

WEB 自动化测试之 WEBDRIVER(PYTHON)

--从入门到精通

测试开发工程师丛书

宋现锋 著

目 录

前言	- 1 -
第一章 自动化测试简述	- 2 -
1.1 为什么要进行自动化?	- 2 -
1.2 自动化测试能做什么?	- 3 -
1.3 如何实施自动化测试?	- 5 -
第二章 测试环境搭建	- 8 -
2.1 SELENIUM IDE	- 8 -
2.1.1 Selenium IDE 简介	- 8 -
2.1.2 Selenium 安装	- 9 -
2.1.3 Selenium IDE 的使用	- 10 -
2.1.4 Selenium IDE 其他的重要功能	- 14 -
2.2 FIREBUG	- 16 -
2.2.1 FireBug 简介	- 16 -
2.2.2 FireBug 的使用	- 16 -
2.3 WEBDRIVER PYTHON 开发环境搭建	- 19 -
2.3.1 工具选择	- 19 -
2.3.2 Python+Webdriver 安装	- 19 -
2.3.3 Eclipse python 开发环境配置	- 22 -
2.4 本章小结	- 28 -
第三章 WEBDRIVER API 简介	- 29 -
3.1 WEBDRIVER API	- 29 -
3.2 页面元素定位	- 30 -
3.2.1 WebElement 对象提供的各种定位元素策略	- 31 -
3.2.2 定位方法的选择	- 33 -
3.2.3 Xpath 定位方法深入探讨	- 39 -
3.2.4 元素定位不到的原因及解决办法	- 42 -
3.3 检查点的设置	- 46 -
3.3.1 常用的检测点设置方法	- 47 -
3.3.2 检测点设置技巧	- 50 -
3.3.3 检测点设置中常见的错误	- 51 -
3.4 本章小结	- 52 -
第四章 自动化测试用例初探	- 53 -
4.1 第一个测试用例 HELLO WORLD	- 53 -

4.1.1 Selenium IDE 录制	- 53 -
4.1.2 手动编写自动化测试用例	- 59 -
4.2 测试用例代码结构	- 63 -
4.2.1 自动化测试代码结构	- 63 -
4.2.2 测试用例编写过程中常见的问题	- 64 -
4.3 本章小结	- 65 -

第五章 常用页面元素自动化操作 - 66 -

5.1 输入类元素	- 66 -
5.1.1 input 和 textarea 元素	- 66 -
5.1.2 input 上传文件	- 67 -
5.1.3 特殊按键的输入	- 68 -
5.2 单击类元素	- 69 -
5.2.1 按钮类元素单击	- 69 -
5.2.2 超级链接单击操作	- 70 -
5.2.3 鼠标右击和双击操作	- 70 -
5.3 选择类元素	- 71 -
5.3.1 超级链接类选择	- 71 -
5.3.2 单选框选择	- 71 -
5.3.3 复选框选择	- 72 -
5.3.4 下拉菜单类选择	- 72 -
5.4 获取元素的文本	- 73 -
5.5 页面或 IFRAME 切换	- 74 -
5.5.1 页面间的切换	- 74 -
5.5.2 iframe 间的切换	- 75 -
5.6 本章小节	- 75 -

第六章 自动测试实施 - 76 -

6.1 评审被测试对象功能	- 76 -
6.2 评审被测对象编码	- 78 -
6.3 自动化测试框架的选择	- 78 -
6.4 自动化测试用例运行环境	- 80 -
6.5 自动化代码架构的规划	- 80 -
6.5.1 代码架构规划的原因	- 81 -
6.5.2 如何规划代码架构	- 81 -
6.6 编写自动化测试用例	- 83 -
6.6.1 WebDriverHelp 类的内容	- 84 -
6.6.2 QT_Operations 类的内容	- 87 -
6.6.3 DataOperations 类的内容	- 88 -
6.6.4 具体的测试用例	- 90 -
6.6.5 测试用例的具体数据	- 91 -
6.7 本章小结	- 92 -

第七章 自动化无人值守运行..... - 93 -

7.1 TESTSUITE 组织测试用例.....	- 93 -
7.2 利用 JENKINS 来管理自动化测试用例.....	- 95 -
7.3 JENKINS 高级配置.....	- 98 -
7.3.1 自动化执行测试用例.....	- 99 -
7.3.2 程序执行失败邮件通知.....	- 100 -
7.3.3 测试执行失败短信通知.....	- 101 -
7.4 自动化测试用例报告	- 103 -
7.5 本章小结	- 105 -

第八章 影响自动化实施的非技术因素..... - 106 -

8.1 非技术性因素的影响	- 107 -
8.2 学习自动化测试的方法	- 108 -
8.3 本章小结	- 109 -

前言

提起自动化，我们通常会想到工业自动，一组大型机器通过操作面板的控制，可以自己在那运行，又快又好地生产出产品来。而控制这些儿机器的就是单片机技术，也是计算机行业的。而在软件开发，网站设计中难道就没有自动化吗？

当然有，随着软件，网站和手机 App 的盛行，功能越来越大，而每次的优化或是增加新功能后，会不会影响原来的功能也变得越来越重要。所以有上线或是发布前的回归测试，把主要功能在上线前后都回归测试一下。这些主功能是不变的，如果让专一的测试人员日复一日的去回归测试，时间消耗就先不说了，测试人员一定会厌烦的，带着情绪回归，就难免不出错了。

由此就引入了自动化测试，用程序去测试程序，一则覆盖全面，执行效率高；二则不会遗漏，不需要专门的人员去执过。所以越来越多的公司开始招聘这个自动化测试工程师，而这类人才介于开发和测试之间。技术比测试好，没有开发强，造成了这类人才很少，工资也相对来高些儿。于是很多测试人员想学习自动化测试技术，以便去做测试开发的工作，但是网上或是市面上这类教程很少，给大家造成了很多困难。结合我多年做测试开发的经验，就写了这个教程，希望能帮助大家学习自动化测试！

作者：宋现锋，1985 年，男，汉族，高级测试开发工程师!!!!

第一章 自动化测试简述

今天是 2015 年新年的第一天上班，公司来的人还不多，没有具体的工作要做。应年前的承诺，写一下自动化测试相关的教程吧，希望对学习自动化测试的小伙伴们有所帮助。

1.1 为什么要进行自动化？

最近几年自动化测试好像挺火的，去各大招聘网站上一搜，什么自动化测试工程师，测试开发工程师啊，都挺多的而且待遇也挺高！那么回过头来，我们要考虑一下，为什么很多公司都这么重视自动化测试呢？这要考虑到现在的用户群体的特性：

- (1) 要求网站或应用响应快。现在生活节奏这么快，网速也是越来越快，无线，4G 等等，如果你的网站或应用反应慢，大家就没有耐心等待，直接关掉或退出。这也是京东为什么这么受欢迎的原因，送货快。
- (2) 要求网站或是应用稳定。如果一个网站或是应用三天两头出问题，不用说别人，就我们自己也会很头痛。自家的孩子没有办法，只能用了，可是用户呢？现在同类网站或是应用这么多，你让我不爽，我就不需要你。
- (3) 要求操作简单。互联网最基本的原则就是简单，一键操作或是点击一个按钮就能完成的事情，就不要一二三四步的来操作了。如果你的操作太复杂，很多用户就会选择放弃的。

鉴于以上种种原因，我们要求网站或是应用快速，稳定和简单，所以每次更新或是上线前后都会花大量的时间来进行回归测试。而回归测试如果让人工来做的话，费时费力，而且容易造成遗漏；如果用自动化回归的话，配合一些儿管理工具来做自动触发，省时省力，而且可以做到无人值守。这就是为什么越来越多的公司重视自动化的原因，自动化测试工程师或是测试开发工程师的就业范围也比较广。

1.2 自动化测试能做什么？

自动化火起来之后，很多公司都相应地成立了自动化测试团队，可是我们也会经常遇到下面的问题：新开发了一个功能，然后领导就让自动化团队来测试；或是某个用户说哪儿的样式不对，老大就说你的自动化测试怎么没有发现这个 Bug 呢？等等诸如此类的问题，让做自动化测试的同学很头痛。

那自动化测试究竟能做什么呢？首先，自动化测试不是万能的，不要以为有了自动化测试，就万无一失了。以下几种情况，自动化测试就无法实施：

- (1) 样式问题。你很难用自动化代码来判断颜色对不对？这个地方应该不应该换行？字体是不是你想要的，字号不符合要求？等等！
- (2) 新开发的功能。新开发的代码或是功能在测试阶段是不适合做自动化测试的，因为这个时候有很多不确定因素存在。当然简单的录制回放也是能做，不过成本太高，要反复修改，而且一

旦测试完成，录制的代码就没有用了。所以不要有了自动化测试，就轻视手工测试工作。

- (3) 反复改版的功能。自动化测试是根据页面元素来定位操作的，如果被测试的网站或应用正处的改版阶段，是不能实施自动化测试的。此时若实施自动化测试，维护成本相当大，而且每一次改动都有可能影响到脚本的运行结果，得不偿失。
- (4) 需要验证码的功能。很多网站为了防刷，就会在一些儿重要的地方加上验证码，不管是手机验证码或是图形验证码，对自动化测试都是阻碍。虽然可以用程序识别图形验证码，调用接口来截取手机验证码，但是这会降低自动化测试的速度和安全性，不到万不得已，尽量避免。
- (5) 支付相关的功能。涉及到钱的功能，最好不要自动化。因为回归测试很多是针对线上的，如果线下的话无所谓。线上回归，支付一般会跳到第三方页面，而且自动化的支付会给财务结算造成困惑。如果不是上面压下来的死命令，还是不要给其他同事的工作添麻烦了。

其次，要明白，自动化测试其实很简单，就是做主功能的回归！自动化测试主要是确保被测试对象的核心功能正常工作，如：众筹网（www.zhongchou.cn）主要回归以下几个功能：

- (1) 登录注册：这是任何一个需要账号登录网站的主要功能，必须确保没有问题，否则会造成客户的流失。
- (2) 主页显示：主页上的内容必须显示无错，当然我们无法确定显

示的精确内容，这些内容会不断变化的，但是应该显示数据图片的地方必须确保有内容显示。

- (3) 浏览项目：浏览所有的项目，根据分类浏览项目和其他的筛选条件查看项目。项目显示符合条件，项目数据显示完整。
- (4) 搜索项目：根据关键字查找项目功能。根据分词拆分原则，有没有显示相关的项目。
- (5) 支持项目：支持某个项目，注：这里和支付相关，不过可以检测到支付前的操作，此时不会支付，不会产生真实的订单。而支持项目的用户的支持的项目中会有这个项目，支付状态是未支付。
- (6) 发起项目：发起一个项目，不过最后不要提交，保存成草稿就行了，也不会产生垃圾数据。

这几个功能项目就是众筹网最主要的功能，任何一个出现了问题，都会影响用户操作的。而你要测试的时候，首先要评选出被测试对象的主要功能进行自动化，必须确保不影响用户的操作。

1.3 如何实施自动化测试？

经过上面的讨论，我们认识到了自动化的重要性，以及哪些能做自动化，哪些不适合做自动化，可是当我们拿到一个被测试的对象，实施自动化的时候，应该如何着手呢？

此时不要慌张，应该从以下几个方面考虑及实施：

- (1) 评审被测试对象功能。评审被测试对象的时候主要包括：被测

试对象目前是不是稳定版的？有哪些儿主功能？目前自动化测试应该覆盖到什么程度？BVT 或是 80%，95%？最终确认下来自动化测试要编写的测试用例，如果有对应的手工测试用例最好。

(2) 评审被测试对象编码。自动化测试脚本一般都会和被测试对象使用同种语言或是类似的语言，这样兼容性好，支持函数比较多。评审被测试对象的编码，然后选择你最擅长的脚本语言作为自动化测试的编码语言。

(3) 自动化测试框架的选择。目前业内自动化测试框架和自动化测试工具多于牛毛，有点儿让人不知所措，但是也不能乱选，有以下几个参考标准：

一， 开源框架或工具。开源的框架或是工具限制比较少，而且支持的人也比较多，方便自己定制或是做二次开发。

二， 支持语言较多。框架或是工具支持的语言是重要标准之一，因为真正实施的时候，有很多外在的阻碍，支持的语言多以后变换的空间比较大。

三， 比较成熟的框架。不要选择使用人较少，偏僻的框架，这样的框架会存在 Bug，我们是在使用工具，没有必要花很多力气去解决框架的 Bug。而且要选择成熟的版本，如果不是使用最新的函数，还是使用稳定的版本比选择最新的版本好。

(4) 自动化测试脚本的运行环境。自动化测试脚本在 Windows 环境下运行，还是在 Linux 环境下运行？或者是其他的环境，运行

环境的不同，也会影响自动化脚本的选择。

- (5) 自动化代码架构的规划。当上面的问题都选择好后，就要规划一下自动化测试代码的架构了。虽然传统的流水账似有代码也能测试，当测试用例多的时候，就很难维护了。好的代码架构是非常有必要的，清晰明了，可读性和易维护性在后其是相当重要的。
- (6) 编写具体的测试用例。然后将第（1）中确定的手工测试用例转化成自动化脚本编写的测试用例，调试成功即可。
- (7) 自动触发及执行。当自动化测试稳定后，可以接入到 Jenkins 等代码管理工具中，配置触发式执行或是定时执行，对被测试对象进行回归。

经过上面我们探讨，相信你对自动化测试有了一定的了解。那么要想动后写出自动的自动化测试用例，还是需要认真的学习的。下面的文章我们将探讨如何借助于 WebDriver 框架，用 python 语言编写自动的 Web 自动化测试用例。

第二章 测试环境搭建

经过上面的讨论，我们已经窥探到自动化测试的端倪，但是要想写出一个可以运行的自动化测试用例，我们还需要更深入的学习。以下的内容，我们将一步步的来探讨自动化测试的秘密。

“工欲善其事，必先利其器。”当然自动化测试也不例外，我们用到的工具及语言如下：

- (1) Selenium IDE：作为火狐的一个插件，是我们 Web 自动化测试中必不可少的。
- (2) FireBug：火狐的一个插件，帮助我们定位页面元素。
- (3) Eclipse：强大的代码编辑工具，相信大家并不陌生。
- (4) PyDev：Eclipse 的插件，用于编辑 Python 代码。
- (5) Python：我们测试用例的编码语言，请自行学习，本系统教程是不再讲究 Python 编程的内容。
- (6) WebDriver：selenium2.0，Web 自动化测试主要框架。

2.1 Selenium IDE

2.1.1 Selenium IDE 简介

Selenium 是一个用于 Web 应用程序测试的工具。Selenium 测试直接运行在浏览器中，就像真正的用户在操作一样。支持的浏览器包括 IE、Mozilla Firefox、Mozilla Suite 等。这个工具的主要功能包括：

- 测试与浏览器的兼容性——测试你的应用程序看是否能够很好得

工作在不同浏览器和操作系统之上。

- 测试系统功能——创建衰退测试检验软件功能和用户需求。支持自动录制动作和自动生成。
- 支持 python、Net、Java、Perl 等不同语言的测试脚本。Selenium 是 ThoughtWorks 专门为 Web 应用程序编写的一个验收测试工具。

Selenium IDE

Selenium IDE 是基于 FIREFOX 浏览器的一个插件，提供 GUI 界面来运行 Selenium 测试。Selenium IDE 提供脚本录制功能，可以将用户在浏览器中执行的操作记录下来，生成各种形式的脚本，可以将这些脚本保存供以后使用。

2.1.2 Selenium 安装

方法一，安装 Selenium IDE 的步骤如下：

- 从 www.openqa.org/selenium-ide/download.action 下载 Selenium IDE(一个 XPI 后缀的文件)。
- 启动 FIREFOX 浏览器，打开刚才下载的文件。
- 重启 FIREFOX 浏览器，在工具菜单下应该就可以看到 Selenium IDE 菜单项。

方法二，从 Firefox 浏览添加

- 打开浏览器，打开菜单“工具”→“附加组件”，在打开的页面搜索 Selenium。
- 安装 Selenium IDE Button，安装完成的重启浏览器。

➤ 定制工具栏，将 Selenium IDE Button 拖到工具栏上，然后单击这个 Button，按提示安装 Selenium IDE，这个时候选择最新的版本即可。

注：Selenium IDE 安装完成后，默认打开了所有支持语言的格式，如图 2.1.2 所示，我们可以禁用到不用的语言项。



图 2.1.2 Selenium IDE 支持的语言格式

2.1.3 Selenium IDE 的使用

下面我们将逐步讲 Selenium IDE 的使用方法：

- 安装 Selenium IDE。
- 启动 Selenium IDE: 打开“工具” —> “Selenium IDE”，IDE 启动后，弹出如图 2.1.3.1 对话框：



图 2.1.3 Selenium IDE 对话框

上图标明了一些 Selenium IDE 的主要功能。其中，由 Command, Target, Value 组成的表格就是脚本，每个脚本都是由一条一条的 Action(行为)组成，而每个 Action 又由 (Command, Target, Value) 三者组成。Command 就是《API 参考手册》提到的内容，Target 指的是 Web 中的某个对象，比如：文字，输入框等等，如果选取对象呢？呵呵，这里就用到了 XPath，不熟悉可以参考《XPath 的使用》，后面的课程我们也将讲到相关内容；而 Value 就是这个对象的值。

➤ 脚本的录制及运行

当弹出上面的 IDE 窗口后，我们就可以开始 Selenium 的脚本录制了，右上角有个红色的圆点，当它下按时(如上图)就表示 IDE 正在进行脚本录制。OK，开始录制，录制的时候，直接操作 Firefox 浏

览器窗口就可以了，IDE 会自动记录你的操作的，下面我演示一个例子：

（1）打开 IDE，在 Base URL 框内输入你要录制的网站，如：
<http://www.baidu.com>。

（2）在地址栏输入：<http://www.baidu.com/>，登录到百度首页。

（3）IDE 打开的时候就是录制状态，如果录制按钮没有按钮，按下该按钮然后刷新页面，在查询框输入“hyddd”。

（4）按“百度一下”按钮，查询结果。

（5）进入搜索结果页面后，右键单击第一条记录（即：hyddd - 博客园），在右键弹出菜单中，单击“Verify TestPersent hyddd - 博客园”。

（6）单击第一条记录（即：进入 hyddd - 博客园）

（7）Firefox 弹出一个新 Tab 页面，并进入了我的博客。

OK，现在看看我们的 Selenium IDE 录制的结果吧！如图 2.1.3.2 所示：

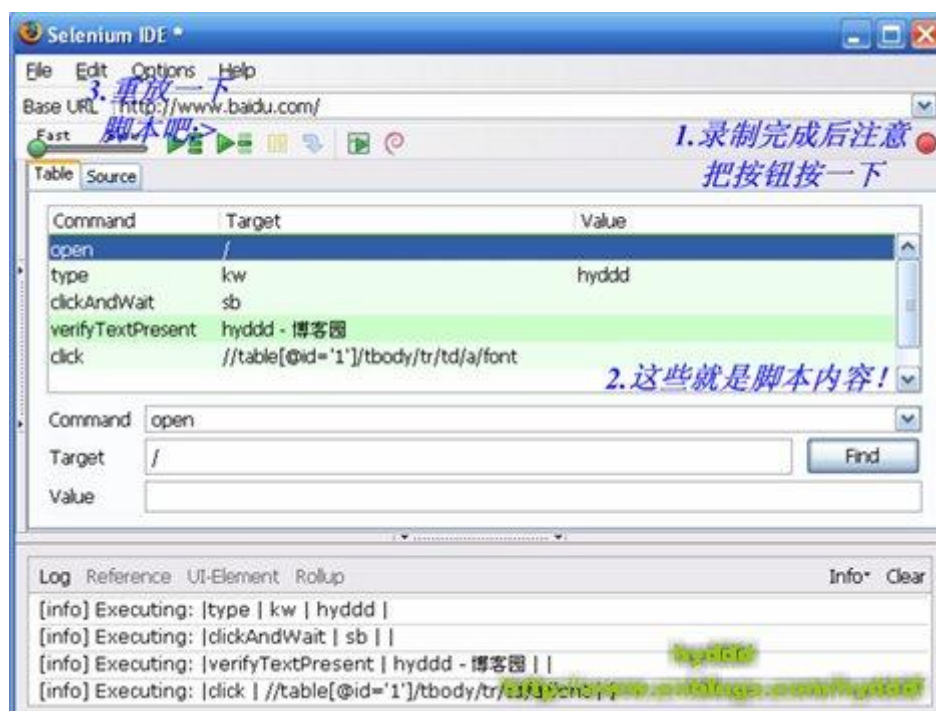


图 2.1.3.2 脚本录制结果

上图中，中间的表格就是录制的结果，你可以按“运行脚本”重新回放脚本看看，值得注意的是，在运行时，Firefox 可能会认为脚本中最后一个操作（即：步骤 7）为非法弹出框，浏览器会自动阻止其弹出，这个需要设置一下 Firefox，具体位置是：

Firefox→Menubar→Tools→options→content→Block pop-up

Window，你可以把钩去掉或者在 Exceptions 里面添加相应的网址。

到此为止，脚本录制圆满完成！在运行脚本后，你会发现 IDE 表格的颜色发生了变化，运行前，脚本表格为白色，成功运行完毕后，表格为青色，其中还分为深青色和浅青色两种，浅青色表示：动作成功，如：打开网页成功，点击按钮成功等等，而深青色表示：判断正确，如：“hyddd - 博客园”这段文字在页面中存在等等。

那如果运行失败会是什么样子呢？现在我们看看出错时的情况吧。如图 2.1.3.3 所示：

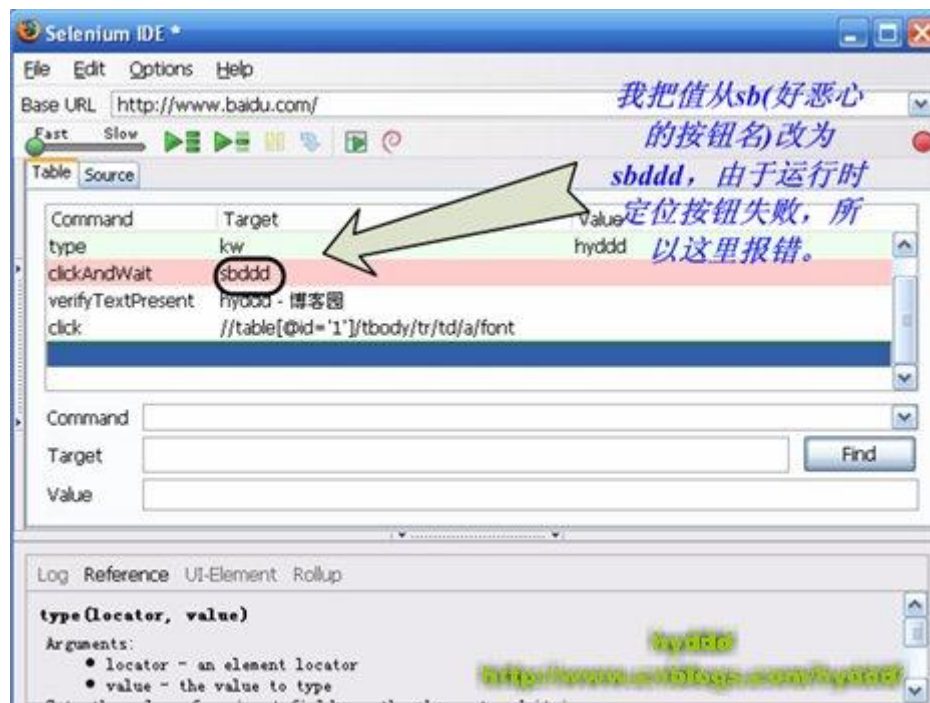


图 2.1.3.3 脚本运行错误的情况

出错时，表格可能会出现两种颜色，一种是浅粉红色，一种是深粉红色。浅粉红色表示判断结果为 false，这种情况案例还是会继续执行下去，判断的失败不会影响案例的运行，深粉红色表示动作失败，如：没有找到按钮等（如上图），这种情况下案例会停止运行。

2.1.4 Selenium IDE 其他的重要功能

➤ 脚本转化保存

Selenium IDE 录制完成脚本后，经过我们回放验证，发现可以达到我们的要求，此时我们最想做的事情就是把脚本转化成我们期望的形式保存来了。

不用担心，Selenium IDE 有这个功能，如图 2.1.4.1 所示：

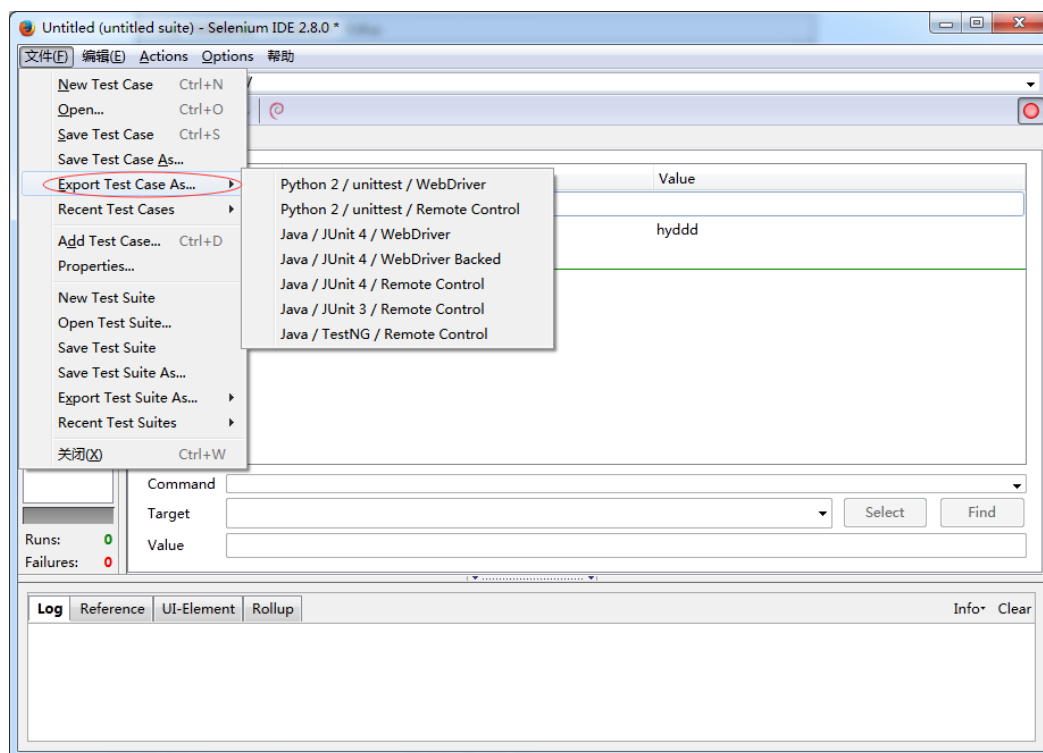


图 2.1.4.1 导出录制的脚本

打开“文件”-->“Export Test Case As...”，打开的二级菜单中就是可以转化成的格式。在我们安装 Selenium IDE 的时候，我们只保存了 python 和 java 格式的，而在此只有这两个格式。此时我们需要第一种格式“python2/unittest/WebDriver”，然后在打开的对话框中输入文件名，扩展名输入“.py”，保存即可！

➤ 验证定位方式是否正确

Selenium IDE 虽然可以录制脚本，可是在我们使用过程中，如果测试用例是采用模块化编码时，不需要反复录制的。此时，IDE 就可以帮我们验证我们的定位方法是否正确。

如图 2.1.4.2 所示：具体验证方法，我们将在定位页面元素章节讲解。

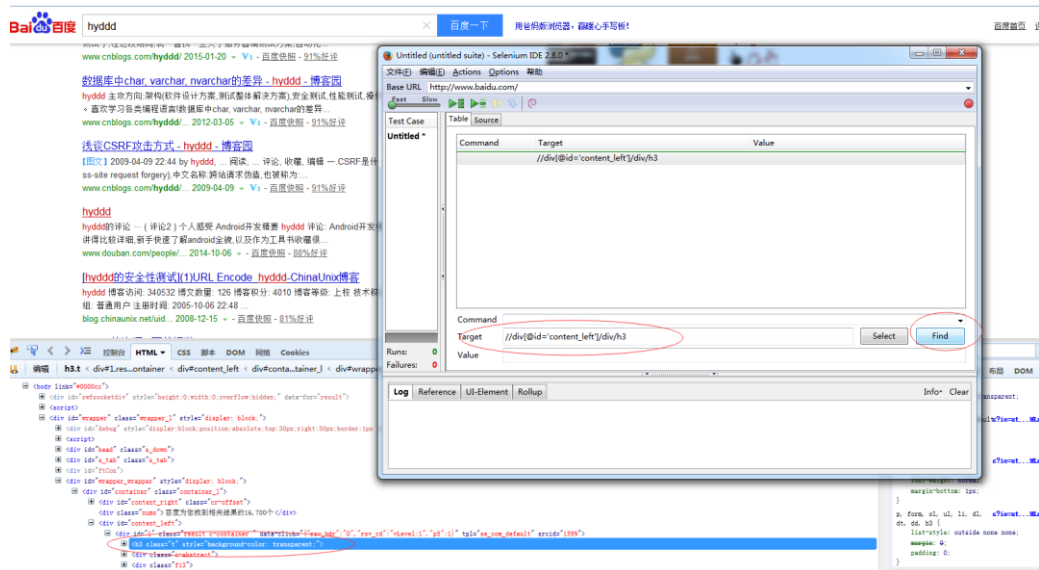


图 2.1.4.2 Selenium IDE 验证元素定位

2.2 FireBug

2.2.1 FireBug 简介

Firebug 是网页浏览器 Mozilla Firefox 下的一款开发类插件，现属于Firefox的五星级强力推荐插件之一。它集HTML查看和编辑、JavaScript 控制台、网络状况监视器于一体，是开发 JavaScript、CSS、HTML 和 Ajax 的得力助手。Firebug 如同一把精巧的瑞士军刀，从各个不同的角度剖析 Web 页面内部的细节层面，给 Web 开发者带来很大的便利。

2.2.2 FireBug 的使用

FireBug 的安装和使用，可以参考这个文档：

<http://jingyan.baidu.com/article/fd8044fa97e08c5030137a6c.html>，此文

档把 FireBug 的基本使用方法都介绍了一下，还是比较详细的，在此就不一一介绍了！

下面我就把我们自动化测试中对 Firebug 的应用，以一个实例来简单介绍一下：

➤ 页面元素定位

(1) 打开微博，登录一个账号，以本人的账号为例。

(2) 查看我的微博昵称“潜龙 0318”，在页面 HTML 编码中的节点，按图 2.2.2.1 中的步骤所示：

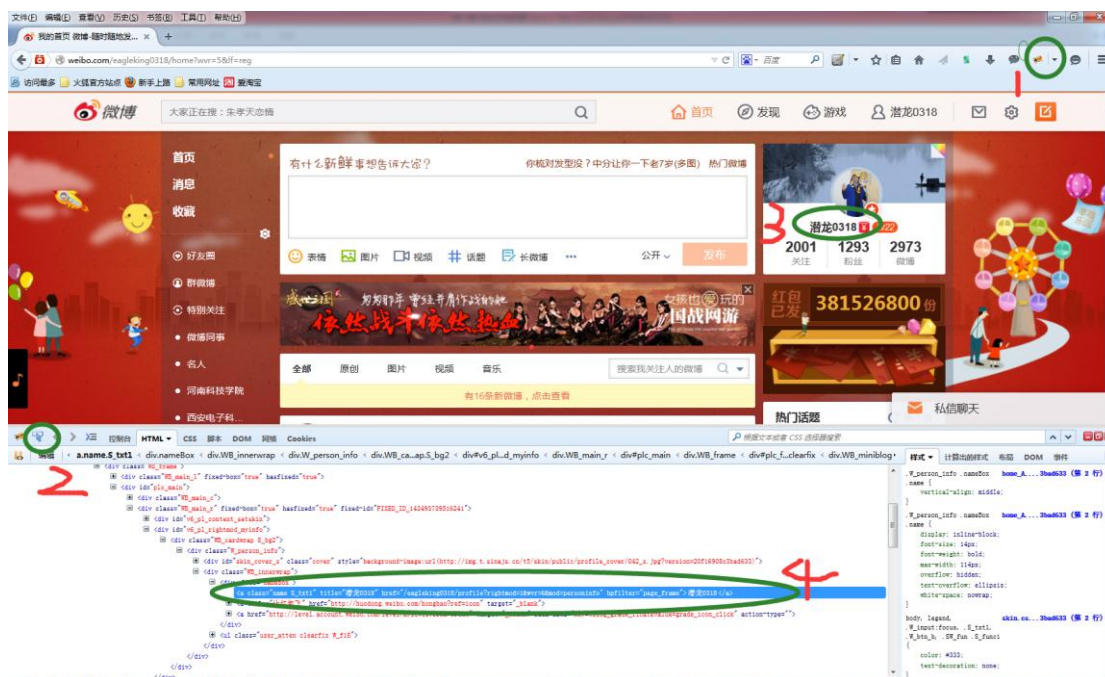


图 2.2.2.1 查看页面元素

步骤讲解：

- (a) 单击浏览器工具栏上 Firebug 启动按钮，图中 1 处，启动 Firebug。
- (b) 单击 firebug 元素查看按钮，图中 2 处，然后将鼠标移到要定位的元素上，如图中标 3 处，当昵称“潜龙 0318”被框中后，单击鼠标。

(c) 此是 Firebug HTML 标签下显示的内容被选中的内容，就是昵称对应的元素在 HTML 中的位置，找到这个位置后，我们就可以编写元素定位的方法了。

➤ 定位方法唯一性查找

当我们用 Firebug 查到一个元素的某个属性值为“feed_list”，欲用 xpath 定位，但是如果这个元素的这个属性在页面中不唯一，而且这个元素不是第一个符合条件的元素时，则会定位失败。所以定位前我们需要查看一下唯一性！

如图 2.2.2.2 所示：

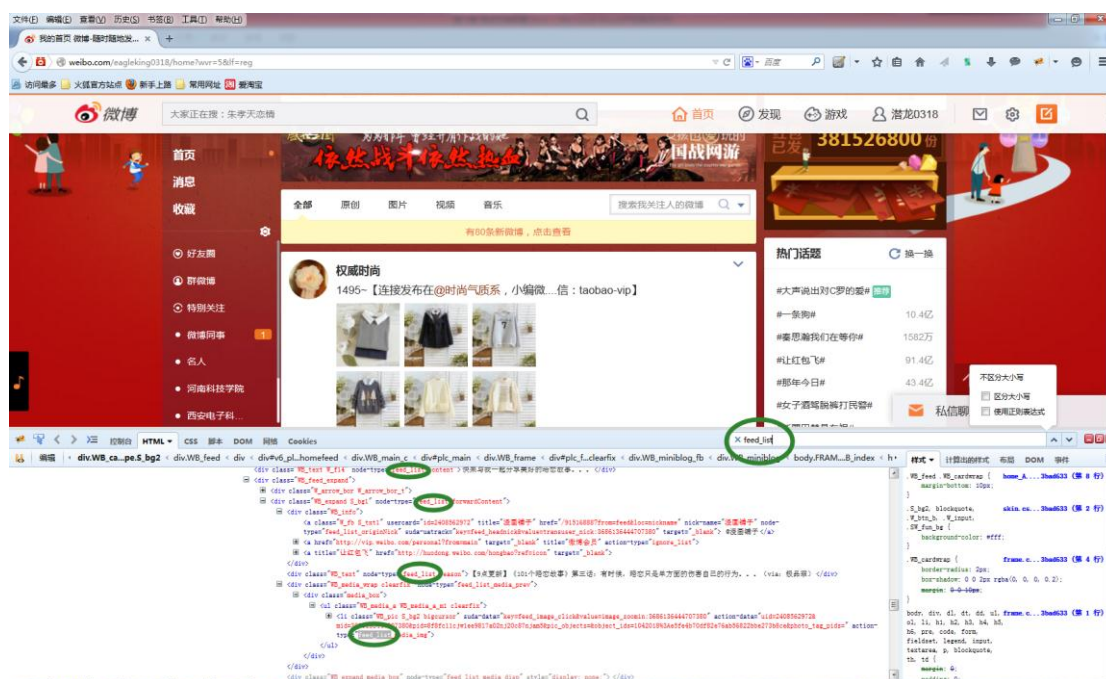


图 2.2.2.2 查找 feed_list 的结果

在 Firebug 的查找框输入“feed_list”，然后单击回车查找。此时就会发现有很多条件条件的元素，证明这个属性值不唯一，所以不能用这个属性值定位。

2.3 WebDriver Python 开发环境搭建

上面是所用到的工具的介绍，下面我们开始着手搭建 WebDriver+Python 在 windows 下的运行环境。

2.3.1 工具选择

- **操作系统：**Windows 7 64 位。
- **Python 版本：**选择 2.7.X。目前大部分第三方库和工具对 2.7 都有简单的安装包，不需要自己做太多处理，比 2.6 内置了一些包，不需要再安装； Python3.x 不支持 Selenium2.0。
- **Selenium 版本：**python 自动安装最新的包，如果手动安装不能低于 2.0，因为从 2.0 开始，Selenium 已经和 WebDriver 集成在一起了，WebDriver 提供了非常多的 API 和自动化测试处理方法。
- **脚本开发工具：**Eclipse (JDK：选择 1.6 版本)，其中插件选择：PyDev，专门对 python 进行开发。

2.3.2 Python+Webdriver 安装

请按以下步骤安装 python+Webdriver 运行环境：

第一步：安装 Python

- 根据下面的地址，直接一键安装，全部默认方式。

下载地址：

<http://www.python.org/ftp/python/2.7.2/python-2.7.2.msi>。

- 设置 Python 的环境变量:，修改我的电脑->属性->高级->环境变量->系统变量中的 PATH 为 PATH: “C:\Python27;”

上述环境变量设置成功之后，就可以在命令行直接使用 python 命令。或执行“python *.py”运行 python 脚本了。

- 此时，还是只能通过“python *.py”运行 python 脚本，若希望直接运行 *.py，只需再修改另一个环境变量 PATHEXT 为：
PATHEXT=PATHEXT;.PY;.PYM

第二步：安装 Python 的 SetupTools

其实 SetupTools 就是一个帮助你安装第三方工具包的增强工具软件，根据下面的地址下载，然后一键安装。下载地址：

<http://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11.win32-py2.7.exe#md5=57ele64f6b7c7f1d2eddfc9746bbaf20>

第三步：安装 Python 的包管理工具

pip 有点类似 SetupTools，打开 DOS 界面，进入到目录：C:\Python27\Scripts，然后敲入命令：easy_install pip，等待完成就 OK。如图 2.3.2.1 所示：

```

C:\Python27\Scripts>easy_install pip
Searching for pip
Reading http://pypi.python.org/simple/pip/
Reading http://pip.openplans.org
Reading http://www.pip-installer.org
Best match: pip 1.1
Downloading http://pypi.python.org/packages/source/p/pip/pip-1.1.tar.gz#md5=62a9f08dd5dc69d76734568a6c040508
Processing pip-1.1.tar.gz
Running pip-1.1\setup.py -q bdist_egg --dist-dir c:\docume~1\admini~1\locals~1\temp\easy_install-rrs8ls\pip-1.1\egg-dist-tap-ukxjfp
warning: no files found matching '*.html' under directory 'docs'
warning: no previously-included files matching '*.txt' found under directory 'docs\build'
no previously-included directories found matching 'docs\build\sources'
Adding pip 1.1 to easy-install.pth file
Installing pip-script.py script to C:\Python27\Scripts
Installing pip.exe script to C:\Python27\Scripts
Installing pip.exe.manifest script to C:\Python27\Scripts
Installing pip-2.7-script.py script to C:\Python27\Scripts
Installing pip-2.7.exe script to C:\Python27\Scripts
Installing pip-2.7.exe.manifest script to C:\Python27\Scripts

Installed c:\python27\lib\site-packages\pip-1.1-py2.7.egg
Processing dependencies for pip
Finished processing dependencies for pip

C:\Python27\Scripts>

```

图 2.3.2.1 pip 安装过程

第四步：安装基于 Python 的 Selenium 包

打开 DOS 界面，进入到目录： C:\Python27\Scripts，然后敲入命令： pip install selenium，回车后就会自动下载最新的 selenium 包，并进行安装。

第五步：验证 Selenium 安装是否成功

在记事本中编写下面的代码：（保存为 pytest.py，然后直接运行即可！）

```

#pytest.py
from selenium import webdriver
browser=webdriver.Firefox()#获取本地火狐浏览器
browser.get("http://www.yahoo.com")#打开雅虎首页
assert "Yahoo!" in browser.title

browser.close()

```

将上面代码保存，然后在命令行下找到此文件，python pytest.py 运行。如果能成功打开火狐浏览器，并打开了雅虎首页，则说明 Selenium 安装成功。

2.3.3 Eclipse python 开发环境配置

Eclipse 是强大的开发工具，所以我们也采用这个工具编写我们的自动化测试用例。其插件 `pydev`，更能方便地编辑和运行 `python` 脚本文件。下面我们就开始配置这个开发环境：

第一步：安装 JDK6、Eclipse

注：JDK 和 Eclipse 都要用 64 位，否则有可能遇到问题。

JDK 需要配置环境变量（详细见

<http://jingyan.baidu.com/article/6dad5075d1dc40a123e36ea3.html>）

下载 JDK：<http://download.csdn.net/download/xiaoxiaoxinyuan8/5796753>

eclipse 下载：

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR1/eclipse-java-indigo-SR1-win32-x86_64.zip

当然你也可以自行在网上搜索下载，JKD 最好用 6，不过好像 8 也支持。可能尝试用最新版本，如果有问题，就降下来，毕竟低版本用的时间长，比较稳定，但也存在不包括最新的功能的风险。

第二步：给 Eclipse 安装 PyDev 插件

启动 Eclipse，在 Help 菜单中，选择 Install New Software；

选择 Add 按钮，Name：PyDev，Location：<http://pydev.org/updates>

（PyDev 的更新地址），单击 OK，开始查询。如图 2.3.3.1 所示：

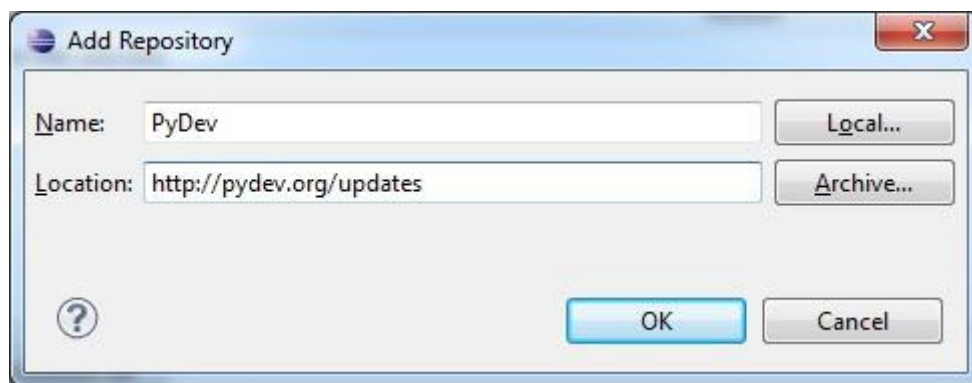


图 2.3.3.1 安装 Pydev 插件

然后在打开的对话框中选择 PyDev 下的 PyDev for Eclipse，别的都不要选，否则依赖检查那关过不去。如图 2.3.3.2 所示：

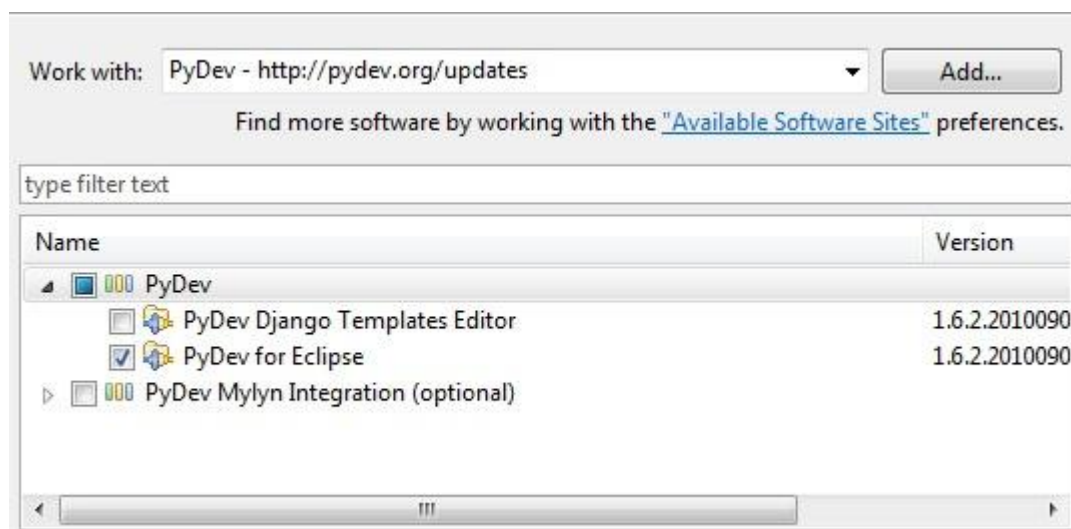


图 2.3.3.2 pydev 安装项选择

点下一步安装，中间会出现是否信任 Aptana、Pydev，选择信任即可，最后重启 Eclipse（如果第二步不能成功，尝试从官网下载 Pydev2.8.1 版本，直接覆盖到纯净版 eclipse 插件文件夹下，重启 eclipse 即可）。

另外需要下载 MySQL-python.rar 插件，直接安装，重启 eclipse。不同的环境和版本可能会出现不同的问题，大家需要去网上查询相关的解决办法，在此就不一一穷尽。

第三步：配置 PyDev 插件

在 Window - Preference - PyDev - Interpreter-Python，单击 New... 按钮，在弹出的 Select interpreter 窗口中单击 Browse... 按钮，找到已经安装的 Python 解释器。如图 2.3.3.3 所示：

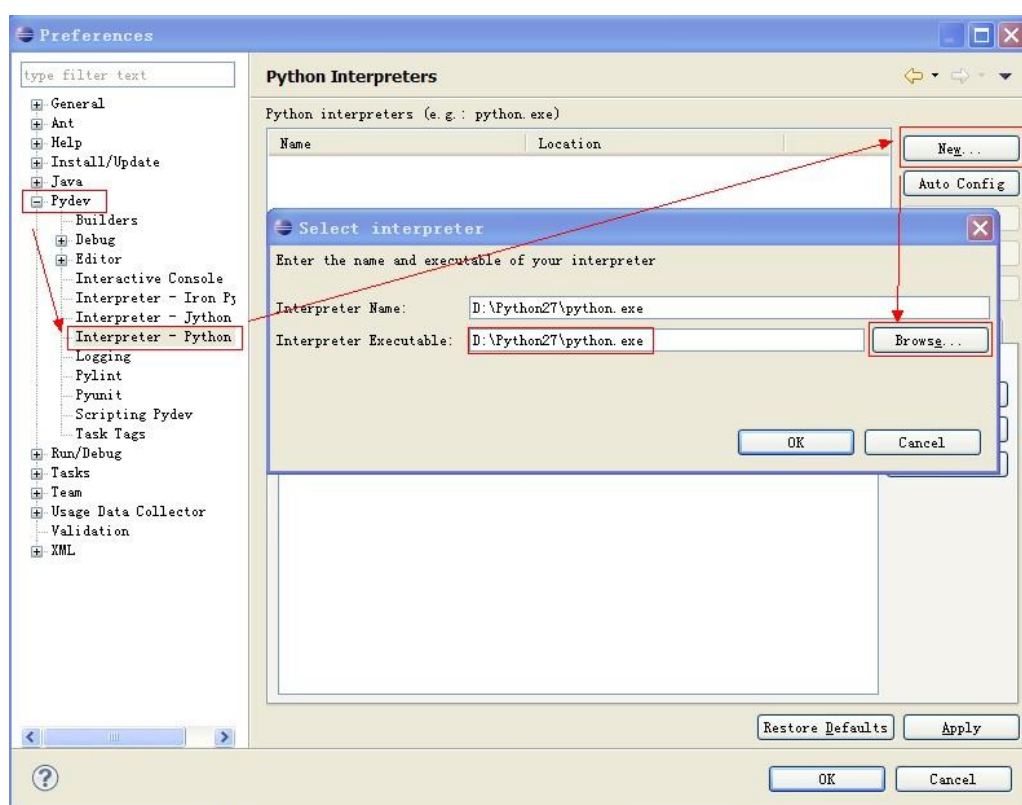


图 2.3.3.3 选择 python 解释器

单击 Select interpreter 窗口中单击 OK 按钮，在弹出的 Selection Needed 窗口中单击 Select All 按钮，然后单击 OK 按钮完成设置。如图 2.3.3.4 所示：

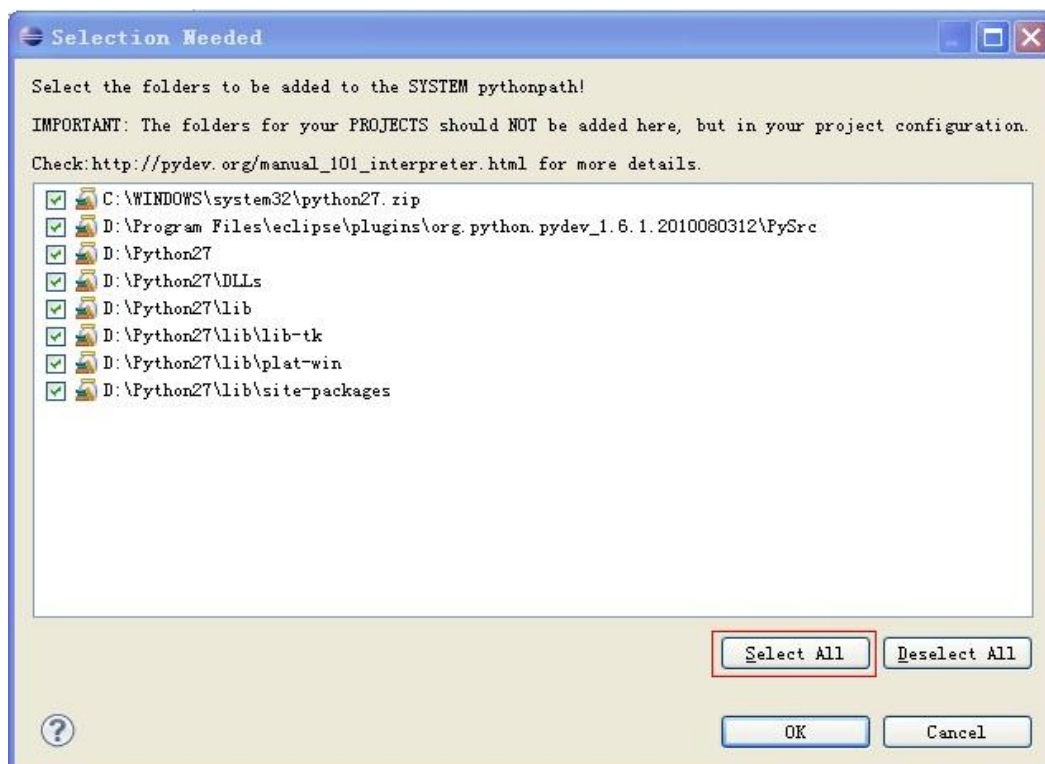


图 2.3.3.4 选择所有相关内容

然后在 Preferences 窗口中选择 Apply→OK 完成设置。如图 2.3.3.5 所示：

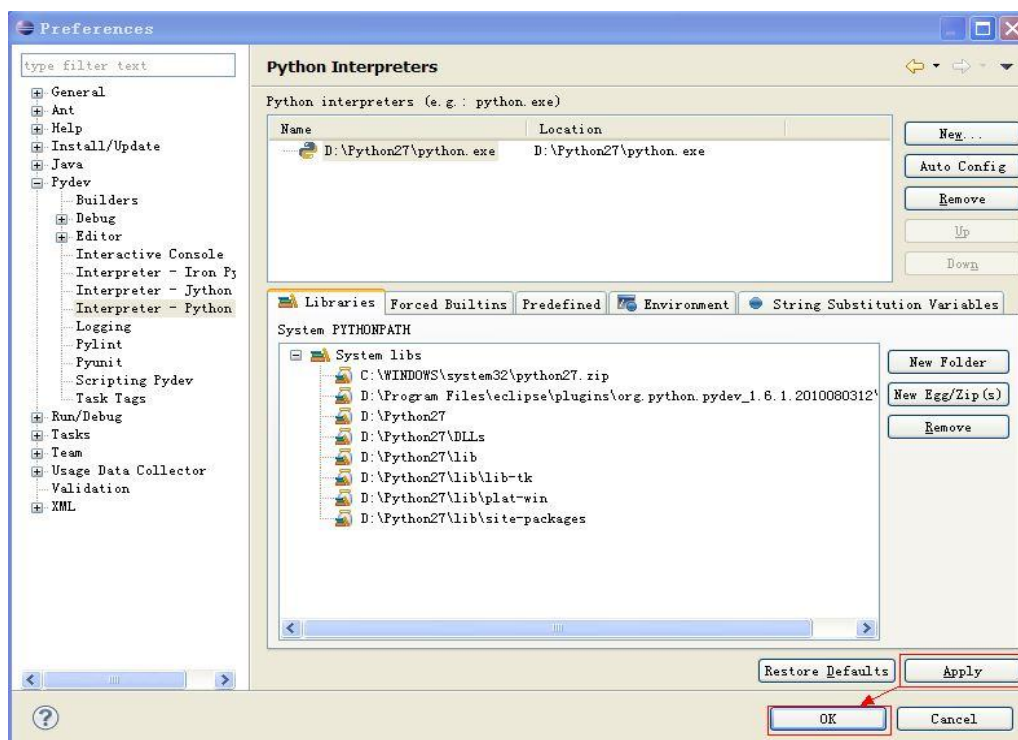


图 2.3.3.5 应用所有配置

第四步：配置 Python 编码格式

统一编码格式，把 PyDev 的编码格式修改成 UTF-8，默认采用 GBK 编码。取消 u' ...' 形式的 unicode 文本表示，保存文本的数据类型是 str，保存数据的数据类型是 bytes。由于默认采用 utf-8 编码，只要保持.py 文件的字符编码也为 utf-8 格式，不用再在头部声明程序的编码类型，即不用写# -*- coding: utf-8 -*-；采用如下方法：

- 修改 PyDev 编码格式，在 Window-->Preference→General->Editors->Text Editors->Spelling, Encoding 改成 Other:UTF-8。
- 修改 Workspace 编码格式 General->Workspace, Text file encoding 改成 Other:UTF-8。
- 修改 python 编码格式：找到安装目录下的 \plugins\org.python.pydev.debug_x.x.x.yyyymmddhh\pysrc\pydevd.py, 920 行的 encoding=Non 改成 encoding="UTF-8", 保存（有的环境可能没有这个文件，如果找不到就忽略）。

至此，PyDev 的配置就完成了。

注：如果没有配置编码规范，运行 python 脚本时会报错。

第五步：测试安装是否成功

File->New->Project, 选 PyDev 下的 PyDev Project, Grammer 和 Interpreter 选相应的版本（如 2.7），单击 Finish。如图 2.3.3.6 所示：

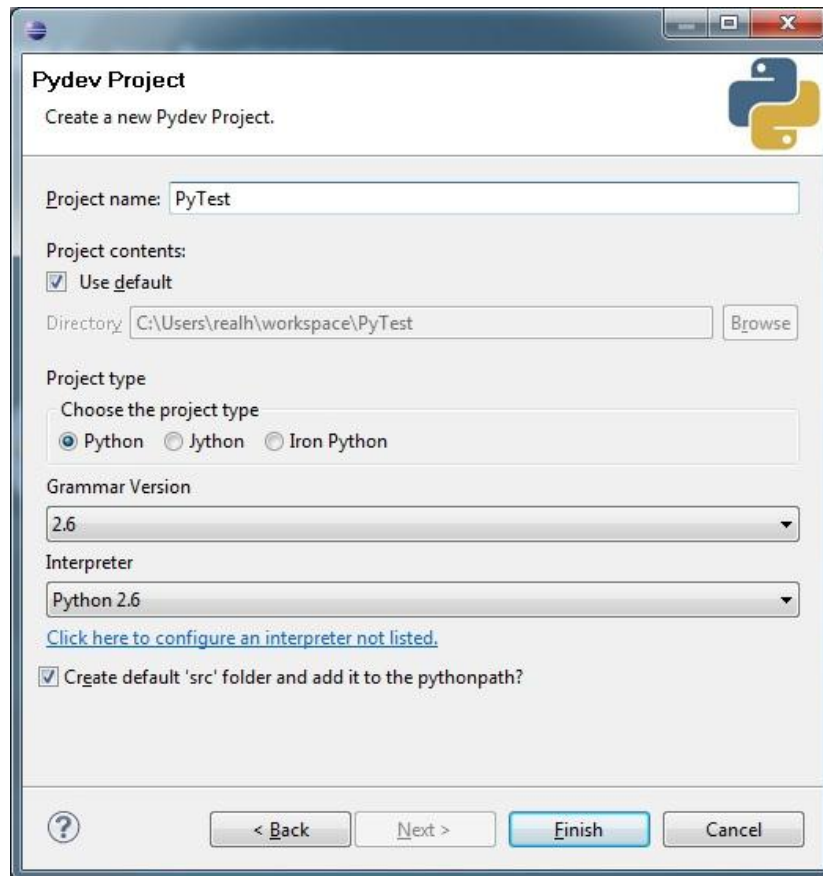


图 2.3.3.6 创建 pydev 工程

在 PyDev Package Explorer 的项目上右键，New->PyDev Module，随便写个名字，Finish。然后随便写几行代码，Run 在弹出的对话框中选择 Python Run，如果运行成功，则说明 Pydev 环境配置没有问题。如图 2.3.3.7 所示：

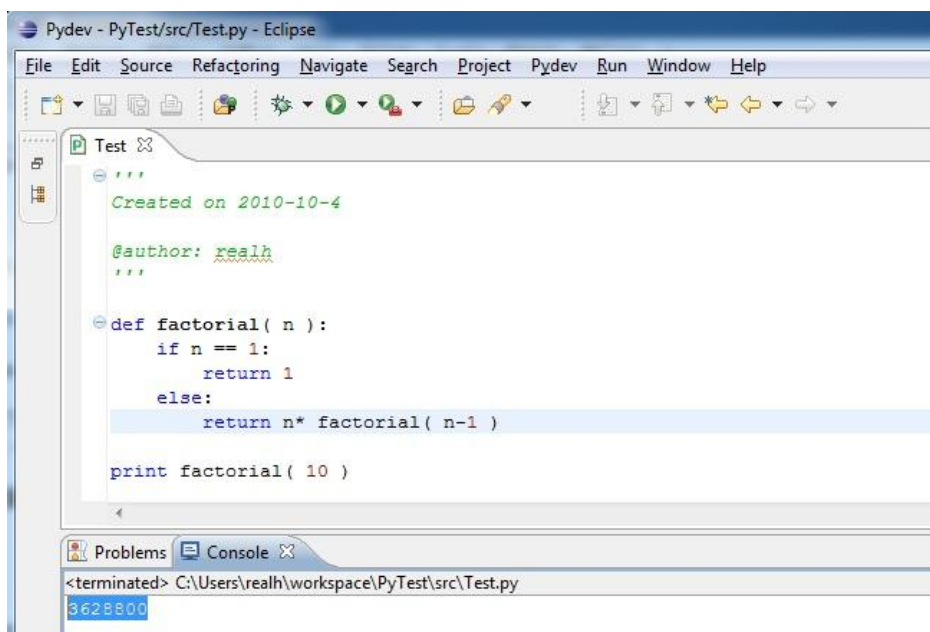


图 2. 3. 3. 7 检测 Pydev 环境

2.4 本章小结

经过本章的学习，我们可以完成 Windows 环境下 Webdriver+Python 开发环境的配置。当配置完成后，你也可以把先前我们用记事本编辑的测试用例拿到 Eclipse 中测试运行一下，结果和前面完全相同。在 Linux 和 MAC 环境下配置类似，只是安装方法有所不同，找到相应环境的插件及工具，自行安全即可。再用本章中提到的例子，去验证一下，如果没有问题，说明环境配置成功。

有了好的开发环境，我们还需要努力学习，才能真正地掌握如何在这个环境下编写自动化测试用例。从下面的章节，我们将开始讲解自动化测试用例的编写知识。

第三章 Webdriver API 简介

Selenium 2.0 主要的特性就是与 WebDriver API 的集成。WebDriver 旨在提供一个更简单，更简洁的编程接口以及解决一些 Selenium-RC API 的限制。Selenium-Webdriver 更好的支持页面本身不重新加载而页面的元素改变的动态网页。WebDriver 的目标是提供一个良好设计的面向对象的 API，提供了对于现代先进 web 应用程序测试问题的改进支持。

3.1 Webdriver API

Web 应用程序的测试主要是基于调用 Webdriver API 来模拟用户操作，然后判断操作结果是否与预期的一致，从而达到自动化测试的目的。所以熟悉 Webdriver 的 API 的使用很重要，也是我们在做自动化测试的先决条件。

Webdriver API 网上有官方文档，不过由于种种原因吧，官网不太容易打开。所以我们可以百度中去搜索相关的文档，结果也非常多。不同语言的 API 有点儿不太一样，由于我们采用的是 Python 作为脚本语言，所以建议看出 Python 版的，推荐以下两个网页：

(1) selenium_webdriver(python)第一版：

<http://wenku.baidu.com/link?url=PDcKQYNL-iVRlahMunWoY1BDMw5vyUvv-AFCtC6eUCfG0R5XdC0SnBCHdp742uY6riA25FdBfaUtl-N2uZiXj6PXxiyRGcl-bV1QEYSZH>

(2) 虫师的翻译:

<http://www.cnblogs.com/fnng/archive/2013/06/16/3138283.html>

这两篇文档刚好是一个人的，我看了一下写的比较详细，大家就去自行学习一下吧。

关于 Webdriver API 的一点儿说明:

- API 只是一些儿封闭的方法，大致浏览一下，知道有哪儿些 API，完成什么操作即可，没有必要花很多时间去学习，边用边学。
- API 的学习要灵活，最好把同类的操作放到一起比较一下。因为在编写测试用例的过程中，不仅仅只有一种方法可以达到预期的结果。
- 要会使用 Eclipse 的联想功能。在写测试用例的时候，如果一时想不起来用什么方法了，可以利用联想功能进行查询。

本章我们就不详细讲解这些儿 API 的使用方法了，重点将放到页面元素的定位及检查点的设置，这些儿是体现一个自动化测试工程师水平的重点。

3.2 页面元素定位

自动化测试是模拟用户对页面元素进行操作的，所以在操作之前，需要先定位到要操作的页面元素。如果页面元素都定位不到，其他的操作将无从谈起。而对页面元素定位技巧，随着经验的增加，将会越来越精准。而在此，我们将从基础谈起，然后再逐步加深！

3.2.1 WebElement 对象提供的各种定位元素策略

下面我们先来讲解一下 WebElement 对象提供的各种常用的定位元素策略：

➤ 通过 ID 定位元素：

ID: `driver.find_element_by_id(<elementID>)`

示例：当一个页面元素如下，明显包含 id 属性，而且属性值是固定的时候，可以使用这个定位方法。

```
<div id="nav" class="m-subnav">.....</div>
```

Ex: driver.find_element_by_id("nav")

➤ 通过 Name 定位元素：

Name: `driver.find_element_by_name(<elementName>)`

示例：当一个页面元素如下，明显包含 name 属性，而且属性值是固定的时候，可以使用这个定位方法。

```
<input type="text" wx-validator-placeholder="用户名/手机号/邮箱" wx-validator-username-required="*请输入账号" wx-validator-rule="required" name="username" class="foc" placeholder="用户名/手机号/邮箱">
```

Ex: driver.find_element_by_name("username")

➤ 通过 ClassName 定位元素：

className: `driver.find_element_by_class_name(<elementClassName>)`

示例：当一个页面元素如下，明显包含 name 属性，而且属性值是固定的时候，可以使用这个定位方法。

```
<div class="bx-wrapper" style="max-width: 100%;">.....</div>
```

Ex: driver.find_element_by_class_name("bx-wrapper")

➤ 通过 TagName 定位元素：

tagName: `driver.find_element_by_tag_name (<htmlTagName>)`

示例：当一个页面元素如下，这是一个 form 元素，如果本页中只有一个 form，可以使用这个定位方法。

```
<form wx-validator="" method="get" action="/deals" autocomplete="off">
  <span>
```

```

        <input type="text" wx-validator-notip="" wx-validator-rule="required" wx-validator-plac
eholder="输入关键词" name="k" placeholder="输入关键词">
        <a class="bh-mhnbo" type="submit" href="javascript:;"></a>
    </span>
</form>

```

Ex: driver.find_element_by_tag_name("form")

➤ 通过 LinkText 定位元素:

linkText: driver.find_element_by_link_text (<linkText>)

示例: 当一个页面元素如下, 明显是一个超级链接, 可以使用这个定位方法。

```

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

```

Ex: driver.find_element_by_link_text("cheese")

➤ 通过 PartialLinkText 定位元素:

partialLinkText: driver.find_element_by_partial_link_text (<partialLinkText>)

示例: 当一个页面元素如下, 明显是一个超级链接, 但是超级链接显示的文字比较长的时候, 可以使用这个定位方法, 通过部分文字定位。

```

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

```

Ex: driver.find_element_by_link_text("cheese")

➤ 通过 CSS 定位元素:

css: driver.find_element_by_css_selector (<cssSelector>)

示例: 当一个页面元素如下, 这个可以使用 CSS 定位。

```

<div class="m-right clearfix">
    <span class="z-Login">
        .....
    </span>
</div>

```

Ex: driver.find_element_by_css_selector ("#div.m-right.clearfix span.z-Login")

➤ 通过 Xpath 定位元素:

xpath: driver.find_element_by_xpath (<xpathQuery>)

示例: 当一个页面元素如下, div 中嵌套 h3, h3 中嵌套 a 标签, 现在我们要定位这个 a 标签的元素。现在我们采用 xpath 定位方法:

```

<div class="lev_Box lev_Box_noborder">
    <h3 class="lev">

```

```
<a class="S_txt1" suda-uatrack="key=V6update_leftnavigate&value=homepage" nm="status" bpfiler="main" node-type="item" href="/eagleking0318/home?leftnav=1">
</h3>
</div>
```

Ex: driver.find_element_by_xpath (“//div[@class=’ lev_Box lev_Box_noborder’]/h3/a”)

以上是 WebElement 对象提供的定位方法，这八种方法是最基本的。不过大家要有这样的一个共识，元素定位不只是有一种方法能定位到。一个页面元素可以通过很多种方法来定位，要选择一个比较恰当的方法，这就需要一些儿技巧和经验了。

3.2.2 定位方法的选择

页面元素的定位，是页面自动化测试过程中的首要任务及重中之重。如果连元素都定位不到，再好的测试框架，高超的编程技巧，也无法完成自动化测试用例的编写。所以本节我们就从最基本的开始，当我们欲定位一个页面元素的时候，应该如何选择定位方法。

为了简单期间，我们就以百度为例，来讲解页面元素定位方法选择的思路。现有一个测试用例如下：

测试步骤：

- (1) 打开百度首页，输入“自动化测试”。
- (2) 百度一下，检测搜索结果

分析：

(1) 打开百度没有任何问题，直接调用 WebDriver API 就行了。可是要输入“自动化测试”，我们首先要定位到输入框架，然后再能输

入要查询的关键词。

(2) 百度一下，要定位到“百度一下”按钮，才能执行单击操作。
检测搜索结果的时候，需要在查询结果页定位查询到的网页标题或是内容，才能检测。

好了，目前我们明白自动化要做的内容了，就可以着手去编写自动化测试用例。至于如何编写是以后章节的内容，本节我们主要关注元素定位的部分。现在我们按如下步骤进行定位：

第一步：用火狐打开百度首页。

我们之所以有火狐打开百度，是想利用在第二章我们讲到的火狐的插件来定位元素。

第二步，用 Firebug 查找定位的元素。

Firebug 想必大家已经会使用了，我们打开 firebug,利用“点击查看页面中的元素”按钮，点击输入框，则 firebug 会以选中状态显示输入框在 HTML 中的标签。如图 3.2.2.1 所示：

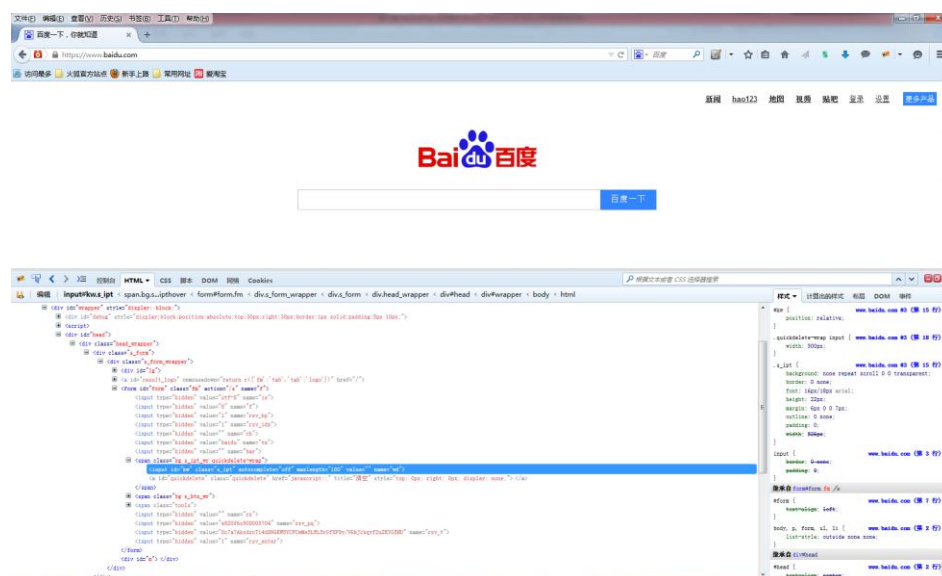


图 3.2.2.1 选中状态显示输入框

第三步，分析选择定位方法。

我们先分析一下这个标签的特点：

```
<input autocomplete="off" maxlength="100" value="" class="s_ipt" name="wd" id="kw">
```

- 这是一个 input 标签，所以说 Tag Name 是 input，通过我们查看网页源码，发现不是只有一个这样的标签，所以不能用 find_element_by_tag_name 来定位。
- 分析标签的属性，发现标签中具有我们特别感兴趣的属性 class,name,id，这三个属性可以用来定位，而 autocomplete,maxlength,value 经过分析是不可用的，所以舍弃。
- 假如标签不存上面的三个属性，我们就考虑一下能否用 Xpath 和 CSS。

方法：从此标签向上查找，遇到一个它的上级标签，就去找有没有唯一的属性，如果有，从此层往下写 Xpath 或 Css，如果没有，接着向上层查找。

本标签的上一级标签是 span,span 有 Class 属性，但其值中有空格，不能使用 Xpath(原因见下节)，但可以使用 Css，Css 的定位方法是：input#kw.s_ipt。

- Xpath 定位方法，我们接着上面继续向上查找，再上一级是 form 标签，这个标签有 id,name,class 属性，比较适合用来定位，所以从此层开始写 Xpath 是：//form[@id='form']/span/input 或 //form[@name='f']/span/input 或 //form[@class='fm']/span/input。
- 这个标签不是超级链接，所以不能用 link 相关的定位方法。

第四步，编写定位代码

针对该标签，我们现在把能用的定位方法全部写下来：

- id: driver.find_element_by_id("kw")
- name: driver.find_element_by_name("wd")
- class: driver.find_element_by_class_name("s_ipt")
- css: driver.find_element_by_css_selector("input#kw.s_ipt")
- xpath: driver.find_element_by_xpath("//input[@id='kw']")

由此可见一个页面元素的定位方法是相当多的，我们可是根据需要选择。然后再放到代码中去调试，如果不行，就换另外的办法。**原则就是越简单的定位方法越好，因为这样的定位方法受到网站改版的影响也越小。**

第五步，工具使用定位

经过上面四步我们已经可以定位页面上的元素了，不过这都是通过我们人工来查看的，然后手动编写的代码。有没有更加简单的方法来定位呢？答案是 Yes，就是我们前面提到的 FireBug 和 Selenium IDE。下面我们简单地介绍一下使用方法：

(1) Firebug 提取元素的 Xpath，Css 路径。

元素的 id,name,class 属性一目了然，直接可以使用，如果元素没有这几个属性，就需要用 Xpath 和 Css 路径定位了。但是这两个路径不太容易写出来，所以 Firebug 提供了方法。

首先，我们用 Firebug 找到要定位的元素。然后右击这个元素在 Firebug 中的位置，从弹出的菜单中选择“复制 Xpath”，“复制最简

Xpath”或是“复制 CSS 路径”。最后将复制到的内容粘贴出来，这就是对应页面元素的 xpath 或是 Css 路径。如图 3.2.2.2 所示：



图 3.2.2.2 提取页面元素的 CSS or Xpath 路径

(2) Selenium IDE 验证提取的路径是否正确。

通过 Firebug 我们可以提取出元素的 Css 或者 Xpath 的路径，可是提取的究竟对不对呢？工具有的时候也不太靠谱，所以我们要验证一下。最直接的办法就是放到测试用例中去执行一下，但是一直在执行测试用例，这样比较耗时。我们可以借助于 Selenium IDE 来验证一下。

验证方法如下：

- 用 FireBug 提取出要定位的元素的 Xpath 或者 Css 路径。
- 打开 Selenium IDE 界面，右击界面 “Insert new command”。
- 将复制的路径粘贴到 IDE 的 Target 文本框中，如果复制的是 Css 路径，需要在复制的路径前加上 “Css=”。
- 单击 Find 按钮，此时 Firebug 中显示的要定位的元素会标黄显示，表示定位正确。如图 3.2.2.3 所示：

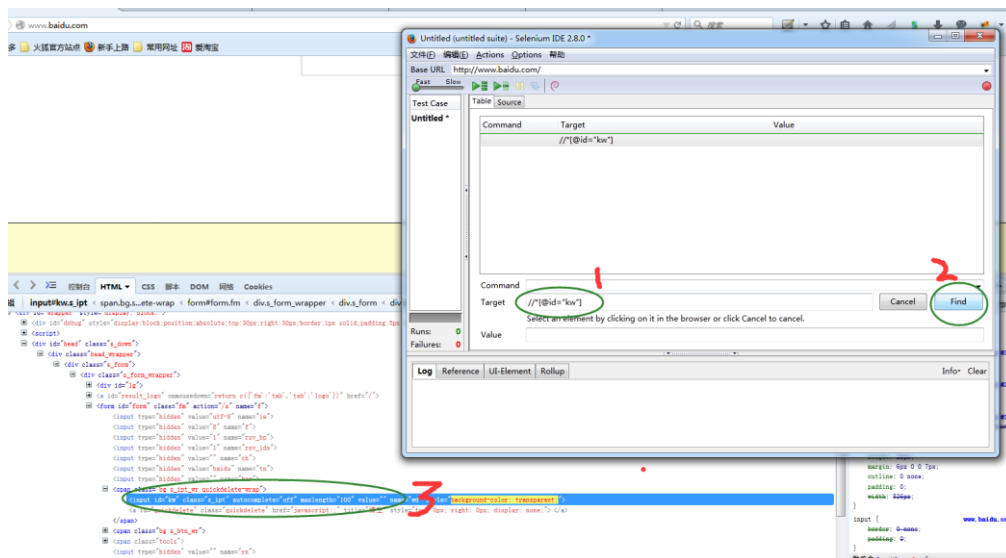


图 3.2.2.3 验证定位元素正确的情况

- 如果复制的路径不对，则 IDE 会在 log 区以红色的信息提示定位不到。如图 3.2.2.4 所示：

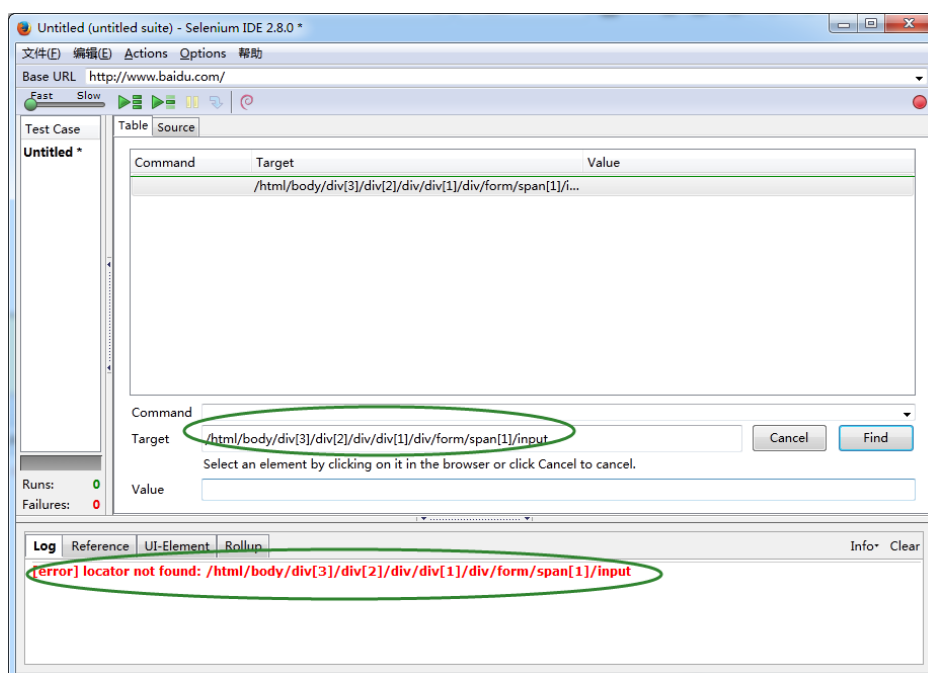


图 3.2.2.4 验证定位元素定位不到的情况

注意：

Firebug 提交的路径一般都是从页面起始位置<html>标签开始，一直提取到要定位的元素，是相对路径，容易受到网页变化的影响。

不建议直接使用，只可以作为参考。

Selenium IDE 验证定位方法，不仅仅可以验证 Firebug 复制的路径，还可以验证我们自己编写的路径，比用代码验证方便快捷。这种验证方法是我的最爱，节省了不省时间。

3.2.3 Xpath 定位方法深入探讨

相比 `cssSelector`，`xpath` 是我比较常用的一种定位元素的方式，因为它很方便，缺点是，消耗系统性能。如果 `Xpath` 使用的比较好，几乎可以定位到任何页面元素，而且受页面变化影响较小。

(1) 常用的 Xpath 定位方法及其特点

➤ 使用绝对路径定位元素。

例如：`driver.find_element_by_xpath ("/html/body/div/form/input")`。

特点：这个路径是从网页起始标签`<html>`开始一直到要定位的元素的路径，如果要定位的元素在页面最下面，则这个 `Xpath` 路径会非常长。如果在要定位的元素与页面开始之间的元素有任何增减，元素定位就会失败。

➤ 使用相对路径定位元素。

例如：`driver.find_element_by_xpath ("//input")` 返回查找到的第一个符合条件的元素。

特点：相对路径一般只会包含与被定位元素最近的几层元素有关，相对路径写的好的话，页面变动影响最小，而且定位准确。

➤ 使用索引定位元素，索引的初始值为 1，注意与数组等区分开。

例如：`driver.find_element_by_xpath ("//input[2]")` 返回查找到的第二个符合条件的元素。

特点：如果一个页面中有多个相似的元素，或是一个层下面有多个同样的元素的时候，需要用索引的方法来定位，否则无法区分。

➤ 结合属性值来定位元素。

例如：`driver.find_element_by_xpath ("//input[@id='username']");`
`driver.find_element_by_xpath ("//img[@alt='flowr']");`

特点：属性定位也是比较常用的方法，如果元素中没有常见的 `id,name,class` 等直接有方法可调用的属性，也可以查找元素中是否有其他能唯一标识元素的属性，如果有，就可以用此方法定位。

➤ 使用逻辑运算符，结合属性值定位元素, **and** 与 **or**。

例如：`driver.find_element_by_xpath ("//input[@id='username' and @name='userID']");`

特点：多个属性值联合定位，更能准确定位到元素。并且如果多个相同标签的元素，如果其包含的属性值有不同的，也可以用这个方法区分开来。

➤ 使用属性名来定位元素。

例如：`driver.find_element_by_xpath ("//input[@button]")`

特点：此方法可以区分同一种标签，含有不同属性名的元素。定位相对简单一些儿，但也同样存在着无法区分同种标签含有同种属性名的多个元素，这个时候要配合索引定位才行。

➤ 类似于 **cssSlector**，使用部分属性值匹配元素。

例如：

(a) starts-with()

```
driver.find_element_by_xpath ("//input[starts-with(@id,'user')]")
```

(b) ends-with()

```
driver.find_element_by_xpath ("//input[ends-with(@id,'name')]")
```

(c) contains()

例如： `driver.find_element_by_xpath`

```
("//input[contains(@id,"ernam")]")
```

特点：此方法更加灵活，可以定位属性值不太规律，或是部分变动，中间有空格的情况。注：如果属性值中间包含空格，Webdriver 定位的时候容易出错，时而能定位到时而定位不到，所以应该避免用含用空格的属性值定位。可以采用此方法，进行部分属性值定位。

➤ 使用任意属性值匹配元素。

例如： `driver.find_element_by_xpath ("//input[@*='username']")`

特点：此方法相当于模糊查询，只要欲定位的标签，如 input 中任何属性值等于 ‘username’ ,就能匹配成功。缺点，可能会匹配含有这个属性值的其他元素，所以我们在定位的时候要查看一下这个元素值在页面中是否唯一。

(2) 运用 Xpath 定位元素的思路

当我们在做自动化测试的时候，欲对一个页面元素定位，通过上面我们讲到的选择定位方法筛选后，决定用 Xpath 定位了，此时我们应该怎么写 Xpath 呢？请按以下步骤来分析：

(a) 先看一个这个元素是否有明显的，唯一的属性值。如果有，我们就用相对路径加属性值定位，这是最简单准确的定位方法。如：`://input[@alog-alias='search']`。

(b) 如果要定位的元素，不符合上面的特征，元素属性要么是动态的，要么就是不能区分这个元素的，还有就是属性值中间有空格的情况，都无法定位。所以从此元素开始，向他的上一层查找。

(c) 当遇到了一个符合条件的元素时，对其写 Xpath，然后在 Selenium IDE 中验证是否能定位到该元素。如：`://div[@type='good']`，在 Selenium IDE 中验证能定位到这个 div。

(d) 然后从这个元素开始，一级级往下写，直到要定位的元素为止。如果你比较肯定写的是正确的，可以写完后验证，如果不肯定，就写一层，用 Selenium IDE 验证一下，以确保安全。如：
`://div[@type='good']/div/input`

(e) 当 Selenium IDE 定位成功后，再放到测试用例中去调试运行。虽然 Selenium IDE 能定位到的代码也能定位到，不过还有因为延迟，操作顺序等会影响代码定位的因素存在。

3.2.4 元素定位不到的原因及解决办法

在我们编写自动化测试用例的过程中，经常会遇到元素定位不到的现象，有的时候我们用 Selenium IDE 检查的时候也能在 Firebug 中看到，可是运行代码的时候，总是提示元素找不到。经过我以往和经验和大家在网上的讨论，我总结了以下几种情况：

(1) 定位属性值是动态变化的情况

现象：在我们定位元素的时候，发现有 `id`, `name` 或其他的属性存在，于是就用相应的定位方法去定位。可是运行的时候提示定位不到，然后我们再去查看元素的时候，发现属性值和我们写代码的时候不一样了。

原因：通常产生这种情况的原因就是你使用的属性值是动态变化的，主要表现有属性值是一串数据，或是字符加一串数据等情况。页面加载一次变化一次，每次都不相同。

解决办法：我们应尽量避免用这样的属性值去定位，而采用这个元素下的其他固定不变的属性值。或是向上层查找，采用 `xpath` 定位。

(2) `Iframe` 中的元素定位出错的情况

现象：我们在定位元素的时候，查看网页源码，发现有 `iframe` 存在。可是我们没有做特殊处理，而是直接用通用的定位方法，`name`, `id`, `xpath` 或者 `CSS` 来定位。用 `Selenium IDE` 验证能查找到元素，可是运行测试用例的时候，总是元素找不到。

原因：在我们运行测试脚本的时候，代码获取的是页面的句柄，而 `iframe` 在句柄中是当成一个元素来处理的。脚本是没有办法自己去 `iframe` 中去定位元素的，所以当搜索完页面时，发现找不到要定位的元素，就当错误处理。

解决办法：当需要定位 `iframe` 中的元素的时候，先将句柄切换到 `iframe` 中（`driver.switchTo().frame("framename");`），然后再去定位，就

能定位到要测试的元素。

（3）不同页面或 iframe 切换时元素定位情况

现象：当我们在编写测试用例的时候，会遇到打开一个新页面，或是切换到一个新的 iframe 中，然后再去定位元素进行操作。但是我们的定位方法写的没有问题，而且在 Selenium IDE 中也验证通过，可是代码运行的时候还是会提示找不到元素。

原因：其实这个和定位 iframe 中元素的情况是一样的，在打开一个页面或是切换到一个 iframe 的时候，driver 获取的是当前页面或是 iframe 的句柄。当你的操作切换到新的页面或是 iframe 的时候，如果代码不去做相应的切换，查找元素的时候还会在原来的句柄下查找，当然会出现查找不到的情况。

解决办法：当操作切换页面或是 iframe 的时候，我们的测试脚本也要做相应的切换，选择新打开的页面或是切换到新的 iframe 下。然后再去定位的时候，就会在新页面或是 iframe 下定位了。

（4）Xpath 编写出错的情况

现象：如果我们对一个元素编写了对应的 Xpath，然后在没有通过 Selenium IDE 进行验证的情况吧，就去编写代码执行测试用例。会出现查找不到元素的情况，或是页面发生了变化，导致 Xpath 路径有了变化，也会查找不到元素。

原因：主要的问题就是 Xpath 编写出错了，或是页面有改动。不管是增加了新的模块或是隐藏的 div，都会影响 Xpath 路径的。

解决办法：将代码中的 Xpath 拷出来，放到 Selenium IDE 中进行

验证。如果出错了，就做相应的修改。这个也是代码维护中当遇到的问题，被测试对象变化，导致测试用例的修改。

（5）操作速度过快，被定位的元素没有加载出来的情况

现象：在测试用例运行过程中，会出现被定位的元素有的时候能定位的到，有的时候却定位不到的现象。而我们去页面上验证我们的定位方法的时候，没有一点儿问题，显示不是定位方法写错了。

原因：这种情况多半是因为测试用例执行到代码的时候，被定位元素没有加载出来造成的。网速原因，执行代码的机器原因，都会造成加载比程序执行的慢的情况。

解决办法：在我们定位元素之前，评估一下页面的加载情况，如果有加载慢的地方，需要添加一定等待时间 `self.sleep(5000)`,等上几秒后再去定位操作。

（6）定位页面嵌入式元素的情况

现象：在页面中会有一些儿嵌入式元素，如 `object`,播放器等。这个时候，我们对其操作的时候，是无法定位到上面的元素的。

原因：嵌入式元素对 `webdriver` 来说是一个元素，不管里面包含多少元素，都无法操作。对于 `object` 对象，网上有说要对相应的 `Flash` 重新编译，添加相应的代码或是控件才能定位。但这样一样又不安全了，所以嵌入式对象一直是自动化测试的盲区。

解决办法：嵌入式对象如果是简单的单击操作，可是用模拟鼠标单击相应的区域，就能完成操作。如果是输入操作，我们可以先模拟点击输入区，然后模拟键盘进行输入。除此之外，好像也没有什么好

的办法。

(7) firefox 安全性报错的情况

现象: firefox 安全性强, 不允许跨域调用出现报错, 错误描述: `uncaught exception: [Exception... "Component returned failure code: 0x80004005 (NS_ERROR_FAILURE) [nsIDOMNSHTMLDocument.execCommand]" nsresult: "0x80004005 (NS_ERROR_FAILURE)" location:`

原因: 这是因为 firefox 安全性强, 不允许跨域调用。

解决办法: Firefox 要取消 XMLHttpRequest 的跨域限制的话, 第一是从 `about:config` 里设置 `signed.applets.codebase_principal_support = true;` (地址栏输入 `about:config` 即可进行 firefox 设置)。

第二就是在 `open` 的代码函数前加入类似如下的代码:

```
try { netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");
}
catch (e)
{
    alert("Permission UniversalBrowserRead denied.");
}
```

对错误进行处理。

3.3 检查点的设置

自动化测试不像手工测试, 在执行测试用例的过程中, 我们可以随时看到结果, 然后能判断正确与否。而自动化测试对应的就是检查点, 如果不设置检查点, 只有测试步骤的自动化测试是没有任何作用的。因为执行步骤执行完了, 结果不是我们想要的时候, 测试用例也是正确的。所以检查点才是自动化测试执行成功或是失败的检验标准,

而检查点的设置是发现 Bug 的关键。

3.3.1 常用的检测点设置方法

我们通过了一系列的操作后，就需要检查测试执行的结果，常用的就是 Assert 相关的函数。下面我们谈一下最常用的两种方法：

(1) 手工设置检测点

在我们测试用例执行完成之后，对测试结果进行检测。我们还是个例子来说明一下：

例子：在众筹网上喜欢一个项目，在进行了一系列的喜欢操作后，我们要检测操作是否成功。

- 进入到“喜欢的项目”列表页，检查是否有刚刚喜欢的项目。
- 我们可以直观地看到有喜欢的项目，可是怎么用程序判断呢？

如图 3.3.1.1 所示：



图 3.3.1.1 检查喜欢的项目

- 我们要先定位项目名称“[Girls Summit 组合首张 EP 众筹](#)”对应的 Xpath: “//div[@class='m-location']/table/tbody/tr[2]/td/div/div/p/a”，然后获取这个元素对应的 Text。

- 将获取到的 Text 与 “Girls Summit 组合首张 EP 众筹” 字符串进行 `assertEquals()`, 如果相同, 则说明喜欢成功。否则, 喜欢操作失败。
- 同样方法验证一下日期是不是今天, 如果不是, 可能是先前喜欢的操作, 测试用例仍然失败。

(2) Selenium IDE 设置检测点

我们也可以用 Selenium IDE 录制测试用例, 在操作完成后, 需要添加检测点, 此时只要利用 Selenium IDE 提供的 “Show all Available Commands” 菜单选择合适的检测点即可。

例如: 同样是上面的那个喜欢项目操作的例子, 我们的设置步骤如下:

- 进入到 “喜欢的项目” 列表页, 检查是否有刚刚喜欢的项目。
- 如图 3.3.1.2 所示, 右击项目名称, 选择 “Show all Available Commands” 菜单, 然后在打开的子菜单中选择合适的 Assert 菜单。

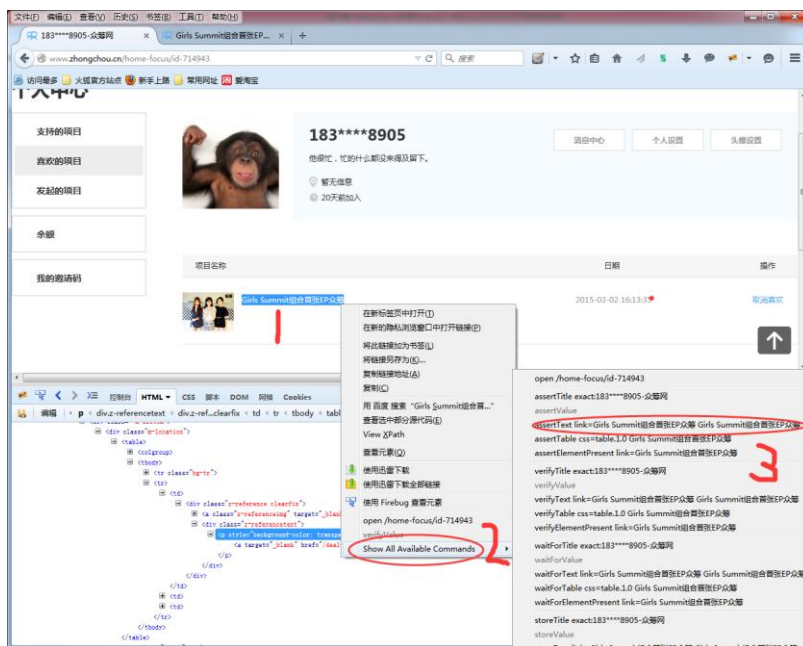


图 3.3.1.2 selenium IDE 设置检测点

- 然后 Selenium IDE 中就会出现相应的 assert 命令，如图 3.3.1.3 所示：

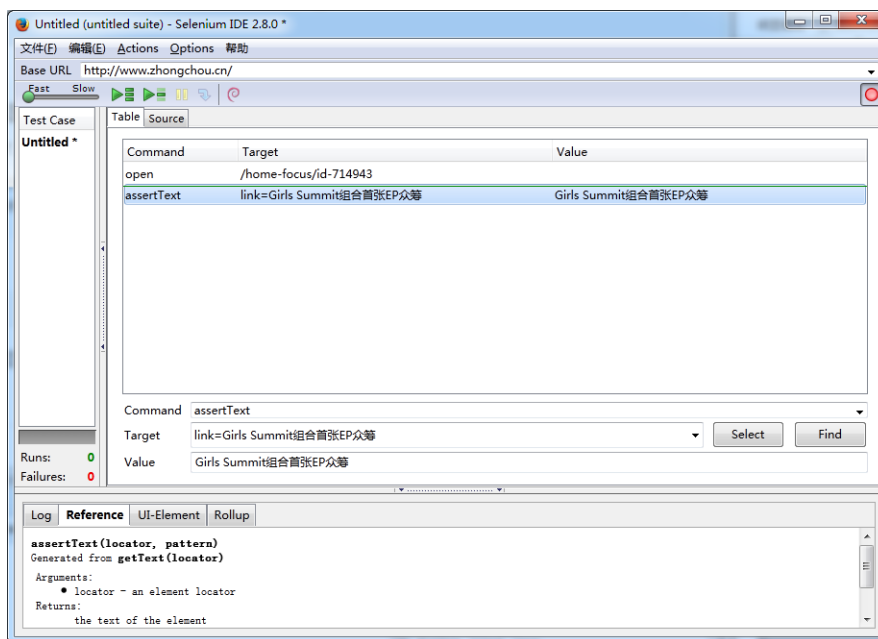


图 3.3.1.3 Selenium IDE 记录 Assert 操作

- 转化成相应的编码，如 python,就可以直接拷到测试用例中使用。转化后的编码如：`self.assertEqual(u"Girls Summit 组合首张 EP 众筹", driver.find_element_by_link_text(u"Girls Summit 组合首张 EP 众筹").text)`

3.3.2 检测点设置技巧

正如我们上面所说的，好的检测点是发现 Bug 的关键。但是并不是说检测点设置的越多越好，因为检测点会消耗机器资源，测试用例出错的时候增加排查难度。所以如何设置检测点呢？通常可以参考如下方法：

（1）根据测试用例的侧重点设置检测点

每个测试用例都有测试的重点，比如说，我们测试登录的时候，登录是否成功，就需要检测。但是在我们的测试项目的时候，需要先登录，这个时候登录就不需要设置检测点了，因为在登录测试用例中已经测试过了。

（2）设置检测点要全面

我们在编写测试用例的时候，一定要全面了解测试操作影响了哪些方面。对影响到的地方，都设置一下检测点，防止出现遗漏的地方。

（3）设置检测点要灵活

设置检测点的时候，我们通常会比较一下实际的结果和预期结果是否相同。可是有些时候，我们不能简单地进行是否相等来判断。比如说：检测图片的时候，可能会检测图片是否显示；有的检测对象在某些页面会换行或是添加空格，与预期有变化，这个时候我们可以判断是否包含关键字即可。灵活使用各种判断函数，才能使自动化测试用例更加健壮。

3.3.3 检测点设置中常见的错误

在测试过程中，我们编写了测试用例，设置了检测点，可是在测试用例投入使用的过程中，我们不得不反复修改测试用例。因为测试用例总是通不过，维护成本很高。虽然这一部分是因为被测对象变化造成的，还有一部分原因是检测点设置的不对。所以常见的检测点设置中的错误如下：

(1) 检测动态变化的元素

检测点不能随着操作而变化，比如说翻页。我们想要测试翻页是否成功，就不能去检测第二页第一个元素是否是某个项目。因为如果项目增加的话，第二页第一个元素的项目可能会变化。应该先取一下第一个位置的项目名称，然后翻页，再判断现在第一个位置的项目是不是和刚刚获取的项目名称相同，如果不同，就证明翻页成功。

(2) 遗漏检测点

在一个测试用例中，我们要检测所有影响到的地方。如喜欢项目操作，如果我们只检测我的喜欢项目列表中有没有刚刚喜欢的项目，这是不够的。还要检测一下这个项目的喜欢数据是否+1，喜欢项目的入口是否变成已喜欢等相关检测点。

(3) 检测点设置过多

既然你说了，检测点是检测 Bug 的关键，我们就在每一步操作后添加检测点。这样做也是多余的，虽然检测点多了，更加安全一点儿，但是过多的检测点儿影响测试用例运行。而且测试用例如果出错了，我们去定位错误的时候，也非常困难，或是一个测试用例出错会导致

相关的测试用例无法执行。

(4) 忘记设置检测点或是检测点不是测试重点

新手写自动化测试用例的时候，往往会写了每一步的测试操作代码，没有添加对应的检测点或是检测点设置不正确。明明是登录操作，操作完成之后却检测页面显示是否正确，这样会不管操作成功与否，测试用例都不会报错，使自动化测试用例失去了意义。

(5) 检测需要刷新才有反映的元素

在测试的时候，有些儿元素在操作完成后需要刷新一下页面才能显示出操作的结果。手工测试的时候，一般会触发刷新操作，可是自动化的时候，如果不刷新，就不符合预期结果。所以我们要添加刷新页面的代码，然后再去检测。

这几种是常见的错误，当然也会有一些儿比较奇葩的检测点设置错误的情况。在此也不能一下列举了，遇到问题，要多尝试几种方法，会在网上搜索解决办法，这也是学习自动化测试必备的技能。

3.4 本章小结

本章的标题虽然是 Webdriver API 简介，可是我们并没有把重点放在 API 介绍上，因为 Webdriver API 网上有官方的文档，也有相当多的相关的文档。我们的重点是页面元素的定位及检测点的设置，而且还包含常见的错误，这也是自动化测试中非常关键的部分，是写好自动化测试必备的技术。在后面的章节中，将主要讲解自动化测试用例的编写，其中涉及到的元素定位和检测点的设置，我们将不在详述了。

第四章 自动化测试用例初探

经过前三章我们对自动化相关内容的学习，我们了解了什么是自动化测试，测试环境的搭建和 WebDriver API 的介绍，定位方法及检测点设置等相关知识点。从本章开始，我们讲探讨测试用例的编写，以及其相关的知识点。

4.1 第一个测试用例 Hello World

几乎所有编程语言的第一个程序就是输出 Hello World，那我们也沿袭这个传统，第一个测试用例就是用百度搜索 Hello World。

首先我们先写一下这个测试用例的手工测试步骤：

- (1) 用浏览器打开百度首页。
- (2) 输入 Hello World，然后单击“百度一下”按钮，进行查找。
- (3) 检测查找到的结果页面，第一项是否包含 Hello World。

这是我们手工测试的步骤，如果要转化成自动化测试用例，需要转化成我们需要的脚本语言编写的程序。

4.1.1 Selenium IDE 录制

如我们前面讲到的，可以利用 Selenium IDE 来录制测试步骤，然后添加上检测点，就可以形成一个测试用例。转化成我们需要的脚本语言格式，如 python，然后保存文件，放到我们的开发环境下就可以调试运行了。

具体的录制方法如下：

- (1) 打开火狐浏览器，打开 Selenium IDE，在 IDE 的 Base URL 中输入要录制的网站地址：<http://www.baidu.com>。
- (2) IDE 默认是处于录制状态的，如果没有，单击红色按钮，录制。
在火狐浏览器的地址栏中输入 <http://www.baidu.com>，打开百度首页。
- (3) 输入“Hello World”，单击“百度一下”按钮，进行搜索。
- (4) 在搜索结果页，右击第一个结果项，如图 4.1.1.1 所示，选择 assert 判断项，设置检测点。



图 4.1.1.1 设置检测点

- (5) 单击 Selenium IDE 的回放按钮，检测录制的测试步骤，如图 4.1.1.2 所示，不过此时检测点会报错，原因是执行太快，页面没有刷新出来。

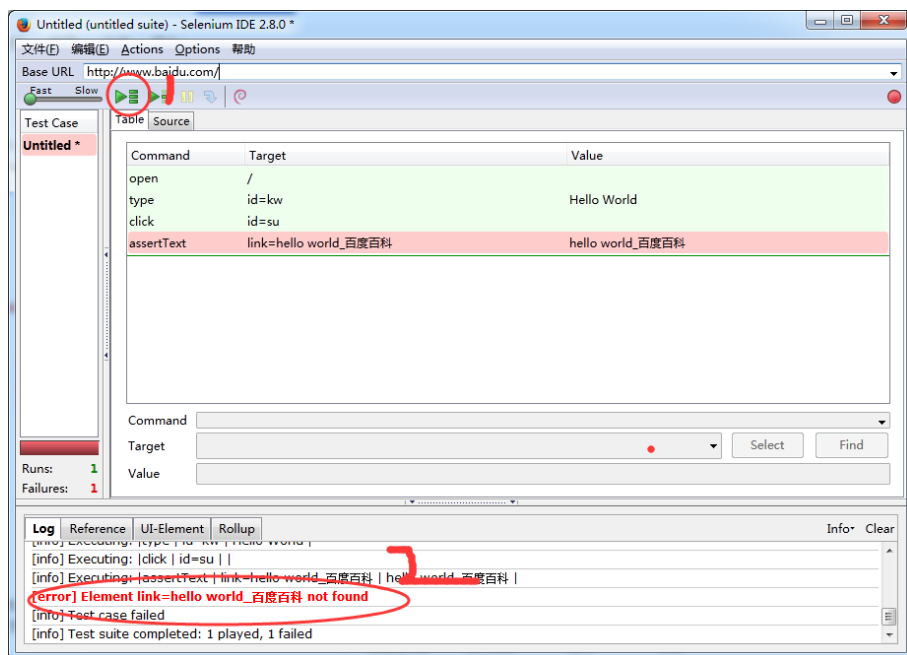


图 4.1.1.2 回放录制的测试步骤

- (6) 为了解决这个问题，我们在检测点前添加一个等待。等待我们要检测的内容出现后，再去执行判断。如图：4.1.1.3 所示，回放验证成功。

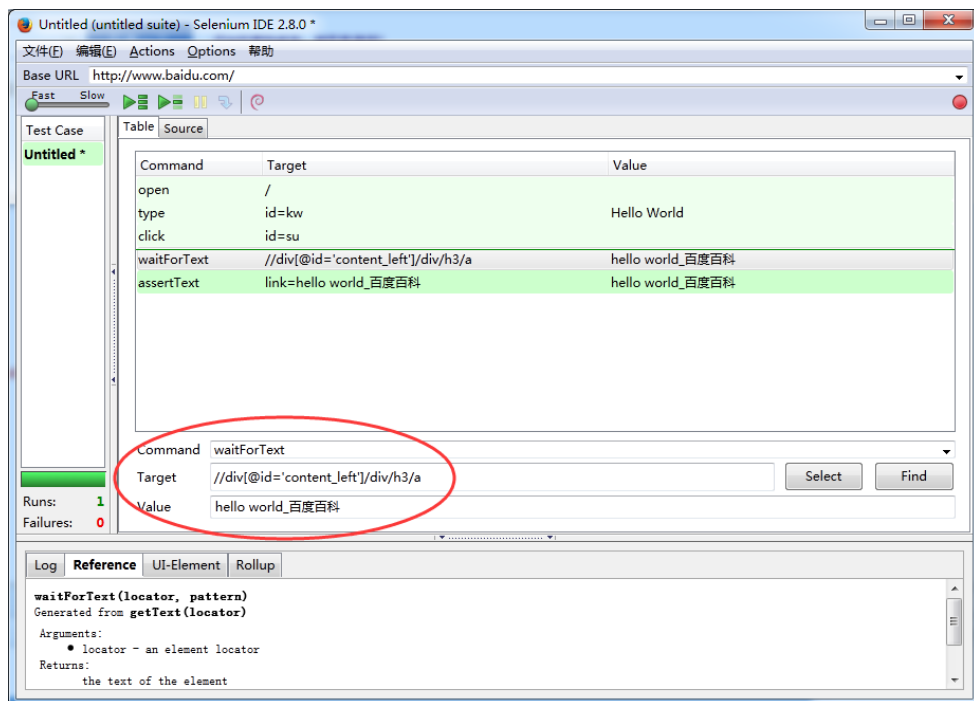


图 4.1.1.3 添加等待命令

- (7) 现在说明我们录制的测试用例没有问题，导出录制的测试用例

为 python(webdriver)格式的，保存为 helloworld.py.如图 4.1.1.4 所示：

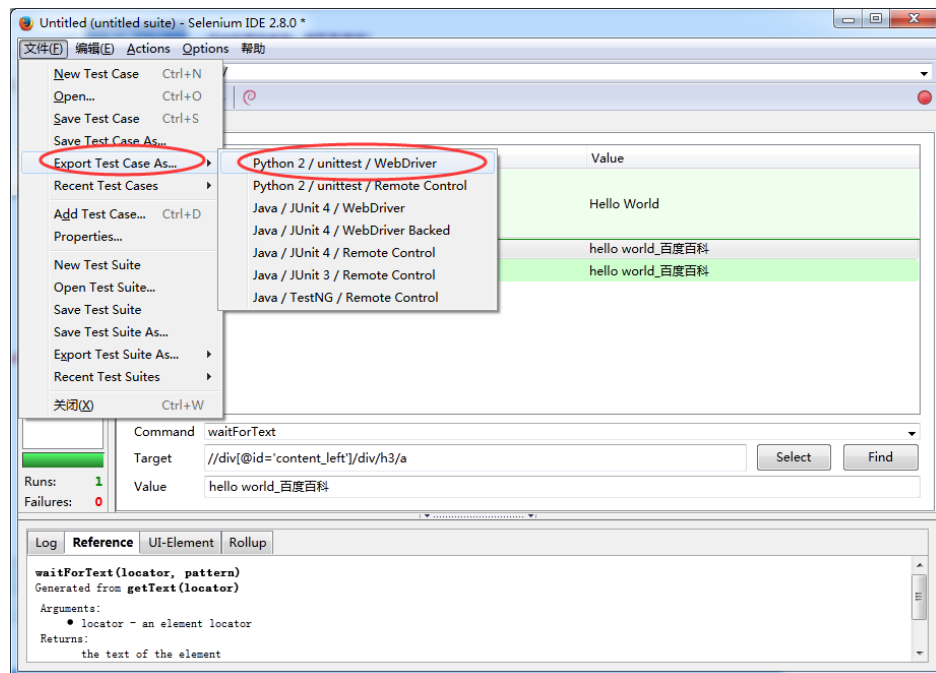


图 4.1.1.4 导出录制的测试用例

(8) 检查导出的代码，Selenium IDE 帮我们导出的代码如下：

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import NoAlertPresentException
import unittest, time, re

class Helloworld(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://www.baidu.com/"
        self.verificationErrors = []
        self.accept_next_alert = True

    def test_helloworld(self):
        driver = self.driver
```

```

driver.get(self.base_url + "/")
driver.find_element_by_id("kw").clear()
driver.find_element_by_id("kw").send_keys("Hello World")
driver.find_element_by_id("su").click()
for i in range(60):
    try:
        if u"hello world_ 百 度 百 科 " ==
driver.find_element_by_xpath("//div[@id='content_left']/div/h3/a").text: break
    except: pass
    time.sleep(1)
else: self.fail("time out")
self.assertEqual(u"hello world_ 百 度 百 科 ",
driver.find_element_by_link_text(u"hello world_百度百科").text)

def is_element_present(self, how, what):
    try: self.driver.find_element(by=how, value=what)
    except NoSuchElementException, e: return False
    return True

def is_alert_present(self):
    try: self.driver.switch_to_alert()
    except NoAlertPresentException, e: return False
    return True

def close_alert_and_get_its_text(self):
    try:
        alert = self.driver.switch_to_alert()
        alert_text = alert.text
        if self.accept_next_alert:
            alert.accept()
        else:
            alert.dismiss()
        return alert_text
    finally: self.accept_next_alert = True

def tearDown(self):
    self.driver.quit()
    self.assertEqual([], self.errors)

if __name__ == "__main__":
    unittest.main()

```

代码讲解：

- 最前面是引用 Webdriver 相关的一些儿函数或方法。
- 创建一个测试类 HelloWorld, 类里是具体的一些儿测试方法。
- 函数 setUp, 初始化火狐浏览器, 要测试的网站 URL 等一些儿常用的变量。
- 函数 test_helloworld() 是我们具体的操作步骤及验证点的检测。
- 函数 is_element_present(): 判断元素是否在当前页面显示;
is_alert_present(): 当前页面是否有浮层;
close_alert_and_get_its_text(): 关闭浮层, 并返回浮层标题。
这几个函数由 Selenium IDE 自动生成, 在本例是没有用到。
- 函数 tearDown() 关闭浏览器, 判断是否有错误。
- 运行测试用例, 这是 python 的功能, 不做详解。

(9) 将代码导入到开发环境中。打开 Eclipse, 新建 PyDev 工程如: WebAuto。将刚刚保存的 helloworld.py 文件导入到这个工程中, 如图 4.1.1.5 所示:

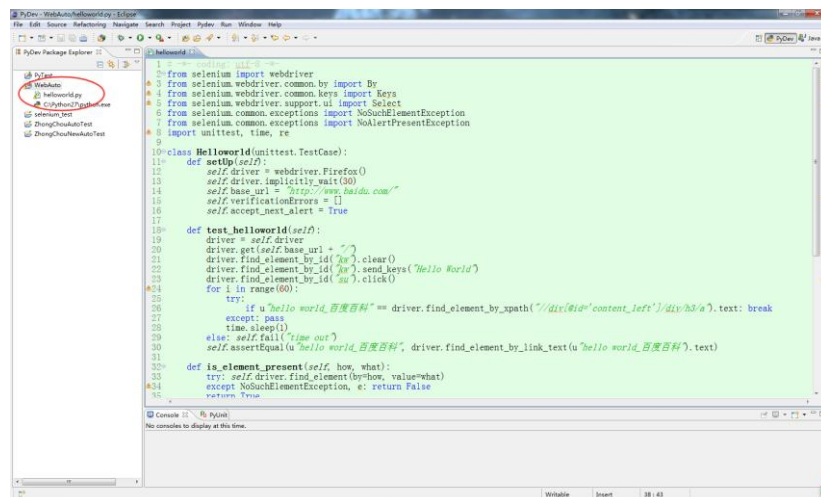


图 4.1.1.5 导入保存的文件

- (10) 运行验证刚刚导入的代码，右击这个文件，“Run as”
-->” python run”，查看运行结果。程序打开火狐，执行了测试用例，执行完成后关闭浏览器。如图 4.1.1.6 所示：

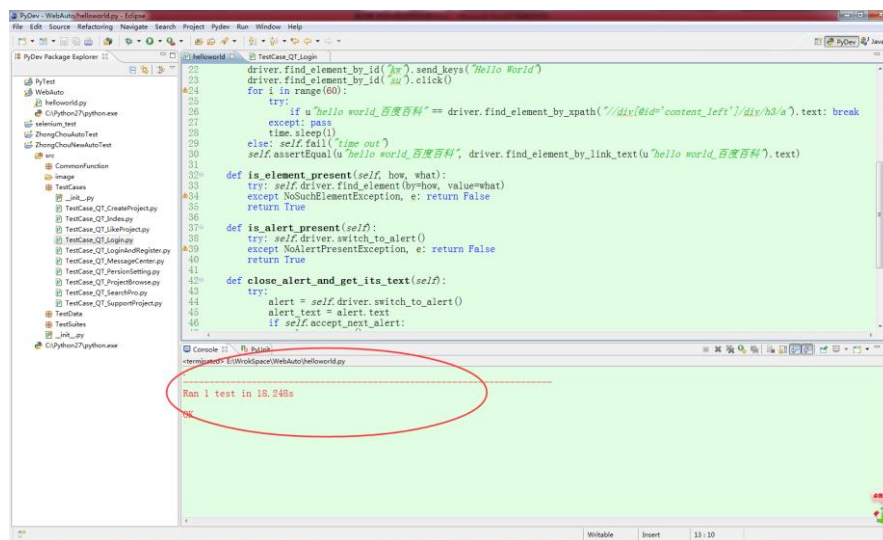


图 4.1.1.6 程序运行结果

- (11) 至此，利用 Selenium IDE 来录制测试用例，并且在开发环境下运行通过。可是我们发现这样录制测试用例还是存在问题的，转化后会有很多用不到的代码存在，而且不符合模块化编程的规范。所以我们还是喜欢自己编写测试用例，录制转化只能当作参考。

4.1.2 手动编写自动化测试用例

兼于用 Selenium IDE 录制转化测试用例会出现很多没有用的代码，并且代码复用性差，转化后的方法也不是最完美的，所以很多自动化高手是不会用这种方法来写自动化测试用例的。

现在我们来讲解一下高手最常用的方法，自己编写自动化测试用例。当然编写之前也会有代码架构组织，公用函数编写，测试数据和

测试用例代码的分离等工作要做。我们在本章就不涉及这些儿方面，只按一个普通的测试用例，以测试流程为序来编写测试用例，其他的内容将在后面的章节讲解。

手工编写自动化测试用例的步骤，我们还以上面的 Hello World 为例：

- 打开 Eclipse，创建 pydev 工程，为了方便起见，我们就用上面创建的 WebAuto 工程。
- 创建测试脚本文件：文件→新建→other→PyDev&Module,然后在打开的对话框中输入 HelloWorld_demo，单击“fininsh”。最后在弹出的对话框中选择“Module:Unittest”。如图 4.1.2.1 所示：

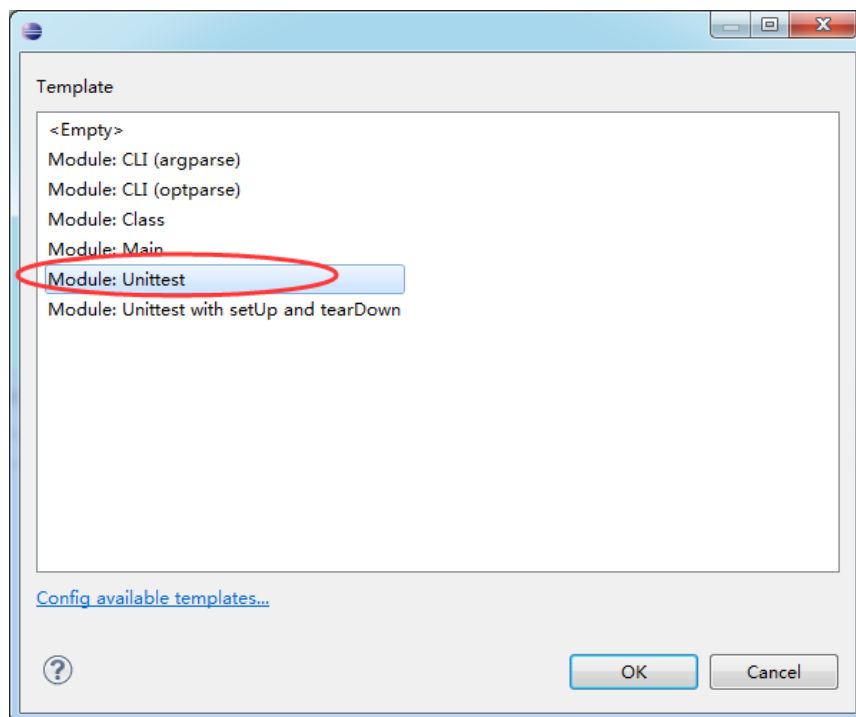


图 4.1.2.1 新建文件 unittest

- 此时会自动生成代码文件，内容如下：

```
'''  
  
Created on 2015-3-4  
@author: sxf
```

```
'''
import unittest
class Test(unittest.TestCase):

    def testName(self):
        pass

if __name__ == "__main__":
    #import sys;sys.argv = ['', 'Test.testName']
    unittest.main()
```

这是一个默认的文件内容，引用了 unittest, 创建 Test 类，测试方法 testName, 以及运行接口 unittest.main().

- 我们需要在此基础上加上我们的测试用例，首先要修改类名为 HelloWorld_Demo，测试函数为 testSearch(注：测试函数必须以 test 开头，否则 unittest 将不识别。) 然后开始加入我们的测试步骤及设置检测点。
- 测试用例代码及分析。

最后我们的测试用例代码如下：

```
# -*- coding: utf-8 -*-
'''
Created on 2015-3-4
@author: sxf
'''

import unittest
from selenium import webdriver
import time

class HelloWorld_Demo(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.base_url = "http://www.baidu.com/"

    def testSearch(self):
```

```

'''
Search Hello World!
'''

driver = self.driver
driver.get(self.base_url + '/')
driver.find_element_by_id("kw").clear()
driver.find_element_by_id("kw").send_keys("Hello World")
driver.find_element_by_id("su").click()
time.sleep(3)
#检查查询的结果
self.assertEqual(u"hello world_百度百科",
driver.find_element_by_xpath("//div[@id='content_left']/div/h3/a").text)

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

if __name__ == "__main__":
    #import sys;sys.argv = ['', 'Test.testName']
    unittest.main()

```

代码讲解：

- (1) 首先我们要引入用例要用到的 Webdriver, time.
- (2) 创建初始化函数 setUp(), 用于初始化火狐浏览器，及测试的 baseURL.
- (3) 创建具体的测试函数 testSearch(), 将测试步骤及检测点添加进去。此处我们用的检测点是利用 Xpath 定位获取要检测的文本，然后和我们预期的相比较。其中的定位操作，还有测试步骤相对应的代码，我们可以从刚刚录制的代码中拷过来，也可以自己写，这样看你的编码水平了。
- (4) 创建测试用例清理函数 tearDown(), 关闭浏览器。
- (5) 这是一个测试用例最基本的结构，再复杂的测试用例，也是这样的结构，只不过测试步骤变多，检测点多些儿而已。

- 用 Eclipse 运行刚刚编写的测试代码，发现和 Selenium IDE 录制的结果一样。而我们的代码比录制转换后的代码简洁很多，这个才是我们想要的结果。

4.2 测试用例代码结构

通过上面我们对百度搜索 hello world 测试用例的讲解，我们对自动化测试用例有了一定程度的了解。就是用代码来完成我们手工测试的步骤，然后利用检测点来检测执行的结果。可是要写好自动化测试用例，单单知道这些儿还是不够的，我们需要了解一下自动化测试用例的结构。

4.2.1 自动化测试代码结构

通常一个完整的自动化测试用例包含以下几部分：

- (1) 测试用例执行的先决条件。

测试用例执行之前的公用变量的定义，或是测试用例执行之前的一些儿先决条件，如用户 A 和 B 要相互发私信操作时，必须相互关注等。而 Python Unittest 是用 setUp() 来处理的，因为所有测试用例执行前，先执行这个函数。

- (2) 具体的测试用例。

根据不同的测试需要，来组织不同的测试用例。测试用例必须包含测试步骤，测试步骤执行完后对执行结果的检测。测试用例命名要以 test 开头，一个测试类中可以包含多个测试用例。

(3) 测试用例执行完后的清理。

测试用例执行完成后，如果对被测试对象产生了一些测试数据，则必须清除。如发布了一些文字，或是测试操作影响了两个测试账号间的关系等。如果不清除，则会影响下次测试用例的执行或是在并发过程中其他的测试用例的执行。

(4) 测试用例的调用。

在 Python 编写的自动化测试用例，可加上对测试用例的调用，否则测试用例将无法执行。

4.2.2 测试用例编写过程中常见的问题

在我们编写测试用例的过程中，会经常遇到一些问题的，这些问题影响着我们的测试用例的质量。所以我们要在编写之前就在脑海里对它们提高警惕，避免这样的错误：

(1) 随意命名测试用例类和函数名。

这涉及到编码规范的问题，变量，函数，类的命名必须有意义，不可随意命名。在自动化测试用例中，不规范的命名，会影响测试用例的可读性，而且一旦执行出错，不便于定位问题。

(2) 前提条件写到具体的测试用例中。

`setup()`函数是每个测试用例执行之前都会调用的，所以这个函数里面写的是所有测试用例都要用到的前提条件。而有很多初学的同学，会把测试步骤都写到具体的测试函数中，这样就会造成把前提条件写了很多次，不符合代码重用机制。

（3）测试用例耦合度高。

测试用例之间应该没有太多的耦合性，各自执行完测试用例，判断检测点之后，退出浏览器。这样不影响其他测试用例的执行，如果我们测试用例之间有耦合，可是使用相同的测试账号的话，单线程执行没有问题，多线程运行的时候，就会造成死锁或是相互影响。

（4）执行完测试用例，不对测试数据进行清理。

执行完测试用例后，要对测试数据进行清理。而执行线上回归任务的自动化测试用例原则是不产生测试数据的。如果本次执行的测试数据不清理，下次就算执行失败了，检测执行结果的时候，如果检测到上次产生的测试数据了，也不会报错的。

（5）测试用例中没有注释。

不管是什么样的语言写的程序，都要求有注释的。由于我们做自动化测试，大多没有经过详细的编码规范的培训，所有不喜欢写注释。可是如果不加注释，时间长了，你就不清楚这个测试用例是做什么的了，需要一步一步地去看代码分析，多费时啊！如果加上注释，只需要看一下注释即可。

4.3 本章小结

本章我们以一个传统的 `hello world` 测试用例讲解了如何利用 Selenium IDE 录制自动化测试用例，以及如何手工编写自动化测试用例。同时还讲解了自动化测试用例的结构及常见错误，从具体的自动化测试用例来介绍如何编写自动化测试用例。接下来的章节，我们将

讲述常用的页面元素的自动化测试用例操作，请大家继续学习。

第五章 常用页面元素自动化操作

经过我们对自动化测试环境的搭建，页面元素的定位，测试点的验证以及自动化测试用例的初探等的学习，可以说我们可以编写自动化测试用例了。本章节我们将对常用的页面元素的自动化操作，作一下详细的介绍，以便大家在以后编写自动化测试用例的时候更加得心应手。

5.1 输入类元素

现在很多网站都实行会员制或是实名制的，所以注册和登录是非常常见的操作，在一个网站上输入各种信息也是必不可少的，我们首先讲解输入类元素。

5.1.1 input 和 textarea 元素

这两个是最常见的操作，这是的 input 的 type 为 text 和 password 类型，不是上传图片的，上传图片我们下节再讲。对于这类输入文本的元素操作对于 python 是一种操作，先对元素进行定位，然后调用 send_keys () 函数，进行输入。

例:

```
def inputvalue(self, findby, elmethod, value):  
    '''  
    通过定制定位方法，在输入框中输入值  
    @param findby: 定位方法，如: byid, byname, byclassname, byxpath 等  
    @param elmethod: 要定位元素的属性值，如: id, name, class name, xpath 等  
    @param value: 要给文本框输入的值  
    '''  
  
    if(findby == 'byid'):  
        self.driver.find_element_by_id(elmethod).send_keys(value)  
    elif(findby == 'byname'):  
        self.driver.find_element_by_name(elmethod).send_keys(value)  
    elif(findby == 'byclassname'):  
        self.driver.find_element_by_class_name(elmethod).send_keys(value)  
    elif(findby == 'byxpath'):  
        self.driver.find_element_by_xpath(elmethod).send_keys(value)
```

这个 inputvalue 函数就是我对这类操作函数进行的封装，根据不同的定位类型，先对要输入的元素进行操作，然后 send_keys(value) 将关键字输入进去。

注：有些儿网站在输入框失去焦点后不会清除里面默认的字符，为了确保输入正确，我们在定位到元素后先调用 clear() 函数，将默认的字符清除后再进行输入操作。

5.1.2 input 上传文件

当 input 的 type 为 file 时，说明这个元素是上传文件的，这个时候我们的普通操作是，单击这个元素，然后在弹出的上传文件对话框中选择要上传的文件，最后是单击上传按钮。

而在我们写自动化测试的时候，如果按这个步骤操作的话，会非常麻烦，因为我们无法定位弹出的上传文件对话框，要借助于第三方

工具。其实没有必要这么麻烦，我们可以按照上面的上传文字的方法，不过参数不是文件，是文件路径，这样就能上传了。

例：

```
self.Driver.find_element_by_xpath (location).send_keys(filepath);
```

这个例子是用 id 来定位的，然后把文件路径作为参数传给这个元素。

注：这个文件路径最好是相对路径，将文件和代码放到一起，这样在代码放到其他的地方运行的时候，就不会出现找不到需要上传文件的现象了。

5.1.3 特殊按键的输入

在输入过程中，有的时候我们需要输入特殊的按键，可是组合键。如按 ESC 或是 CTRL+A 等，这些儿我们无法用键盘输入的键，应该如何输入呢？

其实和普通的字符输入差不多，不过需要调用特殊的类 Key，例如：

(1) 执行按 ESC 键操作

```
self.driver.find_element_by_name ('image_file').send_keys(Keys.ESCAPE)
```

(2) 执行按 CTRL+a 键操作

```
self.driver.find_element_by_name ('image_file').send_keys((Keys.CONTROL, 'a'))
```

当然对元素的输入操作这些儿是常见的，也有不常见的，如果遇到，可以酌情处理，多尝试几种方法。

5.2 单击类元素

好像自从鼠标出现后，用鼠标完成的操作越来越多了，在我们自动化测试过程中，单击类操作是必不可少的操作之一。下面我们来看一下单击类操作的自动化测试代码：

5.2.1 按钮类元素单击

在页面上单击按钮用来完成特定的操作，比如登录，注册，提交等。方法是，先定位到这个元素，然后调用 `click()` 函数。

例如：

```
def clickitem(self, findby, elmethod):
    """
    通过定制定位方法，在对应的项目上执行单击操作
    @param findby: 定位方法，如：byid, byname, byclassname, byxpath 等
    @param elmethod: 要定位元素的属性值，如：id, name, class name, xpath, text
    等
    """
    if findby == 'byid':
        self.driver.find_element_by_id(elmethod).click()
    elif findby == 'byname':
        self.driver.find_element_by_name(elmethod).click()
    elif findby == 'byxpath':
        self.driver.find_element_by_xpath(elmethod).click()
    elif findby == 'bytext':
        self.driver.find_element_by_text(elmethod).click()
    elif findby == 'byclassname':
        self.driver.find_element_by_class_name(elmethod).click()
```

函数 `clickitem()` 通过各种定位方法，先将要定位的元素定位，然后执行 `click()` 操作。

5.2.2 超级链接单击操作

在网站上单击超级链接，从而执行相应的操作。可以像按钮一样，先定位到这个超级链接，然后执行 `click()` 函数。但是针对超级链接，有专门的定位方法：

```
driver.find_element_by_link_text(link_text).click()
```

或

```
driver.find_element_by_partial_link_text(link_text).click()
```

将超级链接全部文字或是部分文字用来定位，然后调用 `click()` 函数，就可以完成单击操作。

5.2.3 鼠标右击和双击操作

虽然在测试过程中，对被测试元素进行右击和双击操作不太常用，可是这两个操作还是很有用的。对于不太常用的操作，`webdriver` 就没有将这个操作封装到 `Element` 类中，而是在 `ActionChains` 类中。以下用例参考了虫师的博客（<http://www.cnblogs.com/fnng/p/3288444.html>，他的博客写的相当好，建议去学习一下。）

引用 `ActionChains` 类：

```
from selenium.webdriver.common.action_chains import ActionChains
```

（1）右击操作：

```
chain = ActionChains(driver)
implement = driver.find_element_by_xpath("location")
chain.context_click(implement).perform()
```

（3）双击操作：

```
qqq =driver.find_element_by_xpath("location")  
#对定位到的元素执行鼠标双击操作  
ActionChains(driver).double_click(qqq).perform()
```

对于其他的鼠标操作，请参考我们推荐的博客，自行学习，我们本章只讲常用的操作。

5.3 选择类元素

在网站创建过一再要求操作要简单化，为了减少用户操作，就会提供很多选择类的操作，如超级链接类品牌选择，单选类操作，复选类操作，下拉菜单类选择操作等。这类操作有的是简单的单击一下，有的需要调用相应的函数，所以我们下面详细讲述一下。

5.3.1 超级链接类选择

超级链接类选择往往出现在购物网站上，如京东，淘宝等。这类型的选择其实没有什么特殊的，可以采用上面提到的超级连接定位法，普通的定位，单击要选择的分类即可。

此时注意，不同分类的选择，有可能显示结果的地方是新页面，或是 iframe，此时要检测搜索的结果，就要先切换到新页面或是 iframe 当中。

5.3.2 单选框选择

在填写信息的时候，经常会遇到性别选择，或是其他类型的单选按钮的选择。其实单选框如同普通元素，先对其进行定位，然后执行 click() 操作。

例如：

```
sexelem= driver.find_element_by_xpath("sex")
sexelem.click()
```

执行了操作后，定位的单选按钮就处于选中状态了。如果不想选中个单选按钮，可以单击其他的单选项，或是招待 `clear()` 函数，就取消选中。判断单选按钮是否处于选中状态，可以调用函数 `isSelected()`。

注：有的单选按钮的选择圆圈和显示文字可以分开定位，此时单击哪一个都可以，要考虑哪个方便定位。

5.3.3 复选框选择

为了增加用户体验，给用户提供更多的选择，复选框的应用也是非常多的。复选框的使用是很简单的，定位到相应的复选框，然后单击。

例如：

```
pricelem= driver.find_element_by_xpath("price")
pricelem.click()
```

复选框的操作和单选框差不多，想取消选择，可以调用 `clear()` 函数，也可以再次单击就可以取消选择。判断是否选中，调用函数 `isSelected()`。判断是否可用，调用函数 `isEnabled()`。

5.3.4 下拉菜单类选择

在填写信息的时候，城市的选择；可是对页面信息的选择，都会用到下拉菜单。在 `webdriver+python` 中，对应的是 `Select` 类，如：

```
from selenium.webdriver.support.ui import Select
```

我们要操作下拉菜单，先用普通元素的定位方法，定位到这个元

素，然后转化为 select 类型的。

```
select = Select(self.driver.find_element_by_id("selected"))
```

接下来调用 `select_by_visible_text()` 函数来选择对应的菜单项。

例：`select.select_by_visible_text("北京")`

在 java 版的 webdriver 还有其他的相应操作，不过 python 好像只有这一个操作。对于下拉菜单，还可以像普通元素一样，先单击下拉菜单，拉出菜单项，然后单击对应的选择项。不过这样操作时而好用，时而不好用，不建议用这种方法，此处只提供一种参考。

5.4 获取元素的文本

在设置检测点的时候，我们经常需要获取操作后影响到页面元素，然后和预期的相比。所以此时用到最多的就是获取元素的文本，与预期的相比，相同则说明测试通过。

获取元素文本的方法，不管你是 `div`,`link`,或是其他的元素，都是一样的。先定位到这个元素，然后获取 `text` 属性。例如：

```
def gettext(self,findby,elmethod):
    """
    通过定制定位方法，获取指定元素的文本
    @param findby: 定位方法，如：byid,byname,byxpath 等
    @param elmethod: 要定位元素的属性值，如：id,name,xpath 等
    @return: 返回获取到的元素文本
    """
    if(findby == 'byid'):
        return self.driver.find_element_by_id(elmethod).text
    elif(findby == 'byname'):
        return self.driver.find_element_by_name(elmethod).text
    elif(findby == 'byxpath'):
        return self.driver.find_element_by_xpath(elmethod).text
    elif (findby=='byclassname'):
```

```
        return self.driver.find_element_by_class_name(elmethod).text
    elif (findby=='bycss'):
        return self.driver.find_element_by_css(elmethod).text
```

`gettext()`函数通过各种定位方法，定位到要获取 `text` 的元素，然后将 `text` 返回。

5.5 页面或 `iframe` 切换

在自动化测试过程中，难免会遇到，打开新页面或是切换到新的 `iframe` 中的情况。如果我们不将代码做相应的切换操作，将句柄切到新的页面或是 `iframe` 中，我们定位的时候，将出现找不到元素的情况。所以，在适当的时候切换一下句柄，然后再进行测试操作。

5.3.1 页面间的切换

在网页中单击链接打开新网页，然后在新打开的页面中操作或是验证新页面中操作的结果等操作。如果我们用 `Selenium IDE` 录制脚本的时候，回放或是调试代码，执行到打开新页面后就会提示元素找不到。可是我们用 `Selenium IDE` 验证的时候，能找到要定位的元素，这是什么原因啊？

其实就是当前的 `driver` 句柄在第一次打开页面的时候，取的是页面句柄，但是当我们打开新页面后，句柄还是原来的页面的。在原来的页面上查找新页面的元素，当然会找不到了。所以在我们验证新页面的元素的时候，需要先调用 `switch_to_window(0)` 将句柄切换到新打开的页面。

例如：


```
self.driver.switch_to_window(self.driver.window_handles[-1])
```

`self.driver.window_handles` 为获取所有打开窗口的句柄, -1 为获取最后一个窗口的句柄, 即最新打开的窗口的句柄。

5.3.2 iframe 间的切换

iframe 间的切换在正常的页面中偶尔会遇到, 如果要自动化测试后台相关的内容, 则 iframe 是非常多的。几乎所有的操作都要在不同的 iframe 之间进行切换, 本节我们只讲一种方法, 更多的方法请参考: http://blog.sina.com.cn/s/blog_68f262210101mcxp.html

例如现在页面上有两个 iframe, 一个 name 属性为 “frame1”, 另一个 name 属性为 “frame2”, 我们现在需要将句柄切换到 frame2 中:

```
this.driver.switchTo().frame("frame2");
```

注: 切换过 iframe 后, 就可以在这个里面定位元素或是执行操作了。有的时候如果程序招待太快, 可能会出现句柄切换不成功的现象, 此时在切换句柄语句后, 添加等待操作, 然后再去查找就可以成功了。

5.6 本章小节

本章我们讲述了自动化测试中最常见的元素操作, 就像我们在盖高楼大厦之前的砖, 水泥和钢筋。虽然这只有这些儿东西, 我们不能建起我们的大厦, 可是这也是先决条件。必须掌握的东西, 本章节我们没有列出所有的元素, 因为网站应用的时候非常灵活, 我们只讲基

本的，对于不常用的，请自行学习。第六章，我们将着手绘制建筑蓝图，讲解自动化测试的实施及测试框架的搭建。

第六章 自动测试实施

从本章开始，我们将讲述如何实施自动化测试，在第一章的时候，我们也提供了自动化实施的步骤。那些儿步骤是指导方针，可以按着这一步步地去实施，可是有点儿笼统。本章我们将从具体实例入手，按前面的我们提到的步骤来讲解，通过本章的学习，你可以从一个被测试的网站着手，从零开始建立起普通的回归测试用例。

6.1 评审被测试对象功能

现在你的老大给你分配个任务：我们公司的业务已经成熟，网站主要功能也基本上不会变化了，我们想在每次上线后自动回归一下主功能。所以呢，我们想让你对我们的网站建立起自动化测试，用例要求覆盖主要功能。

当你接到这个任务的时候，你应该如何去做呢？不是去想用什么来写自动化，或是马上用你擅长的语言马上去写测试脚本。而是要先分析一下被测网站的功能，哪些儿是主要的，哪些儿必须自动化，哪些儿不能自动化。

现在我们以众筹网（www.zhongchou.cn）为例，分析一下主要功能：

- （1）登录注册
- （2）浏览项目
- （3）搜索项目
- （4）查看项目详情
- （5）支持项目
- （6）个人信息设置
- （7）发起项目
- （8）查看自己相关的项目
- （9）发起的项目管理和订单发货
- （10）消息中心

等等，这十个主要功能是网站最基本的功能，然后我们分析一下哪些儿能自动化，哪些儿不能？

能自动化的部分：1，2，3，4，6，8，10

不能自动化的部分：5，7，9

为什么不能自动化呢？5，支持项目，涉及到真实的支付，影响项目对账，因为不管是测试环境还是线上环境，都是真实的支付，如果有支付的测试环境，也是可以自动化的。

7，发起项目，发起项目通过审核后，会在线上产生测试数据，影响用户体验。这违反了测试不能对线上产生垃圾数据的原则，所不建议自动化，但是可以测试发起项目的流程，不对项目进行上线。

9, 发起的项目管理和订单发货。发起项目我们就没有自动化, 所以这块功能是没有数据的, 再者订单发货会短信通知支持者, 会影响到用户, 不建议自动化。

当你接到任何一个测试任务的时候, 需要先这样分析一下被测试对象的功能及是否合适自动化测试。然后查找一下是否存在对应功能的手工测试用例, 一般公司都会做测试用例的收集工作的。如果没有, 就先编写对应的测试用例, 然后再去转化成自动化测试用例。

6.2 评审被测对象编码

评审完了主功能, 然后开始评审被测试对象的编码, 选择与被测试对象相同或是同一系列的脚本语言来写测试用例。一般网站编码无非是 ASP.NET, PHP, JSP 等, 编写测试用例的脚本语言也可以用 php, python, java 等, 根据实际情况, 做出相应的选择。

经查看, 我们的众筹网是用 php 编写的, 按原则来说, 我们应该用 php 来编写测试用例。Webdriver 也支持 php, 但是 php 支持库不多, 在用例组织和报告生成等方面也存在着不足, 所以目前用 php 编写自动化测试用例的不多。综合考虑各种因素, 我们决定用 python 来编写自动化测试用例。

6.3 自动化测试框架的选择

目前业界做页面自动化的都是选用 selenium1.0 RC 或是 selenium2.0(webdriver), 而它们之间是存在着区别的:

Selenium RC 工作原理:

- (1) RC server 在服务端启动 浏览器 并将Core 注入到浏览器中
(为了解决浏览器的同源策略)
- (2) 我们的测试脚本调用Client API, Client将操作转化成标准的selenese语句发送给RC Server。
- (3) Selenium Core 解释selenese 语句,通过js的方式操作浏览器。

Webdriver 工作原理:

- (1) WebDriver 启动目标浏览器, 并绑定到指定端口。该启动的浏览器实例, 做为web driver的remote server。
- (2) Client 端通过CommandExcuter 发送HTTPRequest 给remote server 的侦听端口(通信协议: the webdriver wire protocol)。
- (3) Remote server 需要依赖原生的浏览器组件(如: IEDriver.dll, chromedriver.exe), 来转化转化浏览器的native调用。

那么remote server端的这些功能是如何实现的呢? 答案是浏览器实现了webdriver的统一接口, 这样client就可以通过统一的restful的接口去进行浏览器的自动化操作。目前webdriver支持ie, chrome, firefox, opera等主流浏览器, 其主要原因是这些浏览器实现了webdriver约定的各种接口。

优缺点:

- (1) RC 需要启动一个 RCserver, 就直观感觉上多了一个步骤。

(2) RC 采用js 的方式，稳定性和兼容行方面还是取决与js的代码的质量。有时需要自己去写js代码来扩展相应的功能。由于自己是从RC用起的，对于RC js的方式还是比较接受，感觉比较灵活。

(3) 至于弹出对话框和警告框的处理，RC 原生是不支持的。不过可以通过第三方软件来解决，比如 autoit。我用的就是 autoit，使用简单方便，是一个不错的工具。

通过上面的分析，结合现在的时代潮流，我们采用高版本的 selenium 2.0 来作为我们的自动化测试的框架。

6.4 自动化测试用例运行环境

由于我们采用的是 python 开发的脚本语言，对运行环境要求不是太高，可是相应的包还是要安装的。在脚本编写期间，我们一般是在自己机器上编写和调试的，运行环境是 windows 环境。在后期的交付运行，自动执行回归的时候，我们要放到服务器上，服务器是 linux 环境的。

在 Windows 环境和 linux 环境下，python 运行环境差别不大，所以不要太担心环境的问题。可是也有需要注意的地方，比如说文件路径的写法，两个环境下还是不同的。这也是我们一再要求用相对路径，最好用函数来获取路径，而不是把路径写死到代码中的原因。

6.5 自动化代码架构的规划

任何一个好的程序，都需要有好的规划。考虑到现在的结构化程

序设计，我们的自动化测试代码也需要好好规划一下架构，不能像记流水帐似的按测试用例的步骤罗列下来。

6.5.1 代码架构规划的原因

自动化测试用例的主要工作就是回归测试，由于其任务的特殊性，就决定了自动化测试用例是成体系的，大里的自动化测试用例放在一起，如果不好好规划一下，会给后期的工作带来很大的影响。

代码规划的原因：

（1）增加可读性。根据用途，将代码进行分类存放，方便团队其他成员进行代码管理与合作开发。

（2）便于维护。自动化测试用例不可能是一成不变的，被测试的网站变化了，我们对应的自动化测试用例也要进行相应的维护。如果不进行规划，出现问题后不方便定位，更加会耗费大量的成本去维护。

（3）提高代码利用率。结构化程序设计，要求我们对代码重复利用率要高，同样的代码要写在公用函数或是方便，减少代码量。

6.5.2 如何规划代码架构

目前自动化的用法主要有两种：1，简单录制回放转化成的测试用例，此用例用于反复测试的功能点，一旦功能上线后便不再用。目的是加快回归测试的执行。特点就是对功能进行反复测试，不考虑通用性，健壮性等。2，用于回归测试的系统化的测试用例。此种用例是需要反复执行的，一定要考虑程序的通用性，健壮性以及执行时间

等。

针对第一种测试用例，虽然使用时间比较短，通用率也不高，不过有些儿公司还是会收集这样的测试用例的，以便以后同样的或是类似的功能回归测试的时候用。此时可以按业务线对测试用例进行存放，测试用例中包含测试脚本和测试数据。

回归测试自动化测试用例是我们的重点，所以我们要详细规划一下：

（1）为了增加代码的通用性，我们把大部分测试用例需要执行的操作步骤封装成函数，放到公共的类中，并把公共的类也按功能进行分类，统一放到一个文件夹下，如 **CommonFunction**。

（2）将所有的测试用例放到一个文件夹下，测试用例与测试数据分离开，以便网站有变化的时候，我们只需要修改相应该的测试数据即可。测试用例放到一个文件夹下，如：**TestCases**。

（3）测试数据放到和测试用例同名的 **Xml** 文件中，然后通过公用的数据读取方法，对测试数据进行读取，然后在测试用例中调用通用的函数，读取到的数据传递过去。所有数据存放到统一的文件夹下，如：**TestData**。

（4）测试用例集的存放。由于不同的测试目的，我们会运行不同的测试用例。如果测试目的 **A**，我们将运行测试用例 **1, 2, 3**；测试目的 **B**，我们运行测试用例 **1,3,4,5**。所以我们会有不同的测试用例集文件，将这些儿文件统一存放到一个文件夹下，如：**TestSuites**。

（5）其他的资源文件，可以创建不同的文件夹来进行存放，如图片

文件夹 image。

所以，经过规划后，我们的测试用例结构如图 6.5.2 所示，图中的文件是我先前写的测试用例。下面的章节，我们会具体讲述测试用例：

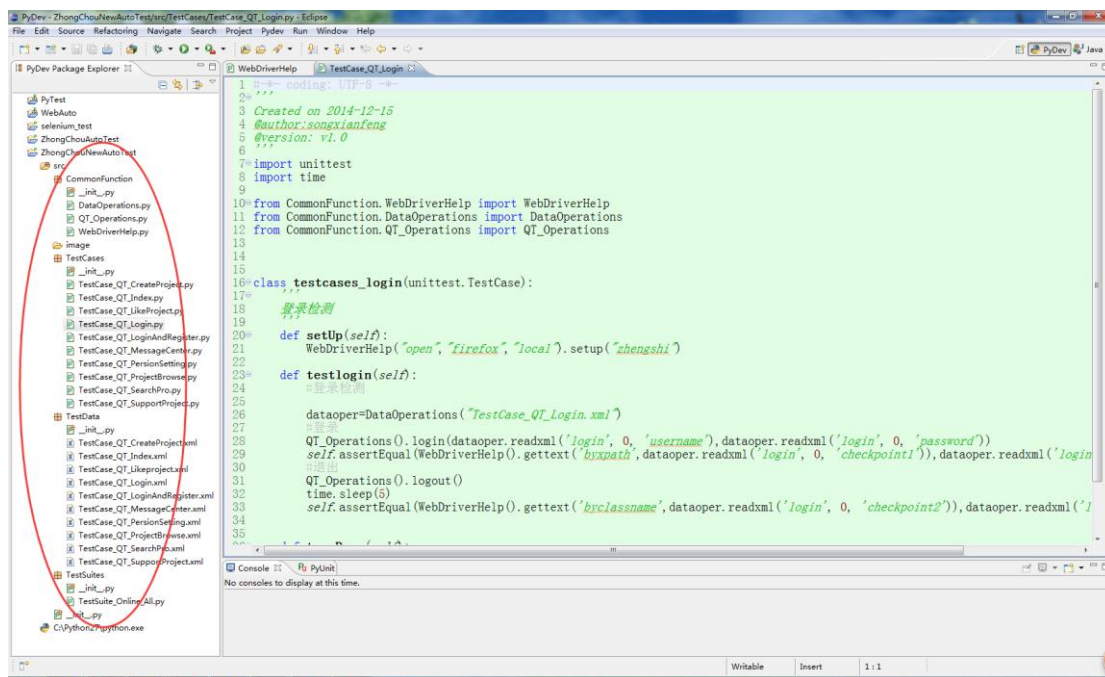


图 6.5.2 自动化测试用例代码结构规划

6.6 编写自动化测试用例

我们规划好了代码架构，就可以编写具体的测试用例了。首先，我们写登录的测试用例，在写自动化测试用例的时候，我们要先写一下公用函数类。根据需要，我们写了三个通用的类放到 CommonFunction 文件夹下：WebDriverHelp 用来存放所有页面操作用到公用方法；QT_Operations 用来存放具体操作功能块，如登录，退出等；DataOperations 用来存放所有数据操作，用来读取 xml 中的测试数据。

然后我们在 TestCases 文件夹下编写登录的测试用例

TestCase_QT_Login.py 执行具体的测试操作，验证测试用例执行是否成功。

并把测试用例需要的测试数据，放到 TestData 文件夹下 TestCase_QT_Login.xml 文件中。如果网站有所变化，可以修改这个里面的测试数据，从而减少代码维护的成本。

6.6.1 WebDriverHelp 类的内容

WebDriverHelp 是我们所有测试用例能用到的方法，可以在编写测试过程中不断抽象出来，写到这个类里面，相当于对 Webdriver 再次做了一下封装。

下面是部分代码，以供大家参考：

```
@author:songxianfeng
@copyright: V1.0
'''
import time
import datetime
from selenium import webdriver
from selenium.webdriver.support.ui import Select

from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.keys import Keys
global G_WEBDRIVER, G_BROWSER_TYPTPE, DRIVER

class WebDriverHelp(object):
    '''
    本类主要完成页面的基本操作，如打开指定的URL，对页面上在元素进行操作等
    '''

    def __init__(self, btype="close", atype="firefox", ctype="local"):
        '''
        根据用户定制，打开对应的浏览器
        @param bType: 开关参数，如果为close则关闭浏览器
        '''
```

```

@param aType: 打开浏览器的类型, 如chrome,firefox,ie 等要测试的浏览器类型
@param cType: 打开本地或是远程浏览器: local,本地; notlocal: 远程
"""
global DRIVER
if( btype == "open" ):
    if( atype == "chrome" ):
        if(ctype == "local"):
            DRIVER = webdriver.Chrome()
            DRIVER.maximize_window()
        elif(ctype == "notlocal"):
            DRIVER = webdriver.Remote(command_executor='http://124.65.151.158:4444/wd/hub',
desired_capabilities=webdriver.DesiredCapabilities.CHROME)
            DRIVER.maximize_window()
        elif( atype == "ie" ):
            if(ctype == "local"):
                DRIVER = webdriver.Ie()
                DRIVER.maximize_window()
            elif(ctype == "notlocal"):
                DRIVER =
webdriver.Remote(command_executor='http://124.65.151.158:4444/wd/hub',desired_capabilities=webdriver.Desir
edCapabilities.INTERNETEXPLORER)
                DRIVER.maximize_window()
            elif( atype == "firefox" ):
                if(ctype == "local"):
                    DRIVER = webdriver.Firefox()
                    DRIVER.maximize_window()
                elif(ctype == "notlocal"):
                    DRIVER =
webdriver.Remote(command_executor='http://10.20.5.56:4444/wd/hub',desired_capabilities=webdriver.DesiredC
apabilities.FIREFOX)
                    DRIVER.maximize_window()

self.DRIVER = DRIVER

def setup(self,logintype):
    """
    定制测试URL
    @param loginplace: 指定测试的URL: qiantai: 前台测试地址, houtai: 后台测试地址, zhengshi: 正
式环境测试地址
    ysh: 原始会测试地址 zhengshiysh: 正式原始会测试地址
    """
    try:
        qiantai_url = "http://test.zhongchou.cn"
        ysh_url = "http://test.ysh.zhongchou.cn"
        houtai_url = "http://test.admin.zhongchou.cn"

```

```

zhengshi_url = "http://www.zhongchou.cn"
zhengshi_ysh_url = "http://ysh.zhongchou.cn"

if(logintype=="qiantai"):
    self.DRIVER.get(qiantai_url)
elif(logintype=="houtai"):
    self.DRIVER.get(houtai_url)
elif(logintype=="zhengshi"):
    self.DRIVER.get(zhengshi_url)
elif(logintype=="ysh"):
    self.DRIVER.get(ysh_url)
elif(logintype=="zhengshiysh"):
    self.DRIVER.get(zhengshi_ysh_url)
else:
    print '路径错误!'

    self.DRIVER.implicitly_wait(1)
except NoSuchElementException:
    print '您选择的测试地址出错!!'

def teardown(self):
    """
    关闭浏览器
    """
    self.DRIVER.quit()

def geturl(self,url):
    """
    打开指定的网址
    @param url: 要打开的网址
    """
    self.DRIVER.get(url)

def selectvalue(self,findby,select,selectvalue):
    """
    通过定制定位方法和要选择项的文本，选择指定的项目
    @param findby: 定位方法，如: byid,byname,byclassname 等
    @param select: 要执行选择操作的下拉框句柄
    @param selectvalue: 下拉框中要选择项的文本
    """
    if(findby == 'byid'):
        select = Select(self.findelementbyid(select))
    elif(findby == 'byname'):
        select = Select(self.findelementbyname(select))
    elif(findby == 'byclassname'):
        select = Select(self.findelementbyclassname(select))
    select.select_by_visible_text(selectvalue)

```

```

def inputvalue(self,findby,elmethod,value):
    """
    通过定制定位方法，在输入框中输入值
    @param findby: 定位方法，如: byid,byname,byclassname,byxpath 等
    @param elmethod: 要定位元素的属性值，如: id,name,class name,xpath 等
    @param value: 要给文本框输入的值
    """
    if(findby == 'byid'):
        self.findelementbyid(elmethod).send_keys(value)
    elif(findby == 'byname'):
        self.findelementbyname(elmethod).send_keys(value)
    elif(findby == 'byclassname'):
        self.findelementbyclassname(elmethod).send_keys(value)
    elif(findby == 'byxpath'):
        self.findelementbyxpath(elmethod).send_keys(value)
    .....

```

上面的代码都有很详细的注释，在此就不一一讲述了。

6.6.2 QT_Operations 类的内容

QT_Operations 类是业务相关的公用功能块的封装，以便增加函数的公用性，减少代码量。例如，登录，在很多测试用例的第一步都需要先登录再操作的。所以你可以抽象出测试用例中的功能模块，封装后放到这个类中。

QT_Operations 类的部分代码展示：

```

#-*- coding: UTF-8 -*-
'''
Created on 2014-12-15
@author:songxianfeng
@copyright: V1.0
'''

import time
import win32api
import win32con

from WebDriverHelp import WebDriverHelp

```

```

class QT_Operations(object):
    '''
    众筹前台相关操作
    '''

    def login(self, userName, passwd):
        '''
        从首页直接登录
        @param userName: 用户名
        @param passwd: 密码
        @param type1: 指示登录方式, 1为从主页登录, 2, 从登录页登录
        '''

        WebDriverHelp().clickitem("byclassname", "Js-showLogin")
        time.sleep(3)
        WebDriverHelp().clearvalue('byname', 'username')
        WebDriverHelp().inputvalue('byname', 'username', userName)
        WebDriverHelp().clearvalue('byname', 'user_pwd')
        WebDriverHelp().inputvalue('byname', 'user_pwd', passwd)
        time.sleep(1)
        WebDriverHelp().clickitem("byclassname", "a-btn")
        time.sleep(5)

    def logout(self):
        '''
        退出登录
        '''

        WebDriverHelp().geturl("http://www.zhongchou.cn/usernew-logout")
        .....

```

展示部分只包含了登录和退出功能，其他的功能可以根据测试用例的需要进行添加。

6.6.3 DataOperations 类的内容

DataOperations 类是对测试数据进行读取的操作，我们是用 xml 来存放测试数据的，所以测试用例执行的时候，需要先将测试数据读取出来，传递给相应的函数来对测试用例进行执行。然后根据执行的结果，判断测试用例是否执行成功。

DataOperations 的内容:

```

#-*- coding: UTF-8 -*-
'''
Created on 2014-12-15
@author:songxianfeng
@copyright: V1.0
'''

import MySQLdb
from xml.dom import minidom
global DOC,CONN

class DataOperations(object):
    '''
    数据读取相关操作
    '''

    def __init__(self,filename):
        '''
        初始化xml文档
        '''
        global DOC,CONN
        DOC = minidom.parse("../testData/"+filename)

    def readxml(self,ftagname,num,stagname):
        '''
        从指定的文件中读取指定节点的值
        @param ftagname: 起始节点的名称, 如: project
        @param num: 取与起始节点相同的第num个节点
        @param stagname: 起始节点下的二级节点
        @return: 返回二级节点的值
        '''
        root = DOC.documentElement
        message=root.getElementsByTagName(ftagname)[num]
        return message.getElementsByTagName(stagname)[0].childNodes[0].nodeValue

    def readxml_attribute(self,ftagname,num,stagname,attributeName):
        '''
        从all_case.xml文件中读取节点的属性值
        @param ftagname: 起始节点的名称, 如: project
        @param num: 取与起始节点相同的第num个节点
        @param stagname: 起始节点下的二级节点
        @param attributeName: 二级节点的属性名
        @return: 返回二级节点指定的属性值
        '''

```

```
'''

root = DOC.documentElement
message=root.getElementsByTagName(ftagname)[num]
    return message.getElementsByTagName(stagname)[0].getAttribute(attributeName)
```

python 对 xml 的读取操作，如果你不太明白，可以去自行学习相关的内容，要教程不讲解相关的操作。

6.6.4 具体的测试用例

上面的公用方法类创建以后，我们就可以着手编写具体的测试用例了。在 TestCases 文件夹下创建测试用例 TestCase_QT_Login.py，然后编写下面的测试步骤：

- (1) 打开众筹网。
- (2) 点击登录按钮，输入用户名和密码。
- (3) 验证是否登录成功，用户昵称是不是刚刚登录的账号。
- (4) 退出登录，关闭浏览器。

测试用例代码如下：

```
-*- coding: UTF-8 -*-
'''
Created on 2014-12-15
@author:songxianfeng
@version: v1.0
'''

import unittest
import time
#导入需要的公共函数类
from CommonFunction.WebDriverHelp import WebDriverHelp
from CommonFunction.DataOperations import DataOperations
from CommonFunction.QT_Operations import QT_Operations
class testcases_login(unittest.TestCase):
    '''
```



```

    登录检测
    '''

    def setUp(self):
        WebDriverHelp("open", "firefox", "local").setup("zhengshi") #打开浏览器，并打开众筹网

    def testlogin(self):
        #登录检测
        dataoper=DataOperations("TestCase_QT_Login.xml") #读取测试数据
        #登录
        QT_Operations().login(dataoper.readxml('login', 0, 'username'),
        dataoper.readxml('login', 0, 'password'))
        self.assertEqual(WebDriverHelp().gettext('byxpath', dataoper.readxml('login', 0,
        'checkpoint1')), dataoper.readxml('login', 0, 'value1')) #判断登录是否成功
        #退出
        QT_Operations().logout()
        time.sleep(5)
        self.assertEqual(WebDriverHelp().gettext('byclassname', dataoper.readxml('login', 0,
        'checkpoint2')), dataoper.readxml('login', 0, 'value2')) #判断退出是否成功

    def tearDown(self):
        WebDriverHelp().teardown() #关闭浏览器

if __name__ == "__main__":
    unittest.main()

```

经过我们上面的封装，现在具体的测试用例只是传参数，调用具体的函数，验证执行结果。是不是非常简单？有点儿像我们小时候玩积木，用一块块现成的积木堆积出魔幻的城堡。

6.6.5 测试用例的具体数据

由于我们把测试用例和测试数据完全分离开了，所以我们用和测试用例同名的文件名命名对应的测试数据文件。登录测试用例的数据文件是 TestCase_QT_Login.xml，具体内容如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<TestData>
    <login name="登录测试数据">

```

```
<username>183*****905</username>
<password>*****</password>
<checkpoint1>//div[@id='Js-header-loginBtn']/span/i[2]</checkpoint1>
<value1>潜龙0318</value1>
<checkpoint2>Js-showLogin</checkpoint2>
<value2>登录</value2>
</login>
</TestData>
```

我们将要验证的数据，获取验证元素的 Xpath 都写到这个里面，这样就算网页有所变化，我们只需要改这个数据文件中对应的 Xpath 即可，不需要更改测试用例。这样可以将网站变化影响到测试用例降到最低，同时也减少了我们维护测试用例的成本。

编写完上面所有的代码后，我们只要右击测试用例，选择 Run as->python run, 调试运行测试用例即可。

6.7 本章小结

本章我们讲解了如何从接触到一个网站开始，从零着手去编写自动化测试用例。通过上面各个方法的考虑，选择合适的框架，脚本语言，规划代码架构，编写公用函数，并以众筹网的登录注册为例，我们编写了具体的测试用例。通过这样的讲解，相信大家一定能编写测试用例了，要建立测试用例集，不过是按照先前我们选择的测试用例，全部转化成测试用例代码即可。第七章我们将讲述测试用例集的组织，以及将测试用例放到代码管理工具 **jenkins** 上实现自动化运行。从编写自动化测试用例，讲述到实现无人值守的回归测试用例的建立，希望你能学到点儿实用的东西。

第七章 自动化无人值守运行

在你根据手工测试用例完善自动化测试用例后，每个测试用例都调试通过了，说明我们已完成了对被测对象的测试用例覆盖。可是在平时的上线回归中，你不可能一个一个地手工执行测试用例的，我们希望的是运行一个命令或是执行一次菜单，测试用例会一个一个地去执行。所以本章开始，我们将讲解如何组织测试用例，以及利用 `jenkins` 来自动执行测试用例。

7.1 TestSuite 组织测试用例

平时我们编写测试用例的时候，都是继承 `unittest.TestCase` 类来编写测试用例的，重写了 `setUp()`，`tearDown()` 方法，并且定义以 `'test'` 开头的具体方法，来组织成一个个测试用例。而很多的测试用例文件，我们可以用 `unittest.TextTestRunner()` 来组织运行测试用例。而根据不同的测试需要，来编写不同的 `suite` 文件。

例如：我编写的 `TestSuite_Online_All.py` 文件：

```
# -*- coding: utf-8 -*-  
'''
```

```
Created on 2014-6-12
```

```

@author: songxianfeng
'''

import unittest
import sys
import os
sys.path.append("../")
sys.path.append(os.getcwd()+"/src/")
#引用测试用例文件
from TestCases.TestCase_QT_Login import testcases_login
from TestCases.TestCase_QT_Index import testcases_index
class testsuit_online_all():
    def test(self):
        if __name__ == "__main__":
            #1, 登录检测: testlogin
            #2, 首页检测: testindex
            #.....
            #构造测试集
            suite = unittest.TestSuite()
            suite.addTest(testcases_login('testlogin'))
            suite.addTest(testcases_index('testindex'))
            # 运行测试用例集
            runner = unittest.TextTestRunner()
            runner.run(suite)
if __name__ == "__main__":
    testsuit_online_all().test()

```

先引用测试用例文件中的测试类，如：testcases_index, 然后利用“测试类（测试函数）”的方法，将具体的测试函数添加 suite 中，然后利用 unittest.TextTestRunner() 的 run() 函数来运行测试用例集中的测试用例。

测试用例的运行

通过 testsuite 将需要的测试用例组织起来后，当需要执行这一系列的测试用例的时候，只需要执行 python TestSuite_Online_All.py 命令即可。或是右击 TestSuite_Online_All.py 文件，选择“Run As”—>“python run”

运行测试集文件即可。

为了方便管理，我们将所有的测试用例集文件放在 TestSuites 文件夹下，在以后配置 Jenkins 自动运行 Job 的时候就可以根据需要，进行不同的配置。

7.2 利用 Jenkins 来管理自动化测试用例

Jenkins 是基于 Java 开发的一种持续集成工具，用于监控持续重复的工作，功能包括：

- (1) 持续的软件版本发布/测试项目。
- (2) 监控外部调用执行的工作。

由于其是开源的，所以现在很多公司都用它来管理代码，当然要做到持续集成还是有很多工作要做的，我们只用它来调用我们的自动化测试用例。

Jenkins 基本配置

Jenkins 这个开源工具的安装及配置管理，在网上有很多相关的教程，如：<http://blog.csdn.net/wangmuming/article/details/22925127>。这个上面讲的就很详细，所以我们不在此详细讲述相关的操作。本节我们只讲解，如何把我们的自动化测试用例配置到 jenkins 上。

下面我们以一个实例来讲解：

- (1) 我们编写了测试用例文件以及 suite 文件，并将所有的文件放到

SVN: svn://svn.corp.ncfgroup.com/selenium_test/yyzhongchou 下

面。

- (2) 我们在要运行测试用例的服务器上安装并配置好了 jenkins,这个配置你可以在网上查找相关的方法自行安装配置。
- (3) 新建一个 job ,命名为 Zhongchou-web-test-testsuite, 然后打开这个 job,单击配置菜单。
- (4) 给我们这个 Job 添加描述, 允许执行项目的人员“启动安全项目”, 如图 7.2.1 所示:

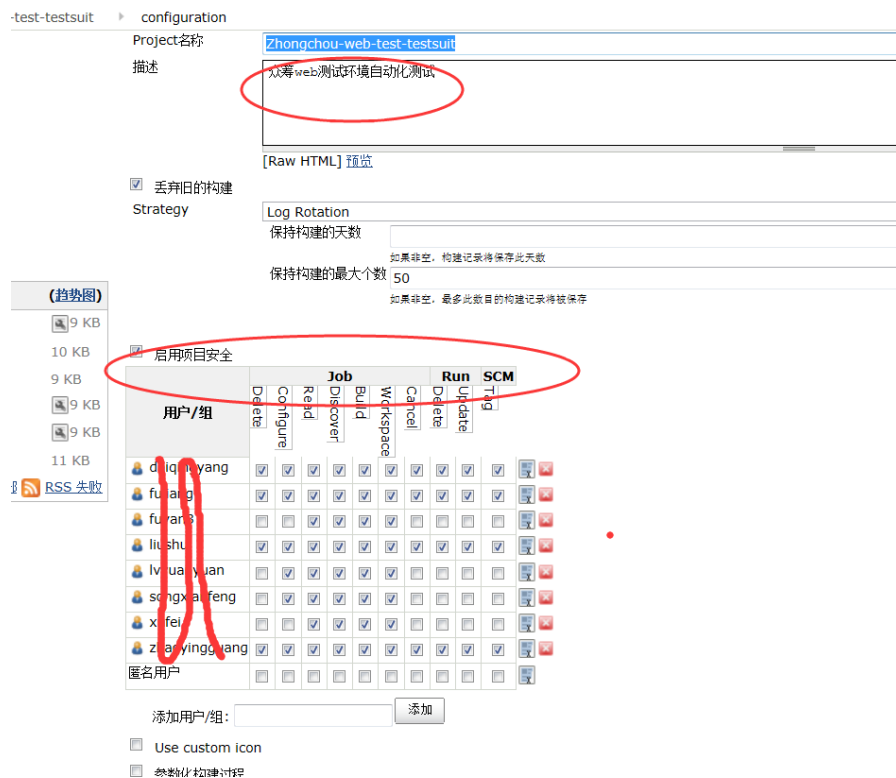


图 7.2.1 添加描述及启动项目安全

- (5) 对我们测试用例的源码进行管理, 根据我们源码管理的工具不同, 选择不同的方法。我们放的是 SVN, 所以选择 Subversion 项, 然后在后面填写我们的 SVN 地址, 如图 7.2.2 所示:



图 7.2.2 源码管理

(6) 添加构建命令。当我们执行 Job 的时候，怎么运行我们的测试用例呢，在这一步中，我们要添加上执行我们测试用例的命令：
`python .\src\testSuit\testSuit_zhongchou_all.py`，这样运行的时候就会调用 `testsuite`，执行测试集中的所有测试用例了，如图 7.2.3 所示：

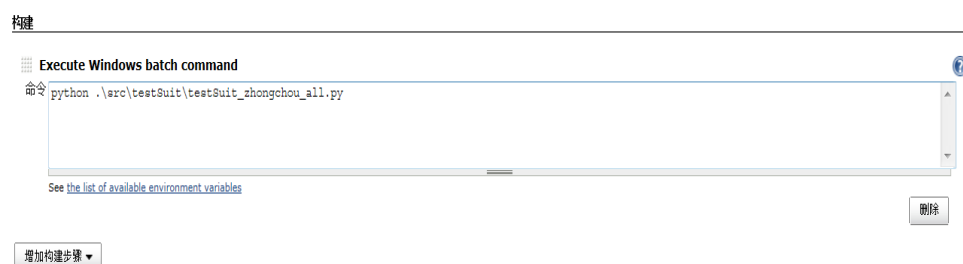


图 7.2.3 添加构建命令

(7) 构建 Job。填写完成之后，点击保存按钮，返回 job 页。当我们想执行测试用例的时候，点击页面左侧的“立即构建”，即可运行测试用例。如图 7.2.4 所示：



图 7.2.4 构建 Job

(8) 查看执行结果。要想查看测试用例执行的结果，我们可以单击左侧的“Build History”下面的构建项目，在打开的页面中单击左侧的“控制台输出”，就能看出用例执行的结果了。这样就可以定位错误，进行调试。如图 7.2.5 所示：



图 7.2.5 查看控制台输出

7.3 Jenkins 高级配置

经过上面的配置，我们已实现了把自动化测试用例接入到 jenkins

中，虽然比手工执行高端一点儿，但这不是我们想要的结果。我们想要自动化执行测试用例，监控执行结果，如果出错给我们发邮件或是短信，这样我们才能及时处理。所以我们要进一步去配置或是优化测试用例。

7.3.1 自动化执行测试用例

Jenkins 是用来管理和配置持续化集成的，在持续化集成中，自动运行脚本语言是最基本的功能。下面我们就配置一下：

(1) 定时执行测试用例

如图 7.3.1.1 所示，选择构建触发器中选择 build periodically，配置执行的方法：

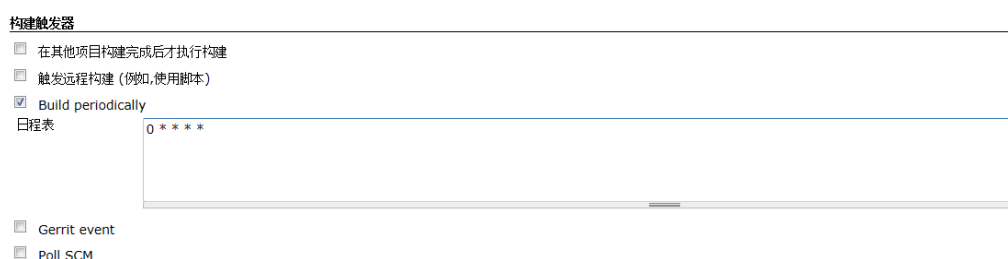


图 7.3.1.1 配置定时执行

选择 Build periodically，在 Schedule 中填写 0 * * * *。

第一个参数代表的是分钟 minute，取值 0~59；

第二个参数代表的是小时 hour，取值 0~23；

第三个参数代表的是天 day，取值 1~31；

第四个参数代表的是月 month，取值 1~12；

最后一个参数代表的是星期 week，取值 0~7，0 和 7 都是表示星期天。

所以 0 * * * * 表示的就是每个小时的第 0 分钟执行一次构建。

(2) 触发式执行测试用例

如图 7.3.1.2 所示，选择构建触发器中选择“在其他项目完成后构建”或是“触发无私构建”，然后写上触发的项目了。如在开发上传完代码，完成构建后，自动触发回归测试自动用例来进行回归测试。

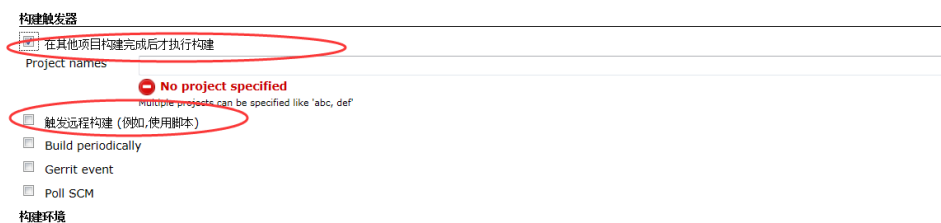


图 7.3.1.2 触发式构建

7.3.2 程序执行失败邮件通知

当测试用例实现自动化执行后，我们需要知道执行的结果。一般测试用例执行成功后，我们并不太关心。但是如果执行失败了，我们需要知道为什么失败，是被测试对象存在 bug,还是我们的自动化测试代码有问题了？然后去排查，找出问题所在，这也是自动化测试的意义。

而用例执行失败后发邮件通知，这个功能是jenkins自带的功能，我们只需要对其进行如下配置即可。

(1) 打开要进行配置的 job, 单击“配置”项。

(2) 在“构建后操作”下单击“Add post build actions”,然后选

择“E-mail Notification”项。

(3) 然后在 Recipients 后面的框中填写要接收错误报告的邮箱地址，以空格分隔多个地址。如图 7.3.2.1 所示：

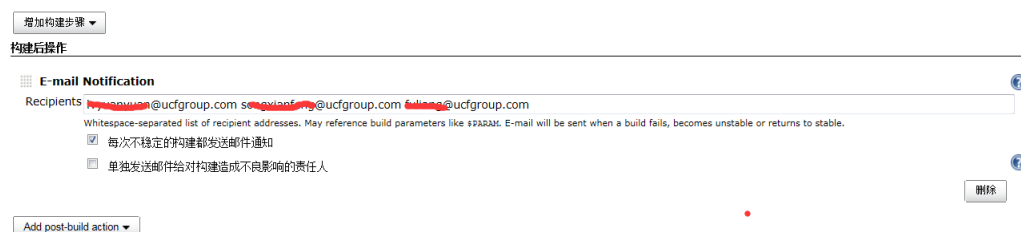


图 7.3.2.1 填写接收报告邮件地址

(4) 在下面勾选“每次不稳定的构建都发送邮件通知”复选框，然后单击保存，这样在测试用例执行失败或者 job 构建失败后就会发邮件给所填写的邮件地址了。

7.3.3 测试执行失败短信通知

现在手机使用这么发达，邮件通知感觉到不那么及时了，如果不时常查看一下邮箱，即使是 Jenkins 给我们发了邮件我们也不能及时收到。所以如果自动化测试用例执行失败了，能发送短信，这不是很好吗？

遗憾的是，Jenkins 没有这个功能，如果想实现这个目标，我们只有自己努力了！下面我们分析一下思路：

(1) 我们要根据执行结果来分析出测试用例执行成功了或是失败了，这个虽然 Jenkins 会自己分析，可是我们没有办法获取它分析

的结果。所以我们要保存执行结果。

(2) 解析保存的执行结果，如果失败了，解析出哪些测试用例失败了，把测试用例名字保存下来，以便作为我们失败通知短信的内容。

(3) 需要一个发短信给固定手机的方法，我的办法是我们公司有一个公共短信平台，申请了相关的权限后就可以直接调用发短信接口向任何手机号发短信。读者要想达到这个目的，请自行想办法，因为没有免费的短信接口。

(4) 上面三方面我们需要写一个脚本来做这些事儿，我们暂时命名为 **TestGetResult.php**，因为调用接口 **php** 最方便。脚本都是在 **linux** 下执行的，所有 **python,php** 都不会相互影响的。脚本内容涉及公司接口，就不给大家展示了。

(5) 如果测试用例执行成功，则直接执行 **Jenkins** 返回成功，如果失败，则调用此脚本发送短信，并同时调用 **Jenkins** 发送邮件。所以我们的脚本执行不能影响 **Jenkins** 对执行结果的判断。

(6) 为了达到这个目的，我们需要在“构建”—>“Excute Shell”下添加脚本语言，如下所示：

```
01  #!/bin/sh
02  python .\src\testSuit\testSuit_zhongchou_all.py >./Result/WebResult.log
03  if [ $? -ne 0 ];
04  then
05  php TestGetResult.php
06  exit 1
07  else
08  exit 0
09  fi
```

脚本讲解：

A, 第 02 行我们是调用执行 suite 文件，执行测试用例，并将结果保存到 ./Result/WebResult.log 文件中。

B, 第 03 行判断第一行执行的结果，如果执行结果不等于 0，说明测试用例执行失败，然后调用 05 行我们处理测试结果，发短信的脚本。

C, 第 06 行很关键，当测试用例执行失败后，第 02 行会返回一个非 0 的结果，此时我们调用 05 行执行。05 行执行成功后会返回 0，此时 jenkins 接到的结果代码是 0，会把执行结果置成成功，并且不会发邮件。所以 06 行，我们人为的返回一个非零的代码给 Jenkins，为了不影响 Jenkins 的结果。

D, 第 08 行和 06 行的目的是一样的，当 03 行判断结果为失败时，返回代码是非零的，如果不在 08 行添加一个人为地返回 0 的语句，测试用例执行成功时，Jenkins 根据返回码也会认为是失败。为了不影响 Jenkins 原来的判断结果，我们添加了 06 和 08 两个强制返回语句。

至此，我们完成了 Jenkins 的配置，这些儿配置可以满足我们对自动化测试用例的失败监控。当然，如果你们有其他需要，可以去网上学习 Jenkins 的其他配置及插件的使用，学无止境嘛！

7.4 自动化测试用例报告

Jenkins 可以随时监控自动化测试用例的执行，如果有错误就会

及时通知我们。如果我们想向我们的领导展示一下我们自动化的成果，这时一个漂亮的执行报告就是非常必要的了。虽然 Python 不能生成像 testng 那样漂亮的报告，不过也是可以生成清晰的报告的。

Python+Webdriver 生成报告的方法如下：

(1) 下载 HTMLTestRunner.py 文件：地址

<http://tungwaiyip.info/software/HTMLTestRunner.html>

(2) 将该文件保存在 python 安装路径下的 lib 文件夹中。在文件中能 import HTMLTestRunner 成功，即配置成功。

注：如果失败，在项目中新建一个这样的文件也是可以的，只要达到能引入和使用就行。

(3) 修改 TestSuite 文件，添加生成测试报告的语句，如下所示：

```
# -*- coding: utf-8 -*-
'''
Created on 2014-6-12

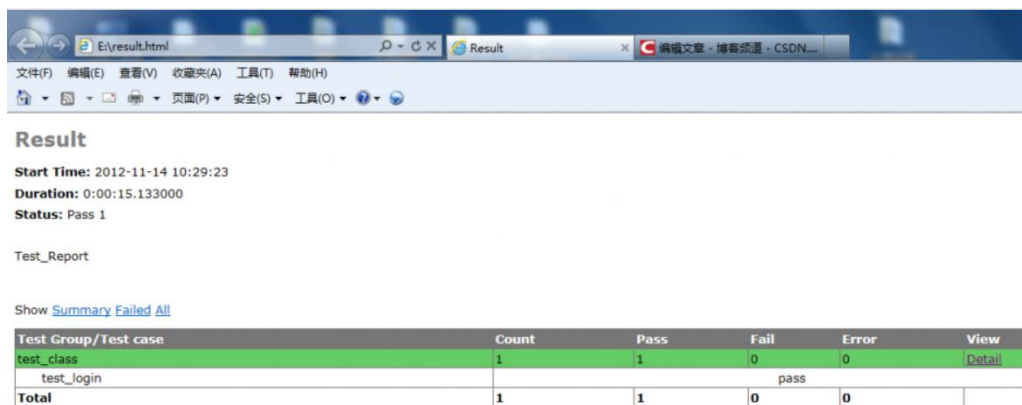
@author: songxianfeng
'''
import unittest
import sys
import os
sys.path.append("../")
sys.path.append(os.getcwd()+"/src/")
#引用测试用例文件
from TestCases.TestCase_QT_Login import testcases_login
from TestCases.TestCase_QT_Index import testcases_index
class testsuit_online_all():
    def test(self):
        if __name__ == "__main__":
            #1,登录检测: testlogin
            #2,首页检测: testindex
            #.....
            #构造测试集
            suite = unittest.TestSuite()
```

```

suite.addTest(testcases_login('testlogin'))
suite.addTest(testcases_index ('testindex'))
# 运行测试用例集
filename="./Reoport/TestReport.html"#测试报告路径
fp=file(filename,'wb')
runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='Result',description='
Test_Report')#添加测试报告
runner.run(suite)
if __name__ == "__main__":
    testsuit_online_all().test()

```

(4) 当执行完测试用例后，去打开./Report/TestReport.html 文件，就可以看到，类似图 7.4.1 样式的报告。现在你就可以将此报告发给你们老大，以数据形式展示你的成绩了。



Result

Start Time: 2012-11-14 10:29:23
Duration: 0:00:15.133000
Status: Pass 1

Test_Report

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
test_class	1	1	0	0	Detail
test_login				pass	
Total	1	1	0	0	

图 7.4.1 测试报告

7.5 本章小结

本章我们讲述了如何使用 python unittest 的 testsuite 组织测试用例，如果将自动化测试用例配置到 Jenkins 中，配置测试用例自动执行，执行失败后发邮件和短信通知相关人员，以及如何生成测试报告等内容。这也是我们自动化测试的具体使用，展示我们自动如何牛的地方。

当然不管你用什么来做自动化测试，也不管你测试的是网页，接口还是手机 App，如果不拿出点儿数据来展示给老大们，他们是不会

支持你的。像用例覆盖率啊，执行时间啊，发现 Bug 的数据啊，节省人力成本什么的，本章节的内容是你的加分项目，平时的编写自动化测试用例的努力，全靠此章来展示了！

第八章 影响自动化实施的非技术因素

通过前七章的讲解，我们了解了什么是自动化，自动化如何实施，而后又分章节讲解了如何实施自动化测试，自动化测试在 Jenkins 上的配置及报告的生成。经过这七章的学习，读者应该完全可以编写自动的自动化测试用例，实现对被测试项目的无人值守的自动化运行。本章我们分析一下影响自动化测试实施的非技术因素，并作为本教程的结束章节。

8.1 非技术性因素的影响

初次接触自动化测试的同学，一般都会把注意力放到测试框架，编写测试用例的脚本语言，测试用例的运行环境，以及多线程或是多线程执行测试用例的问题。但是真正在公司实施自动化测试的时候，你会发现能不能完全实施自动化测试用例，很大程度是不是你技术行不行，还有众多因素在影响着你。

影响自动化测试实施的非技术因素：

- (1) 被测试对象反复改版。这个是影响最大的因素，当我们接到编写自动化测试用例后，立马投入测试用例的编写，调试用例，接入 Jenkins，生成漂亮的测试报告。结果被测试对象大改版，你编写的自动化测试用例几乎要重新编写。如此反复几次，你就会感觉到自己一直在做无用功。
- (2) 自动化测试人员的流失。在公司决定实施自动化测试的时候，招进了大批自动化测试的牛人，然后就是创建自动化测试小组，划分各自的职能，编写相应的模块。可是后来，相关人员离职了，他编写的那一部分没有人管了，只好转交给他人，他人要花费很多时间来看代码。最糟糕的时候，离职人员编码习惯不好，没有注释，这时交接人员会疯掉的。
- (3) 领导的不支持。领导 A 比较注重自动化测试，然后给你了大量的时间让你去写自动化测试用例，然后运行起来。后来公司职能调整，你这一块归领导 B 来管理，他不注重自动化测试，给你分了很多其他的任务，自动化测试就没有时间和精力去维护

和执行了。

- (4) 公司不重视。就算你的直属领导很重视自动化测试，可是如果公司不注重的话，也是很难实施的。申请自动化测试用例执行的服务器，申请不下来。其他的相关资源也很难申请，你总不至于用自己的机器来执行自动化测试用例吧？执行过程中你不能做其他工作，会不会有人说你不工作呢！！
- (5) 其他不确定的因素。在《自动化测试最佳实践》上面讲到很多国内外自动化实施成功失败的案例，各种各样的因素都会影响到自动化测试的实施。

8.2 学习自动化测试的方法

现在我们讨论一下自动化测试用例的学习方法。我是从研二实习的时候开始接触自动化测试的，到现在有五六年了吧，页面自动化，接口自动化，手机 App 自动化，虽然学习的不算精通，也都鼓捣过。在学习的过程中遇到了不少困难，也积累了不少经验，现在和大家分享一下：

- (1) 勤动手实践。学习任何一种编程语言都是一样的，不是说你看了几本相关的书，就掌握了一门编程语言，一定要去动手编写和调试。就算你照着书中的例子输入到电脑，也会出现错误的，书中例子的错误，少输入符号或是空格的错误，如果不去动手，就不能有所感悟的。
- (2) 大胆尝试。自动化测试用例编写的时候，首先要对元素进行定

位，定位方法很多。Name, id, css 如果定位不到，还有 Xpath 的嘛，要多尝试各种定位方法，可以提高测试用例的健壮性和执行效率。

- (3) 学会积累。在编写自动化测试的过程中，你可能会遇到各种各样的问题，在网上查询或是经过尝试后，问题解决了，这个时候你要对问题写一下总结。好记性不如烂笔头嘛，以后再遇到的时候就不用这么费劲去查找了。
- (4) 善于沟通。开发在编写网站的时候，由于不好的编程习惯，造成页面元素随意编写，没有固定的属性。这样的情况吧，会增加我们自动化测试用例编写的难度，而且如果以后要统计页面数据的时候，也很难统计。我们要善于和他们沟通，帮其养成良好的编程习惯。

8.3 本章小结

作为本教程的最后章节，我们分析了影响自动化测试实施的非技术因素和自动化测试学习中的一些方法问题。由于本人能力有限，这个系列的教程中难免存在着问题和不足之处，希望大家在学习中遇到问题及时沟通。同时也希望我写的东西能给你的自动化学习带来点儿帮助，这是我最欣慰的地方。

最后祝大家学有所成，开心工作!!