

一、准备篇——webdriver 的工作原理

通过研究 selenium-webdriver 的源码，发现其实 webdriver 的实现原理并不高深莫测无法揣度。在这里以 webdriver ruby binding 的 firefox-webdriver 实现为例，简单介绍一下 webdriver 的工作原理。

当测试脚本启动 firefox 的时候，selenium-webdriver 会首先在新线程中启动 firefox 浏览器。如果测试脚本指定了 firefox 的 profile，那么就以该 profile 启动，否则的话就新启 1 个 profile，并启动 firefox；

firefox 一般是以-no-remote 的方法启动，启动后 selenium-webdriver 会将 firefox 绑定到特定的端口，绑定完成后该 firefox 实例便作为 webdriver 的 remote server 存在；

客户端(也就是测试脚本)创建 1 个 session，在该 session 中通过 http 请求向 remote server 发送 restful 的请求，remote server 解析请求，完成相应操作并返回 response；

客户端接受 response，并分析其返回值以决定是转到第 3 步还是结束脚本；

这就是 webdriver 的工作流程，看起来很复杂实际上当了解了 webdriver 的实现原理后，理解上述问题应该比较简单。

webdriver 是按照 server - client 的经典设计模式设计的。

server 端就是 remote server，可以是任意的浏览器。当我们的脚本启动浏览器后，该浏览器就是 remote server，它的职责就是等待 client 发送请求并做出相应；

client 端简单说来就是我们的测试代码，我们测试代码中的一些行为，比如打开浏览器，转跳到特定的 url 等操作是以 http 请求的方式发送给被测试浏览器，也就是 remote server；remote server 接受请求，并执行相应操作，并在 response 中返回执行状态、返回值等信息；

举个实际的例子，下面代码的作用是”命令”firefox 转跳到 google 主页：

```
driver = Selenium::WebDriver.for :firefoxdriver.navigate.to "http://google.com"在执行 driver.navigate.to “http://google.com” 这句代码时，client，也就是我们的测试代码向 remote server 发送了如下的请求：
```

POST session/285b12e4-2b8a-4fe6-90e1-c35cba245956/urlpost_data

{"url":"http://google.com"} 通过 post 的方式请求

localhost:port/hub/session/session_id/url 地址，请求浏览器完成跳转 url 的操作。

如果上述请求是可接受的，或者说 remote server 是实现了这个接口，那么 remote server 会跳转到该 post data 包含的 url,并返回如下的 response

```
{"name":"get","sessionId":"285b12e4-2b8a-4fe6-90e1-c35cba245956","status":0,"value":""}
```

该 response 中包含如下信息

name: remote server 端的实现的方法的名称，这里是 get，表示跳转到指定 url;

sessionId: 当前 session 的 id;

status: 请求执行的状态码，非 0 表示未正确执行，这里是 0，表示一切 ok 不许担心;

value: 请求的返回值，这里返回值为空，如果 client 调用 title 接口，则该值应该是当前页面的 title;

如果 client 发送的请求是定位某个特定的页面元素，则 response 的返回值可能是这样的:

```
{"name":"findElement","sessionId":"285b12e4-2b8a-4fe6-90e1-c35cba245956","status":0,"value":{"ELEMENT":{"2192893e-f260-44c4-bdf6-7aad3c919739"}}}
```

name,sessionId, status 跟上面的例子是差不多的，区别是该请求的返回值是 ELEMENT:{2192893e-f260-44c4-bdf6-7aad3c919739}，表示定位到元素的 id，通过该 id，client 可以发送如 click 之类的请求与 server 端进行交互。

那么 remote server 端的这些功能是如何实现的呢？答案是浏览器实现了 webdriver 的统一接口，这样 client 就可以通过统一的 restful 的接口去进行浏览器的自动化操作。目前 webdriver 支持 ie, chrome, firefox, opera 等主流浏览器，其主要原因是这些浏览器实现了 webdriver 约定的各种接口。

具体见 [http://code.google.com/p/seleniu ... l#Command_Reference](http://code.google.com/p/seleniu...l#Command_Reference)。

二、实战篇——快速入门

selenium webdriver 学习历程（一）-----快速开始

学习 selenium 已经两年了，从 1.X 到 2.X，一直在关注它。中间由于工作原因中断了一段时间，但是一直无法割舍，最近又去官网看了一下，更新还挺快的。selenium1.X 的时代将被取代，selenium-webdriver 的大航海时代开始了。。。

安装 selenium webdriver（eclipse+jdk+selenium webdriver2.20+firefox 15）

- 1、安装 firefox，本人使用 firefox10。确保 firefox 安装在默认环境下（不是的话会报错）。
- 2、安装 jdk，确保安装了 jdk，本人喜欢使用 java。但 selenium webdriver 也支持其它语言，如 ruby、python、C#等。
- 3、安装 eclipse，个人喜好。
- 4、安装 selenium webdriver。解压下载的 selenium webdriver 包，可以在 eclipse 建一个 user library，便与项目的引入。

第一个 test

现在以第一个 selenium webdriver 的 test 来感受一下它的魅力。

Java 代码 ☆

```
1. import org.openqa.selenium.By;
2. import org.openqa.selenium.WebDriver;
3. import org.openqa.selenium.WebElement;
4. import org.openqa.selenium.firefox.FirefoxDriver;
5.
6.
7. public class FirstExampe {
8.
9.
10.     public static void main(String[] args) {
11.         WebDriver driver = new FirefoxDriver();
12.
13.
14.         driver.get("http://www.google.com.hk");
15.         WebElement element = driver.findElement(By.name("q"));
```

```
16.     element.sendKeys("hello Selenium!");
17.     element.submit();
18.     try {
19.         Thread.sleep(3000);
20.     } catch (InterruptedException e) {
21.         e.printStackTrace();
22.     }
23.     System.out.println("Page title is: " + driver.getTitle());
24.
25.     driver.quit();
26. }
27.
28. }
```

正常运行后，这几行代码将会打开 firefox 浏览器，然后转跳到 google 首页。在搜索框中输入 hello Selenium 并提交搜索结果。等待 3 秒后会在命令行打印出当前页面的 title，输出如下：

Java 代码 ☆

1. Page title is: hello Selenium! - Google 搜尋

并关闭 ff 浏览器。

selenium webdriver 学习（二）——对浏览器的简单操作

selenium webdriver 对浏览器的简单操作

打开一个测试浏览器

对浏览器进行操作首先需要打开一个浏览器，接下来才能对浏览器进行操作。但要注意的是，因为 Chrome Driver 是 [Chromium](#) 项目自己支持和维护的，所以你必需另外下载安装 Chrome Driver，详细介绍查下他们的 [wiki](#)。

Java 代码 ☆

```

1. import java.io.File;
2.
3. import org.openqa.selenium.WebDriver;
4. import org.openqa.selenium.firefox.FirefoxBinary;
5. import org.openqa.selenium.firefox.FirefoxDriver;
6. import org.openqa.selenium.ie.InternetExplorerDriver;
7.
8. public class OpenBrowsers {
9.
10.
11.     public static void main(String[] args) {
12.         //打开默认路径的 firefox
13.         WebDriver driver = new FirefoxDriver();
14.
15.         //打开指定路径的 firefox,方法 1
16.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
fox\\firefox.exe");
17.         WebDriver dr = new FirefoxDriver();
18.
19.         //打开指定路径的 firefox,方法 2
20.         File pathToFirefoxBinary = new File("D:\\Program Files\\Mozilla Firefox\\fi
refox.exe");
21.         FirefoxBinary firefoxbin = new FirefoxBinary(pathToFirefoxBinary);
22.         WebDriver driver1 = new FirefoxDriver(firefoxbin,null);
23.
24.         //打开 ie
25.         WebDriver ie_driver = new InternetExplorerDriver();
26.
27.         //打开 chrome
28.         System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe");
29.         System.setProperty("webdriver.chrome.bin",
30.             "C:\\Documents and Settings\\gongjf\\Local Settings"
31.             +"\\Application Data\\Google\\Chrome\\Application\\
chrome.exe");
32.
33.
34.     }
35.
36. }

```

打开指定路径 ie 和 chrome 方法和 ff 一样。

打开 1 个具体的 url

打开一个浏览器后，我们需要跳转到特定的 url 下，看下面代码：

Java 代码 ☆

```
1. import org.openqa.selenium.WebDriver;
2. import org.openqa.selenium.firefox.FirefoxDriver;
3.
4. public class OpenUrl {
5.     public static void main(String []args){
6.         String url = "http://www.51.com";
7.         WebDriver driver = new FirefoxDriver();
8.
9.         //用 get 方法
10.        driver.get(url);
11.
12.        //用 navigate 方法，然后再调用 to 方法
13.        driver.navigate().to(url);
14.    }
15. }
```

如何关闭浏览器

测试完成后，需要关闭浏览器

Java 代码 ☆

```
1. import org.openqa.selenium.WebDriver;
2. import org.openqa.selenium.firefox.FirefoxDriver;
3.
4. public class CloseBrowser {
5.     public static void main(String []args){
6.         String url = "http://www.51.com";
7.         WebDriver driver = new FirefoxDriver();
8.
9.         driver.get(url);
10.
11.        //用 quit 方法
12.        driver.quit();
13.
14.        //用 close 方法
15.        driver.close();
16.    }
17. }
```

如何返回当前页面的 url 和 title

有时候我们需要返回当前页面的 url 或者 title 做一些验证性的操作等。代码如下：

Java 代码 ☆

```
1. import org.openqa.selenium.WebDriver;
2. import org.openqa.selenium.firefox.FirefoxDriver;
3.
4. public class GetUrlAndTitle {
5.     public static void main(String []args){
6.         String url = "http://www.51.com";
7.         WebDriver driver = new FirefoxDriver();
8.
9.         driver.get(url);
10.
11.         //得到 title
12.         String title = driver.getTitle();
13.
14.         //得到当前页面 url
15.         String currentUrl = driver.getCurrentUrl();
16.
17.         //输出 title 和 currenturl
18.         System.out.println(title+"\n"+currentUrl);
19.
20.     }
21. }
```

其他方法

- getWindowHandle() 返回当前的浏览器的窗口句柄
- getWindowHandles() 返回当前的浏览器的所有窗口句柄
- getPageSource() 返回当前页面的源码

小结

从上面代码可以看出操作浏览器的主要方法都来自 `org.openqa.selenium.WebDriver` 这个接口中。看了一下源代码这些方法都是在 `org.openqa.selenium.remote.RemoteWebDriver` 这个类中实现的，然后不同浏览的 `driver` 类继承 `RemoteWebDriver`。

selenium webdriver 学习（三）-----执行 js 脚本

在用 selenium 1.X 的时候常常会用到 `getEval()`方法来执行一段 js 脚本来对页面进行处理，以处理一些遇到的问题。当然 selenium webdriver 也提供这样的一个方法：`executeScript()`

Java 代码 ☆

```
1. import org.openqa.selenium.JavascriptExecutor;
2. import org.openqa.selenium.WebDriver;
3.
4. public class SimpleExample {
5.
6.
7.     public static void main(String[] args) {
8.         WebDriver driver = new FirefoxDriver();
9.
10.        ((JavascriptExecutor)driver).executeScript("alert(\"hello,this is a alert!\")");
11.    }
12.
13. }
```

上面是一个最简单的例子，打开一个浏览器，然后弹层一个 `alert` 框。注意这里的 `driver` 要被强制转换成 `JavascriptExecutor`。

下面演示在打开 51.com 首页如何得到帐号输入框中显示的字符，并打印输出。

Java 代码 ☆

```
1. import org.openqa.selenium.JavascriptExecutor;
2. import org.openqa.selenium.WebDriver;
3.
4. public class FirstExampe {
```



```
5.
6.
7.   public static void main(String[] args) {
8.       WebDriver driver = new FirefoxDriver();
9.       driver.get("http://www.51.com");
10.      String js = "var user_input = document.getElementById(\"passport_51_user\
    \").title;return user_input;";
11.
12.      String title = (String)((JavascriptExecutor)driver).executeScript( js);
13.      System.out.println(title);
14.  }
15.
16. }
```

输出结果为:

Java 代码 ☆

1. 用户名/彩虹号/邮箱

selenium webdriver 学习（四）-----定位页面元素

seleniumwebdriver 定位页面元素 findElementBy

selenium-webdriver 提供了强大的元素定位方法，支持以下三种方法。

- 单个对象的定位方法
- 多个对象的定位方法
- 层级定位

定位单个元素

在定位单个元素时,selenium-webdriver 提示了如下一些方法对元素进行定位。

- By.className(className)
- By.cssSelector(selector)
- By.id(id)

- By.linkText(linkText)
- By.name(name)
- By.partialLinkText(linkText)
- By.tagName(name)
- By.xpath(xpathExpression)

注意：selenium-webdriver 通过 `findElement()`\`findElements()`等 `find` 方法调用"By"对象来定位 和查询元素。By 类只是提供查询的方式进行分类。`findElement` 返回一个元素对象否则抛出异常，`findElements` 返回符合条件的元素 List，如果不存在符合条件的就返回一个空的 list。

使用 `className` 进行定位

当所定位的元素具有 `class` 属性的时候我们可以通过 `classname` 来定位该元素。

下面的例子定位了 51.com 首页上 `class` 为"username"的 li。

Java 代码 ☆

```
1. import org.openqa.selenium.WebDriver;
2. import org.openqa.selenium.WebElement;
3.
4. import org.openqa.selenium.By;
5.
6. public class ByClassName {
7.
8.
9.     public static void main(String[] args) {
10.         WebDriver driver = new FirefoxDriver();
11.         driver.get("http://www.51.com");
12.         WebElement element = driver.findElement(By.className("username"));
13.         System.out.println(element.tagName());
14.
15.     }
16. }
```

输出结果：

Java 代码 ☆

```
1. li
```

使用 id 属性定位

51.com 首页的帐号输入框的 html 代码如下:

Java 代码 ☆

1. `<input id="passport_51_user" type="text" value="" tabindex="1" title="用户名/彩虹号/邮箱"`
2. `name="passport_51_user">`

在下面的例子中我们用 id 定位这个输入框，并输出其 title,借此也可以验证代码是否工作正常。

Java 代码 ☆

```
1. import org.openqa.selenium.By;
2. import org.openqa.selenium.WebDriver;
3. import org.openqa.selenium.WebElement;
4. import org.openqa.selenium.firefox.FirefoxDriver;
5.
6. public class ByUserId {
7.
8.     /**
9.      * @param args
10.     */
11.    public static void main(String[] args) {
12.        // TODO Auto-generated method stub
13.        WebDriver dr = new FirefoxDriver();
14.        dr.get("http://www.51.com");
15.
16.        WebElement element = dr.findElement(By.id("passport_51_user"));
17.        System.out.println(element.getAttribute("title"));
18.    }
19.
20. }
```

输出结果:

Java 代码 ☆

1. 用户名/彩虹号/邮箱

使用 name 属性定位

51.com 首页的帐号输入框的 html 代码如下:

Java 代码 ☆

1. `<input id="passport_51_user" type="text" value="" tabindex="1" title="用户名/彩虹号/邮箱"`
2. `name="passport_51_user">`

使用 name 定位

Java 代码 ☆

1. `WebElement e = dr.findElement(By.name("passport_51_user"));`

使用 css 属性定位

51.com 首页的帐号输入框的 html 代码如下:

Java 代码 ☆

1. `<input id="passport_51_user" type="text" value="" tabindex="1" title="用户名/彩虹号/邮箱"`
2. `name="passport_51_user">`

使用 css 定位

Java 代码 ☆

1. `WebElement e1 = dr.findElement(By.cssSelector("#passport_51_user"));`

使用其他方式定位

在定位 link 元素的时候, 可以使用 link 和 link_text 属性;

另外还可以使用 tag_name 属性定位任意元素;

定位多个元素

上面提到 `findElements()`方法可以返回一个符合条件的元素 List 组。看下面例子。

Java 代码 ☆

```
1. import java.io.File;
2. import java.util.List;
3.
4. import org.openqa.selenium.By;
5. import org.openqa.selenium.WebDriver;
6. import org.openqa.selenium.WebElement;
7. import org.openqa.selenium.firefox.FirefoxBinary;
8. import org.openqa.selenium.firefox.FirefoxDriver;
9.
10. public class FindElementsStudy {
11.
12.     /**
13.      * @author gongjf
14.      */
15.     public static void main(String[] args) {
16.         WebDriver driver = new FirefoxDriver();
17.         driver.get("http://www.51.com");
18.
19.         //定位到所有<input>标签的元素，然后输出他们的 id
20.         List<WebElement> element = driver.findElements(By.tagName("input"));
21.         for (WebElement e : element){
22.             System.out.println(e.getAttribute("id"));
23.         }
24.
25.         driver.quit();
26.     }
27. }
```

输出结果:

Java 代码 ☆

```
1. passport_cookie_login
2. gouri
3. passport_login_from
4. passport_51_user
5. passport_51_password
6. passport_qq_login_2
7. btn_reg
8. passport_51_ishidden
```

9. passport_auto_login


上面的代码返回页面上所有 `input` 对象。很简单，没什么可说的。

层级定位

层级定位的思想是先定位父元素，然后再从父元素中精确定位出其我们需要选取的子元素。

层级定位一般的应用场景是无法直接定位到需要选取的元素，但是其父元素比较容易定位，通过定位父元素再遍历其子元素选择需要的目标元素，或者需要定位某个元素下所有的子元素。

下面的代码演示了如何使用层级定位 `class` 为"login"的 `div`，然后再取得它下面的所有 `label`，并打印出他们的文本

Java 代码 

```
1. import java.io.File;
2. import java.util.List;
3.
4. import org.openqa.selenium.By;
5. import org.openqa.selenium.WebDriver;
6. import org.openqa.selenium.WebElement;
7. import org.openqa.selenium.firefox.FirefoxBinary;
8. import org.openqa.selenium.firefox.FirefoxDriver;
9.
10. public class LayerLocator {
11.
12.     /**
13.      * @author gongjf
14.      */
15.     public static void main(String[] args) {
16.
17.         WebDriver driver = new FirefoxDriver();
18.         driver.get("http://www.51.com");
19.
20.         //定位 class 为"login"的 div，然后再取得它下面的所有 label，并打印出他们的
           值
21.         WebElement element = driver.findElement(By.className("login"));
22.         List<WebElement> el = element.findElements(By.tagName("label"));
```

```
23.     for(WebElement e : el)
24.     System.out.println(e.getText());
25.
26.     }
27.
28. }
```

输出结果:

Java 代码 ☆

1. 帐号:
2. 密码:
3. 隐身
4. 下次自动登录

定位页面元素 over 了，下次写一下对 frame 的处理。

[selenium webdriver 学习（五）-----iframe 的处理](#)

[如何定位 frame 中元素](#)

有时候我们在定位一个页面元素的时候发现一直定位不了，反复检查自己写的定位器没有任何问题，代码也没有任何问题。这时你就要看一下这个页面元素是否在一个 iframe 中，这可能就是找不到的原因之一。如果你在一个 default content 中查找一个在 iframe 中的元素，那肯定是找不到的。反之你在一个 iframe 中查找另一个 iframe 元素或 default content 中的元素，那必然也定位不到。

selenium webdriver 中提供了进入一个 iframe 的方法:

```
WebDriver org.openqa.selenium.WebDriver.TargetLocator.frame(String nameOrId)
```

也提供了一个返回 default content 的方法:

```
WebDriver org.openqa.selenium.WebDriver.TargetLocator.defaultContent()
```

这样使我们面对 iframe 时可以轻松应对。

以下面的 html 代码为例，我们看一下处理 iframe。

Html 代码 ☆

```
1. main.html
2.
3. <html>
4.   <head>
5.     <title>FrameTest</title>
6.   </head>
7.   <body>
8.     <div id = "id1">this is a div!</div>
9.     <iframe id = "frame" frameborder="0" scrolling="no" style="left:0;position:
      absolute;" src = "frame.html"></iframe>
10.  </body>
11. </html>
12.
13.
14.
15. frame.html
16.
17. <html>
18.   <head>
19.     <title>this is a frame!</title>
20.   </head>
21.   <body>
22.     <div id = "div1">this is a div, too!</div>
23.     <label>input:</label>
24.     <input id = "input1"></input>
25.   </body>
26. </html>
```

Java 代码 ☆

```
1. import org.openqa.selenium.By;
2. import org.openqa.selenium.WebDriver;
3. import org.openqa.selenium.firefox.FirefoxDriver;
4.
5. public class FameStudy {
6.
7.
8.   public static void main(String[] args) {
9.     WebDriver dr = new FirefoxDriver();
10.    String url = "\\Your\\Path\\to\\main.html";
```



```

11.     dr.get(url);
12.
13.     //在 default content 定位 id="id1"的 div
14.     dr.findElement(By.id("id1"));
15.
16.     //此时，没有进入到 id="frame"的 frame 中时，以下两句会报错
17.     dr.findElement(By.id("div1"));//报错
18.     dr.findElement(By.id("input1"));//报错
19.
20.     //进入 id="frame"的 frame 中，定位 id="div1"的 div 和 id="input1"的输入
    框。
21.     dr.switchTo().frame("frame");
22.     dr.findElement(By.id("div1"));
23.     dr.findElement(By.id("input1"));
24.
25.     //此时，没有跳出 frame，如果定位 default content 中的元素也会报
    错。
26.     dr.findElement(By.id("id1"));//报错
27.
28.     //跳出 frame,进入 default content;重新定位 id="id1"的 div
29.     dr.switchTo().defaultContent();
30.     dr.findElement(By.id("id1"));
31. }
32.
33. }

```

`switch_to` 方法会 new1 个 `TargetLocator` 对象，使用该对象的 `frame` 方法可以将当前识别的”主体”移动到需要定位的 `frame` 上去。

[selenium webdriver 学习（六）-----如何得到弹出窗口](#)

在 selenium 1.X 里面得到弹出窗口是一件比较麻烦的事，特别是新开窗口没有 `id`、`name` 的时候。当时还整理了处理了几种方法，详见：

<http://seleniumcn.cn/read.php?tid=791>。在 selenium webdriver 中得到新开窗口相对简单的多，它无关新开窗口的 `id`、`name` 等属性。以下面的 html 为例：

Html 代码 ☆

```

1. <span style="white-space: normal; background-color: #ffffff;">test.html</span>
2.

```

```
3.
4. <html>
5.
6.   <head><title>Test Popup Window</title></head>
7.
8.   <body>
9.
10.     <a id = "51" href = "http://www.51.com/" target = "_blank">Let's go!</a>
11.
12.   </body>
13.
14. </html>
```

下面的代码演示了如何去得到弹出的新窗口

Java 代码 ☆

```
1. import java.util.Iterator;
2. import java.util.Set;
3.
4. import org.openqa.selenium.By;
5. import org.openqa.selenium.WebDriver;
6. import org.openqa.selenium.firefox.FirefoxDriver;
7.
8. public class PopupWindowTest {
9.
10.
11.   /**
12.    * @author gongjf
13.    */
14.   public static void main(String[] args) {
15.     System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
fox\\firefox.exe");
16.     WebDriver dr = new FirefoxDriver();
17.     String url = "\\Your\\Path\\to\\main.html";
18.     dr.get(url);
19.     dr.findElement(By.id("51")).click();
20.     //得到当前窗口的句柄
21.     String currentWindow = dr.getWindowHandle();
22.     //得到所有窗口的句柄
23.     Set<String> handles = dr.getWindowHandles();
24.     Iterator<String> it = handles.iterator();
25.     while(it.hasNext()){
26.       if(currentWindow == it.next()) continue;
```

```
27.         WebDriver window = dr.switchTo().window(it.next());
28.         System.out.println("title,url = "+window.getTitle()+","+window.getCurrent
           tUrl());
29.     }
30. }
31.
32. }
```

输出结果:

Java 代码 ☆

1. title,url = 51.com 真人配对玩游戏,http://www.51.com/

捕获或者说定位弹出窗口的关键在于获得弹出窗口的句柄。（句柄，我的理解是浏览器窗口的一个唯一标识，记得以前玩"按键精灵"也有这玩样。）

在上面的代码里，使用 `windowhandle` 方法来获取当前浏览器窗口的句柄，使用了 `windowhandles` 方法获取所有弹出的浏览器窗口的句柄，然后通过排除当前句柄的方法来得到新开窗口的句柄。

在获取新弹出窗口的句柄后，使用 `switchto.window(newwindow_handle)` 方法，将新窗口的句柄作为参数传入既可捕获到新窗口了。

[selenium webdriver 学习（七）-----如何处理 alert、confirm、prompt 对话框](#)

[alertpromptconfirmseleniumwebdriver](#)

alert、confirm、prompt 这样的 js 对话框在 selenium1.X 时代也是难啃的骨头，常常要用 autoit 来帮助处理。

试用了一下 selenium webdriver 中处理这些对话框十分方便简洁。以下面 html 代码为例：

Html 代码 ☆

1. Dialogs.html

Html 代码 ☆

```
1. <html>
2.
3.   <head>
4.
5.     <title>Alert</title>
6.
7.   </head>
8.
9.   <body>
10.
11.     <input id = "alert" value = "alert" type = "button" onclick = "alert('欢迎！请
    按确认继续！');" />
12.   <input id = "confirm" value = "confirm" type = "button" onclick = "confirm('确
    定吗？');" />
13.   <input id = "prompt" value = "prompt" type = "button" onclick = "var name = p
    rompt('请输入你的名字:','请输入
14.
15. 你的名字'); document.write(name) " />
16.
17.
18. </body>
19.
20. </html>
```

以上 html 代码在页面上显示了三个按钮，点击他们分别弹出 alert、confirm、prompt 对话框。如果在 prompt 对话框中输入文字点击确定之后，将会刷新页面，显示出这些文字。

selenium webdriver 处理这些弹层的代码如下：

Java 代码 ☆

```
1. import org.openqa.selenium.Alert;
2. import org.openqa.selenium.By;
3. import org.openqa.selenium.WebDriver;
4. import org.openqa.selenium.firefox.FirefoxDriver;
5.
6. public class DialogsStudy {
7.
8.     /**
9.      * @author gongjf
10.     */
11.     public static void main(String[] args) {
12.         // TODO Auto-generated method stub
13.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
fox\\firefox.exe");
14.         WebDriver dr = new FirefoxDriver();
15.         String url = "file:///C:/Documents and Settings/gongjf/桌面
/selenium_test/Dialogs.html";// "/Your/Path/to/main.html"
16.         dr.get(url);
17.
18.         //点击第一个按钮，输出对话框上面的文字，然后又掉
19.         dr.findElement(By.id("alert")).click();
20.         Alert alert = dr.switchTo().alert();
21.         String text = alert.getText();
22.         System.out.println(text);
23.         alert.dismiss();
24.
25.         //点击第二个按钮，输出对话框上面的文字，然后点击确认
26.         dr.findElement(By.id("confirm")).click();
27.         Alert confirm = dr.switchTo().alert();
28.         String text1 = confirm.getText();
29.         System.out.println(text1);
30.         confirm.accept();
31.
32.         //点击第三个按钮，输入你的名字，然后点击确认，最后
33.         dr.findElement(By.id("prompt")).click();
34.         Alert prompt = dr.switchTo().alert();
35.         String text2 = prompt.getText();
36.         System.out.println(text2);
37.         prompt.sendKeys("jarvi");
38.         prompt.accept();
39.
40.     }
41.
42. }
```

从以上代码可以看出 `dr.switchTo().alert()`;这句可以得到 `alert\confirm\prompt` 对话框的对象，然后运用其方法对它进行操作。对话框操作的主要方法有：

- `getText()` 得到它的文本值
- `accept()` 相当于点击它的"确认"
- `dismiss()` 相当于点击"取消"或者叉掉对话框
- `sendKeys()` 输入值，这个 `alert\confirm` 没有对话框就不能用了，不然会报错。

[selenium webdriver 学习（八）-----如何操作 select 下拉框](#)

下面我们来看一下 `selenium webdriver` 是如何来处理 `select` 下拉框的，以 `http://passport.51.com/reg2.5p` 这个页面为例。这个页面中有 4 个 下拉框，下面演示 4 种选中下拉框选项的方法。`select` 处理比较简单，直接看代码吧：)

Java 代码 ☆

```
1. import org.openqa.selenium.By;
2. import org.openqa.selenium.WebDriver;
3. import org.openqa.selenium.WebElement;
4. import org.openqa.selenium.firefox.FirefoxDriver;
5. import org.openqa.selenium.support.ui.Select;
6.
7. public class SelectsStudy {
8.
9.     /**
10.      * @author gongjf
11.      */
12.     public static void main(String[] args) {
13.         // TODO Auto-generated method stub
14.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
15.         fox\\firefox.exe");
16.         WebDriver dr = new FirefoxDriver();
17.         dr.get("http://passport.51.com/reg2.5p");
18.         //通过下拉列表中选项的索引选中第二项，即 2011 年
19.         Select selectAge = new Select(dr.findElement(By.id("User_Age")));
20.         selectAge.selectByIndex(2);
21.
22.         //通过下拉列表中的选项的 value 属性选中"上海"这一项
23.         Select selectShen = new Select(dr.findElement(By.id("User_Shen")));
```

```

24.     selectShen.selectByValue("上海");
25.
26.     //通过下拉列表中选项的可见文本选中"浦东"这一项
27.     Select selectTown = new Select(dr.findElement(By.id("User_Town")));
28.     selectTown.selectByVisibleText("浦东");
29.
30.     //这里只是想遍历一下下拉列表所有选项，用 click 进行选中选项
31.     Select selectCity = new Select(dr.findElement(By.id("User_City")));
32.     for(WebElement e : selectCity.getOptions())
33.         e.click();
34. }
35.
36. }

```

从上面可以看出，对下拉框进行操作时首先要定位到这个下拉框，new 一个 Selcet 对象，然后对它进行操作。

[selenium webdriver 学习（九）-----如何操作 cookies](#)

Web 测试中我们会经常会接触到 Cookies，一个 Cookies 主要属性有"所在域、name、value、有效日期和路径",下面来讲一下怎么操作 Cookies。

Java 代码 ☆

```

1. import java.util.Set;
2.
3. import org.openqa.selenium.Cookie;
4. import org.openqa.selenium.WebDriver;
5. import org.openqa.selenium.firefox.FirefoxDriver;
6.
7. public class CookiesStudy {
8.
9.     /**
10.      * @author gongjf
11.      */
12.     public static void main(String[] args) {
13.         // TODO Auto-generated method stub
14.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
fox\\firefox.exe");

```

```

15.     WebDriver dr = new FirefoxDriver();
16.     dr.get("http://www.51.com");
17.
18.     //增加一个 name = "name",value="value"的 cookie
19.     Cookie cookie = new Cookie("name", "value");
20.     dr.manage().addCookie(cookie);
21.
22.     //得到当前页面下所有的 cookies，并且输出它们的所在域、name、
    value、有效日期和路径
23.     Set<Cookie> cookies = dr.manage().getCookies();
24.     System.out.println(String.format("Domain -> name -> value -> expiry -
    > path"));
25.     for(Cookie c : cookies)
26.         System.out.println(String.format("%s -> %s -> %s -> %s -> %s",
27.             c.getDomain(), c.getName(), c.getValue(),c.getExpiry(),c.getPath()));

28.
29.
30.     //删除 cookie 有三种方法
31.
32.     //第一种通过 cookie 的 name
33.     dr.manage().deleteCookieNamed("CookieName");
34.     //第二种通过 Cookie 对象
35.     dr.manage().deleteCookie(cookie);
36.     //第三种全部删除
37.     dr.manage().deleteAllCookies();
38. }

```

上面的代码首先在页面中增加了一个 cookie,然后遍历页面的所有 cookies，并输出他们的主要属性。最后就是三种删除 cookie 的方法。遍历 cookies 输出的结果：

Java 代码 ☆

```

1. Domain -> name -> value -> expiry -> path
2. .51.com -> FO_RFLP -> %7CaHR0cDovL3d3dy41MS5jb20v%7C%7C%7C -
    > null -> /
3. .51.com -> __utmz -
    > 67913429.1331544776.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none) -
    > Tue Sep 11 05:32:56 CST 2012 -> /
4. www.51.com -> name -> value -> Tue Mar 12 17:33:00 CST 2030 -> /
5. www.51.com -> PHPSESSID -> 51d37fc72eb0ea66e4ef1971b688698b -> null -
    > /

```


6. .51.com -> __utma -
> 67913429.453585250.1331544776.1331544776.1331544776.1 -
> Wed Mar 12 17:32:56 CST 2014 -> /
7. www.51.com -> www_cookie_adv -> 1 -> Mon Mar 12 18:32:55 CST 2012 -> /
8. .51.com -> __utm -> 67913429 -> null -> /
9. www.51.com -> NSC_xxx -> 44595a553660 -> null -> /
10. .51.com -> __utmb -> 67913429.1.10.1331544776 -
> Mon Mar 12 18:02:56 CST 2012 -> /
11. www.51.com -> www_jiaoyou_guide -> 0c83c0b5f569512d5a832bf0b4397a05 -
> null -> /

[selenium webdriver 学习（十）-----如何把一个元素拖放到另一个元素里面](#)

Q 群里有时候会有人问，selenium webdriver 怎么实现把一个元素拖放到另一个元素里面。这一节总一下元素的拖放。

下面这个页面是一个演示拖放元素的页面，你可以把左右页面中的条目拖放到右边的 div 框中。

<http://koyoz.com/demo/html/drag-drop/drag-drop.html>

现在来看看 selenium webdriver 是怎么实现 drag and drop 的吧。let's go!

Java 代码 ☆

```
1. import org.openqa.selenium.By;
2. import org.openqa.selenium.WebDriver;
3. import org.openqa.selenium.WebElement;
4. import org.openqa.selenium.firefox.FirefoxDriver;
5. import org.openqa.selenium.interactions.Actions;
6.
7. public class DragAndDrop {
8.
9.     /**
10.      * @author gongjf
11.      */
12.     public static void main(String[] args) {
13.         // TODO Auto-generated method stub
14.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
15.         fox\\firefox.exe");
16.         WebDriver dr = new FirefoxDriver();
17.         dr.get("http://koyoz.com/demo/html/drag-drop/drag-drop.html");
```

```

17.
18. //首先 new 出要拖入的页面元素对象和目标对象， 然后进行拖入。
19. WebElement element = dr.findElement(By.id("item1"));
20. WebElement target = dr.findElement(By.id("drop"));
21. (new Actions(dr)).dragAndDrop(element, target).perform();
22.
23. //利用循环把其它 item 也拖入
24. String id="item" ;
25. for(int i=2;i<=6;i++){
26.     String item = id+i;
27.     (new Actions(dr)).dragAndDrop(dr.findElement(By.id(item)), target).perform();
28. }
29. }
30.
31. }

```

代码很简单，需要注意的是(new Actions(dr)).dragAndDrop(element, target).perform();这句话中，dragAndDrop(element, target)这个方法是定义了“点击 element 元素对象，然后保持住，直到拖到目标元素对象里面才松开”这一系列动作的 Actions,如果你不调用 perform()方法，这个 Actions 是不会执行的。over!

[selenium webdriver 学习（十一）-----如何等待页面元素加载完成](#)

web 的自动化测试中，我们经常会遇到这样一种情况：当我们的程序执行时需要页面某个元素，而此时这个元素还未加载完成，这时我们的程序就会报错。怎么办？等待。等待元素出现后再进行对这个元素的操作。

在 selenium-webdriver 中我们用两种方式进行等待：明确的等待和隐性的等待。

明确的等待

明确的等待是指在代码进行下一步操作之前等待某一个条件的发生。最不好的情况是使用 Thread.sleep()去设置一段确认的时间去等待。但为什么说最不好呢？因为一个元素的加载时间有长有短，你在设置 sleep 的时间之前要自己把握长短，太短

容易超时，太长浪费时间。selenium webdriver 提供了一些方法帮助我们等待正好需要等待的时间。利用 `WebDriverWait` 类和 `ExpectedCondition` 接口就能实现这一点。

下面的 html 代码实现了这样的一种效果：点击 click 按钮 5 秒钟后，页面上会出现一个红色的 div 块。我们需要写一段自动化脚本去捕获这个出现的 div，然后高亮之。

Html 代码 ☆

```
1. Wait.html
2.
3. <html>
4.   <head>
5.     <title>Set Timeout</title>
6.     <style>
7.       .red_box {background-
      color: red; width = 20%; height: 100px; border: none;}
8.     </style>
9.     <script>
10.      function show_div(){
11.        setTimeout("create_div()", 5000);
12.      }
13.
14.      function create_div(){
15.        d = document.createElement('div');
16.        d.className = "red_box";
17.        document.body.appendChild(d);
18.      }
19.    </script>
20.  </head>
21.  <body>
22.    <button id = "b" onclick = "show_div()">click</button>
23.  </body>
24. </html>
```

下面的代码实现了高亮动态生成的 div 块的功能:

Java 代码 ☆

```
1. import org.openqa.selenium.By;
2. import org.openqa.selenium.JavascriptExecutor;
3. import org.openqa.selenium.WebDriver;
4. import org.openqa.selenium.WebElement;
5. import org.openqa.selenium.firefox.FirefoxDriver;
6. import org.openqa.selenium.support.ui.ExpectedCondition;
7. import org.openqa.selenium.support.ui.WebDriverWait;
8.
9.
10. public class WaitForSomething {
11.
12.     /**
13.      * @author gongjf
14.      */
15.     public static void main(String[] args) {
16.         // TODO Auto-generated method stub
17.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
18.             fox\\firefox.exe");
19.         WebDriver dr = new FirefoxDriver();
20.         String url = "file:///C:/Documents and Settings/gongjf/桌面
21.             /selenium_test/Wait.html"; // "/Your/Path/to/Wait.html"
22.         dr.get(url);
23.         WebDriverWait wait = new WebDriverWait(dr, 10);
24.         wait.until(new ExpectedCondition<WebElement>(){
25.             @Override
26.             public WebElement apply(WebDriver d) {
27.                 return d.findElement(By.id("b"));
28.             }
29.         }).click();
30.
31.         WebElement element = dr.findElement(By.cssSelector(".red_box"));
32.         ((JavascriptExecutor)dr).executeScript("arguments[0].style.border = \"5px so
33.             lid yellow\"", element);
34.
35.     }
36. }
```

上面的代码 `WebDriverWait` 类的构造方法接受了一个 `WebDriver` 对象和一个等待最长时间（10 秒）。然后调用 `until` 方法，其中重写了 `ExpectedCondition` 接口中的

apply 方法，让其返回一个 WebElement,即加载完成的元素，然后点击。默认情况下，WebDriverWait 每 500 毫秒调用一次 ExpectedCondition，直到有成功的返回，当然如果超过设定的值还没有成功的返回，将抛出异常。

隐性等待

隐性等待是指当要查找元素，而这个元素没有马上出现时，告诉 WebDriver 查询 Dom 一定时间。默认值是 0,但是设置之后，这个时间将在 WebDriver 对象实例整个生命周期都起作用。上面的代码就变成了这样：

Java 代码 ☆

```
1. import java.util.concurrent.TimeUnit;
2.
3. import org.openqa.selenium.By;
4. import org.openqa.selenium.JavascriptExecutor;
5. import org.openqa.selenium.WebDriver;
6. import org.openqa.selenium.WebElement;
7. import org.openqa.selenium.firefox.FirefoxDriver;
8. import org.openqa.selenium.support.ui.ExpectedCondition;
9. import org.openqa.selenium.support.ui.WebDriverWait;
10.
11.
12. public class WaitForSomething {
13.
14.     /**
15.      * @author gongjf
16.      */
17.     public static void main(String[] args) {
18.         // TODO Auto-generated method stub
19.         System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
20.         fox\\firefox.exe");
21.         WebDriver dr = new FirefoxDriver();
22.
23.         //设置 10 秒
24.         dr.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
25.
26.         String url = "file:///C:/Documents and Settings/gongjf/桌面
27.         /selenium_test/Wait.html";// "/Your/Path/to/Wait.html"
```

```

28.     /*WebDriverWait wait = new WebDriverWait(dr,10);
29.     wait.until(new ExpectedCondition<WebElement>(){
30.         @Override
31.         public WebElement apply(WebDriver d) {
32.             return d.findElement(By.id("b"));
33.         }}).click();*/
34.     dr.findElement(By.id("b")).click();
35.     WebElement element = dr.findElement(By.cssSelector(".red_box"));
36.     ((JavascriptExecutor)dr).executeScript("arguments[0].style.border = \"5px so
    lid yellow\"",element);
37.
38. }
39. }

```

两者选其一，第二种看起来一劳永逸呀。哈哈

[selenium webdriver 学习（十二）-----如何利用 selenium-webdriver 截图](#)

在自动化测试中常常会用到截图功能。最近用了一下 selenium-webdriver 的截图功能还算不错，可以截取页面全图，不管页面有多长。

下面的代码演示了如何使用 webdriver 进行截图：

Java 代码 ☆

```

1. import java.io.File;
2. import java.io.IOException;
3.
4. import org.apache.commons.io.FileUtils;
5. import org.openqa.selenium.OutputType;
6. import org.openqa.selenium.TakesScreenshot;
7. import org.openqa.selenium.WebDriver;
8. import org.openqa.selenium.firefox.FirefoxDriver;
9. public class ShotScreen {
10.
11.     /**

```

```

12.  * @author gongjf
13.  * @throws IOException
14.  * @throws InterruptedException
15.  */
16.  public static void main(String[] args) throws IOException, InterruptedException
    n {
17.
18.      System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fire
fox\\firefox.exe");
19.      WebDriver dr = new FirefoxDriver();
20.      dr.get("http://www.51.com");
21.
22.      //这里等待页面加载完成
23.      Thread.sleep(5000);
24.      //下面代码是得到截图并保存在 D 盘下
25.      File screenShotFile = ((TakesScreenshot)dr).getScreenshotAs(OutputType.F
ILE);
26.      FileUtils.copyFile(screenShotFile, new File("D:/test.png"));}}

```

看了一下 `OutputType` 接口和 `TakesScreenshot` 接口，吐槽一下，貌似这两个接口不是同一个开发写的或者注释没有更新什么的。在 `OutputType` 里面的注释说：

Java 代码 ☆

```

1.  /**
2.   * Defines the output type for a screenshot. See org.openqa.selenium.Screenshot f
or usage and
3.   * examples.
4.   ...

```

然后在那找了半天的 `org.openqa.selenium.Screenshot` 接口，晕，后来想应该是 `org.openqa.selenium.TakesScreenshot`。

在 `TakesScreenshot` 里有如下注释：

Java 代码 ☆

```

1.  /**
2.   * Capture the screenshot and store it in the specified location.
3.   *
4.   * <p>For WebDriver extending TakesScreenshot, this makes a best effort

```

5. * depending on the browser to return the following in order of preference:
6. *
7. * Entire page
8. * Current window
9. * Visible portion of the current frame
10. * The screenshot of the entire display containing the browser
11. *
12. *
13. * <p>For WebElement extending TakesScreenshot, this makes a best effort
14. * depending on the browser to return the following in order of preference:
15. * - The entire content of the HTML element
16. * - The visible portion of the HTML element
17. *
18. * @param <X> Return type for getScreenshotAs.
19. * @param target target type, @see OutputType
20. * @return Object in which is stored information about the screenshot.
21. * @throws WebDriverException on failure.
22. */

试了一下截取 WebElement 最终发现 WebElement 接口没有实现这个类。搞了半天也只是会了截取页面的全图。截取当前的 frame 也截取的页面全图。难道这个功能没有完善，好吧，这样说自我安慰一下。

selenium-webdriver 面向接口编程，找一个需要的功能还真是挺难的。

[selenium webdriver 学习（十三）-----如何利用 Actions 类模拟鼠标和键盘的操作](#)

在 [selenium webdriver 学习（十）-----如何把一个元素拖放到另一个元素里面的时候](#)，用到了一个 Actions 类。这一节主要分析一下这个 Actions 类。

这个 actions 类，主要定义了一些模拟用户的鼠标 mouse，键盘 keyboard 操作。对于这些操作，使用 perform()方法进行执行。

actions 类可以完成单一的操作，也可以完成几个操作的组合。

单一的操作

单一的操作是指鼠标和键盘的一个操作。如鼠标左键按下、弹起或输入一个字符串等。

前面涉及到鼠标键盘操作的一些方法，都可以使用 `actions` 类中的方法实现，比如：`click`，`sendKeys`。

Java 代码 ☆

1. `WebElement element = dr.findElement(By.id("test"));`
2. `WebElement element1 = dr.findElement(By.id("test1"));`
3. `element.sendKeys("test");`
4. `element1.click;`

用 `Actions` 类就可以这样实现：

Java 代码 ☆

1. `//新建一个 action`
2. `Actions action=new Actions(driver);`
3. `//操作`
4. `WebElement element=dr.findElement(By.id("test"));`
5. `WebElement element1=dr.findElement(By.id("su"));`
6. `action.sendKeys(element,"test").perform();`
7. `action.moveToElement(element1);`
8. `action.click().perform();`

看起来用 `Actions` 类实现 `click` 和 `sendKeys` 有点烦索

组合操作

组合操作就是几个动作连在一起进行操作。如对一个元素的拖放。

Java 代码 ☆

1. `(new Actions(dr)).dragAndDrop(dr.findElement(By.id(item)), target).perform();`

可以直接调用 `dragAndDrip()`方法，也可以像下面演示的一样把几个操作放在一起实现

Java 代码 ☆

1. Action dragAndDrop = builder.clickAndHold(someElement)
2. .moveToElement(otherElement)
3. .release(otherElement)
4. .build().perform();

其他鼠标或键盘操作方法可以具体看一下 API 里面的
org.openqa.selenium.interactions.Actions 类

[selenium webdriver 学习（十四）-----如何处理 table](#)

以前在 selenium RC 里面有一个 getTable 方法，是得到一个单元格中的文本。其详细描述如下：

Java 代码 ☆

1. /** Gets the text from a cell of a table. The cellAddress syntax tableLocator.row.column
2. , where row and column start at 0.
3. @param tableCellAddress a cell address, e.g. "foo.1.4"
4. @return the text from the specified cell
5. */
6. String getTable(String tableCellAddress);

就是传入一个参数，这个参数的格式必须是 tableLocator.row.column，如"foo.1.4"，foo 用于得到 table 对象，1.4 代表在 table 里第 1 行第 4 列。行、列从 0 开始。

在 selenium webdriver 里，没有这样的方法，也就是说没有专门操作 table 的类。但我们可以自己封装一个，这并不难。以上面的 getTable 方法为例，我们自己也可以创建这样功能的一个方法。

Java 代码 ☆

1. public String getCellText(By by,String tableCellAddress)

我叫它 `getCellText`,它有两个参数, 第一个是 `By` 对象用于得到 `table` 对象, `tableCellAddress` 如 "1.4",代表在 `table` 里第 1 行第 4 列。行、列从 0 开始。

以下面 `html` 代码为例:

Html 代码 ☆

```
1. <html>
2.   <head>
3.     <title>Table</title>
4.
5.   </head>
6.   <body>
7.     <table border="1" id="myTable">
8.       <tr>
9.         <th>Heading(row 0 ,cell 0)</th>
10.        <th>Another Heading(row 0 ,cell 1)</th>
11.        <th>Another Heading(row 0 ,cell 2)</th>
12.      </tr>
13.      <tr>
14.        <td>row 1, cell 0</td>
15.        <td>row 1, cell 1</td>
16.        <td>row 1, cell 2</td>
17.      </tr>
18.      <tr>
19.        <td>row 2, cell 0</td>
20.        <td>row 2, cell 1</td>
21.        <td>row 2, cell 2</td>
22.      </tr>
23.    </table>
24.  </body>
25. </html>
```

示例代码如下:

Java 代码 ☆

```
1. import java.util.List;
2. import org.openqa.selenium.By;
3. import org.openqa.selenium.NoSuchElementException;
4. import org.openqa.selenium.WebDriver;
5. import org.openqa.selenium.WebElement;
6. import org.openqa.selenium.firefox.FirefoxDriver;
7.
8. public class Table {
9.
10.     /**
11.      * @author gongjf
12.      */
13.     private WebDriver driver;
14.     Table(WebDriver driver){
15.         this.driver = driver;
16.     }
17.
18.     /** 从一个 table 的单元格中得到文本值. 参数 tableCellAddress 的格式为
19.     row.column, 行列从 0 开始.
20.     @param by 用于得到 table 对象
21.     @param tableCellAddress 一个单元格地址, 如. "1.4"
22.     @return 从一个 table 的单元格中得到文本值
23.     */
24.     public String getCellText(By by,String tableCellAddress) {
25.         //得到 table 元素对象
26.         WebElement table = driver.findElement(by);
27.         //对所要查找的单元格位置字符串进行分解, 得到其对应行、列。
28.         int index = tableCellAddress.trim().indexOf('.');
29.         int row = Integer.parseInt(tableCellAddress.substring(0, index));
30.         int cell = Integer.parseInt(tableCellAddress.substring(index+1));
31.         //得到 table 表中所有行对象, 并得到所要查询的行对象。
32.         List<WebElement> rows = table.findElements(By.tagName("tr"));
33.         WebElement theRow = rows.get(row);
34.         //调用 getCell 方法得到对应的列对象, 然后得到要查询的文本。
35.         String text = getCell(theRow, cell).getText();
36.         return text;
37.     }
38.     private WebElement getCell(WebElement Row,int cell){
39.         List<WebElement> cells;
40.         WebElement target = null;
41.         //列里面有"<th>"、"<td>"两种标签, 所以分开处理。
```

```

42.     if(Row.findElements(By.tagName("th")).size(>0){
43.         cells = Row.findElements(By.tagName("th"));
44.         target = cells.get(cell);
45.     }
46.     if(Row.findElements(By.tagName("td")).size(>0){
47.         cells = Row.findElements(By.tagName("td"));
48.         target = cells.get(cell);
49.     }
50.     return target;
51.
52. }
53.
54.
55. public static void main(String[] args) {
56.     WebDriver driver;
57.     System.setProperty("webdriver.firefox.bin", "D:\\Program Files\\Mozilla Fir
58.     efox\\firefox.exe");
59.     driver = new FirefoxDriver();
60.     driver.get("file:///C:/Documents and Settings/Gongjf/桌面
61.     /selenium_test/table.html");
62.
63.     Table table = new Table(driver);
64.     By by = By.id("myTable");
65.     String address = "0.2";
66.
67.     System.out.println(table.getCellText(by, address));
68.
69. }
70. }

```

运行代码将输出

Java 代码 ☆

1. Another Heading(row 0 ,cell 2)

ps: 这里我只是以得到一个 table 中单元格的文本为例，但是从代码可以看出，对 table 的基本操作都有涉及到。有用到的同学可以自己包装一个完整的 table 类。

[selenium webdriver 学习（十五）-----如何处理 FirefoxProfile](#)

这一节主要涉及 selenium webdriver 处理 Firefox profile 的一些知识。

什么是 Firefox profile

要了解 Firefox profile 它详细介绍了 Firefox profile。在 Firefox 里，如何管理 Firefox profile 看完它们，相信你对 Firefox profile 会有所了解。好了，必备的知识准备完了，让我们来看看 selenium webdriver 是怎么操作 Firefox profile 的吧。

设置 profile 中的一个 preference

Java 代码 ☆

1. `FirefoxProfile profile = new FirefoxProfile();`
2. `profile.setPreference("aaa", "bbbb");`
3. `WebDriver driver = new FirefoxDriver(profile);`

以上代码在 Firefox Profile 文件中设置一个名 aaa，值为 bbb 的 preference。(ps:这个 preference 只是一个举例，没有任何意义。要看 firefox profile 有哪些 preference, 可以在 firefox 浏览器地址栏中输入:about:config). 代码运行后，在 firefox 浏览器地址栏中输入:about:config，可以看到它。

启用已经存在的 profile

首先来了解一下为什么要已经存在的 profile，其中一个原因是已经存在的 profile 里面保存有 cookie 等信息，可以保持用户的登录状态。

启动已经存在的 profile，因 profile 不同而有两种方法。一种是如果这个 profile 使用 firefox 配置管理器（Firefox's profile manager）而已经存在了。我们用下面的方法：

Java 代码 ☆

1. ProfilesIni allProfiles = new ProfilesIni();
2. FirefoxProfile profile = allProfiles.getProfile("WebDriver");
3. WebDriver driver = new FirefoxDriver(profile);

另一种是没有在自己的 firefox 里面注册过的，比如从另一台机器中的 firefox 得到的，我们可以用下面的代码：

Java 代码 ☆

1. File profileDir = new File("path/to/your/profile");
2. FirefoxProfile profile = new FirefoxProfile(profileDir);
3. WebDriver driver = new FirefoxDriver(profile);

临时指定插件

有时我们需要临时让启动的 firefox 带一个插件，如 firebug,来定位问题等。首先我们要下载这个插件的 xpi 安装包。剩下的就让 selenium webdriver 来完成，如下：

Java 代码 ☆

1. File file = new File("path/to/your/firebug-1.8.1.xpi");

Java 代码 ☆

1. FirefoxProfile firefoxProfile = new FirefoxProfile();
2. firefoxProfile.addExtension(file);
3. firefoxProfile.setPreference("extensions.firebug.currentVersion", "1.8.1"); //避免启动画面
4. WebDriver driver = new FirefoxDriver(firefoxProfile);

这样启动的 firefox 中就安装了插件 firebug.

启用默认情况下被 firefox 禁用的功能

以本地事件例，很简单直接设置为 true 就可以了。

Java 代码 ☆

1. `FirefoxProfile profile = new FirefoxProfile();`
2. `profile.setEnableNativeEvents(true);`
3. `WebDriver driver = new FirefoxDriver(profile);`

其它设置见 [selenium webdriver API](#) 中的 `org.openqa.selenium.firefox.FirefoxProfile`.

启用 **firefox** 代理

这个更简单，直接上代码了。

Java 代码 ☆

1. `String PROXY = "localhost:8080";`//如果不是本机，localhost 替换成 IP 地址
- 2.
3. `org.openqa.selenium.Proxy proxy = new org.openqa.selenium.Proxy();`
4. `proxy.setHttpProxy(PROXY)`
5. `.setFtpProxy(PROXY)`
6. `.setSslProxy(PROXY);`
7. `DesiredCapabilities cap = new DesiredCapabilities();`
8. `cap.setPreference(CapabilityType.PROXY, proxy);`
9. `WebDriver driver = new FirefoxDriver(cap);`

over !