1.1 下载 selenium2.0 的 lib 包

http://code.google.com/p/selenium/downloads/list

官方 UserGuide：http://seleniumhq.org/docs/

1.2 用 webdriver 打开一个浏览器

我们常用的浏览器有 firefox 和 IE 两种，firefox 是 selenium 支持得比较成熟的浏览器。但是做页面的测试，速度通常很慢，严重影响持续集成的速度，这个时候建议使用 HtmlUnit，不过 HtmlUnitDirver 运行时是看不到界面的，对调试就不方便了。使用哪种浏览器，可以做成配置项，根据需要灵活配置。

1. 打开 firefox 浏览器：

```
//Create a newinstance of the Firefox driver
WebDriver driver = newFirefoxDriver();
```

1. 打开 IE 浏览器

```
//Create a newinstance of the Internet Explorer driver
WebDriver driver = newInternetExplorerDriver ();
```

1. 打开 HtmlUnit 浏览器

```
//Createa new instance of the Internet Explorer driver
WebDriverdriver = new HtmlUnitDriver();
```

1.3 打开测试页面

对页面对测试，首先要打开被测试页面的地址（如：http://www.google.com）,web driver 提供的 get 方法可以打开一个页面：

```
// And now use thedriver to visit Google
driver.get("http://www.google.com");
```

1.4 GettingStarted

```
package org.openqa.selenium.example;

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 voidmain(String[] args) {
        // Create a newinstance of the Firefox driver
        // Notice that theremainder of the code relies on the interface,
        // not the implementation.
        WebDriver driver = newFirefoxDriver();

        // And now use this tovisit Google
       driver.get("http://www.google.com");
        // Alternatively thesame thing can be done like this
        // driver.navigate().to("http://www.google.com");

        // Find the text inputelement by its name
        WebElement element =driver.findElement(By.name("q"));

        // Enter something tosearch for
       element.sendKeys("Cheese!");

        // Now submit the form.WebDriver will find the form for us from the element
        element.submit();

        // Check the title ofthe page
       System.out.println("Page title is: " + driver.getTitle());

        // Google's search isrendered dynamically with JavaScript.
        // Wait for the pageto load, timeout after 10 seconds
        (newWebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
         public Booleanapply(WebDriver d) {
                returnd.getTitle().toLowerCase().startsWith("cheese!");
            }
        });

        // Should see:"cheese! - Google Search"
       System.out.println("Page title is: " + driver.getTitle());

        //Close the browser
        driver.quit();
    }
}
```

第 2 章　　Webdirver 对浏览器的支持

2.1 HtmlUnit Driver

优点：HtmlUnit Driver 不会实际打开浏览器，运行速度很快。对于用 FireFox 等浏览器来做测试的自动化测试用例，运行速度通常很慢，HtmlUnit Driver 无疑是可以很好地解决这个问题。
缺点：它对 JavaScript 的支持不够好，当页面上有复杂 JavaScript 时，经常会捕获不到页面元素。
使用：

```
WebDriver driver = new HtmlUnitDriver();
```

2.2 FireFox Driver

优点：FireFox Dirver 对页面的自动化测试支持得比较好，很直观地模拟页面的操作，对 JavaScript 的支持也非常完善，基本上页面上做的所有操作 FireFox Driver 都可以模拟。
缺点：启动很慢，运行也比较慢，不过，启动之后 Webdriver 的操作速度虽然不快但还是可以接受的，建议不要频繁启停 FireFox Driver。
使用：

```
WebDriver driver = new FirefoxDriver();
```

Firefox profile 的属性值是可以改变的, 比如我们平时使用得非常频繁的改变 useragent 的功能, 可以这样修改：

```
FirefoxProfile profile = new FirefoxProfile();
profile.setPreference("general.useragent.override", "some UAstring");
WebDriver driver = new FirefoxDriver(profile);
```

2.3 InternetExplorer Driver

优点：直观地模拟用户的实际操作，对 JavaScript 提供完善的支持。
缺点：是所有浏览器中运行速度最慢的，并且只能在 Windows 下运行，对 CSS 以及 XPATH 的支持也不够好。
使用：

```
WebDriver driver = new InternetExplorerDriver();
```

第 3 章 使用操作

3.1 如何找到页面元素

Webdriver 的 findElement 方法可以用来找到页面的某个元素，最常用的方法是用 id 和 name 查找。下面介绍几种比较常用的方法。

3.1.1 By ID

假设页面写成这样：

```
<input type="text" name="passwd"id="passwd-id" />
```

那么可以这样找到页面的元素：
通过 id 查找：

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

3.1.2 By Name

或通过 name 查找：

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

3.1.3 By XPATH

或通过 xpath 查找：

WebElement element =driver.findElement(By.xpath("//input[@id='passwd-id']"));

3.1.4 By Class Name

假设页面写成这样：

Cheddar <divclass="cheese">Gouda

可以通过这样查找页面元素：

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

3.1.5 By Link Text

假设页面元素写成这样：

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

那么可以通过这样查找：

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

3.2 如何对页面元素进行操作

找到页面元素后，怎样对页面进行操作呢？我们可以根据不同的类型的元素来进行一一说明。

3.2.1 输入框（text field or textarea）

找到输入框元素：

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

在输入框中输入内容：

element.sendKeys(“test”);

将输入框清空：

element.clear();

获取输入框的文本内容：

element.getText();

3.2.2 下拉选择框(Select)

找到下拉选择框的元素：

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

选择对应的选择项：

select.selectByVisibleText(“mediaAgencyA”);
或
select.selectByValue(“MA_ID_001”);

不选择对应的选择项:
select.deselectAll();
select.deselectByValue(“MA_ID_001”);
select.deselectByVisibleText(“mediaAgencyA”);
或者获取选择项的值:
select.getAllSelectedOptions();
select.getFirstSelectedOption();

3.2.3 单选项(Radio Button)

找到单选框元素:
WebElement bookMode =driver.findElement(By.id("BookMode"));
选择某个单选项:
bookMode.click();
清空某个单选项:
bookMode.clear();
判断某个单选项是否已经被选择:
bookMode.isSelected();

3.2.4 多选项(checkbox)

多选项的操作和单选的差不多:
WebElement checkbox =driver.findElement(By.id("myCheckbox."));
checkbox.click();
checkbox.clear();
checkbox.isSelected();
checkbox.isEnabled();

3.2.5 按钮(button)

找到按钮元素:
WebElement saveButton = driver.findElement(By.id("save"));
点击按钮:
saveButton.click();
判断按钮是否 enable:

saveButton.isEnabled ();

3.2.6 左右选择框

也就是左边是可供选择项，选择后移动到右边的框中，反之亦然。例如:
Select lang = new Select(driver.findElement(By.id("languages")));

lang.selectByVisibleText(“English”);

WebElement addLanguage =driver.findElement(By.id("addButton"));

addLanguage.click();

3.2.7 弹出对话框(Popup dialogs)

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

alert.accept();

alert.dismiss();

alert.getText();

3.2.8 表单(Form)

Form 中的元素的操作和其它的元素操作一样，对元素操作完成后对表单的提交可以:

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

approve.click();

或

approve.submit();//只适合于表单的提交

3.2.9 上传文件 (Upload File)

上传文件的元素操作:

WebElement adFileUpload = driver.findElement(By.id("WAP-upload"));

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

adFileUpload.sendKeys(filePath);

3.2.10 Windows 和 Frames 之间的切换

一般来说，登录后建议是先:

driver.switchTo().defaultContent();

切换到某个 frame:

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

从一个 frame 切换到另一个 frame:

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

切换到某个 window:

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

3.2.11 拖拉(Drag andDrop)

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

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

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

3.2.12 导航 (Navigationand History)

打开一个新的页面:

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

通过历史导航返回原页面：

```
driver.navigate().forward();
driver.navigate().back();
```

3.3 高级使用

3.3.1 改变 user agent

User Agent 的设置是平时使用得比较多的操作：

```
FirefoxProfile profile = new FirefoxProfile();
profile.addAdditionalPreference("general.useragent.override","some UA string");
WebDriver driver = new FirefoxDriver(profile);
```

3.3.2 读取 Cookies

我们经常要对的值进行读取和设置。

增加 cookie:

```
// Now set the cookie. This one's valid for the entire domain
Cookie cookie = new Cookie("key", "value");
driver.manage().addCookie(cookie);
```

获取 cookie 的值：

```
// And now output all the available cookies for the current URL
Set<Cookie> allCookies = driver.manage().getCookies();
for (Cookie loadedCookie : allCookies) {
   System.out.println(String.format("%s -> %s",loadedCookie.getName(), loadedCookie.getValue()));
}
```

根据某个 cookie 的 name 获取 cookie 的值：

```
driver.manage().getCookieNamed("mmsid");
```

删除 cookie:

```
// You can delete cookies in 3 ways
// By name
driver.manage().deleteCookieNamed("CookieName");
// By Cookie
driver.manage().deleteCookie(loadedCookie);
// Or all of them
driver.manage().deleteAllCookies();
```

3.3.3 调用 Java Script

Web driver 对 Java Script 的调用是通过 JavascriptExecutor 来实现的，例如：

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("(function(){inventoryGridMgr.setTableFieldValue('"+ inventoryId + "','" + fieldName +
"','"
                + value + "');})()");
```

3.3.4 Webdriver 截图

如果用 webdriver 截图是：

```
driver = webdriver.Firefox()
driver.save_screenshot("C:\error.jpg")
```

3.3.5 页面等待

因为 Load 页面需要一段时间，如果页面还没加载完就查找元素，必然是查找不到的。最好的方式，就是设置一个默认等待时间，在查找页面元素的时候如果找不到就等待一段时间再找，直到超时。

Webdriver 提供两种方法，一种是显性等待，另一种是隐性等待。

显性等待：

```
WebDriver driver =new FirefoxDriver();
driver.get("http://somedomain/url_that_delays_loading");
WebElementmyDynamicElement = (new WebDriverWait(driver, 10))
  .until(newExpectedCondition<WebElement>(){
  @Override
  public WebElementapply(WebDriver d) {
    returnd.findElement(By.id("myDynamicElement"));
  }});
```

隐性等待：

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement =driver.findElement(By.id("myDynamicElement"));
```

第 4 章　　RemoteWebDriver

当本机上没有浏览器，需要远程调用浏览器进行自动化测试时，需要用到 RemoteWebDirver.

4.1　使用 RemoteWebDriver

```
import java.io.File;
import java.net.URL;

import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.Augmenter;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class Testing {
```

```
    public void myTest()throws Exception {
        WebDriver driver = newRemoteWebDriver(
                        new URL("http://localhost:4446/wd/hub"),
                        DesiredCapabilities.firefox());

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

        // RemoteWebDriverdoes not implement the TakesScreenshot class
        // if the driver doeshave the Capabilities to take a screenshot
        // then Augmenter willadd the TakesScreenshot methods to the instance
        WebDriveraugmentedDriver = new Augmenter().augment(driver);
        File screenshot =((TakesScreenshot)augmentedDriver).
                        getScreenshotAs(OutputType.FILE);
    }
}
```

4.2 SeleniumServer

在使用 RemoteDriver 时，必须在远程服务器启动一个 SeleniumServer:

```
java -jar selenium-server-standalone-2.20.0.jar -port 4446
```

4.3 How to setFirefox profile using RemoteWebDriver

```
profile = new FirefoxProfile();
        profile.setPreference("general.useragent.override",testData.getUserAgent());
capabilities = DesiredCapabilities.firefox();
capabilities.setCapability("firefox_profile", profile);
driver = new RemoteWebDriver(new URL(“http://localhost:4446/wd/hub”),capabilities);
driverWait = new WebDriverWait(driver,TestConstant.WAIT_ELEMENT_TO_LOAD);
driver.get("http://www.google.com");
```

第 5 章 封装与重用

WebDriver 对页面的操作，需要找到一个 WebElement，然后再对其进行操作，比较繁琐：

```
 // Find the text inputelement by its name
WebElement element = driver.findElement(By.name("q"));

// Enter something to search for
element.sendKeys("Cheese!");
```

我们可以考虑对这些基本的操作进行一个封装，简化操作。比如，封装代码：

```
    protected void sendKeys(Byby, String value){
      driver.findElement(by).sendKeys(value);
    }
```

那么，在测试用例可以这样简化调用：

```
sendKeys(By.name("q"),”Cheese!”);
```

看，这就简洁多了。

类似的封装还有：

```
package com.drutt.mm.end2end.actions;


import java.util.List;
import java.util.NoSuchElementException;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.openqa.selenium.support.ui.WebDriverWait;

import com.drutt.mm.end2end.data.TestConstant;

public class WebDriverAction {

   //protected WebDriverdriver;
   protected RemoteWebDriverdriver;
   protected WebDriverWaitdriverWait;


    protected booleanisWebElementExist(By selector) {
        try {
            driver.findElement(selector);
            return true;
        } catch(NoSuchElementException e) {
            return false;
        }
    }

    protected StringgetWebText(By by) {
```

```
        try {
        return driver.findElement(by).getText();
        } catch (NoSuchElementException e) {
            return "Textnot existed!";
        }
    }

    protected voidclickElementContainingText(By by, String text){
        List<WebElement>elementList = driver.findElements(by);
        for(WebElement e:elementList){
            if(e.getText().contains(text)){
                e.click();
                break;
            }
        }
    }

    protected StringgetLinkUrlContainingText(By by, String text){
        List<WebElement>subscribeButton = driver.findElements(by);
        String url = null;
        for(WebElement e:subscribeButton){
            if(e.getText().contains(text)){
                url =e.getAttribute("href");
                break;
            }
        }
        return url;
    }

    protected void click(Byby){
      driver.findElement(by).click();
      driver.manage().timeouts().implicitlyWait(TestConstant.WAIT_ELEMENT_TO_LOAD,TimeUnit.SEC
ONDS);
    }

    protected StringgetLinkUrl(By by){
        return driver.findElement(by).getAttribute("href");
    }

    protected void sendKeys(Byby, String value){
```

```
        driver.findElement(by).sendKeys(value);
    }
```

第 6 章　　在 selenium2.0 中使用 selenium1.0 的 API

Selenium2.0 中使用 WeDriver API 对页面进行操作，它最大的优点是不需要安装一个 selenium server 就可以运行，但是对页面进行操作不如 selenium1.0 的 Selenium RC API 那么方便。Selenium2.0 提供了使用 Selenium RC API 的方法：

```
// You may use any WebDriver implementation. Firefox is used hereas an example
WebDriver driver = new FirefoxDriver();

// A "base url", used by selenium to resolve relativeURLs
 String baseUrl ="http://www.google.com";

// Create the Selenium implementation
Selenium selenium = new WebDriverBackedSelenium(driver, baseUrl);

// Perform actions with selenium
selenium.open("http://www.google.com");
selenium.type("name=q", "cheese");
selenium.click("name=btnG");

// Get the underlying WebDriver implementation back. This willrefer to the
// same WebDriver instance as the "driver" variableabove.
WebDriver driverInstance = ((WebDriverBackedSelenium)selenium).getUnderlyingWebDriver();

    //Finally, close thebrowser. Call stop on the WebDriverBackedSelenium instance
    //instead of callingdriver.quit(). Otherwise, the JVM will continue running after
    //the browser has beenclosed.
    selenium.stop();
```

我分别使用 WebDriver API 和 SeleniumRC API 写了一个 Login 的脚本，很明显，后者的操作更加简单明了。

WebDriver API 写的 Login 脚本：

```
    public void login() {
        driver.switchTo().defaultContent();
        driver.switchTo().frame("mainFrame");

        WebElement eUsername= waitFindElement(By.id("username"));
```

```
        eUsername.sendKeys(manager@ericsson.com);

        WebElement ePassword= waitFindElement(By.id("password"));
        ePassword.sendKeys(manager);

        WebElementeLoginButton = waitFindElement(By.id("loginButton"));
       eLoginButton.click();

    }
```

SeleniumRC API 写的 Login 脚本：

```
    public void login() {
        selenium.selectFrame("relative=top");
        selenium.selectFrame("mainFrame");
        selenium.type("username","manager@ericsson.com");
        selenium.type("password","manager");
        selenium.click("loginButton");
}
```