# 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。

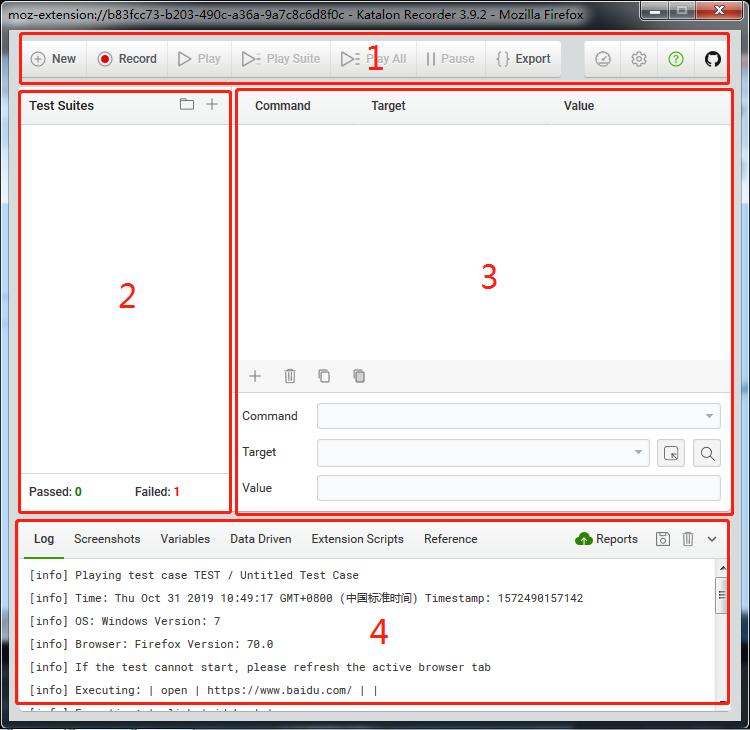
自动化测试可以在较少的时间内运行更多的测试用例，在短时间内可以进行重复性校验。但自动化测试不能完全代替手动测试，手工测试比自动化测试更能发现软件的缺陷，往往很多缺陷通过自动化测试难以发现。自动化测试是在完成手工测试之后才能进行，一般来讲周期较长、需求变动不频繁、需要进行回归测试的项目比较适合做自动化测试。

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：

from selenium import webdriver(导入webdriver类)

driver=webdriver.Ie()(实例化浏览器打开Ie浏览器)

driver=webdriver.FireFox()(实例化浏览器打开火狐浏览器)

driver=webdriver.Chrome()(实例化浏览器打开谷歌浏览器)

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  
  
options = webdriver.ChromeOptions()  
options.binary\_location=r'E:\常用软件\360浏览器\360se6\Application\360se.exe'  
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。

## 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：

1.from selenium import webdriver（从selenium包中导入webdriver）

2.browser = webdriver.Ie()（实例化浏览器）

3.browser.get(‘http://lcoalhost:7001/’)（打开网页）（扩展：如果是本地文件路径地址，例如D:\自动化测试\web自动化测试视频selenium\day1，由于\是转义符，所以需要填写的URL为’D:\\自动化测试\\web自动化测试视频selenium\\day1’。或者是r’D:\自动化测试\web自动化测试视频selenium\day1’，r的作用是让转义符失效）

4.username = browser.find\_element\_by\_id(‘username’)（使用id的定位方法查找元素）

5.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。

## 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。

## 拓展

**（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’)

对比（1）跟（2）其实会发现，By类里面的NAME就是等于name，同理PARTIAL\_LINK\_TEXT就是等于partial link text。其实find\_element\_by\_方法名()源代码里是find\_elemen(by=By.方法名，value=参数值)，find\_element\_by\_方法名()封装了find\_element(By.方法名,‘‘)这个方法，find\_element(By.方法名,‘‘)封装了find\_element(‘方法‘,‘参数值‘)这个方法。

**（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。

## JS与jQuery定位

**（1）JS定位**

id定位document.getElementById()、name定位document.getElementsByName()、css定位document.querySelectorAll()、tag定位document.getElementsByTagName()、class定位document.getElementsByClassName()。

只有id对象用的是Element返回是单个对象，其他都是Elements返回的是一个list，需要标注下标。具体的用法是先写好对应的js语句，可以先赋值给一个变量，然后后调用execute\_script进行执行一下js语句即可，用法如下：

username = "document.getElementsById('username')[0].value='admin'"

password = "document.getElementByName('password').value='000000'"

button= "document.getElementsByTagName('button')[0].click()"

browser.execute\_script(username)

browser.execute\_script(password)

browser.execute\_script(button)

**（2）jQuery**

jQuery语法是为HTML元素的选取编制的，可以对元素执行一些具体的操作，基础语法是$(selector).action()。$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()在jQuery中有这么几种：1.$(selector).val('input\_value')，输入操作，其中input\_value表示要输入的文本的值；2.$(selector).click()，点击操作。用法如下：

username = "$('#username).val(admin)"

password = "$('#password).val(000000)"

button= "$('button).click()"

browser.execute\_script(username)

browser.execute\_script(password)

browser.execute\_script(button)

# 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()：单击元素。

例如单击元素：button.click()

## 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。
9. get\_screenshot\_as\_file(r"D:\测试工作\baidu.png")：截取当前页面图并保存为。

例如浏览器刷新：browser.refresh()。

## 其他操作方法

（1）size：获取元素的尺寸。

（2）text：获取元素的文本。

（3）get\_attribute(name)：获取属性的值。

（4）location：获取元素坐标。

（5）is\_displayed() ：元素是否可见。

（6）is\_enabled()：元素是否可使用。

（7）is\_selected() ：元素是否被选中。

（8）tag\_name：元素的标签名。

（9）submit()：提交表单。

（10）title：获取页面的标题。

（11）current\_url：获取当前页面的URL。

（12）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) ——拖拽到某个元素然后松开。

（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()——根据选项<option>的索引进行定位。
2. select\_by\_value()——根据选项<option>的属性value值进行定位。
3. select\_by\_visiable\_text()——根据选项<option>显示的文本进行定位。

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：返回alert、confirm、prompt中的文字信息。

accept()：接受弹出框选项。

dismiss()：取消/拒绝弹出框选项。

send\_keys(keysToSend)：发送文本到警告框。

使用这些方法前需要找到弹出框，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()。

## 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，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)。

# 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。

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。

# 窗口截图、验证码的处理

## 窗口截图

自动化测试由程序去执行，因此有时候打印错误的信息并不是十分明确。如果出错的时候对当前页面进行截图保存，那么可以通过图片就非常直观得看出出错的原因，截图的方法为driver.get\_screenshot\_as\_file(D:\截图\图片.png)

## 验证码

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

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，因为unittest.TestCase类批量运行的方法是搜索执行test开头的方法：

import unittest

class test(unittest.TestCase):

def test1(self):

print('11111')

def test2(self):

print('22222')

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## Testsuite

Testsuite即测试套件，多条测试用例集合在一起就是测试套件，使用方法如下：

实例化：suite=unittest.TestSuite()，suite为测试套件的实例化名称。

添加用例：suite.addTest(classname(methonname))，classname为类名，methonname为方法名。

添加扩展：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。

TestSuite需要配合TextTestRunner才能被执行，执行的方法为：runner=unittest.TextTestRunner，runner(suite)。

## TextTestRunner

TextTestRunner用来执行测试套件，实例化为runner=unittest.TextTestRunner，执行为runner.run(suite)。

## Fixture

Fixture即测试固件，对测试用例环境的搭建与销毁就是一个Fixture，使用方法如下：

初始化（搭建）：def setUp(self)，首先执行，setUp方法继承于unittest.TestCase。

结束（销毁）：def tearDown(self)，最后执行，tearDown方法继承于unittest.TestCase。

一般代码执行都是从上往下执行，setUp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的作用是让方法后执行。所以setUP一般做初始化工作，比如实例化浏览器、浏览器最大化、隐式等待设置等。tearDown一般做结束工作，比如退出登陆，关闭浏览器。这两个方法的使用都要继承unittest.TestCase类，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。

## DefaultTestLoader

使用unittest.defaultTestLoader()类的discover()方法自动搜索指定目录下指定开头的.py文件，并将找到的.py文件组装到测试套件中。用法为：suite=unittest.defaultTestLoader.discover(“./TestCase”,pattern=”test\*.py”)，意思是查找同级目录的TestCase下以test开头的.py文件的test开头的方法组装到suite测试套件中。（../为上一级目录，.为当前目录）

## UnitTest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。

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。

（2）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()。

（3）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：

#抓取异常

try:

assertEqual(1,2)

#如果抓取到断言异常那么执行以下操作

except AssertionError:

#获取时间戳

Nowtime=time.strftime(“%Y\_%m\_%d\_%H\_%M\_%S”)

#获取错误信息

Error=sys.exc\_info()[1]

#截图

Driver.get\_screenshot\_as\_file(“./image/%s--%s.png”%(Nowtime,Error))

#抛出错误

raise AssertionError

## 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：

#添加测试套件

suite=unittest.defaultTestLoader.discover(“./TestCase”,pattern=”test\*.py”)

#报告的路径及名称

File\_name=dir\_path(文件路径)+nowtime(时间戳)+”Report.html”

#写入报告(wb二进制)

with open(File\_name,”wb”) as f:

runner=HTMLTestRunner(stream=f,title=’测试报告’,description=’说明’)

runner.run(suite)