# 自动化测试基本概念知识

自动化测试指的是利用程序代替人来对软件进行测试的一种测试方法，自动化测试可以对软件进行回归测试、压力测试以及兼容性测试等，帮助提高测试的效率，使软件测试更加全面，更加保证了软件的质量。

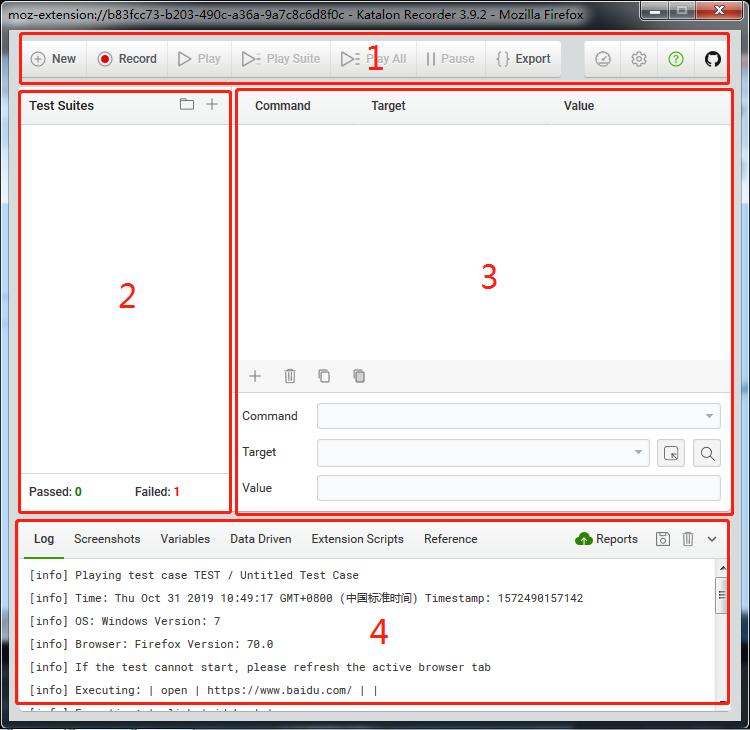
自动化测试可以在较少的时间内运行更多的测试用例，在短时间内可以进行重复性校验。但自动化测试不能完全代替手动测试，手工测试比自动化测试更能发现软件的缺陷，往往很多缺陷通过自动化测试难以发现。自动化测试是在完成手工测试之后才能进行，一般来讲周期较长、需求变动不频繁、需要进行回归测试的项目比较适合做自动化测试。

自动化测试可以分成以下几类：1.WEB应用自动化测试；2.客户端应用自动化测试；3.移动APP自动化测试；4.接口自动化测试；5.单元自动化测试。

# WEB自动化测试脚本录制

WEB自动化测试脚本录制不仅支持回放，还可以导出脚本语言，通过导出的脚本语言，可以帮助定位分析元素，对WEB自动化测试脚本的开发有很大的帮助。Katalon Recorder是一款浏览器插件，支持FireFox、Chrome浏览器，在浏览器官方插件商店中可以直接下载安装。Katalon Recorder的主要功能包括：录制回放脚本、定位网页元素、转化成脚本语言。（在火狐55版本之后，Katalon Recorder代替了原来的Selenuim IDE）

## Katalon Recorder简介



Katalon Recorder由4个部分组成：1.工具栏；2.测试用例管理器；3.用例详情；4.控制台。

主工具栏：NEW（新建一个测试用例）、Record（录制脚本的开关按钮）、Play（回放选中的测试用例）、Play Suite（回放选中的测试集）、Play All（回放所有）、Pause（暂停）、Export（导出脚本，可以选择不同的语言）、（调节脚本回放的速度）。



测试用例管理器：打开测试集、添加测试集、（1.保存测试集；2.关闭测试集；3.添加新的测试用例；4.重命名测试集）、（1.添加新的测试用例；2.移除测试用例；重命名测试用例；回放该测试用例）



用例详情：Command（操作命令：例如open打开一个URL、click单击网页上的链接或按钮）、Target（目标：指定必须在其上回放操作的web元素以及locator属性，例如id=kw就是文本输入框，id=su就是按钮）、Value（值：需要发送一些实际参数时使用。例如，在文本框中账号与密码）、右键（添加操作、删除操作、删除所有操作、复制操作、粘贴操作、断点、回放这个操作（可以双击触发回放）、从这开始回放）、（添加一个操作、删除选中的操作、复制选中的操作、粘贴操作）、选中操作可以对操作的属性进行修改编辑、（点击一下，工具会帮助解析鼠标选中的元素）、（点击一下，工具会在当前页面寻找并标亮出该元素所在的位置）。



控制台：LOG（日志信息）、Screenshots（截图：遇到执行错误的步骤会截图，并且打印出执行错误的步骤号）。

## Katalon Recorder脚本录制

点击录制按钮，在浏览器界面上进行操作，Katalon Recorder对用户的每一个操作都会生成对应的脚本，每个操作脚本包含命令、元素、值。录制完成，点击stop停止录制，生成的脚本支持回放、编辑修改以及导出不同语言的脚本。

## Katalon Recorder常用命令

1. open：打开一个链接地址，地址写在Target里面。如果地址写的域名加上路径或者地址加上路径的完整URL，那么会直接访问填写的地址。如果写的是路径，那么会打开当前页面地址加上路径的URL。
2. pause：推迟下个操作的执行时间，延迟的时间写在Target中，单位为毫秒。
3. goback：模拟浏览器的后退按钮。
4. refresh：刷新当前页面。
5. click：单击一个链接、按钮、复选框等。
6. type：向指定输入域中输入值，可以是文本框、下拉框等，值填写在value中。
7. close：模拟浏览器窗口关闭按钮。

# Selenuim与WebDriver

## Selenuim与WebDriver简介

Selenium是一种用于Web应用程序测试的工具，用Selenium进行测试时会直接运行在浏览器中，就像真正的用户在操作一样。Selenium可以使用多种编程语言的脚本进行调用，并且 Selenium能够在一个或多个浏览器中执行这些测试脚本。

Selenium中集成了WebDriver，WebDriver提供了一套友好的API，通过调用WebDriverAPI，可以达到控制浏览器驱动程序驱动浏览器进行操作。WebDriverAPI可以使用多种语言代码进行访问，WebDriverAPI支持Chrome、Firefox、IE等浏览器驱动。

## Selenuim进行WEB自动化测试的原理

Selenium在进行自动化测试时，可以看做是经典的Client—Server模式。Python等语言编写的脚本执行时，Selenium Client初始化一个Service服务，通过Webdriver启动浏览器驱动程序去打开浏览器。启动浏览器的同时，Selenium启动基于WebDriver Wire协议的Web Service作为 Selenium 的 Remote Server，并与浏览器绑定。之后，Remote Server 就开始监听Client 端的操作请求。执行测试脚本时，测试脚本会作为Client端，操作请求以Http Request 的方式发送给Remote Server（该Http Request 的body，是以WebDriver Wire 协议规定的 JSON 格式来描述需要浏览器执行的具体操作）。Remote Server 接收到请求后，会对请求进行解析，并将解析结果发给浏览器驱动，由浏览器驱动实际执行浏览器的操作。浏览器驱动可以看做是直接操作浏览器的原生组件，每种浏览器都有不同的驱动程序，所以搭建测试环境时，通常都需要先下载浏览器对应的浏览器驱动。

## Selenium与浏览器驱动的安装

1. Selenuim3安装

在DOS命令中调用Python的pip包管理工具进行下载，命令为pip install selenium，下载完毕之后，输入命令pip show selenium来查看是否下载安装成功。

1. 浏览器器驱动安装及配置

IE:IEDriverServer.exe，下载地址http://selenium-release.storage.googleapis.com/index.html

Chrome:chromdriver.exe，下载地址http://chromedriver.storage.googleapis.com/index.html

Firefox:geckodriver.exe，下载地址https://github.com/mozilla/geckodriver/releases

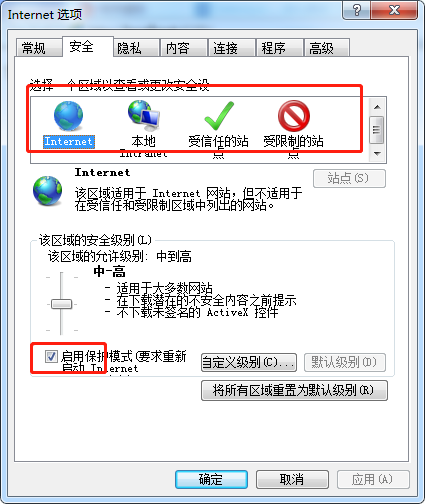
需要注意版本的问题，selenium与浏览器驱动版本的兼容，浏览器驱动与浏览器版本的兼容。下载的驱动程序可以放在Python安装目录下，也可以放在其他地方，放在其他地方需要在环境变量配置path中加上驱动路径。

（3）selenium控制浏览器

Selenium控制浏览器首先要从Selenium中导入webdriver类，然后通过webdriver类实例化浏览器，即打开浏览。具体步骤如下：

# 从selenium中导入webdriver  
 from selenium import webdriver  
 # 实例化浏览器对象，即打开浏览器。  
 driver=webdriver.Ie() # 打开IE浏览器  
 driver=webdriver.Firefox() # 打开火狐浏览器  
 driver=webdriver.Chrome() # 打开谷歌浏览器

注意：IE安全里面所有项是否启用保护模式必须要统一，否则启动IE浏览器将会报错。



拓展：Selenium主要支持Ie、FireFox、Chrome浏览器，对于市面上其他一些浏览器，例如360浏览器、QQ浏览器等，它们使用的内核大部分跟这三款浏览器的内核是一样的，所以理论上也能使用Selenium来进行控制。比如市面上很多浏览器使用了Chrome的内核，所以能利用webdriver类中的Chrome来实例化浏览器。在实例化时需要知道浏览器使用的Chrome内核的版本（打开浏览器F12开发者工具，在控制台中输入navigator.appVersion回车即可查看内核信息），然后根据版本下载相对应版本的浏览器驱动。在实例化浏览器时，需要指定使用的浏览器驱动以及启动浏览器的exe文件。以360浏览器为例，方法如下：

from selenium import webdriver  
# 指定浏览器的运行内核。  
options = webdriver.ChromeOptions()  
# 指定启动程序。  
options.binary\_location=r'E:\常用软件\360浏览器\360se6\Application\360se.exe'  
# 实例化浏览器指定驱动。  
driver = webdriver.Chrome(r'E:\WebDriver\chromedriver\_v2.36.exe', options=options)

# Webdriver元素定位

Selenium对网页的控制是基于各种前端元素的，在使用过程中，对于元素的定位是基础，只有准确抓取到对应元素才能进行后续的自动化操作，Selenium提供了基本元素的八种定位方法，此外还可以利用JavaScript、jQuery对元素进行操作。

## 通过元素的属性进行定位

**（1）ID属性**

通过元素的id属性进行定位，一般一个页面的id属性只会有一个，这种唯一性使得使用元素的id属性更加容易定位元素，定位的方法为find\_element\_by\_id()，操作步骤如下：

from selenium import webdriver # 从selenium包中导入webdriver。  
Browser = webdriver.Ie() # 实例化浏览器对象,即打开浏览器。  
Browser.get('http://lcoalhost:7001/') # 打开网页。  
username = Browser.find\_element\_by\_id('username') # 使用id定位查找元素。  
username.send\_keys('admin') # 使用send\_keys()方法填写元素。

**（2）NAME属性**

通过元素的name属性进行定位，页面可能会有多个元素含有相同的name属性，定位的方法为find\_element\_by\_name()，操作步骤参考id。

**（3）CLASS属性**

通过元素的calss属性进行定位，calss指定了元素的类名，页面可能会有多个元素含有相同的calss属性，定位的方法为find\_element\_by\_class\_name()，操作步骤参考id。

## 通过元素的标签名称进行定位

TAG\_NAME通过标签的名称进行定位，页面可能会有多个相同的标签，定位的方法为find\_element\_by\_tag\_name()，操作步骤参考id。

## 通过文本信息类型的超链接进行定位

**（1）LINK\_TEXT**

通过文本信息类型的超链接进行定位，定位的方法为find\_element\_by\_link\_text()，注意需要传入链接的全部文本，操作为点击，即find\_element\_by\_link\_text(‘链接’).click()。

**（2）PARTIAL\_LINK\_TEXT**

对link\_text的扩展，可以对长文本信息超链接进行模糊定位，不需要传入链接的全部文本，但要保证模糊条件的唯一性，操作为点击，即find\_element\_by\_partial\_link\_text(‘链接’).click()。

## 通过元素的路径信息进行定位

xpath是xml路径语言，它可以用来确定xml文档中的元素位置，通过元素的路径来完成对元素的查找。HTML就是XML的一种实现方式，所以可以用xpath来进行定位。定位的方法为find\_element\_by\_xpath，Xpath定位方式有以下几种：

1. **绝对路径**

这种定位方式是利用html标签名的层级关系来定位元素的绝对路径，一般从<html>标签开始依次往下进行查找。如百度搜索框的绝对路径xpath定位可以是这样的：find\_element\_by\_xpath("/html/body/div[1]/div[1]/div/div[1]/div/form/span[1]/input")([1]代表body第1个div标签)

1. **相对路径的属性**

这种方式是利用html标签名以及元素的属性进行定位，例如：find\_element\_by\_xpath("//input[@id='kw']")、find\_element\_by\_xpath("//\*[@name='wd']")

其中的标签名input也可以用\*来代替，而且只要是在该标签内，任意属性都可以，比如搜索框的maxlength属性：find\_element\_by\_xpath("//input[@maxlength='255']")

而且如果标签里面有多个属性，那可以利用多个属性条件来进行查找，例如：

find\_element\_by\_xpath("//input[@maxlength='255'and@name='wd’]")

**（3）相对路径与绝对路径结合**

有的时候会发现绝对路径定位路径太长，而且相对路径下相同属性值有多个，这个时候可以将相对路径与绝对路径结合起来使用。比如查找搜索框的时候发现其上级元素form又唯一的id方便定位，就可以先查找到form元素然后依次往下写路径：find\_element\_by\_xpath("//form[@id='form']/span/input")

**（4）延伸扩展**

find\_element\_by\_xpath("//\*[text()='xxx']")：查找文本内容为xxx的元素。

find\_element\_by\_xpath("//input[@id]")：查找包含id属性的input元素。

find\_element\_by\_xpath("//input[not(@id)]")：查找不包含id属性的input元素。

find\_element\_by\_xpath(“//\*[starts-with(@id,’username’)]”)：开始属性为id=username的元素。

find\_element\_by\_xpath(“//\*[contains(@id,’username’)]”)：包含属性为id=username的元素。

find\_element\_by\_xpath(“//\*[count(input)=2]”)：查找包含两个input元素的元素。

find\_element\_by\_xpath(“//\*[contains(@id,’username’)]/**..**”)：包含属性为id=username的元素的父元素。

find\_elements\_by\_xpath("//form[@id='form']/span/input[1]")：1表示span元素下面的第一个input元素。

find\_element\_by\_xpath(“//\*[local-name()=’xxx’]”)：找到标签名称为xxx的元素。

find\_element\_by\_xpath(“//\*[starts-with(local-name(),’x’)]”)：找到标签名称以x开头的元素。

find\_element\_by\_xpath(“//\*[contains(local-name(),’x’)]”)：找到标签名称包含x的元素。

当元素定位没什么头绪时，可以考虑利用元素的父元素或者子元素的属性特征使用相对路径与绝对路径结合来进行定位。

## 通过元素的显示样式进行定位

CSS是一种语言，他用来描述HTML和XML的元素显示样式，而在CSS语言中有CSS选择器，这种选择器可以使用不同的策略选择元素，在Selenium中可以使用这种选择器，定位的方法为find\_element\_by\_css\_selector()，CSS定位有以下几种常用的策略：

1. **Id选择器**

根据元素的id属性进行定位，格式为#id，选择器以#修饰，定位的方法为find\_element\_by\_css\_selector(‘#属性值)，例如元素有id属性id=username，定位的表达式为：find\_element\_by\_css\_selector(‘#username)。

1. **Class选择器**

根据元素的class属性进行定位，格式为.class，选择器以.修饰，定位的方法为find\_element\_by\_css\_selector(.属性值)，例如元素有class属性calss=cssLoginBtn\_L1，定位的表达式为：find\_element\_by\_css\_selector(‘.cssLoginBtn\_L1’)。

1. **元素选择器**

根据元素的标签名称进行定位，例如元素的标签名称为button，那么定位的表达式为find\_element\_by\_css\_selector(button)。

1. **属性选择器**

根据元素的属性名与属性值进行定位，格式为[属性名=属性值]，例如元素有属性type=”password”，那么定位的表达式为find\_element\_by\_css\_selector(“[type=’password’]”)。

1. **层级选择器**

根据元素的层级关系进行定位，格式为标签>标签或者是标签空格标签，一般都是配合元素的属性来使用，例如含有属性id=text的标签p标签下面的input标签，表达式为find\_element\_by\_css\_selector(“p[id=’text’]>input”)或者(“p[id=’text’]空格input”)。

**（6）延伸扩展**

find\_element\_by\_css\_selector(“Input[id^=’pa’])：元素input标签的id属性值的开头是pa，如果id的属性值的开头是pa的标签只有一个，那么标签可以不写。

find\_element\_by\_css\_selector(“input[name$=’pas’]])：元素input标签name的属性值的结尾是pas，如果name的属性值的结尾是pas的标签只有一个，那么标签可以不写。

find\_element\_by\_css\_selector(“Input[class\*=’pa’]])：元素input标签class属性值包含pa，如果class的属性值包含pa的标签只有一个，那么标签可以不写。

## 基本定位方法的其他写法

**（1）browser.find\_element(‘方法‘,‘参数值‘)**

browser.find\_element(‘name‘,‘username‘)

browser.find\_element(‘id‘,‘username‘)

browser.find\_element(‘class name‘,‘password‘)

browser.find\_element(‘tag name‘,‘button‘)

browser.find\_element(‘link text‘,‘链接‘)

browser.find\_element(‘partial link text‘,‘链接‘)

browser.find\_element(‘xpath‘,”//put[@id=’id’]”)

browser.find\_element(‘css selector‘,’#id’)

**（2）通过By类**

前提是要从selenium.webdriver.common.by 导入By包（快速导包快捷键：Ctrl+alt+空格、快速复制一行：Ctrl+d、注释多行Ctrl+/）,from selenium.webdriver.common.by import By。

browser.find\_element(By.NAME,‘username‘)

browser.find\_element(By.ID,‘username‘)

browser.find\_element(By.CLASS\_NAME,‘password‘)

browser.find\_element(By.TAG\_NAME,‘button‘)

browser.find\_element(By.LINK\_TEXT‘链接‘)

browser.find\_element(By.PARTIAL\_LINK\_TEXT,‘链接‘)

browser.find\_element(By.XPATH,”//put[@id=’id’]”)

browser.find\_element(By.CSS\_SELECTOR,’#id’)

**（3）多个元素选择**

find\_elements\_by\_元素，带有s这种定义方法为找到所有符合条件的元素，返回的元素为列表格式，列表数据格式的读取需要制定下标，下标从0开始。例如：find\_elements\_by\_tag\_name(‘input’)[1]，这是查找页面的第二个input标签元素。

**（4）通过父元素定位子元素**

调用查找元素的方法不仅仅实例化浏览器对象可以使用，元素也可以进行使用，所以也可以先通过查到到的父元素调用查找元素的方法去查找相应的子元素。

## \*JavaScript操作元素

JavaScript是世界上最流行的脚本语言，在电脑、手机、平板上浏览的所有的网页，以及无数基于HTML5的手机App，交互逻辑都是由JavaScript驱动的。JavaScript是一种运行在浏览器中的解释型的编程语言，在Web中，只有JavaScript能跨平台、跨浏览器驱动网页，与用户交互，运行JavaScript脚本可以直接对页面上的元素进行操作。Webdriver提供了运行JavaScript脚本的方法execute\_script(JavaScript)，只要编写好JavaScript脚本就能运行。

**（1）通过元素的id属性。**

document.getElementById(‘id’)

**（2）通过元素的name属性。**

document.getElementsByName(‘name’)

**（3）通过元素的class属性。**

document.getElementsByClassName(‘class’)

**（4）通过元素的标签名称。**

document.getElementsByTagName(‘tagname’)

**（5）通过选择器（单个）。**

document.querySelector()通过元素的id/class/tagname来定位，例如：

document.querySelector(‘input#username’)：id属性值是username的第一个input标签，标签名称不写的话就是id属性值是username的第一个标签。

document.querySelector(‘input.login’)：class属性值是login的第一个input标签，标签名称不写的话就是class属性值是login的第一个标签。

document.querySelector(‘button’)：第一个button标签。

**（6）通过选择器（多个）。**

document.querySelectorAll()：document.querySelector()的复数形式，返回多个元素的list。

上述方法中除了document.getElementById()与document.querySelector()查找的元素是单个，其它是列表形式返回，需要加上索引，例如document.getElementsByName(‘name’)[0]。执行JavaScript脚本需要加上元素的动作，不能通过执行JavaScript脚本获取元素，然后在对元素进行操作。常用的操作比如赋值、单击，直接在元素后面加上操作的代码即可：document.getElementById(‘id’).value=’username’，document.getElementsByTagName(‘tagname’)

.click()（JavaScript还有许多其他的操作元素的方法，有必要学习JavaScript）

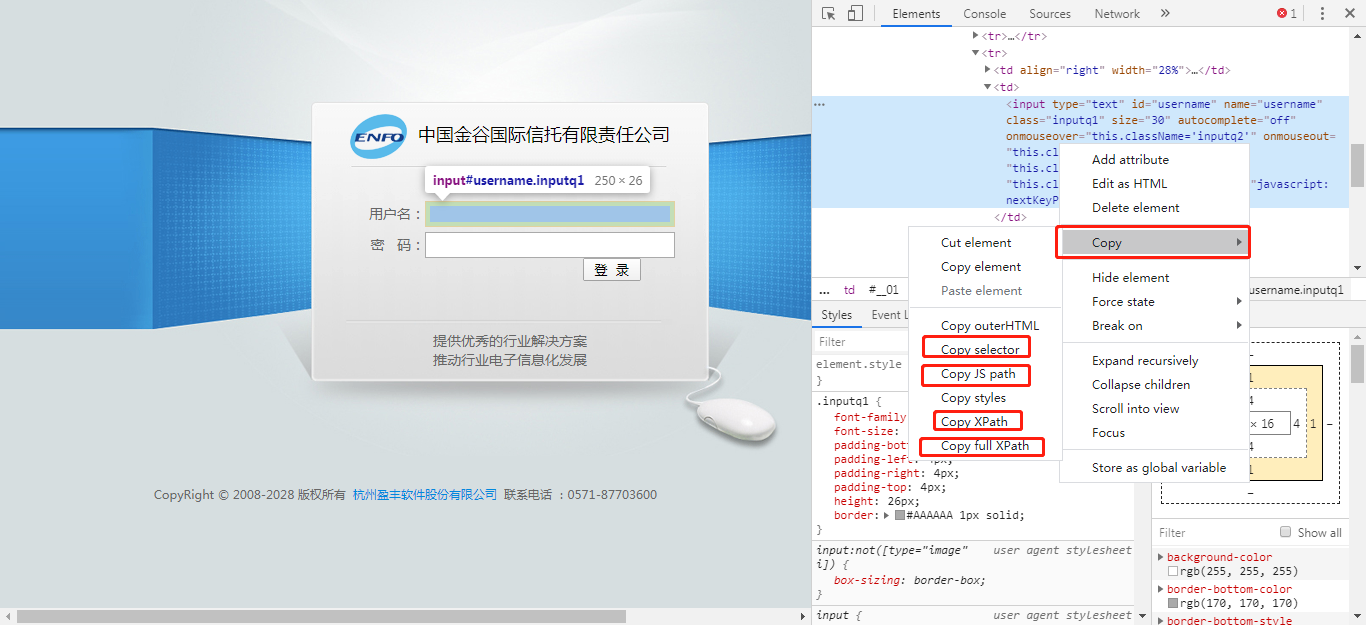
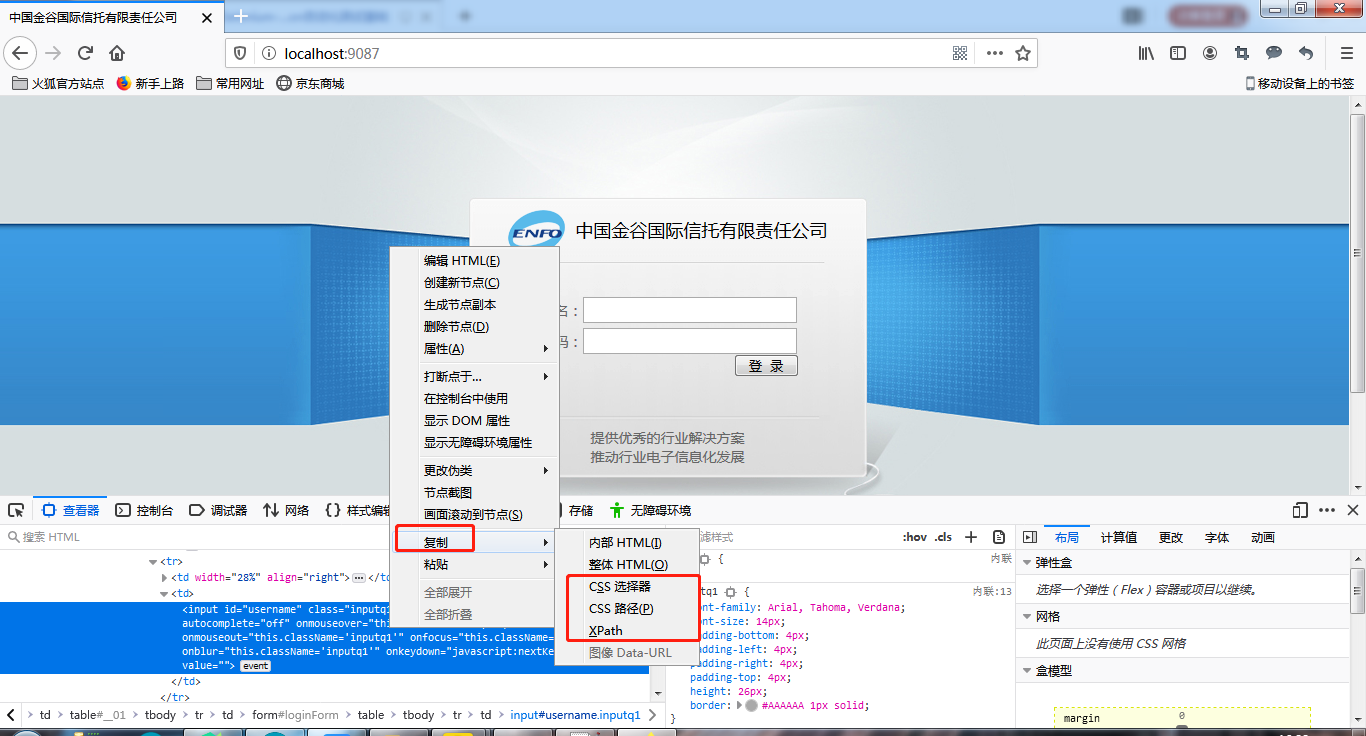
## jQuery操作元素

jQuery语法是为HTML元素的选取编制的，是一个JavaScript库，极大地简化了JavaScript编程。基本语法是$(selector).action()，$符号定义jQuery，selector选择器用来查询具体的HTML元素，通过action()来执行对元素的具体操作。其中经常用到的action()也是输入值与点击：（1）$(selector).val('value')其中value是输入的值，value为空时表示清空的意思；（2）

$(selector).click()点击，基本定位方式也是通过元素的id/class属性以及标签名称：$(‘#id’)、$(‘.class’)、$(‘tagname’)。

## 浏览器使用小技巧

使用FireFox、Chrome浏览器的查看器可以在查看器中直接复制元素的路径信息、样式信息。



# Webdriver元素及浏览器操作方法

## 元素常用操作方法

（1）clear()：清除文本信息。

（2）send\_keys('value')：输入文本信息。

（3）click()：点击元素。

（4）size：获取元素的尺寸。

（5）text：获取元素的文本。

（6）get\_attribute(attributename)：获取属性的值。

（7）location：获取元素坐标。

（8）is\_displayed() ：元素是否可见。

（9）is\_enabled()：元素是否可使用。

（10）is\_selected() ：元素是否被选中。

（10）tag\_name：元素的标签名。

使用元素的send\_keys(value)方法输入信息，有时候元素最终输入值可能会与提供给的value值不一致。那么这时候可以在输入信息之后，可以调用该元素的get\_attribute(‘value’)来获取真正输入到元素中的值。

## 浏览器常用操作方法

1. maximise\_window()：浏览器窗口最大化。
2. set\_window\_size(200,100)：设置浏览器大小，宽与高，单位为像素。
3. set\_window\_position(200,100)：设置浏览器的位置，X与Y，单位为像素。
4. back()：浏览器后退。
5. forword()：浏览器前进。
6. refresh()：浏览器刷新。
7. close()：关闭当前窗口。
8. quit()：关闭由webdriver打开的所有窗口，每次实例化浏览器之后都需要关闭，否则将会有越来越多的webdriver进程，导致服务器变卡。
9. get\_screenshot\_as\_file(r"D:\测试工作\baidu.png")：截取当前页面图并保存为，此功能可以帮助验证功能或者分析错误。

（10）submit()：提交表单。

（11）title：获取页面的标题。

（12）current\_url：获取当前页面的URL。

（13）page\_source：获取页面源码。

# Webdriver的鼠标及键盘操作

## Webdriver鼠标操作

Webdriver鼠标操作的方法封装在ActionChains类中，使用时需要从webdriver中导入，from selenium.webdriver.common.action\_chains import ActionChains，鼠标常用的操作方法：

（1）click(on\_element=None) ——单击鼠标左键。

（2）click\_and\_hold(on\_element=None) ——点击鼠标左键，不松开。

（3）context\_click(on\_element=None) ——点击鼠标右键。

（4）double\_click(on\_element=None) ——双击鼠标左键。

（5）drag\_and\_drop(source, target) ——在source源元素上按下鼠标左键，然后移动到target目标元素上释放。

（6）drag\_and\_drop\_by\_offset(source, xoffset, yoffset) ——拖拽到某个坐标然后松开。

（7）move\_by\_offset(xoffset, yoffset) ——鼠标从当前位置移动到某个坐标。

（8）move\_to\_element(to\_element) ——鼠标移动到某个元素。

（9）move\_to\_element\_with\_offset(to\_element, xoffset, yoffset) ——移动到距某个元素（左上角坐标）多少距离的位置。

（10）release(on\_element=None) ——在某个元素位置松开鼠标左键。

（11）send\_keys(\*keys\_to\_send) ——发送某个键到当前焦点的元素。

（12）send\_keys\_to\_element(element, \*keys\_to\_send) ——发送某个键到指定元素。

（13）perform() ——执行以上所有动作。

使用ActionChains类首先需要实例化，实例化必须需要实例化浏览器作为参数，方法为action = ActionChains(browser)（括号中需要输入参数，在括号中Ctrl+p会给予提示）。ActionChains提供的所有鼠标操作的方法，在调用的时候所有的行为都会存储在ActionChains类中，必须调用perform()方法才能执行。例如在元素element上悬停的操作为：action.move\_to\_element(element).perform()。

## Webdriver键盘操作

Webdriver键盘操作的方法封装在Keys类中，使用时需要从webdriver中导入，from selenium.webdriver.common.keys import Keys，键盘常用的操作方法：

（1）key\_down(value, element=None)——按下某个键盘上的键

（2）key\_up(value, element=None)——松开某个键

（3）send\_keys(Keys.BACK\_SPACE)——删除键（backspace）(执行一次只模拟一次按键)

（4）send\_keys( Keys. SPACE)——空格键（space）

（5）send\_keys( Keys.TAB)——制表键（Tab）

（6）send\_keys( Keys. ESCAPE)——回退键（esc）

（7）send\_keys( Keys. ENTER)——回车键（enter）

（8）send\_keys(Keys.CONTROL,’a’)——全选（ctrl+A）

（9）send\_keys(Keys.CONTROL,’c’)——复制（ctrl+C）(复制的前提是要选中内容才行)

（10）send\_keys(Keys.CONTROL,’x’)——剪切（ctrl+X）

（11）send\_keys(Keys.CONTROL,’v’)——粘贴（ctrl+v）

（12）send\_keys(keys.F1)——键盘F1

（13）send\_keys(keys.F12)——键盘F12

复制、粘贴：element.send\_keys(Keys.CONTROL,’c’)，element.send\_keys(Keys.CONTROL,’v')。当需要操作组合键时，send\_keys需要传入两个参数，例如ALT+X为send\_keys(Keys.ALT,’a’)。只操作一个建是，send\_keys只需要传入一个参数，send\_keys(Keys.a)。

# 元素等待

元素等待指的是WebDriver定位页面元素时如果没有找到，会在指定时间内一直等待的过程。设置元素等待是因为网络速度、电脑配置、服务器处理请求等原因，导致页面元素加载速度过慢。

## 固定等待

time包的sleep()方法，其机制就是让程序运行暂停几秒，例如sleep(5)让程序暂停5秒之后再往下执行。

## 元素的显式等待

显示等待指定义一个等待条件和一个最长等待时间，程序会判断在等待时间内条件是否满足，如果满足则执行后续代码，如果不满足会继续等待，超过时间就会抛出异常。WebDriver把显式等待的相关方法封装在WebDriverWait类中，而判断条件是否满足的方法封装在expected\_conditions类中，使用时需要导入：from selenium.webdriver.support.wait import WebDriverWait、from selenium.webdriver.support import expected\_conditions as EC。

WebDriverWait类是由WebDirver提供的等待方法，在设置时间内，默认每隔一段时间检测一次当前页面元素是否存在，如果超过设置时间检测不到则抛出异常。具体格式如下：

WebDriverWait(driver, timeout, poll\_frequency=0.5, ignored\_\_exceptions=None)：driver(浏览器驱动)、timeout(最长超时时间，默认以秒为单位)、poll\_\_frequency(检测的间隔时间，调用until或until\_not中的方法的间隔时间，默认是0.5秒)、ignored\_exceptions(忽略的异常，如果在调用until或until\_not的过程中抛出这个元组中的异常，则不中断代码，继续等待，如果抛出的是这个元组外的异常，则中断代码，默认为NoSuchElementException)。

WebDriverWait()一般由until()或until\_not()方法配合使用，until：当某元素出现或什么条件成立则继续执行，until\_not：当某元素消失或什么条件不成立则继续执行。EC(expected\_conditions的别名)使用presence\_of\_element\_located(element)判断元素是否存在，element为通过By类查找元素的方法，具体用法为：WebDriverWait(browser,10,poll\_frequency=1).until(EC.presence\_of\_element\_located((By.ID,’id’))

## 元素的隐式等待

隐式等待是通过一定的时长等待页面上某元素加载完成。如果超出了设置的时长元素 还没有被加载，则抛出 NoSuchElementException 异常。WebDriver 提供了implicitly\_wait() 方法来实现隐式等待，默认设置为0。隐式等待的作用域为全局元素，而显式等待的作用域为单个元素。隐式等待直接通过实例化浏览器对象进行调用即可，实际工作中我们一般都使用隐式等待，调用方法为browser.implicitly\_wait(timeout)，timeout为具体的超时时间，单位为秒。隐式等待并不是固定等待时间，与sleep()有本质的区别，sleep()就是等待多长时间然后再继续往下执行，而隐式等待只有在第一次未定位到元素时才会触发，然后以循环的方式去判断元素是否被找到，如果在指定的隐式等待时间内找到则往下继续执行，否则将抛出NoSuchElementException 异常。

# 下拉框、弹出框、滚动条操作

## 下拉框

下拉框就是HTML中的<select>元素，选项的<option>元素，一般通过option的标签及其属性可以定位选项，但这种定位有时候比较繁琐，WebDriver提供了Select类来定位复选框<select>下的选项<option>，有以下三种常用的方法：

1. select\_by\_index()——通过下拉选项的索引选择下拉选项。
2. select\_by\_value()——通过下拉选项的value值选择下拉选项。
3. select\_by\_visiable\_text()——通过下拉选项显示的文本信息选择下拉选项。

需要导入Select类：from selenium.webdriver.support.select import Select，使用Select类需要进行实例化，实例化的方法为Select(selectelement)(selectelement为<select>元素)，用法为Select(selectelement).select\_by\_index(num)。

## 弹出框

html中常用的弹出框有三种：alert、confirm、prompt，Webdriver中对处理弹出框的操作，有专用的处理方法：

text：获取弹出框的文本信息。

accept()：接受弹出框信息。

dismiss()：拒绝弹出框选项。

send\_keys(values)：发送文本到弹出框。

使用这些方法前需要获取弹出框，Webdriver获取弹出框的方法为：alert = driver.switch\_to.alert（切换到弹出框）。那么处理弹出框的方法为alert.text、alert.accept()、alert().dismiss()。

遇到一个情况：当处理完第一个弹出框时，随之弹出第二个弹出框，操作第二个弹出框也需要获取弹出框，但是这个时候直接如果在操作完第一个弹出框时就去获取第二个弹出框将会失败，而需要通过sleep()函数将代码运行停一小段时间再去获取。

## 滚动条

滚动条是一种控制显示范围的组件，在html页面中，页面的元素为动态显示，元素根据滚动条下拉才会被加载，元素只有被加载出来后才能被获取，所以需要控制滚动条。Webdriver类库中没有提供直接对滚动条进行操作的方法，但是提供了调用JavaScript的方法execute\_script()，所以可以通过调用JavaScript脚本来控制滚动条。window.scrollTo(x,y)为移动滚动条水平的距离x，垂直的距离为y，单位为像素，距离需要看实际的屏幕大小。具体用法如下：

JS = window.scrollTo(0,1000)：滚动条拉倒底部。

JS = window.scrollTo(0,0)：滚动条拉倒顶部。

执行为driver.execute\_script(JS)。

使用过程中发现不能控制弹出框的滚动条，而且发现该滚动条是在div里，然后通过网上查询控制div里滚动条的方法为：（1）document.getElementById('d1').scrollTop=100：通过scrollTop设置垂直滚动条到100的位置。（2）document.getElementById('d1').scrollLeft=200：通过scrollTop设置水平滚动条到200的位置。这里方法首先是通过方法查询到div元素，然后在对div元素进行操作，此方法值得借鉴，也看到了JavaScript的强大之处。

# frame表单以及多窗口切换

## frame表单切换

frame/iframe是HTML页面中的一种框架，主要作用是在当前页面中指定区域显示另一页面元素，就是常说的嵌入式表单。在Web应用中经常会遇到frame/iframe表单嵌套页面的应用，WebDriver只能在一个页面上对元素识别与定位，对于frame/iframe表单内嵌页面上的元素无法直接定位，这时就需要通过WebDriver提供的表单切换方法将当前定位的主体切换为frame/iframe表单的内嵌页面中。常见的方法如下：

driver.switch\_to.frame(‘name/id’)：切换到对应表单，参数为frame/iframe标签的name属性或者id属性。如果frame/iframe表单没有id、name属性，name先要定位到frame/iframe的元素，然后通过switch\_to(frame/iframe)切换到frame/iframe表单，或者通过frame/iframe的索引进行切换switch\_to.frame(‘index’)。

driver.switch\_to.default\_content()：恢复到默认界面，如果操作frame表单之外的元素，需要先回到默认的界面。

**driver.switch\_to.parent\_frame()：switch\_to中独有的方法，可以切换到上一层的frame，对于层层嵌套的frame很有用。**

## 多窗口切换

在HTML页面中，经常有些链接在点击时会在新的窗口去打开，这时候定位链接页面的元素则需要切换到新的窗口中才能进行定位，Webdriver中封装了获取当前窗口句柄、所有窗口句柄以及指定窗口句柄的方法（句柄：英文名为handle，窗口的唯一识别码）。方法如下：

driver.current\_window\_handle：获取当前窗口句柄。

driver.window\_handles：获取所有窗口句柄。

driver.switch\_to.window(handle)：切换到指定的句柄窗口。

（遇到一个问题：IE浏览器（其他浏览器未尝试过）获取句柄时，鼠标停留在窗口上会导致弹出框获取不到，鼠标移出浏览器的窗口即能获取到。）

# 验证码的处理

验证码是一种随机生成的信息（图片、数字、字母、汉字、算术题）等为了防止恶意的请求行为，增加应用的安全性。在web应用中，大部分系统在用户登录的时候应用验证码，所以设计自动化脚本时需要考虑到验证码的问题。

Webdriver类库中没有对验证码的处理方法，针对验证码有以下几种处理方式：

1. 去掉验证码：测试环境下采用。
2. 设置万能验证码：生产环境采用。
3. 验证码识别技术：通过Python-tesseract来识别图片类型验证码，但是识别率很难达到100%。
4. 记录cookie：通过记录cookie进行登录。

通过记录cookie的方式能够绕过登陆的步骤，所以在自动化测试中经常会使用记录cookie的这种方式去处理验证码，Webdriver提供了以下几种cookie处理方法：

get\_cookie(name)：获取指定的cookie。

get\_cookies()：获取本网站所有本地cookie。

add\_cookie(str)：添加cookie，str为python中的字典格式（键值对格式），例如str={“username” : ”admin” , ”password” : “000000”}

# UnitTest框架

UnitTest框架是用来执行代码测试的框架，UnitTest能够同时执行多个用例，提供了丰富的断言方法与测试结果展示，同时更加方便查看日志信息。UnitTest的核心要素为：TestCase、TestSuite、TextTestRunner、Fixture，使用UnitTest框架需要进行导入：import unittest。

## TestCase测试用例

TestCase即测试用例，在自动化测试中，一条用例就是一个完整的测试流程。新建测试类必须继承unittest.TestCase，测试方法的名称必须以test开头，unittest.TestCase类批量运行的方法是搜索执行test开头的方法，通过函数unittest.main()运行测试用例：

import unittest  
# 测试类必须继承unittest.TestCase类。  
class Test(unittest.TestCase):  
# 测试方法必须以test开头。  
 def testcase1(self):  
 print('测试用例1被执行！')  
 def testcase2(self):  
 print('测试用例2被执行！')  
 def testcase3(self):  
 print('测试用例3被执行！')  
if \_\_name\_\_ == '\_\_main\_\_':  
# 通过函数unittest.main()来运行test开头的方法。  
 unittest.main()

## Testsuite测试套件

Testsuite测试套件，多条测试用例集合在一起就是测试套件。测试套件是一个对象，实例化方法为：suite=unittest.TestSuite()。测试套件添加测试用例的方法如下：

（1）suite.addTest(classname(defname))：添加单个用例，classname为类名，defname为方法名。

（2）suite.addTest(Iterable[classname(defname)])：添加多个个用例，参数是类名(方法名)组成的一个可迭代的对象。

（3）suite.addTest(unittest.makeSuite(classname))：搜索指定classname内test开头的方法并添加到测试套件中。

suite测试套件添加了测试用例后，可以使用unittest.TextTestRunner()的run(suite)方法批量执行测试用例。

import unittest  
# 导入测试用例  
from 基础.UnitTest框架.Case.case1 import Case1  
from 基础.UnitTest框架.Case.case2 import Case2  
from 基础.UnitTest框架.Case.case3 import Case3  
if \_\_name\_\_ == '\_\_main\_\_':  
# 实例化测试套件对象：suite=unittest.TestSuite()。  
 suite=unittest.TestSuite()  
# 给测试套件添加单个测试用例：suite.addTest(classname(defname))。  
 suite.addTest(Case1('test1'))  
# 给测试套件添加多个测试用例：suite.addTest(Iterable[classname(defname)]),参数是类名(方法名)组成的一个可迭代的对象。  
 suite.addTests([Case1('test2'),Case1('test3')])  
# 批量添加test开头的用例：suite.addTest(unittest.makeSuite(classname))  
 suite.addTest(unittest.makeSuite(Case2))  
 suite.addTest(unittest.makeSuite(Case3))  
# 使用unittest.TextTestRunner的run()方法执行测试套件。  
 runner=unittest.TextTestRunner(verbosity=2)  
 runner.run(suite)

还有一种添加测试用例是通过路径的文件搜索test开头的方法：unittest.defaultTestLoader.discover(path,pattern=’filename’)。

import unittest  
if \_\_name\_\_ == '\_\_main\_\_':  
suite=unittest.defaultTestLoader.discover('./Case',pattern='case\*.py')  
 runner=unittest.TextTestRunner(verbosity=2)  
 runner.run(suite)

测试套件如果添加的用例模块中有Fixture以及@classmethod，那么在执行用例时这两个也会去作用。

## Fixture测试固件

Fixture测试固件，由setUP(self)方法与tearDown(self)方法组成，这两个方法在继承unittest.TestCase的类中才能发生作用。一般代码执行都是从上往下执行，setUP(self)的作用是让方法先执行，不管写在代码页的中间还是底部都是先执行，而tearDown(self)的作用是让方法最后执行。所以setUP(self)一般做初始化工作，比如实例化浏览器、浏览器最大化、隐式等待、登录系统等。tearDown(self)一般做结束工作，比如退出登陆，关闭浏览器。如果一个测试类中包含多个test开头的方法，那么每个test开头的方法执行前都会运行setUp，结束时都会运行tearDown。

from selenium import webdriver  
from time import sleep  
import unittest  
# Fxieture由setUp(self)/tearDown(self)组成。  
class Fixture(unittest.TestCase):  
 # setUp会最先执行。  
 def setUp(self):  
 self.browser=webdriver.Ie()  
 browser=self.browser  
 browser.maximize\_window()  
 browser.implicitly\_wait(5)  
 browser.get('http://localhost:9087')  
 # tearDown会最后执行。  
 def tearDown(self):  
 browser=self.browser  
 browser.switch\_to.frame('topFrame')  
 browser.find\_element\_by\_link\_text('退出系统').click()  
 alert=browser.switch\_to.alert  
 alert.accept()  
 sleep(3)  
 browser.quit()  
 # 每个test开头的方法执行都会先执行setUp()然后执行tearDown()。  
 def testLoin1(self):  
 browser=self.browser  
 browser.find\_element\_by\_id('username').send\_keys('admin')  
 browser.find\_element\_by\_id('password').send\_keys('000000')  
 browser.find\_element\_by\_tag\_name('button').click()  
 sleep(3)  
 def testLoin2(self):  
 browser=self.browser  
 browser.find\_element\_by\_id('username').send\_keys('admin')  
 browser.find\_element\_by\_id('password').send\_keys('000000')  
 browser.find\_element\_by\_tag\_name('button').click()  
 sleep(3)  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Assert断言

断言就是让程序来判断执行结果是否符合预期结果，UnitTest提供了三种类型的断言方法：布尔型断言、比较断言、复杂断言，使用这些断言方法必须要要继承unittest.TestCase，继承该类的实例化对象可以通过self.方法名称使用这些断言方法。

1. **布尔型断言**

assertEqual(a,b,msg=None)：检查a==b，不等则fail。

assertNotEqual(a,b,msg=None)：检查a!=b，相等则fail。

assertTrue(expr,msg=None)：验证expr是true，如果为false，则fail。

assertFalse(expr,msg=None)：验证expr是false，如果为true，则fail。

assertIs(a,b,msg=None)：验证a是b，如果不是，则fail。

assertIsNot(a,b,msg=None)：验证 a不是b，如果是，则fail。

assertIsNone(expr,msg=None)：验证expr是none，如果不是none，则fail。

assertIsNotNone(expr,msg=None)：验证expr不是none，如果是none，则fail。

assertIn(a,b,msg=None)：验证a是b的子串，如果不是，则fail。

assertNotIn(a,b,msg=None)：验证a不是b的子串，如果是，则fail。

1. **比较断言**

assertAlmostEqual (first, second, places = 7, msg = None, delta = None)：验证first约等于second。palces: 指定精确到小数点后多少位，默认为7；delta：first与second的差值不能大于delta。

assertNotAlmostEqual (first, second, places = 7, msg = None, delta = None)：验证first不约等于second。palces: 指定精确到小数点后多少位，默认为7。delta：first与second的差值不能大于delta。

assertGreater (first, second, msg = None)：验证first > second，否则fail。

assertGreaterEqual (first, second, msg = None)：验证first ≥ second，否则fail。

assertLess (first, second, msg = None)：验证first < second，否则fail。

assertLessEqual (first, second, msg = None)：验证first ≤ second，否则fail。

assertRegexpMatches (text, regexp, msg = None)：验证正则表达式regexp搜索匹配的文本text，regexp：通常使用re.search()。

assertNotRegexpMatches (text, regexp, msg = None)：验证正则表达式regexp搜索不匹配的文本text，regexp：通常使用re.search()。

1. **复杂断言**

assertListEqual(list1,list2,msg=None)：验证列表list1、list2相等，不等则fail，同时报错信息返回具体不同的地方。

assertTupleEqual(tuple1,tuple2,msg=None)：验证元组tuple1、tuple2相等，不等则fail，同时报错信息返回具体的不同地方。

assertSetEqual(set1,set2,msg=None)：验证集合set1、set2相等，不等则fail，同时报错信息返回具体的不同地方。

assertDictEqual(expected,actual,msg=None)：验证字典expected、actual相等，不等则fail，同时报错信息返回具体的不同地方。

以assertEqual(a,b,msg=None)为例，使用方法如下：

from selenium import webdriver  
from selenium.webdriver.support.select import Select  
from time import sleep  
import unittest  
from datetime import datetime  
class Test(unittest.TestCase):  
 def setUp(self):  
 self.browser=webdriver.Ie()  
 browser=self.browser  
 browser.maximize\_window()  
 browser.implicitly\_wait(5)  
 browser.get('http://localhost:9087')  
 browser.find\_element\_by\_id('username').send\_keys('admin')  
 browser.find\_element\_by\_id('password').send\_keys('000000')  
 browser.find\_element\_by\_tag\_name('button').click()  
 sleep(3)  
 def test(self):  
 try:  
 browser=self.browser  
 browser.switch\_to.frame('topFrame')  
 browser.find\_element\_by\_link\_text('集团报送').click()  
 browser.switch\_to.default\_content()  
 browser.switch\_to.frame('leftFrame')  
 browser.switch\_to.frame('leftiframe')  
 browser.find\_element\_by\_link\_text('信托上报信息').click()  
 browser.find\_element\_by\_id('itemTextLink9').click()  
 browser.switch\_to.default\_content()  
 browser.switch\_to.frame('content')  
 browser.find\_element\_by\_css\_selector('input[value=496]').click()  
 browser.find\_element\_by\_css\_selector('button[title="作废 "]').click()  
 alert=browser.switch\_to.alert  
 alert.accept()  
 sleep(3)  
 alert=browser.switch\_to.alert  
 alert.accept()  
 browser.find\_element\_by\_id('queryButton').click()  
 element=browser.find\_element\_by\_css\_selector('select[name="delete\_flag"]')  
 options=Select(element)  
 options.select\_by\_value('1')  
 browser.find\_element\_by\_name('btnQuery').click()  
 element=browser.find\_element\_by\_name('serial\_no')  
 value=element.get\_attribute('value')  
 self.assertEqual(value,'496','项目成员信息作废失败')  
 except Exception as E:  
 print(E)  
 browser.get\_screenshot\_as\_file(r'D:\自动化测试\失败%s.png'%datetime.now().strftime('%Y年%m月%d日 %H点%M分%S秒'))  
 else:  
 browser.get\_screenshot\_as\_file(r'D:\自动化测试\成功%s.png'%datetime.now().strftime('%Y年%m月%d日 %H点%M分%S秒'))  
 finally:  
 browser.switch\_to.default\_content()  
 def tearDown(self):  
 browser=self.browser  
 browser.switch\_to.frame('topFrame')  
 browser.find\_element\_by\_link\_text('退出系统').click()  
 alert=browser.switch\_to.alert  
 alert.accept()  
 sleep(3)  
 browser.quit()  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Html测试报告

Html测试报告就是执行完测试用例后以html方式将执行结果生成报告，生成Html报告的方法有两种：Export Test Results（unittest自带）、HTMLTestRunner（第三方模板），一般采用HTMLTestRunner。使用HTMLTestRunner前，需要下载HTMLTestRunner.py复制到Python中，然后从HTMLTestRunner中导入HTMLTestRunner类。用法如下：

from HTMLTestRunner import HTMLTestRunner  
from datetime import datetime  
import unittest  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 定义报告文件名。  
 filename=r'D:\自动化测试\%s'%datetime.now().strftime('%Y年%m月%d日 %H点%M分%S秒')+'Report.html'  
 # 添加测试套件。  
 suite=unittest.defaultTestLoader.discover('./Case',pattern='case\*

.py')  
 with open(filename,'wb') as file:  
 # 实例化runner，并执行测试套件。  
 runner=HTMLTestRunner(stream=file,title='测试报告',description='测试内容')  
 runner.run(suite)

## classmethod装饰器

如果想在全部案例执行之前先执行一个案例以及全部案例执行完毕完毕后再执行一个案例，则可以使用unittest的一个装饰器@calssmethod，用法如下：

import unittest  
class Test(unittest.TestCase):  
 @classmethod  
 # 前置函数。  
 def setUpClass(cls):  
 print('全部测试案例执行前执行！')  
 @classmethod  
 # 后置函数。  
 def tearDownClass(cls):  
 print('全部测试案例执行完毕后再执行！')  
 def setUp(self):  
 print('每个案例的前置函数执行！')  
 def tearDown(self):  
 print('每个案例的后置函数执行！')  
 def test1(self):  
 print('执行测试1')  
 def test2(self):  
 print('执行测试2')  
 def test3(self):  
 print('执行测试3')  
if \_\_name\_\_ =='\_\_main\_\_':  
 unittest.main()

## unittest.skip装饰器

unittest框架中的测试用例执行时都是批量全部执行的，使用@unittest.skip()可以让某一条用例不执行，只需要在用例前加上@unittest.skip()。

import unittest  
class Test(unittest.TestCase):  
 def testcase1(self):  
 print('测试用例1被执行！')  
 # 屏蔽用例的执行  
 @unittest.skip('先屏蔽一下')  
 def testcase2(self):  
 print('测试用例2被执行！')  
 def testcase3(self):  
 print('测试用例3被执行！')  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

执行时将会屏蔽testcase2，把用例添加到suite里，执行suite同样也会屏蔽testcase2。