**Selenium2.17.0 + Maven3.0.4 + TestNG6.3.1**

**前提：**

1. Jdk，环境变量配置
2. eclipse SDK
3. 以下红字请根据实际情况调整

**动手：**

1. Maven下载：[**http://labs.mop.com/apache-mirror//maven/binaries/apache-maven-3.0.4-bin.zip**](http://labs.mop.com/apache-mirror/maven/binaries/apache-maven-3.0.4-bin.zip)
2. 安装
3. 添加环境变量：
* 添加变量名为*M2\_HOME*，变量值为Maven的安装目录*D:\apache-maven-3.0*
* 在Path变量的末尾加上*%M2\_HOME%\bin;*
1. 新建个文件夹test，在cmd进入到test目录：
* 运行mvn archetype:generate
* 回车，看到：“Define value for groupId: :”停止
* 输入groupID，格式：“com.随意A”，回车
* 输入artifactId，格式：“随意B”，回车
* 回车
* 输入package，格式：“com. 随意A. 随意B”,回车
* 输入Y,回车
* 看到BUILD SUCCESS，成功
1. 进入test文件夹，有个pom.xml打开，

<project xmlns=<http://maven.apache.org/POM/4.0.0> xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>

 <groupId>com.随意A</groupId>

 <artifactId>随意B</artifactId>

 <version>1.0-SNAPSHOT</version>

 <packaging>jar</packaging>

 <name>随意B</name>

 <url>http://maven.apache.org</url>

* 把下面的内容粘贴进去，覆盖掉“junit”

<properties>

 <project.build.sourceEncoding>GBK</project.build.sourceEncoding>

 </properties>

 <build>

 <finalName>随意B </finalName>

 <plugins>

 <plugin>

 <groupId>org.apache.maven.plugins</groupId>

 <artifactId>maven-compiler-plugin</artifactId>

 <version>2.3.2</version>

 <configuration>

 <source>1.6</source>

 <target>1.6</target>

 <encoding>GBK</encoding>

 </configuration>

 </plugin>

 </plugins>

 </build>

 <dependencies>

 <dependency>

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

 <artifactId>selenium-java</artifactId>

 <version>2.17.0</version>

 <scope>test</scope>

 </dependency>

 <dependency>

 <groupId>org.testng</groupId>

 <artifactId>testng</artifactId>

 <version>6.3.1</version>

 <scope>test</scope>

 </dependency>

</dependencies>

1. 在test文件夹下，新建一个批处理文件：build.bat
* 粘贴内容

@echo off

setlocal

call mvn eclipse:clean

call mvn -npu eclipse:eclipse -Dwtpversion=1.0 -DdownloadSources=true

pause

endlocal

* 运行build.bat
1. 打开eclipse—》help，安装TestNG plug-in
* 搜索TestNG，具体此处不述
* 提示：
* For Eclipse 3.4 and above, enter <http://beust.com/eclipse>.
* For Eclipse 3.3 and below, enter <http://beust.com/eclipse1>.
1. Classpath设置，打开eclipse—》 window—》



Name：M2\_REPO

Path：C:/Documents and Settings/Administrator/.m2/repository

注：Path为Maven本地仓库

1. 重启eclipse—》import—》existing projects into workspace，导入项目“随意B”
2. Maven是约定优于配置，测试脚本都建在src/test/java下面。

**(一)   Selenium2.0 VS Selenium1.0**

我们已经有了Selenium1.0为什么还需要Selenium2.0呢？

Selenium1.0不能处理一下事件：

1)         本机键盘和鼠标事件

2)         同源策略XSS/HTTP（S）

3)         弹出框，对话框（基本身份认证，自签名的证书和文件上传/下载）

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

同时Selenium2.0进行了架构的调整和升级：

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

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

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

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

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

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

**(二)   WebDriver**

**a)         WebDirver小实例**

WebDriver是一个用来进行复杂重复的web自动化测试的工具。意在提供一种比Selenium1.0更简单易学，有利于维护的API。它没有和任何测试框架进行绑定，所以他可以很好的在单元测试和main方法中调用。一旦创建好一个Selenium工程，你马上会发现WebDriver和其他类库一样：它是完全独立的，你可以直接使用而不需要考虑其他配置，这个Selenium RC是截然相反的。

下面我们就开始进入WebDriver的实际应用，首先大家下载好eclipse，还有从selenium官方网站下载[selenium2.0](http://http/seleniumhq.org/download/)的jar包和[chromedriver](http://code.google.com/p/chromedriver/downloads/list)的驱动文件。构建整个工程如下模式：



 现在我们开始写代码：

package demo;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class Selenium2Example {
 public static void main(String[] args) {
 //创建一个WebDriver实例
 WebDriver driver = new FirefoxDriver();
 // 访问google
 driver.get("http://www.google.com");
 // 另一种访问方法
 // driver.navigate().to("http://www.google.com");
 // 找到文本框
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 //提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("Selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 //关闭浏览器
 driver.quit();
 }
}

**b)         介绍WebDriver的Drivers**

WebDriver是测试中必须要写的关键接口的名字，但是他有多种实现，包括下面：

***HtmlUnit Driver****：*这是目前最快，最轻巧的实施的WebDriver。顾名思义，这是基于HtmlUnit。 HtmlUnit是Java一个WebBrowser没有图形用户界面的实现。

*用法：*

WebDriver driver = **new** HtmlUnitDiver();

*例子：*

package demo;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class DemoUseHtmlUnit {
 public static void main(String[] args) {
 //创建一个WebDriver实例
 WebDriver driver = new HtmlUnitDriver();
 // 访问google
 driver.get("http://www.google.com");
 // 找到文本框
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 //提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 //关闭浏览器
 driver.quit();
 }
 }

*优点：*

  WebDriver最快的实现方式

  纯java跨平台性好

  支持javascript

*缺点*：

  模拟其他浏览器的JavaScript行为（见下文）

现今流行的浏览器没有使用HtmlUnit所使用的JavaScript引擎。如果使用HtmlUnit测试JavaScript的结果可能会与实际使用浏览器不同。当我们说“JAVASCRIPT”其实我们的意思是“JavaScript和DOM”。虽然DOM是由W3C定义的，但是每个浏览器都有自己的方式使用JavaScript来实现DOM。 HtmlUnit对JavaScript操作DOM具有良好的支持和完整实现，给人留下了深刻的印象，但它和任何其他浏览器一样：它有自己的方式和W3C标准的主流浏览器的DOM实现差异，尽管其有模仿其他浏览器的能力。

支持javascript：

HtmlUnitDriver driver = new HtmlUnitDriver(true);

注：HtmlUnitDriver默认是模拟firefox3.6来处理javascript。（具体作用还未研究）

***Firefox Driver****：*这是一个比较好的WebDriver，目前已经支持到了10.0版本。运行Firefox需要加载Selenium WebDriver.xpi插件。

*用法：*

WebDriver driver = new FirefoxDriver();

*例子：*

package demo;
import java.io.File;
import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxProfile;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class DemoUseFirefox {
 public static void main(String[] args) {
 //创建一个WebDriver实例
 WebDriver driver = new FirefoxDriver();
 // 访问google
 driver.get("http://www.google.com");
 // 找到文本框
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 //提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 //关闭浏览器
 driver.quit();
 }
}

*优点：*

  在真正的浏览器上运行，且支持javascript

  运行速度快于IE

*缺点：*

  运行速度低于HtmlUnit Driver

*修改Firefox的配置文件：*FirefoxDriver可以通过自己设置Firefox的配置文件，如启动浏览器时，加载Firebug插件。

 File file = new File(".\\res\\firebug-1.9.1-fx.xpi");
 FirefoxProfile firefoxProfile = new FirefoxProfile();
 try {
 firefoxProfile.addExtension(file);
 } catch (IOException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }
 firefoxProfile.setPreference("extensions.firebug.currentVersion","1.9.1");

*例子：*

package demo;
import java.io.File;
import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxProfile;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class DemoUseFirefox {
 public static void main(String[] args) {
 // 创建一个WebDriver实例
 File file = new File(".\\res\\firebug-1.9.1-fx.xpi");// firebug插件的本地位置
 FirefoxProfile firefoxProfile = new FirefoxProfile();
 try {
 firefoxProfile.addExtension(file);
 } catch (IOException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }
 firefoxProfile.setPreference("extensions.firebug.currentVersion","1.9.1");
 WebDriver driver = new FirefoxDriver(firefoxProfile);
 // 访问google
 driver.get("http://www.google.com");
 // 另一种访问方法
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 // 提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 // 关闭浏览器
 driver.quit();
 }
}

***Internet Explorer Driver****：*该驱动需要一个dll文件，故只能在windows系统下使用。所有Selenium版本的都有对xp下的IE6,7,8和windows7下的IE9支持。

*用法：*

WebDriver driver = new InternetExplorerDriver();

*例子：*

package demo;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class DemoUseIE {
 public static void main(String[] args) {
 // 创建一个WebDriver实例
 WebDriver driver = new InternetExplorerDriver();
 // 访问google
 driver.get("http://www.google.com");
 // 找到文本框
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 // 提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 // 关闭浏览器
 driver.quit();
 }
}

*优点：*

  运行于真正的浏览器，且支持各种用户终端的javascript。

*缺点：*

  IE Driver只能工作在windows下

  运行速度相对较慢

  大部分版本下XPach没有原生支持，由于Sizzle是自动注入，所以运行速度低于其他浏览器，CSS渲染也比较慢；

  CSS不是原生支持IE6和IE7的，Sizzle是强行注入的；

  IE8,9是原生支持CSS的，但是他们不完全支持CSS3；

*需要的配置项：*

  在ie7还有win7和vista系统下，我们必须将浏览器设置成安全模式

  如果要使用鼠标事件，必须确定浏览器是100%缩放的

***Chrome Driver：***是一个独立的服务器，是由Chromium项目组协助开发的。ChromeDriver由三个部件组成：chrome浏览器本身，支持ChromeDriver的语言和支持ChromeDriver和Chrome之间通信的工程。使用ChromeDriver必须启动一个后台程序（即：支持ChromeDriver的语言和支持ChromeDriver和Chrome之间通信的工程）。

用法：

WebDriver driver = new ChromeDriver();

例子：

package demo;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class DemoUseChrome {
 public static void main(String[] args) {
 // 配置服务器
 System.setProperty("webdriver.chrome.driver",
 ".\\res\\chromedriver.exe");
 // 创建一个WebDriver实例
 WebDriver driver = new ChromeDriver();
 // 访问google
 driver.get("http://www.google.com");
 // 找到文本框
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 // 提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 // 关闭浏览器
 driver.quit();
 }
}

*优点：*

  运行在真正的浏览器上，且支持javascript；

  由于Chrome是Webkit内核的浏览器，所以Chrome Driver可以一定程度上测试Safari。但是需要注意的是Chrome使用自己的V8 javascript引擎，而Safari是Nitro引擎，在javascript的执行上还是有区别的。

*缺点：*

  运行速度低于HtmlUnit Driver

*Chrome高级设置：*

我们可以使用一些命令行指令来配置浏览器，类似Firefox。如最大化浏览器：

 DesiredCapabilities capabilities = DesiredCapabilities.chrome();

  capabilities.setCapability("chrome.switches", Arrays.asList("--start-maximized"));

  WebDriver driver = new ChromeDriver(capabilities);

 例子：

package demo;
import java.util.Arrays;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
public class DemoUseChrome {
 public static void main(String[] args) {
 // 配置服务器
 System.setProperty("webdriver.chrome.driver",
 ".\\res\\chromedriver.exe");
 // 创建一个WebDriver实例
 DesiredCapabilities capabilities = DesiredCapabilities.chrome();
 capabilities.setCapability("chrome.switches",
 Arrays.asList("--start-maximized"));
 WebDriver driver = new ChromeDriver(capabilities);
 // 访问google
 driver.get("http://www.google.com");
 // 另一种访问方法
 // driver.navigate().to("http://www.google.com");
 // 找到文本框
 WebElement element = driver.findElement(By.name("q"));
 // 输入搜索关键字
 element.sendKeys("Selenium");
 // 提交表单 WebDriver会自动从表单中查找提交按钮并提交
 element.submit();
 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());
 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });
 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());
 // 关闭浏览器
 driver.quit();
 }
}

 ***WebDriver-Backed Selenium-RC：***WebDriver提供了Selenium-RC的实现，这表示你可以通过Selenium-RC API来使用WebDriver的基础功能。这主要是为了向后兼容。他允许通过适当的转换，那些已经用Selenium-RC API编写的用例能在WebDriver下使用。

用法：

Selenium selenium = new WebDriverBackedSelenium(driver, baseUrl);

例子：

package demo;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebDriverBackedSelenium;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
import com.thoughtworks.selenium.Selenium;

public class DemoUseBacked {
 public static void main(String[] args) {
 //创建一个WebDriver实例
 WebDriver driver = new FirefoxDriver();
 String baseUrl = "http://www.youdao.com";
 //启动一个Selenium
 Selenium selenium = new WebDriverBackedSelenium(driver, baseUrl);
 // 访问google
 selenium.open(baseUrl);
 // 输入搜索关键字
 selenium.type("id=query", "selenium");
 //点击 搜索
 selenium.click("id=qb");

 // 检查页面title
 System.out.println("Page title is: " + driver.getTitle());

 // google查询结果是通过javascript动态呈现的.
 // 设置页面等待10秒超时
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("selenium");
 }
 });

 // 显示查询结果title
 System.out.println("Page title is: " + driver.getTitle());

 //停止Selenium
 selenium.stop();
 }
}

*优点：*

  允许WebDriver和Selenium API直接兼容

  提供selenium RC向WebDriver迁移

  不需要对立的运行Selenium RC server

*缺点：*

   不是所有方法都可以使用

  由于底层实现的差别，切换后有些方法会运行变慢

**(三)   命令和操作**

这一部分将介绍一下WebDriver的一些具体操作和命令，实际操作中，我们需要两大工具来帮助我们：FireBug和Xpath工具，这两者都是Firefox上的插件。接下来我们所讲解的都是以FirefoxDriver为基础的，且基于WebDriver driver = new FirefoxDriver();创建的一个driver实例：

a)         **访问一个页面**

第一件你想使用WebDriver做的事情肯定是访问一个页面，最基础的方法是调用“get”方法：

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

同样我们可以使用：

driver.navigate().to("http://www.google.com");

WebDriver会自动等待到该页面完全加载才执行接下来的测试和脚本的执行。但是如果你的页面存在很多的AJAX加载，此时WebDriver是无法知道是否完成加载。检查此类页面是否加载完成，那么我们就需要Explicit和Implicit Wait（这两个将在后面文章解释）。

b)       **定位UI元素**

WebDriver可以通过WebDriver实例来定位元素，任何语言库都含有“Find Element”和“Find Elements”的方法。第一个方法返回一个WebElement或者抛出异常。后者返回所有WebElement的列表，或者空列表。

获取和定位元素我们调用“By”方法。下面具体解释下“By”方法：

*By ID*

这是一个极为有效定位元素的方法。普遍的现状是UI工程师在实际编写页面时很少写id或者自动生产一个ID，这些都是需要避免的。对于一个页面Element来说，class比自动生产的id更好。

通过id定位元素的例子：

<div id="coolestWidgetEvah">...</div>

WebElement element = driver.findElement(By.id("coolestWidgetEvah"));

 *By Class Name*

        这里的class指的是DOM中的元素，在实际使用过程中，我们也会发现很多DOM元素含有相同的class名。

通过class name定位元素例子：

<div class="cheese">

<span>Cheddar</span>

</div>

<div class="cheese">

<span>Gouda</span>

</div>

List<WebElement> cheeses = driver.findElements(By.className("cheese"));

*By Tag Name*

DOM的Tag元素

用Tag name 定位元素的例子：

<iframe src="..."></iframe>

WebElement frame = driver.findElement(By.tagName("iframe"));

*By Name*

例子：

<input name="cheese" type="text"/>

WebElement cheese = driver.findElement(By.name("cheese"));

*By Link Text*

例子：

<a href="http://www.google.com/search?q=cheese">cheese</a>>

WebElement cheese = driver.findElement(By.linkText("cheese"));

*By Partial Link Text*

根据链接的部分文字

例子：

<a href="http://www.google.com/search?q=cheese">search for cheese</a>>

WebElement cheese = driver.findElement(By.partialLinkText("cheese"));

*By CSS*

从名字上看，这是根据CSS来定位元素。

例子：

<div id="food">

         <span class="dairy">milk</span>

         <span class="dairy aged">cheese</span>

</div>

WebElement cheese = driver.findElement(By.cssSelector("#food span.dairy aged"));

*By XPATH*

在高级的水平下，WebDriver尽可能使用浏览器的原生的XPath能力。在那些没有原生的XPath支持的浏览器，我们提供自己的实现方式。但是三个Driver有一定的区别。

 例子：

 <input type="text" name="example" />

 <INPUT type="text" name="other" />

 List<WebElement> inputs = driver.findElements(By.xpath("//input"));

查找结果：

HTML元素有时并不需明确声明，因为他们将默认为已知值的属性。例如，input标签，就不需要设置type为text，默认属性就是text，经验原则：WebDriver在使用中的XPath时，不应该期望能够对这些隐含属性相匹配。



*使用javascript*

您可以执行任意JavaScript找到一个元素，只要你返回一个DOM元素，它会自动转换到一个WebElement对象。

例子：

jQuery的页面加载一个简单的例子：

WebElement element = (WebElement) ((JavascriptExecutor)driver).executeScript("return $('.cheese')[0]");

寻求所有的页面上的input元素：

List<WebElement> labels = driver.findElements(By.tagName("label"));

List<WebElement> inputs = (List<WebElement>) ((JavascriptExecutor)driver).executeScript(

     "var labels = arguments[0], inputs = []; for (var i=0; i < labels.length; i++){" +

"inputs.push(document.getElementById(labels[i].getAttribute('for'))); } return inputs;", labels);

*用户表单填充*

例子：

遍历select标签

WebElement select = driver.findElement(By.tagName("select"));

List<WebElement> allOptions = select.findElements(By.tagName("option"));

for (WebElement option : allOptions) {

             System.out.println(String.format("Value is: %s", option.getAttribute("value")));

             option.click();

}

*选择某一个选项：*

Select select = new Select(driver.findElement(By.tagName("select")));

select.deselectAll();

select.selectByVisibleText("Edam");

*上传文件：*

WebElement FileUpload =driver.findElement(By.id("upload"));

String filePath = "C:\test\\uploadfile\\media\_ads\\test.jpg";

FileUpload.sendKeys(filePath);

*提交：*

Submit在form中

driver.findElement(By.id("submit")).click();

submit不在form中

WebElement.submit();

*拖拽操作：*

WebElement element = driver.findElement(By.name("source"));

WebElement target = driver.findElement(By.name("target"));

(new Actions(driver)).dragAndDrop(element, target).perform();

*Windows和Frames之间的切换*

一些web应用程序有许多Frames或多个Windows。 WebDriver支持使用“switchTo”的方法实现的窗口之间切换。

driver.switchTo().window("windowName");

所有对driver的调用都会指向特定的窗口，但是我们怎么知道窗口的名字呢？我们可以查看javascript代码和打开他的链接：

<a href="somewhere.html" target="windowName">Click here to open a new window</a>

另外，还可以通过“window handle”去调用“switchTo().window()”，通过这个，我们就遍历来找到所有打开的窗口：

for (String handle : driver.getWindowHandles()) {

 driver.switchTo().window(handle);

 }

*Switch同样支持frame：*

driver.switchTo().frame("frameName");

同样可以使用他访问subframe，找frameName的第一个subframe中叫做child的frame：

driver.switchTo().frame("frameName.0.child");

*弹出框：*

从selenium2.0开始，已经支持对弹出框的获取

Alert alert = driver.switchTo().alert();

这个方法会返回当前被打开打警告框，你可以进行统一，取消，读取提示内容，后则进入到提示，这个同样使用alerts，confirms，prompts。

 *Navigation：History and Location*

之前我们就可以通过get方法来打开一个网页，像我们所看到的，WebDriver同样还有许多小接口，Navigation就是其中一个小接口：

driver.navigate().to("http://www.example.com");

navigate().to和get()其实作用是一样的，但是navigate还可以进行浏览器的前进后退操作：

driver.navigate().forward();

driver.navigate().back();