基于 SELENIUM 的自动化测试架构

> 非常感谢各位付费打赏本篇文章，笔者在此感谢各位。

目前市面上有分门别类的自动化测试工具，这篇文章将讨论开源自动化测试工具 `Selenium` 的使用，以及围绕该工具进行自动化测试的理念、方案以及测试架构的构建。

1. 工具的使用

1.1 Selenium 介绍

`Selenium` 是开源的自动化测试工具，它主要是用于Web 应用程序的自动化测试，不只局限于此，同时支持所有基于web 的管理任务自动化。

`Selenium` 官网的介绍如下：

> Selenium is a suite of tools to automate web browsers across many platforms.
>
> - runs in many browsers and operatingsystems
> - can be controlled by many programming languages and testing frameworks.

Selenium 是用于测试 Web 应用程序用户界面 (UI) 的常用框架。它是一款用于运行端到端功能测试的超强工具。您可以使用多个编程语言编写测试，并且 Selenium 能够在一个或多个浏览器中执行这些测试。

Selenium 经历了三个版本：Selenium 1，Selenium 2 和 Selenium 3。Selenium 也不是简单一个工具，而是由几个工具组成，每个工具都有其特点和应用场景。

Selenium 诞生于 2004 年，当在 ThoughtWorks 工作的 Jason Huggins 在测试一个内部应用时。作为一个聪明的家伙，他意识到相对于每次改动都需要手工进行测试，他的时间应该用得更有价值。他开发了一个可以驱动页面进行交互的 Javascript 库，能让多浏览器自动返回测试结果。那个库最终变成了 Selenium 的核心，它是 Selenium RC（远程控制）和 Selenium IDE 所有功能的基础。Selenium RC 是开拓性的，因为没有其他产品能让你使用自己喜欢的语言来控制浏览器。这就是 Selenium 1。

然而，由于它使用了基于 Javascript 的自动化引擎，而浏览器对 Javascript 又有很多安全限制，有些事情就难以实现。更糟糕的是，网站应用正变得越来越强大，它们使用了新

浏览器提供的各种特性，都使得这些限制让人痛苦不堪。

在 2006 年，一名 Google 的工程师， Simon Stewart 开始基于这个项目进行开发，这个项目被命名为 WebDriver。此时，Google 早已是 Selenium 的重度用户，但是测试工程师们不得不绕过它的限制进行工具。Simon 需要一款能通过浏览器和操作系统的本地方法直接和浏览器进行通话的测试工具，来解决Javascript 环境沙箱的问题。WebDriver 项目的目标就是要解决 Selenium 的痛点。

到了 2008 年，Selenium 和 WebDriver 两个项目合并。Selenium 有着丰富的社区和商业支持，但 WebDriver 显然代表着未来的趋势。两者的合并为所有用户提供了一组通用功能，并且借鉴了一些测试自动化领域最闪光的思想。这就是 Selenium 2。

2016 年，Selenium 3 诞生。移除了不再使用的 Selenium 1 中的 Selenium RC，并且官方重写了所有的浏览器驱动。

1.2 Selenium 工具集

- Selenium IDE

  Selenium IDE (集成开发环境) 是一个创建测试脚本的原型工具。它是一个 Firefox 插件，实现简单的浏览器操作的录制与回放功能，提供创建自动化测试的建议接口。Selenium IDE 有一个记录功能，能记录用户的操作，并且能选择多种语言把它们导出到一个可重用的脚本中用于后续执行。

- Selenium RC

  Selenium RC 是selenium 家族的核心工具，Selenium RC 支持多种不同的语言编写自动化测试脚本，通过selenium RC 的服务器作为代理服务器去访问应用从而达到测试的目的。

  selenium RC 使用分Client Libraries 和Selenium Server。

  - Client Libraries 库主要主要用于编写测试脚本，用来控制selenium Server 的库。
  - Selenium Server 负责控制浏览器行为，总的来说，Selenium Server 主要包括 3 个部分：Launcher、Http Proxy、Core。

- Selenium Grid

  Selenium Grid 使得 Selenium RC 解决方案能提升针对大型的测试套件或者哪些需要运行在多环境的测试套件的处理能力。Selenium Grid 能让你并行的运行你的测试，也就是说，不同的测试可以同时跑在不同的远程机器上。这样做有两个有事，首先，如果你有一个大型的测试套件，或者一个跑的很慢的测试套件，你可以使用 Selenium Grid 将你的测试套件划分成几份同时在几个不同的机器上运行，这样能显著的提升它的性能。同时，如果你必须在多环境中运行你的测试套件，你可以获得多个远程机器的支持，它们将同时运行你的测试套件。在每种情况下，Selenium Grid 都能通过并行处理显著地缩短你的测试套件的处理时间。

- Selenium WebDriver

  WebDriver 是 Selenium 2 主推的工具，事实上WebDriver是Selenium RC的替代品，因为Selenium需要保留向下兼容性的原因，在 Selenium 2 中， Selenium RC才没有被彻底的抛弃，如果使用Selenium开发一个新的自动化测试项目，那么我们强烈推荐使用Selenium2 的 WebDriver进行编码。另外， 在Selenium 3 中，Selenium RC 被移除了。

1.3 Selenium WebDriver 的使用

接下来的内容，我们将会主要讨论本文的核心重点， Selenium WebDriver 的使用。Selenium WebDriver 是从 Selenium 2 开始使用并流行， 在 Selenium 3 中得到进一步发展的工具，是当前 Selenium 的最核心的工具。WebDriver 具有清晰面向对象 API，能以最佳的方式与浏览器进行交互。

Selenium WebDriver 就好比是一个懂浏览器的司机，它可以在浏览器的网页上行走，走到网页内容的任何地方，可以参观网页的任何地方，并且和网页进行交互。那么作为测试工程师，如果想和这样的一个司机打交道，就必须要掌握和这样的司机打交道的技能。

学习司机会使用的语言，并使用该语言，以及合适的沟通工具与司机进行交流：

- Java
- Python
- C#
- JavaScript
- PHP
- Ruby

给司机找到合适的浏览器，以便司机在浏览器上行走。

- 支持多种浏览器，包括 Chrome，Firefox，IE，Edge，Safari，Opera 等

Selenium WebDriver 的使用主要分为两个场景：

1）懂浏览器的司机，WebDriver 类

- 用 WebDriver 提供的模板，制造一个司机。
- WebDriver 的第一个应用场景，就是这个司机的各种能力，包括但不限于以下的部分：

  1. 用浏览器打开指定的 URL
  2. 清理浏览器的Cookie
  3. 在浏览器中寻找页面元素（Web Element）
     - 查找单个的指定元素

- 查找一组有共同属性的元素，并进行遍历等。

4. 控制浏览器的基本操作：
   - 前进： forward()
   - 后退： backward()
   - 刷新： refresh()
   - 关闭： close()
   - 最大化： maximize_window()
   - 弹窗： switch_to_alert()

5. 返回浏览器的属性
   - current_url
   - title

6. 执行 JavaScript 脚本

2）在浏览器中找到的元素，WebElement 类

司机在浏览器中找到页面元素以后，对它做的任何操作，都是 WebDriver 的第二个主要的场景：

1. 点击该元素： `click()`
2. 清除该元素原有的文字： `clear()`
3. 给该元素填写新的文字： `send_keys()`
4. 获取该元素的文字： text
5. 获取该元素的指定属性： `get_attribute()`
6. 对该元素进行二次加工
   - 构成 frame 并切换进去： `switch_to.frame(元素)`
   - 构成 select 并进行操作： `Select(元素).select_by_value()`

1.4 Selenium 环境搭建

Selenium 的环境搭建基本上分为三个部分：

1. 安装编程语言以及IDE（集成编程环境），用来操作 WebDriver
2. 安装 Selenium WebDriver，实现浏览器的测试
3. 安装浏览器，和指定的驱动，完成自动化测试的执行

接下来分别用目前市面上主流的 Java 和 Python 环境进行搭建。

Java 版本

1. 安装 Java 语言，即 JDK。推荐 1.8 的版本。

2. 安装 IDE，推荐 JetBrains IDEA Community Edition，这款是目前主流的 Java 开发工具，而且社区版是免费使用的，拥有出色的用户交互，以及使用界面，完全能够应对一般的自动化测试的编程。当然如果你更加熟悉Eclipse，也是可以使用的。

3. 安装 Selenium，推荐使用 Maven 直接引入依赖。当自动化测试作为团队共同的解决方案，而不是一个人单独维护的方案的时候，团队需要统一的 Selenium 版本以及共同的 API 操作，Maven 的使用，无疑简化了项目的难度，很好的解决了此类问题。

```xml
<!--
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.3.1</version>
</dependency>
<!--
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-remote-driver -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-remote-driver</artifactId>
    <version>3.3.1</version>
</dependency>
```

当然，你也可以直接下载 Selenium Standalone Server，并且手工引用 `Jar 包` 到项目中。最新版3.3.1的下载地址：3.3.1

4. 安装 浏览器和浏览器的驱动。

以上的步骤，便完成了 Java + Selenium 的环境搭建。

Python 版本

1. 安装 Python 语言。

   Python 目前并行了两套版本，2.x 和 3.x。如果你之前没有 Python 的使用经验，建议使用 Python 3.x 版本。两套版本互相不兼容，并且 Python 从 3.5（含）开始，不再支持 Windows XP 系统，请注意。

2. 安装 Python IDE，推荐 JetBrains Pycharm Community Edition。

3. 安装 Selenium，推荐使用 pip 的方式直接安装。在命令行下，直接输入：

```shell
# 在Selenium 3 发布之前，可以用下面命令直接装selenium
# -U = --upgrade 升级安装
# 自动安装最新版
# 目前3.0发布以后，这个命令直接安装 3.3.1 的最新版
```

   pip install -U selenium

```
# 如果要装2.53.6版本
pip install selenium==2.53.6
```

如果你处于没有外网的情况下，可以采用源码解压安装，前往https://pypi.python.org/pypi/selenium下载最新版的PyPI版本的Selenium，解压后执行：

```
shell
python setup.py install
```

4. 安装 浏览器和浏览器的驱动。

以上的步骤，便完成了 Python + Selenium 的环境搭建。

1.5 Selenium 编程

通过前面的介绍，我们知道 Selenium 支持多种语言，并且推荐使用面向对象的方式进行编程。接下来我们将着重介绍如何使用面向对象的方式进行编程。

在面向对象的理念看来，任何的编码，都是由对象而来的，这里也不例外。和之前介绍 WebDriver 时候的描述对应，我们需要用到两种主要的类，并将其实例化。

- WebDriver 类：主要靠直接实例化该类为对象，然后用其对象直接调用该类的方法和属性
- WebElement 类：主要通过 WebDriver 类实例化的对象，通过对页面元素的查找，得到 WebElement 类的对象，然后调用该类的方法和属性。

具体的使用如下，以 Java 语言 和 火狐浏览器为例

```
// 声明 Web司机，司机是一个火狐类的对象
// 需要用 new 关键字来实例化对象， () 代表构造方法
WebDriver driver = new FirefoxDriver();

// Web司机去打开网站
driver.get("http://demo.ranzhi.org");

// 线程停止 3000 毫秒，使得 Web司机有足够的时间打开网址
Thread.sleep(3000);

// 选择 用户名 密码 并依次输入 demo 和 demo （用户名和密码都是 demo）
weAccount = driver.findElement(By.cssSelector("#account"));
weAccount.clear();
weAccount.sendKeys("demo");

wePassword = driver.findElement(By.cssSelector("#password"));
wePassword.clear();
wePassword.sendKeys("demo");
```

```
// 选择 登录 按钮，并点击 click
driver.findElement(By.cssSelector("#submit")).click();
Thread.sleep(5000);
```

上述代码中，使用了一个 WebDriver 类 的对象，即第3行，声明了该类的对象，并赋值给变量 driver，接着变量 driver 作为 WebDriver 类的对象，使用了多个 WebDriver 类的方法。

- get(url): 第6行，打开网址
- findElement(by, selector): 第12、16、21行都使用了该方法，同时通过对该方法的调用，分别各产生了一个 WebElement类的对象， `weAccount` ， `wePassword` 和最后一个匿名的对象，并通过产生的三个对象，调用 WebElement 类的方法
    - clear()：清理页面元素中的文字
    - sendKeys(text)：给页面元素中，输入新的文字
    - click()：鼠标左键点击页面元素

正是通过这样的面向对象的方式，产生 Web司机（ WebDriver类的对象 ），并且通过 Web 司机不懈的努力，寻找到各种 Web元素（ WebElement类的对象 ）进行操作，这样便实现了 Selenium WebDriver 作为一款出色的浏览器测试工具，进行浏览器UI界面的自动化测试的代码编写和用例执行。

上述代码，也同样可是使用 Python 作为编程语言进行操作，如下所示：


```
# 声明一个司机，司机是个Firefox类的对象
driver = webdriver.Firefox()

# 让司机加载一个网页
driver.get("http://demo.ranzhi.org")

# 给司机3秒钟去打开
sleep(3)

# 开始登录
# 1. 让司机找用户名的输入框
we_account = driver.find_element_by_css_selector('#account')
we_account.clear()
we_account.send_keys("demo")

# 2. 让司机找密码的输入框
we_password = driver.find_element_by_css_selector('#password')
we_password.clear()
we_password.send_keys("demo")

# 3. 让司机找 登录按钮 并 单击
driver.find_element_by_css_selector('#submit').click()
sleep(3)
```

常用的重点编程对象有如下几种：

WebDriver 类

- get(url)： 打开web页面
- findElement(by, selector)：查找一个页面元素。配合浏览器的开发者工具（推荐 Chrome Developer Tools），有8中方式定位元素：
    - id：元素标签的 id
    - css selector：元素标签的 selector
    - xpath：元素标签的 XPath
    - link text：元素标签的完整文字
    - name：元素标签的 name
    - class name：元素标签的 class name
    - tag name：元素标签的 tag name
    - partial link text：元素标签的部分文字

- findElements(by, selector)：查找一组具有同一属性的页面元素，方式同上。
- deleteAllCookies()：清理 Cookies
- executeJs(js)：执行 JavaScript
- quit()： 退出浏览器
- getTitle()：当前浏览器的标题属性
- getCurrentUrl()：获取当前浏览器的 URL

WebElement 类

- click()：点击改元素
- clear()：清除当前的文本
- sendKeys(text)：给当前的元素输入文字
- getAttribute(attribute)：获取当前元素的某一种属性
- getText()：获取当前元素的文字属性
- isDisplayed()：获取当前元素的 Displayed 属性
- Select：针对 `<select>` 元素进行的操作
    - selectByValue(value)
    - selectByIndex(index)

```
// 找到该 <select> 元素 we
WebElement we = driver.findElement(by, selector);
// 使用该元素，实例化一个 Select 类的对象 s
Select s = new Select(we);
s.selectByValue(value);
// 或者用index
s.selectByIndex(value)
```

- 鼠标事件，有关鼠标的操作，不只是单击，有时候还要做右击、双击、拖动等操作。这些操作包含在ActionChains类中。

    - contextClick()：右击
    - douchClick()：双击
    - dragAndDrop()：拖拽
    - moveToElement()：鼠标停在一个元素上
    - clickAndHold()：按下鼠标左键在一个元素上

- Frame: 针对 <iframe> 元素标签进行的操作

  很多的页面中，都包含有内联框架（iframe），那么如果需要获取到其内部的元素并进行操作，必须首先切换到该内联框架中，当操作完成以后，再退出到最外层的网页中

```
// 找到该内联框架的元素
WebElement we = driver.findElement(by, selector);
// 利用WebDriver 的对象driver，切换到该内联框架中
driver.switchTo().frame(we);
// TODO: 进行各种操作
// 退出该内联框架，返回到外层的网页中
driver.switchTo().defaultContent();
```

2. 理念与方案

在第一部分，工具的使用中，我们重点介绍了 Selenium 工具的编程，但是这样其实对于自动化测试来讲，还远远不够。自动化测试的重点，其实依旧是测试用例的编写和执行，要求代码中，具备测试用例的属性；同时要求测试的代码能够很好的组织起来，通过抽取和分离的理念，实现良好的测试。主要达到以下的几个目的：

- 具备测试用例的属性

  测试代码，可以轻松的具备测试用例的属性，主要包括测试前置条件、清理操作、和断言（检查）。

- 避免重复代码的编写和复制

  通过模块化拆分页面功能，避免 WebDriver类的重复实例化和调用，也避免同样的测试步骤，多次的编写和复制

- 测试数据单独存放

  测试代码中不需要包含需要输入的测试数据，而是把测试数据单独存放在 文本文件，或者数据库中。

- 封装底层的测试工具

  对 Selenium WebDriver 这种第三方的工具，进行封装起来，避免代码中直接调用

- 必须使用源代码管理工具

  无论是否是 一人团队 ，源代码管理工具的使用都是积极地和必要的，推荐使用 Git 。

接下来的描述，将会对上述的理念依次进行讲解，实现自动化测试的方案。

2.1 使用单元测试框架

在第一部分，我们对 Selenium WebDriver 的使用，仅仅停留在让网页自动的进行操作的阶段，并没有对任何一个步骤进行“检查”。当然，这样没有“检查”的操作，实际上是没有测试意义的。那么第一项，我们需要解决的便是“检查”的问题。

所谓“检查”，实际上就是断言。对需要检查的步骤操作，通过对预先设置的期望值，和执行结果的实际值之间的对比，得到测试的结果。在这里，我们并不需要单独的写 `if` 语句进行各种判定，而是可以使用编程语言中对应的单元测试框架，即可解决好此类问题。

目前 Java 语言主流的单元测试框架有 JUnit 和 TestNG。Python 语言主流的单元测试框架有 unittest 。本小节的内容，主要介绍 TestNG 和 unittest 的使用，探讨单元测试框架如何帮助自动化测试。

TestNG

接下来我们将会使用Java语言的 `TestNG` 框架展开“检查”。TestNG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，通过注解（ annotation ）的方式进行操作。

在 `TestNG` 提供了 `@BeforeMethod` 和 `@AfterMethod` ，在每个测试函数调用之前/后都会调用。

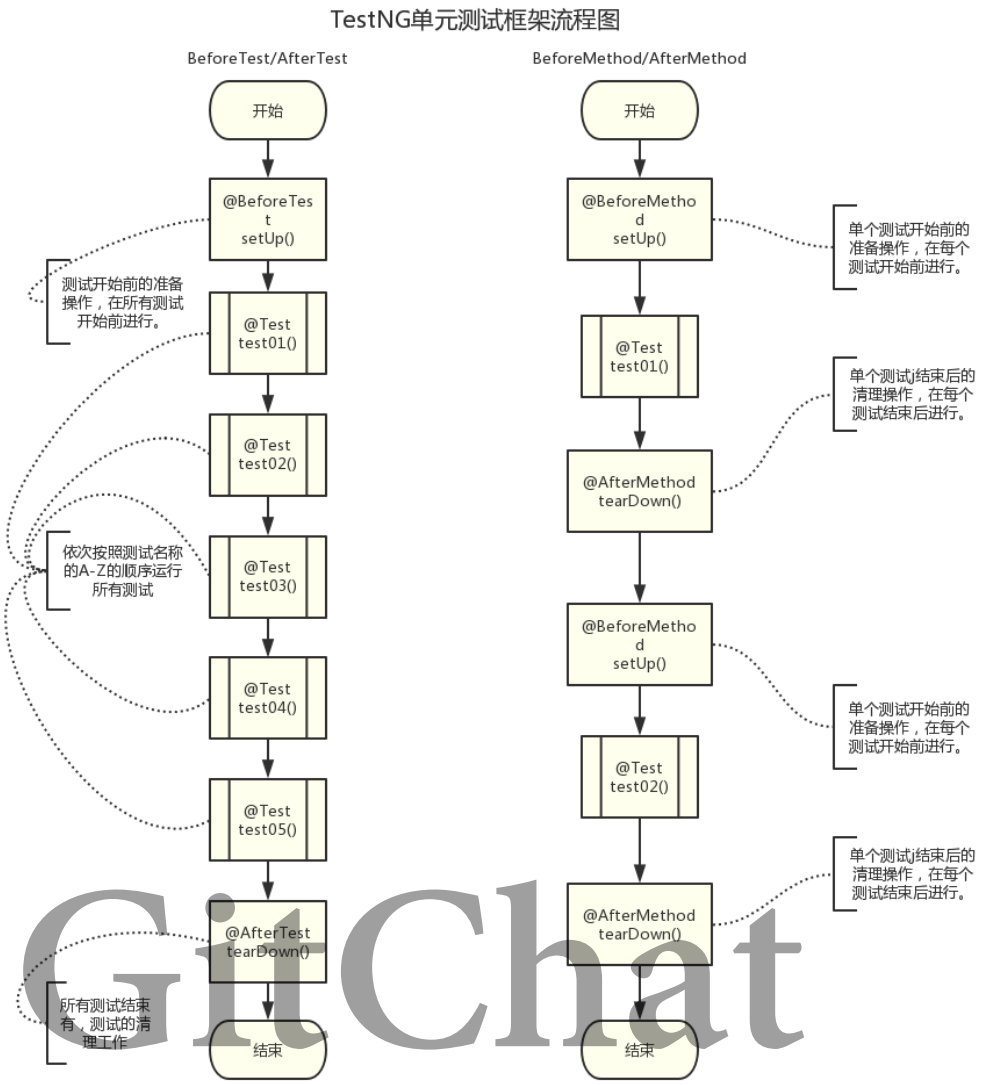
- `@BeforeMethod` : Method annotated with `@BeforeMethod` executes before every test method.
- `@AfterMethod` : Method annotated with `@AfterMethod` executes after every test method.

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，那就用下面两个来标注：

- `@BeforeTest` : 在第一个 test method 开始执行前，执行。
- `@AfterTest` : 在最后一个 test method 执行后再执行。

接下来我们用具体的代码示例，解释单元测试框架的使用

- TestNG 框架图



- TestNG 断言

enter image description here

- TestNG 的引入

  这里我们依旧使用 Maven 的方式，引入 TestNG 到项目中。

```
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>6.8</version>
</dependency>
```

- TestNG 的使用

```
/**
 * Created by Linty on 1/8/2017.
```

```
 * 使用 @BeforeTest 和 @AfterTest 进行测试框架操作
 * 如果直接运行整个测试，运行步骤如下
 * 首先运行 @BeforeTest
 * 然后运行 test01ChangeLanguage
 * 接着运行 test02LogIn
 * 最后运行 @AfterTest
 */
public class RanzhiMainTest {
    // 声明两个全局变量，每个方法都可以用下面的变量。
    private WebDriver baseDriver = null;
    private String baseUrl = null;
    /**
     * 测试登录
     * 需要用 @Test 注解进行标注
     * 这样 不用main()方法 便可直接运行测试
     *
     * @throws InterruptedException
     */
    @Test
    public void test02LogIn() throws InterruptedException {
        WebDriver driver = this.baseDriver;
        driver.get(this.baseUrl);
        Thread.sleep(2000);

driver.findElement(By.id("account")).sendKeys("admin");

driver.findElement(By.id("password")).sendKeys("123456");
        driver.findElement(By.id("submit")).click();
        // 点击登录按钮后，需要等待浏览器刷新
        Thread.sleep(2000);
        String expectedUrl = this.baseUrl +
"sys/index.html";
        // driver.getCurrentUrl() -- 获取当前的浏览器URL
        Assert.assertEquals(driver.getCurrentUrl(),
expectedUrl);
    }
    /**
     * 测试切换语言
     * 把系统语言切换成 English
     * 然后查询 语言的按钮 是不是变成了 English
     * @throws InterruptedException
     */
    @Test
    public void test01ChangeLanguage() throws
InterruptedException {
        WebDriver driver = this.baseDriver;
        driver.get(this.baseUrl);
        Thread.sleep(2000);
        // 点击语言按钮
        driver.findElement(By.cssSelector("#langs >
button")).click();
        Thread.sleep(500);
```

```
        // 用Css Selector 选择 英文
        driver.findElement(By.cssSelector("#langs > ul > li:nth-child(3) > a")).click();
        // 浏览器需要刷新，等待2秒钟
        Thread.sleep(2000);
        // 检查按钮上的字是不是变成了 English
        String expected_language = "English";
        String actual_language =
                driver.findElement(By.cssSelector("#langs > button")).getText();
        Assert.assertEquals(actual_language, expected_language);
    }
    /**
     * 测试前置条件
     * 在所有的测试开始前 执行一次
     */
    @BeforeTest
    public void setUp() {
        this.baseDriver = new FirefoxDriver();
        this.baseUrl = "http://demo.ranzhi.org/";
    }
    /**
     * 测试清理操作
     * 在所有的测试结束后 执行一次
     */
    @AfterTest
    public void tearDown() {
        this.baseDriver.quit();
    }
}
```


- unittest

  接下来我们将会使用 Python 语言的 `unittest` 框架展开“检查”。 `unittest` 框架的原本的名字是PyUnit。是从JUnit 这样一个被广泛使用的 经典的Java应用开发的单元测试框架创造而来。类似的框架还有NUnit（.Net开发的单元测试框架）等。我们可以使用unittest框架为任意Python项目编写可理解的单元测试集合。现在这个unittest已经作为Python的标准库模块发布。我们安装完Python以后，便可以直接使用unittest。

  使用unittest需要以下简单的三步：

  - 引入unittest模组
  - 继承unittest.TestCase基类
  - 测试方法以 `test` 开头

  unittest 并未使用 Java 语言常见的注解方式，依旧停留在 比较早期的 Java 版本中依靠方法名称进行识别的方式。主要有以下两个固定名字的方法：

- setUp()：在每个测试方法运行前，执行。是测试前置条件。
- tearDown()：在每个测试方法运行后执行，是测试清理操作。

具体的代码如下：

```
class RanzhiMainTest(unittest.TestCase):
  """
  第一步：import unittest
  第二步：继承 unittest.TestCase 类
  第三步：测试的方法，以test_ 开头
  第四步：重写 setUp() 作为 测试前置条件，注意setUp的大小写，必须一致
  第五步：重写 tearDown() 作为 测试清理操作，注意 tearDown的大小
写，必须一致
  """
  # 全局变量
  base_driver = None
  base_url = None

  def setUp(self):
      self.base_driver = webdriver.Firefox()
      self.base_url = "http://demo.ranzhi.org/"

  def tearDown(self):
      self.base_driver.quit()

  def test_01_change_language(self):
      driver = self.base_driver
      driver.get(self.base_url)
      sleep(2)

      driver.find_element_by_css_selector("#langs >
button").click()
      sleep(1)

      driver.find_element_by_css_selector("#langs > ul >
li:nth-child(3) > a").click()
      sleep(2)

      # 页面应该换成英语了
      actual_lang =
driver.find_element_by_css_selector("#langs > button").text
      expected_lang = "English"
      # 与Java的TestNG 相反，先写期待值，再写实际值
      self.assertEqual(expected_lang, actual_lang)

  def test_02_log_in(self):
      driver = self.base_driver
      driver.get(self.base_url)
      sleep(2)
```

```
        driver.find_element_by_id("account").send_keys("admin")

        driver.find_element_by_id("password").send_keys("123456")
              driver.find_element_by_id("submit").click()

              sleep(3)
              actual_url = driver.current_url
              expected_url = self.base_url + "sys/index.html"
              self.assertEqual(expected_url, actual_url)
```

2.2 使用 Page Object 设计模式

Page Object设计模式是Selenium自动化测试项目的最佳设计模式之一，强调测试、逻辑、数据和驱动相互分离。

Page Object模式是Selenium中的一种测试设计模式，主要是将每一个页面设计为一个Class，其中包含页面中需要测试的元素（按钮，输入框，标题等），这样在Selenium测试页面中可以通过调用页面类来获取页面元素，这样巧妙的避免了当页面元素id或者位置变化时，需要改测试页面代码的情况。当页面元素id变化时，只需要更改测试页Class中页面的属性即可。

它的好处如下：

- 集中管理元素对象，便于应对元素的变化
- 集中管理一个page内的公共方法，便于测试用例的编写
- 后期维护方便，不需要重复的复制和修改代码

具体的做法如下：

1. 创建一个页面的类
2. 在类的构造方法中，传递 WebDriver 参数。
3. 在测试用例的类中，实例化页面的类，并且传递在测试用例中已经实例化的WebDriver对象。
4. 在页面的类中，编写该页面的所有操作的方法
5. 在测试用例的类中，调用这些方法

实现的示例：

Page 基类

- 设计了一个基本的Page类，以便所有的页面进行继承，该类标明了一个sub page类的基本功能和公共的功能。

- 全局变量： this.baseDriver，让所有的子类都使用的。

```
// 基类的变量，所有继承的类，都可以使用
BoxDriver baseDriver;
```

- 构造方法：
  - 默认的构造方法，无参数的构造方法

```
public BasePage() {
}
```

  - 传递 driver的构造方法

```
public BasePage(BoxDriver driver) {
 this.baseDriver = driver;
}
```

  - 私有的常量：存放元素的定位符

```
java
private String START_BUTTON_SELECTOR = "s,#start > div";
private final String EXIT_MENU_TEXT = "l,%s";
```


- 成员方法：
  - 每个子类都需要的系统功能：
    - open

```
public void open(String url) throws
InterruptedException {
    this.baseDriver.navigate(url);
    Thread.sleep(2000);
}
```

  - 所有子类（页面）都具有的业务功能
    - selectApp
    - logout
  - Sub Pages(s)子类
  - 具体的页面的类，定义了某个具体的页面的功能
  - 必须继承基类

```
  java
public class AdminPage extends BasePage {
```

```
}
```

- 创建构造方法，带driver 参数

```
 java
public AdminPage(BoxDriver driver) {
super(driver);
}
```

- 特定页面的业务
- 使用基类的 `this.baseDriver` 成员变量
- Tests 类
- 这部分描述的是具体的测试用例。
- 声明全局变量

```
java
private BoxDriver baseDriver = null;
private String baseUrl = null;
private LoginPage loginPage = null;
private AdminPage adminPage = null;
```


- 调用各种页面（ pages ）

1. 实例化Page

```
this.loginPage = new LoginPage(this.baseDriver);
this.adminPage = new AdminPage(this.baseDriver);
```

2. 使用page的对象，调用成员方法

```
java
loginPage.open(this.baseUrl);
loginPage.changeLanguage(lang);
loginPage.login("admin", "123456", true);
loginPage.selectApp(AppType.Admin);
adminPage.clickAddMemberButton();
adminPage.addMemberData(member);
```

2.3 使用数据驱动

主要的数据驱动方式有两种：

- 通过文本文件或者 Excel 文件存储数据，并通过程序读取数据，遍历所有的行
- 通过数据库存储数据，并通过程序和 SQL 脚本读取数据，遍历所有的行

通过 CSV 文件 或者 MySQL 数据库，是主流的数据驱动方式。当然数据驱动也可以结合单元测试框架的参数化测试进行编写（此部分本文不做具体描述）。

无论使用了 哪一种（CSV 或者 MySQL），读取数据后都要进行遍历操作。

Java 代码

```
 java
   // 布尔型 true false
   boolean isFirstLine = true;
   // 循环每一个行，接下来根据每一行的值（数据），进行测试
   for (CSVRecord row : csvData) {
     if (isFirstLine) {
       isFirstLine = false;
       continue;
       // continue的作用
       // 当前循环到此为止，直接进入下一条循环
     }
Member member = new Member();
member.setAccount(row.get(0));
member.setRealName(row.get(1));
if (Objects.equals(row.get(2), "f")) {
  member.setGender(Member.Gender.Female);
} else {
  member.setGender(Member.Gender.Male);
}


member.setDept(Integer.parseInt(row.get(3)));
member.setRole(Integer.parseInt(row.get(4)));
member.setPassword(row.get(5));
member.setEmail(row.get(6));
// TODO: 进行测试

   }
```

Python 代码

```
python
  is_header = True
  for row in csv_data:
      if is_header:
          is_header = False
          continue
      # dict 类型的数据
      member_data = {
          "account": row[0],
```

```
        "real_name": row[1],
        "gender": row[2],
        "dept": row[3],
        "role": row[4],
        "password": row[5],
        "email": row[6]
    }
    # TODO: 进行测试
```

2.4 封装 Selenium WebDriver

封装是一个面向对象编程的概念，是面向对象编程的核心属性，通过将代码内部实现进行密封和包装，从而简化编程。对Selenium进行封装的好处主要有如下三个方面：

- 使用成本低
  1. 不需要要求所有的测试工程师会熟练使用Selenium，而只需要会使用封装以后的代码。
  2. 不需要对所有的测试工程师进行完整培训。也避免工作交接的成本。
  3. 测试人员使用统一的代码库。

- 维护成本低
  1. 通过封装，在代码发生大范围变化和迁移的时候，不需要维护所有代码，只需要变更封装的部分即可。
  2. 维护代码不需要有大量的工程师，只需要有核心的工程师进行封装的维护即可。

- 代码安全性
  1. 对作为第三方的Selenium进行封装，是代码安全的基础。
  2. 对于任何的代码的安全隐患，必须由封装来解决，使得风险可控。
  3. 使用者并不知道封装内部的代码结构。

封装的具体示例：

- 找到一个指定输入框(selector)，并且输入指定的字符(text)

  type(selector, text)

  不用在业务逻辑中，使用多次的 `findElement(By.id(...))`

  ```
  public void type(String selector, String text) {
  WebElement we = this.locateElement(selector);
  we.clear();
  we.sendKeys(text);
  }
  ```

- 找到一个可以点击的元素(selector)，并且点击(click)

click(selector)

```
public void click(String selector) {
this.locateElement(selector).click();
}
```

- 找到一个指定的frame，并且切换进去

  switchToFrame(selector)

```
public void switchToFrame(String selector) {
WebElement we = this.locateElement(selector);
this.baseDriver.switchTo().frame(we);
}
```

- 找到一个指定的select，并且通过index进行选择

  selectByIndex(selector, index)

```
public void selectByIndex(String selector, int index) {
WebElement we = this.locateElement(selector);
Select s = new Select(we);
s.selectByIndex(index);
}
```


以上的代码是封装了 `locateElement()` 的几种方法，在具体使用封装过的代码的时候，只需要简单的调用即可。接下来的重点，是介绍 `locateElement(selector)` 的封装方式。

- 查找元素： `findElement(By...)`
- 支持各种的查找：8种方式都需要支持，必须通过 `selector` 显示出分类
  - `selector` 中需要包含一个特殊符号
  - 实例化 封装好的类的时候，需要约定好是什么特殊符号
    1. 强制性用 硬编码 `hard code` 来实例化，例如 `,` 或者 `?` 或者 其他非常用字符 `=>`
    2. 或者，构造方法中，传递 `this.byChar`
- 要把查找到元素的返回给调用的地方：必须要有返回值，类型是 `WebElement`

```
private WebElement locateElement(String selector) {
  WebElement we;
  // 如果定位符中 有 分隔符，那么就从分隔符处分成两段
  // 第一段是By
  // 第二段是真正的定位符
  // 如果没有分隔符，就默认用 id 定位
```

```
    if (!selector.contains(this.byChar)) {
      // 用 id 定位
      we = this.baseDriver.findElement(By.id(selector));
    } else {
      // 用 分隔符 分成两个部分
      String by = selector.split(this.byChar)[0];
      String value = selector.split(this.byChar)[1];
      we = findElementByChar(by, value);
    }
    return we;
  }
```

- 接下来的重点，是实现 `findElementByChar(by, value)`

```
  private WebElement findElementByChar(String by, String value) {
    WebElement we = null;
    switch (by.toLowerCase()) {
      case "id":
      case "i":
        we = this.baseDriver.findElement(By.id(value));
        break;
      case "css_selector":
      case "css":
      case "cssselector":
      case "s":
        we = this.baseDriver.findElement(By.cssSelector(value));
        break;
      case "xpath":
      case "x":
        we = this.baseDriver.findElement(By.xpath(value));
        break;
      case "link_text":
      case "link":
      case "text":
      case "linktext":
      case "l":
        we = this.baseDriver.findElement(By.linkText(value));
        break;
        //TODO: other by type
    }
    return we;
  }
```


使用上面的封装类，就需要指定特定的 selector。

类型	示例(分隔符以逗号，为例)	描述
id	“account” 或者 “i,account” 或者 “id,account”	分隔符左右两侧不可以空格
xpath	“x,//*[@id=”s-menu-dashboard”]/button/i”	
css selector	“s,#s-menu-dashboard > button > i”	
link text	“l,退出”	
partial link text	“p,退”	
name	“n,name1”	
tag name	“t,input”	
class name	“c,dock-bottom	

调用的具体示例：

```
void logIn(String account, String password) throws
InterruptedException {
  BoxDriver driver = this.baseDriver;
  driver.type("account", account);
  driver.type("password", password);
  driver.click("submit");
  // 点击登录按钮后，需要等待浏览器刷新
  Thread.sleep(2000);
}
```


至此，自动化测试的方案如下图所示：

1. 封装 Selenium 为 BoxDriver
2. 在 测试用例中，实例化 BoxDriver，产生 bd 对象
3. 使用 bd 对象，构造 业务模块的实例化对象，产生 common
4. 使用 common 在测试用例中，构建测试步骤
5. 使用数据驱动的外部数据，通过读取，进行测试
6. 执行整个用例

2.5 使用 Git 进行源代码管理

Git 是目前主流的源代码管理工具，本文推荐的两个 IDE 工具： JetBrains IDEA 和 JetBrains Pycharm 都是默认支持 Git 的。只需要按照以下步骤进行配置，便可以通过 IDE 工具对代码进行提交，这样可以防止代码丢失，以及方便的查询代码的修改历史，同时很方便团队的编码。

使用编程工具提交代码(用IDEA 或者 PyCharm)

1. 在本地的Git项目文件夹中创建项目
    - 比如： `git\selenium_pro`
    - 用IDEA 创建 `Maven` 项目
        - 注意 `project location` 务必在 `git\selenium_pro`
        - 比如 项目名字 `HelloSelenium`
        - 项目路径： `git\selenium_pro\HelloSelenium`

2. 编写 代码
3. 选择 IDEA 的 VCS | `Enable Version Control Integration`
4. 弹出的窗口选择 Git
    - 所有的代码文件名字变成红色

5. 右键 左侧的项目名字 `HelloSelenium`
    - 选择 `Git | Add`
    - 所有的代码文件名字变成绿色

6. 右键 左侧的项目名字 `HelloSelenium`
    - 选择 `Git | Commit Directory`
    - 左侧填写 说明
    - 右侧勾选 `Reformat Code`
    - 选择右下角 `Commit And Push`
    - “XX文件已经committed”以后，点击 `Push`
    - 输入用户名 + 密码，勾选 remember password
    - push successful

3. 架构的构建

将第二部分的自动化测试方案付诸实践，再对自动化测试的结果生成测试报告，便基本上实现了自动化架构的构建。比较重要的地方是第三方工具 Selenium WebDriver 的封装工作。事实上，如果封装了别的工具，便可以实现其他方面的自动化测试。

3.1 代码的构建

在第二部分的基础上，我们添加上去测试组织和测试报告的功能，并且将不同作用的代码分开放入不同的文件夹中，实现代码的构建。

测试用例的组织和执行

- 测试集合 test suite
- 测试用例的集合
    - 多个测试用例的类
    - 测试用例类的方法

- 测试运行运行 test suite

具体的方法

- TestNG

  在项目中，创建 testng.xml 文件如下

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Default Suite">
    <test name="SolutionSelenium">
        <classes>
            <class name="cases.LoginTests">
                <methods>
                    <!-- 添加指定的测试 -->
                    <include name="testLoginByCsv"/>
                    <include name="testLoginByCsv2"/>
                    <include name="testLoginByCsv3"/>
                </methods>
            </class> <!-- cases.LoginTests -->
            <class name="cases.AdminTests">
                <methods>
                    <!-- 去除指定的测试 -->
                    <exclude name="testAddMember"/>
                </methods>
            </class>
        </classes>
    </test> <!-- SolutionSelenium -->
</suite> <!-- Default Suite -->
```


    - 指定测试的类
    - 指定测试的类中的方法
        - include: 一个个方法包含进来
        - exclude: 去除指定的方法

  然后，编写测试入口的脚本：Main.java

```
public class Main {
    public static void main(String[] args) {
        TestNG test = new TestNG();
        List<String> suites = new ArrayList<>();
        suites.add("testng.xml");
        test.setTestSuites(suites);
        test.run();
    }
}
```

    - 测试报告的生成

- TestNG测试报告
  - TestNG 自带的测试报告 xml / html
  - ReportNG 测试报告的插件，停止开发和支持。

  - ExtentReport 测试报告
    - TestNG有默认的测试报告生成器 Listener
    - 使用 ExtentReport 重写一个 Listener
    - 让 TestNG 使用我们写好的 Listener 生成报告
  - 步骤
    - 引入 ExtentReport 到 pom.xml

```
<dependency>
  <groupId>com.relevantcodes</groupId>
  <artifactId>extentreports</artifactId>
  <version>2.41.2</version>
</dependency>
```


    - 目前 ExtentReport 有两个版本：2 和 3
      - 2 全部开源的
      - 3 有付费版和开源版
    - 官方网址：http://extentreports.com/community/
    - 编写 Listener : ExtentReporterNgListener

```
@Override
public void generateReport(List<XmlSuite> xmlSuites,
List<ISuite> suites, String outputDirectory) {
  Date date = new Date();
  SimpleDateFormat formatter = new
SimpleDateFormat("yyyyMMdd_HHmmss");
  String time = formatter.format(date);

  String reportName =
String.format("ExtentReportTestNG_%s.html", time);
  // 创建报告
  // NetworkMode.OFFLINE  支持断网查看报告
  this.extent = new ExtentReports(
    outputDirectory + File.separator + reportName,
    true, NetworkMode.OFFLINE);
```

```
      for (ISuite suite : suites) {
        Map<String, ISuiteResult> result =
    suite.getResults();

        for (ISuiteResult r : result.values()) {
          ITestContext context = r.getTestContext();
          // 创建测试节点
          buildTestNodes(context.getPassedTests(),
    LogStatus.PASS);
          buildTestNodes(context.getFailedTests(),
    LogStatus.FAIL);
          buildTestNodes(context.getSkippedTests(),
    LogStatus.SKIP);
        }
      }
      extent.flush();
      extent.close();
    }
```

- 修改 testng.xml

在 `<suite />` 中添加 `<listener/>`


```
<listeners>
  <listener class-
name="runner.ExtentReporterNgListener"/>
</listeners>
```

- 正常运行测试

- unittest 的操作

- 添加 test suite

```
suite = unittest.TestSuite()
suite.addTest(LoginTests("test_login_by_csv"))
suite.addTest(LoginTests("test_login_by_csv2"))
suite.addTest(AdminTests("test_add_member_by_csv"))
```

这里可以配置测试，到外部文件，数据库中等。

```
shell
测试类,测试方法
LoginTests,test_login_by_csv
LoginTests,test_login_by_csv2
AdminTests,test_add_member_by_csv
```

- 实例化 test runner

```
# 测试报告的文件

test_time = time.strftime("%Y%m%d_%H%M%S", time.localtime())
report_file = open("reports\\ranzhi_automate_report_%s.html"
% test_time,
                   mode="wb")
runner = HtmlTestRunner(stream=report_file,
                        verbosity=2,
                        title="然之系统自动化测试报告",
                        description="具体测试报告内容如下: ")
```

HtmlTestRunner : 第三方测试报告运行器

- 用 test runner 去执行测试，产生报告。

```
runner.run(suite)
```

3.2 使用持续集成

持续集成，Continuous integration，简称CI。随着软件开发复杂度的不断提高，团队开发成员间如何更好地协同工作以确保软件开发的质量已经慢慢成为开发过程中不可回避的问题。尤其是近些年来，敏捷（Agile）在软件工程领域越来越红火，如何能再不断变化的需求中快速适应和保证软件的质量也显得尤其的重要。

持续集成正是针对这一类问题的一种软件开发实践。首先我们看一下，敏捷教父 `Martin Fowler` 对持续集成的定义：

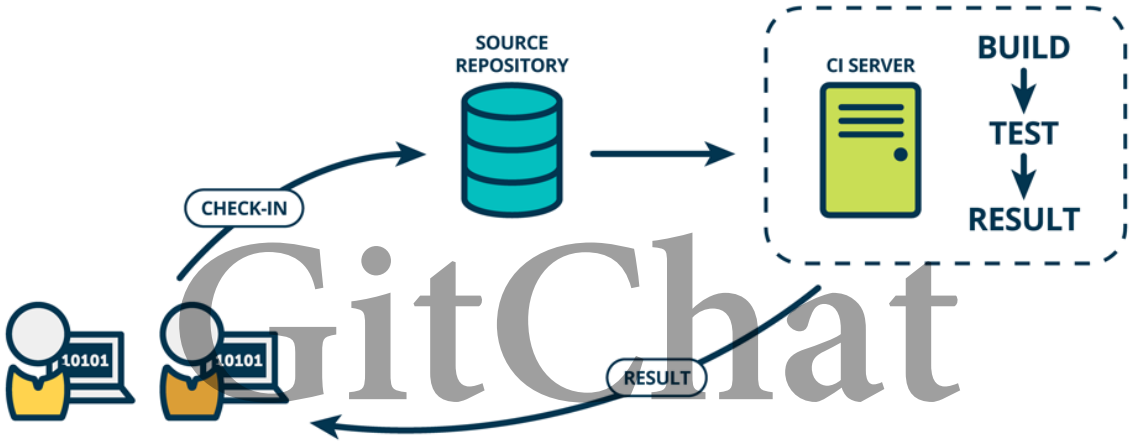
> Martin Fowler ： **Continuous Integration** is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

具体定义：持续集成式一种软件开发实践。它倡导团队的成员必须经常的集成他们的工作，通常至少每天一次甚至更多次集成。每次集成都需要通过自动化的构建（包括编译代码、构建应用、部署程序以及自动化测试）来验证，从而尽早尽快的发现集成中的错误。大量的团队利用这样的方式来更快的开发内聚的软件。大大减少此过程中的集成问题。

持续集成强调开发人员提交了新代码之后，立刻进行构建、（单元、自动化）测试。根据测试结果，我们可以确定新代码和原有代码能否正确地集成在一起。

首先，解释下集成。我们所有项目的代码都是托管在SVN服务器上。每个项目都要有若干个单元测试，并有一个所谓集成测试。所谓集成测试就是把所有的单元测试跑一遍以及其它一些能自动完成的测试。只有在本地电脑上通过了集成测试的代码才能上传到SVN服务器上，保证上传的代码没有问题。所以，集成指集成测试。

再说持续。不言而喻，就是指长期的对项目代码进行集成测试。既然是长期，那肯定是自动执行的，否则，人工执行则没有保证，而且耗人力。对此，我们有一台服务器，它会定期的从SVN中检出代码，并编译，然后跑集成测试。每次集成测试结果都会记录在案。完成这方面工作的就是下面要介绍的Jenkins软件。当然，它的功能远不止这些。在我们的项目中，执行这个工作的周期是1天。也就是，服务器每1天都会准时地对SVN服务器上的最新代码自动进行一次集成测试。



通过持续基础，可以将自动化测试良好的应用起来，只要是代码发生了变动，或者是进行代码的构建，那么在构建后都可以通过持续基础，自动的执行测试脚本，验证改进的功能。

当前主要的持续基础工具有：

- Jenkins
- TeamCity

通过持续集成，可以进一步完善自动化测试的架构，使得自动化测试真正的帮助项目，保证测试质量。

3.3 参考代码

> 本文的示例代码，放到了Github上面，以下附上Github的地址。为了简化操作，本代码使用了 Selenium 2 的环境，搭配 46.0（含）以下的火狐浏览器，以便各位参考。

> 代码将会持续更新。

代码地址：https://github.com/lintyleo/seleniumpro

再次感谢各位花时间阅读本文，由于时间紧迫，以及才疏学浅，文中难免出现各种疏漏，还请各位多多指教。不当之处，还欢迎各位指出，在此谢过。