**Holmos 自动化测试框架**

**——webtest使用文档**

作者：吴银龙（独行者）QQ：307087558

Holmos框架QQ群：181124238

目录

[1：selenium家族介绍 3](#_Toc335665268)

[2：Holmos 自动化测试框架webtest模块介绍 4](#_Toc335665269)

[3：Holmos-webtest模块使用指南 5](#_Toc335665270)

[3.1：运行环境准备 5](#_Toc335665271)

## 1：selenium家族介绍

Selenium是一款web UI自动化测试框架，有了selenium，你能通过selenium对浏览器进行任何手动能够完成的操作；编写selenium的目的就在于完成web 应用的测试工作，但是selenium的功能又不仅仅是这些，selenium还能完成非常令人无聊的基于web的任务调度管理工作。Selenium得到了很多浏览器供应商的支持，这些供应商能够使selenium作为他们作为本地浏览器的一部分而存在；有有一部分已经实现（IE，FF，Chrome，Safari，Opera，Android，Iphone），有一部分正在实现。Selenium同时也作为了其他的开源web测试框架的基础底层。

Selenium是针对Web应用的测试框架，支持多种浏览器和多种编程语言。它是一个浏览器自动化工具，准确执行你所期望的行为——自动化浏览器的某个控件，从而达到自动重复执行任务的目的。

下面我们简要说明一下selenium家族成员：

1：selenium RC（Remote Control）

2：Selenium IDE

3：Selenium 2（Selenium WebDriver）

4：Selenium Grid

原理不在这里一一介绍：需要了解原理，参考官方文档：

RC：<http://seleniumhq.org/docs/05_selenium_rc.html>

IDE：<http://seleniumhq.org/docs/02_selenium_ide.html>

Grid：<http://seleniumhq.org/docs/07_selenium_grid.html>

Webdriver：<http://seleniumhq.org/docs/03_webdriver.html>

使用selenium的优点如下：

● 可直接运行在浏览器之上，就像真实用户所做的一样。

● Selenium 测试可以在 Windows、Linux 和 Mac IOS 等系统下的 Internet Explorer、Firefox 和Chrome，Safari，Opera等10几种浏览器中运行。

● 测试脚本的运行不需要依赖其它任何的工具。

● selenium是开源的不收费，能和其他框架无缝集成，结合使用

## 2：Holmos 自动化测试框架webtest模块介绍

Webtest模块是Holmos自动化测试框架中的web UI测试模块，用来完成web Application的自动化测试工作，底层基于Selenium RC和Selenium Webdriver；webdriver本身针对各个浏览器而开发，通过原生浏览器支持或者浏览器扩展直接控制浏览器，取代了嵌入到被测Web应用中的JavaScript，把端到端的测试与底层的测试工具隔离开来。RC是采用嵌入到浏览器中js，webtest模块对RC模式和webdriver模式均支持。

Webtest模块能够实现PC和移动端web端自动化测试，采用webdriver和seleniumRC作为此模块的底层，采用Page-Object模式组织页面结构，页面结构清晰，并且将业务流程代码和页面结构代码解耦，大幅度降低维护成本，所有的操作均被封装到了各种结构里面，完全面相对象式的编程模型，支持多维度定位页面元素，可以同时用多种方法定位一个元素，只要有一种定位方式成功，便可以成功操作，加上丰富的校验方法，以及详细的日志引擎，可以确定到一个确定的元素确定的操作的操作信息，提供了人性化的多页面操作，可以同时操作多种浏览器，支持各种主流浏览器的主流的操作系统平台，学习成本低廉。

Webtest部分的层次结结构：

Model层：页面元素组织，对Dom元素进行抽象

Action层：页面元素定位，页面元素操作和交互

业务层：根据业务线的需求的业务代码层，包括测试流程，校验流程，测试结果展示层

Model 层

业务层

Action 层

## 3：Holmos-webtest模块使用指南

## 3.1：运行环境准备

#### 3.1.1：JDK的安装

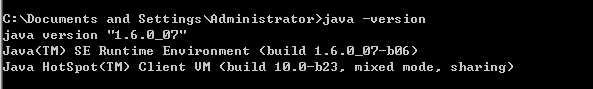
下载jdk1.6及以上版本，下载后进行安装，安装完毕后配置环境变量，设置方法如下：

1）设置JAVA\_HOME=C:\Program Files\Java\jdk1.6.0\_07 注意这里最后没有 ";"

2）设置CLASSPATH=.;%Java\_Home%\lib\dt.jar;%Java\_Home%\lib\tools.jar;%Java\_Home%\lib\jconsole.jar;%Java\_Home%\lib\htmlconverter.jar;%Java\_Home%\bin;%Java\_Home%\lib\javaee.jar 注意这里最前面需要".;"

1. 设置 path=%Java\_Home%\bin;

4）验证是否正确：命令行中输入java –version出现对应版本信息说明jdk安装正确



大家根据自己的路径，设置，这个只是我下载的版本及其安装位置的设置，记住，JAVA\_HOME配置的是你的JDK安装的位置。

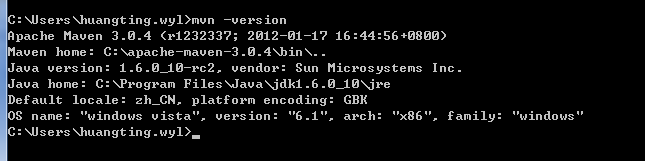
#### 3.1.2：maven的安装

[**http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.0.4/binaries/apache-maven-3.0.4-bin.zip**](http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.0.4/binaries/apache-maven-3.0.4-bin.zip)

1)上面是maven的下载地址，下载下来之后，最好放到C盘根目录，接下来开始设置maven的环境变量：新建MAVEN\_HOME："C:\Program Files\apache-maven-2.2.1"

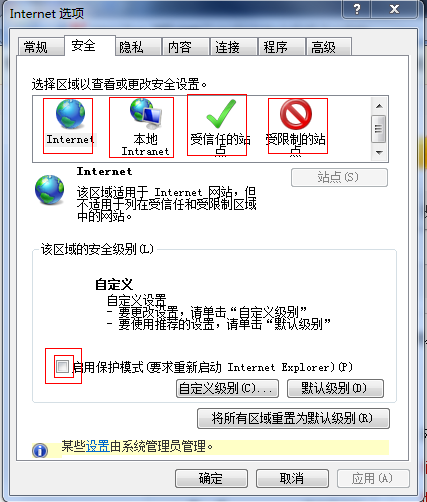
2)设置 path=%MAVEN\_HOME%\bin;

3)验证是否正确：命令行中输入mvn –version出现对应版本信息说明maven安装正确



### 3.1.3:IE 浏览器设置

1）Internet Explorer属性->安全



分别点击上面四个红框中的选项，将这写选项下的启用保护模式的选项一致（要么都勾选，要么都不选）

图2-1-1(a) Internet Explorer安全属性设置

2）Internet Explorer属性->高级->安全，不要勾选下面两项，具体如图2-1-1(b)所示：

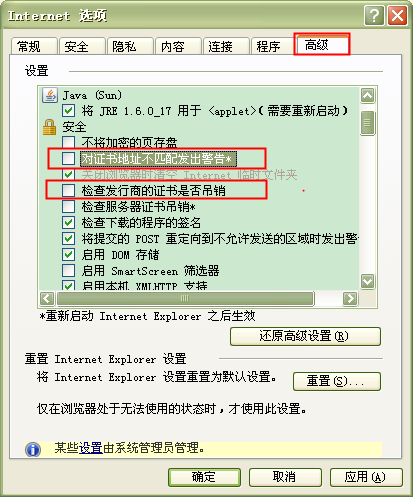


图2-1-1(b) Internet Explorer高级属性设置

### 3.1.4:火狐 浏览器设置

1）工具->选项->高级->更新，选择【不检查更新】 ，去掉【自动更新】中的附加组件勾选，具体如图2-1-1(c)所示：

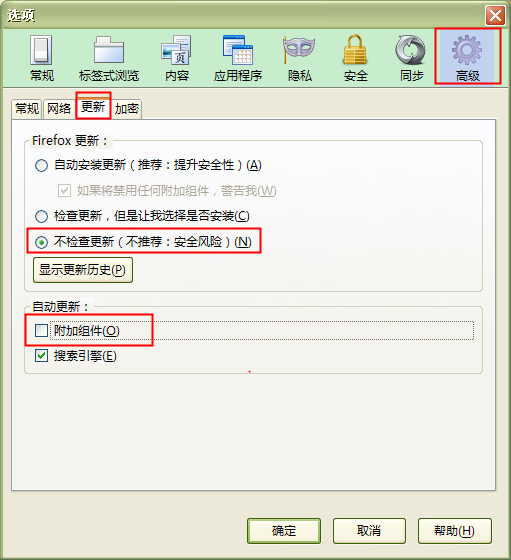
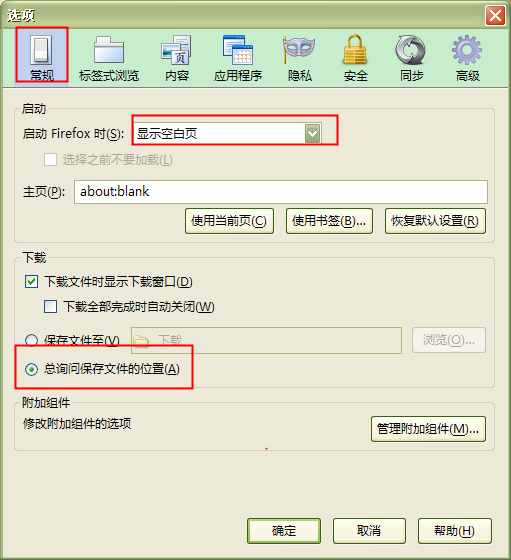


图2-1-1(c) FireFox 高级属性设置

2）工具->选项->常规，【启动】选择显示空白页 ，【下载】选择总询问保存文件的位置，具体如图2-1-1(d)所示：



### 3.1.5:Chrome（谷歌）浏览器设置

1）下载chromedriver.exe文件：

下载地址：<http://chromedriver.googlecode.com/files/chromedriver_win_23.0.1240.0.zip>

将其放到chrome的安装路径下与chrome.exe同目录的文件夹中。

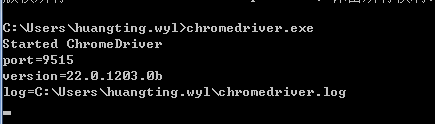
2）设置chrome的环境变量

新建CHROME 将其值设置为：

C:\Users\huangting.wyl\AppData\Local\Google\Chrome\Application 我这是win7系统，xp系统的会不一样，但设置方法一样。然后将;%CHROME%添加到Path变量的后边即可。

3）验证环境变量设置成功和chromedriver.exe文件完好的办法，在命令行里面输入：

chromedriver.exe



我们主要以PC端的三种主要浏览器（IE，火狐，Chrome为例）来说明设置。设置完毕后，那么对三种浏览器的使用就是透明和相同的了。

## 3.2：创建Holmos-webtest程序

### 3.2.1：搭建编程环境

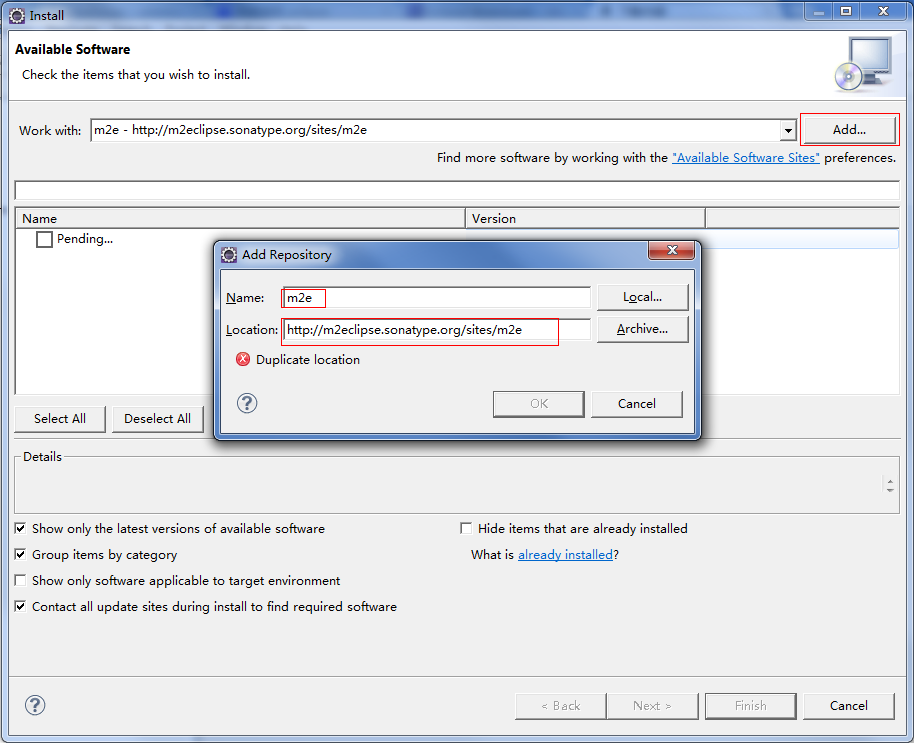
1）下载编程IDE eclipse：地址如下：

<http://mirror.bit.edu.cn/eclipse/technology/epp/downloads/release/juno/R/eclipse-jee-juno-win32.zip>

下载后放到计算机的任意一个位置，打开eclipse

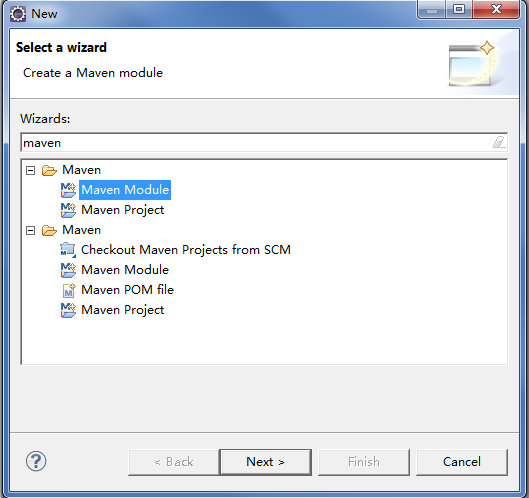
2）安装m2e插件

点击Help->Install new software

点击add：然后在出现的对话框的地方的Name处输入名字m2e(可以任意)；在location的地方输<http://m2eclipse.sonatype.org/sites/m2e>

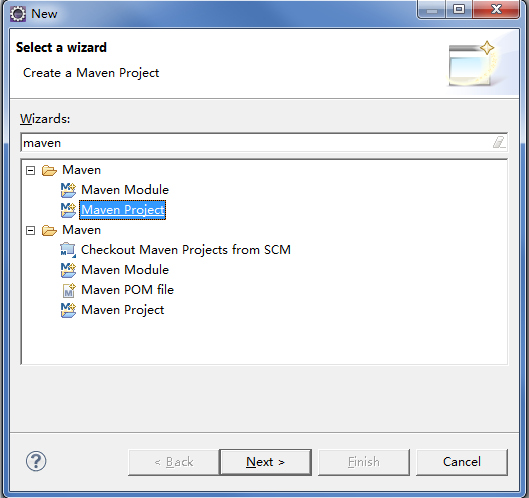
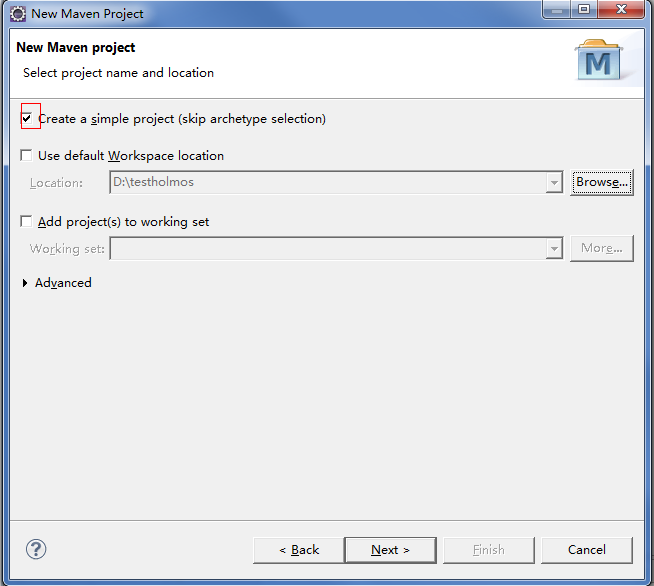
然后等Pending结束后，点击finish按钮，等待安装结束，重启eclipse，安装完成。  
3）验证安装成功

方法一：点击File->New->Other

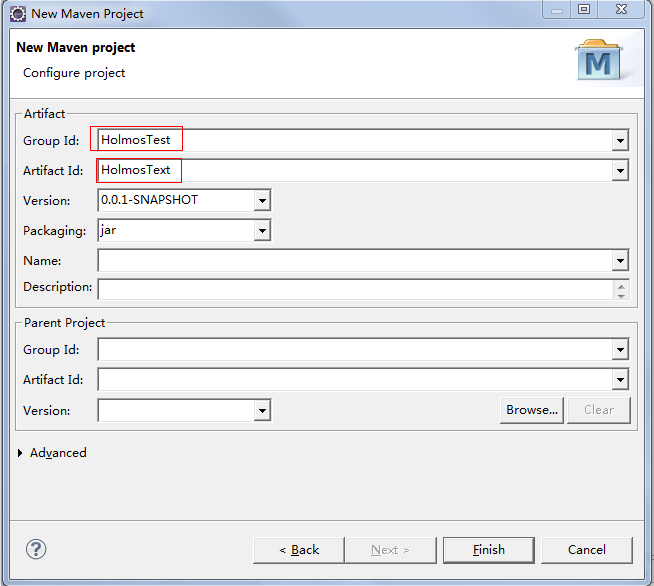
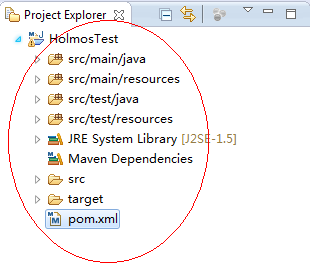
如果出现了左边的新建列表，便安装成功了。  
方法二：点击Window->preferences:输入classpath 然后点击Classpath Variable选项

### 如果左图中出现M2\_REPO即安装成功 3.2.2：get started！！！

下面我们开始写第一个Holmos程序：

->

1 2

 ****

2)点开pom文件，添加maven依赖

<dependency>

<groupId>com.holmos</groupId>

<artifactId>holmos-webtest</artifactId>

<version>1.0.2</version>

</dependency>

如果需要高级校验功能：可以引入如下模块：

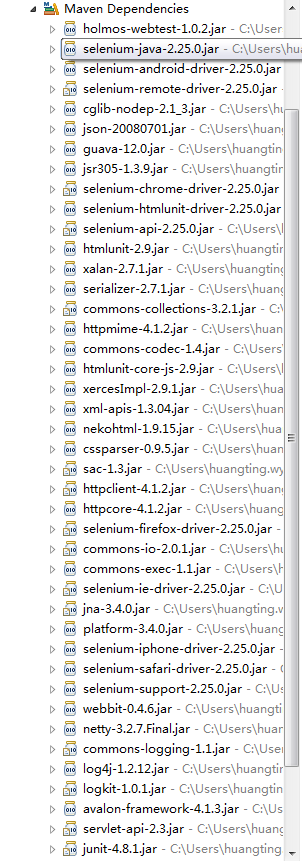
<dependency>

<groupId>com.holmos</groupId>

<artifactId>holmos-reflectAssert</artifactId>

<version>1.0.2</version>

</dependency>

左边即为添加进来的依赖  
依赖添加完毕，我们以Chrome浏览器为例，写第一个Holmos程序，功能是打开一个百度搜索页面，写入搜索值，并点击搜索按钮，查看搜索结果，然后关闭浏览器：

1. Holmos程序由两部分组成，页面库和程序流程，首先我们需要先写页面库，这里只写一个简单的，页面库的知识后边会详细介绍。
2. 将holmos配置文件holmosconf.properties 和log4j.properties文件放在src/main/resoureces目录下面
3. 写BaiduPage页面信息

**package** com.holmos.pagestore;

**import** Holmos.webtest.element.Button;

**import** Holmos.webtest.element.TextField;

**import** Holmos.webtest.struct.Page;

**public** **class** BaiduPage **extends** Page{

**public** BaiduPage(){

**this**.comment="百度首页";//此page的说明，在打印操作日志的时候用到

**this**.init();//page的初始化操作，这个操作一定要加上

}

**public** TextField input=**new** TextField("百度搜索框");//定义百度搜索框

**public** Button search=**new** Button("百度一下");//定义百度一下按钮

{

input.addIDLocator("kw");//添加定位信息(此为id)

search.addIDLocator("su");//添加定位信息(此为id)

}

}

1. 写操作流程

@Test

**public** **void** getStarted(){

BaiduPage baidu=**new** BaiduPage();//定义baidupage对象

HolmosWindow.*openNewWindow*(EngineType.*WebDriverChrome*, "http://www.baidu.com");

//新开百度页面，浏览器选择为webdriver环境的chrome浏览器

baidu.input.setText("hello!holmos!");//在输入框里面输入"hello!holmos!"

baidu.search.click();//点击百度一下按钮，加载搜索结果

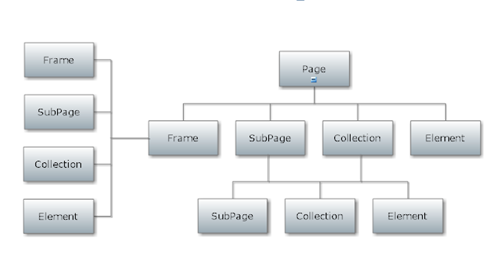
HolmosWindow.*closeAllWindows*();//关闭所有由这个方法打开的页面

}

### 3.2.3：Holmos-webtest 页面库介绍

我们刚才说过，用Holmos-webtest写的webUI测试程序大体分为两个模块，页面库和流程测试用例，经过getStarted的BaiduPage介绍，我们大体了解了页面库是一个什么样子的东西。那么我们下面就详细介绍下。

1. 页面库的结构图如下：

分为如下几个部分：

Page

Frame

SubPage

Collection

各种Element

各种定制组件

1. 页面Page

Page，顾名思义：是对一个网页内容的抽象，指代一个页面，比如说百度首页，google首页，淘宝页面，百度的搜索结果页，也就是说Page对应了一个url下面的页面。页面上可以包含的组织有Frame，SubPage，Collection，Element四种元素

* + 1. 淘宝页面 百度搜索页面  
       
    2. 腾讯首页 新浪首页

等等，也就是说，Holmos里面的Page类就是对实际网页页面的一个抽象，Page类里面主要包含的信息有：

|  |  |
| --- | --- |
| Page.comment | 页面的描述信息，输出操作日志的时候用到 |
| Page.elements | 页面的元素信息，指代该页面上所有定义的Element，为节点类型 |
| Page.subpages | 次级页面信息，指代该页面上所有定义的SubPage对象，为容器类型 |
| Page.frames | iframe或frameSet信息，该页面上所有的定义的Frame，为容器类型 |
| Page.collections | Holmos框架定义的集合信息，用来指代页面上dom结构类似的一组数据，比如百度的搜索结果，比如微博和校内发出的消息列表信息，为容器类型 |
| getBodyText | 获取这个页面<body>标签中的内容 |
| getHtmlText | 获取这个页面<html>标签中的内容 |
| getTitle | 获取这个页面的title |
| deleteAllCokies | 删除所有的由Holmos-webtest产生的cookie信息 |
| deleteCookieByName | 根据cookie的名字来删除指定名字的cookie |
| getAllCookies | 获得所有的由Holmos-webtest产生的cookie信息，各个cookie之间用|隔开 |

* + 1. Page类的定义方法

**package** com.holmos.pagestore;

**import** Holmos.webtest.element.Button;

**import** Holmos.webtest.element.TextField;

**import** Holmos.webtest.struct.Page;

**public** **class** BaiduPage **extends** Page{

--任何一个定义的Page类需要继承Holmos-webtest的Page类，由此继承一些Page类的公共功能，和对页面的结构组织方法

**public** BaiduPage(){

--这个地方是Page的构造函数地方，最好设置comment信息，当然，也可以不设置，不过如果不设置的话，在输出操作日志的时候将无从知晓是哪个页面的操作出问题了，或者成功了

**this**.comment="百度首页";

**this**.init();

--这个是页面的初始化函数，这个函数完成对Page内部Comment的组装，各个元素locator信息的组装，以及对Page里面定义的各种各样的变量信息的获取，此初始化方法不能缺少，要不然Page没有完成初始化，Holmos-webtest模块是没有办法操作元素的

}

--下面是定义页面元素的地方，对于Element类型的元素，直接调用构造方法进行定义，如果是其他（容器）类型的元素，需要先定义容器类型，定义方法下面会给出，然后定义过后，需要给这些元素添加定位信息，定位信息可以是很多种，下面介绍定位信息的时候会一一介绍

**public** TextField input=**new** TextField("百度搜索框");

**public** Button search=**new** Button("百度一下");

{

input.addIDLocator("kw");

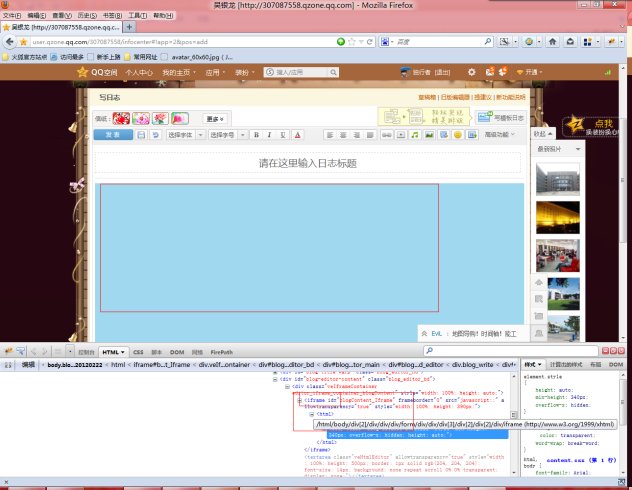
search.addIDLocator("su");

}

}

1. Frame容器

意指一个页面上一个iframe或者一个frameset，根据我们对这两种结构的了解，他也应该包含Page所包含的四种页面结构，Frame是一个递归定义；Frame中可以包含Frame，SubPage，Collection，Element；比如如下所示的一个Frame

Frame的操作方法如下：

a):如果要想操作当前Frame里面的元素，需要先调用这个Frame的select();

b):不管这个Page里面有多少层Frame也不管每层有多少个Frame，用之前只要调用当前Frame的select方法就可以了，框架已经处理好他们之间的转换关系了  
 Frame的用法如下：定义Frame的源代码

**package** com.holmos.pagestore;

**import** Holmos.webtest.element.Button;

**import** Holmos.webtest.element.TextField;

**import** Holmos.webtest.struct.Frame;

**public** **class** ShareFrame **extends** Frame{

--定义Frame的时候一定要继承Frame类，这样的话能够用Frame的功能

**public** ShareFrame(String comment) {

**super**(comment);

}

--下面定义Frame里面的元素，和Page里面定义元素一致

**public** TextField shareContent=**new** TextField("分享内容");

**public** Button shareButton=**new** Button("分享按钮");

{

shareContent.addClassLocator("J\_SnsShareComment");

shareButton.addLinkTextLocator("立即分享");

}

}

定义含有Frame的Page

**package** com.holmos.pagestore;

**import** Holmos.webtest.element.Link;

**import** Holmos.webtest.struct.Page;

**public** **class** MyTaobaoPage **extends** Page{

**public** MyTaobaoPage(){

**this**.comment="淘宝日志页面";

**this**.init();

}

**public** Link toolbarShare=**new** Link("toolbar上的分享给好友连接");

**public** ShareFrame shareFrame=**new** ShareFrame("toolbar分享");

{--在Page里面定义Frame的方法和定义元素的方法一致

shareFrame.addClassLocator("sns-iframe-popup");

toolbarShare.addXpathLocator(".//\*[@id='tstart-plugin-share']/span/a");

}

}

测试含有Frame的页面

**package** com.holmos.casestore;

**import** org.junit.Test;

**import** Holmos.webtest.basetools.HolmosWindow;

**import** com.holmos.pagestore.MyTaobaoPage;

**import** com.holmos.pagestore.TaobaoLoginPage;

**public** **class** TestPageObject {

@Test

**public** **void** testFrame(){

MyTaobaoPage taobao=**new** MyTaobaoPage();

TaobaoLoginPage loginpage=**new** TaobaoLoginPage();

loginpage.login("黄庭", "hello1234");

taobao.toolbarShare.click();

--在用Frame之前，要先进行select然后才能用其中的元素

taobao.shareFrame.select();

taobao.shareFrame.shareContent.setText("不错哦");

taobao.shareFrame.shareButton.click();

HolmosWindow.*closeAllWindows*();

}

}

1. SubPage容器

意指页面上的一个模块结构，代表了一个dom节点及其此节点下的子节点的集合,SubPage是一个递归的定义，其里面可包含SubPage，Collection，Element，但是Holmos框架做了限制，里面不能再有Frame和Page类型

SubPage特点：

a):一定是页面上dom结构的一个节点

b):其中的元素，无论是SubPage，Collection，还是Element都是此SubPage的子节点

SubPage的用法：我们以上图为例来说明：

**package** com.holmos.pagestore;

**import** Holmos.webtest.element.Label;

**import** Holmos.webtest.struct.SubPage;

**public** **class** TaobaoSubPage **extends** SubPage{

--定义SubPage的时候需要继承SubPage类

**public** TaobaoSubPage(String comment) {

--构造函数的形式

**super**(comment);

}

--定义该SubPage里面元素的地方，与Page里面定义的形式一致

**public** Label buy=**new** Label("购物标签");

**public** Label life=**new** Label("生活标签");

**public** Label tool=**new** Label("工具标签");

**public** Label social=**new** Label("社区标签");

**public** Label other=**new** Label("其他标签");

{

--如果是xpath那么需要在这个地方写相对于该SubPage

--的xpath，切要写成相对xpath，前面加’.’

buy.addXpathLocator("./dl[1]/dt");

life.addXpathLocator("./dl[2]/dt");

tool.addXpathLocator("./dl[3]/dt");

social.addXpathLocator("./dl[4]/dt");

other.addXpathLocator("./dl[5]/dt");

}

}

Page 里面定义SubPage的格式

**package** com.holmos.pagestore;

**import** Holmos.webtest.struct.Page;

**public** **class** TaobaoPage **extends** Page{

**public** TaobaoPage(){

**this**.comment="淘宝首页";

**this**.init();

}

--定义SubPage的地方，跟定义Element的形式一样

**public** TaobaoSubPage taobaoService=**new** TaobaoSubPage("淘宝服务");

{

taobaoService.addXpathLocator(".//\*[@id='J\_ProductList']/div");

}

}

测试含有SubPage的用例：

@Test

**public** **void** testSubPage(){

TaobaoPage taobao=**new** TaobaoPage();

HolmosWindow.*openNewWindow*(EngineType.*WebDriverChrome*,"http://www.taobao.com");

--判断该SubPage是否存在

System.*out*.println(taobao.taobaoService.isExist());

--在用SubPage的时候，不必像Frame那样，先Select

taobao.taobaoService.buy.outputText();

taobao.taobaoService.life.outputText();

taobao.taobaoService.other.outputText();

taobao.taobaoService.social.outputText();

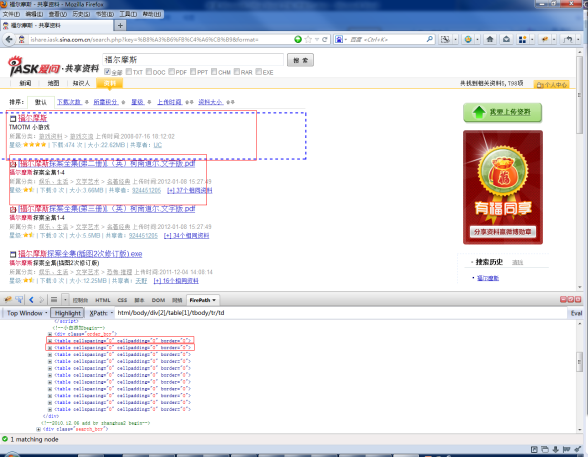
taobao.taobaoService.tool.outputText();

HolmosWindow.*closeAllWindows*();

}

1. Collection容器：

指的是页面上一系列dom结构相似的单元的集合，定义Collection的时候只需要定义Collection的一个单元的结构即可，框架会自动处理其他单元的定位，Collection和SubPage很相似，只是其中的元素的定位具有规律性

Collection特点：

1. 当为xpath定位方式的时候，定位collection的时候需要写到[index]前面，不能加上[index]
2. 在用collection的第index 个元素的时候，先调用collection的.select(index)方法，index从1开始  
   Collection的构造：

**package** com.holmos.pagestore;

**import** Holmos.webtest.element.Label;

**import** Holmos.webtest.element.Link;

**import** Holmos.webtest.struct.Collection;

**public** **class** ResultCollection **extends** Collection{--需要继承Collection

**public** ResultCollection(String comment) {

**super**(comment);

}—定义Collection中的变量

**public** Link resultTitle=**new** Link("结果title");

**public** Label resultExtend=**new** Label("结果附属");

{

resultTitle.addXpathLocator("./tbody/tr/td/div[1]/a/span");

resultExtend.addXpathLocator("/tbody/tr/td/span/span[1]");

}

}

在页面中定义Collection类型对象：

**package** com.holmos.pagestore;

**import** Holmos.webtest.struct.Page;

**public** **class** SinaResultPage **extends** Page{

**public** SinaResultPage(){

**this**.comment="新浪资料搜索结果页面";

**this**.init();

}

--定义Collection类型变量，与一般Element类型一致

**public** ResultCollection result=**new** ResultCollection("新浪搜索结果");

{

result.addXpathLocator("html/body/div[2]/table");

}

}

测试含有Collection类型的用例：

@Test

**public** **void** testCollection(){

SinaResultPage sina=**new** SinaResultPage();

SinaHomePage sinaHome=**new** SinaHomePage();

sinaHome.search("福尔摩斯");

--判断这个集合是否存在

System.*out*.println(sina.result.isExist());

--循环获取前4个元素的信息

**for**(**int** i=1;i<5;i++){

--选中第i个元素

sina.result.select(i);

--判断集合中第i个元素是否存在

System.*out*.println(sina.result.isExist(i));

--获取第i个元素的搜索title

sina.result.resultTitle.outputText();

}

HolmosWindow.*closeAllWindows*();

}

1. Element元素：元素是页面dom树的一个节点，这个节点可以是叶子节点，也可以是非叶子节点，分为如下几种

|  |  |  |  |
| --- | --- | --- | --- |
| 元素类型 | 描述 | 元素类型 | 描述 |
| Element | 普通元素 | TextField | 字符输入框 |
| Button | 按钮 | CheckBox | 复选框 |
| RadioButton | 单选按钮 | Combobox | 下拉框 |
| Image | 图片类型 | Link | 链接类型 |
| ListElement | 叶子节点集合类型 | RichTextField | 富文本框 |
| Table | Table类型 | Custom | 定制类型包 |

具体的用法不在一一熬述，在上面的几个例子中已经给出。

至于多级Frame，多级SubPage，多级Collection，多级Element的用法跟上述描述的单级用法是一致的。

### 3.2.4：Holmos-webtest 多维定位方式介绍

页面组织的定位管理器:我们知道，要在页面上的一个元素上完成一个操作，我们必须定位这个元素，那么Locator类就是完成这个功能的，页面元素的定位方式有很多种：下面是Holmos-webtest支持的定位方式

|  |  |  |  |
| --- | --- | --- | --- |
| 定位方式 | 描述 | 定位方式 | 描述 |
| Id | 根据元素id来定位 | Name | 根据元素name来定位 |
| Class | 根据元素class来定位 | Xpath | 根据元素xpath路径来定位 |
| Css | 根据元素的css路径定位 | link text | 根据元素文本来定位 |
| Tag mame | 根据dom节点的节点名 |  |  |

我们针对不同的定位方式给出用法：

1. id定位：这个是最有效的定位方式，速度快，准确率高，几乎不会失败，并且一般的前端编程原则能够保证确定元素id唯一，在holmos-webtest中，Page-Object除了Page之外，其他类型的Object都是需要定位的，定位方式都是一致的.
2. className classname 是dom节点的一个属性，在页面上，会同时有很多个节点一个含有同样的一个className，那么className一般用来找寻多个节点的场合，一般用来作为Collection和ListElement类型的定位器。
3. tagName 根据dom结构的tag name 来定位元素，一般用于比较特殊的tag定位，比如多用于定位iframe
4. name name是dom节点的一个属性，根据这个属性来定位此元素，一般也是唯一的
5. linkText 根据dom节点上的文本来定位元素，是夹在<>text<>之间的text文本
6. partialLinkText 根据部分文本来定位元素，这个文本的意思和link text的意思一致
7. css 采用css路径来定位元素，css的查找引擎，一般用的是浏览器默认的，但是也有一些浏览器没有默认的css查找引擎，比如IE6,7；FF3.0系列，这个时候采用Sizzle作为css查找引擎
   1. xpath 这是一个高级的定位方式，如果本地浏览器有xpath引擎的话，就用这个默认的引擎，如果没有的话，webdriver自己实现了一套xpath引擎，那么就用这个他们实现的；但是会造成一些没法预期的结果。

| **Driver类型** | **Dom结构的tag和属性** | **属性值** | **有没有本地xpath引擎支持** |
| --- | --- | --- | --- |
| [HtmlUnit Driver](http://seleniumhq.org/docs/03_webdriver.html#htmlunit-driver) | 全是小写 | 跟他们在html出现的一致 | Yes |
| [Internet Explorer Driver](http://seleniumhq.org/docs/03_webdriver.html#internet-explorer-driver) | 全是小写 | 跟他们在html出现的一致 | No |
| [Firefox Driver](http://seleniumhq.org/docs/03_webdriver.html#firefox-driver) | 大小写敏感 | 跟他们在html出现的一致 | Yes |

Locator的使用方法列表：对于Frame，SubPage，Collection，各种Element的用法都是一致的，方法如下，我们定义一个变量 ele，ele可以为以上四种变量里的任何一种。

|  |  |
| --- | --- |
| 定位方式 | 使用方法 |
| Id | ele.addIDLocator(id); |
| Name | ele.addNameLocator(name); |
| Class | ele.addClassLocator(class); |
| Xpath | ele.addXpathLocator(xpath); |
| Css | ele.addCSSLocator(css); |
| Tag name | ele.addTagNameLocator(tagName); |
| Link text | ele.addLinkTextLocator(linkText); |
| Partiallinktext | ele.addPartialLinkTextLocator(partialLinkText); |

### 3.2.5：Holmos-webtest 执行javascript脚本

Holmos-webtest将其做了很好的封装，只需要调用HolmosWindow.sunScript(String js);即可

### 3.2.6：Holmos-webtest 处理js弹出窗口

1） 处理alert弹框 HolmosWindow.dealAlert();

2） 处理confirm弹框HolmosWindow.dealConfirm(isYes);

3） 处理prompt弹框HolmosWindow.dealPrompt(input, isYes)**;**

**具体用法不再一一介绍，java doc文档里面写的很详细。**

### 3.2.7：Holmos-webtest 之样式校验

1） 参考Holmos.webtest.css.HolmosCSSTool类的javadoc文档

### 3.2.8：Holmos-webtest 之多窗口切换

**1） 比如点击了页面的一个组件，然后弹出了一个新的页面**

**2） 多次调用HolmosWindow.openNewWindow（）方法，会出现多窗口**

**调用HolmosWindow.attach()系列方法，具体用法，参考javadoc文档**