

# Table of Contents

UI自动化测试课程	1.1
Web自动化测试基础	1.2
Web自动化测试框架	1.2.1
环境搭建	1.2.2
元素定位	1.2.3
元素定位-XPath、CSS	1.2.4
元素操作 浏览器操作方法	1.2.5
鼠标和键盘操作	1.2.6
元素等待	1.2.7
下拉选择框、弹出框、滚动条操作	1.2.8
frame切换、多窗口切换	1.2.9
窗口截图、验证码处理	1.2.10

# UI自动化测试课程

序号	章节	知识点
1	UI自动化测试介绍	1. UI自动化测试
2	Web自动化测试基础	1. Web自动化测试框架 2. 环境搭建 3. 元素定位和元素操作 4. 鼠标和键盘操作 5. 元素等待 6. HTML特殊元素处理 7. 验证码处理
3	移动自动化测试基础	1. 移动自动化测试框架 2. ADB调试工具 3. UIAutomatorViewer工具 4. 元素定位和元素操作 5. 滑动和拖拽事件 6. 高级手势TouchAction 7. 手机操作
4	PyTest框架	1. PyTest基本使用 2. PyTest常用插件 3. PyTest高级用法
5	PO模式	1. 方法封装 2. PO模式介绍 3. PO模式实战
6	数据驱动	1. 数据驱动介绍 2. 数据驱动实战
7	日志收集	1. 日志相关概念 2. 日志的基本方法 3. 日志的高级方法
8	黑马头条项目实战	1. 自动化测试流程 2. 项目实战演练

## 课程目标

1. 掌握使用Selenium实现Web自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
2. 掌握使用Appium实现移动自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
3. 掌握如何通过PyTest管理用例脚本，并使用Allure生成HTML测试报告。
4. 掌握使用PO模式来设计自动化测试代码的架构。
5. 掌握使用数据驱动来实现自动化测试代码和测试数据的分离。
6. 掌握使用logging来实现日志的收集。

# Web自动化测试基础

## 目标

1. 理解自动化测试的相关概念
2. 了解Selenium的特点
3. 掌握如何搭建web自动化测试的相关环境
4. 熟练掌握web自动化测试脚本编写的基本步骤
5. 熟练应用八种元素定位方式
6. 掌握对元素和浏览器的操作方法
7. 掌握键盘鼠标的操作
8. 掌握元素等待的操作
9. 掌握下拉选择框、警告框和滚动条的操作
10. 掌握如何切换frame框架和多窗口
11. 掌握如何实现窗口截图

# Web自动化测试框架

## 目标

1. 了解Web自动化测试常用工具
2. 熟悉Selenium的特点

## 1. 主流的Web自动化测试工具

1. QTP  
QTP是一个商业化的功能测试工具，收费，支持web，桌面自动化测试。
2. Selenium（本阶段学习）  
Selenium是一个开源的web自动化测试工具，免费，主要做功能测试。
3. Robot framework  
Robot Framework是一个基于Python可扩展地关键字驱动测试的测试自动化框架。

## 2. 什么是Selenium?

Selenium是一个用于Web应用程序的自动化测试工具；中文的意思（硒）

### 2.1 Selenium特点

1. 开源软件：源代码开放可以根据需要来增加工具的某些功能
2. 跨平台：linux、windows、mac
3. 支持多种浏览器：Firefox、Chrome、IE、Edge、Opera、Safari等
4. 支持多种语言：Python、Java、C#、JavaScript、Ruby、PHP等
5. 成熟稳定：目前已经被google、百度、腾讯等公司广泛使用
6. 功能强大：能够实现类似商业工具的大部分功能，因为开源性，可实现定制化功能

### 2.2 Selenium发展史【了解】



# 环境搭建

## 目标

1. 掌握如何搭建web自动化测试的相关环境
2. 熟练掌握web自动化测试脚本编写的基本步骤

## 1. 环境搭建

基于Python环境搭建

1. Python 开发环境
2. 安装selenium包
3. 安装浏览器
4. 安装浏览器驱动 -- 保证能够用程序驱动浏览器，实现自动化测试

### 1.1 安装selenium包

前提：Python3 安装完毕且能正常运行

#### PIP工具

pip是一个通用的 Python 包管理工具，提供了对 Python 包的查找、下载、安装、卸载的功能。

#### 安装

```
pip install selenium
```

#### 卸载

```
pip uninstall selenium
```

#### 查看

```
pip show selenium
```

## 1.2 安装浏览器驱动

### 火狐浏览器

1. Firefox 48 以上版本  
selenium 3.x + Firefox驱动(geckodriver)  
驱动下载地址: <https://github.com/mozilla/geckodriver/releases>
2. Firefox 48 以下版本  
selenium 2.x + 内置驱动

### 谷歌浏览器

selenium 2.x/3.x + Chrome驱动(chromedriver)  
驱动下载地址: <https://sites.google.com/a/chromium.org/chromedriver/downloads>

chromedriver版本	支持的Chrome版本
2.41	v67-69
2.40	v66-68
2.39	v66-68
2.38	v65-67
2.37	v64-66
2.36	v63-65
2.35	v62-64
...	...

### Edge浏览器(了解)

selenium 3.x + Edge驱动(MicrosoftWebDriver)  
驱动下载地址: <https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>

### 安装浏览器驱动的步骤

1. 下载浏览器驱动
  - 各个驱动下载地址: <http://www.seleniumhq.org/download/>
  - 浏览器的版本和驱动版本要一致!
2. 把驱动文件所在目录添加到Path环境变量中
  - 或者直接放到Python安装目录, 因为Python已添加到Path中

## 2. 入门示例

### 2.1 需求

通过程序启动浏览器，并打开百度首页，暂停3秒，关闭浏览器

## 2.2 实现步骤

1. 导包  
`from selenium import webdriver`
2. 创建浏览器驱动对象  
Firefox浏览器: `driver = webdriver.Firefox()`  
Chrome浏览器: `driver = webdriver.Chrome()`  
Edge浏览器: `driver = webdriver.Edge()`
3. 打开Web页面  
`driver.get("http://www.baidu.com/")`
4. 暂停  
`time.sleep(3)`
5. 关闭驱动对象  
`driver.quit()`

## 2.3 示例代码

```
# 导包
from selenium import webdriver
import time

# 创建浏览器驱动对象
driver = webdriver.Firefox()
# driver = webdriver.Chrome()
# driver = webdriver.Edge()

# 加载web页面
driver.get("http://www.baidu.com/")

# 暂停3秒
time.sleep(3)

# 关闭驱动对象
driver.quit()
```

## 3. 总结

1. web自动测试环境搭建中涉及到的软件?
2. selenium 安装、卸载、查看命令?
3. web自动化测试脚本编写的基本步骤?



# 元素定位

## 目标

1. 掌握id、name、class\_name、tag\_name、link\_text、partial\_link\_text定位方式的使用

思考：为什么要学习元素定位？

让程序操作指定元素，就必须先找到此元素。

## 1. 如何进行元素定位？

html页面由标签构成，标签的基本格式如下：

<标签名 属性名1="属性值1" 属性名2="属性值2">文本</标签名>

示例：

```
<input id="username" type="text" name="username" placeholder="用户名" />
<div id="my_cart">
  <span>我的购物车</span>
</div>
```

元素定位就是通过元素的信息或元素层级结构来定位元素的。

思考：如何快速的查看一个元素的相关信息？

## 2. 浏览器开发者工具

浏览器开发者工具就是给专业的web应用和网站开发人员使用的工具。包含了对HTML查看和编辑、Javascript控制台、网络状况监视等功能，是开发JavaScript、CSS、HTML和Ajax的得力助手。

作用：快速定位元素，查看元素信息



## 2.1 如何使用浏览器开发者工具

### 安装

浏览器已默认安装，可以直接使用

### 启动

- 快捷键：一般在windows系统上打开浏览器开发者工具都是按F12
- 火狐浏览器：在页面上点击右键选择'查看元素'
- 谷歌浏览器：在页面上点击右键选择'检查'

### 使用

- 方法一：在要查看的元素上点击右键选择'查看元素'或者'检查'
- 方法二：先打开浏览器开发者工具，点击选择元素的图标，移动鼠标到要查看的元素，然后点击

## 3. 元素定位方式

Selenium提供了八种定位元素方式

1. id
2. name
3. class\_name
4. tag\_name
5. link\_text
6. partial\_link\_text
7. XPath
8. CSS

## 3.1 id定位

说明: id定位就是通过元素的id属性来定位元素, HTML规定id属性在整个HTML文档中必须是唯一的;  
前提: 元素有id属性

### id定位方法

```
element = driver.find_element_by_id(id)
```

### 案例

案例演示环境说明:

受限于网络速度的影响, 我们案例采用本地的html页面来演示。这样可以提高学习效率和脚本执行速率。

需求: 打开注册A.html页面, 完成以下操作

- 1). 使用id定位, 输入用户名: admin
- 2). 使用id定位, 输入密码: 123456
- 3). 3秒后关闭浏览器窗口

### 案例实现步骤分析

1. 导入selenium包 --> `from selenium import webdriver`
2. 导入time包 --> `import time`
3. 实例化浏览器驱动对象 --> `driver = webdriver.Firefox()`
4. 打开注册A.html --> `driver.get(url)`
5. 调用id定位方法 --> `element = driver.find_element_by_id("")`
6. 使用send\_keys()方法输入内容 --> `element.send_keys("admin")`
7. 暂停3秒 --> `time.sleep(3)`
8. 关闭浏览器驱动对象 --> `driver.quit()`

说明: 为了更好的学习体验, 我们先暂时使用下send\_keys()方法来输入内容

## 3.2 name定位

说明: name定位就是根据元素name属性来定位。HTML文档中name的属性值是可以重复的。  
前提: 元素有name属性

### name定位方法

```
element = driver.find_element_by_name(name)
```

## 案例

需求：打开注册A.html页面，完成以下操作

- 1).使用name定位用户名，输入：admin
- 2).使用name定位密码，输入：123456
- 3).3秒后关闭浏览器窗口

## 3.3 class\_name定位

说明：class\_name定位就是根据元素class属性值来定位元素。HTML通过使用class来定义元素的样式。

前提：元素有class属性

注意：如果class有多个属性值，只能使用其中的一个

### class\_name定位方法

```
element = driver.find_element_by_class_name(class_name)
```

## 案例

需求：打开注册A.html页面，完成以下操作

- 1).通过class\_name定位电话号码A，并输入：18611111111
- 2).通过class\_name定位电子邮箱A，并输入：123@qq.com
- 3).3秒后关闭浏览器窗口

## 3.4 tag\_name定位

说明：tag\_name定位就是通过标签名来定位：

HTML本质就是由不同的tag组成，每一种标签一般在页面中会存在多个，所以不方便进行精确定位，一般很少使用

### tag\_name定位方法

```
element = driver.find_element_by_tag_name(tag_name)
```

如果存在多个相同标签，则返回符合条件的第一个标签

如何获取第二个元素？稍后讲解

## 案例

需求：打开注册A.html页面，完成以下操作

- 1).使用tag\_name定位用户名输入框，并输入：admin
- 2).3秒后关闭浏览器窗口

## 3.5 link\_text定位

说明：link\_text定位是专门用来定位超链接元素(<a>标签</a>)，并且是通过超链接的文本内容来定位元素。

### link\_text定位方法

```
element = driver.find_element_by_link_text(link_text)
```

link\_text: 为超链接的全部文本内容

## 案例

需求：打开注册A.html页面，完成以下操作

- 1).使用link\_text定位(访问 新浪 网站)超链接，并点击
- 2).3秒后关闭浏览器窗口

## 3.6 partial\_link\_text定位

说明：partial\_link\_text定位是对link\_text定位的补充，link\_text使用全部文本内容匹配元素，而partial\_link\_text可以使用局部来匹配元素，也可以使用全部文本内容匹配元素。

### partial\_link\_text定位方法

```
element = driver.find_element_by_partial_link_text(partial_link_text)
```

partial\_link\_text: 可以传入a标签局部文本-能表达唯一性

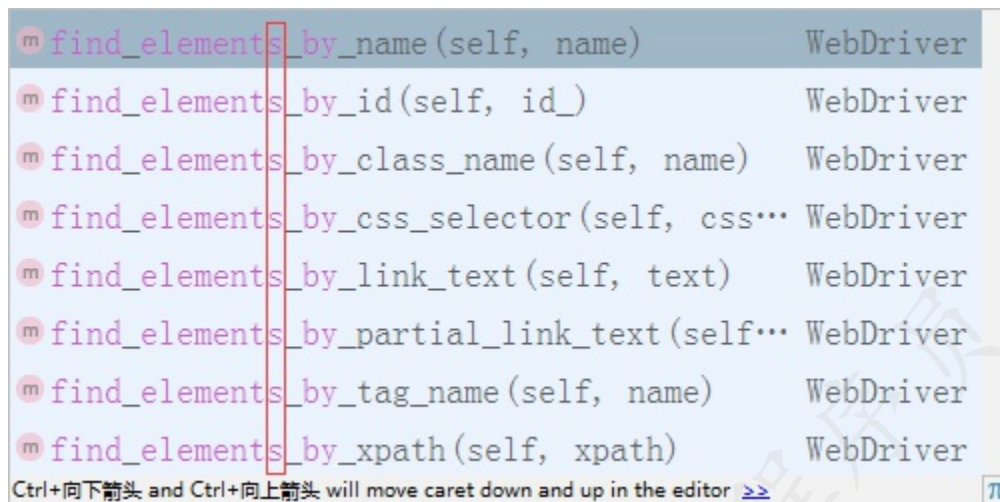
## 案例

需求：打开注册A.html页面，完成以下操作

- 1).使用partial\_link\_text定位(访问 新浪 网站)超链接，并点击
- 2).3秒后关闭浏览器窗口

## 4. 定位一组元素

在我们学习使用以上方法的时候，发现有个共同的相似方法：



### 4.1 find\_elements\_by\_xxx()

作用：

- 1). 查找定位所有符合条件的元素
- 2). 返回的定位元素格式为数组(列表)格式；

说明：

- 1). 列表数据格式的读取需要指定下标(下标从0开始)

### 4.2 案例

需求：打开注册A.html页面，完成以下操作

- 1). 使用tag\_name定位密码输入框(第二个input标签)，并输入：123456
- 2). 3秒后关闭浏览器窗口

### 4.3 示例代码

```
driver.find_elements_by_tag_name("input")[1].send_keys("123456")
```

# 元素定位-XPath、CSS

## 目标

1. 掌握XPath定位策略
2. 掌握CSS定位策略

## 为什么要学习XPath、CSS定位？

1. 如果要定位的元素没有id、name、class属性，该如何进行定位？
2. 如果通过name、class、tag\_name无法定位到唯一的元素，该如何进行定位？

示例：

```
<input type="submit" value="提交" />
```

## 1. 什么是XPath？

1. XPath即为XML Path的简称，它是一门在 XML 文档中查找元素信息的语言。
2. HTML可以看做是XML的一种实现，所以Selenium用户可以使用这种强大的语言在Web应用中定位元素。

XML：一种标记语言，用于数据的存储和传递。 后缀.xml结尾

```
<?xml version="1.0" encoding="UTF-8" ?>
<node>
  <db id="db" desc="三条边的长度都一样">
    <b1>3</b1>
    <b2>3</b2>
    <b3>3</b3>
    <expect>等边三角形</expect>
  </db>
  <dy>
    <b1>4</b1>
    <b2>4</b2>
    <b3>5</b3>
    <expect>等腰三角形</expect>
  </dy>
</node>
```

XPath定位方式之所以强大，是因为它有非常灵活的定位策略

## 2. XPath定位策略(方式)

1. 路径-定位
2. 利用元素属性-定位
3. 属性与逻辑结合-定位
4. 层级与属性结合-定位

### XPath定位方法

```
element = driver.find_element_by_xpath(xpath)
```

#### 2.1 路径定位(绝对路径、相对路径)

绝对路径：从最外层元素到指定元素之间所有经过元素层级的路径

- 1). 绝对路径以/html根节点开始，使用/来分隔元素层级；  
如：/html/body/div/fieldset/p[1]/input
- 2). 绝对路径对页面结构要求比较严格，不建议使用

相对路径：匹配任意层级的元素，不限制元素的位置

- 1). 相对路径以//开始
- 2). 格式：//input 或者 //

#### 练习

需求：打开注册A.html页面，完成以下操作

- 1). 使用绝对路径定位用户名输入框，并输入：admin
- 2). 暂停2秒
- 3). 使用相对路径定位用户名输入框，并输入：123

#### 2.2 利用元素属性

说明：通过使用元素的属性信息来定位元素

格式：//input[@id='userA'] 或者 //\*[@id='userA']

#### 练习

需求：打开注册A.html页面，完成以下操作

- 1). 利用元素的属性信息精确定位用户名输入框，并输入：admin



## 2.3 属性与逻辑结合

说明：解决元素之间个相同属性重名问题

格式： `//*[@name='tel' and @class='tel']`

### 练习

需求：打开注册A.html页面，完成以下操作

1).使用属性与逻辑结合定位策略，在test1对应的输入框里输入：admin

## 2.4 层级与属性结合

说明：如果通过元素自身的信息不方便直接定位到该元素，则可以先定位到其父级元素，然后再找到该元素

格式： `//*[@id='p1']/input`

### 练习

需求：打开注册A.html页面，完成以下操作

1).使用层级与属性结合定位策略，在test1对应的输入框里输入：admin

## 2.5 XPath-延伸

<code>//*[text()='xxx']</code>	文本内容是xxx的元素
<code>//*[contains(@attribute,'xxx')]</code>	属性中含有xxx的元素
<code>//*[starts-with(@attribute,'xxx')]</code>	属性以xxx开头的元素

## 2.6 XPath-总结

1. XPath定位策略有哪些？

## 3. CSS定位

### 3.1 什么是CSS定位？

1. CSS (Cascading Style Sheets) 是一种语言，它用来描述HTML元素的显示样式；
2. 在CSS中，选择器是一种模式，用于选择需要添加样式的元素；
3. 在Selenium中也可以使用这种选择器来定位元素。

提示：

1. 在selenium中推荐使用CSS定位，因为它比XPath定位速度要快
2. css选择器语法非常强大，在这里我们只学习在测试中常用的几个

## CSS定位方法

```
element = driver.find_element_by_css_selector(css_selector)
```

### 3.2 CSS定位常用策略 (方式)

1. id选择器
2. class选择器
3. 元素选择器
4. 属性选择器
5. 层级选择器

#### id选择器

说明：根据元素id属性来选择

格式：#id

例如：#userA <选择id属性值为userA的元素>

#### class选择器

说明：根据元素class属性来选择

格式：.class

例如：.telA <选择class属性值为telA的所有元素>

#### 元素选择器

说明：根据元素的标签名选择

格式：element

例如：input <选择所有input元素>

#### 属性选择器

说明：根据元素的属性名和值来选择

格式：`[attribute=value]`      `element[attribute=value]`

例如：`[type="password"]` <选择type属性值为password的元素>

## 练习

需求：打开注册A.html页面，完成以下操作

- 1).使用CSS定位方式中id选择器定位用户名输入框，并输入：admin
- 2).使用CSS定位方式中属性选择器定位密码输入框，并输入：123456
- 3).使用CSS定位方式中class选择器定位电话号码输入框，并输入：18600000000
- 4).使用CSS定位方式中元素选择器定位注册按钮，并点击

## 层级选择器

说明：根据元素的父子关系来选择

格式1: `element1>element2`      通过element1来定位element2，并且element2必须为element1的直接子元素

例如1: `p[id='p1']>input`      <定位指定p元素下的直接子元素input>

格式2: `element1 element2`      通过element1来定位element2，并且element2为element1的后代元素

例如2: `p[id='p1'] input`      <定位指定p元素下的后代元素input>

## 练习

需求：打开注册A.html页面，完成以下操作

- 1).使用CSS定位方式中的层级选择器定位用户名输入框，并输入：admin

## 3.3 CSS延伸[了解]

<code>input[type^='p']</code>	type属性以p字母开头的元素
<code>input[type\$='d']</code>	type属性以d字母结束的元素
<code>input[type*='w']</code>	type属性包含w字母的元素

## 3.4 CSS总结

1. 常用的CSS定位选择器有哪些？

## 4. 八种元素定位方式分类-汇总

1. id、name、class\_name: 为元素属性定位
2. tag\_name: 为元素标签名称
3. link\_text、partial\_link\_text: 为超链接定位(a标签)
4. XPath: 为元素路径定位
5. CSS: 为CSS选择器定位

## XPath与CSS类似功能对比

定位方式	XPath	CSS
元素名	//input	input
id	//input[@id='userA']	#userA
class	//*[@class='telA']	.telA
属性	1. //*[text()='xxx'] 2. //input[starts-with(@attribute,'xxx')] 3. //input[contains(@attribute,'xxx')]	1. input[type^='p'] 2. input[type\$='d'] 3. input[type*='w']

## 5. 定位元素的另一种写法--延伸[了解]

方法: find\_element(by=By.ID, value=None)

备注: 需要两个参数, 第一个参数为定位的类型由By提供, 第二个参数为定位的具体方式

示例:

1. driver.find\_element(By.CSS\_SELECTOR, '#emailA').send\_keys("123@126.com")
2. driver.find\_element(By.XPATH, '//\*[@id="emailA"]').send\_keys('234@qq.com')
3. driver.find\_element(By.ID, "userA").send\_keys("admin")
4. driver.find\_element(By.NAME, "passwordA").send\_keys("123456")
5. driver.find\_element(By.CLASS\_NAME, "telA").send\_keys("18611111111")
6. driver.find\_element(By.TAG\_NAME, 'input').send\_keys("123")
7. driver.find\_element(By.LINK\_TEXT, '访问 新浪 网站').click()
8. driver.find\_element(By.PARTIAL\_LINK\_TEXT, '访问').click()

### 5.1 导入By类

导包: from selenium.webdriver.common.by import By

```
class By(object):
    """
    Set of supported locator strategies.
    """

    ID = "id"
```

```
XPATH = "xpath"
LINK_TEXT = "link text"
PARTIAL_LINK_TEXT = "partial link text"
NAME = "name"
TAG_NAME = "tag name"
CLASS_NAME = "class name"
CSS_SELECTOR = "css selector"
```

## 5.2 find\_element\_by\_xxx()和find\_element() 区别

说明：通过查看find\_element\_by\_id底层实现方法，发现底层是调用find\_element方法进行的封装；

```
def find_element_by_id(self, id_):
    """Finds an element by id.

    :Args:
        - id_ - The id of the element to be found.

    :Usage:
        driver.find_element_by_id('foo')
    """
    return self.find_element(by=By.ID, value=id_)
```

# 元素操作|浏览器操作方法

## 目标

1. 掌握常用的元素操作方法
2. 掌握常用的操作浏览器方法
3. 知道常用的获取元素信息的方法

## 1. 元素操作

### 1.1 为什么要学习操作元素的方法？

1. 需要让脚本模拟用户给指定元素输入值
2. 需要让脚本模拟人为删除元素的内容
3. 需要让脚本模拟点击操作

### 1.2 元素常用操作方法

- |                                  |      |
|----------------------------------|------|
| 1. <code>click()</code>          | 单击元素 |
| 2. <code>send_keys(value)</code> | 模拟输入 |
| 3. <code>clear()</code>          | 清除文本 |

### 1.3 案例

需求：打开注册A页面，完成以下操作

- 1).通过脚本执行输入用户名: **admin**; 密码: **123456**; 电话号码: **18611111111**; 电子邮件: **123@qq.com**
- 2).间隔3秒, 修改电话号码为: **18600000000**
- 3).间隔3秒, 点击‘注册’按钮
- 4).间隔3秒, 关闭浏览器
- 5).元素定位方法不限

## 实现步骤难点分析

1. 修改电话号码, 先清除再输入新的号码; 清除 --> `clear()`
2. 点击按钮 --> `click()`

## 2. 浏览器操作

脚本启动浏览器窗口大小默认不是全屏？

如何刷新页面？

通过调用Selenium的API来实现浏览器的操作

### 2.1 操作浏览器常用方法

1. maximize_window()	最大化浏览器窗口 --> 模拟浏览器最大化按钮
2. set_window_size(width, height)	设置浏览器窗口大小 --> 设置浏览器宽、高(像素点)
3. set_window_position(x, y)	设置浏览器窗口位置 --> 设置浏览器位置
4. back()	后退 --> 模拟浏览器后退按钮
5. forward()	前进 --> 模拟浏览器前进按钮
6. refresh()	刷新 --> 模拟浏览器F5刷新
7. close()	关闭当前窗口 --> 模拟点击浏览器关闭按钮
8. quit()	关闭浏览器驱动对象 --> 关闭所有程序启动的窗口
9. title	获取页面title
10. current_url	获取当前页面URL

### 2.2 示例代码

```
# 最大化浏览器
driver.maximize_window()
# 刷新
driver.refresh()
# 后退
driver.back()
# 前进
driver.forward()
# 设置浏览器大小
driver.set_window_size(300,300)
# 设置浏览器位置
driver.set_window_position(300,200)
# 关闭浏览器单个窗口
driver.close()
# 关闭浏览器所有窗口
driver.quit()
# 获取title
title = driver.title
# 获取当前页面url
url = driver.current_url
```

## 3. 获取元素信息

### 3.1 为什么要学习获取元素信息的方法？

1. 如何获取元素的文本？
2. 如何获取元素属性值？
3. 如何让程序判断元素是否为可见状态？

我们想解决以上问题，就需要学习Selenium封装的获取元素信息的方法

### 3.2 获取元素信息的常用方法

- |                                      |                            |
|--------------------------------------|----------------------------|
| 1. <code>size</code>                 | 返回元素大小                     |
| 2. <code>text</code>                 | 获取元素的文本                    |
| 3. <code>get_attribute("xxx")</code> | 获取属性值，传递的参数为元素的属性名         |
| 4. <code>is_displayed()</code>       | 判断元素是否可见                   |
| 5. <code>is_enabled()</code>         | 判断元素是否可用                   |
| 6. <code>is_selected()</code>        | 判断元素是否选中，用来检查复选框或单选按钮是否被选中 |

提示：

1. `size`、`text`：为属性，调用时无括号；如：`xxx.size`

### 3.3 案例

需求：使用‘注册A.html’页面，完成以下操作：

- 1). 获取用户名输入框的大小
- 2). 获取页面上第一个超链接的文本内容
- 3). 获取页面上第一个超链接的地址
- 4). 判断页面中的span标签是否可见
- 5). 判断页面中取消按钮是否可用
- 6). 判断页面中‘旅游’对应的复选框是否为选中的状态

## 4. 总结

1. 常用的元素操作方法？
2. 常用的操作浏览器方法？
3. 常用的获取元素信息的方法？



# 鼠标和键盘操作

## 目标

1. 掌握鼠标操作的方法
2. 掌握键盘操作的方法

## 1. 鼠标操作

常见的鼠标操作有：点击、右击、双击、悬停、拖拽等，对于这些鼠标操作Selenium都封装了相应的操作方法。

### 1.1 为什么要操作鼠标？

现在Web产品中存在丰富的鼠标交互方式，作为一个Web自动化测试框架，需要应对这些鼠标操作的应用场景。

### 1.2 鼠标操作的方法

说明：在Selenium中将操作鼠标的方法封装在ActionChains类中

实例化对象：

```
action = ActionChains(driver)
```

方法：

- |                                  |                        |
|----------------------------------|------------------------|
| 1. context_click(element)        | 右击 --> 模拟鼠标右键点击效果      |
| 2. double_click(element)         | 双击 --> 模拟鼠标双击效果        |
| 3. drag_and_drop(source, target) | 拖动 --> 模拟鼠标拖动效果        |
| 4. move_to_element(element)      | 悬停 --> 模拟鼠标悬停效果        |
| 5. perform()                     | 执行 --> 此方法用来执行以上所有鼠标操作 |

为了更好的学习其他方法，我们先学习perform()执行方法,因为所有的方法都需要执行才能生效

### 1.3 鼠标执行-perform()

说明：在ActionChains类中所有提供的鼠标事件方法，在调用的时候所有的行为都存储在ActionChains对象中，

而perform()方法就是真正去执行所有的鼠标事件。

强调：必须调用perform()方法才能执行鼠标事件

## 1.4 鼠标右键-context\_click()

说明：对于点击鼠标右键，如果弹出的是浏览器默认的菜单，Selenium没有提供操作菜单选项的方法；如果是自定义的右键菜单，则可以通过元素定位来操作菜单中的选项。

### 练习

需求：打开注册页面A，在用户名文本框上点击鼠标右键

### 代码实现关键点分析

1. 导包：from selenium.webdriver.common.action\_chains import ActionChains
2. 实例化ActionChains对象：action = ActionChains(driver)
3. 调用右键方法：action.context\_click(element)
4. 执行：action.perform()

## 1.5 鼠标双击-double\_click()

说明：模拟双击鼠标左键操作

### 练习

需求：打开注册页面A，输入用户名admin，暂停3秒钟后，双击鼠标左键，选中admin

## 1.6 鼠标拖动-drag\_and\_drop()

说明：模拟鼠标拖动动作，选定拖动源元素释放到目标元素

### 拖动关键点分析

1. 源元素 source = driver.find\_element\_by\_id(xxx)
2. 目标元素 target = driver.find\_element\_by\_id(xxx)
3. 调用方法 action.drag\_and\_drop(source, target).perform()

### 练习

需求：打开'drag.html'页面，把红色方框拖拽到蓝色方框上

## 1.7 鼠标悬停-move\_to\_element()

说明：模拟鼠标悬停在指定的元素上

### 练习

需求：打开注册页面A，模拟鼠标悬停在‘注册’按钮上

## 1.8 鼠标操作总结

1. 鼠标右击
2. 鼠标双击
3. 鼠标拖拽
4. 鼠标悬停
5. 鼠标执行

## 2. 键盘操作

思考：如何实现复制、粘贴的操作？

说明：

- 1). 模拟键盘上一些按键或者组合键的输入 如：Ctrl+C 、Ctrl+V;
- 2). Selenium中把键盘的按键都封装在Keys类中

### 2.1 Keys类

导包：from selenium.webdriver.common.keys import Keys

### 2.2 常用的键盘操作

- |                                 |                |
|---------------------------------|----------------|
| 1. send_keys(Keys.BACK_SPACE)   | 删除键(BackSpace) |
| 2. send_keys(Keys.SPACE)        | 空格键(Space)     |
| 3. send_keys(Keys.TAB)          | 制表键(Tab)       |
| 4. send_keys(Keys.ESCAPE)       | 回退键(Esc)       |
| 5. send_keys(Keys.ENTER)        | 回车键(Enter)     |
| 6. send_keys(Keys.CONTROL, 'a') | 全选(Ctrl+A)     |
| 7. send_keys(Keys.CONTROL, 'c') | 复制(Ctrl+C)     |

提示：以上方法就不一个一个讲解了，因为调用方法都一样；

## 2.3 案例

需求：打开注册A页面，完成以下操作

- 1). 输入用户名: admin1, 暂停2秒, 删除1
- 2). 全选用户名: admin, 暂停2秒
- 3). 复制用户名: admin, 暂停2秒
- 4). 粘贴到密码框

## 2.4 示例代码

```
# 定位用户名
element = driver.find_element_by_id("userA")
# 输入用户名
element.send_keys("admin1")
# 删除1
element.send_keys(Keys.BACK_SPACE)
# 全选
element.send_keys(Keys.CONTROL, 'a')
# 复制
element.send_keys(Keys.CONTROL, 'c')
# 粘贴
driver.find_element_by_id('passwordA').send_keys(Keys.CONTROL, 'v')
```

## 2.5 键盘-总结

1. Keys类的作用
2. 键盘操作调用方法

# 元素等待

## 目标

1. 掌握元素的隐式等待
2. 掌握元素的显式等待

## 1. 元素等待

### 1.1 什么是元素等待？

概念：在定位页面元素时如果未找到，会在指定时间内一直等待的过程；

### 1.2 为什么要设置元素等待？

1. 网络速度慢
2. 电脑配置低
3. 服务器处理请求慢

Selenium中元素等待有几种类型呢？

### 1.3 元素等待类型

1. 隐式等待
2. 显式等待

## 2. 隐式等待

概念：定位元素时，如果能定位到元素则直接返回该元素，不触发等待； 如果不能定位到该元素，则间隔一段时间后再去定位元素； 如果在达到最大时长时还没有找到指定元素，则抛出元素不存在的异常 `NoSuchElementException` 。

### 2.1 实现方式

方法: `driver.implicitly_wait(timeout)`  
(timeout: 为等待最大时长, 单位: 秒)

说明: 隐式等待为全局设置 (只需要设置一次, 就会作用于所有元素)

## 2.2 案例

需求: 打开注册A页面, 完成以下操作

- 1). 使用隐式等待定位用户名输入框, 如果元素存在, 就输入admin

## 3. 显式等待

概念: 定位指定元素时, 如果能定位到元素则直接返回该元素, 不触发等待; 如果不能定位到该元素, 则间隔一段时间后再去定位元素; 如果在达到最大时长时还没有找到指定元素, 则抛出超时异常 `TimeoutException`。

在Selenium中把显式等待的相关方法封装在WebDriverWait类中

### 3.1 实现方式

```
1. 导包 等待类 --> from selenium.webdriver.support.wait import WebDriverWait
2. WebDriverWait(driver, timeout, poll_frequency=0.5)
   1). driver: 浏览器驱动对象
   2). timeout: 超时的时长, 单位: 秒
   3). poll_frequency: 检测间隔时间, 默认为0.5秒
3. 调用方法 until(method): 直到...时
   1). method: 函数名称, 该函数用来实现对元素的定位
   2). 一般使用匿名函数来实现: lambda x: x.find_element_by_id("userA")
4. element = WebDriverWait(driver, 10, 1).until(lambda x: x.find_element_by_id("userA"))
```

### 3.2 案例

需求: 打开注册A页面, 完成以下操作

- 1). 使用显式等待定位用户名输入框, 如果元素存在, 就输入admin

### 3.3 示例代码

```
import time
```

```
from selenium import webdriver
from selenium.webdriver.support.wait import WebDriverWait

driver = webdriver.Firefox()
driver.get("file:///D:/webAutoTest/page/注册A.html")

element = WebDriverWait(driver, 10, 1).until(lambda x: x.find_element_by_id("userA"))
element.send_keys("admin")

time.sleep(3)
driver.quit()
```

---

## 4. 显式与隐式区别

1. 作用域：隐式为全局元素，显式等待为单个元素有效
2. 使用方法：隐式等待直接通过驱动对象调用，而显式等待方法封装在WebDriverWait类中
3. 达到最大超时时长后抛出的异常不同：隐式为NoSuchElementException，显式等待为TimeoutException

---

## 5. 元素等待-总结

1. 为什么要设置元素等待
2. 元素等待的过程
3. 元素等待的类型
4. 显式等待与隐式等待区别

# 下拉选择框、弹出框、滚动条操作

## 目标

1. 掌握下拉选择框的操作方法
2. 掌握处理弹出框的方法
3. 掌握调用JavaScript方法

## 1. 下拉选择框操作

说明：下拉框就是HTML中<select>元素；

### 1.1 如何操作下拉选择框？

#### 案例

需求：使用‘注册A.html’页面，完成对城市的下拉框的操作

- 1). 选择‘广州’
- 2). 暂停2秒，选择‘上海’
- 3). 暂停2秒，选择‘北京’

#### 案例实现方式一

思路：先定位到要操作的option元素，然后执行点击操作

#### 问题

操作起来比较繁琐：要先定位到要操作的选项，然后再执行点击操作

### 1.2 Select类

说明：Select类是Selenium为操作select标签特殊封装的。

实例化对象：

```
select = Select(element)
```

element：<select>标签对应的元素，通过元素定位方式获取，

例如：driver.find\_element\_by\_id("selectA")



操作方法:

- |                                 |                          |
|---------------------------------|--------------------------|
| 1. select_by_index(index)       | --> 根据option索引来定位, 从0开始  |
| 2. select_by_value(value)       | --> 根据option属性 value值来定位 |
| 3. select_by_visible_text(text) | --> 根据option显示文本来定位      |

## Select类实现步骤分析

1. 导包 Select类 --> `from selenium.webdriver.support.select import Select`
2. 实例化Select类 `select = Select(driver.find_element_by_id("selectA"))`
3. 调用方法: `select.select_by_index(index)`

## 示例代码

```
#导包
from selenium.webdriver.support.select import Select

select = Select(driver.find_element_by_id("selectA"))
select.select_by_index(2) # 根据索引实现
select.select_by_value("sh") # 根据value属性实现
select.select_by_visible_text("A北京") # 根据文本内容实现
```

## 1.3 总结

1. 通过Select类如何选择下拉框?

## 2. 弹出框处理

网页中常用的弹出框有三种

1. alert 警告框
2. confirm 确认框
3. prompt 提示框

### 2.1 案例

需求: 打开注册A.html页面, 完成以下操作:

- 1). 点击 alert 按钮
- 2). 关闭警告框
- 3). 输入用户名: admin

## 问题

1. 按钮被点击后弹出警告框，而接下来输入用户名的语句没有生效
2. 什么问题导致的？
3. 如何处理警告框？

## 2.2 弹出框处理方法

说明：Selenium中对处理弹出框的操作，有专用的处理方法；并且处理的方法都一样

1. 获取弹出框对象

```
alert = driver.switch_to.alert
```

2. 调用

```
alert.text          --> 返回alert/confirm/prompt中的文字信息
alert.accept()      --> 接受对话框选项
alert.dismiss()     --> 取消对话框选项
```

## 2.3 示例代码

```
# 定位alerta按钮
driver.find_element_by_id("alerta").click()
# 获取警告框
alert = driver.switch_to.alert
# 打印警告框文本
print(alert.text)
# 接受警告框
alert.accept()
# 取消警告框
# alert.dismiss()
```

## 2.4 总结

1. 如何处理弹出框？

## 3. 滚动条操作

滚动条：一种可控制页面显示范围的组件

### 3.1 为什么要学习滚动条操作？

1. 在HTML页面中，由于前端技术框架的原因，页面元素为动态显示，元素根据滚动条的下拉而被加载
2. 页面注册同意条款，需要滚动条到最底层，才能点击同意

## 3.2 实现方式

说明：selenium中并没有直接提供操作滚动条的方法，但是它提供了可执行JavaScript脚本的方法，所以我们可以用JavaScript脚本来达到操作滚动条的目的。

1. 设置JavaScript脚本控制滚动条  
`js = "window.scrollTo(0,1000)"`  
(0:左边距; 1000: 上边距; 单位像素)
2. selenium调用执行JavaScript脚本的方法  
`driver.execute_script(js)`

## 3.3 案例

需求：打开注册页面A，暂停2秒后，滚动条拉到最底层

### 示例代码

```
# 最底层
js1 = "window.scrollTo(0,10000)"
driver.execute_script(js1)

# 最顶层
js2 = "window.scrollTo(0,0)"
driver.execute_script(js2)
```

## 3.4 总结

1. JavaScript如何控制滚动条?
2. Selenium执行JavaScript脚本的方法?

# frame切换、多窗口切换

## 目标

1. 掌握切换frame的方法
2. 掌握多窗口切换的技巧

## 1. frame切换

**frame:** 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；

形式一：[了解]

```
<frameset cols="25%,75%">
    <frame src="frame_a.htm">
    <frame src="frame_b.htm">
</frameset>
```

形式二：

```
<iframe name="iframe_a" src="demo_iframe.htm" width="200" height="200"></iframe>
```

### 1.1 为什么要学习frame切换

案例：打开‘注册实例.html’页面，完成以下操作

- 1). 填写主页面的注册信息
- 2). 填写注册页面A中的注册信息
- 3). 填写注册页面B中的注册信息

## 问题

1. 当前页面内无法定位注册页面A和注册页面B

### 1.2 frame切换方法

说明：在Selenium中封装了如何切换frame框架的方法

方法：

- 1). `driver.switch_to.frame(frame_reference)` --> 切换到指定frame的方法  
    **frame\_reference:** 可以为frame框架的name、id或者定位到的frame元素
- 2). `driver.switch_to.default_content()` --> 恢复默认页面方法

在frame中操作其他页面，必须先回到默认页面，才能进一步操作

## 1.3 案例解决方案

1. 完成主页面注册信息;
2. 调用frame切换方法(`switch_to.frame("myframe1")`)切换到注册用户A框架中
3. 调用恢复默认页面方法(`switch_to.default_content()`)
4. 调用frame切换方法(`switch_to.frame("myframe2")`)切换到注册用户B框架中

## 1.4 frame切换-总结

1. HTML中常用的frame框架
2. 切换框架的方法
3. 恢复到默认页面的方法

## 2. 多窗口切换

说明：在HTML页面中，当点击超链接或者按钮时，有的会在新的窗口打开页面。

### 2.1 为什么要切换窗口？

#### 案例

需求：打开‘注册实例.html’页面，完成以下操作

- 1). 点击‘注册A页面’链接
- 2). 在打开的页面中，填写注册信息

#### 问题

- 1). 无法定位注册A页面中的元素

### 2.2 如何实现多窗口切换？

说明：在Selenium中封装了获取当前窗口句柄、获取所有窗口句柄和切换到指定句柄窗口的方法；  
句柄：英文handle，窗口的唯一识别码

方法：

- 1). `driver.current_window_handle` --> 获取当前窗口句柄

2). driver.window_handles	--> 获取所有窗口句柄
3). driver.switch_to.window(handle)	--> 切换指定句柄窗口

## 2.3 案例解决方案分析

1. 获取‘注册实例.html’当前窗口句柄
2. 点击‘注册实例.html’页面中注册A页面
3. 获取所有窗口句柄
4. 获取注册A页面对应的窗口句柄，并切换
5. 操作注册A页面元素

## 2.4 多窗口切换-总结

1. 什么是句柄？
2. 获取当前窗口句柄方法
3. 获取所有窗口句柄方法
4. 切换指定句柄窗口方法

# 窗口截图、验证码处理

## 目标

1. 掌握窗口截图方法
2. 熟悉验证码处理的方式

思考：如果自动化测试脚本运行时出现了异常，该如何定位问题？

## 1. 窗口截图

说明：把当前操作的页面，截图保存到指定位置

### 1.1 为什么要窗口截图？

自动化脚本是由程序去执行的，因此有时候打印的错误信息并不是十分明确。如果在执行出错的时候对当前窗口截图保存，那么通过图片就可以非常直观地看到出错的原因。

### 1.2 窗口截图的方法

说明：在Selenium中，提供了截图方法，我们只需要调用即可

方法：

```
driver.get_screenshot_as_file(imgpath)
imgpath: 图片保存路径
```

### 1.3 案例

需求：打开‘注册A.html’页面，完成以下操作

- 1). 填写注册信息
- 2). 截图保存

### 1.4 示例代码

```
driver.find_element_by_id("userA").send_keys("admin")
```

```
driver.get_screenshot_as_file("./img/img01.jpg")
```

## 2. 验证码

说明：一种随机生成的信息（数字、字母、汉字、图片、算术题）等为了防止恶意的请求行为，增加应用的安全性。

### 2.1 为什么要学习验证码？

在Web应用中，大部分系统在用户登录注册的时候都要求输入验证码，而我们在设计自动化测试脚本的时候，就需要面临处理验证码的问题。

### 2.2 验证码的处理方式

说明：Selenium中并没有对验证码处理的方法，在这里我们介绍一下针对验证码的几种常用处理方式

方式：

- 1). 去掉验证码  
(测试环境下-采用)
- 2). 设置万能验证码  
(生产环境和测试环境下-采用)
- 3). 验证码识别技术  
(通过Python-tesseract来识别图片类型验证码；识别率很难达到100%)
- 4). 记录cookie  
(通过记录cookie进行跳过登录)

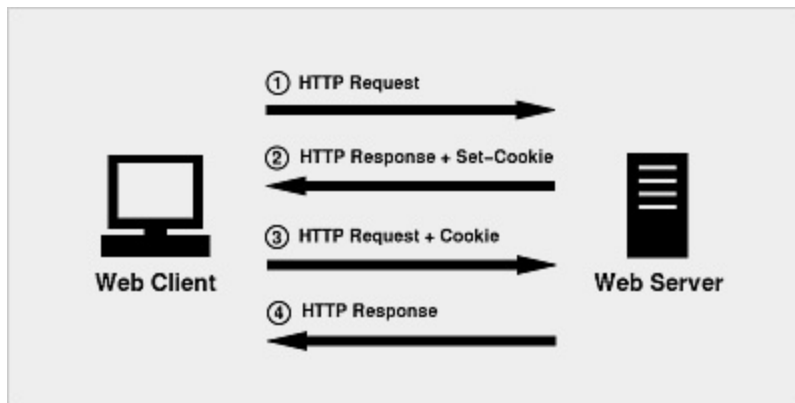
提示

1. 去掉验证码、设置万能验证码：都是开发来完成，我们在这里不做讲解
2. 验证码识别技术：成功率不高，验证码种类繁多，不太适合
3. 记录cookie：比较实用，我们对它进行下讲解

## 3. cookie

### 3.1 cookie是什么？





1. Cookie是由Web服务器生成的，并且保存在用户浏览器上的小文本文件，它可以包含用户相关的信息。
2. Cookie数据格式：键值对组成（python中的字典）
3. Cookie产生：客户端请求服务器，如果服务器需要记录该用户状态，就向客户端浏览器颁发一个Cookie数据
4. Cookie使用：当浏览器再次请求该网站时，浏览器把请求的数据和Cookie数据一同提交给服务器，服务器检查该Cookie，以此来辨认可用户状态。

## 3.2 cookie的应用场景

1. 实现会话跟踪，记录用户登录状态
2. 实现记住密码和自动登录的功能
3. 用户未登录的状态下，记录购物车中的商品

## 4. Selenium操作cookie

说明：Selenium中对cookie操作提供相应的方法

方法：

1. `get_cookie(name)` --> 获取指定cookie  
    name: 为cookie的名称
2. `get_cookies()` --> 获取本网站所有本地cookies
3. `add_cookie(cookie_dict)` --> 添加cookie  
    cookie\_dict: 一个字典对象，必选的键包括: "name" and "value"

### 4.1 案例

需求：使用cookie实现跳过登录

- 1). 手动登录百度，获取cookie
- 2). 使用获取到的cookie，达到登录目的，然后就可以执行登录之后的操作

## 4.2 案例实现步骤分析

BDUSS是登录百度后的唯一身份凭证(\*.baidu.com)，拿到BDUSS就等于拿到帐号的控制权，通行贴吧、知道、百科、文库、空间、百度云等百度主要产品。

1. 登录baidu，登录成功后抓取 (BDUSS)
2. 使用add\_cookie()方法，添加 (BDUSS)键和值
3. 调用刷新方法 driver.refresh()

## 4.3 示例代码

```
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.add_cookie({'name': 'BDUSS', 'value': '根据实际情况填写'})
time.sleep(3)
driver.refresh()
time.sleep(3)
driver.quit()
```

## 5. 窗口截图、验证码总结

1. 窗口截图的方法？
2. 验证码常用的处理方式？
3. Selenium操作cookie的方法？