## 使用firefoxdriver

1.导入selenium库赫尔firefoxdriver库

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

2.实例一个firefoxdriver对象

WebDriver driver = new FirefoxDriver();

3.操作firefoxdriver对象

driver.get(url)打开网页

driver.getCurrentUrl()获取当前网页url

### FirefoxProfile

FirefoxProfile用于定制待测试的Firefox浏览器的特定属性，包括存储的密码、书签、历史、Cookies等。

import org.openqa.selenium.firefox.FirefoxProfile;

FirefoxProfile profile = new FirefoxProfile();

try {

//使用addExtension(对象)方法加载插件

profile.addExtension(new File("files/firebug-2.0.7-fx.xpi"););

} catch (IOException e) {

e.printStackTrace();

}

//激活插件

profile.setPreference("extensions.firebug.currentVersion", "2.0.7");

profile.setPreference("extensions.firebug.allPagesActivation", "on");

//设置代理参数

profile.setPreference("network.proxy.type", 1);

profile.setPreference("network.proxy.http", proxyIp);//proxyIp代理主机ip

profile.setPreference("network.proxy.http\_port", proxyPort);// proxyPort代理端口

//设置默认下载路径

profile.setPreference("browser.download.folderList", 2);

profile.setPreference("browser.download.dir", "D:\\");

WebDriver driver = new FirefoxDriver(profile);设置好profile后实例化WebDriver时传递profile对象可以加载设置的profile

### 加载本机的Profile

import org.openqa.selenium.firefox.internal.ProfilesIni;

ProfilesIni pi = new ProfilesIni();

FirefoxProfile profile = pi.getProfile("default");

WebDriver driver = new FirefoxDriver(profile);

### 加载导出的profile文件

File file = new File("files\\lg6mie1i.default");

FirefoxProfile profile = new FirefoxProfile(file);

FirefoxProfile profile = new FirefoxProfile(file);

## 使用Chrome

import org.openqa.selenium.chrome.\*;

System.setProperty("webdriver.chrome.driver","D:\\selenium\\chromedriver.exe");

//System.setProperty("webdriver.ie.driver","D:\\selenium\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

## WebDriver与浏览器

### WebDriver的元素状态检测

WebElement element = driver.findElement(By.xpath("//form/input[@value=\"female\"]"));

element.isSelected()//检测元素是否被选中

element.isEnabled()//检测元素是否启用

element.isDisplayed()//检测元素是否可见

### 选择元素

By.id()//通过id属性查找元素

By.name()//通过name属性查找元素

//通过xpath查找元素

WebElement element = driver.findElement(By.xpath("//form/input[@value=\"female\"]"));

//通过CSS方式定位元素，查找条件和xpath一致

WebElement element = driver.findElement(By.cssSelector("input[value='female']"));

### 操作弹出窗口

//获取driver打开窗口的句柄

Set<String> allWindowsld = driver.getWindowHandles();//driver打开了多个窗口。

String MasterHandle = driver.getWindowHandle();

driver.switchTo().window(MasterHandle);//通过句柄切换窗口

driver.switchTo().window(windowld).getTitle().contains("登录")//切换窗口并获取窗口的标题，检查是否包含”登录”

driver.switchTo().window(wind).getPageSource().contains("四川代表团参加审议");//检查页面内容是否包含”四川代表团参加会议”

driver.manage().window().maximize();//最大化窗口

driver.manage().window().fullscreen();//全屏窗口

### 操作JavaScript弹框

import org.openqa.selenium.Alert;

Alert a = driver.switchTo().alert();//获取弹窗对象

操作Alert对象

//alert() 方法用于显示带有一条指定消息和一个 OK 按钮的警告框。

a.getText();//获取弹窗内容

a.accept();//点击弹窗的确定按钮

Confirm

//confirm() 方法用于显示一个带有指定消息和 OK 及取消按钮的对话框。

a.dismiss();//点击取消按钮

a.accept();//点击弹窗的确定按钮

Prompt

//prompt() 方法用于显示可提示用户进行输入的对话框。

a.sendKeys("");//向弹框中输入

操作cookie

获取cookie

//通过创建文件对象，写文件对象，写文件流的方式把获取cookies写入文件中

//保持登录后，通过driver.manage().getCookies()获取cookies对象

File cookiefile = new File("D:\\cookie.txt");

cookiefile.delete();

cookiefile.createNewFile();

FileWriter filewriter = new FileWriter(cookiefile);

BufferedWriter bufferedWriter = new BufferedWriter(filewriter);

for(Cookie cookies : driver.manage().getCookies()){

bufferedWriter.write(cookies.getName() + ";" + cookies.getValue() + ";" + cookies.getDomain() + ";"

+ cookies.getPath() + ";" + cookies.getExpiry() + ";" + cookies.isSecure() + ";");

bufferedWriter.newLine();

}

bufferedWriter.flush();

bufferedWriter.close();

filewriter.close();

使用cookie

浏览器在request添加cookie要跟当前请求的URL是一个域的 ，webdriver 实例化浏览器的时候默认给浏览器的URL是：data:. 所以你添加的cookie的域和当前URL不是一个域名下的，会报InvalidCookieDomain！

解决办法 ：

先get你的目标网站再添加cookie保持域一致：

1.webdriver.get(http://www.xxx.com);

2.webdriver.manage().addCookie(cookie)

3.webdriver.get(http://www.xxx.com); //重新请求目标网站

//使用读取文件流的方式读取多个cookies，再使用driver.manage().addCookie(cookie)向浏览器中添加cookie，最后webdriver.get(http://www.xxx.com); //重新请求目标网站

public static void main(String[] args) throws FileNotFoundException, InterruptedException{

WebDriver driver = new FirefoxDriver();

driver.get("http://182.92.218.212:8888/zentaopms/www/index.php?m=user&f=login");

try {

File file = new File("D:\\cookie.txt");

FileReader filereader = new FileReader(file);

BufferedReader bufferedreader = new BufferedReader(filereader);

String line;

Thread.sleep(5000);

while((line = bufferedreader.readLine()) != null){

StringTokenizer stringtokenizer = new StringTokenizer(line,";");

while(stringtokenizer.hasMoreTokens()){

String name = stringtokenizer.nextToken();

System.out.println(name);

String value = stringtokenizer.nextToken();

System.out.println(value);

String domain = stringtokenizer.nextToken();

System.out.println(domain);

String path = stringtokenizer.nextToken();

System.out.println(path);

Date expiry = null;

String dt;

if(!(dt= stringtokenizer.nextToken()).equals("null")){

expiry = new Date(dt);

}

System.out.println(expiry);

boolean isSecure = new Boolean(stringtokenizer.nextToken()).booleanValue();

System.out.println(isSecure);

Cookie cookie = new Cookie(name,value,domain,path,expiry,isSecure);

driver.manage().addCookie(cookie);

}

}

} catch (IOException e) {

e.printStackTrace();

}

driver.get("http://182.92.218.212:8888/");

driver.quit();

}

### 操作浏览器前进后退

driver.navigate().to("http://59.108.56.172/redmine");//跳转到

driver.navigate().back();//后退

driver.navigate().refresh();//刷新

driver.navigate().forward();//前进

### 操作页面标题

driver .getTitle()//获取当前页面标题

driver. getCurrentUrl()//获取当前页面url

driver .getTitle().toLowerCase()//把字母变成小写

driver .getTitle().toUpperCase()//把字母变成大写

d.getTitle().toLowerCase().startsWith(a,start);检查标题是否包含字符串a，可选参数start，可以指定从start开始，start为int类型。

### 通过WebDriver执行JavaScript

执行js一般有两种场景：

1. 一种是在页面上直接执行JS
2. 另一种是在某个已经定位的元素上执行JS

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebElement;

WebDriver driver = new FirefoxDriver();

JavascriptExecutor jsExecutor = (JavascriptExecutor)driver;

//把webdriver对象转换为JavascriptExecutor对象才能调用executeScript方法

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

jsExecutor.executeScript("JavascriptExecutor",element);//定位的元素上执行JS

jsExecutor.executeScript("JavascriptExecutor");//直接执行js

### 等待方式

最近在项目过程中使用selenium 判断元素是否存在的时候 遇到一个很坑爹的问题， 用以下方法执行的时候每次都会等待很长一段时间，原因是因为对selenium实现方法了解不足导致一直找不到解决方法。

private boolean isElementPresent(By by) {

try {

driver.findElement(by);

return true;

} catch (NoSuchElementException e) {

return false;

}

}

不明所以， 经过追踪源码才发现启动Driver的时候 使用driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS); 此处设置的等待时间 是针对全局设置的，webdriver中执行所有命令 的超时时间都设置为30秒了， 如上面的findElement方法，找不到元素会默认等待三十秒。 有些时候只需简单判断元素是否存在，立马执行，而此处的设置导致 脚本执行缓慢。 将Selenium中的等待时间贴出来给大家张张经验。

Selenium的延迟等待分为 显式等待(Explicit Wait) & 隐式等待(Implicit Wait).

1.显式等待

显式等待,就是明确的要等到某个元素的出现或者是某个元素的可点击等条件,等不到,就一直等,除非在规定的时间之内都没找到,那么就跳出Exception.

如:

1 new WebDriverWait(driver, 15 ).until(

2 ExpectedConditions.presenceOfElementLocated(By.cssSelector( "css locator" ))

3 );

这里,15是要等待的秒数.如果没有满足until()方法中的条件,就会始终在这里wait 15秒,依然找不到,就抛出异常.

也可以这样写:

01 WebDriver driver = new FirefoxDriver();

02 driver.get( www.baidu.com );

03 WebElement e = ( new WebDriverWait( driver, 10 )) .until(

04 new ExpectedCondition< WebElement>(){

05 @Override

06 public WebElement apply( WebDriver d) {

07 return d.findElement( By.id( "id locator" ));

08 }

09 }

10 );

这样就通过回调函数,直接获得了这个WebElement.也就是页面元素.

如果只是仅仅想判断页面是不是加载到某个地方了,就可以用第一种方法; 但如果需要得到某个WebElement,两种方式都可以,只是第一种方式还需要再多一步获取的操作.

1 Wait w = new Wait(){

2 @Override

3 public boolean until() {

4 return webElement.isDisplayed();

5 }

6 };

另外这种等待的方式,在加载js代码的时候做判断会比较方便,反正我只在代码里这么见过,但是没用过,有兴趣的朋友可以研究一下.

2.隐式等待

1 driver.manage().timeouts().implicitlyWait(second, TimeUnit.SECONDS);

隐式等待, 此处的隐式等待是针对Driver 每次执行命令的 最长执行时间也可以理解为超时时间， 一些人对此处有误解，认为是让Driver等一段时间， 确实某些时候能让Driver等一段时间， 但是影响是全局的，每次Driver执行 找不到元素都会等待此处设置的时间， 假设某处将此值设置的太长，必须在执行完成之后还原回来，否则判断一个元素是否存在的时候，就会遇到很坑爹的问题。 改进后的方法如下：

WebDriver会进行一个隐式等待,但参数只有时间,这就导致我需要什么元素出现,我不一定能等到它,

private boolean isElementPresent(By by) {

try {

driver.manage().timeouts().implicitlyWait(1, TimeUnit.SECONDS);

driver.findElement(by);

return true;

} catch (NoSuchElementException e) {

return false;

}

}

得不到某个元素,我们就延迟一下...

3.线程休眠

1. Thread.sleep()

等待抛出的异常The element reference is stale. Either the element is no longer attached to

出现场景，当使用显示等待，查找元素发送字符串，submit，再使用隐式等待，查找元素发送字符串，submit，运行时就提示这个，搜索结果应该是dom认为元素在隐式等待中使用一次后变得不够新，不能做两次操作。

## 文件操作

### 截图

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

//指定了OutputType.FILE做为参数传递给getScreenshotAs()方法，其含义是将截取的屏幕以文件形式返回。

File srcFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

//利用FileUtils工具类的copyFile()方法保存getScreenshotAs()返回的文件对象。

FileUtils.copyFile(srcFile, new File("screenshot.png"));

### 复制

import org.openqa.selenium.io.FileHandler;

FileHandler filehandler = new FileHandler();

filehandler.copy(source, dest, suffix);//source源文件对象，dest目的文件对象，suffix筛选条件以suffix为后缀名的文件。

### 创建目录

filehandler.createDir(new File("D:\\Baidu\\BaiduReaderLite\\ClientStat\\test1"));

### 删除目录

filehandler.delete(new File("D:\\Baidu\\BaiduReaderLite\\ClientStat\\test1"));

### 读取文件

String f = filehandler.readAsString(new File("D:\\Baidu\\BaiduReaderLite\\ClientStat\\test.txt"));

### 压缩文件

import org.openqa.selenium.io.Zip;

Zip zip = new Zip();

zip.zip(inputDir, output);//讲input文件对象压缩为output文件对象

filehandler.isZipped("d:1.txt")//判断是否是压缩包

zip.unzip(inputDir, output);//讲input文件对象解压缩为output文件对象

### 操作文件权限

filehandler.makeExecutable(file)//修改file对象为可执行

filehandler.makeWritable(file)// 修改file对象为可写

filehandler.canExecute(file);//判断是否为可执行

## WebDriver与HTML5

### Video

Video标签可以通过js方法来获取起源地址、播放、暂停视频

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebElement;

WebDriver driver = new FirefoxDriver();

driver.get(“http://videojs.com/”);

WebElement video = driver.findElement(By.id(“preview-player\_html5\_api”));

JavascriptExecutor jsExecutor = (JavascriptExecutor)driver;

//把webdriver对象转换为JavascriptExecutor对象才能调用executeScript方法

String source = (String)jsExecutor.executeScript("return arguments[0].currentSrc",video);//定位的元素上执行JS,返回播放源

jsExecutor.executeScript("return arguments[0].play()",video);//通过js代码播放视频

jsExecutor.executeScript("return arguments[0].pause()",video); //通过js代码暂停视频

canvas

该标签提供给了通过javascript绘制图形的方法，他再HTML页面上以一块以<canvas>为标签的矩形区域

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.interactions.Actions

WebDriver driver = new ChromeDriver()

driver.get(url)

WebElement canvas = driver.findElement(By.id(“”))//定位到canvas

Actions drawing = new Actions(driver);

drawing.clickAndHold(canvas).moveByOffset(10,50). moveByOffset(50,10). moveByOffset(-10,-50). moveByOffset(-50,-10).release().perform();//通过Actions在canvas上绘制一个图形

## remote Server

### Selenium与jetty

//代码：以嵌入模式启动Jetty

import org.mortbay.http.HttpContext;

import org.mortbay.http.HttpServer;

import org.mortbay.http.SocketListener;

import org.mortbay.http.handler.ResourceHandler;

public class JettySample{

public static void main(String[]args)throws Exception{

//创建JettyHttpServer对象

HttpServer server=new HttpServer();

//在端口8080上给HttpServer对象绑上一个listener，使之能够接收HTTP请求

SocketListener listener=new SocketListener();

listener.setPort(8080);

server.addListener(listener);

//创建一个HttpContext，处理HTTP请求。

HttpContext context=new HttpContext();

//用setContextPath把Context映射到（/web）URL上。

context.setContextPath("/web");

//setResourceBase方法设置文档目录以提供资源

context.setResourceBase("C:\\j2sdk1.4.1\_05");

//添加资源处理器到HttpContext，使之能够提供文件系统中的文件

context.addHandler(new ResourceHandler());

server.addContext(context);

//启动服务器

server.start();

}

}

### Remote server截图

在remoteWebdriver中没有实现takesScreenshot类，所以需要使用浏览器本身的截图功能，可以通过takesScreenshot接口的Augmenter类来完成remote截图

import org.apache.commons.io.FileUtils;

import org.junit.\*;

import java.io.File;

import java.io.IOException;

import java.net.URL;

import java.net.MalformedURLException;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.remote.Augmenter;//远程增量类,使用截图时需要引入

import org.openqa.selenium.remote.DesiredCapabilities;//远程控制类

import org.openqa.selenium.remote.RemoteWebDriver;//远程wdriver类

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

WebDriver augmentdriver = new Augmenter().augment(driver);//实例化WebDriver类指向Augmenter().augment(driver)，在截图中使用

File Screenshot = ((TakesScreenshot)augmentdriver).getScreenshotAs(OutputType.FILE);//调用getScreenshotAs截图赋值给File Screenshot

try {

FileUtils.copyFile(Screenshot, new File("D:\\13.png"));//调用FileUtils类把截图复制到本地

} catch (IOException e) {

// TODO Auto-generated catch block

e. .printStackTrace();

}

WebDriver监听  
1.我可以用log记录我的driver的所有事件。注意，我只要对每种事件写一行代码，一共撑死了10来行代码。以后就可以自动监听，自动执行这些代码，不用再写一大堆log.info，也不用面对一个没log的测试用例。

2.我可以对事件截图。不仅能在出错时截图，我还可以回溯到错误前一个步骤时截图。甚至对每个步骤截图。而不必在测试用例里写一大堆screenshot的调用。

3. 我可以隐式等待。对五六个主要事件进行隐式等待。五六行的代码量换取你在测试用例里一次一次的调用等待。你也不需要再因为为了加个等待而对selenium提供的每个方法都做二次封装。（开发转的自动化测试人员特别爱干这事。）

#### 监听类

WebDriver存在两个监听类，一个是EventFiringWebDriver，一个是WebDriverEvenListener。所有的WebDriverEvenListener监听都需要注册到EventFiringWebDriver上。

#### 自定义事件监听

有两种方法，一种是实现WebDriverEvenListener在实现中添加自定义逻辑，由于是接口所以需要用户将所有接口方法全部实现一遍。另一种是通过继承AbstractWebDriverEventListener抽象类来自定义事件侦听类并添加自定义逻辑，只需要对有必要添加自定义逻辑的方法编写覆盖的逻辑代码即可。

继承AbstractWebDriverEventListener抽象类来自定义事件侦听，添加自定义逻辑

package button;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.events.AbstractWebDriverEventListener;

public class WebDriverListen extends AbstractWebDriverEventListener{

@Override//重写方法时使用这个参数，编译器会去校验重写方法的正确性，不正确会抛异常，不加这个参数编译器会以为是新写了一个方法

public void afterNavigateTo(String url,WebDriver driver){

System.out.println("After Navigate To" + url);

}

@Override

//重写了浏览器后退的侦听事件

public void afterNavigateBack(WebDriver driver){

System.out.println("After Navigate Back To" + driver.getCurrentUrl());

}

@Override

//重写了click事件的侦听日志

public void afterClickOn(WebElement webElement,WebDriver driver){

System.out.println("After Click on" + webElement.getText());

}

}

使用自定义侦听

import button.WebDriverListen;

public class UseListen {

public static void main(String[] args){

WebDriver driver = new FirefoxDriver();

EventFiringWebDriver eventFiriingDriver = new EventFiringWebDriver(driver);

WebDriverListen mylisten = new WebDriverListen();

eventFiriingDriver.register(mylisten);//

eventFiriingDriver.get("https://www.baidu.com");

eventFiriingDriver.navigate().to("http://news.baidu.com/");

eventFiriingDriver.navigate().back();

eventFiriingDriver.findElement(By.id("su")).click();

driver.quit();

}

}