

Selenium 深入浅出

By 沈东良

<http://blog.csdn.net/shendl/>

2009 年 3 月 8 日

目录

Selenium 深入浅出.....	1
Selenium—Web 界面测试工具.....	2
需要的软件.....	2
Selenium IDE 的使用.....	2
Selenium Java 测试.....	2
Selenium 工作原理.....	5
Java 执行 Selenium 测试.....	5
Selenium 执行内部原理.....	6
我应该使用哪种 Selenium 工具？	7
Selenium Remote Control: 服务器命令行参数.....	8
Selenium 中文参考手册.....	9
Selenium 与 JavaScript.....	18
Selenium 与 EXT.....	18
动态执行 JavaScript 代码的注意点.....	19
Selenium 执行 Ext 的例子:	19

Selenium—Web 界面测试工具

- Selenium 可以使用录制工具录制脚本，测试页面。
- Selenium 可以生成类 html 代码，java 代码，ruby 代码等。
- Selenium 录制工具根据 id 属性定位 html 元素

需要的软件

- 1, 安装 Firefox, Selenium IDE 插件, Firebug 插件。
- 2, 解压 Selenium 的 selenium-remote-control 包，得到一个服务器。
- 3, 获取 selenium-java-client-driver.jar，用于 java 语言的 Selenium 开发。

Selenium IDE 的使用

- 1, FireFox--工具--selinum IDE
- 2, 点击录制按钮，开始记录一个测试。可以使用浏览器进行操作。
最后点击 Selenium IDE 的停止录制按钮。
可以看到 html,Java,Ruby 等形式的脚本语言。
可以执行。
可以根据选择的浏览器，使用选择的浏览器执行脚本中的命令。

SeleniumIDE 除了记录 Html 请求和 AJAX 调用的操作。其他不涉及通讯的操作，也会被录制。其他的自动脚本录制工具，如 WebLoad，LoadRunner 都是只记录 http 请求，不能真正记录对页面的操作。

3, 脚本 可以格式化为多种语言。 这个是使用.js 文件实现的。

4, selenium 使用 xpath 和 css 等选择 html 元素。

Selenium Java 测试

1, java 项目中, 加入 selenium-java-client-driver.jar。

2, 启动 selenium-remote-control 服务器

3, 把 Selenium IDE 录制的 Selenium 脚本转换成 Java 格式, Java 代码复制到 Java 项目中。

```
package com.example.tests;
```

```
import com.thoughtworks.selenium.*;
```

```
import java.util.regex.Pattern;
```

```
public class NewTest extends SeleneseTestCase {
    public void setUp() throws Exception {
        setUp("http://auto.sohu.com/", "*chrome");
    }
    public void testNew() throws Exception {
        selenium.open("/");
        selenium.select("price", "label=10-15 万");
        selenium.select("bodywork", "label=SUV/越野车");
        selenium.select("brand", "label=B 比亚迪");
        selenium.click("//form[@id='SBL']/div/div[2]/a/img");
        selenium.click("link=比亚迪 F0");
    }
}
```

看一个真实的例子:

```
public class ScrComplaintControllerTest extends Demo006ControllerTest
{
    private IProcessHtmlService processHtmlService=new
ProcessHtmlService();
    /*
    * 下面是简单单独的测试
    *
    * */
    @Test
    public void prepareAddCase() {
```

```

        this.browser.open("http://localhost:8080/macodemo/complaint/prepa
reAddCase.page");
        Assert.assertEquals("prepareAddCase出错", "新增個案",
this.browser.getTitle());
    }

```

基类

```

public class Demo006ControllerTest extends SeleniumControllerTest {
    /**
     * @throws java.lang.Exception
     */
    @Before
    public void setUp() throws Exception {
        browser = new DefaultSelenium("localhost", 4444, "*iexplore",
            "http://localhost:8000/macodemo/");
        browser.start();
        SeleniumLoginUtil.managerLogin(browser);
    }

    /**
     * @throws java.lang.Exception
     */
    @After
    public void tearDown() throws Exception {
        this.browser.stop();
    }
}

public abstract class SeleniumControllerTest {
    protected Selenium browser;
    /**
     * @throws java.lang.Exception
     */
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {

    }
    /**
     * @throws java.lang.Exception
     */
    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }
}

```

```
}
```

在每一次 JUnit 测试之前应该执行：

```
Selenium browser= new DefaultSelenium("localhost", 4444, "*iexplore",
    "http://localhost:8000/macodemo/");
    browser.start();
```

结束时执行 `this.browser.stop()`；

Selenium 类用来执行 Selenium 操作，控制浏览器执行 Selenium 指令。

DefaultSelenium 的构造器的参数是你连接的 Selenium 服务器的 ip 地址和端口，第三个参数是用来执行 Selenium 指令的浏览器，最后一个参数是浏览器的起始 URL。

执行简单的 Selenium 指令：

```
this.browser.open("http://localhost:8080/macodemo/complaint/prepareAddCase.page");
```

通过 Selenium 服务器向服务器所在的机器的浏览器发出指令，打开一个 URL。

```
Assert.assertEquals("prepareAddCase 出错 ", " 新增 个案 ",
this.browser.getTitle());
```

返回浏览器页面的 Title 值，判断是否是预期的，如果不是，那么就是出现了错误。

Selenium 工作原理

Selenium 不同于一般的测试工具。一般的脚本测试工具录制脚本，实际上都是通过拦截浏览器收发的 http 请求来实现的。事实上并没有办法录制用户对 html 页面的操作。

当然，对那些执行压力测试的工具来说，这类模拟已经足够。

Selenium 的脚本录制工具是通过监听用户对 html 页面的操作来录制脚本的。Selenium 是真正能够监听用户对 html 页面的操作的录制工具。Selenium 完全了解用户操作的 html 页面。

Selenium 可以生成 7 种语言的脚本：html,java,C#,ruby,python,perl,php。

你可以直接在 SeleniumIDE 中执行 html 格式的脚本。你机器上的 FireFox 将会执行 Selenium 脚本。

Selenium 生成的 Html 格式的脚本，是 Selenium 使用 Html 的语言元素自己开发的一套脚本语言。

如果你要执行其他语言格式的 Selenium 脚本，那么，你需要使用 Selenium 服务器。6 种语言的使用方法都是一样的。下面，我以 Java 为例进行说明。

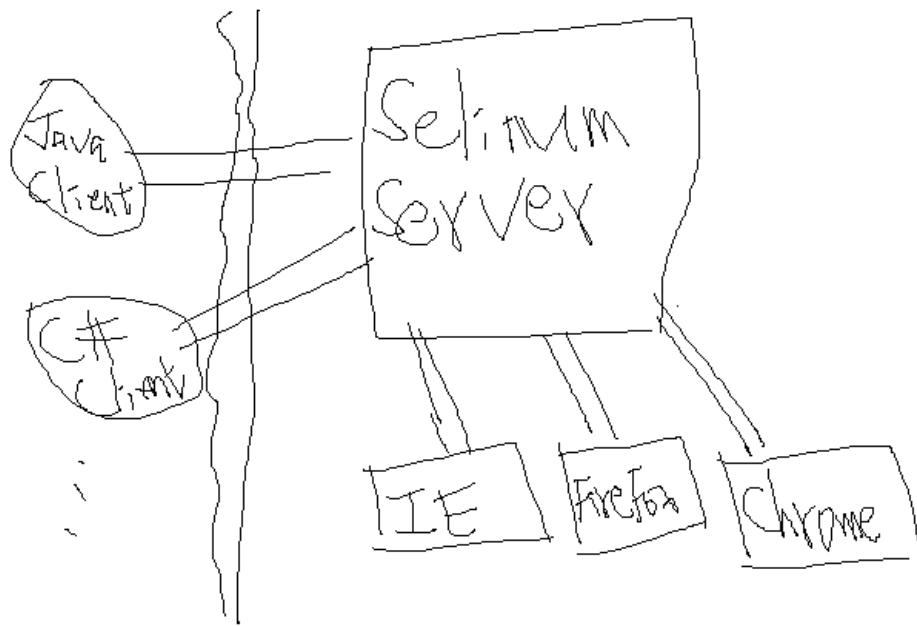
Java 执行 Selenium 测试

首先，需要启动 Selenium 服务器，然后就可以执行 java 版本的 Selenium 测试。

Selenium 服务器将会解释 Java 发来的 Selenium 脚本，并在 Selenium 服务器本地打开浏览器执行客户端发来的 Selenium 脚本，并把结果返回给 Java 客户端。

其他语言执行 Selenium 测试也是如此工作的。

Selenium 的架构:



Java 或者其他客户端，一般作为单元测试程序被执行。它们通过网络与一个或多个 Selenium 服务器进行通讯。

请注意，浏览器是在服务器端打开的，而不是 Java 或者其他客户端打开的。

我们可以把客户端看作是发号施令的主人，Selenium 服务器是执行 Selenium 测试的奴隶。

Selenium 执行内部原理

SeleniumServer 通过网络与 Selenium 客户端通讯，接收 Selenium 测试指令。

SeleniumServer 通过向浏览器发出 JavaScript 调用实现对 Html 页面的全面追踪,并通过网络把执行结果返回给 Selenium 客户端。

Selenium 客户端一般使用单元测试技术实现，通过判断返回的结果与预期是否一致来决定程序是否运行正确。

Selenium 是通过 javascript 来实现对 Html 页面的操作的。它提供了丰富的指定 Html 页面元素和操作页面元素的方法。

Selenium 打开浏览器时，把自己的 JavaScript 文件嵌入网页中。然后 Selenium 的网页通过 frame 嵌入目标网页。这样，就可以使用 Selenium 的 JavaScript 对象来控制目标网页。

Selenium 的 JavaScript 对象中，最重要的就是 Selenium 对象。它的作用是代表 Java 中的 Selenium 接口执行一系列的命令，让浏览器执行。

我应该使用哪种 Selenium 工具？

考虑如下功能矩阵：

	Selenium IDE	Selenium Remote Control	Selenium Core	Selenium Core HTA
浏览器支持	仅 Firefox	很多	所有	仅 IE

需要远程安装	否	否	是	否
--------	---	---	---	---

支持 HTTPS/SSL	是	是*	是	是
---------------------	---	----	---	---

支持跨域	是	是*	否	是
------	---	----	---	---

需要 Java	否	是	否	否
----------------	---	---	---	---

将测试结果保存到磁盘	是	是	否	是
多语言支持	仅 Selenese	很多	仅 Selenese	仅 Selenese

* = Selenium RC 中实验性的支持

浏览器支持：Selenium IDE 仅可以在 Firefox 中工作。Selenium Remote Control 直接支持 Firefox 和 IE，此外还支持手动配置很多其它浏览器。Selenium Core 几乎可以在任何浏览器中工作，因为它是纯粹的 JavaScript。Selenium Core HTA 可以在 IE 最高安全等级（特权）下工作，这意味着它仅能在 IE 下工作。

需要远程安装：Selenium Core 需要在被测试应用程序（Application Under Test, AUT）的网站内安装，因为 [同源安全策略](#)，一个阻止进行跨网站脚本访问的安全策略。那意味着你不能坐在那里写一个 Selenium Core 测试而运行在 google.com；如果要那样做，你需要在 google.com 安装 Selenium Core。如果你不能/不会在你的被测应用程序 AUT 上面安装 Selenium Core，不能保证它在 AUT 相同的服务器上公开可见，那么 Selenium Core 可能不适合你。

同源安全策略不会限制 Selenium IDE，因为它作为 Firefox 扩展实现；它对于 Selenium Core HTA 也无效，因为它运行于 IE 的最高安全级别（特权）上。Selenium RC 通过提供一个代理服务器来保证 Selenium JS 文

件看似来自相同的远程服务器，从而符合同源策略；代理服务器欺骗浏览器，让它认为这里的确有像 <http://www.google.com/selenium/> 这样的目录。

支持 HTTPS/SSL: 最新版本的 Selenium Remote Control 的代理服务器可以支持 HTTPS 网站（当前所指版本为 0.9.0），但是这个支持还是实验性的。参考[使用试验性的浏览器加载器](#)

支持跨域: 同源策略意味着测试一个服务器/域不能操作另外一个服务器/域。这意味着 Selenium Core 不能处理跨多个不同域的应用程序。

Selenium RC 可以处理切换域的问题，但是这个支持是实验性的。参考[使用试验性的浏览器加载器](#)获取更多信息。

需要 Java: Selenium Core 和 Selenium IDE 直接在浏览器中运行。Selenium RC 需要安装 Java（运行代理服务器）。注意，虽然 Selenium RC 需要 Java，你也可以用 .NET、Perl、Python 和 Ruby 编写你的 RC 测试，但是你需要 Java 来运行代理。

将测试结果保存到磁盘: Selenium Core 不能将任何测试结果写到磁盘上（因为它用 javascript 写的，它不允许向磁盘写数据），当然你可以将测试结果发送到另外一台服务器保存。（你可以通过 Selenium Core HTA 保存测试结果。）Selenium Remote Control 提明确供支持运行测试并用多种语言讲测试结果写入磁盘的功能；它还可以为 Selenium Core 结果处理服务器。Selenium IDE 是一个 Firefox 扩展，所以当然支持将测试结果存盘。

语言支持: Selenium Remote Control 允许你用任何语言写测试，包括 Java、.NET、Perl、Python 和 Ruby。（你还可以为其它语言添加测试的支持。）Selenium IDE 和 Selenium Core 仅支持使用"Selenese"测试，一种简单的脚本语言。Selenese 有一些严格的限制：它没有条件（没有"if"表达式），并且它没有循环（没有"For"表达式）。这会使编写复杂的测试变得困难甚至不可能。

使用 SeleniumServer 执行 Selenium 测试，是功能最强的，建议使用 Unit 单元测试和 SeleniumServer 这种方式执行测试。

Selenium Remote Control: 服务器命令行参数

使用示例: `java -jar selenium-server.jar [-interactive] [options]`

- **-port <nnnn>:** selenium 服务器使用的端口号（默认 4444）
- **-timeout <nnnn>:** 我们放弃前（超时）所等待的秒数
- **-interactive:** 进入交互模式。参考[教程](#)获取更多信息
- **-multiWindow:** 进入被测试网站都在单独窗口打开的模式，并且 selenium 支持 frame
- **-forcedBrowserMode <browser>:** 设置浏览器模式（例如，所有的会话都使用 "*iexplore"，不管给 `getNewBrowserSession` 传递什么参数）
- **-userExtensions <file>:** 指定一个被载入到 selenium 的 JavaScript 文件
- **-browserSessionReuse:** 停止在测试间重新初始化和替换浏览器。
- **-alwaysProxy:** 默认情况下，我们尽量少的进行代理；设置这个标志将会强制所有的浏览器通讯都通过代理
- **-firefoxProfileTemplate <dir>:** 一般情况，我们在每次启动之前都生成一个干净的 Firefox 设置。您可以指定一个目录来让我们将您的设置拷贝过来，代替我们生成的。

- **-debug:** 进入 debug 模式，会有更多的跟踪调试信息
- **-htmlSuite <browser> <startURL> <suiteFile> <resultFile>:** 使用指定的浏览器（例如"*firefox"）在指定的 URL（例如"http://www.google.com"），运行一个单独的 HTML Selenese (Selenium Core)测试套件后立即退出。您需要指定 HTML 测试套件的绝对路径还有我们将会生成的 HTML 测试结果文件的路径。
- **-proxyInjectionMode:** 进入代理注入模式，这个模式中 selenium 服务器作为进入测试程序的所有内容的代理服务器。在这个模式下，可以跨多个域访问，并且还支持如下附加参数：
 - **-dontInjectRegex <regex>:** 附加的正则表达式，代理注入模式能够使用它决定是否进行注入
 - **-userJsInjection <file>:** 指定一个 JavaScript 文件，将它注入到所有页面中
 - **-userContentTransformation <regex> <replacement>:** 一个正则表达式，对所有被测 HTML 内容进行匹配；第二个 string 将会对替换所有匹配的内容。这个标志能够使用多次。一个简单的适合使用这个参数的例子：如果你添加"-userContentTransformation https http"那么测试应用程序的 HTML 中的所有"https"字符串都会被替换为"http"。

We also support two Java system properties: 我们还支持两种 Java 系统属性：**-Dhttp.proxyHost** 和 **-Dhttp.proxyPort**。使用 Selenium 服务器作为代理服务器，Selenium RC 一般重载你的代理服务器配置。使用这个参数适合在使用 Selenium 服务器代理的同时使用你自己的代理服务器。使用代理服务器时这样配置：

```
java -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=1234 -jar selenium-server.jar
```

如果你的 HTTP 代理服务器需要验证，你还可以在 http.proxyHost 和 http.proxyPort 后面设置 **-Dhttp.proxyUser** 和 **-Dhttp.proxyPassword**。

```
java -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=1234 -Dhttp.proxyUser=joe -Dhttp.proxyPassword=example -jar selenium-server.jar
```

Selenium 中文参考手册

（本部分由江南白衣翻译）

一、Commands (命令)

- **Action**
对当前状态进行操作
失败时，停止测试
- **Assertion**
校验是否有产生正确的值
- **Element Locators**
指定 HTML 中的某元素
- **Patterns**
用于模式匹配

1. Element Locators (元素定位器)

- **id=id**
id locator 指定 HTML 中的唯一 id 的元素
- **name=name**
name locator 指定 HTML 中相同 name 的元素中的第一个元素
- **identifier=id**
identifier 标识符 locator 首先查找 HTML 是否存在该 id 的元素, 若不存在, 查找第一个该 name 的元素
- **dom=javascriptExpression**
dom locator 用 JavaScript 表达式来定位 HTML 中的元素, 注意必须要以 "document" 开头
例如:
`dom=document.forms['myForm'].myDropdown`
`dom=document.images[56]`
- **xpath=xpathExpression**
xpath locator 用 XPath 表达式来定位 HTML 中的元素, 必须注意要以 "/" 开头
例如:
`xpath=//img[@alt='The image alt text']`
`xpath=//table[@id='table1']//tr[4]/td[2]`
- **link=textPattern**
link locator 用 link 来选择 HTML 中的连接或锚元素
例如:
`link=The link text`
- 在没有 locator 前序的情况下:
如果以 "document." 开头, 则默认是使用 dom locator, 如果是以 "/" 开头, 则默认使用 xpath locator, 其余情况均认作 identifier locator

2. String Matching Patterns (字符串匹配模式)

- **glob:pattern**
glob 模式, 用通配符 "*" 代表任意长度字符, "?" 代表一个字符
- **regexp:regexp**
正则表达式模式, 用 JavaScript 正则表达式的形式匹配字符串
- **exact:string**
精确匹配模式, 精确匹配整个字符串, 不能用通配符
- 在没有指定字符串匹配前序的时候, selenium 默认使用 glob 匹配模式

3. Select Option Specifiers (Select 选项指定器)

- **label=labelPattern**
通过匹配选项中的文本指定选项
例如: `label=regexp:^(Oo)ther`
- **value=valuePattern**
通过匹配选项中的值指定选项
例如: `value=other`

- **id=id**
通过匹配选项的 id 指定选项
例如: id=option1
- **index=index**
通过匹配选项的序号指定选项, 序号从 0 开始
例如: index=2
- 在没有选项选择前序的情况下, 默认是匹配选项的标签

二、Actions


描述了用户所会作出的操作。

Action 有两种形式: action 和 actionAndWait, action 会立即执行, 而 actionAndWait 会假设需要较长时间才能得到该 action 的响应, 而作出等待, open 则是会自动处理等待时间, 是同步方法。

- **click**
click(elementLocator)
 - 点击连接, 按钮, 复选和单选框
 - 如果点击后需要等待响应, 则用 "clickAndWait"
 - 如果是需要经过 JavaScript 的 alert 或 confirm 对话框后才能继续操作, 则需要调用 verify 或 assert 来告诉 Selenium 你期望对对话框进行什么操作。

click	aCheckbox	
clickAndWait	submitButton	
clickAndWait	anyLink	

- **open**
open(url)
 - 在浏览器中打开 URL, 可以接受相对和绝对路径两种形式
 - 注意: 该 URL 必须与浏览器相同的安全限定范围之内

open	/mypage	
open	http://localhost/ 	

- **type**
type(inputLocator, value)
 - 模拟人手的输入过程, 往指定的 input 中输入值
 - 也适合给复选和单选框赋值
 - 在这个例子中, 则只是给勾选了的复选框赋值, 注意, 而不是改写其文本

type	nameField	John Smith
------	-----------	------------

typeAndWait	textBoxThatSubmitsOnChange	newValue
-------------	----------------------------	----------

- **select**

select(dropDownLocator, optionSpecifier)

- 根据 optionSpecifier 选项选择器来选择一个下拉菜单选项
- 如果有多于一个选择器的时候，如在用通配符模式，如"f*b*"，或者超过一个选项有相同的文本或值，则会选择第一个匹配到的值

select	dropDown	Australian Dollars
select	dropDown	index=0
selectAndWait	currencySelector	value=AUD
selectAndWait	currencySelector	label=Auslian D*rs

- **goBack,close**

goBack()

模拟点击浏览器的后退按钮

close()

模拟点击浏览器关闭按钮

- **selectWindow**

select(windowId)

- 选择一个弹出窗口
- 当选中那个窗口的时候，所有的命令将会转移到那窗口中执行

selectWindow	myPopupWindow	
selectWindow	null	

- **pause**

pause(millisecnds)

- 根据指定时间暂停 Selenium 脚本执行
- 常用在调试脚本或等待服务器段响应时

pause	5000	
pause	2000	

可以使用 Thread.sleep(2000);实现相同的功能。

- **fireEvent**

fireEvent(elementLocatore,evenName)

模拟页面元素事件被激活的处理动作

fireEvent	textField	focus
fireEvent	dropDown	blur

- **waitForCondition**

waitForCondition(JavascriptSnippet,time)

- 在限定时间内，等待一段 JavaScript 代码返回 true 值，超时则停止等待

waitForCondition	var value=selenium.getText("foo"); value.match(/bar/);	3000
------------------	--	------

- **waitForValue**

waitForValue(inputLocator, value)

- 等待某 input(如 hidden input)被赋予某值。

- 会轮流检测该值，所以要注意如果该值长时间一直不赋予该 input 该值的话，可能会导致阻塞

waitForValue	finishIndication	isfinished

- **store,storeValue**

store(valueToStore, variablename)

保存一个值到变量里。

该值可以由自其他变量组合而成或通过 JavaScript 表达式赋值给变量

store	Mr John Smith	fullname
store	\$. {title} \$. {firstname} \$. {surname}	fullname
store	javascript. {Math.round(Math.PI*100)/100}	PI
storeValue	inputLocator	variableName

- 把指定的 input 中的值保存到变量中

storeValue	userName	userID
type	userName	\$. {userID}

- **storeText, storeAttribute**

storeText(elementLocator, variablename)

把指定元素的文本值赋予给变量

storeText	currentDate	expectedStartDate
-----------	-------------	-------------------

verifyValue	startDate	\$. { expectedStartDate }
-------------	-----------	---------------------------

- **storeAttribute(. { } elementLocator@attributeName,variableName. { })**

把指定元素的属性的值赋予给变量

storeAttribute	input1@class	classOfInput1
verifyAttribute	input2@class	\$. { classOfInput1 }

- **chooseCancel., answer..**

chooseCancelOnNextConfirmation()

- 当下次 JavaScript 弹出 confirm 对话框的时候,让 selenium 选择 Cancel
- 如果没有该命令时，遇到 confirm 对话框 Selenium 默认返回 true，如手动选择 OK 按钮一样

chooseCancelOnNextConfirmation		
--------------------------------	--	--

- - 如果已经运行过该命令，当又一次又有 confirm 对话框出现时，也会同样地再次选择 Cancel

answerOnNextPrompt(answerString)

- 在下一次 JavaScript 弹出 prompt 提示框时，赋予其 answerString 的值，并选择确定

answerOnNextPrompt	Kangaroo	
--------------------	----------	--

三、Assertions

允许用户去检查当前状态。两种模式：Assert 和 Verify，当 Assert 失败，则退出测试；当 Verify 失败，测试会继续运行。

- **assertLocation, assertTitle**

assertLocation(relativeLocation)

判断当前是在正确的页面

verifyLocation	/mypage	
assertLocation	/mypage	

- **assertTitle(titlePattern)**

检查当前页面的 title 是否正确

verifyTitle	My Page	
assertTitle	My Page	

- **assertValue**

assertValue(inputLocator, valuePattern)

- 检查 input 的值
- 对于 checkbox 或 radio，如果已选择，则值为"on",反之为"off"

verifyValue	nameField	John Smith
assertValue	document.forms[2].nameField	John Smith

- **assertSelected, assertSelectedOptions**

assertSelected(selectLocator, optionSpecifier)

检查 select 的下拉菜单中选中的选项是否和 optionSpecifer(Select 选择选项器)的选项相同

verifySelected	dropdown2	John Smith
verifySelected	dorpdwn2	value=js*123
assertSelected	document.forms[2].dropDown	label=J*Smith
assertSelected	document.forms[2].dropDown	index=0

- **assertSelectOptions(selectLocator, optionLabelList)**

- 检查下拉菜单中的选项的文本是否和 optionLabelList 相同
- optionLabelList 是以逗号分割的一个字符串

verifySelectOptions	dropdown2	John Smith,Dave Bird
assertSelectOptions	document.forms[2].dropdown	Smith,J,Bird,D

- **assertText**

assertText(elementLocator,textPattern)

- 检查指定元素的文本
- 只对有包含文本的元素生效
- 对于 Mozilla 类型的浏览器，用 textContent 取元素的文本，对于 IE 类型的浏览器，用 innerText 取元素文本

verifyText	statusMessage	Successful
assertText	//div[@id='foo']//h1	Successful

- **assertTextPresent, assertAttribute**

assertTextPresent(text)

检查在当前给用户显示的页面上是否有出现指定的文本

verifyTextPresent	You are now logged in	
-------------------	-----------------------	--

assertTextPresent	You are now logged in	
-------------------	-----------------------	--

- **assertAttribute(. { } elementLocator@attributeName. { } , ValuePattern)**

检查当前指定元素的属性的值

verifyAttribute	txt1@class	bigAndBlod
assertAttribute	document.images[0]@alt	alt-text
verifyAttribute	//img[@id='foo']/alt	alt-text

- **assertTextPresent, etc.**

assertTextPresent(text)

assertTextNotPresent(text)

assertElementPresent(elementLocator)

verifyElementPresent	submitButton	
assertElementPresent	//img[@alt='foo']	assertElementNotPresent(elementLocator)

- **assertTable**

assertTable(cellAddress, valuePattern)

- 检查 table 里的某个 cell 中的值

- cellAddress 的语法是 tableName.row.column, 注意行列序号都是从 0 开始

verifyTable	myTable.1.6	Submitted
assertTable	results0.2	13

- **assertVisible, nonVisible**

assertVisible(elementLocator)

- 检查指定的元素是否可视的

- 隐藏一个元素可以用设置 css 的'visibility'属性为'hidden', 也可以设置'display'属性为'none'

verifyVisible	postcode	
assertVisible	postcode	

- **assertNotVisible(elementLocator)**

verfyNotVisible	postcode	
------------------------	-----------------	--

assertNotVisible	postcode	

- **Editable, non-editable**

assertEditable(inputLocator)

检查指定的 input 是否可以编辑

verifyEditable	shape	
assertEditable	colour	

- **assertNotEditable(inputLocator)**

检查指定的 input 是否不可以编辑

- **assertAlert**

assertAlert(messagePattern)

- 检查 JavaScript 是否有产生带指定 message 的 alert 对话框

- alert 产生的顺序必须与检查的顺序一致

- 检查 alert 时会产生与手动点击'OK'按钮一样的效果。如果一个 alert 产生了，而你却没有去检查它，selenium 会在下个 action 中报错。

- 注意：Selenium 不支持 JavaScript 在 onload()事件时 调用 alert();在这种情况下，Selenium 需要你自己手动来点击 OK。

- **assertConfirmation**

assertConfirmation(messagePattern)

- 检查 JavaScript 是否有产生带指定 message 的 confirmation 对话框和 alert 情况一样，confirmation 对话框也必须在它们产生的时候进行检查

- 默认情况下，Selenium 会让 confirm() 返回 true，相当于手动点击 Ok 按钮的效果。你能够通过

chooseCancelOnNextConfirmation 命令让 confirm()返回 false.同样地，如果一个 cofirmation 对话框出现了，但你却没有检查的话，Selenium 将会在下个 action 中报错

- 注意：在 Selenium 的环境下，confirmation 对话框框将不会再出现弹出显式对话框

- 注意：Selenium 不支持在 onload()事件时调用 confirmation 对话框，在这种情况下，会出现显示 confirmatioin 对话框，并需要你自己手动点击。

- **assertPrompt**

assertPrompt(messagePattern)

- 检查 JavaScript 是否有产生带指定 message 的 Prompt 对话框

- 你检查的 prompt 的顺序 Prompt 对话框产生的顺序必须相同

- 必须在 verifyPrompt 之前调用 answerOnNextPrompt 命令

- 如果 prompt 对话框出现了但你却没有检查，则 Selenium 会在下个 action 中报错

answerOnNextPrompt	Joe	
click	id=delegate	

verifyPrompt	Delegate to who?	
--------------	------------------	--

四、Parameters and Variables

参数和变量的声明范围由简单的赋值到 JavaScript 表达式赋值。

Store, storeValue 和 storeText 为下次访问保存值。

在 Selenium 内部是用一个叫 storeVars 的 map 来保存变量名。

- **Variable Substitution 变量替换**

提供了一个简单的方法去访问变量,语法 \$. {xxx}

store	Mr	title
storeValue	nameField	surname
store	\$. {title} \$. {surname}	fullname
type	textElement	Full name is: \$. {fullname}

- **JavaScript Evaluation JavaScript 赋值**

你能用 JavaScript 来构建任何你所需要的值。

这个参数是以 javascript 开头,语法是 javascript. {'with a trailing'}。

可以通过 JavaScript 表达式给某元素赋值。

store	javascript. {'merchant'+(new Date()).getTime() }	merchantId
type	textElement	javascript. { storedVars['merchantId'].toUpperCase() }

- **Generating Unique values 产生唯一值。**

问题: 你需要唯一的用户名

解决办法: 基于时间来产生用户名, 如'fred'+(new Date()).getTime())

Selenium 与 JavaScript

Selenium 与 EXT

- EXT 使用 JS 动态生成 Html 和 CSS 代码。

- 如果未给 EXT 元素指定 id，会使用自增 id。无法确定 id 的值。
- 如果 EXT 的界面元素发生变化，id 也会发生变化。
- Selenium 录制工具因此无法定位 EXT 生成的 Web 界面
根本原因就是 Selenium 需要定位 html 元素，而 Ext 的很多 html 元素都是 Ext 生成的。
而且生成的元素的 id 都是 Ext 自动创建的。

解决方法：

- Ext 为每一个组件分配 ID
- 但是有一些 Html 元素无法指定 ID
- Selenium 录制工具辅助手工编写测试代码
- Selenium 可以使用 JS 代码，操作用户界面的 EXT 对象。
可以使用 Selenium 接口的 `getEval` 方法在浏览器中执行 JS 代码。

Selenium 接口

`java.lang.String getEval(java.lang.String script)`

方法，动态执行 JavaScript 代码。

storeEval (script, variableName)

Arguments:

- script - the JavaScript snippet to run
- variableName - the name of a [variable](#) in which the result is to be stored.

Returns:

the results of evaluating the snippet

结果存储在 JS 的变量名中。

动态执行 JavaScript 代码的注意事项

通过调用 Selenium 接口的 `getEval()` 等方法可以直接把 Java 中的 JavaScript 代码发送到浏览器中执行。

但是，这里实际上执行的并不是标准的 JavaScript 的 `eval()` 方法。`getEval()` 方法中的 JavaScript 代码的 `this` 是 JavaScript 中的 Selenium 对象，而不是 `window` 对象。

如果你需要调用当前页面的 `window` 对象的方法，你需要直接指出 `window` 对象，如：
“`window.document.getElementById('foo');`”。

`getEval()` 方法中的 JavaScript 代码的 `this` 是 JavaScript 中的 Selenium 对象。请注意这不是 Java 中的 Selenium 接口，而是 JavaScript 中的 Selenium 对象。JavaScript 中的 Selenium 对象的作用是代表 Java 中的 Selenium 接口执行一系列的命令，让浏览器执行。

另外，Selenium 接口的 `selectFrame` 和 `selectWindow` 方法可以修改当前 Selenium 命令的目标窗口。也就是说，当前的 `window` 对象会改变。

Selenium 执行 Ext 的例子:

protected Selenium browser;

.....

```
this.browser.getEval("window.frames['1000'].window.Ext.getCmp('testId-09').setValue('应严办! ');");
```

说明:

Window 是当前 Selenium 命令的目标窗口，这个当前窗口可以通过 Selenium 接口的 selectFrame 和 selectWindow 方法修改！

Selenium 动态执行 JavaScript 的例子:

也可以使用这种方法，在 Java 端注入 javascript 代码到页面中动态执行，如:

```
rowNumStr = this.browser
    .getEval("(function() {"
        + "var tableLength=0;"
        + " var
tables=window.frames['iframe_00300000000000000003'].window.document.g
etElementsByTagName('table'); "
        + "for(var j=0;j<tables.length;j++){ "
        + "  if(tables[j].className=='x-grid3-row-
table'){ "
        + "    tableLength++; "
        + "
if(tables[j].childNodes[0].childNodes[0].childNodes[1].childNodes[0].
innerText==' "
        + complainIdValue + " ' ) { "
        + "    return tableLength;" + "    }" + "  }"
        + "}" + "return -1;" + "})();");
```