面命令

```
test -Dtest=AccountImageServiceImplTest
```

这个是指定具体运行哪个测试用例。当然需要将 pom 文件中忽略测试用例的配置注释掉。

也可以测试多个测试用例

```
mvn test -Dtest=AccountImageServiceImplTest,AccountImageUtilTest
```

也可以使用模糊匹配进行测试

```
mvn test -Dtest=*Test
```

我们也可以通过 pom 文件配置我们想要测试的类与不想测试的类

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.5</version>
            <configuration>
                <includes>
                    <include>**/*Test.java</include>
                </includes>
                <excludes>

<exclude>**/AccountImageUtilTest.java</exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>
```

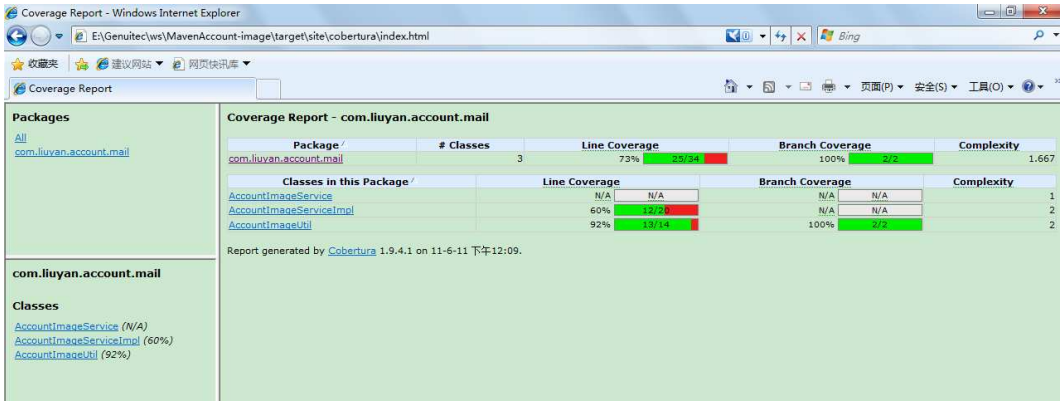
includes 是需要测试的类，excludes 是要排除之外测试用例。可以使用模糊匹配。**用来匹配任意文件路经，*匹配任意类。

4. 测试报告

基本的测试报告上面已经介绍过了，下面我们看看测试覆盖率的报告。运行如下命令

```
mvn cobertura:cobertura
```

在 target 文件夹下出现了一个 site 目录，下面是一个静态站点，里面就是单元测试的覆盖率报告。



5. 总结

这次我们介绍了 Maven 的测试，可以运行项目的单元测试用例，并生成报告。使用者可以根据自己的需要配置测试选项以满足项目的测试需求。最后说一下，测试十分重要，往往大手笔的产品测试人员和开发人员的比例是 2:1。