



快速回答常见问题

Selenium测试工具食谱

基于Selenium WebDriver有超过90个秘籍菜谱去建立、维护和
改进自动化测试

Unmesh Gundecha

[PACKT] open source*
PUBLISHING community experience distilled

Selenium 测试工具食谱

基于 Selenium WebDriver 有超过 90 个秘籍菜谱去建立

维护和改进自动化测试

Unmesh Gundecha



BIRMINGHAM - MUMBAI

Selenium Testing Tools Cookbook

Copyright © 2012 Packt Publishing

版权所有。本书的任何一个部分在没有得到出版商的授权之前都不允许复制，储存在检索系统中，或者以任何形式或是任何方式进行传播。除了对本书有疑义的文章或是引用本书的部份内容。

为本书做的每一份努力是为了确保展现出准确的信息。但是，本书所包含的内容是没有保质期的，工具可能会随着版本的改变，可会出现与书中所描述不符合的情况。

任何对本书造成了直接或是间接损害的无论是作者，Packt 出版社，或是经销商都将会被追究法律责任。

Packt 出版社努力用大写字母标识出本书所涉及到的公司、产品提供商标信息，但是，Packt 出版社无法保证这些信息是正确的。

第一次印刷：2012

产品参考：1161112

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-84951-574-0

www.packtpub.com

Cover Image by Faiz Fattohi (faizfattohi@gmail.com)

工作人员

Author

Unmesh Gundecha

Project Coordinator

Yashodhan Dere

Reviewers

V.Vamsi Chandra

Dave Hunt

Proofreaders

Matthew Humphries

Lydia May Morris

Acquisition Editor

Usha Iyer

Indexer

Hemangini Bari

Lead Technical Editor

Azharuddin Sheikh

Technical Editors

Mayur Hule

Ankita Shashi

Graphics

Valentina D'silva

Aditi Gajjar

Production Coordinator

Arvindkumar Gupta

Copy Editors

Brandt D'Mello

Laxmi Subramanian

Alfida Paiva

Cover Work

Arvindkumar Gupta

关于作者

Unmesh Gundecha : 一位拥有着软件工程研究学历和近 10 软件开发和测试工作经验的人员。Unmesh 利用商业的和开源的自动化测试工具构建了符合行业标准的内部和自定义的功能自动化测试项目。目前, 他正在印度普纳的一个跨国公司中担任测试架构师。

如果我们不说感谢所有人, 可能会疏忽感谢某些人, 所以感谢所有给予这本书帮助的人们。这包括了非常了不起的 Packt 出版社的人们, 尤其是 Usha Iyer - 组稿编辑。正是他建议我来写这本书。我也非常感谢 Packt 出版社编辑部的工作人员对于本书的审查 - Azharuddin Sheikh, Sonali Tharwani, Mayur Hule, Veronica Fernandes, Prashant Salvi, 尤其是项目协调员 Yashodhan Dere, 正是他协调着整个项目的进度, 确保我一直在正确的轨道上。

这本书也得益于有一个非常伟大的技术审查小组。在此也要非常感谢每一位自愿者贡献出来他们的私人时间来审阅最初的草稿并提供了非常有价值的反馈, Dave Hunt - Selenium 贡献者, 他对某些领域有着非常深入的了解; Vamsi Chandra 确保了书中代码的例子是可以正确执行的; Tarun Kumar 给予了最早的反馈。

我还要感谢我的妻子 Punam 的支持, 让我可以准时的完成这本书, 还有我的朋友、同事这么多年的支持

最后, 最由衷的感谢 Selenium 开发者和用户社区建立了这么一个伟大的工具。

关于审查者

V.Vamsi Chandra 在 Evolution 科技公司是一位 QA 自动化主管，在软件行业有着 5 年以上的工作经验。他有着移动计算机网络的硕士学历，和计算机科学工程的学士学历。同时他也获得了很多的证书，如 ISTQB、MCP、MCSE、MCITP and ITIL v3。他参与了利用各种自动化和手动工具来改善产品的质量，作品有软件开发生命周期(SDLC),软件测试生命周期(STLC),和敏捷(scrum)测试方法交付高标准/复杂的产品给客户。

他在 Evolution 公司很多的部门工作过，比如财务部，银行，商业和零售，处理复杂的项目并为 Fifth Third Bank-USA，Sainsbury's，Myindospace，Nationwide Building Society，and Mergermarket 设计了定制的框架。公司也提供了基于 Web 的产品。

我要感谢 Unmesh Gundecha 先生给我提供了这个美妙和令人激动的机会来展现我的想法并可以在技术外做更多的一些探索。我也非常感谢给我支持的团队和充分的沟通来实现目标的成功。

Dave Hunt 和他的妻子和儿子生活在英国的肯特郡。他一直热爱把普通的任务可以转变成一键式的解决方案，在 2005 年发现 Selenium 之前，他的软件测试和自动测试工作基乎是没有的。他在家为 Mozilla 公司工作，帮助团队对他们的项目建立自动化测试，从 Mozilla 网页到 Firefox 浏览器，Thunderbird e-mail 客户端。

关于译者

这是我的第二本关于 selenium 翻译的书籍，第一本为<Selenium Beginners' s Guide>，名为 Selenium 初学者指南，主要介绍了使用 Selenium IDE，和一些关于 Selenium RC 相关的知识。虽然再后期的自动化开发过程当中 IDE 并不常用。但是帮助我们定位元素，录一些简单的脚本后转成相应的语言的功能还是非常方便的。

关于此书，原版英文书为<Selenium Testing CookBook>，直译起来名为 Selenium 的测试食谱，听起来也不是那么的好听，如取名为 Selenium 测试饕餮也觉得没有达到这样的高度，太有吹嘘的感觉。本书主要介绍了关于 WebDriver 一些主要 API 的使用，对于了解一些基础的朋友，有着非常大的帮助，由于我本身是使用 Java 语言编写的脚本，里面所有的测试用例也都是使用 Java 的，所以对于其它类的语言就没有那么认真的检查，说白了我也不懂，很多就是拷贝原来的东西。如果有读者发些有问题，敬请联系我进行改正。本书删除了一些其他的章节。如 BDD 的使用 (Behavior Driven Development)，和手机平台的测试，有兴趣的去自己研究下。

在这里给大家一些使用自动化框架的人一些学习的方向建议，当大家会使用了一些基础的测试自动化脚本完成了一定的场景后，这仅仅是开始，完成一个好的自动化框架还需要框架的分层，是否可以用 PageFactory 模式，实现功能与页面元素的分离。持续集成的编译 (如 Hudson,Maven)，如通过脚本自动执行 (可以通过 Ant 执行 TestNG 的 xml)，当脚本出错后进行页面的截图，如果可能的话生成一个漂亮的结果报告，是否可以和测试管理工具做集成，当你完成某些测试用例后在测试工具上可以标记哪些用例执行过了，和其状态都是需要大家慢慢去解决的问题。

此书所用的测试用例需要大家存放在 D:\demo 的目录下，这样才可能保证测试用例的正确运行。

虽然经过一些检查但是肯定还是有问题存在，如打错字，代码不能执行的问题，难免会发生，请大家帮助一起来检查一下，如有遇到请通过 Email 联系我进行改正 ryannj@126.com

这里还需要强调一下，当前我使用的 selenium 版本为 2.25，目前已经到了 2.29，有些读者可以遇到了使用最新的 2.29 在启动浏览器的时候报错，当然是有解决方法，大家自己搜索一下即可，有的是使用的是 2.25 旧版本启动最新的 firefox18 的时候打不开页面等问题，大家都需要注意，每个版本都是会有变化的，配置好你的环境和相应的代码。

www.PacktPub.com

文档，电子书，折扣提供等更多信息

你可以访问 www.packtpub.com，得到相关的支持文档，下载与书相关的资料。

你知道吗，Packt 提供很一本书的电子书版本支持 PDF 和 ePub。你可以在 www.packtpub.com 上升级你的电子书版本。作一个想打印书的客户，你有权通过折扣来享受电子的拷贝。详细信息请联系 service@packtpub.com。

在 www.packtpub.com 上，你也可以阅读收集来的免费的技术文章，注册后你就可以享受独家折扣和 Packt 书和电子书的最新信息。



<http://PacktLib.PacktPub.com>

你需要即时解决 IT 相关的问题吗?PacktLib 是 Packt 的在线数字图书馆。在这里,你可以访问、阅读和搜索整个 Packt 图书馆的书。

账户持有人免费访问 Packt

如果你有一个 www.PacktPub.com 的账户,你可以用这个来访问 PacktLib 和免费查看 9 本书。使用你的登录凭证立即访问吧。

目录

目录.....	9
引言.....	12
1. 元素定位.....	16
1.1. 介绍.....	16
1.2. 使用浏览器工具来检查页面元素结构.....	17
1.3. 使用 findElement 方法定位元素.....	23
1.4. 使用 findElements 方法定位元素.....	27
1.5. 定位链接.....	29
1.6. 使用标签名称定位元素.....	30
1.7. 使用 CSS 选择器定位元素.....	30
1.8. 使用 XPath 定位元素.....	34
1.9. 使用文本元素.....	38
1.10. 使用高级的 CSS 选择器定位元素.....	39
1.11. 使用 jQuery 选择器.....	42
1.12. 定位表格行和单元格.....	45
1.13. 定位表格中的子元素.....	47
2. 使用 Selenium API.....	49
2.1 简介.....	50
2.2 检查元素的文本.....	50
2.3 检查元素的属性值.....	51
2.4 检查元素的 CSS 属性值.....	51
2.5 针对鼠标和键盘事件使用高级的用户交互 API.....	52
2.6 在元素上执行双击操作.....	53
2.7 执行拖拽操作.....	53
2.8 执行 JavaScript 代码.....	54
2.9 使用 Selenium WebDriver 进行截图.....	56
2.10 使用 RemoteWebDriver/Grid 进行截图.....	57
2.11 将浏览器最大化.....	58

2.12	自动选择下拉列表.....	59
2.13	检查下拉列表中的选项	60
2.14	自动选择单选按钮.....	63
2.15	自动选择多选框	64
2.16	处理 windows 进程.....	65
2.17	通过 WebDriver 中阅读 windows 注册表中值.....	66
2.18	通过 WebDriver 中修改 windows 注册表中值.....	67
3.	测试流的控制	67
3.1.	简介	68
3.2.	使用隐式的等待同步测试	68
3.3.	使用显式的等待同步测试	70
3.4.	使用自定义条件同步测试	71
3.5.	检查元素是否存在	75
3.6.	检查元素的状态.....	76
3.7.	通过名称识别和处理一个弹出窗口.....	78
3.8.	通过标题识别和处理一个弹出窗口.....	79
3.9.	通过网页内容识别和处理一个弹出窗口	80
3.10.	处理一个简单的 JavaScript 警告框.....	82
3.11.	处理一个确认框	83
3.12.	处理一个提示框	84
3.13.	识别处理框架.....	85
3.14.	通过页面内容识别和处理框架	87
3.15.	处理 IFRAM	88
4.	数据驱动测试	89
4.1.	简介	89
4.2.	使用 JUnit 创建一个数据驱动测试.....	91
4.3.	使用 TestNG 创建一个数据驱动测试.....	93
4.4.	使用 JUnit 从 CSV 读取数据	95
4.5.	使用 JUnit 和 Apach POI 读取 Excel 数据	98
4.6.	使用 JUnit 和 JDBC 来读取数据库中的数据	101
4.7.	创建一个 NUnit 数据驱动测试.....	104
4.8.	使用 ROO 创建一个 Ruby 的数据驱动测试.....	106

4.9. 使用 Python 创建一个数据驱动测试.....	108
4.10. 使用 MSTEST 创建一个数据驱动测试用例.....	111
5. 使用页面对象模型.....	113
5.1. 简介	114
5.2. 使用 PageFactory 类来分离页面元素.....	114
5.3. 使用 PageFactory 类来分离页面操作.....	117
5.4. 实现嵌套页面对象实例.....	119
5.5. 用.NET 实现对象模型.....	122
5.6. 用 Python 实现对象模型.....	123
5.7. 用 Ruby 实现对象模型	124
6. Selenium 扩展.....	126
6.1. 简介	126
6.2. 创建一个表格的扩展类.....	126
6.3. 给元素设置属性.....	128
6.4. 高亮元素.....	129
6.5. 截取元素图片.....	129
7. 测试 HTML5 的网页应用	131
7.1. 简介	131
7.2. 处理 HTML5 的视频播放.....	131
7.3. 网页存储 – 测试本地存储	132
7.4. 网页存储 – 测试 Session 存储.....	134
7.5. 清除 Local 和 Session 的存储数据	134

引言

这本书将帮助你学习先进的使用 WebDriver API 和相关工具来测试 Web 应用。在这本书中，你会学到如何利用 Selenium WebDriver 高效的在桌面，移动网页浏览器和分布式环境中高效的测试 Web 应用。

本书同时也涵盖了设计模式，如数据驱动，页面对象，对象映射，这样可以设计出一个更易维护和稳定的自动化测试框架。你也会将学习如何将 Selenium WebDriver 和行为驱动开发框架集成如 Cucumber-JVM, SpecFlow.NET, Capybara。

这本书也涵盖了通过扩展 Selenium 达到定制自己需求的技术。有超过 90 的菜谱秘籍，你可以用来建立或扩展现有的测试自动化框架。

本书内容简介

第一章 元素定位：介绍了 Selenium WebDriver 的元素定位技术，用来定位你的网页应用上面的元素。Selenium WebDriver 提供了一种高级的技术使用多种定位策略来定位网页上的元素，如 XPath，CSS，DOM。我们也可以执行自定义的定位策略来定位元素。本章将从介绍 Selenium WebDriver Locator API 开始。

第二章 使用 Selenium API：演示了如何使用 Selenium WebDriver API 构建测试。我们将探索 API 和高级的用户交互，执行复杂的鼠标和键盘操，处理网页应用中 UI 元素的各种类型。

第三章 元素定位：测试流控制，演示如何使用 Selenium WebDriver API 构建一个稳定的自动化测试框架。好的自动化测试具有健壮性，稳定性，可以从异常事件中恢复。本章讨论了在测试执行时使用显式和隐式等待来处理同步还有多窗口，警告窗口的处理。

第四章 数据驱动测试：介绍了数据驱动的测试方法，在自动化测试中广为使用。Selenium WebDriver 不支持数据驱动的功能，但是我们可以扩展 Selenium WebDriver API。本章介绍了使用 JUnit/Apache POI 和 JDBC 技术来达到数据驱动测试。

第五章 元素定位：

第六章 元素定位：

第七章 元素定位：

第八章 元素定位：

第九章 元素定位：

你需要为本书估的准备

你需要准备好下列软件

- ◆ 浏览器：Microsoft Internet Explorer, Google Chrome, or Mozilla Firefox
- ◆ Selenium 浏览器的驱动：Chrome Driver, InternetExplorer Driver
- ◆ Selenium 工具：Selenium WebDriver 客户端驱动（基于你喜欢的编程语言）和 Selenium Standalone Server。
- ◆ IDE：Eclipse，IntelliJ IDEA，Microsoft Visual Studio (用于 .NET)。
- ◆ 行为驱动框架：Cucumber-JVM，JBehave（用于 Java），SpecFlow.NET（用于.NET），Capybara（用于 Ruby）
- ◆ 构建集成工具：Maven，ANT，Jenkins。
- ◆ 性能工具：dynaTrace AJAX Edition，HttpWatch，BrowserMob Proxy
- ◆ 其他工具：AutoIt，Sikuli
- ◆ 录相捕捉工具：Monte Media Library（用于 Java），Microsoft Expression Encoder SDK (用于 .NET), and Castro (用于 Python)
- ◆ 移动设备工具：Apple Xcode (用于 iOS 浏览器测试), Android SDK, Android
- ◆ 运行时的语言：JDK 1.6 (用于 Java), Ruby 1.9 (用于 Ruby), and Python 2.7 (用于 Python)

本书适读人群

本书是为从事质量保证，专业测试人员，测试经理，软件开发员，之前有过使用 Selenium 或 Java 测试过基于 Web 的应用经验。本书也提供了 C#，Python，Ruby 的例子。

约定

在这本书中，你会发现许多风格的文本，区分不同种类的信息。下面有一些风格的示例，并解释了其代表的含义。

代码中的字段将显示如下：“WebElement 类也支持查找工能来查找其子元素”

代码块将显示如下

```
<form name="loginForm">
  <label for="username">UserName: </label> <input type="text"
```

```
class="username" /></br>
<label for="password">Password: </label> <input
  type="password" class="password" /></br>
<input name="login" type="submit" value="Login" />
</form>
```

当我们让你重点注意某一段代码块部分的时候，相应的代码为会加粗显示。

```
//Locate all the Checkbox which are checked by calling jQuery
//find() method.
//find() method returns elements in array
List<WebElement> elements = (List<WebElement>)
js.executeScript("return jQuery.find(':checked')");
```

任何命令行的输入和输出将会显示如下

```
mvn clean test
```

专业词汇和**重要的单词**会被加粗显示。以你在屏幕上看到的单词，在菜单或是对话框中的为例，在文本框有一段“右击弹出一个菜单，选择 **Inspect Element** 选项”

约定

读者的反馈向来是很受欢迎的。让我们知道你的看法 – 哪些是你喜欢的 哪些是不喜欢的。读者的有效反馈对我们来说也是一个很重要的发展。

通过发送邮件到 feedback@packtpub.com 给我们提供你宝贵的建议，在邮件的主题上标明反馈的书名。

如果有一个主题，你拥有较高的专业知识且你有兴趣要写一本书，到这里 www.packtpub.com/authors 看看我们的作者指南。

客户支持

你是一位另 Packt 书骄傲的拥有者，我们还有一些事情要做让你的购买得到更多。

下载示例代码

你可以从通过你在 <http://www.packtpub.com> 的账户下载你所购买的书的示例代码的文件。如果你在其他地方购买本书，你可以访问 <http://www.packtpub.com> 注册后可以通过 Email 将文件发送给你。

勘误

尽管我们已经采取所有措施，确保我们的内容的准确性，但是错误还是会发生。如果你在我們的书中发现了错误 - 也许是一个文字或是代码错误，如果你将这它报告给我们，我们会非感谢你。这样做，你也可以解决了别的读者的疑惑并帮助我们提高后续版本的质量。如果你发现任何勘误，请将报们报告到 <http://www.packtpub.com/support>，选择你的书,单击勘误表，点击提交表单的链接，进入详细页面。一旦你的勘误验证，你的提交将被接受，勘误表将会上传到我们的网站，或者添加到任何现有的勘误表中。

版权

盗版在互联网上是一个持续的问题。在 Packt，我们会严肃的保护着我们的版权和许句。如果你遇到任何非法的复制，任何形式在互联网上，请给我们地址或网站名称以便我们能立即寻求补救。

请通过 copyright@packtpub.com 联系我们，附上链接和涉嫌盗版的材料。

我们感激你对我们作者的保护，我们能够给你带来有价值的内容。

疑问

如果你在书中有任何问题，可以通过 questions@packtpub.com 联系我们,我们将尽力解决它。

1.元素定位

在本章中，我们将讨论

- ◆ 使用浏览器工具来检查页面中元素的结构
- ◆ 使用 findElement 方法定位元素
- ◆ 使用 findElements 方法来定位元素
- ◆ 定位链接
- ◆ 通过标签名称定位元素
- ◆ 使用 CSS 选择器定位元素
- ◆ 使用 XPath 定位元素
- ◆ 使用文本定位元素
- ◆ 使用高级 CSS 选择器定位元素
- ◆ 使用 jQuery 选择器
- ◆ 定位表格的行和列
- ◆ 定位表格中的子元素

1.1. 介绍

成功的自动化 **GUI(图形用户界面)**测试取决于从被测试的应用程序中识别和定位 GUI 元素，然后执行操作和验证这些元素来实现测试流。这可以归结为测试工具有效的识别各种 GUI 元素的能力。

Selenium WebDriver 提供一个先进的技术来定位 web 页面元素。Selenium 功能丰富的 API 提供了多个定位策略如:Name、ID、CSS 选择器、XPath 等等。我们也可以执行自定义的定位策略来定位元素。

在本章中，我们将探索如何使用定位策略，从简单的 ID，Name 和 Class 开始。

在任何一个 Web 项目中,给 GUI 的元素附上属性如 Name ,ID 或 Class 是非常好的做法。这使得应用程序更加容易测试符合易访问性的标准。但是,有时候并不是所期望的那样。遇到这些场景,我们就需要使用高级的定位策略如 CSS 选择器和 XPath。

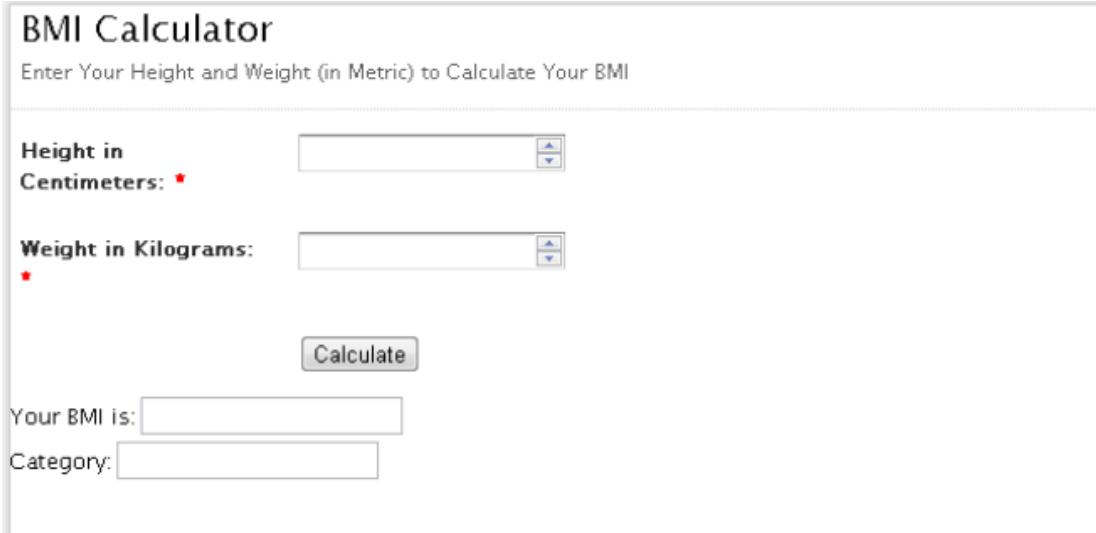
CSS 选择器和 XPath 在 Selenium 用户中非常流行,但是 CSS 选择器相比 XPath 从难易、速度、效率来说更为推荐大家使用。

1.2. 使用浏览器工具来检查页面元素结构

在我们开始探测定位器之前,我们需要先分析一下页面和元素,了解一下他们的结构,元素有哪些属性,JavaScript 或 AJAX 是怎么调用的。等等。

浏览器为最终用户渲染的视觉元素通过隐藏掉 HTML 代码和其他资源。当我们想要用 Selenium WebDriver 自动的进行交互的时候,我们需要仔细查浏览器背后渲染页面和元素的代码。我们需要识别出有用的信息如属性值和元素结构来定位元素,再利用 Selenium WebDriver API 模拟执行用户的操作。

这里有一个 BMI 体重计算的页面和 HTML 代码,下图所示



The image shows a web form titled "BMI Calculator". Below the title is a subtitle: "Enter Your Height and Weight (in Metric) to Calculate Your BMI". The form contains two input fields: "Height in Centimeters:" and "Weight in Kilograms:", each with a red asterisk indicating a required field. Below these is a "Calculate" button. At the bottom, there are two output fields: "Your BMI is:" and "Category:", both with empty text boxes.

```
108 <header id="header" class="info">
109 <h2>BMI Calculator</h2>
110 <div>Enter Your Height and Weight (in Metric) to Calculate Your BMI</div>
111 </header>
112
113 <ul>
114
115 <li id="foli1" class="notranslate">
116 <label class="desc" id="title1" for="heightCMS">
117 Height in Centimeters:
118 <span id="req_1" class="req">*</span>
119 </label>
120 <div>
121 <input name="heightCMS" id="heightCMS" type="number" class="field text medium" value="" maxlength="255" tabindex="1"
122 onkeyup=""/>
123 </div>
124 </li><li id="foli2" class="notranslate">
125 <label class="desc" id="title2" for="weightKg">
126 Weight in Kilograms:
127 <span id="req_2" class="req">*</span>
128 </label>
129 <div>
130 <input name="weightKg" id="weightKg" type="number" class="field text medium" value="" maxlength="255" tabindex="2"
131 onkeyup=""/>
132 </div>
133 </li>
134 <li class="buttons">
135 <div>
```

你可以通过右击浏览器窗口，在弹出的菜单中选择 **View Page Source** 来查看页面的代码。代码将会显示在一个独立的窗口中。但这也可能看起来有一点混乱和难以理解。

我们需要一个特别的工具让显示的信息有结构的更易于理解的格式。在这个秘籍中在深入到定位器之前我们将会介绍一些这样的工具。

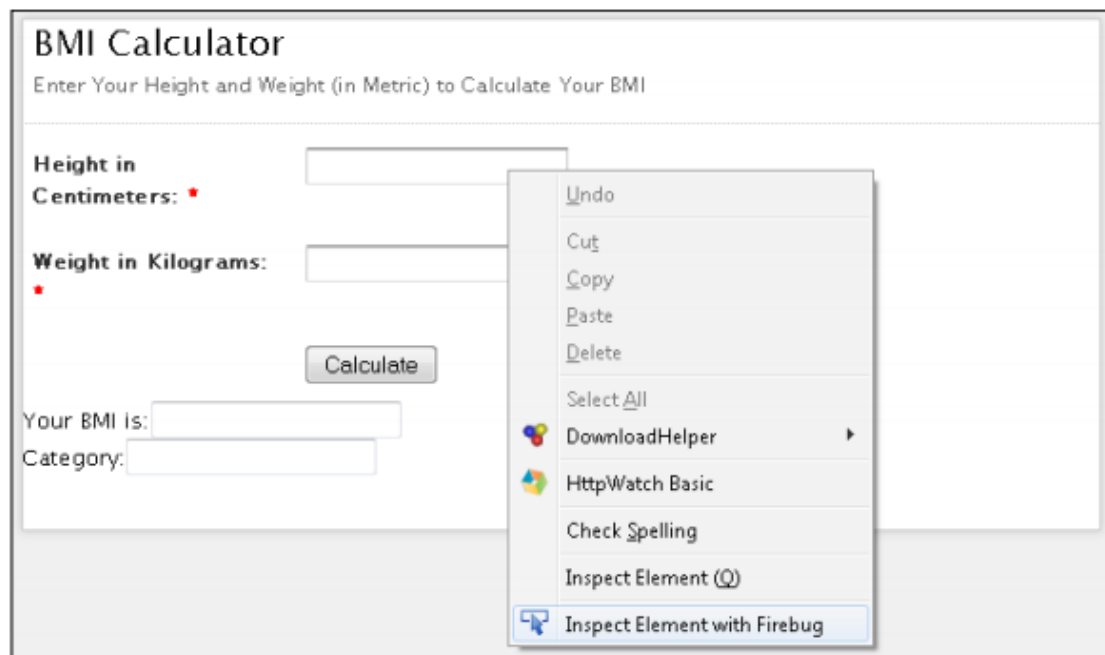
如何实现

在下面几个部分中，我们将探讨一些内置在浏览器的工具和插件来分析元素和页面结构。这些工具将帮助我们了解页面上的元素和它们的属性，DOM 结构，JavaScript 调用，CSS 样式属性等等。

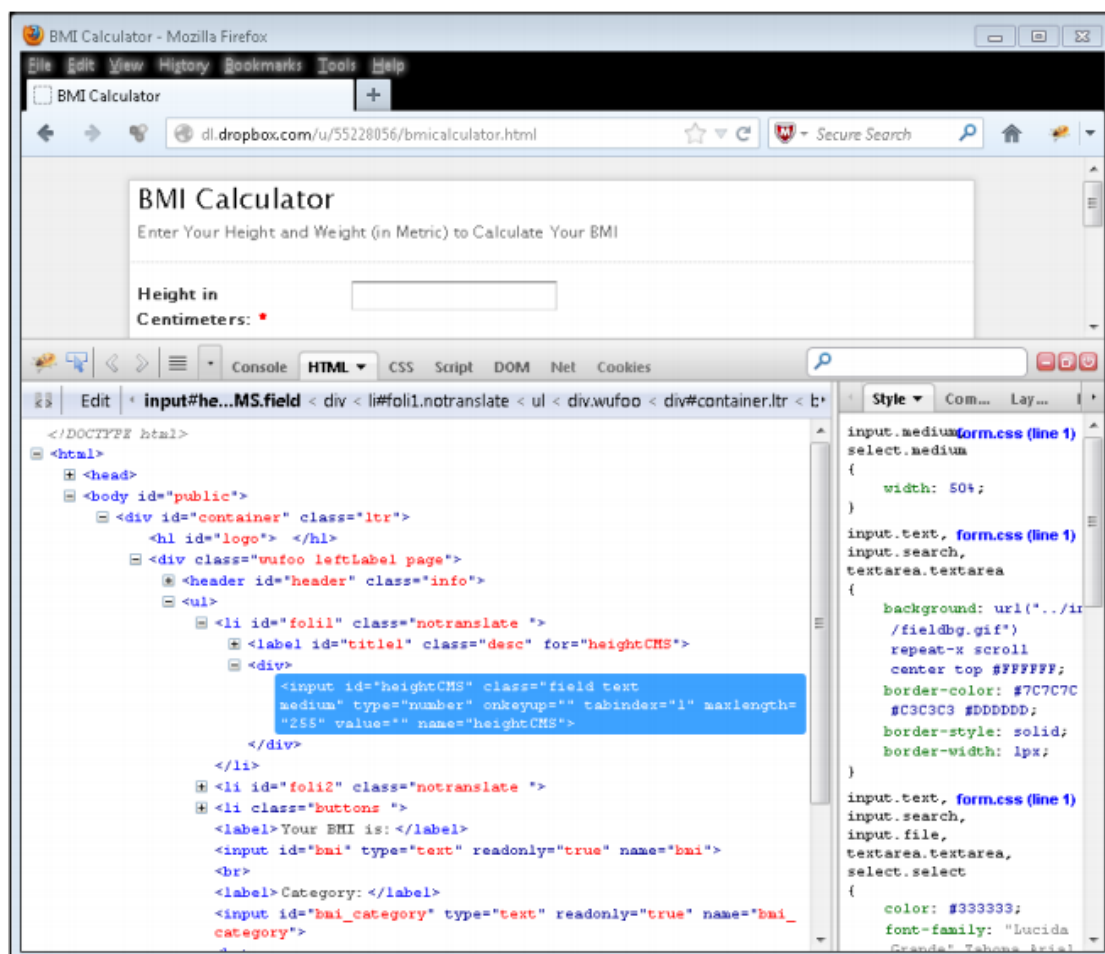
利用 Firefox 的 Firebug 插件来检查页面中的元素

新版本的 Firefox 提供内置的方法来分析页面和元素；然而，我们将使用 Firebug 插件具有更强大的功能。你需要从 <https://addons.mozilla.org/en-us/firefox/addon/firebug/> 安装 Firebug。

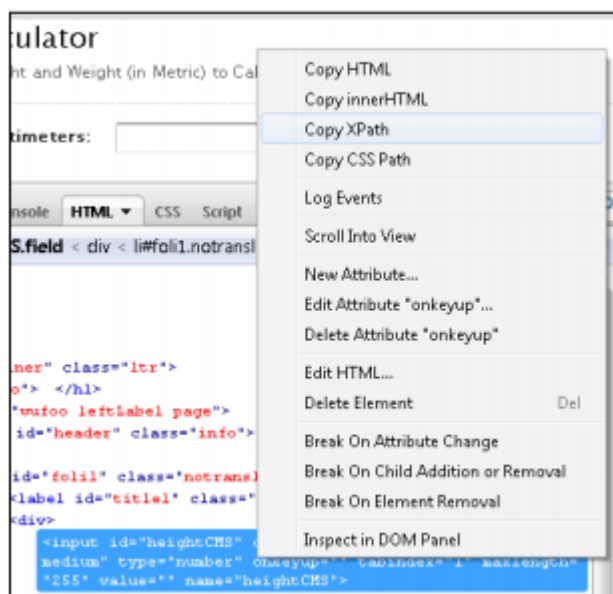
检查页面的元素，将鼠标移向所需查看的元素后右键鼠标打开弹出菜单。选择 **Inspect Element with Firebug** 选项，如下图所示：



HTML 代码在 Firebug 下以树型结构显示出来，如下图所示：



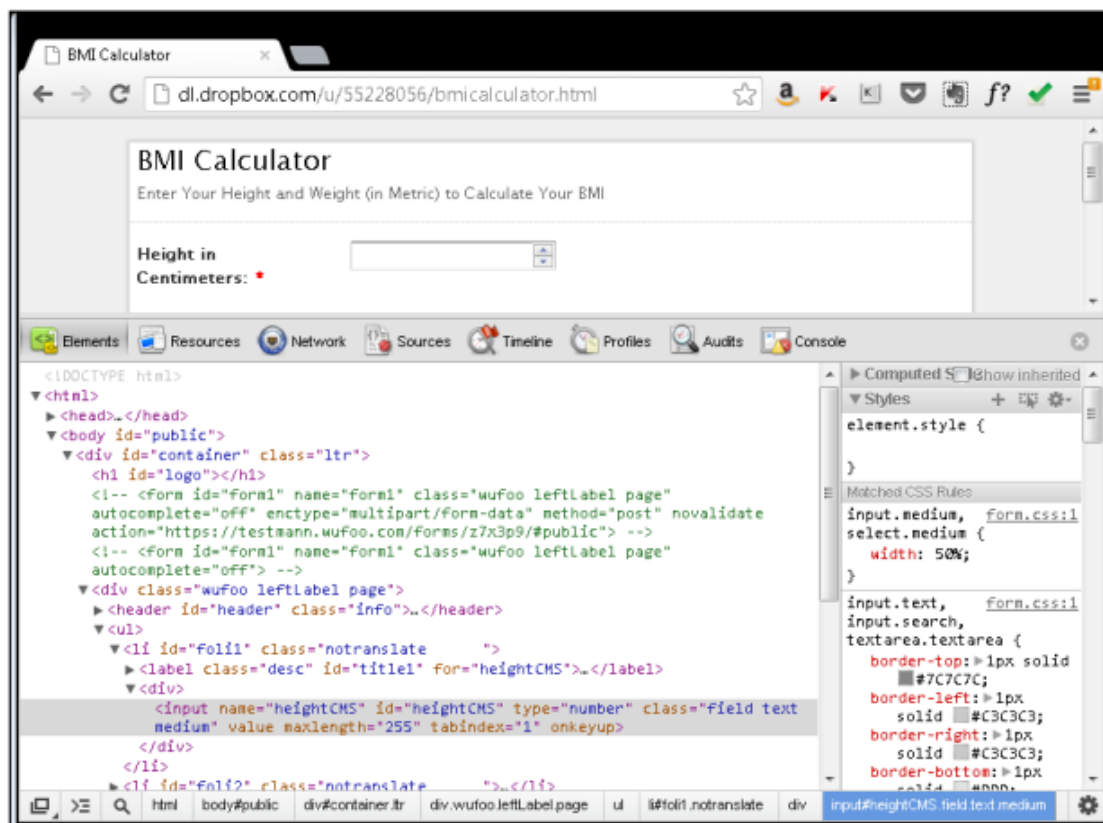
Firebug 提供了各种其他调试特性。它还可以为指定元素生成 XPath 和 CSS 选择器。为此, 选择所需的元素, 然后右击鼠标, 选择 **Copy XPath** 或 **Copy CSS Path** 选项, 如下图所示



XPath 或 CSS 选择器的值将会复制剪贴板上。

利用 Chrome 检查页面中的元素

Chrome 提供了一个内置的功能来分析页面和元素。这和 firebug 很相似。你可以将鼠标移向所需查看的元素, 右击弹出菜单, 然后选择 **Inspect Element** 选项。这将在浏览器中打开开发人员工具。显示信息类似于 Firebug, 如下图所示:

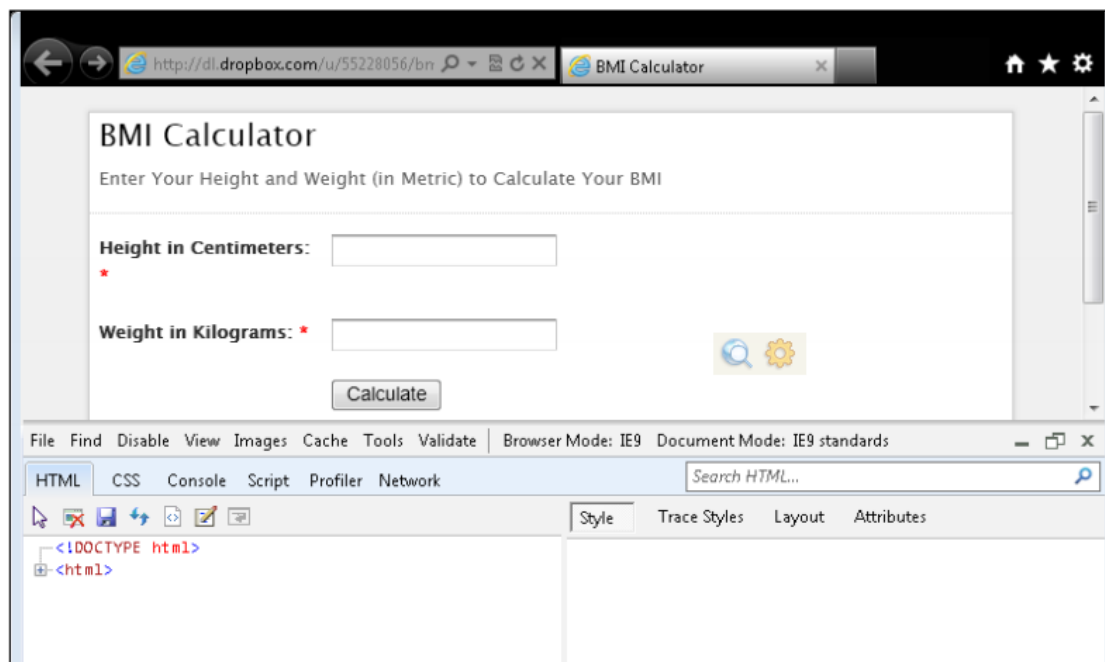



Chrome 开发工具还提供一个特性，在那里你可以得到一个元素的 XPath，右键单击所需的元素，从弹出菜单中选择 **Copy XPath**。

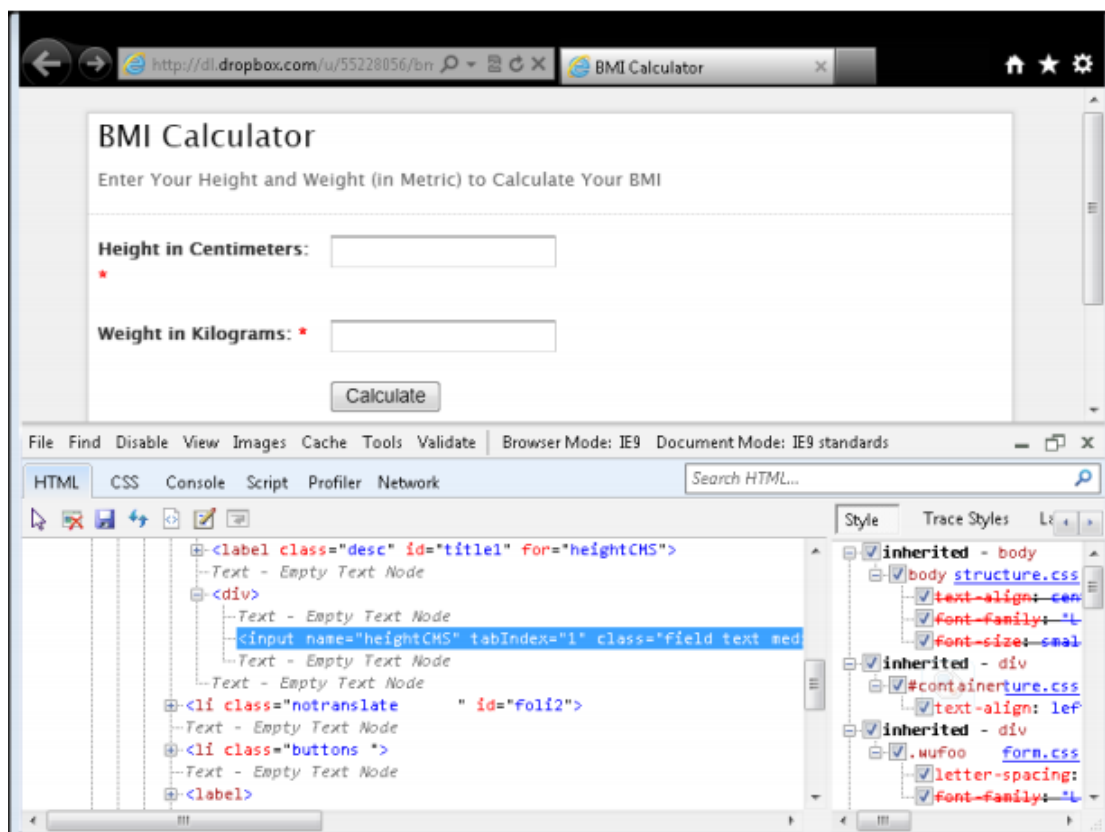
利用 Internet Explorer 检查页面中的元素

类似于 Google Chrome，Microsoft Internet Explorer 也提供了一个内置的分析页面和元素的特性。

按 F12 键打开开发人员工具，如下图所示



检查一个元素,单击指针 () 图标,将鼠标悬停在所需查看的元素上。开发工具高亮此元素为蓝色,HTML 代码将以树型结构显示,如下图所示:



如何实现

浏览器开发工具在测试开发的过程当中派的上用场。这些工具将帮助你找到定位元素，解析代码，以树结构显示出来。这些工具还提供了样式的应用、页面资源，页面的 DOM(文档对象模型)，JavaScript 代码信息等等。

有些工具还可以运行 JavaScript 代码进行测试和调试。

在下面的秘籍中我们将探索 Selenium WebDriver 的各种定位器，这些工具都将帮助你找到和决定使用哪种 Selenium WebDriver API 的定位策略和方法。

1.3. 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。

`findElements()` 方法会返回匹配指定查询条件的 `WebElements` 的集合，如果没有找到则返回为空。

查询方法会将 `By` 实例作为参数传入。Selenium WebDriver 提供了 `By` 类来支持各种查询策略。

下面的表格列出了各种 Selenium WebDriver 支持的定位策略。

策略	语法	描述
By ID	Java: <code>driver.findElement(By.id(<element ID>))</code>	通过元素 ID 属性定位元素
	C#: <code>driver.FindElement(By.Id(<elementID>))</code>	
	Python: <code>driver.find_element_by_id(<elementID>)</code>	
	Ruby: <code>driver.find_element(:id,<elementID>)</code>	
By Name	Java: <code>driver.findElement(By.name(<element name>))</code>	通过元素 Name 属性定位元素
	C#: <code>driver.FindElement(By.Name(<element name>))</code>	
	Python: <code>driver.find_element_by_name(<element name>)</code>	
	Ruby: <code>driver.find_element(:name,<element name>)</code>	

By class name	<p>Java:driver.findElement(By.className(<element class>))</p> <p>C#: driver.FindElement(By.ClassName(<elementclass>))</p> <p>Python: driver.find_element_by_class_name(<elementclass>)</p> <p>Ruby: driver.find_element(:class,<element class>)</p>	通过元素 class name 属性定位元素
By tag name	<p>Java: driver.findElement(By.tagName(<htmltagname>))</p> <p>C#: driver.FindElement(By.TagName(<htmltagname>))</p> <p>Python: driver.find_element_by_tag_name(<htmltagname>)</p> <p>Ruby: driver.find_element(:tag_name,<htmltagname>)</p>	
By link text	<p>Java: driver.findElement(By.linkText(<linktext>))</p> <p>C#: driver.FindElement(By.LinkText(<linktext >))</p> <p>Python: driver.find_element_by_link_text(<linktext >)</p> <p>Ruby: driver.find_element(:link_text,< linktext >)</p>	通过文本定位链接
By partial link text	<p>Java: driver.findElement(By.partialLinkText(<linktext>))</p> <p>C#: driver.FindElement(By.PartialLinkText(<linktext >))</p> <p>Python : driver.find_element_by_partial_link_text(<linktext >)</p> <p>Ruby : driver.find_element(:partial_link_text,<linktext >)</p>	
By CSS	<p>Java: driver.findElement(By.cssSelector(<css selector>))</p> <p>C#: driver.FindElement(By.CssSelector(<cssselector >))</p>	通过 CSS 定位元素

By XPath	Python:	driver.find_elements_by_css_selector (<css selector>)	通过 XPath 定位元素
	Ruby:	driver.find_element(:css,< css selector >)	
	Java:	driver.findElement(By.xpath(<xpath query expression>))	
	C#:	driver.FindElement(By.XPath(<xpath query expression>))	
	Python:	driver.find_elements_by_xpath (<xpath query expression>)	
	Ruby:	driver.find_element(:xpath,<xpath query expression>)	

在这个秘籍中，我们将使用 `findElement()` 方法来定位元素。

如何实现

使用 `id`, `name` 或 `class` 属性是定位元素的首选方法。让我们试试用上面描述的这些方法来定位元素

通过 ID 属性来查找元素

用元素的 `id` 是最首选的方法来定位页面元素。W3C 的标准中推荐开发人员为每一个元素都提供一个独一无二的 `id` 属性。拥有 `id` 属性，就可以提供一个明确可靠的方法来定位页面上的元素。

当处理 DOM 的时候，浏览器使用 `id` 作为首选识别元素的方法，同时这也是最快速的策略。

现在我们看看如何在一个登录表单中使用 `id` 属性来定位元素。

```
<form name="loginForm">
  <label for="username">UserName: </label> <input type="text"
    id="username" /><br/>
  <label for="password">Password: </label> <input
    type="password" id="password" /><br/>
  <input name="login" type="submit" value="Login" />
</form>
```

为定位 **User Name** 和 **Password** 字段，我们可以通过下面方法使用 `id` 属性来定位元素：

```
WebElement username = driver.findElement(By.id("username"));
WebElement password = driver.findElement(By.id("password"));
```

通过 Name 属性来查找元素

使用元素的 `id` 属性来定位是最为推荐的方法，但是你也可能会因为下列原因不能使用 `id` 属性：

- 不是所有的页面上元素都会指定 `id` 属性
- `id` 属性的值是动态生成的

在下面的例子中，登录表单使用了 `name` 属性而不是 `id` 属性

```
<form name="loginForm">
  <label for="username">UserName: </label> <input type="text"
    name="username" /><br/>
  <label for="password">Password: </label> <input
    type="password" name="password" /><br/>
  <input name="login" type="submit" value="Login" />
</form>
```

我们可以使用通过下面的方法使用 `name` 属性来定位元素：

```
WebElement username = driver.findElement(By.name("username"));
WebElement password = driver.findElement(By.name("password"));
```

和 `id` 不同，`name` 属性未必是页面上唯一的属性。你可能会找到多个具有相同 `name` 属性的元素，在这样的情况下，页面上的第一个出现的元素将会被选择，但是个元素未必是你想寻找的，这将会导致测试失败。

通过 CSS 属性来查找元素

除了使用 `id` 和 `name` 属性，你还可以使用 `class` 属性来定位元素。`class` 属性是用来指定元素所应用的 CSS 样式。

在这个例子中，表单元素使用了 `class` 属性而不是 `id` 属性：

```
<form name="loginForm">
  <label for="username">UserName: </label> <input type="text"
    class="username" /><br/>
  <label for="password">Password: </label> <input
    type="password" class="password" /><br/>
  <input name="login" type="submit" value="Login" />
</form>
```

我们可以使用通过下面的方法使用 `class` 属性来定位元素

```
WebElement username =  
    driver.findElement(By.className("username"));  
WebElement password =  
    driver.findElement(By.className("password"));
```

如何实现

Selenium WebDriver 提供了 `findElement()` 方法来定位页面中需要测试的元素。

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。

更多说明

`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。我们第一步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"));
```

NoSuchElementException

`findElement()` 和 `findElements()` 方法当找不到相应的元素的时候就会抛出 `NoSuchElementException` 异常。

1.4. 使用 `findElements` 方法定位元素

Selenium WebDriver 提供了 `findElements()` 方法，可以得到匹配指定规则的集合。当我们需要在一组相似的元素上操作的时候，这个方法会是非常有用的。例如，我们可以得到页面上所有的链接或是表格中所有的行等等。

在这个秘籍中 我们将用 `findElements()` 方法得到所有的链接并打印他们的目标超链接。

如何实现

测试需求以百度首页为例，我们要验证百度首页导航链接的数量，并打印出他们的超链接地址



```
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();
    }
}
```

执行后打印的结果为

```
http://news.baidu.com/
http://tieba.baidu.com/
http://zhidao.baidu.com/
http://music.baidu.com/
http://image.baidu.com/
http://video.baidu.com/
http://map.baidu.com/
http://baike.baidu.com/
http://wenku.baidu.com/
http://www.baidu.com/more/
```

`findElements()` 方法返回所有匹配定位策略的 `WebElement` 的集合, 我们可以使用 `java` 中 `List` 类来创建 `WebElements` 的实例

`List` 类中的 `size()` 方法会告诉你这个集合中元素的总数量。

通过 `for` 循环将得到 `List` 中的所有的元素, 再调用 `getAttribute()` 方法得到元素的属性。

1.5. 定位链接

Selenium WebDriver 提供了多种定位链接的方法。你可以通过链接的名称和部分名称来定位一个链接。

当链接名是动态变化的时候, 使用部分匹配就变得很方便。在这个秘籍中, 我们将看如何使用这些方法来定位页面中的链接。

如何实现

让我们举个例子来看看如何使用 Selenium WebDriver 来定位链接。

通过链接名来定位链接

Selenium WebDriver 的 `By` 类中提供了 `linkText()` 方法来定位链接。在下面的例子中, 我们将定位 Gmail 的链接:

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

通过部分链接名称定位链接

Selenium WebDriver 的 By 类也提供了使用部分链接名来定位链接。当开发人员建立了动态的链接名时这种方法就非常的有用。在这个例子中，有一个链接是用来打开收件箱。这个链接同时也会动态的显示收件箱的数量所以他是在动态变化的过程中。这里我们就可以使用 `partialLinkText()` 来定位固定的或已知不变的一部份名称，在这个例子中就是 "inbox"。

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

如果开发人员提供了 `id`, `name` 或是 `class` 属性，还是尽量使用这些方法来定位。

1.6. 使用标签名称定位元素

Selenium WebDriver 的 By 类中提供了 `tagName()` 方法来定位 HTML 标记名称。这和 `getElementsByTagName()` 很相似。

使用标签名称可以很方便地定位元素。例如，定位所有的表格中的 `<tr>` 等等。

在这个秘籍中，我们将学习如何使用 `tagName` 来定位元素。

如何实现

我们假设你的页面中只有一个按钮。你可以使用他的标签来定位此按钮，方法如下：

```
WebElement loginButton =  
    driver.findElement(By.tagName("button"));  
loginButton.click();
```

另一个例子如果你想统计 `<table>` 中有多少行，你可以这样做：

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

`tagName` 方法查询 DOM，然后返回所有匹配条件的元素。当定位单独的一个元素时，这种方法是不可靠的，页面上可能有很多相同的元素。

1.7. 使用 CSS 选择器定位元素

Cascading Style Sheets (CSS) 是一种样式风格语言用来描述元素的外观和格式。

主流的浏览器实现 CSS 解析引擎使用 CSS 语法来格式化和样式化页面。CSS 的引进是为了让页面信息和样式信息可以分开。更多的 CSS 信息和 CSS 选择器请访问 http://en.wikipedia.org/wiki/Cascading_Style_Sheets.

Selenium WebDriver 使用同样的 CSS 选择器的原则来定位 DOM 里的元素。这是一个相对 XPath 更可靠更快速的方式来定位复杂的元素。

在这个秘籍中,我们将探索一些基本的 CSS 选择器,后面我们会逐步深入到更高级的用法。

如何实现

让我们探索一些基本的 CSS 选择器。Selenium WebDriver 的 By 类提供了 `cssSelector()` 方法,让我们可以使用 CSS 选择器来定位元素。

使用绝对路径来定位元素

CSS 绝对路径指的是在 DOM 结构中具体的位置。下面一个例,使用绝对路径来定位用户名输入字段。在使用绝对路径的时候,每个元素之间要有一个空格。

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

你也可以以父子关系的方式“>”来描述这个选择器

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

但是,这个策略会有一些的限制,他取决于页面的整个结构。如果有些许改变,选择器将找不到这个元素。

使用相对路径来定位元素

使用相对路径的时候我们可以直接定位元素。不用考虑他在 DOM 中的位置。例如,我们可以用这样的方法来定位用户输入字段,假设他在 DOM 中是第一个<input>元素:

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

使用相对路径来定位元素

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"));
```

这同样可以找到按钮的<input>标签 class 为 login 的元素。

你还可以简写查询表达式，只用 . 和 class 属性值，省略掉 HTML 的标签。但是，这将会返回所有 class 为 login 的元素，导致结果并不一定是你所期望的那样。

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

此方法和 className() 很相似。

使用相对 ID 选择器来定位元素

我们也可以使用元素的 ID 来定位。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"));
```

这将会返回 input 标签中 id 为 username 的元素。

你可以通过这样来简化一下表达式，输入 “#” 符号，跟上 id 的名称即可，省略掉 HTML 的标签。但是，这也将会返回所有 id 为 username 的元素，结果未必是你所期望。用的时候要非常小心。

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

这个方法和 id 选择器策略很像。

使用属性来定位元素

除了 class 和 id 属性，CSS 选择器也可以使用其他的元素属性来定位。下面的例子中，将使用<input>中的 Name 属性。

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

使用 name 属性来定位元素和直接用 By 类中的 name() 方法来定位很相似。

让我们试试使用其他的属性，下面的例子中，命名 alt 属性来定位元素。

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


你可以会遇到一个属性不足以来定位到一个元素的情况,你需要联合使用其他的属性来达到精确匹配。下面的例子中,使用多个属性来定位<input>元素。

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

使用属性名称选择器来定位元素

这个策略和之前的有些不同,我们只通过指定元素中属性的名称而不是属性的值来定位元素。例如,我们想要查找所有标签中,含有 alt 属性的元素。

```
List<WebElement> imagesWithAlt =
driver.findElements(By.cssSelector("img[alt]"));
```

not() 伪类也可以使用来匹配不满足规则的元素。例如,想要定位那些标签中不含有 alt 属性,方法如下:

```
List<WebElement> imagesWithoutAlt =
driver.findElements(By.cssSelector("img:not([alt])"));
```

部分属性值的匹配

CSS 选择器提供了一个部分属性值匹配定位元素的方法。这为了测试那些页面上具有动态变化的属性的元素是非常有用的。例如,在 ASP.NET 应用中,元素 id 是动态生成的。下面的表格介绍了如何使用部分匹配的语法:

语法	例子	描述
^=	Input[id^= ' ctrl1']	以 XXX 开始 例如,如果一个元素的ID是ctrl_12,就可以定位到此元素,匹配到 id 的头部 ctrl。
\$=	input[id\$='_userName']	以 XXX 结尾 例如,如果一个元素的 ID 是 a_1_userName,这将会匹配到 id 的尾部_userName。
=	Input[id='userName']	包含 例如,如果一个元素的 ID 是 panel_login_userName_textfield,这将会匹配到此 id 值的_userName,从而定位到元素。

CSS 选择器是 CSS 匹配 HTML 或 XML 一系元素中的规则中的一个模式和部分。

主流的浏览器都支持对 CSS 的解析，应用到相应的元素上去。Selenium WebDriver 使用 CSS 解析引擎来定位页面上的元素。CSS 选择器提供多样的方法，规则和模式来定位页面上的元素。这也是相对我们后面说的 XPath 更加稳定和快速的方法。

使用 CSS 选择器，可以通过多样的方法来定位元素如 Class , ID , 属性值和在此秘籍中我们描述的一些文本内容方法。

1.8. 使用 XPath 定位元素

XPath 是 XML 路径语言，用来查询 XML 文档里的节点。主流的浏览器都支持 Xpath, 因为 HTML 页面在 DOM 中表示为 XHTML 文档。

Xpath 语言是基于 XML 文档的树结构，并提供了浏览树的能力，通过多样的标准来选择结点。

Selenium WebDriver 支持使用 Xpath 表达式来定位元素。

利用 Xpath 来定位元素非常方便，但是，便捷的定位策略牺牲了系统的性能。

XPath 和 CSS 中最重要的区别在于 Xpath 可以向前和向后查询 DOM 结构的元素 而 CSS 只能向前查询。这意味着使用 XPath 可以通过子元来定位父元素。

在这个秘籍中，我们将探索一些基本的 XPath 查询来定位元素然后再学习一些 XPath 的高级应用。

如何实现

让我们探索一些 Selenium WebDriver 中基本的 XPath 表达式。Selenium WebDriver 提供了 `xpath()` 方法来定位元素。

通过绝对路径定位元素

和 CSS 绝对路径相似，XPath 绝对路径适用于指定元素的位置。这里的一个例子就是使用绝对路径来定位用户名的字段。在每一个元素之间需要有一个空格。

```
WebElement userName =
```

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

但是, 这个策略有局限性, 他需要参考整个页面的文档结构。如改变了, 此元素的定位将会失败。

通过相对路径定位元素

用相对路径, 我们可以直接找到元素而不管其在 DOM 中的位置。例如, 我们可以通过如下方法来定位用户名字段, 假设这个<input>元素处于 DOM 中第一个:

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

使用索引来定位元素

在前面的示例中, XPath 查询将返回第一个 DOM 中<input>元素。可能会有多个元素都匹配了 XPath 查询。如果元素不是第一个元素, 我们也可以指定他的个数来找到它。例如在我们的登录表单, 我们可以找到密码字段 - 第二个<input>, 方法如下:

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

使用 XPath 及属性值定位元素

和 CSS 相似, 我们可以在 XPath 中使用元素的属性来定位元素。在下面的例子中, 可以使用 ID 属性来定位用户名字段。

```
WebElement userName =  
    driver.findElement(By.xpath("//input[@id='username']"));
```

另一个使用 alt 属性来定位 image 属性的例子:

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

你可以会遇到一个属性不足以来定位到一个元素的情况, 你需要联合使用其他的属性来达到精确匹配。下面的例子中, 使用多个属性来定位<input>元素。

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

使用 XPath 和 and 操作符也同样可以达到相同的效果

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

下面的例子中, 使用 or 操作符任何一个属性满足也将可以对元素进行定位

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

使用 XPath 及属性名称定位元素

这个策略和之前的有些不同,我们只通过指定元素中属性的名称而不是属性的值来定位元素。例如,我们想要查找所有标签中,含有 alt 属性的元素。

```
List<WebElement> imagesWithAlt = driver.findElements
    (By.xpath ("img[@alt]"));
```

部分属性值的匹配

类似于 CSS 选择器, XPath 还提供了一种一些方法部分匹配属性来定位元素。这对于网页中的属性是动态变化的时候是非常有用的。例如, ASP.NET 应用程序中动态生成 id 属性值。下面的表说明了使用这些 XPath 功能:

语法	例子	描述
starts-with()	input[starts-with(@id, 'ctrl_12')]	例如, 如果元素的 ID 为 ctrl_12, 将会匹配以 ctrl 开始的属性值。
ends-with()	input[ends-with(@id, '_userName')]	例如, 如果元素的 ID 为 a_1_userName, 将会匹配以 userName 结尾的属性值。
contains()	Input[contains(@id, 'userName')]	例如, 如果元素的 ID 为 panel_login_userName_textfield, 将会匹配含有 userName 属性值。

使用值来匹配任意属性及元素

XPath 可以匹配任意元素属性中指定的值。例如, 在下面的 XPath 查询中, "userName" 是指定的。XPath 将会检查所有元素中是否有属性等于"userName", 并将其返回。

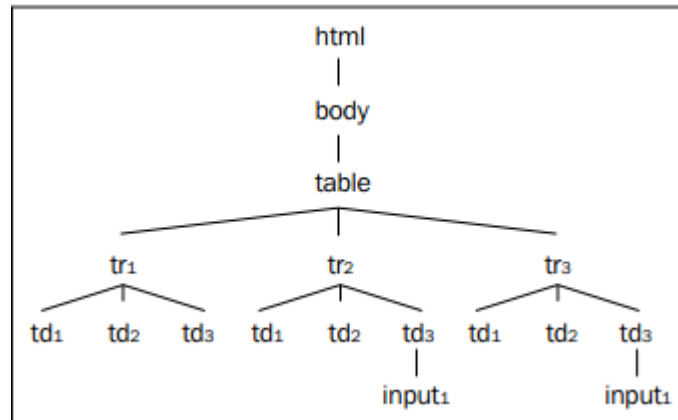
```
WebElement userName =
    driver.findElement(By.xpath("//input[@*='username']"));
```

使用 XPath 轴来定位元素

XPath 轴是借助于文档中元素与元素之间的关系来定位。下面有一个简单的<table>的 XPath 轴的例子。

Product	Price	Qty
Product 1	\$100	12
Product 2	\$150	5

下面是用图形来表示 HTML 元素间的关系



轴	描述	例子	结果
ancestor	选择当前节点所有的父类元素,包括祖先元素	<code>//td[text()='Product 1']/ancestor::table</code>	得到 table 元素
descendant	选择当前节点所有子元素	<code>//table/descendant::td/input</code>	得到第三例第二行的 input 元素
following	选择当前元素结束标签后的所有元素	<code>//td[text()='Product 1']/following::tr</code>	得到第到第二行的 tr 元素
following-sibling	选择当前元素后的兄弟元素	<code>//td[text()='Product 1']/following-sibling::td</code>	得到第二行第二列的 td 元素
preceding	选取文档中当前节点的开始标签之前的所有节点	<code>//td[text()=' \$150']/preceding::tr</code>	得到第一行 tr
preceding-sibling	选取当前节点之前的所有同级节点。	<code>//td[text()=' \$150']/preceding-sibling::td</code>	得到第三行第一列的 td

更多关于 XPath 轴请访问 http://www.w3schools.com/xpath/xpath_axes.asp。
Xpath 是处理、查询浏览器 DOM 强大的语言,可以浏览 DOM 中的元素和属性。XPath 也提供了一些规则、函数和语法来定位元素。

主流的浏览器都支持 XPath, Selenium WebDriver 也提供了通过 XPath 来定位元素的能力。

使用 `By` 类中的 `Xpath()` 方法可以定位元素。XPath 的查询是慢于 CSS 选择器，因为 XPath 支持双向的查询。你可以通过元素的父，兄弟，子节点来定位元素。

1.9. 使用文本元素

当你测试网页应用的时候，你也可能遇到开发人员没有分配任何属性给元素，这时候定位元素就会变得很困难。

使用 CSS 选择器或 XPath，我们可以使用他的本文内容来定位元素。在这个秘籍中，我们将探索如何使用文本值来定位元素。

如何实现

使用文本来定位元素，CSS 选择器和 XPath 提供了方法来通过文本定位元素。如果一个元素包含一个指定的文本，将会返回到测试。

使用 CSS 选择器伪类定位元素

CSS 选择器提供了 `contains()` 伪类来通过指定文本定位元素。例如，你可以通过表格里的内容来定位单元格，方法如下：

```
WebElement cell =  
    driver.findElement(By.cssSelector("td:contains('Item 1')"));
```

`contains()` 伪类接受一个被查询的文本作为参数。然后查询所有 `<td>` 标签里包含指定文本的元素。（注：`contains()` 方法已经被 CSS3 弃用了）



以百度首页为例，现在我想定位新闻链接

```
WebElement news =  
    driver.findElement(By.cssSelector("a:contains('新')"));
```

作为 `contains()` 的替代, 你可以使用 `innerText` 属性 (不支持 Firefox) 或 `textContent` 属性 (支持 Firefox), 方法如下:

```
WebElement news =  
    driver.findElement(By.cssSelector("a[innerText='新闻']"));
```

或

```
WebElement news = driver.findElement  
    (By.cssSelector("a[textContent='新闻']"));
```

使用 XPath 的 text 函数

XPath 提供了 `text()` 方法来定位指定文本的元素, 方法如下:

```
WebElement cell = driver.findElement  
    (By.xpath("//td[contains(text(),'Item 1')]"));
```

这里 `contains()` 与 `text()` 函数一起使用。`text()` 函数, 返回完整的文本, `contains()` 函数将检查是否包含此文本。

使用 XPath 精确文本定位元素

利用 XPath, 通过精确的文本来定位元素, 方法如下:

```
WebElement cell = driver.findElement  
    (By.xpath("//td[.='Item 1']"));
```

CSS 选择器和 XPath 提供基于元素文本内容方法来定位元素。这种当元素没有足够的属性或当没有其他策略时会很方便找到这些元素。

对于使用文本定位, 无论是 CSS 选择器还是 XPath 都是搜索整个 DOM, 返回匹配的元素。

1.10. 使用高级的 CSS 选择器定位元素

我们在之前学习了一些基本的 CSS 选择器, 在此秘籍中, 我们将探索用一些高级的 CSS 选择器来定位元素

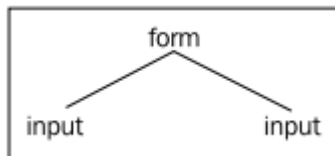
如何实现

在 1.7 章节中, 我们已经探索了一些基本的 CSS 选择器, 让我们探索一些高级的 CSS 选择器, 如相邻的兄弟结合伪类的使用。

查询子元素

CSS 选择器提供了多种方法通过父元素来定位它的子元素。还是之前的例子

```
<form name="loginForm">
  <label for="username">UserName: </label> <input type="text"
    class="username" /></br>
  <label for="password">Password: </label> <input
    type="password" class="password" /></br>
  <input name="login" type="submit" value="Login" />
</form>
```



如果想定位表单中的用户名输入框，我们使用在其父子元素间使用 “>” 符号。

```
WebElement userName = driver.findElement
    (By.cssSelector("form#loginForm > input"));
```

相似的使用 `nth-child()` 方法也可以定位成功，方法如下：

```
WebElement userName = driver.findElement
    (By.cssSelector("form#loginForm :nth-child(2)"));
```

下列表格列出了使用伪类来定位子元素的例子：

伪类	例子	描述
<code>:first-child</code>	<code>form#loginForm :first-child</code>	定位表单里第一个子元素 username 标签
<code>:last-child</code>	<code>form#loginForm :last-child</code>	定位表单最后一个子元素 Login 按钮
<code>:nth-child(2)</code>	<code>form#loginForm :nth-child(2)</code>	定位表单中第二个子元素 username 的输入框

查询兄弟元素



利用 CSS 选择器，我们可以使用 “+” 操作符来定位兄弟元素。还是以百度首页为例，可以使用下列方法来定位百度导航中的“网页”超链接

```
WebElement web =  
    driver.findElement(By.cssSelector("#nv a + b"));
```

#nv a 定位到“新闻”链接，“+ b”后就找到其兄弟元素

使用用户操作伪类

使用用户的操作行为: focus 伪类，定位焦点在 input 框中的元素，方法如下：

```
WebElement productDescription =  
    driver.findElement(By.cssSelector("input:focus"));
```

你也可以使: hover 和: active 伪类来定位元素

使用 UI 状态伪类

使用 UI 状态伪类，我们可以通过元素的各种状态来定位，如 enabled, disabled, checked。下例表格给予了详细的说明

伪类	例子	描述
:enabled	input:enabled	定位所有属性为 enable 的 input 的元素
:disabled	input:disabled	定位所有属性为 disabled 的 input 的元素
:checked	input:checked	定位所有多选框属性为 checked 的元素

访问 http://www.w3schools.com/cssref/css_selectors.asp 查询更多 CSS 选择器的用法

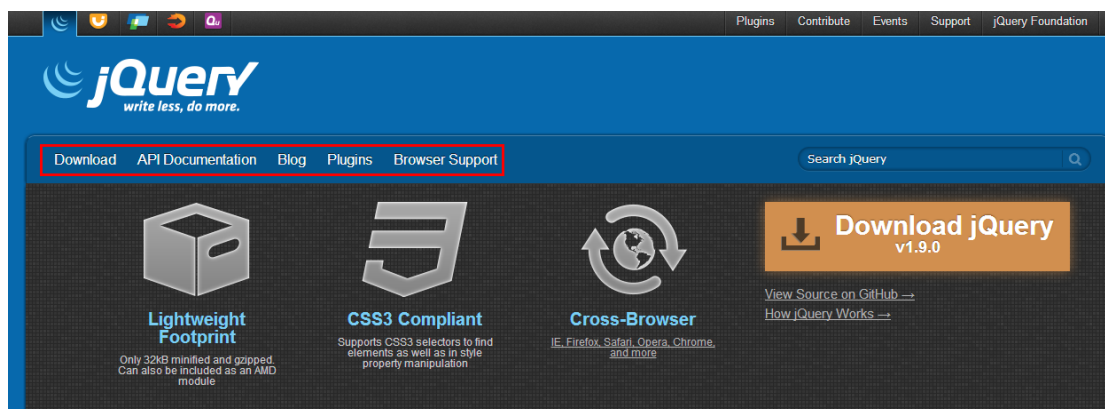
1.11. 使用 jQuery 选择器

jQuery 选择器是 jQuery 库中非常重要的功能。jQuery 选择器是基于 CSS1-3 选择器，加上一些额外的选择器。这些选择器和 CSS 选择器的使用方法很相似，允许开发人员简单快速的识别页面上的元素。同样可以定位 HTML 中的元素作为一个单独的元素或是一个元素集合。

jQuery 选择器可以使用在那些不支持 CSS 选择器的浏览器上。

在这个秘籍中，我们将简单的探索如何在 Selenium WebDriver 中使用 jQuery 选择器

如何实现



以 jQuery 官网为例，网站本身已经自动加载了 jQuery 的库，所以我们可以脚本中直接使用 jQuery 语法。下面的例子中我们想使用 jQuery 的选择器：even 选出右侧导航栏中偶数(第一位是索引是 0)超链接 **Download**、**Blog**、**Browser Support**

```
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.jquery.com/");
```

```
List<WebElement> elements =
    (List<WebElement>)jse.executeScript
        ("return jQuery.find" +
         "('.menu-item a:even')");
assertEquals(3,elements.size());
assertEquals("Download",elements.get(0).getText());
assertEquals("Blog",elements.get(1).getText());
assertEquals("Browser Support",elements.get(2).getText());
driver.close();
}
}
```

Selenium WebDriver 使用 jQuery API 增强了 jQuery 选择器，但是，我们需要确认页面已经加载了 jQuery。jQuery API 提供了 `find()` 方法来查询元素。我们需要使用 `JavaScriptExecutor` 类来执行 jQuery 的 `find()` 方法。

`find()` 方法返回了符合查询条件的元素集合。更多 jQuery 选择器信息请查询 <http://api.jquery.com/category/selectors/>。

在使用 jQuery 选择器的时候，有的页面并未加载 jQuery 库，这时候你可以在加载页面的时候注入 jQuery 库，方法如下：

```
private void injectjQueryIfNeeded() {
    if (!jQueryLoaded())
        injectjQuery();
}
public Boolean jQueryLoaded() {
    Boolean loaded;
    try {
        loaded = (Boolean) driver.executeScript("return
            jQuery() != null");
    } catch (WebDriverException e) {
        loaded = false;
    }
    return loaded;
}
public void injectjQuery() {
    //在 head 中拼出加载 jquery 的 html
    driver.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);");
}
```



[新闻](#) [网页](#) [贴吧](#) [知道](#) [音乐](#) [图片](#) [视频](#) [地图](#) [百科](#) [文库](#) [更多>>](#)

百度一下

还是以百度首页为例，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();
    }
    //判断是否已加载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，你可以修改例子中的版本，使用最新的 jQuery 版本。

1.12. 定位表格行和单元格

在处理表格时，我们可以通过 By 类中的一些方法快速有效的来定位表格的行和单元格。在这个秘籍中，我们将学习如何定位表格中的行和列。

如何实现

让我们创建一个简单的测试来打印表格中的数据，通过下面的方法来定位表格的行和列：

以 W3C School 的网页为例 http://www.w3school.com.cn/html/html_tables.asp
我们的需求是先判断一下这个表格是多少行，然后将每一个单元格的数据打印出来。

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

}

1.13. 定位表格中的子元素

对于简单的表格处理还相对容易些。但是，你可能会遇到相对复杂的表格，表格单元格中还有子元素要和用户进行交互。例如，在一个电子商务网的购物车页面，表格中内嵌了很多复杂的元素。

Your Shopping Basket
You have selected the following products so far:

Product	Price incl. VAT	Quantity	Value incl. VAT
 Large Bags James Wellbeloved + JWB USB Stick Free! - Junior Large Breed Turkey & Rice (15 kg) 315777.29	Was £47.89 Now £42.90	<input type="text" value="1"/> X ↺	£42.90
Subtotal			£42.90
Discount Only one coupon can be accepted with each order.			10% New Customer Discount change coupon -£4.29
Shipping Fees Please choose a delivery country so that we can calculate your shipping costs for you.			<input type="text" value="Great Britain"/> £0.00
Grand Total All prices include tax.			£38.61

[back to shop](#)[proceed to checkout](#)

此外,这些元素是基于用户操作动态生成的。定位这些元素是一个比较有挑战的任何。

在此秘籍中，我们将探索使用 CSS 和 XPath 定位表格的子元素。

如何实现

这里有一个表格的例子，列出了系统用户和各自的权限。我们要创建一个测试选择相应的多选框并检查选择的东西是否授予给了用户。

GMail Inbox (10 New E-mails)		
User Name	E-mail	Access
Nash	Nash@test.com	Admin <input checked="" type="checkbox"/> Content Manager <input type="checkbox"/> Browser <input type="checkbox"/>
John	John@test.com	Admin <input type="checkbox"/> Content Manager <input type="checkbox"/> Browser <input checked="" type="checkbox"/>

数据中每一行代码如下：

```
<tr>
  <td>Nash</td>
  <td><a href="mailto:nash@test.com">Nash@test.com</a></td>
  <td>
    <div>
      <label for="user128_admin">Admin</label>
      <input type="checkbox" id="user128_admin"
        checked="true"/>
      <label for="user128_cm">Content Manager</label>
      <input type="checkbox" id="user128_cm"/>
      <label for="user128_browser">Browser</label>
      <input type="checkbox" id="user128_browser"/>
    </div>
  </td>
</tr>
```

多选框中的 ID 是动态的不能和用户关联,但是我们处理这样的问题可以使用 CSS 选择器和 XPath。在这个例子中,我们想要给 nash 用户授予 admin 的权限。这可能通过 CSS 选择器来实现,方法如下：

```
WebElement adminCheckBox = driver.findElement
  (By.cssSelector("td:contains('Nash')+td>div>label:contains
    ('Admin')+input"));

adminCheckBox.click();
```

我们也可以使用 XPath 来实现：

```
WebElement adminCheckBox = driver.findElement
  (By.xpath("//td[contains(text(),'Nash')]/following-
    sibling::td/descendant::div/label
    [contains(text(),'Admin')]/following-sibling::input"));

adminCheckBox.click();
```

父,子,兄弟元素,使用 CSS 或是 XPath 方法将对于关联用户设计出通用的定位策略有非常大的帮助。简言之,这个策略就是基于元素与元素之间的关系进行定位。

回到我们的问题,首先我们需要找到一个唯一方法来识别表中的用户。为此,我们可以使用单元格中的用户名来定位,方法如下:

CSS
td:contains('Nash')

XPath
//td[contains(text(),'Nash')]

下一步，我们需要找到子元素的单元格，对于用户名来说是后面的第 2 个单元格

CSS	XPath
<code>td:contains('Nash')+td+td</code>	<code>//td[contains(text(),'Nash')]/following-sibling::td/descendant::div</code>

下一步，就要定位到正确的多选框

CSS	XPath
<code>td:contains('Nash')+td+td>div>label:contains('Admin')+input</code>	<code>//td[contains(text(),'Nash')]/following-sibling::td/descendant::div/label[contains(text(),'Admin')]/following-sibling::input</code>

2.使用 Selenium API

在本章中，我们将要讨论

- ◆ 检查元素的文本
- ◆ 检查元素的属性值
- ◆ 检查元素的 CSS 属性值
- ◆ 针对鼠标和键盘事件使用高级的用户交互 API
- ◆ 在元素上执行双击操作
- ◆ 执行拖拽操作
- ◆ 执行 JavaScript 代码
- ◆ 使用 Selenium WebDriver 进行截图
- ◆ 使用 RemoteWebDriver/Grid 进行截图
- ◆ 将浏览器窗口最大化
- ◆ 自动选择下拉列表
- ◆ 检查下拉列表中的选项
- ◆ 检查下拉菜单和列表中选中的选项
- ◆ 自动选择单选按钮
- ◆ 自动选择多选项框
- ◆ 处理 windows 的进程
- ◆ 通过 WebDriver 中阅读 windows 注册表中的值
- ◆ 通过 WebDriver 修改 windows 注册表的值

2.1 简介

Selenium WebDriver 实现了一些操作元素复杂的 API，高级的用户交互，执行 JavaScript 代码，并支持控制各种类似的列表，下拉框，单选框，多选择。

在本章中，我们要探索如何使用这些 API 创建一些复杂的测试步骤。本章也会解决一些在使用 Selenium WebDriver 创建测试时一些常见的问题。本章的例子是使用 Java 的写。

2.2 检查元素的文本

当测试一个网页应用的时候，我们需要验证元素显示的是否正确。Selenium WebDriver API 提供了多种方法来得到和验证文本。有时候，我们需要先将元素的实时的文本值保存下来以便测试后面的某些地点使用。

在此秘籍中，我们要使用 WebElement 类中的 `getText()` 方法来获得和验证元素文本。

如何实现

这里，我们创建了一个测试，先定位元素再将它的文本保存到一个字符串变量中。我们将验证字符串的内容是否正确。

```
@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 my color 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

我们也可以使用 Java String 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"));
```

2.3 检查元素的属性值

开发人员通过配置各种元素属性来控制元素行为或样式。例如，`<input>` 元素可以被设置为 `readonly` 属性。

我们需要创建一个测试来验证元素属性是正确的。我们可以使用 `WebElement` 类的 `getAttribute()` 方法来获得元素的属性。

这此秘籍中，我们需要使用 `getAttribute()` 方法来检查元素的属性。

如何实现

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"));
}
```

此例子中我们验证了元素的属性 `align` 的值是否为 `justify`。

2.4 检查元素的 CSS 属性值

不同的样式应用在页面的元素上，让网页变得更加丰富和美丽。开发人员使用 CSS 来增加这些样式。测试人员就需要验证那些样式是否正确应用到元素上面。这可以通过 `WebElement` 类中的 `getCSSValue()` 方法实现，方法将返回指定样式属性的值。

在此秘籍中，我们将要使用 `getCSSValue()` 方法来检查元素的样式属性。

如何实现

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);
}

```

2.5 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。

在此秘籍中，我们将使用 `Actions` 类来创建一个事件链来选择表格中的一行。

如何实现

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(tableRows.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。`Actions` 不能在隐藏的元素上进行操作，所以如果要使用，确保元素是可见的。

(想吐槽一下这个例子，仅仅是一个方法，也没有实例看看，况且一般人选表格行会用 `ctrl+点击`，貌似这样也选不中啊。)

2.6 在元素上执行双击操作

在网页应用中，有可能出现要对某一元素进行双击才能触发其相应的事件。比如，在某一个表格中双击启动一个新的窗口。在高级用户交互中提供了来执行双击操作的 API。

在此秘籍中我们将使用 `Actions` 类来执行双击操作。

如何实现

```
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()` 来模拟真实的双击动作，并验证了字体属性说明确实有效了。

2.7 执行拖拽操作

Selenium `WebDriver` 在 `Actions` 类中也实现了 Selenium RC 里的 `dragandDrop` 命令。我们之前看到了 `Actions` 类支持高级的用户交互操作，如鼠标和键盘的事件。通过这个类我们可以建立一系列的动作链。

在此秘籍中，我们使用 `Actions` 类来执行元素的拖拽。

如何实现

让我们使用 `Actions` 类来实现一个拖拽操作的测试。

```
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.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，我们需要先定位这些元素（原元素，目标元素）然后作为参数传给 `dragAndDrop()`。

2.8 执行 JavaScript 代码

Selenium WebDriver API 提供了执行 JavaScript 代码的能力。当测试需要使用 JavaScript 进行交互的时候，这将会非常的有用。利用这个 API，客户端的 JavaScript 代码可以通过 Selenium WebDriver 来执行。Selenium WebDriver 提供了 `JavascriptExecutor` 接口，可以任意执行 JavaScript 代码。

在这个秘籍中，我们将探索如何使用 `JavascriptExecutor` 来执行 JavaScript 代码。本书还有一些其它的秘籍也使用了 `JavascriptExecutor` 来执行一些高级的 Selenium WebDriver 目前不支持的操作

如何实现

让我们建立了一个测试调用 JavaScript 代码来返回网页的 `title` 并统计超链接的数量。

这可以通过调用 `driver.getTitle()` 方法。

```
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。基于返回值的类型，我们需要对 `executeScript()` 方法进行转型。对于带小数点的值，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，如果返回的是 HTML 节点，可以使用 `WebElement` 类型，文本值，可以使用 `String` 类型。如果返回的是对象列表，基于对象类型的任何值都可以。

2.9 使用 Selenium WebDriver 进行截图

Selenium WebDriver 提供了 `TakesScreenshot` 接口来捕捉网页的截屏。这可以在测试执行遇到异常或错误的时候将屏幕截取下来,可以知道当时发生了什么。我们也可以在验证元素状态,显示的值或是某个操作完成后的状态进行截屏。

截屏也可以帮助我们验证布局,字段对齐,图片的对比。

在此秘籍中,我们将使用 `TakesScreenshot` 接口来捕捉测试运动时的截屏。

如何实现

让我们创建一个测试,打开一个页面应用进行截屏并保存 PNG 格式。

```
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 `getScreenshotAs()` 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。

`OutputType` 类提供了多种数据类型，在前面的例子中我们使用了文件类型，它也可以保存为 Base64 编码的字符串，代码如下

```
String base64 =  
    ((TakesScreenshot) driver).getScreenshotAs(OutputType.BASE64);
```

2.10 使用 RemoteWebDriver/Grid 进行截图

当测试运行在 `RemoteWebDriver/Grid` 时就不能使用 `TakeScreenshot` 接口来进行截图。

但是我们可以使用 `TakesScreenshot` 接口的 `Augmenter` 类来截取 `RemoteWebDriver` 中的屏幕

如何实现

创建一个 `RemoteWebDriver` 的测试。

加入下列代码到测试中去：

```
driver = new Augmenter().augment(driver);  
File scrFile =  
    ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);  
FileUtils.copyFile(scrFile, new File("D:\\screenshot.png"));
```

`Augmenter` 类通过增加了多种接口，包括 `TakesScreenshot` 接口增加了 `RemoteWebDriver`。

```
package com.example.tests;  
import java.io.File;  
import java.io.IOException;  
import java.net.MalformedURLException;  
import java.net.URL;  
import org.apache.commons.io.FileUtils;  
import org.junit.*;  
import org.openqa.selenium.*;  
import org.openqa.selenium.remote.*;  
public class test {
```

```

@Test
public void testRemoteWebDriverScreenShot() {
    //指定使用的浏览器
    DesiredCapabilities capability =
        DesiredCapabilities.internetExplorer();
    WebDriver driver = null;
    try {
        driver = new RemoteWebDriver( //我使localhost地来测试
            new URL("http://localhost:4444/wd/hub"), capability);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    driver.get("http://www.sina.com.cn");
    //对远程系统进行截图
    driver = new Augmenter().augment(driver);
    File scrFile =
        ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
    try {
        FileUtils.copyFile(scrFile, new File("D:\\screenshot.png"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

2.11 将浏览器最大化

在 Selenium RC 中我们可以使用 `windowMaximize()` 方法来最大化浏览器。从 2.21 版本后，Selenium WebDriver 也可以将浏览器最大化了。

如何实现

通过调用 `maximize()` 方法可以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();
}
}
```

2.12 自动选择下拉列表

Selenium WebDriver 对于下拉列表可以使用指定的 `Select` 类替代 `WebElement` 类。

`Select` 类提供了多种方法和属性来和 HTML 中的 `<select>` 元素进行交互。

在这个秘籍中，我们将要使用 `Select` 类来操作下拉列表。

如何实现

```
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();
    }
}
```

创建另一个多选下拉框的测试例子,执行一些基本的检查然后调用方法进行下拉列表的多选操作,验证所选的选项然后调用取消选项的方法,取消已选的选项。

```
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);
    }
}
```

在执行上面的例子时需要导入 `org.openqa.selenium.support.ui.Select` 类。首先创建一个 `Select` 的对象, `isMultiple()` 用来判断是不是多选下拉框。 `Select` 类提供了 3 种方法来选择下拉选项。 `selectByVisibleText()`, `selectByValue()`, `selectByIndex()`。在使用这些方法的时候要注意下拉列表是不是动态变化的。

2.13 检查下拉列表中的选项

在之前的秘籍中,我们知道了如何选择下拉列表中的选项,并做了一些验证的工作,我们还需要验证被选中的选项是不是正确的,无论是默认的还是用户选择的。

在这个秘籍中，我们将要学习如何检查选中的下拉列表的选项。

如何实现

在之前的例子中，我们需要加一些步骤，所以对上面的例子做一些修改（加粗部分为新增）

修改 `testDropdown()`

```
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());
    }
}
```

同样的修改 `testMultipleSelectLis()`

```
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;
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
        assertEquals
            (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。

2.14 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 `isSelected()` 方法来判断是否选中了单选按钮。

在此秘籍中，我们将学习如何处理单选按钮和按钮组

如何实现

```
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;
        }
    }
}
}
}

```

2.15 自动选择多选框

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 `isSelected()` 方法来判断是否选中。

在此秘籍中，我们将学习如何处理多选框。

如何实现

```

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());
}
}
```

2.16 处理 windows 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。

在此秘籍中, 我们需要使用 `WindowsUtils` 类来关闭已经打开的浏览器。

如何实现

```
@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。

2.17 通过 WebDriver 中阅读 windows 注册表中值

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。

在此秘籍中，我们将要使用 WindowsUtil 读取测试运行的操作系统的名称。我们也许会将此信息打印在测试日志里。

如何实现

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);
    }
}
```

似乎并不是所有的注册表都可以读取，下面注册表的值就在读取的时候就会报异常。

```
HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\WindowsNT\\CurrentVersion\\P
roductName
```

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。

2.18 通过 WebDriver 中修改 windows 注册表中值

WindowsUtil 类也提供了修改或创建 windows 注册表值的方法，同读取注册表一样，也提供了多种方法来修改键值。

在此秘籍中，我们将要使用 WindowsUtil 类来创建一个新的注册表键值。

```
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()。

3. 测试流的控制

在本章中，我们将讨论

- ◆ 使用隐式的等待同步测试
- ◆ 使用显式的等待同步测试
- ◆ 使用自定义的期望条件同步测试

- ◆ 检查元素是否存在
- ◆ 检查元素的状态
- ◆ 通过名称识别和处理一个弹出窗口
- ◆ 通过标题识别处理一个弹出窗口
- ◆ 通过网页内容识别处理一个弹出窗口
- ◆ 处理一个简单的 JavaScript 警告窗
- ◆ 处理一个确认框
- ◆ 处理一个提示框
- ◆ 识别处理框架
- ◆ 通过页面内容识别和处理框架
- ◆ 处理 IFRAM

3.1. 简介

在用 Selenium WebDriver 构建一个复杂的 web 应用程序自动化测试的时候，我们需要确保测试流是稳定的，可维护的自动化测试。

在运行测试时，测试可能并不总是以相同的速度响应。例如，可能需要几秒后等待进度条到 100% 时，状态消息出现，一个按钮变成可点击的状态，或是打开一个提示窗口或警告。你可以通过同步你的测试来处理这些有预期条件的问题，确保 Selenium WebDriver 等待到一定时机再执行下一步操作。你可以使用不同的方法来同步你的测试。Selenium RC 有多种的 `waitFor` 命令，但是作为一个纯 Web 自动化的 API，Selenium WebDriver 提供了有限的方法来同步。在本章中，你将要学习如何使用 `WebDriverWait` 类来执行同步。

弹出窗和警告框

为建立一个良好的用户界面，开发人员需要使似于桌面应用程序中的弹出窗口和警报。在测试复杂的工作流时，测试需要通过一些弹窗和警告来完成业务流的工作。本章还说明了一些处理弹出窗口和警告的常见问题，和如何更好的使用 Selenium WebDriver API。

3.2. 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执行的时间。

在本秘籍中，我们将简单的探索如何使用隐式等待，但是还是要避免使用或是减少使用隐式等待。

如何实现

```
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的时间参数。在这个例子中，我们需要等待 10 秒。

`Thread.sleep(4000)` 意思义等待 4 秒钟，如果没有这句加上 `pageContent` 本身就存在，所以它的内容就是最初没有点击时候的文本，不是我们所期望的。可见这样的方法本不是很好。所以尽量去使用显示的等待，提供了更多的方法来精确的控制同步。

3.3. 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，自定义等待的条件。

显式的等待只需要执行在需要同步的地方而不影响脚本其他的地方。

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的条件。

预定义条件	方法名
元素可见可点击	<code>elementToBeClickable(By locator)</code>
元素被选中	<code>elementToBeSelected(WebElement element)</code>
存在一个元素	<code>presenceOfElementLocated(By locator)</code>
元素中出现指定的文本	<code>textToBePresentInElement(By locator, String text)</code>
元素的值	<code>textToBePresentInElementValue(By locator, String text)</code>
标题	<code>titleContains(String title)</code>

更多的信息请访问：

<http://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html>

修改之前的例子：

```
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-we
bsite-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。
可见这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的条件。上例中如果文本出现也证明我们的点击和文本的显示都是正确的。

3.4. 使用自定义条件同步测试

Selenium WebDriver 也可以结合 ExpectedCondition 类来定义自己期望的条件。

如何实现

```

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ExpectedCondition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，我们这里只是做一个示范。

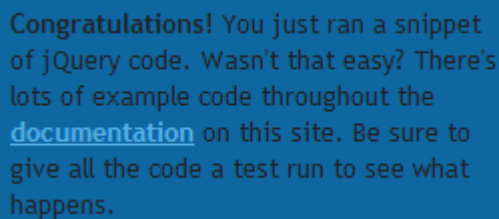
所以，我们可以建立各种的自定义的期望条件。下面的部分中，我们将去学习一些常见的例子。

等待元素的属性值改变

基于某些事件的操作后，元素的属性值可能会改变。例如，一个不可输入的文本框变成可输入的状态。自定义的等待可以在元素的属性上实现。在下面的例子中，`ExpectedCondition` 类将等待返回 `Boolean` 值，


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

以 jquery.com 这个网站为例，当我们点击 RUN CODE 按钮后，下面的文本就会显示



Congratulations! You just ran a snippet of jQuery code. Wasn't that easy? There's lots of example code throughout the [documentation](#) on this site. Be sure to give all the code a test run to see what happens.

其原因是这个 p 元素增加了一个 ohmy 的类。

```
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);
        //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

开发人员会隐藏或是在一系列操作后显示某元素。指定的元素一开始已经存在于 DOM 中，但是为隐藏的状态，当用户经过指定的操作后变为可见。那么这样的自定义期望条件应该如下：

```

(new WebDriverWait(driver, 10)).until(new
ExpectedCondition<Boolean>()
{
    public Boolean apply(WebDriver d) {
        return d.findElement(By.id("page4")).isDisplayed();
    }
});

```

还是以上面的 jquery 网站为例，当我们点击 RUN CODE 后，这个 p 元素才会出现，所以我们将上面的例子修改为

```

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ExpectedCondition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");
    }
});
```

3.5. 检查元素是否存在

Selenium WebDriver 并没有实现 Selenium RC 的 `isElementPresent()` 方法来检查页面上的元素是否存在。在进行下一步操作的时候检查页面的元素是否存在, 对于构建一个稳定的测试是很有帮助的。

在此秘籍中, 我们将编写一个类似 `isElementPresent()` 的方法。

如何实现

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

在测试中调用 `isElementPresent()` 方法来检查页面的元素是否存在, 如果存在则点击此元素, 否则测试将失败。

```
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("sul"))) {
            //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;
        }
    }
}
```

3.6. 检查元素的状态

许多测试失败是因为点击一个元素失败或是在一个不可见的字段中输入文本,或是在不可输入的文本中输入文字。

我们可以在执行操作的时候检查一下元素的状态。WebElement 类提供了这样的方法

方法

isEnabled()

目的

检查元素是否启用

<code>isSelected()</code>	检查元素是否被选中（单选，多选，下拉框）
<code>isDisplayed()</code>	检查元素是否可见

在此秘籍中，我们将使用这些方法来处理元素的状态

如何实现

其实我们在前面的例子中，已经使用到了这样的方法，参考处理单选按钮的例子

```
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;
                }
            }
        }
    }
}
```

```
    }  
  }  
}  
}
```

3.7. 通过名称识别和处理一个弹出窗口

Selenium WebDriver 测试弹出窗口，包括识别弹出窗口，将 driver 转到新的窗口，在新的窗口中执行测试步骤，然后再转换到最初的窗口。

Selenium WebDriver 允许我们通过 name 属性或窗口的句柄来识别窗口，然后通过 `Webdriver.switchTo().window()` 方法在不同的窗口之前进行转换。

在此秘籍中，我们通过 name 属性来识别和处理弹出窗口。开发人员为窗口提供的 name 属性和它的 title 是不一样的。在下面的例子中，一位用户可以通过点击 **Help** 按钮弹出一个新的窗口，开发人员为此窗口提供了 name 属性为 HelpWindow

```
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();
}
}
```

Selenium WebDriver 提供了 `driver.switchTo().window()` 方法来转换窗口，此方法可以接收 `name` 和 `handle` 属性。如果使用 `name` 属性，则就是

```
driver.switchTo().window("HelpWindow");
```

在测试中，当你想回到父窗口的时候，你可以先保存父窗口的 `handle`。

```
String parentWindowId = driver.getWindowHandle();
```

然后通过 `driver.switchTo().window("parentWindowId")` 回到父窗口。如果没有找到窗口则会抛出 `NoSuchWindowException` 的异常。

你可以通过 `driver.close()` 来关闭窗口，但是一般来说子窗口都会提供按钮，通过点击来关闭窗口，所以直接关闭窗口可能会导致错误或异常的发生。

3.8. 通过标题识别和处理一个弹出窗口

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。

如何实现

```
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();
    }
}
```

3.9. 通过网页内容识别和处理一个弹出窗口

某些特殊情况下，开发人员既没有给弹出的窗口分配一个 `name` 属性也没有提供页面的 `title`。当需要处理多少窗口的时候就变得非常复杂。

理想的情况是，你可以建议开发人员让他们去加上这些属性。作为一个解决方案，我们也可以通过检查页面的内容来识别我们需要的弹出窗口。

如何实现

创建一个测试，获得他们的 handles，然后检查页面的内容来识别窗口。

```
package com.example.tests;

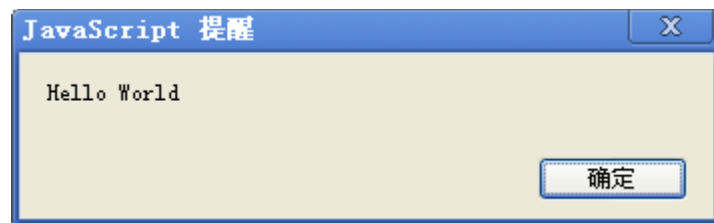
import static org.junit.Assert.*;
import java.util.Set;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

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。

3.10. 处理一个简单的 JavaScript 警告框

前端开发人员使用 JavaScript 警告框给用户一些信息的验证、错误、警告、从一个行为中得到一个响应、接收一个输入的值等等。



测试人员需要验证在测试时显示了正确的警告框。WebDriver 提供了 `Alert` 类来处理 JavaScript 警告框。

在此秘籍中,我们将使用 WebDriver 的 `Alert` 类来处理一个简单的警告框。用户常必须通过点击 `ok` 按钮来处理,如下图所示:

如何实现

创建一个测试,当点击一个按钮的时候,警告框弹出,同时验证警告框里信息。

```
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");
        //点击按钮弹出alert框
        WebElement button = driver.findElement(By.id("alert"));
        button.click();
        //获取alert窗口
        Alert alertBox = driver.switchTo().alert();
        alertBox.accept();
    }
}
```

```

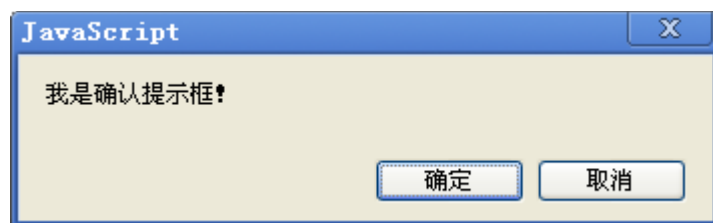
//验证alert窗口里的文字
assertEquals("Hello World", alertBox.getText());
driver.close();
}
}

```

`driver.switchTo().alert()` 将窗口移动到警告框上。`alert.getText()` 得到警告框上的信息。如果找不到警告框则会抛出 `NoAlertPresentException` 异常。

3.11. 处理一个确认框

确认框经常是用来让用户确认或取消一个操作。当一个确认框弹出的时候，用户必须选择 **Ok** 或 **Cancel** 中一个按钮，如下图所示：



如果用户点击了**确定**，返回 `true`；如果点击了取消则返回 `false`，在本秘籍中，我们要用 `Alert` 类来处理一个确认框。

如何实现

```

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。

3.12. 处理一个提示框

提示框经常是用来让用户在进行页面前输入数据的。当弹出一个提示框的时候，用户必须点击**确认**或**取消**按钮之一。



如何实现

我们将要在提示框中输入一些数据，然后再去验证我们输入的数据成功的显示在页面上了。

```

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();
    }
}

```

3.13. 识别处理框架

HTML 框架允许开发人员在多个视图中展现页面，这可能是一个独立的窗口也可能是一个子窗口。多视图给开发者提供了一个让某些信可见，而其它视图是可以进行滚动条的操作。比如，在同一个窗口中，一个框架显示了一个静态的横幅，第二个框架显示了一个导航菜单，第三主框架用来显示主要信息并可以滚动和当点击不同的菜单显示相应的页面。

通过使用<frameset>或是<iframe>标签来创建框架。下面的一个例子中，一个页面显示了三个框架，每一个框架加载不同的 HTML 页面。

```

<html>
  <frameset cols="25%,*,25%" frameborder="0" framespacing="0"
border="0">
    <frame id="left" src="left.html">
    <frame src="middle.html">
    <frame name="right" src="right.html">
  </frameset>
</html>

```

通过 ID 或是 name 可以识别框架。在本秘籍中，我们先识别然后通过 `driver.switchTo().frame()` 方法移动到框架上。

如何实现

```
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 testFrameByIdOrName() {
        driver.get("D:\\demo\\frame\\frame.html");
        //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();
    }
}
```

对于没有 id 或是 name 的框架，我们可以通过 index 来获取焦点，方法如下

```
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 testFrameByIndex() {
```

```

driver.get("D:\\demo\\frame\\frame.html");
//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);
driver.close();
}
}

```

3.14. 通过页面内容识别和处理框架

在处理框架的时候，你发现框架没有 id 或 name 属性的时候，可以通过索引来识别框架。当系统是动态变化的时候，这将不是非常的可靠，我们需要确保激活了正确的框架：

如何实现

让我们创建一个测试得到所有的<frame>元素，然后循环的找到我们需要的框架：

```

package com.example.tests;
import static org.junit.Assert.*;
import java.util.List;
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\\frame\\frame.html");
        //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);
driver.close();
}
}

```

你可以将此方法封装起来，就变成一个通过内容来定位需要的框架。

3.15. 处理 IFRAM

开发人员有时候也会使用<iframe>标签嵌内部的网页或是外部域的网页。这里我们嵌入一个新浪微博的网页。加入下列代码

```

<iframe allowtransparency="true" frameborder="0" scrolling="no"
src="http://v.t.sina.com.cn/widget/widget_topic_publish.php?tags=
舜天&skin=1&isShowPic=1&width=100%&height=500&wordLength=&mblogNum
=20" style="width:300px; height:800px;"></iframe>

```

识别和处理<iframe>标签和<frameset>很相似，

如何实现

我们将在一个嵌入<iframe>标签的页面上创建一个测试。首先找到其父框架，然后再定位到<iframe>标签。

```

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\\iframe\\iframe.html");
        //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();
driver.close();
}
}
```

4. 数据驱动测试

在本章，我们将讨论

- ◆ 使用 JUnit 创建一个数据驱动测试
- ◆ 使用 TestNG 创建一个数据驱动测试
- ◆ 使用 JUnit 从 CSV 中读取数据
- ◆ 使用 JUnit 和 Apache POI 从 Excel 中读取数据
- ◆ 使用 JUnit 和 JDBC 从数据库中读取数据
- ◆ 使用 NUnit 创建一个数据驱动测试
- ◆ 使用 Roo 创建一个 Ruby 的数据驱动测试
- ◆ 创建一个 Python 的数据驱动测试

4.1. 简介

测试是相当重复的工作，不仅仅是因为我们要一次又一次的运行相同的测试。例如，我们可能运行同一个测试需要有不同的输入或者测试条件，然后验证相应的输出结果。每一个测试都有相同的步骤，仅仅是测试数据不同而已。

我们可以使用数据驱动方法来完成这样的测试，我们使用 BMI 计算应用作为例子来理解数据驱动方法。**Body Mass Index(BMI)**是一个基于身高和体重的身体肥胖度量，应用于任何 18 到 65 岁的男女。BMI 指数可以显示出你是否有超重，肥胖，偏轻，或是正常。下面的表格列出了 BMI 的分类：

分类	BMI 指数
偏轻	低于 18.5
正常	18.5-24.9
超重	25-29.9

肥胖

大于 30

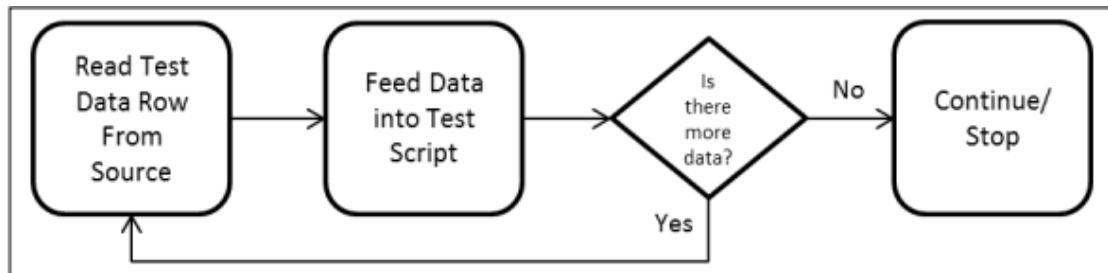
为了测试 BMI 计算的数据是否正确，我们不需要为每一个分类建立一个单独的测试，通过一个脚本输入一组不同的身高和体重数据然后验证输出的结果。我们可以使用下面的测试数据来测试这个 BMI 计算器：

身高（厘米）	体重（公斤）	BMI	分类
160	45	17.6	偏轻
168	70	24.8	正常
181	89	27.2	超重
178	100	31.6	肥胖

数据驱动方法 – 工作流

在数据驱动的方法中，我们可以用多种的形式来维护测试数据，如 CSV 文件，Excel 表格或数据库。我们从数据文件中一行一行的读取输入和输出，然后传递给测试代码执行测试脚本。

数据驱动的测试非常适合应用程序涉及到计算范围的值，边界值和极端案例。



数据驱动测试的好处

数据驱动测试的好处是：

- ◆ 使用数据驱动测试，我们可以获得更多的测试覆盖，减少测试的代码和后期的维护工作。
- ◆ 数据驱动测试可以非常容易的创建和运行很多条件的测试。
- ◆ 数据驱动测试不需要什么编码的技巧。甚至初学者都可以快速的建立一个属于自己的自动化测试数据表格。
- ◆ 测试数据可以提前设计，不需要关心被测程序是否可测。
- ◆ 数据驱动测试也一样可以用于手动测试。

总的来说构建一个大型的自动化测试工程，数据驱动方法的关键是易用。

Selenium WebDriver，作为一个纯浏览器自动化测试的 API，本身没有提供数据驱动的功能。本章，我们将使用 JUnit，TestNG 来创建一些基本的数据驱动的测试。然后，会使用一些其他格式的数据源。

我们也是创建不同语言的数据驱动测试.NET，Ruby，Python。

4.2. 使用 JUnit 创建一个数据驱动测试

JUnit 是一个用来创建 Java 的 Selenium WebDriver 测试非常流行的框架。我们可以使用 JUnit4 参数化的特性来创建 Selenium WebDriver 的数据驱动测试。

在本秘籍中，我们将创建一个简单的 BMI 的 JUnit 测试用例，在测试类中指定测试数据。

如何实现

```
package com.example.tests;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.*;
import org.junit.runners.Parameterized.Parameters;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

@RunWith(value = Parameterized.class)
public class simpleDDT {
    private static WebDriver driver;
    private String height;
    private String weight;
    private String bmi;
    private String bmiCategory;

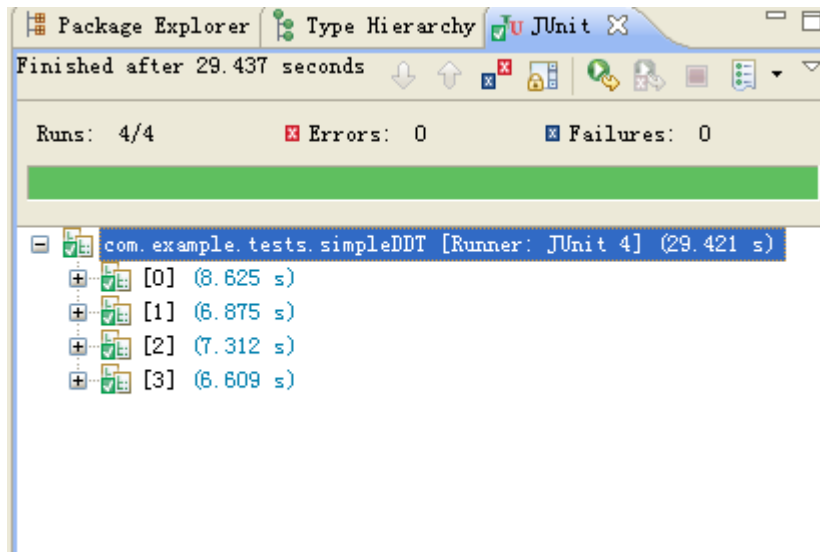
    @Parameters
    public static Collection testData() {
        return Arrays.asList(new Object[][]{
            {"160", "45", "17.6", "Underweight"},
            {"168", "70", "24.8", "Normal"},
            {"181", "89", "27.2", "Overweight"},
            {"178", "100", "31.6", "Obesity"},
        });
    }
}
```

```
//simpleDDT的构造函数，用来赋值
public simpleDDT(String height,String weight,String bmi,String
bmiCategory) {
    this.height = height;
    this.weight = weight;
    this.bmi = bmi;
    this.bmiCategory = bmiCategory;
}

@Test
public void testBMICalculator() {
    driver = new FirefoxDriver();
    driver.get("d:\\demo\\BMICalculator.html");
    //输入身高
    WebElement heightField =
driver.findElement(By.name("heightCMS"));
    heightField.sendKeys(height);
    //输入体重
    WebElement weightField =
driver.findElement(By.name("weightKg"));
    weightField.sendKeys(weight);
    //点击计算
    WebElement calculateButton =
driver.findElement(By.id("Calculate"));
    calculateButton.click();
    //验证计算后的BMI值
    WebElement bmiLabel = driver.findElement(By.name("bmi"));
    assertEquals(bmi, bmiLabel.getAttribute("value"));
    //验证分类的结果
    WebElement bmiCategoryLabel =
        driver.findElement(By.name("bmi_category"));
    assertEquals(bmiCategory,
        bmiCategoryLabel.getAttribute("value"));
    driver.close();
}

@After
public void tearDown() {
    driver.close();
}
}
```

当测试开始执行的时候，测试数据集中的每一行数据都会传递给 simpleDDT 类。测试执行结束后，结果显示如下，4 组数据都执行完毕。



这是一个简单的参数化的例子，测试数据是硬编码在测试类当中，不是很好的维护，所以还是推荐大家将测试数据保存在一个数据源当中，如 CSV，Excel 或是数据库中，这样可以更容易维护。

4.3. 使用 TestNG 创建一个数据驱动测试

TestNG 是另一个被 WebDriver 广泛应用的框架，和 JUnit 很相似。TestNG 增加了更多的测试功能，如参数化，并行运行测试等等。

TestNG 提供了 `DataProvider` 功能来创建数据驱动的测试。在此秘籍中，我们将使用 `DataProvider` 来创建一个简单的测试。

如何实现

```
package com.example.tests;

import static org.junit.Assert.assertEquals;
import org.openqa.selenium.*;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.*;

public class TestNG {
    WebDriver driver;
```

```
@DataProvider
public Object[][] testData() {
    return new Object[][]{
        {"160", "45", "17.6", "Underweight"},
        {"168", "70", "24.8", "Normal"},
        {"181", "89", "27.2", "Overweight"},
        {"178", "100", "31.6", "Obesity"},
    };
}

@BeforeTest
public void setUp() {
    driver = new FirefoxDriver();
}

@Test(dataProvider = "testData")
public void testBMICalculator(String height, String weight
    , String bmi, String bmiCategory) {
    driver.get("d:\\demo\\BMICalculator.html");
    //输入身高
    WebElement heightField =
        driver.findElement(By.name("heightCMS"));
    heightField.sendKeys(height);
    //输入体重
    WebElement weightField =
        driver.findElement(By.name("weightKg"));
    weightField.sendKeys(weight);
    //点击计算
    WebElement calculateButton =
        driver.findElement(By.id("Calculate"));
    calculateButton.click();
    //验证计算后的BMI值
    WebElement bmiLabel =
        driver.findElement(By.name("bmi"));
    assertEquals(bmi, bmiLabel.getAttribute("value"));
    //验证分类的结果
    WebElement bmiCategoryLabel =
        driver.findElement(By.name("bmi_category"));
    assertEquals(bmiCategory,
        bmiCategoryLabel.getAttribute("value"));
}

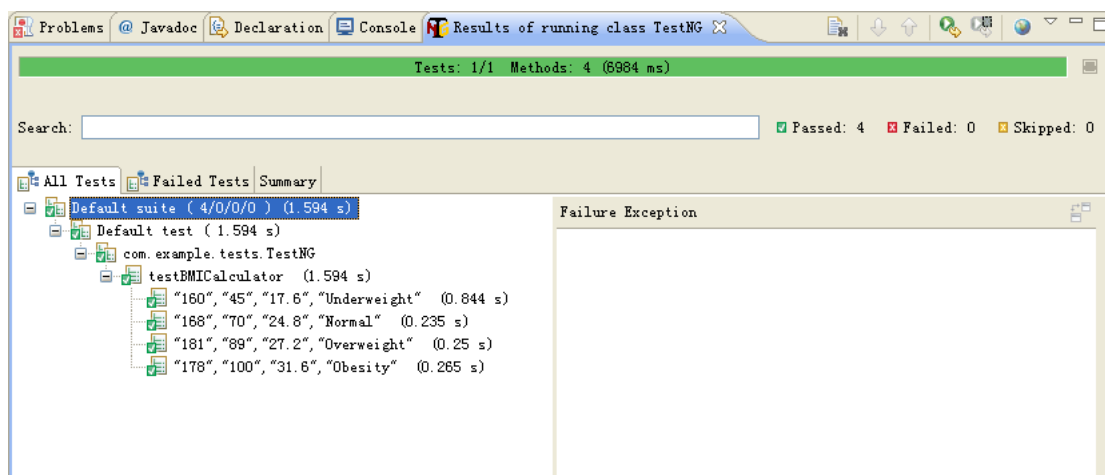
@AfterTest
public void tearDown() {
```

```

        driver.quit();
    }
}

```

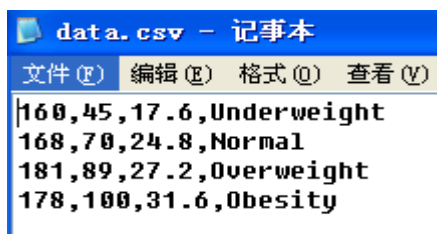
和 JUnit 不同的是，参数化是完成在类级别的，TestNG 的参数化是在测试级别的。TestNG 不需要通过构造函数来传递参数，它会自动的映射。在测试结束后也会在 eclipse 中生成一个简单的报告。



4.4. 使用 JUnit 从 CSV 读取数据

我们前面看到了如何使用 JUnit 和 TestNG 来创建数据驱动的测试。测试数据是硬编码在测试脚本当中。这让维护工作比较困难。比较推荐的方式是将测试的数据和脚本分开存储。

我们经常使用产生环境下的数据来进行测试。这些数据可以导出成一个 CSV 格式的文件。我们可以通过 Java IO 工具类来读取 CSV 的文件，然后就数据传递给测试脚本。在本秘籍中，我们将创建一个从 CSV 文件中读取测试数据的脚本。



如何实现

```

package com.example.tests;

import static org.junit.Assert.*;

import java.io.*;

```

```
import java.util.*;
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.*;
import org.junit.runners.Parameterized.Parameters;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

@RunWith(value = Parameterized.class)
public class CSVData {
    private WebDriver driver;
    private String height;
    private String weight;
    private String bmi;
    private String bmiCategory;
    //参数化
    @Parameters
    public static Collection<String[]> testData() {
        return getTestData("d:\\demo\\data\\data.csv");
    }
    //构造函数赋值
    public CSVData(String height,String weight
        ,String bmi ,String bmiCategory) {
        this.height = height;
        this.weight = weight;
        this.bmi = bmi;
        this.bmiCategory = bmiCategory;
    }
    //读取CSV中的文件
    public static Collection<String[]> getTestData(String path){
        List<String[]> records =
            new ArrayList<String[]>();
        String row;
        try {
            BufferedReader br =
                new BufferedReader(new FileReader(path));
            while ((row = br.readLine())!=null) {
                String fields[] = row.split(",");
                records.add(fields);
            }
            br.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```
        return records;
    }

@Before
public void setUp() {
    driver = new FirefoxDriver();
    driver.get("d:\\demo\\BMICalculator.html");
}

@Test
public void testBMICalculatior() {
    //输入身高
    WebElement heightField =
        driver.findElement(By.name("heightCMS"));
    heightField.sendKeys(height);
    //输入体重
    WebElement weightField =
        driver.findElement(By.name("weightKg"));
    weightField.sendKeys(weight);
    //点击计算
    WebElement calculateButton =
        driver.findElement(By.id("Calculate"));
    calculateButton.click();
    //验证计算后的BMI值
    WebElement bmiLabel =
        driver.findElement(By.name("bmi"));
    assertEquals(bmi, bmiLabel.getAttribute("value"));
    //验证分类的结果
    WebElement bmiCategoryLabel =
        driver.findElement(By.name("bmi_category"));
    assertEquals(bmiCategory,
        bmiCategoryLabel.getAttribute("value"));
}

@After
public void tearDown() {
    driver.close();
}
}
```

当测试执行的时候, `testData()` 方法调用 `getTestData()` 方法来获得测试的数据, 通过逗号将字符串分割成数组再添加到集合当中。 每一行的数据都会通过构造函数来进行初始化赋值。

4.5. 使用 JUnit 和 Apach POI 读取 Excel 数据

在维护测试用例和测试数据的时候，Microsoft Excel 是最常用的工具。和 CSV 相比，Excel 提供了更多的功能和结构化的方式来存储数据。测试人员建立维护一个 Excel 数据表格更加容易。

	A	B	C	D
1	Height	Weight	BMI	Category
2	160	45	17.6	Underweight
3	168	70	24.8	Normal
4	181	89	27.2	Overweight
5	178	100	31.6	Obesity

在此秘籍中，我们将使用 Excel 电子表格作为数据源。我们将使用 Apache POI API（一个 Apache 的开源项目）来操作 Excel 电子表格。（注，为了简化操作，在此例子中我将单元格的数据都使用了字符串格式，在取值的时候我就没有判断 Excel 的数据格式，如果你没有这样做，用了其他的数据格式，则你在取单元格的数据时要先判断是什么数据才能输出）

从 <http://poi.apache.org/> 下载 POI 文件，并导入下列 jar 包到项目中。

- ◆ poi-3.8.jar
- ◆ poi-ooxml-3.8.jar
- ◆ poi-ooxml-schemas-3.8.jar
- ◆ dom4j-1.6.1.jar
- ◆ stax-api-1.0.1.jar
- ◆ xmlbeans-2.3.0.jar

这里我们用的 Excel 的格式为.xlsx。

如何实现

```
package com.example.tests;
import static org.junit.Assert.*;
import java.io.*;
import java.util.*;
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.*;
import org.junit.runners.Parameterized.Parameters;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.apache.poi.xssf.usermodel.*;
```

```
@RunWith(value = Parameterized.class)
public class ExcelData {
    private WebDriver driver;
    private String height;
    private String weight;
    private String bmi;
    private String bmiCategory;
    //参数化
    @Parameters
    public static Collection<String[]> testData () throws IOException{
        InputStream is =
            new FileInputStream("d://demo/data//data.xlsx");
        XSSFWorkbook workbook = new XSSFWorkbook(is);
        //获得工作表
        XSSFSheet sheet = workbook.getSheetAt(0);
        //得到总行数
        int rowNum = sheet.getLastRowNum();
        List<String[]> records =
            new ArrayList<String[]>();
        for (int i = 1; i < rowNum; i++) {
            //当前行
            XSSFRow row = sheet.getRow(i);
            int colNum = row.getLastCellNum();
            String[] data = new String[colNum];

            for (int j = 0; j < colNum; j++) {
                data[j] = row.getCell(j).getStringCellValue();
                System.out.println(data[j]);
            }
            records.add(data);
        }
        return records;
    }
    //构造函数赋值
    public ExcelData(String height,String weight
        ,String bmi ,String bmiCategory) {
        this.height = height;
        this.weight = weight;
        this.bmi = bmi;
        this.bmiCategory = bmiCategory;
    }
    //读取EXCEL中的文件
    public static Collection<String[]> getTestData(String path){
        List<String[]> records =
```

```
        new ArrayList<String[]>();
String row;
try {
    BufferedReader br =
        new BufferedReader(new FileReader(path));
    while ((row = br.readLine()) != null) {
        String fields[] = row.split(",");
        records.add(fields);
    }
    br.close();
} catch (Exception e) {
    e.printStackTrace();
}
return records;
}
```

@Before

```
public void setUp() {
    driver = new FirefoxDriver();
    driver.get("d:\\demo\\BMICalculator.html");
}
```

@Test

```
public void testBMICalculatior() {
    //输入身高
    WebElement heightField =
        driver.findElement(By.name("heightCMS"));
    heightField.sendKeys(height);
    //输入体重
    WebElement weightField =
        driver.findElement(By.name("weightKg"));
    weightField.sendKeys(weight);
    //点击计算
    WebElement calculateButton =
        driver.findElement(By.id("Calculate"));
    calculateButton.click();
    //验证计算后的BMI值
    WebElement bmiLabel =
        driver.findElement(By.name("bmi"));
    assertEquals(bmi, bmiLabel.getAttribute("value"));
    //验证分类的结果
    WebElement bmiCategoryLabel =
        driver.findElement(By.name("bmi_category"));
    assertEquals(bmiCategory,
```

```

        bmiCategoryLabel.getAttribute("value"));
    }









    @After
    public void tearDown() {
        driver.close();
    }
}

```

4.6. 使用 JUnit 和 JDBC 来读取数据库中的数据

在之前的秘籍中，我们使用了 CSV 和 Excel 来维护测试数据。测试数据也一样可以从数据库中读取。这里我们用 JDBC 来连接 MySQL 数据库，当然它可以应用在任何的数据库上。

在项目中需导入 mysql-connector-java-5.1.22-bin.jar

← T →			height	weight	bmi	bmicategory
<input type="checkbox"/>			168	70	24.8	Normal
<input type="checkbox"/>			181	89	27.2	Overweight
<input type="checkbox"/>			178	100	31.6	Obesity
<input type="checkbox"/>			160	45	17.6	Underweight

在数据库保存的测试数据

如何实现

```

package com.example.tests;
import static org.junit.Assert.*;
import java.sql.*;
import java.util.*;
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.*;
import org.junit.runners.Parameterized.Parameters;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

@RunWith(value = Parameterized.class)
public class MySqlData {
    private WebDriver driver;
    private String height;

```

```
private String weight;
private String bmi;
private String bmiCategory;
//参数化
@Parameters
public static Collection<String[]> testData() throws SQLException
{
    List<String[]> records =
        new ArrayList<String[]>();
    //连接数据库
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }

    Connection con = DriverManager.getConnection
        ("jdbc:mysql://127.0.0.1:3306/test","root","");
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("select * from testdata");
    //取得结果集结构
    ResultSetMetaData rsMetaData = rs.getMetaData();
    int cols = rsMetaData.getColumnCount(); //取得例数
    while(rs.next()) {
        String fields[] = new String[cols];
        int col = 0;
        for(int colIdx=1;colIdx<=cols;colIdx++) {
            fields[col] = rs.getString(colIdx);
            col++;
        }
        records.add(fields);
    }
    rs.close();
    st.close();
    con.close();
    return records;
}
//构造函数赋值
public MySqlData(String height,String weight
    ,String bmi ,String bmiCategory) {
    this.height = height;
    this.weight = weight;
    this.bmi = bmi;
    this.bmiCategory = bmiCategory;
}
```

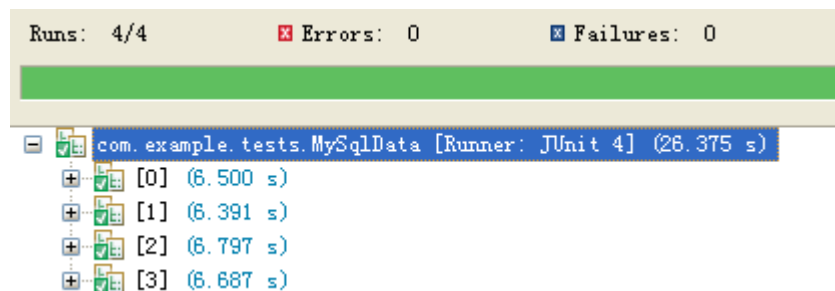
```
}

@Before
public void setUp() {
    driver = new FirefoxDriver();
    driver.get("d:\\demo\\BMICalculator.html");
}

@Test
public void testBMICalculatior() {
    //输入身高
    WebElement heightField =
        driver.findElement(By.name("heightCMS"));
    heightField.sendKeys(height);
    //输入体重
    WebElement weightField =
        driver.findElement(By.name("weightKg"));
    weightField.sendKeys(weight);
    //点击计算
    WebElement calculateButton =
        driver.findElement(By.id("Calculate"));
    calculateButton.click();
    //验证计算后的BMI值
    WebElement bmiLabel =
        driver.findElement(By.name("bmi"));
    assertEquals(bmi, bmiLabel.getAttribute("value"));
    //验证分类的结果
    WebElement bmiCategoryLabel =
        driver.findElement(By.name("bmi_category"));
    assertEquals(bmiCategory,
        bmiCategoryLabel.getAttribute("value"));
}

@After
public void tearDown() {
    driver.close();
}
}
```

测试结果



4.7. 创建一个 NUnit 数据驱动测试

NUnit 框架在 Selenium WebDriver 社区中也非常的常用。可以通过.NET 的语言来编写测试脚本。

同 JUnit 一样，NUnit 框架也支持数据区别的测试。在此秘籍中，我们将使用 NUnit 来创建一个测试。从 XML 中读取测试数据。(注,这里的代码我是直接从书里复制出来，我只用 java 的语言，所以后面的其他类型的语言我就没有一一测试过)

```
<testdata>
  <vars height="160" weight="45" bmi="17.6"
bmi_category="Underweight" />
  <vars height="168" weight="70" bmi="24.8"
bmi_category="Normal" />
  <vars height="181" weight="89" bmi="27.2"
bmi_category="Overweight" />
  <vars height="178" weight="100" bmi="31.6"
bmi_category="Obesity" />
</testdata>
```

如何实现

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NUnit.Framework;
using System.Collections;
using System.Xml.Linq;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Support;
using OpenQA.Selenium.Support.UI;
namespace BMICalculator
{
    [TestFixture]
    public class BMICalculatorNUnitTest
    {
        IWebDriver driver;
        [SetUp]
```



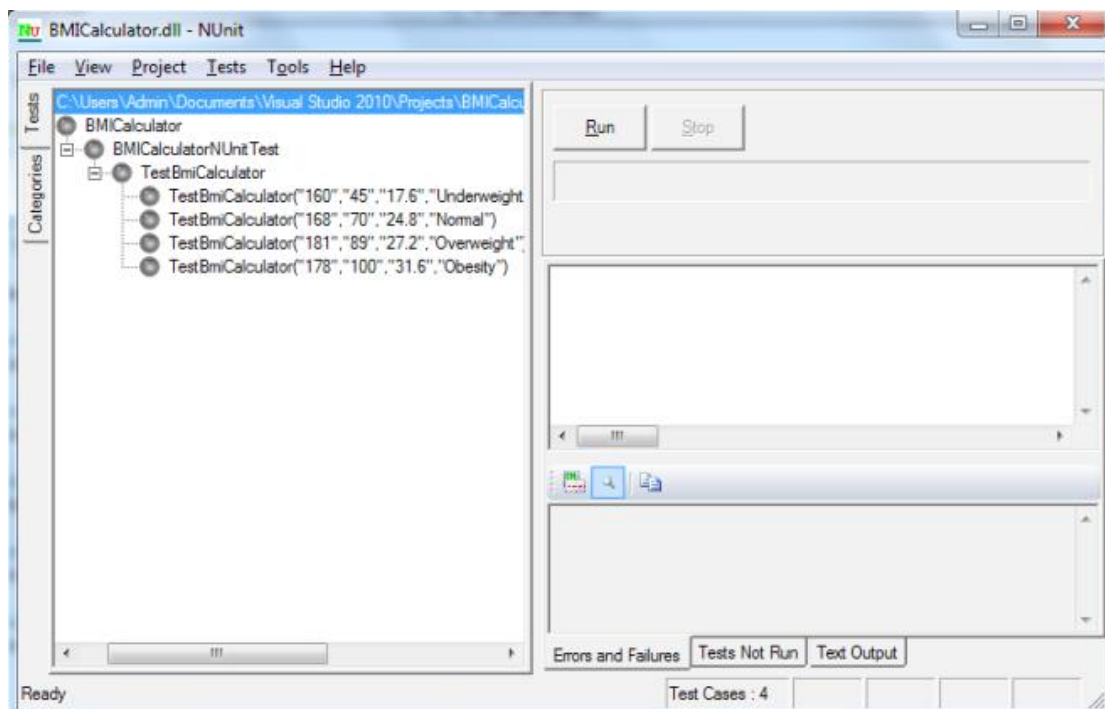
```

public void TestSetup()
{
    // Create a instance of the Firefox driver using
    IWebDriver Interface
    driver = new FirefoxDriver();
}
[TestCaseSource("BmiTestData")]
public void TestBmiCalculator(string height, string weight,
string expected_bmi, string expected_category)
{
    driver.Navigate().GoToUrl("http://dl.dropbox.
com/u/55228056/mobilebmiccalculator.html");
    IWebElement heightElement = driver.FindElement(By.
Name("heightCMS"));
    heightElement.SendKeys(height);
    IWebElement weightElement = driver.FindElement(By.
Name("weightKg"));
    weightElement.SendKeys(weight);
    IWebElement calculateButton = driver.FindElement(By.
Id("Calculate"));
    calculateButton.Click();
    IWebElement bmiElement = driver.FindElement(By.
Name("bmi"));
    Assert.AreEqual(expected_bmi, bmiElement.
GetAttribute("value"));
    IWebElement bmiCatElement = driver.FindElement(By.
Name("bmi_category"));
    Assert.AreEqual(expected_category, bmiCatElement.
GetAttribute("value"));
}
[TearDown]
public void TestCleanUp()
{
    // Close the browser
    driver.Quit();
}

private IEnumerable BmiTestData
{
    get { return GetBmiTestData(); }
}
private IEnumerable GetBmiTestData()
{
    var doc = XDocument.Load(@"c:\data.xml");
    return
        from vars in doc.Descendants("vars")
        let height = vars.Attribute("height").Value
        let weight = vars.Attribute("weight").Value
        let expected_bmi = vars.Attribute("bmi").Value
        let expected_category = vars.Attribute
        ("bmi_category").Value
        select new object[] { height, weight,
        expected_bmi, expected_category };
}
}

```

同样 NUnit 也可以从其他的数据源中读取数据，Nuint 的 GUI 测试执行的结果，如下图



4.8. 使用 ROO 创建一个 Ruby 的数据驱动测试

在前面的章节中，我们看到了如何使用 Java 和 .NET 来进行参数化，Ruby 也是一种非常常用的脚本语言被广大 Selenium 的用户使用。

同样，Ruby 本身不支持参数化脚本，但是我们可以使用 ROO 来读取数据，ROO 支持多种的格式如 Excel(xls), OpenOffice(ods), Google 电子表格。ROO 是一个非常好的工具来处理 Excel，它甚至都不需要安装 office，自身就可以读取这些文件。

在本秘籍中，我们将使用 Excel 作为数据源，使用 Ruby 来参数化 Selenium 的测试脚本。

如何实现

首先必须安装 ROO，此命令将会安装所有的依赖关系的软件。

```
gem install roo
```

通过下面的步骤，我们将创建一个简单的 Ruby 的参数化脚本的例子，读取 Excel 里面的数据

导入下面的模块

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

1. 创建一个 WebDriver 实例，使用 Firefox 浏览器。

```
#Create an instance of WebDriver for Firefox
driver = Selenium::WebDriver.for :firefox
```

2. 声名一些测试总结需要变量

```
#Variables for Printing Test Summary
test_executed = 0
test_passed = 0
test_failed = 0
```

3. 创建一个 Excel 的实例来读取电子表格

```
#Create an instance of a Excel Spreadsheet
data = Excel.new("C:\\Data.xls")
data.default_sheet = data.sheets.first
```

4. 增加下面的代码，读取电子表格并进行验证

```
#Iterate through the Sheet reading Rows line by line
data.first_row.upto(data.last_row) do |line|
  if data.cell(line,1) != "Height" #Ignore the first line for
  Headers
    begin
      test_status = true
      test_executed = test_executed + 1
      puts "Test " + test_executed.to_s()

      driver.get "http://dl.dropbox.com/u/55228056/bmicalculator.
      html"
      height = driver.find_element :name => "heightCMS"
      height.send_keys data.cell(line,1).to_s()
      weight = driver.find_element :name => "weightKg"
      weight.send_keys data.cell(line,2).to_s()
      calculateButton = driver.find_element :id => "Calculate"
      calculateButton.click
      bmi = driver.find_element :name => "bmi"
      bmi_category = driver.find_element :name => "bmi_category"

      if bmi.attribute("value").to_s() == data.cell(line,3).to_s()
        puts "Pass, expected value for BMI <" + data.cell(line,3).
          to_s() + ">, actual <" + bmi.attribute("value").to_s() + ">"
      else
        puts "Fail, expected value for BMI <" + data.cell(line,3).
          to_s() + ">, actual <" + bmi.attribute("value").to_s() + ">"
        test_status=false
      end

      if bmi_category.attribute("value").to_s() == data.
        cell(line,4).to_s()
        puts "Pass, expected value for BMI Category <" +
          data.cell(line,4).to_s() + ">, actual <" + bmi_category.
            attribute("value").to_s() + ">"
        else
          puts "Fail, expected value for BMI Category <" +
```

```

        data.cell(line,4).to_s() + ">", actual <" + bmi_category.
        attribute("value").to_s() + ">"
        test_status=false
    end

    if test_status == true
        test_passed = test_passed + 1
    else
        test_failed = test_failed + 1
    end
end
rescue
    puts "An error occurred: #{$!}"
end
end
end
end

```

5. 最后打印出一个执行结果的报告

```

puts "-----"
puts "Total (" + test_executed.to_s() + ") Tests Executed"
puts "Total (" + test_passed.to_s() + ") Tests Passed"
puts "Total (" + test_failed.to_s() + ") Tests Failed"
driver.quit

```

通过 `data.first_row.upto(data.last_row) do |line|` 方法读取每一行的数据。我们也需要一个定制的报靠来显示测试执行的结果，如下图所示：

```

C:\>ruby webdriver_roo_dtd.rb
C:/Ruby193/lib/ruby/site_ruby/1.9.1/rubygems/custom_require.rb:36:in `require':
iconv will be deprecated in the future, use String#encode instead.
Test 1
Pass, expected value for BMI <17.6>, actual <17.6>
Pass, expected value for BMI Category <Underweight>, actual <Underweight>
Test 2
Pass, expected value for BMI <24.8>, actual <24.8>
Pass, expected value for BMI Category <Normal>, actual <Normal>
Test 3
Pass, expected value for BMI <27.2>, actual <27.2>
Pass, expected value for BMI Category <Overweight>, actual <Overweight>
Test 4
Pass, expected value for BMI <31.6>, actual <31.6>
Pass, expected value for BMI Category <Obesity>, actual <Obesity>
-----
Total (4) Tests Executed
Total (4) Tests Passed
Total (0) Tests Failed
C:\>_

```

关于 ROO 更多的信息，请访问：<http://roo.rubyforge.org/>。

4.9. 使用 Python 创建一个数据驱动的测试

Python 也是 Selenium WebDriver 测试员经常用的一种语言。它也提供了很多参数化数据的方法。

在本秘籍中，我们将用 CSV 格式为数据源，创建一个 Python 的参数化测试。

如何实现

1. 导入下列模块

```
from selenium import webdriver
import csv, sys
```

2. 声音一些测试总结的变量

```
#Variables for Printing Test Summary
test_executed = 0
test_passed = 0
test_failed = 0
test_status = True
```

3. 加入下在的代码，循环出 CSV 里面的数据，进行操作和验证。

```
try:
    #Create an instance of WebDriver for Firefox
    driver = webdriver.Firefox()
    driver.get("http://dl.dropbox.com/u/55228056/bmicalculator.html")

    #Open the CSV file
    datafile = open('c:\data.csv', "rb")
    #Create a CSV Reader from CSV file
    reader = csv.reader(datafile)
    test_executed = 0
    #Iterate through the CSV Rows line by line
    for row in reader:
        test_executed += 1
        print "Test " + str(test_executed)

        heightField = driver.find_element_by_name("heightCMS")
        heightField.clear()
        heightField.send_keys(row[0])

        weightField = driver.find_element_by_name("weightKg")
        weightField.clear()
        weightField.send_keys(row[1])

        calculateButton = driver.find_element_by_id("Calculate")
        calculateButton.click()

        bmiLabel = driver.find_element_by_name("bmi")
        bmiCategoryLabel = driver.find_element_by_name("bmi_category")

        if bmiLabel.get_attribute("value") == row[2]:
            print "Pass, expected value for BMI <" + row[2] + ">,"
            actual <" + bmiLabel.get_attribute("value") + ">"
        else:
            print "Fail, expected value for BMI <" + row[2] + ">,"
            actual <" + bmiLabel.get_attribute("value") + ">"
```

```

test_status = False

if bmiCategoryLabel.get_attribute("value") == row[3]:
    print "Pass, expected value for BMI Category <" + row[3] +
">, actual <" + bmiCategoryLabel.get_attribute("value") + ">"
else:
    print "Fail, expected value for BMI Category <" + row[3] +
">, actual <" + bmiCategoryLabel.get_attribute("value") + ">"
    test_status = False

if test_status == True:
    test_passed = test_passed + 1
else:
    test_failed = test_failed + 1
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
finally:
    print "-----"
    print "Total (" + str(test_executed)+ ") Tests Executed"
    print "Total (" + str(test_passed)+ ") Tests Passed"
    print "Total (" + str(test_failed) + ") Tests Failed"
    driver.close()
    datafile.close()

```

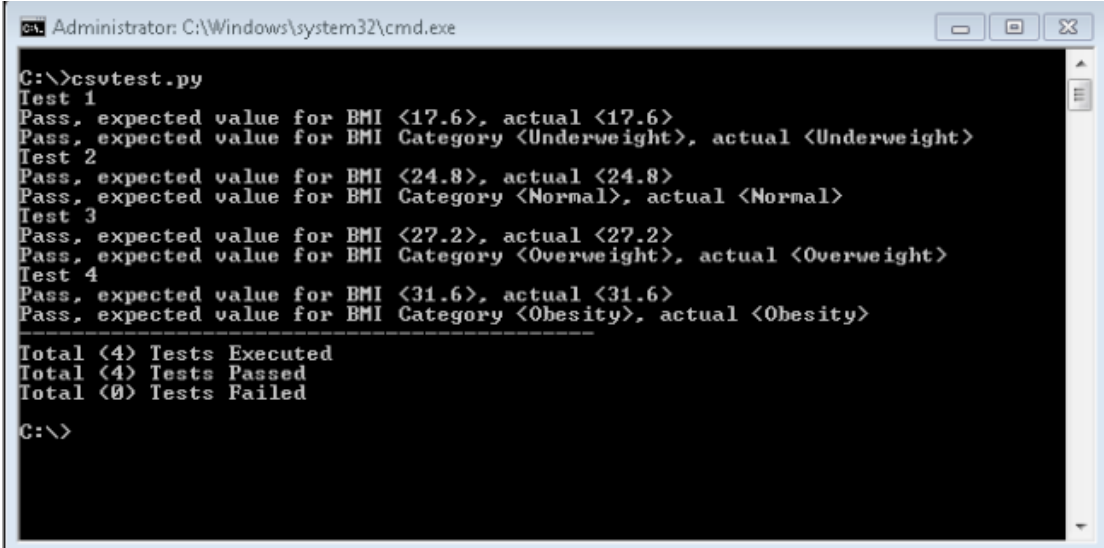
在测试执行的时候，Python 将会把 CSV 文件里的数据读取到 reader 对象中。

```

#Open the CSV file
datafile = open('c:\data.csv', "rb")
#Create a CSV Reader from CSV file
reader = csv.reader(datafile) ets.first

```

我们还添加了一个自定义报告,生成一个良好格式化的报告,如下图所示:



```

Administrator: C:\Windows\system32\cmd.exe

C:\>csvtest.py
Test 1
Pass, expected value for BMI <17.6>, actual <17.6>
Pass, expected value for BMI Category <Underweight>, actual <Underweight>
Test 2
Pass, expected value for BMI <24.8>, actual <24.8>
Pass, expected value for BMI Category <Normal>, actual <Normal>
Test 3
Pass, expected value for BMI <27.2>, actual <27.2>
Pass, expected value for BMI Category <Overweight>, actual <Overweight>
Test 4
Pass, expected value for BMI <31.6>, actual <31.6>
Pass, expected value for BMI Category <Obesity>, actual <Obesity>
-----
Total <4> Tests Executed
Total <4> Tests Passed
Total <0> Tests Failed
C:\>

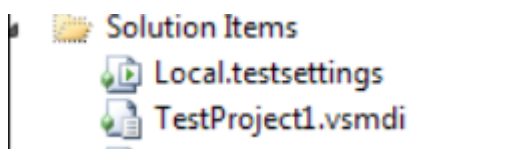
```

4.10. 使用 MSTEST 创建一个数据驱动测试用例

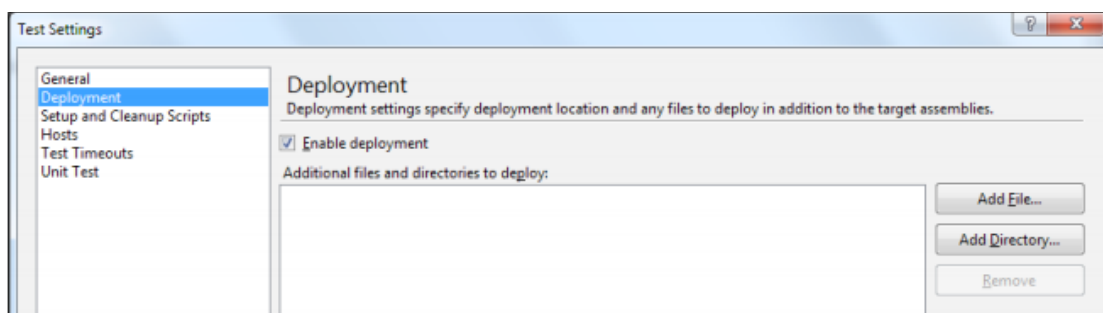
使用 Microsoft Visual Studio 有一个最简单的方法来创建.NET 的数据驱动测试用例。MSTEST 通过简单的配置就可以支持数据驱动的测试。在此秘籍中，我们将使用 MSTEST 来创建 Selenium 的测试，从 Excel 中读取测试数据。

如何实现

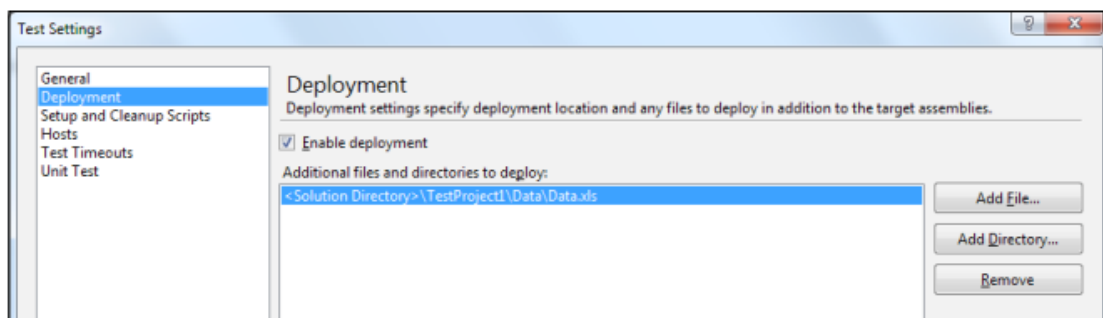
1. 点击 **Solution items** 下的 **Local.testsettings**，如下图所示



2. 在 **Test Settings** 对话框中，在 **Deployment** 部分中点击 **Add File** 按钮，如下图所示



3. 一旦你新加了文件到 **Deployment**，就会列出来，如下图所示



4. 创建一个新的类名为 **BMICalculatorTests**，复制下列代码到类中

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

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Support;
using OpenQA.Selenium.Support.UI;
using System.Data;
namespace BMICalculator
{
    [TestClass]
    public class BMICalculatorTests
    {
        IWebDriver driver;
        [TestInitialize]
        public void TestSetup()
        {
            //Create a instance of the Firefox driver using
            //IWebDriver Interface
            driver = new FirefoxDriver();
        }
        private TestContext testContextInstance;
        /// <summary>
        ///Gets or sets the test context which provides
        ///information about and functionality for the current
        ///test run.
        ///</summary>
        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
        [TestMethod]
        [DeploymentItem("Data.xls")]
        [DataSource("System.Data.OleDb", "Provider=Microsoft.
ACE.OLEDB.12.0;Data Source=Data.xls;Persist Security
Info=False;Extended Properties='Excel 12.0;HDR=Yes'",
        "Data$", DataAccessMethod.Sequential)]
        public void TestBMICalculator()
        {

```



```
driver.Navigate().GoToUrl("http://dl.dropbox.
com/u/55228056/bmicalculator.html");

IWebElement height = driver.FindElement(By.
Name("heightCMS"));
height.SendKeys(TestContext.DataRow["Height"].
ToString());
IWebElement weight = driver.FindElement(By.
Name("weightKg"));
weight.SendKeys(TestContext.DataRow["Weight"].
ToString());
IWebElement calculateButton = driver.FindElement(
Id("Calculate"));
calculateButton.Click();
IWebElement bmi = driver.FindElement(By.Name("bmi"));

Assert.AreEqual(TestContext.DataRow["Bmi"].ToString(),
    bmi.GetAttribute("value"));
IWebElement bmi_category = driver.FindElement(By.
Name("bmi_category"));
Assert.AreEqual(TestContext.DataRow["Category"].
ToString(), bmi_category.GetAttribute("value"));
}
[TestCleanup]
public void TestCleanUp()
{
    // Close the browser
    driver.Quit();
}
}
```

5.使用页面对象模型

在本章中我们将讨论下面的主题：

- ◆ 使用 PageFactory 类分离页面元素
- ◆ 使用 PageFactory 类分离页面的操作
- ◆ 实现页面嵌套对象实例
- ◆ 用.NET 实现页面对象模型
- ◆ 用 Python 实现页面对象模型

◆ 用 Ruby 实现页面对象模型

5.1. 简介

开发一个可维护的自动化代码是自动化测试成功的关键。自动化测试的代码需要和产品的代码一样对待，应用相似的标准和模式。

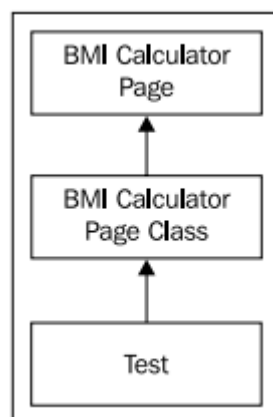
在开发一个 Selenium WebDriver 测试，我们可以使用页面对象模型。这个模型可以使测试脚本有更高的可维护性，减少了重复的代码，把页面抽象出来。

通过应用面向对象的开发原则，我们可以开发一个类作为一个 web 页面的应用程序的接口，建模其属性和行为。通过隐藏页面元素的定位，这有助于将测试的代码和页面的分离。页面对象设计模式提供了测试一个接口，测试可以像用户行为一样来操作页面。例如，如果我们创建了一个登录页面对象模型，提供了一个登录的方法，接受用户名和密码参数。测试不需要关注如何在登录页面输入数据，如何定位元素等。

测试在使用页面对象模型的时候必须有较高的分离度，任何页面布局属性字段的变化，不能影响到测试的业务逻辑。

5.2. 使用 PageFactory 类来分离页面元素

为实现测试的页面对象模型，我们需要为每一个需要测试的页面建立一个类。例如，测试 BMI Calculator 应用，就需要定义一个 BMI Calculator 页面的类，此类中分离出所有需要的页面元素，如下图所示，通过 Selenium WebDriver API 的 PageFactory 类来完成这样的设计。



如何实现

1. 创建一个包 `com.example.pageobjects`，专门用来测试页面对象模型

2. 创建一个 Java 类,取名为我们需要测试页面的名称。例如,我们要测试 BMI Calculator 应用,我们需要创建一个 `BmiCalculatePage` 类,这仅仅是一个页面的应用。
3. 在此类中定义在此页面中需要的元素变量,使用元素的 `name` 或 `id` 属性。

```
public WebElement heightCMS;
public WebElement weightKg;
public WebElement Calculate;
public WebElement bmi;
    public WebElement bmi_category;
```

4. 创建一个构造函数,并调用 `PageFactory.initElements()` 方法来初始化元素。换句话说,将元素映射到我们定义好的变量上。

```
public BmiCalculatePage(WebDriver driver) {
    PageFactory.initElements(driver, this);
}
```

这样我们的页面类就定义好了

```
package com.example.pageobjects;
import org.openqa.selenium.*;
import org.openqa.selenium.support.PageFactory;
public class BmiCalculatePage {
    //使用name或id定义页面元素
    public WebElement heightCMS;
    public WebElement weightKg;
    public WebElement Calculate;
    public WebElement bmi;
    public WebElement bmi_category;
    //构造函数来初始化元素
    public BmiCalculatePage(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }
}
```

5. 再创建对于页面 `BmiCalculatePage` 的真正的测试类 `BmiCalculatorTests`,在此类中写测试的业务逻辑。

```
package com.example.pageobjects;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.junit.Test;
import static org.junit.Assert.*;
public class BmiCalculatorTests {
    @Test
    public void testBmiCalculatorTest() {
        WebDriver driver = new FirefoxDriver();
```

```
driver.get("d:\\demo\\BMICalculator.html");
//创建页面实例
BmiCalculatePage bmiCalclaterPage =
    new BmiCalculatePage(driver);
//直接调用页面的元素无需关心元素定位，并输入数据
bmiCalclaterPage.heightCMS.sendKeys("181");
bmiCalclaterPage.weightKg.sendKeys("80");
bmiCalclaterPage.Calculate.click();
//断言
assertEquals("24.4",
    bmiCalclaterPage.bmi
        .getAttribute("value"));
assertEquals("Normal",
    bmiCalclaterPage.bmi_category
        .getAttribute("value"));
driver.close();
}
}
```

FindBy 注释语法

使用 name 或 id 查询页面元素未必总是有用 我们可能需要使用高级的定位策略如 XPath , CSS 选择器。使用 FindBy 注释语法结合 PageFactory 类就可以定位元素了

修改 BmiCalculatePagte 类的定位方法

```
package com.example.pageobjects;
import org.openqa.selenium.*;
import org.openqa.selenium.support.*;
public class BmiCalculatePage {
    //使用xpath或css定义页面元素
    @FindBy(css = "ul input")
    public WebElement heightCMS;
    @FindBy(xpath = "//ul/li[2]/div/input")
    public WebElement weightKg;
    public WebElement Calculate;
    public WebElement bmi;
    public WebElement bmi_category;
    //构造函数来初始化元素
    public BmiCalculatePage(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }
}
```

```
}
```

FindBy 注释语法

使用@FindBy 的一个缺点就是，每次方法中需要调用页面元素的时候，driver 都需要再次对当前的页面进行查询。这对于元素如果是动态变化的或是使用了 AJAX 技术的页面是非常有用的。

但是，一个应用的页面的元素，总是在那里不会变的情况下。如果我们将查询出来的元素进行缓存将会非常的方便再次调用，在@FindBy 下面接着使用@CacheLookup 注释语法。

```
@FindBy(id = "heightCMS")
@CacheLookup
public WebElement heightField;
```

这将告诉 PageFactory.initElements() 方法，一旦定位后就进行缓存。重复的使用这些元素，将使测试的速度大大加快。

5.3. 使用 PageFactory 类来分离页面操作

在前面的介绍中，创建了一个 BmiCalculatePage 的类，提供了页面所需要的元素。和元素一样，我们也定义了页面的操作和行为。在 BMI Calculator 应用中，我们通过输入身高和体重来计算 BMI。我们可以创建一个方法名为 calculateBmi，直接调用，而不需要单独的来调用元素来操作。

在本秘籍中，让我们修改 BmiCalculatePage 类，把一些常用的操作，封装成方法。

如何实现

1. 把页面的元素都修改为私有属性，体现更好的封装性，增加一个 url 的变量。

```
private WebElement heightCMS;
private WebElement weightKg;
private WebElement Calculate;
private WebElement bmi;
private WebElement bmi_category;
private String url = "d://demo//BMICalculate.html";
```

2. 修改 BmiCalculatePage() 的构造函数并初始化 WebDriver 的实例

```
public BmiCalculatePage() {
    driver = new FirefoxDriver();
    PageFactory.initElements(driver, this);
}
```

3. 新增 load() 和 close() 方法

```
public void load() {  
    this.driver.get(url);  
}  
public void close() {  
    this.driver.close();  
}
```

4. 新增一个 calculateBmi() 方法

```
public void calculateBmi(String height, String weight) {  
    heightCMS.sendKeys(height);  
    weightKg.sendKeys(weight);  
    Calculate.click();  
}
```

5. 新增 getBmi() 和 getBmiCategory() 方法

```
public String getBmi() {  
    return bmi.getAttribute("value");  
}  
public String getBmiCategory() {  
    return bmi_category.getAttribute("value");  
}
```

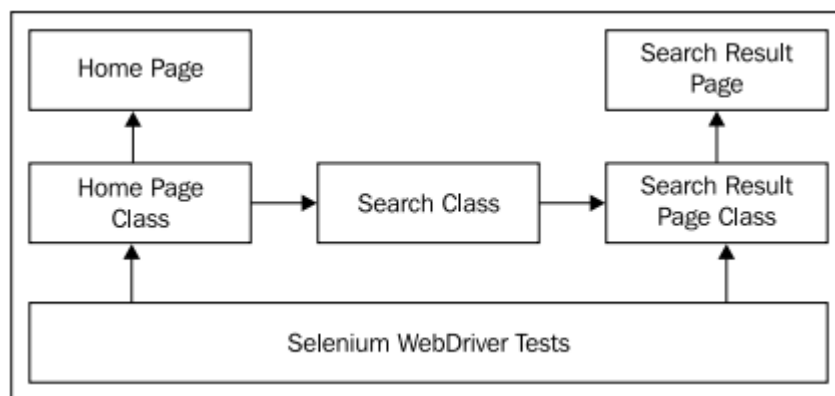
6. 最后修改 BmiCalculatorTests 类, 对于任何操作只需要调用前面封装好的方法

```
package com.example.pageobjects;  
import org.openqa.selenium.*;  
import org.openqa.selenium.firefox.FirefoxDriver;  
import org.junit.Test;  
import static org.junit.Assert.*;  
public class BmiCalculatorTests {  
    @Test  
    public void testBmiCalculatorTest() {  
        BmiCalculatePage bcp = new BmiCalculatePage();  
        //打开页面  
        bcp.load();  
        //计算BMI  
        bcp.calculateBmi("180", "80");  
        //验证BMI值  
        assertEquals("24.7", bcp.getBmi());  
        //验证分类  
        assertEquals("Normal", bcp.getBmiCategory());  
        //关闭页面  
        bcp.close();  
    }  
}
```

5.4. 实现嵌套页面对象实例

到目前为止，我们探索了简单的一个页面应用的页面对象实现。我们也可以在复杂的页面应用中使用页面对象模型来简化测试。

在此秘籍中，我们将为一个电子商务的应用 <http://demo.magentoocommerce.com/> 的查询功能来创建页面对象模型。甚至指定的功能并不属于其页面本身也可以使用页面对象功能。在这个例子中，查询功能是贯穿整个页面的。让我们来看看如何在首页使用查询功能。



每个页面都提供了查询产品的功能。当查询提交后，返回一个新的相应的查询结果页面。

如何实现

1. 创建一个 HomePage 类

```
package com.example.pageobjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.PageFactory;
public class HomePage {
    private static WebDriver driver;
    //用来传递WebDriver
    public static WebDriver driver(){
        return driver;
    }
    public HomePage() {
        driver = new FirefoxDriver();
        PageFactory.initElements(driver, this);
    }

    public void load() {
```

```
        driver.get("http://demo.magentocommerce.com");
    }

    public void close() {
        driver.close();
    }

    public Search Search() {
        Search search = new Search();
        return search;
    }
}
```

2. 创建一个 Search 类，主要用来提供查询的方法

```
package com.example.pageobjects;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
public class Search {
    //定位元素
    private WebElement search;
    @FindBy(css="button.button")
    private WebElement searchButton;

    public Search() {
        PageFactory.initElements(HomePage.driver(), this);
    }
    //输入查询内容的方法
    public SearchResults searchInStore(String query) {
        search.sendKeys(query);
        searchButton.click();
        return new SearchResults(query);
    }
}
```

3. 创建一个 SearchResults 类，提供得到查询结果的方法

```
package com.example.pageobjects;
import java.util.ArrayList;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.PageFactory;
public class SearchResults {
```



```
public SearchResults(String query){
    PageFactory.initElements(HomePage.driver(), this);
}
//得到查询结果
public List<String> getProducts() {
    List<String> products = new ArrayList<String>();
    List<WebElement> productList = HomePage.driver()
        .findElements(By.cssSelector("ul.products-grid >
li"));
    for(WebElement item : productList)
        products.add(item.findElement(By.cssSelector("h2 >
a")));
    return products;
}
```

4. 创建查询功能的主测试方法

```
package com.example.pageobjects;
import org.junit.Test;
import static org.junit.Assert.*;
public class SearchTest {
    @Test
    public void testProductSearch()
    {
        HomePage homePage = new HomePage();
        homePage.load();
        //查询sony的产品
        SearchResults searchResult =
            homePage.Search().searchInStore("sony");
        //验证sony产品的数量
        assertEquals(2, searchResult.getProducts().size());
        assertTrue(searchResult.getProducts()
            .contains("Sony Ericsson W810i"));
        homePage.close();
    }
}
```

5.5. 用.NET 实现对象模型

同 Java 一样，Selenium WebDriver 也提供了 PageFactory 类来实现页面对象模型。但是，在使用的时候确有一点小区别，.NET 需要使用 FindsBy 注释语言来定位元素，并且也不像 Java 那样可以直接使用 name 或 id 来定位元素并作用变量。

在本秘籍中，我们将使用 C#，PageFactory 类来实现 BMI Calculator 的页面对象模型。

如何实现

1. 定义页面对象 BmiCalcPage 类

```
using System;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.PageObjects;
namespace PageFactoryTests
{
    public class BmiCalcPage
    {
        static string Url = "d:\\demo\\BMICalculator.html";

        private static string Title = "BMI Calculator";
        [FindsBy(How = How.Id, Using = "heightCMS")]
        [CacheLookup]
        private IWebElement HeightField;

        [FindsBy(How = How.Id, Using = "weightKg")]
        private IWebElement WeightField;

        [FindsBy(How = How.Id, Using = "Calculate")]
        private IWebElement CalculateButton;

        [FindsBy(How = How.Name, Using = "bmi")]
        private IWebElement BmiField;

        [FindsBy(How = How.Name, Using = "bmi_category")]
        private IWebElement BmiCategoryField;
        private IWebDriver driver;
        public BmiCalcPage() {
            driver = new ChromeDriver(@"C:\ChromeDriver");
            PageFactory.InitElements(driver, this);
        }
        public void Load()
        {
            driver.Navigate().GoToUrl(Url);
        }
        public void Close()
        {
            driver.Close();
        }
        public bool IsLoaded
        {
            get { return driver.Title.Equals(Title); }
        }
    }
}
```

```

    }
    public void CalculateBmi(String height, String weight)
    {
        HeightField.SendKeys (height);
        WeightField.SendKeys (weight);
        CalculateButton.Click();
    }
    public String Bmi
    {
        get { return BmiField.GetAttribute("value"); }
    }
    public String BmiCategory
    {
        get { return BmiCategoryField.GetAttribute("value"); }
    }
}
}

```

2. 创建一个 BMI 计算的测试

```

using NUnit.Framework;
namespace PageFactoryTests
{
    public class BmiCalcTests
    {
        [TestCase]
        public void TestBmiCalculator()
        {
            BmiCalcPage bmiCalcPage = new BmiCalcPage();
            bmiCalcPage.Load();
            Assert.IsTrue (bmiCalcPage.IsLoaded);
            bmiCalcPage.CalculateBmi ("181", "80");
            Assert.AreEqual ("24.4", bmiCalcPage.Bmi);
            Assert.AreEqual ("Normal", bmiCalcPage.BmiCategory);
            bmiCalcPage.Close();
        }
    }
}

```

5.6. 用 Python 实现对象模型

1. 创建 BmiCalcPage 类

```

class bmicalcpage(object):
    def __init__(self, driver):
        self._driver = driver
        self._url = "d:\\demo\\BMICalculator.html"

        self._title = 'BMI Calculator'

    @property
    def is_loaded(self):
        return self._driver.title == self._title

    @property

```

```

def bmi(self):
    bmi_field = self._driver.find_element_by_id('bmi')
    return bmi_field.get_attribute('value')

@property
def bmi_category(self):
    bmi_category_field = self._driver.find_element_by_id('bmi_
category')
    return bmi_category_field.get_attribute('value')

def open(self):
    self._driver.get(self._url)

def calculate(self, height, weight):
    height_field = self._driver.find_element_by_
id('heightCMS')
    weight_field = self._driver.find_element_by_id('weightKg')
    calc_button = self._driver.find_element_by_id('Calculate')

    height_field.send_keys(height)
    weight_field.send_keys(weight)
    calc_button.click()
def close(self):
    self._driver.close()

```

2. 创建 BMI 计算的测试

```

import unittest
from selenium.webdriver.chrome.webdriver import WebDriver
from bmi_calc_page import bmicalcpage
class BmiCalcTest(unittest.TestCase):
    def testCalc(self):
        driver = WebDriver()
        bmi_calc = bmicalcpage(driver)
        bmi_calc.open()
        self.assertEqual(True, bmi_calc.is_loaded)
        bmi_calc.calculate('181', '80')
        self.assertEqual('24.4', bmi_calc.bmi)
        self.assertEqual('Normal', bmi_calc.bmi_category)
        bmi_calc.close()
if __name__ == '__main__':
    unittest.main()

```

5.7. 用 Ruby 实现对象模型

1. 创建 BmiCalcPage 类

```

require 'page-object'
class BmiCalcPage
    include PageObject

```

```
text_field(:height, :id => 'heightCMS')
text_field(:weight, :id => 'weightKg')
button(:calculate, :value => 'Calculate')
text_field(:bmi, :id => 'bmi')
text_field(:bmi_category, :id => 'bmi_category')

def calculate_bmi(height, weight)
  self.height = height
  self.weight = weight
  calculate
end

def open()
  @browser.get "d:\\demo\\BMICalculator.html"
end

def close()
  @browser.close()
end
end
```

2. 创建 BMI 计算的测试

```
require 'rubygems'
require 'watir-webdriver'
require 'test/unit'
require_relative 'bmicalcpage.rb'
class BmiCalcTest < Test::Unit::TestCase
  def test_bmi_calculation
    @driver = Selenium::WebDriver.for :chrome
    bmi_calc = BmiCalcPage.new(@driver)
    bmi_calc.open()
    bmi_calc.calculate_bmi('181','80')
    assert_equal '24.4', bmi_calc.bmi
    assert_equal 'Normal', bmi_calc.bmi_category
    bmi_calc.close()
  end
end
```

6.Selenium 扩展

6.1. 简介

Selenium WebDriver 提供了灵活和健壮的 API 来扩展它的功能和命令，增加一些自定义的方法来构建一个可扩展的自动化测试框架。本章将探讨一下在各种不同的场景中一些重要的 Selenium WebDriver 扩展。

6.2. 创建一个表格的扩展类

Selenium WebDriver 针对 HTML 的元素提供了一个通用的 `WebElement` 类。也提供了对于下拉选择框的 `Select` 类。但是，并没有提供对于 `<table>` 元素处理的类。

如何实现

让我们建立一个关于表格相关的扩展类 `WebTable.java`

```
package com.selenium.util;
import java.util.*;
import java.util.NoSuchElementException;
import org.openqa.selenium.*;
public class WebTable {
    private WebElement webTable;
    public WebTable(WebElement webElement) {
        this.webTable = webElement;
    }
}
```

增加一个得到表格中行数的方法

```
// 得到表格的行数
public int getRowCount() {
    List<WebElement> rowCounts =
        webTable.findElements(By.tagName("tr"));
    return rowCounts.size();
}
```

增加一个得到指定行的列数，传入一个指定行数为参数，从 0 开始。

```
// 得到指定行的列数
public int getColCount(int rowIdx) {
    try {
        List<WebElement> rowCounts =
            webTable.findElements(By.tagName("tr"));
        // 取得当前的tr
        WebElement rowNum = rowCounts.get(rowIdx);
        // 计算当前的td数
        List<WebElement> colCounts =
            rowNum.findElements(By.tagName("td"));
        return colCounts.size();
    } catch (NoSuchElementException e) {
        throw new NoSuchElementException("Failed to get the cell");
    }
}
```

增加一个得到指定单元格内容的方法，传入指定的行数，和列数作为参数。

```
// 得到指定单元格的内容
public String getCellText(int rowIdx, int colIdx) {
    try {
        List<WebElement> rowCounts = webTable
            .findElements(By.tagName("tr"));
        // 得到对应的行数
        WebElement currentRow = rowCounts.get(rowIdx);
        List<WebElement> td = currentRow.findElements(By.tagName("td"));
        // 取得对应的单元格
        WebElement cell = td.get(colIdx);
        return cell.getText();
    } catch (Exception e) {
        throw new NoSuchElementException("Failed to get the cell");
    }
}
```

你还可以创建更多的符合自己要求的方法来扩展对于<table>元素的处理，下面创建一个测试类来测试我们上面写的方法对不对。

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

public class TestWebTable {
```

```

@Test
public void testTableWebElement() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.w3school.com.cn/tags/tag_table.asp");
    //得到table元素
    WebTable webTable =
        new WebTable(driver.findElement(By.tagName("table")));
    //验证总行数
    assertEquals(10, webTable.getRowCount());
    //验证第2行的例数
    assertEquals(4, webTable.getColCount(1));
    //验证第4行，第3列的单元格内容
    assertEquals("规定表格边框的宽度.", webTable.getCellText(3, 2));
    driver.close();
}
}

```

6.3. 给元素设置属性

当测试需要操作元素属性的时候，给元素设置属性这个方法就会变得非常的有用。例如，有时候对某些输入框 `sendKeys()` 不能工作的时候，你可能需要去设置他的 `value` 属性来解决这样的问题。

如何实现

创建一个设置属性的方法。需要传入元素对象，属性名，属性值做为参数。

```

public static void setAttribute(WebElement element,
    String attributeName, String value) {
    WrapsDriver wrappedElement = (WrapsDriver) element;
    JavascriptExecutor driver =
        (JavascriptExecutor) wrappedElement.getWrappedDriver();

    driver.executeScript("arguments[0].setAttribute(arguments[1],arguments[2])",
        , element, attributeName, value);
}

```

写一个例子来测试这个方法，Util 为我自己的工具类，

```

@Test
public void testSetAttribute() {

```



```
WebDriver driver = new FirefoxDriver();
driver.get("http://www.baidu.com");
WebElement wb = driver.findElement(By.id("kw"));
Util.setAttribute(wb, "value", "test" );
}
```

6.4. 高亮元素

在测试过程中，是没有办法来高亮元素的。高亮元素可以非常方便的来调试脚本。

如何实现

```
// 高亮元素
public static void highlightElement(WebElement element){
    for (int i = 0; i < 3; i++) {
        WrapsDriver wrappedElement = (WrapsDriver) element;
        JavascriptExecutor driver = (JavascriptExecutor) wrappedElement
            .getWrappedDriver();
        //为元素设置style来高亮
        try {
            driver.executeScript(
                "arguments[0].setAttribute('style',arguments[1]);",
                element, "color: green; border: 2px solid yellow;");
            //取消高亮将style清掉
            Thread.sleep(800);
            driver.executeScript(
                "arguments[0].setAttribute('style',arguments[1]);",
                element, "");
            Thread.sleep(800);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

6.5. 截取元素图片

TakeScreenshot 接口可以截取整个页面的作为图片，但是不能截取某个元素。我们可以扩展一下这个截取图片的功能让它可以截取某一个元素。

如何实现

```
// 页面元素截图
public static File captureElement(WebElement element) throws
Exception {
    WebDriver wrapsDriver = (WebDriver) element;
    // 截图整个页面
    File screen = ((TakesScreenshot) wrapsDriver
        .getWrappedDriver())
        .getScreenshotAs(OutputType.FILE);
    BufferedImage img = ImageIO.read(screen);
    // 获得元素的高度和宽度
    int width = element.getSize().getWidth();
    int height = element.getSize().getHeight();
    // 创建一个矩形使面上面的高度，和宽度
    Rectangle rect = new Rectangle(width, height);
    // 得到元素的坐标
    Point p = element.getLocation();
    BufferedImage dest = img.getSubimage(p.getX()
        , p.getY(), rect.width,
        rect.height);
    // 存为png格式
    ImageIO.write(dest, "png", screen);
    return screen;
}
```

也一个测试方来来验证，截取百度首页面里新闻超链接的为图片

```
@Test
public void testTableWebElement() {
    WebDriver driver = new FirefoxDriver();
    driver.manage().window().maximize();
    driver.get("http://www.baidu.com");
    WebElement wb = driver.findElement(By.cssSelector("#nv a"));
    try {
        FileUtils.copyFile(Util.captureElement(wb), new File("c:\\a.png"));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

7.测试 HTML5 的网页应用

在本章中，我们将讨论

- ◆ 处理 HTML5 视频播放
- ◆ 网页存储 – 测试本地存储
- ◆ 网页存储 – 测试 Session 存储
- ◆ 清除本地和 Session 的存储

7.1. 简介

HTML5 越来越流行，主流的浏览器现在都支持了 HTML5。越来越多的应用使用了 HTML5 的元素如 canvas,video 等等，还有网页存储功能更增加了用户的网络体验。越来越多的开发者在使用这样的标准，所以我们也学习如何使用自动化技术来测试它们。

Selenium WebDriver 支持在指定的浏览器上测试 HTML5。但是，我们也可以使用 JavaScript 来测试这些功能，这就可以工作在任何的浏览器上了。

在本章中，我们将关注如何使用 Selenium WebDriver 来测试 HTML5 的最重要的功能。

7.2. 处理 HTML5 的视频播放

到目前为止，如何在网页上展示一个视频还没有一个标准。多数的浏览器使用了插件如 Flash 来播放视频。但是，不同的浏览器就需要使用不同的插件。

HTML5 定义了一个新的元素<video>，指定了一个标准的方式来嵌入电影片段，IE9+，Firefox,Opera,Chrome 都已支持。

在此秘籍中，我们将探索如何将<video>来自动化测试，该元素提供了 JavaScript 接口和多种方法和属性。

如何实现

创建一个测试，名为 `testHTML5VideoPlayer` 来测试`<video>`元素，我们将使用 `JavaScriptExecutor` 类和`<video>`元素进行交互。

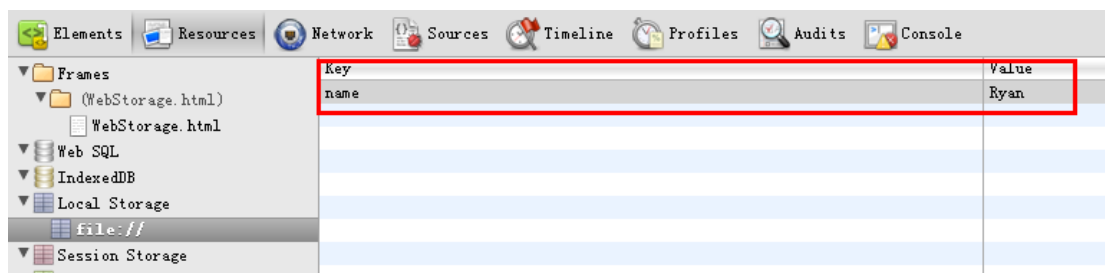
```
package com.example.tests;
import static org.junit.Assert.*;
import org.junit.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Selenium2 {
    @Test
    public void testHTML5Video() throws InterruptedException {
        WebDriver driver = new FirefoxDriver();
        driver.get("http://videojs.com/");
        WebElement video = driver.findElement
            (By.id("home_video_html5_api"));
        JavascriptExecutor jse = (JavascriptExecutor)driver;
        //获得视屏的URL
        String source = (String) jse.executeScript
            ("return arguments[0].currentSrc;", video);
        assertEquals("http://video-js.zencoder.com/oceans-clip.webm"
            , source);
        //播放视屏
        jse.executeScript("return arguments[0].play()", video);
        Thread.sleep(5000);
        //暂停视屏
        jse.executeScript("arguments[0].pause()", video);
    }
}
```

在调用 `JavaScript` 的时候 `currentSrc` 为视频播放的 `URL`，`play()` 播放视频，`pause()` 为暂停

7.3. 网页存储 – 测试本地存储

HTML5 引入一个更加安全和快速的方式来利用用户的浏览器存储数据到本地。之前，我们都是通过 `Cookies` 来完成。它可以存储更加大量的数据也不影响网站的性能。数据通过键值对进行存储，只有本自的网页应用才可以去存储去访问它。

HTML5 提供了一个 `localStorage` 的 `JavaScript` 的接口来存储数据并且没有过期时间。这个数据即便是关闭了浏览器也是不会删除的。你可以通过 `Google Chrome` 开发工具中的 **Resources** 标签来查看



如何实现

```
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 testHTML5Video() throws InterruptedException {
        WebDriver driver = new FirefoxDriver();
        driver.get("D:\\demo\\WebStorage.html");
        JavascriptExecutor jse = (JavascriptExecutor)driver;
        WebElement localStorageButton =
            driver.findElement(By.cssSelector("input"));
        //保存localStorage
        localStorageButton.click();
        //通过js获取localStorage
        String name = (String)jse.executeScript
            ("return localStorage.name");
        assertEquals("Ryan ---- Show By Local Storage ", name);
        //删除storage
        WebElement deleteStorageButton =
            driver.findElement(By.cssSelector("input + input"));
        deleteStorageButton.click();
        int temp = (Integer)jse.executeScript
            ("return localStorage.length");
        int length = (int)temp;
        assertEquals(0, length); //长度为0, 验证被删除
        driver.close();
    }
}
```

7.4. 网页存储 – 测试 Session 存储

session storage 仅将数据存储在 session 中，当浏览器窗口被关闭的时候，数据就被删除了。我们可以使用 sessionStorage 方法来访问存储的数据。

在本秘籍中，我们将验证网页存储在 session 中的数据是我们所期望的。

如何实现

还是先增加一个通过 sessionStorage 存储一个 name，然后再进行验证

```
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 testHTML5Video() throws InterruptedException {
        WebDriver driver = new FirefoxDriver();
        driver.get("D:\\demo\\WebStorage.html");
        JavascriptExecutor jse = (JavascriptExecutor)driver;
        WebElement sessionStorageButton = driver.findElement
            (By.cssSelector("input+input+input"));
        sessionStorageButton.click();
        String sessionStorage = (String) jse.executeScript
            ("return sessionStorage.name");
        assertEquals("Ryan ---- Show By Session Storage", sessionStorage);
        driver.close();
    }
}
```

7.5. 清除 Local 和 Session 的存储数据

在这里简单的介绍一下如何清除 Local 与 Session 产生的存储数据，对于测试来说，一般来说我们不会主动去清除都是通页面触发某些事件，所以这里仅作为了解一下。

在 JS 中，我们可以调用 localStorage.removeItem(name);

sessionStorage.removeItem(name); 来删除指的数据。要想清理全部的数据就调用 localStorage.clear(); session.clear();