Selenium WebDriver 菜谱

◆ 使用 findElement 方法定位元素

selenium WebDriver定位元素是通过使用`findElement()`和`findElements()`方法。

`findElement()`方法返回一个基于指定查寻条件的`WebElement`对象或是抛出一个没有找到符合条件元素的异常。
查询方法会将`By`实例作为参数传入。

By.id　通过元素id属性定位元素

By.name　通过元素name属性定位元素

By.classname 通过classname属性定位元素

By.tagname 通过tagname属性定位元素

By.linktext 通过文本定位链接

By.partiallinktext 通过部分文本定位链接

By.cssselector 通过CSS定位元素

By.xpath 通过xpath定位元素

先定位父级元素，再定位具体子元素：

`WebElement`类也可以支持查询子类元素。例如，假设页面上有一些重复的元素。但是，他们在不同的`<div>`中。可以先定位到其父元素`<div>`然后在定位其子元素，方法如下：

```
WebElement div = driver.findElement(By.id("div1"));
WebElement topLink = div.findElement(By.linkText("top"));
```

你也可以将他们缩写成一行：

```
WebElement topLink = driver.findElement
(By.id("div1")).findElement(By.linkText("top"));
```

对于findElement()方法：

当开始寻找符合指定条件的元素时，它将会查询整个DOM，然后返回第一个找到的匹配的元素。

◆ 使用 findElements 方法来定位元素

findElements()方法会返回匹配指定查询条件的WebElements的集合（即：可以得到匹配指定规则的集合）。如果没有找到则返回为空。

JAVA实例代码：

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2
{
@Test
public void test()
{
WebDriver driver = new InternetExplorerDriver();
driver.get("http://www.baidu.com");
List<WebElement> links = driver.findElements(By.cssSelector("#nv a"));
//验证链接数量
assertEquals(10, links.size());
//打印href属性
for (int i = 0; i < links.size(); i++)
{
System.out.println(links.get(i).getAttribute("href"));
}
driver.close();
}
}
```

◆ 定位链接

```
WebElement gmailLink = driver.findElement(By.linkText("GMail"));
assertEquals("http://mail.google.com/",gmailLink.getAttribute("hr
ef"));
```

通过部分链接名定位链接

```
WebElement inboxLink =
driver.findElement(By.partialLinkText("Inbox"));
System.out.println(inboxLink.getText());
```

◆ 通过标签名称定位元素

```
WebElement table = driver.findElement(By.id("summaryTable"));
List<WebElement> rows = table.findElements(By.tagName("tr"));
assertEquals(10, rows.size());
```

◆ 使用 CSS 选择器定位元素

使用绝对路径定位元素：

```
WebElement userName = driver.findElement(By.cssSelector("html body   div div
form input"));
```

使用相对路径定位元素：

当我们使用CSS选择器来查找元素的时候，我们可以使用class属性来定位元素。我们可以先指定一个HTML的标签，然后加一个“.”符号，跟上class属性的值，方法如下：

```
WebElement loginButton =
driver.findElement(By.cssSelector("input.login"));
```

ID选择器定位元素：

先指定一个HTML标签，然后加上一个“#”符号，跟上id的属性值，如下所示：

```
WebElement userName =
driver.findElement(By.cssSelector("input#username"));
```

使用其他属性的选择器定位元素：

使用name选择器：

```
WebElement userName =
driver.findElement(By.cssSelector("input[name=username]"));
```

使用alt选择器：

```
WebElement previousButton =
driver.findElement(By.cssSelector("img[alt='Previous']"));
```

使用多个属性选择器定位元素：

```
WebElement previousButton =

driver.findElement(By.cssSelector("input[type='submit'][value='Login']"));
```

◆ 使用 XPath 定位元素

使用绝对路径：

```
WebElement userName =
driver.findElement(By.xpath("html/body/div/div/form/input"));
```

使用相对路径：

处于DOM中第一个<input>元素：

```
WebElement userName = driver.findElement(By.xpath("//input"));
```

使用索引定位DOM中的第二个<input>元素：

WebElement passwd=driver.findElement(By.xpath("//input[2]"));

使用Xpath和属性值定位元素：

```
WebElement previousButton = driver.findElement
(By.xpath("//input[@type='submit'and @value='Login']"));
```

◆ 使用 jQuery 选择器

以百度首页为例，百度首页没有jQuery库。我们想定位百度导航栏上面的所有超链接元素，并输出结果。

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2 {
```

WebDriver driver = new InternetExplorerDriver();

JavascriptExecutor jse = (JavascriptExecutor)driver;

@Test

public void jQueryTest() {

```
driver.get("http://www.baidu.com/");
```

injectjQueryIfNeeded();

List<WebElement> elements =

```
(List<WebElement>)jse.executeScript
("return jQuery.find('#nv a')");
assertEquals(7,elements.size()); //验证超链接的数量
for (int i = 0; i < elements.size(); i++) {
System.out.print(elements.get(i).getText() + "、");
}
driver.close();
}
private void injectjQueryIfNeeded() {
if (!jQueryLoaded())
injectjQuery();
}
//判断是已加载jQuery
public Boolean jQueryLoaded() {
Boolean loaded;
try {
loaded = (Boolean)jse.executeScript("return " +
"jQuery()!=null");
} catch (WebDriverException e) {
loaded = false;
}
return loaded;
}
//通过注入jQuery
public void injectjQuery() {
jse.executeScript(" var headID = "
+"document.getElementsByTagName(\"head\")[0];"
+ "var newScript = document.createElement('script');"
+ "newScript.type = 'text/javascript';"
```

```
+ "newScript.src = "
+"'http://ajax.googleapis.com/ajax/"
+"libs/jquery/1.7.2/jquery.min.js';"
+ "headID.appendChild(newScript);");
}
}
```

injectjQueryIfNeeded()方法首先通过jQueryLoaded()方法来判断网页中是否加有jQuery对象。如果没有，再调用injectjQuery()方法通过增加一个<Script>元素来实时加载jQuery库，参考的是Google在线库，可修改例子中的版本，使用最新的jQuery版本。

◆ 定位表格的行和列

表格相关的页面元素

表格	描述
<table>	定义表格
<caption>	定义表格标题。
<th>	定义表格的表头。
<tr>	定义表格的行。
<td>	定义表格单元。
<thead>	定义表格的页眉。
<tbody>	定义表格的主体。
<tfoot>	定义表格的页脚。
<col>	定义用于表格列的属性。
<colgroup>	定义表格列的组。

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2
```

```
{
WebDriver driver = new InternetExplorerDriver();
JavascriptExecutor jse = (JavascriptExecutor)driver;
@Test
public void tableTest()
    {
driver.get
("http://www.w3school.com.cn/html/html_tables.asp");
//首先得到所有tr的集合
List<WebElement> rows =
driver.findElements(By.cssSelector(".dataintable tr"));
//验证表格的行数
assertEquals(11,rows.size());
//打印出所有单元格的数据
for (WebElement row : rows)
    {
//得到当前tr里td的集合
List<WebElement> cols =
row.findElements(By.tagName("td"));
for (WebElement col : cols)
    {
System.out.print(col.getText());//得到td里的文本
    }
System.out.println();
    }
driver.close();
    }

}
```

◆ 检查元素的文本

```
@Test
public void testElementText()
{
//取得元素
WebElement message = driver.findElement(By.id("message"));
//得到元素文本
String messageText = message.getText();
//验证文本为"Click on me and mycolor will change"
```

```
assertEquals("Click on me and my color will change",  messageText);
//获得area元素
WebElement area = driver.findElement(By.id("area"));
//验证文本为"Div's Text\nSpan's Text"
assertEquals("Div's Text\nSpan's Text",area.getText());
}
```

WebElement中的getText()方法返回元素的innerText属性。所以元素里面如果有子节点一样也会被反回出来

也可以使用JavaString API方法如contains()，startsWith()，endsWith()来进行部分匹配。方法如下：

```
assertTrue(messageText.contains("color"));
assertTrue(messageText.startsWith("Click on"));
assertTrue(messageText.endsWith("will change"));
```

◆ 检查元素的属性值

此处需要使用getAttribute()方法来检查元素的属性。
创建一个测试，定位元素再检查它的属性，方法如下：

```
@Test
public void testElementAttribute()
{
WebElement message = driver.findElement(By.id("message"));
assertEquals("justify",message.getAttribute("align"));
}
```

此例子中验证了元素的属性align的值是否为justify。

◆ 检查元素的 CSS 属性值

让我们创建一个测试，读取元素的CSS的width属性并验证它的值。

```
@Test
public void testElementStyle()
{
```

```
WebElement message = driver.findElement(By.id("message"));
String width = message.getCssValue("width");
assertEquals("150px",width);
}
```

◆ 针对鼠标和键盘事件使用高级的用户交互 API

Selenium WebDriver高级用户交互API允许我们通过使用Actions类执行从键盘事件到简单或复杂的鼠标事件，如拖拽操作，按住一个按键然后执行鼠标操作，创建一个复杂的事件链就像用户真正的在手动操作一样。

我们创建一个测试使用Ctrl按键来选择表格的多个行。我们可以先选择第一行，然后按住ctrl键，再选择另一行后释放ctrl键。这样就可以选择所需要的行。

```
@Test
public void testRowSelectionUsingControlKey()
{
List<WebElement> tableRows = driver.findElements
(By.xpath("//table[@class='iceDatTbl']/tbody/tr"));
//Select second and fourth row from table using Control Key.
//Row Index start at 0
Actions builder = new Actions(driver);
builder.click(tableRows.get(1)).keyDown(Keys.CONTROL).click(table
Rows.get(3)).keyUp(Keys.CONTROL).build().perform();


//Verify Selected Row table shows two rows selected

List<WebElement> rows = driver.findElements

(By.xpath("//div[@class='icePnlGrp

exampleBox']/table[@class='iceDatTbl']/tbody/tr"));

assertEquals(2,rows.size());

}
```

首先创建一个Actions的实例，再调用相应的事件方法，然后调用build()方法，建立这么一组操作方法链，最后调用perform()来执行。

◆ 在元素上执行双击操作

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
public class Selenium2 {
WebDriver driver = new FirefoxDriver();
@Test
public void actionsTest() {
driver.get("D:\\demo\\DoubleClickDemo.html");
WebElement message = driver.findElement(By.id("message"));
// 验证初始字体为14px
assertEquals("14px", message.getCssValue("font-size"));
Actions builder = new Actions(driver);
builder.doubleClick(message).build().perform();
// 验证点击后字体变为20px
assertEquals("20px", message.getCssValue("font-size"));
driver.close();
}
}
```

当鼠标双击的时候触发了字体变化的事件，我们可以使用doubleClick()来模拟真实的双击。

◆ 执行拖拽操作

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.interactions.Actions;
public class Selenium2 {
@Test
public void testDragDrop() {
WebDriver driver = new InternetExplorerDriver();
```

```
driver.get("D:\\demo\\DragAndDrop.html");
WebElement source = driver.findElement(By.id("draggable"));
WebElement target = driver.findElement(By.id("droppable"));
Actions builder = new Actions(driver);
builder.dragAndDrop(source, target).perform();
try {
assertEquals("Dropped!", target.getText());
} catch (Error e) {
e.printStackTrace();
}finally{
driver.close();
}
}
}
```

拖拽一个元素到另一个元素再放下，我们需要先定位返些元素（原元素，目标元素）然后作为参数传给dragAndDrop()。

◆ 执行 JavaScript 代码

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2 {
@Test
public void testJavaScriptCalls() {
WebDriver driver = new InternetExplorerDriver();
driver.get("http://www.baidu.com");
JavascriptExecutor js = (JavascriptExecutor) driver;
String title = (String) js.executeScript("return document.title");
assertEquals("百度一下，你就知道", title);
long links = (Long) js.executeScript("var links = "
+ "document.getElementsByTagName('A'); "
+ "return links.length");
assertEquals(26, links);
driver.close();
}
}
```

从javaScript代码中返回数据，我们需要使用return关键字。基于返回值乩类型，我们需要对executeScript()方法迕行转型。对于带小数点乩值，使用Double类型，非小数值可以使用Long类型，布尔值可以使用Boolean类型，如果返回乩是HTML节点，可以使用 WebElement类型，文本值，可以使用String类型。如果返回乩是对象列表，基于对象类型乩伕何值都可以。

◆ 使用 Selenium WebDriver 进行截图

Selenium WebDriver提供了TakesScreenshot接口来捕捉网页的全屏。返可以在测试执行遇到异常错误时候将屏幕戛支下来，可以知道弼时収生了什么。我们也可以在验证元素状态，显示的值是某个操作完成后的状态迕行截屏。

```
package com.example.tests;
import java.io.File;
import org.apache.commons.io.FileUtils;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2 {
@Test
public void testTakesScreenshot() {
WebDriver driver = new InternetExplorerDriver();
driver.get("http://www.baidu.com");
try {
File srcFile = ((TakesScreenshot)driver).
getScreenshotAs(OutputType.FILE);
FileUtils.copyFile
(srcFile,new File("d:\\screenshot.png"));
} catch (Exception e) {
e.printStackTrace();
}
driver.close();
}
}
```

TakesScreenshot接口提供了getScreenshotAs()方法来捕捉屏幕。上面的例子中，我们指定了OutputType.FILE作为参数传递给getScreenshoAs()方法，告诉它将截取的屏幕以文件形式返回。

使用org.apache.commons.io.FileUtils类中的copyFile()方法来保存getScreenshot()返回的文件对象。TakesScreenshot接口依赖于浏览器中的API来捕捉屏幕。所以在HtmlUnit Driver中不支持这样使用。

◆ 将浏览器窗口最大化

```
package com.example.tests;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2 {
@Test
public void testTakesScreenshot() {
WebDriver driver = new InternetExplorerDriver();
driver.get("http://www.baidu.com");
driver.manage().window().maximize();
driver.close();
}
}
```

◆ 自动选择下拉列表

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
public class Selenium2 {
@Test
public void testDropdown() {
WebDriver driver = new FirefoxDriver();
driver.get("D:\\demo\\Droplist.html");
//得到下拉列表框
```

```
Select make =
new Select(driver.findElement(By.name("make")));
//验证下拉列表的不支持多选
assertFalse(make.isMultiple());
//验证下拉列表的数量
assertEquals(4,make.getOptions().size());
//命名用可见的本文来选择选项
make.selectByVisibleText("Honda");
//通过value属性来选择选项
make.selectByValue("Audi");
//通过索引来选择选项
make.selectByIndex(2);
driver.close();
}
}
```

在执行下面的例子时需要导入org.openqa.selenium.support.ui.Select类。首先创建一个Select乩对象，

isMultiple()用来判断是丌是多选下拉框。

Select类提供了3种方法来选择下拉选项：

selectByVisibleText()，selectByValue(),selectByIndex()。

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
public class Selenium2 {
@Test
public void testMultipleSelectLis() {
WebDriver driver = new FirefoxDriver();
driver.get("D:\\demo\\Droplist.html");
// 得到下拉列表框
Select color = new Select(driver.findElement(By.name("color")));
// 验证下拉列表支持多选
assertTrue(color.isMultiple());
// 验证下拉列表的数量
```

```
assertEquals(4, color.getOptions().size());
// 使用可见的本文来选择选项
color.selectByVisibleText("Black");
color.selectByVisibleText("Red");
color.selectByVisibleText("Silver");
// 通过可见的文本取消已选选项
color.deselectByVisibleText("Silver");
// 通过value属性取消已选选项
color.deselectByValue("red");
// 通过选项索引取消已选选项
color.deselectByIndex(0);
}
}
```

◆ 检查下拉列表中的选项

检查单选的下拉框：

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
public class Selenium2 {
@Test
public void testDropdown() {
WebDriver driver = new FirefoxDriver();
driver.get("D:\\demo\\Droplist.html");
//得到下拉列表框
Select make =
new Select(driver.findElement(By.name("make")));
//验证下拉列表的不支持多选
assertFalse(make.isMultiple());
//验证下拉列表的数量
assertEquals(4,make.getOptions().size());
//使用可见的本文来选择选项
make.selectByVisibleText("Honda");
assertEquals
("Honda",make.getFirstSelectedOption().getText());
```

```
//通过value属性来选择选项
make.selectByValue("Audi");
assertEquals("Audi",
make.getFirstSelectedOption().getText());
//通过索引来选择选项
make.selectByIndex(2);
assertEquals("BMW",
make.getFirstSelectedOption().getText());
}
}
```

检查多选的下拉框：

```
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
public class Selenium2 {
@Test
public void testMultipleSelectLis() {
WebDriver driver = new FirefoxDriver();
driver.get("D:\\demo\\Droplist.html");
//得到下拉列表框
Select color = new Select(driver.findElement(By.name("color")));
//验证下拉列表支持多选
assertTrue(color.isMultiple());
//验证下拉列表的数量
assertEquals(4,color.getOptions().size());
//用可见的本文来选择选项
color.selectByVisibleText("Black");
color.selectByVisibleText("Red");
color.selectByVisibleText("Silver");
//验证所选的选项
List<String> exp_sel_options =
Arrays.asList(new String[] {"Black","Red","Silver"});
List<String> act_sel_options = new ArrayList<String>();
for(WebElement option:color.getAllSelectedOptions()){
act_sel_options.add(option.getText());
}
//验证选择的选项和我们期望的是一样的
assertArrayEquals
```

```
(exp_sel_options.toArray(), act_sel_options.toArray());
//验证3个选项已经被选择了
assertEquals(3, color.getAllSelectedOptions().size());
//通过可见的文本取消已选选项
color.deselectByVisibleText("Silver");
assertEquals(2, color.getAllSelectedOptions().size());
//通过value属性取消已选选项
color.deselectByValue("red");
assertEquals(1, color.getAllSelectedOptions().size());
//通过选项索引取消已选选项
color.deselectByIndex(0);
assertEquals(0, color.getAllSelectedOptions().size());

}

}
```

如果只是单选的下拉列表，通过getFirstSelectedOption()就可以得到所选择的选项，再调用getText()就可以得到本文。如果是多选的下拉列表，使用getAllSelectedOptions()得到所有已选择的选项，此方法会返回元素的集合。使用assertArrayEquals()方法来对比期望和实际所选的选项是否正确。调用getAllSelectedOptions().size()方法来判断已选的下拉列表选项数量。如果想检查某一个选项是否被选择了，可以使用assertTrue(act_sel_options.contains("Red"))方法。

◆ 自动选择单选按钮

Selenium WebDriver的WebElement类支持单选按钮和按钮组。我们可以通过click()方法来选择单选按钮和取消选择，使用isSelect()方法来判断是否选中了单选按钮。

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
@Test
public void testRadioButton() {
WebDriver driver = new FirefoxDriver();
```

```
driver.get("D:\\demo\\RadioButton.html");
//使用value值来定位单选按钮
WebElement apple =
driver.findElement(By.cssSelector("input[value='Apple']"));
//检查是否已选择，如果没有则点击选择
if(!apple.isSelected()){
apple.click();
}
//验证apple选项已经选中
assertTrue(apple.isSelected());
//也可以得到所有的单选按钮
List<WebElement> fruit =
driver.findElements(By.name("fruit"));
//查询Orange选项是否存在，如果存在则选择
for(WebElement allFruit : fruit){
if(allFruit.getAttribute("value").equals("Orange")){
if(!allFruit.isSelected()){
allFruit.click();
assertTrue(allFruit.isSelected());
break;
}
}
}
}
}
```

◆ 自动选择多选项框

Selenium WebDriver的WebElement类也支持多选框。我们可以使用click()方法来选择和取消选择，使用isSelect()方法来判断是否选中。

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
@Test
public void testRadioButton() {
WebDriver driver = new FirefoxDriver();
driver.get("D:\\demo\\checkbox.html");
//使用value值来选择单选按钮
```

```
WebElement apple = driver.findElement
(By.cssSelector("input[value='Apple']"));
WebElement pear = driver.findElement
(By.cssSelector("input[value='Pear']"));
WebElement orange = driver.findElement
(By.cssSelector("input[value='Orange']"));
//检查是否已选择，如果没有则点击选择
if(!apple.isSelected()){
apple.click();
}
if(!pear.isSelected()){
pear.click();
}
if(!orange.isSelected()){
orange.click();
}
//验证选项已经选中
assertTrue(apple.isSelected());
assertTrue(pear.isSelected());
assertTrue(orange.isSelected());
//再次点击apple多选框，取消选择
if(apple.isSelected()){
apple.click();
}
assertFalse(apple.isSelected());
}
```

◆ 处理 windows 的进程

Selenium WebDriver java提供了windowsUtils类来和Windows操作系统交互。在测试开始时，我们需要关掉已经一些进程。

如下例子，关闭已打开的火狐浏览器：

```
@Before
public void setUp()
{
WindowsUtils.tryToKillByName("firefox.exe");
driver = new FirefoxDriver();
driver.get("http://www.google.com");
```

```
driver.manage().window().maximize();
}
```

我们可以使用tryToKillByName方法来关闭任何的windows的进程。如果这个进程不存在则会抛出一个异常，但是，测试还是会正常的执行下去。

◆ 通过 WebDriver 读取 windows 注册表中的值

WindowsUtils类提供了多种方法和windows操作系统的注册表进行交互，如果测试是运行在windows操作系统上的IE浏览器，则可能需要修改一些IE注册表里的设置。使用WindowsUtils类就可以很方便的解决。

我们需要导入org.openqa.selenium.os.WindowsUtils类然后使用readStringRegistryValue()方法来读取注册表里的键值。

```
package com.example.tests;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.os.WindowsUtils;
public class Selenium2 {
@Test
public void testRegistry() {
WebDriver driver = new InternetExplorerDriver();
driver.get("D:\\demo\\checkbox.html");
String osname = WindowsUtils.readStringRegistryValue
("HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\OS");
System.out.println(osname);
}
}
```

WindowsUtil也基于返回值的不同提供了多种方法来读取注册表的值，例子中返回的是String所以使用readStringRegistryValue()，还可以据不同的数据类型使用readIntegerRegistryValue()，readBooleanRegistryValue()根。

◆ 通过 WebDriver 修改 windows 注册表的值

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.os.WindowsUtils;
public class Selenium2 {
@Test
public void testRegistry() {
WebDriver driver = new InternetExplorerDriver();
driver.get("D:\\demo\\checkbox.html");
WindowsUtils.writeStringRegistryValue
("HKEY_CURRENT_USER\\SOFTWARE\\Selenium\\SeleniumVersion",
"2.24");
assertEquals("2.24",
WindowsUtils.readStringRegistryValue
("HKEY_CURRENT_USER\\SOFTWARE\\Selenium\\SeleniumVersion"));
}
}
```

writeStringRegistryValue()通过注册表的路径找到相应的位置，如果值存在则修改，如果不存在则新建一条新的，同时基于写入数据类型也提供了其他两种方法writeIntegerRegistryValue(),writeBooleanRegistryValue()。

◆ 使用隐式的等待同步测试

Selenium WebDriver提供了隐式等待来同步测试。当使用了隐式等待执行测试的时候，如果WebDriver没有在DOM中找到元素，将继续等待，超出设定时间后则抛出找不到元素的异常。

换句话说，当查找元素或元素并没有立即出现的时候，隐式等待将等待一段时间再查找DOM。默认的时间是0 。

一旦设置了隐式等待，则它存在在整个WebDriver对象实例的生命周期中，但是，隐式的等待会让一个正常响应的应用的测试变慢，它将会在寻找每个元素的时候都进行等待，这样就增加了整个测试过程的执行时间。

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.concurrent.TimeUnit;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.os.WindowsUtils;
public class Selenium2 {
@Test
public void testWithImplicitWait() throws InterruptedException {
WebDriver driver = new InternetExplorerDriver();
driver.get
("http://demo.tutorialzine.com/2009/09/" +
"simple-ajax-website-jquery/demo.html");
//等待10秒
driver.manage().timeouts(). implicitlyWait(10, TimeUnit.SECONDS);
WebElement page4button = driver. findElement(By.linkText("Page 4"));
page4button.click();
WebElement message = driver.findElement(By.id("pageContent"));
//等待Ajax的内容出现
Thread.sleep(4000);
assertTrue(message.getText().contains("Nunc nibh tortor"));
}
}
```

Selenium WebDriver提供了Timeouts接口来设置隐式等待。Timeouts接口下又提供了

implicitlyWait()方法，接收一个查询元素所需要等待乩时间参数。
Thread.sleep(4000)意思义等待4秒钟，如果没有返句加上pageContent本身就存在，所以它乩内容就是最初没有点击时候的文本，不是我们所期望的。。

◆ 使用显式的等待同步测试

Selenium WebDriver也提供了显式的等待，相对于隐式来说更好的控制方法来同步测试。

不像隐式等待，你可以在执行下一次操作时，自定义等待条件。
显式的等待另需要执行在需要同步的地方而不影响脚本其他乩地方。

Selenium WebDriver提供了WebDriverWait和ExpectedCondition类来执行显式等待。
ExpectedCondition类提供了一系列预定义好的条件来等待。下面的表格中显示了一些我们经常在自动化测试中遇到的常用条件。

预定义条件	方法名
元素可见可点击	elementToBeClickable(By locator)
元素被选中	elementToBeSelected(WebElement element)
存在一个元素	presenceOfElementLocated(By locator)
元素中出现指定乩文本	textToBePresentInElement(By locator, String text)
元素乩值	textToBePresentInElementValue(By locator, String text)
标题	titleContains(String title)

```
package com.example.tests;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.*;

public class Selenium2 {
@Test
public void testWithImplicitWait() throws InterruptedException {
WebDriver driver = new InternetExplorerDriver();
driver.get("http://demo.tutorialzine.com/2009/09/simple-ajax-website-jquery/demo.html");
WebElement page4button = driver.findElement(By.linkText("Page 4"));
page4button.click();
//设置等待时间10秒
WebDriverWait wait = new WebDriverWait(driver, 10);
```

```
//等待直到符合元素文本内容出现。
wait.until(ExpectedConditions.textToBePresentInElement
(By.id("pageContent"),
"Nunc nibh tortor, " +
"congue pulvinar rhoncus quis, " +
"porta sed odio. Quisque ornare, " +
"velit elementum porta consequat, " +
"nibh augue tincidunt magna, " +
"at ullamcorper ligula felis vitae felis."));
```

WebDriverWait每500毫秒调用一次ExpectedCondition直到正确的返回值。可见返样的好处就是随时控制所需要等待的地方，更加精确的控制所需要乩条件

◆ 使用自定义的期望条件同步测试

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.*;
public class Selenium2 {
@Test
public void testWithImplicitWait() throws InterruptedException {
WebDriver driver = new InternetExplorerDriver();
driver.get("http://www.w3school.com.cn/" +
"ajax/ajax_example.asp");
WebElement button = driver.findElement(By.tagName("button"));
//验证点击之前的文本
assertTrue(driver.findElement
(By.cssSelector("#myDiv h3"))
.getText().contains("Let AJAX"));
button.click();
//设置等待时间为5秒
WebDriverWait wait = new WebDriverWait(driver, 5);
//创建一个新的ExpecctedCondition接口，就必须实现apply方法
WebElement message = wait.until
```

```
(new ExpectedCondition<WebElement>(){
public WebElement apply(WebDriver d){
return d.findElement(By.cssSelector("#myDiv p"));
}
});
//验证点击后的文本
assertTrue(message.getText().contains("AJAX is"));
driver.close();
}
}
```

这里通过ExpectedCondition和WebDriverWait类我们自定义了一个期望条件。在这个例子中我们需要在5秒内定位到指定的元素，其实通过自身的presenceOfElementLocated(By locator)方法也能实现.

等待元素的属性值改变：

```
(new WebDriverWait(driver, 10)).until(
new ExpectedCondition<Boolean>()
{
public Boolean apply(WebDriver d)
{
return
d.findElement(By.id("userName")).getAttribute("readonly").contain
s("true");
}});
```

如下例子也是等待某属性改变：

```
driver.get("http://www.jquery.com");
WebElement button =
driver.findElement(By.linkText("RUN CODE"));
button.click();
//设置等待时间为5秒
WebDriverWait wait = new WebDriverWait(driver, 5);
//这里apply的返回值为Boolean
Boolean classname = wait.until
(new ExpectedCondition<Boolean>(){
public Boolean apply(WebDriver d){
return d.findElement(
By.cssSelector(".jq-codeDemo p"))
.getAttribute("class")
.contains("ohmy");
}
});
driver.close();
```

等待元素变为可见：

```
package com.example.tests;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.*;
public class Selenium2 {
@Test
public void testWithImplicitWait() throws InterruptedException {
WebDriver driver = new InternetExplorerDriver();
driver.get("http://www.jquery.com");
WebElement button =
driver.findElement(By.linkText("RUN CODE"));
button.click();
//设置等待时间为5秒
WebDriverWait wait = new WebDriverWait(driver, 5);
//创建一个新的ExpecctedCondition接口，就必须实现apply方法
Boolean classname = wait.until
(new ExpectedCondition<Boolean>(){
public Boolean apply(WebDriver d){
return d.findElement(
By.cssSelector(".jq-codeDemo p"))
.isDisplayed();
}
});
driver.close();
}
}
```

等待DOM的事件

Web应用可能使用了AJAX的框架如jQuery来操作网页内容。例如，jQuery经常异步的会去从服务器加载一个很大的JSON文件。当jQuery正在处理文件的时候，测试可以使用active属性。自定义的等待可以通过执行一段JavaScript代码并检查返回值来完成。

```
(new WebDriverWait(driver, 10)).until(
new ExpectedCondition<Boolean>()
{
public Boolean apply(WebDriver d)
 {
JavascriptExecutor js = (JavascriptExecutor) d;
return (Boolean)js.executeScript("return jQuery.active == 0");
}});
```

◆ 检查元素是否存在

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class Selenium2 {
WebDriver driver = new InternetExplorerDriver();
@Test
public void testisElementPresent(){
driver.get("http://www.baidu.com");
driver.findElement(By.id("kw")).sendKeys("selenium");
//判断搜索按钮是否存在
if(isElementPresent(By.id("su1"))){
//点击按钮
driver.findElement(By.id("su")).click();
}else{
fail("元素不存在");
}
}
private boolean isElementPresent(By by)
{
try
{
driver.findElement(by);
return true;
}
catch (Exception e)
{
e.printStackTrace();
return false;
}
}
```

```
}
```

◆ 检查元素的状态

isSelected()	检查元素是否被选中（单选，多选，下拉框）
disDisplayed()	检查元素是否可见

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
@Test
public void testRadioButton() {
WebDriver driver = new FirefoxDriver();
driver.get("D:\\demo\\RadioButton.html");
//使用value值来定位单选按钮
WebElement apple =
driver.findElement(By.cssSelector("input[value='Apple']"));
//检查是否已选择，如果没有则点击选择
if(!apple.isSelected()){
apple.click();
}
//验证apple选项已经选中
assertTrue(apple.isSelected());
//也可以得到所有的单选按钮
```

```
List<WebElement> fruit =
driver.findElements(By.name("fruit"));
//查询Orange选项是否存在，如果存在则选择
for(WebElement allFruit : fruit){
if(allFruit.getAttribute("value").equals("Orange")){
if(!allFruit.isSelected()){
allFruit.click();
assertTrue(allFruit.isSelected());
break;
}
```

◆ 通过名称识别和处理一个弹出窗口

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
WebDriver driver = new FirefoxDriver();
@Test
public void testWindowPopup(){
driver.get("D:\\demo\\window.html");
//保存父窗口
String parentWindowId = driver.getWindowHandle();
//点击按钮弹出窗口
WebElement helpButton = driver.findElement(By.id("helpbutton1"));
helpButton.click();
try
 {
```

```
//转到HelpWindow
driver.switchTo().window("HelpWindow");
}
catch (NoSuchWindowException e)
{
e.printStackTrace();
}

//验证新窗口里的文本
assertEquals("PopUpWindow", driver.findElement(By.tagName("p")).getText());
//关闭子窗口
driver.close();
//回到父窗口
driver.switchTo().window(parentWindowId);
//验证父窗口的title
assertTrue(driver.getTitle().equals("help"));
driver.close();
}
}
```

◆ 通过标题识别处理一个弹出窗口

很多时候开发人员并没有给弹出的窗口分配一个name属性。这种情况下，我们可以使用handle属性。但是，handle的值是不停的变化的，这让识别窗口变的有些困难，尤其是多个窗口的时候。我们使用handle和title来识别一个弹出窗口。

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.Set;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Selenium2 {
WebDriver driver = new FirefoxDriver();
@Test
public void testWindowPopup(){
driver.get("D:\\demo\\window.html");
//保存父窗口
```

```
String parentWindowId = driver.getWindowHandle();
//点击按钮弹出窗口
WebElement helpButton =
driver.findElement(By.id("helpbutton2"));
helpButton.click();
//得到所有的窗口
Set<String> allWindowsId = driver.getWindowHandles();
//通过title得到新的窗口
for (String windowId : allWindowsId)
 {
if (driver.switchTo().window(windowId)
.getTitle().equals("PopUpWindow"))
{
driver.switchTo().window(windowId);
break;
}
}
//验证新窗口的文本
assertEquals("PopUpWindow",
driver.findElement(By.tagName("p")).getText());
//关闭弹出窗口
driver.close();
//关闭父窗口
driver.switchTo().window(parentWindowId);
driver.close();
}
}
```

◆ 通过网页内容识别处理一个弹出窗口

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.Set;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
WebDriver driver = new FirefoxDriver();
@Test
public void testWindowPopup(){
driver.get("D:\\demo\\window.html");
```

```
//保存父窗口
String parentWindowId = driver.getWindowHandle();
//点击按钮弹出窗口
WebElement helpButton =
driver.findElement(By.id("helpbutton2"));
helpButton.click();
//得到所有的窗口
Set<String> allWindowsId = driver.getWindowHandles();
//通过查找页面内容得到新的窗口
for (String windowId : allWindowsId) {
driver.switchTo().window(windowId);
if (driver.getPageSource().contains("Welcome")){
driver.switchTo().window(windowId);
break;
}
}
//验证新窗口的文本
assertEquals("PopUpWindow",
driver.findElement(By.tagName("p")).getText());
//关闭弹出窗口
driver.close();
//关闭父窗口
driver.switchTo().window(parentWindowId);
driver.close();
}
}
```

通过driver.getPageSource()方法得到页面的html内容,再调用contains()方法来判断是否含有指定的内容。这样做的缺点就是如果页面内容很多，获得的速度会慢，而且contains()里面查找内容必须是唯一，否则可能识别的窗口不是你所期望的。

◆ 处理一个简单的 JavaScript 警告窗

```
//获取alert窗口
```

```
Alert alertBox = driver.switchTo().alert();
alertBox.accept();
//验证alert窗口里的文字
assertEquals("Hello World",alertBox.getText());
driver.close();
```

driver.switchTo().alert()将窗口移劢到警告框上。
alert.getText()得到警告框上乩信息。
如果找不到警告框则会抛出NoAlertPresentException异常。

◆ 处理一个确认框

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
WebDriver driver = new FirefoxDriver();
@Test
public void testWindowPopup(){
driver.get("D:\\demo\\alert.html");
//点击确定按钮
getConfirmBox().accept();
//验证点击后的文字
assertEquals("你点击了确定按钮！",
driver.findElement(By.cssSelector("span")).getText());
//点击取消按钮
getConfirmBox().dismiss();
assertEquals(″你点击了取消按钮！",
driver.findElement(By.cssSelector("span")).getText());
driver.close();
}
//封装得到窗口的方法
private Alert getConfirmBox(){
//点击按钮弹出确认提示框
```

```
WebElement button = driver.findElement(By.id("confirm"));
button.click();
//获取确认提示框
Alert confirmBox = driver.switchTo().alert();
assertEquals("我是确认提示框！",confirmBox.getText());
return confirmBox;
}
}
```

点击确认是使用accept()方法，点击取消使用dismiss()方法。

◆ 处理一个提示框

```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Selenium2 {
WebDriver driver = new FirefoxDriver();
@Test
public void testPromptAlert(){
driver.get("D:\\demo\\alert.html");
WebElement button = driver.findElement(By.id("prompt"));
button.click();
//获得提示框
Alert promptAlert = driver.switchTo().alert();
assertEquals("点都点了，就输入点什么吧",promptAlert.getText());
//输入一些数据
promptAlert.sendKeys("洛阳亲友如相问，就说我在写代码");
//点击确定按钮
promptAlert.accept();
//验证输入的数据
String actualTest = driver.findElement(By.tagName("span"))
.getText();
assertEquals("洛阳亲友如相问，就说我在写代码", actualTest);
driver.close();
}
}
```

◆ 识别处理框架

```
//通过id定位到左边的框架
driver.switchTo().frame("left");
String leftMsg = driver.findElement(By.tagName("p")).getText();
assertEquals("i am left page",leftMsg);
//回到初始的焦点
driver.switchTo().defaultContent();
//通过name定位到右边的框架
driver.switchTo().frame("right");
String rightMsg = driver.findElement(By.tagName("p")).getText();
assertEquals("i am right page",rightMsg);
driver.close();
```

通过index获取frame框架：

```
//通过index来定位框架
driver.switchTo().frame(1);
//验证中间框架的文本
String middleMsg = driver.findElement(By.tagName("p"))
.getText();
assertEquals("i am middle page",middleMsg);
```

◆ 通过页面内容识别和处理框架

```
//得到所有的frame元素
List<WebElement> frames = driver.findElements(By.tagName("frame"));
//通过页面的内容得到中间的框架
for (int i = 0; i < frames.size(); i++)
 {
driver.switchTo().frame(i);
if (driver.getPageSource().contains("middle"))
```

```
{
break;
//没有匹配的时候需要回到最初的页面
}
else
{
driver.switchTo().defaultContent();

}

String actualText = driver.findElement(By.tagName("p")).getText();
assertEquals("i am middle page", actualText);
```

◆ 处理 IFRAM

```
//首先获得父窗口
driver.switchTo().frame("left");
//取得iframe元素
WebElement weiboIframe = driver.findElement(By.tagName("iframe"));
//获得iframe窗口
driver.switchTo().frame(weiboIframe);
//验证iframe里面的页面内容
String actualText = driver.findElement(By.linkText("新浪微博")).getText();
assertEquals("新浪微博", actualText);
driver.switchTo().defaultContent();
```