

Web自动化——Selenium+Unittest

一、自动化测试的基本概念

1.1 什么是自动化测试？

自动化测试是把以人为驱动测试行为转化为机器执行的一种过程。

自动化测试的目的是节省人力、时间和硬件资源，提高测试效率，保障软件的质量。

生活中的自动化例子：

- 洗衣机洗衣服
- 电饭煲煮饭
- 自动门
- 自动开灯
- 工厂里面的机器
- 工厂里面汽车组装
-

发展：人工完成相关操作 -----> 机器代替手工（效率高、不怕累，准确率高）

1.2 自动化的类型

当前软件公司里开展的自动化主要包括UI自动化、接口自动化和性能自动化三种，其中UI自动化又包括Web自动化和App自动化。

- UI自动化
 - Web自动化：针对Web项目，利用代码来操作Web页面的元素从而对Web功能进行自动化测试
 - App自动化：针对App项目，利用代码来操作App页面的元素从而对App功能进行自动化测试
- 接口自动化：利用代码来实现对后台接口的功能进行自动化测试
- 性能自动化：利用代码来实现对后台的性能进行自动化测试（1.使用代码模拟大批量用户，让用户并发请求；2.多页面多用户并发请求；3.采集参数，统计系统负载能力；4.生成报告）

1.3 什么样的项目适合做（需要做）自动化？

1. 测试任务明确，不会频繁变动
2. 每日构建后的测试验证
3. 比较频繁的回归测试
4. 软件系统界面稳定，变动少
5. 需要在多平台上运行的相同测试案例、组合遍历性的测试、大量的重复任务
6. 软件维护周期长
7. 项目进度压力不太大
8. 被测软件系统开发比较规范，能够保证系统的可测试性
9. 测试人员具备较强的编程能力（前提）

1.4 自动化能不能完全代替手工？

不能，因为迭代过程中，刚开发出来的功能还不稳定，可能会有很多问题，如果用自动化测试，那么有可能要经常修改自动化代码，浪费人力物力。

1.5 自动化测试适用于项目的哪些阶段？

回归测试（比如：版本发布前对已实现有关联的功能进行测试、对已实现的功能定期测试）。

1.6 自动化需要写用例吗？可以直接用手动测试的用例吗？

一般来说，自动化测试的用例可以参考手动测试的用例，但是需要设计特定的用例模板来组织测试用例，选取合适的数据文件格式存放测试数据供代码读取。

我们可以抽取手动测试的用例来开发成自动化测试用例。在工作中，有可能是对系统的部分功能实现自动化测试，手动+自动化同时进行。

抽取原则1：用户使用频繁的功能点（比如淘宝系统的下订单、支付、搜索）

抽取原则2：基本稳定、变更不大的功能点

1.7 自动化脚本谁来写和维护，怎么管理？

自动化测试脚本是由测试来写和维护的。

代码管理工具：重点是Git（管理代码）、SVN（一些测试文档，比如测试用例等）

1.8 自动化测试工具有哪些？

- Selenium
- Requests
- Appium
-

二、Web自动化环境搭建-Python+Selenium库

2.1 Selenium介绍

Selenium 是一个用于Web应用程序测试的工具（是一款免费的、开源的、基于web页面的UI自动化测试工具）。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（7, 8, 9, 10, 11),Firefox, Safari, Google Chrome, Opera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——创建回归测试检验软件功能和用户需求是否一致。支持自动录制动作和自动生成 .Net、Java、Perl、Python等不同语言的测试脚本。补：它提供了一套完善的测试函数，功能非常灵活，能够完成界面元素的定位、窗口的跳转、结果的比较（引入unittest进行断言）等

selenium实现网页的控制操作主要是通过控制前端的元素来完成，在这个过程中，元素的定位是基础，只要准确抓取到对应元素才能进行后续的自动化控制

2.2 官网和发展

官网: <https://www.selenium.dev/>

支持的语言: Python、Java、Ruby、PHP等等

跨平台、跨浏览器

selenium1.0: selenium IDE (录制和回放脚本)、selenium Grid (多浏览器运行)、selenium Remote Control (RC) (核心, 主要是用来跟浏览器做交互)

selenium2.0 = selenium1.0 + WebDriver

selenium3.0是在selenium2.0的基础上改进, 去掉了RC, 全面拥抱了webdriver, 从此webdriver一统江湖

2.3 搭建自动化环境

环境: PyCharm+Python+Selenium+Unittest (从来写测试用例的) +谷歌浏览器+浏览器驱动

- IDE工具: PyCharm
- 语言: Python
- Web自动化工具: Selenium (它其实就是python中的一个第三方包)
 - 可以在dos下通过pip install selenium来安装, 也可以在PyCharm上安装
- 浏览器: 谷歌Chrome、火狐、IE (可选)
- 浏览器驱动: 与浏览器的类型和版本号保持一致, 下载对应的驱动包放到任何一个环境变量Path的路径下即可, 建议放到Python的安装目录。(在selenium 3.x版本中没有默认的浏览器的支持, 需要使用哪个浏览器完成自动化, 就需要获取官方提供的对应版本的驱动,)
 - Chrome: <http://chromedriver.storage.googleapis.com/index.html>
 - Firefox: <https://github.com/mozilla/geckodriver/releases/>
 - IE: <http://selenium-release.storage.googleapis.com/index.html>
 - Opera: <https://github.com/operasoftware/operachromiumdriver/releases>

我们使用Selenium实现自动化测试, 主要涉及3个东西:

1. 测试脚本, 可以是python, java编写的脚本程序 (也可以叫做client端);
2. 浏览器驱动, 这个驱动是根据不同的浏览器开发的, 不同的浏览器使用不同的Webdriver驱动程序且需要对应相应的浏览器版本, 比如: chromedriver.exe(Chrome);
3. 浏览器, 目前Selenium支持市面上大多数浏览器, 如: 火狐, 谷歌, IE等。

2.4 演示代码

演示代码:

```
#导入webdriver包
from selenium import webdriver
#实例化谷歌浏览器对象
driver = webdriver.Chrome()
#浏览器窗口最大化
driver.maximize_window()
#打开百度首页
driver.get('https://www.baidu.com')
```

2.5 Selenium的运行原理

简单说就是：模拟用户对浏览器进行操作

导入selenium第三方包并编写脚本后（运行）发送请求给浏览器驱动，它来解析这些自动化测试的代码，解析后把它们发送给浏览器，浏览器接受并执行驱动发来的指令，并最终完成工程师想要的操作

2.6 浏览器驱动

接受并解析自动化测试的代码（脚本），解析后把它们发送给浏览器，从而驱动浏览器执行相关指令，完成工程师想要的操作

三、元素定位的8大基本方法

3.1 什么是元素定位

元素定位就是根据元素的**某个特征**在网页中找到对应的元素，定位元素的目的是为了操作元素，在Web自动化中，定位元素是后续一切操作的前提条件。

自动化测试前提条件：需要识别HTML页面上的信息---也就是我们所说的元素；所以做自动化测试需要我们首先去定位到我们要找的元素（元素由属性和属性值组成），然后就是模拟用户点击（click）、模拟用户输入信息（send keys）

Selenium中有8大基本的元素定位方法：

- find_element (By.ID,属性值)
- find_element (By.NAME, 属性值)
- find_element (By.CLASS_NAME, 属性值)
- find_element (By.TAG_NAME,属性值)
- find_element (By.LINK_TEXT,属性值)
- find_element (By.PARTIAL_LINK_TEXT, 属性值)
- find_element (By.XPATH,属性值)
- find_element (By.CSS_SELECTOR, 属性值)

以上8个方法，会根据传入的条件，在整个页面中查找对应的元素，找到第一个元素（对应元素）后就返回，不再往下面继续找。所以在指定条件时要准确（确保唯一性），否则会定位不到元素。

3.2 通过id定位元素

- 方法名：find_element (By.ID,属性值)
- 作用：通过元素的id值来定位元素，一般情况下元素的id是唯一的，所以通过这个方法可以很轻松地找到元素。find_element (By.ID) 方法的返回值是一个WebElement对象，这个对象对应的就是Web页面上对应的元素。

- 举例：

如下元素的id是kw

```
<input type="text" class="s_input" name="wd" id="kw" maxlength="100" autocomplete="off">
```

```
# 通过id定位这个元素并输入Selenium
driver.find_element_by_id("kw").send_keys("Selenium")
```

- 有些特性情况，不能通过id来定位元素，比如：
 1. 有的元素的id是动态变化的，动态id一般由固定部分+动态部分组成，一般情况下元素的id是一个有具体含义的，看到带数字的id就要怀疑可能是动态的。
动态id举例：163邮箱登录页面的输入框的id，<https://mail.163.com/>

```
<input data-placeholder="邮箱帐号或手机号码" name="email" data-type="email" data-loginname="loginEmail" data-required="true" class="j-inputtext dlemail j-nameforslide" type="text" autocomplete="off" tabindex="1" spellcheck="false" id="auto-id-1637386500727" placeholder="邮箱帐号或手机号码" style="width: 201px;">
```
 2. 有些Web元素也可能没有id，这种情况就要换成其它方法来定位。
 3. 有可能id的属性值不唯一（重复）

3.3 通过name（名称）定位元素

- 方法名：find_element (By.NAME, '属性值')
- 作用：通过元素的名称来定位元素，一般情况下元素的名称也是唯一的，所以通过名称可以找到元素。find_element (By.NAME, '属性值') 方法的返回值是一个WebElement对象。
- 举例：

如下元素的名称是wd

```
<input type="text" class="s_ip" name="wd" id="kw" maxlength="100" autocomplete="off">
```

```
# 通过name定位这个元素并输入WebDriver
driver.find_element_by_name("wd").send_keys("WebDriver")
```

3.4 通过class（类名）定位——类名定位：一般用来定位一组元素

- class 属性表示元素的类名，指向样式表中的类，一个元素可以有一个或多个类名，一个类名也可能对应多个元素。当元素有多个类名时，类名与类名之间用间隔符隔开(但是定位时需要使用点号代替间隔符)。
- 方法：find_element (By.CLASS_NAME, '属性值')
- 作用：根据元素的类名定位元素，需要注意的是作为定位条件的类名必须是唯一的，否则会定位不到想找的元素，当元素有多个类名时，选择其中一个唯一的类名来定位。
find_element (By.CLASS_NAME, '属性值') 方法的返回值是一个WebElement对象。
- 举例：

如下元素的类名是s_ip

```
<input type="text" class="s_ip" name="wd" id="kw" maxlength="100" autocomplete="off">
```

```
# 通过类名来定位这个元素，并输入Selenium WebDriver
driver.find_element_by_class_name("s_ip").send_keys("Selenium WebDriver")
```

如下元素有4个类名，我们可以选择其中一个唯一的类名s_btn来定位元素

```
<input type="submit" value="百度一下" id="su" class="btn self-btn bg s_btn">
```

```
# 选择其中一个唯一的类名s_btn来定位元素
driver.find_element_by_class_name('s_btn').click()
```

3.5 通过tag_name（标签名称）定位——标签定位，一般用来定位一组元素

- 方法：find_element (By.TAG_NAME,属性值)
- 作用：根据元素的标签名称来定位元素，因为一个页面上往往有多个同名的标签，所以这种方法的局限性较大。find_element (By.TAG_NAME,属性值) 方法的返回值是一个WebElement对象。
- 举例：

<http://106.13.12.109:8080/korei/login.jsp> 登录页面

如下元素的标签名是input

```
▼ <div class="land_left">
  <p class="land_title">科睿教务管理系统</p>
  ▼ <div class="land_inputSite">
    <span class="land_icon01"></span>
    <span class="land_line"></span>
    <input type="text" class="land_input" name="username">
  </div>
```

```
# 根据标签名input来定位这个元素
driver.find_element_by_tag_name('input').send_keys("admin")
```

3.6 通过链接文本定位

- 方法：find_element (By.LINK_TEXT, 属性值)
- 作用：根据元素的链接文本来定位元素，适用于超链接元素，超链接元素的标签是a，在一个页面上，超链接文本一般是唯一的。find_element_by_link_text方法的返回值是一个WebElement对象。
- 举例：

```
▼ <div id="s-top-left" class="s-top-left s-isindex-wrap">
  <a href="http://news.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">新闻</a>
  <a href="https://www.hao123.com" target="_blank" class="mnav c-font-normal c-color-t">hao123</a>
  <a href="http://map.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">地图</a>
  <a href="https://live.baidu.com/" target="_blank" class="mnav c-font-normal c-color-t">直播</a>
  <a href="https://haokan.baidu.com/?sfrom=baidu-top" target="_blank" class="mnav c-font-normal c-color-t">视频</a>
  <a href="http://tieba.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">贴吧</a>
  <a href="http://xueshu.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">学术</a>
```

```
#定位文本是新闻的超链接元素
driver.find_element_by_link_text("新闻").click()
```

3.7 通过部分链接文本定位（模糊文本定位）

- 方法：find_element_by_partial_link_text(部分链接文本)
- 作用：根据元素的链接文本的一部分来定位元素，要注意选取的部分文本要唯一，否则定位不到需要的元素。find_element_by_partial_link_text方法的返回值是一个WebElement对象。
- 举例：

```
<div id="s-top-left" class="s-top-left s-isindex-wrap">
  <a href="http://news.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">新闻</a>
  <a href="https://www.hao123.com" target="_blank" class="mnav c-font-normal c-color-t">hao123</a>
  <a href="http://map.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">地图</a>
  <a href="https://live.baidu.com/" target="_blank" class="mnav c-font-normal c-color-t">直播</a>
  <a href="https://haokan.baidu.com/?sfrom=baidu-top" target="_blank" class="mnav c-font-normal c-color-t">视频</a>
  <a href="http://tieba.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">贴吧</a>
  <a href="http://xueshu.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">学术</a>
```

#定位文本包含闻的超链接元素

```
driver.find_element_by_partial_link_text("闻").click()
```

3.8 通过XPath路径定位

3.8.1 什么是XPath路径表达式

一句话：XPath 是一门在 XML 文档中查找信息的语言，XPath 使用路径表达式来选取 XML 文档中的节点或节点集。我们可通过XPath路径表达式在HTML中定位元素。**通俗一点讲就是通过元素的路径来查找到这个标签元素**

方法：find_element(BY.XPATH,路径表达式的属性值)

3.8.2 XPath路径表达式的语法

3.8.2.1 绝对路径

- 格式：与Linux的绝对路径相似，XPath绝对路径以/（根目录）开始写，第一层是/html，依次往下写，直到要定位的标签，比如/html/body/form/div/div/div/div/input
- 举例：

<http://106.12.70.245:8080/korei/login.jsp> 登录页面

#通过XPath定位用户名输入框

```
driver.find_element_by_xpath("/html/body/form/div/div/div/div/input").send_keys("admin")
```

注意：当父标签下有多个相同的子标签，可通过索引进行选择，索引值是从1开始，与python中的索引值从0开始不一样。

3.8.2.2 相对路径+属性

- 格式：//标签名[@属性名=属性值]；其中，//表示当前页面的任意目录下，因为同一个页面，往往同名的标签很多，所以经常把相对路径+属性组合起来使用
- 举例：

```
#定位百度首页输入框，输入Python
driver.find_element_by_xpath('//input[@id="kw"]').send_keys("Python")
#定位百度首页输入框，输入Selenium
driver.find_element_by_xpath('//input[@name="wd"]').send_keys("Selenium")
#定位百度一下按钮，点击
driver.find_element_by_xpath('//input[@id="su"]').click()
```

3.8.2.3 相对路径+层级

- 格式：绝对路径是完全依靠层级定位，相对路径是标签+属性定位，我们也可以把路径+属性+层级组合来定位，比如说有的元素它本身不好定位，但它的上一级或者上上一级定位比较方便，这种情况下就可以通过先定位上一级元素再通过上一级来定位你想定位的元素。
- 举例：

```
#定位百度输入框，输入Python
driver.find_element_by_xpath("//span[@id='s_kw_wrap']/input[1]").send_keys("Python")
#定位百度一下按钮，点击
driver.find_element_by_xpath("//form[@id='form']/span[2]/input[1]").click()
```

3.8.2.4 使用逻辑运算符

- 格式：//标签名[@属性名1=属性值1 and @属性名2=属性值2]；如果通过一个属性不能准确地定位到元素，可以使用逻辑运算符and来指定多个属性定位元素。
- 举例：

```
#定位百度输入框，输入Python
driver.find_element_by_xpath("//input[@id='kw' and @name='wd']").send_keys("Python")
#定位百度一下按钮，点击
driver.find_element_by_xpath("//input[@id='su' and @value='百度一下']").click()
```

3.8.2.5 文本内容

- 格式：//标签名[text()='文本']，如果一个标签有文本，并且文本在页面中是唯一的，可以通过指定文本来定位元素，比如//a[text()='新闻']
- 举例：

```
<div id="s-top-left" class="s-top-left s-isindex-wrap">
  <a href="http://news.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">新闻</a>
  <a href="https://www.hao123.com" target="_blank" class="mnav c-font-normal c-color-t">hao123</a>
  <a href="http://map.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">地图</a>
  <a href="https://live.baidu.com/" target="_blank" class="mnav c-font-normal c-color-t">直播</a>
  <a href="https://haokan.baidu.com/?sfrom=baidu-top" target="_blank" class="mnav c-font-normal c-color-t">视频</a>
  <a href="http://tieba.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">贴吧</a>
  <a href="http://xueshu.baidu.com" target="_blank" class="mnav c-font-normal c-color-t">学术</a>
```

```
#定位文本是新闻的标签
driver.find_element_by_xpath("//a[text()='新闻']").click()
```


3.8.2.6 模糊定位

- 格式：
 - //标签名[contains(@属性名,属性对应的部分值)]
 - //标签名[starts-with(@属性名,属性对应的部分值)]
 - //标签名[ends-with(@属性名,属性对应的部分值)]———已被移除

注意：分析属性值是否是动态的，是否由动态+固定值所构成的；如果分析出动态值中的部分固定值则可以使用contains完成

- 举例：//input[contains(@id,'kw')]

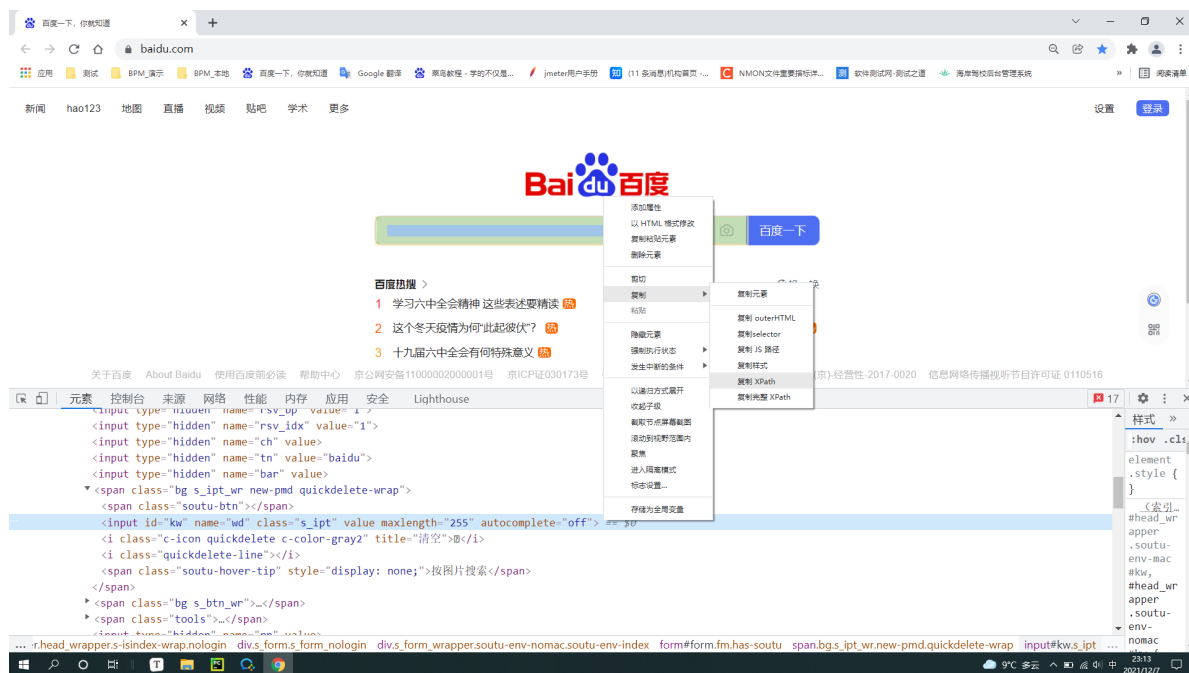
- 注意：ends-with是xml2.0的语法，有的浏览器不支持。

- 面试题：对于动态属性值的元素怎么定位？

首先要分析出动态值是否是由固定值+动态值两部分组成，如果是，那就可以用XPath中的模糊定位来实现，contains、starts-with。

3.8.2.7 通过浏览器生成XPath表达式

可以在浏览器开发者工具界面，选择元素，右键选择Copy，选择Copy XPath来自动生成XPath路径表达式。这可以作为一种辅助手段，有时复制出来的可能也不对，所以最终必须掌握手写XPath表达式的能力。



3.9 通过CSS选择器定位

3.9.1 什么是CSS选择器表达式

CSS选择器表达式可以实现对HTML页面中的元素实现一对一、一对多或者多对一的控制。

方法：find_element_by_css_selector()

3.9.2 CSS选择器表达式语法

3.9.2.1 通过CSS id属性值定位

- 语法: #代表id, 比如#kw
- 举例:

```
#定位百度一下输入框, 输入Python
driver.find_element_by_css_selector("#kw").send_keys("Python")
```

3.9.2.2 通过CSS class属性值定位

- 语法: .类名, 如果一个元素有多个类名, 可以用.类名1.类名2.类名n来定位, 也可以选择其中一个唯一的类名来定位
- 举例:

```
#定位百度输入框, 输入Python
driver.find_element_by_css_selector(".s_ip").send_keys("Python")
#定位百度一下按钮, 这个元素有多个class值, 可以指定多个类名定位, 也可以选择其中一个唯一的类名来定位
driver.find_element_by_css_selector(".btn.self-btn.bg.s_btn").click()
driver.find_element_by_css_selector(".s_btn").click()
```

3.9.2.3 通过其它属性值定位

- 语法: [属性名=属性值]
- 举例:

```
#定位百度输入框, 输入Python
driver.find_element_by_css_selector("[name='wd']").send_keys("Python")
```

3.9.2.4 通过标签名定位

- 语法: 直接写标签名, 前面不需要加任何符号 (会存在很多同名的标签, 不建议单独使用, 一般结合其他属性进行定位)
- 举例:

```
#定位百度输入框, 输入Python
driver.find_element_by_css_selector("input").send_keys("Python")
```

3.9.2.5 通过标签名+id组合定位 (常用)

- 语法: 标签名#id属性值
- 举例:

```
#定位百度输入框，输入Python
driver.find_element_by_css_selector("input#kw").send_keys("Python")
```

3.9.2.6 通过标签名+class组合定位

- 语法：标签名.类名
- 举例：

```
#定位百度输入框，输入Python
driver.find_element_by_css_selector("input.s_ipt").send_keys("Python")
```

3.9.2.7 使用标签名+其它属性组合定位

- 语法：标签名[属性名=属性值]，id和class也可以用这种形式写
- 举例：

```
#定位百度输入框，输入Python
driver.find_element_by_css_selector("input[name='wd']").send_keys("Python")
```

3.9.2.8 使用多个属性组合定位

- 语法：标签名[属性名1=属性值1][属性名2=属性值2]；如果通过一个属性不好准确定位到元素，也可以通过多个属性值定位；
- 举例：

```
#定位百度输入框，输入Python
driver.find_element_by_css_selector("input[name='wd']
[type='text']").send_keys("Python")
```

3.9.2.9 通过页面元素的属性值的一部分定位（模糊匹配）

- 语法：
 - 属性名^='XX'：表示以XX开头
 - 属性名\$='XX'：表示以XX结尾
 - 属性名*='XX'：表示包含XX；
- 举例：

```
#定位百度一下按钮，点击
driver.find_element_by_css_selector("input[value*='百度']").click()
driver.find_element_by_css_selector("input[value^='百度']").click()
driver.find_element_by_css_selector("input[value$='一下']").click()
```

3.9.2.10 通过页面元素定位子元素

- 语法：如果直接定位某个元素不好定位，可以尝试先找它的上级元素或上上级元素，再找这个元素，在CSS中用>表示层级。

css的绝对路径是从html标签开始的，比如html>body>form>div>div>div>div>input，这种方式类似于XPath的绝对路径，在实际使用中不可取。

表示某个元素下的第x个子元素的方法：

- nth-child(x)：A>B:nth-child(x)——表示A标签下面的所有子标签中的第x个并且标签名是B的子元素，可以用，**第一个也可以用first-child表示，最后一个也可以用last-child表示。**
- nth-of-type(x)：表示A标签下面的所有B标签中的第x个，可以用A>B:nth-of-type(x)，第一个也可以用first-of-type表示，最后一个也可以用last-of-type表示。

两者的区别：nth-child是在所有子标签中找，nth-of-type是在某一类子标签中找。

- 举例：

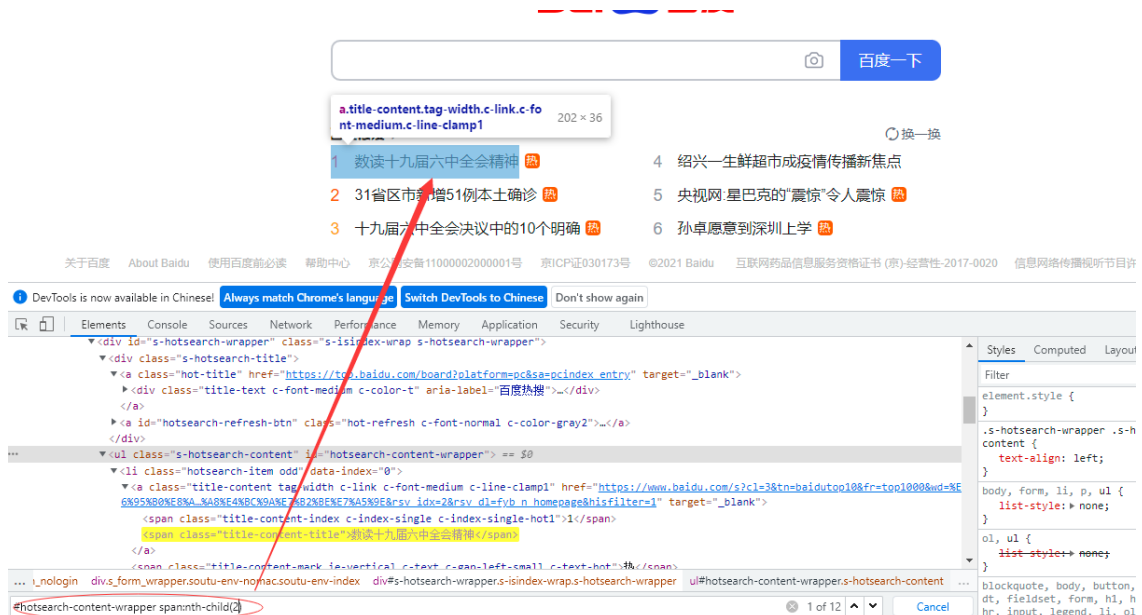
```
#定位百度输入框，输入Python
#使用nth-child(x)
driver.find_element_by_css_selector("span#s_kw_wrap>input:nth-child(2)").send_keys("Python")
#使用nth-of-type(x)
driver.find_element_by_css_selector("span#s_kw_wrap>input:nth-of-type(1)").send_keys("Python")
```

3.9.2.11 通过页面元素定位后代元素

- 语法：

A B：表示B是A的后代元素，后代包含儿子、孙子、曾孙等。这种方式往往再结合属性来定位B元素。

- 举例：



The screenshot shows a web browser with a search bar and a list of search results. A red arrow points from the search bar to the first search result. Below the browser, the Chrome DevTools 'Elements' panel is open, showing the HTML structure of the search results. A red arrow points from the search result in the browser to the corresponding HTML element in the DevTools. The element is a span with class 'title-content-tag-width-c-link-c-font-medium-c-line-clamp1'. The HTML structure shows a div with class 's-hotsearch-wrapper' containing a title and a list of search results. The search results are contained in a div with class 's-hotsearch-content'.

```
#定位百度输入框，输入Python，表达式form#form input[type='text']表示找id等于form的
form标签的后代元素中type等于text的input标签
driver.find_element_by_css_selector("form#form
input[type='text']").send_keys("Python")
```

3.9.2.12 通过页面元素定位同级兄弟元素

- 语法：
 - 表达式1: element+element, 比如div#div1 > input + a, 表示定位与id为div1的div标签的子元素input同级的后面紧挨着的元素a
 - 表达式2: element1~element2, 比如div#div1 > input ~ a, 表示定位与id为div1的div标签的子元素input同级的后面所有元素a, 若满足条件的元素不止一个, 需要使用find_elements_by_css_selector()方法
- 举例:

```
#定位百度输入框, 输入Python, 表达式form#form span.soutu-btn+input表示找id等于form的form标签的后代元素中class等于soutu-btn的span标签后紧邻的input标签
driver.find_element_by_css_selector("form#form span.soutu-btn+input").send_keys("Python")
```

3.9.3 XPath表达式和CSS选择器的语法对比

1. 在XPath中路径用/表示, 在CSS中用>表示
2. 如果要根据标签名定位, 在XPath中用//标签名表示, 在CSS中用"标签名"表示
3. 如果要指定属性, XPath的格式是:[@属性名=属性值], CSS的格式是:[属性名=属性值]
4. 如果要表示多个属性的组合, XPath的格式是[@属性名1=属性值1 and @属性名2=属性值2], CSS的格式是:[属性名1=属性值1][属性名2=属性值2]
5. 模糊匹配, XPath和CSS的格式对比:
 - //标签名[contains(@属性名,属性值)]" <---> 标签名[属性名*=属性值]
 - //标签名[starts-with(@属性名,属性值)]" <---> 标签名[属性名^=属性值]
 - //标签名[ends-with(@属性名,属性值)]" (已被弃用) <---> 标签名[属性名\$=属性值]

3.9.4 八大定位方法的选择

- 1) 如果有固定且唯一的属性 (id/name/class等), 直接通过属性定位;
- 2) 如果是超链接, 优先选择link_text()方法;
- 3) 如果上面几种简单的方法解决不了, 选择find_element_by_css_selector()方法, CSS的功能强大、速度快、语法稍微复杂;
- 4) 如果以上方法都解决不了, 选择find_element_by_xpath()方法, 通常XPath定位的速度会比CSS慢 (特别是在IE上);

四、定位一组元素

以下八个方法用于定义一组具有共同属性的元素, 返回值是一个列表。用前面讲的八个单数的方法, 如果有多个满足条件的元素, 只会定位到第一个满足条件的元素就返回了; 用复数方法, 则会得到一个列表, 得到列表后我们可以很方便地对列表中的任意元素操作。

- find_elements (By.ID,属性值)
- find_elements (By.NAME, 属性值)

- find_elements (By.CLASS_NAME, 属性值)
- find_elements (By.TAG_NAME, 属性值)
- find_elements (By.LINK_TEXT, 属性值)
- find_elements (By.PARTIAL_LINK_TEXT, 属性值)
- find_elements (By.XPATH, 属性值)
- find_elements (By.CSS_SELECTOR, 属性值)

举例：

```
from selenium import webdriver
driver=webdriver.Chrome()
driver.maximize_window()
driver.get("https://www.baidu.com")
#定位类名为mnav的所有a标签
ele_list=driver.find_elements_by_css_selector("a.mnav")
#推出一个文本的列表
text_list=[i.text for i in ele_list]
print(text_list)
#点击列表中的第一个元素，也就是新闻超链接
ele_list[0].click()
```

五、无

六、二次定位

有时我们需要定位的元素没有可利用的属性和其他的信息，不方便直接定位到它，可以通过先定位上级的元素，返回对应的WebElement对象，再调用WebElement对象中的find_element_by_x方法定位它下面的元素。

举例：

```
#定位百度输入框
driver.find_element_by_id("s_fm").find_element_by_class_name("s_ipt").send_keys(
"Python")
```

七、定位子页面中的元素（iframe框架切换）

在网页中页面嵌套页面的做法比较常见，当WebDriver打开一个页面时，默认是在主页面，如果要定位子页面里的元素，需要先切换到子页面再定位。如果在定位完子页面中的元素后再想定位主页面的元素，需要再切换到主页面。

方法：

- driver.switch_to.frame(): 进入到子页面，传入子页面iframe标签的id或者name的属性值；也可以传入索引值（默认是从0开始），如果这个iframe既没有id也没有name，可以先定位到这个元素，把这个元素作为参数传到switch_to.frame()里。
- driver.switch_to.default_content(): 返回主页面
- driver.switch_to.parent_frame(): 返回父页面

如果主页面中有两个同级的子页面，子页面之间不能直接切换，只能先返回到主页面再切换到另一个子页面。

举例：

```
url="https://music.163.com/#/discover/toplist"
driver.get(url)
#进入到id等于g_iframe的iframe中
driver.switch_to.frame("g_iframe")
#定位播放按钮，点击
driver.find_element_by_css_selector("#toplist > div.g-mn3 > div > div.g-wrap > div > div.cnt > div > div.btns.f-cb > a.u-btn2.u-btn2-2.u-btni-addply.f-fl").click()
#返回到主页面
driver.switch_to.default_content()
#定位歌单，点击
driver.find_element_by_css_selector("#g_nav2 > div > ul > li:nth-child(3) > a > em").click()
```

元素定位不到的原因可能是：

- 1、是否有数据加载
- 2、属性值是否写正确
- 3、是否有句柄切换
- 4、是否有iframe框架

项目介绍：然之项目

八、通过JS脚本定位元素

8.1 通过JS脚本定位元素

除了前面的WebDriver的基本定位方法，还有一些使用基本定位方法无法解决的，如Windows窗口、浏览器滚动条等，这时就需要使用JS脚本来定位。

JS定位实际是使用DOM树的定位方式（js:页面动态处理和及时响应用户的操作），那何为DOM树呢？

DOM树表示树形展示的层级结构，层级结构很好地体现了元素与元素之间的联系，当指出树中所含的DOM节点时，简单来说就是这个树实现了DOM接口的元素构成，同时这些实现包含了一些其它浏览器内部所需的属性信息。

常用的几种定位方式的总结如下：

- id定位：document.getElementById()——返回带有指定ID的元素
- name定位：document.getElementsByName()——返回带有指定name的元素
- tag定位：document.getElementsByTagName()——返回包含带有指定标签名称的所有元素的节点列表（集合/节点数组）
- class定位：document.getElementsByClassName()——返回包含带有指定类名的所有元素的节点列表
- CSS定位：document.querySelectorAll()

举例：

```
from selenium import webdriver
driver=webdriver.Chrome()
```

```

driver.maximize_window()
driver.get("https://www.baidu.com")
#id定位
js_script_1="document.getElementById('kw').value='Python';"
#name定位
js_script_2="document.getElementsByName('wd')[0].value='Selenium';"
#tagname定位
js_script_3 = "document.getElementsByTagName('input')[7].value='laowang'"
#class定位
js_script_3="document.getElementsByClassName('s_ipt')[0].value='WebDriver';"
#css定位
js_script_4="document.querySelectorAll('#kw')[0].value='Hello world';"
js_script_5="document.querySelectorAll('#su')[0].click();"
driver.execute_script(js_script_1)
time.sleep(3)
driver.execute_script(js_script_2)
time.sleep(3)
driver.execute_script(js_script_3)
time.sleep(3)
driver.execute_script(js_script_4)
time.sleep(3)
driver.execute_script(js_script_5)

```

8.2 通过JS脚本操作滚动条

操作滚动条脚本: `js = 'arguments[0].scrollIntoView()'`

先定位元素, 再把元素作为JS脚本的参数传进去, 调用`scrollIntoView()`脚本来移动到元素所在的位置。

```

ele_1=driver.find_element_by_css_selector("body > div.footer > div.footer-txt >
p:nth-child(1) > span:nth-child(2)")
driver.execute_script("arguments[0].scrollIntoView();",ele_1)

```

```

document.documentElement.scrollTop=5000#滑动到最底部
driver.execute_script("window.scrollTo(500,300)")#左右上下滑动

```

九、对元素的常用操作

常用操作:

- `send_keys()`: 在文本框输入内容
- `clear()`: 清除文本框的内容
- `click()`: 点击
- `get_attribute('属性名')`: 获取指定属性的值
- `text`: 获取元素的文本

举例:


```
from selenium import webdriver
from time import sleep
#获取百度一下按钮的value属性值
button_text=driver.find_element_by_id("su").get_attribute("value")
#获取直播超链接的元素文本
link_text=driver.find_element_by_css_selector("#s-top-left > a:nth-child(4)").text
```

十、浏览器的常用操作

浏览器的常用操作：

- driver.forward(): 浏览器页面的前进
- driver.back(): 浏览器页面的后退
- ps:在操作浏览器时，前进以及后退需要注意是否存在历史操作
- driver.maximize_window(): 浏览器最大化
- driver.minimize_window(): 浏览器最小化
- driver.refresh(): 刷新浏览器
- driver.get_window_size(): 获取窗口大小
- driver.close(): 关闭当前对象所处页面的窗口
- driver.quit(): 关闭所有的页面窗口并关闭浏览器，如果只存在一个窗口，driver.close()与driver.quit()的效果是一样的
- driver.current_url: 获取当前对象的url地址
- driver.current_window_handle: 获取当前对象的句柄
- driver.title: 获取当前对象的标题
- driver.window_handles: 获取当前对象的所有句柄
- 句柄：有些页面的链接打开后，会重新打开一个窗口，对于这种情况，想在新页面上操作，就得先切换窗口了。获取窗口的唯一标识用句柄表示，所以只需要切换句柄，我们就能在多个页面上灵活自如的操作了。
- 1.元素有属性，浏览器的窗口其实也有属性的，只是你看不到，浏览器窗口的属性用句柄(handle)来识别。

2.人为操作的话，可以通过眼睛看，识别不同的窗口点击切换。但是脚本没长眼睛，它不知道你要操作哪个窗口，这时候只能句柄来判断了

ps:我们在使用浏览器打开链接时，通常会存在2种情况

- 1、在当前窗口打开，即覆盖
- 2、新建窗口打开，保留原有窗口

这个时候虽然web页面已跳转至新窗口，但是代码逻辑还在原来的窗口，为解决该问题，我们需要引入句柄的概念：窗口句柄。粗略的理解：每个窗口对应一个句柄，句柄可认为是一个唯一长字符串

举例：

```
from selenium import webdriver
from time import sleep
```

```

# 实例化对象
driver = webdriver.Chrome()
# 获取窗口大小
print(driver.get_window_size())
# 窗口最大化，有了实例化对象就可以执行
driver.maximize_window()
# 获取窗口大小
print(driver.get_window_size())
sleep(1)
# 设置窗口大小
# driver.set_window_size(500,300)
# 窗口最小化
# driver.minimize_window()
try:
    # 打开网址
    driver.get(r"https://www.baidu.com")
    # 定位百度搜索框并输入Python
    ele = driver.find_element_by_id("kw")
    ele.send_keys("Python")
    sleep(2)
    ele.clear()
    ele.send_keys("Selenium")

    # 点击百度一下按钮
    driver.find_element_by_id("su").click()
    sleep(3)
    driver.back()
    sleep(3)
    driver.forward()
    sleep(3)
    driver.refresh()

finally:
    sleep(20)
    driver.quit()

```

十一、Select下拉框处理

在WebDriver中提供了专门操作下拉选择框的方法，需要导入Select类。

选择下拉选择框的方法：（看到select标签就要想到导入select类来进行下拉列表操作，是select标签一定会有 option；通过下标方式选元素、value属性选元素、文本内容选元素）

1. select_by_visible_text(): 根据显示的文本来选择，如果传入的文本在选项中不存在，会报 NoSuchElementException
2. select_by_index(): 根据选项的索引来选择，索引从0开始，如果传入的索引不存在，则会报 NoSuchElementException
3. select_by_value(): 根据选择的value属性值选择；

举例：

```

import time
from selenium.webdriver.support.select import Select
from selenium import webdriver

```

```

driver=webdriver.Chrome()
driver.maximize_window()
url="https://kyfw.12306.cn/otn/leftTicket/init?linktypeid=dc"
driver.get(url)
driver.find_element_by_id("qd_closeDefaultwarningwindowDialog_id").click()
time.sleep(3)
#根据索引选择，索引为1表示第2个选项
Select(driver.find_element_by_id("cc_start_time")).select_by_index(1)
time.sleep(3)
#根据value选择
Select(driver.find_element_by_id("cc_start_time")).select_by_value("06001200")
time.sleep(3)
#根据文本选择
Select(driver.find_element_by_id("cc_start_time")).select_by_visible_text("18:00
--24:00")

```

十二、切换窗口（句柄）

如果WebDriver打开了几个窗口，想要定位不同窗口的元素，需要先切换到目标窗口。

常用方法和属性：

- current_window_handle：表示当前WebDriver所在窗口的句柄
- window_handles：表示返回WebDriver打开的所有窗口的句柄，以列表的形式返回
- switch_to.window()：用于切换窗口，传入窗口的句柄来完成切换

举例：

```

first_handle=driver.current_window_handle
all_handles=driver.window_handles
for i in all_handles:
    if i!=first_handle:
        driver.switch_to.window(i)
        driver.find_element_by_xpath("//*
[@id=\"TANGRAM__PSP_4__userName\"]").send_keys("zhangsan")
        driver.switch_to.window(first_handle)

```

十三、在代码中设置等待时间

为什么要设置等待时间？在网页中有的元素加载快，有的加载慢，如果元素还没有加载出来就去定位是定位不到的，所以在代码中要考虑设置等待时间等元素可以被定位和操作。

在WebDriver中有三种设置等待时间的方法：

13.1 强制等待（死等）

time.sleep(n)，表示代码运行到这里就休眠n秒钟，n秒之后再往下执行；但这种方法有很大的局限性，你不知道应该设置多长的最合理，设置短了可能还是会定位不到，设置长了就降低了代码的运行效率，在实际操作中，很少设置强制等待；

13.2 隐式等待 (implicitly_wait)

隐式等待相当于设置了一个全局的等待时间，一般在初始化浏览器对象后就设置一个隐式等待，在后面所有的元素定位中，如果页面在第一时间没有加载出来，就会等待固定的时间（设置的隐式等待时间），如果页面在第一时间加载出来了，就继续往下执行不需要等；隐式等待是全局的，针对整个页面的元素加载来说的，一旦设置在整個WebDriver实例的生命周期都有效。可以把隐式等待当做全局变量，它影响整个页面，所以程序需要等待整个页面加载完成（就是浏览器标签栏那个小圈不再转）时，才会执行下一步，但可能页面加载未完成的时候，需要定位的元素已经加载完成了，但受限于某些JS文件、图片加载特别慢，我们不能执行下一步，必须得等到网页所有东西都加载完了才能下一步【增加不必要的加载时间】**

隐式等待举例：driver.implicitly_wait(30)

13.3 显式等待 (explicit_wait)

可以实现单个元素的定位的等待设置，需要导入WebDriverWait类和expected_conditions模块，在WebDriverWait模块中只有两个方法：until、until_not；until、until_not方法传入参数只能是lambda函数或者expected_conditions模块

显式等待举例：

```
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

#设置定位新增客户元素的最大等待时间为10s，每0.5s轮询检查，如果元素可用就继续往下执行；如果超过10s还没有等到，就报TimeoutException
WebDriverWait(driver,10).until(lambda d: d.find_element_by_link_text("新增客户")).click()
driver.switch_to.default_content()

#切换到子页面OpenOpen101
#在until方法中传入EC模块，frame_to_be_available_and_switch_to_it()表示判断子页面是否可用，如果可用就切换进去
WebDriverWait(driver,10).until(EC.frame_to_be_available_and_switch_to_it("OpenOpen101"))

#在until方法中传入EC模块，presence_of_element_located()表示判断元素是否可见，如果可见就继续往下执行代码；注意在#presence_of_element_located()里面传入的定位器的格式不是我们平常写的完整的定位语句，要写成(By.ID,"company")的格式
WebDriverWait(driver,10).until(EC.presence_of_element_located((By.ID,"company")) ,message="元素不见了!").send_keys("zhangsan")

#判断title是否一致,返回布尔值
WebDriverWait(driver,10,0.1).until(EC.title_is("title_text"))

#判断title是否与包含预期值,返回布尔值
WebDriverWait(driver,10,0.1).until(EC.title_contains("title_text"))

#判断某个元素是否被加到了dom树里，并不代表该元素一定可见，如果定位到就返回元素
WebDriverWait(driver,10,0.1).until(EC.presence_of_element_located((locator)))

#判断某个元素是否被添加到了dom里并且可见，可见代表元素可显示且宽和高都大于0
WebDriverWait(driver,10,0.1).until(EC.visibility_of_element_located((locator)))

#判断元素是否可见，如果可见就返回这个元素
```

```

WebDriverWait(driver,10,0.1).until(EC.visibility_of(driver.find_element(element)
))

#判断是否至少有1个元素存在于dom树中，如果定位到就返回列表
WebDriverWait(driver,10,0.1).until(EC.presence_of_all_elements_located((locator)
))

#判断是否至少有一个元素在页面中可见，如果定位到就返回列表
WebDriverWait(driver,10,0.1).until(EC.visibility_of_any_elements_located((locato
r)))

#判断指定的元素中是否包含了预期的字符串，返回布尔值
WebDriverWait(driver,10,0.1).until(EC.text_to_be_present_in_element((locator),'预
期的text'))

#判断指定元素的value属性值中是否包含了预期的字符串，返回布尔值(注意：只是value属性)
WebDriverWait(driver,10,0.1).until(EC.text_to_be_present_in_element_value((locat
or),'预期的text'))

#判断该frame是否可以switch进去，如果可以的话，返回True并且switch进去，否则返回False
WebDriverWait(driver,10,0.1).until(EC.frame_to_be_available_and_switch_to_it(loc
ator))

#判断某个元素在是否存在于dom或不可见,如果可见返回False,不可见返回这个元素
WebDriverWait(driver,10,0.1).until(EC.invisibility_of_element_located((locator))
)

#判断某个元素是否可见并且是可点击的，如果是的就返回这个元素，否则返回False
WebDriverWait(driver,10,0.1).until(EC.element_to_be_clickable((locator)))

#等待某个元素从dom树中移除
WebDriverWait(driver,10,0.1).until(EC.staleness_of(driver.find_element(locator))
)

#判断某个元素是否被选中了，一般用在下拉列表
WebDriverWait(driver,10,0.1).until(EC.element_to_be_selected(driver.find_element
(locator)))

#判断某个元素的选中状态是否符合预期
WebDriverWait(driver,10,0.1).until(EC.element_selection_state_to_be(driver.find_
element(locator),True))

#判断某个元素的选中状态是否符合预期
WebDriverWait(driver,10,0.1).until(EC.element_located_selection_state_to_be((loc
ator),True))

#判断页面上是否存在alert,如果有就切换到alert并返回alert的内容
accept = WebDriverWait(driver,10,0.1).until(EC.alert_is_present())

```

十四、文件的上传

我们遇到的上传文件的标签是input，一般情况下type是file，对于这种类型的文件上传，我们可以直接使用send_keys()方法来上传文件，传入文件的路径。如果type不是file，就不能通过send_keys()方法实现，需要借助第三方的库来实现。（input标签是可以直接send_keys的，这种情况下就简单粗暴）

举例：driver.find_element_by_name("file").send_keys(r"C:\Users\Administrator\Desktop\123.png")

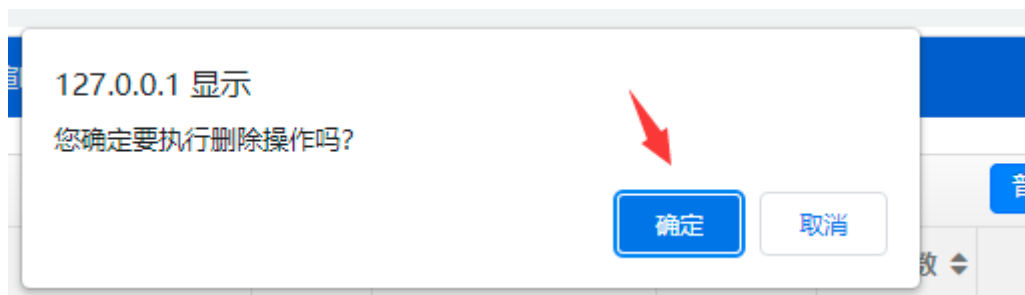
十五、alert框处理

- alert框无法定位，可以通过内置方法直接处理（显示一个警告对话框，而非 HTML 文本）；js对话框三种类型：alert警告对话框，confirm确认对话框，prompt提示对话框

◆ 点击取消按钮



◆ 点击接受/确定按钮



获取文本

```
print(driver.switch_to.alert.text)
```

点击确认

```
driver.switch_to.alert.accept()
```

点击取消

```
driver.switch_to.alert.dismiss()
```

针对prompt输入：driver.switch_to.alert.sendkeys()

十六、断言

- assert语句详解

assert 语句又称为断言语句，其作用和 if 分支非常类似，都是对一个 bool 表达式进行判断，具体功能为：

- 若被判断的 bool 表达式结果为 True，则继续执行后续代码
- 若被判断的 bool 表达式结果为 False，程序会引发 AssertionError 错误
 - 1、获取元素文本：元素定位.text
 - 2、获取当前窗口url来断言：driver.current_url
 - 3、获取当前窗口标题来断言：driver.title

- ```

获取文本(作为实际结果) 元素.text
get_text = driver.find_element_by_xpath('//*[@id="mainNavbar"]/div/ul[1]/li/a').text
print("真实姓名是: ",get_text)

print("url:",driver.current_url)
print("标题: ",driver.title)

if get_text == "admin1":
print("断言成功")
else:
print("断言失败")
assert get_text=="admin" # 如果断言失败，就停止程序

```

```

#捕获URL地址
print(driver.current_url)
#捕获title, 用title断言
print(driver.title)
#获取文本用的比较多, 并作为实际结果
name = driver.find_element(By.XPATH,'//*[@id="mainNavbar"]/div/ul[1]/li/a').text
print(name)
if name == "admin":
print("断言成功, 就执行下面的代码")
else:
raise NameError("断言失败, 请排查错误")
assert name=="admin"#python自带的断言方法

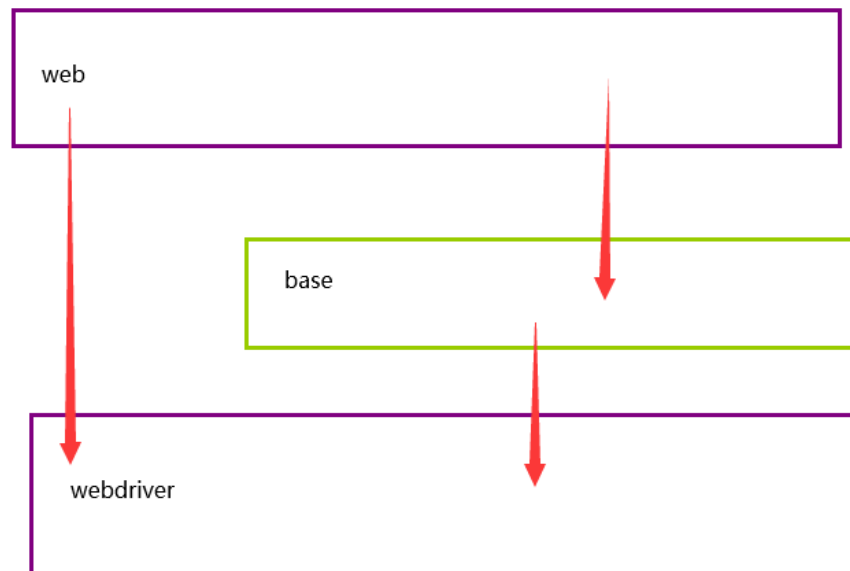
```

## 十七、封装——对于重复的代码采用类的形式进行封装、继承等

步骤:

- 初学者:
  - 把线性代码封装成面向对象的形式 (类和方法)
  - 把里面的参数全部提成形参, 在调用的时候传入实参
- 1、
  - 编写线性业务流程脚本;
  - 业务数据转换成参数形式;
  - 封装到不同类的方法中;
  - 调用不同方法合成流程脚本;
  - 从文件读取数据到脚本中;
- 2、为什么要封装:
 

使代码更简洁, 思路如图:






## 十八、UI自动化之分层思想pom模式

- 1、什么是POM? (page\_object\_model)
  - 页面对象模型(POM)是一种设计模式, 用来管理维护一组web元素集的对象库;  
在POM下, 应用程序的每一个页面都有一个对应的page class;  
每一个page class维护着该web页的表现层和操作层;  
page class中的方法命名最好根据其对应的业务场景进行命名,
- 2、pom的优点
  - 2.1.POM提供了一种在UI层操作、业务流程与验证分离的模式, 这使得测试代码变得更加清晰和高可读性
  - 2.2.对象库与用例分离, 使得我们更好的复用对象, 甚至能与不同的工具进行深度结合应用
  - 2.3.可复用的页面方法代码会变得更加优化
  - 2.4.更加有效的命名方式使得我们更加清晰的知道方法所操作的UI元素。例如我们要添加用户, 方法名命名为: adduserPage(),通过方法名即可清晰的知道具体的功能实现。
- 3、在pom的模式结构基础上搭建的web自动化框架如下:
  - common-公共层: base.py-selenium方法二次封装等
  - page-页面层: 封装项目所有页面,每一个pageClass都继承Base类
    - loginpage.py-登录页面封装
    - adduserpage.py-添加用户页面封装
  - case-用例层
    - test\_login.py-登录功能测试用例;登录流程,引用page中的loginpage文件
  - dataconfig-存放配置文件
    - data-存放测试数据:Excel,txt,yaml,json
  - result-结果层
    - log-测试脚本日志文件
    - report-测试报告



## 十九、Unittest:单元测试框架及组成

-  python中的unittest框架是python自带的一套测试框架，pytest(第三方库) 学习起来也相对较容易，unittest框架最核心的四个部分：
    - 1、test case：就是我们的测试用例，unittest中提供了一个基本类TestCase，可以用来创建新的测试用例，一个TestCase的实例就是一个测试用例；unittest中测试用例方法都是以test开头的，且执行顺序会按照方法名的ASCII码排序。
    - 2、test fixture：测试夹具，用于测试用例环境的搭建和销毁。即用例测试前准备环境的搭建（SetUp前置条件），测试后环境的还原（TearDown后置条件），比如测试前需要打开浏览器等就是测试用例需要的环境，运行完后执行下一个用例前需要还原环境，以免影响下一条用例的测试结果。
    - 3、test suite：测试套件，用来把需要一起执行的测试用例集中放到一块执行，相当于一个篮子。我们可以使用TestLoader来加载测试用例到测试套件中。
    - 4、test runner：用来执行测试用例的，并返回测试用例的执行结果。它还可以用图形或者文本接口，把返回的测试结果更形象的展现出来，如：HTMLTestRunner，TestRunner等
  - unittest组成
    - test fixture：测试夹具，有两种使用方式
      - 一种是以测试方法为维度的setUp()和tearDown(),
      - 一种是以测试类方法为维度的setUpClass()和tearDownClass()
    - TestCase：（测试案例）——所有测试用例的基类，它是软件测试中最基本的组成单元；
      - 每个用例类必须继承unittest.TestCase类
      - 在运行每个用例之前会执行setUp()方法，
      - 运行完每个用例执行tearDown()方法。
      - 运行所有用例之前只执行一遍setUpClass(),
      - 运行完所有用例之后只执行一遍tearDownClass()
    - TestSuite：（测试套件）——测试案例的集合；
      - unittest框架中测试套件（testsuite）：一组测试用例的集合
- 添加单个用例
-  test\_suite = unittest.TestSuite()
   
test\_suite.addTest(LoginTest('test\_login\_fail\_0\_111'))
   
test\_login\_fail\_0\_111中的111是测试用例的第一列数据。
- 添加批量用例
-  test\_suite = unittest.TestSuite()
   
test\_suite.addTests(unittest.TestLoader().discover('用例文件路径',pattern='login\_test.py'))
   
pattern为匹配规则，可以写成 \*test.py，意思是查找以 \*test.py 结尾的所有文件。

```
import unittest

from case.login_test import LoginTest
from common.read_ini import ReadIni
from mfyreport import TestRunner
```

```

import time

class RanZhiRunner:

 def ranzhi_runner(self):

 # 实例化测试套件
 # suite = unittest.TestSuite()

 # 把用例加载到测试套件中
 # 添加一个用例
 # suite.addTest(LoginTest("test_login_fial_0_"))
 # 批量加载用例
 case_path = ReadIni().get_case_path() # 调用封装路径找到用例位置
 # 通过addTests来添加用例，通过TestLoader来加载用例，通过discover来找到用例，通过pattern匹配用例
 # 第一种加载用例的方法
 #
 suite.addTests(unittest.TestLoader().discover(case_path,pattern="login_test.py"))
 # 第二种加载用例的方法
 # suite =
 unittest.defaultTestLoader.discover(case_path,pattern="login_test.py")
 # 第三种加载用例的方法
 suite =
 unittest.defaultTestLoader.loadTestsFromTestCase(LoginTest)

 tt = time.strftime("%Y_%m_%d_%H_%M_%S")
 test_run =
 TestRunner(suite,filename="ranzhi{}.html".format(tt),report_dir=ReadIni().get_report_path(),
 title="然之测试报告",tester="老王",templates=1,desc="下面是然之测试的具体情况")

 test_run.run()

 # 发送邮件：以后在实际工作中不会在代码里面发送邮件，要么手动，要么就是使用jenkins工具

if __name__ == '__main__':
 run = RanZhiRunner()
 run.ranzhi_runner()

```

- test runner：（执行测试）——测试用例的执行。

- ```
import unittest
```



```
# 必须继承unittest.TestCase
class Test11(unittest.TestCase):
    @classmethod
```

```

def setUpClass(self):
    print("这是setUpClass")
    @classmethod
def tearDownClass(self):
    print("这是tearDownClass")

# def setUp(self):
#     print("这是setUp")
#
# def tearDown(self):
#     print("这是tearDown")

def test_01(self):
    print("这是test_01")

def test_a(self):
    print("这是test_a")

def test_A(self):
    print("这是test_A")

if __name__ == '__main__':
    unittest.main() # 直接运行所有的用例方法

```

二十、常用三种断言方法：



如图：

```

# 断言:必须完全相等,才能算断言成功
self.assertEqual(box_text,except1,msg="实际结果和期望结果不一致,请排查")
# 断言:必须完全相等,才能算断言成功
# self.assertTrue(box_text==except1,msg="实际结果和期望结果不一致,请排查")
# 包含: 第二个参数包含第一个参数
# self.assertIn(except1,box_text,msg="实际结果和期望结果不一致,请排查")

```

二十一、参数化文件（数据驱动）

- 数据驱动：
 -  unittest框架中本身没有自带参数的模块，但是存在扩展模块、引用模块：
ddt (datadrivertest) 或者模块parameterized
安装parameterized
 -  安装方式：
在pycharm中安装
pip install parameterized

- ```

from parameterized import parameterized
@parameterized.expand([('admin', '123456'), ('da76', '123456')])
def test_login_success(self, name, pwd):
 '''用户名和密码正确，登录成功'''
 self.login.login(name, pwd)
 self.login.logout()
 print('用户名是%s，密码是%s，登录成功!' % (name, pwd))

```

- 读取excel文件:

- ```

import openpyxl

# [[],[ ]]
def read_excel(filename, sheetname):
    # 打开文件
    wb = openpyxl.load_workbook(filename)
    # 选择sheet页
    ws = wb[sheetname]
    all_list = []
    # 第一个for循环读取的是一行的数据
    for hang_tuple in ws:
        # 第二个for循环是读取的单元格的数据
        list1 = []
        for i in hang_tuple:
            if i.value == None:
                i.value = ""
            list1.append(i.value)
        all_list.append(list1)
    return all_list[1:]

```

- 读取yaml文件:

- 安装yaml文件

- 1、pip install pyyaml
 - 2、在pycharm里面安装

- 数据格式

- ```

test_login_success:
 - ["xiao204724", "123456", "hua204724", 1]
 - ["xiao329550", "123456", "hua329550", 2]

test_login_fail:
 - ["", "123456", "登录失败，请检查您的成员名或密码是否填写正确。", 1]
 - ["dfsdf", "", "登录失败，请检查您的成员名或密码是否填写正", 2]

test_adduser_success:
 pass

```

- 读取yaml文件数据

- ```
import yaml

def read_yaml(filename):
    # 打开文件
    with open(filename,encoding="utf8") as yaml_data:
        return yaml.safe_load(yaml_data)
```

- 读取json文件：自带的，不用下载(常用，占用内存小，运行效率高)

- json数据格式

-

```
{
  "test_login_success": [
    ["xiao204724", "123456", "hua204724", 1],
    ["xiao329550", "123456", "hua329550", 2]
  ],
  "test_login_fial": [
    ["", "123456", "登录失败，请检查您的成员名或密码是否填写正确。", 1],
    ["dfsdf", "", "登录失败，请检查您的成员名或密码是否填写正", 2]
  ]
}
```

- 读取json文件数据

- ```
import json

def read_json(filename):
 with open(filename,encoding="utf8") as json_data:
 return json.load(json_data) # 把json数据格式变为字典

if __name__ == '__main__':

 print(read_json(r"D:\Projects\selenium35\Day4\case_json.json"))
```

- ini文件:

- 数据格式:

- ```
[path]
excel_path = data_config\ranzhi_case.xlsx
json_path = data_config\case_json.json
yaml_path = data_config\case_yaml.yaml
screenshot_path = result\screenshot\
log_path = result\log\

[mysql]
ip = 192.168.2.2
port = 3306
```

- 下载安装:

- 导入configparser

- 安装方式

- 1、pip install configparser
 - 2、在pycharm中安装
- 代码:

```
import configparser
import os

class ReadIni:

    def __init__(self):
        # 实例化对象
        self.config = configparser.ConfigParser()
        # 自动获取项目在你电脑上的路径
        self.before_path =
os.path.dirname(os.path.abspath(__file__)).split("common")
[0]
        # 找到ini文件路径
        self.config.read(str(self.before_path) +
r"\data_config\file_path.ini")

    def get_yaml_path(self):
        # 获取ini文件数据获取自己配置的文件路径
        file_path = self.config.get("path", "yaml_path")
        # 实现路径拼接
        return os.path.join(self.before_path, file_path)

    def get_json_path(self):
        # 获取ini文件数据获取自己配置的文件路径
        file_path = self.config.get("path", "json_path")
        # 实现路径拼接
        return os.path.join(self.before_path, file_path)

    def get_excel_path(self):
        # 获取ini文件数据获取自己配置的文件路径
        file_path = self.config.get("path", "excel_path")
        # 实现路径拼接
        return os.path.join(self.before_path, file_path)

    def get_screenshot_path(self):
        # 获取ini文件数据获取自己配置的文件路径
        file_path = self.config.get("path",
"screenshot_path")
        # 实现路径拼接
        return os.path.join(self.before_path, file_path)

    def get_log_path(self):
        # 获取ini文件数据获取自己配置的文件路径
        file_path = self.config.get("path", "log_path")
        # 实现路径拼接
        return os.path.join(self.before_path, file_path)

if __name__ == '__main__':
    get = ReadIni()
    print(get.get_yaml_path())
    print(get.get_json_path())
    print(get.get_excel_path())
```

二十二、跳过用例的装饰器（跳过不用执行用例）

```
# @parameterized.expand(json_data["test_login_succeed"])
@parameterized.expand(json_data["test_login_succeed"])
# @unittest.skip # 跳过比执行, 没有任何提示
# @unittest.skip("想跳过")
# @unittest.skipIf(os.name == "nt", "跳过") # 满足条件就跳过
@unittest.skipUnless(os.name == "nt", "没有跳过") # 条件成立就执行, 不成立就不执行
def test_login_succeed(self, user, pwd, expect, xuhao):
```

二十三、截图：在base中封装截图方法（二选一）

```
def save_screenshot(self, filename):
    """
    保存当前窗口的截图
    :param filename: 图片路径
    :return:
    """
    self.driver.save_screenshot(filename)

def get_screenshot_as_file(self, filename):
    """
    保存当前窗口的截图
    :param filename: 图片路径
    :return:
    """
    self.driver.get_screenshot_as_file(filename)
```

二十四、日志输出【时间】【模块】【级别】：描述信息

- 介绍
 - logging模块是Python内置的标准模块，主要用于输出运行日志，可以灵活配置输出日志的各项信息，比如设置输出日志的等级、日志保存路径等；相比print，具备如下优点：
 - 可以通过设置不同的日志等级，只输出重要信息，而不必显示大量的调试信息；
 - print将所有信息都输出到标准输出中，严重影响编程人员从标准输出中查看其它数据，logging则可以由编程人员决定将信息输出到什么地方，以及怎么输出；
- 日志级别
 - logging中可以选择很多消息级别，如debug、info、warning、error。
- 日志格式
 - %(levelNo)s: 打印日志级别的数值
 - %(levelName)s: 打印日志级别的名称
 - %(pathname)s: 打印当前执行程序的路径
 - %(filename)s: 打印当前执行程序名
 - %(funcName)s: 打印日志的当前函数
 - %(lineno)d: 打印日志的当前行号
 - %(asctime)s: 打印日志的时间

- %(thread)d: 打印线程ID
- %(threadName)s: 打印线程名称
- %(process)d: 打印进程ID
- %(message)s: 打印日志信息
- 代码

```
import logging

def get_log(filename):
    # 实例化
    log = logging.Logger("pro_ranzhi30")
    # 打印日志到指定位置:
    fh = logging.FileHandler(filename, encoding="utf8")
    # 定义日志格式
    formatter = logging.Formatter("[%%(asctime)s] [%%(filename)s] [%%(levelname)s]: %(message)s")
    # 调用设置好的格式来规定日志输出的样式
    fh.setFormatter(formatter)
    # 把设置好格式的日志添加到实例化对象中
    log.addHandler(fh)

    return log
```

二十五、dos命令运行

- 打开dos命令窗口，进入到项目根目录，输入python + 模块名 回车就可以执行

二十六、总结：项目框架

- 主要是说清楚你的分层思想，以及每个层里面放的是什么内容
 - 我们的代码有几部分，比如有common部分，common主要是对一些常用的操作方法做了封装，比如有对元素进行定位，点击，输入内容，切换frame，退出frame，显示/隐式等待，另外读取文件数据，生成日志，生成报告，发送邮件的这些方法也会在里面封装。common部分大概就这些了。
 - 除了common部分的代码，我们还有page部分代码，page部分代码主要是把不同的业务流程写在不同的page和方法里面，方法里面的操作是调用前面在base里面封装好的方法，该页面是用来存放元素定位和对元素的操作的
 - 业务流程代码写完后，我们会抽取调用不同的业务流程方法以及引用unittest框架组装成不同的测试用例写在case里面，同时还会引用unittest单元测试框架对用例进行断言，还有会把用例的测试数据分离出去，比如把数据放在excel或者json,yaml等文件中然后通过参数化框架进行读取调用数据到用例中执行。除了这些以外，我们还在用例中添加日志代码，方便后期运行代码出错后排查和定位问题。
 - 最后我们还有runner部分的代码，runner部分的代码主要是抽取想要运行的用例进行测试，测试执行完成后会自动生成测试报告和运行的日志。而我们执行自动化测试也会从这个runner入口运行。

二十七、你用Python代码做自动化是怎么做的？

- 答：我们自动化开发语言用的是Python语言，代码是分层管理的，分别是：公共层、数据与配置层、用例层、结果层，运行层等，其中公共层实现公共工具的代码，比如数据文件的读取，数据与配置层存放用例数据文件和配置文件，用例层是利用Unittest框架编写测试用例以及实现断言等，结果层存放测试报告、日志等。最终通过HTMLTestRunner模块运行测试用例，生成HTML格式的报告，通过Jenkins工具来实现自动化代码的持续集成，这就是我做自动化的基本实现方法。

二十八、自动化的周期

- 接口自动化：从后端接口开发完成就可以做了
- web UI自动化，需要前端页面的支撑(功能稳定的情况下)。
 - 1、元素定位和操作封装到不同的page类里面，实现代码（模块）分离（登录，添加用户）
 - 3、数据分离（excel,json,yaml,ini）便于数据管理
 - 4、执行测试用例（unittest）
 - 5、出结果（日志，报告，截图）
 - 6、发送报告到邮箱（都没有使用代码来做）（实际是手动或者用配置（Jenkins）文件来进行）

