目录

目录.....1
一、 环境准备.....2
1.1 TestNG+cobertura 在 Eclipse 中的集成.....2
1.2 TestNG+cobertura 在 Eclipse+maven 中的集成.....2
二、 开发工具环境集成.....3
2.1 Eclipse 开发环境.....3
2.2 Eclipse 开发环境+maven 项目管理.....3
三、 项目结构搭建与测试代码编写.....4
3.1 建立项目目录与文件结构.....4
3.2 测试代码编写.....4
3.3 引用测试断言.....5
3.4 常见测试功能说明.....6
四、 测试执行与输出.....7
4.1 在 Eclipse 中开发项目.....7
4.2 使用 maven 在 Eclipse 中构建项目.....9
五、 测试实例演示.....10
5.1 基于 Eclipse 的测试实例演示.....10
5.2 基于 Eclipse+maven 的测试实例演示.....15
六、 单元测试问题与解决.....18
6.1 项目测试管理.....18
6.2 测试代码编写.....20

一、环境准备

针对我公司各事业部 JAVA 开发环境与项目管理工具使用的情况，采用以下两种方式进行单元测试开发：

1.1 TestNG+cobertura 在 Eclipse 中的集成

在压缩包中提供开发所需开发包:
压缩包的 readExcel.jar 实现从 excel 中读取简单类型测试用例的功能。
Eclipse 单独开发：工具在" java 单元测试工具包--EcLipse 单独开发"文件夹下，包括：lib.rar 与 cobertura.rar。进行测试时需将这两个文件解压到项目目录下的 testsrc 目录下。
其中，lib.rar 为 jar 包文件夹，包括 testng-6.8.1.jar 与 result_eclipse.jar。
1、testng-6.8.1.jar 为 TestNG 断言定制包，在 testng-6.8.jar 的断言类中加入 map、collection 数据类型的断言，并改进了原有对数组断言的缺陷。
2、result_eclipse.jar 为 Eclipse 单独开发项目情况下的报告集成工具.
cobertura.rar 为覆盖度统计工具包。

- 将 TestNG-6.8.1jar 测试开发包加载 Eclipse 环境中，并下载 TestNG 的 Eclipse 插件。
- 解压 cobertura-1.9.4.1.rar 到项目的根目录下，下载 cobertura 在 Eclipse 中的插件 eCobertura.
- 将 result_eclipse.jar 放入项目的 lib 文件夹中，若项目中无 lib 文件夹，需创建 lib 文件夹。

1.2 TestNG+cobertura 在 Eclipse+maven 中的集成

- 在 maven 中集成了 TestNG 与 cobertura 的插件，仅需在 pom.xml 文件对实现对 TestNG 与 cobertura 进行依赖，maven 会在编译过程中自行下载，pom.xml 的具体见第五部分 Eclipse+Maven 的实例演示。
- 将" java 单元测试工具包--EcLipse+maven 开发"文件夹下的 lib.rar 与 bat 文件复制到项目目录下的 src/test 目录下，并将 lib.rar 解压到当前目录下，lib.rar 为 jar 包文件夹，包括 testng-6.8.1.jar 与 result_maven.jar。
  1、testng-6.8.1.jar 为 TestNG 断言定制包，在 testng-6.8.jar 的断言类中加入 map、collection 数据类型的断言，并改进了原有对数组断言的缺陷。
  2、result_eclipse.jar 为 Eclipse 单独开发项目情况下的报告集成工具.

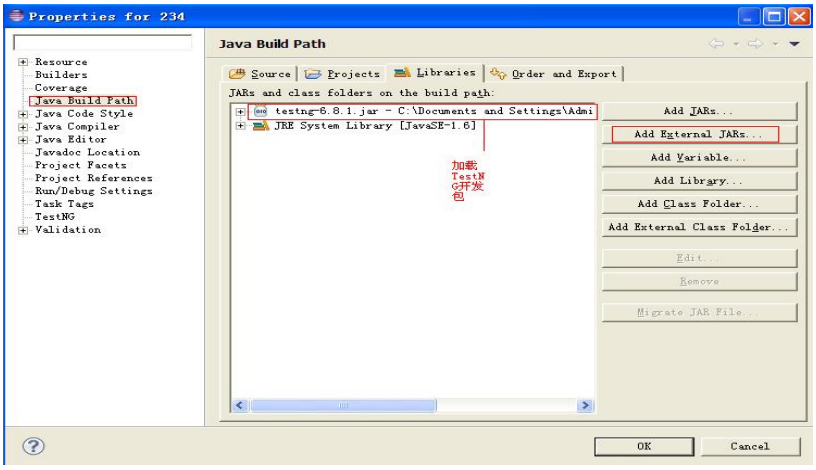
二、开发工具环境集成

2.1 Eclipse 开发环境

➢ 加载 TestNG 框架包文件

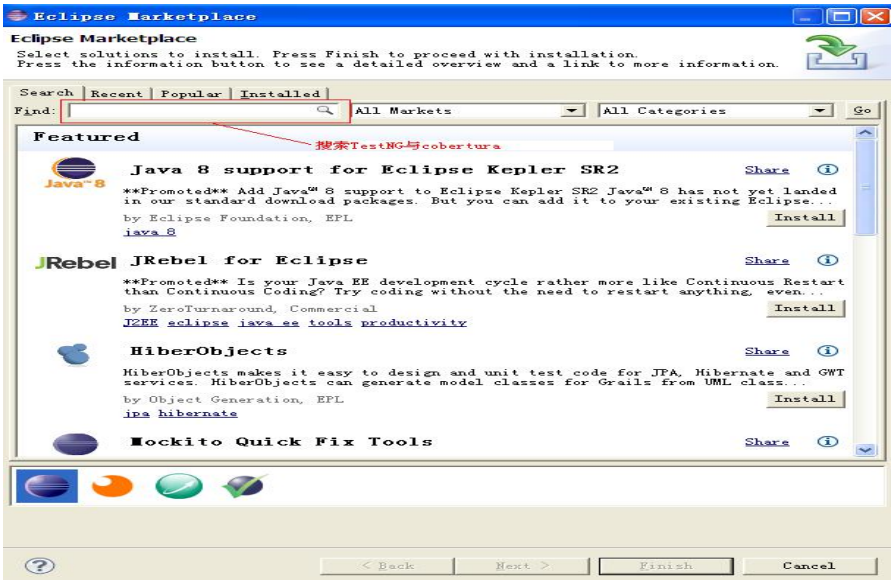
在对原有的开发包进行改进后定制了 TestNG-6.8.1.jar。将改 jar 包加载到开发环境中，具体步骤：

在对应的项目点击右键，选择属性，弹出属性设置对话框：



➢ TestNG 与 cobertura 的 Eclipse 插件下载安装

在 help->Eclipse maketplace 中进行插件在线下载安装，分别搜索 TestNG 与 cobertura,进行下载



2.2 Eclipse 开发环境+maven 项目管理

➢ Maven 集成 TestNG 与 cobertura 工具

使用 maven 对开发项目进行构建，在 pom.xml 文件中加入对 testNG 与 cobertura

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.8.1</version>
   <scope>test</scope>
</dependency>
```

对 Testng 的依赖

```
 <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>cobertura-maven-plugin</artifactId>
</plugin>                                   //cobertura 依赖
```

三、项目结构搭建与测试代码编写

基于以上环境的集成与配置，已经可以进行对待测代码进行测试构建与代码编写。

3.1 建立项目目录与文件结构

- 为了实现测试代码与被测代码相隔离，为测试代码单独建立文件夹（用 maven 构建自动创建测试文件夹） src/main/java src/test/java ，在 scr/test/java 进行测试代码编写，若没有用 maven 构建项目，需在项目中创建 src 与 testsrc 目录分别存放待测程序与测试程序 src testsrc 。
- 在项目根目录的 testsrc(Eclipse 单独开发)或 src/test(Eclipse+maven 构建)目录下创建 Testng.xml 文件

  在 testng.xml 文件对可以对测试属性进行配置，可以选择需要测试包、类、方法等，也可进行参数化测试。

```
<?xml version="1.0" encoding="UTF-8"?>
    <suite name="Suite1"   verbose="1" >
        <test name="Regression1" >
        <packages>                          //加入需要执行的测试包
            <package name="hyhTest" />
            <package name="otherTest" />
        </packages>
    </test>
</suite>
```

3.2 测试代码编写

- Eclipse+maven 构建：在 src/test/java 目录下建立测试类包，注意用 maven 构建项目时，测试包名需要是*Test 形式。

  Eclipse 开发：在 testsrc 目录下建立测试类包，为了对测试管理方便，测试包名

需要是*Test 形式。

- 用@Test 标注测试方法，可以根据需要加入相应的测试属性。如下

```
@Test(groups = { "param"}，……，……)
public void testCalculateAppriasal()
{
}
```

还有以下几种标注形式：

@DataProvider、@Parameters ----------参数化测试

@ExpectedExceptions----------进行异常检测

@Factory ----------标记工厂方法

- @Test 常用属性

alwaysRun ----------标志依赖测试失败后，测试方法是否执行

dependsOnGroups ----------这个方法依赖的组列表

dependsOnMethods -------这个方法依赖的方法列表

enabled --------这个类的方法是否激活

groups -------这个类或方法所属的分组列表

3.3 引用测试断言

- AssertEquals(Object , Object ,message)/ AssertEquals(Object , Object)
  其中，Object 可以为基础数据类型：int,double，char 等，也可为 vector，list，map,set 等复杂数据类型。Message 为报告中打印的相关提示信息。
- AssertEquals(Object[] , Object[] ,message)/ AssertEquals(Object , Object)
- assertFalse(boolean condition , String message) / assertFalse(boolean condition) 判断状态错误
- AssertNotEquals(Object , Object ,message)/ AssertEquals(Object , Object)
  其中，Object 可以为基础数据类型：int,double，char 等，也可为 vector，list，map,set 等复杂数据类型。Message 为报告中打印的相关提示信息。
- assertNotNull(Object)判断参数是否为 NULL
- assertNotSame(Object, Object,String) /assertNotSame(Object, Object,String) 若两个 Object 相等，抛出异常。
- assertSame(Object, Object,String) /assertNotSame(Object, Object,String) 若两个 Object 不相等，抛出异常。
- assertTrue(Boolean) 状态为假时，抛出异常。
- assertEqualsNoOrder((Object[] , Object[])/ assertEqualsNoOrder((Object[] , Object[] ,message) 若两个数组中有元素不同，抛出异常。不用对应元素进行比较

  单元测试开发中对 Collection、map 等数据类型断言进行了扩充与改进。

3.4 常见测试功能说明

- 参数化测试：@ DataProvider 与@ Parameters
  - ✓ @ Parameters 结合 TestNG.xml 文件可对带参数的测试方法进行参数化测试，仅对参数为基本数据类型有效（如 int,String 等），具体编写如下：

    TestNG.xml 文件设置参数：

```
<test name="Regression1" >
  <parameter name="data1" value="7000"></parameter>
  <parameter name="data2" value="500"></parameter>
```

参数名　　参数值

    在需使用参数的测试方法前加入如下标注：

    @Parameters({"data1","data2"})　　///有多少参数就要加入多少个参数名
  - ✓ @ DataProvider 可进行高级参数化测试，及对复杂数据类型进行测试（如 collection、map 等类集或自定义类），具体实现举例如下

说明: @ DataProvider 标注的参数提供方法返回的数据可以从 Excel 表格中读取简单数据类型，这样便于数据的管理。压缩包的 readExcel.jar 可实现 Excel 数据的读取(仅为简单的数据类型读取：int、String),具体实现如下

从 Excel 中读取简单数据类型的测试参数

```
@DataProvider()
public static Object[][] employ() throws IOException
{
    File file = new File("./testsrc/params_excel/11.xlsx");
    int[][] arrayStr =ReadExcel.getRunDataInt(file);
    return ReadExcel.intToObject(arrayStr);
}
}
```

- 异常检测

  两种编写形式均可：

方式一：

```
@Test(expectedExceptions = java.lang.ArithmeticException.class) ///完成异常处理测试
```

方式二：

```
@ExpectedExceptions(NumberFormatException.class) //在进行异常测试方法前加入此标注
@Test
```

- 工厂测试：@Factory

  动态创建测试案例。被@Factory 标注的方法返回测试用例数组（Object[]）,举例如下

```
public class WebTestFactory {
  @Factory                                  //标注方法为工厂测试
  public Object[] createInstances() {
   Object[] result = new Object[10];
   for (int i = 0; i < 10; i++) {
      result[i] = new WebTest(i * 10);
    return result;
```

```
  }
}
```

测试类如下

```
public class WebTest {
  private int m_numberOfTimes;
  public WebTest(int numberOfTimes) {
    m_numberOfTimes = numberOfTimes;
  }
```

注意：测试类不需要@Test 标注。

四、测试执行与输出

4.1 在 Eclipse 中开发项目

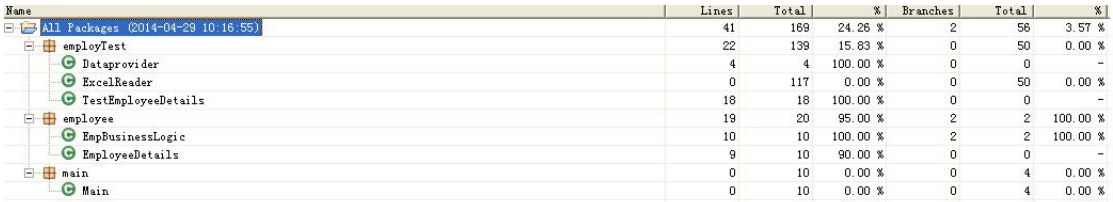
有两种方式执行测试程序：

- 利用 TestNG 与 cobertura 的 Eclipse 中的插件进行测试

选择待测试的项目，点击红框中按钮 生成测试报告与覆盖度报告，可在 Eclipse 中方便查阅：

测试报告

覆盖度报告



Name	Lines	Total	%	Branches	Total	%
All Packages (2014-04-29 10:16:55)	41	169	24.26 %	2	56	3.57 %
employTest	22	139	15.83 %	0	50	0.00 %
Dataprovider	4	4	100.00 %	0	0	-
ExcelReader	0	117	0.00 %	0	50	0.00 %
TestEmployeeDetails	18	18	100.00 %	0	0	-
employee	19	20	95.00 %	2	2	100.00 %
EmpBusinessLogic	10	10	100.00 %	2	2	100.00 %
EmployeeDetails	9	10	90.00 %	0	0	-
main	0	10	0.00 %	0	4	0.00 %
Main	0	10	0.00 %	0	4	0.00 %

在待测代码中以不同颜色区分覆盖到的行

```
public void setMonthlySalary(double monthlySalary) {
    this.monthlySalary = monthlySalary;
}
/**
* @return the age
*/
public int getAge() {
    return age;
}
/**
* @param age the age to set
*/
public void setAge(int age) {
this.age = age;
}
```

绿色覆盖到

红色没有覆盖

➢ 利用命令行执行测试

将提供压缩包中" java 单元测试工具包--EcLipse 单独开发"文件夹下 cobertura.rar 与 lib.rar 解压到 testsrc 目录下中，并编写执行命令 run.bat：

```
cd ..
md uninstrumented
md instrumented
```

进入项目根目录，创建临时文件夹

```
javac -cp testsrc\lib\testng-6.8.1.jar -d uninstrumented src\aaa\*.java
javac -cp testsrc\lib\testng-6.8.1.jar; -d uninstrumented testsrc\aaaTest\*.java
```

编译被测代码与测试代码，放于 uninstrumented 文件夹中

```
call testsrc\cobertura\cobertura-instrument.bat --destination instrumented
    uninstrumented\aaa\*.class
```

在被测代码中加入字节码，便于覆盖度统计

```
java -cp testsrc\lib\testng-6.8.1.jar; testsrc\cobertura\cobertura.jar;
instrumented;uninstrumented; -Dnet.sourceforge.cobertura.datafile=cobertura.ser
org.testng.TestNG testsrc/testng.xml
```

根据 testng.xml 文件执行测试程序，生成测试结果

```
call testsrc\cobertura\cobertura-report --format html --datafile cobertura.ser
--destination reports src
```

生成覆盖度统计报告

```
del cobertura.ser
java -jar testsrc\lib\result_eclipse.jar
rd /s/q uninstrumented
rd /s/q instrumented
rd /s/q reports
rd /s/q test-output
.\testsrc\Result_Report\Test_Result.html
```

报告集成，删除临时文件夹及文件

其中 result_maven.jar 为编写的测试报告与覆盖度报告集成工具。

集成测试报告输出形式

Coverage Report

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	1	66%	4/6	0%	0/2	1
aaa	1	66%	4/6	0%	0/2	1

Report generated by Cobertura 1.9.4.1 on 14-4-25 下午2:51.

Test Report

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Command line suite						
Command line test	1	0	0	31		

Class	Method	Start	Time (ms)
Command line suite			
Command line test — passed			
aaaTest.AaaTest	ffTest	1398408779015	16

Command line test

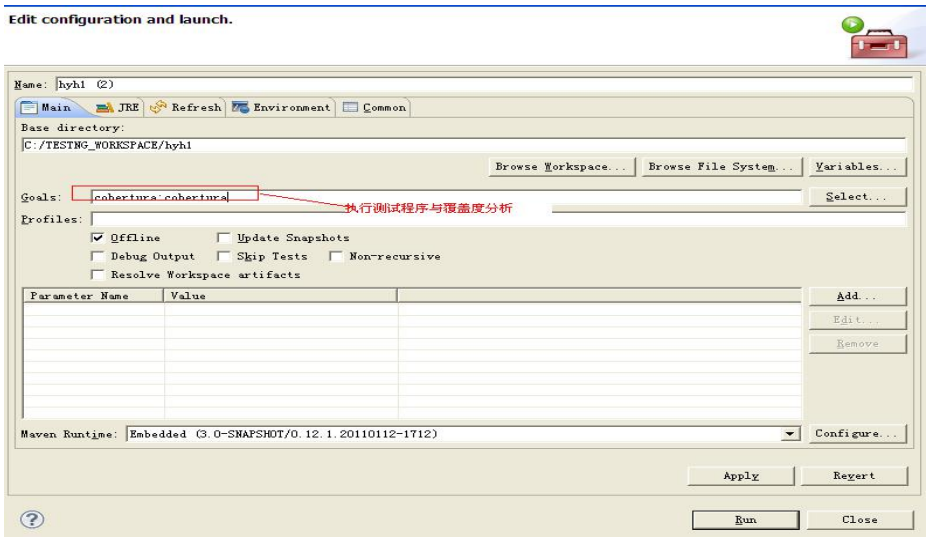
aaaTest.AaaTest#ffTest

4.2 使用 maven 在 Eclipse 中构建项目

有两种方式执行测试程序：

- 利用 maven 的 Eclipse 中的插件进行测试

在项目目录下，选择 run as ->maven build, 弹出如下对话框，输入 cobertura:cobertura



运行后在相应的文件夹中生成测试报告

\target\site\cobertura　包括覆盖度分析报告

target\surefire-reports　测试代码报告

- 利用 maven 的命令行执行测试

注意：第一次执行测试时，需将工具包“java 单元测试工具包--EcLipse+maven 开发”文件夹下的”加载断言库—执行一次即可.bat”复制到项目目录下的 scr/test 目录下，并执行一次，完成自定制库 testng-6.8.1.jar 在 maven 中的安装。

在项目文件夹下的 src\test 文件夹下解压 lib.rar(testng-6.8.1.jar 与 result_maven.jar)压缩文件，执行以下的 run.bat 文件：

cd ..

cd ..

进行项目根目录，删除原有的测试结果

```
rd /s /q .\src\test\Result_Report
call mvn cobertura:clean
call mvn cobertura:cobertura
```

Maven 编译、执行、生成报告命令，无需更改

```
java -jar .\src\test\lib\result_maven.jar
. \src\test \Result_Report\Test_Result.html
```

执行报告集成程序，将 result.jar 开发包放入 lib 文件夹下，无需更改

运行后在 src\test\Result_Report 文件夹下生产最终集成的测试报告 Test_Result.html 如下：

Coverage Report

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	5	70%	29/41	25%	2/8	1.235
calculator	1	78%	11/14	N/A	N/A	1
employee	2	94%	18/19	100%	2/2	1.125
org.hyh.number	2	0%	0/8	0%	0/6	2.5

Report generated by Cobertura 2.0.3 on 14-4-21 上午10:44.

Test Report

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite1						
Regression1	11	0	2	46		

Class	Method	Start	Time (ms)
Suite1			
Regression1 — failed			
employeeTest.TestEmployeeDetails	testCalculateAppriasal	1398048285171	0
org.hyh.numberTest.IntArrayTest	MapTest	1398048285140	0
Regression1 — passed			
employeeTest.TestEmployeeDetails	testCalculateAppriasal	1398048285171	0
	testCalculateAppriasal		

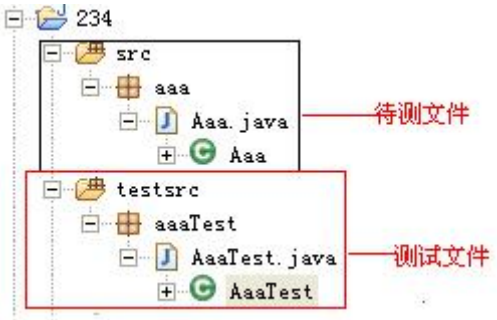
五、测试实例演示

通过两个简单实例简单说明基于 TestNG 的 java 单元测试流程

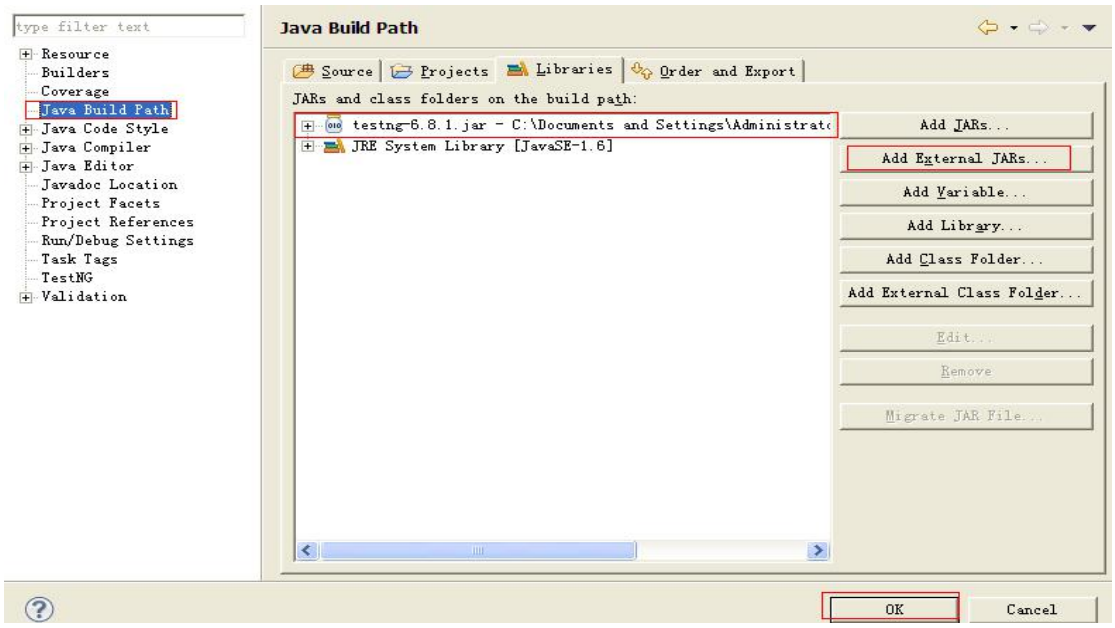
5.1 基于 Eclipse 的测试实例演示

- 在项目中创建于 src 并列的测试文件夹 testsrc src testsrc ，然后根据需要创建测试包 aaaTest ，根据被测代码创建测试类 AaaTest.java

项目的基本目录如下：



➢ 在项目中加入 testng-6.8.1.jar



➢ 编写测试代码

```
package aaaTest;
import aaa.Aaa;                                  //被测类的声明
import static org.testng.Assert.assertEquals;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
```

Testing 包方法的声明

```
public class AaaTest                             //测试类
{
 private static Aaa calculator1 = new Aaa();
   @Test                                         //测试标注,可带属性
public void testAdd()                            //测试方法
{
     calculator1.add(7);                         //被测方法调用
     calculator1.add(8);
     assertEquals(14, calculator1.getReuslt()); //断言
```

```
        System.out.println("通过测试");          //相关信息打印
    }
      @Test
    public void testSub()
      {
        calculator1.substract(9);
        assertEquals(-9,calculator1.getReuslt());
    }
       @Test(timeOut = 2000L)   //timeout是Test的属性，完成限时测试
    public void testSquare()
    {
        calculator1.square(6);
        assertEquals(36,calculator1.getReuslt());
    }
    public void PrintResult()
    {
        System.out.println("方法Square被忽略，不进行测试");
    }
@Test(expectedExceptions=java.lang.ArithmeticException.class)
     ///完成异常处理测试
    public void testDivide()
    {
        calculator1.divide(0);
    }
    @Test （enable = “false”）  //////配置Test属性，不进行测试
    public void testmultiply()
    {
        calculator1.multiply(4);
        assertEquals(10,calculator1.getReuslt());
    }
    @BeforeMethod
    public void setUp() throws Exception
    {
    calculator1.clear();
        System.out.println("This is Before!!!");
    }
    @AfterMethod
    public void tearDown() throws Exception
    {
        System.out.println("This is After!!!");
    }
    @BeforeClass        ///完成测试方法前置工作，在测试方法执行前会完成
   前置工作
    public static void setUp1() throws Exception
```

```
    {
        calculator1.clear();
        System.out.println("This is BeforeClass!!!");
    }
    @AfterClass
    public static void tearDown1() throws Exception
    {
        System.out.println("This is AfterClass!!!");

    }
}
```

其中被测代码如下：

```
package aaa;
public class Aaa {
 ///////////////成员函数///////////////
 /*********************/
 /********加法*********/
 /*********************/
 public void add(int n){
     result += n;
 }
     /*********************/
     /********减法*********/
     /*********************/
 public void substract(int n){
         result -= n;
 }
 /*********************/
 /********乘法*********/
 /*********************/
 public void multiply(int n){
         result *= n;
     }
 /*********************/
 /********除法******/
/ *********************/
public void divide(int n){
     result /= n;
 }
/*********************/
/********平方*********/
/*********************/
public void square(int n){
    result = n * n;
```

```
    }
```

Coverage Report

Test Report

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Command line suite						
Command line test	3	0	1	47		

Class	Method	Start	Time (ms)
Command line suite			
Command line test — failed			
aaaTest.AaaTest	testAdd	1398752016828	15
Command line test — passed			
aaaTest.AaaTest	testDivide	1398752016843	0
	testSquare	1398752016843	0
	testSub	1398752016843	16

Command line test

aaaTest.AaaTest#testAdd

Exception

```
        /**********************/
        /********取值*********/
        /**********************/
    public int getReuslt(){
            return result;
    }
        /**********************/
        /********清值********/
        /**********************/
    public void clear(){
            result = 0;
        }
    ////////////成员变量////////
        /**********************/
        /********结果*********/
        /**********************/
    private static int result;
    }
```

- 在 testsrc 目录下创建 testing.xml：对测试进行配置

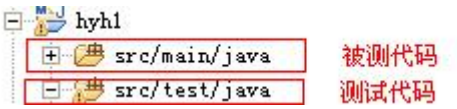
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="none">
<test name="Test">
    <classes>            //需要执行的测试类
      <class name="aaaTest.AaaTest"/>
    </classes>
  </test>
</suite>
```

- 两种方式运行：第四部分已经有过介绍，这里就不再累述。命令行生成的报告如下图

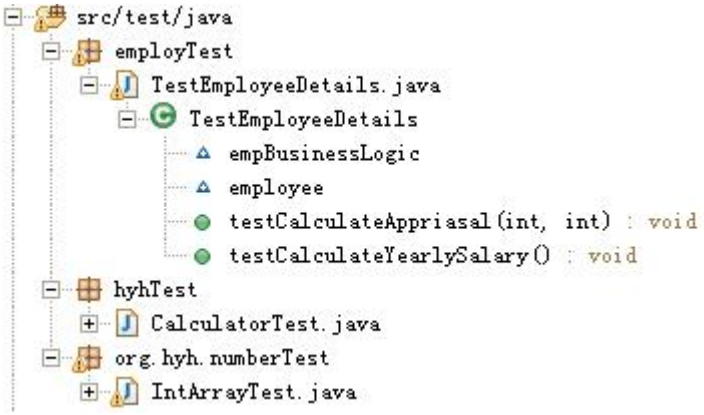
5.2 基于 Eclipse+maven 的测试实例演示

- 用 maven 进行构建项目构建

  在使用 Maven 进行构建后自动生成测试文件夹与被测程序文件夹，



- 同样的方法创建测试包与测试类



- 在项目的 src/test 目录下创建 testng.xml 文件对测试进行配置

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Suite1"    verbose="1" >
    <test name="Regression1" >                          //测试组
     <parameter name="data1" value="7000"></parameter>  ///参数化测试
    <parameter name="data2" value="500"></parameter>
    <classes>                  //选择需要执行的测试类
        <class name = "employTest.TestEmployeeDetails">
          <methods>          //选择需要测试的方法
               <include name = "testCalculateAppriasal"/>
             </methods>
            </class>
          </classes>
          <packages>                ////这里可以对需要执行的测试包进行选择
             <package name="hyhTest"/>
             <package name="employTest"/>
             <package name="org.hyh.numberTest"/>
          </packages>
          <classes>
             <class name = "org.hyh.numberTest.IntArrayTest"/>
             </classes>
        </test>
       </suite>
```

- 在 pom.xml 中加入对 TestNG 与 cobertura 的依赖，并配置对 testng.xml 的操作。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>hyh1</groupId>
  <artifactId>hyh1</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>hyh1</name>
  <url>http://maven.apache.org</url>
  <properties>
   <project.build.sourceEncoding>UTF-8</project.build.source
   Encoding>
  </properties>
  <dependencies>
    <dependency>
    <groupId>org.testng1</groupId>
    <artifactId>testng1</artifactId>
    <version>6.8.1</version>
    </dependency>
  </dependencies>
  <build>
  <plugins>
     <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-surefire-plugin</artifactId>
     <version>2.5</version>
     <configuration>
     <useSystemClassLoader>true</useSystemClassLoader>
       <testFailureIgnore>true</testFailureIgnore>
         <parallel>false</parallel>
         <forkMode>once</forkMode>
           <suiteXmlFiles>
           <suiteXmlFile>testng.xml</suiteXmlFile>
<!-- 配置testng.xml文件中的 信息-->>
</suiteXmlFiles>
      </configuration>
     </plugin>
   </plugins>
  </build>
</project>
```

在 pom.xml 中加入对 testng 的依赖

加载 cobertura

执行 TestNG.xml

- 在项目 src/test/lib 文件夹中加入 testng-6.8.1.jar 与 result_maven.jar 文件

特别注意：在用命令行执行测试时，需要先将开发的 testng-6.8.1.jar 加载到本地仓库中，并在 pom.进行如下配置

```
    <dependency>
        <groupId>org.testng1</groupId>
        <artifactId>testng1</artifactId>
    <version>6.8.1</version>
</dependency>
```

执行如下命令行：（仅需执行一次即可）

```
call mvn install:install-file -DgroupId=org.testng1
-DartifactId=testng1 -Dversion=6.8.1 -Dpackaging=jar
-Dfile=.\lib\testng-6.8.1.jar
```

Coverage Report

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	5	70%	29/41	25%	2/8	1.235
calculator	1	78%	11/14	N/A	N/A	1
employee	2	94%	18/19	100%	2/2	1.125
org.hyh.number	2	0%	0/8	0%	0/6	2.5

Report generated by Cobertura 2.0.3 on 14-4-29 下午3:00.

Test Report

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite1						
Regression1	11	0	2	32		

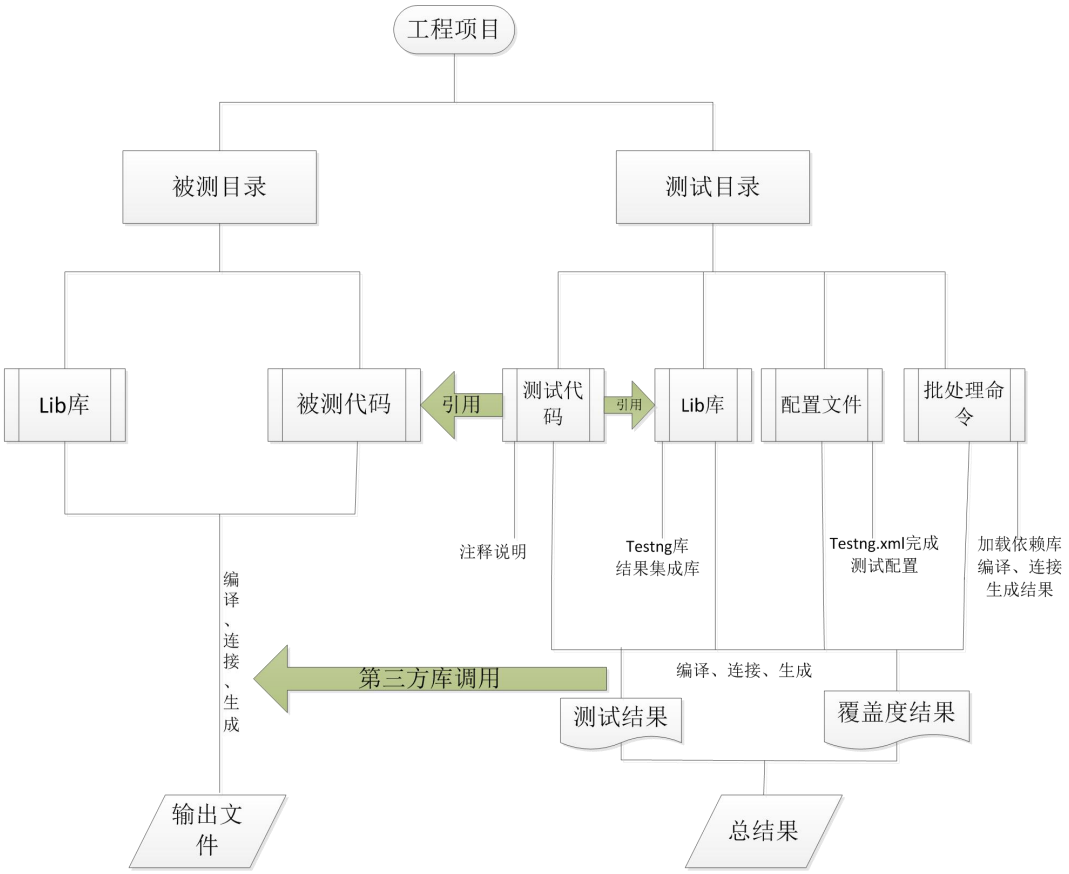
Class	Method	Start	Time (ms)
Suite1			
Regression1 — failed			
employTest.TestEmployeeDetails	testCalculateAppriasal	1398754829625	0
org.hyh.numberTest.IntArrayTest	MapTest	1398754829609	0
Regression1 — passed			
employTest.TestEmployeeDetails	testCalculateAppriasal	1398754829625	0
	testCalculateAppriasal		
	testCalculateAppriasal		
	testCalculateAppriasal		
	testCalculateAppriasal		
	testCalculateYearlySalary	1398754829625	0
hyhTest.CalculatorTest	testAdd	1398754829609	0
	testDivide	1398754829609	0
	testSquare	1398754829609	16
	testSub	1398754829625	0
org.hyh.numberTest.IntArrayTest	ArrayListTest	1398754829593	16

- 执行 src/test 目录下的 run.bat（命令行的编写格式见 4.2）,生成如下测试报告

六、单元测试问题与解决

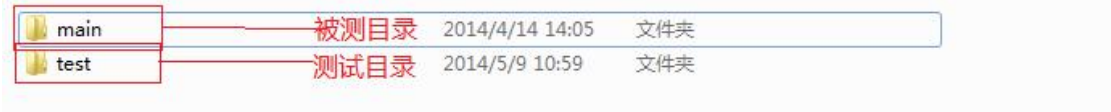
6.1 项目测试管理

为了使项目与测试的条理更加清晰，设计测试管理框图如下：

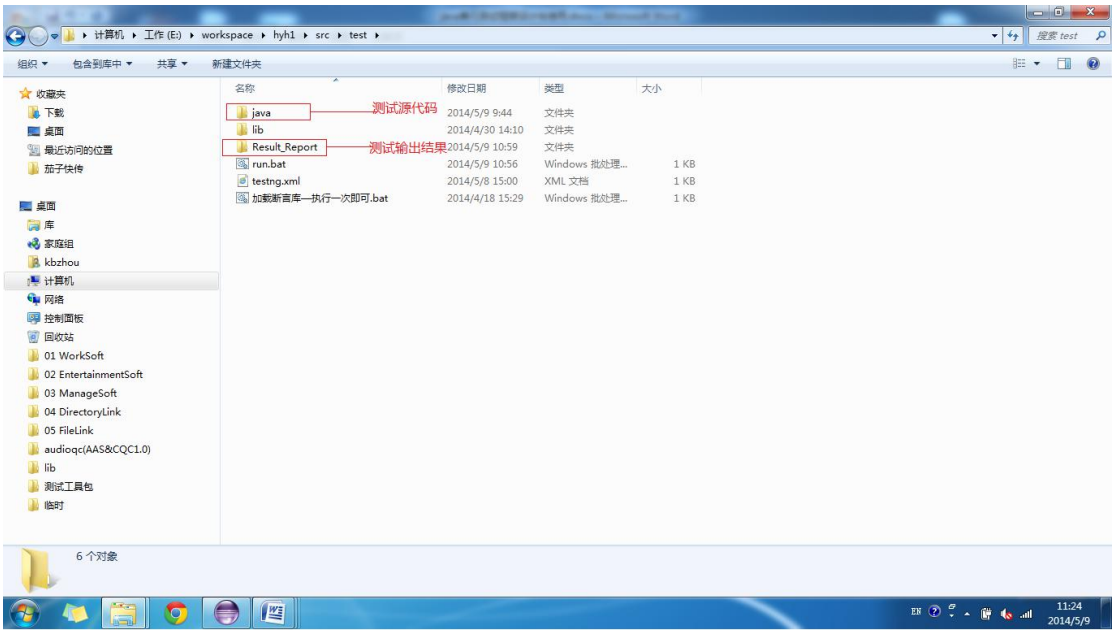


6.1.1 测试目录的管理

为了实现对测试代码与被测代码相隔离，对测试代码与被测代码建立独立目录进行管理。在项目根目录下创建 src/test 目录，该目录下存放包括：测试源代码、配置文件、需调用的库文件、批处理命令。工程的物理逻辑如下图：



Test 目录下：



6.1.2 配置 xml 文件完成对测试的管理

在测试过程中对 tesng.xml 文件进行配置即可实现对测试的管理，如制定需执行的测试套件、测试包、测试类、测试方法等，也可实现参数化测试的配置。

具体配置示例见 5.2 中的介绍。

6.1.3 测试代码注释编写

在测试代码编写的同时需要对测试功能进行注释，可以使测试代码更加清晰。示例如下：

```
/*
 * 测 试 类:  employee.EmpBusinessLogic.class
 * 测试方法: CalculateYearlySalary();
  * 输入参数: employee.name="Rajeev",employee.MonthlySalary =
  8000,employee.age=25;
 * 期 望 值: 500
 */
@Test
public void testCalculateYearlySalary() {
    employee.setName("Rajeev");
    employee.setAge(25);
    employee.setMonthlySalary(8000);
    double salary = empBusinessLogic
        .calculateYearlySalary(employee);
```

```
        Assert.assertEquals(salary, 500);
    }
```

6.2 测试代码编写

6.2.1 Private 及 protected 成员函数测试方法

当 java 的类有一个方法的访问修饰符为 Private 或 protected 时，我们在写测试类时需要将其私有方法抽出，让其可以访问，就可以按要求进行测试了。可以使用 Method 这个类将我们要测试的方法赋给它，其中它有一个设置，可是方法变成可进入，这样就可达到可测试的目的。

示例：

```
package calculator;
public class Calculator {
    ///////////////成员函数///////////////
    /*********************/
    /********加法*********/
    /*********************/
    private void add(int n)            //私有方法
    {
        += n;
       }
……................................................ // 以下代码省略
……................................................
```

测试代码：

```
package calculatorTest;
import static org.testng.Assert.assertEquals;

import java.lang.reflect.Method;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.ExpectedExceptions;
import org.testng.annotations.Test;
import calculator.Calculator;
public class CalculatorTest
{
    private static Calculator calculator1 = new Calculator();
    @Test
    public void testAdd()
    {
```

```
        Object calculatorObject = null;
        try
        {
            Method method =
            calculator1.getClass().getDeclaredMethod("add", new
            Class[]{int.class});
            method.setAccessible(true);    //设置抽出方法可用
            calculatorObject=method.invoke(calculator1, new
            Object[]{3});
            calculatorObject=method.invoke(calculator1, new
            Object[]{12});
            /////模拟方法调用，Object[]{12}为传入参数
        }
        catch(Exception e)
        {

        }
        assertEquals(15, calculator1.getReuslt());
    }
……................................................
……................................................    ////// 以下代码省略
```

注意：导入 Method 类的时候要为引用 java.lang.reflect.Method 类。

6.2.2 桩代码编写

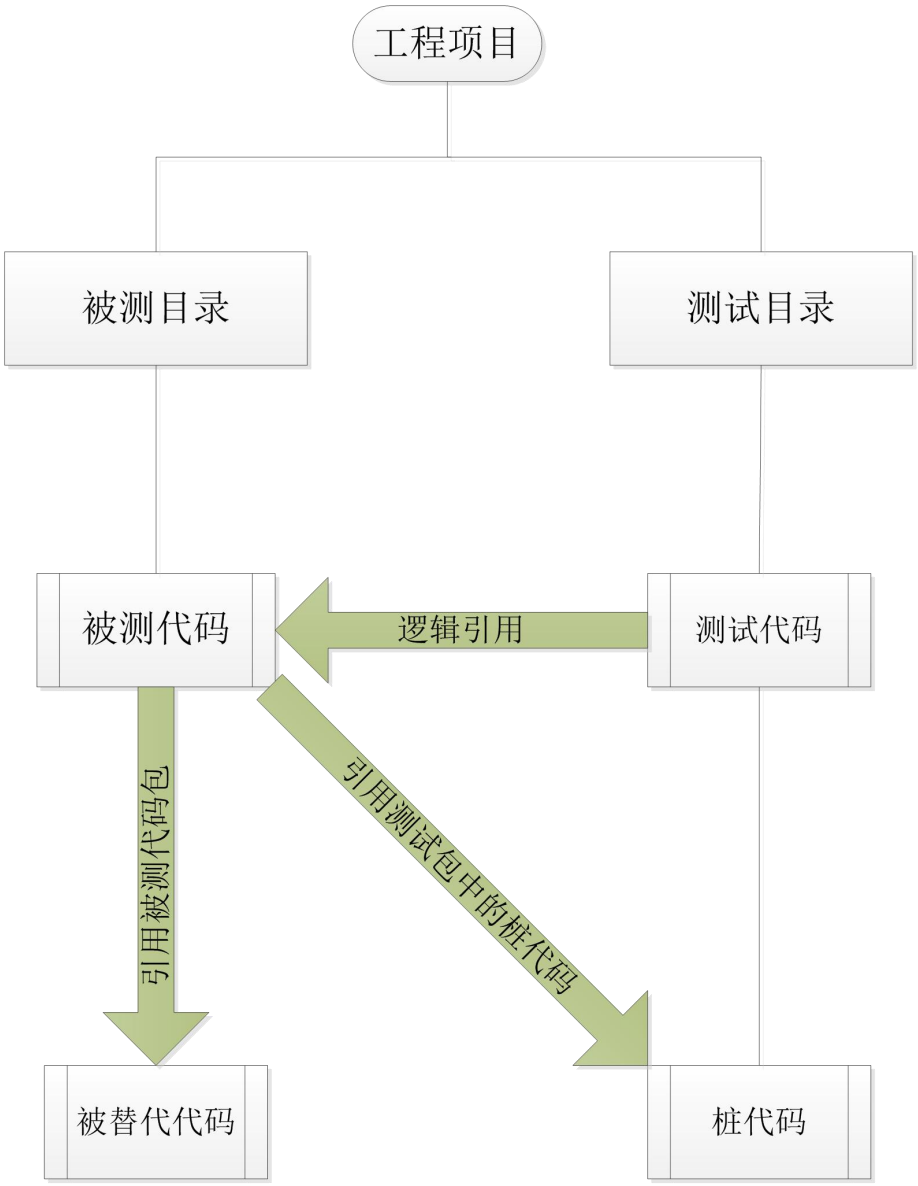
由于单元测试是针对代码开发过程中对方法或模块进行测试，但在测试中会遇到被测代码中包含未完成或暂不可实现的函数及代码块。可以使用 “桩”代码替代，从而可以针对被测函数的代码逻辑进行提前测试。

- 何时需要编写桩代码
  1. 从被处理数据来源看，如果某段未完成代码（函数）是从被测程序外部获取数据时，我们可以用桩代码进行代替。
  2. 从复杂度方面看，如果某段代码（函数）在被测程序层面为顺序执行代码时，我们可以用桩代码进行代替。
  3. 如果未完成代码传入数据为被测代码中的局部变量或者被测代码中频繁操作的全局变量，并且未完成代码中存在比较复杂的代码逻辑，存在判断、循环等逻辑语句，此时不建议用桩代码来替代这段代码，最好能够在这段代码编写完成之后再进行单元测试。
- 桩代码写在哪里

  桩代码是测试代码的一部分，只有执行测试时，才进行桩代码替换，因此应该将桩代码写在测试代码目录下。
- 如何将桩代码与原程序进行物理隔离

  通过存放不同的物理目录，使源代码与桩代码进行隔离，在项目工程中通过引用不同包

名中的同名类，来分别调用桩代码或原代码。



具体示例：

在示例中需要对被测包 stub 包中的 DriveModul 类中的 readFile()方法进行测试，在 readfile() 会调用被测代码包 read 中的 StubModul.readTargetFile() 方法，通过 StubModul.readTargetFile() 的返回值获取读入文件的路径。在实际测试中，StubModul.readTargetFile() 方法并未完成，需在测试代码中编写桩代码对 StubModul.readTargetFile()方法进行替换，获取合适的返回值。

被测代码的 DriveModul 类如下：

```
package stub;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
```

```
import stubTest.StubModul;        //测试过程中引用桩代码类
//import read.StubModul;          //若被测代码已完成编写，可引用被测
代码中的类
public class DriveModul
{
    private String path;
    public String readFile()
    {

        StubModul stubModul = new StubModul();
         path = stubModul.readTargetFile("C:\\01.txt","GBK");
                                                  桩代码调用
         //read.stubModul.readTargetFile未完成
         //测出调用哪个包中的stubModul.readTargetFile由
         // import stubTest.StubModul或import read.StubModul决定
        StringBuffer sbStr = new StringBuffer();
         try
         {
              File ff = new File(path);
               InputStreamReader read = new InputStreamReader(new
               FileInputStream(ff),
                    "GBK");
                BufferedReader ins = new BufferedReader(read);

                String dataLine = "";
                while (null != (dataLine = ins.readLine()))
                {
                   sbStr.append(dataLine);
                }
                ins.close();
            }
            catch (Exception e)
            {
            }

            return sbStr.toString();
    }
}
```

测试代码中的桩代码如下：

```
package stubTest;

public class StubModul
```

```
{
    public String readTargetFile(String filePath,String sEncode)
    {
        return "C:\\02.txt";      //直接返回所需要的返回值
    }
}
```

在测试或项目执行时，仅需改动如下的注释即可控制调用桩代码或源代码。

```
import stubTest.StubModul;        //测试过程中引用桩代码类
//import read.StubModul;          //若被测代码已完成编写，可引用被测
代码中的类
```