# **一、前言**

在二十三天中我们介绍了使用maven来下载工程的依赖库文件，用ant来进行war包的建立。今天我们在这个基础上将使用junit+dbunit来进行带有单元测试报告的框架的架构。

目标：

1. 每次打包之前自动进行单元测试并生成单元测试报告
2. 生成要布署的打包文件即war包
3. 单元测试的代码不能够被打在正式的要布署的war包内，单元测试仅用于unit test用
4. 使用模拟数据对dao层进行测试，使得dao方法的测试结果可被预料

# **二、Junit+Ant生成的单元测试报告**





上面是一份junit生成的测试报告，它可以与ant任务一起运行然后自动生成这么一份html的测试报告，要生成这样的一份junit test report我们需要调用ant任务中的<junitreport>这个task，示例代码如下：

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. <target name="junitreport">
2. <junit printsummary="on" haltonfailure="false" failureproperty="tests.failed" showoutput="true">
3. <classpath>
4. <pathelement path="${dist.dir}/${webAppQAName}/WEB-INF/classes" />
5. <fileset dir="${lib.dir}">
6. <include name="\*.jar" />
7. </fileset>
8. <fileset dir="${ext-lib.dir}">
9. <include name="\*.jar" />
10. </fileset>
11. </classpath>
12. <formatter type="xml" />
13. <batchtest todir="${report.dir}">
14. <fileset dir="${dist.dir}/${webAppQAName}/WEB-INF/classes">
15. <include name="org/sky/ssh/ut/Test\*.\*" />
16. </fileset>
17. </batchtest>
18. </junit>
19. <junitreport todir="${report.dir}">
20. <fileset dir="${report.dir}">
21. <include name="TEST-\*.xml" />
22. </fileset>
23. <report format="frames" todir="report" />
24. </junitreport>
25. <fail if="tests.failed">
26. ---------------------------------------------------------
27. One or more tests failed, check the report for detail...
28. ---------------------------------------------------------
29. </fail>
30. </target>

在一般的产品级开发时或者是带有daily building/nightly building的项目组中我们经常需要检查最新check in的代码是否影响到了原有的工程的编译，因为每天都有程序员往源码服务器里check in代码，而有时我们经常会碰到刚刚被check in的代码在该程序员本地跑的好好的，但是check in源码服务器上后别人从源码服务器“拉”下来的最新代码跑不起来，甚至编译出错，这就是regression bug，因此我们每天的打包要干的事情应该是：

1. 程序员check in代码时必须把相关的unit test也check in源码服务器
2. 次日的零晨由持续集成构件如：cruisecontrol自动根据设好的schedule把所有的源码服务器的代码进行编译
3. 运行单元测试
4. 生成报告
5. 打包布署到QA服务器上去

如果考究点的还会生成一份“单元测试覆盖率”报告。

那么有了这样的单元测试报告，项目组组长每天早上一上班检查一下单元测试报告就知道昨天代码check in的情况，有多少是成功多少是失败，它们分别是哪些类，哪些方法，以找到相关的负责人。

同时，有了单元测试报告，如果测试报告上显示的是有fail的地方，该版本就应被视之为fail，不能被送给QA进行进一步的测试，直到所有的单元测试成功才能被送交QA。

# **三、如何在Spring下书写一个单元测试方法**

## **3.1使用spring的注入特性书写一个单元测试**

Spring是一个好东西，一切依赖注入，连单元测试都变成了依赖注入了，这省去我们很多麻烦。

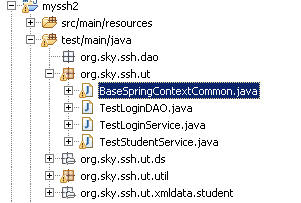
我们可以将web工程中的applicationContext、Datasource甚至iBatis或者是Hibernate的配署都可以注入给junit，这样使得我们可以用IoC的方法来书写我们的单元测试类。

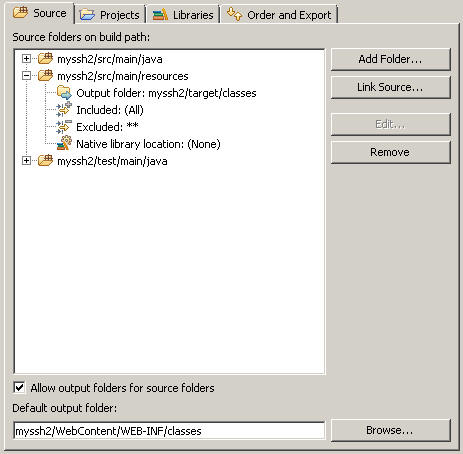
此处，我们使用的junit为4.7, 而相关的spring-test库文件为3.1，我都已经在pom.xml文件中注明了.

我们先在eclipse里建立一个专门用来放单元测试类的src folder：test/main/java。

****注意一下单元测试类的coding convention：****

* *****所有的测试类必须以Test开头*****
* *****所有的测试方法名必须为public类型并且以test开头*****
* *****所有的测试类全部放在test/main/java目录下，不可和src/main/java混放*****





### **类 org.sky.ssh.ut.BaseSpringContextCommon**

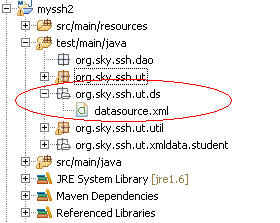
**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **package** org.sky.ssh.ut;
3. **import** org.junit.runner.RunWith;
4. **import** org.springframework.test.context.ContextConfiguration;
5. **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
6. **import** org.springframework.test.context.transaction.TransactionConfiguration;
7. **import** org.springframework.transaction.annotation.Transactional;
9. @RunWith(SpringJUnit4ClassRunner.**class**)
10. @ContextConfiguration({ "/spring/appconfig/applicationContext.xml", "/org/sky/ssh/ut/ds/datasource.xml",
11. "/spring/hibernate/hibernate.xml" })
12. **public** **class** BaseSpringContextCommon {
13. }

该类为一个基类，我们所有的单元测试类全部需要继承自该类，大家可以把这个类认为一个spring的context加载器，****注意这边的datasource.xml。****

因为我们在做测试方法时势必会涉及到对一些数据进行操作，因此我们在数据库里除了平时开发和布署用的数据库外，还有一个专门用于运行“单元测试”的“单元测试数据库”或者“单元测试数据库****实例****”，因此我们在单元测试时会把我们当前的数据库连接“硬”指向到“单元测试用数据库”上去.

这个datasource.xml文件位于/org/sky/ssh/ut/ds目录下，见下图（当然它也必须被放在test/main/java目录里哦：



该文件内容如下：

### **org.sky.ssh.ut.ds.datasource.xml**

**[html]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns:p="http://www.springframework.org/schema/p" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
4. xmlns:context="http://www.springframework.org/schema/context" xmlns="http://www.springframework.org/schema/beans"
5. xsi:schemaLocation="
6. http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
7. http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
8. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
9. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd"**>**



14. **<bean** class="org.springframework.jdbc.core.JdbcTemplate" p:dataSource-ref="dataSource" **/>**
16. <!-- configure data base connection pool by using JNDI -->
17. <!--
18. **<bean** id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean"**>**
19. **<property** name="jndiName"**>**
20. **<value>**${jdbc.jndiname}**</value>**
21. **</property>**
22. **</bean>**
24. --**>**
25. <!-- configure data base connection pool by using C3P0 -->
27. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close"**>**
29. **<property** name="driverClass" value="${jdbc.driverClassName}" **/>**
31. **<property** name="jdbcUrl" value="${jdbc.databaseURL}" **/>**
33. **<property** name="user" value="alpha\_test" **/>**
35. **<property** name="password" value="password\_1" **/>**
37. **<property** name="initialPoolSize" value="10" **/>**
39. **<property** name="minPoolSize" value="10" **/>**
41. **<property** name="maxPoolSize" value="15" **/>**
43. **<property** name="acquireIncrement" value="1" **/>**
45. **<property** name="maxIdleTime" value="5" **/>**
46. **</bean>**
48. **<bean** id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
50. **<property** name="dataSource" ref="dataSource" **/>**
51. **</bean>**
53. **<tx:advice** id="txAdvice" transaction-manager="transactionManager"**>**
55. **<tx:attributes>**
57. **<tx:method** name="submit\*" propagation="REQUIRED" rollback-for="java.lang.Exception" **/>**
59. **<tx:method** name="add\*" propagation="REQUIRED" rollback-for="java.lang.Exception" **/>**
61. **<tx:method** name="del\*" propagation="REQUIRED" rollback-for="java.lang.Exception" **/>**
63. **<tx:method** name="upd\*" propagation="REQUIRED" rollback-for="java.lang.Exception" **/>**
65. **<tx:method** name="save\*" propagation="REQUIRED" rollback-for="java.lang.Exception" **/>**
67. **<tx:method** name="query\*" read-only="true" **/>**
69. **<tx:method** name="find\*" read-only="true" **/>**
71. **<tx:method** name="get\*" read-only="true" **/>**
73. **<tx:method** name="view\*" read-only="true" **/>**
75. **<tx:method** name="search\*" read-only="true" **/>**
77. **<tx:method** name="check\*" read-only="true" **/>**
79. **<tx:method** name="is\*" read-only="true" **/>**
81. **<tx:method** name="\*" propagation="REQUIRED" rollback-for="java.lang.Exception" **/>**
82. **</tx:attributes>**
83. **</tx:advice>**
85. **<aop:config>**
87. **<aop:pointcut** id="serviceMethod" expression="execution(\* org.sky.ssh.service.impl.\*.\*(..))" **/>**
89. **<aop:advisor** advice-ref="txAdvice" pointcut-ref="serviceMethod" **/>**
90. **</aop:config>**
92. **</beans>**

注意两行:

****<property name="user" value="alpha\_test" />  
  
<property name="password" value="password\_1" />****

可以得知我们测试时用的是同一个数据库上的另一个实例，该实例是专门为我们的单元测试用的.

我们先来书写一个单元测试类吧

### **org.sky.ssh.ut.TestLoginDAO**

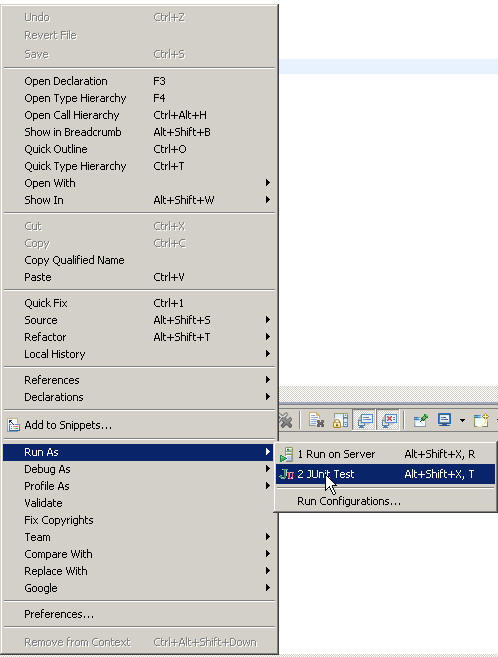
**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **package** org.sky.ssh.ut;
3. **import** **static** org.junit.Assert.assertEquals;
5. **import** javax.annotation.Resource;
7. **import** org.junit.Test;
8. **import** org.sky.ssh.dao.LoginDAO;
9. **import** org.springframework.test.annotation.Rollback;
11. **public** **class** TestLoginDAO **extends** BaseSpringContextCommon {
12. @Resource
13. **private** LoginDAO loginDAO;
15. @Test
16. @Rollback(**false**)
17. **public** **void** testLoginDAO() **throws** Exception {
18. String loginId = "alpha";
19. String loginPwd = "aaaaaa";
20. **long** answer = loginDAO.validLogin(loginId, loginPwd);
21. assertEquals(1, answer);
22. }
23. }

很简单吧，把原来的LongDAO注入进我们的单元测试类中，然后在test方法前加入一个@Test代码该方法为“单元测试”方法即可被junit可识别，然后我们调用一下LoginDAO中的.validLogin方法，测试一下返回值。

运行方法为：

在eclipse打开该类的情况下右键->run as Junit Test



然后选junit4来运行，运行后直接出错抛出：

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

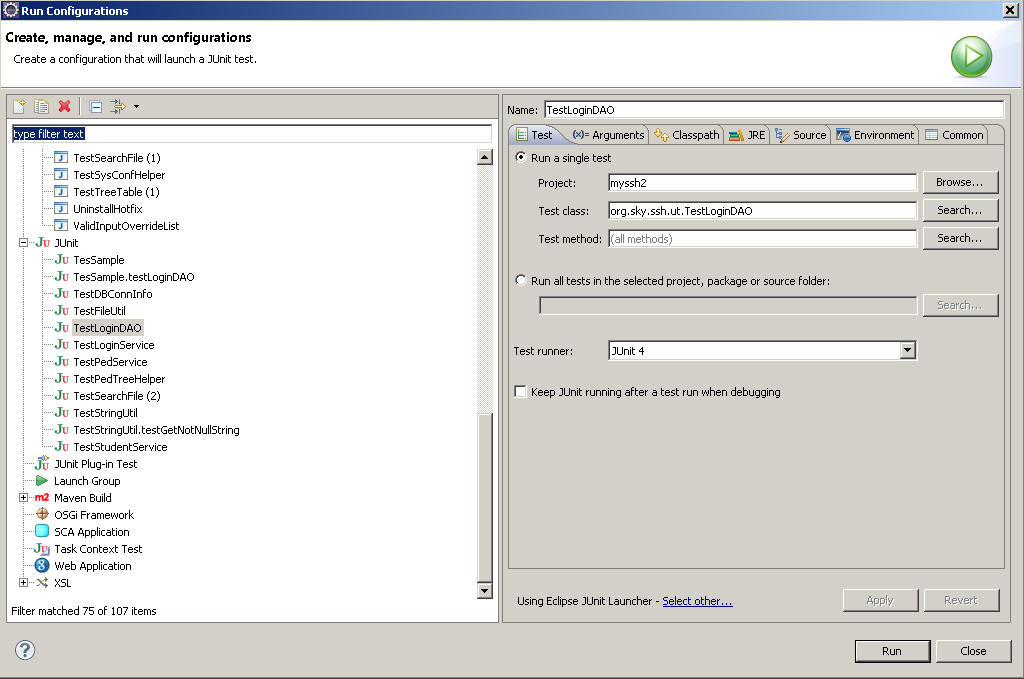
1. Class not found org.sky.ssh.ut.TestLoginDAO
2. java.lang.ClassNotFoundException: org.sky.ssh.ut.TestLoginDAO
3. at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
4. at java.security.AccessController.doPrivileged(Native Method)
5. at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
6. at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
7. at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
8. at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
9. at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.loadClass(RemoteTestRunner.java:693)
10. at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.loadClasses(RemoteTestRunner.java:429)
11. at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:452)
12. at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:683)
13. at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:390)
14. at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:197)

这样一个错误，为什么？

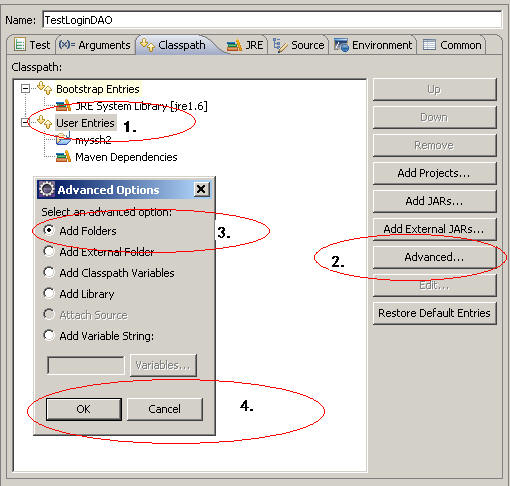
其原因在于我们的工程是在eclipse里使用的m2 eclipse这个插件生成的，因此在做单元测试时由于我们的unit test的类是放在test/main/java这个目录下，而这个目录是我们手工建的，因此eclipse不知道这个目录的对应的编译输出的class的目录了.

没关系，按照下面的方法：

右键->选择run as->run configuration，打开如下的设置

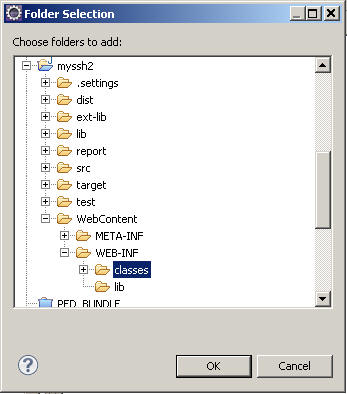


选择classpath这个选项栏



1. 单击user Entries
2. 单击Advanced按钮
3. 在弹出框中选择Add Folders
4. 点ok按钮

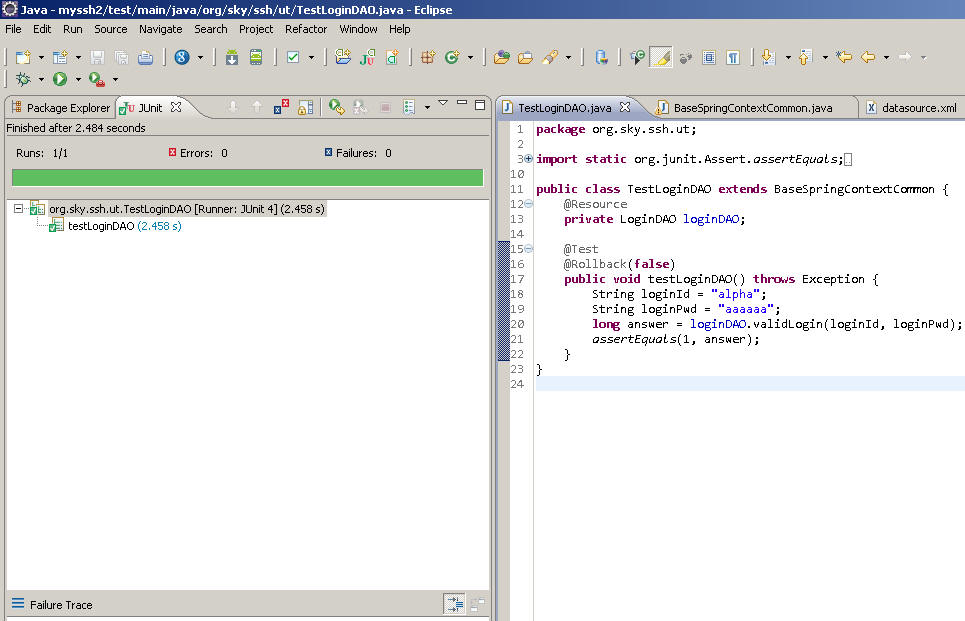
在下一个弹出框中选择我们的junit test的源码在被编译后输出的目录即myssh2工程的WebContent/WEB-INF/classes目录，对吧。



点OK按钮

点Apply按钮

点Run按钮，查看运行效果



运行成功，说明该unit test书写的是对的。

## **3.2 结合dbunit来做单元测试**

我们有了junit为什么还要引入一个dbunit呢？这不是多此一举吗？

试想一下下列场景：

我们开发时连的是开发用的数据库，一张表里有一堆的数据，有些数据不是自己的插的是其它的开发人员插的，那么我想要测试一个dao或者是service方法，获得一个List，然后判断这个List里的值是否为我想要的时候，有可能会碰到下属这样的情况：

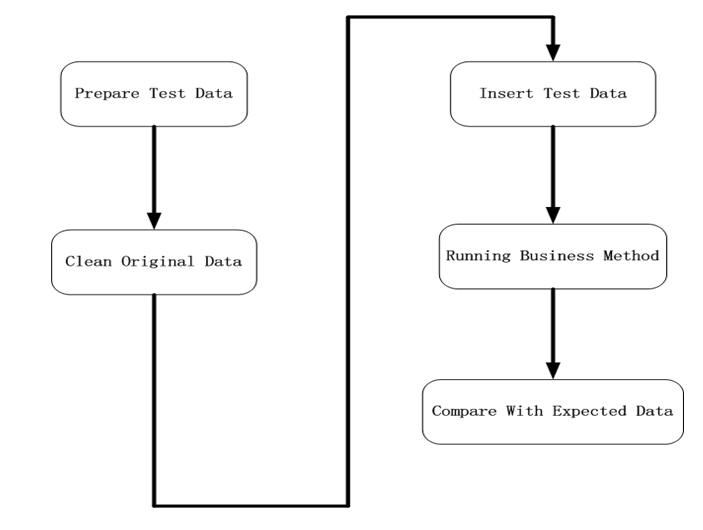
运行我的service或者dao方法得到一个list，该list含有6个值，但正好在运行时另一个开发人员因为测试需要往数据库里又插了一些值，导致我的测试方法失败，对不对，这种情况是有可能的。

怎么办呢？比较好的做法是我们需要准备一份自己的业务数据即prepare data，因为是我们自己准备的数据数据，因此它在经过这个方法运行后得到的值，这个得到的值是要经过一系列的业务逻辑的是吧？因此这个得到的值即：expected data是可以被精确预料的。

因此，我们拿着这个expected data与运行了我们的业务方法后得到的结果进行比对，如果比对结果一致，则一定是测试成功，否则失败，对吧？

这就是我们常说的，****测试用数据需要是一份干净的数据****。

那么为了保持我们的数据干净，我们在测试前清空我们的业务表，插入数据，运行测试地，比对结果，删除数据（也可以不删除，因为每次运行时都会清空相关的业务表），这也就是为什么我们事先要专门搞一个数据库或者是数据库实例，在运行单元测试时我们的数据库连接需要指向到这个单元测试专用的数据库的原因了，见下面的测试流程表：

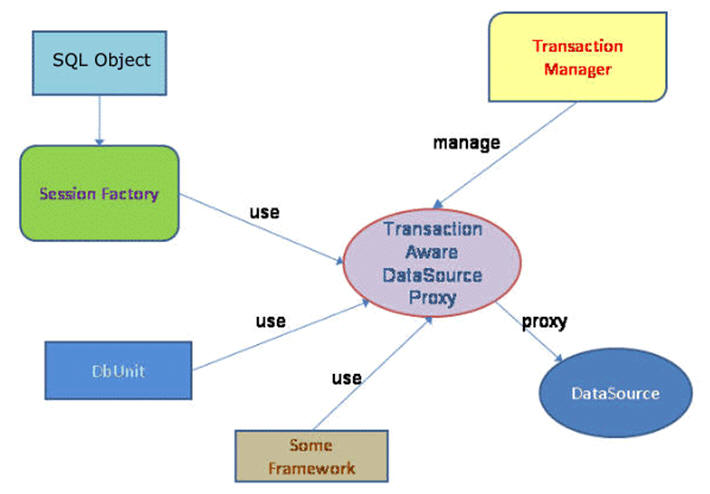


有了DbUnit，它就可以帮助我们封装：

* 准备测试用数据
* 清空相关业务表
* 插入测试数据
* 比对结果
* 清除先前插入的业务数据

这一系列底层的操作。

现在我们可以开始搭建我们的单元测试框架了，下面是这个单元测试框架的”逻辑表达图“（一个架构设计文档不仅需要有logic view还要有physical view。。。当然还有更多，以后会一点点分享出来）



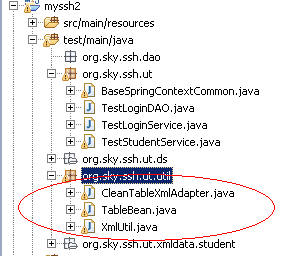
这边的Session Factory是结合的原有框架的Hibernate的Session Factory，我们也可以把它改成iBatis，Jdbc Template等等等。。。它可以稍作变动就可适用于一切SSX这样的架构。

该框架的优点如下：



## **3.3 构建spring+junit+dbunit的框架**

除去上述的一些类和配置我们还需要3个基类，它们分别位于test/main/java目录下（因为它们都属于unit test对吧）



### **org.sky.ssh.ut.util.CleanTableXmlAdapter**

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **package** org.sky.ssh.ut.util;
3. **import** org.dom4j.Element;
4. **import** org.dom4j.VisitorSupport;
5. **import** java.util.\*;
7. **public** **class** CleanTableXmlAdapter **extends** VisitorSupport {
9. **private** ArrayList tableList = **new** ArrayList();
11. **public** CleanTableXmlAdapter() {
12. }
14. **public** **void** visit(Element node) {
15. **try** {
17. **if** ((node.getName().toLowerCase()).equals("table")) {
18. TableBean tBean = **new** TableBean();
19. tBean.setTableName(node.getText());
20. tableList.add(tBean);
21. }
23. } **catch** (Exception e) {
24. }
25. }
27. **public** ArrayList getTablesList() {
28. **if** (tableList == **null** || tableList.size() < 1) {
29. **return** **null**;
30. } **else** {
31. **return** tableList;
32. }
33. }
34. }

### **org.sky.ssh.ut.util.TableBean**

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **package** org.sky.ssh.ut.util;
2. **import** java.io.\*;
3. **public** **class** TableBean **implements** Serializable{
5. **private** String tableName = "";
7. **public** String getTableName() {
8. **return** tableName;
9. }
11. **public** **void** setTableName(String tableName) {
12. **this**.tableName = tableName;
13. }
14. }

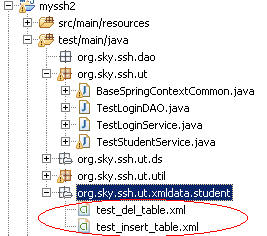
### **org.sky.ssh.ut.util.XmlUtil**

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **package** org.sky.ssh.ut.util;
3. **import** java.util.\*;
4. **import** java.io.\*;
5. **import** org.dom4j.Document;
6. **import** org.dom4j.Element;
7. **import** org.dom4j.VisitorSupport;
8. **import** org.dom4j.io.SAXReader;
9. **import** org.springframework.core.io.ClassPathResource;
11. **public** **class** XmlUtil {
13. **public** ArrayList getCleanTables(String xmlFile) {
14. ArrayList tablesList = **new** ArrayList();
15. **try** {
16. SAXReader reader = **new** SAXReader();
17. File file = **new** File(xmlFile);
18. Document doc = reader.read(file);
19. CleanTableXmlAdapter xmlAdapter = **new** CleanTableXmlAdapter();
20. doc.accept(xmlAdapter);
21. tablesList = xmlAdapter.getTablesList();
22. **return** tablesList;
23. } **catch** (Exception e) {
24. e.printStackTrace();
25. **return** **null**;
26. }
27. }
29. }

## **3.4使用框架**

我们准备两份测试用数据



### **test\_del\_table.xml文件**

**[html]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<Tables>**
3. **<table>**t\_student**</table>**
4. **</Tables>**

### **test\_insert\_table.xml文件**

**[html]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

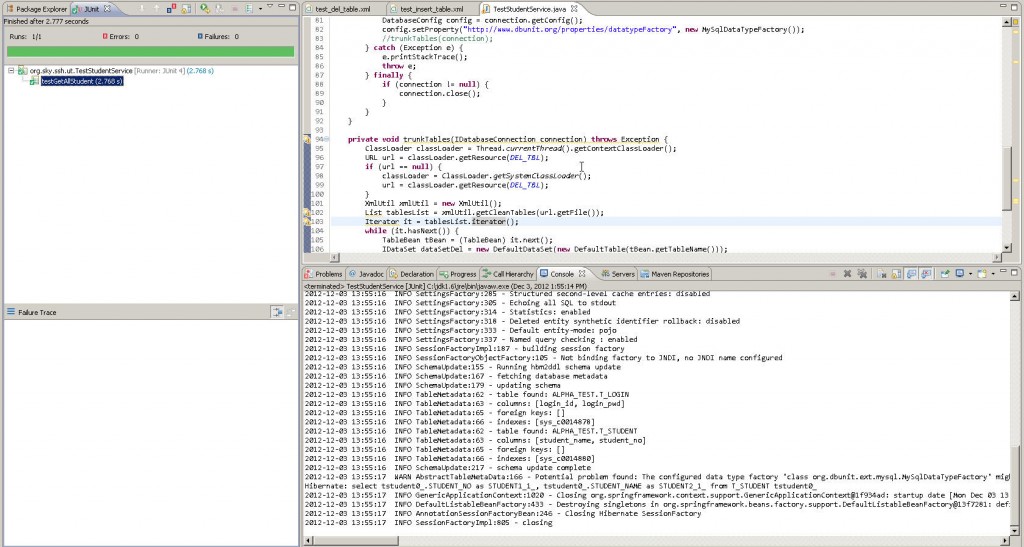
1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<dataset>**
3. **<t\_student** student\_no="101" student\_name="alice"**/>**
4. **<t\_student** student\_no="102" student\_name="jil"**/>**
5. **<t\_student** student\_no="103" student\_name="leon"**/>**
6. **<t\_student** student\_no="104" student\_name="chris"**/>**
7. **<t\_student** student\_no="105" student\_name="Ada Wong"**/>**
8. **</dataset>**

### **测试类org.sky.ssh.ut.TestStudentService**

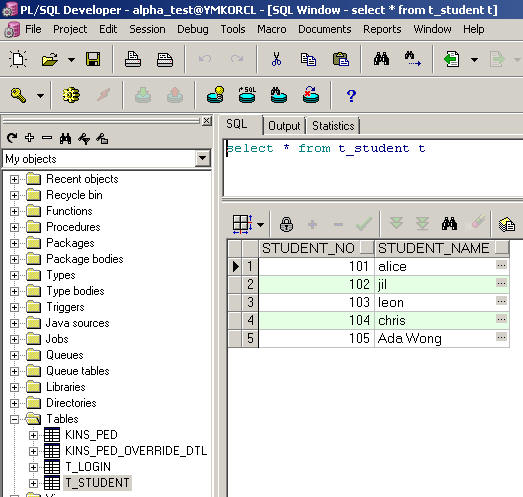
**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

1. **package** org.sky.ssh.ut;
3. **import** **static** org.junit.Assert.assertEquals;
5. **import** java.io.File;
6. **import** java.io.FileInputStream;
7. **import** java.net.URL;
8. **import** java.util.ArrayList;
9. **import** java.util.Iterator;
10. **import** java.util.List;
12. **import** javax.annotation.Resource;
13. **import** javax.sql.DataSource;
15. **import** org.dbunit.database.DatabaseConfig;
16. **import** org.dbunit.database.DatabaseConnection;
17. **import** org.dbunit.database.IDatabaseConnection;
18. **import** org.dbunit.dataset.DefaultDataSet;
19. **import** org.dbunit.dataset.DefaultTable;
20. **import** org.dbunit.dataset.IDataSet;
21. **import** org.dbunit.dataset.xml.FlatXmlDataSet;
22. **import** org.dbunit.dataset.xml.FlatXmlDataSetBuilder;
23. **import** org.dbunit.ext.mysql.MySqlDataTypeFactory;
24. **import** org.dbunit.operation.DatabaseOperation;
25. **import** org.junit.After;
26. **import** org.junit.Before;
27. **import** org.junit.Test;
28. **import** org.sky.ssh.service.StudentService;
29. **import** org.sky.ssh.ut.util.TableBean;
30. **import** org.sky.ssh.ut.util.XmlUtil;
31. **import** org.sky.ssh.vo.StudentVO;
32. **import** org.springframework.beans.factory.annotation.Autowired;
33. **import** org.springframework.core.io.ClassPathResource;
34. **import** org.springframework.jdbc.datasource.DataSourceUtils;
35. **import** org.springframework.test.annotation.Rollback;
37. **public** **class** TestStudentService **extends** BaseSpringContextCommon {
38. **private** **final** **static** String INSERT\_TBL = "org/sky/ssh/ut/xmldata/student/test\_insert\_table.xml";
39. **private** **final** **static** String DEL\_TBL = "org/sky/ssh/ut/xmldata/student/test\_del\_table.xml";
40. @Autowired
41. **private** DataSource dataSource;
43. @Resource
44. **private** StudentService stdService;
46. @SuppressWarnings("deprecation")
47. @Before
48. **public** **void** setUp() **throws** Exception {
49. IDatabaseConnection connection = **null**;
50. **try** {
51. connection = **new** DatabaseConnection(DataSourceUtils.getConnection(dataSource));
52. DatabaseConfig config = connection.getConfig();
53. config.setProperty("http://www.dbunit.org/properties/datatypeFactory", **new** MySqlDataTypeFactory());
55. //trunkTables(connection);
56. ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
57. URL url = classLoader.getResource(INSERT\_TBL);
58. **if** (url == **null**) {
59. classLoader = ClassLoader.getSystemClassLoader();
60. url = classLoader.getResource(INSERT\_TBL);
61. }
63. IDataSet dateSetInsert = **new** FlatXmlDataSetBuilder().build(**new** FileInputStream(url.getFile()));
64. DatabaseOperation.CLEAN\_INSERT.execute(connection, dateSetInsert);
65. } **catch** (Exception e) {
66. e.printStackTrace();
67. **throw** e;
68. } **finally** {
69. **if** (connection != **null**) {
70. connection.close();
71. }
72. }
73. }
75. @After
76. **public** **void** tearDown() **throws** Exception {
77. IDatabaseConnection connection = **null**;
78. **try** {
80. connection = **new** DatabaseConnection(DataSourceUtils.getConnection(dataSource));
81. DatabaseConfig config = connection.getConfig();
82. config.setProperty("http://www.dbunit.org/properties/datatypeFactory", **new** MySqlDataTypeFactory());
83. //trunkTables(connection);
84. } **catch** (Exception e) {
85. e.printStackTrace();
86. **throw** e;
87. } **finally** {
88. **if** (connection != **null**) {
89. connection.close();
90. }
91. }
92. }
94. **private** **void** trunkTables(IDatabaseConnection connection) **throws** Exception {
95. ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
96. URL url = classLoader.getResource(DEL\_TBL);
97. **if** (url == **null**) {
98. classLoader = ClassLoader.getSystemClassLoader();
99. url = classLoader.getResource(DEL\_TBL);
100. }
101. XmlUtil xmlUtil = **new** XmlUtil();
102. List tablesList = xmlUtil.getCleanTables(url.getFile());
103. Iterator it = tablesList.iterator();
104. **while** (it.hasNext()) {
105. TableBean tBean = (TableBean) it.next();
106. IDataSet dataSetDel = **new** DefaultDataSet(**new** DefaultTable(tBean.getTableName()));
107. DatabaseOperation.DELETE\_ALL.execute(connection, dataSetDel);
108. }
109. }
111. @Test
112. @Rollback(**false**)
113. **public** **void** testGetAllStudent() **throws** Exception {
114. List<StudentVO> stdList = **new** ArrayList<StudentVO>();
115. stdList = stdService.getAllStudent();
116. assertEquals(5, stdList.size());
117. }
118. }
119. 该测试方法每次都清空t\_student表
120. 往t\_student表里注入5条数据
121. 运行业务方法getAllStudent
122. 比较getAllStudent方法返回的list里的size是否为5
123. 清空注入的数据（也可不用去清空）

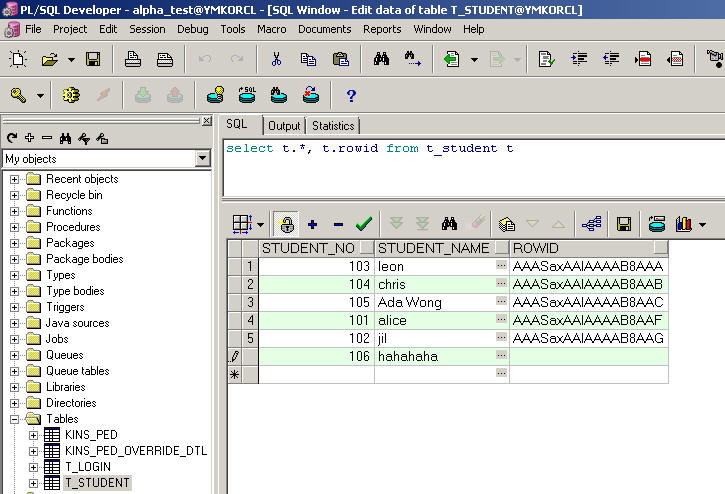
然后我们在eclipse里用junit来运行我们这个测试类吧。



我们现在用我们的单元测试用数据库帐号连入我们的数据库，查询t\_student表

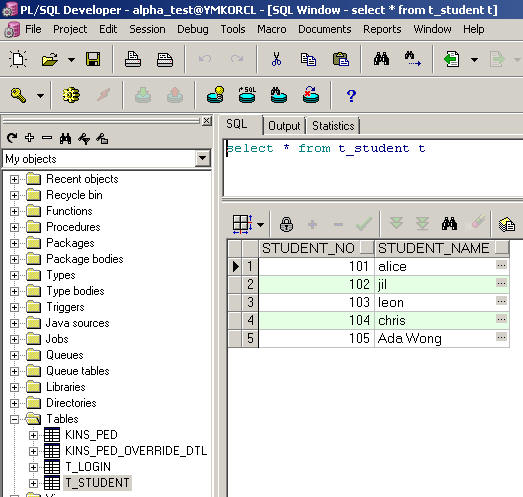


我们往该表中手动插入一条数据



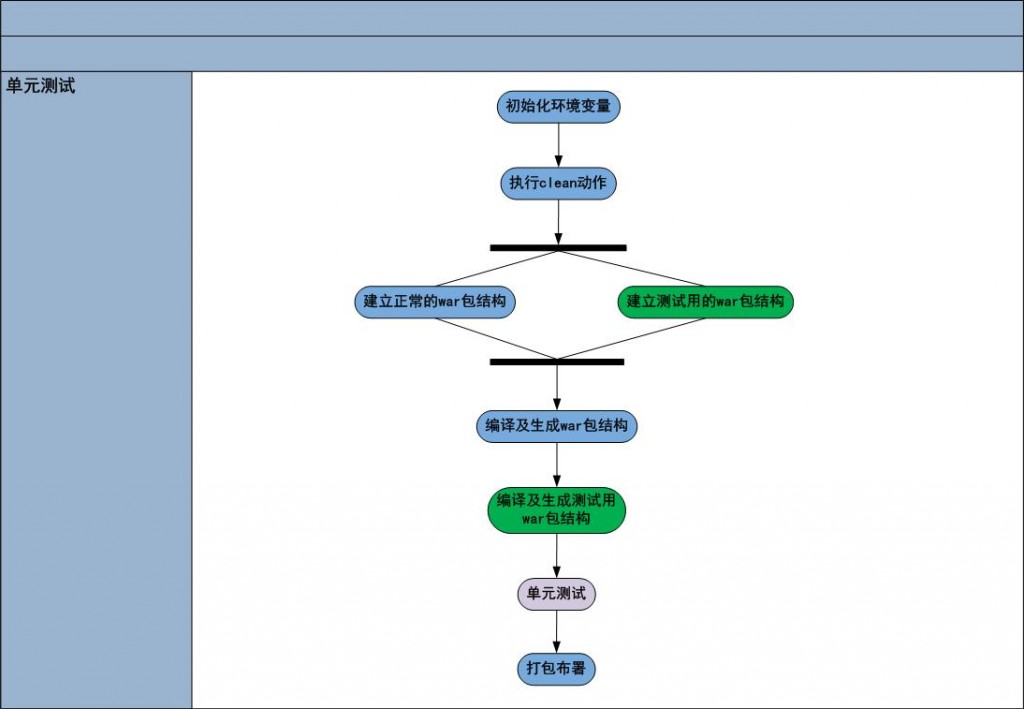
再重新运行一遍我们的单元测试

测试结果还是成功，再重新连入我们单元测试用数据库实例查询t\_student表，发觉还是5条记录，说明我们的框架达到了我们的目标。



# **四、将ant与我们的单元测试框架连接起来并生成单元测试报告**

先来看一下我们的nightly building，即每天次日的零晨将要生成的单元测试与打包布署的流程吧



****(需要ant1.8及以上版本运行)****

然后下面给出build.xml文件(需要ant1.8及以上版本运行)（结合了maven的依赖库机制）

## **build.properties文件**

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

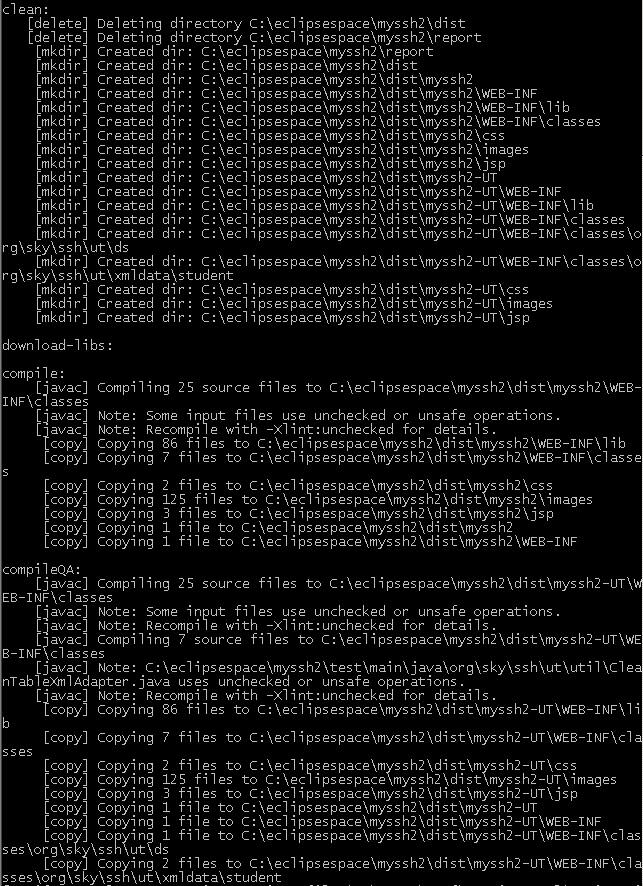
1. # ant
2. appName=myssh2
3. webAppName=myssh2
4. webAppQAName=myssh2-UT
5. local.dir=C:/eclipsespace/${appName}
6. src.dir=${local.dir}/src/main/java
7. test.src.dir=${local.dir}/test/main/java
8. dist.dir=${local.dir}/dist
9. report.dir=${local.dir}/report
10. webroot.dir=${local.dir}/src/main/webapp
11. lib.dir=${local.dir}/lib
12. ext-lib.dir=${local.dir}/ext-lib
13. classes.dir=${webroot.dir}/WEB-INF/classes
14. resources.dir=${local.dir}/src/main/resources

## **build.xml文件**

**[html]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8251056" \o "copy)

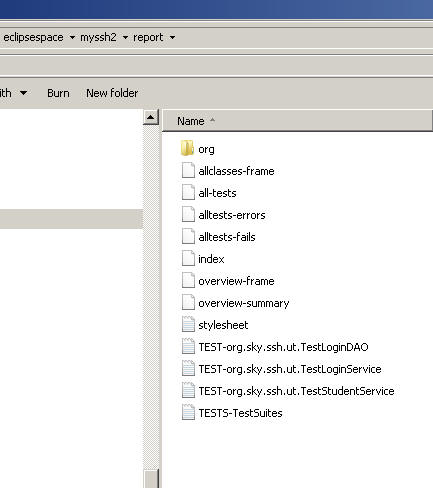
1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<project** name="myssh2" default="buildwar" xmlns:artifact="urn:maven-artifact-ant"**>**
4. **<property** file="build.properties" **/>**
5. **<property** name="classes.dir" value="${dist.dir}/${webAppName}/WEB-INF/classes" **/>**
6. **<path** id="maven-ant-tasks.classpath" path="C:/ant/lib/maven-ant-tasks-2.1.3.jar" **/>**
7. **<typedef** resource="org/apache/maven/artifact/ant/antlib.xml" uri="urn:maven-artifact-ant" classpathref="maven-ant-tasks.classpath" **/>**
8. **<artifact:pom** id="maven.project" file="pom.xml" **/>**
10. **<artifact:dependencies** filesetId="deps.fileset.compile" useScope="compile"**>**
11. <!--<pom file="pom.xml"/>-->
12. **<pom** refid="maven.project" **/>**
13. **</artifact:dependencies>**
15. **<path** id="compile.classpath"**>**
16. **<fileset** dir="${lib.dir}"**>**
17. **<include** name="\*.jar" **/>**
18. **</fileset>**
19. **</path>**
21. **<target** name="clean" description="Delete old build and dist directories"**>**
23. **<delete** dir="${dist.dir}" **/>**
24. **<delete** dir="${report.dir}" **/>**
25. **<mkdir** dir="${report.dir}" **/>**
26. **<mkdir** dir="${dist.dir}" **/>**
28. <!-- create war structure for production env-->
29. **<mkdir** dir="${dist.dir}/${webAppName}" **/>**
30. **<mkdir** dir="${dist.dir}/${webAppName}/WEB-INF" **/>**
31. **<mkdir** dir="${dist.dir}/${webAppName}/WEB-INF/lib" **/>**
32. **<mkdir** dir="${dist.dir}/${webAppName}/WEB-INF/classes" **/>**
33. **<mkdir** dir="${dist.dir}/${webAppName}/css" **/>**
34. **<mkdir** dir="${dist.dir}/${webAppName}/images" **/>**
35. **<mkdir** dir="${dist.dir}/${webAppName}/jsp" **/>**
37. <!-- create war structure for qa env -->
38. **<mkdir** dir="${dist.dir}/${webAppQAName}" **/>**
39. **<mkdir** dir="${dist.dir}/${webAppQAName}/WEB-INF" **/>**
40. **<mkdir** dir="${dist.dir}/${webAppQAName}/WEB-INF/lib" **/>**
41. **<mkdir** dir="${dist.dir}/${webAppQAName}/WEB-INF/classes" **/>**
42. **<mkdir** dir="${dist.dir}/${webAppQAName}/WEB-INF/classes/org/sky/ssh/ut/ds" **/>**
43. **<mkdir** dir="${dist.dir}/${webAppQAName}/WEB-INF/classes/org/sky/ssh/ut/xmldata/student" **/>**
44. **<mkdir** dir="${dist.dir}/${webAppQAName}/css" **/>**
45. **<mkdir** dir="${dist.dir}/${webAppQAName}/images" **/>**
46. **<mkdir** dir="${dist.dir}/${webAppQAName}/jsp" **/>**
47. **</target>**
49. **<target** name="download-libs" depends="clean"**>**
50. **<copy** todir="${lib.dir}"**>**
51. **<fileset** refid="deps.fileset.compile" **/>**
52. **<mapper** type="flatten" **/>**
53. **</copy>**
54. **</target>**
56. **<target** name="compile" description="Compile java sources" depends="download-libs"**>**
58. <!-- compile main class -->
60. **<javac** debug="true" destdir="${dist.dir}/${webAppName}/WEB-INF/classes" includeAntRuntime="false" srcdir="${src.dir}"**>**
61. **<classpath** refid="compile.classpath" **/>**
62. **</javac>**
63. **<copy** todir="${dist.dir}/${webAppName}/WEB-INF/lib"**>**
64. **<fileset** dir="${lib.dir}"**>**
65. **<include** name="\*.jar" **/>**
66. **</fileset>**
67. **</copy>**
68. **<copy** todir="${dist.dir}/${webAppName}/WEB-INF/classes"**>**
69. **<fileset** dir="${resources.dir}"**>**
70. **<include** name="\*\*/\*.\*" **/>**
71. **</fileset>**
72. **</copy>**
73. **<copy** todir="${dist.dir}/${webAppName}/css"**>**
74. **<fileset** dir="${webroot.dir}/css"**>**
75. **<include** name="\*\*/\*.\*" **/>**
76. **</fileset>**
77. **</copy>**
78. **<copy** todir="${dist.dir}/${webAppName}/images"**>**
79. **<fileset** dir="${webroot.dir}/images"**>**
80. **<include** name="\*\*/\*.\*" **/>**
81. **</fileset>**
82. **</copy>**
83. **<copy** todir="${dist.dir}/${webAppName}/jsp"**>**
84. **<fileset** dir="${webroot.dir}/jsp"**>**
85. **<include** name="\*\*/\*.\*" **/>**
86. **</fileset>**
87. **</copy>**
88. **<copy** todir="${dist.dir}/${webAppName}"**>**
89. **<fileset** dir="${webroot.dir}"**>**
90. **<include** name="\*.\*" **/>**
91. **</fileset>**
92. **</copy>**
93. **<copy** todir="${dist.dir}/${webAppName}/WEB-INF"**>**
94. **<fileset** dir="${webroot.dir}/WEB-INF"**>**
95. **<include** name="\*.\*" **/>**
96. **</fileset>**
97. **</copy>**
98. **</target>**
99. **<target** name="compileQA" description="Compile java sources" depends="compile"**>**
101. <!-- compile main class -->
103. **<javac** debug="true" destdir="${dist.dir}/${webAppQAName}/WEB-INF/classes" includeAntRuntime="false" srcdir="${src.dir}"**>**
104. **<classpath** refid="compile.classpath" **/>**
105. **</javac>**
106. **<javac** debug="true" destdir="${dist.dir}/${webAppQAName}/WEB-INF/classes" includeAntRuntime="false" srcdir="${test.src.dir}"**>**
107. **<classpath** refid="compile.classpath" **/>**
108. **</javac>**
109. **<copy** todir="${dist.dir}/${webAppQAName}/WEB-INF/lib"**>**
110. **<fileset** dir="${lib.dir}"**>**
111. **<include** name="\*.jar" **/>**
112. **</fileset>**
113. **</copy>**
114. **<copy** todir="${dist.dir}/${webAppQAName}/WEB-INF/classes"**>**
115. **<fileset** dir="${resources.dir}"**>**
116. **<include** name="\*\*/\*.\*" **/>**
117. **</fileset>**
118. **</copy>**
119. **<copy** todir="${dist.dir}/${webAppQAName}/css"**>**
120. **<fileset** dir="${webroot.dir}/css"**>**
121. **<include** name="\*\*/\*.\*" **/>**
122. **</fileset>**
123. **</copy>**
124. **<copy** todir="${dist.dir}/${webAppQAName}/images"**>**
125. **<fileset** dir="${webroot.dir}/images"**>**
126. **<include** name="\*\*/\*.\*" **/>**
127. **</fileset>**
128. **</copy>**
129. **<copy** todir="${dist.dir}/${webAppQAName}/jsp"**>**
130. **<fileset** dir="${webroot.dir}/jsp"**>**
131. **<include** name="\*\*/\*.\*" **/>**
132. **</fileset>**
133. **</copy>**
134. **<copy** todir="${dist.dir}/${webAppQAName}"**>**
135. **<fileset** dir="${webroot.dir}"**>**
136. **<include** name="\*.\*" **/>**
137. **</fileset>**
138. **</copy>**
139. **<copy** todir="${dist.dir}/${webAppQAName}/WEB-INF"**>**
140. **<fileset** dir="${webroot.dir}/WEB-INF"**>**
141. **<include** name="\*.\*" **/>**
142. **</fileset>**
143. **</copy>**
144. **<copy** todir="${dist.dir}/${webAppQAName}/WEB-INF/classes/org/sky/ssh/ut/ds"**>**
145. **<fileset** dir="${test.src.dir}/org/sky/ssh/ut/ds"**>**
146. **<include** name="\*.xml" **/>**
147. **</fileset>**
148. **</copy>**
149. **<copy** todir="${dist.dir}/${webAppQAName}/WEB-INF/classes/org/sky/ssh/ut/xmldata/student"**>**
150. **<fileset** dir="${test.src.dir}/org/sky/ssh/ut/xmldata/student"**>**
151. **<include** name="\*.xml" **/>**
152. **</fileset>**
153. **</copy>**
154. **<antcall** target="junitreport"**>**
155. **</antcall>**
156. **</target>**
157. **<target** name="buildwar" depends="compileQA"**>**
158. **<war** warfile="${dist.dir}/${webAppName}.war"**>**
159. **<fileset** dir="${dist.dir}/${webAppName}" **/>**
160. **</war>**
161. **</target>**
162. **<target** name="junitreport"**>**
163. **<junit** printsummary="on" haltonfailure="false" failureproperty="tests.failed" showoutput="true"**>**
164. **<classpath>**
165. **<pathelement** path="${dist.dir}/${webAppQAName}/WEB-INF/classes" **/>**
166. **<fileset** dir="${lib.dir}"**>**
167. **<include** name="\*.jar" **/>**
168. **</fileset>**
169. **<fileset** dir="${ext-lib.dir}"**>**
170. **<include** name="\*.jar" **/>**
171. **</fileset>**
172. **</classpath>**
173. **<formatter** type="xml" **/>**
174. **<batchtest** todir="${report.dir}"**>**
175. **<fileset** dir="${dist.dir}/${webAppQAName}/WEB-INF/classes"**>**
176. **<include** name="org/sky/ssh/ut/Test\*.\*" **/>**
177. **</fileset>**
178. **</batchtest>**
179. **</junit>**
180. **<junitreport** todir="${report.dir}"**>**
181. **<fileset** dir="${report.dir}"**>**
182. **<include** name="TEST-\*.xml" **/>**
183. **</fileset>**
184. **<report** format="frames" todir="report" **/>**
185. **</junitreport>**
186. **<fail** if="tests.failed"**>**
187. ---------------------------------------------------------
188. One or more tests failed, check the report for detail...
189. ---------------------------------------------------------
190. **</fail>**
191. **</target>**
192. **</project>**

对照着上面的build的流程图，很容易看懂

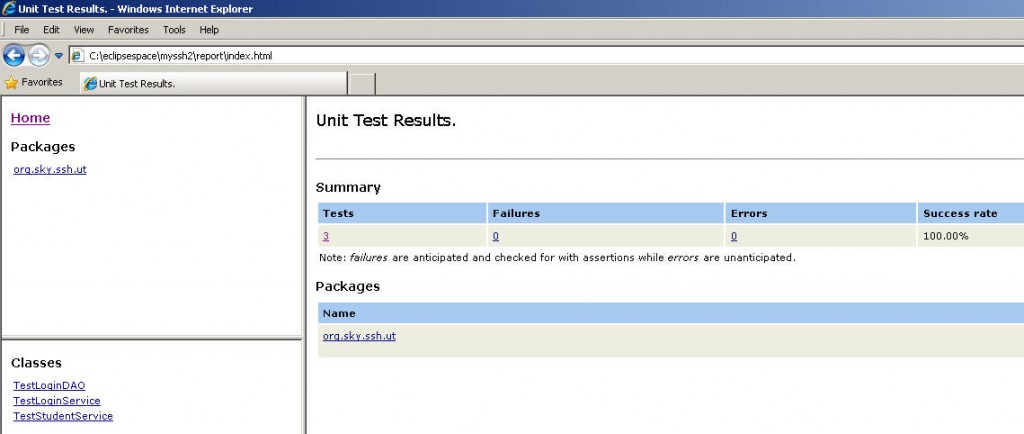
打开一个command窗口，进入到我们的工程的根目录下，设置好ANT\_HOME并将%ANT\_HOME%\bin目录加入到path中去，然后在工程的根据目录下运行ant，就能看到打包和运行unit test的效果了。  




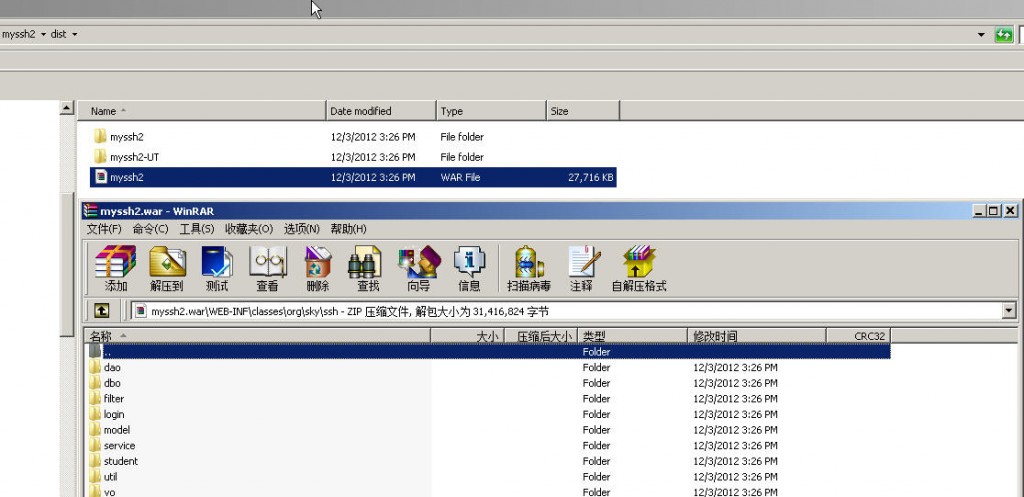
build完后可以在工程的根目录下找到一个report目录，打开后里面有一堆的html文件



双击index.htm这个文件查看单元测试报告



我们在windows的资源管理器中打开我们的工程，在根目录下有一个dist目录，打开这个目录，我们会开到两个目录与一个.war文件，它们分别是：



其中myssh2-UT是专门用来run unit test的，而myssh2是可以用于发布到production environment的，我们打开myssh2.war这个包，我们可以看到，由于这个是正确布署的war，因此里面是不能够含有unit test的相关类与方法的，完全按照上述的打包流程图来做的。

结束今天的教程！！！