目录

1、 自动化测试环境搭建

1.1 为什么选择 Python

什么是 python，引用 python 官方的说法就是“一种解释型的、面向对象、带有动态语义的高级程序设计语言”，对于很多测试人员来说，这段话包含了很多术语，而测试人员大多是希望利用编程语言来帮助他实现自动化的测试，而不需要花费大量的精力来学习一门编程语言，所以在本文档中不会过多强调 python 的内容，只是通过 python 配合 selenium 实现自动化的测试。

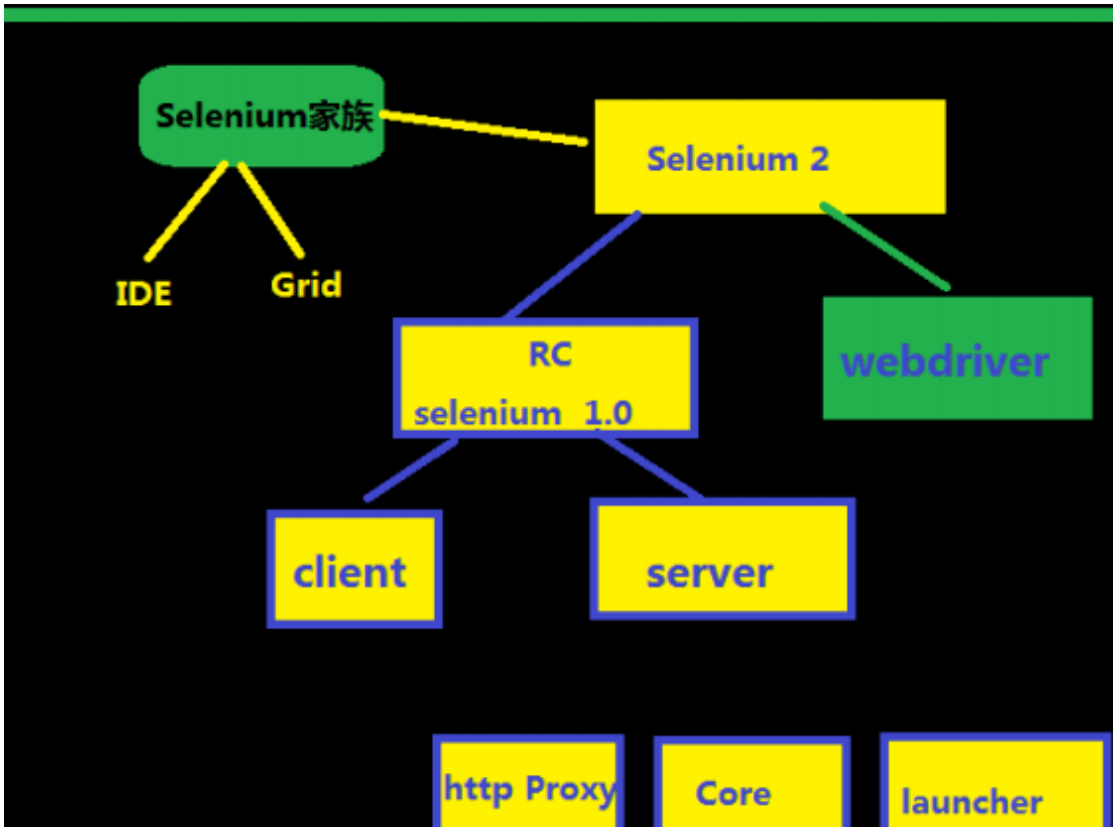
为什么选择 python，python 的优势在于是一种解释型语言，学习容易，使用范围广泛。其他语言学习起来，太复杂，过一段时间后，就会忘记。而 python 是目前测试推广最多的语言，翻翻各大招聘公司，测试要求会 python 的比比皆是，而且易学易用。

1.2 Selenium 简介

Selenium 是 Thought Works 公司开发的一套基于 web 应用的自动化测试工具，直接运行在浏览器中，模拟用户操作。它可以被用于单元测试、集成测试、回归测试、系统测试、冒烟测试、验收测试，并且可以运行在各种浏览器和操作系统上。

Selenium 分为 1.0 和 2.0 两个大版本，1.0 主要包含 ide、core 和 rc 三大部分。2.0 集成了 1.0 的功能，同时集成了 webdriver，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。

Selenium 主要结构：



IDE：可以通过 IDE 完成测试过程的录制和回放。主要用来给初学者了解 selenium，但不适合直接作为日常自动化的测试。

Grid：是 selenium 部署、测试及执行。

RC：selenium Remote Control,一个代理与控制器。

Core：selenium 的测试机制核心部分，包含测试用例集的执行，断言，由 js 代码组成，支持夸平台运行。

Webdriver 结构：

selenium 分为四层：

Selenium test：业务脚本层，支持各种编程语言脚本 java、C#、Ruby、python、js 等。

Webdriver：实现模拟用户在浏览器中的各种操作。

浏览器：几乎支持所有浏览器。

业务层：即被测对象。

Selenium 的目录结构：

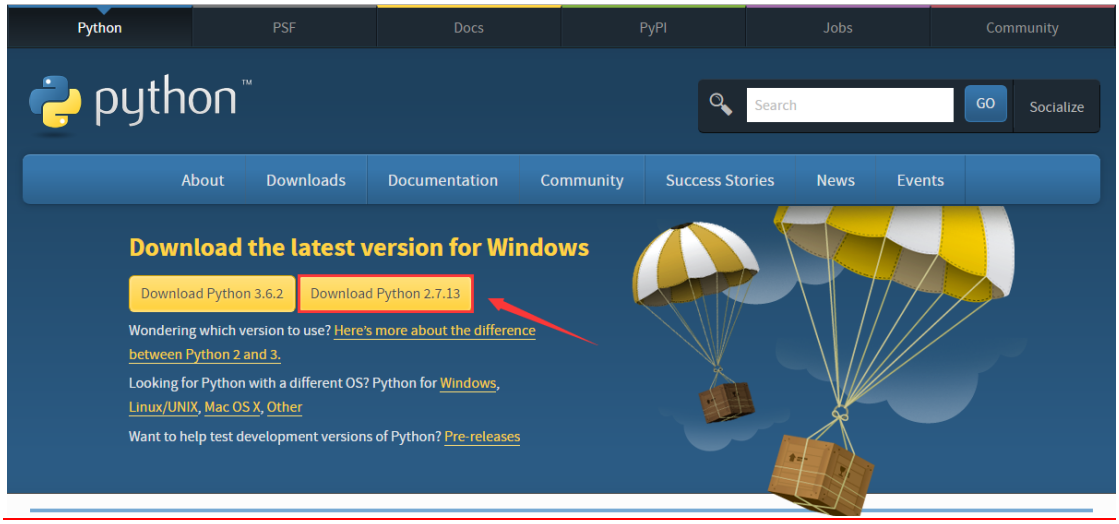
Selenium 异常处理部分：

1.3 Python 安装

搭建 python 环境：

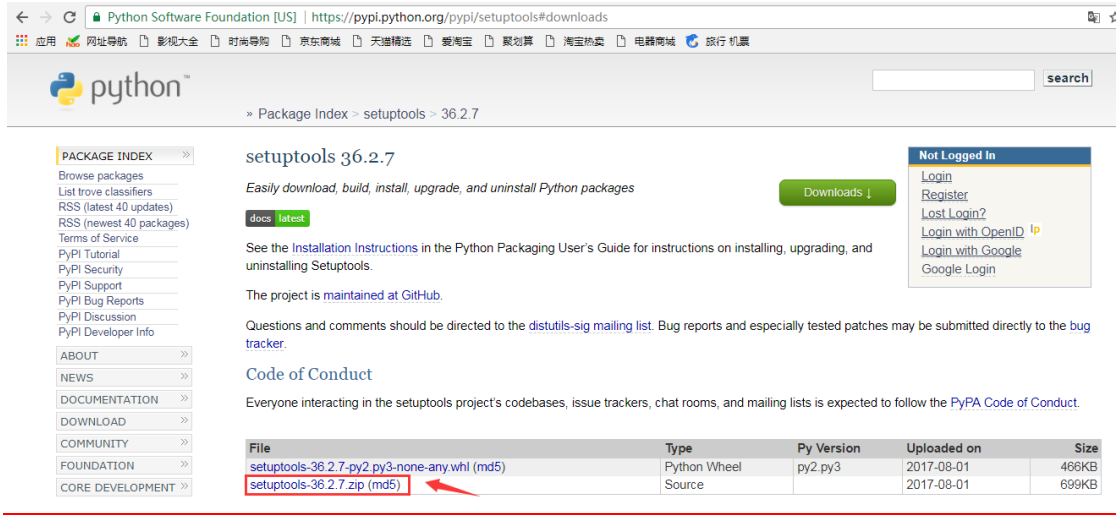
1、 下载 python

https://python.org/getit



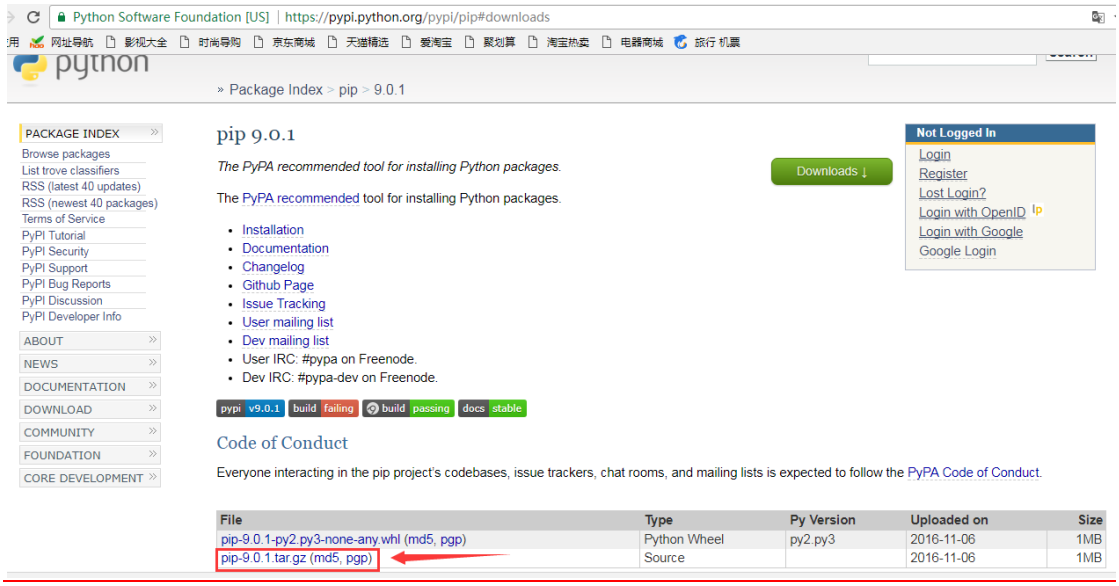
2、 下载 python 的基础包工具 setuptools

https://pypi.python.org/pypi/setuptools



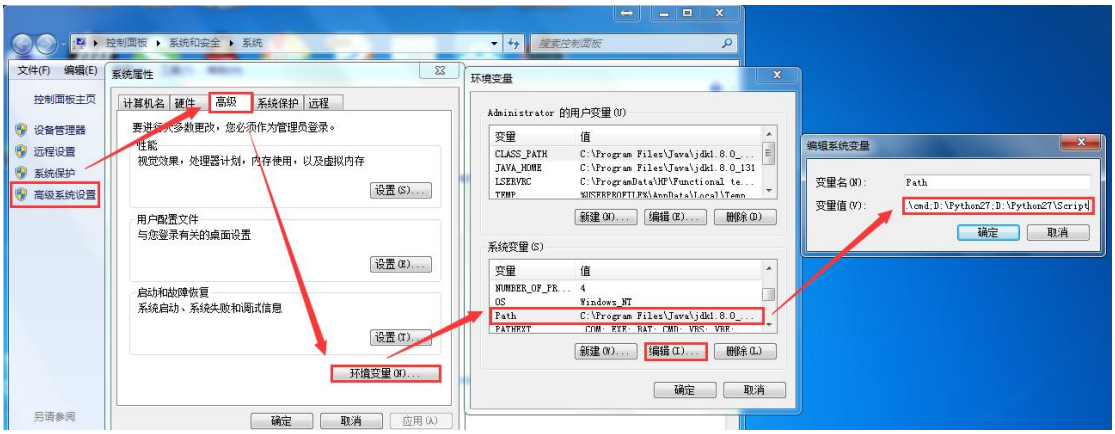
3、 下载 python 安装包管理工具 pip

https://pypi.python.org/pypi/pip



安装步骤

Python 安装，下载自己系统对应的 python 版本，32 位的下载对应 32 位安装包，64 位下载对应版本。双击安装程序。默认安装路径为 c:\python27。Python 安装完成后，需要将 python 的安装路径加入到 path 变量中。



；C:\python27; C:\Python27\Scripts

检查 python 安装是否成功：

在 cmd 中执行 python，如果出现下列界面，则表示 python 安装成功。

```
C:\Users\Administrator>python
Python 2.7.10 (default, May 23 2015, 09:44:00) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Setuptools 的安装相同，默认会找到 python 的安装路径，将安装在 c:\python27\Lib\site-packages

将 Setuptools 文件包进行解压，解压到 D:\setuptools-36.2.7

打开 cmd，执行 cd D:\setuptools-36.2.7

执行 python setup.py install 进行安装。

1、 安装 pip，将 pip 的包解压，解压后通过 cmd 进入该目录，执行 python setup.py install.

例如：

pip 安装包在 D:\下，解压后为 D:\pip-9.0.1

打开 cmd，执行 cd D:\pip-9.0.1，再执行 python setup.py install 进行安装。

第二种安装方式为：

打开 cmd 窗口，执行 easy_install pip。

1.4 selenium 环境搭建

selenium 下载：

https://pypi.python.org/pypi/selenium

selenium 安装，将安装包下载后，解压，解压完成后，通过 cmd 进入解压后的目录，执行 python setup.py install。

如果你的机器没有联网，则使用上述方法，如果已经联网，则可以使用 pip install selenium。

```
C:\Users\Administrator>python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> webdriver.Firefox()
```

检查 selenium 是否安装成功：

```
C:\Users\Administrator>python
Python 2.7.10 (default, May 23 2015, 09:44:00) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> driver=webdriver.Firefox()
>>> driver.get('https://www.baidu.com')
>>>
```

执行上述命令后，如果已经安装成功，则会打开一个 firefox 浏览器界面。

命令解释：

Python 进入 python 开发界面。

from selenium import webdriver:加载 selenium 中的 webdriver

driver=webdriver.Firefox():打开一个 firefox 浏览器，并将操作浏览器的句柄赋给 driver 变量。

driver.get('https://www.baidu.com')：打开浏览器后，在浏览器中输入百度 url 地址，转到百度页面。

接下来开始介绍 webdriver 对浏览器操作的 API。

2、 页面元素定位

通过自动化操作 web 页面，首先要解决的问题就是定位到要操作的对象，比如要模拟用户在页面上的输入框中输入一段字符串，那就必须得定位到这个输入框，然后才能输入。这些对象也可以称为页面的元素，每个元素都会有很多属性，可以根据不同属性来定位元素。

Web 中常见元素有文本输入框、单选框、复选框、按钮、下拉框等，每个元素又提供了很多属性，比如 id、name、文本等。

图 1：web 页面元素获取

Webdriver 提供了这些元素的定位方法，主要有以下几种：

- id
- name
- class name
- link text
- partiallink text
- tag name

- xpath
- cssSelector

3.1 id 定位

1、 打开 LMD 系统后台,输入邮箱地址及密码。

图 2：页面元素 id

脚本如下：

```
#coding=utf-8
from selenium import webdriver
driver=webdriver.Firefox()
driver.get('http://www.chuangyijia.com/admin/login')
driver.find_element_by_id('email').send_keys('lib@163.com')
driver.find_element_by_id('password').send_keys('password')
```

#coding=utf-8

该句表示编码格式，主要为了防止后期执行乱码。

from selenium import webdriver

从 selenium 的包中导入 webdriver 的函数。

driver=webdriver.Firefox()

通过 webdriver 的 firefox 打开一个 firefox 浏览器，firefox 也可以是其他浏览器，比如 IE，chrome 等，然后将打开后的浏览器操作句柄赋给 driver 变量，变量名可以自己定义，注意，后面对浏览器的操作都会用到这个对象。

driver.get('http://www.chuangyijia.com/admin/login')

通过 webdriver 的 get 方法让浏览器重定向到 LMD 的 url 地址。

driver.find_element_by_id('email').send_keys('lib@163.com') 和

driver.find_element_by_id('password').send_keys('password')

这两句的定位方式相同，从图 2 中可以看到，邮箱和密码都是 input，并且都有很多属性，在这些属性中就有 id，邮箱的 id 为 email，password 的 id 为 password，通过 webdriver 的 find_element_by_id()方法定位邮箱和密码输入框。Send_keys 是往输入框中输入值。

PS：查看页面中元素属性，可以通过 firefox 浏览器的查看元素方式查看，在页面中，选择你要查看的元素，右键找到“查看元素”，则可以在浏览器下方看到该元素的属性信息。如图 3

图 3：页面元素定位

3.2 name 定位

除了上述的 id 方法，还可以通过 webdriver 的 find_element_by_name()来定位邮箱地址的输入。例如：

```
driver.find_element_by_name('email').send_keys('test@163.com')
```

执行该条命令，也可以在邮箱输入框中输入邮箱地址，password 输入框一样，主要是观察该元素是否有 name 属性，及 name 属性的值。

PS：，如果该页面中存在多个元素都包含 name 属性，并且属性值相同，则该方法无法定位到该元素。

3.3 tag name 定位

tag name，在一个页面中，每个元素都是一种标签，列入上面用到的邮箱地址和密码都是 input，那么他们的 tag name 就是 input，如果想要使用 tag name 来定位一个元素，最好确定该页面只有这一个标签，如果是多个，则会定位出一组元素，那么 webdriver 将不知道你要操作的究竟是哪个，所以这种状况下就不能使用 tag name 来定位了。在这个登陆界面中只有一个登陆按钮，标签为 button，这个标签在这个页面中只有一个，我们可以用 tag name 来定位该元素，代码如下：

```
driver.find_element_by_tag_name('button').click()
```

注：click()点击的意思

3.4 Class name 定位

在查看元素中，除了看到了 tag name、id、name 等属性外，还有其他属性，比如 class 属性，接下来，我们通过 class 来定位要操作的对象。

在登陆成功以后，页面上的左上角有一个 LMD 系统的图片头，点击该图片，可以回到管理界面的首页，在这里我们通过 class 来定位，从下图中，可以看到它的标签为 span，有一个 class 属性，属性值为 first，定位代码如下：


```
driver.find_element_by_class_name('first').click()
```

3.5 css 定位

css 是 Cascading Style Sheet 的缩写，是用于（增强）控制网页样式并允许将样式信息与网页内容分离的一种标记性语言。Css 本身是使用选择器来定位元素，并对 html 中元素进行格式化。Selenium 也可以利用 css 的选择器进行选择被操作元素，操作相对上面几种定位方式而言更加灵活。

例如：定位之前登陆页面的邮箱和密码，使用 css 来定位，代码如下：

```
<form id="login" method="post" action="/admin/login">
<label>邮箱</label>
<input id="email" class="span12" type="text" name="email"></input>
<label>密码</label>
<input id="password" class="span12" type="password" name="password"></input>
<button class="btn btn-primary pull-right" type="submit">登录</button>
```

```
driver.find_element_by_css_selector( '#emai' ).send_keys( 'lib@163.com
```

其中#email 为 css 选择器选择方式，以#来标示，说明在这里 css 通过该元素的 id 来选择的。

通过 css 来定位登陆页面中的登陆按钮，代码如下：

```
<button class="btn btn-primary pull-right" type="submit">登录</button>
```

```
driver.find_element_by_css_selector('.btn').click()
```

在 find_element_by_css_selector()函数来实现定位，将定位元素的 css 语句放入参数位置。

通过 css 选择器方式定位邮箱输入框：

```
<label>邮箱</label>
<input id="email" class="span12" type="text" name="email"></input>
```

```
driver.find_element_by_css_selector('input[name="email"]').clear()
```

通过 css 方式定位登陆按钮

```
<button class="btn btn-primary pull-right" type="submit">登录</button>
```

```
driver.find_element_by_css_selector('button.btn').click()
```

css 的定位方式很多，也狠灵活，这里介绍了几种常见操作。如果对 css 选择器定位不是很理解，可以借助 firefox 浏览器的功能，完成 css 定位，例如在登陆界面中，不知道如何使用 css 定位登陆按钮，则可以参考如下操作：

在登陆页面，鼠标选中登陆按钮，右键查看元素，在下方开发者工具栏中，鼠标选中登陆的代码，右键，选择复制唯一选择器，然后将复制的内容粘贴到 find_element_by_css_selector()的方法中即可。

也可以参考 css 样式的选择器方法。

3.6 xpath 定位

什么是 xpath，是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。XPath 是 W3C XSLT 标准的主要元素，并且 XQuery 和 XPointer 都构建于 XPath 表达之上。因此，对 XPath 的理解是很多高级 XML 应用的基础。

因为 html 可以看作另一种形式的 xml，所以 selenium 也可以通过 xpath 定位页面元素，xpath 扩展了常规 web 元素属性的定位方式，提供了更多的操作方式。

通过 xpath 方式定位登陆页面的邮箱和密码：

```
driver.find_element_by_xpath('//input[@id="email"]').send_keys('lib@163.com')
定位到当前页面的 input 属性，在获取该类属性中 id 为 email 的。
driver.find_element_by_xpath('//input[@id="password"]').send_keys('123456')
```

通过 xpath 的层级定位方式定位登陆页面的登陆按钮：

```
driver.find_element_by_xpath('//form[@id="login"]/button').click()
先定位到 button 的上级属性，button 是属于 form 元素的下级元素，定位到 id 为 login 的 form，然后查找 form 下的 button，在这个页面中，id 为 login 的 form 下只有 1 个 button，所以这里这么写是可以的，如果这里 form 下有多个 button，则需要其他方式了。
```

通过 xpath 定位时，如果通过上级元素来定位它的子元素，而子元素又有多个标签是相同的，就像上面的案例中，如果有 form 中有多个 button 标签，则会出错。

下面通过 xpath 的层级定位方式来定位邮箱和密码，邮箱和密码的标签都是 input，他们的上级标签为 form，那么先定位上级的 form，通过//form[@id="login"]来定位 form。如果此时通过//form[@id="login"/input]来定位邮箱或密码，则会出现问题，webdriver 会发现该 form 下有多个 input，因此不知道用户究竟要操作哪个，则可以按照下列方法定位，邮箱和密码标签都是 input，邮箱为第一个 input，密码为第二个 input，则代码如下：

```
driver.find_element_by_xpath('//form[@id="login"]/input[1]').send_keys('lib@163.com')
定位 id 为 login 的 form，并操作该 form 下的第一个 input，即邮箱地址的输入框
driver.find_element_by_xpath('//form[@id="login"]/input[2]').send_keys('lib@163.com')
定位 id 为 login 的 form，并操作该 form 下的第二个 input，即密码地址的输入框
```

Xpath 中除了可以用 id 之外，也可以使用其他属性，例如：

```
driver.find_element_by_xpath('//input[@name="email"]').send_keys('lib@163.com')
这里是通过 xpath，定位 name 为 email 的输入框。
```

利用 xpath，通过页面元素上的文本来定位操作对象。

```
driver.find_element_by_xpath('//button[contains(text(),"登录")]').click()
```

```
定位元素标签为 button，并且该标签上的文字为“登录”.
```

3.7 Link text 定位

在页面定位元素时，有时候定位对象不是一个文本框，也不是按钮，而是一个超链接此时我们可以通过 link text 方式定位对象。

例如：在 LMD 系统中，登录成功之后，点击待审核创意后，会出现项目管理，以及项目管理中的子功能列表，而列表是通过超链接方式实现的，想要点击预售管理时，通过文字定位无疑是最容易看懂该步骤在做什么，代码如下

```
from selenium import webdriver
导入 selenium 的 webdriver 模块
driver=webdriver.Firefox()
打开 firefox 浏览器
driver.get('http://www.chuangyijia.com/admin/login')
打开 LMD 后台登陆页面
sleep(2)
等待 2 秒
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys('lib@163.com')
输入登陆的邮箱地址
driver.find_element_by_xpath('//form[@id="login"]/input[2]').
send_keys('12345678')
输入登陆密码
driver.find_element_by_xpath('//form[@id="login"]/button').cl
ick()
点击登陆按钮
driver.find_element_by_css_selector('#dashboard-menu >
li:nth-child(2) > a:nth-child(1)').click()
这里是通过 css 方式定位，点击待审核创意
driver.find_element_by_link_text('预售管理').click()
点击待审核创意后，页面会出现图 4 的页面，这里是点滴子功能列表
中的预售管理。
                    图 4
```

3.8 partialinktext 定位

在页面通过文本定位超链接时，有些链接文本很长，selenium 的 webdriver 提供了可以通过匹配链接中的部分文本来定位该元素，则通过 find_element_by_partialink_link_text()方法来定位该元素。

例如：在 LMD 系统登陆后，点击待审核创意后，可以看到项目管理，及项目管理的子功能，在子功能中，想点击预售订单管理，用 link_text 方法时，需要将所有文本都写上去，而 partialink_link_text()时，可以只写部分文本，代码如下：

```
driver.find_element_by_partial_link_text('售订单').click()
在点击待审核创意之后，会显示项目管理及项目管理子功能，在此页面中，只有预售订单管理 的文本中包含“售订单”字符，可以用 partial_link_text 方法来定位该元素。
```

3.9 check box 定位

在 web 页面的元素中，还会有单选框和复选框的操作。下列代码是对百度的设置中的搜索设置页面的操作，将“是否希望在搜索时显示搜索框提示”设置为不显示，下面的代码稍微复杂了一些，牵扯到还没有讲到的内容，这里先简单介绍下：

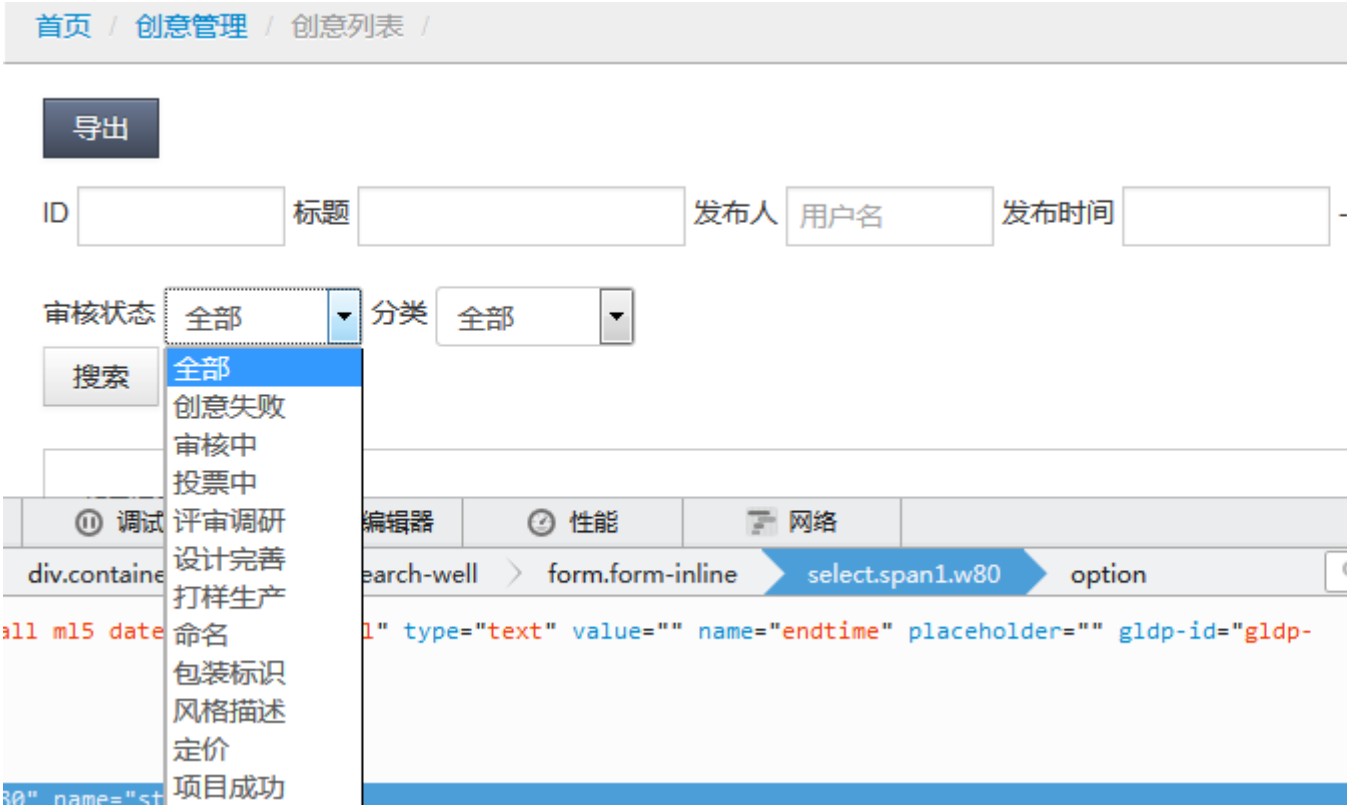
```
from selenium import webdriver
导入 webdriver
from selenium.webdriver.common.action_chains import
ActionChains
```

```
因为百度的设置页面比较特殊，需要将鼠标放到设置上，然后才会显示下拉框，才能从下拉框中选择搜索设置，所以需要用到鼠标的操作，这里就需要导入鼠标操作的模块
driver=webdriver.Firefox()
driver.get('https://www.baidu.com')
打开百度页面
seting=driver.find_element_by_link_text('设置')
找到设置所在的位置
ActionChains(driver).move_to_element(seting).perform()
将鼠标移动到设置上
driver.find_element_by_link_text('搜索设置').click()
点击搜索设置
sleep(1)
driver.find_element_by_xpath('//label[contains(text(),"不显示")]').click()
点击搜索设置中的不显示的单选框

也可以使用 xpath 的方式定位到单选框，直接点击
driver.find_element_by_xpath('//label[@for="s1_2"]').click()
```

3.10 下拉框定位

在 web 页面中，进行各种操作时，无法避免要对下拉框进行操作，下面对 LMD 后台管理中，对创意进行查询时，查询条件就是通过下拉框来操作，页面如下：



下面通过 selenium 的 webdriver，对审核状态下拉框的处理，在下拉框中，选择定价。代码如下（登陆代码没有贴上来，这里是登陆后的操作代码）：

```
driver.find_element_by_css_selector('#dashboard-menu > li:nth-child(2) > a:nth-child(1)').click()
点击登陆后页面中的待审核创意
driver.find_element_by_css_selector('select.span1:nth-child(8)').click()
定位到下拉框，然后点击，点击后，会显示该下拉框中所有选项。
driver.find_element_by_xpath('//option[contains(text(),"定价")]').click()
在下拉框的所有选项中，找到定价。
```

第二种方式：

```
check_status=driver.find_element_by_xpath('//select[@name="status"]')
找到下拉框的元素，将操作句柄赋给变量 check_status。
check_status.find_element_by_xpath('//option[@value="8"]').click()
通过操作句柄 check_status 来选择下拉框中的选项
```

3、 时间等待

在做自动化测试时，难免会碰到一些问题，比如你在脚本中操作某个对象时，页面还没有加载出来，你的操作语句已经被执行，从而导致脚本执行失败，针对这样的问题 webdriver 提供了等待操作，等待一定的时间，或在一个时间段内发现对象，则继续操作。

Webdriver 提供了隐式等待和显示等待，当然，我们也可以借助 time 包的 sleep 模块，实现强制等待。

3.1 sleep 等待

sleep 是等待多少秒后，再继续执行后面的代码，要想使用 sleep，必须先导入 time 包。示例如下：

```
from selenium import webdriver
from time import sleep
导入 time 包的 sleep 模块
driver=webdriver.Firefox()
driver.get('http://www.chuangyijia.com/admin/login')
sleep(2)
等待 2 秒
```

也可以直接导入 time 包，然后通过 time.sleep(seconds)实现：

```
from selenium import webdriver
import time
导入 time 包的 sleep 模块
driver=webdriver.Firefox()
driver.get('http://www.chuangyijia.com/admin/login')
time.sleep(2)
等待 2 秒
```

3.2 智能等待

隐式等待：implicitly_wait()

当使用了隐式等待执行测试的时候，如果 WebDriver 没有在 DOM 中找到元素，将继续等待，超出设定时间后则抛出找不到元素的异常， 换句话说，当查找元素或元素并没有立即出现的时候，隐式等待将等待一段时间再查找 DOM，默认的时间是 0，一旦设置了隐式等待，则它存在整个 WebDriver 对象实例的声明周期中，隐式的等到会让一个正常响应的应用的测试变慢，–它将会在寻找每个元素的时候都进行等待，这样会增加整个测试执行的时间。

```
driver.get('http://www.chuangyijia.com/admin/login')
driver.implicitly_wait(10)
等待 10 秒
```

显式等待：WebDriverWait()

在 web 界面操作时，如果使用 sleep 等待，需要明确知道等待多长时间，如果时间太短，则容易产生超时，未能找到操作元素，如果时间太长，则容易浪费时间。如果使用 implicitly_wait，则是全局等待。WebDriverWait 可以配合 webdriver 的 expected_conditions 实现针对某个元素的等待操作。示例：

```
from selenium import webdriver
from selenium.webdriver.support import expected_conditions
导入 expected_conditions 模块
from selenium.webdriver.common.by import By
导入 By 模块
from selenium.webdriver.support.ui import WebDriverWait
导入 WebDriverWait
driver.get('http://www.chuangyijia.com/admin/login')
WebDriverWait(driver,10).until(expected_conditions.visibility
_of_element_located((By.ID,'email')))
```

WebDriverWait(driver,10).until(expected_conditions. visibility-_of_element_located (By.is_valid('email')))

这段代码需要解释，WebDriverWait（driver,10），driver 为打开浏览器的操作句柄，10 为超时时间，until 将 expected_conditions. visibility-_of_element_located ((By.Id ,'email'))作为参数，直到返回 True。Until_not 直到参数返回为 false。(By.ID, 'email')通过 id 查找邮箱地址输入框，expected_conditions. visibility _of_element_located 判断邮箱地址输入框是否可见，并且该元素的高和宽不为 0。总结该句代码的意思为，判断邮箱地址输入框是否加载完成，并可见，如果没有完成，

则默认每隔 0.5 秒检查一次，直到 10 秒后超时，如果在 10 秒内完成，则继续执行之后的代码。

更多关于 expected_conditions 的方法，在后面继续说明。

4、 浏览器操作

4.1 浏览器最大化

Webdriver 打开浏览器后，默认不是最大化，如果需要界面最大化，需要通过 maximize_window()方法来实现，代码如下：

```
from selenium import webdriver
driver=webdriver.Firefox()
driver.maximize_window()
将浏览器最大化
```

4.2 浏览器的高、宽

在 webdriver 中，除了可以通过 maximize_window()方法将浏览器最大化之外，也可以自定义界面的大小。

```
from selenium import webdriver
driver=webdriver.Firefox()
driver.set_window_size(480,800)
将界面设置高为 800，宽为 480
refox()
```

4.3 浏览器的前进、后退

在浏览器界面中，可以手动点击前进、后退，实现访问下一个页面或退回到前一个页面，在 selenium 的 webdriver 中，可以通过 back()方法实现后退，forword()方法实现前进，代码如下：

```
from selenium import webdriver
导入 webdriver
driver=webdriver.Firefox()
打开 firefox 浏览器
driver.get('http://www.chuangyijia.com/admin/login')
打开 LMD 后台登陆页面
driver.maximize_window()
窗口最大化
WebDriverWait(driver,10).until(expected_conditions.presence_of_element_located((By.ID,'email')))
显示等待，等待邮箱输入框
driver.find_element_by_id('email').send_keys('lib@163.com')
输入用户名
driver.find_element_by_id('password').send_keys('12345678')
输入密码
driver.find_element_by_css_selector('button.btn').click()
点击登陆
driver.implicitly_wait(3)
隐式等待 3 秒
driver.find_element_by_css_selector('#dashboard-menu > li:nth-child(2) > a:nth-child(1)').click()
点击待审核创意
driver.find_element_by_css_selector('select.span1:nth-child(8)').click()
点击审核状态的下拉框
driver.back()
后退
driver.forword()
前进
```

4.4 浏览器的关闭

在一个测试结束的时候，往往会将已经打开的浏览器关闭，这里顺便将浏览器的打开操作也做一次总结，浏览器的关闭可以通过 quit()方法完成。

前面的案例中，很多地方都有用到浏览器的打开，打开方式简单：

```
from selenium import webdriver
driver=webdriver.Firefox()
打开一个 firefox 浏览器
browser=webdriver.Ie()
打开 ie 浏览器
chrome=webdriver.Chrome()
打开 google 浏览器
driver.quit()
```

5、 鼠标键盘操作

在浏览器中，通常会用到鼠标来进行操作，比如右键菜单中选择一个操作，在 selenium 中提供了下列鼠标相关操作。

ActionChains 类提供了以下方法：

context_click() 右击

double_click() 双击

drag_and_drop() 拖动

5.1 鼠标右击

模拟用户在 LMD 登陆界面，在输入邮箱地址的输入框右键，但是这里本身没有定义右键，所以只能打开右键，而无法操作右键，如果在项目中有用到，那右键后的菜单也可以进行定位，并操作。示例：

```
from selenium import webdriver
```

```
driver=webdriver.Firefox()
driver.get('http://www.chuangyijia.com/admin/login')
driver.implicitly_wait(3)
test=driver.find_element_by_id('email')
找到要执行右键操作的元素
ActionChains(driver).context_click(test).perform()
对被操作元素执行右键
```

5.2 鼠标双击

```
from selenium import webdriver
driver=webdriver.Firefox()
driver.get('http://www.chuangyijia.com/admin/login')
driver.implicitly_wait(3)
test=driver.find_element_by_id('email')
找到要执行右键操
ActionChains(driver).double_click(test).perform()
对被操作元素执行双击
```

5.3 鼠标拖放

在一些 web 页面中，一些菜单需要将鼠标放上去，才会显示它的子菜单，在这种情况下，自动化需要模拟人为将鼠标放到菜单上。

下面通过百度页面来实现这个操作，在百度页面中，要对搜素的设置进行设置，这种操作需要将鼠标放到页面的设置菜单中，才能看到搜索设置，才能进行下一步的操作。代码如下：

```
driver.get('https://www.baidu.com')
打开百度页面
seting=driver.find_element_by_link_text('设置')
找到设置
ActionChains(driver).move_to_element(seting).perform()
将鼠标移动到设置菜单上
driver.find_element_by_link_text('搜索设置').click()
点击设置下的搜索设置
```

5.4 按键用法

使用键盘时，需要导入 selenium.webdriver.common.keys 中的 Keys 模块。

下面代码模拟用户通过键盘向邮箱地址中输入一个数字。

```
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys(Keys.NUMPAD3)
Keys.NUMPAD3 表示从键盘输入数字 3
```

下面模拟操作 tab 键和 enter 键

```
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys('lib@163.com')
输入邮箱
driver.find_element_by_xpath('//form[@id="login"]/input[2]').
send_keys('12345678')
输入密码
driver.find_element_by_xpath('//form[@id="login"]/input[2]').
send_keys(Keys.TAB)
按下 tab 键
driver.find_element_by_xpath('//form[@id="login"]/button').se
nd_keys(Keys.ENTER)
按下回车键
```

通过上面的代码能够看出，输入邮箱和密码之后，按下 tab 键，操作会切换到登陆按钮上，然后在登陆按钮上模拟用户按下 enter 键。

当然这样的操作需要按业务的顺序来的，否则会出错。

5.5 组合键

在 web 页面使用键盘除了上面的操作之外，还可能会有其他操作，比如组合键。

接下来，通过代码模拟用户在界面输入邮箱地址之后，使用 ctrl+a 的方式，将其全选，然后在使用 ctrl+c 的方式将内容复制出来，登陆成功后，将复制的内容粘贴到创意列表的标题中。代码如下：

```
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys('lib@163.com')
```

```
输入邮箱地址
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys(Keys.CONTROL,'a')
将输入的字符串使用 ctrl+a 键全选
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys(Keys.CONTROL,'c')
在按下 ctrl+c，将全选的内容复制到剪切板
driver.find_element_by_xpath('//form[@id="login"]/input[2]').
send_keys('12345678')
输入密码
driver.find_element_by_xpath('//button[contains(text(),"登录
")]').click()
登陆
#sleep(2)
driver.implicitly_wait(3)
driver.find_element_by_css_selector('#dashboard-menu >
li:nth-child(2) > a:nth-child(1)').click()
点击待审核管理
driver.find_element_by_css_selector('#input01').send_keys(Key
s.CONTROL,'v')
在搜索栏中的标题输入框中，用 ctrl+v 粘贴到输入框
```

6、 多层窗口定位

7.1 多层框架定位

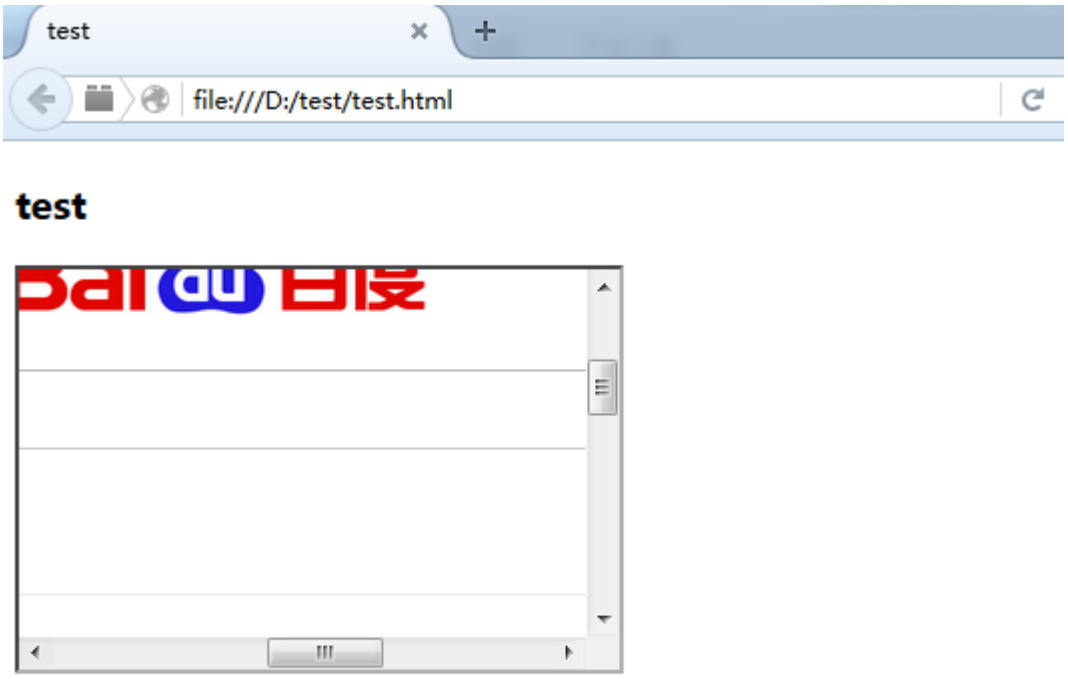
在 web 的自动化测试工作中，通常会碰到一个元素无法定位的问题，检查了很多次，依然得不到解决，此时就需要了解下 html 的 frame 框架了，frame 可以实现一个窗口中显示多个 html 文件，而当我们使用 selenium 打开页面后，定位元素时，发现无法定位，此时需要确定自己要操作的元素在哪个 frame 中。

下面这段 html 代码是将百度的页面封装在一个 frame 中，如果还想之前那样操作，则无法定位百度页面的元素，此时需要先定位到它所在的 frame 中，才能定位，使用 switch_to_frame 方法切换不同的 frame。

```
<html>
<head>
```

```
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>test</title>
</head>
<body>
<div class="row-fluid">
<div class="span6 well">
<h3>test</h3>
<iframe id="f2" src="http://www.baidu.com"
width="700"height="500"></iframe>
<h3>********************</h3>
<a href="javascript:alert('watir-webdriver better than
selenium webdriver;')">click</a>
</div>
</div>
</body>
</html>
```

将上面代码保存为 test.html，运行结果如下:



在这个案例中，我们可以直接定位到 click 这个链接，却无法定位到百度的搜索输入框，通过上面的 html 代码能看出，百度的页面是放在 id 为 f2 的框架中，所以需要先切换到 f2 的框架，然后在定位到输入框。

```
from selenium import webdriver
```

```
导入 webdriver
driver = webdriver.Firefox()
打开浏览器
driver.get('file:///D:/test/test.html')
打开 test.html
driver.switch_to_frame('f2')
先切换到 f2 的 frame 框架中
driver.find_element_by_id('kw').send_keys('test')
在定位百度的输入框
driver.switch_to_default_content()
重新回到之前的 frame
driver.find_element_by_tag_name('a').click()
此时才能操作 click 这个链接
```

在切换到 f2 的 frame 之后，对里面的操作完成后，往往还要回到之前的 frame，才能继续下一步的操作，此时可以通过 switch_to_default_content()方法返回。

7.2 多层窗口定位

在页面操作时，有些时候会出现多个窗口的情况，比如，点及一个链接后，会打开一个新的窗口，此时想要对新窗口进行操作时，就必须先切换到新的窗口才能继续操作，可以通过 switch_to_window()方法来实现。下面通过代码来实现点击一个创意项目之后，会弹出该项目的具体信息页面，此时需要切换到新页面才能操作。

```
from selenium import webdriver
导入 webdriver
driver=webdriver.Firefox()
打开 firefox 浏览器
driver.get('http://www.chuangyijia.com/admin/login')
打开 LMD 的登陆页面
driver.implicitly_wait(3)
driver.find_element_by_xpath('//form[@id="login"]/input[1]').
send_keys('lib@163.com')
输入邮箱
driver.find_element_by_xpath('//form[@id="login"]/input[2]').
send_keys('12345678')
输入密码
driver.find_element_by_xpath('//button[contains(text(),"登录
")]').click()
```

```
点击登陆
driver.implicitly_wait(3)
等待 3 秒
driver.find_element_by_css_selector('#dashboard-menu > li:nth-child(2) > a:nth-child(1)').click()
点击待审核项目
driver.find_element_by_link_text('预售管理').click()
点击预售管理
driver.find_element_by_css_selector('.table > tbody:nth-child(2) > tr:nth-child(1) > td:nth-child(4) > a:nth-child(1)').click()
点击预售管理中的第一个项目的标题，此时会弹出一个新的窗口
print driver.title
打印当前窗口的 title，输出结果说明，此时还是在之前的窗口操作的
并没有切换到新的窗口
window=driver.window_handles
获取当前所有的浏览器操作句柄
driver.switch_to_window(window[1])
切换到新窗口
print driver.title
打印新窗口的 title
```

从上面的案例中，我们需要先获取窗口的句柄，然后再进行切换的，句柄的规则是，按打开顺序来看，管理页面的窗口是第一个被打开的，那么他的句柄下标为 0，新窗口是第二个被打开的，那么它的句柄下标是 1.window 变量接收当前所有窗口的句柄，通过 switch_to_window（window[1]）切换到新窗口，同样也可以通过 switch_to_window(window[0])回到之前的页面。

7、 警告框处理

在 web 中，除了上面提到的元素和操作之外，还有就是页面的提示框的处理了，页面的警告框通常分为这几类 js alert 、 confirm 以及 prompt，这些警告框，我们都可以通过 switch_to_alert()来处理。

对警告框的处理有以下几种：

- text 获取警告框中的文本
- accept 点击警告框中的确定

- dismiss　　如果警告框中有取消，则可以通过该方法点击取消。

将下列代码保存为.html 文件，点击上面的清空数据的按钮，会产生一个 confirm 的警告框，点击确定，则输入框内显示确定，点击取消，则输入框显示取消，接下来对这个警告框进行处理。代码如下：

```
<script>
function clear1()
 {
  if(confirm("确定要清空数据吗？"))
  {
   document.main.text1.value = "确定";
  }
  else
  {
  document.main.text1.value = "取消";
  }
 }
</script>
<body>
<form name="main">
 <input type="text" name="text1" />
 <input type="button" name="Submit" value="清空数据" onClick="return
clear1();">
</form>
</body>
```

```
from selenium import webdriver
导入 webdriver
from time import sleep
需要用到 sleep 来暂停，所以这里导入 time 的 sleep 模块
driver = webdriver.Firefox()
打开 firefox 浏览器
driver.get('file:///D:/test/test1.html')
打开 html 文件
driver.find_element_by_css_selector('body > form:nth-
child(1) > input:nth-child(2)').click()
点击按钮
text=driver.switch_to_alert().text
获取警告框的文本，赋给 text 变量
print text
```

```
将文本输出
driver.switch_to_alert().dismiss()
点击警告框中的取消
sleep(3)
等待 3 秒
driver.find_element_by_css_selector('body > form:nth-
child(1) > input:nth-child(2)').click()
再次点击按钮
driver.switch_to_alert().accept()
点击警告框中的确定
```

8、 Cookie 处理

通过 webdriver 可以对浏览器中的 cookie 进行处理，常见处理方式有获取 cookie、添加 cookie、删除指定 cookie、删除所有 cookie。

8.1 获取 cookie 信息

```
from selenium import webdriver
from time import sleep
drvier=webdriver.Firefox()
drvier.get('http://www.chuangyijia.com/login')
打开前台登陆页面
drvier.implicitly_wait(3)
drvier.find_element_by_id('email').send_keys('8101550
67@qq.com')
输入用户名
drvier.find_element_by_id('pwd').send_keys('a654321')
输入密码
drvier.find_element_by_css_selector('#submit').click()
点击登陆
cookie = drvier.get_cookies()
获取登陆后的 cookie 信息
print cookie
打印获取到的 cookie 信息
```

8.2 向 cookie 中添加信息

```
from selenium import webdriver
from time import sleep
drvier=webdriver.Firefox()
drvier.get('http://www.chuangyijia.com/login')
打开前台登陆页面
cookie = drvier.add_cookie({ ‘name’ :’ key-
test’ ,’ value’ :’ key-test’ })
添加 cookie 信息
```

添加 cookie，可以使用 add_cookie 方式添加。

8.3 删除 cookie 中的信息

```
drvier.delete_cookie('ci_session')
删除 cookie
drvier.delete_all_cookies()
删除所有 cookie
```

9、 expected_conditions

在自动化测试过程中，通常需要对测试结果做出判断，在此可以通过 expected_conditions 来实现预期结果的判定，以此来断言执行状况。

expected_conditions 提供了很多方法，常用的方法如下:

- title_is：判断当前页面的 title 是否为预期结果
- title_contains：判断当前页面的 title 是否包含预期字符
- presence_of_element_located：判断一个元素是否存在，但是不表示该元素可见，如果该元素存在，则返回该元素，否则抛出异常。
- visibility_of_element_located：判断页面是否存在元素，并且该元素可见，如果存在并可见，则返回该元素，否则抛异常。

- presence_of_all_elements_located：判断至少有一个页面存在，只要有一个，则返回一个所有元素的列表，否则返回空列表。
- text_to_be_present_in_element：判断一个元素的文本中是否包含了预期字符串，匹配则返回 True，否则返回 False。

```
from selenium import webdriver
导入 webdriver
from selenium.webdriver.support import expected_conditions
导入 expected_conditions 模块
from selenium.webdriver.common.by import By
在 expected_conditions 中需要使用定位，by 提供统一使用
find_element()方法，简化了定位操作
from time import sleep
导入 sleep 模块

drvier=webdriver.Firefox()
打开浏览器

drvier.get('http://www.chuangyijia.com/login')
打开登陆页面

drvier.implicitly_wait(3)
等待 3 秒

drvier.find_element_by_id('email').send_keys('810155067@qq
.com')
输入用户名

drvier.find_element_by_id('pwd').send_keys('a654321')
输入密码

drvier.find_element_by_css_selector('#submit').click()
登陆

sleep(2)
此时等待 2 秒，主要是为了获取 title，太快的话，获取的 title
是登陆成功之前的 title
```

```
is_title=expected_conditions.title_is(u'首页-创意家')
判断页面的 title 是否为预期的字符串

is_title(drvier)
如果与预期字符串相等，这里返回结果为 True，否则为 False。
Title_is 是一个 Class，该 class 中实现了__call__方法，那么这个类对象就能像函数一样调用了。
is_in_title=expected_conditions.title_contains(u'创意家')
判断 title 中是否包含预期字符串

is_in_title(drvier)
可以用 print 打印他的返回结果为 True 还 False

is_exist=expected_conditions.presence_of_element_located((By.CSS_SELECTOR,'.sq_menu > a:nth-child(3)'))
判断元素是否存在页面上，不一定会显示在页面上

print is_exist(drvier)
如果存在，这里返回元素信息，否则这里会出现 NoSuchElementException 的异常。

in_ele=expected_conditions.presence_of_all_elements_located((By.TAG_NAME,'li'))
页面是否至少存在一个指定元素

print in_ele(drvier)
如果存在，这里返回一个列表，否则返回列表为空

visibility_exist=expected_conditions.visibility_of_element_located((By.TAG_NAME,'li'))
检查元素是否可见
print visibility_exist(drvier)
如果元素存在并可见，则返回元素信息，元素不可见，则返回异常，NoSuchElementException 的异常，如果元素存在，但是不可见，则返回 False。
```

```
is_text_in_ele=expected_conditions.text_to_be_present_in_element((By.CSS_SELECTOR,'.menu > ul:nth-child(1) > li:nth-child(1) > a:nth-child(1)'),u'意')
检查元素中是否包含字符串

print is_text_in_ele(drvier)
如果检查中包含字符串，则返回 True，否则返回 False。
```

10、Python 的 unittest 框架

9.1 Unittest 框架介绍

Unittest 是 python 的单元测试框架，原名为 PyUnit，由 java 的 junit 演化而来。

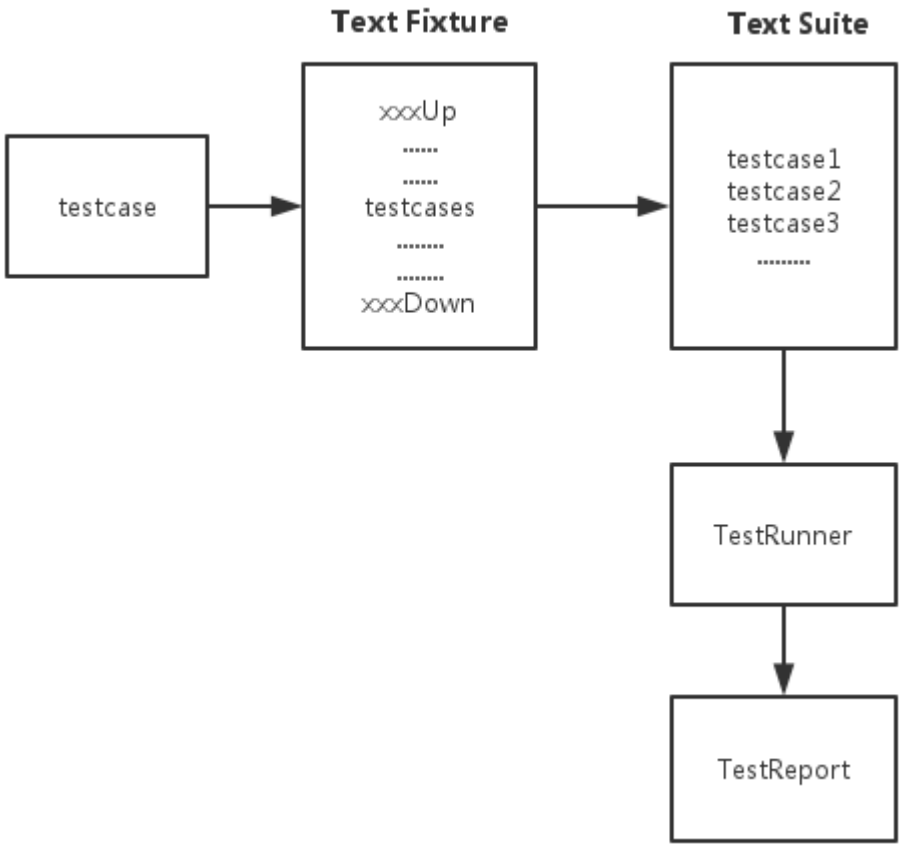
Unittest 提供了 test case、test suites、test fixtures、test runnet。

Test case ：通过继承 TestCase 类，实现创建 test 或 tests

Test suite：测试套，通常把一组相关的测试称为一个测试套，通过测试套件，将服务于同一个测试目的或同一运行环境下的一系列测试用例有机的组合起来。测试套件是按照测试计划所定义的各个阶段的测试目标决定的，即先有测试计划，后面才有测试套件。

test fixtures：setup + test case + teardown 的组合

Unittest 结构：



9.2 Unittest 框架详解

9.2.1 测试用例

在 unittest 中没有明确究竟 test 的 class 是测试用例，还是 class 中的 test 开头的方法是测试用例，在下面的案例中，我们将 test 开头的方法作为测试用例来执行。

先看一个简单的案例：

```
import unittest
在使用 unittest 框架时，需要先导入 unittest
class login_test(unittest.TestCase):
创建一个测试类，该类继承 unittest 的 TestCase。
   def setUp(self):
Setup 是测试执行之前的操作，会在用例执行之前先执行这里的内容，通常用来初始化环境
     self.testa=10
```

```
        self.testb=10
    def test_login(self):
    要执行的测试 1
        self.assertTrue(self.testa == self.testb)
    def test_login2(self):
    要执行的测试 2
        try:
            self.testc=30
            self.assertTrue(self.testa == self.testc)
        except AssertionError:
            self.fail("test is false")
    def tearDown(self):
    执行结束之后的操作，通常用来执行还原测试环境的操作
        print('Test Runner End')
if __name__ == '__main__':
    unittest.main()
    调用 unittest 的开始执行
```

在使用 unittest 前，需要先导入 unittest 的包，在这个包中包含了 unittest 提供的各种类，如：TestCase、TestSuite、TextTestRunner、TestLoader、main 等等。

创建一个 login_test 的 class，用来测试，该类继承 TestCase，从而能够使用 TestCase 中的方法。

Setup 方法是 TestCase 中的方法，这里是重写 TestCase 的 setup 方法，该方法用来做测试执行前的操作，比如构建测试数据，打开浏览器、链接数据库等操作。下面的 tearDown 方法相同，区别在与该方法用于执行测试执行结束后的操作，比如关闭浏览器，释放数据库连接操作。

中间以 test 开头的方法，则会被当做测试用例来执行，运行该脚本时，unittest 会自动加载这些 test 开头的方法，来执行测试。

Unitest.main 是调用 unittest 的执行方法，这里是测试执行的入口。

9.2.2 测试套

测试套又称测试集，也叫测试套件，通常是一组测试用例的集合，通过 suite 可以执行一组测试用例。下面的代码是针对 LMD 的登陆测试，如果登陆成功，则获取他

的 title，判断 title 是否与预期结果一致，如果一致，则测试成功，否则用例执行失败。这个案例通过 suite 实现的。代码如下：

```
    #coding=utf-8
__author__ = 'Administrator'
import unittest
from selenium import webdriver
import HTMLTestRunner
from selenium.webdriver.support import expected_conditions
class login_test2(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()

    def test_login(self):
        driver = self.driver
        driver.get('http://www.chuangyijia.com/login')
        driver.find_element_by_id('email').send_keys('810155067
@qq.com')
        driver.find_element_by_id('pwd').send_keys('a654321')
        driver.find_element_by_css_selector('#submit').click()
        test=expected_conditions.title_is(u'创意家，登录')
        self.assertTrue(test(driver))

    def tearDown(self):
        self.driver.quit()
if __name__ == '__main__':

suite=unittest.TestLoader().loadTestsFromTestCase(login_test2)
    unittest.TextTestRunner().run(suite)
```

这段代码中，上面的内容和之前没有上面差别，主要在 if __name__=='__main__'的内容，这句话的意思为“**Make a script both importable and executable**”，解释过来就是**让你写的脚本模块既可以导入到别的模块中用，另外该模块自己也可执行。**试代码的时候，在” if __name__ == '__main__' “中加入一些我们的调试代码，我们可以让外部模块调用的时候不执行我们的调试代码，但是如果我们想排查问题的时候，直接执行该模块文件，调试代码能够正常运行！

Unittest.TestLoader：源代码解释为“This class is responsible for loading tests according to various criteria and returning them wrapped in a TestSuite”

意思可以通过该类完成测试用例的加载，将用例加载到 siuter 中，但是需要调用该类的 loadTestFromTestCase 方法来加载用例，该方法接收参数为测试类，也就是我们上面定义的 login_test2 的类名。这句代码就会将该类中以 test 开头的方法加载到 suiter 中。Suiter 变量用来接收加载后的结果。

通过 unittest 调用他的 TextTestRunner 类中的 run 方法来执行测试，run 方法需要将 suiter 作为参数传入。TextTestRunner 是以文本形式显示测试结果，如果测试失败，会显示失败的测试名称，并统计测试的结果。run 方法是运行传入的 case 或 suiter。

9.2.3 断言

在编程中，通常会做一些假设，使用断言，去判断一个代码处理的结果是否和自己的预期相符，如果符合则为 True，否则为 false。在测试中，我们也会有这样的需求，执行一个测试，给一个预期结果，来断言实际结果与预期结果是否符合，不符合则测试失败，符合则执行成功。

在 unittest 的 TestCase 中给我们提供了很丰富的断言操作，下面罗列出了常用的一些断言。

assertTrue：检查表达式返回值是否为真 True，返回值类型为布尔类型。

assertFalse：与上面相反，检查表达式返回是否为假 False，布尔类型。

assertEqual(arg1, arg2, msg=None)：验证 arg1 和 arg2 是否相等。

assertNotEqual(arg1, arg2, msg=None)：验证 arg1 和 arg2 是否不相等。

assertIs(arg1, arg2, msg=None)：验证 arg1 和 arg2 是否为同一个对象。

assertIsNot(arg1, arg2, msg=None)：验证 arg1 和 arg2 是否不为同一个对象。

assertIsNone(expr, msg=None)：验证 expr 是否为 none，是则返回 True。

ssertIsNotNone(expr, msg=None)：验证 expr 是否不为 none，是则返回 True。

assertIn(arg1, arg2, msg=None)：验证 arg1 是否在 arg2 中，是则 True

assertNotIn(arg1, arg2, msg=None)：验证 arg1 不在 arg2 的中，是则 fail

示例：

```
#coding=utf-8
__author__ = 'Administrator'
import unittest
class Test_Asser(unittest.TestCase):
    def test_assertTrue(self):
        self.assertTrue(5>4)
        5>4 则返回 True，否则返回 False
    def test_assertFalse(self):
        self.assertFalse(5<4)
        表达式成立则返回 False，否则返回 True
    def test_assertEqual(self):
        self.assertEqual(u"中国",u"中国")
        两个字符串相等，则返回 True，否则返回 False
    def test_assertNotEqual(self):
        self.assertNotEqul(u"中国 1",u"中国 2")
        两个字符串不相等，返回 True，否则返回 false
    def test_assertIs(self):
        a=10
        a=20
        self.assertIs(a,a)
      a 是一个变量，下面的是对 a 重新赋值，所以依然是一个对象，所以此处返回 True，否则返回 false。
    def test_assertIsNot(self):
        a=10
        b=20
        self.assertIsNot(a,b)
      a 和 b 是两个变量，不属于同一个对象 ，所以此处返回 True，否则返回 False

    def test_assertIsNone(self):
        a=None
        self.assertIsNone(a)
        None 在 python 中是一个特殊的空值，即表示空值，也可以表示为空对象。在这里给 a 赋值为 None，那么 a 就是一个空对象，此处返回 True，否则返回 false
    def test_assertIsNotNone(self):
```

```
        c=10
        self.assertIsNotNone(c)
         与上面相反，c 是一个有值的对象，这里判断 c 是不是不为
None，是则返回 True，否则返回 False。
    def test_assertIsIn(self):
        text="abcdef"
        t='c'
        self.assertIn(t,text)
         判断 t 的值是否在 text 中，如果在则返回 True，否则返回
False。
    def test_assertIsNotIn(self):
        text="abcdef"
        self.assertNotIn('h',text)
         判断字符 h 是否不在 text 中，不在则返回 True，否则返回
false.
if __name__ == '__main__':
    unittest.main()
```

当然在 unittest 中不只是这些断言，还有其他断言，后续会慢慢加入。

9.3 测试批量执行

下面做两个测试的案例，先创建目录结构：

主目录为：Lmd_auto_test

次目录：Lmd_auto_test\test_case

在 test_case 目录中有 login_auto.py 和 Release_Creative.py 两个脚本。

将下面的代码保存为 login_auto.py

```
    #coding=utf-8
__author__ = 'Administrator'
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
```

```
import time
import unittest

class login_test_case(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.chuangyijia.com/login')
    def tearDown(self):
        self.driver.quit()
    def test_login(self):

self.driver.find_element_by_id('email').send_keys('810155067@
qq.com')

self.driver.find_element_by_id('pwd').send_keys('a654321')
        self.driver.find_element_by_id('submit').click()
        #self.driver.implicitly_wait(5)
        time.sleep(3)

WebDriverWait(self.driver,30).until(expected_conditions.visib
ility_of_element_located((By.CSS_SELECTOR,'.logo')))
        print self.driver.title
        is_title = expected_conditions.title_is(u'首页-创意家')
        self.assertTrue(is_title(self.driver))


if __name__ == '__main__':
    unittest.main()
```

下面的文件保存为 Release_Creative.py

```
    #coding=utf-8
__author__ = 'Administrator'
from selenium import webdriver
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.action_chains import
ActionChains
```

```
import unittest
import sys
import os,time
class release_creat(unittest.TestCase):
   def setUp(self):
      self.driver = webdriver.Firefox()
      self.driver.maximize_window()
      self.driver.get('http://www.chuangyijia.com/login')

self.driver.find_element_by_id('email').send_keys('810155067@
qq.com')

self.driver.find_element_by_id('pwd').send_keys('a654321')
      self.driver.find_element_by_id('submit').click()
      time.sleep(3)
      #is_title = expected_conditions.title_is(u'首页-创意家')
      #self.assertTrue(is_title(self.driver))
   def test_release(self):
      self.driver.find_element_by_css_selector('.menu >
ul:nth-child(1) > li:nth-child(1) > a:nth-child(1)').click()
      self.driver.find_element_by_css_selector('.c01 > a:nth-
child(1)').click()

self.driver.find_element_by_name('mobile').send_keys('1943235
923')

self.driver.find_element_by_css_selector('.idea_name').send_k
eys(u"发布中文创意")
      self.driver.find_element_by_id('province').click()

self.driver.find_element_by_xpath('//option[contains(text(),"
江苏")]').click()
      self.driver.find_element_by_id('city').click()

self.driver.find_element_by_xpath('//option[contains(text(),"
南京")]').click()
      text=u"预售管理，后台管理员可以修改时间，并且只能修改为
当月的日期，不能修改年月"
```

```
        self.driver.find_element_by_css_selector('#pForm >
table:nth-child(11) > tbody:nth-child(1) > tr:nth-child(1) >
td:nth-child(1) > table:nth-child(1) > tbody:nth-child(1) >
tr:nth-child(6) > td:nth-child(2) > textarea:nth-
child(1)').send_keys(text)
        self.driver.find_element_by_css_selector('#pForm >
table:nth-child(11) > tbody:nth-child(1) > tr:nth-child(1) >
td:nth-child(1) > table:nth-child(1) > tbody:nth-child(1) >
tr:nth-child(7) > td:nth-child(2) > textarea:nth-
child(1)').send_keys(u"没有问题。。。。")
        self.driver.find_element_by_css_selector('#pForm >
table:nth-child(11) > tbody:nth-child(1) > tr:nth-child(1) >
td:nth-child(1) > table:nth-child(1) > tbody:nth-child(1) >
tr:nth-child(8) > td:nth-child(2) > textarea:nth-
child(1)').send_keys(u"不需要解决")

self.driver.find_element_by_name('support_end_date').send_key
s('2017-8-18')
        time.sleep(2)

self.driver.find_element_by_css_selector('.c_btn').click()

self.driver.find_element_by_css_selector('.c_btn').click()
        os.system("H:\\pydj\\Lmd_auto_test\\testjpg.exe")
        time.sleep(2)
        put2 =
self.driver.find_element_by_css_selector('#upimg_1')
        put1 =
self.driver.find_element_by_css_selector('.cy_pic > ul:nth-
child(1) > li:nth-child(1) > input:nth-child(3)')


ActionChains(self.driver).move_to_element(put2).perform()
        self.driver.find_element_by_css_selector('.cy_pic >
ul:nth-child(1) > li:nth-child(1) > input:nth-
child(3)').click()
        os.system("H:\\pydj\\Lmd_auto_test\\testjpg.exe")
        #self.driver.find_element_by_css_selector('.cy_pic >
```

```
ul:nth-child(1) > li:nth-child(2) > input:nth-
child(3)').click()
      #os.system("H:\\pydj\\Lmd_auto_test\\testjpg.exe")
      self.driver.find_element_by_id('saveBtn').click()
      time.sleep(2)
      is_exist =
self.driver.find_element_by_css_selector('div.item:nth-
child(1) > div:nth-child(2) > a:nth-child(1)')
      Creat_Name = is_exist.text
      self.assertEqual(Creat_Name,u"发布中文创意")
      #self.assertTrue(is_exist)

   if __name__ == '__main__':
      unittest.main()
```

上面的两个文件，一个是测试登陆功能的用例，另一个是测试创意发布的用例，这里对成功和失败只做了简单的判断，例如登陆，只判断登陆后，title 是否为登陆成功的 title，是则执行通过，否则执行失败。创意发布时，判断提交后，该创意是否在发布的列表中，并在第一个，如果是，则执行成功，否则执行失败。

在 test_case 的目录中创建__init__.py 文件，文件内容如下：

```
#coding=utf-8
__author__ = 'Administrator'
import login_auto
import Release_Creative
```

在 Lmd_auto_test 目录下创建 run_test_case.py 文件，文件主要用来实现执行 test_case 目录下的测试脚本。文件内容如下：

```
    #coding=utf-8
__author__ = 'Administrator'
# import test_case
import unittest
from test_case import login_auto,Release_Creative
suiter = unittest.TestSuite()
suiter.addTest(unittest.makeSuite(login_auto.login_test_case))
suiter.addTest(unittest.makeSuite(Release_Creative.release_cr
```

```
eat))
unittest.TextTestRunner().run(suiter)
```

11、HTMLTestRunner 生成测试报告

在之前的案例中，我们完成了自动化测试的基本能力，也能完成测试的执行工作，但是还没有做到将测试的结果以报表的形式输出，接下来，在之前的测试基础上加上测试报告的输出。

10.1 HTMLTestRunner 介绍

HTMLTestRunner 是 python 标准库 unittest 的扩展，可以生成一个直观的测试报告。在使用之前需要将 HTMLTestRunner.py 文件放如到 python 的安装目录下，例如我的就是 c:\\Python27 目录。

10.2 生成测试报告

下我们在之前的 LMD 登陆测试的脚本中先来看看 HTMLTestRunner 是如何使用的，将 login_auto.py 的内容修改如下：

```
    #coding=utf-8
__author__ = 'Administrator'
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
import time
import unittest
import HTMLTestRunner

class login_test_case(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.chuangyijia.com/login')
    def tearDown(self):
```

```
        self.driver.quit()
    def test_login(self):

self.driver.find_element_by_id('email').send_keys('810155067@
qq.com')

self.driver.find_element_by_id('pwd').send_keys('a654321')
        self.driver.find_element_by_id('submit').click()
        #self.driver.implicitly_wait(5)
        time.sleep(3)

WebDriverWait(self.driver,30).until(expected_conditions.visib
ility_of_element_located((By.CSS_SELECTOR,'.logo')))
        print self.driver.title
        is_title = expected_conditions.title_is(u'首页-创意家')
        self.assertTrue(is_title(self.driver))


if __name__ == '__main__':
    suite = unittest.TestSuite()
    suite.addTest(login_test_case("test_login"))
    Report_file =
u"H:\\pydj\\Lmd_auto_test\\Report\\Result.html"
    Rf = file(Report_file,'wb')
    Case_run =
HTMLTestRunner.HTMLTestRunner(stream=Rf,title=u'LMD 登陆测试
',description=u"测试报告输出")
    Case_run.run(suite)
```

上面的代码只是在原有基础上做了修改，加入了 import HTMLTestRunner 这句，还有后面的

```
suite = unittest.TestSuite()
创建一个测试套对象
suite.addTest(login_test_case("test_login"))
将登陆的测试用例添加到测试套中
Report_file = u"H:\\pydj\\Lmd_auto_test\\Report\\Result.html"
设置测试报告输出的位置及文件名
```

```
Rf = file(Report_file,'wb')
```
使用 python 标准库 file 打开测试报告文件，wb 是以二进制写的模式打开。
```
Case_run=HTMLTestRunner.HTMLTestRunner(stream=Rf,title=u'LMD 登陆测试',description=u"测试报告输出")
```
创建一个 HTMLTestRunner 的对象，并且将上面打开的用于输出测试报告的对象传入，title 是 html 报告页面的 title，description 对测试报告的描述
```
Case_run.run(suite)
```
开始运行测试套