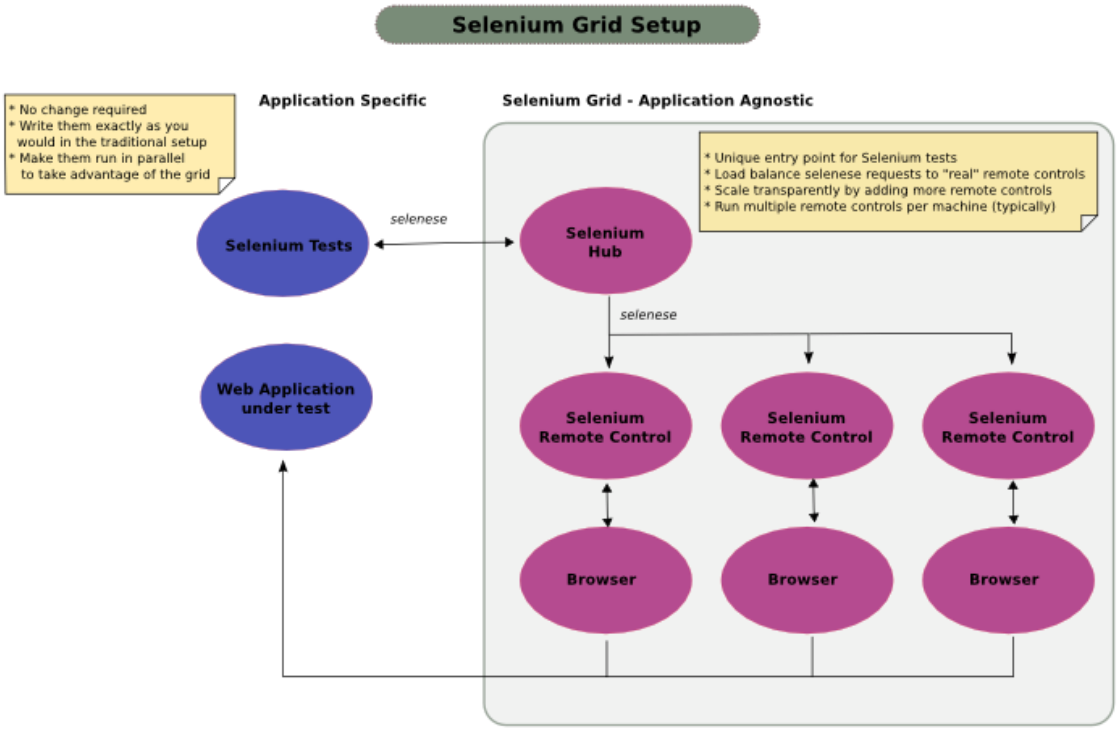
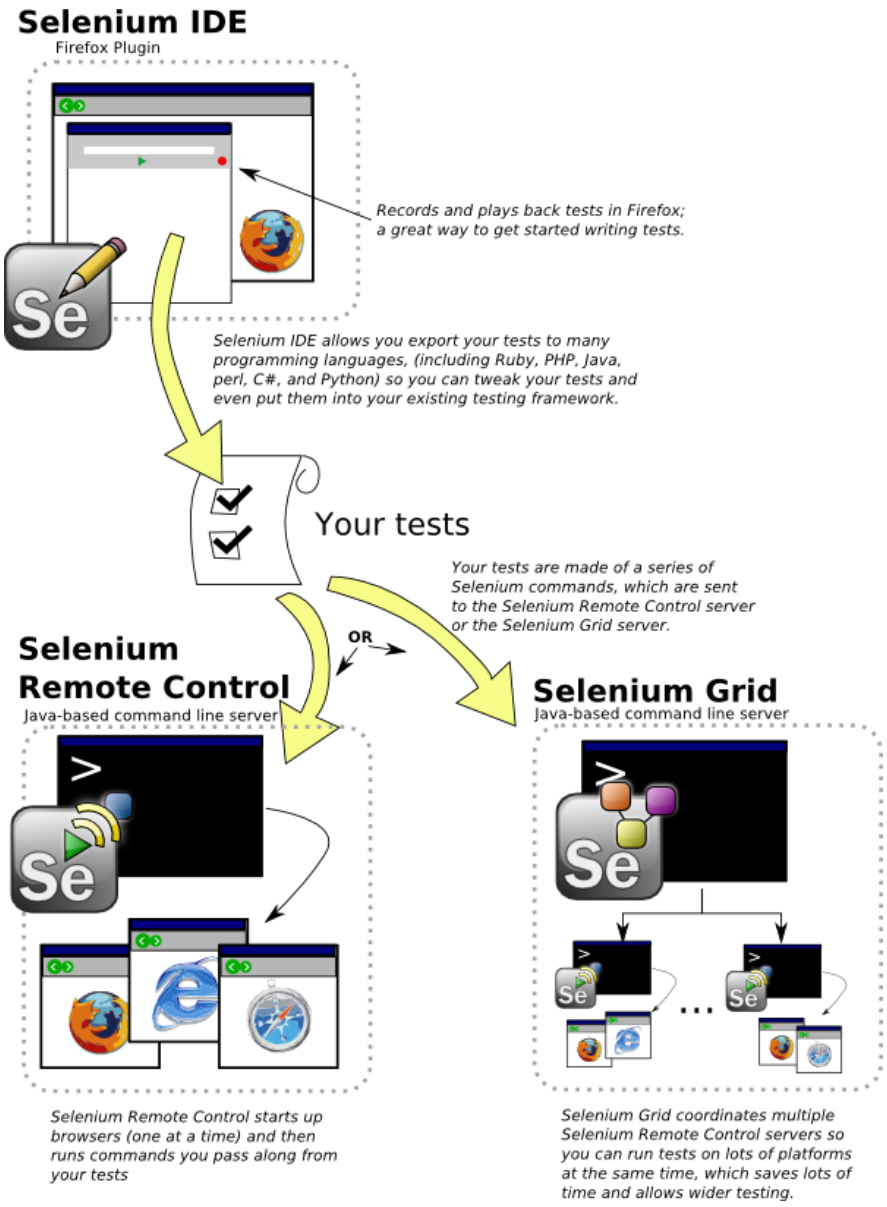
Selenium 简介

Selenium 官方网站：
http://seleniumhq.org/

Selenium 是一款开源的 Web 自动化测试工具，主要由 3 部分组成：
1、Selenium IDE：Firefox 的插件，可用于录制回放脚本
2、Selenium Remote Control（RC）
可以采用各种编程语言编写自动化测试脚本并运行在多种浏览器上。
3、Selenium Grid
可以让 Selenium Remote Control 的自动化测试脚本并行在多个服务器上执行，节省大量运行测试的时间。
http://selenium-grid.seleniumhq.org/how_it_works.html



Selenium 的工作示意图：
http://seleniumhq.org/about/how.html



Selenium 支持多款浏览器的各种版本，包括：Firefox 3、Firefox 2、IE8、IE7、Safari 3、Safari 2、Opera 9、Opera 8 等。并且支持使用多种编程语言开发脚本，包括：C#、Java、Perl、PHP、Python、Ruby 等。
参考：
http://seleniumhq.org/about/platforms.html

selenium 在线文档：
http://seleniumhq.org/docs/

watir 设计理念和 selenium 有很大差异。watir 便于测试工程师快速上手,对 ie 支持非常好;而 selenium 擅长浏览器兼容性。

Selenium 的历史（Brief History of The Selenium Project）

Selenium first came to life in 2004 when Jason Huggins was testing an internal application at ThoughtWorks. Being a smart guy, he realized there were better uses of his time than manually stepping through the same tests with every change he made. He developed a Javascript library that could drive interactions with the page, allowing him to automatically rerun tests against multiple browsers. That library eventually became Selenium Core, which underlies all the functionality of Selenium Remote Control (RC) and Selenium IDE. Selenium RC was ground-breaking because no other product allowed you to control a browser from a language of your choice.

While Selenium was a tremendous tool, it wasn't without its drawbacks. Because of its Javascript based automation engine and the security limitations browsers apply to Javascript, different things became impossible to do. To make things "worst", webapps became more and more powerful over time, using all sorts of special features new browsers provide and making this restrictions more and more painful.

In 2006 a plucky engineer at Google named Simon Stewart started work on a project he called WebDriver. Google had long been a heavy user of Selenium, but testers had to work around the limitations of the product. Simon wanted a testing tool that spoke directly to the browser using the 'native' method for the browser and operating system, thus avoiding the restrictions of a sandboxed Javascript environment. The WebDriver project began with the aim to solve the Selenium' pain-points.

Jump to 2008. The Beijing Olympics mark China's arrival as a global power, massive mortgage default in the United States triggers the worst international recession since the Great Depression, The Dark Knight is viewed by every human (twice), still reeling from the untimely loss of Heath Ledger. But the most important story of that year was the merging of Selenium and WebDriver. Selenium had massive community and commercial support, but WebDriver was clearly the tool of the future. The joining of the two tools provided a common set of features for all users and brought some of the brightest minds in test automation under one roof. Perhaps the best explanation for why WebDriver and Selenium are merging was detailed by Simon Stewart, the creator of WebDriver, in a joint email to the WebDriver and Selenium community on August 6, 2009.

"Why are the projects merging? Partly because webdriver addresses some

shortcomings in selenium (by being able to bypass the JS sandbox, for example. And we've got a gorgeous API), partly because selenium addresses some shortcomings in webdriver (such as supporting a broader range of browsers) and partly because the main selenium contributors and I felt that it was the best way to offer users the best possible framework."

学习大纲

参考《Selenium 1.0 Testing Tools》

1 Selenium IDE 的使用
录制回放脚本、添加断言、添加注释
处理多窗口
处理 AJAX
保存页面数据到变量
调试脚本
创建测试集

2 定位器
Firebug、Firefinder、IE Developer Tools、Google Chrome Developer Tools
通过 ID 定位页面元素
通过 Name 定位页面元素
通过链接文本定位页面元素
通过 JS 访问 DOM 定位页面元素
通过 XPath 定位页面元素
CSS 选择器

3 通过模式匹配验证页面元素
检查文本精确匹配（exact:）
使用 glob 进行匹配
使用正则表达式匹配（regexp:）

4 使用 JavaScript
通过 JavaScript 调用函数返回值给 Selenium
把 JavaScript 结果存储到变量（storeEval）
在 JavaScript 中使用保存的变量（storedVars）
使用 JavaScript 访问浏览器（BrowserBot）
WaitForCondition
fireEvent

5 Selenium 插件扩展
创建自定义函数并关联到 Selenium

在插件扩展中设置 Selenium 变量
在插件扩展中使用定位器
在插件扩展中使用 Browserbot

6 Selenium RC
Selenium RC 配置
用 Selenium RC 运行 Selenium IDE 测试用例
Selenium RC 支持的编程语言
从 Selenium IDE 导出指定语言的测试用例脚本
JAVA + JUnit/TestNG 搭建 Selenium RC 测试框架
页面对象设计模式
与持续集成框架整合

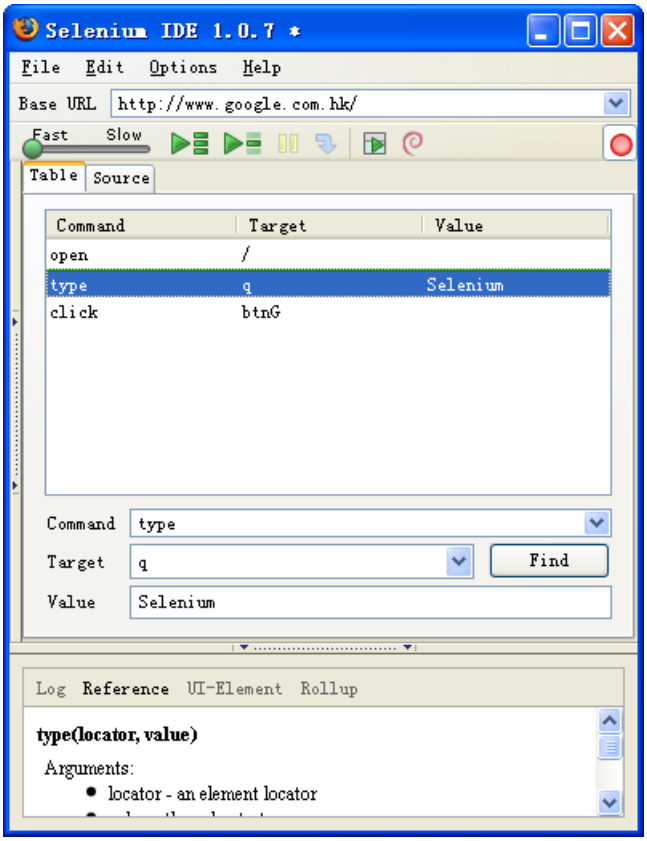
7 高级 Selenium 技巧
读取 Cookie
删除 Cookie
添加一个新的位置策略（addLocationStrategy）
捕获网络消息（captureNetworkTraffic）
截屏（captureScreenshot、captureEntirePageScreenshot）
录制测试视频（castro）

8 Selenium Grid
Selenium Grid 简介
Selenium Grid Hub
往 Hub 添加 Selenium RC
针对 Grid 编写测试用例
并行执行 Selenium 测试用例（TestNG 的 XML 配置文件）

9 Selenium 2.0
Selenium1.0 + WebDriver
把 Selenium1.0 脚本转换成 Selenium2.0 脚本
Selenium2.0 中查找和访问页面元素的脚本编写方式
在 Selenium2.0 中模拟用户操作
在 Selenium2.0 中执行 JavaScript（JavascriptExecutor）

使用 Selenium 开发 Web 自动化测试脚本

1、下载 Firefox 的插件 Selenium IDE（http://seleniumhq.org/download/）后直接打开 Firefox 并把 Selenium IDE 安装文件拖拽进去，重启 Firefox 后，就可以在 Tools->Selenium IDE 菜单项中找到并打开 Selenium IDE 进行脚本的录制，如图所示。



录制的脚本可以保存成 TestCase，还可以导出成指定语言的 Selenium RC 脚本，例如，如果想把刚才录制的测试用例导出成 C#脚本，则可以选择 Selenium IDE 中的菜单“File->Export Test Case As -> C# - Selenium RC”，导出的 C#脚本如下所示：

```
using System;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using NUnit.Framework;
using Selenium;

namespace SeleniumTests
{
    [TestFixture]
    public class GoogleTest
    {
        private ISelenium selenium;
        private StringBuilder verificationErrors;

        [SetUp]
        public void SetupTest()
        {
            selenium = new DefaultSelenium("localhost", 4444, "*chrome",
"http://change-this-to-the-site-you-are-testing/");
            selenium.Start();
```

```
            verificationErrors = new StringBuilder();
        }

        [TearDown]
        public void TeardownTest()
        {
            try
            {
                selenium.Stop();
            }
            catch (Exception)
            {
                // Ignore errors if unable to close the browser
            }
            Assert.AreEqual("", verificationErrors.ToString());
        }

        [Test]
        public void TheGoogleTest()
        {
            selenium.Open("/");
            selenium.Type("q", "Selenium");
            selenium.Click("btnG");
        }
    }
}
```

这些代码可以作为基本的录制脚本，在 VS2005 中进行后续的编辑和修改。

2、下载 Selenium RC。在安装之前需要确保 1.5 版本以上的 JDK 已经正确安装。把 Selenium RC 的文件包解压缩到某个目录，然后在命令行中转到其中的 selenium-server-0.9.2 目录，运行 java -jar selenium-server.jar 来启动 Selenium RC，如图所示。

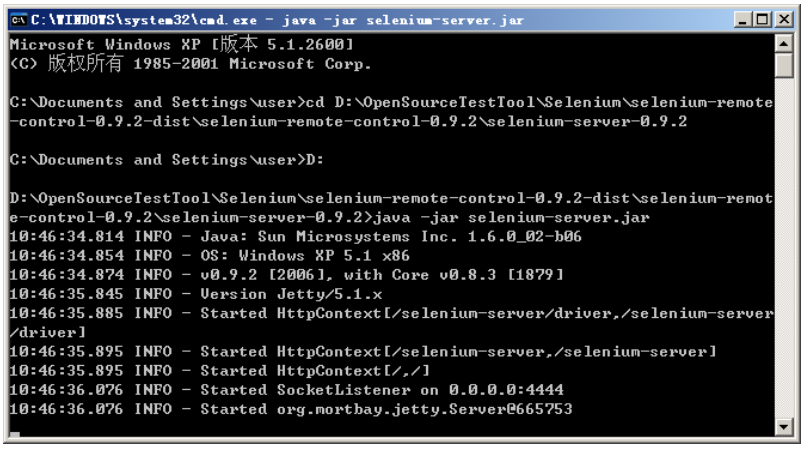


图　启动 Selenium RC

启动后即可以编写测试类来使用 Selenium RC，从而访问和控制各种浏览器进行 Web 页面的功能测试。

技巧：如果碰到启动不成功的情况，则很可能是因为端口号问题，可尝试使用不同的端口号来启动 Selenium RC，默认端口号为 4444，可以使用参数-port，改成其它端口号，如图所示。

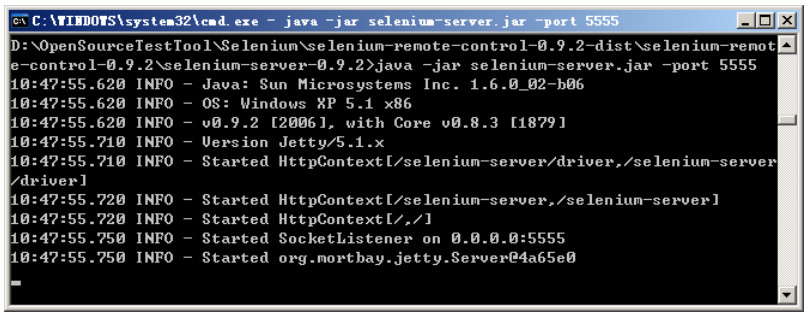


图　指定端口启动 Selenium RC

3、Selenium RC 支持在多种编程语言下编写测试脚本，如果想使用 C#进行脚本的编写，则可以在 Selenium RC 的安装目录下找到 selenium-dotnet-client-driver-0.9.2 目录中的 ThoughtWorks.Selenium.Core.dll，并加到 C#的项目中，然后在 C#代码中添加如下脚本：

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
// 使用 Selenium 的接口
using Selenium;

namespace SeleniumTest1
{
    static class Program
    {
      static void Main()
      {
        // 创建 Selenium 的对象实例
       ISelenium selenium;
        // 启动 firefox 浏览器并导航到 Google 网站
         selenium = new DefaultSelenium("localhost", 5555, @"*firefox C:\Program Files\Mozilla Firefox\firefox.exe", "http://www.google.com.hk/");
         // 开始测试
         selenium.Start();
        // 打开指定的页面
        selenium.Open("http://www.google.com.hk/webhp");
        // 检查页面的 Title 属性是否等于“Google”
        Assert.AreEqual("Google", selenium.GetTitle());
        // 输入字符串
        selenium.Type("q", "Selenium");
        // 检查属性值
        Assert.AreEqual("Selenium OpenQA", selenium.GetValue("q"));
        // 单击按钮
        selenium.Click("btnG");
        // 等待页面加载完成
        selenium.WaitForPageToLoad("5000");
        // 检查指定的文本是否出现
```

```
            Assert.IsTrue(selenium.IsTextPresent("www.openqa.org"));
            // 检查标题
            Assert.AreEqual("Selenium OpenQA - Google Search", selenium.GetTitle());
            // 停止测试
            selenium.Stop();
        }
    }
}
```

说明：该脚本实现了启动 Firefox 浏览器，并加载指定页面进行各种功能测试的效果。

4、基于上面的脚本，仅需要把其中的浏览器加载代码略作修改，即可在另外一种类型的浏览器中执行后续的测试脚本，从而实现浏览器兼容性测试。例如，Firefox 的浏览器加载代码是：

```
// 启动 firefox 浏览器并导航到 Google 网站
selenium = new DefaultSelenium("localhost", 5555, @"*firefox C:\Program Files\Mozilla Firefox\firefox.exe", "http://www.google.com.hk/");
```

改成如下代码则可以支持在 IE 浏览器中进行测试：

```
//启动 iexplore 浏览器并导航到 Google 网站
//selenium = new DefaultSelenium("localhost", 5555, @"*iexplore C:\Program Files\Internet Explorer\iexplore.exe", "http://www.google.com.hk/");
```

说明：类似地，逐一改成其它浏览器类型，则可以完成兼容性测试。可以把这段脚本适当地参数化，例如，从外部配置文件读取浏览器类型、浏览器执行文件路径等信息，便可自动地切换加载浏览器来执行测试。

Selenium IDE

参考：

Selenium IDE 使用、常见问题总结

http://wenku.baidu.com/view/f039670d844769eae009ede9.html###

Selenium IDE 插件

http://seleniumhq.org/download/

Selenium IDE can be extended through its own plugin system. Here are a number of plugins that have been created using it. For more information on how to create your own plugin or have it listed, see the plugin tutorial page.

Please note that these are not supported by the Selenium project and all issues need to be raised with the relevant developers

ScreenShot on Fail

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: February 20, 2012

Version: 1.5

This plugin for Selenium-IDE automatically takes a screen-shot when a command fails while running the test suite. More info.

Favorites

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: April 7, 2011

Version: 1.14

This plugin for Selenium-IDE gives you a way to mark your favorite test suites and open and execute them with a SINGLE click. More info.

Page Coverage

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: January 16, 2012

Version: 1.7

This plugin for Selenium-IDE generates page coverage reports highlighting areas of a web page touched by the Selenese test cases / test suites. More info.

Test Results

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: October 8, 2011

Version: 1.10

This plugin for Selenium-IDE saves the results of the test execution for the test suite and included test cases with a single click. It also allows you to export the test results for individual test cases. More info.

Implicit Wait

Links: Download - Support

Author: Florent Breheret

License: Apache 2

Released: February 20, 2012

Version: 1.0.20

This plugin allows Selenium IDE to automatically wait until the element is found before executing each command using a locator. It is equivalent to the implicit wait function available with Selenium 2 WebDrivers. It avoids having to insert waitForElementPresent before click, type, select..., and provides a command to deal with AJAX processing status. For more information, see the project page.

Perl Formatters

Links: Download - Support

Author: Nate Broderick

License: Apache 2

Released: January 3, 2012

Version: 1.0.3

A plugin for Selenium-IDE that bundles the Perl formatters that used to be included in Se-IDE.

PHP Formatters

Links: Download - Support

Author: Dan Chan

License: Apache 2

Released: October 12, 2011

Version: 1.3.0

A plugin for Selenium-IDE that bundles the PHP formatters that used to be included in Se-IDE.

Play! framework

Links: Download - Support

Author: Manuel Bernhardt

License: Apache 2

Released: June 28, 2011

Version: 1.3

A plugin for Selenium-IDE that adds support for the simplified markup used by the Play! framework.

Highlight Elements

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: August 3, 2010

Version: 1.2

This plugin for Selenium-IDE highlights elements specified in the Selenese commands on the web page as the test case is executed. Once this plugin is installed, a Highlight elements button will be available on the Selenium-IDE main window and Sidebar. More info.

Test Suite Batch Converter

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: January 13, 2011

Version: 1.5

A plugin for Selenium-IDE to convert one or more test suites, including their test cases, from the html format to any other format supported by the Selenium-IDE. More info.

Selenium Expert

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: May 3, 2011

Version: 0.2

This plugin is my attempt to bring the wonderful world of inspections, tips, hints, fixes and refactoring to Selenese! The Selenium Expert goes through your selenium test cases suggesting improvements, giving tips and even lets you apply them with a single click. Recently, Selenium IDE has brought a some improvements that will break some test scripts. The good news is that most of the essential breaking changes introduced in Selenium IDE 1.0.10 have been incorporated into the Selenium Expert. This should make your migration task a few clicks affair. More info.

Power Debugger

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: November 15, 2010

Version: 1.0

This plugin for Selenium-IDE improves debugging and troubleshooting issues with scripts. This plugin adds the pause on fail tool to the Selenium IDE toolbar. When pause on fail is turned on, Selenium IDE would pause the execution of the test case when there is an error or a command failure, allowing you to troubleshoot the problem. More info.

Flex Pilot X

Links: Download - Support

Author: Adam Christian

License: Apache 2

Released: August 28, 2010

Version: 0.8.0

A Selenium IDE plugin for integrating with Flex-Pilot, for Flex automation. More info.

File Logging

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: August 18, 2010

Version: 1.7

This plugin for Selenium-IDE saves log messages to a file in real time at a user selectable log level. Once this plugin is installed, a FileLogging tab will be added to the options dialog and a FileLogging menu will be added to the log pane. More info.

Log Search Bar

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: August 2, 2010

Version: 1.1

A plugin for Selenium-IDE to show a find toolbar in the log pane making it easy to search the displayed log messages. Once this plugin is installed, the log pane will be changed to contain the find toolbar similar to the one found in the Firefox browser. More info.

Stored Variables Viewer

Links: Download - Support

Author: Samit Badle

License: Unknown/Free

Released: September 28, 2010

Version: 1.3

A plugin for Selenium-IDE to view the stored variables within Selenium IDE. Stored variables are created using the store, storeText, storeExpression and other similar store* commands. This plugin allows you to view these variables when the test is running. More info.

Grails Formatters

Links: Download - Support

Author: Robert Fletcher

License: Unknown/Free

Released: June 15, 2010

Version: 1.0

Adds Grails Formatters. More info.

FlexMonkium

Links: Download - Support

Author: Gorrila Logic

License: GPL

Released: January 6, 2011

Version: 4.1.5

A plugin that adds Adobe Flex recording and playback to Selenium via the popular FlexMonkey open source testing framework. With FlexMonkium, Flex recording and playback is seamlessly interleaved with native Selenium recording and playback so you can easily automate the testing of hybrid web applications that mix html and Javascript with Flex. In addition to creating and running Flex tests inside the Selenium IDE, you can export your hybrid testing scripts as JUnit 4 tests that run with Selenium-RC to easily enable automated testing and continuous integration.

Flow Control

Links: Download - Support

Author: Dave Hunt

License: Apache

Released: February 24, 2010

Version: 1.0.3

Incorporates the flow control extension available here and here.

WebDriver Backed Formatters

Links: Download - Support

Author: Dave Hunt

License: Mozilla Public License

Released: July 7, 2011

Version: 1.0.4

Adds WebDriver backed Selenium formatters, which allows users to take advantage of WebDriver without having to modify their tests to use the new API.

Separated Values Formatter

Links: Download - Support

Author: Dave Hunt

License: Apache

Released: February 21, 2011

Version: 1.0.0

Adds a simple reversible formatter to Selenium IDE. Useful for sharing test commands via copy/paste.

User Extensions

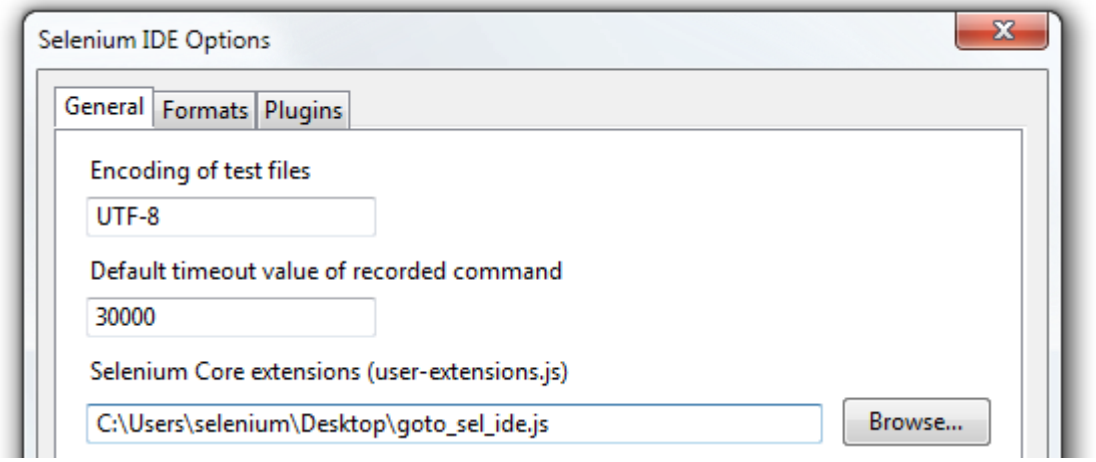
User extensions are JavaScript files that allow one to create his or her own customizations and features to add additional functionality. Often this is in the form of customized commands although this extensibility is not limited to additional commands.

There are a number of useful extensions created by users.

IMPORTANT: THIS SECTION IS OUT OF DATE–WE WILL BE REVISING THIS SOON.

Perhaps the most popular of all Selenium-IDE extensions is one which provides flow control in the form of while loops and primitive conditionals. This extension is the goto_sel_ide.js. For an example of how to use the functionality provided by this extension, look at the page created by its author.

To install this extension, put the pathname to its location on your computer in the **Selenium Core extensions** field of Selenium-IDE's Options=>Options=>General tab.



After selecting the **OK** button, you must close and reopen Selenium-IDE in order for the extensions file to be read. Any change you make to an extension will also require you to close and reopen Selenium-IDE.

Information on writing your own extensions can be found near the bottom of the SeleniumReference document.

Extending Selenium（如何扩展 Selenium）

http://release.seleniumhq.org/selenium-core/1.0/reference.html#extending-selenium

It can be quite simple to extend Selenium, adding your own actions, assertions and locator-strategies. This is done with javascript by adding methods to the Selenium object prototype, and the PageBot object prototype. On startup, Selenium will automatically

look through methods on these prototypes, using name patterns to recognise which ones are actions, assertions and locators.

The following examples try to give an indication of how Selenium can be extended with javascript.

Actions

All *doFoo* methods on the Selenium prototype are added as actions. For each action *foo* there is also an action *fooAndWait* registered. An action method can take up to 2 parameters, which will be passed the second and third column values in the test.

Example: Add a "typeRepeated" action to Selenium, which types the text twice into a text box.

```
        Selenium.prototype.doTypeRepeated = function(locator, text) {
            // All locator-strategies are automatically handled by
"findElement"
            var element = this.page().findElement(locator);

            // Create the text to type
            var valueToType = text + text;

            // Replace the element text with the new text
            this.page().replaceText(element, valueToType);
        };
```

Accessors/Assertions

All *getFoo* and *isFoo* methods on the Selenium prototype are added as accessors (storeFoo). For each accessor there is an *assertFoo*, *verifyFoo* and*waitForFoo* registered. An assert method can take up to 2 parameters, which will be passed the second and third column values in the test. You can also define your own assertions literally as simple "assert" methods, which will also auto-generate "verify" and "waitFor" commands.

Example: Add a *valueRepeated* assertion, that makes sure that the element value consists of the supplied text repeated. The 2 commands that would be available in tests would be *assertValueRepeated* and *verifyValueRepeated*.

```
Selenium.prototype.assertValueRepeated = function(locator, text)
{
        // All locator-strategies are automatically handled by
"findElement"
        var element = this.page().findElement(locator);

        // Create the text to verify
        var expectedValue = text + text;

        // Get the actual element value
        var actualValue = element.value;

        // Make sure the actual value matches the expected
        Assert.matches(expectedValue, actualValue);
    };
```

Automatic availability of storeFoo, assertFoo, assertNotFoo, waitForFoo and waitForNotFoo for every getFoo

All *getFoo* and *isFoo* methods on the Selenium prototype automatically result in the availability of storeFoo, assertFoo, assertNotFoo, verifyFoo, verifyNotFoo, waitForFoo, and waitForNotFoo commands.

Example, if you add a getTextLength() method, the following commands will automatically be available: storeTextLength, assertTextLength, assertNotTextLength, verifyTextLength, verifyNotTextLength, waitForTextLength, and waitForNotTextLength commands.

```
Selenium.prototype.getTextLength = function(locator, text) {
    return this.getText(locator).length;
};
```

Also note that the *assertValueRepeated* method described above could have been implemented using isValueRepeated, with the added benefit of also automatically getting assertNotValueRepeated, storeValueRepeated, waitForValueRepeated and waitForNotValueRepeated.

Locator Strategies

All *locateElementByFoo* methods on the PageBot prototype are added as locator-strategies. A locator strategy takes 2

parameters, the first being the locator string (minus the prefix), and the second being the document in which to search.

Example: Add a "valuerepeated=" locator, that finds the first element a value attribute equal to the the supplied value repeated.

```
        // The "inDocument" is a the document you are searching.
       PageBot.prototype.locateElementByValueRepeated = function(text,
inDocument) {
            // Create the text to search for
            var expectedValue = text + text;

            // Loop through all elements, looking for ones that have
            // a value === our expected value
            var allElements = inDocument.getElementsByTagName("*");
            for (var i = 0; i < allElements.length; i++) {
                var testElement = allElements[i];
                if (testElement.value && testElement.value ===
expectedValue) {
                    return testElement;
                }
            }
            return null;
        };
```

user-extensions.js

By default, Selenium looks for a file called "user-extensions.js", and loads the javascript code found in that file. This file provides a convenient location for adding features to Selenium, without needing to modify the core Selenium sources.

In the standard distibution, this file does not exist. Users can create this file and place their extension code in this common location, removing the need to modify the Selenium sources, and hopefully assisting with the upgrade process.

Contributed User-Extensions（已有的扩展）

http://wiki.openqa.org/display/SEL/Contributed+User-Extensions

Alert

http://wiki.openqa.org/display/SEL/alert

locateByLabelText

http://wiki.openqa.org/display/SEL/locateByLabelText

Locate form elements in selenium tests by matching the text in their associated labels. A label can be associated with a form control by either setting its "for" attribute to the id of the form control or by nesting the form control within the label tag. This locator supports both methods and will support nested input, select, textarea and button tags. It supports some filters (notably index) and also a pseudofilter: within, which restricts the search for matching labels to a subtree of the infoset whose tree is identified by a given xpath expression.

example of use:

For a form containing the HTML:

```
<label for="inputWithId">Input with ID:</label> <input
id="inputWithId" />
<label>Nested Input: <input id="nestedInput" /></label>
```

Text can be typed into the two inputs using the following selenium commands:

type	labelText=Input with ID:	input found by id

type	labelText=Nested Input*	nested input found

下载：

locateByLabelText_1_1

locateByLabelText_1_0

在 Selenium RC 中使用：

1、启动 selenium server 时添加 userExtensions 参数：

java -jar selenium-server.jar -userExtensions user-extensions.js

2、在代码中使用 locateByLabelText：

```
package com.userextensions;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.util.regex.Pattern;

public class locateByLabelText {

    DefaultSelenium selenium;

    @Before
    public void setUp() throws Exception {
        selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"file:///C:/test_locate_by_label.html");
        selenium.start();
    }

    @Test
    public void testLocateByLabelText() throws Exception {
        selenium.open("file:///C:/test_locate_by_label.html");
        selenium.type("labelText=Input with ID:", "hello!");
    }

    @After
    public void tearDown() throws Exception {
        //selenium.stop();
```

```
    }
}
```

locateElementByPartialId

http://wiki.openqa.org/display/SEL/locateElementByPartialId

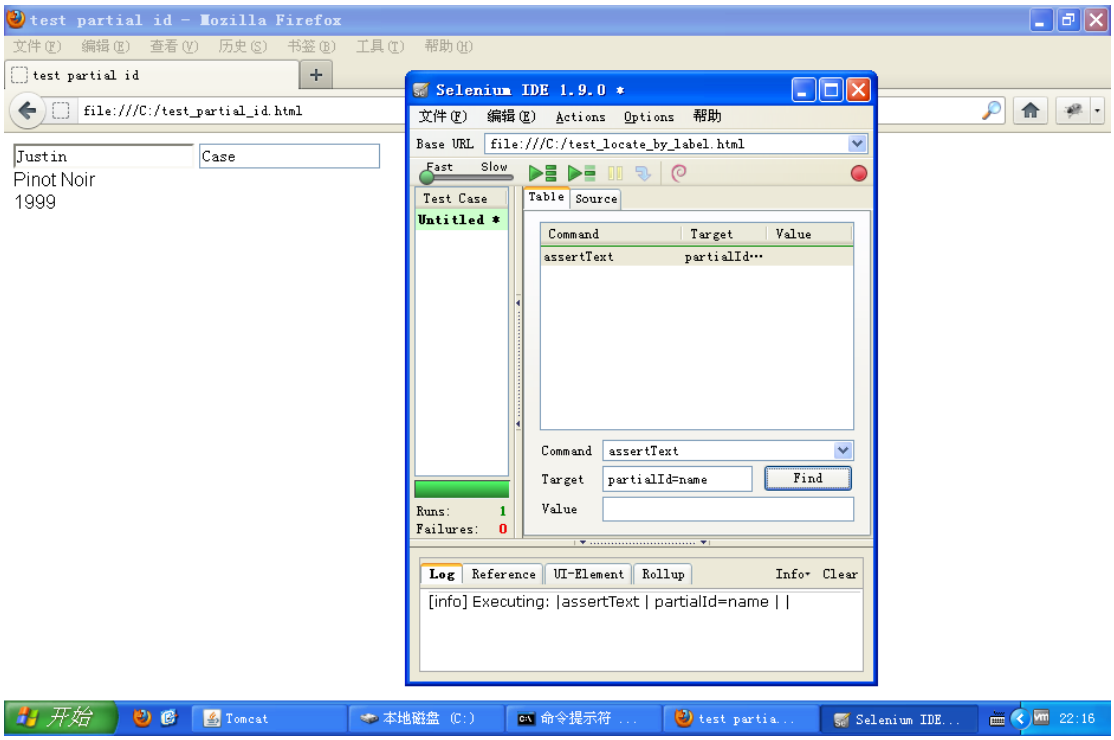
下载：

partialId 1.0.zip

Locate an element on the page by a partial match in ID.

Given an element ... , asserts that it contains the text 'Hello world':

assertText	partialId=main	Hello world



逻辑控制（Sequence of Evaluation and Flow Control）

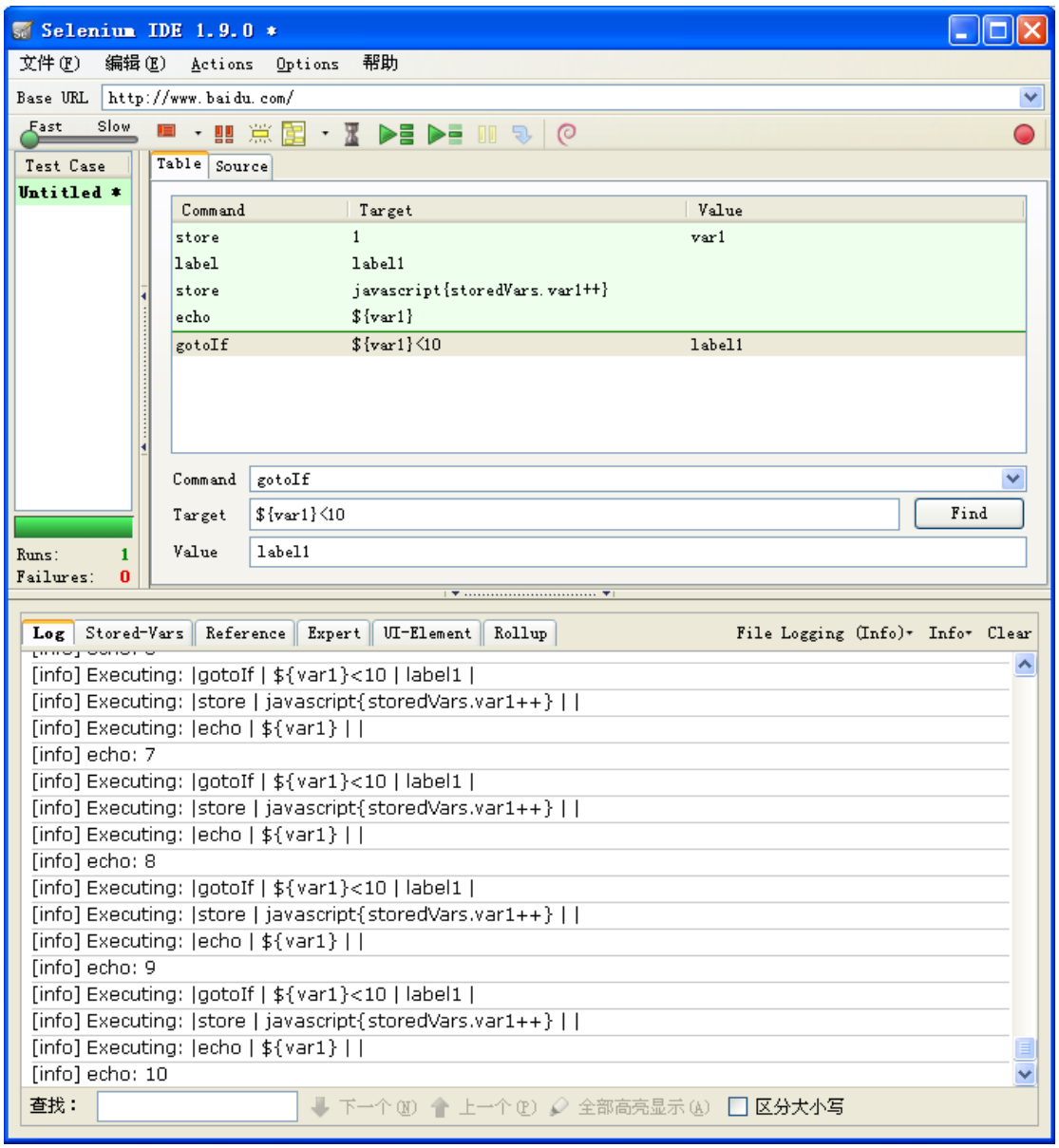
When a script runs, it simply runs in sequence, one command after another.

Selenese, by itself, does not support condition statements (if-else, etc.) or iteration (for, while, etc.). Many useful tests can be conducted without flow control. However, for a functional test of dynamic content, possibly involving multiple pages, programming logic is often needed.

When flow control is needed, there are three options:

1. Run the script using Selenium-RC and a client library such as Java or PHP to utilize the programming language's flow control features.
2. Run a small JavaScript snippet from within the script using the storeEval command.
3. Install the goto_sel_ide.js extension.

Most testers will export the test script into a programming language file that uses the Selenium-RC API (see the Selenium-IDE chapter). However, some organizations prefer to run their scripts from Selenium-IDE whenever possible (for instance, when they have many junior-level people running tests for them, or when programming skills are lacking). If this is your case, consider a JavaScript snippet or the goto_sel_ide.js extension.



参考:
http://wiki.openqa.org/display/SEL/flowControl

Examples of implemented commands:

gotolabel / label

Unconditional jump

gotolabel testlabel1

....

label testlabel1

gotolabel parameters:

1. a label to jump to

gotoIf / label

Conditional jump

gotoIf /corrrect caption/i.test(storedVars.pagetext) testlabel1
....
label testlabel1

gotoIf parameters:

1. javascript expression
2. label to jump to if javascript expression evaluated as true

(obsolete) assertNoFailureOnNextAndGoto / assertNoErrorOnNextAndGoto / label

this command is dropped in extension for Selenium Core v.0.8+

Jump if following command passed

assertNoFailureOnNextAndGoto testlabel1
verifyText element correct page
open ./pageneeded.html
label testlabel1

while/endWhile

Cycle

store 0 loop2
while storedVars.loop2 < 2
store javascript{storedVars.loop2++}
endWhile

while parameters:

1. javascript expression.

FAQ

Q: Can the flowControl extension be used in Selenium IDE?

A: according to the following thread it can:
http://clearspace.openqa.org/message/20977#20977

however, I do not use S-IDE and therefore will recommend to ask guys on corresponding (S-IDE) forum

Q: I am using this extension along with 'include' extension and getting error like "'While' is not found". What do I do?

A: add following code at the end of IncludeCommand.prototype.doInclude function. (as per http://forums.openqa.org/thread.jspa?threadID=10163)

```
if( htmlTestRunner.currentTest.initialiseLabels )
{ htmlTestRunner.currentTest.initialiseLabels(); }
```

NOTE:This answer refers to the IncludeCommand.prototype.doInclude function, which did not work for me. Farther down in this forum there's a reference by Jerry Qianto putting this code in the Selenium.prototype.doInclude function, which did work for me. In addtion, since I was using IDE, changed the code to:

```
if( this.initialiseLabels )
{ this.initialiseLabels(); }
```

You will also need to add this at the end of doEnd$Template$ so that goto's after includes to work.

在 Selenium IDE 中添加程序控制功能

http://www.cnblogs.com/bluesfeng/archive/2011/05/20/2052439.html

下面介绍如何在 Selenium IDE 中添加程序控制功能。

1）下载 Selenium 插件 （sideflow.js）：

如果不想下载，直接把下面的代码保存到本机也可。

?

```
var gotoLabels= {};
```

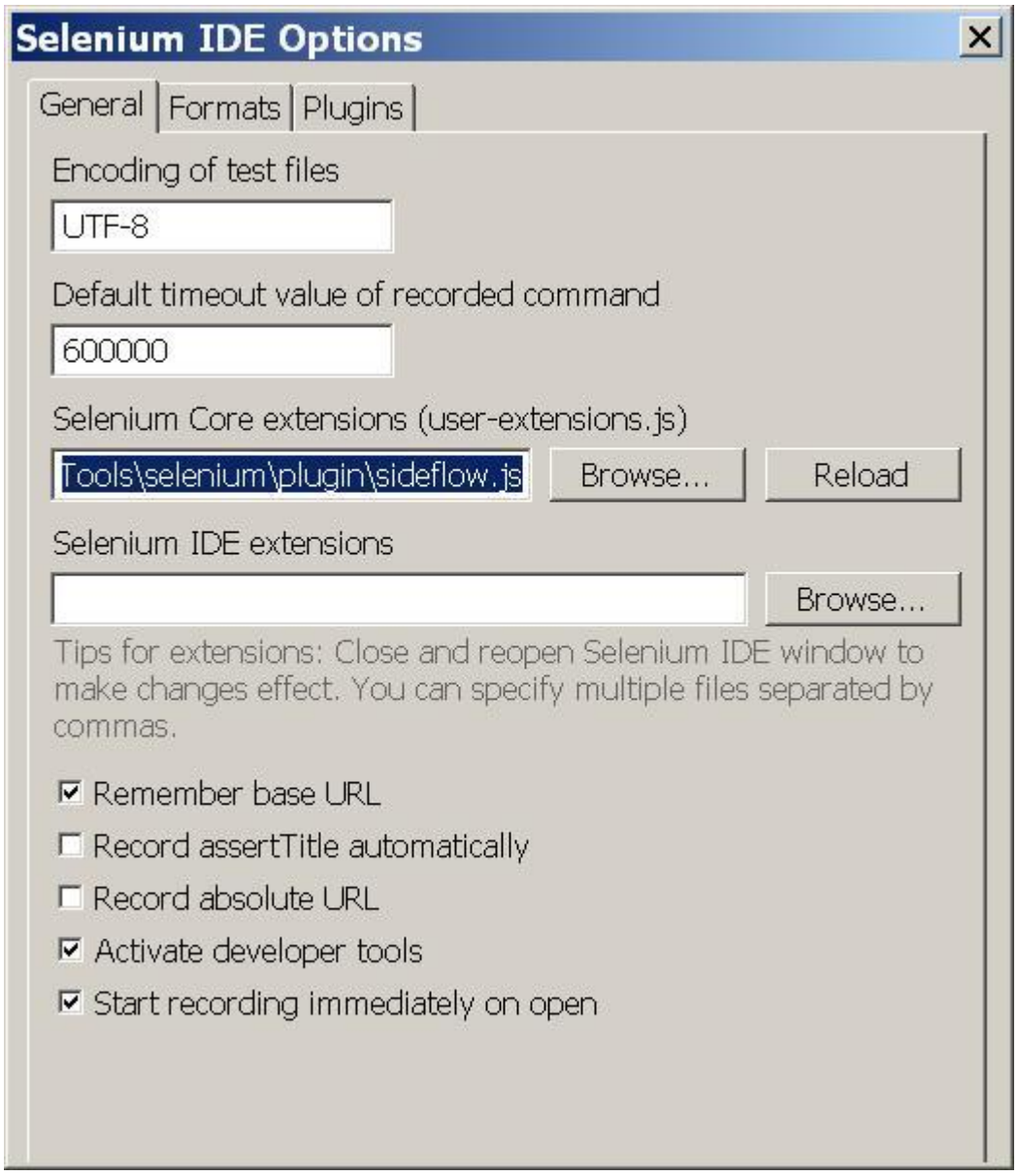
```
var whileLabels = {};

// overload the original Selenium reset function
Selenium.prototype.reset = function() {
        // reset the labels
        this.initialiseLabels();
        // proceed with original reset code
        this.defaultTimeout = Selenium.DEFAULT_TIMEOUT;
        this.browserbot.selectWindow("null");
        this.browserbot.resetPopups();
}

Selenium.prototype.initialiseLabels = function()
{
        gotoLabels = {};
        whileLabels = { ends: {}, whiles: {} };
        var command_rows = [];
        var numCommands = testCase.commands.length;
        for (var i = 0; i < numCommands; ++i) {
                var x = testCase.commands[i];
                command_rows.push(x);
        }
        var cycles = [];
        for( var i = 0; i < command_rows.length; i++ ) {
                if (command_rows[i].type == 'command')
                switch( command_rows[i].command.toLowerCase() ) {
                        case "label":
                                gotoLabels[ command_rows[i].target ] = i;
                                break;
                        case "while":
                        case "endwhile":
                                cycles.push( [command_rows[i].command.toLowerCase(),
                                break;
                }
        }
        var i = 0;
        while( cycles.length ) {
                if( i >= cycles.length ) {
                        throw new Error( "non-matching while/endWhile found");
                }
                switch( cycles[i][0] ) {
                        case "while":
                                if( ( i+1 < cycles.length ) && ( "endwhile" == cycles
                                        // pair found
```

```
                                        whileLabels.ends[ cycles[i+1][1] ] = cycles[
                                        whileLabels.whiles[ cycles[i][1] ] = cycles[
                                        cycles.splice( i, 2 );
                                        i = 0;
                                } else ++i;
                                break;
                        case "endwhile":
                                ++i;
                                break;
                }
        }
}

Selenium.prototype.continueFromRow = function( row_num )
{
        if(row_num == undefined || row_num == null || row_num < 0) {
                throw new Error( "Invalid row_num specified.");
        }
        testCase.debugContext.debugIndex = row_num;
}

// do nothing. simple label
Selenium.prototype.doLabel = function(){};

Selenium.prototype.doGotolabel = function( label )
{
        if( undefined == gotoLabels[label] ) {
                throw new Error( "Specified label '" + label + "' is not found.");
        }
        this.continueFromRow( gotoLabels[ label ] );
};

Selenium.prototype.doGoto = Selenium.prototype.doGotolabel;

Selenium.prototype.doGotoIf = function( condition, label )
{
        if( eval(condition) ) this.doGotolabel( label );
}

Selenium.prototype.doWhile = function( condition )
{
        if( !eval(condition) ) {
                var last_row = testCase.debugContext.debugIndex;
                var end_while_row = whileLabels.whiles[ last_row ];
```

```
                if( undefined == end_while_row ) throw new Error( "Corresponding 'endW
                this.continueFromRow( end_while_row );
        }
}

Selenium.prototype.doEndWhile = function()
{
        var last_row = testCase.debugContext.debugIndex;
        var while_row = whileLabels.ends[ last_row ] - 1;
        if( undefined == while_row ) throw new Error( "Corresponding 'While' is not fo
        this.continueFromRow( while_row );
}
```

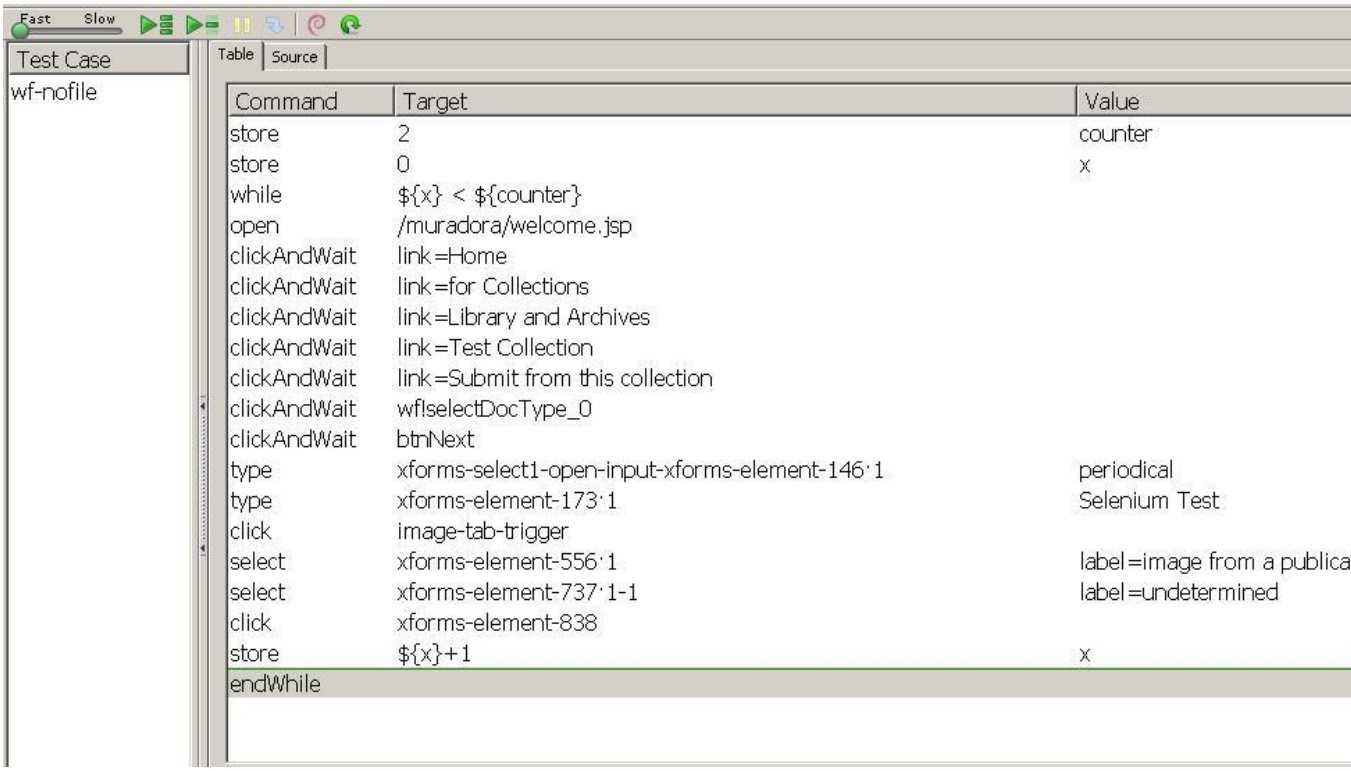
下载地址： https://github.com/darrenderidder/sideflow

2）在 Selenium 中配置插件。如下图所示，添加 sideflow.js 到 "Selenium Core extensions (user-extensions.js)" 文本框中，重新启动 Selenium IDE。



3） 这样就可以在 Selenium 中使用流程控制了

，例如



使用变量（Store Commands and Selenium Variables）

You can use Selenium variables to store constants at the beginning of a script. Also, when combined with a data-driven test design (discussed in a later section), Selenium variables can be used to store values passed to your test program from the command-line, from another program, or from a file.

The plain *store* command is the most basic of the many store commands and can be used to simply store a constant value in a selenium variable. It takes two parameters, the text value to be stored and a selenium variable. Use the standard variable naming conventions of only alphanumeric characters when choosing a name for your variable.

Command	**Target**	**Value**
store	paul@mysite.org	userName

Later in your script, you'll want to use the stored value of your variable. To access the value of a variable, enclose the variable in curly brackets ({}) and precede it with a dollar sign like this.

Command	Target	Value
verifyText	//div/p	${userName}

A common use of variables is for storing input for an input field.

Command	Target	Value
type	id=login	${userName}

Selenium variables can be used in either the first or second parameter and are interpreted by Selenium prior to any other operations performed by the command. A Selenium variable may also be used within a locator expression.

An equivalent store command exists for each verify and assert command. Here are a couple more commonly used store commands.

storeElementPresent

This corresponds to verifyElementPresent. It simply stores a boolean value–"true" or "false"–depending on whether the UI element is found.

storeText

StoreText corresponds to verifyText. It uses a locater to identify specific page text. The text, if found, is stored in the variable. StoreText can be used to extract text from the page being tested.

storeEval

This command takes a script as its first parameter. Embedding JavaScript within Selenese is covered in the next section. StoreEval allows the test to store the result of running the script in a variable.

使用 JavaScript（JavaScript and Selenese Parameters）

JavaScript can be used with two types of Selenese parameters: script and non-script (usually expressions). In most cases, you'll want to access and/or manipulate a test case variable inside the JavaScript snippet used as a Selenese parameter. All variables created in your test case are stored in a JavaScript *associative array*. An associative array has string indexes rather than sequential numeric indexes. The associative array containing your test case's variables is named**storedVars**. Whenever you wish to access or manipulate a

variable within a JavaScript snippet, you must refer to it as **storedVars[‘yourVariableName’]**.

JavaScript Usage with Script Parameters

Several Selenese commands specify a **script** parameter including **assertEval**, **verifyEval**,**storeEval**, and **waitForEval**. These parameters require no special syntax. A Selenium-IDE user would simply place a snippet of JavaScript code into the appropriate field, normally the**Target** field (because a **script** parameter is normally the first or only parameter).

The example below illustrates how a JavaScript snippet can be used to perform a simple numerical calculation:

Command	**Target**	**Value**
store	10	hits
storeXpathCount	//blockquote	blockquotes
storeEval	storedVars[‘hits’]-storedVars[‘blockquotes’]	paragraphs

This next example illustrates how a JavaScript snippet can include calls to methods, in this case the JavaScript String object’s `toUpperCase` method and `toLowerCase` method.

Command	**Target**	**Value**
store	Edith Wharton	name
storeEval	storedVars[‘name’].toUpperCase()	uc
storeEval	storedVars[‘name’].toLowerCase()	lc

JavaScript Usage with Non-Script Parameters

JavaScript can also be used to help generate values for parameters, even when the parameter is not specified to be of type **script**. However, in this case, special syntax is required–the JavaScript snippet must be enclosed inside curly braces and preceded by the label `javascript`, as in `javascript {*yourCodeHere*}`. Below is an example in which the `type`command’s second parameter `value` is generated via JavaScript code using this special syntax:

Command	**Target**	**Value**
store	league of nations	searchString
type	q	javascript{storedVars[‘searchString’].toUpperCase()}

弹出窗口的处理（Alerts, Popups, and Multiple Windows）

Suppose that you are testing a page that looks like this.

```
<!DOCTYPE HTML>
<html>
<head>
  <script type="text/javascript">
    function output(resultText){

document.getElementById('output').childNodes[0].nodeValue=resultText;
    }

    function show_confirm(){
      var confirmation=confirm("Chose an option.");
      if (confirmation==true){
        output("Confirmed.");
      }
      else{
        output("Rejected!");
      }
    }

    function show_alert(){
      alert("I'm blocking!");
      output("Alert is gone.");
    }
    function show_prompt(){
      var response = prompt("What's the best web QA tool?","Selenium");
      output(response);
    }
    function open_window(windowName){
      window.open("newWindow.html",windowName);
    }
    </script>
</head>
<body>

  <input type="button" id="btnConfirm" onclick="show_confirm()"
value="Show confirm box" />
  <input type="button" id="btnAlert" onclick="show_alert()" value="Show
alert" />
```

```
39   <input type="button" id="btnPrompt" onclick="show_prompt()"
40 value="Show prompt" />
41   <a href="newWindow.html" id="lnkNewWindow" target="_blank">New Window
42 Link</a>
43   <input type="button" id="btnNewNamelessWindow"
44 onclick="open_window()" value="Open Nameless Window" />
45   <input type="button" id="btnNewNamedWindow"
   onclick="open_window('Mike')" value="Open Named Window" />

   <br />
   <span id="output">
   </span>
 </body>
 </html>
```

The user must respond to alert/confirm boxes, as well as moving focus to newly opened popup windows. Fortunately, Selenium can cover JavaScript pop-ups.

But before we begin covering alerts/confirms/prompts in individual detail, it is helpful to understand the commonality between them. Alerts, confirmation boxes and prompts all have variations of the following

Command	**Description**
assertFoo(*pattern*)	throws error if *pattern* doesn't match the text of the pop-up
assertFooPresent	throws error if pop-up is not available
assertFooNotPresent	throws error if any pop-up is present
storeFoo(*variable*)	stores the text of the pop-up in a variable
storeFooPresent(*variable*)	stores the text of the pop-up in a variable and returns true or false

When running under Selenium, JavaScript pop-ups will not appear. This is because the function calls are actually being overridden at runtime by Selenium's own JavaScript. However, just because you cannot see the pop-up doesn't mean you don't have to deal with it. To handle a pop-up, you must call its `assertFoo(pattern)` function. If you fail to assert the presence of a pop-up your next command will be blocked and you will get an error similar to the following `[error]Error: There was an unexpected Confirmation! [Chose an option.]`

Alerts

Let's start with alerts because they are the simplest pop-up to handle. To begin, open the HTML sample above in a browser and click on the "Show alert" button. You'll notice that after you close the alert the text "Alert is gone." is displayed on the page. Now run through the same steps with Selenium IDE recording, and verify the text is added after you close the alert. Your test will look something like this:

Command	Target	Value
open	/	
click	btnAlert	
assertAlert	I'm blocking!	
verifyTextPresent	Alert is gone.	

You may be thinking "That's odd, I never tried to assert that alert." But this is Selenium-IDE handling and closing the alert for you. If you remove that step and replay the test you will get the following error `[error] Error: There was an unexpected Alert! [I'm blocking!]`. You must include an assertion of the alert to acknowledge its presence.

If you just want to assert that an alert is present but either don't know or don't care what text it contains, you can use `assertAlertPresent`. This will return true or false, with false halting the test.

Confirmations

Confirmations behave in much the same way as alerts, with `assertConfirmation` and `assertConfirmationPresent` offering the same characteristics as their alert counterparts. However, by default Selenium will select OK when a confirmation pops up. Try recording clicking on the "Show confirm box" button in the sample page, but click on the "Cancel" button in the popup, then assert the output text. Your test may look something like this:

Command	Target	Value
open	/	
click	btnConfirm	
chooseCancelOnNextConfirmation		
assertConfirmation	Choose an option.	
verifyTextPresent	Rejected	

The `chooseCancelOnNextConfirmation` function tells Selenium that all following confirmation should return false. It can be reset by calling chooseOkOnNextConfirmation.

You may notice that you cannot replay this test, because Selenium complains that there is an unhandled confirmation. This is because the order of events Selenium-IDE records causes the click and chooseCancelOnNextConfirmation to be put in the wrong order (it makes sense if you think about it, Selenium can't know that you're cancelling before you open a confirmation) Simply switch these two commands and your test will run fine.

```
<!DOCTYPE HTML>
<html>
<head>
  <script type="text/javascript">
    function output(resultText){

document.getElementById('output').childNodes[0].nodeValue=resultText;
    }

    function show_confirm(){
      var confirmation=confirm("Chose an option.");
      if (confirmation==true){
        output("Confirmed.");
      }
      else{
        output("Rejected!");
      }
    }

    function show_alert(){
      alert("I'm blocking!");
      output("Alert is gone.");
    }
    function show_prompt(){
      var response = prompt("What's the best web QA tool?","Selenium");
      output(response);
    }
    function open_window(windowName){
      window.open("newWindow.html",windowName);
    }
    </script>
```

```
</head>
<body>

  <input type="button" id="btnConfirm" onclick="show_confirm()"
value="Show confirm box" />
  <input type="button" id="btnAlert" onclick="show_alert()" value="Show
alert" />
  <input type="button" id="btnPrompt" onclick="show_prompt()"
value="Show prompt" />
  <a href="newWindow.html" id="lnkNewWindow" target="_blank">New Window
Link</a>
  <input type="button" id="btnNewNamelessWindow"
onclick="open_window()" value="Open Nameless Window" />
  <input type="button" id="btnNewNamedWindow"
onclick="open_window('Mike')" value="Open Named Window" />

  <br />
  <span id="output">
  </span>
</body>
</html>
```

调试技巧（Debugging）

Debugging means finding and fixing errors in your test case. This is a normal part of test case development.

We won't teach debugging here as most new users to Selenium will already have some basic experience with debugging. If this is new to you, we recommend you ask one of the developers in your organization.

Breakpoints and Startpoints

The Sel-IDE supports the setting of breakpoints and the ability to start and stop the running of a test case, from any point within the test case. That is, one can run up to a specific command in the middle of the test case and inspect how the test case behaves at that point. To do this, set a breakpoint on the command just before the one to be examined.

To set a breakpoint, select a command, right-click, and from the context menu select *Toggle Breakpoint*. Then click the Run button to run your test case from the beginning up to the breakpoint.

It is also sometimes useful to run a test case from somewhere in the middle to the end of the test case or up to a breakpoint that follows the starting point. For example, suppose your test case first logs into the website and then performs a series of tests and you are trying to debug one of those tests. However, you only need to login once, but you need to keep rerunning your tests as you are developing them. You can login once, then run your test case from a startpoint placed after the login portion of your test case. That will prevent you from having to manually logout each time you rerun your test case.

To set a startpoint, select a command, right-click, and from the context menu select*Set/Clear Start Point*. Then click the Run button to execute the test case beginning at that startpoint.

Stepping Through a Testcase

To execute a test case one command at a time ("step through" it), follow these steps:

1. Start the test case running with the Run button from the toolbar.

2. Immediately pause the executing test case with the Pause button.

3. Repeatedly select the Step button.

Find Button

The Find button is used to see which UI element on the currently displayed webpage (in the browser) is used in the currently selected Selenium command. This is useful when building a locator for a command's first parameter (see the section on *locators* in the Selenium Commands chapter). It can be used with any command that identifies a UI element on a webpage, i.e. *click*, *clickAndWait*, *type*, and certain *assert* and *verify* commands, among others.

From Table view, select any command that has a locator parameter. Click the Find button. Now look on the webpage: There should be a bright green rectangle enclosing the element specified by the locator parameter.

Page Source for Debugging

Often, when debugging a test case, you simply must look at the page source (the HTML for the webpage you're trying to test) to determine a problem. Firefox makes this easy. Simply right-click the webpage and select 'View->Page Source. The HTML opens in a separate

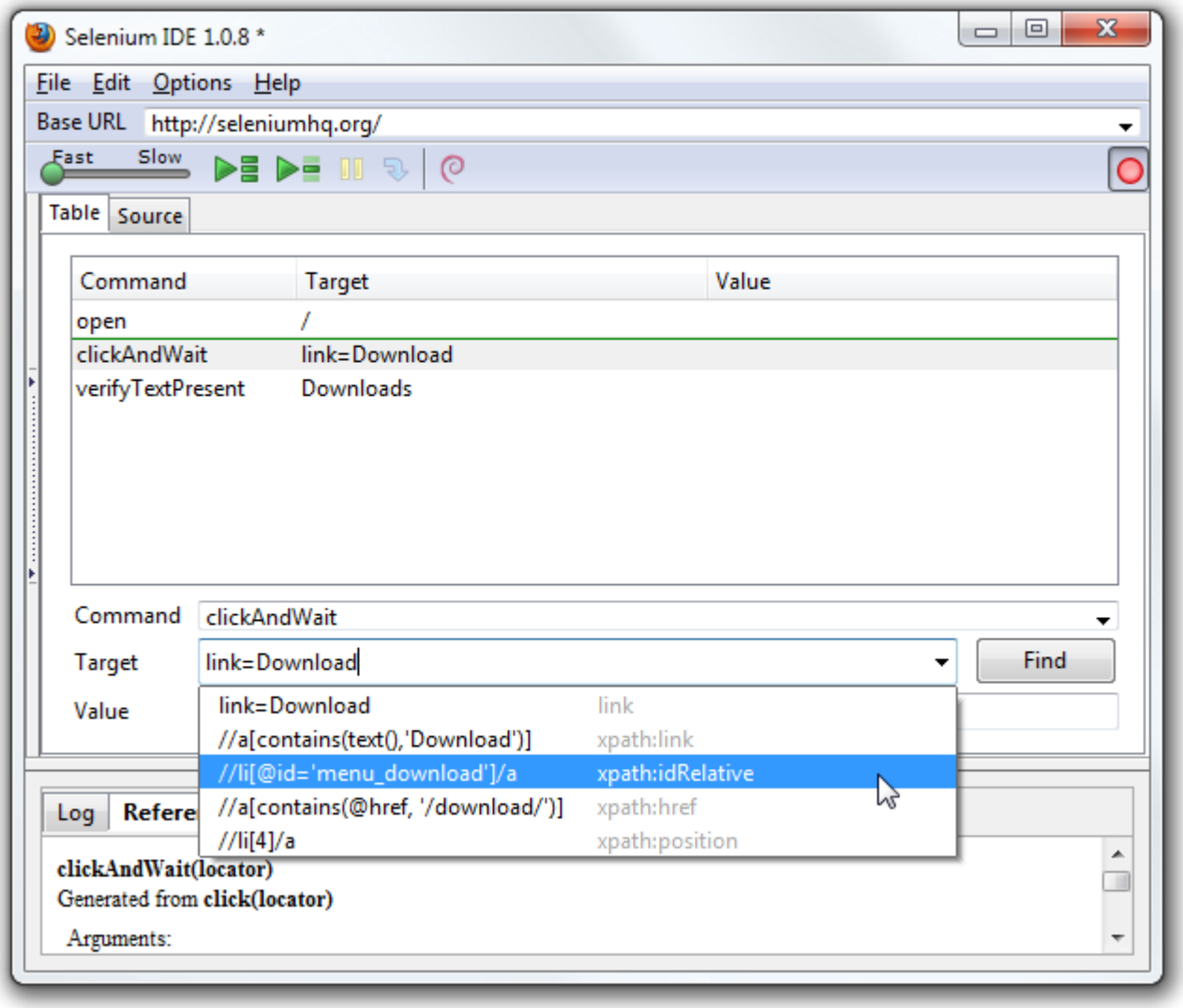
window. Use its Search feature (Edit=>Find) to search for a keyword to find the HTML for the UI element you're trying to test.

Alternatively, select just that portion of the webpage for which you want to see the source. Then right-click the webpage and select View Selection Source. In this case, the separate HTML window will contain just a small amount of source, with highlighting on the portion representing your selection.

Locator Assistance

Whenever Selenium-IDE records a locator-type argument, it stores additional information which allows the user to view other possible locator-type arguments that could be used instead. This feature can be very useful for learning more about locators, and is often needed to help one build a different type of locator than the type that was recorded.

This locator assistance is presented on the Selenium-IDE window as a drop-down list accessible at the right end of the Target field (only when the Target field contains a recorded locator-type argument). Below is a snapshot showing the contents of this drop-down for one command. Note that the first column of the drop-down provides alternative locators, whereas the second column indicates the type of each alternative.



辅助工具

Firebug

Xpath Checker

Xpather

Firefinder
https://getfirebug.com/wiki/index.php/Firebug_Extensions

对象识别

http://seleniumhq.org/docs/02_selenium_ide.html

Locating Elements

For many Selenium commands, a target is required. This target identifies an element in the content of the web application, and consists of the location strategy followed by the location in the format `locatorType=location`. The locator type can be omitted in many cases. The various locator types are explained below with examples for each.

Locating by Identifier

This is probably the most common method of locating elements and is the catch-all default when no recognized locator type is used. With this strategy, the first element with the id attribute value matching the location will be used. If no element has a matching id attribute, then the first element with a name attribute matching the location will be used.

For instance, your page source could have id and name attributes as follows:

```
1 <html>
2  <body>
3   <form id="loginForm">
4    <input name="username" type="text" />
5    <input name="password" type="password" />
6    <input name="continue" type="submit" value="Login" />
7   </form>
8  </body>
9 <html>
```

The following locator strategies would return the elements from the HTML snippet above indicated by line number:

- identifier=loginForm (3)
- identifier=password (5)
- identifier=continue (6)
- continue (6)

Since the `identifier` type of locator is the default, the `identifier=` in the first three examples above is not necessary.

Locating by Id

This type of locator is more limited than the identifier locator type, but also more explicit. Use this when you know an element's id attribute.

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
 </body>
<html>
```

- id=loginForm (3)

Locating by Name

The name locator type will locate the first element with a matching name attribute. If multiple elements have the same value for a name attribute, then you can use filters to further refine your location strategy. The default filter type is value (matching the value attribute).

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
</body>
<html>
```

- name=username (4)
- name=continue value=Clear (7)
- name=continue Clear (7)
- name=continue type=button (7)

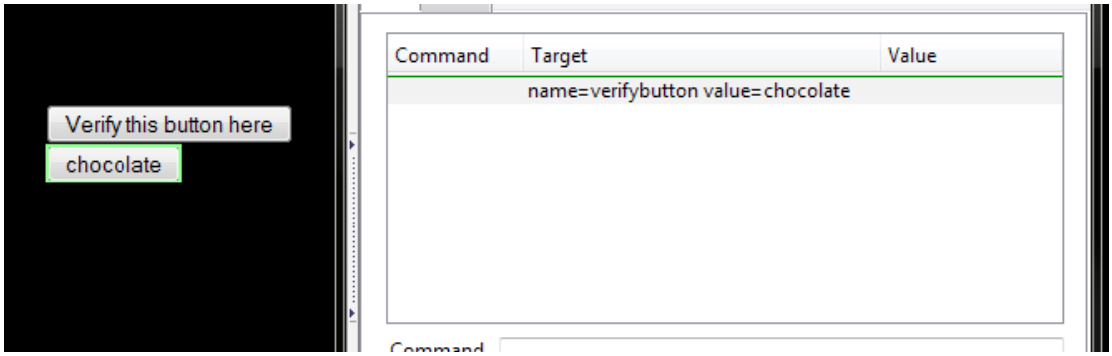
Note

Unlike some types of XPath and DOM locators, the three types of locators above allow Selenium to test a UI element independent of its location on the page. So if the page structure and organization is altered, the test will still pass. You may or may not want to also test whether the page structure changes. In the case where web designers frequently alter the page, but its functionality must be regression tested, testing via id and name attributes, or really via any HTML property, becomes very important.

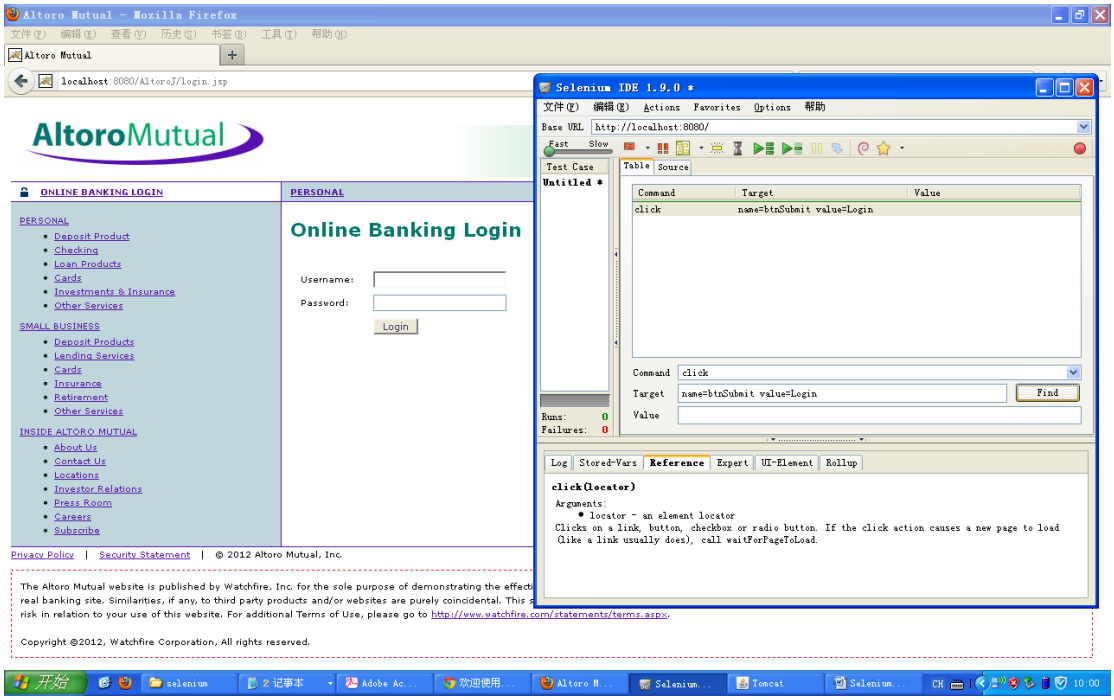
Adding filters to the name

There are times that there may be elements on the page that have the same name but a different attribute. When this happens, we can apply filters to the locator so that it can find the element that we are looking for.

An example of this on the page would be `name=verifybutton value=chocolate;`. This will find the second button with the name `verifybutton`. See an example of this in the next screenshot:



例子：



WebDriver 的做法是不一样的

```
    @Test
    public void testWebDriverSameNameButton() throws Exception {
        driver.get(baseUrl+"AltoroJ/login.jsp");
        //driver.findElement(By.name("btnSubmit
value=Login")).click();
        List<WebElement> eles =
driver.findElements(By.name("btnSubmit"));
        if(eles.get(0).getAttribute("value").equals("Login")){
            System.out.println(eles.get(0).getAttribute("value"));
        }
    }
```

Locating by XPath

XPath is the language used for locating nodes in an XML document. As HTML can be an implementation of XML (XHTML), Selenium users can leverage this powerful language to target elements in their web applications. XPath extends beyond (as well as supporting) the simple methods of locating by id or name attributes, and opens up all sorts of new possibilities such as locating the third checkbox on the page.

One of the main reasons for using XPath is when you don't have a suitable id or name attribute for the element you wish to locate. You can use XPath to either locate the element in absolute terms (not advised), or relative to an element that does have an id or name attribute. XPath locators can also be used to specify elements via attributes other than id and name.

Absolute XPaths contain the location of all elements from the root (html) and as a result are likely to fail with only the slightest adjustment to the application. By finding a nearby element with an id or name attribute (ideally a parent element) you can locate your target element based on the relationship. This is much less likely to change and can make your tests more robust.

Since only `xpath` locators start with "//", it is not necessary to include the `xpath=` label when specifying an XPath locator.

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
</body>
<html>
```

- xpath=/html/body/form[1] (3) - *Absolute path (would break if the HTML was changed only slightly)*
- //form[1] (3) - *First form element in the HTML*
- xpath=//form[@id='loginForm'] (3) - *The form element with attribute named 'id' and the value 'loginForm'*
- xpath=//form[input/@name='username'] (3) - *First form element with an input child element with attribute named 'name' and the value 'username'*
- //input[@name='username'] (4) - *First input element with attribute named 'name' and the value 'username'*
- //form[@id='loginForm']/input[1] (4) - *First input child element of the form element with attribute named 'id' and the value 'loginForm'*
- //input[@name='continue'][@type='button'] (7) - *Input with attribute named 'name' and the value 'continue' and attribute named 'type' and the value 'button'*
- //form[@id='loginForm']/input[4] (7) - *Fourth input child element of the form element with attribute named 'id' and value 'loginForm'*

These examples cover some basics, but in order to learn more, the following references are recommended:

- W3Schools XPath Tutorial
- W3C XPath Recommendation

There are also a couple of very useful Firefox Add-ons that can assist in discovering the XPath of an element:

- XPath Checker - suggests XPath and can be used to test XPath results.
- Firebug - XPath suggestions are just one of the many powerful features of this very useful add-on.

Locating Hyperlinks by Link Text

This is a simple method of locating a hyperlink in your web page by using the text of the link. If two links with the same text are present, then the first match will be used.

```
<html>
 <body>
  <p>Are you sure you want to do this?</p>
  <a href="continue.html">Continue</a>
  <a href="cancel.html">Cancel</a>
</body>
<html>
```

- link=Continue (4)
- link=Cancel (5)

Locating by DOM

The Document Object Model represents an HTML document and can be accessed using JavaScript. This location strategy takes JavaScript that evaluates to an element on the page, which can be simply the element's location using the hierarchical dotted notation.

Since only `dom` locators start with "document", it is not necessary to include the `dom=` label when specifying a DOM locator.

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
```

```
  </form>
</body>
<html>
```

- dom=document.getElementById('loginForm') (3)
- dom=document.forms['loginForm'] (3)
- dom=document.forms[0] (3)
- document.forms[0].username (4)
- document.forms[0].elements['username'] (4)
- document.forms[0].elements[0] (4)
- document.forms[0].elements[3] (7)

You can use Selenium itself as well as other sites and extensions to explore the DOM of your web application. A good reference exists on W3Schools.

Locating by CSS

CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents. CSS uses Selectors for binding style properties to elements in the document. These Selectors can be used by Selenium as another locating strategy.

```
<html>
 <body>
  <form id="loginForm">
   <input class="required" name="username" type="text" />
   <input class="required passfield" name="password"
type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
</body>
<html>
```

- css=form#loginForm (3)
- css=input[name="username"] (4)
- css=input.required[type="text"] (4)
- css=input.passfield (5)
- css=#loginForm input[type="button"] (4)
- css=#loginForm input:nth-child(2) (5)

For more information about CSS Selectors, the best place to go is the W3C publication. You'll find additional references there.

Note

Most experienced Selenium users recommend CSS as their locating strategy of choice as it's considerably faster than XPath and can find the most complicated objects in an intrinsic HTML document.

Implicit Locators

You can choose to omit the locator type in the following situations:

- Locators without an explicitly defined locator strategy will default to using the identifier locator strategy. See Locating by Identifier.
- Locators starting with "//" will use the XPath locator strategy. SeeLocating by XPath.
- Locators starting with "document" will use the DOM locator strategy. See Locating by DOM

XPath

XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。XPath 是 W3C XSLT 标准的主要元素，并且 XQuery 和 XPointer 同时被构建于 XPath 表达之上。因此，对 XPath 的理解是很多高级 XML 应用的基础。

XPath 作为 XML 的强悍的查询语言，可以简洁有效的方式快速查询、定位 XML 节点对象，而 GUI 对象元素可以组成一棵有层次结构的 XML 树（典型的是 WEB 页面对象的 DOM 树），Selenium、Watir、Ranorex 等自动化测试工具都支持通过 XPath 定位测试对象，对于某些不直接支持 XPath 的测试工具，例如 IBM 的 RFT（Rational Functional Tester），也可以自己扩展，让其支持通过 XPath 查找和定位测试对象，具体方法请参考文章《使用 XPath 在 Rational Functional Tester 中动态识别对象》：
http://www.ibm.com/developerworks/cn/rational/r-cn-rftxpathdynobj/
因此，在 Web 测试过程中，尤其是使用某些工具进行自动化测试时，需要我们掌握 XPath 的相关知识，另外，在安全测试方面，检测软件系统的 XPath 注入的漏洞也需要我们深入掌握 XPath 相关的知识。

下面以一个 XML 文件为例，说明 XPath 的一些简单语法和使用方法：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>
```

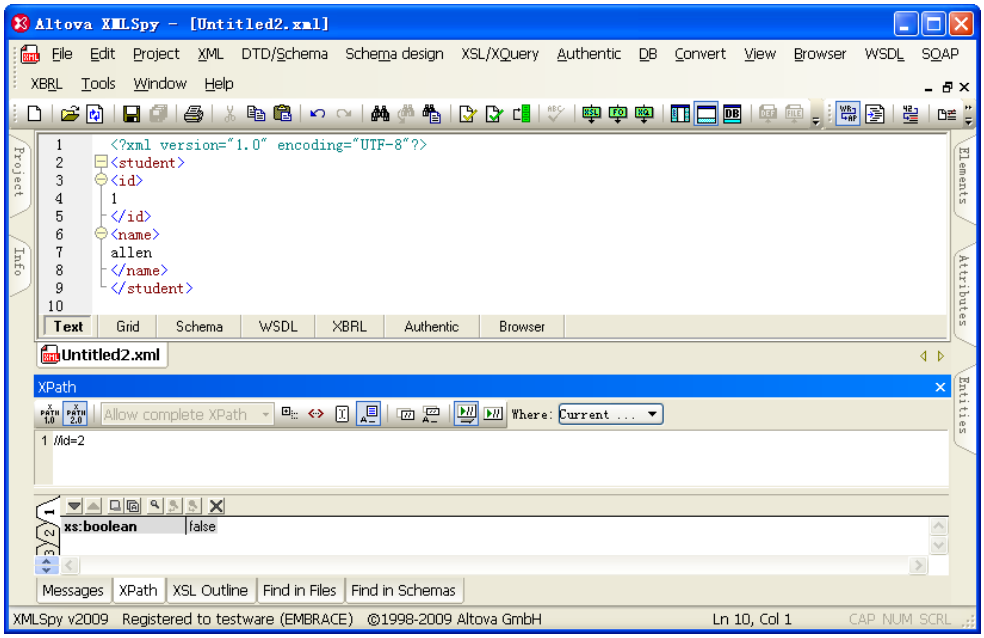
</bookstore>

对于这样的 XML 文件，我们可以通过 XPath 的路径表达式来查找需要的节点，例如：

- bookstore　选取 bookstore 元素的所有子节点
- /bookstore　选取根元素 bookstore，假如路径起始于正斜杠(/)，则此路径始终代表到某元素的绝对路径！
- bookstore/book　选取所有属于 bookstore 的子元素的 book 元素。
- //book　选取所有 book 子元素，而不管它们在文档中的位置。
- bookstore//book　选择所有属于 bookstore 元素的后代的 book 元素，而不管它们位于 bookstore 之下的什么位置。
- //@lang　选取所有名为 lang 的属性。
- /bookstore/book[1]　选取属于 bookstore 子元素的第一个 book 元素。
- /bookstore/book[last()]　选取属于 bookstore 子元素的最后一个 book 元素。
- /bookstore/book[last()-1]　选取属于 bookstore 子元素的倒数第二个 book 元素。
- /bookstore/book[position()<3]　选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
- //title[@lang]　选取所有拥有名为 lang 的属性的 title 元素。
- //title[@lang='eng']　选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
- /bookstore/book[price>35.00]　选取所有 bookstore 元素的 book 元素，且其中的 price 元素的值须大于 35.00。
- /bookstore/book[price>35.00]/title　选取所有 bookstore 元素中的 book 元素的 title 元素，且其中的 price 元素的值须大于 35.00。
- /bookstore/*　选取 bookstore 元素的所有子节点
- //*　选取文档中的所有元素
- //title[@*]　选取所有带有属性的 title 元素。
- //book/title | //book/price　选取所有 book 元素的 title 和 price 元素。
- //title | //price　选取所有文档中的 title 和 price 元素。
- /bookstore/book/title | //price　选取所有属于 bookstore 元素的 book 元素的 title 元素，以及文档中所有的 price 元素。

关于 XPath 的详细内容的学习，请读者参考 W3School 的教程：
http://www.w3school.com.cn/xpath/

技巧：XMLSpy 中提供了一个 XPath 测试功能（XML –> Evaluate XPath），可用于针对 XML 文档编写 XPath 查询和测试，如图所示。读者可利用这个工具来学习 XML 中的 XPath 知识。



Xpather

https://addons.mozilla.org/zh-CN/firefox/addon/xpather/

高级 XPath
教程：
http://www.zvon.org/xxl/XPathTutorial/General_chi/examples.html

参考：
http://www.cnblogs.com/zhaozhan/archive/2009/09/10/1563725.html

XPath 轴(XPath Axes)可定义某个相对于当前节点的节点集：

1、child 选取当前节点的所有子元素
2、parent 选取当前节点的父节点
3、descendant 选取当前节点的所有后代元素（子、孙等）
4、ancestor 选取当前节点的所有先辈（父、祖父等）
5、descendant-or-self 选取当前节点的所有后代元素（子、孙等）以及当前节点本身
6、ancestor-or-self 选取当前节点的所有先辈（父、祖父等）以及当前节点本身
7、preceding-sibling 选取当前节点之前的所有同级节点
8、following-sibling 选取当前节点之后的所有同级节点
9、preceding 选取文档中当前节点的开始标签之前的所有节点
10、following 选取文档中当前节点的结束标签之后的所有节点

11、self 选取当前节点

12、attribute 选取当前节点的所有属性

13、namespace 选取当前节点的所有命名空间节点

preceding-sibling

preceding-sibling 轴(axis)包含上下文节点之前的所有兄弟节点

/AAA/XXX/preceding-sibling::*

```
<AAA>
   <BBB>
      <CCC/>
      <DDD/>
   </BBB>
   <XXX>
      <DDD>
         <EEE/>
         <DDD/>
         <CCC/>
         <FFF/>
         <FFF>
            <GGG/>
         </FFF>
      </DDD>
   </XXX>
   <CCC>
      <DDD/>
   </CCC>
</AAA>
```

在 XLab 中打开实例 | 树视图 (JPG)

//CCC/preceding-sibling::*

```
<AAA>
   <BBB>
      <CCC/>
      <DDD/>
   </BBB>
   <XXX>
      <DDD>
         <EEE/>
         <DDD/>
         <CCC/>
         <FFF/>
         <FFF>
            <GGG/>
         </FFF>
      </DDD>
   </XXX>
   <CCC>
      <DDD/>
   </CCC>
</AAA>
```

在 XLab 中打开实例 | 树视图 (JPG)

空格 normalize-space

normalize-space()	将字符串中前后空格删除后返回	normalize-space(' The XML') 结果: 'The XML'

对象识别技巧（Locating Techniques）

Useful XPATH patterns

text

Not yet written - locate elements based on the text content of the node.

starts-with

Many sites use dynamic values for element's id attributes, which can make them difficult to locate. One simple solution is to use XPath functions and base the location on what you do know about the element. For example, if your dynamic ids have the format `<input id="text-12345" />` where `12345` is a dynamic number you could use the following
XPath: `//input[starts-with(@id, 'text-')]`

contains

If an element can be located by a value that could be surrounded by other text, the contains function can be used. To demonstrate, the element `<span class="top heading bold">` can be located based on the 'heading' class without having to couple it with the 'top' and 'bold' classes using the following
XPath: `//span[contains(@class, 'heading')]`. Incidentally, this would be much neater (and probably faster) using the CSS locator strategy `css=span.heading`

siblings

Not yet written - locate elements based on their siblings. Useful for forms and tables.

Starting to use CSS instead of XPATH

Locating elements based on class

In order to locate an element based on associated class in XPath you must consider that the element could have multiple classes and defined in any order. However with CSS locators this is much simpler (and faster).

- XPath: //div[contains(@class, 'article-heading')]
- CSS: css=div.article-heading

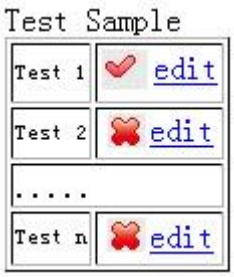
如何高效地认识不带 ID 属性的 web 元素

http://www.ibm.com/developerworks/cn/opensource/os-webautoselenium/?S_TACT=105AGX52&S_CMP=tec.cto

使用一个有含义的 ID 或名称是一种高效且方便的定位元素的方式。它也可以改善测试用例的可读性。但是为了每个元素具有一个有含义的、惟一的 ID（尤其是动态元素），Selenium 提供多种策略来认识元素，比如说 Xpath、DOM 和 CSS。

下面是一个样例，使用三种策略来定位图 6 中提供的动态表格中的一个元素。HTML 代码在清单 11 中。

图 6. 动态表格样例



清单 11. 第一个表格列的 HTML 代码

```
    <table id="test_table" border="1">
            <tbody>
<tr>
   <td align="left">
      <div class="test_class">Test 1</div>
   </td>
   <td align="center" style="vertical-align: top;">
      <table id="AUTOGENBOOKMARK_4">
         <tbody>
            <tr>
               <td align="center" style="vertical-align: top;">
                  <div>
                    <img alt="supported" src="supported.png"/>
                  </div>
               </td>
               <td align="center" style="vertical-align: top;">
                  <div>
                     <a href="test?name=test1">edit</a>
                  </div>
               </td>
    …….
```

Xpath 是一种找到不带特定 ID 或名称的元素的简单方式。

- 如果知道 ID 或名称之外的一个属性，那么直接使用 `@attribute=value` 定位元素。
- 如果只知道属性值的一些特定部分，那么使用 `contains(attribute, value)` 定位元素。
- 如果元素没有指定的属性，那么利用 Firebug 搜索最近的具有指定属性的父元素，然后使用 Xpath 从这个元素开始定位想要找到的那个元素。

表 1. 定位元素的 Xpath 表达式

定位元素	Xpath 表达式
n 行的第一列	`//table[@id='test_table']//tr[n]/td`
n 行的图像	`//table[@id='test_table']//tr[n]//img`
'Test 1' 的编辑链接	`//a[contains(@href,test1)]`

表 1 展示了定位元素的 Xpath 表达式。在 Firebug 的帮助下，Xpath 可以定位元素和复制的元素。在元素没有 ID 和名称时，Selenium IDE 将会采用 Xpath。尽管 Xpath 利用已经录的脚本，有助于保持一致性，但是它高度依赖于 web 页面的结构。这使得测试用例可读性差，增加了维护难度。此外，在 Internet Explorer 7 和 Internet Explorer 8 中运行具有多个复杂 Xpath 表达式的测试用例可能会太慢了。在这种情况下，将 Xpath 更换为 DOM，后者是另一种高效的定位策略。

DOM 是 Document Object Model（文档对象模型）的缩写。Selenium 允许您利用 JavaScript 遍历 HTML DOM。Java 的灵活性允许在 DOM 表达式中有多个语句，用分号隔开，以及在语句中定义函数。

表 2. 定位元素的 DOM 表达式

定位元素	DOM 表达式
n 行的第一列	`dom=document.getElementById('test_table').rows[n-1].cells[0]`
n 行的图像	`dom=element=document.getElementById('test_table').rows[n-1].cells[` `element.getElementsByTagName('IMG')[0]`

‘Test 1’ 的编辑链接	`dom=function test(){ var array=document.getElementsByTagName('a'); var element;for(var i=0; i<array.length; i++) {if(array[i].attributes.getNamedItem("href").\ value.indexOf('test2')!=-1){element=array[i];break;}}return element}; test()`

表 2 展示了定位元素的 DOM 表达式。DOM 定位器在 Firefox 和 Internet Explorer 中也有很好的性能。组织 DOM 表达式需要一些 JavaScript 知识。有时，DOM 表达式对于复杂的元素来说太长了，难以看懂（参见表 2 中提到的 Test 1 的编辑链接的表达式）。

CSS 定位器用于利用 CSS 选择器选择元素。当 HTML 代码具有良好的样式时，可以高效地利用 CSS 定位器。样例表达式展示在表 3 中。

表 3. 定位元素的 CSS 表达式

定位元素	CSS 表达式
n 行的第一列	`css=#test_table .test_class:nth-child(n)`
n 行的图像	`css=#test_table tr:nth-child(n) > td:nth-child(2) > table td:nth-child(1) > div > img`
‘Test 1’ 的编辑链接	`css=a[href*='test2']`

一般来说，选用熟悉的定位器表达式，并在脚本结构中保持一致。如果有多种表达式可执行，那么使用最高效的方式在 web 页面中定位元素。

文本模式匹配（Matching Text Patterns）

Like locators, *patterns* are a type of parameter frequently required by Selenese commands. Examples of commands which require patterns are **verifyTextPresent**, **verifyTitle**, **verifyAlert**,**assertConfirmation**, **verifyText**, and **verifyPrompt**. And as has been mentioned above, link locators can utilize a pattern. Patterns allow you to *describe*, via the use of special characters, what text is expected rather than having to specify that text exactly.

There are three types of patterns: *globbing*, *regular expressions*, and *exact*.

Globbing Patterns

Most people are familiar with globbing as it is utilized in filename expansion at a DOS or Unix/Linux command line such as `ls *.c`. In this case, globbing is used to display all the files ending with a `.c` extension that exist in the current directory. Globbing is fairly limited. Only two special characters are supported in the Selenium implementation:

***** which translates to "match anything," i.e., nothing, a single character, or many characters.

[ ] (*character class*) which translates to "match any single character found inside the square brackets." A dash (hyphen) can be used as a shorthand to specify a range of characters (which are contiguous in the ASCII character set). A few examples will make the functionality of a character class clear:

`[aeiou]` matches any lowercase vowel

`[0-9]` matches any digit

`[a-zA-Z0-9]` matches any alphanumeric character

In most other contexts, globbing includes a third special character, the **?**. However, Selenium globbing patterns only support the asterisk and character class.

To specify a globbing pattern parameter for a Selenese command, you can prefix the pattern with a **glob:** label. However, because globbing patterns are the default, you can also omit the label and specify just the pattern itself.

Below is an example of two commands that use globbing patterns. The actual link text on the page being tested was "Film/Television Department"; by using a pattern rather than the exact text, the **click** command will work even if the link text is changed to "Film & Television Department" or "Film and Television Department". The glob pattern's asterisk will match "anything or nothing" between the word "Film" and the word "Television".

Command	Target	Value

Command	Target	Value
click	link=glob:Film*Television Department	
verifyTitle	glob:*Film*Television*	

The actual title of the page reached by clicking on the link was "De Anza Film And Television Department - Menu". By using a pattern rather than the exact text, the `verifyTitle` will pass as long as the two words "Film" and "Television" appear (in that order) anywhere in the page's title. For example, if the page's owner should shorten the title to just "Film & Television Department," the test would still pass. Using a pattern for both a link and a simple test that the link worked (such as the `verifyTitle` above does) can greatly reduce the maintenance for such test cases.

Regular Expression Patterns

Regular expression patterns are the most powerful of the three types of patterns that Selenese supports. Regular expressions are also supported by most high-level programming languages, many text editors, and a host of tools, including the Linux/Unix command-line utilities **grep**, **sed**, and **awk**. In Selenese, regular expression patterns allow a user to perform many tasks that would be very difficult otherwise. For example, suppose your test needed to ensure that a particular table cell contained nothing but a number. `regexp: [0-9]+`is a simple pattern that will match a decimal number of any length.

Whereas Selenese globbing patterns support only the ***** and **[ ]** (character class) features, Selenese regular expression patterns offer the same wide array of special characters that exist in JavaScript. Below are a subset of those special characters:

PATTERN	MATCH
.	any single character
[ ]	character class: any single character that appears inside the brackets
*	quantifier: 0 or more of the preceding character (or group)
+	quantifier: 1 or more of the preceding character (or group)
?	quantifier: 0 or 1 of the preceding character (or group)
{1,5}	quantifier: 1 through 5 of the preceding character (or group)
|	alternation: the character/group on the left or the character/group on the right

PATTERN	MATCH
( )	grouping: often used with alternation and/or quantifier

Regular expression patterns in Selenese need to be prefixed with either `regexp:` or `regexpi:`. The former is case-sensitive; the latter is case-insensitive.

A few examples will help clarify how regular expression patterns can be used with Selenese commands. The first one uses what is probably the most commonly used regular expression pattern–**.*** ("dot star"). This two-character sequence can be translated as "0 or more occurrences of any character" or more simply, "anything or nothing." It is the equivalent of the one-character globbing pattern ***** (a single asterisk).

Command	Target	Value
click	link=regexp:Film.*Television Department	
verifyTitle	regexp:.*Film.*Television.*	

The example above is functionally equivalent to the earlier example that used globbing patterns for this same test. The only differences are the prefix (**regexp:** instead of **glob:**) and the "anything or nothing" pattern (**.*** instead of just *****).

The more complex example below tests that the Yahoo! Weather page for Anchorage, Alaska contains info on the sunrise time:

Command	Target	Value
open	http://weather.yahoo.com/forecast/USAK0012.html	
verifyTextPresent	regexp:Sunrise: *[0-9]{1,2}:[0-9]{2} [ap]m	

Let's examine the regular expression above one part at a time:

Sunrise: *	The string **Sunrise:** followed by 0 or more spaces
[0-9]{1,2}	1 or 2 digits (for the hour of the day)
:	The character **:** (no special characters involved)
[0-9]{2}	2 digits (for the minutes) followed by a space
[ap]m	"a" or "p" followed by "m" (am or pm)

\s 是空格 ，官方解释: __匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。

\S 非空格之外的所有字符 ，官方解释: __ 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。

[\s\S] 就是一个字符匹配以上两种情况，就是什么都匹配..

[\s\S]* 是匹配 0 到 n 次的意思在加?号是非贪婪匹配

^匹配字符串开始

$匹配字符串结束

Exact Patterns

The **exact** type of Selenium pattern is of marginal usefulness. It uses no special characters at all. So, if you needed to look for an actual asterisk character (which is special for both globbing and regular expression patterns), the **exact** pattern would be one way to do that. For example, if you wanted to select an item labeled "Real *" from a dropdown, the following code might work or it might not. The asterisk in the `glob:Real *` pattern will match anything or nothing. So, if there was an earlier select option labeled "Real Numbers," it would be the option selected rather than the "Real *" option.

select	//select	glob:Real *

In order to ensure that the "Real *" item would be selected, the `exact:` prefix could be used to create an **exact** pattern as shown below:

select	//select	exact:Real *

But the same effect could be achieved via escaping the asterisk in a regular expression pattern:

select	//select	regexp:Real *

It's rather unlikely that most testers will ever need to look for an asterisk or a set of square brackets with characters inside them (the character class for globbing patterns). Thus, globbing patterns and regular expression patterns are sufficient for the vast majority of us.

等待

The "AndWait" Commands

The difference between a command and its *AndWait* alternative is that the regular command (e.g. *click*) will do the action and continue

with the following command as fast as it can, while the *AndWait* alternative (e.g. *clickAndWait*) tells Selenium to **wait** for the page to load after the action has been done.

The *AndWait* alternative is always used when the action causes the browser to navigate to another page or reload the present one.

Be aware, if you use an *AndWait* command for an action that does not trigger a navigation/refresh, your test will fail. This happens because Selenium will reach the *AndWait*'s timeout without seeing any navigation or refresh being made, causing Selenium to raise a timeout exception.

The waitFor Commands in AJAX applications

In AJAX driven web applications, data is retrieved from server without refreshing the page. Using *andWait* commands will not work as the page is not actually refreshed. Pausing the test execution for a certain period of time is also not a good approach as web element might appear later or earlier than the stipulated period depending on the system's responsiveness, load or other uncontrolled factors of the moment, leading to test failures. The best approach would be to wait for the needed element in a dynamic period and then continue the execution as soon as the element is found.

This is done using *waitFor* commands, as *waitForElementPresent* or *waitForVisible*, which wait dynamically, checking for the desired condition every second and continuing to the next command in the script as soon as the condition is met.

Selenium 测试 AJAX

1、加 ID
2、fireevent 触发事件、keypress、mouseOver、mouseDown、mouseUp
3、用 WebDriver
4、加延时判断、WaitForCondition

fireevent

http://luocb1980.iteye.com/blog/1114550

```
1.    selenium.select("A", "label=Boy");
2.    selenium.fireEvent("A","blur");
3.    selenium.fireEvent("A","change");
4.    pause(1000);
5.    selecium.select("B","label=Football");
6.
7.    selenium.select("A", "label=Boy");
8.    selenium.fireEvent("A","blur");
9.    selenium.fireEvent("A","change");
10.   pause(1000);
11.   selecium.select("B","label=Football");
```

http://www.iteye.com/topic/215091

```
1.    selenium.type("xpath=//*[@autotest='" + element + "']", type);
2.    selenium.fireEvent("xpath=//*[@autotest='" + element + "']", "change");
3.    selenium.fireEvent("xpath=//*[@autotest='" + element + "']", "blur");
```

Actions

http://nbkhic.iteye.com/blog/1456936
webdriver 里面已经没有了 fire_event 方法，就像世界上再也没有萨达姆，本拉登和卡扎菲一样。

不过我们可以通过其他方法来实现 fire_event 的相似功能。

考虑下面的 html，当鼠标悬停到 Mouse Over Here 链接上时，js 的 mouseover 事件被触发，show_tips() 函数将被执行，隐藏的 tips div 会显示在页面上。

Html 代码

```
1.   <html>
2.       <head>
3.           <title>FireEvent</title>
4.           <style>
5.               .mo {color: blue;}
6.               .tips {display:none;border: 1px solid #ccc; background-color:#EFEFEF;widt
     h: 100px;height:100px}
```

```
        </style>
        <script>
            function show_tips(){
                document.getElementById("t").style.display = "block";
            }
            function hide_tips(){
                document.getElementById("t").style.display = "none";
            }
        </script>
    </head>
    <body>
        <a class = "mo" href = "#" onmouseover = "show_tips()" onmouseout = "hide_tips()">Mouse Over Here</a>
        <div id = "t" class = "tips">This is the tips of link</div>
    </body>
</html>
```

如果存在 fire_event 方法的话，直接在 Mouse Over Here 链接上触发 oumouseover 就能达到显示隐藏 div 的效果了，但是 webdriver 取消了 fire_event，所以这时候我们就需要求助于另一个功能强大的类，Action 类。

Action 类给用户提供了一些模拟用户交互方法，比如模拟 key_down，key_up, double_click 等。

下面的代码使用 Action 类的 move_to 方法模拟了鼠标的悬停事件，需要注意 3 点：

Action 类并不需要显示的实例化，调用时只需要通过 driver.action 直接调用该实例既可；

move_to 方法执行完毕后悬停的效果也就消失了，所以代码中使用了循环 10 次的方法来人为"延长"事件的持续时间；

调用 move_to 方法只是注册但并未真正的触发实际动作，需要调用 perform 方法来执行注册了的动作；

Ruby 代码

```
require 'rubygems'
require 'selenium-webdriver'
dr = Selenium::WebDriver.for :firefox
select_file = 'file:///'.concat File.expand_path(File.join(File.dirname(__FILE__), 'fire_event.html'))
dr.navigate.to select_file

m = dr.find_element(:css => '.mo')
10.times do
    dr.action.move_to(m).perform
```

```
10.  end
```

JavascriptExecutor

参考：
http://blog.csdn.net/shandong_chu/article/details/7074812
http://www.cxyclub.cn/n/15001/
http://code.google.com/p/selenium/wiki/FrequentlyAskedQuestions
http://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/JavascriptExecutor.html#executeScript(java.lang.String, java.lang.Object...)

另外也可以直接调用 js 引擎执行 show_tips 函数，这样就不需要去模拟事件了。

Ruby 代码

```
1.  js = %q[show_tips();]
2.  dr.execute_script js
```

Java 代码：

```
    @Test
    public void test_fireEvent_JS(){

        WebDriver driver = new FirefoxDriver();

        driver.get("file:///D:/selenium/MyTest/AUT/fireEvent.html");

        WebElement mo = driver.findElement(By.cssSelector(".mo"));

        JavascriptExecutor js = (JavascriptExecutor) driver;
        //System.out.println(js.executeScript("return
document.title").toString());
        js.executeScript("show_tips();");

    }
```

测试 GWT 控件

参考：
http://dingyichen.livejournal.com/23628.html
http://blog.blackpepper.co.uk/black-pepper-blog/Simulating-clicks-on-GWT-push-buttons-with-Selenium-RC.html
http://stackoverflow.com/questions/2084233/selenium-testing-of-gwt-2-0

http://www.cnblogs.com/samwu/archive/2012/06/16/2551792.html

mouseMove、mouseDown、mouseUp、keyPress

```
package com.testscripts;

import org.junit.Test;
import org.openqa.selenium.Keys;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.Selenium;


public class SeleniumRC_AJAX {

    @Test
    public void test_SekeniumRC_AJAX_ClickButton() throws
InterruptedException{

        Selenium selenium = new DefaultSelenium("localhost", 4444,
"*firefox", "http://www.baidu.com/");
        selenium.start();


    selenium.open("http://gwt.google.com/samples/Showcase/Showcase.ht
ml#!CwTree");

        boolean elementPresent = false;
        for(int i=0;i<30;i++)
        {
            if
(selenium.isElementPresent("gwt-debug-cwTree-staticTree-root-child0-c
ontent"))
            {
                elementPresent = true;
                break;
            }
            Thread.sleep(1000);
        }

        if(elementPresent)
        {
```

```
            System.out.println("OK!");


    selenium.mouseMove("gwt-debug-cwTree-staticTree-root-child0-conte
nt");

    selenium.mouseDown("gwt-debug-cwTree-staticTree-root-child0-conte
nt");

    selenium.mouseUp("gwt-debug-cwTree-staticTree-root-child0-content
");


    //selenium.mouseMove("gwt-debug-cwTree-staticTree-root-child0-ima
ge");

    //selenium.mouseDown("gwt-debug-cwTree-staticTree-root-child0-ima
ge");

    //selenium.mouseUp("gwt-debug-cwTree-staticTree-root-child0-image
");


    selenium.keyPress("gwt-debug-cwTree-staticTree-root-child0-conten
t","\\39");//39 就是向右键的ASCII码

        }

    }
}
```

WebDriverBackedSelenium

```
package com.testscripts;

import java.util.concurrent.TimeUnit;

import org.junit.Ignore;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebDriverBackedSelenium;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

import com.thoughtworks.selenium.Selenium;


public class WebDriver_GWT {

    @Ignore
    public void test_ClickButton() throws InterruptedException{

        WebDriver driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);


    driver.get("http://gwt.google.com/samples/Showcase/Showcase.html#
!CwTree");

        WebElement element =
driver.findElement(By.id("gwt-debug-cwTree-staticTree-root-child0-con
tent"));
        //WebElement element =
driver.findElement(By.xpath("//*[@id='gwt-debug-cwTree-staticTree-roo
t-child0-content']"));

        element.click();

        //element =
driver.findElement(By.id("gwt-debug-cwTree-staticTree-root-child0-ima
ge"));// //*[@id="gwt-debug-cwTree-staticTree-root-child0-image"]/img
        //element.click();
        Actions action = new Actions(driver) ;
        //action.moveToElement(element).perform();
        action.sendKeys(Keys.RIGHT).perform();

        driver.close();
    }

    @Test
    public void test_test_ClickButton_WebDriverBackedSelenium() throws
InterruptedException{
```

```
        WebDriver driver = new FirefoxDriver();
        //driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);

        Selenium selenium = new WebDriverBackedSelenium(driver,
"http://gwt.google.com/samples/Showcase/Showcase.html#!CwTree");

selenium.open("http://gwt.google.com/samples/Showcase/Showcase.html#!
CwTree");
        selenium.waitForPageToLoad("30000");

selenium.click("gwt-debug-cwTree-staticTree-root-child0-content");


//selenium.click("gwt-debug-cwTree-staticTree-root-child0-image"); //
/html/body/div[4]/div[2]/div/div[4]/div/div[3]/div/div[3]/div/div[2]/
div/div[2]/div/div[3]/div/div/table/tbody/tr[2]/td/table/tbody/tr[2]/
td[2]/div/div/div/div/div[2]/table/tbody/tr/td/img
                                                                    //
//*[@id="gwt-debug-cwTree-staticTree-root-child0-image"]

//selenium.mouseOver("id=gwt-debug-cwTree-staticTree-root-child0-imag
e");

//selenium.mouseDown("id=gwt-debug-cwTree-staticTree-root-child0-imag
e");

//selenium.mouseUp("id=gwt-debug-cwTree-staticTree-root-child0-image"
);


selenium.keyPress("gwt-debug-cwTree-staticTree-root-child0-content",
Keys.RIGHT.toString());

    }
}
```

Testing Ext JS & Ext GWT Applications With Selenium

参考：

http://www.sencha.com/blog/testing-ext-js-ext-gwt-applications-with-selenium/

Selenium RC

```
package com.testscripts;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.SeleneseTestCase;
import com.thoughtworks.selenium.Selenium;


public class SeleniumRC_GXT extends SeleneseTestCase {

	private Selenium selenium;

	public void setUp() {
	  selenium = new DefaultSelenium("localhost", 4444, "*iexplore",
"http://extjs.com");
	  selenium.start();
	}

	public void testForm() {
	  selenium.open("http://extjs.com/playpen/gxt/selenium/");
	  pause(10000);

	  selenium.type("//input[@name='name']", "John");
	  selenium.fireEvent("//input[@name='name']", "blur"); //onFocus
事件就是当光标落在文本框中时发生的事件。 onBlur事件是光标失去焦点时发生的事件。
	  assertTrue(selenium.isElementPresent("//input[@name='name' and
contains(@class, 'x-form-invalid')]"));

	  selenium.type("//input[@name='name']", "Darrell");
	  selenium.fireEvent("//input[@name='name']", "blur");
	  assertFalse(selenium.isElementPresent("//input[@name='name'
and contains(@class, 'x-form-invalid')]"));

	  selenium.type("//input[@name='email']", "darrell@foo.com");
	  assertEquals("Darrell",
selenium.getValue("//input[@name='name']"));
	  assertEquals("darrell@foo.com",
selenium.getValue("//input[@name='email']"));

	  selenium.focus("//input[@name='company']");
```

```
selenium.click("//input[@name='company']/following-sibling::img");
        assertEquals("43",
selenium.getXpathCount("//div[@class='x-combo-list-item']"));

        selenium.click("//div[@id='Apple']");
        assertEquals("Apple Inc.",
selenium.getValue("//input[@name='company']"));


selenium.click("//input[@name='birthday']/following-sibling::img");
        selenium.click("//button[contains(text(), \"Today\")]");

assertTrue(selenium.getValue("//input[@name='birthday']").matches("^[
\\s\\S]*$"));

        selenium.check("//input[@value='Classical']");
        assertEquals("on",
selenium.getValue("//input[@value='Classical']"));

        selenium.check("//input[@value='Blue']");
        assertEquals("on",
selenium.getValue("//input[@value='Blue']"));

        selenium.uncheck("//input[@value='Red']");
        assertEquals("off",
selenium.getValue("//input[@value='Red']"));
      }

      public void tearDown() {
        //selenium.stop();
      }

    }
```

WebDriver

```
package com.testscripts;

import java.util.List;
import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.junit.*;
```

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.Select;

public class WebDriver_GXT {
    private WebDriver driver;
    private String baseUrl;
    private StringBuffer verificationErrors = new StringBuffer();
    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl = "http://dev.sencha.com/";
        driver.manage().timeouts().implicitlyWait(30,
TimeUnit.SECONDS);
    }

    @Test
    public void test1() throws Exception {
        driver.get(baseUrl + "/playpen/gxt/selenium/");

        driver.findElement(By.id("x-auto-4")).clear();
        driver.findElement(By.id("x-auto-4")).sendKeys("John");

        //selenium.fireEvent("//input[@name='name']", "blur");
        assertTrue(isElementPresent(By.xpath("//input[@name='name' and
contains(@class, 'x-form-invalid')]")));
        WebElement ele = driver.findElement(By.id("x-auto-24"));
        assertTrue(ele.isDisplayed());
        //System.out.println(driver.getPageSource());

        driver.findElement(By.id("x-auto-4")).clear();
        driver.findElement(By.id("x-auto-4")).sendKeys("Darrell");
        driver.findElement(By.id("x-auto-5")).click();

        //assertFalse(isElementPresent(By.xpath("//input[@name='name'
and contains(@class, 'x-form-invalid')]")));
        ele = driver.findElement(By.id("x-auto-24"));
        assertFalse(ele.isDisplayed());
        //System.out.println("--------------------");
        //System.out.println(driver.getPageSource());
```

```
        driver.findElement(By.id("x-auto-5")).clear();

driver.findElement(By.id("x-auto-5")).sendKeys("Darrell@126.com");
        //driver.findElement(By.id("x-auto-5")).sendKeys(Keys.TAB);

        //assertTrue(isTextPresent("x-auto-5","Darrell@126.com"));

////System.out.println(driver.findElement(By.id("x-auto-5")).getAttri
bute("value"));

    assertEquals("Darrell@126.com",driver.findElement(By.id("x-auto-5
")).getAttribute("value"));

        WebElement ele_company =
driver.findElement(By.xpath("//input[@name='company']"));
        ele_company.click();

    driver.findElement(By.xpath("//input[@name='company']/following-s
ibling::img")).click();

        //assertEquals("43",
selenium.getXpathCount("//div[@class='x-combo-list-item']"));
        //Select select = new
Select(driver.findElement(By.id("x-auto-6")));
        //assertEquals("43",select.getAllSelectedOptions().size());
//org.openqa.selenium.support.ui.UnexpectedTagNameException: Element
should have been "select" but was "div"
        List<WebElement> List =
driver.findElements(By.xpath("//div[@class='x-combo-list-item']"));
        assertEquals(43,List.size());

        driver.findElement(By.xpath("//div[@id='Apple']")).click();
        assertEquals("Apple Inc.",
driver.findElement(By.xpath("//input[@name='company']")).getAttribute
("value"));


driver.findElement(By.xpath("//input[@name='birthday']/following-sibl
ing::img")).click();
        driver.findElement(By.xpath(("//button[contains(text(),
\"Today\")]"))).click();

assertTrue(driver.findElement(By.xpath("//input[@name='birthday']")).
getAttribute("value").matches("^[\\s\\S]*$"));
```

```
driver.findElement(By.xpath("//input[@value='Classical']")).click();
        assertEquals("true",
driver.findElement(By.xpath("//input[@value='Classical']")).getAttrib
ute("checked"));


//driver.findElement(By.xpath("//input[@value='Blue']")).click();
        //assertEquals("true",
driver.findElement(By.xpath("//input[@value='Blue']")).getAttribute("
checked"));

        WebElement fcolor
=driver.findElement(By.xpath("//input[@value='Blue']"));
        fcolor.click();
        assertTrue(fcolor.isSelected());

assertFalse(driver.findElement(By.xpath("//input[@value='Red']")).isS
elected());

    }

    @After
    public void tearDown() throws Exception {
        /*
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
        */
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }

    public boolean isTextPresent(String elementId, String str) {
```

```
        WebElement bodyElement = driver.findElement(By.id(elementId));
        return bodyElement.getText().contains(str);
    }

}
```

测试 SmartGWT

参考：
http://www.taobaotest.com/blogs/qa?bid=9756
https://docs.jboss.org/author/display/RHQ/Testing+SmartGWT+with+Selenium?_sscc=t
http://blog.codecentric.de/en/2010/12/testing-smartgwt-applications-with-selenium-and-robot-framework/
http://www.thomas-letsch.de/2012/automated-tests-of-smartgwt-application-with-selenium/
http://forums.smartclient.com/showthread.php?t=19858

Selenium RC

https://docs.jboss.org/author/display/RHQ/Testing+SmartGWT+with+Selenium?_sscc=t
http://forums.smartclient.com/showthread.php?t=13058

Smart GWT includes a free, custom Selenium extension for robust record and playback of tests, including the ability to record on one browser and play back on others, support for Selenium Remote Control allowing tests to be written in a variety of programming languages and run as scripts, as well as Smart GWT-specific enhancements to the Selenium IDE.

These extensions and a brief user guide can be found in the `selenium/` directory in the top level of the SDK.

```
protected static DefaultSelenium selenium;

    @BeforeClass
    public static void setUp() throws Exception {
        selenium = new DefaultSelenium("localhost", 4444, "*firefox",
getBaseUrl());

selenium.setExtensionJs("src/main/resources/user-extensions.js");
        selenium.start();
```

```
        selenium.windowMaximize();
        //BaseSeleniumChecks.selenium = selenium;
    }
```

```
package com.testscripts;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.openqa.selenium.server.RemoteControlConfiguration;
import org.openqa.selenium.server.SeleniumServer;

import java.io.File;
import java.util.regex.Pattern;

public class SeleniumRC_SmartGWT {

    public static SeleniumServer seleniumserver;
    public static DefaultSelenium selenium;

    @BeforeClass
    public static void setUpClass() throws Exception {

        seleniumserver=new SeleniumServer();
        RemoteControlConfiguration a=
seleniumserver.getConfiguration();
        File uimap=new
File("D:/selenium/MyTest/AUT/smartgwt-3.0/selenium/user-extensions.js
");
        a.setUserExtensions(uimap);
        seleniumserver.boot();
        seleniumserver.start();
    }

    @Before
    public void setUp(){
        selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://www.smartclient.com/");

//selenium.setExtensionJs("D:/selenium/MyTest/AUT/smartgwt-3.0/seleni
um/user-extensions.js");
```

```
        selenium.start();
        selenium.windowMaximize();
    }


    @Test
    public void test1() throws Exception {
        selenium.open("/smartgwt/showcase/#main");
        selenium.type("id=isc_A3","123");

    selenium.click("scLocator=//DynamicForm[ID=\"isc_DynamicForm_3\"]
/item[name=description||title=Search||value=123||index=0||Class=TextI
tem]/[icon='_0']");

        //selenium.waitForCondition("1==0", "10000");
    }

    @After
    public void tearDown(){
        selenium.stop();
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
        seleniumserver.stop();
    }
}
```

WebDriver

SmartGWT does not support WebDriver as of now. See http://forums.smartclient.com/showthread.php?t=19858 for an official statement.

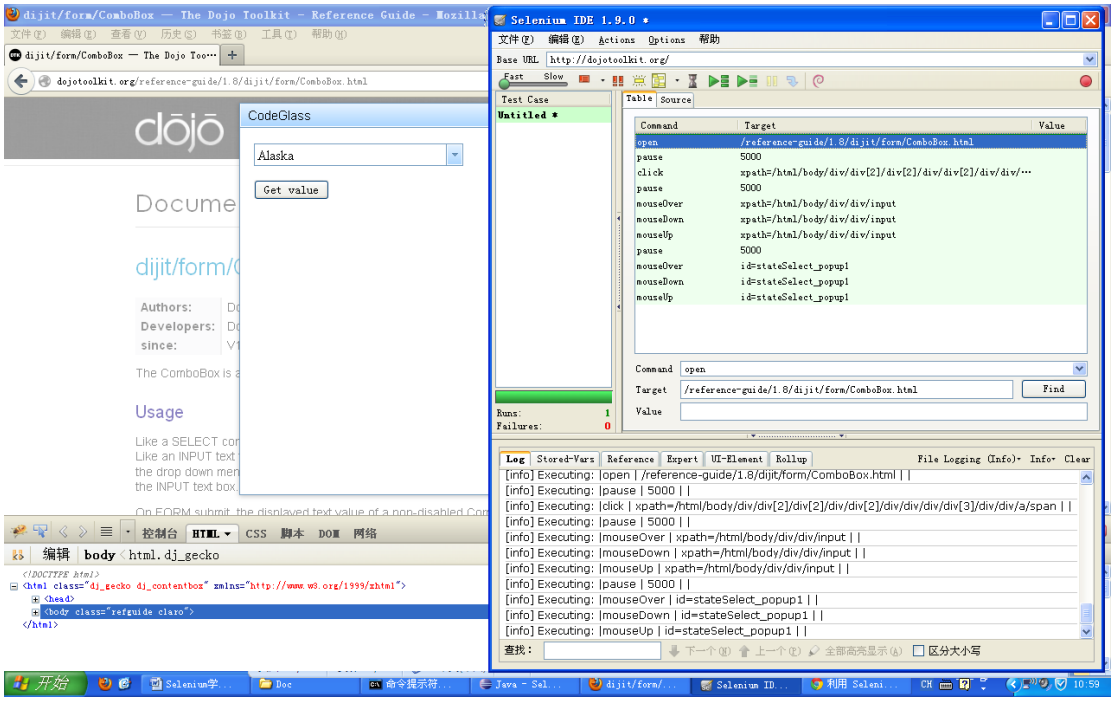
```
package com.testscripts;

import static org.junit.Assert.assertEquals;

import java.util.List;
import java.util.concurrent.TimeUnit;

import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;


public class WebDriver_SmartGWT {

    @Test
    public void test1(){
        WebDriver driver = new FirefoxDriver();
driver.get("http://www.smartclient.com/smartgwt/showcase/#featured_master_detail");
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);

        //find WebElement which contains SmartGWT/ComboBox
        WebElement element =
driver.findElement(By.xpath("//td[starts-with(text(),'Units')]/following-sibling::td[1]") );
        element.click();
        //System.out.println("id:[" + element.getAttribute("id")+ "] $89=["+element.getAttribute("$89")+"]" +element.getText());

        List<WebElement> List =
driver.findElements(By.xpath("/html/body/div/div/div/div/div/table/tbody/tr/td/div/nobr"));
        assertEquals(11,List.size());


    driver.findElement(By.xpath("/html/body/div/div/div/div/div/table/tbody/tr[8]")).click();
        assertEquals("Tin",
driver.findElement(By.xpath("/html/body/div[3]/div/div[2]/div/div[2]/div/div/div/div[2]/div/div/div/div[2]/div/div/div/div/div/div[2]/div/form/table/tbody[2]/tr[4]/td[2]/table/tbody/tr")).getAttribute("textContent"));


    }
}
```

测试 DOJO

参考：
http://hi.baidu.com/onlysis/blog/item/110814ecf0a19e3363d09ffb.html
http://www.silverwareconsulting.com/index.cfm/2010/6/7/Checking-for-JavaScript-Errors-with-Selenium
http://blog.csdn.net/oychw/article/details/6339289

pause、mouseOver、mouseDown、mouseUp



等待

使用类 `com.thoughtworks.selenium.Wait` 来等待一个元素或文本在页面上出现或消失。可以在 `until()` 函数中定义等待的退出条件，或者扩展 `Wait` 类来实现等待退出。清单 1 是使用 `Wait` 类的样例代码。它将在条件满足时停止等待，或者在超出最大等待时间时返回一个超时异常。

清单 1. 等待元素或文本出现

```
        Wait wait = new Wait() {
        public boolean until() {
        return selenium.isElementPresent(locator);
        // or selenium.isTextPresent(pattern);
```

```
        }
};
wait.wait("", timeoutInMilliseconds);
```

另一种选择是使用 Selenium 中的 `waitForCondition` 函数，一个 JavaScript 代码片段将被作为参数传递给该函数。一旦 Selenium 检测到条件返回为真，它将停止等待。您可以等待一些元素或文本出现或者不出现。

Handle AJAX elements in Selenium 2 (WebDriver)
http://www.thoughtworks-studios.com/twist/2.3/help/how_do_i_handle_ajax_in_selenium2.html

The implicit wait in Selenium 2 might not work for Ajax elements. We recommend you to use any one of the following workaround to handle Ajax elements.

One approach is to use FluentWait and a Predicate available with Selenium2. The advantage of this approach is that element polling mechanism is configurable. The code example below waits for 1 second and polls for a textarea every 100 milliseconds.

```
    FluentWait<By> fluentWait = new
FluentWait<By>(By.tagName("TEXTAREA"));
        fluentWait.pollingEvery(100, TimeUnit.MILLISECONDS);
        fluentWait.withTimeout(1000, TimeUnit.MILLISECONDS);
        fluentWait.until(new Predicate<By>() {
            public boolean apply(By by) {
                try {
                    return browser.findElement(by).isDisplayed();
                } catch (NoSuchElementException ex) {
                    return false;
                }
            }
        });
        browser.findElement(By.tagName("TEXTAREA")).sendKeys("text to
enter");
```

Another approach is to use ExpectedCondition and WebDriverWait strategy. The code below waits for 20 seconds or till the element is available, whichever is the earliest.

```
      public ExpectedCondition<WebElement>
visibilityOfElementLocated(final By by) {
        return new ExpectedCondition<WebElement>() {
          public WebElement apply(WebDriver driver) {
            WebElement element = driver.findElement(by);
            return element.isDisplayed() ? element : null;
          }
        };
      }

      public void performSomeAction() {
        ..
        ..
        Wait<WebDriver> wait = new WebDriverWait(driver, 20);
        WebElement element =
wait.until(visibilityOfElementLocated(By.tagName("a")));
        ..
      }
```

Converting Selenium waitForCondition to WebDriverWait

http://www.tarnowski.se/2011/09/11/converting-selenium-waitforcondition-to-webdriverwait/

I usually don't write simple tutorial posts, but I'd like to share this one, because it took me some time to google the answer, and I still had to adapt it to my case.

I had the simplest possible vanilla Ajax application, that would react to keystrokes in a textfield and then return a list of words based on the entered value. You see this everywhere! My particular HTML snippet looked like this:

```
<body onload="document.getElementById('prefixText').focus()">
<input type="text" id="prefixText"
onkeyup="checkForExpansion(document.getElementById('prefixText').valu
e)"/>
<br/>
<span id="ajaxResults"></span>
</body>
```

checkForExpansion just made an asynchronous call the a servlet that returned a list of completions. *However*, one thing was a little untypical with this application: The results went in as strings separated by
 tags into the span element. The consequence of this was that there was no obvious element to wait for when testing this page; the span was there already.

Now, my first test went like this:

```
@Test
public void trivialAjaxListCompletionExampleUsingSelenium1StyleWait()
{
    Selenium selenium = new WebDriverBackedSelenium(webDriver,
testedUrl);
    webDriver.get(testedUrl);
    webDriver.findElement(By.id("prefixText")).sendKeys("a");

selenium.waitForCondition("selenium.browserbot.getCurrentWindow().doc
ument.getElementById('ajaxResults').innerHTML.indexOf('ALEX') > 0",
"1000");
}
```

It worked, but it relied on WebDriverBackedSelenium, which I really didn’t want it to. After some googling and experiementing, I came up with the followging:

```
@Test
public void trivialAjaxListCompletionExampleUsingWebDriverStyleWait()
{
    webDriver.get(testedUrl);
    webDriver.findElement(By.id("prefixText")).sendKeys("a");

    boolean isListPopulated = (new WebDriverWait(webDriver, 1000))
            .until(new ExpectedCondition() {
                public Boolean apply(WebDriver d) {
                    JavascriptExecutor javascriptExecutor =
(JavascriptExecutor) webDriver;
                    return
(Boolean)javascriptExecutor.executeScript("return
document.getElementById('ajaxResults').innerHTML.indexOf('ALEX') >
0");
                }
            });
    assertTrue(isListPopulated);
}
```

As a test, this sucks, but this snippet shows several interesting things. First we have the WebDriverWait class. Waiting can be quite fine-tuned using the FluentWait class.

Then we have an example of the browserbot being dropped (since it's a Selenium 1 thing) and Javascript with a return (mandatory in Selenium 2).

Finally, we have the boolean test. Many examples on the web are written so that they wait for a certain element to appear, and then return it. Since the element (the span) was already present, I wanted to make the example a little odd.

WaitForCondition

http://www.cnblogs.com/cnblogsfans/archive/2010/07/27/1785834.html

使用 Selenium 进行 Ajax 测试

如果我们点击 Get 按钮后，直接判断页面是否返回”Hello World”时，这样会出错，因为现在还没有立即返回。

这时我们就需要使用 Selenium 的 WaitForCondition 方法，这个方法会不停的去判断一个 javascript 表达式是否是 true. 知道返回 true 时才继续执行。

```
using ThoughtWorks.Selenium;
using ThoughtWorks.Selenium.UnitTests;
using ThoughtWorks.Selenium.IntegrationTests;
using Selenium;

namespace MvcApplication1.Tests.Controllers
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
        public void TestAjax()
        {
            DefaultSelenium selenium = new DefaultSelenium("localhost",
                                       4444, "*firefox", @"http://localhost:4288/");

            selenium.Start();

            selenium.Open(@"http://localhost:4288/");

            selenium.WaitForPageToLoad("3000");

            selenium.Click("id=GetContent");

            selenium.WaitForCondition(
                string.Format(" var allText = selenium.page().bodyText(); allText.indexOf('{0}')>0 ;",
                "Hello World"), "30000");

            Assert.IsTrue(selenium.IsTextPresent("Hello World"));
        }

    }
```

我们可以看到，我们需要写 js 表达式，如果我们需要判断的很复杂，那么些表达式也是一件很麻烦的事，同时这个方法经过我的使用，它对 IE 的支持不好。那么有没有更好的方法呢？

更好的方法判断 Ajax 调用是否结束

事实上我们测试 Ajax 程序最大的麻烦，就是想上面一样来判断 Ajax 调用是否结束，上面我们是判断页面中如果出现”Hello World”，证明 Ajax 已经结束了，那么什么是更好的方法呢，我们经常使用的 javascript 的类库是 JQuery，如何判断 JQuery 的 Ajax 调用已经结束了呢？

经过研究发现当没有 ajax 调用时，Jquery.active=0.

所以，我们可以使用这个万能的表达式判断 Ajax 是否返回。

```
selenium.WaitForCondition("selenium.browserbot.getCurrentWindow().jQuery.active == 0", "50000");
```

```
using ThoughtWorks.Selenium;
using ThoughtWorks.Selenium.UnitTests;
using ThoughtWorks.Selenium.IntegrationTests;
using Selenium;

namespace MvcApplication1.Tests.Controllers
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
        public void TestAjax()
        {
            DefaultSelenium selenium = new DefaultSelenium("localhost",
                                       4444, "*firefox", @"http://localhost:4288/");

            selenium.Start();

            selenium.Open(@"http://localhost:4288/");

            selenium.WaitForPageToLoad("3000");

            selenium.Click("id=GetContent");

            selenium.WaitForCondition("selenium.browserbot.getCurrentWindow().jQuery.active == 0", "50000");

            Assert.IsTrue(selenium.IsTextPresent("Hello World"));
        }

    }
}
```

是不是整个世界清静多了 

下面我收集了判断其它类库的 Ajax 活动状态

jQuery: “jQuery.active”

Prototype: “Ajax.activeRequestCount”

Dojo: “dojo.io.XMLHTTPTransport.inFlight.length”

ASP.NET AJAX:

Javascript:

```
function isInAsyncPostBack() {
instance = Sys.WebForms.PageRequestManager.getInstance();

return instance.get_isInAsyncPostBack();
}
```

C#

selenium.WaitForCondition(“!selenium.browserbot.getCurrentWindow().isInAsyncPostBack()”, “1000″);

等待窗口就绪的状态

```
package com.testscripts;

import org.junit.Test;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.Selenium;


public class SeleniumRC_Wait {


    //等待窗口就绪的状态
    @Test
    public void test_WaitWindow(){
        Selenium selenium = new DefaultSelenium("localhost", 4444,
"*firefox", "http://www.baidu.com/");
        selenium.start();

        selenium.open("http://www.AutomationQA.com");

        String script = "var my_window =
selenium.browserbot.getCurrentWindow();" ;
        script += "var bool;";
        script += "var readyState = (my_window.document.readyState);";
        script += "if (readyState == 'complete'){";
        script += "bool = 'true';";
        script += "}";
        script += "bool;";

        selenium.waitForCondition(script, "30000");
    }
}
```

Browserbot:

参考：

https://community.neustar.biz/community/wpm/blog/2011/03/14/selenium-tips--wait-with-waitforcondition

As the first part of this Selenium Tips series, this article intends to summarize the different ways to interact with dynamic sites that refresh content asynchronously wherein the script will have to wait for certain elements to appear or disappear before proceeding further. As you are already aware of,

Selenium has several 'waitFor' commands that fulfill this purpose. waitForCondition is one among them. 'waitForCondition' basically takes in two arguments, a JavaScript snippet and a timeout period in milliseconds. The snippet is executed either until it returns true or until the timeout period, after which the command will return an error. Now enough of theory and let's get into action.

Example:

Variation 1:

Consider the following example where there are two drop-downs, one for the states and the other for counties.

The county drop-down is populated based on the state selection.

(Example courtesy of bitrepository.com)

You can see in the site above, when the state is selected, a spinner shows up for a few seconds before the county drop-down is populated. Using waitForCondition and some JavaScript, we wait until display style attribute changes to 'none' in the snippet below.

```
1. selenium.open("http://www.bitrepository.com/apps/viewDemo/?originalPost=http://www.bitrepository.com/dynamic-dependant-dropdown-list-us-states-counties.html&demoPage=http://www.bitrepository.com/demo/dynamic-dependent-dropdown-list/");
2. selenium.selectFrame("mainFrame");
3. selenium.select("state", "label=California");
4. selenium.waitForCondition("var value = selenium.browserbot.findElementOrNull('loading_county_drop_down'); value.style.display == 'none'", "10000");
5. selenium.select("county", "label=Amador County");
```

Variation 2:

You may also use logical operators to add additional logic.

```
1. selenium.waitForCondition("var value = selenium.browserbot.findElementOrNull('loading_county_drop_down'); value.style.display == 'none' || selenium.browserbot.getUserWindow().document.form.county.options[0].text == 'Select a county of California';'', "10000");
```

Variation 3:

The same can also be accomplished by simply using waitForNotVisible and/or waitForSelectOptions.

```
1. selenium.waitForNotVisible("loading_county_drop_down");
2. selenium.waitForSelectOptions("county","regexpi:Amador County");
```

You may wonder why you need to go into the trouble of using complex JavaScript with waitForCondition while you can accomplish the same with one of the other selenium waitFor commands. The additional advantage of using waitForCondition is that you are able to declare a timeout value explicitly whereas the other waitFor commands simply honor the default selenium timeout value.

Other useful variations:

You may have seen many AJAX intensive sites displaying a 'Loading..' or 'Saving..' or 'Processing…' message when a button or a link is clicked. In such cases you may find this to be helpful to wait for the status text to change or disappear.

Assuming that a 'Saving' message is displayed within a div tag when a fictional 'Save' button is clicked in the snippet below:

```
1. selenium.waitForCondition("var value = selenium.getText(\"//div[@class='save-text']\");value != \"Saving\"",
   "20000");
```

In general, .Net sites that use the web form controls, panel controls etc, trigger an asynchronous postback everytime a form control is filled or a .Net tab or panel control is clicked etc. In such cases, I've found the following to be very helpful in waiting for the postback to complete.

```
1. selenium.waitForCondition("selenium.browserbot.getUserWindow().Sys.WebForms.PageRequestManager.
   getInstance().get_isInAsyncPostBack() == false;", "10000");
```

Now for the killer part, for sites that use jQuery, if all you need is to confirm there aren't any active asynchronous requests, then the following does the trick:

```
1. selenium.waitForCondition("selenium.browserbot.getUserWindow().$.active == 0", "10000");
```

As you can see, there are several possible variations to a single command. The possibilities are endless with Selenium.

Happy Testing !!

selenium-implicit-wait

http://code.google.com/p/selenium-implicit-wait/

This plugin allows Selenium IDE to automatically wait until the element is found before executing each command using a locator.
It is designed to :

- **Avoid the user to add waitForElementPresent before click, type, select...**
- **Implement the implicit wait function available with Selenium 2 WebDrivers**
- **Handle Ajax synchronisation issues**

An hourglass button is added to SeleniumIDE to enable/disable the implicitWaitLocator function and also provides 2 new script commands. The first one is `setImplicitWaitLocator` which has the same purpose as the hourglass button and the second one is `setImplicitWaitCondition` which set a JavaScript condition that need to be true to execute each command.

An error is raised when an element is not found and when the timeout defined in Selenium IDE Options is reached.

Release status

- Beta. Feel free to report defects or wanted feature : http://code.google.com/p/selenium-implicit-wait/issues/list

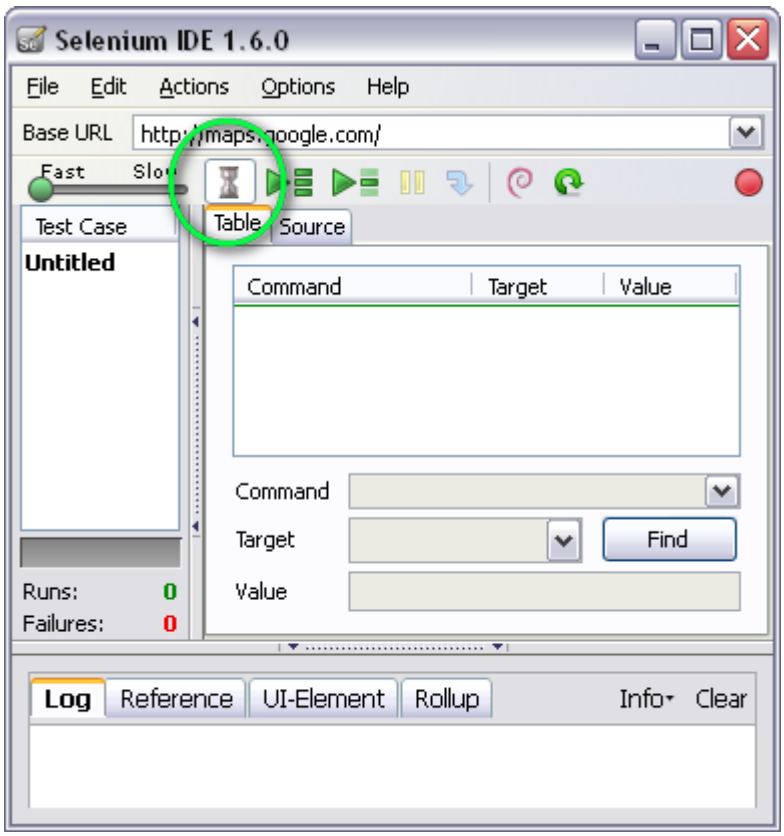
Minimum Requirements

- Firefox 8
- Selenium IDE 1.1.0 http://seleniumhq.org/download/

How to use in SeleniumIDE ?

- Download and install the plugin through Firefox : SeleniumImplicitWait-x.x.x.xpi
- Restart Firefox
- Launch Selenium IDE
- Open or record a script
- Click on the hourglass button to activate the implicit wait function
- Run the script en enjoy!

Screen Capture



How to run exported Html test suites using implicit wait commands ?

- Download the latest selenium-server-standalone-x.x.x.jar [Download page]
- Download and extract the user-extensions file SeleniumImplicitWait-user-extensions-x.x.x.zip
- In Selenium IDE, save tour test suite as "c:\testsuite.html"
- Start the Selenium Server and run the html test suite :

```
java -jar selenium-server-standalone-x.x.x.jar -userExtensions
"c:\user-extensions.js" -htmlSuite "*iexplore"
"http://www.google.com" "c:\testsuite.html" "c:\results.html"
```

How to remotely invoke implicit wait commands ?

- Download the latest selenium-server-standalone-x.x.x.jar [Download page]
- Download and extract the user-extensions file SeleniumImplicitWait-user-extensions-x.x.x.zip
- Start the Selenium Server :

```
java -jar selenium-server-standalone-x.x.x.jar -userExtensions
"c:\user-extensions.js"
```

- Start the browser and invoke implicit wait commands :

```
HttpCommandProcessor cp = new  HttpCommandProcessor("localhost",
4444, "*ieplore", "http://www.google.com/");
DefaultSelenium selenium =new  DefaultSelenium(cp);
selenium.start();
selenium.open("/");
cp.doCommand("setImplicitWaitLocator", new String[] {"10000",});
cp.doCommand("setImplicitWaitCondition", new String[]
{"10000","1==1"});
...
selenium.close();
```

New Selenium commands

- setImplicitWaitLocator | timeout | |

Waits until each locator is found before executing each command.
Ex: setImplicitWaitCondition | 5000 |

- setImplicitWaitCondition | timeout | condition_js |

Waits until the condition is true before executing each command.
Examples using Ajax libraries to wait the end of the transaction :

```
  setImplicitWaitCondition | 5000 | window.dojo?
window.dojo.io.XMLHTTPTransport.inFlight.length==0 : true |
  setImplicitWaitCondition | 5000 | window.Ajax?
window.Ajax.activeRequestCount==0 : true |
  setImplicitWaitCondition | 5000 | window.tapestry ?
```

```
window.tapestry.isServingRequests()==false : true |
  setImplicitWaitCondition | 5000 | window.jQuery ?
window.jQuery.active==0 : true |
  setImplicitWaitCondition | 5000 | window.Sys ?
window.Sys.WebForms.PageRequestManager.getInstance().get_isInAsyncPos
tBack()==false : true |
```

Tested environments

- Firefox 10.0.2 / Selenium IDE 1.6.0

Release note

- 1.0.10 - Added user-extensions for Selenium Server, separated locator and condition timeout
- 1.0.8 - Added setImplicitWaitLocator and setImplicitWaitCondition functions
- 1.0.7 - first release

使用 JQuery 定位元素

http://www.51testing.com/?uid-174712-action-viewspace-itemid-808032
http://www.muranosoft.com/Outsourcingblog/How-To-Use-JQuery-Instead-Of-XPath-Locators-In-Selenium-Testing-Framework.aspx

Selenium server 本身不支持对 jquery 的元素定位，如果需要这个能力那么就需要进行定制。网上找了很多这方面的资料很多试过了都不 work，今天在一个有经验的同事的帮助一下。终于把这个功能加进去了。具体方法如下：

******注意，我用的是 selenium-server.jar v2.19.0******

- 从 JQuery 网站下载 jquery.min.js 文件备用。可以用 jquery-latest.min.js，不过别忘了改名哦!
- 用{code}jar xf selenium-server.jar {code}来解压缩。
- 将准备好的 jquery.min.js 文件 拷贝到 core/lib/目录下。

- 编辑 core/RemoteRunner.html，在 head 部分加入对 jquery.min.js 的引用。如(绿色)：

[code] <script. language="JavaScript" type="text/javascript" src="lib/prototype.js"></script>
<script. language="JavaScript" type="text/javascript" src="lib/jquery.min.js"></script>
<script. language="JavaScript" type="text/javascript" src="lib/sizzle.js"></script> [/code]

- 编辑 core/scripts/selenium-remoterunner.js，并添加如下代码(绿色):

{code} selenium = Selenium.createForWindow(testAppWindow, proxyInjectionMode);
var jqueryLocator =
" var loc = locator; " +
" var attr = null; " +
" var isattr = false; " +
" var inx = locator.lastIndexOf('@'); " +
"" +
" if (inx != -1) { " +
" loc = locator.substring(0, inx); " +
" attr = locator.substring(inx + 1); " +
" isattr = true " +
" } " +
"" +
" var selectors = loc.split('<'); " +
" var found = jQuery(inDocument); " +
"" +
" for (var i = 0; i < selectors.length; i++) { " +
" if (i > 0) {" +
" found = jQuery(found.parents()[0]); " +
" } " +
"" +
" if (jQuery.trim(selectors[i]) != '') {" +
" found = found.find(selectors[i]); " +
" } " +
" }" +
"" +
" if (found.length > 0) { " +
" if (isattr) { " +
" return found[0].getAttributeNode(attr); " +
" } " +
" else { " +
" return found[0]; " +
" } " +
" } " +

```
"    else { " +
"        return null; " +
"    } "
    ;
    selenium.doAddLocationStrategy("jquery", jqueryLocator);
if (runOptions.getBaseUrl()) {
        selenium.browserbot.baseUrl = runOptions.getBaseUrl();
```

{code}

- 讲编辑好的文件更新到 selenium-server.jar 包里。操作如下：

{code}

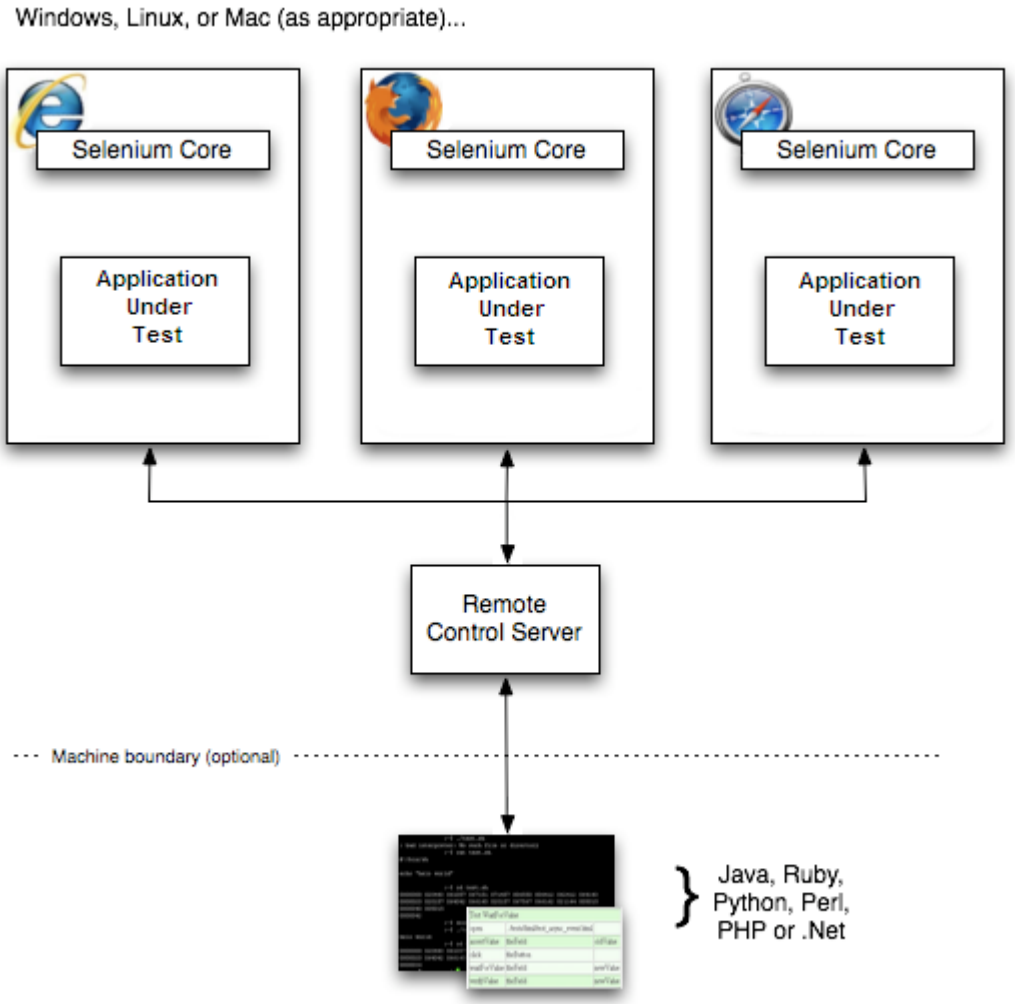
```
jar -uf selenium-server.jar core/lib/jquery.min.js
jar -uf selenium-server.jar core/RemoteRunner.html
jar -uf selenium-server.jar core/scripts/selenium-remoterunner.js
```

{code}

这样就可以了，用新的包来做 selenium server 就可以在 step 里边用 jquery 来定位元素了。例如：

{code}@selenium.click "jquery=a.edit-exception-workflow:nth(0)"{code}

Selenium RC



Selenium Server (formerly the Selenium RC Server)

The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium Webdriver ones. The 2.x server is a drop-in replacement for the old Selenium RC server and is designed to be backwards compatible with your existing infrastructure.

Download version 2.25.0

To use the Selenium Server in a Grid configuration see the wiki page.

Installing Selenium Server

The Selenium RC server is simply a Java *jar* file (*selenium-server-standalone-<version-number>.jar*), which doesn't require any special installation. Just downloading the zip file and extracting the server in the desired directory is sufficient.

Running Selenium Server

Before starting any tests you must start the server. Go to the directory where Selenium RC's server is located and run the following from a command-line console.

java -jar selenium-server-standalone-<version-number>.jar
This can be simplified by creating a batch or shell executable file (.bat on Windows and .sh on Linux) containing the command above. Then make a shortcut to that executable on your desktop and simply double-click the icon to start the server.

For the server to run you'll need Java installed and the PATH environment variable correctly configured to run it from the console. You can check that you have Java correctly installed by running the following on a console.

java -version
If you get a version number (which needs to be 1.5 or later), you're ready to start using Selenium RC.

服务器选项（Server Options）

When the server is launched, command line options can be used to change the default server behaviour.

Recall, the server is started by running the following.

```
$ java -jar selenium-server-standalone-<version-number>.jar
```

To see the list of options, run the server with the `-h` option.

```
$ java -jar selenium-server-standalone-<version-number>.jar -h
```

You'll see a list of all the options you can use with the server and a brief description of each. The provided descriptions will not always be enough, so we've provided explanations for some of the more important options.

Proxy Configuration

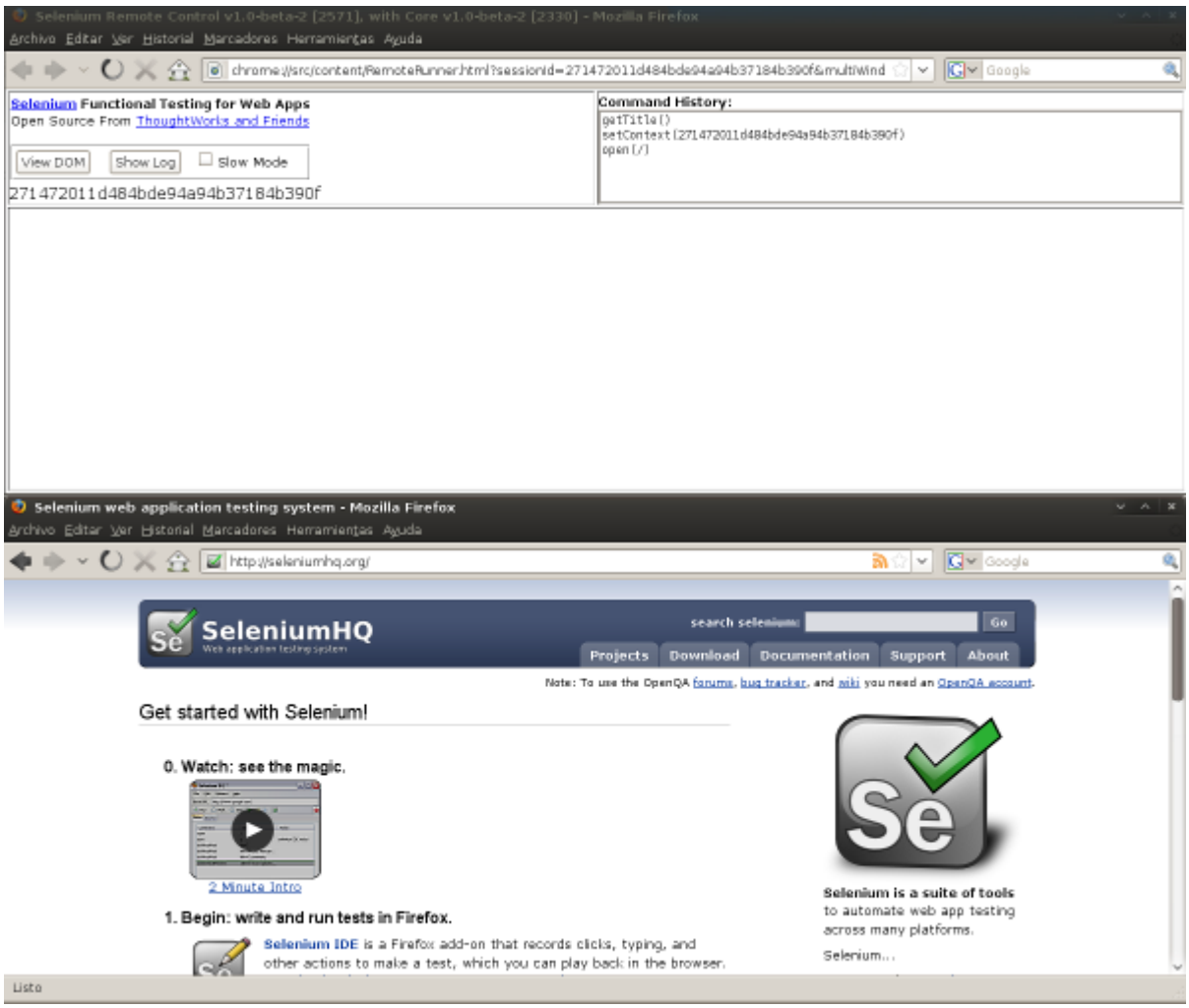
If your AUT is behind an HTTP proxy which requires authentication then you should configure http.proxyHost, http.proxyPort, http.proxyUser and http.proxyPassword using the following command.

```
$ java -jar selenium-server-standalone-<version-number>.jar
-Dhttp.proxyHost=proxy.com -Dhttp.proxyPort=8080
-Dhttp.proxyUser=username -Dhttp.proxyPassword=password
```

Multi-Window Mode

If you are using Selenium 1.0 you can probably skip this section, since multiwindow mode is the default behavior. However, prior to version 1.0, Selenium by default ran the application under test in a sub frame as shown here.

Some applications didn't run correctly in a sub frame, and needed to be loaded into the top frame of the window. The multi-window mode option allowed the AUT to run in a separate window rather than in the default frame where it could then have the top frame it required.



For older versions of Selenium you must specify multiwindow mode explicitly with the following option:

```
-multiwindow
```

As of Selenium RC 1.0, if you want to run your test within a single frame (i.e. using the standard for earlier Selenium versions) you can state this to the Selenium Server using the option

```
-singlewindow
```

Specifying the Firefox Profile

Firefox will not run two instances simultaneously unless you specify a separate profile for each instance. Selenium RC 1.0 and later runs in a separate profile automatically, so if you are using Selenium 1.0, you can probably skip this section. However, if you're using an older version of Selenium or if you need to use a specific profile for your

tests (such as adding an https certificate or having some addons installed), you will need to explicitly specify the profile.

First, to create a separate Firefox profile, follow this procedure. Open the Windows Start menu, select "Run", then type and enter one of the following:

```
firefox.exe -profilemanager
firefox.exe -P
```

Create the new profile using the dialog. Then when you run Selenium Server, tell it to use this new Firefox profile with the server command-line option *-firefoxProfileTemplate* and specify the path to the profile using its filename and directory path.

```
-firefoxProfileTemplate "path to the profile"
```

Warning

Be sure to put your profile in a new folder separate from the default!!! The Firefox profile manager tool will delete all files in a folder if you delete a profile, regardless of whether they are profile files or not.

More information about Firefox profiles can be found in Mozilla's Knowledge Base

Run Selenese Directly Within the Server Using -htmlSuite

You can run Selenese html files directly within the Selenium Server by passing the html file to the server's command line. For instance:

```
java -jar selenium-server-standalone-<version-number>.jar -htmlSuite
"*firefox"
"http://www.google.com" "c:\absolute\path\to\my\HTMLSuite.html"
"c:\absolute\path\to\my\results.html"
```

This will automatically launch your HTML suite, run all the tests and save a nice HTML report with the results.

Note

When using this option, the server will start the tests and wait for a specified number of seconds for the test to complete; if the test

doesn't complete within that amount of time, the command will exit with a non-zero exit code and no results file will be generated.

This command line is very long so be careful when you type it. Note this requires you to pass in an HTML Selenese suite, not a single test. Also be aware the -htmlSuite option is incompatible with `-interactive` You cannot run both at the same time.

Selenium Server Logging

Server-Side Logs

When launching selenium server the **-log** option can be used to record valuable debugging information reported by the Selenium Server to a text file.

```
java -jar selenium-server-standalone-<version-number>.jar -log selenium.log
```

This log file is more verbose than the standard console logs (it includes DEBUG level logging messages). The log file also includes the logger name, and the ID number of the thread that logged the message. For example:

```
20:44:25 DEBUG [12] org.openqa.selenium.server.SeleniumDriverResourceHandler - Browser 465828/:top frame1 posted START NEW
```

The message format is

```
TIMESTAMP(HH:mm:ss) LEVEL [THREAD] LOGGER - MESSAGE
```

This message may be multiline.

Browser-Side Logs

JavaScript on the browser side (Selenium Core) also logs important messages; in many cases, these can be more useful to the end-user than the regular Selenium Server logs. To access browser-side logs, pass the **-browserSideLog** argument to the Selenium Server.

```
java -jar selenium-server-standalone-<version-number>.jar -browserSideLog
```

-browserSideLog must be combined with the **-log** argument, to log browserSideLogs (as well as all other DEBUG level logging messages) to a file.

Specifying the Path to a Specific Browser

You can specify to Selenium RC a path to a specific browser. This is useful if you have different versions of the same browser and you wish to use a specific one. Also, this is used to allow your tests to run against a browser not directly supported by Selenium RC. When specifying the run mode, use the *custom specifier followed by the full path to the browser's executable:

```
*custom <path to browser>
```

Handling HTTPS and Security Popups

Many applications switch from using HTTP to HTTPS when they need to send encrypted information such as passwords or credit card information. This is common with many of today's web applications. Selenium RC supports this.

To ensure the HTTPS site is genuine, the browser will need a security certificate. Otherwise, when the browser accesses the AUT using HTTPS, it will assume that application is not 'trusted'. When this occurs the browser displays security popups, and these popups cannot be closed using Selenium RC.

When dealing with HTTPS in a Selenium RC test, you must use a run mode that supports this and handles the security certificate for you. You specify the run mode when your test program initializes Selenium.

In Selenium RC 1.0 beta 2 and later use *firefox or *iexplore for the run mode. In earlier versions, including Selenium RC 1.0 beta 1, use *chrome or *iehta, for the run mode. Using these run modes, you will not need to install any special security certificates; Selenium RC will handle it for you.

In version 1.0 the run modes *firefox or *iexplore are recommended. However, there are additional run modes of *iexploreproxy and *firefoxproxy. These are provided for backwards compatibility only,

and should not be used unless required by legacy test programs. Their use will present limitations with security certificate handling and with the running of multiple windows if your application opens additional browser windows.

In earlier versions of Selenium RC, *chrome or *iehta were the run modes that supported HTTPS and the handling of security popups. These were considered ???experimental modes although they became quite stable and many people used them. If you are using Selenium 1.0 you do not need, and should not use, these older run modes.

Security Certificates Explained

Normally, your browser will trust the application you are testing by installing a security certificate which you already own. You can check this in your browser's options or Internet properties (if you don't know your AUT's security certificate ask your system administrator). When Selenium loads your browser it injects code to intercept messages between the browser and the server. The browser now thinks untrusted software is trying to look like your application. It responds by alerting you with popup messages.

To get around this, Selenium RC, (again when using a run mode that support this) will install its own security certificate, temporarily, to your client machine in a place where the browser can access it. This tricks the browser into thinking it's accessing a site different from your AUT and effectively suppresses the popups.

Another method used with earlier versions of Selenium was to install the Cybervillians security certificate provided with your Selenium installation. Most users should no longer need to do this however; if you are running Selenium RC in proxy injection mode, you may need to explicitly install this security certificate.

Java 与 Selenium

Using the Java Client Driver

- Download Selenium java client driver zip from the SeleniumHQ downloads page.
- Extract selenium-java-<version-number>.jar file
- Open your desired Java IDE (Eclipse, NetBeans, IntelliJ, Netweaver, etc.)

- Create a java project.
- Add the selenium-java-<version-number>.jar files to your project as references.
- Add to your project classpath the file selenium-java-<version-number>.jar.
- From Selenium-IDE, export a script to a Java file and include it in your Java project, or write your Selenium test in Java using the selenium-java-client API. The API is presented later in this chapter. You can either use JUnit, or TestNg to run your test, or you can write your own simple main() program. These concepts are explained later in this section.
- Run Selenium server from the console.
- Execute your test from the Java IDE or from the command-line.

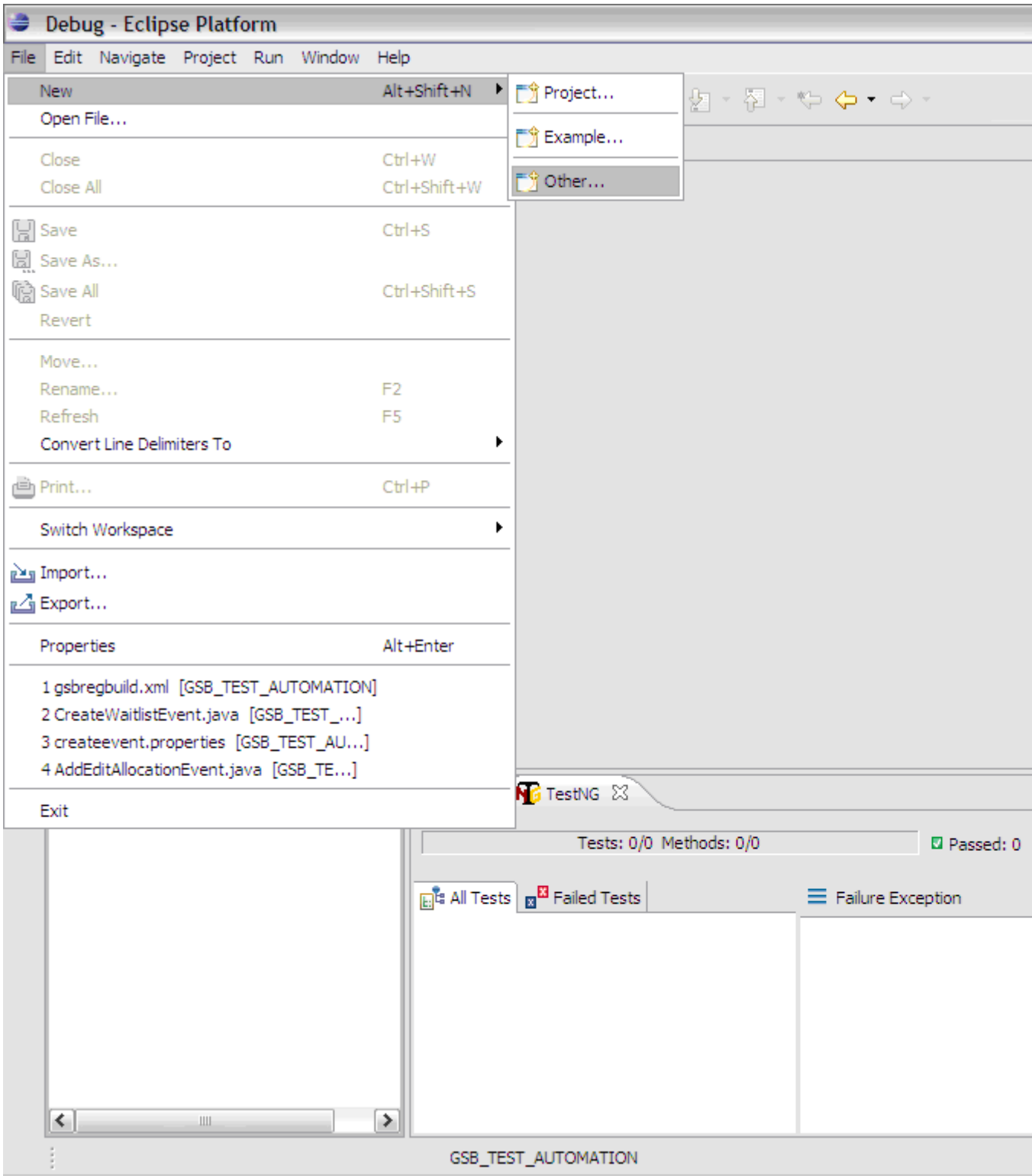
For details on Java test project configuration, see the Appendix sections *Configuring Selenium RC With Eclipse* and *Configuring Selenium RC With Intellij*.

Configuring Selenium-RC With Eclipse

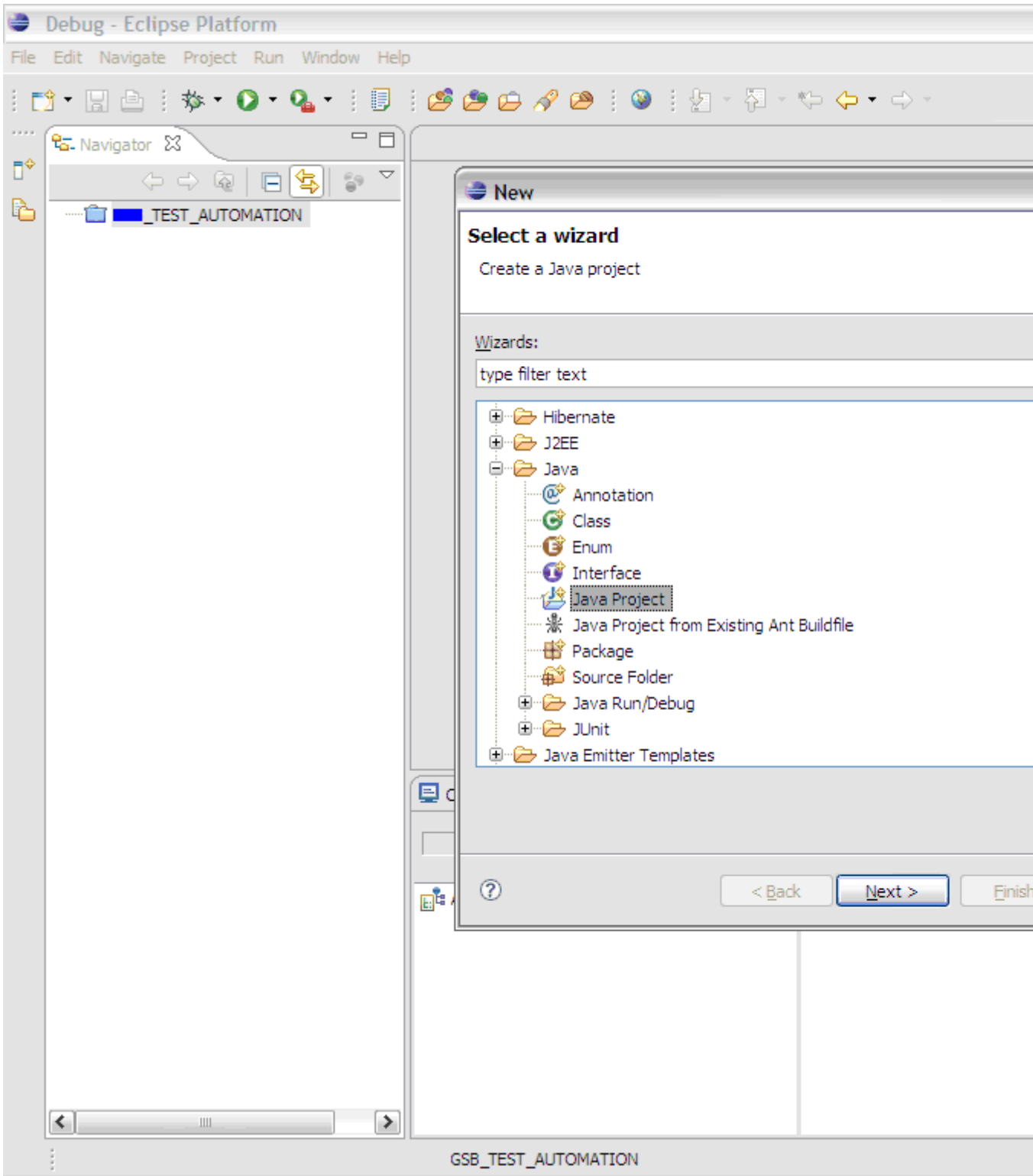
Eclipse is a multi-language software development platform comprising an IDE and a plug-in system to extend it. It is written primarily in Java and is used to develop applications in this language and, by means of the various plug-ins, in other languages as well as C/C++, Cobol, Python, Perl, PHP and more.

Following lines describes configuration of Selenium-RC with Eclipse - Version: 3.3.0. (Europa Release). It should not be too different for higher versions of Eclipse

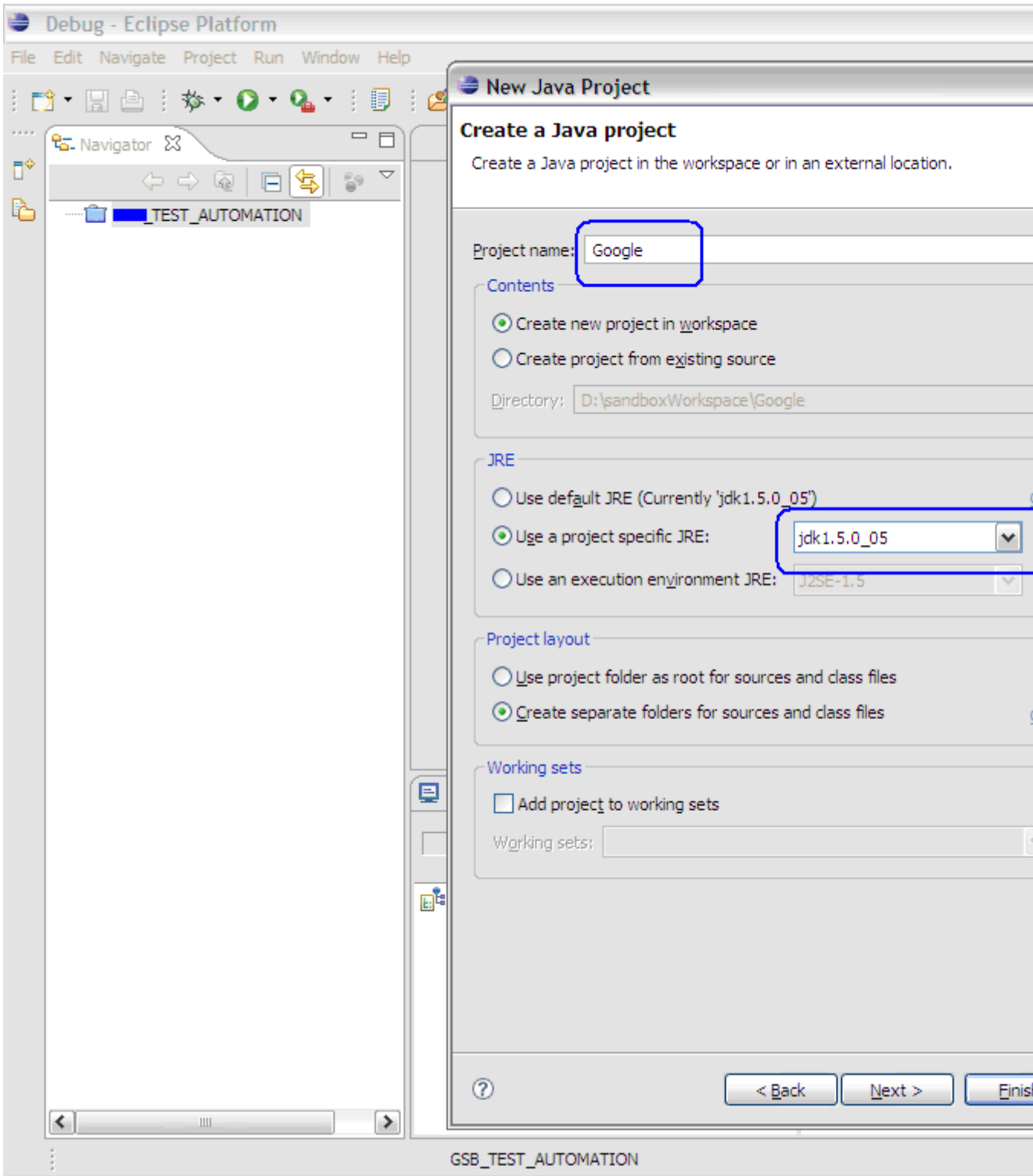
- Launch Eclipse.
- Select File > New > Other.



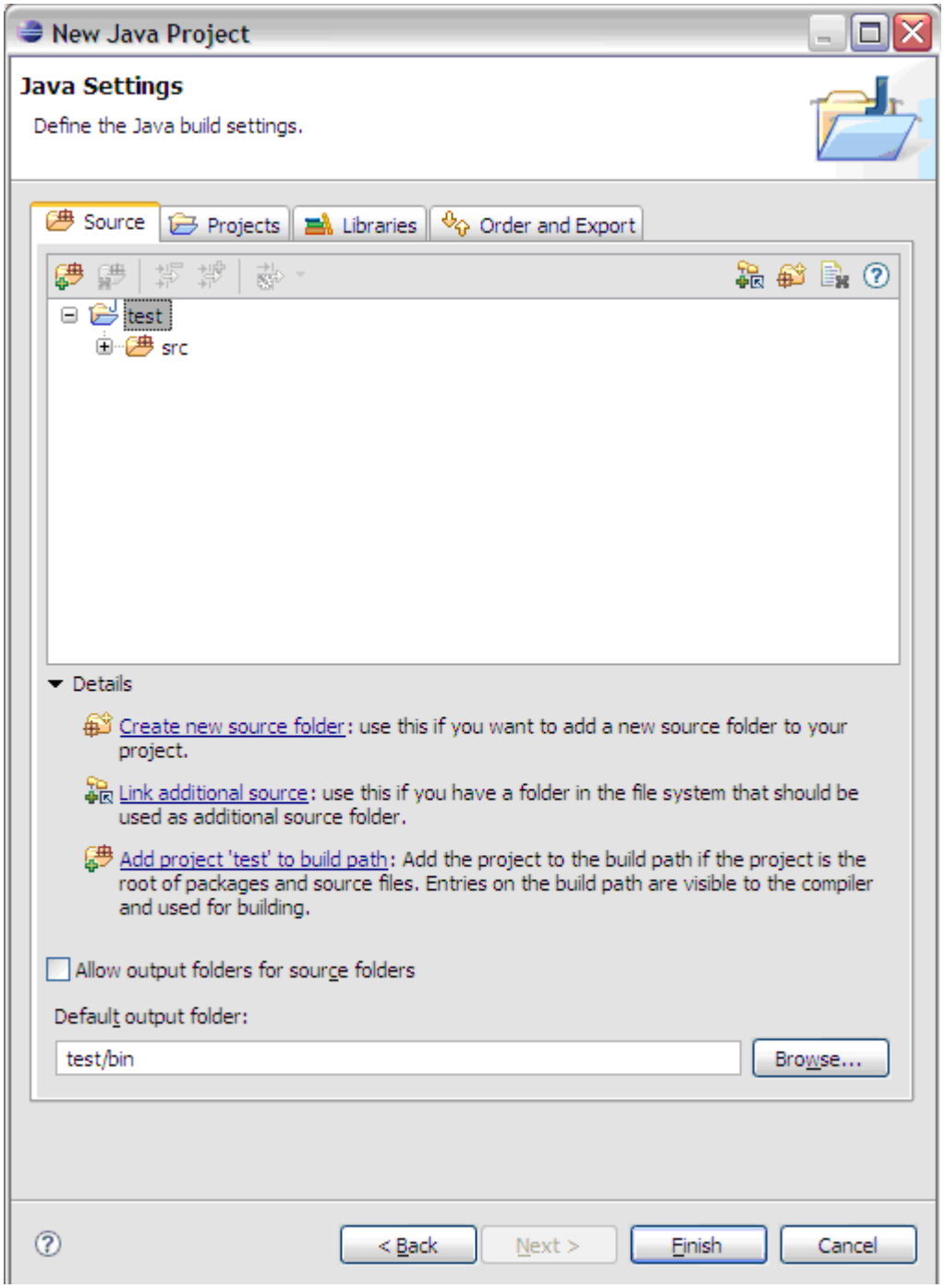
- Java > Java Project > Next



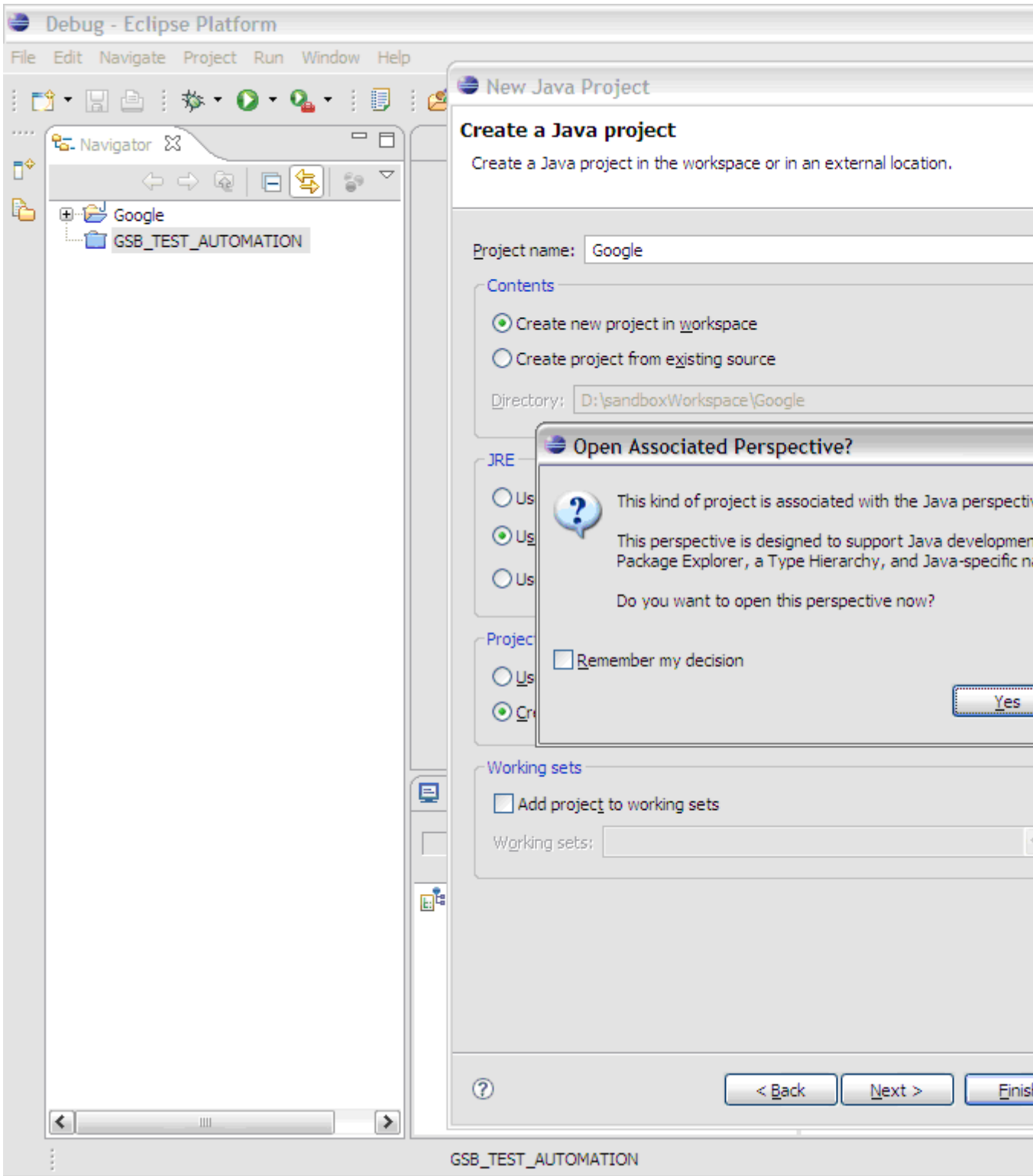
- Provide Name to your project, Select JDK in 'Use a project Specific JRE' option (JDK 1.5 selected in this example) > click Next



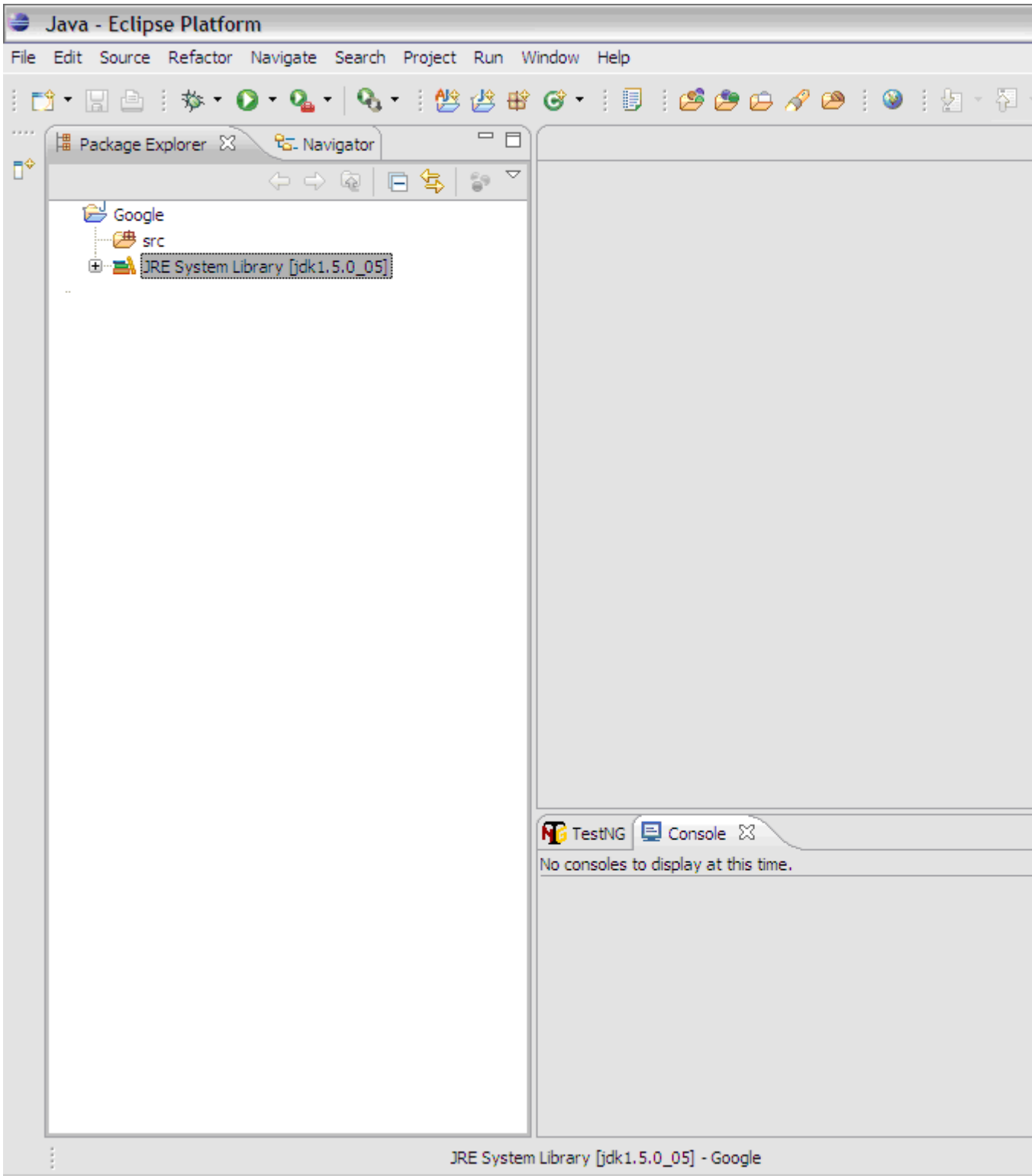
- Keep 'JAVA Settings' intact in next window. Project specific libraries can be added here. (This described in detail in later part of document.)



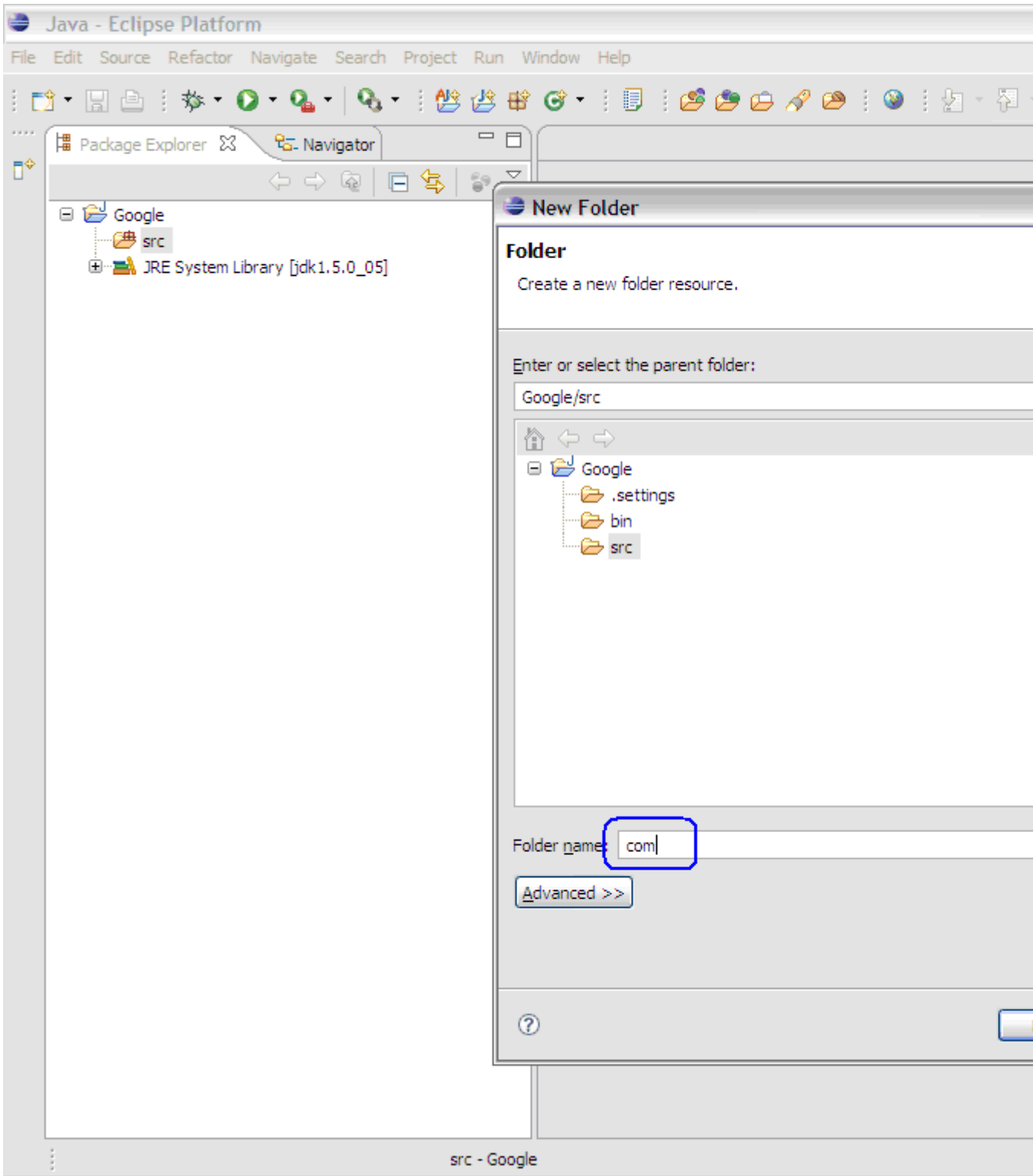
- Click Finish > Click on Yes in Open Associated Perspective pop up window.



This would create Project Google in Package Explorer/Navigator pane.

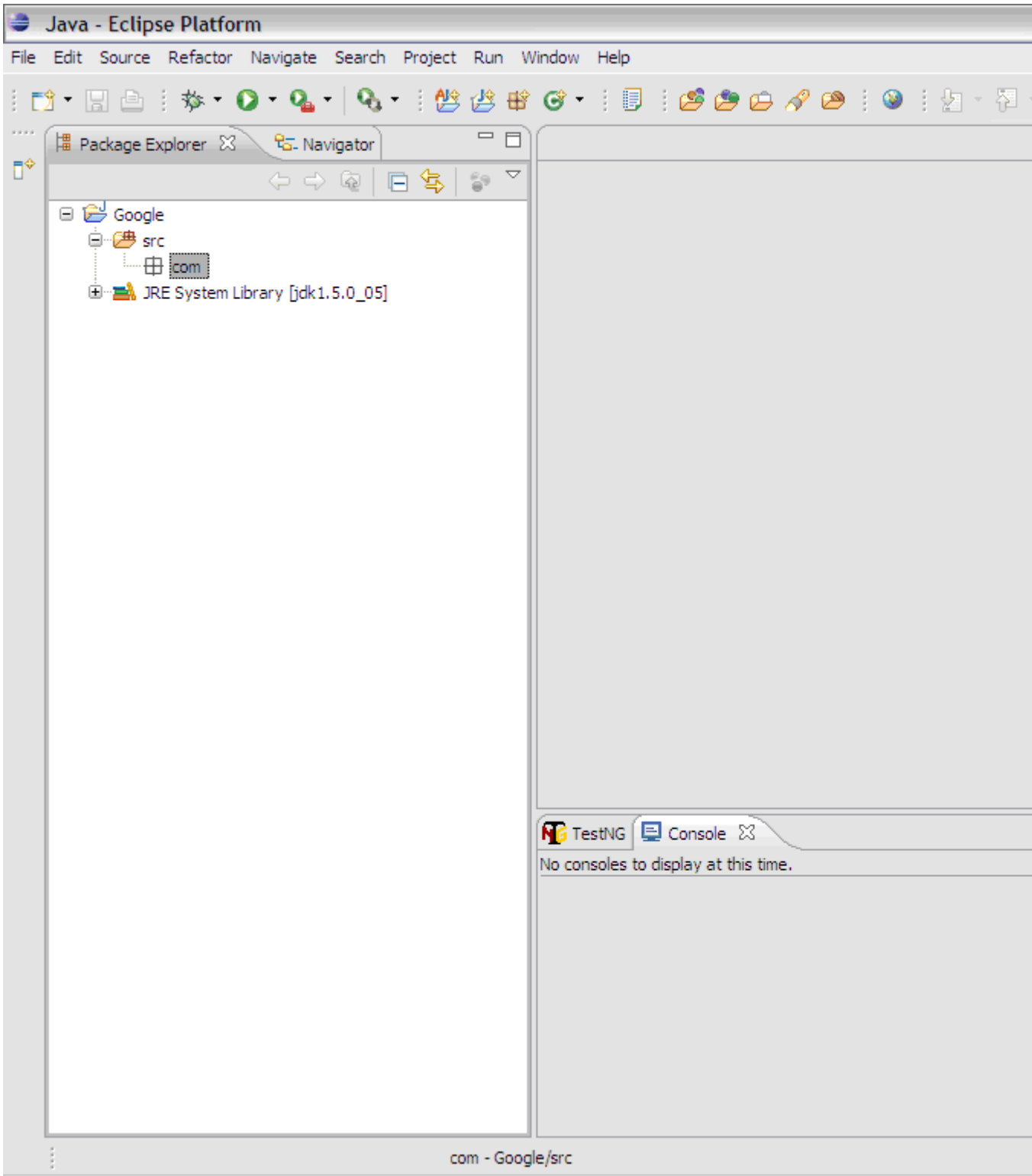


- Right click on src folder and click on New > Folder

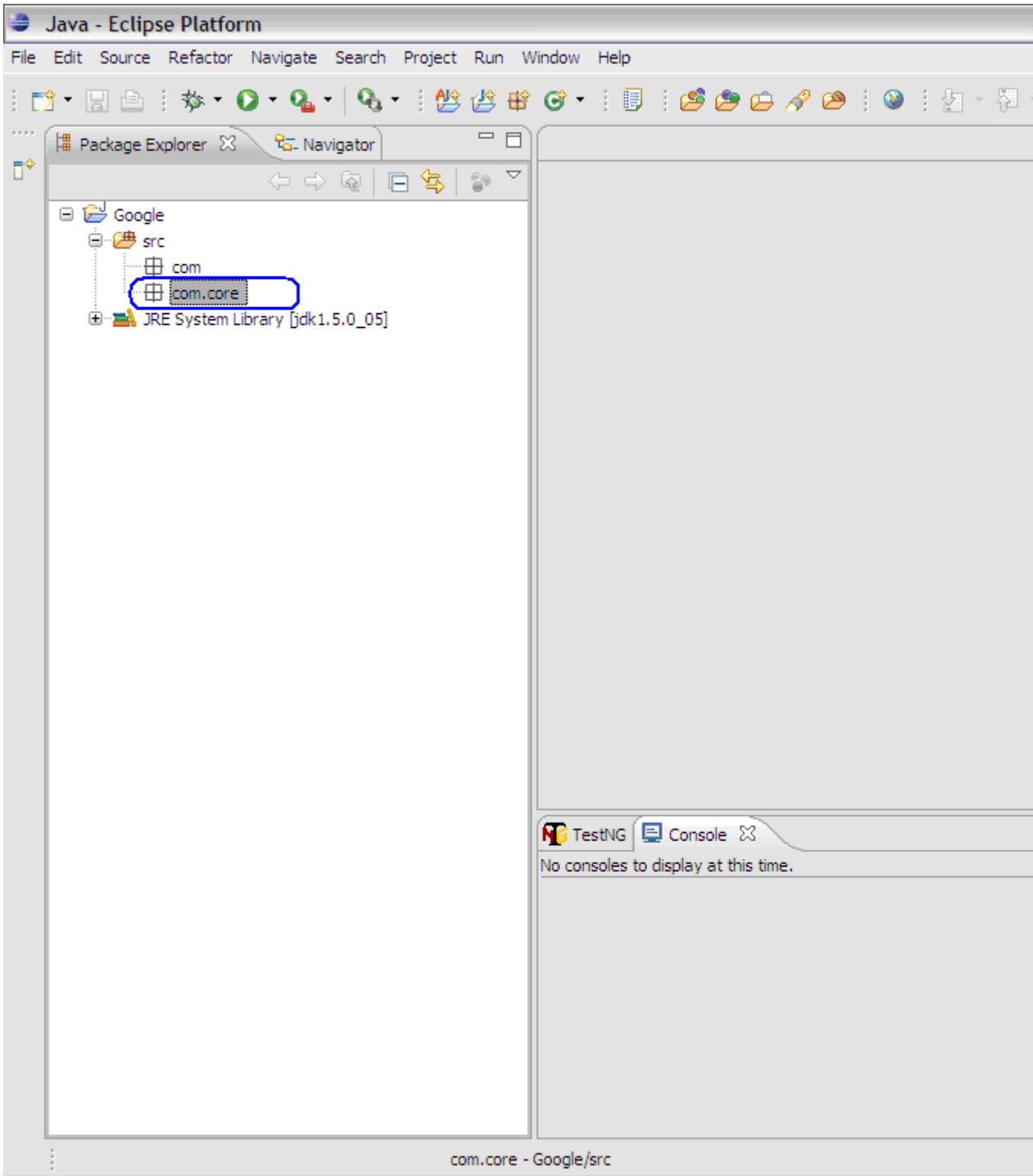


Name this folder as com and click on Finish button.

- This should get com package insider src folder.



- Following the same steps create *core* folder inside *com*

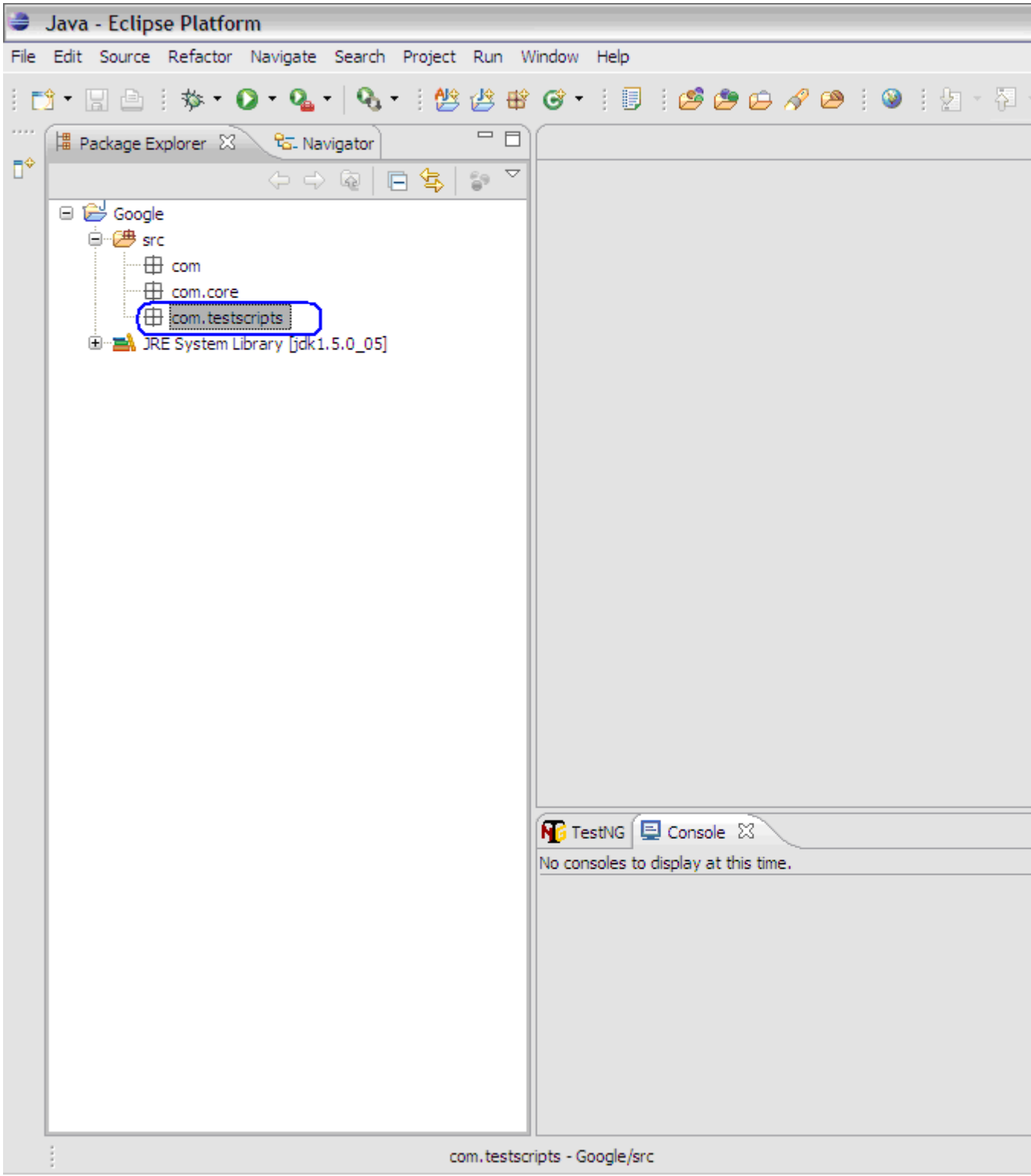


SelTestCase class can be kept inside *core* package.

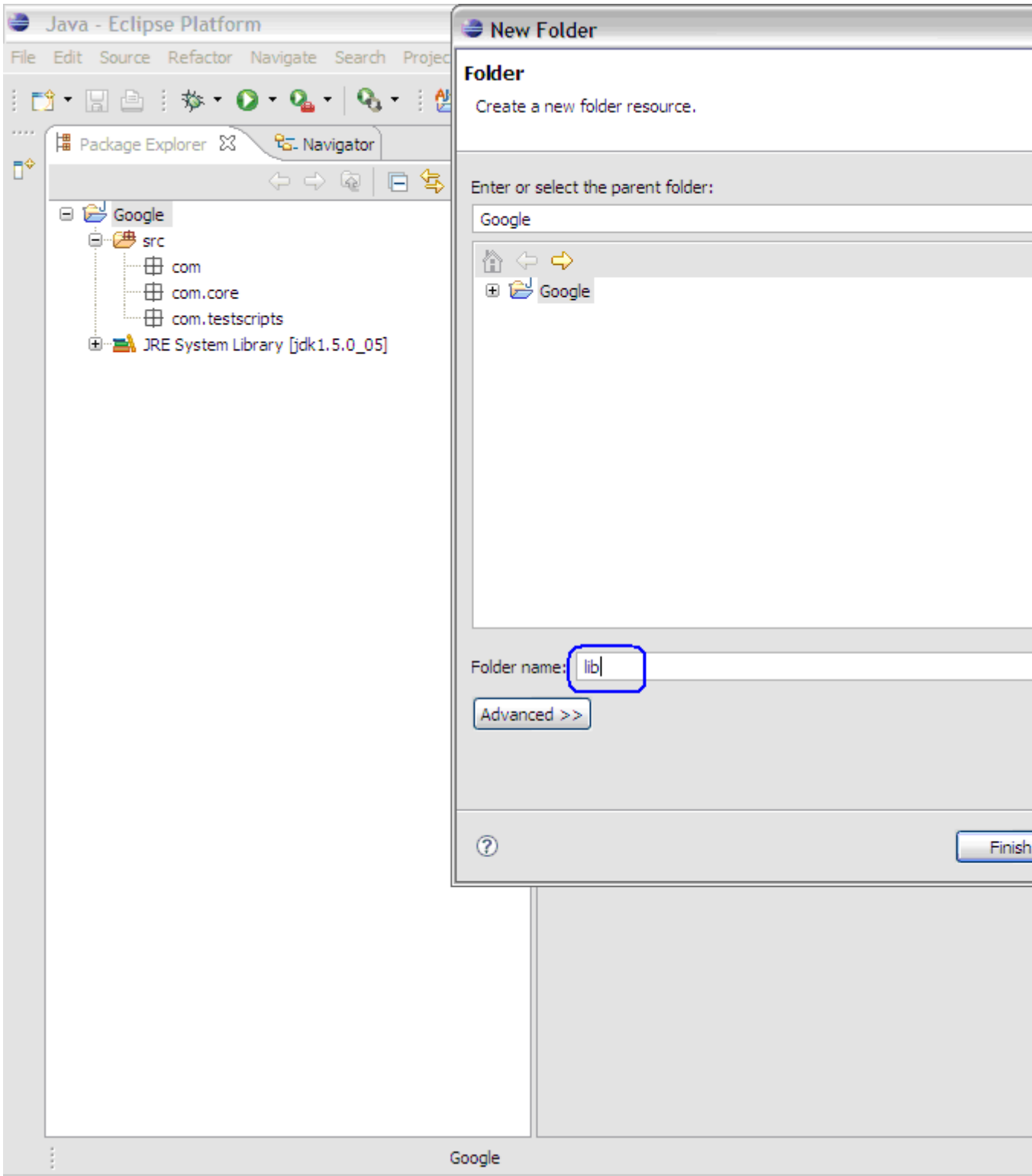
Create one more package inside *src* folder named *testscripts*. This is a place holder for test scripts.

Please notice this is about the organization of project and it entirely depends on individual's choice / organization's standards. Test scripts

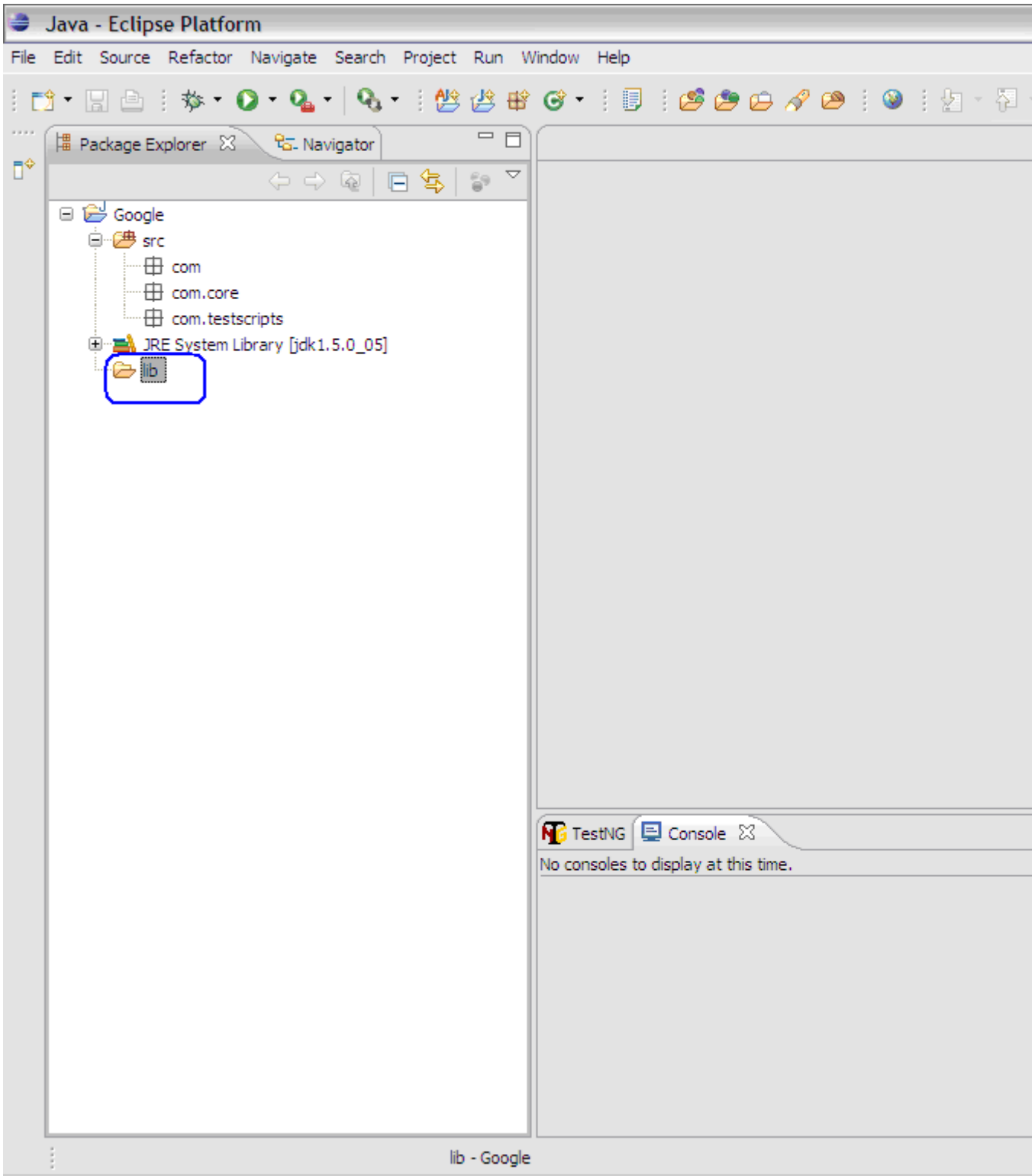
package can further be segregated depending upon the project requirements.



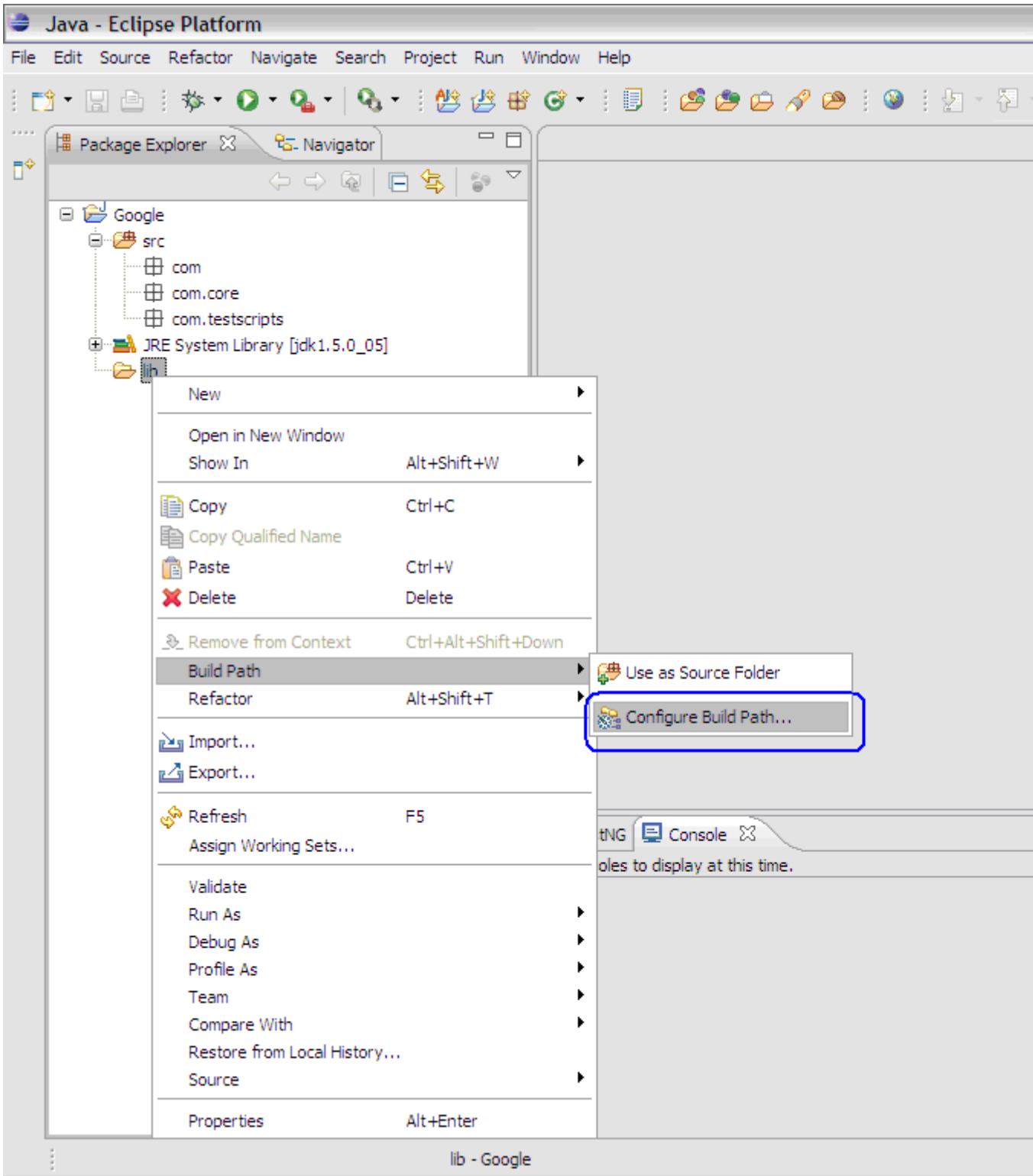
- Create a folder called lib inside project Google. Right click on Project name > New > Folder. This is a place holder for jar files to project (i.e. Selenium client driver, selenium server etc)



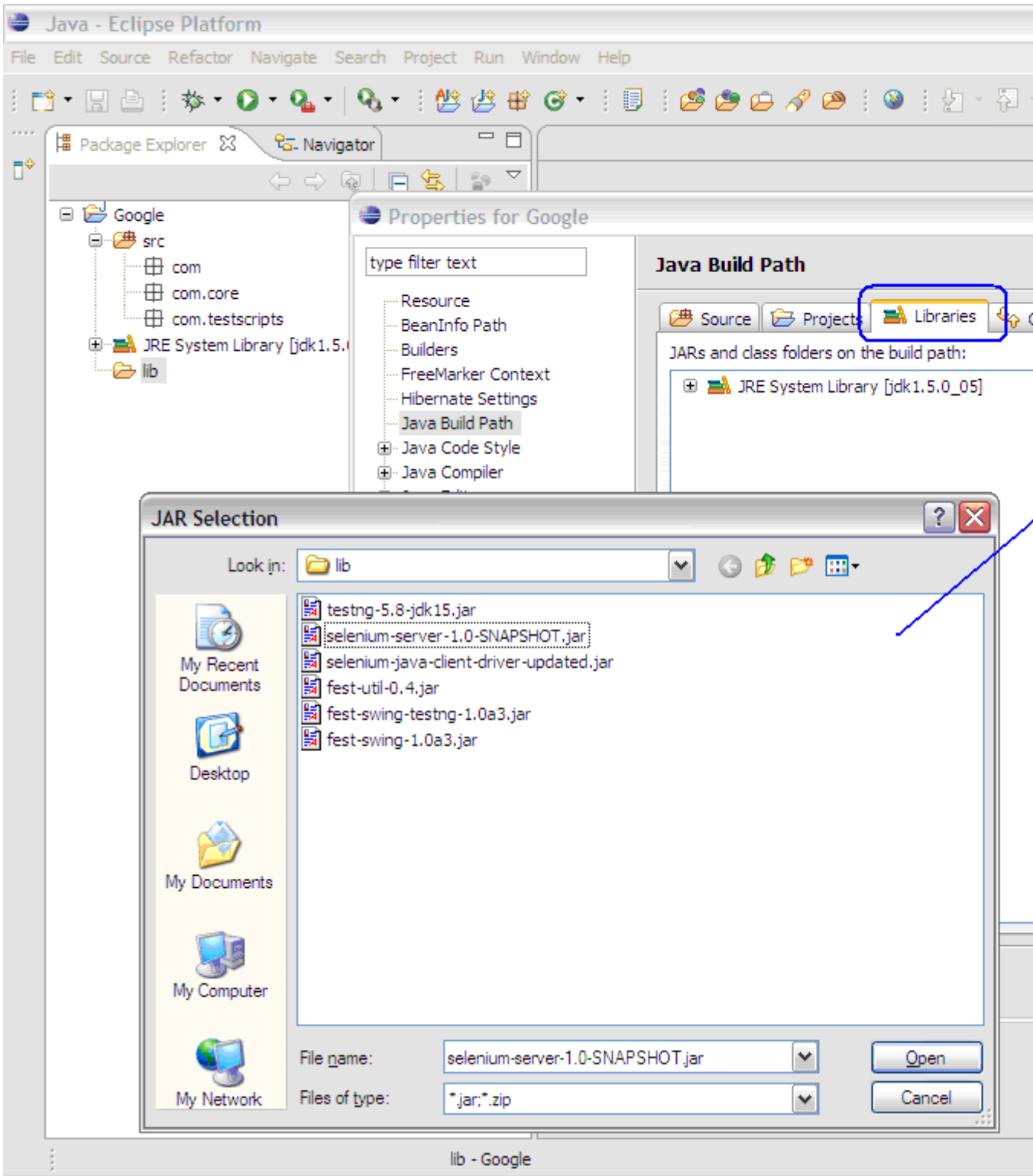
This would create lib folder in Project directory.



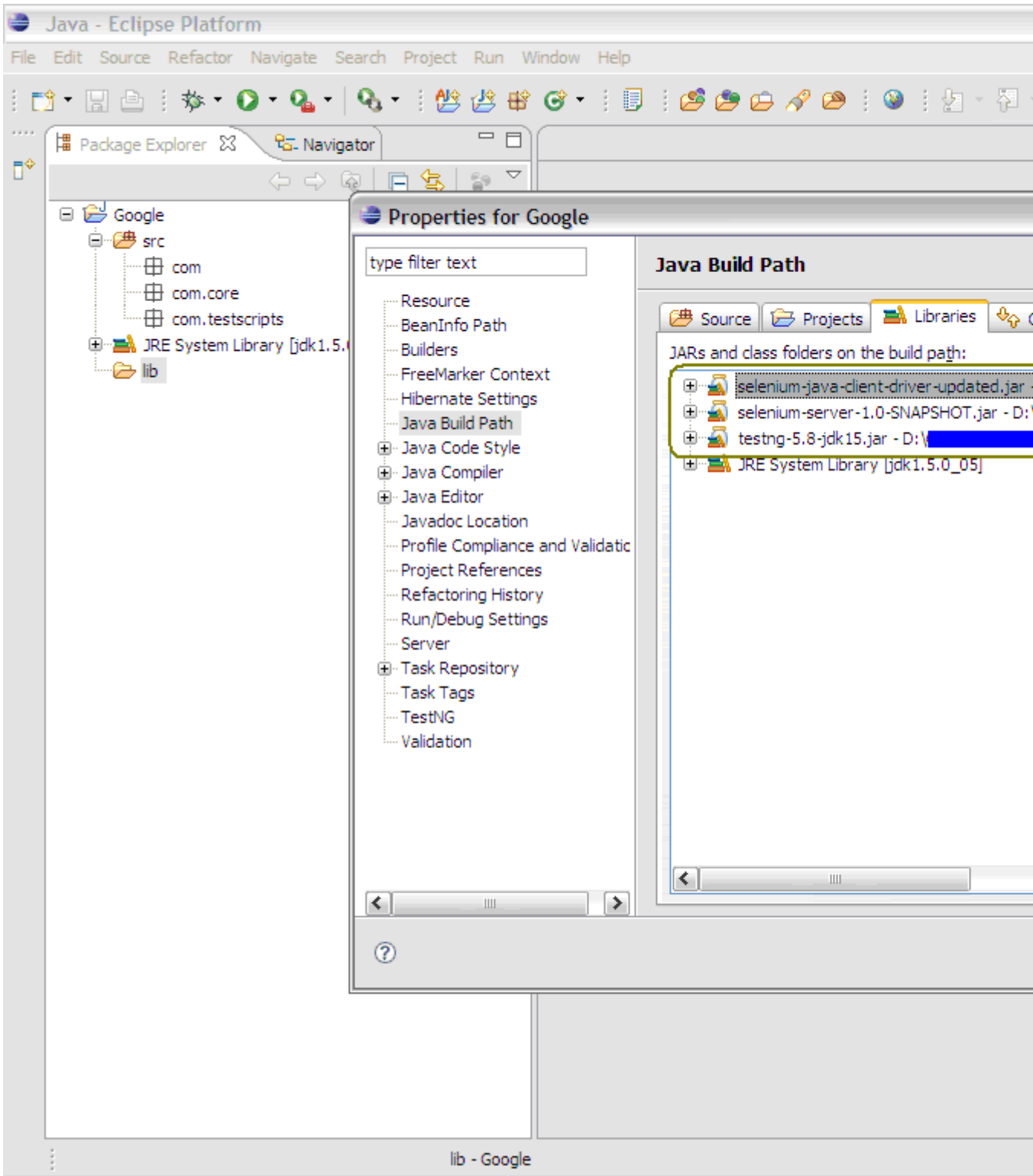
- Right click on *lib* folder > Build Path > Configure build Path



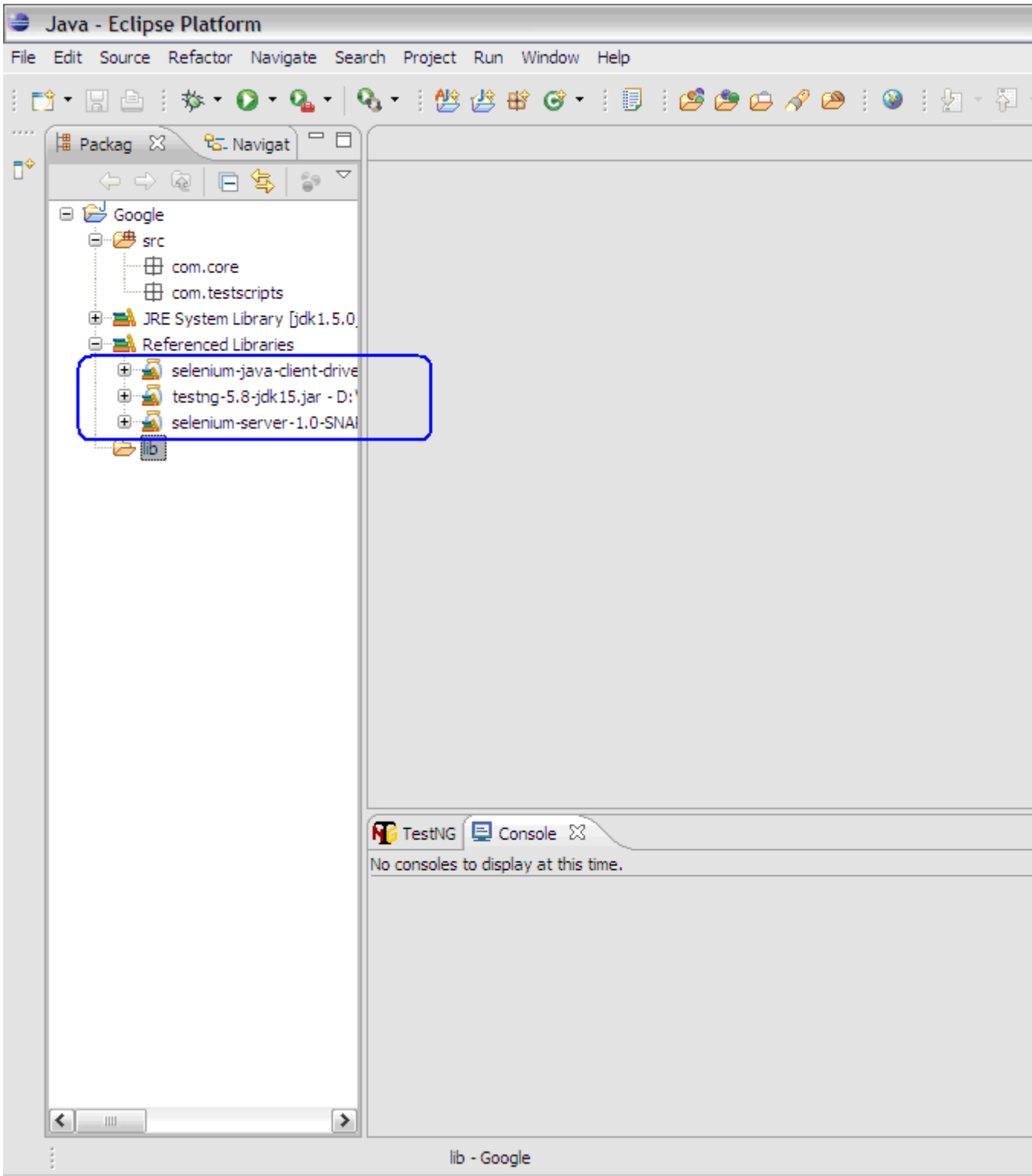
- Under Library tab click on Add External Jars to navigate to directory where jar files are saved. Select the jar files which are to be added and click on Open button.



After having added jar files click on OK button.



Added libraries would appear in Package Explorer as following:



Eclipse+TestNG+Selenium

Eclipse 安装 TestNG 插件：

http://testng.org/doc/download.html

testing.xml:
http://www.blogjava.net/aoxj/archive/2008/03/19/187382.html
https://confluence.sakaiproject.org/display/QA/Selenium+Remote+Control+and+TestNG

http://testng.org/doc/selenium.html
http://www.51testing.com/?uid-375367-action-viewspace-itemid-245315

Modeling your test case

Before writing a test case, you need to know how and what will be validated. Let's use the WordPress "Create New Post" test case.

1. Go to http://demo.opensourcecms.com/wordpress/wp-login.php
2. Enter "admin" in the "Username" field
3. Enter "demo123" in the "Password" field
4. Click on the "Log In" button
5. Verify that the text "Howdy, admin" is present
6. Click on the "Posts" link
7. Click on the "Add New" button
8. Type "Selenium Demo Post" in the title field
9. Click on the "Publish" button
10. Verify that the text "Post published" is present

Considering this scenario, the first thing that comes to mind is creating a long test case that goes through all the steps. This might be a good approach if you are writing a manual test case. However, since we are writing an automated test, we want to write our script as modular as possible to be able to reuse parts of it in future scenarios.

This is how I would break down the test:

1. Launch the WordPress site
2. Open the Admin Login page
3. Enter valid login data
4. Navigate to the Write Post page
5. Write the post
6. Publish the post
7. Verify that it was actually post

Keep in mind that this is just an example. You are free to model your tests in any way you want, as long as they have business value and will validate your business logic.

Let's see how to do that with actual Java code:

```
@Test(description="Launches the WordPress site")
public void launchSite(){
    selenium.open("");
    selenium.waitForPageToLoad("30000");
    assertEquals(selenium.getTitle(), "Demo | Just another WordPress
site");
}

@Test(description="Navigates to the admin page")
    public void openAdminPage() {
    selenium.open("wp-admin");
    selenium.waitForPageToLoad("30000");
    assertEquals(selenium.getTitle(), "Demo 鈥 Log In");
}

@Test(description="Enters valid login data")
    public void loginAsAdmin() {
    selenium.type("user_login", "admin");
    selenium.type("user_pass", "demo123");
    selenium.click("wp-submit");
    selenium.waitForPageToLoad("30000");
    assertTrue(selenium.isTextPresent("Howdy, admin"));
}

@Test(description="Navigates to the New Post screen")
public void navigateNewPost() {
    selenium.click("//a[contains(text(),'Posts')]/following::a[contains(text(),'Add
New')][1]");
    selenium.waitForPageToLoad("30000");
    assertTrue(selenium.isTextPresent("Add New Post"));
}

@Test(description="Writes the new post")
public void writeBlogPost() {
    selenium.type("title", "New Blog Post");
```

```
    selenium.click("edButtonHTML");
    selenium.type("content", "This is a new post");
    //TODO:Assert
}

@Test(description="Publishes the post")
public void publishBlogPost() {
    selenium.click("submitdiv");
    selenium.click("publish");
    selenium.waitForPageToLoad("30000");
    assertTrue(selenium.isTextPresent("Post published."));
}

@Test(description="Verifies the post")
public void verifyBlogPost() {
    selenium.click("//a[contains(text(),'Posts') and
contains(@class,'wp-first-item')]");
    selenium.waitForPageToLoad("30000");
    assertTrue(selenium.isElementPresent("//a[text()='New Blog
Post']"));
}

@Test(description="Logs out")
public void logout() {
    selenium.click("//a[text()='Log Out']");
    //TODO:Assert
}
```

These are the test methods (or steps) we are going to use.

Configuration methods

If you are familiar with unit testing frameworks, you probably know about the setup and teardown methods. TestNG goes beyond that idea and allows you to define methods that will be run after or before your test suites, test groups or test methods. This is very useful for our Selenium tests because you can create a Selenium server and browser instance before you start running your test suite.)

To achieve this, we will use two TestNG annotations: `@BeforeSuite` and `@AfterSuite`:

```
@BeforeSuite(alwaysRun = true)
public void setupBeforeSuite(ITestContext context) {
    String seleniumHost = context.getCurrentXmlTest().getParameter("selenium.host");
    String seleniumPort = context.getCurrentXmlTest().getParameter("selenium.port");
    String seleniumBrowser = context.getCurrentXmlTest().getParameter("selenium.browser");
    String seleniumUrl = context.getCurrentXmlTest().getParameter("selenium.url");

    RemoteControlConfiguration rcc = new RemoteControlConfiguration();
    rcc.setSingleWindow(true);
    rcc.setPort(Integer.parseInt(seleniumPort));

    try {
        server = new SeleniumServer(false, rcc);
        server.boot();
    } catch (Exception e) {
        throw new IllegalStateException("Can't start selenium server", e);
    }

    proc = new HttpCommandProcessor(seleniumHost, Integer.parseInt(seleniumPort),
            seleniumBrowser, seleniumUrl);
    selenium = new DefaultSelenium(proc);
    selenium.start();
}

@AfterSuite(alwaysRun = true)
public void setupAfterSuite() {
    selenium.stop();
    server.stop();
}
```

PS: Did you notice those weird parameters? They are stored in the XML file (we are going to see in the next section) and accessed by a `ITestContext` object, which was injected.

By adding these annotations, the TestNG engine will invoke the configuration methods automatically before/after your test suite (make sure the test methods are annotated with `@Test`), launching the Selenium server and instantiating the Selenium client object only once, reusing the same browser session across the tests.

Creating the XML file

To define the order of the tests, we will have to create an XML file listing the test methods we would like to run. Make sure that the test methods are annotated with `@Test`, or else the TestNG engine will not invoke them.

Before TestNG 5.13.1, you had to use Method Interceptors if you wanted to run the tests in the order defined in the XML file. I have posted my implementation of a Method Interceptor on my Github account. From TestNG 5.13.1+, you can just add the `preserve-order` parameter to your test tag and include the methods you would like to run, reducing unecessary code in your test suite.

Here is the XML file:

```
<suite name="Knorrium.info - Wordpress Demo" verbose="10">
    <parameter name="selenium.host" value="localhost" />
    <parameter name="selenium.port" value="3737" />
    <parameter name="selenium.browser" value="*firefox" />
    <parameter name="selenium.url" value="http://demo.opensourcecms.com/wordpress/" />

    <test name="Write new post" preserve-order="true">
        <classes>
            <class name="test.Wordpress">
                <methods>
                    <include name="launchSite" />
                    <include name="openAdminPage" />
                    <include name="loginAsAdmin" />
                    <include name="navigateNewPost" />
                    <include name="writeBlogPost" />
                    <include name="publishBlogPost" />
                    <include name="verifyBlogPost" />
                </methods>
            </class>
        </classes>
    </test>
```

```
</suite>
```

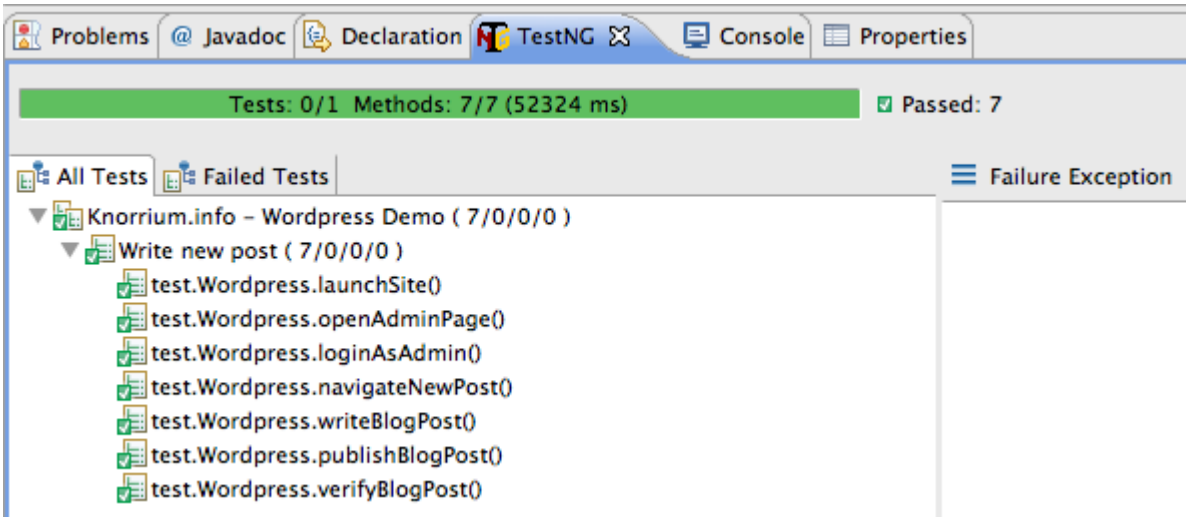
Launching your tests in Eclipse

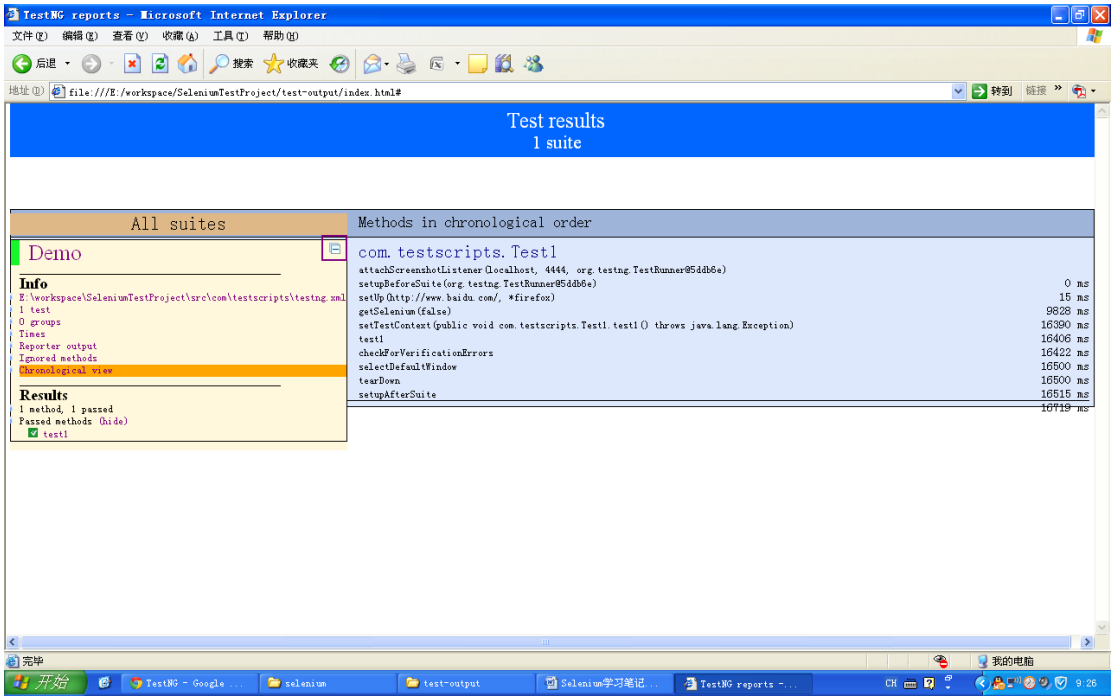
We finished writing our tests, now how can we run them?

You can launch TestNG from the command line, using a Eclipse plugin or even programatically. We are going to use the Eclipse plugin. Follow the steps described on the official TestNG documentation over here

If you installed TestNG correctly, you will see this menu when you right click on the XML file:

Click on "Run as TestNG Suite" and your test will start running. You will then see this nice results tree:





Thinking about the future

If you really want to think about the future of your test suite, I would recommend you to read Adam Goucher's article published on PragPub. He talks about Selenium 2 and the Page Objects Model (a very nice way to model your tests, especially if you use Selenium 2).

Since there are lots of people still using Selenium 1, I'll stick to that for a while, but Selenium 2 will eventually be covered here.

As the number of tests in your test suite grows, you will find that grouping them in different test classes is a good idea. If you do that, you can take advantage of object oriented programming and create a new class named BaseTest (for example), and leave your configuration logic there. That way, every test class must extend the BaseTest class and use static attributes.

```
public class WordPressAdmin extends BaseTest {
@Test
public void test1(){
    selenium.open("");
    //...
}

@Test
public void test2(){
```

```
    selenium.open("");
    //...
}
}
```

This is better than leaving your configuration methods in the test class.

API

API 文档：
http://selenium.googlecode.com/svn/trunk/docs/api/java/index.html

D:\selenium\API 1.0.3

启动浏览器

```
setUp("http://www.google.com/", "*firefox");
```

Each of these examples opens the browser and represents that browser by assigning a "browser instance" to a program variable. This program variable is then used to call methods from the browser. These methods execute the Selenium commands, i.e.
like *open* or *type* or the *verify* commands.

The parameters required when creating the browser instance are:

host

Specifies the IP address of the computer where the server is located. Usually, this is the same machine as where the client is running, so in this case *localhost* is passed. In some clients this is an optional parameter.

port

Specifies the TCP/IP socket where the server is listening waiting for the client to establish a connection. This also is optional in some client drivers.

browser

The browser in which you want to run the tests. This is a required parameter.

url

The base url of the application under test. This is required by all the client

libs and is integral information for starting up the browser-proxy-AUT communication.

Note that some of the client libraries require the browser to be started explicitly by calling its*start()* method.

Running Commands

Once you have the browser initialized and assigned to a variable (generally named "selenium") you can make it run Selenese commands by calling the respective methods from the browser variable. For example, to call the *type* method of the selenium object:

selenium.type("field-id","string to type")
In the background the browser will actually perform a *type* operation, essentially identical to a user typing input into the browser, by using the locator and the string you specified during the method call.

调整和完善脚本（**Adding Some Spice to Your Tests**）

Now we'll get to the whole reason for using Selenium RC, adding programming logic to your tests. It's the same as for any program. Program flow is controlled using condition statements and iteration. In addition you can report progress information using I/O. In this section we'll show some examples of how programming language constructs can be combined with Selenium to solve common testing problems.

You will find as you transition from the simple tests of the existence of page elements to tests of dynamic functionality involving multiple web-pages and varying data that you will require programming logic for verifying expected results. Basically, the Selenium-IDE does not support iteration and standard condition statements. You can do some conditions by embedding javascript in Selenese parameters, however iteration is impossible, and most conditions will be much easier in a programming language. In addition, you may need exception handling for error recovery. For these reasons and others, we have written this section to illustrate the use of common programming techniques to give you greater 'verification power' in your automated testing.

The examples in this section are written in C# and Java, although the code is simple and can be easily adapted to the other supported languages. If you have some basic knowledge of an object-oriented programming language you shouldn't have difficulty understanding this section.

循环（Iteration）

Iteration is one of the most common things people need to do in their tests. For example, you may want to to execute a search multiple times. Or, perhaps for verifying your test results you need to process a "result set" returned from a database.

Using the same Google search example we used earlier, let's check the Selenium search results. This test could use the Selenese:

open	/	
type	q	selenium rc
clickAndWait	btnG	
assertTextPresent	Results * for selenium rc	
type	q	selenium ide
clickAndWait	btnG	
assertTextPresent	Results * for selenium ide	
type	q	selenium grid
clickAndWait	btnG	
assertTextPresent	Results * for selenium grid	

The code has been repeated to run the same steps 3 times. But multiple copies of the same code is not good program practice because it's more work to maintain. By using a programming language, we can iterate over the search results for a more flexible and maintainable solution.

In C#:

```
// Collection of String values.
String[] arr = {"ide", "rc", "grid"};

// Execute loop for each String in array 'arr'.
foreach (String s in arr) {
    sel.open("/");
```

```
    sel.type("q", "selenium " +s);
    sel.click("btnG");
    sel.waitForPageToLoad("30000");
    assertTrue("Expected text: " +s+ " is missing on page."
    , sel.isTextPresent("Results * for selenium " + s));
}
```

条件判断（Condition Statements）

To illustrate using conditions in tests we'll start with an example. A common problem encountered while running Selenium tests occurs when an expected element is not available on page. For example, when running the following line:

```
selenium.type("q", "selenium " +s);
```

If element 'q' is not on the page then an exception is thrown:

```
com.thoughtworks.selenium.SeleniumException: ERROR: Element q not found
```

This can cause your test to abort. For some tests that's what you want. But often that is not desirable as your test script has many other subsequent tests to perform.

A better approach is to first validate whether the element is really present and then take alternatives when it it is not. Let's look at this using Java.

```
// If element is available on page then perform type operation.
if(selenium.isElementPresent("q")) {
    selenium.type("q", "Selenium rc");
} else {
    System.out.printf("Element: " +q+ " is not available on page.")
}
```

The advantage of this approach is to continue with test execution even if some UI elements are not available on page.

调用 JavaScript 脚本

JavaScript comes very handy in exercising an application which is not directly supported by selenium. The **getEval** method of selenium API can be used to execute JavaScript from selenium RC.

Consider an application having check boxes with no static identifiers. In this case one could evaluate JavaScript from selenium RC to get ids of all check boxes and then exercise them.

```
package com.testscripts;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import com.thoughtworks.selenium.*;

public class Test_getEval {

    DefaultSelenium selenium;

    @Before
    public void setUp() throws Exception {
        selenium = new DefaultSelenium("localhost", 4444,
"*firefox","http://www.baidu.com");
        selenium.start();
    }

    @After
    public void tearDown() throws Exception {
        selenium.stop();
    }

    @Test
    public void test_getEval1()
    {

    selenium.open("http://automationqa.com/member.php?mod=UserRegiste
r");
        String[] checkboxIds = getAllCheckboxIds();
        for(int i=0;i<checkboxIds.length;i++){
            System.out.println(checkboxIds[i].toString());
        }

        System.out.println("Images Count = " + getImagesCount());
    }
```

```
    public String[] getAllCheckboxIds () {
        String script = "var inputId  = new Array();";// Create array in
java script.
        script += "var cnt = 0;"; // Counter for check box ids.
        script += "var inputFields  = new Array();"; // Create array in
java script.
        script += "inputFields =
window.document.getElementsByTagName('input');"; // Collect input
elements.
        script += "for(var i=0; i<inputFields.length; i++) {"; // Loop
through the collected elements.
        script += "if(inputFields[i].id !=null " +
        "&& inputFields[i].id !='undefined' " +
        "&& inputFields[i].getAttribute('type') == 'checkbox') {"; // If
input field is of type check box and input id is not null.
        script += "inputId[cnt]=inputFields[i].id ;" + // Save check box
id to inputId array.
        "cnt++;" + // increment the counter.
        "}" + // end of if.
        "}"; // end of for.
        script += "inputId.toString();" ;// Convert array in to string.
         String[] checkboxIds = selenium.getEval(script).split(","); //
Split the string.
        return checkboxIds;
    }

    public String getImagesCount(){
        return selenium.getEval("window.document.images.length;");
    }

}
```

Python 与 Selenium

中文编码问题

测试报告（Reporting Results）

Selenium RC does not have its own mechanism for reporting results. Rather, it allows you to build your reporting customized to your needs using features of your chosen programming language. That’s great,

but what if you simply want something quick that's already done for you? Often an existing library or test framework can meet your needs faster than developing your own test reporting code.

Test Framework Reporting Tools

Test frameworks are available for many programming languages. These, along with their primary function of providing a flexible test engine for executing your tests, include library code for reporting results. For example, Java has two commonly used test frameworks, JUnit and TestNG. .NET also has its own, NUnit.

We won't teach the frameworks themselves here; that's beyond the scope of this user guide. We will simply introduce the framework features that relate to Selenium along with some techniques you can apply. There are good books available on these test frameworks however along with information on the internet.

Test Report Libraries

Also available are third-party libraries specifically created for reporting test results in your chosen programming language. These often support a variety of formats such as HTML or PDF.

What's The Best Approach?

Most people new to the testing frameworks will begin with the framework's built-in reporting features. From there most will examine any available libraries as that's less time consuming than developing your own. As you begin to use Selenium no doubt you will start putting in your own "print statements" for reporting progress. That may gradually lead to you developing your own reporting, possibly in parallel to using a library or test framework. Regardless, after the initial, but short, learning curve you will naturally develop what works best for your own situation.

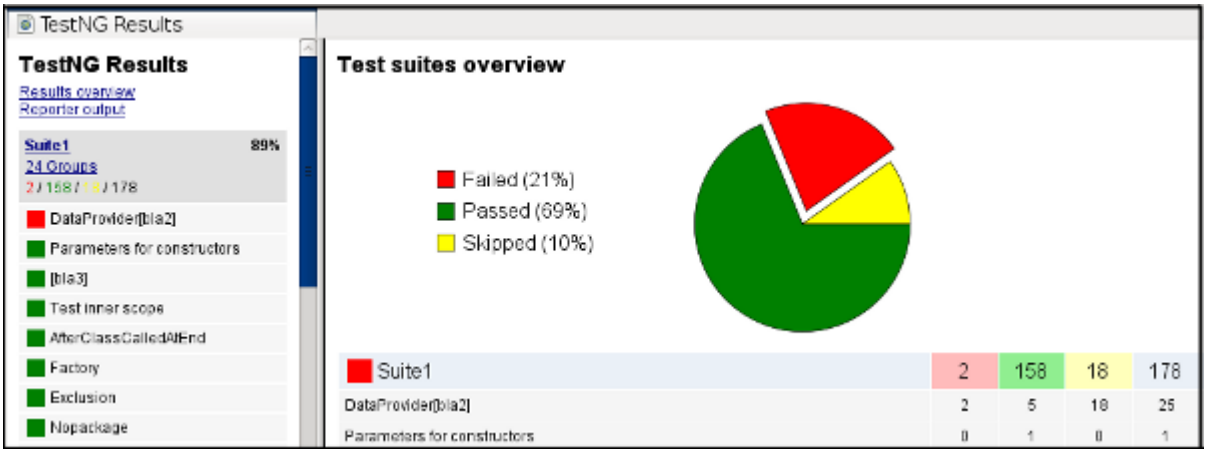
Test Reporting Examples

To illustrate, we'll direct you to some specific tools in some of the other languages supported by Selenium. The ones listed here are commonly used and have been used extensively (and therefore recommended) by the authors of this guide.

Test Reports in Java

- If Selenium Test cases are developed using JUnit then JUnit Report can be used to generate test reports. Refer to JUnit Report for specifics.

- If Selenium Test cases are developed using TestNG then no external task is required to generate test reports. The TestNG framework generates an HTML report which list details of tests. See TestNG Report for more.
- ReportNG is a HTML reporting plug-in for the TestNG framework. It is intended as a replacement for the default TestNG HTML report. ReportNG provides a simple, colour-coded view of the test results.
  See ReportNG for more.
- Also, for a very nice summary report try using TestNG-xslt. A TestNG-xslt Report looks like this.



See TestNG-xslt for more.

Logging the Selenese Commands

- Logging Selenium can be used to generate a report of all the Selenese commands in your test along with the success or failure of each. Logging Selenium extends the Java client driver to add this Selenese logging ability. Please refer to Logging Selenium.

Test Reports for Python

- When using Python Client Driver then HTMLTestRunner can be used to generate a Test Report. See HTMLTestRunner.

Test Reports for Ruby

- If RSpec framework is used for writing Selenium Test Cases in Ruby then its HTML report can be used to generate a test report. Refer to RSpec Report for more.

Note

If you are interested in a language independent log of what's going on, take a look atSelenium Server Logging

WebDriver 和 Selenium 2.0

参考：
http://seleniumhq.org/docs/03_webdriver.html
http://www.qaautomation.net/?p=230
http://www.qaautomation.net/?p=373

RC VS WebDriver

http://qa.blog.163.com/blog/static/19014700220122231779/
http://www.infoq.com/cn/news/2011/07/selenium-arch-2

我们已经有了 Selenium1.0 为什么还需要 Selenium2.0 呢？

Selenium1.0 不能处理一下事件：

1) 本机键盘和鼠标事件
2) 同源策略 XSS/HTTP（S）
3) 弹出框，对话框（基本身份认证，自签名的证书和文件上传/下载）

Selenium2.0 有简洁的 API，WebDriver 和 WebElement 对象，更好的抽象。且支持多中操作系统，多语言，多浏览器。

同时 Selenium2.0 进行了架构的调整和升级：

Selenium2.0 = Selenium1.0 + WebDriver（也就是说 Selenium2.0 合并了这两个项目）

Selenium1.0 可以使用任何编程语言，但是有个先决条件就是必须支持 HTTP 库。Selenium1.0 起初就是一个 Javascript 库，到后面引入了 SeleniumRC。SeleniumRC 作为一个代理服务器并且发送操作命令给 Selenium Core（javascript 代码，且为 SeleniumRC 的一部分）。SeleniumRC 从测试程序接收指令并翻译，返回测试结果给测试程序。Selenium Core 在 client API 打开浏览器后就注入到浏览器中，然后 Selenium Core 接收测试程序的指令，解释成 selenese 命令，在浏览器执行。

Selenium1.0 作为第一个基于 javascript 开源的 web 测试框架，迅速的得到了其他浏览器的支持。但是和任何其他大项目一样，Selenium1.0 也不是完美的。正因为他完全是使用 javascript 写的，这也给他带来了致命的缺陷。为了防止恶意的 javascript，所有浏览器都加强了对 javascript 的安全策略。这样势必导致了在一些特定场景无法使用 Selenium1.0。作为一个大项目，随着时间的日积月累，Selenium1.0 的 API 变得越来也大，也越来也复杂，导致了都不知道更好的使用它改善它。

WebDriver 项目是由 Simon Stewart 提出的，它是一个轻便简洁的自动化测试框架。WebDriver 通过尝试不同的方法去解决 Selenium1.0 所面临的问题。不单单是使用 javascript，WebDriver 会使用任何一种更合适的机制来操作浏览器。IE 通过使用 C++，FF 通过使用 javascript in a XPCOM component。

通过更灵活的机制去操控浏览器，那我们就能很好的绕过浏览器 javascript 的安全限制。当这些技术还不够用时，我们可以调用系统设备操作，尤其是当你需要一些键盘和鼠标操作时，通过这些技术，我们可以更好的模拟用户的真实浏览器操作。

当这两个框架被合并后，一个框架的缺陷被另一个框架所弥补。WebDriver 对浏览器的支持需要对应框架开发工程师做对应的开发；同样 Selenium 必须操作真实浏览器，但是 WebDriver 可以 HTML unit Driver 来模拟浏览器，在内存中执行用例，更加的轻便。Selenium1.0 解决了自动化测试中的一些常见问题，WebDriver 更好的解决了沙箱限制。WebDriver 不支持并行，但是 Selenium Grid 解决了这个问题。

WebDriver 如何驱动浏览器？

Selenium-WebDriver makes direct calls to the browser using each browser's native support for automation. How these direct calls are made, and the features they support depends on the browser you are using. Information on each 'browser driver' is provided later in this chapter.

For those familiar with Selenium-RC, this is quite different from what you are used to. Selenium-RC worked the same way for each supported browser. It 'injected' javascript functions into the browser when the browser was loaded and then used its javascript to drive the AUT within the browser. WebDriver does not use this technique. Again, it drives the browser directly using the browser's built in support for automation.

WebDriver 的底层框架采用了与 Selenium 的 JavaScript 的 Selenium-Core 不一样的实现，WebDriver 不依赖于嵌入浏览器的 JavaScript，因此可以避免一些长期困扰着 Selenium 的限制。

WebDriver 的目标是提供一个 API，给 Web 测试一个设计良好的、标准的编程接口，改进浏览器之间的一致性，提供 Selenium 1.0 不能够很好支持的功能。

Selenium 的开发者们一直尝试不断地改进 Selenium，把 WebDriver 整合进来是这个过程中的一大步。Selenium 的开发者们和 WebDriver 的开发者们觉得整合两者的开发力量、理念和技术将极大地推动开源测试自动化的发展。因此有了 Selenium 2.0。

什么时候应该使用 WebDriver（或者说 Selenium 2.0）呢？当出现以下需求时你应该考虑升级到 Selenium2.0：

1、需要测试多浏览器，而 Selenium 1.0 不能很好支持的时候
2、需要处理多 frames、多浏览器窗口、弹出窗口（popups）、警告窗口（alerts）时。
3、需要处理拖拽（Drag-and-drop）时。
4、需要处理基于 AJAX 的 UI 元素时。

下面简单介绍 Selenium 2.0 的使用方法：
1、下载 selenium-server-2.0a5.zip（http://code.google.com/p/selenium/downloads/list），解压缩后，在命令行中进入目录，使用下面命令启动服务：
java - jar selenium-server-standalone-2.0a5.jar
2、下载 selenium-dotnet-2.0a5.zip（http://code.google.com/p/selenium/downloads/list），解压缩，打开 VS2005，新建一个控制台项目，引用 selenium-dotnet-2.0a5 目录中的类库，这样就可以准备开始 WebDriver 代码的开发了。

3、编写一个简单的例子：

```
using System;
using System.Collections.Generic;
using System.Text;

using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;

namespace WebDriverTest1
{
    class Program
    {

        static void Main(string[] args)
        {
            // 实例化浏览器对象
            IWebDriver driver = new FirefoxDriver();
            // 导航到指定页面
            driver.Navigate().GoToUrl("http://www.google.com.hk");
            // 查找页面对象
            IWebElement element = driver.FindElement(By.Name("q"));
            // 输入
            element.SendKeys("Selenium 2.0");
            // 提交
            element.Submit();
            // 获取标题
            Console.WriteLine("Page title is: " + driver.Title);
            // 关闭
            driver.Quit();
            // 释放对象
            driver.Dispose();
```

```
        }
    }
}
```

在 VS2005 中运行项目即可执行测试脚本。

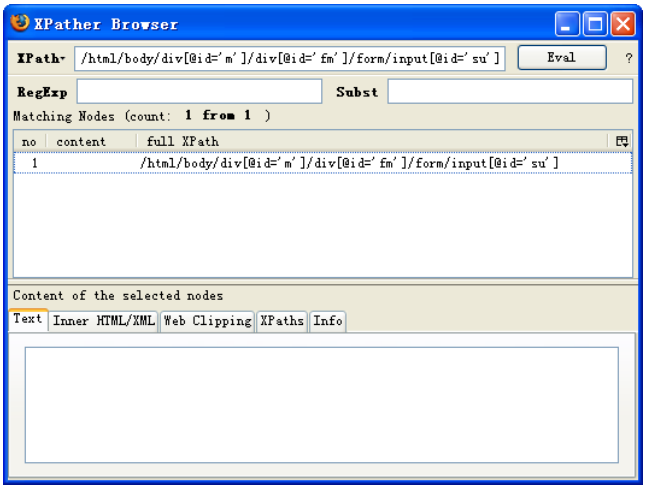
4、再看一个相对复杂的例子：

```
            // 实例化浏览器对象
            IWebDriver driver = new FirefoxDriver();
            // 导航到指定页面
            driver.Navigate().GoToUrl("http://www.baidu.com");
            // 查找页面对象
            IWebElement element = driver.FindElement(By.Name("wd"));
            // 输入
            element.SendKeys("Selenium 2.0");
            // 点击“百度一下”按钮
            driver.FindElement(By.XPath("//input[@type='submit'  and  @value='百度一下']")).Click();
            // 获取标题
            Console.WriteLine("Page title is: " + driver.Title);
           // 获取返回页面的 body 内容
            IWebElement body = driver.FindElement(By.XPath("/html/body"));
            if (body.Text.Contains("WebDriver"))
                Console.WriteLine("Pass!");
            else
                Console.WriteLine("Fail!");
            // 关闭
            driver.Quit();
            // 释放对象
            driver.Dispose();
```

在这个例子中，使用了XPath来查找页面元素，关于XPath的基本概念和使用，请读者参考：

http://www.zvon.org/xxl/XPathTutorial/General_chi/examples.html

可以利用Firefox的一个插件“XPather”（http://xpath.alephzarro.com/），来协助获取和测试页面元素的XPath路径值。在Firefox安装XPather后，选择Web页面中的任何元素，然后单击右键，选择“Show in XPather”，则会在XPather Browser中显示该页面元素的XPath全路径，如图所示：



也可以自己在“XPath”输入框中输入XPath路径，单击“Eval”按钮，测试是否能定位和查找到指定的元素。

*注：Selenium2.0目前仍在持续开发和改进中，读者可到google代码开源网站查看其进展情况：

http://code.google.com/p/selenium/wiki/RoadMap

加载浏览器插件

http://www.cnblogs.com/nbkhic/archive/2011/10/28/2228067.html

有时候我们需要在使用 firefox 测试时启动 firebug，这时候就可以用到下面的代码

```
require 'rubygems'
require 'selenium-webdriver'

profile = Selenium::WebDriver::Firefox::Profile.new
profile.add_extension 'where/the/extensions/locate/firebug.xpi'
dr = Selenium::WebDriver.for :firefox, :profile => profile
```

这段代码首先创建了 1 个新的 firefox profile，然后将 firebug 的扩展配置到该 profile 中去。

另外也可以看出，通过指定不同的 profile 可以起到配置 firefox 的功能

设置代理：

http://www.cnblogs.com/nbkhic/archive/2011/10/30/2229221.html

通过设置 firefox 可执行文件地址的方式来加速 firefox 的启动速度

http://www.cnblogs.com/nbkhic/archive/2011/11/05/2237198.html

加载 Profile

Custom Firefox profile for Selenium

http://www.qaautomation.net/?p=45

selenium 配置 firefox profile，创建 firefox profile
http://blog.csdn.net/andyhong110/article/details/5766925

1． 打开运行窗口。输入 firefox –p 回车。会出现下面类似的窗口。

如果没有出现，可能是你已经打开了 firefox ， 如果是请先关闭。

2． 点击创建配置文件。然后单击下一步。然后选择一个存放 profile 的文件夹。

3．给你的新的配置文件输入一个名称。然后点击结束按钮。

这样你的 firefox profile 就配置好了，接下来我们要给我们的 profile 下的 firefox 实例，添加一些插件。这样你在使用这个 profile 的时候就会有相应的插件。

4．打开运行窗口输入 firefox –p 回车。在弹出串口中选择你要使用的 profile 然后点击启动 firefox 按钮就可以使用你的新 profile 了。

5． 在 selenium 的 start.bat 文件中加入 -firefoxProfileTemplate "your firefox profile path"，这样你的 selenium server 启动的 firefox 就是指定的 profile 了。

浏览器操作

http://blog.sina.com.cn/s/blog_6966650401012a5f.html

```
1.启动浏览器
A.firefox
//打开默认路径的 firefox（路径指的是  firefox  的安装路径）
        WebDriver diver = new FirefoxDriver();
//打开指定路径的 firefox,方法 1
        System.setProperty("webdriver.firefox.bin","D:\\ProgramFiles\\Mozilla Firefox\\firefox.exe");
        WebDriver dr = new FirefoxDriver();
//打开指定路径的 firefox,方法 2
        File pathToFirefoxBinary  = new File("D:\\Program Files\\Mozilla Firefox\\firefox.exe");
        FirefoxBinary firefoxbin = new FirefoxBinary(pathToFirefoxBinary);
        WebDriver driver1 = new FirefoxDriver(firefoxbin,null);
        
B.ie
//打开 ie
        WebDriver ie_driver = new InternetExplorerDriver();

C.chrome
因为 Chrome Driver 是 Chromium 项目自己支持和维护的，所以你必需另外下载 chromedriver.exe，放在目录下 C:\WINDOWS\system32
下载地址：  http://vdisk.weibo.com/s/5Bf3L
//打开 chrome
                WebDriver driver = new ChromeDriver();
```

```
另一种启动 chrome 的方法
wiki 介绍：http://code.google.com/p/selenium/wiki/ChromeDriver
//打开 chrome
                System.setProperty("webdriver.chrome.driver",
"D:\\chromedriver.exe");
                System.setProperty("webdriver.chrome.bin",
                                                        C:\\Documents and
Settings\\fy\\Local Settings"
                                                        +"\\Application
Data\\Google\\Chrome\\Application\\chrome.exe");

Chromium 介绍：http://code.google.com/p/chromium/



2.页面跳转 url
String url = "http://www.baidu.com";
WebDriver driver = new FirefoxDriver();

A//用 get 方法
                driver.get(url);

B//用 navigate 方法，然后再调用 to 方法,chrome 不支持这种方法
                driver.navigate().to(url);



3.关闭浏览器
//quit  关闭所有页面   close  关闭本次执行打开的页面
A.//用 quit 方法
                driver.quit();
B.//用 close 方法
                driver.close();

4.获取页面信息
//得到 title
String title = driver.getTitle();
//得到当前页面 url
String currentUrl = driver.getCurrentUrl();

getWindowHandle()         返回当前的浏览器的窗口句柄
getWindowHandles()    返回当前的浏览器的所有窗口句柄
getPageSource()                  返回当前页面的源码
//String s=driver.getPageSource();s=s.substring(s.indexOf("{"), s.indexOf("}"));
//System.out.println("当前页面的源码:"+s);
```

5. 总结
操作浏览器的主要方法都来自 org.openqa.selenium.WebDriver 这个接口中。
源代码这些方法都是在　org.openqa.selenium.remote.RemoteWebDriver 这个类中实现的，然后不同浏览的 driver 类继承　RemoteWebDriver。

WebDriver 与 Selenium-Server

不需要用 **Server**：

You may, or may not, need the Selenium Server, depending on how you intend to use Selenium-WebDriver. If you will be only using the WebDriver API you do not need the Selenium-Server. If your browser and tests will all run on the same machine, and your tests only use the WebDriver API, then you do not need to run the Selenium-Server; WebDriver will run the browser directly.

需要用 **Server**：

There are some reasons though to use the Selenium-Server with Selenium-WebDriver.

- You are using Selenium-Grid to distribute your tests over multiple machines or virtual machines (VMs).
- You want to connect to a remote machine that has a particular browser version that is not on your current machine.
- You are not using the Java bindings (i.e. Python, C#, or Ruby) and would like to use HtmlUnit Driver

关于HtmlUnit Driver：
http://code.google.com/p/selenium/wiki/HtmlUnitDriver

RemoteWebDriver

http://blog.sina.com.cn/s/blog_6966650401012dye.html

当本机上没有浏览器，需要远程调用浏览器进行自动化测试时，需要用到 RemoteWebDirver.

一、使用 RemoteWebDriver

```
import java.io.File;
import java.net.URL;

import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.Augmenter;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
public class Testing {
        public void myTest()throws Exception {
                WebDriver driver = newRemoteWebDriver(
                                                        new
URL("http://localhost:4446/wd/hub"),
                                                        DesiredCapabiliti
es.firefox());
              driver.get("http://www.google.com");

                // RemoteWebDriverdoes not implement the TakesScreenshot class
                // if the driver doeshave the Capabilities to take a screenshot
                // then Augmenter willadd the TakesScreenshot methods to the
instance
                WebDriveraugmentedDriver  = new Augmenter().augment(driver);
                File screenshot =((TakesScreenshot)augmentedDriver).
                                                        getScreenshotAs(OutputT
ype.FILE);
        }
}

二、SeleniumServer
在使用 RemoteDriver 时，必须在远程服务器启动一个 SeleniumServer:
java -jar selenium-server-standalone-2.20.0.jar -port 4446

三、How to setFirefox profile using RemoteWebDriver
            profile = new FirefoxProfile();
            profile.setPreference("general.useragent.override",testData.getUser
Agent());
```

```
            capabilities = DesiredCapabilities.firefox();
            capabilities.setCapability("firefox_profile", profile);
            driver = new RemoteWebDriver(new
URL(“http://localhost:4446/wd/hub”),capabilities);
            driverWait = new
WebDriverWait(driver,TestConstant.WAIT_ELEMENT_TO_LOAD);
            driver.get("http://www.google.com");
```

创建 Selenium-WebDriver 项目（Maven+Eclipse+Selenium）

To install Selenium means to set up a project in a development so you can write a program using Selenium. How you do this depends on your programming language and your development environment.

The easiest way to set up a Selenium 2.0 Java project is to use Maven. Maven will download the java bindings (the Selenium 2.0 java client library) and all its dependencies, and will create the project for you, using a maven pom.xml (project configuration) file. Once you’ve done this, you can import the maven project into your preferred IDE, IntelliJ IDEA or Eclipse.

First, create a folder to contain your Selenium project files. Then, to use Maven, you need a pom.xml file. This can be created with a text editor. We won’t teach the details of pom.xml files or for using Maven since there are already excellent references on this. Your pom.xml file will look something like this. Create this file in the folder you created for your project.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>MySel20Proj</groupId>
        <artifactId>MySel20Proj</artifactId>
        <version>1.0</version>
        <dependencies>
```

```
            <dependency>
                <groupId>org.seleniumhq.selenium</groupId>
                <artifactId>selenium-java</artifactId>
                <version>2.24.1</version>
            </dependency>
            <dependency>
                <groupId>com.opera</groupId>
                <artifactId>operadriver</artifactId>
            </dependency>
        </dependencies>
        <dependencyManagement>
            <dependencies>
                <dependency>
                    <groupId>com.opera</groupId>
                    <artifactId>operadriver</artifactId>
                    <version>0.14</version>
                    <exclusions>
                        <exclusion>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-remote-driver</artifactId>
                        </exclusion>
                    </exclusions>
                </dependency>
            </dependencies>
        </dependencyManagement>
</project>
```

Be sure you specify the most current version. At the time of writing, the version listed above was the most current, however there were frequent releases immediately after the release of Selenium 2.0. Check the Maven download pagefor the current release and edit the above dependency accordingly.

Now, from a command-line, CD into the project directory and run maven as follows.

```
mvn clean install
```

This will download Selenium and all its dependencies and will add them to the project.

Finally, import the project into your preferred development environment. For those not familiar with this, we've provided an appendix which shows this.

Importing a maven project into IntelliJ IDEA. *Importing a maven project into Eclipse*.

参考：

http://www.cnblogs.com/fnng/archive/2012/02/04/2337919.html
http://www.cnblogs.com/fnng/archive/2011/12/02/2272610.html
http://mojo.codehaus.org/selenium-maven-plugin/

Alternative Back-Ends: Mixing WebDriver and RC Technologies

WebDriver-Backed Selenium-RC（兼容 RC）

The Java version of WebDriver provides an implementation of the Selenium-RC API. These means that you can use the underlying WebDriver technology using the Selenium-RC API. This is primarily provided for backwards compatibility. It allows those who have existing test suites using the Selenium-RC API to use WebDriver under the covers. It's provided to help ease the migration path to Selenium-WebDriver. Also, this allows one to use both APIs, side-by-side, in the same test code.

Selenium-WebDriver is used like this:

```
// You may use any WebDriver implementation. Firefox is used here as an
example
WebDriver driver = new FirefoxDriver();

// A "base url", used by selenium to resolve relative URLs
 String baseUrl = "http://www.google.com";

// Create the Selenium implementation
Selenium selenium = new WebDriverBackedSelenium(driver, baseUrl);

// Perform actions with selenium

selenium.open("http://www.google.com");
selenium.type("name=q", "cheese");
selenium.click("name=btnG");

// Get the underlying WebDriver implementation back. This will refer to
the
// same WebDriver instance as the "driver" variable above.
```

```
WebDriver driverInstance = ((WebDriverBackedSelenium)
selenium).getWrappedDriver();

//Finally, close the browser. Call stop on the WebDriverBackedSelenium
instance
//instead of calling driver.quit(). Otherwise, the JVM will continue
running after
//the browser has been closed.
selenium.stop();
```

Pros

- Allows for the WebDriver and Selenium APIs to live side-by-side
- Provides a simple mechanism for a managed migration from the Selenium RC API to WebDriver's
- Does not require the standalone Selenium RC server to be run

Cons

- Does not implement every method
- More advanced Selenium usage (using "browserbot" or other built-in JavaScript methods from Selenium Core) may not work
- Some methods may be slower due to underlying implementation differences

Backing WebDriver with Selenium（调用 Selenese 命令）

WebDriver doesn't support as many browsers as Selenium RC does, so in order to provide that support while still using the WebDriver API, you can make use of the`SeleneseCommandExecutor`

Safari is supported in this way with the following code (be sure to disable pop-up blocking):

```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setBrowserName("safari");
CommandExecutor executor = new SeleneseCommandExecutor(new
URL("http://localhost:4444/"), new URL("http://www.google.com/"),
capabilities);
WebDriver driver = new RemoteWebDriver(executor, capabilities);
```

There are currently some major limitations with this approach, notably that findElements doesn't work as expected. Also, because we're using Selenium Core for the heavy lifting of driving the browser, you are limited by the JavaScript sandbox.

Running Standalone Selenium Server for use with RemoteDrivers

From Selenium's Download page download selenium-server-standalone-<version>.jar and optionally IEDriverServer. If you plan to work with Chrome, download it from Google Code.

Unpack IEDriverServer and/or chromedriver and put them in a directory which is on the $PATH / %PATH% - the Selenium Server should then be able to handle requests for IE / Chrome without additional modifications.

Start the server on the command line with

```
java -jar <path_to>/selenium-server-standalone-<version>.jar
```

If you want to use native events functionality, indicate this on the command line with the option

```
-Dwebdriver.enable.native.events=1
```

For other command line options, execute

```
java -jar <path_to>/selenium-server-standalone-<version>.jar -help
```

In order to function properly, the following ports should be allowed incoming TCP connections: 4444, 7054-5 (or twice as many ports as the number of concurrent instances you plan to run). Under Windows, you may need to unblock the applications as well.

RemoteWebDriver：
http://code.google.com/p/selenium/wiki/RemoteWebDriverServer
http://code.google.com/p/selenium/wiki/RemoteWebDriver

Native events versus synthetic events

http://code.google.com/p/selenium/wiki/AdvancedUserInteractions#Native_events_versus_synthetic_events

In WebDriver advanced user interactions are provided by either simulating the Javascript events directly (i.e. synthetic events) or by letting the browser generate the Javascript events (i.e. native events). Native

events simulate the user interactions better whereas synthetic events are platform independent, which can be important in Linux when alternative window managers are used, see native events on Linux. Native events should be used whenever it is possible.

The following table shows which browsers support which kind of events:

Browser	Operating system	Native events	Synthetic events
Firefox	Linux	supported	supported (default)
Firefox	Windows	supported (default)	supported
Internet Explorer	Windows	supported (default)	not supported
Chrome	Linux/Windows	supported*	not supported
Opera	Linux/Windows	supported (default)	not supported
HtmlUnit	Linux/Windows	supported (default)	not supported

*) ChromeDriver provides two modes of supporting native events called WebKit events and raw events. In the WebKit events the ChromeDriver calls the WebKit functions which trigger Javascript events, in the raw events mode operating systems events are used.

In the FirefoxDriver, native events can be turned on and off in the FirefoxProfile.

```
FirefoxProfile profile = new FirefoxProfile();
profile.setEnableNativeEvents(true);
FirefoxDriver driver = new FirefoxDriver(profile);
```

Examples

These are some examples where native events behave different to synthetic events:

- With synthetic events it is possible to click on elements which are hidden behind other elements. With native events the browser sends the click event to the top most element at the given location, as it would happen when the user clicks on the specific location.
- When a user presses the 'tab' key the focus jumps from the current element to the next element. This is done by the browser. With synthetic events the browser does not know that the 'tab' key is pressed and therefore won't change the focus. With native events the browser will behave as expected.

启动 Selenium 服务器时指定 native events 选项：

If you want to use native events functionality, indicate this on the command line with the option

```
-Dwebdriver.enable.native.events=1
```

参考：
http://watirmelon.com/2011/09/16/disabling-native-events-when-using-firefox-with-webdriver/

WebDriver+TestNG+ANT 实现多浏览器兼容性测试

http://www.cnblogs.com/rbai/archive/2012/07/22/2603395.html

一 使用说明

整个 Demo 通过 ANT 和 TestNG 的结合使用， 同时引入 log4J 方便记录日志。
最后 通过 testng-results.xsl 文件将 TestNG 生成的报告美化。

Demo 运行 4 个百度搜索脚本，每一个脚本都会在 IE firefox Chorme 中分别执行一次。

Demo 下载地址：
文件名 ： ATS-WD.rar
115 网盘下载
CSDN 下载

1. 安装 ANT

apache-ant-1.7.1-bin.zip
下载地址：
http://archive.apache.org/dist/ant/binaries/apache-ant-1.7.1-bin.zip

第一步 解压 apache-ant-1.7.1-bin.zip 到某个盘符，这里以 C 盘做为例。

第二步 将 bin 所在目录设置为 path 环境变量的值之一：C:\apache-ant-1.7.1\bin

第三步 设置环境变量 ANT_HOME 为：C:\apache-ant-1.7.1

第四步 打开 Dos 窗口，输入 ant -version 检查是否正确，如果正确如下图

```
C:\Documents and Settings\Administrator>ant -version

Apache Ant version 1.7.1 compiled on June 27 2008
```

2. 配置 firefox 的安装目录

修改 src 目录下的 testng.xml 文件

[code]<parameter name="firefox_dir"

value="D:/Program Files/Mozilla Firefox/Mozilla Firefox/firefox.exe" />[/code]

value 为 本地 firefox 的安装目录

3. 执行脚本

第一种方法 ：在 eclipse 中导入项目，运行 ant 脚本 build.xml

第二种方法 ：用 dos 进入到 ATS-WD 的目录，直接输入命令 ant 即可。

4. 查看报告

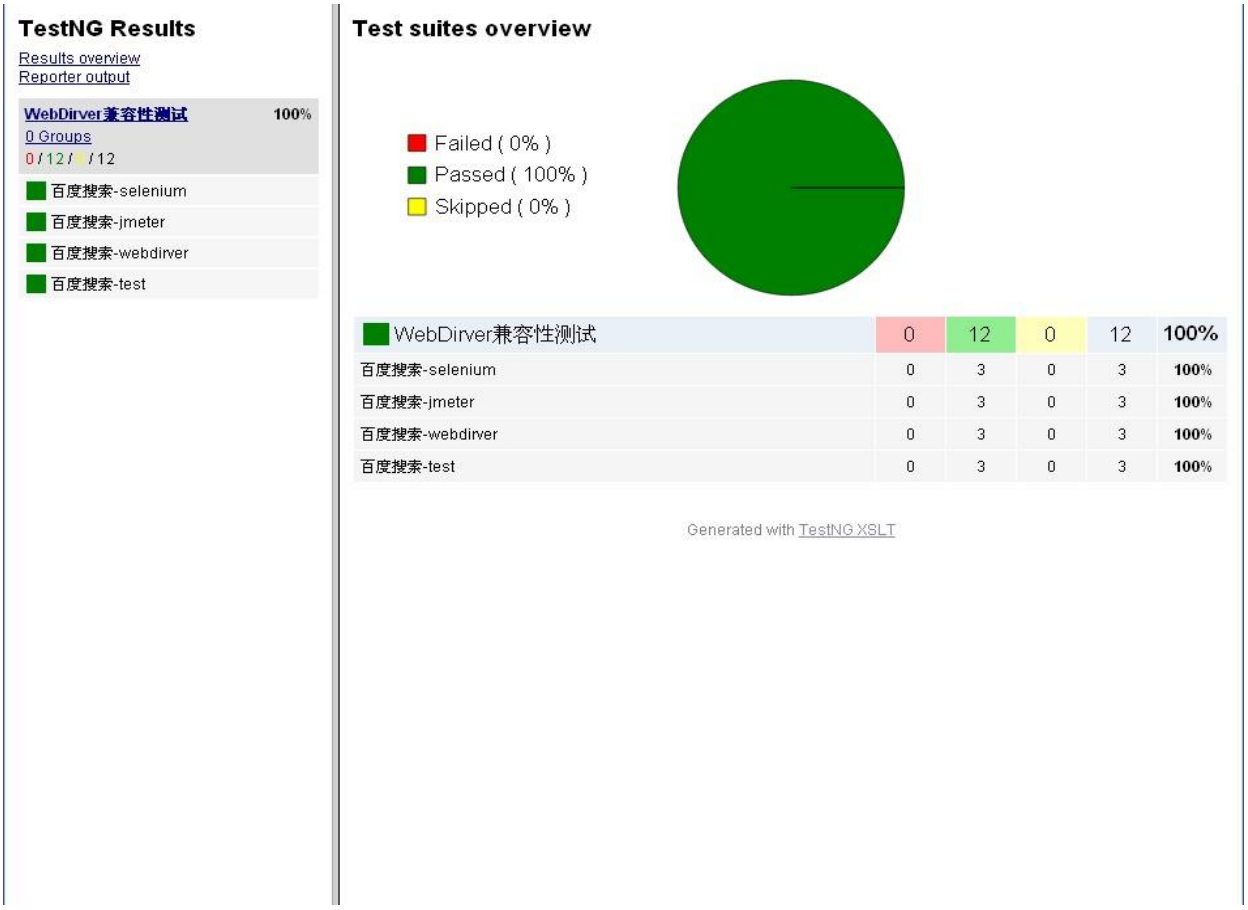
通过 ANT 运行后的测试报告如下：

在线展示：

http://opentest.cn/test-output/Report.html

截图中的 饼图需要第三方插件支持，firefox 和 chrome 可以正常显示。ie 中不会显示。

在本地运行后，开打 test-output 目录下的 Report.html 可以查看报告。



二 脚本设计

具体实现说明：

主要方法是使用 InternetExplorerDriver FirefoxDriver ChromeDriver 运行相同的测试脚本。

将每一种 Driver 的启动写成 @Test 的方法，如下所示：

@Test

```
@Parameters( { "webSite" })
public void setUp_InternetExplorerDriver(String webSite) throws Exception {
    //.\\lib\\IEDriverServer.exe  是 lib 目录下的驱动
    System.setProperty("webdriver.ie.driver", ".\\lib\\IEDriverServer.exe");
    DesiredCapabilities capabilities = DesiredCapabilities.internetExplorer();
    capabilities.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNORING_SECURITY_DOMAINS,
                               true);
    driver = new InternetExplorerDriver(capabilities);
    baseUrl = webSite;
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    script(driver);
}

@Test
@Parameters( { "firefox_dir", "webSite" })
public void setUp_FirefoxDriver(String firefox_dir, String webSite)
        throws Exception {
    //firefox_dir  为本机  firefox 安装目录
    System.setProperty("webdriver.firefox.bin", firefox_dir);
    driver = new FirefoxDriver();
    baseUrl = webSite;
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    script(driver);
}

@Test
@Parameters( { "webSite" })
public void setUp_ChromeDriver(String webSite) throws Exception {
    //.\\lib\\IEDriverServer.exe  是 lib 目录下的驱动
    System.setProperty("webdriver.chrome.driver", "./lib/chromedriver.exe");
    driver = new ChromeDriver();
    baseUrl = webSite;
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    script(driver);
}
```

对于具体的测试脚本，写成一个新的方法，每一个 driver 都会调用该方法，如下所示：

```
private void script(WebDriver driver) throws Exception {
        try {
            logger.log("开始");
            driver.get(baseUrl);
            driver.findElement(By.id("kw")).clear();
            driver.findElement(By.id("kw")).sendKeys("selenium");
            driver.findElement(By.id("kw")).submit();
            driver.findElement(By.linkText("下一页>")).click();
            driver.findElement(By.linkText("下一页>")).click();
            driver.findElement(By.linkText("下一页>")).click();
            assertEquals("selenium", driver.findElement(By.id("kw"))
                                        .getAttribute("value"));

            logger.log("结束");

        } catch (Exception e) {
            throw e;
        } finally {
            tearDown();
        }
    }
```

三 ANT TestNG 配置

ANT 脚本文件为 根目录下 的 build.xml

主要任务是 ： 编译脚本、调用 TestNG 执行脚本、将 TestNG 生成的报告美化。

TestNG 配置文件为 src 目录下的 testng.xml

通过配置 testng.xml 可以到达 脚本 参数化 ， 并发执行脚本的目的，如下所示

```
<suite name="WebDriver 兼容性测试" thread-count="4" parallel="tests">

        <parameter name="firefox_dir"
                value="D:/Program Files/Mozilla Firefox/Mozilla Firefox/firefox.exe" />
        <parameter name="webSite" value="http://www.baidu.com" />
```

thread-count="4" 为同时执行的线程数 parallel="tests" 为同时执行的测试级别

四 日志配置

将 log4J 和 TestNG 中自带的日志功能结合。

在脚本中写入的日志会分别记录在以下地方：

1. 测试报告 -- Report.html
2. logs 文件夹中的日志文件
3. 在控制台中打印

对于每一个 webdriver 脚本 开始先初始化一下：

```
private static final SelLogger logger = SelLogger.getLogger(脚本名.class);
```

在脚本中 使用 ：

```
logger.log("开始");
logger.log("结束");
```

五 目录结构

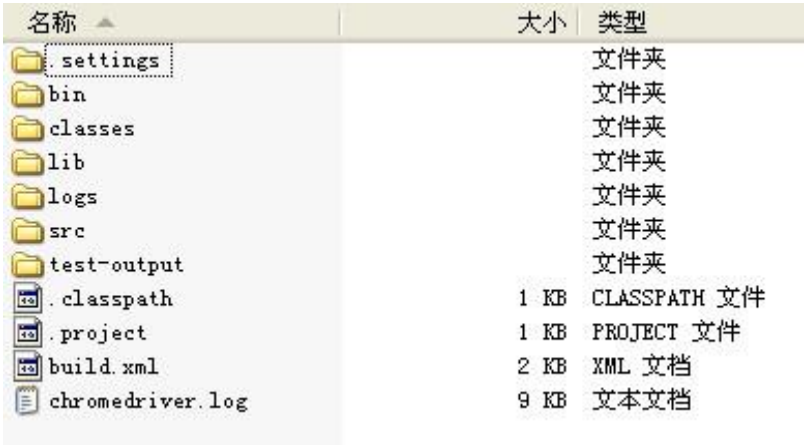
src 目录为 源码

lib 目录为 需要使用到的 jar 包文件，包括 ChromeDriver InternetExplorerDriver 需要使用的 exe 文件

logs 目录为 日志文件，在脚本中使用 logger.log("XXXXXX")
会记录到该文件夹中

classes 目录为 ANT 编译后的 class 文件夹，可删除

test-output 目录为 测试报告目录 TestNG 自动生成，可删除



TestNG 并发执行：
http://blog.sina.com.cn/s/blog_6966650401012u9z.html

各种 Driver

HtmlUnit Driver

Firefox Driver

Controls the Firefox browser using a Firefox plugin. The Firefox Profile that is used is stripped down from what is installed on the machine to only include the Selenium WebDriver.xpi (plugin). A few settings are also changed by default (see the source to see which ones) Firefox Driver is capable of being run and is tested on Windows, Mac, Linux. Currently on versions 3.6, 10, latest - 1, latest

Usage

```
WebDriver driver = new FirefoxDriver();
```

Pros

- Runs in a real browser and supports JavaScript
- Faster than the Internet Explorer Driver

Cons

- Slower than the HtmlUnit Driver

Modifying the Firefox Profile

Suppose that you wanted to modify the user agent string (as above), but you've got a tricked out Firefox profile that contains dozens of useful extensions. There are two ways to obtain this profile. Assuming that the profile has been created using Firefox's profile manager (`firefox -ProfileManager`):

```
ProfileIni allProfiles = new ProfilesIni();
FirefoxProfile profile = allProfiles.getProfile("WebDriver");
profile.setPreferences("foo.bar", 23);
WebDriver driver = new FirefoxDriver(profile);
```

Alternatively, if the profile isn't already registered with Firefox:

```
File profileDir = new File("path/to/top/level/of/profile");
FirefoxProfile profile = new FirefoxProfile(profileDir);
profile.addAdditionalPreferences(extraPrefs);
WebDriver driver = new FirefoxDriver(profile);
```

As we develop features in the Firefox Driver, we expose the ability to use them. For example, until we feel native events are stable on Firefox for Linux, they are disabled by default. To enable them:

```
FirefoxProfile profile = new FirefoxProfile();
profile.setEnableNativeEvents(true);
WebDriver driver = new FirefoxDriver(profile);
```

Info

See the Firefox section in the wiki page for the most up to date info.

Internet Explorer Driver

http://code.google.com/p/selenium/wiki/InternetExplorerDriver

The Internet Explorer Driver Server

http://seleniumhq.org/download/

This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.24.2 for (recommended) 32 bit Windows IE or 64 bit Windows IE

ChromeDriver

http://code.google.com/p/selenium/wiki/ChromeDriver
http://www.cnblogs.com/nbkhic/archive/2011/10/29/2228393.html

Getting running with Chrome Driver

Download the Chrome Driver executable and follow the other instructions on the wiki page

Opera Driver

iPhone Driver

Android Driver

等待（Explicit and Implicit Waits）

Waiting is having the automated task execution elapse a certain amount of time before continuing with the next step.

显式等待（Explicit Waits）

An explicit waits is code you define to wait for a certain condition to occur before proceeding further in the code. The worst case of this is Thread.sleep(), which sets the condition to an exact time period to wait. There are some convenience methods provided that help you write code that will wait only as long as required. WebDriverWait in combination with ExpectedCondition is one way this can be accomplished.

```
WebDriver driver = new FirefoxDriver();
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = (new WebDriverWait(driver, 10))
  .until(new ExpectedCondition<WebElement>(){
        @Override
        public WebElement apply(WebDriver d) {
                return d.findElement(By.id("myDynamicElement"));
        }});
```

This waits up to 10 seconds before throwing a TimeoutException or if it finds the element will return it in 0 - 10 seconds. WebDriverWait by default calls the ExpectedCondition every 500 milliseconds until it returns successfully. A successful return is for ExpectedCondition type is Boolean return true or not null return value for all other ExpectedCondition types.

This example is also functionally equivalent to the first Implicit Waits example.

Expected Conditions

There are some common conditions that are frequently come across when automating web browsers. Listed below are Implementations of each. Java happens to have convienence methods so you don't have to code an ExpectedCondition class yourself or create your own utility package for them.

- Element is Clickable - it is Displayed and Enabled.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));
```

The ExpectedConditions class in contains a set of predefined conditions to use with WebDriverWait in Java.

隐式等待（Implicit Waits）

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement =
driver.findElement(By.id("myDynamicElement"));
```

pageLoadTimeout

http://www.qaautomation.net/?p=490

```
driver.manage().timeouts().pageLoadTimeout(90, TimeUnit.SECONDS);
```

http://blog.sina.com.cn/s/blog_6966650401012ayq.html

```
package com.testscripts;
```

```
import java.util.concurrent.TimeUnit;

import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;

public class WebDriver_Wait {

    @Test
    public void test1() {

        String url = "file:///D:/selenium/MyTest/AUT/wait.html";
        // 打开firefox
        WebDriver dr = new FirefoxDriver();
        dr.get(url);

        dr.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        WebElement button_b = dr.findElement(By.id("b"));
        button_b.click();

        /*
        WebDriverWait wait = new WebDriverWait(dr, 10);
        wait.until(new ExpectedCondition<WebElement>() {
            @Override
            public WebElement apply(WebDriver d) {
                return d.findElement(By.className("red_box"));
            }
        });
        */

        WebElement element =
dr.findElement(By.cssSelector(".red_box"));
        System.out.println("获取.red_box的背景颜色属性值:"
                + element.getCssValue("background-color"));
        // 在红色区域外面加黄框
        ((JavascriptExecutor) dr).executeScript(
```

```
                "arguments[0].style.border = \"5px solid yellow\"",
element);

        //dr.quit();
    }


}
```

截屏

http://blog.sina.com.cn/s/blog_6966650401012c2b.html

```
package com.testscripts;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.commons.io.FileUtils;
import org.junit.Test;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.remote.Augmenter;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class RemoteWebDriver_ScreenShot {

    @Test
    public void test_RemoteWebDriver_ScreenShot() throws Exception {
        WebDriver driver = new RemoteWebDriver(new URL(
            "http://localhost:4444/wd/hub"),
DesiredCapabilities.firefox());
        driver.get("http://www.baidu.com");

        String dir_name = "screenshot"; // 这里定义了截图存放目录名
        if (!(new File(dir_name).isDirectory())) { // 判断是否存在该目录
            new File(dir_name).mkdir(); // 如果不存在则新建一个目录
```

```
        }

        SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd-HHmmss");
        String time = sdf.format(new Date()); // 这里格式化当前时间，例如
20120406-165210，后面用的着

        try {
            // RemoteWebDriver does not implement the TakesScreenshot
class
            // if the driver does have the Capabilities to take a screenshot
            // then Augmenter will add the TakesScreenshot methods to the
instance
            WebDriver augmentedDriver = new Augmenter().augment(driver);
            File source_file = ((TakesScreenshot) augmentedDriver)
                    .getScreenshotAs(OutputType.FILE);

            FileUtils.copyFile(source_file, new File(dir_name +
File.separator
                    + time + ".png")); // 这里将截图另存到我们需要保存的目录，
例如screenshot\20120406-165210.png
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Test
    public void test_TakesScreenshot() throws Exception {

        WebDriver driver = new FirefoxDriver();
        driver.get("http://www.baidu.com");

        String dir_name = "screenshot"; // 这里定义了截图存放目录名
        if (!(new File(dir_name).isDirectory())) { // 判断是否存在该目录
            new File(dir_name).mkdir(); // 如果不存在则新建一个目录
        }

        SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd-HHmmss");
        String time = sdf.format(new Date()); // 这里格式化当前时间，例如
20120406-165210，后面用的着

        try {
            File source_file = ((TakesScreenshot) driver)
                    .getScreenshotAs(OutputType.FILE); // 关键代码，执行屏
幕截图，默认会把截图保存到temp目录
```

```
            FileUtils.copyFile(source_file, new File(dir_name +
File.separator
                    + time + ".png")); // 这里将截图另存到我们需要保存的目录，
例如screenshot\20120406-165210.png
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Selenium1.0 实现

```
Selenium selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://www.google.com.hk");
selenium.start();
selenium.open("http://www.google.com.hk/");
selenium.windowMaximize();
selenium.windowFocus();
selenium.captureEntirePageScreenshot("screenshot.png", "");
```

似乎只能在 Selenium RC 上运行，以及 Firefox 浏览器。。Selenium1.0 有个 CaptureScreenshot 方法也能实现屏幕截图。CaptureScreenshot 截取浏览器内可见部分，而 captureEntirePageScreenshot 截取浏览器内所有内容。

参考：
http://qa.blog.163.com/blog/static/190147002201236437196 38/

Browser Startup Manipulation

Using a Proxy

Internet Explorer

The easiest and recommended way is to manually set the proxy on the machine that will be running the test. If that is not possible or you want your test to run with a different configuration or proxy, then you can use the following technique that uses a Capababilities object. This temporarily changes the system’s proxy settings and changes them back to the original state when done.

```
String PROXY = "localhost:8080";

org.openqa.selenium.Proxy proxy = new org.openqa.selenium.Proxy();
proxy.setHttpProxy(PROXY)
     .setFtpProxy(PROXY)
     .setSslProxy(PROXY);
DesiredCapabilities cap = new DesiredCapabailities();
cap.setPreference(CapabilityType.PROXY, proxy);

WebDriver driver = new InternetExplorerDriver(cap);
```

Chrome

Is basically the same as internet explorer. It uses the same configuration on the machine as IE does (on windows). On Mac it uses the System Preference -> Network settings. On Linux it uses (on Ubuntu) System > Preferences > Network Proxy Preferences (Alternatively in "/etc/environment" set http_proxy). As of this writing it is unknown how to set the proxy programmatically.

Firefox

Firefox maintains it's proxy configuration in a profile. You can preset the proxy in a profile and use that Firefox Profile or you can set it on profile that is created on the fly as is shown in the following example.

```
String PROXY = "localhost:8080";

org.openqa.selenium.Proxy proxy = new org.openqa.selenium.Proxy();
proxy.setHttpProxy(PROXY)
     .setFtpProxy(PROXY)
     .setSslProxy(PROXY);
DesiredCapabilities cap = new DesiredCapabailities();
cap.setPreference(CapabilityType.PROXY, proxy);
WebDriver driver = new FirefoxDriver(cap);
```

WebDriver 定位对象

Locating page elements using WebDriver
http://www.qaautomation.net/?p=388
http://blog.sina.com.cn/s/blog_6966650401012a6u.html

定位器类型（Types of Locators）

Locators have changed with Selenium 2 with the WebDriver API. Out of the box WebDriver supports several ways to locate elements on the page using the `By` abstract class.

Finding elements by ID

ids are the preferred way to locate elements on a page for 2 main reasons:

- According to w3c ids are supposed to be unique on an html page. This makes ids a very explicit and reliable way to locate elements on the page.
- Also, all browsers also have highly efficient methods to get an object on the page using their ids. This makes `id` locators the fastest type of locator in Selenium.

Let us now look at how to use id locators using this html code as an example:

```
<form name="loginForm">
        Login Username: <input id="username" name="login" type="text" />
        Password:
<input id="password" name="password" type="password" />
        <input name="login" type="submit" value="Login" />
</form>
```

In the above code, the username and password text fields are can be set using their ids. The locators for them would be

```
driver.findElement(By.id(username));
```

Even though this is a great locator, it is not realistic for all elements on a page to have ids. The developers add ids to key elements on the page to better control the look and feel or provide the dynamic user interaction. However, ids should also be added to elements that are frequently interacted within tests … to make the pages more testable. Automated test script authors should consider adding, or requesting addition of, ids to these key elements on the page.

In some cases, the ids on an element cannot be reliably used in a test. For instance, if you are displaying objects stored in the database, the objects ids could contain the database id in it, for instance `book1347` could be the id for book who's database id is 1347. In such situations it may not be possible to predict and use the ids and so you may need to use other ways described below to find elements.

Finding elements by name

Generally ids are added to elements when they want to be referenced from css or javascript and names are added to form fields. When referencing element from javascript either can be used. From a test automation standpoint, whenever id is not available/ usable, you should try to use the name instead.

Using the same example above, the way you would find the submit button would be:

```
driver.findElement(By.name("login"));
```

There is one big difference between the id and name attributes though … name attributes don't have to be unique in a page. If there are multiple elements with the same name, then the first element in the page is selected. So, in this example, if another button or form named "login" was present of added later, it could cause the test to fail.

Finding links by text

This locator identifies links by the text in them. Let us look at an example:

```
<html>
  <body>
        ...
        <a href="signin.html">Sign In</a> to modify your account details.
        ...
</body>
</html>
```

To click this hyperlink using the anchor tag's text, you can use the By.linkText() locator:

```
driver.findElement(By.linkText("Sign In"));
```

If there are multiple elements with text "Sign In", the first one is selected.

Btw, this is called linkText because it is used for hyperlinks. In Selenium if you used the link=textPattern locator, you could use it to locate other elements like div, span, td etc. In WebDriver, this locator works only for links.

Another common case is when we need to find links by a portion of the text it contains. In such cases you can find it by specifying the partial text. For example:

```
driver.findElement(By.partialLinkText("Sign"));
```

Finding elements by XPath

XPath is a very powerful language to express which element to identify. If you use it correctly, it can produce very reliable and low maintenance locators, but if you use it incorrectly, it can create very brittle test cases.

Let us see some examples:

```
<table name="cart">
        <tr id="item1">
                <td class="name item">Mp3 Download: foobar.mp3</td>
                <td class="name     item"><input name="qty" class="name
item formfield disabled" /></td>
        </tr>
        <tr id="item2">
                <td class="name item">Mp3 Player</td>
                <td class="name
item"><input id="item2_quantity" name="qty" class="name  item  formfield
required"type="text" /></td>
        </tr>
        ...
</table>
```

You can target the highlighted input field using the following XPath expressions:

```
//table/tr[2]/td/input
//input[@id='item2_quantity']
(//table[@name='cart']//input)[2]
//input[contains(@class,'required')]
//input[contains(@class,'required') and type='text']
```

And for the last option, your java call would look like:

```
driver.findElement(By.xpath("//input[contains(@class='required')                and
type='text']"));
```

As you can guess, some of these expressions will not be as reliable as others. Of these `//table/tr[2]/td/input` is the worst because it would break even with slightest modification to the page structure. It can take some time to learn XPath if you aren't familiar with it, but it is worth the time if you plan to spend a lot of time writing UI automated tests. In any case do not rely on tools, including selenium IDE, to generate the right xpath expression for

you. They can help you get started but they are usually bad at identifying the more reliable XPaths.

XPath namespaces

There are some special cases you should be aware of when you work with XPath, like when you are trying to interact with SVG. Like in the example here, the html looks something like this:

```
<div id="svgchart">
    ...
        <svg xmlns="http://www.w3.org/2000/svg">
                <g>
                        <path .../>
                </g>
                ...
        </svg>
    ..
.
</div>
```

Here, if you use XPath like:

```
//svg
```

you will get `ERROR org.openqa.selenium.NoSuchElementException: Unable to locate element.` This is because the svg element is in a different namespace. You will have to specify your xpath with the namespace uri like this instead:

```
//*[local-name()='svg' and namespace-uri()='http://www.w3.org/2000/svg']
```

XPath performance

So, if XPath's are so versatile, why doesn't everyone prefer these? It's because they are often the slowest, especially in older versions of IE! There are some ways you can make XPath's faster, but they still are a few times slower than ids or names.

Finding elements by CSS

`css` locators can be used to identify a large number of elements on a page. Let us look at this html snippet for instance:

```
<table name="cart">
```

```
        <tr id="item1">
                <td class="label">Mp3 Download: foobar.mp3</td>
                <td class="item"><input name="qty" class="formfield
disabled" /></td>
        </tr>
        <tr id="item2">
                <td class="label">Mp3 Player</td>
                <td class="item"><input id="item2_quantity" name="qty" class="formfield
required" type="text" /></td>
        </tr>
        ...
</table>
```

In this case the css locator variations possible for the highlighted input field are:

```
input.required
input[class~='required']
input.required[type='text']
#item2_quantity
input#item2_quantity
```

The css locator may not be as expressive as XPath, but it generally executes faster.

Finding elements by class

This is more of a convinience mechanism to identify elements. In the css example above, instead of using

```
driver.findElements(By.css("input[class~='required']"));
```

you could use

```
driver.findElements(By.class("required"));
```

Getting elements by DOM

DOM stands for Document Object Model. DOM is convention for representing objects in HTML documents.

```
<form id="loginForm">
```

```
        Login                                                    Username:
<input id="username" name="username" type="text" />
        Password: <input name="password" type="password" />
        <input name="login" type="submit" value="Log in" />
</form>
```

In this page, the dom expression for the highlighted input field would be:

```
document.forms[0].elements[0]
document.forms['loginForm'].elements['username']
document.forms['loginForm'].username
document.getElementById('username')
```

For those who have used Selenium 1 API, you would expect to find a By.dom() equivalent api, but it doesn't exist. However, you still can get a handle to these elements using dom expressions by using the following code snippet instead:

```
1 ...
2         driver.findElement(byDom("document.forms[0].elements[0]"));
3 ...
4
5 public By byDom(String domExpression) {
        final Object o = ((JavascriptExecutor)
6 driver).executeScript("return " + domExpression + ";");
7
8         if (o instanceof WebElement) {
9                 return new By() {
                        @Overrid
10 e
                        public List<WebElement>
11 findElements(SearchContext searchContext) {
                                return new ArrayList<WebElement>()
12 {
13                                      {
                                                add((WebElement)
14 o);
15                                      }
16                              };
17                      }
```

```
18                      };
19              }
20 }
```

The three key aspects of this code that you should note,

- The driver can be casted to `JavascriptExecutor` and other interfaces which will provide additional capabilities.
- `executeScript()` in `JavascriptExecutor` returns an object which can be casted to a `WebElement` if the javascript expression returns a dom element.
- You can create your own implementation of `By`

Some things to note for Selenium 1 users

- You will notice that Implicit locators are not supported in WebDriver.
- When using name in Selenium 1, you could add other attributes in the expressions to filter the results. For example `name=login type=text` was valid. You cannot do that any more with the By.name() call.

`findElement()` and `findElements()`

As you might have noticed in the examples above, the `By` locators were used within `findElement` and `findElements` methods. The purpose of these methods is to return a `WebElement`, in case of `findElement`, and a list of `WebElement`s in the case of `findElements`.

The `WebElement` represents an html element on the page. In the next section you will see that it is this `WebElement` object that you would interact with or read attributes of for validations etc.

When you use these methods on the `WebDriver` object, the scope within with the elements are located is the entire page.

Getting child elements

The `WebElement` interface also has `findElement()` and `findElements()` methods. In this case, it is within the scope of this parent element that child elements are located. This is useful when you want to narrow the scope to find several elements that you would interact with. In most cases, narrowing the scope improves the execution times for locating the elements.

selenium-webdriver 提供了强大的元素定位方法，支持以下三种方法：

单个对象的定位方法

多个对象的定位方法

层级定位

注意：

selenium-webdriver 通过 findElement()\findElements()等 find 方法调用"By"对象来定位和查询元素。By 类只是提供查询的方式进行分类。findElement 返回一个元素对象否则抛出异常，findElements 返回符合条件的元素 List，如果不存在符合条件的就返回一个空的 list。

一、定位单个元素

A.使用 className 进行定位

当所定位的元素具有 class 属性的时候我们可以通过 classname 来定位该元素。

例：下面的例子定位页面上 class 为"username"的 li。

```
WebElement element = driver.findElement(By.className("username"));
System.out.println(element.getTagName());
```

输出结果：Li

B.使用 id 属性定位

例：<input id="passport_user" type="text" value="" title="用户名/邮箱" name="passport_user">

```
WebElement element = dr.findElement(By.id("passport_user"));
System.out.println(element.getAttribute("title"));
```

输出结果：用户名/邮箱

C.使用 name 属性定位

例：<input id="passport_user" type="text" value="" title="用户名/邮箱" name="passport_user">

```
WebElement e = dr.findElement(By.name("passport_user"));
```

D.使用 css 属性定位

例：<input id="passport_user" type="text" value="" title="用户名/邮箱" name="passport_user">

```
WebElement e1 = dr.findElement(By.cssSelector("#passport_user"));
```

详解：

1.css 之后代选择器

```
<p>
    <em>location1</em>
</p>
<ol>
```

```
        <li><em>location2</em></li>
</ol>
```

可以通过 css=p em 这个可以选中文本为 location1 的 em 元素
css=ol em 这个可以选中文本为 location2 的 em 元素
css 后代选择器和 xpath 中//div//a 一样：取得所有 div 下面所有的 a 节点。这个是忽略了父子节点

```
<div>
        <p><em>location1</em></p>
</div>
<div>
<ol>
        <li><strong><em>location2</em></strong></li>
        <li><em>location3</em></li>
        <li><em>location4</em></li>
</ol>
</div>
```

可以通过 css=p>em 来定位 location1
css 之父子节点选择器给后代选择器加了一个限制，类似 xpath 中//div/p/em：所有 div 下的子元素 p 的子元素 em。
css=li+li em 来定位 location3，location4 的 em
css=li+strong+em 来定位文本为 location2 的 em

2.css 之 id 选择器

```
<input id="location1" type="button"/>
<input id="location2" type="radio"/>
```

通过 css=#location1 来定位 type 为 button 的按钮
通过 css=#location2 来定位 type 为 radio 的单选框

3.css 之类选择器

```
<input class="location1" type="button" value="确定"/>
<input class="location2" type="button" value="取消"/>
```

通过 css=input.location1 来选择 value 值为确定的按钮
通过 css=input.location2 来选择 value 值为取消的按钮

4.css 之属性选择器

```
<input class="location1" type="button" value="确定"/>
<input class="location2" type="button" />
```

通过 css=[class=location1]可以定位第一个按钮
通过 css=[class~=1]可以定位第一个按钮
通过 css=[value="确定"]可以定位第一个按钮
通过 css=input[class="location"]可以定位第二个按钮

E.按标记（tag）名称查找
元素的 DOM 标记名称
<iframe src="..."></iframe>
WebElement frame = driver.findElement(By.tagName("iframe"));

F.按链接文本查找
cheese>
WebElement cheese = driver.findElement(By.linkText("cheese"));

按部分链接文本查找
search for cheese>
WebElement cheese = driver.findElement(By.partialLinkText("cheese"));

G.使用 XPATH 定位
例 ： <input id="passport_user" type="text" value="" title="用户名/邮箱" name="passport_user">
WebElement element =dr.findElement(By.xpath("//input[@id='passport_user']"));

parent::返回父节点
following::返回此节点后面的兄弟节点
preceding::返回此节点前面的兄弟节点

div>
<input id="location1" type="button" />
<input type="button" />

通过 id 为 location1 可以很随意的定位到兄弟节点
//div/input[@id='location1']/following::input

也可以通过 location1 很随意的定位到父类节点 div。
//div/input[@id='location1']/parent::div。

也可以通过索引为 2 的 input 定位到 id 为 location1 的 input
//div/input[2]/preceding-sibling::input

有几个非常有用的 Firefox 插件，有助于发现一个元素的 XPath：
XPath Checker - suggests XPath and can be used to test XPath results.
Firebug - XPath suggestions are just one of the many powerful features of this very useful add-on.
XPath Checker - 建议 XPath 并可以用于测试 XPath 的结果
Firebug - XPath 建议仅仅是这非常有用的插件的许多强有力特征中的一个。

参考网站：http://www.w3school.com.cn/xpath/index.asp

```
二、定位多个元素
//定位到所有<input>标签的元素，然后输出他们的 id
        List<WebElement> element = driver.findElements(By.tagName("input"));
        for (WebElement  e : element)
                System.out.println(e.getAttribute("id"));

三、层级定位
层级定位的思想是先定位父元素，然后再从父元素中精确定位出其我们需要选取的子元素。
层级定位一般的应用场景是无法直接定位到需要选取的元素，但是其父元素比较容易定位，通过定位
父元素再遍历其子元素选择需要的目标元素，或者需要定位某个元素下所有的子元素。

下面的代码演示了如何使用层级定位 class 为"login"的 div，然后再取得它下面的所有 label，并打
印出他们的文本
//定位 class 为"login"的 div，然后再取得它下面的所有 label，并打印出他们的值
        WebElement element = driver.findElement(By.className("login"));
        List<WebElement> el = element.findElements(By.tagName("label"));
        for(WebElement e : el)
                System.out.println(e.getText());

四、使用 Javascript
你可以执行任何 Javascript,以找到一个元素,只要你找到一个 DOM 元素,它将自动转换为 WebElement
对
象。
WebElement element = (WebElement) ((JavascriptExecutor)driver).executeScript("return
$('.cheese')[0]");
```

操作页面元素

http://blog.sina.com.cn/s/blog_6966650401012a7q.html
http://blog.sina.com.cn/s/blog_6966650401012pvv.html
http://www.qaautomation.net/?p=443

```
一、输入框（text field or textarea）
//找到输入框元素：
WebElement element = driver.findElement(By.id("passwd-id"));
//将输入框清空：
element.clear();
//在输入框中输入内容：
element.sendKeys(“test”);
//获取输入框的文本内容：
element.getText();
```

二、下拉选择框(Select)

```
//找到下拉选择框的元素：
Select select = new Select(driver.findElement(By.id("select")));

//选择对应的选择项：
select.selectByVisibleText(“mediaAgencyA”);
或
select.selectByValue(“MA_ID_001”);

//不选择对应的选择项：
select.deselectAll();
select.deselectByValue(“MA_ID_001”);
select.deselectByVisibleText(“mediaAgencyA”);
或者获取选择项的值：
select.getAllSelectedOptions();
select.getFirstSelectedOption();
```

对下拉框进行操作时首先要定位到这个下拉框，new 一个 Selcet 对象，然后对它进行操作

例如：

以 http://passport.51.com/reg2.5p 这个页面为例。这个页面中有 4 个下拉框，下面演示 4 种选中下拉框选项的方法。

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class SelectsStudy {
        public static void main(String[] args) {
                System.setProperty("webdriver.firefox.bin","D:\\Program Files\\Mozilla Firefox\\firefox.exe");
                WebDriver dr = new FirefoxDriver();
                dr.get("http://passport.51.com/reg2.5p");

                //通过下拉列表中选项的索引选中第二项，即 2011 年
                Select selectAge = new Select(dr.findElement(By.id("User_Age")));
                selectAge.selectByIndex(2);//Select.selectByIndex

                //通过下拉列表中的选项的 value 属性选中"上海"这一项
                Select selectShen = new Select(dr.findElement(By.id("User_Shen")));
                selectShen.selectByValue("上海");//Select.selectByValue
```

```
                //通过下拉列表中选项的可见文本选 中"浦东"这一项
                Select selectTown = new Select(dr.findElement(By.id("User_Town")));
                selectTown.selectByVisibleText("浦东");Select.selectByVisibleText

                //这里只是想遍历一下下拉列表所有选项，用 click 进行选中选项
                Select selectCity = new Select(dr.findElement(By.id("User_City")));
                for(WebElement e : selectCity.getOptions())//Select.getOptions()
                        e.click();
        }
}
```

三、单选项(Radio Button)

```
//找到单选框元素：
WebElement bookMode =driver.findElement(By.id("BookMode"));
//选择某个单选项：
bookMode.click();
//清空某个单选项：
bookMode.clear();
//判断某个单选项是否已经被选择：
bookMode.isSelected();
```

四、多选项(checkbox)

```
//多选项的操作和单选的差不多：
WebElement checkbox =driver.findElement(By.id("myCheckbox."));
checkbox.click();
checkbox.clear();
checkbox.isSelected();
checkbox.isEnabled();
```

五、按钮(button)

```
//找到按钮元素：
WebElement saveButton = driver.findElement(By.id("save"));
//点击按钮：
saveButton.click();
//判断按钮是否 enable:
saveButton.isEnabled ();
```

六、左右选择框

也就是左边是可供选择项，选择后移动到右边的框中，反之亦然。

例如：

```
Select lang = new Select(driver.findElement(By.id("languages")));
lang.selectByVisibleText(“English”);
WebElement addLanguage =driver.findElement(By.id("addButton"));
addLanguage.click();
```

七、弹出对话框(Popup dialogs)

```
Alert alert = driver.switchTo().alert();
alert.accept();
alert.dismiss();
alert.getText();
```

八、表单(Form)

Form 中的元素的操作和其它的元素操作一样，对元素操作完成后对表单的提交可以：

```
WebElement approve = driver.findElement(By.id("approve"));
approve.click();
```

或

```
approve.submit();//只适合于表单的提交
```

九、上传文件 (Upload File)

上传文件的元素操作：

```
WebElement adFileUpload = driver.findElement(By.id("WAP-upload"));
String filePath = "C:\test\\uploadfile\\media_ads\\test.jpg";
adFileUpload.sendKeys(filePath);
```

十、拖拉(Drag andDrop)

```
WebElement element =driver.findElement(By.name("source"));
WebElement target = driver.findElement(By.name("target"));
(new Actions(driver)).dragAndDrop(element, target).perform();
```

例如：下面这个页面是一个演示拖放元素的页面，你可以把左右页面中的条目拖放到右边的 div 框中。

http://koyoz.com/demo/html/drag-drop/drag-drop.html

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class DragAndDrop {
        public static void main(String[] args) {
                System.setProperty("webdriver.firefox.bin","D:\\Program
Files\\Mozilla Firefox\\firefox.exe");
```

```
                WebDriver dr = new FirefoxDriver();
                dr.get("http://koyoz.com/demo/html/drag-drop/drag-drop.html");

                //首先 new 出要拖入的页面元素对象和目标对象，然后进行拖入。
                WebElement    element = dr.findElement(By.id("item1"));
                WebElement    target = dr.findElement(By.id("drop"));
                (new Actions(dr)).dragAndDrop(element, target).perform();

                //利用循环把其它 item 也拖入
                String id="item" ;
                for(int i=2;i<=6;i++){
                        String item = id+i;
                        (new
Actions(dr)).dragAndDrop(dr.findElement(By.id(item)), target).perform();
                }
        }
}
```

代码很简单，需要注意的是(new Actions(dr)).dragAndDrop(element, target).perform();这句话中，dragAndDrop(element, target)这个方法是定义了“点击 element 元素对象，然后保持住，直到拖到目标元素对象里面才松开”这一系列动作的 Actions,如果你不调用 perform()方法，这个 Actions 是不会执行的。

```
十一、导航 (Navigationand History)
//打开一个新的页面：
  driver.navigate().to("http://www.example.com");
//通过历史导航返回原页面：
driver.navigate().forward();
driver.navigate().back();
```

Frame 处理

http://blog.sina.com.cn/s/blog_6966650401012as9.html
http://blog.sina.com.cn/s/blog_6f8bd7460100zyma.html
http://www.51testing.com/?uid-205906-action-viewspace-itemid-250049

有时候我们在定位一个页面元素的时候发现一直定位不了，反复检查自己写的定位器没有任何问题，代码也没有任何问题。这时你就要看一下这个页面元素是否在一个 iframe 中，这可能就是找不到的原因之一。
如果你在一个 default content 中查找一个在 iframe 中的元素，那肯定是找不到的。反之你在一个 iframe 中查找另一个 iframe 元素或 default content 中的元素，那必然也定位不到。
selenium webdriver 中提供了进入一个 iframe 的方法：

```
WebDriver org.openqa.selenium.WebDriver.TargetLocator.frame(String nameOrId)
```

也提供了一个返回 default content 的方法：

WebDriver org.openqa.selenium.WebDriver.TargetLocator.defaultContent()
这样使我们面对 iframe 时可以轻松应对。
switch_to 方法会 new 1 个 TargetLocator 对象，使用该对象的 frame 方法可以将当前识别的”主体”移动到需要定位的 frame 上去。

以下面的 html 代码为例，我们看一下处现 iframe。

这个 2 个页面放桌面
Html 代码
main.html

```
<html>
        <head>
                <title>FrameTest</title>
        </head>
        <body>
<div id = "id1">this is a div!</div>
                <iframe id = "frame"   frameborder="0" scrolling="no"
style="left:0;position:absolute;" src = "frame.html"></iframe>
        </body>
</html>
```

frame.html

```
<html>
        <head>
                <title>this is a frame!</title>
        </head>
        <body>
<div id = "div2">this is a frame, too!</div>
<label>input:</label>
<input id = "input2" value='frame VALUE'>a frame</input>
        </body>
</html>
```

Java 代码

```
package com.test;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;


public class Test_frame {
public static void main(String[] args) {
```

```
String url = "file:///C:/Documents and Settings/fei yong/桌面/main.html";
      //打开 chrome
      WebDriver dr = new ChromeDriver();
                dr.get(url);

                //在 default content 定位 id="id1"的 div
                dr.findElement(By.id("id1"));

                //此时，没有进入到 id="frame"的 frame 中时，以下两句会报错
                //dr.findElement(By.id("div1"));//报错
                //dr.findElement(By.id("input1"));//报错

                //进入 id="frame"的 frame 中，定位 id="div1"的 div 和 id="input1"的输入框。
                dr.switchTo().frame("frame");
                dr.findElement(By.id("div2"));
                dr.findElement(By.id("input2"));
                System.out.println("div2.getTagName:"+dr.findElement(By.id("div2")).get
TagName()+";");
                System.out.println("input2.getTagName:"+dr.findElement(By.id("input2"))
.getTagName());
                System.out.println("input2.getText:"+dr.findElement(By.id("input2")).ge
tText());

                //此时，没有跳出 frame，如果定位 default content 中的元素也会报错。
                //dr.findElement(By.id("id1"));//报错

                //跳出 frame,进入 default content;重新定位 id="id1"的 div
                dr.switchTo().defaultContent();
                dr.findElement(By.id("id1"));
                System.out.println("id1.getText:"+dr.findElement(By.id("id1")).getText(
));

                dr.quit();
}
}
```

页面输出：

```
Started ChromeDriver
port=50032
version=19.0.1068.0
log=E:\android\selenium\test_wdng_java\chromedriver.log
div2.getTagName:div;
input2.getTagName:input
input2.getText:
id1.getText:this is a div!
```

窗口处理

http://www.thoughtworks-studios.com/twist/2.3/help/how_do_i_handle_popup_in_selenium2.html

弹出窗口

http://blog.sina.com.cn/s/blog_6966650401012au9.html

```
package com.testscripts;

import java.util.Iterator;
import java.util.Set;

import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class WebDriver_PopUpWindowTest {

    @Test
    public void test1() {
        System.setProperty("webdriver.firefox.bin",
                "E:\\Program Files\\Mozilla Firefox\\firefox.exe");
        WebDriver dr = new FirefoxDriver();
        String url = "file:///D:/selenium/MyTest/AUT/testWindow.html";
        dr.get(url);
        dr.findElement(By.id("baidu")).click();

        // 得到当前窗口的句柄
        String currentWindow = dr.getWindowHandle();
        // 得到所有窗口的句柄
        Set<String> handles = dr.getWindowHandles();

        Iterator<String> it = handles.iterator();
        while (it.hasNext()) {
            if (currentWindow == it.next())
                continue;
            WebDriver window = dr.switchTo().window(it.next());
```

```
            System.out.println("title,url = " + window.getTitle() + ","
                    + window.getCurrentUrl());
        }
    }

}
```

对话框

http://blog.sina.com.cn/s/blog_6966650401012aut.html

```
package com.testscripts;

import java.util.List;

import org.junit.Test;
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class WebDriver_AlertTest {

    @Test
    public void test1() {

        String url = "file:///D:/selenium/MyTest/AUT/testDialog.html";
        // 打开firefox
        WebDriver driver = new FirefoxDriver();
        driver.get(url);

        // 点击第一个按钮，输出对话框上面的文字，然后叉掉
        driver.findElement(By.id("alert")).click();
        Alert alert = driver.switchTo().alert();
        String text = alert.getText();
        System.out.println("alert:" + text);
        alert.dismiss();
        // 点击第二个按钮，输出对话框上面的文字，然后点击确认
        driver.findElement(By.id("confirm")).click();
        Alert confirm = driver.switchTo().alert();
```

```
        String text1 = confirm.getText();
        System.out.println("confirm:" + text1);
        confirm.accept();
        // 点击第三个按钮，输入你的名字，然后点击确认，最后
        driver.findElement(By.id("prompt")).click();
        Alert prompt = driver.switchTo().alert();
        String text2 = prompt.getText();
        System.out.println("prompt:" + text2);
        prompt.sendKeys("allen");
        prompt.accept();

        // driver.quit();
    }
}
```

Cookie 操作

http://blog.sina.com.cn/s/blog_6966650401012aw0.html

一个 Cookies 主要属性有：所在域、name、value、有效日期和路径

Java 代码

```
import java.util.Set;

import org.openqa.selenium.Cookie;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class CookiesStudy {
          static void main(String[] args) {
                    System.setProperty("webdriver.firefox.bin","D:\\Program
Files\\Mozilla Firefox\\firefox.exe");
                    WebDriver dr = new FirefoxDriver();
                    dr.get("http://www.51.com");
                    //增加一个 name = "name",value="value"的 cookie
                    Cookie cookie = new Cookie("name", "value");
                    dr.manage().addCookie(cookie);
                    //得到当前页面下所有的 cookies，并且输出它们的所在域、name、
value、有效日期和路径
                    Set<Cookie> cookies = dr.manage().getCookies();
```

```
                    System.out.println(String.format("Domain -> name -> value ->
expiry -> path"));
                    for(Cookie c : cookies)
                    System.out.println(String.format("%s -> %s -> %s -> %s -> %s",
c.getDomain(), c.getName(), c.getValue(),c.getExpiry(),c.getPath()));

                    //删除 cookie 有三种方法
                    //第一种通过 cookie 的 name
                    dr.manage().deleteCookieNamed("CookieName");
                    //第二种通过 Cookie 对象
                    dr.manage().deleteCookie(cookie);
                    //第三种全部删除
                    dr.manage().deleteAllCookies();
}
}
```

小结：

上面的代码首先在页面中增加了一个 cookie,然后遍历页面的所有 cookies，并输出他们的主要属性。最后就是三种删除 cookie 的方法。

执行 JavaScript

http://blog.sina.com.cn/s/blog_6966650401012dxv.html

在用 selenium 1.X 的时候常常会用到 geteval_r()方法来执行一段 js 脚本来对页面进行处理。

当然 selenium webdriver 也提供这样的一个方法:JavascriptExecutor.executeScript(string)

例如：

```
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;

public class   SimpleExample {
      public static void main(String[] args) {
          ChromeDriver driver = new ChromeDriver();
driver.executeAsyncScript("arguments[0](); alert('Hello')");
try {
Thread.sleep(3000);
} catch (InterruptedException e) {
e.printStackTrace();
}
driver.switchTo().alert().accept();
```

```
    }
}
```

上面是一个最简单的例子，打开一个浏览器，然后弹层一个 alert 框。注意这里的 driver 要被强制转换成 JavascriptExecutor。

下面演示在打开 51.com 首页如何得到帐号输入框中显示的字符，并打印输出。

```
<div class="input_wrap">
    <input id="passport_51_user" type="text" value="" tabindex="1" title="用户名/彩虹号/邮箱" name="passport_51_user" style="color: rgb(0, 0, 0)
</div>
```

```
package com.test;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Test_JsExecutor {
public static void main(String[] args) {
String url = "http://www.51.com";
//打开 chrome
WebDriver dr = new ChromeDriver();
String js = "var user_input  =
document.getElementByIdx_x_x_x(\"passport_51_user\").title;return user_input;";
String title =  (String)((JavascriptExecutor)dr).executeScript( js);
System.out.println(title);

                dr.quit();
}
}
```

输出结果为：用户名/彩虹号/邮箱

其他用例：

```
        JavascriptExecutor js = (JavascriptExecutor) driver;
        js.executeScript("(function(){
                                                                    inventoryGridMg
r.setTableFieldValue('"+ inventoryId  + "','"

                    + fieldName + "','"+ value + "');
                                                            }
                                                )()"
                                        );
```

拖拽操作

http://blog.sina.com.cn/s/blog_6966650401012h1l.html

如何把一个元素拖放到另一个元素里面

下面这个页面是一个演示拖放元素的页面，你可以把左右页面中的条目拖放到右边的 div 框中。

http://koyoz.com/demo/html/drag-drop/drag-drop.htm

Java 代码

```
package com.test;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;


public class Test_DragAndDrop {
public static void main(String[] args) {
WebDriver dr = new ChromeDriver();
        dr.get("http://koyoz.com/demo/html/drag-drop/drag-drop.html");
        //首先 new 出要拖入的页面元素对象 element 和目标对象 target，然后进行拖入。
WebElement   element = dr.findElement(By.id("item1"));
WebElement   target = dr.findElement(By.id("drop"));
(new Actions(dr)).dragAndDrop(element, target).perform();
//利用循环把其它 item 也拖入
String id="item" ;
for(int i=2;i<=6;i++){
String item = id+i;
```

```
(new Actions(dr)).dragAndDrop(dr.findElement(By.id(item)), target).perform();
}
      //dr.quit();
}
}
```

==========

附：如何利用 Actions 类模拟鼠标和键盘的操作

actions 类，主要定义了一些模拟用户的鼠标 mouse，键盘 keyboard 操作。对于这些操作，使用 perform()方法进行执行。

actions 类可以完成单一的操作，也可以完成几个操作的组合。

单一的操作

单一的操作是指鼠标和键盘的一个操作。如鼠标左键按下、弹起或输入一个字符串等。

前面涉及到鼠标键盘操作的一些方法，都可以使用 actions 类中的方法实现，比如：click，sendkeys。

```
WebElement   element = dr.findElement(By.id("test"));
  WebElement   element1 = dr.findElement(By.id("test1"));
element.sendKeys("test");
element1.click;
```

用 Actions 类就可以这样实现：

```
//新建一个 action
Actions action=new Actions(driver);
//操作
WebElement element=dr.findElement(By.id("test"));
WebElement element1=dr.findElement(By.id("su"));
action.sendKeys(element,"test").perform();
action.moveToElement(element1);
action.click().perform();
```

看起来用 Actions 类实现 click 和 sendKeys 有点烦索

组合操作

组合操作就是几个动作连在一起进行操作。如对一个元素的拖放。

```
  (new Actions(dr)).dragAndDrop(dr.findElement(By.id(item)), target).perform();
```

可以直接调用 dragAndDrip()方法，也可以像下面濱示的一样把几个操作放一起实现

```
Action dragAndDrop = builder.clickAndHold(someElement)
                .moveToElement(otherElement)
```

```
                .release(otherElement)
                .build().perform();
```

其他鼠标或键盘操作方法可以具体看一下 API 里面的
org.openqa.selenium.interactions.Actions 类

测试脚本设计

验证结果（Validating Results）

Assert vs. Verify

When should you use an assert command and when should you use a verify command? This is up to you. The difference is in what you want to happen when the check fails. Do you want your test to terminate, or to continue and simply record that the check failed?

Here's the trade-off. If you use an assert, the test will stop at that point and not run any subsequent checks. Sometimes, perhaps often, that is what you want. If the test fails you will immediately know the test did not pass. Test engines such as TestNG and JUnit have plugins for commonly used development environments (Chap 5) which conveniently flag these tests as failed tests. The advantage: you have an immediate visual of whether the checks passed. The disadvantage: when a check does fail, there are other checks which were never performed, so you have no information on their status.

In contrast, verify commands will not terminate the test. If your test uses only verify commands you are guaranteed (assuming no unexpected exceptions) the test will run to completion whether the checks find defects or not. The disadvantage: you have to do more work to examine your test results. That is, you won't get feedback from TestNG or JUnit. You will need to look at the results of a console printout or a log output. And you will need to take the time to look through this output every time you run your test. If you are running hundreds of tests, each with its own log, this will be time-consuming, and the immediate feedback of asserts will be more appropriate. Asserts are more commonly used then verifys due to their immediate feedback.

参考：

http://www.51testing.com/?uid-49689-action-viewspace-itemid-205958

想使用 selenium RC 做浏览器兼容性测试,使用了 eclipse+junit+ant 作为测试平台.

在 selenium 中同时支持 assert 和 verify 两种断言模式.

如果使用 assert,当遇到错误时,test case 不会再继续被执行下去；而如果使用 verify,当遇到错误时,test case 仍会继续运行,直到完成.

但在 junit 中创建测试代码时发现,junit 不识别 verify 方法.

从网上搜索了 N 多次,都没有找到彻底的解决方法,后来咨询了单位一个做开发人员,这个问题总得有眉目了.

1. 首先不使用 junit 中的 testcase 类,而是使用 com.thoughtworks.selenium.SeleneseTestCase.

import com.thoughtworks.selenium.*;

2.使用上述类就可以使用 verify 方法了,需要注意一点是:

This class adds a number of "verify" commands, which are like "assert" commands, but they don't stop the test when they fail. Instead, verification errors are all thrown at once during tearDown.

所以需要在 teardown 方法中,加后 check verification 的方法

```
public void tearDown() throws Exception {

        selenium.stop();

        checkForVerificationErrors();

}
```

注意: checkForVerificationErrors();和 selenium.stop();的位置不要反了啊. 如果反了,当遇到 verificationerror 的情况,这个测试就不能关闭不了.

3.teardown()方法,使用 junit 默认定义 teardown 方法是如下

```
protected void tearDown() throws Exception {
```

```
        selenium.stop();

        checkForVerificationErrors();

}
```

但 junit 总是报一个错误:

Cannot reduce the visibility of the inherited method from SeleneseTestCase

原因是:

因为接口定义的方法默认是 public 的，意思就是你没有任何访问修饰符的情况下，系统给接口的方法加上了一个 public 的访问修饰符。你 Test 实现了接口，并且实现了接口定义的方法，于是方法的访问修饰符只能比接口的访问修饰符高，但是类的默认访问修饰符是 freidnly，降低了访问级别，所以会报错. 所以你 Test 实现的方法前面加上 public 就对了(具体不知道什么意思,咨询了一个开发人员后说的,哈哈,本人不懂代码哦.)

后来将 teardown()方法改成如下后,问题解决:

```
public void tearDown() throws Exception {

        selenium.stop();

        checkForVerificationErrors();

}
```

Trade-offs: *assertTextPresent*, *assertElementPresent*, *assertText*

You should now be familiar with these commands, and the mechanics of using them. If not, please refer to Chapter 3 first. When constructing your tests, you will need to decide

- Do I only check that the text exists on the page? (*verify/assertTextPresent*)
- Do I only check that the HTML element exists on the page? That is, the text, image, or other content is not to be checked, only the HTML tag is what is relevant. (*verify/assertElementPresent*)
- Must I test both, the element and its text content? (*verify/assertText*)

There is no right answer. It depends on the requirements for your test. Which, of course, depend on the requirements for the application

you're testing. If in doubt, use *assertText*since this is the strictest type of checkpoint. You can always change it later but at least you won't be missing any potential failures.

Verify/assertText is the *most specific test* type. This can fail if either the HTML element (tag) OR the text is not what your test is expecting. Perhaps your web-designers are frequently changing the page and you don't want your test to fail every time they do this because the changes themselves are expected periodically. However, assume you still need to check that *something* is on the page, say a paragraph, or heading text, or an image. In this case you can use *verify/assertElementPresent*. It will ensure that a particular type of element exists (and if using XPath can ensure it exists relative to other objects within the page). But you don't care what the content is. You only care that a specific element, say, an image, is at a specific location.

Getting a feel for these types of decisions will come with time and a little experience. They are easy concepts, and easy to change in your test.

Common Assertions using WebDriver

http://www.qaautomation.net/?p=498

Page Title

You can get the current page title simply by calling `getTitle()` on the WebDriver instance. Here is how a simple test would look like:

```
1 @Test
2 public void pageTitle() {
3         driver.get("http://referencewebapp.qaautomation.net/");
4         String pageTitle = driver.getTitle();
5         assertEquals("Current page title","Reference Web App - QA Automation", pageTitle);
6 }
```

Validations against an element

Text within an element

There are two approaches you can use for validating text within an element. The first approach is find the element using one of the locator strategies and then call `getText()` on the element object returned.

```
1 driver.get("http://referencewebapp.qaautomation.net/");
2 WebElement header = driver.findElement(By.id("header"));
3 assertEquals("Header text", "Reference Web App", header.getText());
```

The second approach is to search for an element using the text in the locator and check if it is not null.

```
1 driver.get("http://referencewebapp.qaautomation.net/");
2 WebElement header = driver.findElement(
3 By.xpath("//div[@id=\"header\" and normalize-space(.)=\"Reference Web App\"]"));
4 assertNotNull("Header text", header);
```

In this example I used `xpath` but you could use other locators like `linkText` or `partialLinkText`.

Attributes of the element

This is similar to validating text in the element. You would find the element on the page and then use `getAttribute`

```
1 driver.get("http://referencewebapp.qaautomation.net/");
2 WebElement submit = driver.findElement(By.name("submit"));
3 assertEquals("Submit button value", "Login", submit.getAttribute("value"));
```

Does text exist in the entire page

For those who start with Selenium 2, or who have used Selenium 1.x, you would expect to easily find a `doesTextExist` method. It will come as a bit of a shock that there isn't something as obvious in the API.

One option to get the text representation of the entire page is getting the text of the `body` element using something like

```
driver.findElement(By.tagName("body")).getText()
```

Unfortunately, getting text using this mechanism is **extremely** slow for large pages. The second option is to get the html content using`driver.getPageSource()` and then pass it to a html parser. One such a parser is JSoup.

- easy to use,
- is better performant for this sort of thing, and
- the output text is cleaner and easier do validations against

. Here is how you would use it:

```
1 public String getPageContent() {
2         Document doc = Jsoup.parse(driver.getPageSource());
3         return doc.text();
4 }
```

Just to compare the two approaches, getting document text in msn.com took over 5 secs using first approach and less than a second for the second.

Ofcourse for this code to work, you will have to download the jsoup library and put it in your classpath. Also if you are not developing in java you will have to find a different parser because jsoup is a java library that hasn't been ported to other languages.

Validating Javascript return values

There are various reasons why you might want to directly call a javascript method or get the value of a javascript variable on the page. You can do this using only those WebDriver instances that implement the `JavascriptExecutor` interface. The `JavascriptExecutor`interface has a `executeScript` method that you would pass your javascript call. If you expect a return value, your javascript snippet must have a `return`.

Let's look at an example:

```
1 driver.get(REFERENCE_APP_URL +"jsmethods.php");
2 String returnValue = (String) ((JavascriptExecutor) driver).executeScript("return getString();");
```

```
3 assertEquals("JS method with string return type", "foo", returnValue);
```

The return type for `executeScript` method is Object. That is because this script would return an object of different types based on what is returned. The various types of objects returned are Sting, Integer, Boolean, etc. It also can return a WebElement as we saw in the Locating page elements using WebDriver under DOM locator.

定位元素（Location Strategies）

选择定位策略（**Choosing a Location Strategy**）

There are multiple ways of selecting an object on a page. But what are the trade offs of each of these locator types? Recall we can locate an object using

- the element's ID
- the element's name attribute
- an XPath statement
- by a links text
- document object model (DOM)

Using an element ID or name locator is the most efficient in terms of test performance, and also makes your test code more readable, assuming the ID or name within the page source is well-named. XPath statements take longer to process since the browser must run its XPath processor. XPath has been known to be especially slow in Internet Explorer version 7. Locating via a link's text is often convenient and performs well. This technique is specific to links though. Also, if the link text is likely to change frequently, locating by the <a> element would be the better choice.

Sometimes though, you must use an XPath locator. If the page source does not have an ID or name attribute you may have no choice but to use an XPath locator. (DOM locators are no longer commonly used since XPath can do everything they can and more. DOM locators are available simply to support legacy tests.)

There is an advantage to using XPath that locating via ID or name attributes do not have. With XPath (and DOM) you can locate an object with respect to another object on the page. For example, if there is a link that must occur within the second paragraph within a

 section, you can use XPath to specify this. With ID and name locators, you can only specify that they occur on the page that is, somewhere on the page. If you must test that an image displaying the company logo appears at the top of the page within a header section XPath may be the better locator.

定位动态元素（**Locating Dynamic Elements**）

As was described earlier in the section on types of tests, a dynamic element is a page element whose identifer varies with each instance of the page. For example,

```
<a class="button" id="adminHomeForm" onclick="return
oamSubmitForm('adminHomeForm',
    'adminHomeForm:_ID38');" href="#">View Archived Allocation
Events</a>
```

This HTML anchor tag defines a button with an ID attribute of "adminHomeForm". It's a fairly complex anchor tag when compared to most HTML tags, but it is still a static tag. The HTML will be the same each time this page is loaded in the browser. Its ID remains constant with all instances of this page. That is, when this page is displayed, this UI element will always have this Identifier. So, for your test script to click this button you simply need to use the following selenium command.

```
click       adminHomeForm
```

Or, in Selenium 1.0

```
selenium.click("adminHomeForm");
```

Your application, however, may generate HTML dynamically where the identifier varies on different instances of the webpage. For instance, HTML for a dynamic page element might look like this.

```
<input type="checkbox" value="true"
id="addForm:_ID74:_ID75:0:_ID79:0:checkBox"
    name="addForm:_ID74:_ID75:0:_ID79:0:checkBox"/>
```

This defines a checkbox. Its ID and name attributes (both addForm:_ID74:_ID75:0:_ID79:0:checkBox) are dynamically generated values. In this case, using a standard locator would look something like the following.

```
click       addForm:_ID74:_ID75:0:_ID79:0:checkBox
```

Or, again in Selenium-RC

```
selenium.click("addForm:_ID74:_ID75:0:_ID79:0:checkBox");
```

Given the dynamically generated Identifier, this approach would not work. The next time this page is loaded the Identifier will be a different value from the one used in the Selenium command and therefore, will not be found. The click operation will fail with an "element not found" error.

To correct this, a simple solution would be to just use an XPath locator rather than trying to use an ID locator. So, for the checkbox you can simply use

```
click       //input
```

Or, if it is not the first input element on the page (which it likely is not) try a more detailed XPath statement.

```
click       //input[3]
```

Or

```
click       //div/p[2]/input[3]
```

If however, you do need to use the ID to locate the element, a different solution is needed. You can capture this ID from the website before you use it in a Selenium command. It can be done like this.

```
String[] checkboxids  = selenium.getAllFields(); // Collect all input
IDs on page.
             for(String checkboxid:checkboxids) {
                    if(checkboxid.contains("addForm")) {
                selenium.click(expectedText);
            }
             }
```

This approach will work if there is only one check box whose ID has the text 'expectedText' appended to it.

定位 Ajax 元素（Locating Ajax Elements）

As was presented in the Test Types subsection above, a page element implemented with Ajax is an element that can be dynamically refreshed without having to refresh the entire page. The best way to locate and verify an Ajax element is to use the Selenium 2.0 WebDriver API. It was specifically designed to address testing of Ajax elements where Selenium 1 has some limitations.

In Selenim 2.0 you use the waitFor() method to wait for a page element to become available. The parameter is a By object which is how WebDriver implements locators. This is explained in detail in the WebDriver chapters.

To do this with Selenium 1.0 (Selenium-RC) a bit more coding is involved, but it isn't difficult. The approach is to check for the element, if it's not available wait for a predefined period and then again recheck it. This is then executed with a loop with a predetermined time-out terminating the loop if the element isn't found.

Let's consider a page which brings a link (link=ajaxLink) on click of a button on page (without refreshing the page) This could be handled by Selenium using a *for* loop.

```
// Loop initialization.
for (int second = 0;; second++) {

    // If loop is reached 60 seconds then break the loop.
    if (second >= 60) break;

    // Search for element "link=ajaxLink" and if available then break
loop.
    try { if (selenium.isElementPresent("link=ajaxLink")) break; }
catch (Exception e) {}

    // Pause for 1 second.
    Thread.sleep(1000);

}
```

This certainly isn't the only solution. Ajax is a common topic in the user forum and we recommend searching previous discussions to see what others have done.

封装调用（Wrapping Selenium Calls）

As with any programming, you will want to use utility functions to handle code that would otherwise be duplicated throughout your tests. One way to prevent this is to wrap frequently used selenium calls with functions or class methods of your own design. For example, many tests will frequently click on a page element and wait for page to load multiple times within a test.

```
selenium.click(elementLocator);
```

```
selenium.waitForPageToLoad(waitPeriod);
```

Instead of duplicating this code you could write a wrapper method that performs both functions.

```
/**
 * Clicks and Waits for page to load.
 *
 * param elementLocator
 * param waitPeriod
 */
public void clickAndWait(String elementLocator, String waitPeriod) {
        selenium.click(elementLocator);
        selenium.waitForPageToLoad(waitPeriod);
}
```

'Safe Operations' for Element Presence

Another common usage of wrapping Selenium methods is to check for presence of an element on page before carrying out some operation. This is sometimes called a 'safe operation'. For instance, the following method could be used to implement a safe operation that depends on an expected element being present.

```
/**
 * Selenum-RC -- Clicks on element only if it is available on page.
 *
 * param elementLocator
 */
public void safeClick(String elementLocator) {
        if(selenium.isElementPresent(elementLocator)) {
                selenium.click(elementLocator);
        } else {
                // Using the TestNG API for logging
                Reporter.log("Element: " +elementLocator+ ", is not
available on page - "
                                +selenium.getLocation());
        }
}
```

This example uses the Selenium 1 API but Selenium 2 also supports this.

```
/**
 * Selenium-WebDriver -- Clicks on element only if it is available on page.
 *
```

```
 * param elementLocator
 */
public void safeClick(String elementLocator) {
        WebElement webElement =
getDriver().findElement(By.XXXX(elementLocator));
        if(webElement != null) {
                selenium.click(elementLocator);
        } else {
                // Using the TestNG API for logging
                Reporter.log("Element: " +elementLocator+ ", is not
available on page - "
                                + getDriver().getUrl());
        }
}
```

In this second example 'XXXX' is simply a placeholder for one of the multiple location methods that can be called here.

Using safe methods is up to the test developer's discretion. Hence, if test execution is to be continued, even in the wake of missing elements on the page, then safe methods could be used, while posting a message to a log about the missing element. This, essentially, implements a 'verify' with a reporting mechanism as opposed to an abortive assert. But if element must be available on page in order to be able to carry out further operations (i.e. login button on home page of a portal) then this safe method technique should not be used.

Table 类

http://blog.sina.com.cn/s/blog_6966650401012pvv.html

```
package com.core;

import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class WebDriverWrapper {
    private WebDriver driver;

    public WebDriverWrapper(WebDriver driver) {
        this.driver = driver;
```

```
    }

    public String getCellText(By by, String tableCellAddress) {
        // 得到table元素对象
        WebElement table = driver.findElement(by);
        // 对所要查找的单元格位置字符串进行分解，得到其对应行、列。
        int index = tableCellAddress.trim().indexOf('.');
        int row = Integer.parseInt(tableCellAddress.substring(0,
index));
        int cell = Integer.parseInt(tableCellAddress.substring(index +
1));
        // 得到table表中所有行对象，并得到所要查询的行对象。
        List<WebElement> rows = table.findElements(By.tagName("tr"));
        WebElement theRow = rows.get(row);
        // 调用getCell方法得到对应的列对象，然后得到要查询的文本。
        String text = getCell(theRow, cell).getText();
        return text;
    }

    private WebElement getCell(WebElement Row, int cell) {
        List<WebElement> cells;
        WebElement target = null;
        // 列里面有"<th>"、"<td>"两种标签，所以分开处理。
        if (Row.findElements(By.tagName("th")).size() > 0) {
            cells = Row.findElements(By.tagName("th"));
            target = cells.get(cell);
        }
        if (Row.findElements(By.tagName("td")).size() > 0) {
            cells = Row.findElements(By.tagName("td"));
            target = cells.get(cell);
        }
        return target;
    }
}
```

WebDriver Action 类

http://blog.sina.com.cn/s/blog_6966650401012c1s.html

```
package com.core;

import java.util.List;
```

```
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class WebDriverWrapper {
    private WebDriver driver;

    public WebDriverWrapper(WebDriver driver) {
        this.driver = driver;
    }

    public boolean isWebElementExist(By selector) {
        try {
            driver.findElement(selector);
            return true;
        } catch(NoSuchElementException e) {
            return false;
        }
    }

    public String getWebText(By by) {
        try {
        return driver.findElement(by).getText();
        } catch (NoSuchElementException e) {
            return "Textnot existed!";
        }
    }

    public void clickElementContainingText(By by, String text){
        List<WebElement>elementList = driver.findElements(by);
        for(WebElement e:elementList){
            if(e.getText().contains(text)){
                e.click();
                break;
            }
        }
    }

    public String getLinkUrlContainingText(By by, String text){
        List<WebElement>subscribeButton = driver.findElements(by);
        String url = null;
```

```
        for(WebElement e:subscribeButton){
            if(e.getText().contains(text)){
                url =e.getAttribute("href");
                break;
            }
        }
        return url;
    }

    public void click(By by){
       driver.findElement(by).click();
       driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);
    }

    public String getLinkUrl(By by){
        return driver.findElement(by).getAttribute("href");
    }

    public void sendKeys(By by, String value){
       driver.findElement(by).sendKeys(value);
    }

}
```

对象库（UI Mapping）

A UI map is a mechanism that stores all the locators for a test suite in one place for easy modification when identifiers or paths to UI elements change in the AUT. The test script then uses the UI Map for locating the elements to be tested. Basically, a UI map is a repository of test script objects that correspond to UI elements of the application being tested.

What makes a UI map helpful? Its primary purpose is making test script management much easier. When a locator needs to be edited, there is a central location for easily finding that object, rather than having to search through test script code. Also, it allows changing the Identifier in a single place, rather than having to make the change in multiple places within a test script, or for that matter, in multiple test scripts.

To summarize, a UI map has two significant advantages.

- Using a centralized location for UI objects instead of having them scattered throughout the script. This makes script maintenance more efficient.
- Cryptic HTML Identifiers and names can be given more human-readable names improving the readability of test scripts.

Consider the following, difficult to understand, example (in java).

```
public void testNew() throws Exception {
            selenium.open("http://www.test.com");
            selenium.type("loginForm:tbUsername", "xxxxxxxx");
            selenium.click("loginForm:btnLogin");
            selenium.click("adminHomeForm:_activitynew");
            selenium.waitForPageToLoad("30000");
            selenium.click("addEditEventForm:_IDcancel");
            selenium.waitForPageToLoad("30000");
            selenium.click("adminHomeForm:_activityold");
            selenium.waitForPageToLoad("30000");
}
```

This script would be hard to follow for anyone not familiar with the AUT's page source. Even regular users of the application might have difficulty understanding what thus script does. A better script could be:

```
public void testNew() throws Exception {
            selenium.open("http://www.test.com");
            selenium.type(admin.username, "xxxxxxxx");
            selenium.click(admin.loginbutton);
            selenium.click(admin.events.createnewevent);
            selenium.waitForPageToLoad("30000");
            selenium.click(admin.events.cancel);
            selenium.waitForPageToLoad("30000");
            selenium.click(admin.events.viewoldevents);
            selenium.waitForPageToLoad("30000");
}
```

Now, using some comments and whitespace along with the UI Map identifiers makes a very readable script.

```
public void testNew() throws Exception {

            // Open app url.
            selenium.open("http://www.test.com");
```

```
            // Provide admin username.
            selenium.type(admin.username, "xxxxxxxx");

            // Click on Login button.
            selenium.click(admin.loginbutton);

            // Click on Create New Event button.
            selenium.click(admin.events.createnewevent);
            selenium.waitForPageToLoad("30000");

            // Click on Cancel button.
            selenium.click(admin.events.cancel);
            selenium.waitForPageToLoad("30000");

            // Click on View Old Events button.
            selenium.click(admin.events.viewoldevents);
            selenium.waitForPageToLoad("30000");
}
```

There are various ways a UI Map can be implemented. One could create a class or struct which only stores public String variables each storing a locator. Alternatively, a text file storing key value pairs could be used. In Java, a properties file containing key/value pairs is probably best method.

Consider a property file *prop.properties* which assigns as 'aliases' reader-friendly identifiers for UI elements from the previous example.

```
admin.username = loginForm:tbUsername
admin.loginbutton = loginForm:btnLogin
admin.events.createnewevent = adminHomeForm:_activitynew
admin.events.cancel = addEditEventForm:_IDcancel
admin.events.viewoldevents = adminHomeForm:_activityold
```

The locators will still refer to html objects, but we have introduced a layer of abstraction between the test script and the UI elements. Values are read from the properties file and used in the Test Class to implement the UI Map. For more on Java properties files refer to the following link.

面向对象设计模式（Page Object Design Pattern）

Page Object is a Design Pattern which has become popular in test automation for **enhancing test maintenance** and **reducing code**

duplication. A page object is an object-oriented class that serves as an interface to a page of your AUT. The tests then use the methods of this page object class whenever they need to interact with that page of the UI. The benefit is that if the UI changes for the page, the tests themselves don't need to change, only the code within the page object needs to change. Subsequently all changes to support that new UI are located in one place.

The Page Object Design Pattern provides the following advantages.

1. There is clean separation between test code and page specific code such as locators (or their use if you're using a UI map) and layout.

2. There is single repository for the services or operations offered by the page rather than having these services scattered through out the tests.

In both cases this allows any modifications required due to UI changes to all be made in one place. Useful information on this technique can be found on numerous blogs as this 'test design pattern' is becoming widely used. *We encourage the reader who wishes to know more to search the internet for blogs on this subject.* Many have written on this design pattern and can provide useful tips beyond the scope of this user guide. To get you started, though, we'll illustrate page objects with a simple example.

First, consider an example, typical of test automation, that does not use a page object.

```
/***
 * Tests login feature
 */
public class Login {

        public void testLogin() {
                selenium.type("inputBox", "testUser");
                selenium.type("password", "my supersecret password");
                selenium.click("sign-in");
                selenium.waitForPageToLoad("PageWaitPeriod");
                Assert.assertTrue(selenium.isElementPresent("compose
button"),
                                        "Login was unsuccessful");
        }
```

```
}
```

There are two problems with this approach.

1. There is no separation between the test method and the AUTs locators (IDs in this example); both are intertwined in a single method. If the AUT's UI changes its identifiers, layout, or how a login is input and processed, the test itself must change.
2. The id-locators would be spread in multiple tests, all tests that had to use this login page.

Applying the page object techniques this example could be rewritten like this in the following example of a page object for a Sign-in page.

```
/**
 * Page Object encapsulates the Sign-in page.
 */
public class SignInPage {

        private Selenium selenium;

        public SignInPage(Selenium selenium) {
                this.selenium = selenium;
                if(!selenium.getTitle().equals("Sign in page")) {
                        throw new IllegalStateException("This is not
sign in page, current page is: "
                                        +selenium.getLocation());
                }
        }

        /**
         * Login as valid user
         *
         * @param userName
         * @param password
         * @return HomePage object
         */
        public HomePage loginValidUser(String userName, String password)
{
                selenium.type("usernamefield", userName);
                selenium.type("passwordfield", password);
                selenium.click("sign-in");
                selenium.waitForPageToLoad("waitPeriod");

                return new HomePage(selenium);
```

```
        }
}
```

and page object for a Home page could look like this.

```
/**
 * Page Object encapsulates the Home Page
 */
public class HomePage {

        private Selenium selenium;

        public HomePage(Selenium selenium) {
                if (!selenium.getTitle().equals("Home Page of logged in
user")) {
                        throw new IllegalStateException("This is not
Home Page of logged in user, current page" +
                                        "is: "
+selenium.getLocation());
                }
        }

        public HomePage manageProfile() {
                // Page encapsulation to manage profile functionality
                return new HomePage(selenium);
        }

        /*More methods offering the services represented by Home Page
        of Logged User. These methods in turn might return more Page
Objects
        for example click on Compose mail button could return ComposeMail
class object*/

}
```

So now, the login test would use these two page objects as follows.

```
/***
 * Tests login feature
 */
public class TestLogin {

        public void testLogin() {
                SignInPage signInPage = new SignInPage(selenium);
```

```
                HomePage homePage =
signInPage.loginValidUser("userName", "password");
                Assert.assertTrue(selenium.isElementPresent("compose
button"),
                                  "Login was unsuccessful");
        }
}
```

There is a lot of flexibility in how the page objects may be designed, but there are a few basic rules for getting the desired maintainability of your test code. **Page objects themselves should never be make verifications or assertions.** This is part of your test and should always be within the test's code, never in an page object. The page object will contain the representation of the page, and the services the page provides via methods but no code related to what is being tested should be within the page object.

There is one, single, verification which can, and should, be within the page object and that is to verify that the page, and possibly critical elements on the page, were loaded correctly. This verification should be done while instantiating the page object. In the examples above, both the SignInPage and HomePage constructors check that the expected page is available and ready for requests from the test.

A page object does not necessarily need to represent an entire page. The Page Object design pattern could be used to represent components on a page. If a page in the AUT has multiple components, it may improved maintainability if there was a separate page object for each component.

There are other design patterns that also may be used in testing. Some use a Page Factory for instantiating their page objects. Discussing all of these is beyond the scope of this user guide. Here, we merely want to introduce the concepts to make the reader aware of some of the things that can be done. As was mentioned earlier, many have blogged on this topic and we encourage the reader to search for blogs on these topics.

参考：

http://www.cnblogs.com/nbkhic/archive/2011/10/22/2221508.html

数据驱动测试（Data Driven Testing）

Data Driven Testing refers to using the same test (or tests) multiple times with varying data. These data sets are often from external files i.e. .csv file, text file, or perhaps loaded from a database. Data driven testing is a commonly used test automation technique used to validate an application against many varying input. When the test is designed for varying data, the input data can expand, essentially creating additional tests, without requiring changes to the test code.

In Python:

```
# Collection of String values
source = open("input_file.txt", "r")
values = source.readlines()
source.close()
# Execute For loop for each String in the values array
for search in values:
    sel.open("/")
    sel.type("q", search)
    sel.click("btnG")
    sel.waitForPageToLoad("30000")
    self.failUnless(sel.is_text_present("Results * for " + search))
```

The Python script above opens a text file. This file contains a different search string on each line. The code then saves this in an array of strings, and iterates over the array doing a search and assert on each string.

This is a very basic example, but the idea is to show that running a test with varying data can be done easily with a programming or scripting language. For more examples, refer to theSelenium RC wiki for examples of reading data from a spreadsheet or for using the data provider capabilities of TestNG. Additionally, this is a well-known topic among test automation professionals including those who don't use Selenium so searching the internet on "data-driven testing" should reveal many blogs on this topic.

Excel-JXL

Jxl 下载：

http://www.andykhan.com/jexcelapi/download.html

参考：

http://my.oschina.net/u/130226/blog/54997

http://www.2cto.com/kf/201201/116851.html

```
package com.core;

import java.io.File;

import jxl.Cell;
import jxl.Sheet;
import jxl.Workbook;
import jxl.write.Label;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;

public class ExcelTest {

    /**
     * @param args
     */
    public static void main(String[] args) {

        CreateExcelFile();
        ReadExcelFile();
        UpdateExcelFile();
    }

    public static void CreateExcelFile() {
        try {
            // 打开文件
            WritableWorkbook book = Workbook
                    .createWorkbook(new File("test.xls"));
            // 生成名为"第一页"的工作表，参数0表示这是第一页
            WritableSheet sheet = book.createSheet("第一页", 0);
            // 在Label对象的构造子中指名单元格位置是第一列第一行(0,0)
            // 以及单元格内容为test
            Label label = new Label(0, 0, "test");

            // 将定义好的单元格添加到工作表中
            sheet.addCell(label);
```

```
            /*
             * 生成一个保存数字的单元格 必须使用Number的完整包路径,否则有语法歧义
单元格位置是第二列，第一行，值为789.123
             */
            jxl.write.Number number = new jxl.write.Number(1, 0,
555.12541);
            sheet.addCell(number);

            // 写入数据并关闭文件
            book.write();
            book.close();

        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public static void ReadExcelFile() {
        try {
            Workbook book = Workbook.getWorkbook(new File("test.xls"));
            // 获得第一个工作表对象
            Sheet sheet = book.getSheet(0);
            // 得到第一列第一行的单元格
            Cell cell1 = sheet.getCell(0, 0);
            String result = cell1.getContents();
            System.out.println(result);
            book.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public static void UpdateExcelFile() {
        try {
            // Excel获得文件
            Workbook wb = Workbook.getWorkbook(new File("test.xls"));
            // 打开一个文件的副本，并且指定数据写回到原文件
            WritableWorkbook book = Workbook.createWorkbook(
                    new File("test.xls"), wb);
            // 添加一个工作表
            WritableSheet sheet = book.createSheet("第二页", 1);
            sheet.addCell(new Label(0, 0, "第二页的测试数据"));
            book.write();
            book.close();
```

```
        } catch (Exception e) {
            System.out.println(e);
        }

    }

}
```

Excel-JDBC

```
package com.core;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCExcelTest {

    private final String DRIVERCLASSNAME =
"sun.jdbc.odbc.JdbcOdbcDriver";

    private Connection con = null;
    // 注:链接字符串必须加上readonly=false属性，否则无法修改.更多选项看参照
Access数据源配置的高级选项.
    // 更多链接字符串可在 http://www.connectionstrings.com 上查看.
    // Driver={Microsoft Excel Driver
    // (*.xls)};DriverId=790;Dbq=C:\MyExcel.xls;DefaultDir=c:\mypath;
    private String url = "jdbc:odbc:Driver={Microsoft Excel Driver
(*.xls)};DBQ=TestJDBCExcel..xls;READONLY=FALSE";

    public JDBCExcelTest() throws Exception {
        Class.forName(DRIVERCLASSNAME);
        con = DriverManager.getConnection(url);

    }

    public int write() throws Exception {
        Statement statement = con.createStatement();
```

```
        // 创建一个工作簿(表),使用完后一定要关闭,否则会出错!
        int x = statement.executeUpdate("create table 测试(编号 NUMBER,
用户名 TEXT)");
        statement.close();
        return x;
    }

    public void read() throws SQLException {
        Statement statement = con.createStatement();
        /**
         * 查询时表明应该用 [tablename$] 或[worksheetname$] 这是微软ODBC的保
留字，否测会出现找不到引擎.
         * Excel会把首行的值当成字段值.即列名. SQL syntax "SELECT * FROM
[sheet1$]". I.e.
         * excel worksheet name followed by a "$" and wrapped in "[" "]"
         * brackets.
         * */
        ResultSet result = statement.executeQuery("SELECT * FROM [测试
$]");
        while (result.next()) {
            System.out.println(result.getInt(1) + "\t" +
result.getString(2));
        }
        result.close();
    }

    public int update() throws SQLException {
        Statement statement = con.createStatement();
        int x = statement.executeUpdate("insert into 测试 values(1,'孟德
军')");
        /**
         * int x = statement.executeUpdate("insert into 测试 values(1,'
孟德军')");
         * String sql="alter table 测试 add 密码 TEXT";
         * */
        statement.close();
        return x;
    }

    public void close() throws SQLException {
        // 一定要关闭连接对象,否则文档会出错.

        if (con != null) {
            con.close();
```

```
        }
    }

    public static void main(String[] args) throws Exception {
        JDBCExcelTest excel = new JDBCExcelTest();
        // 创建工作簿(表)
        //excel.write();
        // 写入一条数据
        //excel.update();
        // 读取写入的数据
        excel.read();
        // 关闭对象
        excel.close();
    }

}
```

TestNG

```
package testngtest;

import java.io.File;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import jxl.*;


public class ExcelDataProvider implements Iterator<Object[]>{

    private Workbook book = null;
    private Sheet sheet = null;
    private int rowNum = 0;
    private int curRowNo = 0;
    private int columnNum = 0;
    private String[] columnnName;


    public ExcelDataProvider(String classname,String methodname){
        try {
```

```
            File directory = new File(".");
            this.book= Workbook.getWorkbook(new
File(directory.getCanonicalPath() + "\\resources\\" +
                    classname.replaceAll("\\.", "/")+".xls"));
            System.out.println(directory.getCanonicalPath() +
"\\resources\\" +
                    classname.replaceAll("\\.", "/")+".xls");
            this.sheet=book.getSheet(methodname);
            this.rowNum =sheet.getRows();

            Cell [] c = sheet.getRow(0);
            this.columnNum = c.length;
            columnnName = new String[c.length];
            for(int i=0;i< c.length;i++){
                //System.out.println( c[i].getContents().toString());
                columnnName[i] = c[i].getContents().toString();
            }
            this.curRowNo ++;

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public boolean hasNext() {
        if (this.rowNum==0 || this.curRowNo>=this.rowNum){
            try {
                book.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            return false;
        }
        else
            return true;
    }

    @Override
    public Object[] next() {
        Cell [] c = sheet.getRow(this.curRowNo);
        Map<String,String> s = new HashMap<String, String>();
        for(int i=0;i< this.columnNum;i++)
        {
```

```
            String temp="";
            try{
                temp=c[i].getContents().toString();
            }
            catch(ArrayIndexOutOfBoundsException ex){
                temp = "";
            }
            s.put(this.columnnName[i], temp);
        }

        Object r[]=new Object[1];
        r[0]=s;
        this.curRowNo++;
        return r;
    }

    @Override
    public void remove() {
         throw new UnsupportedOperationException("remove unsupported.");
    }
}
```

```
package testngtest;


import java.io.IOException;

import java.lang.reflect.Method;

import java.util.Iterator;

import java.util.Map;


import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

import org.testng.annotations.DataProvider;
```

```
public class NewTest {


    // 参数化
    @Parameters({"name","size"})
    @Test
    public void testParameter(String name,int size) throws Exception
{


        System.out.println("Parameter name = " + name);
        System.out.println("Parameter size = " + size);


    }


    // 数据驱动测试 1
    @Test(dataProvider = "dp")
    public void f(Integer n, String s) {
        System.out.println(s);
    }


    @DataProvider
    public Object[][] dp() {
        return new Object[][] { new Object[] { 1, "a" },
                new Object[] { 2, "b" }, };
    }
```

```
    // 数据驱动测试 2
    @Test(dataProvider = "dp2" )
    public void testA(Map<String,String> data) {
            System.out.print("字段 A:" +data.get("字段 A"));
            System.out.println("");
            System.out.print("字段 B:" +data.get("字段 B"));
            System.out.println("");
            System.out.print("字段 C:" +data.get("字段 C"));
            System.out.println("");
            System.out.println("********************");

    }

    @DataProvider(name = "dp2")
    public Iterator<Object[]> dataFortestMethod(Method method)
throws IOException{
        return new
ExcelDataProvider(this.getClass().getName(),method.getName());
    }

}
```

数据库检查点（Database Validation）

Another common type of testing is to compare data in the UI against the data actually stored in the AUT's database. Since you can also do database queries from a programming language, assuming you have database support functions, you can use them to retrieve data and then use the data to verify what's displayed by the AUT is correct.

Consider the example of a registered email address to be retrieved from a database and then later compared against the UI. An example of establishing a DB connection and retrieving data from the DB could look like this.

In Java:

```
// Load Microsoft SQL Server JDBC driver.
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

// Prepare connection url.
String url =
"jdbc:sqlserver://192.168.1.180:1433;DatabaseName=TEST_DB";

// Get connection to DB.
public static Connection con =
DriverManager.getConnection(url, "username", "password");

// Create statement object which would be used in writing DDL and DML
// SQL statement.
public static Statement stmt = con.createStatement();

// Send SQL SELECT statements to the database via the
Statement.executeQuery
// method which returns the requested information as rows of data in a
// ResultSet object.

ResultSet result =  stmt.executeQuery
("select top 1 email_address from user_register_table");

// Fetch value of "email_address" from "result" object.
String emailaddress = result.getString("email_address");

// Use the emailAddress value to login to application.
selenium.type("userID", emailaddress);
selenium.type("password", secretPassword);
```

```
selenium.click("loginButton");
selenium.waitForPageToLoad(timeOut);
Assert.assertTrue(selenium.isTextPresent("Welcome back" +emailaddress),
"Unable to log in for user" +emailaddress)
```

This is a simple Java example of data retrieval from a database.

Selenium 框架

Testing Frameworks

Testing frameworks aren't required, but they can be helpful if you want to automate your tests.

Framework	Selenium IDE	Selenium Remote Control	Selenium Core
Bromine	Comes with template to add to IDE	Manipulate browser, check assertions via custom driver	Special support**
JUnit	Out-of-the-box code generation	Manipulate browser, check assertions via Java driver	n/a
NUnit	Out-of-the-box code generation	Manipulate browser, check assertions via .NET driver	n/a
RSpec (Ruby)	Custom code generation template*	Manipulate browser, check assertions via Ruby driver	n/a
Test::Unit (Ruby)	Out-of-the-box code generation	Manipulate browser, check assertions via Ruby driver	n/a
TestNG (Java)	Custom code generation template*	Manipulate browser, check assertions via Java driver	n/a
unittest (Python)	Out-of-the-box code generation	Manipulate browser, check assertions via Python driver	n/a
Others	Custom code generation template*	Manipulate browser, check assertions via HTTP requests***	n/a
Robot Framework SeleniumLibrary	n/a	Utilizes the Python driver	n/a

ISFW	Out-of-the-box code generation****	Manipulate browser, check assertions via Java driver, also works with webdriver/remote webdriver, Grid2, Sauce on demand	n/a
Hermes	Comes with template to add to IDE	Manipulate browser, check assertions via Java driver. Full integration with Selenium GRID and Saucelabs infrastructure	n/a

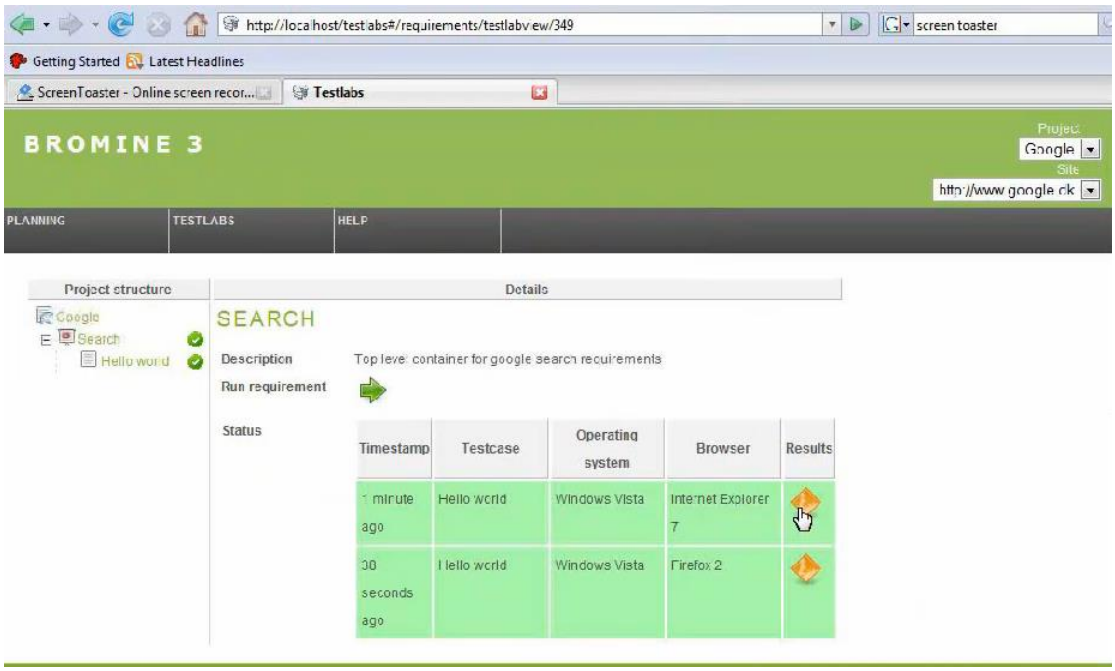
* Built-in code generation creates code specifically for the "out-of-the-box" frameworks, but you can modify the Javascript-based templates to output any kind of code you want.

** Bromine is built specifically for Selenium and it directly supports both Selenium Core and Selenium Remote Control. The other testing frameworks use Selenium Remote Control drivers for testing.

*** Using the appropriate driver (or HTTP commands), you can manipulate and test the browser using any testing framework by calling commands like "click" or "isElementPresent." Selenium Remote Control just deals with the commands and doesn't care about what actually calls those commands.

**** Selenium IDE Plug-in for InfoStretch Framework includes a formatter for ISFW as well as a bunch of other extensions.

Bromine



http://www.brominefoundation.org/

Bromine is an open source QA tool that uses selenium RC as its testing engine.

It provides project management, OS/browser specification, test-case creation as well as user management.

- Supports tests in Java and PHP
- Record tests with provided IDE formats
- Upload tests
- Run multiple tests against multiple OS/browser combinations with a single click
- Setup OS/browser testing needs with checkboxes
- Uses completely unmodified RC servers
- No RC server hassle, just plug in the IP and you're set
- Seemless loadbalancing. If you have multiple RC servers defined, they will be used (not GRID, our own solution)
- Full log of results. All commands executed and their status are stored.
- Define your own user groups and their rights
- Coded in CakePHP, MVC design, accessible API, easily extendable.
- Add plugins easily

下载：

http://www.brominefoundation.org/pages/download

简介：
http://wiki.openqa.org/display/BR/Home
下载：
http://bromine.seleniumhq.org/download.jsp

安装：
1、Setting up a webserver ready for Bromine
http://wiki.openqa.org/display/BR/Setting+up+a+webserver+ready+for+Bromine
2、Installing Bromine
http://wiki.openqa.org/display/BR/Installing+Bromine

3、Setting up Bromine for running tests
http://wiki.openqa.org/display/BR/Setting+up+Bromine+for+running+tests

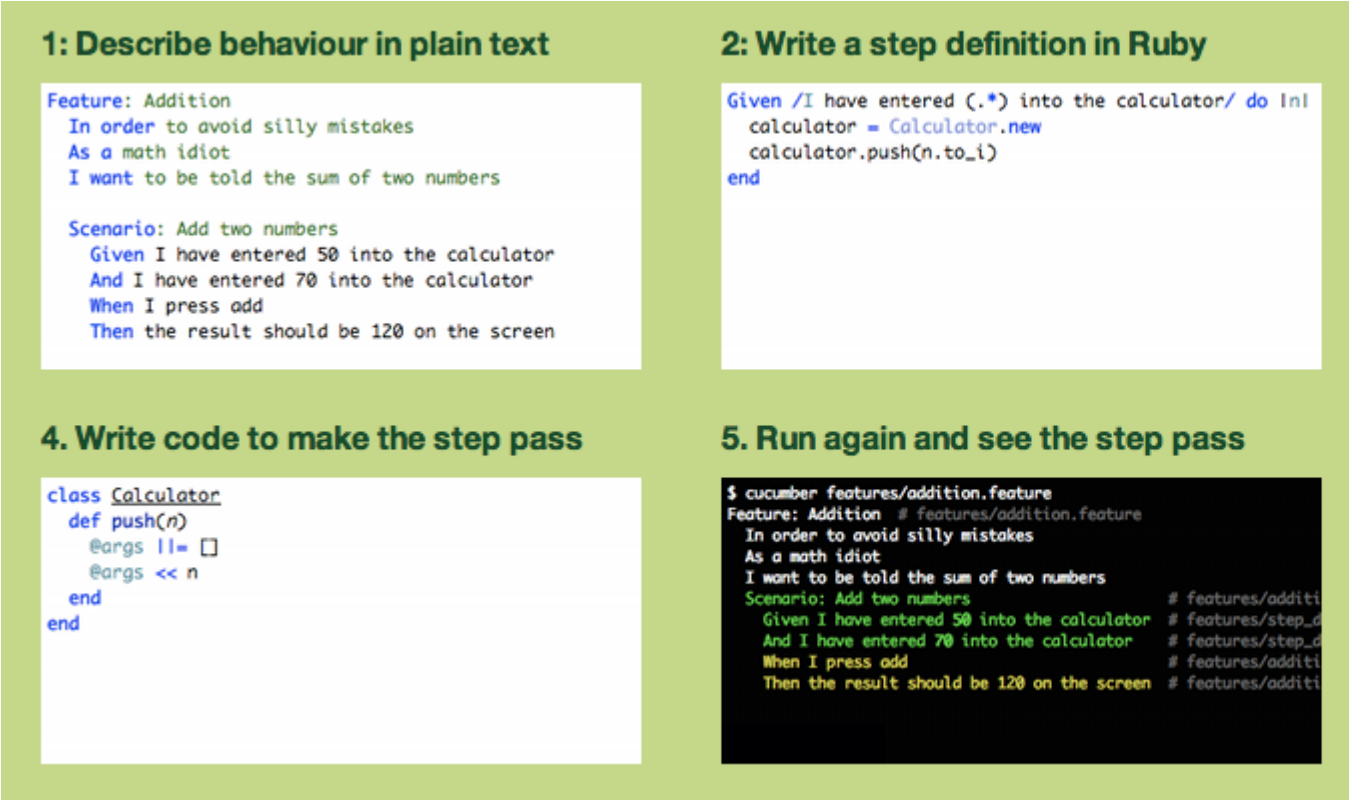
4、Creating a testscript
http://wiki.openqa.org/display/BR/Creating+a+testscript

5、Compiling a java script for Bromine in netbeans
http://wiki.openqa.org/display/BR/Compiling+a+java+script+for+Bromine+in+netbeans

视频教程：
http://bromine.seleniumhq.org/brscreencast/brscreencast.html

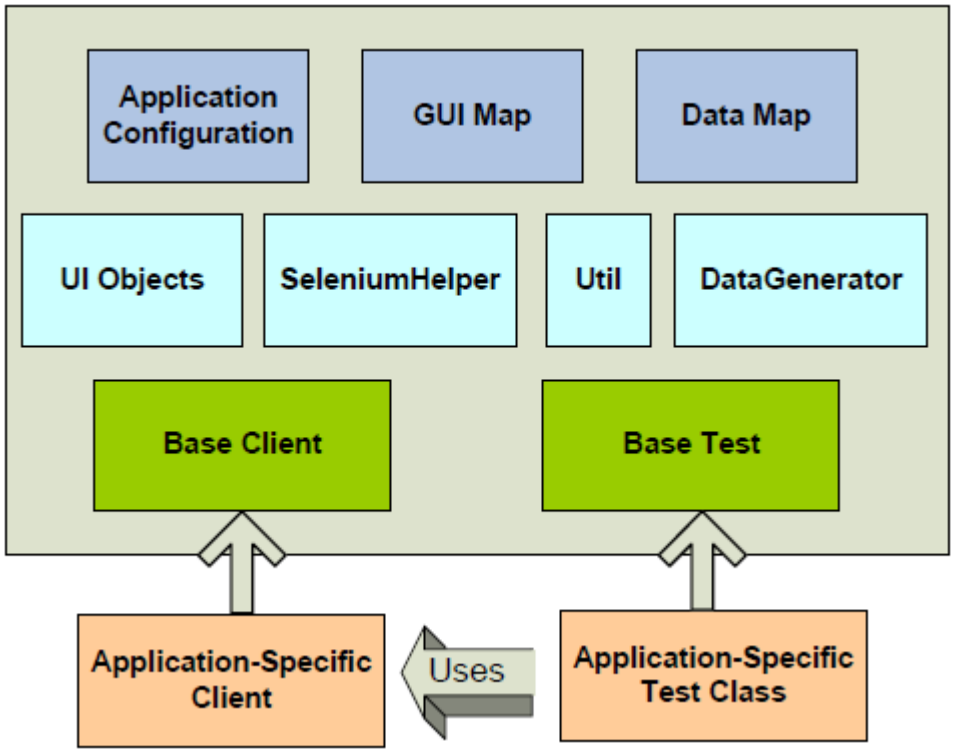
RSpec、Cucumber

BDD(行为驱动开发)测试模式



GTAC（SeleniumRC+TestNG+Firebug）

Ringo

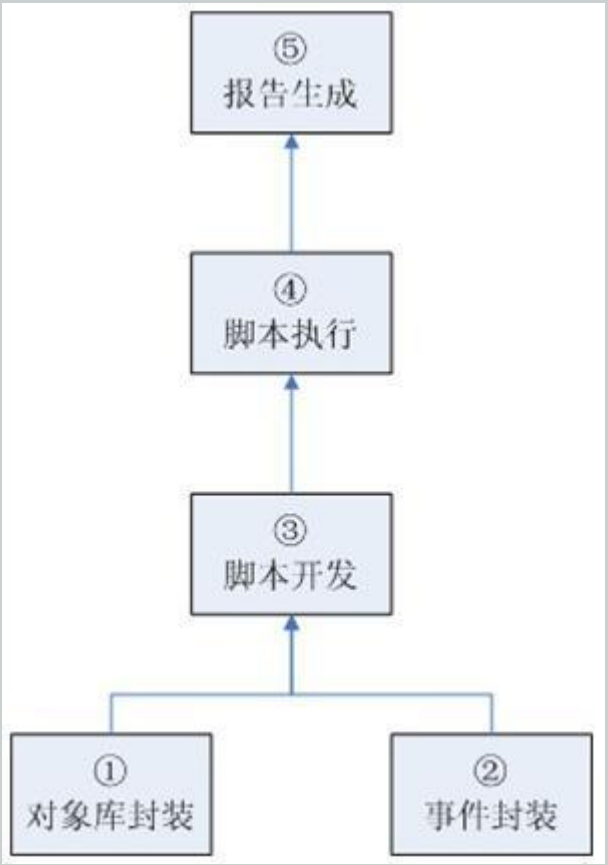


参考：
Building a flexible and extensible framework around Selenium.pdf

WebDriver & TestNG 架构

参考：
http://blog.sina.com.cn/s/blog_6966650401012pnl.html
http://testng.org/doc/selenium.html

一、架构图



1.对象库封装：系统会经历升级改版，每次改动可能会将原来用来定位的 Key（Id,name,ccc...）改变，因而需要修改脚本，而脚本可能无数次的引用该改动的控件对象，查找替换是件非常繁琐的事情更有可能会有遗漏，所以如果将对象库放在数据库中，每次修改只需要修改数据库中的一条数据，这样后期的脚本维护会变得相当方便。考虑到执行效率，可以一次性把数据库中的控件对象库全部读到本地临时文件中，可以避免多次链接数据库操作。

2.事件封装：将控件操作、数据比对操作封装成函数。方便后续事件统一维护。

3.脚本开发：用 java 语言开发 selenium 脚本

4.脚本执行：客户端通过 testNG 起测试脚本，执行用例

5.报告生成：测试完毕，将测试结果转换成 html 文件，如果条件允许可以发邮件通知。

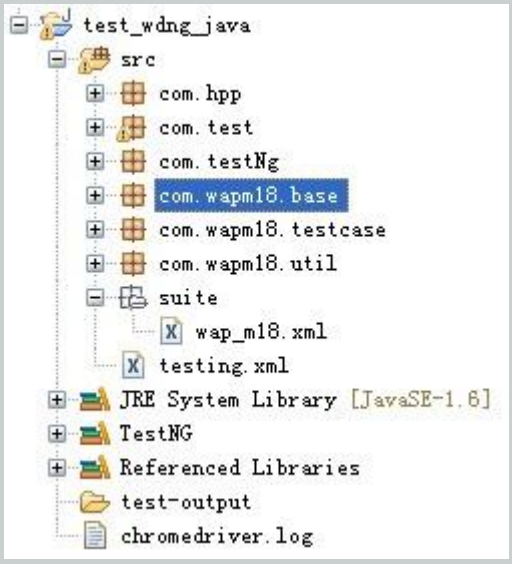
二、代码架构

com.wapm18.base　包：用来存放封装底层方法

com.wapm18.util 包：用来存放公用方法

com.wapm18.testcase 包：用来存放测试用例

suit 文件夹：用来存放执行用例集



三、对象库构建

数据库脚本

```
CREATE TABLE [dbo].[testng_object](
[id] [int] IDENTITY(1,1) NOT NULL,
[website] [varchar](50) NULL,
[page] [varchar](100) NULL,
[area] [varchar](100) NULL,
[object_type] [varchar](50) NULL,
[object_key_type] [varchar](50) NULL,
[object_pagename] [varchar](50) NULL,
[object_key] [varchar](50) NULL,
[description] [varchar](50) NULL,
[Status] [bit] NULL,
[CreateTime] [datetime] NULL,
[CreateUser] [varchar](20) NULL,
[UpdateTime] [datetime] NULL,
[UpdateUser] [varchar](20) NULL,
[DeleteTime] [datetime] NULL,
[DeleteUser] [varchar](20) NULL,
  CONSTRAINT [PK_testng_object] PRIMARY KEY CLUSTERED
(
[Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

四、创建测试用例

testNg 类

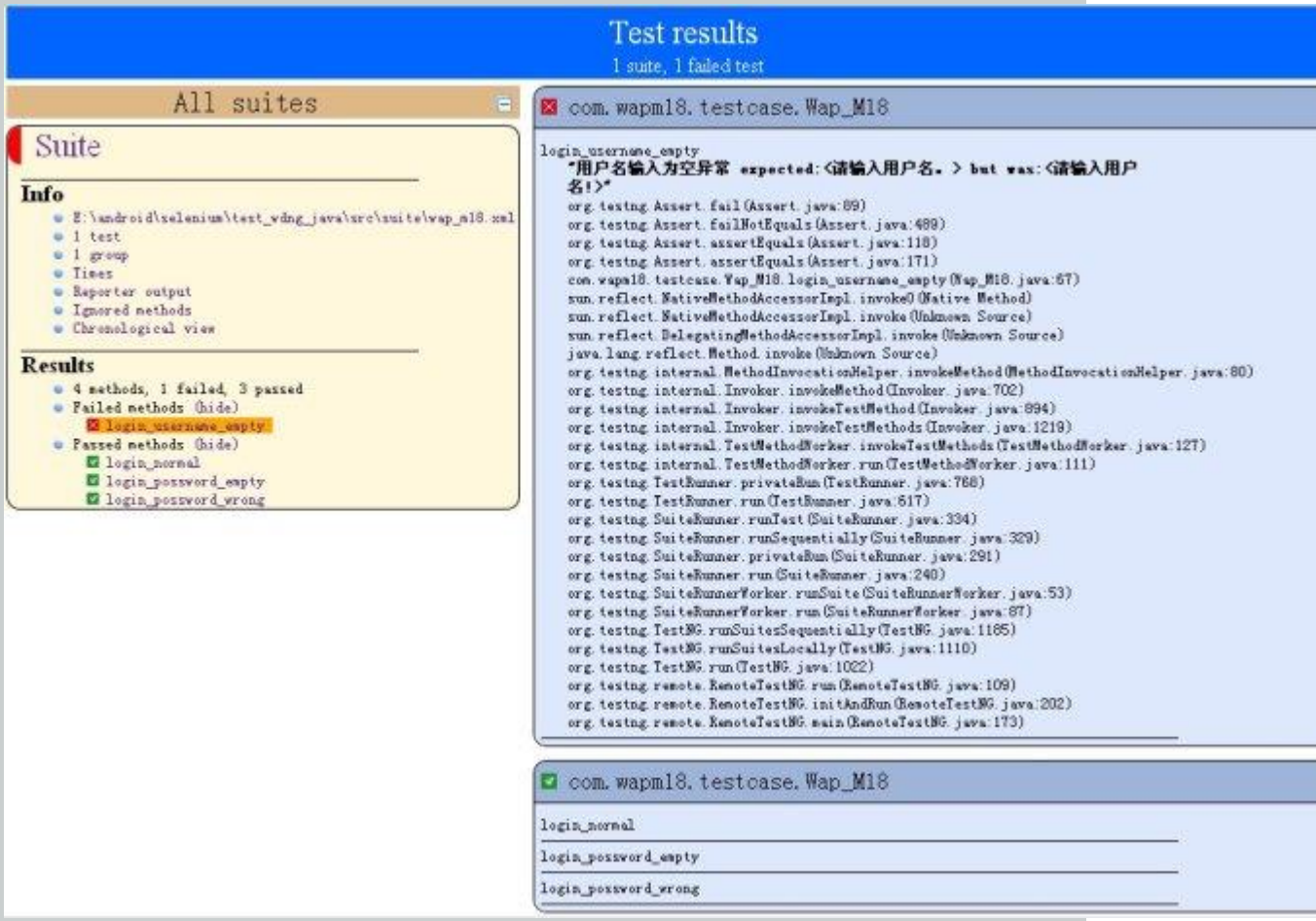
--方法前初始化 驱动浏览器和地址

```
@BeforeMethod(groups = {"login"})
public void initMethodforLogin(){
driver = new ChromeDriver();
driver.get("http://your.url.com/page.aspx");
}

--测试用例 参数化控件对象
@Parameters({"login_name_txtUserName","login_name_txtPwd","login_name_btnLogin"})
@Test(groups = {"login"})
public void login_username_empty(String login_name_txtUserName,String
login_name_txtPwd,String login_name_btnLogin) throws Exception {
System.out.println("login_username_empty");
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.findElement(By.name(login_name_txtUserName)).clear();
driver.findElement(By.name(login_name_txtPwd)).clear();
driver.findElement(By.name(login_name_txtPwd)).sendKeys("123456");
driver.findElement(By.name(login_name_btnLogin)).click();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
assertEquals("请输入用户名。",driver.findElement(By.className("error-tip")).getText(),"
用户名输入为空异常");
driver.quit();
}
-----------------------------------------------------------------
xml 文件
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Suite" parallel="false">
    <parameter name="login_name_txtUserName" value="txtUserName" />
    <parameter name="login_name_txtPwd" value="txtPwd" />
    <parameter name="login_name_btnLogin" value="btnLogin" />
    <test name="wapm18" ><!-- preserve-order="true" -->
    <groups>
        <run>
            <include name="login"/>
        </run>
        </groups>
        <classes>
            <class name="com.wapm18.testcase.Wap_M18_parameter_by_xml"/>
        </classes>
```

```
    </test> <!-- Test -->
</suite> <!-- Suite -->
```

五、执行测试用例

六、生成报告



插入数据脚本

```
INSERT INTO
testng_object([website],[page],[area],[object_type],[object_key_type],[object_pagename]
,[object_key],[description],[Status],[CreateTime],[CreateUser],[UpdateTime],[UpdateUser
```

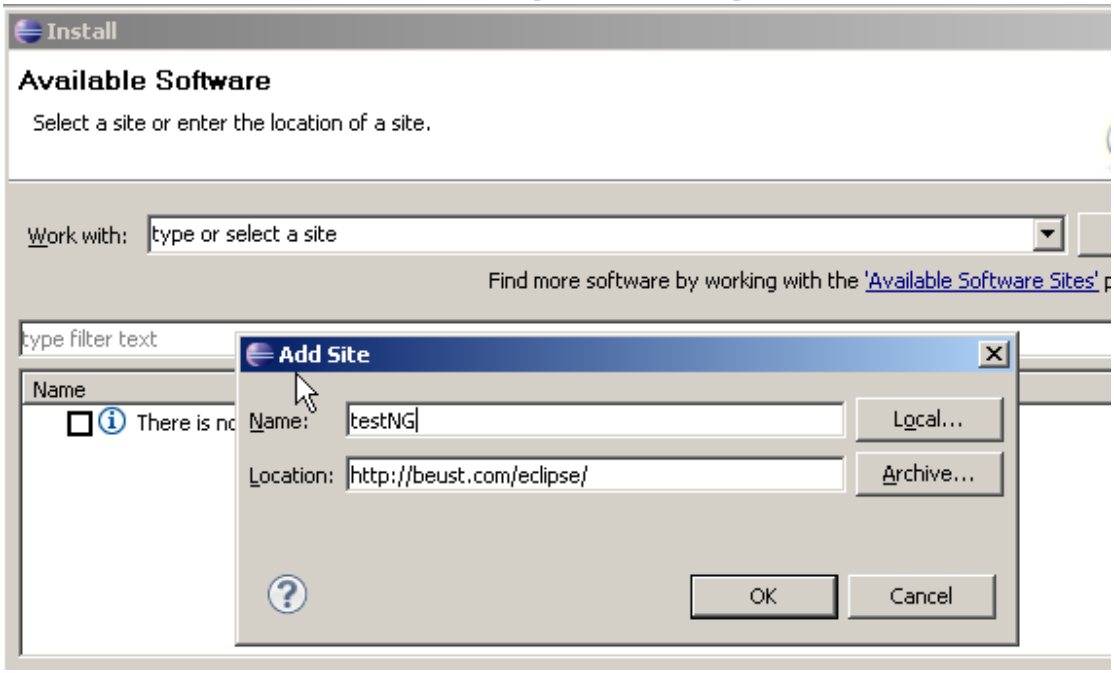
```
],[DeleteTime],[DeleteUser])VALUES('m.m18.com','login.aspx','login','text','name','txtU
serName','txtUserName','用户名输入框','1',GETDATE(),'tools',GETDATE(),'tools','','')
```

安装 TestNG

安装 testNG 插件到 eclipse.

-) 选择菜单 Help / Software updates / Find and Install.

-) 点击 add button 然后在 location 中输入 http://beust.com/eclipse/



-) 确定后会自动安装 testNG 插件。

TestNG 参数化

参考：

http://blog.sina.com.cn/s/blog_6966650401012ra0.html

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd">
<suite name="MyTestSuite-01">

 <test name="MyTest-01" preserve-order="true">
```

```
    <parameter name="name" value="abc"/>
    <parameter name="size" value="123"/>

        <classes>
            <class name="testngtest.NewTest">

                <methods>

                    <include name="testParameter" />
                    <include name="f" />
                    <include name="testA" />

                </methods>

            </class>
        </classes>

    </test>

</suite>
```

```
package testngtest;

import java.io.IOException;
import java.lang.reflect.Method;
import java.util.Iterator;
import java.util.Map;

import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
import org.testng.annotations.DataProvider;

public class NewTest {

    // 参数化
    @Parameters({"name","size"})
    @Test
    public void testParameter(String name,int size) throws Exception {

        System.out.println("Parameter name = " + name);
        System.out.println("Parameter size = " + size);

    }
```

```
    // 数据驱动测试 1
    @Test(dataProvider = "dp")
    public void f(Integer n, String s) {
        System.out.println(s);
    }

    @DataProvider
    public Object[][] dp() {
        return new Object[][] { new Object[] { 1, "a" },
                new Object[] { 2, "b" }, };
    }

    // 数据驱动测试 2
    @Test(dataProvider = "dp2" )
    public void testA(Map<String,String> data) {
            System.out.print("字段 A:" +data.get("字段 A"));
            System.out.println("");
            System.out.print("字段 B:" +data.get("字段 B"));
            System.out.println("");
            System.out.print("字段 C:" +data.get("字段 C"));
            System.out.println("");
            System.out.println("**********");

    }

    @DataProvider(name = "dp2")
    public Iterator<Object[]> dataFortestMethod(Method method) throws IOException{
        return new ExcelDataProvider(this.getClass().getName(),method.getName());
    }

}
```

TestNG 数据驱动

从 Excel 读取数据

参考：
http://www.zhenghongzhi.cn/post/42.html
http://blog.sina.com.cn/s/blog_6966650401012lyn.html#parametric

一、设置参数

测试方法是可以带有参数的。每个测试方法都可以带有任意数量的参数，并且可以通过使用 TestNG 的@Parameters 向方法传递正确的参数。
设置方式有两种方法：使用 testng.xml 或者 Data Providers 。

（一）使用 testng.xml 设置参数

1.如果只使用相对简单的参数，可以在 testng.xml 文件中指定：

```
@Parameters({ "first-name" })
@Test
public void testSingleString(String firstName) {
    System.out.println("Invoked testString " + firstName);
    assert "Cedric".equals(firstName);
}
```

在这段代码中，我们让 firstName 参数能够接到 XML 文件中叫做 first-name 参数的值。这个 XML 参数被定义在 testng.xml：

```
<suite name="My suite">
    <parameter name="first-name"   value="Cedric"/>
    <test name="Simple example">
```

类似的，它也可以用在 @Before/After 和 @Factory 注解上：

```
@Parameters({ "datasource", "jdbcDriver" })
@BeforeMethod
public void beforeTest(String ds, String driver) {
    m_dataSource = ...;                                                    // 查
询数据源的值
    m_jdbcDriver = driver;
}
```

这次有两个 Java 参数 ds 和 driver 会分别接收到来自属性 datasource 和 jdbc-driver 所指定的值。

2.参数也可以通过 Optional 注释来声明：

```
@Parameters("db")
@Test
public void testNonExistentParameter(@Optional("mysql")  String db) { ... }
```

如果在你的 testng.xml 文件中没有找到"db"，你的测试方法就会使用 @Optional 中的值："mysql"。

3.@Parameters 可以被放置到如下位置：
1.在任何已经被 @Test, @Before/After 或 @Factory 注解过的地方。
2.在测试类中至多被放到一个构造函数签。这样，TestNG 才能在需要的时候使用 testng.xml 中特定的参数来实例化这个类。这个特性可以被用作初始化某些类中的值，以便稍后会被类中其他的方法所使用。

注意：
XML 中的参数会按照 Java 参数在注解中出现的顺序被映射过去，并且如果数量不匹配，TestNG 会报错。

参数是有作用范围的。在 testng.xml 中，你即可以在<suite> 标签下声明，也可以在 <test>下声明。如果两个参数都有相同的名字，那么，定义在 <test> 中的有优先权。这在你需要覆盖某些测试中特定参数的值时，会非常方便。

（二）使用 DataProviders 提供参数

在 testng.xml 中指定参数可能会有如下的不足：

1.如果你压根不用 testng.xml.

2.你需要传递复杂的参数，或者从 Java 中创建参数（复杂对象，对象从属性文件或者数据库中读取的 etc...）

这样的话，你就可以使用 Data Provider 来给需要的测试提供参数。所谓**数据提供者，就是一个能返回对象数组的数组的方法，并且这个方法被@DataProvider 注解标注：**

DataProvider 的定义如下：

```
@DataProvider(name = "range-provider")
public Object[][] rangeData() {
int lower = 5;
int upper = 10;
return new Object[][] {
{ lower-1, lower, upper, false },
{ lower, lower, upper, true },
{ lower+1, lower, upper, true },
{ upper, lower, upper, true},
{ upper+1, lower, upper, false },
};
}
```

调用 DataProvider 的方式如下：

```
@Test(dataProvider = "range-provider")
public void testIsBetween(int n, int lower,int upper, boolean expected)
{
println("Received " + n + " " + lower + "-"+ upper + " expected: " + expected);
Assert.assertEquals(expected, isBetween(n, lower, upper));
}
```

被@Test 标注的方法通过 dataProvider 属性指明其数据提供商。这个名字必须与 @DataProvider(name="...") 中的名字相一致。

DataProvider 返回的是一个 Object 的二维数组，二维数组中的每个一维数组都会传递给调用函数，作为参数使用。运行的时候，会发现， @Test 标识的 test method 被执行的次数和 object[][]包含的一维数组的个数是一致的，而@Test 标识的函数的参数个数，也和 object 内一维数组内的元素数是一致的。

运行后的输出结果如下：

```
Received 4 5-10 expected: false
```

```
Received 5 5-10 expected: true
Received 6 5-10 expected: true
Received 10 5-10 expected: true
Received 11 5-10 expected: false

===============================================
Parameter Suite
Total tests run: 5, Failures: 0, Skips: 0
===============================================
```

（三）DataProviders 扩展

默认的情况下，数据提供者会查找当前的测试类或者测试类的基类。如果你希望它能够被其他的类所使用，那么就要将其指定为 static，并且通过 dataProviderClass 属性指定要使用的类：

```
public  static  class  StaticProvider  {
    @DataProvider(name = "create")
    public static Object[][] createData() {
        return new Object[][] {
            new Object[] { new Integer(42) }
        }
    }
}

public class MyTest {
    @Test(dataProvider = "create",  dataProviderClass =  StaticProvider.class)
    public void test(Integer n) {
        // ...
    }
}
```

Data Provider 方法可以返回如下两种类型中的一种：

1.含有多个对象的数组 (Object[][])，其中第一个下标指明了测试方法要调用的次数，第二个下标则完全与测试方法中的参数类型和个数相匹配。上面的例子已经说明。

2.另外一个是迭代器 Iterator<Object[]>。二者的区别是迭代器允许你延迟创建自己的测试数据。TestNG 会调用迭代器，之后测试方法会一个接一个的调用由迭代器返回的值。在你需要传递很多参数组给测试组的时候，这样你无须提前创建一堆值。

下面是使用 JDK5 的例子 (注意 JDK 1.4 的例子不适用泛型)：

```
public Iterator createData() {
    return new MyIterator(DATA);
)
@DataProvider(name = "test1")
public Iterator
```

如果你声明的 @DataProvider 使用 java.lang.reflect.Method 作为第一个参数，TestNG 会把当前的测试方法当成参数传给第一个参数。这一点在你的多个测试方法使用相同的@DataProvider 的时候，并且你想要依据具体的测试方法返回不同的值时，特别有用。
例如，下面的代码它内部的 @DataProvider 中的测试方法的名字：

```
@DataProvider(name = "dp")
public Object[][] createData(Method m) {
    System.out.println(m.getName());
    return new Object[][] { new Object[] { "Cedric" }};
}

@Test(dataProvider = "dp")
public void test1(String s) {
}
@Test(dataProvider = "dp")
public void test2(String s) {
}
```

所以会显示：

```
test1
test2
```

Data provider 可以通过属性 parallel 实现并行运行：

```
@DataProvider(parallel = true)
// ...
```

使用 XML 文件运行的 data provider 享有相同的线程池，默认的大小是 10.你可以通过修改该在 <suite> 标签中的值来更改：

```
<suite name="Suite1" data-provider-thread-count="20" >
...
```

如果你需要让指定的几个 data provider 运行在不同的线程中，那么就必须通过不同的 xml 文件来运行。

TestNG 测试用例组织

调度执行的顺序

```
preserve-order="true"
```

依赖方法

```
    @Test
    public void test1(){
    Assert.assertTrue(false);
    }

    // 依赖
```

```
@Test(dependsOnMethods = { "test1" })
public void test2(){

System.out.println("test2 Run!");

}
```

参考：

http://blog.sina.com.cn/s/blog_6966650401012lyn.html#parametric

有些时候，需要按照特定顺序调用测试方法。

1.确保在进行更多的方法测试之前，有一定数量的测试方法已经成功完成。

2.在初始化测试的时候，同时希望这个初始化方法也是一个测试方法（ @Before/After 不会出现在最后生成的报告中）。

为此，你可以使用 @Test 中的 dependsOnMethods 或 dependsOnGroups 属性。

这两种依赖：

1.**Hard dependencies（硬依赖）**。所有的被依赖方法必须成功运行。只要有一个出问题，测试就不会被调用，并且在报告中被标记为 SKIP。

2.**Soft dependencies（软依赖）**。 即便是有些依赖方法失败了，也一样运行。如果你只是需要保证你的测试方法按照顺序执行，而不关心他们的依赖方法是否成功。那么这种机制就非常有用。可以通过添加 "alwaysRun=true" 到 @Test 来实现软依赖。

硬依赖的例子：

```
@Test
public void serverStartedOk() {}
@Test(dependsOnMethods  = { "serverStartedOk" })
public void method1() {}
```

此例中，method1() 依赖于方法 serverStartedOk()，从而保证 serverStartedOk() 总是先运行。

也可以让若干方法依赖于组：

```
@Test(groups = { "init" })
public void serverStartedOk() {}
@Test(groups = { "init" })
public void initEnvironment() {}

@Test(dependsOnGroups = { "init.* })
public void method1() {}
```

本例中,method1()依赖于匹配正则表达式"init.*"的组，由此保证了 serverStartedOk()和 initEnvironment() 总是先于 method1()被调用。

注意：正如前面所说的那样，在相同组中的调用可是在夸测试中不保证顺序的。

如果你使用硬依赖，并且被依赖方法失败(alwaysRun=false，即默认是硬依赖)，依赖方法则不是被标记为 FAIL 而是 SKIP。被跳过的方法会被在最后的报告中标记出来（HTML 既不用红色也不是绿色所表示），主要是被跳过的方法不是必然失败，所以被标出来做以区别。

无论 dependsOnGroups 还是 dependsOnMethods 都可以接受正则表达式作为参数。对于 dependsOnMethods，如果被依赖的方法有多个重载，那么所有的重载方法都会被调用。如果你只希望使用这些重载中的一个，那么就应该使用 dependsOnGroups。

TestNG 并发执行用例

parallel="tests" thread-count="5"

可以通过使用 parallel 属性要求 TestNG 在单独的线程中运行测试。这个属性可以在两个值中取其一：

<suite name="My suite" parallel="methods" thread-count="5">

<suite name="My suite" parallel="tests" thread-count="5">

* parallel="methods": TestNG 将在单独的线程中运行测试方法，除了那些依赖其他测试方法的，这些将在同一个线程中运行，以保证他们的执行顺序。

* parallel="tests": TestNG 将在一个线程中运行所有在同一个<test>标签中的测试方法，但是每个<test>标签将在单独的线程中运行。这种方式容许把所有不是线程安全的类分组到相同的<test>标签中，保证他们将在相同的线程中运行，有利于 TestNG 使用尽可能多的线程来运行测试。

此外，thread-count 属性容许指定运行时将分配多少线程。

参考：
http://blog.sina.com.cn/s/blog_6f8bd74601011agy.html

TestNG+ANT

build.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="selenium_webtest_framework" default="run_test"
basedir=".">

    <!-- COMPILE TESTS -->
    <path id="libpath">
        <fileset dir="lib">
            <include name="**/*.jar" />
        </fileset>
    </path>
```

```
    <target name="compile">
        <echo message="compiling tests" />
        <delete dir="classes" />
        <mkdir dir="classes" />
        <javac encoding="UTF-8" debug="true" classpathref="libpath"
srcdir="src" destdir="classes" />
    </target>

    <!-- COPY FILES -->
    <target name="copy">
        <echo message="copy files" />
        <copy todir="classes">
            <fileset dir="src">
                <exclude name="**/*.java" />
            </fileset>
        </copy>
    </target>

    <!-- RUN TESTS -->
    <taskdef name="testng" classname="com.beust.testng.TestNGAntTask"
classpathref="libpath" />
    <path id="runpath">
        <path refid="libpath" />
        <pathelement location="classes" />
    </path>
    <target name="run">
        <echo message="running tests" />
        <delete dir="test-result" />
        <mkdir dir="test-result" />
        <testng classpathref="runpath"
outputDir="test-result/iexplore">
            <jvmarg value="-ea" />
            <xmlfileset dir="." includes="testng-iexplore.xml" />
            <!--<xmlfileset dir="." includes="testng-failed.xml" />-->
        </testng>
        <testng classpathref="runpath" outputDir="test-result/firefox">
            <jvmarg value="-ea" />
            <xmlfileset dir="." includes="testng-firefox.xml" />
            <!--<xmlfileset dir="." includes="testng-failed.xml" />-->
        </testng>
    </target>
    <target name="run_test" description="Run Test">
        <sequential>
            <antcall target="compile" />
```

```
            <antcall target="copy" />
            <antcall target="run" />
        </sequential>
    </target>
</project>
```

TestNG 报告

ReportNG

参考：

http://libin0019.iteye.com/blog/1239720

http://blog.sina.com.cn/s/blog_6966650401012m8y.html

在做自动化测试的时候发现，TestNg 原生的报告虽然内容挺全，但是展现效果却不太理想。上网发现 ReportNg 工具用来替换 TestNg 报告再好不过了。以下是我在 Ant build.xml 文件中的配置。

useDefaultListeners = "false" 用来禁止 TestNg 产生报告,但是我们还需要他的错误报告 testng-fails.xml 文件，为了方便我们只关注未通过的测试，所以还要将 TestNg 的 org.testng.reporters.FailedReporter 监听器加上。

注：org.uncommons.reportng.HTMLReporter 为 reportNg 的报告监听器

Xml 代码

```
1.  <!--指定 testNg 需要的 Jar 包-->
2.  <taskdef resource="testngtasks" classpath="${lib.dir}/testng-6.2.jar"/>
3.
4.  <target name="run_tests" depends="compile" description="执行 TestNg 测试用例">
5.      <testng classpathref="compile.path"
6.          outputDir="${output.dir}"
7.          haltOnfailure="true"
8.          useDefaultListeners="false"
9.          listeners="org.uncommons.reportng.HTMLReporter,org.testng.reporters.FailedRepo
    rter" >
10.         <!--设置 TestNg 所包含的 xml 文件-->
11.         <xmlfileset dir="${basedir}" includes="testng.xml" />
12.         <!--设置报告 Title 名称 -->
13.         <sysproperty key="org.uncommons.reportng.title" value="自动化测试报告" />
14.     </testng>
15. </target>
```

ANT+TestNG+ReportNG：

1、ANT build.xml:

```
<?xml version="1.0" ?>
<project name="testng" default="run_testng" basedir=".">
    <property name="src" value="src" />
    <property name="dest" value="classes" />
    <property name="lib.dir" value="${basedir}/lib"/>
    <property name="output.dir" value="${basedir}/OneSight"/>
    <path id="compile.path">
        <fileset dir="${lib.dir}/">
            <include name="*.jar" />
        </fileset>
        <pathelement location="${src}" />
        <pathelement location="${dest}"/>
    </path>

    <target name="init">
        <mkdir dir="${dest}" />
    </target>

    <target name="compile" depends="init">
        <javac srcdir="${src}" destdir="${dest}"
classpathref="compile.path"/>
    </target>


    <taskdef resource="testngtasks"
classpath="${lib.dir}/testng.jar"/>

    <target name="run_testng" depends="compile" description="run
testng">
        <echo>running testng</echo>
        <parallel>
            <antcall target="startServer"/>
            <sequential>
                <echo taskname="waitfor" message="Wait for proxy server
launch" />
                <waitfor maxwait="2" maxwaitunit="minute"
checkevery="100">
                    <http
url="http://localhost:4444/selenium-server/driver/?cmd=testComplete"/
>
                </waitfor>
                <antcall target="run_tests"/>
```

```
            <antcall target="stopServer"/>
        </sequential>
    </parallel>
</target>

<!-- start selenium server -->
<target name="startServer" description="start selenium server">
    <java jar="${lib.dir}/selenium-server.jar" fork="true"
spawn="true">
        <arg line="-timeout 30" />
        <jvmarg value="-Dhttp.proxyHost=proxy.corporate.com"/>
        <jvmarg value="-Dhttp.proxyPort=3128"/>
    </java>
</target>

<!-- stop selenium server -->
<target name="stopServer" description="stop selenium server">
    <get dest="result.txt"
src="http://localhost:4444/selenium-server/driver/?cmd=shutDownSeleni
umServer" ignoreerrors="true"/>
</target>

<!-- run tests -->
<target name="run_tests" depends="compile">
    <echo>running tests</echo>
    <testng classpathref="compile.path" outputdir="${output.dir}"
haltonfailure="true" usedefaultlisteners="false"

listeners="org.uncommons.reportng.HTMLReporter,org.uncommons.repo
rtng.JUnitXMLReporter" failureproperty="test.failed">
        <xmlfileset dir="${src}/" >
            <include name="com/test/testng.xml" />
        </xmlfileset>
        <sysproperty key="org.uncommons.reportng.title" value="My
Test Report"/>
    </testng>

    <fail message="test failed.." if="test.failed"/>
</target>

</project>
```

2、TestNG testing.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<suite name="WebDriverTest" verbose="10">

  <test name="Test1" preserve-order="true">
    <classes>
      <class name="com.test.WebDriver_TestNG_baidu">
        <methods>
          <include name="testWebDriverBaidu" />
          <include name="testWebDriverBaidu2" />
        </methods>
      </class>
    </classes>

  </test>

</suite>
```

3、WebDriver `WebDriver_TestNG_baidu.java`:

```
package com.test;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;


public class WebDriver_TestNG_baidu {
    private WebDriver driver;
    private String baseUrl;
    private StringBuffer verificationErrors = new StringBuffer();

    @BeforeTest
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl = "http://www.baidu.com/";
        driver.manage().timeouts().implicitlyWait(30,
TimeUnit.SECONDS);
```

```
    }

    @Test
    public void testWebDriverBaidu() throws Exception {
        driver.get(baseUrl + "/");
        driver.findElement(By.id("kw")).clear();
        driver.findElement(By.id("kw")).sendKeys("Selenium");
        driver.findElement(By.id("su")).click();
    }

    @Test
    public void testWebDriverBaidu2() throws Exception {
        driver.get(baseUrl + "/");
        driver.findElement(By.id("kw")).clear();
        driver.findElement(By.id("kw")).sendKeys("Selenium");
        driver.findElement(By.id("su1")).click();
    }

    @AfterTest
    public void tearDown() throws Exception {
        /*
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
        */
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

TestNG 报告扩展

当运行 SuiteRunner 的时候会指定一个目录，之后测试的结果都会保存在一个在那个目录中叫做 index.html 的文件中。这个 index 文件指向其他多个 HTML 和文本文件，被指向的文件包含了整个测试的结果。你可以再这里看到一个例子。

通过使用监听器和报表器，可以很轻松的生成自己的 TestNG 报表：
（1）监听器 实现接口 org.testng.ITestListener ，并且会在测试开始、通过、失败等时刻实时通知
（2）报告器 实现接口 org.testng.IReporter ，并且当整个测试运行完毕之后才会通知。IReporter 接受一个对象列表，这些对象描述整个测试运行的情况
例如，如果你想要生成一个 PDF 报告，那么就不需要实时通知，所以用 IReporter。如果需要写一个实时报告，例如用在 GUI 上，还要在每次测试时（下面会有例子和解释）有进度条或者文本报告显示点（"."），那么 ITestListener 是你最好的选择。

例如，如果你想要生成一个 PDF 报告，那么就不需要实时通知，所以用 IReporter。如果需要写一个实时报告，例如用在 GUI 上，还要在每次测试时（下面会有例子和解释）有进度条或者文本报告显示点（"."），那么 ITestListener 是你最好的选择。

1.日志监听器
这里是对每个传递进来的测试显示"."的监听器，如果测试失败则显示 "F" ，跳过则是"S"：

```
public class DotTestListener  extends TestListenerAdapter {
    private int m_count = 0;

    @Override
    public void onTestFailure(ITestResult tr) {
        log("F");
    }

    @Override
    public void onTestSkipped(ITestResult tr) {
        log("S");
    }

    @Override
    public void onTestSuccess(ITestResult tr) {
        log(".");
    }

    private void log(String string) {
        System.out.print(string);
        if (m_count++ % 40 == 0) {
            System.out.println("");
        }
    }
}
```

上例中，我们选择扩展 TestListenerAdapter ，它使用空方法实现了 ITestListener 。所以我不需要去重写那些我不需要的方法。如果喜欢可以直接实现接口。

这里是我使用这个新监听器调用 TestNG 的例子：

```
java -classpath testng.jar;%CLASSPATH% org.testng.TestNG -listener
org.testng.reporters.DotTestListener test/testng.xml
```

输出是：

```
.....
.....
.....
.....
.....
.....
===============================================
TestNG JDK 1.5
Total tests run: 226, Failures: 0, Skips: 0
===============================================
```

2.日志报表

org.testng.IReporter 接口只有一个方法：

```
public void generateReport(List<ISuite> suites, String outputDirectory)
```

这个方法在 TestNG 中的所有测试都运行完毕之后被调用，这样你可以修改这个方法的参数，并且通过它们获得刚刚完成的测试的所有信息。

3.JUnit 报表

TestNG 包含了一个可以让 TestNG 的结果和输出的 XML 能够被 JUnitReport 所使用的监听器。这里有个例子，并且 ant 任务创建了这个报告：

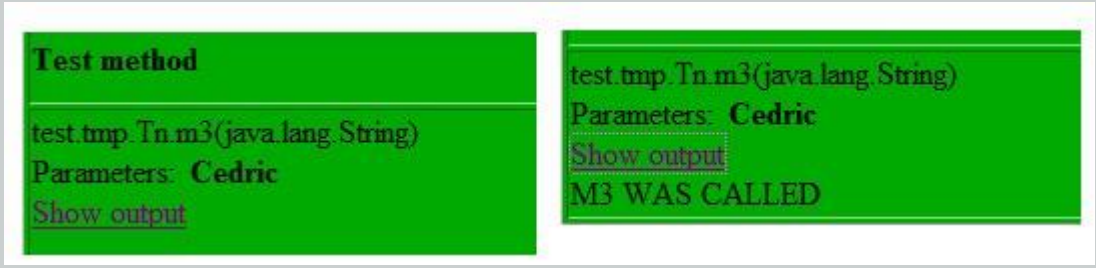
```
<target name="reports">
    <junitreport todir="test-report">
        <fileset dir="test-output">
            <include name="*/*.xml"/>
        </fileset>

        <report format="noframes"   todir="test-report"/>
    </junitreport>
</target>
```

注意： 由于 JDK 1.5 和 JUnitReports 不兼容性，导致了 frame 版本不能够正常工作，所以你需要指定 "noframes" 使其能够正常工作。

4.报表 API

如果你要在 HTML 报告中显示日志信息，那么就要用到类 org.testng.Reporter：

```
        Reporter.log("M3 WAS CALLED");
```



5. XML 报表

TestNG 提供一种 XML 报表器，使得能够捕捉到只适用于 TestNG 而不适用与 JUnit 报表的那些特定的信息。这在用户的测试环境必须要是用 TestNG 特定信息的 XML，而 JUnit 又不能够提供这些信息的时候非常有用。下面就是这种报表器生成 XML 的一个例子：

```
<testng-results>
        <suite name="Suite1">
                <groups>
                        <group name="group1">
                                <method signature="com.test.TestOne.test2()"
name="test2" class="com.test.TestOne"/>
                                <method signature="com.test.TestOne.test1()"
name="test1" class="com.test.TestOne"/>
                        </group>
                        <group name="group2">
                                <method signature="com.test.TestOne.test2()"
name="test2" class="com.test.TestOne"/>
                        </group>
                </groups>
                <test name="test1">
                        <class name="com.test.TestOne">
                                <test-method status="FAIL" signature="test1()"
name="test1" duration-ms="0"
                                                                started-at="2007-05-2
8T12:14:37Z" description="someDescription2"
                                                                finished-at="2007-05-
28T12:14:37Z">
                                                <exception
class="java.lang.AssertionError">
                                                        <short-stacktrace>java.lang.Ass
ertionError
                                                                ... Removed 22 stack
frames
                                                        </short-stacktrace>
                                                </exception>
                                </test-method>
```

```
                                <test-method status="PASS" signature="test2()"
name="test2" duration-ms="0"
                                                started-at="2007-05-2
8T12:14:37Z" description="someDescription1"
                                                finished-at="2007-05-
28T12:14:37Z">
                                </test-method>
                                <test-method status="PASS" signature="setUp()"
name="setUp" is-config="true" duration-ms="15"
                                                started-at="2007-05-2
8T12:14:37Z" finished-at="2007-05-28T12:14:37Z">
                                </test-method>
                        </class>
                </test>
        </suite>
</testng-results>
```

这个报表器是随着默认监听器一起诸如的，所以你默认的情况下就可以得到这种类型的输出。这个监听器提供了一些属性，可以修改报表来满足你的需要。下表包含了这些属性，并做简要说明：

属性	说明	Default value
outputDirectory	一个 String 指明了 XML 文件要存放的目录	TestNG 输出目录
timestampFormat	指定报表器生成日期字段的格式	yyyy-MM-dd'T'HH:mm:ss'Z
fileFragmentationLevel	值为 1，2 或 3 的整数，指定了 XML 文件的生成方式： 1 - 在一个文件里面生成所有的结果 2 - 每套测试都单独生成一个 XML 文件，这些文件被链接到一个主文件 3 - 同 2，不过添加了引用那些套测试的测试用例的 XML 文件	1
splitClassAndPackageNames	boolean 值，指明了对 <class> 元素的生成方式。例如，你在 false 的时候得到 <class class="com.test.MyTest">，在 true 的时候 <class class="MyTest" package="com.test">。	false
generateGroupsAttribute	boolean 值指定了对<test-method> 元素是否生成 groups 属性。这个功能的目的是，对那些无需遍历整个 <group> 元素，且只含有一个测试方法的组来说，可以提供一种更直接的解析组的方式。	false
stackTraceOutputMethod	指定在发生异常的时候生成异常，追踪弹栈信息的类型，有如下可选值： 0 - 无弹栈信息（只有异常类和消息）. 1 - 尖端的弹栈信息，从上到下只有几行 2 - 带有所有内部异常的完整的弹栈方式	2

	3 - 短长两种弹栈方式	
generateDependsOnMethods	对于<test-method>元素，使用这个属性来开启/关闭 depends-on-methods 属性。	true
generateDependsOnGroups	对于<test-method>属性开启 depends-on-groups 属性。	true

为了配置报表器，你可以在命令行下使用 -reporter 选项，或者在 Ant 任务中嵌入 <reporter> 元素。对于每个情况，你都必须指明类 org.testng.reporters.XMLReporter。但是要注意你不能够配置内建的报表器，因为这个只使用默认配置。如果你的确需要 XML 报表，并且使用自定义配置，那么你就不得不手工完成。可以通过自己添加一两个方法，并且禁用默认监听器达到目的。

Robot Framework SeleniumLibrary

http://code.google.com/p/robotframework-seleniumlibrary/
http://www.51testing.com/?uid-128466-action-viewspace-itemid-807706

ISFW（InfoStretch's Framework for Automated Web Testing with Selenium）

下载：
http://www.infostretch.com/QA/selenium-framework.php

What is the framework? Tool of Choice

InfoStretch has developed a framework for Web application testing and test automation using Selenium test tools that focuses on achieving a rapid return on investment – often in less than three months. The InfoStretch framework using Selenium has a robust set of pre-built features that include:

- Parallel compatibility testing for many browsers from the cloud, with zero infrastructure costs..
- Integration scenario testing across UI's and Web services.
- Highly maintainable and repeatable tests that utilize reusable test assets, proper modularity and semantic structure.
- Data-driven Selenium tests with external Excel, XML and CSV-based data.

Bring efficiencies to your testing process

Automate your entire testing needsIntegration into many continuous integration/build automation systems such as Hudson.

Achieve your goals in testing efficienciesBest practices for coding conventions and process guidelines, and coaching on how to make apps "testable."

Realize a solid ROIBuild Selenium tests faster to save time and free up resources.

Selenium IDE Plug-in for InfoStretch Framework

http://blog.infostretch.com/selenium-ide-plug-in-for-infostretch-framework

For non technical testers Selenium IDE is the ideal environment for creating Selenium tests. Selenium IDE provides code formatter for different language and/or testing frameworks.
Choosing the right framework or scripting technique helps in maintaining lower costs. The approach of scripting used during test automation has effect on cost due to development and maintenance efforts.

Various scripting techniques are generally used when developing test automation scripts are:

- **Linear** : Procedural code, possibly generated by tools like Selenium IDE
- **Structured**: Uses control structures – for instance 'if-else', 'switch', 'for', 'while' conditions or flow control statements
- **Data-driven**: Data is persisted outside of tests in a database, csv files, or other mechanism
- **Modularity-driven:** Creation of small, independent scripts that represent modules, sections, and functions of the application-under-test
- **Hybrid**: Two or more of the techniques above are used

Adding Functional programming capability to IDE

If you are using selenium IDE for developing your tests, then using InfoStretch Selenium IDE plug-in you can record scripts that turn out to be **Structured, Data-driven, and Modularity-driven** or **Hybrid.**Plug-in provides code formatter to export recoded steps in IDE to "InfoStretch Test Automation Framework (ISFW)" format. In addition to code formatter for ISFW, following functional programming capability added to IDE:

- Defining reusable steps as modules
  - supports parameters
  - can return value
- Call defined modules in tests
- Conditional Flow control
- Looping

- Nested looping and conditional flow

Through InfoStretch plug-in you can apply different approaches while recording using selenium IDE, for instance

- Scripts generated by Selenium IDE using record facility be the **linear**
- By use of control structures commands – typically '*if*', *'else'*, *'elseif'*, '*while*' conditions/ statements provided by InfoStretch plug-in will make scripts **Structured**
- Creation of small, independent modules of the application-under-test using '*defineModule*' command provided by InfoStretch plug-in be useful in developing **Modularity-driven** test
- Call command provided by InfoStretch plug-in be useful call defined modules in test
- Define module with parameters and use of data-provider will make script **Data-driven**

Thus three steps for non technical users are:

1. Record using Selenium IDE
2. Edit as per requirements
3. Export test case as "ISFW" format from IDE.

Hermes - Selenium For Humans

Automated regression testing is vital for an effective Agile QA approach.
Hermes is the only test automation framework to be fully integrated with Selenium's entire suite of tools - including Selenium GRID.
Hermes comes with advanced reporting, analysis, debugging and enterprise level Java development capabilities, all accessed via Eclipse.
Hermes provides a powerful yet usable automation solution, making it the most popular Selenium framework in the world, with over 4,500 downloads.
It is available under an open source license and for FREE .

下载：
http://sourceforge.net/projects/hrmes/

Hermes provides a powerful yet usable automation solution, making it the most popular Selenium framework in the world, with over 4,000 downloads.

Create human readable, machine executable Selenese style tests in Excel. The world's first automation framework fully integrated with Selenium IDE, RC and GRID; yet built for humans. Be up and running using a feature rich Selenium RC framework in minutes.

Hermes comes with advanced reporting, analysis, debugging and enterprise level Java development capabilities, all accessed via Eclipse.

With Hermes you can run automated tests on 32 different OS and browser version combinations in

parallel and instantly on a Cloud infrastructure.

Contact us today and receive £30 worth of FREE access to cloud infrastructure:www.w3qa.eu.
Hermes - Selenium For Humans Web Site

WatiP

基于 Python 的模块化框架
D:\selenium\Selenium 框架\Watip

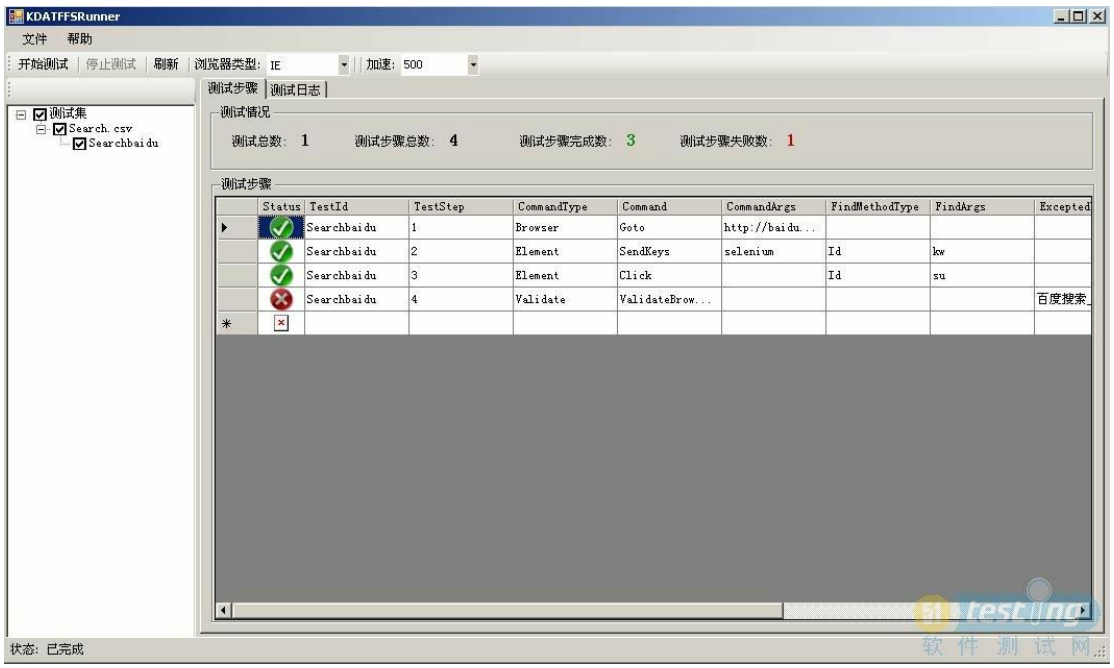
KDATFFS

KDATFFS 作为一个基于 Selenium 的关键字驱动的自动化框架，其中包括 KDATFFS，KDATFFSRecorder,KDATFFSRunner. KDATFFS 是框架的核心，KDATFFSRecorder 是一个测试录制工具，KDATFFSRunner 是一个测试运行工具。

开源下载地址：
http://kdatffs.codeplex.com/

主要实现以下一些功能:
- 不需编写测试代码实现自动化测试
- 多种测试数据载体(Excel,CSV)
- 支持 IE,Firefox 浏览器,Firefox 可以使用代理
- 支持 JS 弹出框,弹出窗口
- 支持 IFrame 结构
- 支持页面元素，图片以及数据库数据的验证功能
- 支持测试结果，测试日志输出
- 良好的 GUI 操作和显示

关键字驱动：



Open2Test Framework for Selenium

http://www.open2test.org/index.html

Selenium RC Frameworks（Ruby）

Selenium 2.0 Framework（Java）

```
package Scripts;

import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.apache.poi.hssf.model.InternalWorkbook;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.junit.*;
import java.util.*;
import java.util.NoSuchElementException;
import java.io.*;
```

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.Select;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Locale;
import jxl.Cell;
import jxl.CellType;
import jxl.Sheet;
import jxl.Workbook;
import jxl.WorkbookSettings;
import jxl.DateCell;
import jxl.read.biff.BiffException;
import java.lang.reflect.*;

public class Sample {
    private WebDriver driver;
    private String baseUrl="";
    private StringBuffer verificationErrors = new StringBuffer();
    @Before
    public void main ()throws Exception {
        driver = new InternetExplorerDriver();
        driver.manage().timeouts().implicitlyWait(30,
TimeUnit.SECONDS);
    }

    @Test
    public void testNewuser() throws Exception {
        {
            Workbook workbook = Workbook.getWorkbook(new
File("D:/selenium/New.xls"));
           Sheet sheet = workbook.getSheet(0);
            String TestSuite = sheet.getCell(1, 1).getContents();
            String TestScript = sheet.getCell(1,2).getContents();
            String ObjectRepository = sheet.getCell(1,3).getContents();
            System.out.println(TestSuite);
            TestSuite(TestSuite, TestScript,ObjectRepository);
        }
    }
```

```
        private void TestSuite(String TestSuite, String TestScript,
String ObjectRepository) throws Exception {
            int TSrowcount = 0;
            FileInputStream fs = null;
            WorkbookSettings ws = null;
            fs = new FileInputStream(new File(TestSuite));
            ws = new WorkbookSettings();
            ws.setLocale(new Locale("en", "EN"));
            Workbook TSworkbook = Workbook.getWorkbook(fs, ws);
            Sheet TSsheet = TSworkbook.getSheet(0);
            TSrowcount = TSsheet.getRows();
            for (int i = 0; i< TSsheet.getRows();i++){
                String TSvalidate = "r";
                if (((TSsheet.getCell(0,
i).getContents()).equalsIgnoreCase(TSvalidate) == true)){
                    String TCStatus = "Pass";
                   String ScriptName= TSsheet.getCell(1,
i).getContents();
                    Keyword(ScriptName,TestScript,ObjectRepository);
                }
                else {
                    System.out.println(TSvalidate);
                }
                }
    }

    private void Keyword(String scriptName, String TestScript, String
ObjectRepository) throws Exception {
        String scriptPath = TestScript + scriptName;
        int TScrowcount = 0;
        String Action = "";
        String cCellData = "";
        String dCellData = "";
        int TScrowcountdata = 0;
        int loopcount = 0;
        int looptimes = 0;
        int loopstart = 0;
        FileInputStream fs1 = null;
        WorkbookSettings ws1 = null;
        fs1 = new FileInputStream(new File(scriptPath));
        ws1 = new WorkbookSettings();
        ws1.setLocale(new Locale("en", "EN"));
        Workbook TScworkbook = Workbook.getWorkbook(fs1, ws1);
        Sheet TScsheet = TScworkbook.getSheet(0);
```

```
        TScrowcount = TScsheet.getRows();
        System.out.println(scriptPath);
        Sheet TScsheetdata = TScworkbook.getSheet(1);
        TScrowcountdata = TScsheet.getRows();
        for (int j=0;j<TScrowcount;j++){
            int loop_flag = 0;
            int Callaction_flag = 0;
            String TSvalidate = "r";
            if (((TScsheet.getCell(0,
j).getContents()).equalsIgnoreCase(TSvalidate) == true)){
                Action = TScsheet.getCell(1, j).getContents();
                cCellData = TScsheet.getCell(2, j).getContents();
                dCellData = TScsheet.getCell(3, j).getContents();
                String ORPath = ObjectRepository;
                String strORvalname = "";
                String ORvalue = "";
                String ObjectVal = "";
                int ORrowcount = 0;
                FileInputStream fs2 = null;
                WorkbookSettings ws2 = null;
                fs2 = new FileInputStream(new File(ORPath));
                ws2 = new WorkbookSettings();
                ws2.setLocale(new Locale("en", "EN"));
                Workbook ORworkbook = Workbook.getWorkbook(fs2, ws2);
                Sheet ORsheet = ORworkbook.getSheet(0);
                ORrowcount = ORsheet.getRows();
                String strActionVal = Action.toLowerCase();
                switch (ActionVal.getActionVal(strActionVal)){

                case perform:
                    String[] cCellDataVal = cCellData.split(";");
                    ObjectVal = cCellDataVal[0];
                    String[] dCellDataVal = dCellData.split(":");

                    String strObjectSet = dCellDataVal[0];
                    String ObjectSetVal ="";
                    if(dCellDataVal.length>1)
                        ObjectSetVal = dCellDataVal[1];
                    for (int k=0;k<ORrowcount;k++){
                        String ORName =
ORsheet.getCell(0,k).getContents();
                        if (((ORsheet.getCell(0,
k).getContents()).equalsIgnoreCase(ObjectVal) == true)){
                            String[] ORcellData =
```

```
(ORsheet.getCell(1,k).getContents()).split(";");
                        strORvalname = ORcellData[0];
                        ORvalue = ORcellData[1];
                        k = ORrowcount+1;
                    }
                }
                switch (ObjectSet.getObjectSet(strObjectSet)){
                case set:
                    switch (ORvalname.getORvalname(strORvalname)){
                    case id:

    driver.findElement(By.id(ORvalue)).sendKeys(ObjectSetVal);
                        break;
                    case name:

    driver.findElement(By.name(ORvalue)).sendKeys(ObjectSetVal);
                        break;
                    }
                    break;
                case click:
                    switch (ORvalname.getORvalname(strORvalname)){
                    case id:
                        driver.findElement(By.id(ORvalue)).click();
                        break;
                    case name:

    driver.findElement(By.name(ORvalue)).click();
                        break;
                    }
                    break;
                case submit:
                    switch (ORvalname.getORvalname(strORvalname)){
                    case id:
                        driver.findElement(By.id(ORvalue)).submit();
                        break;
                    case name:

    driver.findElement(By.name(ORvalue)).submit();
                        break;
                    }
                    break;
                case launchapp:
```

```
                String Apppath = cCellData;
                driver.get(Apppath);
                break;
            case loop:
                cCellDataVal = cCellData.split(";");
                loopstart = Integer.valueOf(cCellDataVal[0]);
                int loopend = Integer.valueOf(cCellDataVal[1]);
                looptimes = Integer.valueOf(dCellData);
            break;
            case endloop:
                if (loopcount != looptimes){
                    j = loopstart;
                    loopcount = loopcount+1;
                } else {
                    loopstart = 0;
                    loopcount = 0;
                    looptimes = 0;
                }
            case function:
                String funcName = cCellData;
                String param = dCellData;
                funcName(param);
            case check:
                String[] cCellDataValChk = cCellData.split(";");
                String ObjectValChk = cCellDataValChk[1];
                String[] dCellDataValChk = dCellData.split(":");
                String ObjectSetChk = dCellDataValChk[0];
                String ObjectSetValChk = dCellDataValChk[1];
                for (int k=0;k<ORrowcount;k++){
                    String ORName =
ORsheet.getCell(0,k).getContents();
                    if (((ORsheet.getCell(0,
k).getContents()).equalsIgnoreCase(ObjectVal) == true)){
                        String[] ORcellData =
(ORsheet.getCell(1,k).getContents()).split(";");
                        strORvalname = ORcellData[0];
                        ORvalue = ORcellData[1];
                        k = ORrowcount+1;
                    }
                }
                switch (ObjectSet.getObjectSet(ObjectSetChk)){
                case enabled:
                    switch (ORvalname.getORvalname(strORvalname)){
                    case id:
```

```
    driver.findElement(By.id(ORvalue)).isEnabled();
                        break;
                    case name:

    driver.findElement(By.name(ORvalue)).isEnabled();
                        break;
                    }
                case selected:
                    switch (ORvalname.getORvalname(strORvalname)){
                    case id:

    driver.findElement(By.id(ORvalue)).isSelected();
                        break;
                    case name:

    driver.findElement(By.name(ORvalue)).isSelected();
                        break;
                    }
                case exist:
                    switch (ORvalname.getORvalname(strORvalname)){
                    case id:

    driver.findElement(By.id(ORvalue)).isDisplayed();
                        break;
                    case name:

    driver.findElement(By.name(ORvalue)).isDisplayed();
                        break;
                    }
                }
                break;
            }
            }
        }
        }


private void funcName(String param) {


    }

/*  @After
```

```
    public void tearDown() throws Exception {
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }
*/
    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

日志和报告框架

LoggingSelenium

仅适用于 Selenium RC

下载：
http://sourceforge.net/projects/loggingselenium/files/

使用手册：
http://loggingselenium.sourceforge.net/usage.html

例子：

```
package com.testscripts;

import static
com.unitedinternet.portal.selenium.utils.logging.LoggingAssert.assert
Equals;
import static
com.unitedinternet.portal.selenium.utils.logging.LoggingAssert.assert
```

```
True;

import java.io.BufferedWriter;
import java.io.File;
import java.io.IOException;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import com.thoughtworks.selenium.HttpCommandProcessor;
import
com.unitedinternet.portal.selenium.utils.logging.HtmlResultFormatter;
import
com.unitedinternet.portal.selenium.utils.logging.LoggingCommandProces
sor;
import
com.unitedinternet.portal.selenium.utils.logging.LoggingDefaultSeleni
um;
import
com.unitedinternet.portal.selenium.utils.logging.LoggingResultsFormat
ter;
import
com.unitedinternet.portal.selenium.utils.logging.LoggingSelenium;
import com.unitedinternet.portal.selenium.utils.logging.LoggingUtils;

public class Test2 {

    protected LoggingSelenium selenium;

    private BufferedWriter loggingWriter;

    private static final String RESULT_FILE_ENCODING = "UTF-8";

    private static final String DEFAULT_TIMEOUT = "30000";

    private static final String OPENQA_URL = "http://openqa.org";

    private static final String SCREENSHOT_PATH = "screenshots";

    private final String RESULTS_BASE_PATH = "target" + File.separator
            + "loggingResults";
```

```
    private String resultsPath = new
File(RESULTS_BASE_PATH).getAbsolutePath();

    private String screenshotsResultsPath = new File(RESULTS_BASE_PATH
            + File.separator + SCREENSHOT_PATH).getAbsolutePath();

    @Before
    public void setUp() throws Exception {

        if (!new File(screenshotsResultsPath).exists()) {
            new File(screenshotsResultsPath).mkdirs();
        }
        // to keep every result use a timestamped filename like this
        // final String resultHtmlFileName = resultsPath
        // + File.separator
        // + "autorunResult"
        // + LoggingUtils.timeStampForFileName()
        // + ".html";
        final String resultHtmlFileName = resultsPath + File.separator
                + "Results.html";
        //System.err.println("resultHtmlFileName=" +
resultHtmlFileName);

        loggingWriter = LoggingUtils.createWriter(resultHtmlFileName,
                Test2.RESULT_FILE_ENCODING, true);

        LoggingResultsFormatter htmlFormatter = new HtmlResultFormatter(
                loggingWriter,
                Test2.RESULT_FILE_ENCODING);
        htmlFormatter
                .setScreenShotBaseUri(Test2.SCREENSHOT_PATH
                        + "/"); // has to be "/" as this is a URI

    htmlFormatter.setAutomaticScreenshotPath(screenshotsResultsPath);
        LoggingCommandProcessor myProcessor = new
LoggingCommandProcessor(
                new HttpCommandProcessor("localhost", 4444, "*chrome",
                        OPENQA_URL), htmlFormatter);
        myProcessor.setExcludedCommands(new String[] {});
        selenium = new LoggingDefaultSelenium(myProcessor);
        selenium.start();

    }
```

```
    @After
    public void tearDown() throws Exception {
        selenium.stop();
        try {
            if (null != loggingWriter) {
                loggingWriter.close();
            }
        } catch (IOException e) {
            // do nothing
        }
    }

    @Test
    public void SeleniumLogging_Test1() {

    selenium.open("file:///D:/selenium/MyTest/AUT/testAlert.html");
        selenium.click("id=btnAlert");

        selenium.captureScreenshot(screenshotsResultsPath +
File.separator
                + "MyTest_" + LoggingUtils.timeStampForFileName() +
".png");

        assertEquals("Expected Alert Show", "I'm blocking!", selenium
                .getAlert().toString(), selenium);

    }

}
```

Log4J

1、引用 log4j 的 jar 包

2、建立 log4j.xml 文件：

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

<appender name="appender1"
class="org.apache.log4j.DailyRollingFileAppender">
<!-- 设置通道ID:org.zblog.all和输出方式:
```

```
org.apache.log4j.DailyRollingFileAppender -->
   <param name="File" value="C:/log4j_1.log" /><!-- 设置File参数：日志输出文件名 -->
   <param name="Append" value="true" /><!-- 设置是否在重新启动服务时，在原有日志的基础添加新日志 -->

    <param name="BufferSize" value="8192" />
    <param name="ImmediateFlush" value="false" />
    <param name="BufferedIO" value="true" />

   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" /><!-- 设置输出文件项目和格式 -->
   </layout>
</appender>


<appender name="appender2"
class="org.apache.log4j.DailyRollingFileAppender">
   <param name="File" value="C:/log4j_2.log" />
   <param name="Append" value="true" />

   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" />
   </layout>
</appender>


<appender name="appender3" class="org.apache.log4j.ConsoleAppender">
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" />
   </layout>
</appender>


<root> <!-- 设置接收所有输出的通道 -->
    <appender-ref ref="appender2" /><!-- 与前面的通道id相对应 -->
   <appender-ref ref="appender3" />
</root>

</log4j:configuration>
```

3、在代码中使用 log4j：

```
package com.test;
```

```
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
import org.junit.Test;


public class Log4J_Test1 {

    @Test
    public void Test1(){
        DOMConfigurator.configure("src/log4j.xml");//加载.xml文件
        Logger log=Logger.getLogger("appender1");
        log.info("My Test1!");
        log.debug("debug message!");
        log.warn("Warning Message!");
        log.error("Error Message!");
        //log.shutdown();
    }

}
```

TestNG Reporter

import org.testng.Reporter;

Reporter.log("TestNG Reporter:Setup!",1,true);

在命令行及 TestNG 报告中可查看输出。

Log4J+TestNG Reporter

封装 log4j 与 Reporter：

package cn.opentest.utils;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.text.SimpleDateFormat;
import java.util.Calendar;

```
import java.util.Properties;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.testng.Reporter;

public class SelLogger {

	private static Logger logger = null;
	private static SelLogger logg = null;

	public static SelLogger getLogger(Class<?> T) {
		if (logger == null) {
			Properties props = new Properties();

			try {
				InputStream is = new
FileInputStream("src//cn//opentest//utils//log4j.properties");
				props.load(is);
			} catch (IOException e) {
				e.printStackTrace();
			}

			PropertyConfigurator.configure(props);
			logger = Logger.getLogger(T);
			logg = new SelLogger();
		}
		return logg;
	}

	// 重写 logger 方法
	public void log(String msg) {
		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
		Calendar ca = Calendar.getInstance();
		logger.info(msg);
		Reporter.log("Reporter:" + sdf.format(ca.getTime()) + "===>" + msg);
	}

	public void debug(String msg) {
		logger.debug(msg);
	}

	public void warn(String msg) {
		logger.warn(msg);
```

```
        Reporter.log("Reporter:" + msg);
    }

    public void error(String msg) {
        logger.error(msg);
        Reporter.log("Reporter:" + msg);
    }
}
```

在代码中使用：

```
package com.test;

import java.util.concurrent.TimeUnit;

import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Reporter;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;


public class WebDriver_TestNG_Log4J_Reporter {
    private WebDriver driver;
    private String baseUrl;
    private StringBuffer verificationErrors = new StringBuffer();
    private Logger log;

    private static final SelLogger logger =
SelLogger.getLogger(WebDriver_TestNG_Log4J_Reporter.class);


    @BeforeTest
    public void setUp() throws Exception {

        driver = new FirefoxDriver();
        baseUrl = "http://www.baidu.com/";
        driver.manage().timeouts().implicitlyWait(15,
```

```
TimeUnit.SECONDS);


    DOMConfigurator.configure("C:/workspace/SeleniumTest1/src/log4j.xml");//加载.xml文件
        log=Logger.getLogger("appender2");

        Reporter.log("TestNG Reporter:Setup!",1,true);
        log.info("log4j:Setup!");
        logger.log("this is SelLogger Message!");
    }

    @Test
    public void testWebDriverBaidu() throws Exception {
        Reporter.log("TestNG Reporter:Start testWebDriverBaidu!",1);
        log.info("log4j:Start My Test1!");
        logger.log("SelLogger:Start My Test1!");

        driver.get(baseUrl + "/");
        driver.findElement(By.id("kw")).clear();
        driver.findElement(By.id("kw")).sendKeys("Selenium");
        driver.findElement(By.id("su")).click();

        Reporter.log("TestNG Reporter:End testWebDriverBaidu!",1);
        log.info("log4j:End My Test1!");
        logger.log("SelLogger:End My Test1!");
    }


    @AfterTest
    public void tearDown() throws Exception {

        /*
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
        */
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
```

```
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

截图

封装 logPicture 方法：

```
    public void logPicture(WebDriver dr) throws IOException {
    File screenShotFile =
((TakesScreenshot)dr).getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(screenShotFile, new
File("C:\\AutoScreenCapture\\" + getCurrentDateTime()+ ".jpg"));
    }
```

使用 logPicture 方法：

logger.logPicture(driver);

录屏

Castro

Selenium Grid

启动 hub：

C:\>java -jar selenium-server-standalone-2.25.0.jar -role hub

启动节点：

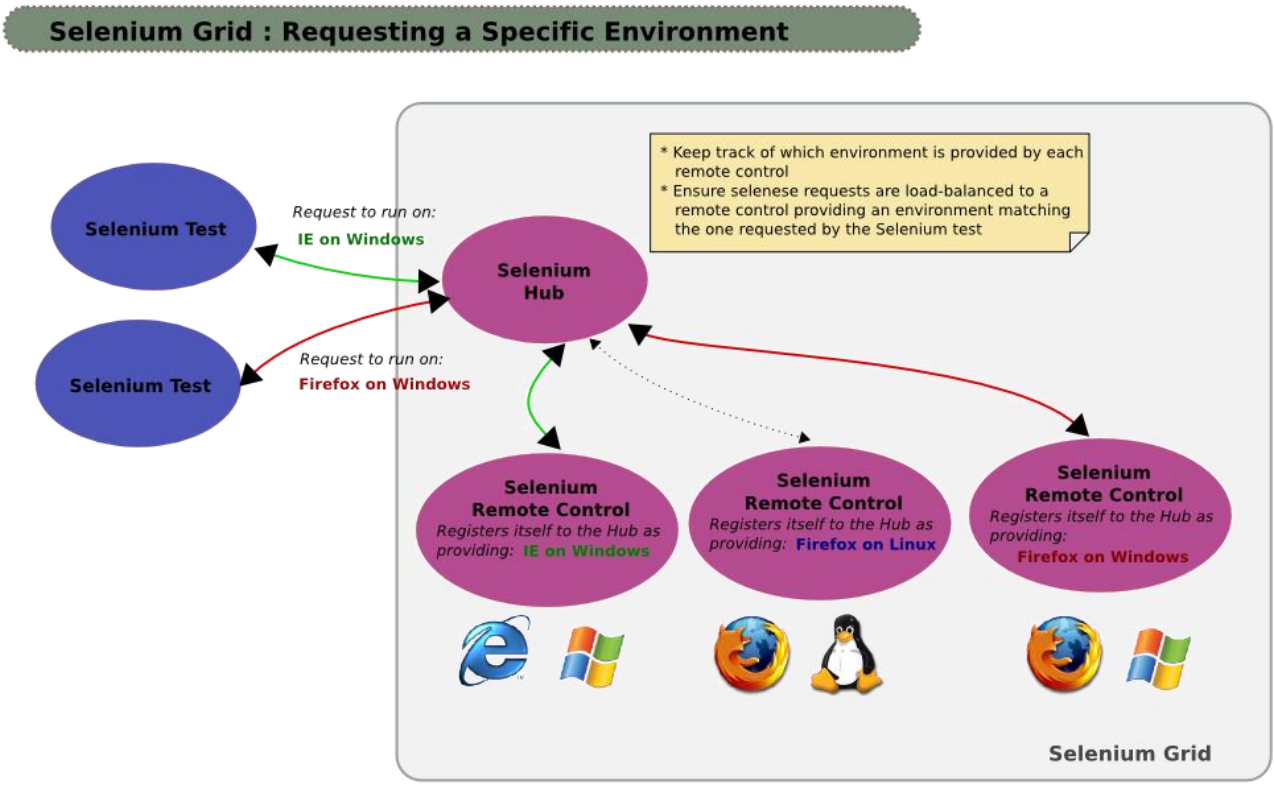
C:\>java -jar selenium-server-standalone-2.25.0.jar -role node -hub http://localhost:4444/grid/register -browser browserName="internet explorer",version=6,platform=WINDOWS

代码：

```
package com.test;

import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class WebDriver_RemoteDriver {

    private WebDriver driver;
    private String baseUrl;
    private StringBuffer verificationErrors = new StringBuffer();
    @Before
    public void setUp() throws Exception {
        DesiredCapabilities capability = new DesiredCapabilities();
        /*
        capability.setBrowserName("internet explorer");
        capability.setPlatform(Platform.WINDOWS);
        capability.setVersion("6");
        */
        capability.setBrowserName("firefox");
        capability.setPlatform(Platform.WINDOWS);
        capability.setVersion("12");

        driver = new RemoteWebDriver(new
URL("http://192.168.0.188:4444/wd/hub"), capability);

        baseUrl = "http://www.baidu.com/";
        driver.manage().timeouts().implicitlyWait(30,
TimeUnit.SECONDS);
    }

    @Test
    public void testWebDriverBaidu() throws Exception {
```

```
        driver.get(baseUrl + "/");
        driver.findElement(By.id("kw")).clear();
        driver.findElement(By.id("kw")).sendKeys("Selenium");
        driver.findElement(By.id("su")).click();
    }

    @After
    public void tearDown() throws Exception {
        /*
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
        */
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }


}
```

参考：
http://code.google.com/p/selenium/wiki/Grid2


Selenium Grid : Requesting a Specific Environment
Selenium Test
Request to run on:
IE on Windows
Selenium Test
Request to run on:
Firefox on Windows
Selenium Hub
* Keep track of which environment is provided by each remote control
* Ensure selenese requests are load-balanced to a remote control providing an environment matching the one requested by the Selenium test
Selenium Remote Control
Registers itself to the Hub as providing: IE on Windows
Selenium Remote Control
Registers itself to the Hub as providing: Firefox on Linux
Selenium Remote Control
Registers itself to the Hub as providing:
Firefox on Windows
Selenium Grid

Introduction

Grid allows you to :

- scale by distributing tests on several machines (parallel execution)
- manage multiple environments from a central point, making it easy to run the tests against a vast combination o
- minimize the maintenance time for the grid by allowing you to implement custom hooks to leverage virtual infra

Quick Start

This example will show you how to start the Selenium 2 Hub, and register both a WebDriver node and a Selenium 1 RC grid from Java. The hub and nodes are shown here running on the same machine, but of course you can copy the seleniu

Note: The selenium-server-standalone package includes the Hub, WebDriver, and legacy R required anymore. You can download the selenium-server-standalone-*.jar from http://code.google.com/p/selenium/downloads/list. This walk-through assumes you

Step 1: Start the hub

The Hub is the central point that will receive all the test request and distribute th

Open a command prompt and navigate to the directory where you copied the selenium-server command:

```
java -jar selenium-server-standalone-2.14.0.jar -role hub
```

The hub will automatically start-up using port 4444 by default. To change the default por -port when you run the command. You can view the status of the hub by opening a brows to: http://localhost:4444/grid/console

Step 2: Start the nodes

Regardless on whether you want to run a grid with new WebDriver functionality, or a gr or both at the same time, you use the same selenium-server-standalone jar file to sta

```
java -jar selenium-server-standalone-2.14.0.jar -role node  -hub http://localhost:4444/grid/register
```

Note: The port defaults to 5555 if not specified whenever the "-role" option is prov

For backwards compatibility "wd" and "rc" roles are still a valid subset of the "node" of remote connections to their corresponding API, while "node" allows both RC and Web

Using grid to run tests

(using java as an example) Now that the grid is in-place, we need to access the grid
1 RC nodes, you can continue to use the DefaultSelenium object and pass in the hub i

```
Selenium selenium = new DefaultSelenium("localhost", 4444, "*firefox", "http://www.google.com");
```

For WebDriver nodes, you will need to use the RemoteWebDriver and the DesiredCapabili
version and platform you wish to use. Create the target browser capabilities you wan

```
DesiredCapabilities capability = DesiredCapabilities.firefox();
```

Pass that into the RemoteWebDriver object:

```
WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);
```

The hub will then assign the test to a matching node.

A node matches if all the requested capabilities are met. To request specific capabili
passing it into the WebDriver object.

```
capability.setBrowserName();
capability.setPlatform();
capability.setVersion()
capability.setCapability(,);
```

Example: A node registered with the setting:

```
 -browser  browserName=firefox,version=3.6,platform=LINUX
```

will be a match for:

```
capability.setBrowserName("firefox" );
capability.setPlatform("LINUX");
capability.setVersion("3.6");
```

and would also be a match for

```
capability.setBrowserName("firefox" );
capability.setVersion("3.6");
```

The capabilities that are not specified will be ignored. If you specify capabilities that
your test specifies Firefox version 4.0, but have no Firefox 4 instance) then there wi
to run.

Configuring the nodes

The node can be configured in 2 different ways; one is by specifying command line para
json file.

Configuring the nodes by command line

By default, this starts 11 browsers : 5 Firefox, 5 Chrome, 1 Internet Explorer. The m
set to 5 by default. To change this and other browser settings, you can pass in parameters
represents a node based on your parameters). If you use the -browser parameter, the defa
what you specify command line will be used.

```
-browser browserName=firefox,version=3.6,maxInstances=5,platform=LINUX
```

This setting starts 5 Firefox 3.6 nodes on a linux machine.

If your remote machine has multiple versions of Firefox you'd like to use, you can map the
version on the same machine:

```
-browser browserName=firefox,version=3.6,firefox_binary=/home/myhomedir/firefox36/firefox,maxInstances=3,
browserName=firefox,version=4,firefox_binary=/home/myhomedir/firefox4/firefox,maxInstances=4,platform=LIN
```

Tip: If you need to provide a space somewhere in your browser parameters, then surround

```
-browser "browserName=firefox,version=3.6,firefox_binary=c:\Program Files\firefox ,maxInstances=3, platfo
```

Optional parameters

- -port 4444 (4444 is default)
- -timeout 30 (30 is default) The timeout in seconds before the hub automatically releases a node that hasn't receiv
  number of seconds. After this time, the node will be released for another test in the queue. This helps to clear cli
  remove the timeout completely, specify -timeout 0 and the hub will never release the node.

Note: This is NOT the WebDriver timeout for all "wait for WebElement" type of commands.

- -maxSession 5 (5 is default) The maximum number of browsers that can run in parallel on the node. This is diff
  (Example: For a node that supports Firefox 3.6, Firefox 4.0 and Internet Explorer 8, maxSession=1 will ensure
  With maxSession=2 you can have 2 Firefox tests at the same time, or 1 Internet Explorer and 1 Firefox test).

- -browser < params > If -browser is not set, a node will start with 5 firefox, 1 chrome, and 1 internet explorer ins
  parameter can be set multiple times on the same line to define multiple types of browsers.

Parameters allowed for -browser: browserName={android, chrome, firefox, htmlunit, in
version={browser version} firefox_binary={path to executable binary} maxInstances={max
platform={WINDOWS, LINUX, MAC}

- -registerCycle = how often in ms the node will try to register itself again.Allow to restart the hub without having
- Relly large (>50 node) Hub installations may need to increase the jetty threads by setting -DPOOL_MAX=512

Configuring timeouts (Version 2.21 required)

Timeouts in the grid should normally be handled through webDriver.manage().timeouts(),
operations time out.

To preserve run-time integrity of a grid with selenium-servers, there are two other

On the hub, setting the -timeout command line option to "30" seconds will ensure all res
a client crashes. On the hub you can also set -browserTimeout 60 to make the maximum t
the browser 60 seconds. This will ensure all resources are reclaimed slightly after 60
values from the hub if they are set. Locally set parameters on a single node has preced
to set these timeouts on the node.

The browserTimeout should be:

- Higher than the socket lock timeout (45 seconds)
- Generally higher than values used in webDriver.manage().timeouts(), since this mechanism is a "last line of defe

Configuring the nodes by JSON

java -jar selenium-server-standalone.jar -role node -nodeConfig nodeconfig.json

A sample nodeconfig file can be seen
athttp://code.google.com/p/selenium/source/browse/trunk/java/server/src/org/openqa/g

Configuring the hub by JSON

java -jar selenium-server-standalone.jar -role hub -hubConfig hubconfig.json

A sample hubconfig.json file can be seen
athttp://code.google.com/p/selenium/source/browse/trunk/java/server/src/org/openqa/g

Hub diagnostic messages

Upon detecting anomalious usage patterns, the hub can give the following message:

```
Client requested session XYZ that was terminated due to REASON
```

Reason	Cause/fix
TIMEOUT	The session timed out because the client did not access it within the timeout. If the client has b up
BROWSER_TIMEOUT	The node timed out the browser because it was hanging for too long (parameter browserTimeout)
ORPHAN	A client waiting in queue has given up once it was offered a new session
CLIENT_STOPPED_SESSION	The session was stopped using an ordinary call to stop/quit on the client. Why are you using it ag
CLIENT_GONE	The client process (*your* code) appears to have died or otherwise not responded to our requests, i
FORWARDING_TO_NODE_FAILED	The hub was unable to forward to the node. Out of memory errors/node stability issues or networ
CREATIONFAILED	The node failed to create the browser. This can typically happen when there are environmental directly to track problem.
PROXY_REREGISTRATION	The session has been discarded because the node has re-registered on the grid (in mid-test)

Tips for running with grid

If your tests are running in parallel, make sure that each thread deallocates its web other tests running on other threads. Starting 1 browser per thread at the start of the t at the end is **not** a good idea. (If one test-case decides to consume abnormal amounts the other tests because they're waiting for the slow test. This can be very confusing

Comment by fischer...@gmail.com, Jun 8, 2011

My observation was that the Grid does select a node by doing a case sensitive string comparison. The enum seems to be converted into an uppercase string, so try to register your node by uppercasing the platform in your browser definition. See the following command line arguments as an example:

-browser browserName=internet explorer,version=8,platform=WINDOWS

Hope that helps,

Johannes

Comment by fischer...@gmail.com, Jun 9, 2011

Either make sure that all the dependent libraries are in the classpath or, the probably easier option, Use the standalone jar file instead. That should solve your problem with the ClassNotFoundException?.

Johannes

参考：
http://seleniumhq.org/docs/07_selenium_grid.html

What is Selenium-Grid?

Selenium-Grid allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, Selenium-Grid support distributed test execution. It allows for running your tests in a *distributed test execution* environment.

When to Use It

Generally speaking, there's two reasons why you might want to use Selenium-Grid.

- To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
- To reduce the time it takes for the test suite to complete a test pass.

Selenium-Grid is used to speed up the execution of a test pass by using multiple machines to run tests in parallel. For example, if you have a suite of 100 tests, but you set up Selenium-Grid to support 4 different machines (VMs or separate physical machines) to run those tests, your test suite will complete in (roughly) one-fourth the time as it would if you ran your tests sequentially on a single machine. For large test suites, and long-running test suite such as those performing large amounts of data-validation, this can be a significant time-saver. Some test suites can take hours to run. Another reason to boost the time spent running the suite is to shorten the turnaround time for test results after developers check-in code for the AUT. Increasingly software teams practicing Agile software development want test feedback as immediately as possible as opposed to wait overnight for an overnight test pass.

Selenium-Grid is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time. For example, a 'grid' of virtual machines can be setup with each supporting a different browser that the application to be tested must support. So, machine 1 has Internet Explorer 8, machine 2, Internet Explorer 9, machine 3 the latest Chrome, and machine 4 the latest Firefox. When the test suite is run, Selenium-Grid receives each test-browser combination and assigs each test to run against it's required browser.

In addition, one can have a grid of all the same browser, type and version. For instance, one could have a grid of 4 machines each running 3 instances of Firefox 12, allowing for a 'server-farm' (in a sense) of available Firefox instances. When the suite runs, each test is passed to Selenium-Grid which assigns the test to the next available Firefox instance. In this manner one gets test pass where conceivably 12 tests are all running at the same time in parallel, significantly reducing the time required to complete a test pass.

Selenium-Grid is very flexible. These two examples can be combined to allow multiple instances of each browser type and version. A configuration such as this would provide both, parallel execution for fast test pass completion and support for multiple browser types and versions simultaneously.

Starting Selenium-Grid

Generally you would start a hub first since nodes depend on a hub. This is not abolutely necessary however, since nodes can recognize when a hub has been started and vice-versa. For learning purposes though, it would easier to start the hub first, otherwise you'll see error messages that may not want to start off with your first time using Selenium-Grid.

Starting a Hub

To start a hub with default parameters, run the following command from a command-line shell. This will work on all the supported platforms, Windows Linux, or MacOs.

```
java -jar selenium-server-standalone-2.21.0.jar -hub
```

This starts a hub using default parameter values. We'll explain these parameters in folowing subsections. Note that you will likely have to

change the version number in the jar filename depending on which version of the selenium-server you're using.

Starting a Node

To start a node using default parameters, run the following command from a command-line.

```
java -jar selenium-server-standalone-2.21.0.jar -role node  -hub http://localhost:4444/grid/register
```

This assumes the hub has been started above using default parameters. The default port the hub uses to listen for new requests is port 4444. This is why port 4444 was used in the URL for locating the hub. Also the use of 'localhost' assumes your node is running on the same machine as your hub. For getting started this is probably easiest. If running the hub and node on separate machines, simply replace 'localhost' with the hostname of the remote machine running the hub.

WARNING: Be sure to turn off the firewalls on the machine running your hub and nodes. Otherwise you may get connection errors.

参考资料

Full Scale Automation Using Selenium
http://www.slideshare.net/AndrewDzynia/full-scale-automation-using-selenium

Selenium API:
http://seleniumhq.org/docs/

http://blog.163.com/lingken2@126/blog/static/30373982201261343019181/

WebDriver(JAVA)拾级而上

WebDriver 拾级而上·之零 WebDriver 理论
WebDriver 拾级而上·之一　环境部署
WebDriver 拾级而上·之二　浏览器操作
WebDriver 拾级而上·之三　定位页面元素
WebDriver 拾级而上·之四　操作页面元素
WebDriver 拾级而上·之五　iframe 的处理
WebDriver 拾级而上·之六　获得弹出窗
WebDriver 拾级而上·之七　处理对话框

WebDriver 拾级而上・之八　操作 cookies
WebDriver 拾级而上・之九　等待页面元素加载
WebDriver 拾级而上・之十　封装与重用
WebDriver 拾级而上・之十一　在 selenium2.0 中使用 selenium1.0 的 API
WebDriver 拾级而上・之十二　利用 selenium-webdriver 截图
WebDriver 拾级而上・之十三　调用 Java Script
WebDriver 拾级而上・之十四　RemoteWebDriver
WebDriver 拾级而上・之十五 拖曳动作模拟
WebDriver 拾级而上・之十六 Table 控件的处理
WebDriver 拾级而上・之十七 断言

TestNG 深入

TestNG・一 基础概念
TestNG・二 测试组
TestNG・三 测试方法
TestNG・四 测试方法之工厂
TestNG・五 运行 TestNG
TestNG・六 测试结果
TestNG・七 annotation
TestNG・八 并发测试

WebDriver・TestNg 学以致用

WebDriver & TestNG・参数化及用例排序
WebDriver & TestNG・架构
追求代码质量: 使用 Selenium 和 TestNG 进行编程式测试

JUnit

JUnit4 单元测试

手工测试用例就是自动化测试脚本——使用 ruby 1.9 新特性进行自动化脚本的编写
http://www.cnblogs.com/nbkhic/archive/2011/07/23/2114841.html

让使用 XPath 的 Selenium 命令在 IE 中执行得更快些：
Making Selenium commands using XPaths faster in Internet Explorer
http://www.qaautomation.net/?p=216

Selenium 2 Tutorial and User Guide with examples

1. Introduction to Selenium
2. What is new in Selenium 2
3. Getting started with Selenium 2 and WebDriver
4. Locating page elements
5. Common Assertions using WebDriver
6. Interacting with page elements
7. Waiting after web page interactions
8. Window, frames and Popup dialog handling
9. Mobile automation

TIB 自动化测试工作室
http://www.cnblogs.com/testware
http://www.AutomationQA.com