

WebDriver基础

欢迎阅读WebDriver基础讲义。本篇讲义将会重点介绍Selenium WebDriver的环境搭建和基本使用方法。

WebDriver环境搭建

Selenium WebDriver 又称为 Selenium2。

Selenium 1 + WebDriver = Selenium 2

WebDriver是主流Web应用自动化测试框架，具有清晰面向对象 API，能以最佳的方式与浏览器进行交互。

支持的浏览器：

- Mozilla Firefox
- Google Chrome
- Microsoft Internet Explorer
- Opera
- Safari
- Apple iPhone
- Android browsers

环境搭建步骤

在上一篇中，我们已经确认使用Python来进行WebDriver的编码和操作。事实上Java+Selenium WebDriver环境的搭建分为两个部分：

1. 安装Java
2. 引入Selenium

标准的安装步骤

1. 选择Java的版本并安装 Java 1.8
2. 使用IDEA新建Project
 - New Project | Maven Project
 - 复制 `selenium-pom.xml` 的内容到 `pom.xml` 中
 - 创建libs文件夹，添加
 -
 - 打开File | Project Structure...
 -
3. 新建一个Java Class如下：

```
public class SeDemo {  
    public static void main(String[] args) {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("https://www.baidu.com");  
        try {  
            sleep(2000);  
        } catch (InterruptedException ignored) {  
        }  
    }  
}
```

使用Maven创建项目

Maven是一个用于项目构建的工具，通过它便捷的管理项目的生命周期。即项目的jar包依赖，开发，测试，发布打包。

下面我自己总结一下它的几个特点，看了这些特点，也许对maven有更多的了解。

1. jar包依赖

这个也许会maven最突出的特点了使用maven不需要上网单独下载jar包，只需要在配置文件pom.xml中配置jar包的依赖关系，就可以

自动的下载jar包到我们的项目中。这样，别人开发或者使用这个工程时，不需要来回的拷贝jar包，只需要复制这个pom.xml就可以自动的下载这些jar包。

而且，我们自己下载jar包，还有可能造成版本的不一致，这样在协同开发的过程中就有可能造成代码运行的不一致。通过使用maven精确的匹配jar包，就不会出现这种问题了。

2. 项目坐标

Maven通过特定的标识来定义项目名称，这样既可以唯一的匹配其他的jar包，也可以通过发布，使别人能使用自己的发布产品。这个标识就被叫做坐标，长的其实很普通，就是简单的xml而已：

```
<groupId>com.test</groupId>
<artifactId>maventest</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>maventest</name>
<url>http://maven.apache.org</url>
```

- **groupId**：所述的项目名称，由于有的项目并不是一个jar包构成的，而是由很多的jar包组成的。因此这个groupId就是整个项目的名称。
- **artifactId**：包的名称。
- **version**：版本号。
- **packaging**：包的类型，一般都是jar，也可以是war之类的。如果不填，默认就是jar。
- **name**和**url**，一个是名称，一个是maven的地址。主要就是上面的几个参数。

开始使用WebDriver

接下来我们尝试几个简单的例子来体会一下Selenium WebDriver的使用。

示例1

```
// 引入WebDriver的包
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

// 创建浏览器对象
WebDriver driver = new FirefoxDriver();

// 打开百度网站
driver.get("https://www.baidu.com");
```

使用JUnit编写测试

在上述环境搭建好以后，我们便可使用JUnit来编写自动化脚本程序，执行Selenium自动化测试。

通过上面的例子，我们可以简单的写出Selenium WebDriver的脚本，但是对于测试工作来说，上述的脚本还远远不够。因为上述的脚本没有“检查”。

接下来我们将会使用Java语言的JUnit框架展开“检查”。

JUnit为我们在项目测试中常用到的单元测试框架，很多程序员的理想套件，本篇文章主要介绍JUnit4的常见annotation。

初始化标注

在JUnit4提供了setUp()和tearDown()，在每个测试函数调用之前/后都会调用。

@Before: Method annotated with **@Before** executes before every test.

@After: Method annotated with **@After** executes after every test.

如果在测试之前有些工作我们只想做一次，用不着每个函数之前都做一次。比如读一个很大的文件。那就用下面两个来标注：

@BeforeClass

@AfterClass

注意：

@Before/@After 可以有多个；**@BeforeClass/@AfterClass** 只有一个

如果我们预计有Exception，那就给**@Test**加参数：

@Test(expected = XXXException.class)

如果出现死循环怎么办？这时timeout参数就有用了：

```
@Test(timeout = 1000)
```

如果我们暂时不用测试一个用例，我们不需要删除或者都注释掉。只要改成：

```
@Ignore
```

你也可以说明一下原因@Ignore("something happens")

使用junit需要以下简单的两步：

- 引入junit包 `import org.junit.*;`
- 测试方法以test开头，方法上加 @Test

junit示例

```
import static org.hamcrest.CoreMatchers.allOf;
import static org.hamcrest.CoreMatchers.anyOf;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.not;
import static org.hamcrest.CoreMatchers.sameInstance;
import static org.hamcrest.CoreMatchers.startsWith;
import static org.junit.Assert.assertThat;
import static org.junit.matchers.JUnitMatchers.both;
import static org.junit.matchers.JUnitMatchers.containsString;
import static org.junit.matchers.JUnitMatchers.everyItem;
import static org.junit.matchers.JUnitMatchers.hasItems;

import java.util.Arrays;

import org.hamcrest.core.CombinableMatcher;
import org.junit.Test;

public class AssertTests {

    @Test
    public void testAssertArrayEquals() {
        byte[] expected = "trial".getBytes();
        byte[] actual = "trial".getBytes();
        org.junit.Assert.assertArrayEquals("failure - byte arrays not same", expected, actual);
    }

    @Test
    public void testAssertEquals() {
        org.junit.Assert.assertEquals("failure - strings not same", 51, 51);
    }

    @Test
    public void testAssertFalse() {
        org.junit.Assert.assertFalse("failure - should be false", false);
    }

    @Test
    public void testAssertNotNull() {
        org.junit.Assert.assertNotNull("should not be null", new Object());
    }

    @Test
    public void testAssertNotSame() {
        org.junit.Assert.assertNotSame("should not be same Object", new Object(), new Object());
    }

    @Test
    public void testAssertNull() {
        org.junit.Assert.assertNull("should be null", null);
    }

    @Test
    public void testAssertSame() {
        Integer aNumber = Integer.valueOf(768);
        org.junit.Assert.assertSame("should be same", aNumber, aNumber);
    }

    // JUnit Matchers assertThat
```

```

@Test
public void testAssertThatBothContainsString() {
    org.junit.Assert.assertThat("albumen", both(containsString("a")).and(containsString("b")));
}

@Test
public void testAssertThatHasItemsContainsString() {
    org.junit.Assert.assertThat(Arrays.asList("one", "two", "three"), hasItems("one", "three"));
}

@Test
public void testAssertThatEveryItemContainsString() {
    org.junit.Assert.assertThat(Arrays.asList(new String[] { "fun", "ban", "net" }), everyItem(containsString("n")));
}

// Core Hamcrest Matchers with assertThat
@Test
public void testAssertThatHamcrestCoreMatchers() {
    assertThat("good", allOf(equalTo("good"), startsWith("good")));
    assertThat("good", not(allOf(equalTo("bad"), equalTo("good"))));
    assertThat("good", anyOf(equalTo("bad"), equalTo("good")));
    assertThat(7, not(CombinableMatcher.<Integer> either(equalTo(3)).or(equalTo(4))));
    assertThat(new Object(), not(sameInstance(new Object())));
}

@Test
public void testAssertTrue() {
    org.junit.Assert.assertTrue("failure - should be true", true);
}
}

```

###理解unit框架提供的各种断言方法

方法 Method	检查条件
<code>assertEquals(msg, a, b)</code>	<code>a == b</code> , <code>msg</code> 可选, 用来解释失败的原因
<code>assertNotEquals(msg, a, b)</code>	<code>a != b</code> , <code>msg</code> 可选, 用来解释失败的原因
<code>assertTrue(msg, x)</code>	<code>x</code> 是真, <code>msg</code> 可选, 用来解释失败的原因
<code>assertFalse(msg, x)</code>	<code>x</code> 是假, <code>msg</code> 可选, 用来解释失败的原因
<code>assertSame(msg, a, b)</code>	<code>a</code> 不是 <code>b</code> , <code>msg</code> 可选, 用来解释失败的原因
<code>assertNull(msg, x)</code>	<code>x</code> 是 <code>null</code> , <code>msg</code> 可选, 用来解释失败的原因
<code>assertNotNull(msg, x)</code>	<code>x</code> 不是 <code>null</code> , <code>msg</code> 可选, 用来解释失败的原因
<code>assertThat(msg, actual, matcher)</code>	高级应用

在IDEA中使用Git集成操作

Git 环境的搭建

1. 在windows系统中安装Git-2.x.1-32-bit.exe或者Git-2.x.1-64-bit.exe，根据电脑系统为32位或者64位选择正确的对应的git版本。安装包可以在git官网下载 <https://git-scm.com>
2. 在Windows系统中安装TortoiseGit，同样需要根据系统为32位和64位进行选择安装。
3. 安装完毕以后，在非系统盘创建'Git'文件夹【这条是建议】
4. 在上一步创建的文件夹中，右键鼠标，选择'Git clone'，里面填写 git 项目的路径。例如在Coding.net 或者 Github.com 网站中的项目地址。"https://git.coding.net/lintyleo/WeekendSelenium.git"
5. 此后，请务必在刚才克隆后的文件夹中，进行操作，便可以在IDEA中集成Git操作。
6. 用IDEA创建新的项目，或者打开老的项目，这些项目必须存放在上一步克隆的地址中，方可进行操作。
7. 如果IDEA中没有发现Git的菜单，请选择 VCS | Enable VCS 菜单进行激活Git。然后再尝试。

##WebDriver API 基本操作

这里我们将会开始WebDriver API的基本操作，从元素定位以及浏览器的基本操作开始。

定位符

元素的定位和操作是自动化测试的核心部分，其操作是建立在定位的基础上的。因此我们首先要开始定位元素。

在html里面，元素具有各种各样的属性。我们可以通过这样唯一区别其他元素的属性来定位到这个元素。WebDriver提供了一系列的元素定位方法。常见的有以下几种：

- id
- name
- class name
- tag
- link text
- partial link text
- xpath
- css selector

查找简单元素

我们从简单的一个元素开始定位。最基本的方法是id和name。大多数元素有这两个属性，在对控件的id和name命名是一般也会使其有意义，而取不同的名字。

例如我们看下面这段html

```
<input id="search" type="text" name="q" value="" class="input-text" maxlength="128" />
```

我们可以使用对应的方法来定位这个input

```
findElement(By.id("search"))
findElement(By.name("q"))
findElement(By.className("input-text"))
```

这里我们开始用最简单的方式来尝试定位

查看下面一个例子

```
driver.get("http://pro.demo.zentao.net")
// 用name定位用户文本输入框
WebElement accountField = driver.findElement(By.name("account"))
// 用name定位密码文本输入框
WebElement passwordField = self.driver.findElement(By.name("password"))
accountField.clear()
passwordField.clear()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

// 输入用户名demo
accountField.sendKeys("demo")
// 输入密码123456
passwordField.sendKeys("123456")
driver.findElement("submit").click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

WebElement companyname = driver.findElement(By.id("companyname"))
assertEquals("标题验证", "demo项目管理系统", companyname.text)
```

示例2

```
driver.findElement(By.id("menuproduct")).click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.findElement(By.id("menuproject")).click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.findElement(By.id("menuqa")).click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.findElement(By.id("menudoc")).click()
```

```
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.findElement(By.id(("menureport"))).click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.findElement(By.id(("menucompany"))).click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.findElement(By.linkText(("退出"))).click()
driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);
```

此外XPath定位和CSS Selector定位，和定位一组元素这样的内容案例，将在后续的讲义继续探讨和讲解。

控制浏览器

浏览器的控制也是自动化测试的一个基本组成部分，我们可以将浏览器最大化，设置浏览器的高度和宽度以及对浏览器进行导航操作等。

```
// 浏览器最大化
driver.manage().window().maximize();

// 设置浏览器的高度为800像素，宽度为480像素
driver.manage().window().setSize(new Dimension(800, 600))

// 隐式等待
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

// 浏览器后退
driver.navigate().back();

// 浏览器前进
driver.navigate().forward()
```