第二十六章　Tomcat与Web测试

在实际的开发中，使用必要的工具进行程序的测试是非常有必要的工作。在本章中介绍使用JUnit和Cactus，其中，JUnit是进行普通Java程序测试用的，而Cactus是JUnit的扩展，是用于服务器端程序测试的工具。本章中就介绍如何使用这两个工具。

26.1　JUnit简介安装配置

在一个软件开发项目中，软件的测试一个必不可少的工作，为了保证工程的质量需要对软件进行测试有：功能测试、性能测试、安全性测试、稳定性测试、浏览器兼容性测试等多项测试。其中功能测试又是最基本的一项测试，它是其他的测试的基础。

对于Java程序而言，JUnit是一个非常优秀的单元测试工具，可以进行有效的功能测试，不过JUnit本身只能对普通的Java程序进行测试，而对于服务器端的程序，如JSP、Servlet等就无能为力了。

JUnit是一个用于编写和运行可重复测试的Java开源测试框架，它最先由Erich Gamma和Kent Beck编写，是进行单元测试框架的xUnit体系结构的一个实例。JUnit的特点包括：

- 使用断言判断期望值和实际值的差异，返回boolean值。
- 测试驱动设备使用共同的初始化变量或者实例。
- 测试包结构便于组织和集成运行。
- 有命令行和字符测试程序。

JUnit是当前Java语言单元测试的一站式解决方案，它把测试驱动的开发思想介绍给Java开发人员并教给他们如何有效地编写单元测试。众多的优点使得它成为一种优秀的测试工具，在本章就介绍如何使用JUnit进行Java的单元测试。

JUnit就是对Java语言进行白箱测试、单元测试的有效工具，可用于类测试、功能测试、接口测试以及针对对象的测试。

26.1.1　JUnit简介

JUnit测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。它本质上是一套框架，即开发者指定了一套条条框框，遵循这此条条框框要求编写测试代码，如继承某个类，实现某个接口，就可以用JUnit进行自动测试了。使用JUnit进行软件测试还是有很多好处的：

- 可以使测试代码与产品代码分开。
- 针对某一个类的测试代码通过较少的改动便可以应用于另一个类的测试。
- 易于集成到测试人员的构建过程中，JUnit和Ant的结合可以实施增量开发。
- JUnit是开放源代码的，可以进行二次开发。
- 可以方便地对JUnit进行扩展。

JUnit的框架包括。

- 对测试目标进行测试的方法与过程集合，可称为测试用例(TestCase)。
- 测试用例的集合，可容纳多个测试用例(TestCase)，将其称作测试包(TestSuite)。
- 测试结果的描述与记录（TestResult。
- 测试过程中的事件监听者(TestListener)。
- 每一个测试方法所发生的与预期不一致状况的描述，称其测试失败元素

(TestFailure)。

- JUnit Framework中的出错异常（AssertionFailedError）。

JUnit框架是一个典型的Composite模式：TestSuite可以容纳任何派生自Test的对象；当调用TestSuite对象的run()方法时，会使用反射机制遍历自己容纳的对象，逐个调用它们的run()方法。

26.1.2 安装配置

JUnit的安装很简单，首先到http://prdownloads.sourceforge.net/junit/junit3.8.1.zip?download下载一个JUnit3.8.1版的软件包，下载完成后将其解压到某个目录下，并将其设定为<JUNIT_HOME>，然后把<JUNIT_HOME>下的junit.jar包加入到系统的CLASSPATH环境变量中。

然后运行如下命令：

```
java junit.swingui.TestRunner junit.samples.AllTests
```

结果如图26.1所示。

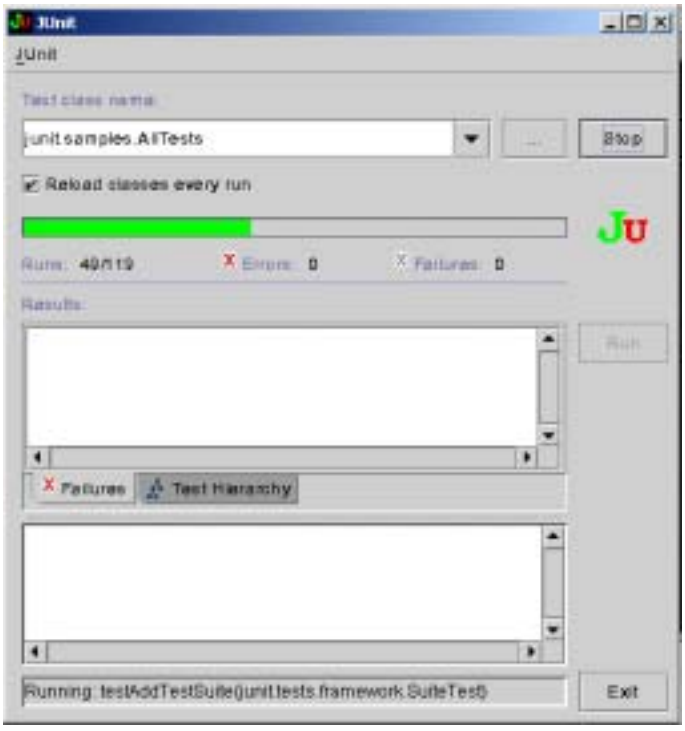

图26.1 JUnit安装成功测试

如果能看到图26.1则表明JUnit安装配置成功了。

26.2 JUnit中常用的接口和类

JUnit中常用的接口和类主要有：Test接口、TestCase抽象类、Assert静态类、TestSuite类等，下面对这几个常用的接口和类进行分别介绍。

26.2.1 Test接口——运行测试和收集测试结果

Test接口使用了Composite设计模式，是单独测试用例（TestCase），聚合测试模式（TestSuite）及测试扩展（TestDecorator）的共同接口。

它的public int countTestCases（）方法，用来统计这次测试有多少个TestCase，另外一个方法就是public void run（TestResult），TestResult是实例接受测试结果，run方法执行本次测试。

26.2.2 TestCase抽象类——定义测试中固定方法

TestCase是Test接口的抽象实现，（不能被实例化，只能被继承）其构造函数TestCase(string name)根据输入的测试名称name创建一个测试实例。由于每一个TestCase在创建时都要有一个名称，若某测试失败了，便可识别出是哪个测试失败。

TestCase类中包含setUp()、tearDown()方法。setUp()方法集中初始化测试所需的所有变量和实例，并且在依次调用测试类中的每个测试方法之前再次执行setUp()方法。tearDown()方法则是在每个测试方法之后，释放测试程序方法中引用的变量和实例。

```
//初始化测试所需的所有变量和实例
        setUp();
        try {
//执行测试
            runTest();
        }
        finally {
//释放测试程序方法中引用的变量和实例
            tearDown();
        }
```

开发人员编写测试用例时，只需继承TestCase，来完成run方法即可，然后JUnit获得测试用例，执行它的run方法，把测试结果记录在TestResult之中。

26.2.3 Assert静态类——一系列断言方法的集合

Assert包含了一组静态的测试方法，用于比对期望值和实际值是否相同，如果不相同，即测试失败，Assert类就会抛出一个AssertionFailedError异常，JUnit测试框架将这种错误归入Failes并加以记录，同时标志为未通过测试。如果该类方法中指定一个String类型的传入参数，则该参数将被做为AssertionFailedError异常的标识信息，告诉测试人员改异常的详细信息。

JUnit提供了6大类31组断言方法，包括基础断言、数字断言、字符断言、布尔断言、对象断言。

- assertEquals（Object expcted,Object actual)：内部逻辑判断使用equals()方法，这表明断言两个实例的内部哈希值是否相等时，最好使用该方法对相应类实例的值进行比较。
- assertSame（Object expected,Object actual）：内部逻辑判断使用了Java运算符“ == ”，这表明该断言判断两个实例是否来自于同一个引用（Reference），最好使用该方法对不同类的实例的值进行比对。
- asserEquals(String message,String expected,String actual)：该方法对两个字符串进行逻辑比对，如果不匹配则显示着两个字符串有差异的地方。ComparisonFailure类提供两个字符串的比对，不匹配则给出详细的差异字符。

26.2.4 TestSuite测试包类——多个测试的组合

TestSuite类负责组装多个Test Cases。待测的类中可能包括了对被测类的多个测试，而TestSuit负责收集这些测试，使开发者可以在一个测试中，完成全部的对被测类的多个测试。TestSuite类实现了Test接口，且可以包含其他的TestSuites。它可以处理加入Test时的所有抛出的异常。TestSuite处理测试用例有6个规约（否则会被拒绝执行测试）

- 测试用例必须是公有类（Public）。
- 测试用例必须继承与TestCase类。
- 测试用例的测试方法必须是公有的（Public）。
- 测试用例的测试方法必须被声明为void。
- 测试用例中测试方法的前置名词必须是test。
- 测试用例中测试方法误传任何传递参数。

26.2.5 TestResult结果类和其他类与接口

TestResult结果类集合了任意测试累加结果，通过TestResult实例传递给每个测试的Run()方法。TestResult在执行TestCase时如果失败会抛出异常。

TestListener接口是个事件监听规约，可供TestRunner类使用。它通知listener对象的相关事件，方法包括测试开始startTest(Test test)，测试结束endTest(Test test)，错误，增加异常addError(Test test,Throwable t)和增加失败addFailure(Test test,AssertionFailedError t)等。

TestFailure失败类是个“失败”状况的收集类，解释每次测试执行过程中出现的异常情况。其toString()方法返回“失败”状况的简要描述。

26.3 使用JUnit进行软件测试

使用JUnit进行软件测试的基本步骤比较简单，它的基本使用步骤如下：

- 创建从junit.framework.TestCase派生unit test需要的test case。
- 书写测试方法，提供类似于如下函数签名的测试方法：public void testXXXXX()。
- 编译，书写完test case后，编译所写的test case类。
- 运行，启动junit test runner，来运行这个test case。

JUnit提供了2个基本的test runner：字符界面和图形界面。启动命令分别如下。

- 图形界面：java junit.swingui.TestRunner XXXXX
- 字符界面：java junit.textui.TestRunner XXXXX

注意：XXXXX 是要进行测试的完整类名。

26.3.1 创建测试实例并书写测试方法

下面举一个小例子，测试一个类的两个方法，首先是在单个测试用例中测试，然后把要测试的类打包为了一个测试包，进行测试。

1. 编写要测试的类

这个类代表一个银行帐号，为了简化，这里只有用户名和钱数两个变量以及相应的getter和setter方法，以及对用于转帐的一个方法，以及判断两个帐户钱数之和与某个帐户是否想等的方法，下面是这个类的源代码。

```
package cn.ac.ict;
```

```
public class Account {
    private String username;
    private float rmb;
    public Account(String name,float money){
        this.username = name;
        this.rmb = money;
    }
    // username属性的设置和获取方法
    public void setUsername(String name){
        this.username = name;
    }
    public String getUsername(){
        return this.username;
    }
    // rmb属性的设置和获取方法
    public void setRmb(float money){
        this.rmb = money;
    }
    public float getRmb(){
        return this.rmb;
    }
//对两个Account对象进行操作
    public boolean Operate(Account acc){
        if(username.equals(acc.getUsername())){
            if(this.rmb>acc.getRmb()){
                rmb-=acc.getRmb();
            }else{
                return false;
            }
        }else if(!username.equals(acc.getUsername())){
            rmb+=acc.getRmb();
        }
        return true;
    }
//用于判断两个Account对象是否相同
    public boolean equals(Object obj1,Object obj2){
        if((obj1 instanceof Account)&&(obj2 instanceof Account)){
            Account acc1 = (Account)obj1;
            Account acc2 = (Account)obj2;
            return (rmb==(acc1.getRmb()+acc2.getRmb()));
        }
        return false;

    }

}
```

2. 编写单个的测试用例

单个的测试用例是指继承了TestCase抽象类的测试类，它使用形如testXXXX()的方法对

XXXX方法进行测试，AccountTest类的源代码如下：

```
package cn.ac.ict;

import junit.framework.Assert;
import junit.framework.TestCase;

public class AccountTest extends TestCase {
    Account acc1 ;
    Account acc2 ;
//初始化
    protected void setUp() throws Exception {
        super.setUp();
        acc1 = new Account("zhw",25663.7F);
        acc2 = new Account("rambler",2563.7F);

    }
//回收资源
    protected void tearDown() throws Exception {
        super.tearDown();
    }
//测试Operate方法
    public void testOperate() {
        Account expected = new Account("all",28227.4F);
        Account temp = new Account("rambler",325.4F);
        acc1.Operate(temp);
        acc2.Operate(temp);
        Assert.assertTrue(expected.equals(acc1,acc2));
    }
//测试equals方法
    public void testEqualsObjectObject() {
        Assert.assertTrue(!acc1.equals(null,null));
        Assert.assertTrue(acc1.equals(acc1,new Account("zero",0.0F)));
        Assert.assertTrue(!acc1.equals(acc1,acc2));
    }

}
```

setUp与tearDown，这两个方法是junit framework中提供初始化和反初始化每个测试方法的。setUp在每个测试方法调用前被调用，负责初始化测试方法所需要的测试环境；tearDown在每个测试方法被调用之后被调用，负责撤销测试环境。它们与测试方法的关系可以描述如下：

```
测试开始 -> setUp -> testXXXX -> tearDown ->测试结束
```

编译AccountTest.java（需要把包junit.jar加入到类路径中），然后在命令行中输入如下命令。

```
java junit.awtui.TestRunner cn.ac.ict.AccountTest
```

运行效果如图26.2。也可以使用如下命令运行。

```
java junit.textui.TestRunner cn.ac.ict.AccountTest
java junit.swingui.TestRunner cn.ac.ict.AccountTest
```

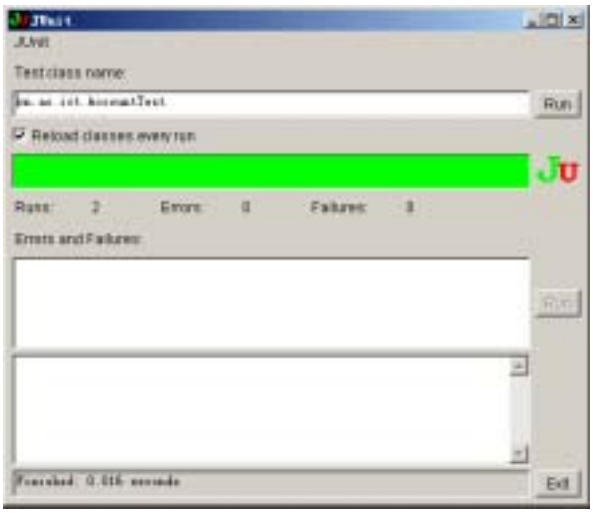

图26.2　单体测试成功

26.3.2　组装TestSuite

利用TestSuite可以将一个TestCase子类中所有testXXX()方法包含进来一起运行，还可将TestSuite子类也包含进来，从而行成了一种等级关系。可以把TestSuite视为一个容器，可以盛放TestCase中的testXXX ()方法，它自己也可以嵌套。这种体系架构，非常类似于现实中程序一步步开发一步步集成的现况。

对于上面的例子，可以编写一个如下所示的TestSuite：

```
package cn.ac.ict;

import junit.framework.Test;
import junit.framework.TestSuite;

public class AccountTests {

    public static void main(String[] args) {
        junit.swingui.TestRunner.run(AccountTests.class);
    }

    public static Test suite() {
        TestSuite suite = new TestSuite("Test for cn.ac.ict");
        //$JUnit-开始测试$
        suite.addTestSuite(AccountTest.class);
        //$JUnit-测试结束$
        return suite;
    }
}
```

在这个例子中，只有一个TestCase，所以在TestSuite也只是加了一个TestCase，如果有更多的TestCase或者其他的TestSuite，可以加到这个TestSuite中来，把很多的测试只需要运行一个程序来实现。

在命令行中输入：

```
java cn.ac.ict.AccountTests或者
```

```
java junit.swingui.TestRunner cn.ac.ict.AccountTests
```
可以看到效果如图26.3。

图26.3 测试包运行成功

26.4 使用Cactus测试Web应用

Cactus是一个基于JUnit框架的简单测试框架，用来单元测试服务端Java代码。Cactus框架的主要目标是能够单元测试服务端的使用Servlet对象的Java方法如HttpServletRequest、HttpServletResponse、HttpSession等。

Cactus的测试分为三种不同的测试类别，JspTestCase、ServletTestCase、FilterTestCase。下面简单介绍一下Cactus的原理和测试过程，如图26.4。

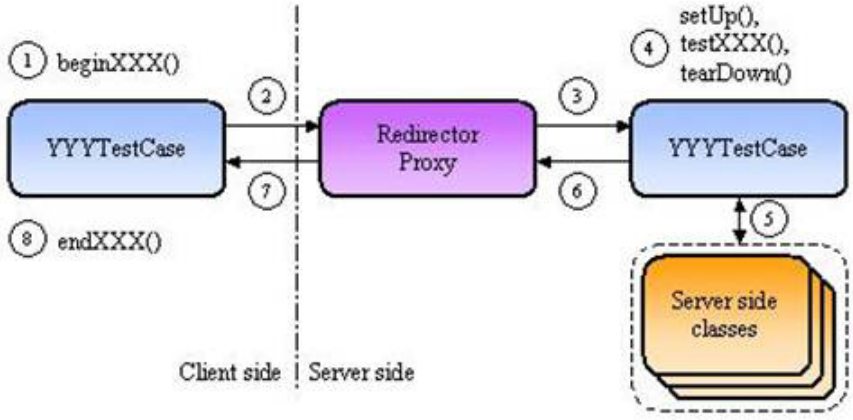

图26.4 Cactus的原理和测试过程

- JUnit Test Runner调用YYYTestCase.runTest()，这个方法寻找beginXXX(ServletTestRequest)。
- YYYTestCase.runTest()打开一个到Redirector Proxy的HTTP连接。

- Redirector Proxy进行如下操作：
  创建Test class的实例。
  创建一些Server对象（HttpServletRequest、ServletConfig、ServletContext）的Cactus wrapper。
  如果需要，创建一个HTTP Session。
- Redirector Proxy通过Reflection，执行Test类的setUP()、testXXX()、tearDown()。
- testXXX()调用Server side classes的方法，并通过JUnit的assert API来验证测试结果。
- 如果测试失败，testXXX()方法抛出例外，Redirector Proxy会捕获例外。
- Redirector Proxy向客户端返回例外的有关信息，JUnit会将这些信息打印出来。
- 如果没有发生例外，YYYTestCase.runTest()寻找并执行。
  endXXX(HttpURLConnection)，在这儿可以使用JUnit asserts检查返回的HTTP Header、servlet output stream。

下面举一个在Tomcat下使用Cactus测试Servlet的例子。

26.4.2 安装Cactus

Cactus是Apache的一个开源项目，使用Cactus对Web应用进行测试，需要下载安装支持Cactus的JAR文件。

Cactus可以从其官方网站http://jakarta.apache.org/cactus/免费下载，其最新版本为1.7.1，下载后，把jakarta-cactus-12-1.7.1.zip解压到硬盘上，然后把解压目录下lib下的包文件拷贝到Web应用的/WEB-INFO/lib目录下就可以了。

26.4.2 使用Cactus测试Servlet

在这个例子中，创建了一个用于验证用户登录的Servlet，为了简化例子的实现，这里只是实现了这个Servlet，而对于用于提供用户名和密码的JSP页面和登录后的页面不做介绍，读者可以自己添加进去，形成完整的功能。

1. 被测试的Servlet

下面是用于验证用户登录的Servlet：

```
package cn.ac.ict;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet {
    public boolean isValidUser(HttpServletRequest request) {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        if(username == null ||password == null ||!username.equals("justin")
||!password.equals("123456")) {
            return false;
        }
        else {
            return true;
            }
        }
    }
```

其中这个Servlet提供了一个方法用于验证用户名和密码是否有效。

2. 编写Servlet的测试类

Cactus由JUnit扩展而来，所以它们还是有很多相似的地方的，但也有不同，例如，要测试Servlet，就不能继承TestCase，而要继承ServletTestCase，下面是Servlet测试类LoginServletTest.java的源代码。

```
package cn.ac.ict.test;
import org.apache.cactus.ServletTestCase;
import org.apache.cactus.WebRequest;
import cn.ac.ict.LoginServlet;

public class LoginServletTest extends ServletTestCase {
//添加合法用户
    public void beginValidUser(WebRequest webRequest) {
        webRequest.addParameter("username", "justin");
        webRequest.addParameter("password", "123456");
        }
//用于验证用户的合法性
    public void testValidUser() {
        LoginServlet loginServlet = new LoginServlet();
        assertTrue(loginServlet.isValidUser(request));
        }
//添加不合法用户
    public void beginInValidUser(WebRequest webRequest) {
        webRequest.addParameter("username", "guest");
        webRequest.addParameter("password", "123456");
        }
//用于验证用户的不合法性
    public void testInValidUser() {
        LoginServlet loginServlet = new LoginServlet();
        assertFalse(loginServlet.isValidUser(request));
        }
    }
```

在运行测试时TestRunner，调用beginXXX()方法（不是必须的），接收一个WebRequest参数，然后可以在这个方法中在WebRequest中添加一些HTTP相关的信息，每一个testXXX()方法都会有一个beginXXX()方法与之相对应，所以名称上必须符合，beginXXX()方法在testXXX()方法之前被调用。

testXXX()方法中使用了request的名称，在接受到请求后，Container会产生相关的资源，ServletTestRedirector会把这些资源关联到request，之后，就可以使用request进行测试了。

3. 定义Redirector Proxy和TestRunner

在本次测试中使用ServletTestRedirector作为Redirector Proxy（前面有介绍），而且使用了Cactus的ServletTestRunner作为测试时使用的TestRunner，所以，在Web应用的web.xml文件中需要声明这两个Servlet，在web.xml文件中添加如下代码：

```
        <servlet>
            <servlet-name>ServletRedirector</servlet-name>

    <servlet-class>org.apache.cactus.server.ServletTestRedirector</servlet-c
lass>
        </servlet>
        <servlet-mapping>
            <servlet-name>ServletRedirector</servlet-name>
```

```
            <url-pattern>/ServletRedirector</url-pattern>
        </servlet-mapping>
        <servlet>
            <servlet-name>ServletTestRunner</servlet-name>

    <servlet-class>org.apache.cactus.server.runner.ServletTestRunner</servle
t-class>
        </servlet>

        <servlet-mapping>
            <servlet-name>ServletTestRunner</servlet-name>
            <url-pattern>/ServletTestRunner</url-pattern>
        </servlet-mapping>
```

4. 运行Servlet测试

要使用Cactus测试，配置好的本Web应用结构如图26.5。

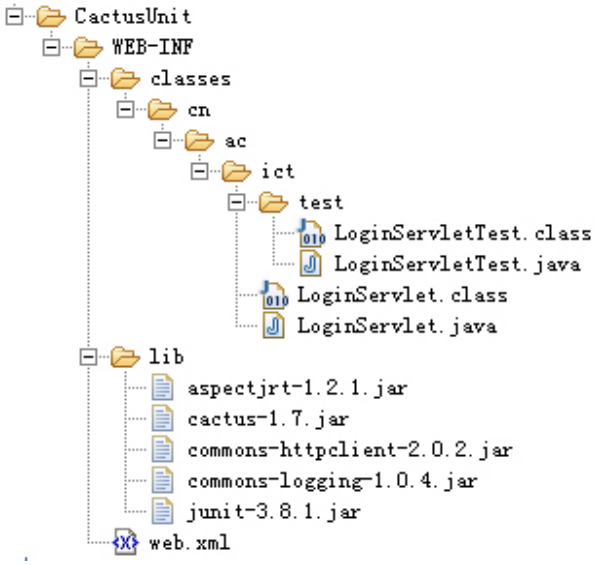

图26.5　Servlet测试应用程序结构

重新启动Tomcat后，在地址栏中输入：

```
    http://localhost:8080/CactusUnit/ServletTestRunner?suite=cn.ac.ict.test.
LoginServletTest
```

可以看到页面显示如图26.6。

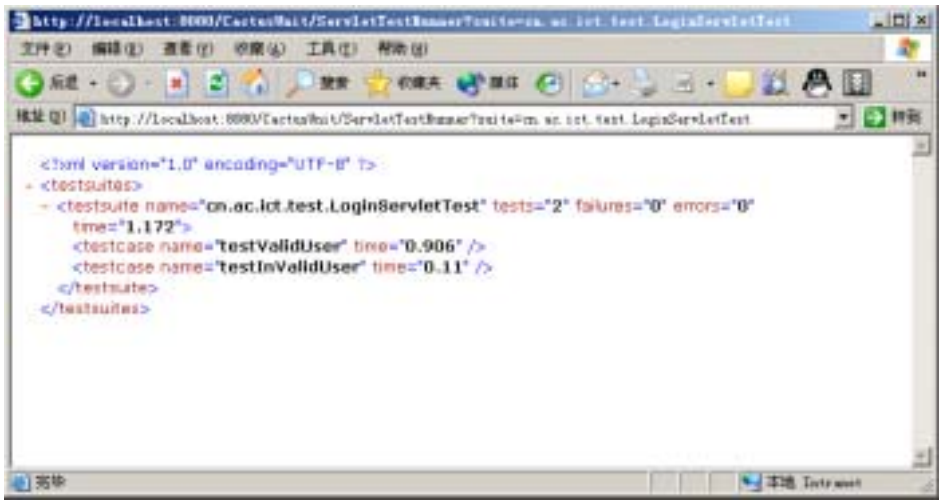

图26.6　Servlet测试应用运行结果

从上面的XML文件中可以看到测试的Suite是cn.ac.ict.test.LoginServletTest，测试了两个方法，没有失败，没有错误，也就是测试通过了！

26.5　相关问题

1. 什么是软件熵？

软件熵（software entropy）是指一个程序从设计很好的状态开始，随着新的功能不断地加入，程序逐渐地失去了原有的结构，最终变成了一团乱麻。

2. 怎么样编写的测试代码才能真正发现问题？

要注意，编写一定能通过的测试代码（如以前举的例子）并没有任何意义，只有测试代码能帮助发现bug，测试代码才有其价值。此外测试代码还应该对工作代码进行全面的测试。如给方法调用的参数传入空值、错误值和正确的值，看看方法的行为是否如你所期望的那样。总之，就是把你的程序放在极端的运行条件下，看它是否依然可以运行。

3. 类的测试方式有哪些？

类有三种不同的测试方式：
1、功能测试接口：测试用例根据类描述确定；
2、构造测试接口：测试用例着重测试类的构造（类可能包含多条构造方法）；
3、交互测试接口：测试用例测试发送消息对一个对象的操作是否正确。
确定这些测试接口是为了后续维护的方便，它也体现了我们遵循何种规则来确定测试用例，并且关系到类的变化对测试用例的影响，表现了使用TestCase的行为实现。
如果类在具体实例化过程种出现了变化，那么就需要调整对应的测试驱动程序。

4. 如何确定类测试用例？

类测试用例的确定和构造取决于准确的类描述，而类描述可能包含类UML设计、类CRC卡片和类状态机等工作。在类测试中介入了两种工作角色（设计人员和实现人员），这样避免了实现人员在类实现期间因误解了类描述，而将错误带入测试用例中，造成测试不准确或者测试失效现象。如果类没有匹配的描述工作，请设计人员不上，虽然可以采用“逆

向工程 ”导出类描述，但是这样的类描述是欠缺的。

（1）根据前置和后置状态确定测试用例

测试用例根据前置和后置条件来确定，其具体实现形式为各个测试方法体，并且在前置条件中指定输入值（包括常见值和边界值）来增加测试用例的测试覆盖率，每一个测试用例都可以包含特殊的前置条件（即错误前置条件）来观察测试用例失败后的异常处理机制是否正确。根据类描述得出测试用例的前置和后置条件，并且根据前置和后置条件的不同组合方式产生不同的测试用例具体测试方法体。

（2）根据状态转换确定测试用例

很多人习惯用状态转换图来确定测试用例而不是根据测试用例的前置条件和后置状态，在状态图中类关联行为明显很容易被测试者辨别，但是这同样要求测试这需要设定更多的特殊值验证类关联关系（例如使用极端的输入值引起了补课预料的状态崩溃），并且使用状态转换图很难确定所有的测试用例。

3、根据代码确定测试用例

5. 什么是断言？

断言是一个布尔语句，该语句不能为假，如果为假，则表明出现了一个bug。

26.6 小结

JUnit是一个非常优秀的开源测试框架，使用它可以对Java的大部分程序进行测试，不过，却不能对服务器端的Servlet、JSP等程序进行测试。

Cactus是一个基于JUnit框架的简单测试框架，用来服务端Java代码进行单元测试。它承袭了JUnit的很多优点，编写测试类、使用方法等都跟JUnit很相似。

本章简单介绍了JUnit框架，以及JUnit框架的扩展实现Cactus，Cactus是进行服务器端程序测试的很好的一个选择。对于使用JUnit进行测试而言，一般是先编写要进行测试的类，然后继承TestCase，实现一个单体测试，当然在需要测试的单体比较多时，可以把多个单体集合为一个Suite。