Table of Contents

Introduction 1.1

第0章-Web自动化入门 1.2

- Web自动化工具选择 1.2.1
- Selenium IDE安装与运行 1.2.2

第1章-WebDriver-定位篇 1.3

- 元素定位 1.3.1
- 元素定位-Xpath、CSS 1.3.2

第2章-WebDriver-操作篇 1.4

- 元素操作|浏览器操作方法 1.4.1
- 元素等待 1.4.2
- 下拉选择框、警告框处理、滚动条操作 1.4.3
- frame表单切换、多窗口切换 1.4.4
- 窗口截图、验证码处理 1.4.5

Web自动化测试

第一章

目标

1. 什么是Web自动化测试
2. Web自动化测试工具

Web自动化测试

目标

了解什么是Web自动化测试

1. 什么是Web自动化测试？

概念：让程序代替人为自动验证Web项目功能的过程

2. 为什么要进行自动化测试?

弄明白下面的几个问题

2.1 Web自动化测试在什么阶段开始？

功能测试完毕(手工测试)

手工测试:由人一个一个输入用例,然后观察结果;

2.2 Web自动化测试属于什么类型的测试?

1. 黑盒测试(功能测试)
2. 白盒测试(单元测试)
3. 灰盒测试(接口测试)

提示:Web自动化测试属于黑盒测试(功能测试)

2.3 自动化测试能解决什么问题？

1. 解决-回归测试
2. 解决-压力测试
3. 解决-兼容性测试
4. 提高测试效率,保证产品质量

回归测试：项目在发新版本之后对项目之前的功能进行验证； 压力测试：可以理解多用户同时去操作软件，统计软件服务器处理多用户请求的能力 兼容性测试：不同浏览器（IE、Firefox、Chrome）等等

2.4 自动化测试优缺点

优点
1．较少的时间内运行更多的测试用例；
2．自动化脚本可重复运行；
3．减少人为的错误；
4．测试数据存储；
缺点
1．自动化测试不可以完全替代手工测试；
2．自动化测试发现新缺陷的能力比手工测试弱；
4．自动化测试对测试人员要求高

Selenium IDE安装与运行

目标

1. 使用Selenium IDE录制脚本
2. 使用Selenium IDE录制的脚本转换成Python语言

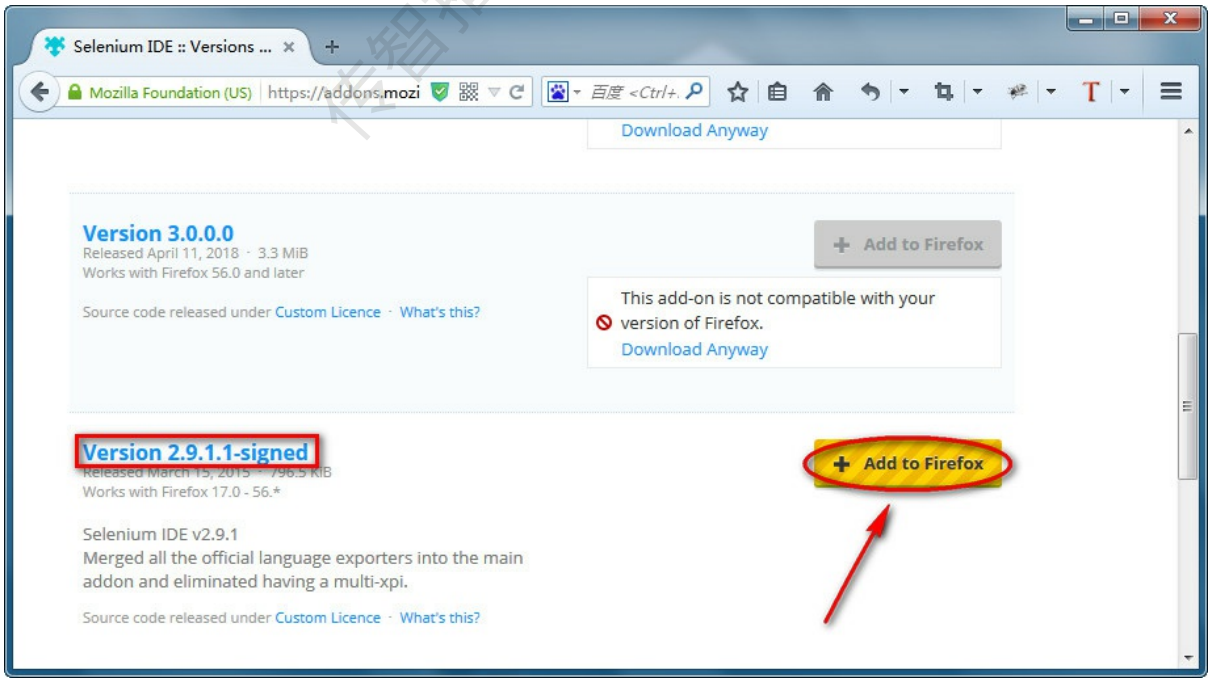
1. Selenium IDE 是什么？

Selenium IDE：是一个Firefox插件，用于记录和播放用户与浏览器的交互。（录制Web操作脚本）

1.1 为什么要学习Selenium IDE？

1. 使用Selenium IDE录制脚本，体验自动化脚本魅力
2. 使用Selenium IDE录制的脚本转换为代码语言
   (在后期我们自己设计脚本时，如果不知道用什么方式定位元素，可使用此方法参考)

1.2 安装方式



1. 官网安装

Version: 2.9.1.1

通过官网安装插件：https://addons.mozilla.org/en-GB/firefox/addon/selenium-ide/versions/

2. 附加组件管理器

1). 火狐浏览器 V24-V35

2). 附加组件管理器-->搜索selenium IDE

提示：

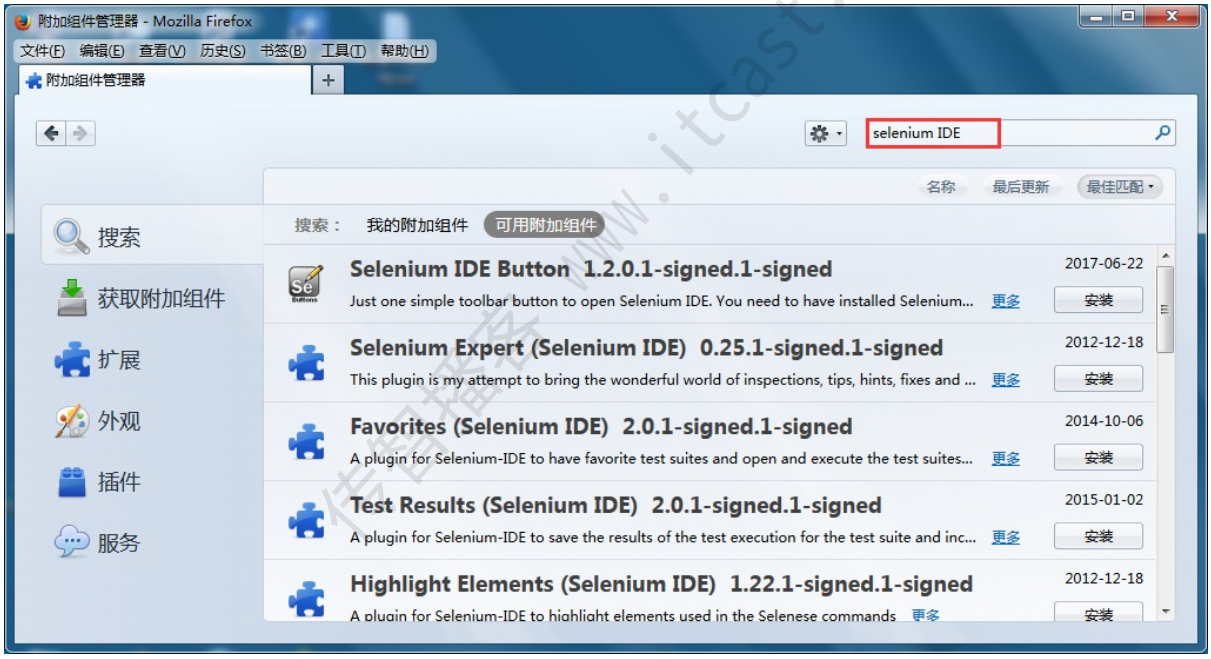
1. IDE前面有个空格
2. 附加组件管理器启动方式-
    1) 工具菜单->附加组件
    2) Ctrl+Shift+A

3. 离线安装

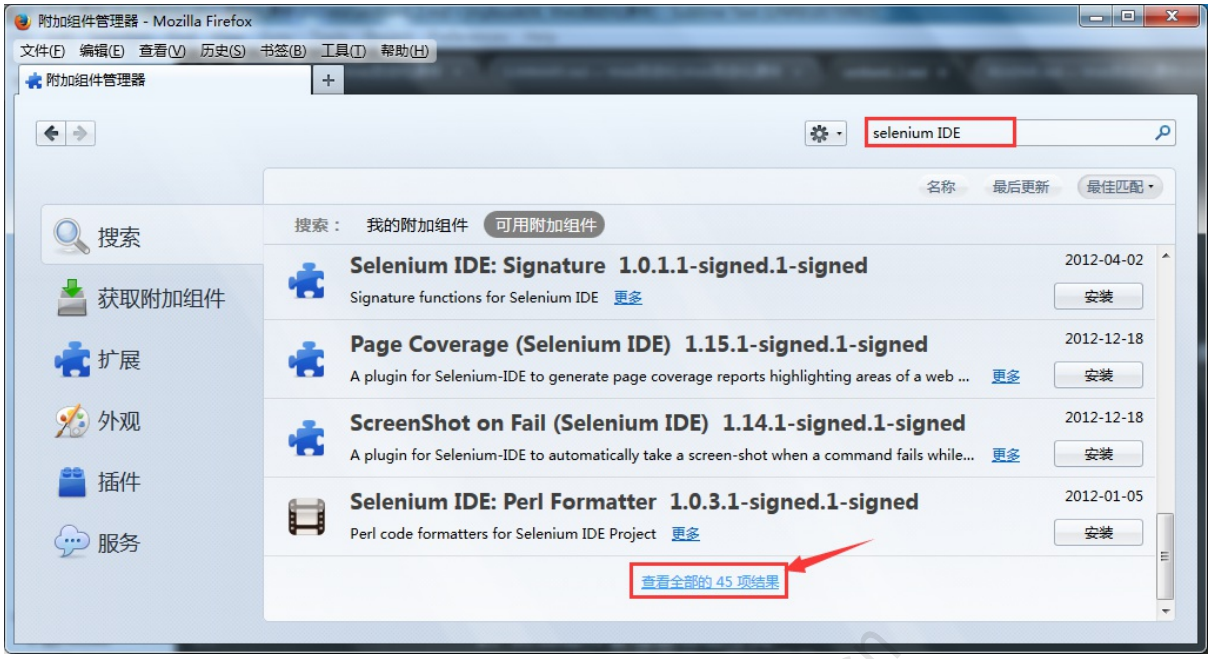
下载：https://github.com/SeleniumHQ/selenium-ide/releases

安装：下载好selenium_ide-2.9.1-fx.xpi直接拖入浏览器安装

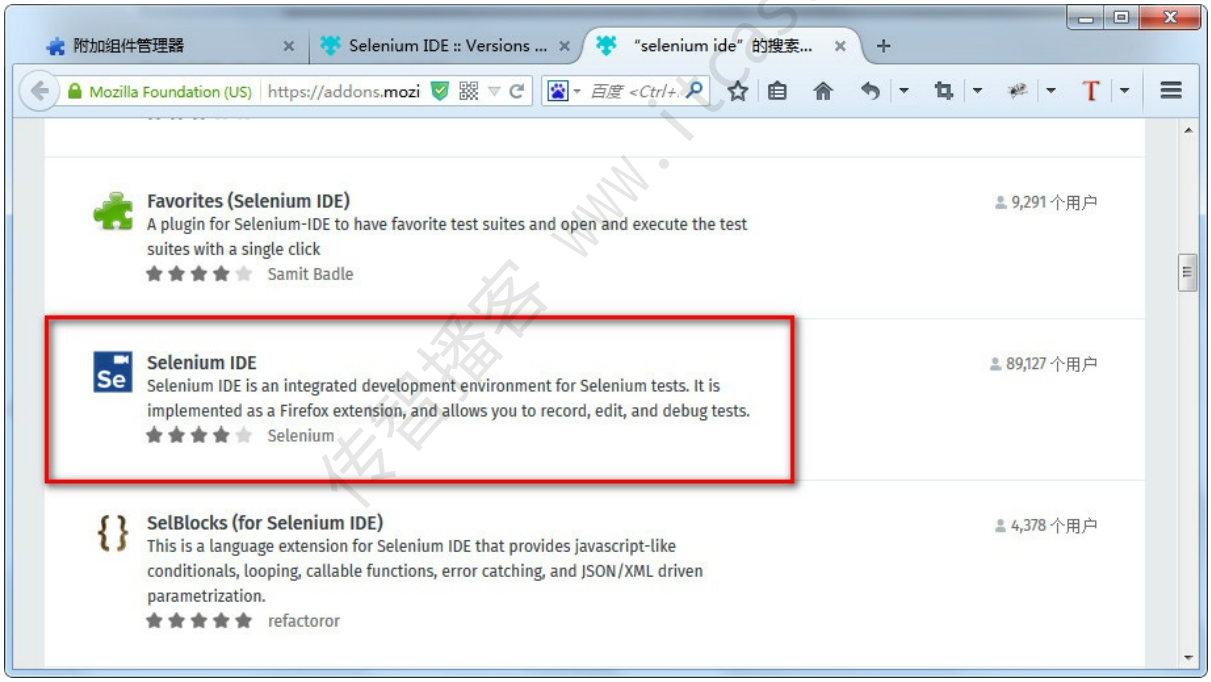
在线安装 搜索**selenium IDE**示意图



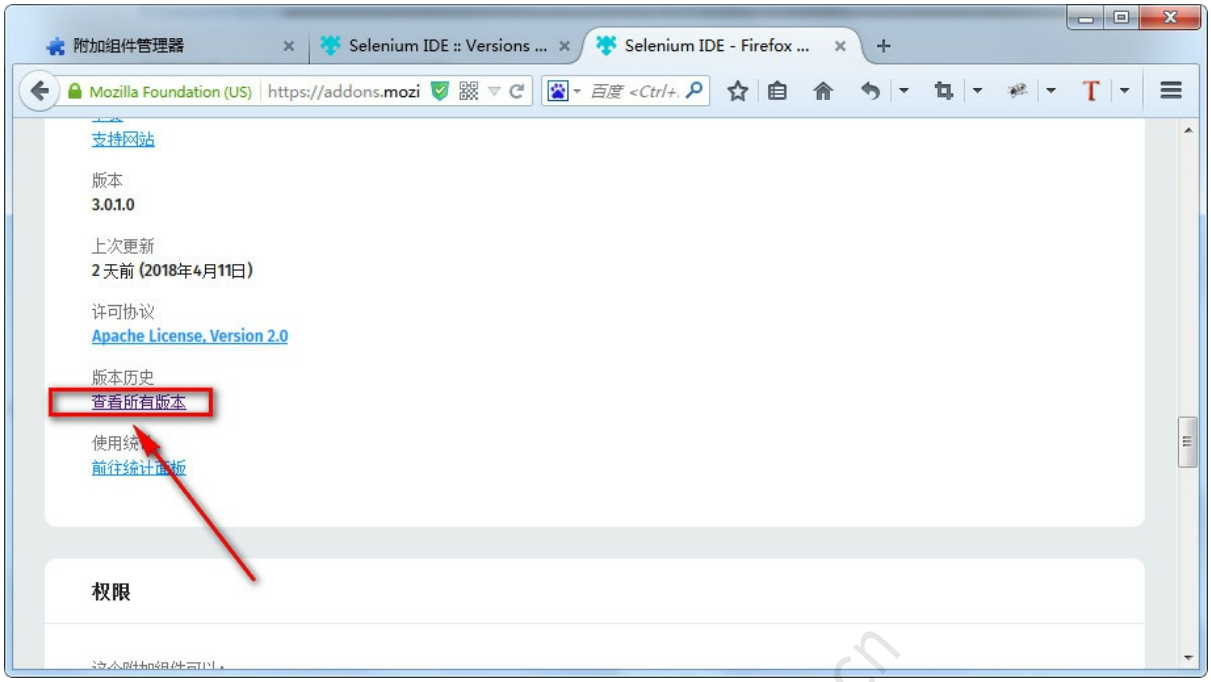
在线安装 点击查看全部**XX**项结果



在线安装 选择**selenium IDE**安装



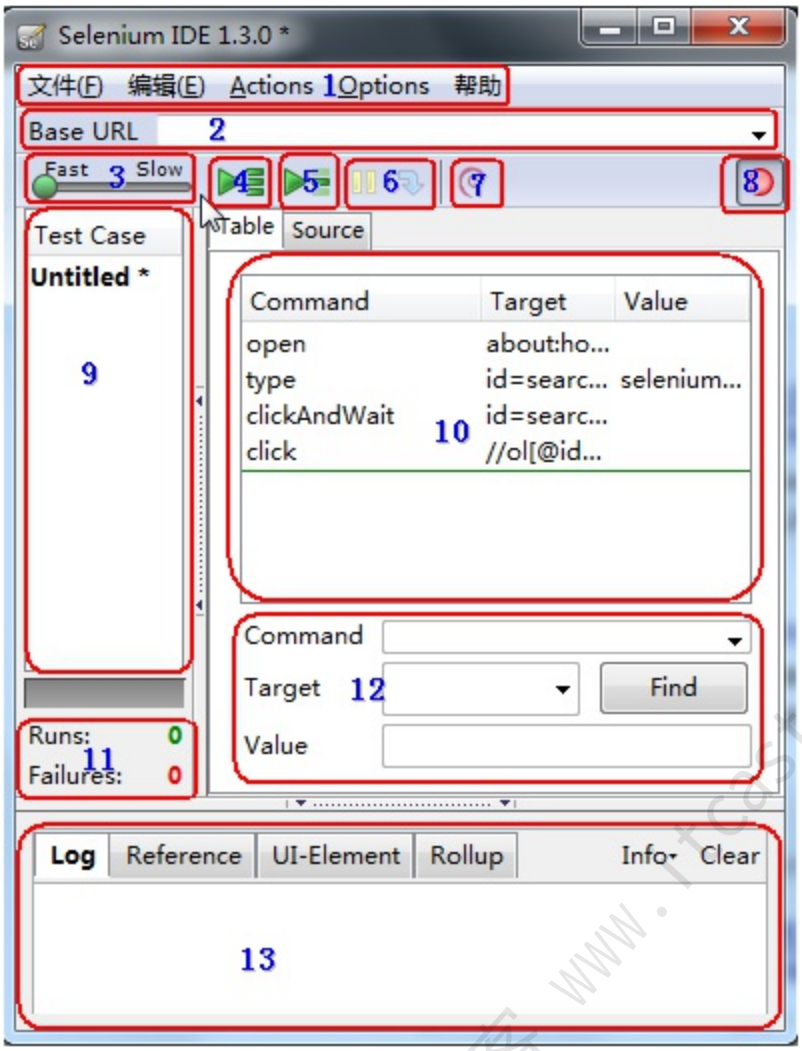
滚动条下拉选择 查看所有版本



火狐浏览器 V35.0 选择 selenium IDE 2.9.1.1

1.3 Selenium IDE运行

1. Ctrl+Alt+S
2. 工具栏—>Selenium IDE



1. 文件：创建、打开和保存测试案例和测试案例集。编辑：复制、粘贴、删除、撤销和选择测试案例中的所有命

令。Options ： 用于设置seleniunm IDE。

2. 用来填写被测网站的地址。
3. 速度控制：控制案例的运行速度。
4. 运行所有：运行一个测试案例集中的所有案例。
5. 运行：运行当前选定的测试案例。
6. 暂停/恢复：暂停和恢复测试案例执行。
7. 单步：可以运行一个案例中的一行命令。
8. 录制：点击之后，开始记录你对浏览器的操作。
9. 案例集列表。
10. 测试脚本；table标签：用表格形式展现命令及参数。source标签：用原始方式展现，默认是HTML语言格式，

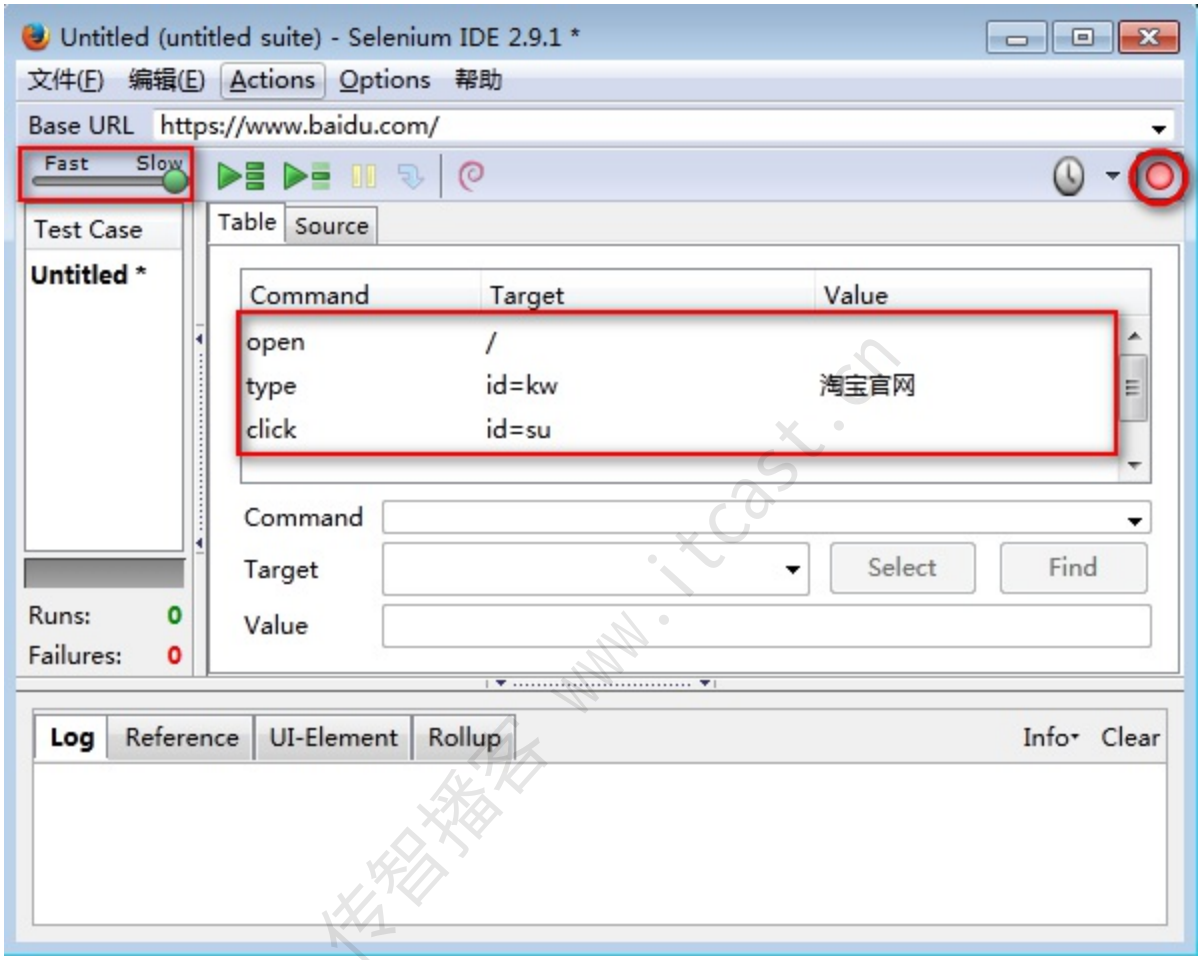
也可以用其他语言展示。

11. 查看脚本运行通过/失败的个数。
12. 当选中前命令对应参数。
13. 日志/参考/UI元素/Rollup

练习1

需求：使用Selenium IDE插件录制->打开百度搜索引擎，搜索框输入[淘宝官网]，点击[百度一下]按钮

练习1 脚本



重点分析：

1. 录制：录制时红色录制按钮一定要打开->按下状态
2. 回放：由于网络延迟原因-建议选择最低
3. 浏览器：回放时浏览器要保持打开状态（否则点击回放，脚本无响应）

重点说明：

1. id=kw：为百度搜索文本框id属性和值
2. id=su：为百度一下按钮id属性和值

思考？

如何快速查找一个元素标签的属性和值？

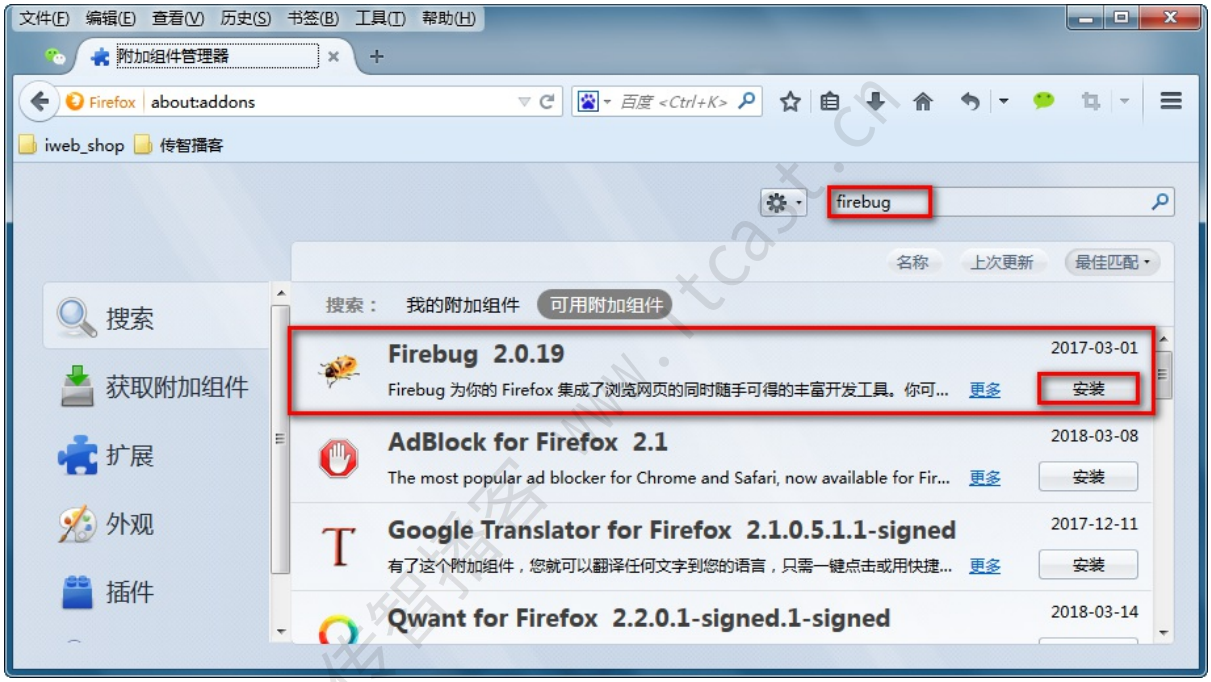
2. 定位调试插件

1. FireBug【重要】

FireBug插件是火狐浏览器一款插件，能够调试所有网站语言，同时也可以快速定位HTML页面中的元素；

作用：定位元素(获取元素定位和查看元素属性)；

2.1 Firebug 插件安装

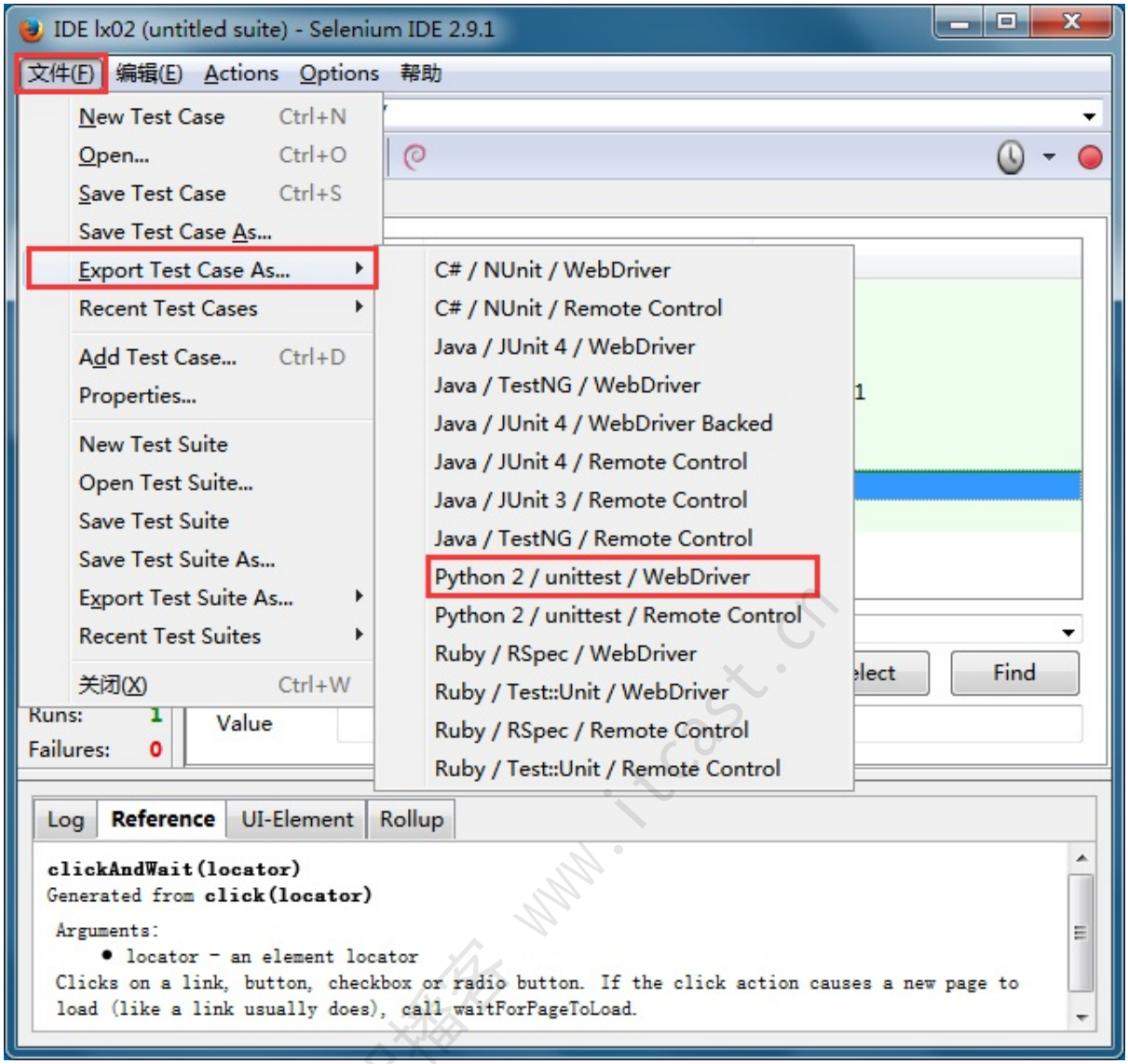


在线安装：

1). 火狐浏览器 V35

2). 附加组件管理器-->搜索FireBug

将练习2脚本 转换成**Python**语言



将录制的动作导出为脚本：

1. 选择“文件”菜单。
2. 选择“Export Test Case As…”
3. 导出为“Python2/unittest/WebDriver”。

切记：导出为脚本时，脚本的名称必须符合python中标识符的命名规则：由字母、数字下划线组成，且只能以字母或下划线开头

第2章-定位

目标

1. 掌握WebDriver元素定位方法

回顾 Selenium家族

对于我们只需要关注以下两点：
1. SeleniumIDE(已学完)
2. Selenium2.0(WebDriver)

提示：
    1). Selenium2.0=Selenium1.0+WebDriver
    2). Selenium2.0以后我们简称WebDriver

WebDriver-元素定位

目标

1. 了解元素各种定位方法
2. 掌握id、name、class_name、tag_name、link_text、partial_link_text定位的使用

1. 为什么要学习元素定位方式？

1. 让程序操作指定元素，就必须先找到此元素；
2. 程序不像人类用眼睛直接定位到元素；
3. WebDriver提供了八种定位元素方式

2. WebDriver 元素定位方式

1. id
2. name
3. class_name
4. tag_name
5. link_text
6. partial_link_text
7. Xpath
8. Css

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

案例-1 注册页面

1. 为了更好的学习这八种方式和网络的关系，我们在案例-1注册页面上来练习自动化脚本设计，提高学习效率和

脚本执行速率
2. 语言使用Python
3. 开发工具使用Pycharm
4. selenium使用2.48.0

2.1 id定位

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

实现案例-1需求：
1). 打开注册A.html页面，使用id定位，自动填写(账号A：admin、密码A:123456)
2). 填写完毕后，3秒钟关闭浏览器窗口

id定位方法

```
find_element_by_id()
```

id定位实现 步骤分析

1. 导入selenium包 --> from selenium import webdriver
2. 导入time包 --> from time import sleep
3. 实例化火狐浏览器 --> driver=webdriver.Firefox()
4. 打开注册A.html --> driver.get(url)
5. 调用id定位方法 --> driver.find_element_by_id("")
6. 使用send_keys()方法发送数据 --> .send_keys("admin")
7. 暂停3秒 --> sleep(3)
8. 关闭浏览器 --> quit()

说明：为了接下来更好的而学习体验，我们先暂时使用下，send_keys()和quit()方法，在2.4节元素操作讲解；

id定位 案例-1代码：

```
from selenium import webdriver
from time import sleep
driver=webdriver.Firefox()
url='E:\\测试\\课件\\Web自动化\\Web自动化课件\\02img\\注册A.html'
driver.get(url)
user=driver.find_element_by_id("userA")
user.send_keys("admin")
pwd=driver.find_element_by_id("passwordA")
pwd.send_keys("123456")
sleep(3)
```

```
driver.quit()
```

id定位-总结

```
1. 导包
2. url中\\转义
3. id定位方法
3. 发送内容方法
4. 暂停方法
5. 关闭浏览器
```

2.2 name定位

```
说明：HTML规定name属性来指定元素名称，因此它的作用更像人名，name的属性值在当前文档中可以不是唯一的
      ，name定位就是根据元素name属性来定位
前提：元素有name属性

实现案例-1需求：
    1). 打开注册A.html页面，使用name定位，自动填写(账号A：admin、密码A:123456)
    2). 填写完毕后，3秒钟关闭浏览器窗口
```

name定位方法

```
find_element_by_name()
```

name定位实现 步骤分析

```
1. 参考id定位
```

2.3 class_name定位

```
说明：HTML规定了class来指定元素的类名，用法和name、id类似；
前提：元素有class属性

实现案例-1需求：
    通过class_name定位电话号码A，并发送18611111111
```

class_name定位方法

```
find_element_by_class_name()
```

class_name定位实现 步骤分析

```
1. 参考id定位
```

2.4 tag_name定位

```
说明：HTML本质就是由不同的tag(标签)组成，而每个tag都是指同一类，所以tag定位效率低，
      一般不建议使用；tag_name定位就是通过标签名来定位；
实现案例-1需求：
    1). 打开注册A.html页面，使用tag_name定位，自动填写(账号A：admin)
    2). 填写完毕后，3秒钟关闭浏览器窗口
```

tag_name定位方法

```
1. find_element_by_tag_name()
        返回：符合条件的第一个标签
2. 如何获取第二个元素？稍后(2.7节)讲解
```

tag_name定位实现 步骤分析

```
1. 参考id定位
```

2.5 link_text定位

```
说明：link_text定位与前面4个定位有所不同，它专门用来定位超链接文本（<a>标签</a>）。

实现案例-1需求：
    1). 打开注册A.html页面，使用link_text定位(访问 新浪 网站)超链接
    2). 3秒钟关闭浏览器窗口
```

link_text定位方法

```
1. 方法：find_element_by_link_text()
2. 说明：需要传入a标签全部文本(访问 新浪 网站)
```

link_text 步骤分析

```
1. 参考id定位
```

```
2. 点击 --> click()
```

2.6 partial_link_text定位

说明：partial_link_text定位是对link_text定位的补充，partial_like_text为模糊匹配；link_text全部匹配

实现案例-1需求：

1). 打开注册A.html页面，使用partial_link_text定位(访问 新浪 网站)超链接
2). 3秒钟关闭浏览器窗口

partial_link_text定位方法

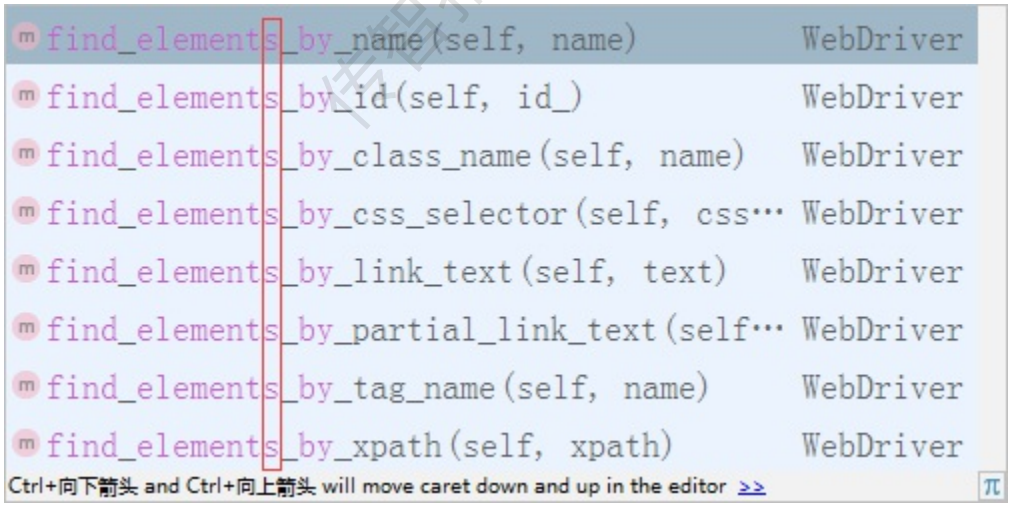
```
1. 方法：find_element_by_partial_link_text()
2. 说明：需要传入a标签局部文本-能表达唯一性(访问 新浪 网站)
```

partial_link_text 步骤分析

```
1. 参考link_text定位
```

提示

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：



2.7 find_element[s]_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)
```

操作**(2.4 tag_name)**

```
说明:使用tag_name获取第二个元素(密码框)

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

---

3. 2.1-2.6定位 总结

```
1. id、name、class_name
2. tag_name
3. link_text、partial_link_text
4. find_elements_by_XXX()
```

思考?

```
1. 在实际项目中标签没有id、name、class属性改如何定位?
2. id、name、class属性值为动态获取,随着刷新或加载而变化,改如何定位?
```

Xpath、CSS定位

目标

1. 熟悉Xpath定位策略
2. 熟悉CSS定位策略

为什么要学习Xpath、CSS定位？

1. 在实际项目中标签没有id、name、class属性
2. id、name、class属性值为动态获取，随着刷新或加载而变化

1. 什么是Xpath？

1. XPath即为XML Path 的简称，它是一种用来确定XML文档中某部分位置的语言。
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

提示：Xpath为强大的语言，那是因为它有非常灵活定位策略；

思考

Xpath有那些策略呢？

2. Xpath定位策略(方式)

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

Xpath定位 方法

```
driver.find_element_by_xpath()
```

2.1 路径(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路径 ；如:/html/body/div/p[2]
          提示：
              1)．绝对路径以/开始
              2)．使用Firebug可以快速生成，元素XPath绝对路径

相对路径：从第一个符合条件元素开始(一般配合属性来区分)；如：//input[@id='userA']
          提示：
              1)．相对路径以//开始
              2)．使用Friebug扩展插件FirePaht可快速生成，元素相对路径

提示：为了方便练习Xpath，可以在FireBug内安装扩展插件-FireFinder插件；
      1)．火狐浏览器-->组件管理器-->搜索FireFinder
```

使用Xpath实现 案例-1

```
需求：
    1)．使用绝对路径和相对路径分别实现，账号A：admin;密码A：123456；自动化脚本设计
```

2.2 利用元素属性

```
说明：快速定位元素，利用元素唯一属性；
示例：//*[@id='userA']
```

2.3 层级与属性结合

```
说明：要找的元素没有属性，但是它的父级有；
示例：//*[@id='p1']/input
```

2.4 属性与逻辑结合

```
说明：解决元素之间个相同属性重名问题
示例：//*[@id='telA' and @class='telA']
```

2.5 Xpath-延伸

```
//*[text()="xxx"]                                文本内容是xxx的元素

//*[starts-with(@attribute,'xxx')]                  属性以xxx开头的元素

//*[contains(@attribute,'Sxxx')]                  属性中含有xxx的元素
```

2.6 Xpath-总结

```
1. 如何通过Friebug快速生成绝对路径
2. 如果通过Friebug快速生成相对路径
3. Xpath策略有哪些
```

---

3. CSS定位

3.1 什么是CSS？

```
1. CSS（Cascading Style Sheets）是一种语言，它用来描述HTML和XML的元素显示样式；
   (css语言书写两个格式：
            1. 写在HTML语言中<style type="text/css">...
            2. 写在单独文件中 后缀.css
       )
2. 在CSS语言中有CSS选择器(不同的策略选择元素)，在Selenium中也可以使用这种选择器；

提示：
    1. 在selenium中极力推荐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

```
driver.find_element_by_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 如：#userA <选择id属性值为userA的所有元素>
```

使用CSS实现 案例-2

```
需求：
    1). 使用CSSid定位实现，账号A：admin;密码A：123456；自动化脚本设计
```

class选择器

```
说明：根据元素class属性来选择
格式：.class 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 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] 如：[type="password"] <选择所有type属性值为password的值>
```

层级选择器

```
说明：根据元素的父子关系来选择
格式：element>element 如：p>input <返回所有p元素下所有的input元素>
提示：> 可以用空格代替 如：p input 或者 p [type='password']
```

3.3 CSS延伸

```
1. input[type^='p'] 说明：type属性以p字母开头的元素
2. input[type$='d'] 说明：type属性以d字母结束的元素
3. input[type*='w'] 说明：type属性包含w字母的元素
```

3.4 CSS总结

选择器	例子	描述
#id	#userA	id选择器，选择id="userA"的所有元素
.class	.telA	class选择器，选择class="telA"的所有元素
element	input	选择所有input元素
[attribute=value]	[type="password"]	选择type="password"的所有元素
element>element	p>input	选择所有父元素为p元素的input元素

4. 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. //※[starts-with(@attribute,'xxx')] 3. //※[contains(@attribute,'xxx')]	1. input[type^='p'] 2. input[type$='d'] 3. input[type*='w']

说明：由于显示排版原因以上所有(※)号代替(*)

5. 八种元素定位总结：

```
1. id
2. name
3. class_name
4. tag_name
5. link_text
6. partial_link_text
7. Xpath
8. Css

说明：
    1). 元素定位我们就学到这里了
    2). WebDriver除了提供以上定位API方法(driver.find_element_by_xxx())
        外，还提供了另外一套写法；
    3). 调用find_element()方法，通过By来声明定位的方法，并且传入对应的方法和参数（了解-熟悉即
可）
```

6. 定位(另一种写法)-延伸【了解】

说明：第二种方法使用By类的封装的方法,所以需要导入By类包

6.1 导入By类

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

6.2 By类的方法

方法：find_element(By.ID,"userA")
　　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()
```

6.3 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，发现底层也是调用的By类方法进行的封装：

```
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_)
```

总结：虽然方法一样，但WebDriver推荐 find_element_by_xxx()这种方法

第3章-操作

目标

0. 掌握元素操作和浏览器操作
1. 掌握元素等待、警告框的操作
2. 掌握下拉选择框和滚动条的操作
3. 掌握切换frame表单及窗口截图
4. 了解截图操作和验证码处理

元素操作方法

目标

1. 掌握WebDriver常用的元素操作方法
2. 掌握WebDriver常用的操作浏览器方法

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浏览器指定元素输入值
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按钮

2. 元素常用操作方法

1. clear()　　清除文本
2. send_keys()　　模拟输入
3. click()　　单击元素

说明：由于这三个方法非常简单，并且有些之前已经使用过，所以在这里用一个案例一起来讲解

2.1 案例-1 用户注册A

需求：

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秒后，修改电话号码为：18600000000
3. 间隔3秒，点击注册用户A
4. 间隔3秒，关闭浏览器
5. 元素定位方法不限

2.2 案例-1 实现步骤难点分析：

1. 间隔3秒 --> sleep(3)
2. 修改电话号码，先清除在输入新的号码； 清除 --> clear()
3. 点击按钮 --> click()

问题

1. 脚本启动浏览器窗口大小默认不是全屏？
2. 脚本执行结束如何关闭浏览器？

---

3. 浏览器常用方法

说明：主要了解通过WebDriver操作浏览器的常用方法

3.1 WebDriver操作浏览器常用方法

```
1. maximize_window()                最大化 --> 模拟浏览器最大化按钮
2. set_window_size(100,100)           浏览器大小 --> 设置浏览器宽、高(像素点)
3. set_window_position(300,200)     浏览器位置 --> 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 --> 模拟浏览器关闭按钮(关闭单个窗口)
8. quit()                           关闭 --> 关闭所有WebDriver启动的窗口
```

3.2 WebDriver 操作浏览器方式-总结

```
# 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()
```

4. WebDriver 其他常用的方法

4.1 为什么要学习WebDriver其他方法？

```
1. 如何获取元素大小？
2. 如果获取元素的文本？
3. 如何获取元素属性值？
4. 如果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WebDriver封装其他操纵元素的方法
```

4.2 WebDriver其他常用方法

```
1. size                  返回元素大小
2. text                  获取元素的文本
3. title                  获取页面title
4. current_url             获取当前页面URL
5. get_attribute("xxx") 获取属性值;xxx：要获取的属性
6. is_display()             判断元素是否可见
7. is_enabled()             判断元素是否可用

提示：
    1. size、text、title、current_url：为属性，调用时无括号；如：xxx.size
    2. title、current_url：使用浏览器实例化对象直接调用；     如： driver.title
```

4.3 WebDriver其他常用方法 总结

```
....
# 获取用户名文本框大小
size=driver.find_element_by_id("userA").size
print('size:',size)
# 获取a标签内容
text=driver.find_element_by_id("fwA").text
print('a标签text:',text)
# 获取title
title=driver.title
print('title:',title)
# 获取当前页面url
url=driver.current_url
print('url:',url)
# 获取a标签href属性值
href=driver.find_element_by_id("fwA").get_attribute("href")
print('href属性值为:',href)
# 判断span是否显示
```

```
display=driver.find_element_by_css_selector('span').is_displayed()
print('span标签是否显示：',display)
# 判断取消按钮是否可用
enabled=driver.find_element_by_id('cancelA').is_enabled()
print('取消按钮是否可用：',enabled)

执行结果：
size: {'height': 30, 'width': 163}
a标签text: 访问 新浪 网站
title: 注册A
url: file:///E:/%E6%B5%8B%E8%AF%95/%E8%AF%BE%E4%BB%B6/Web%E8%87%AA%E5%8A%A8%E5%8C%96/Web%E8%87%AA%E5%8A%A8%E5%8C%96%E8%AF%BE%E4%BB%B6/02img/%E6%B3%A8%E5%86%8CA.html
href属性值为: http://www.sina.com.cn/
span标签是否显示： False
取消按钮是否可用： False
```

设置元素等待

目标

1. 了解元素显式等待
2. 掌握元素隐式等待

1. 元素等待

1.1 什么是元素等待？

概念：WebDriver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. 由于网络速度原因
2. 电脑配置原因
3. 服务器处理请求原因

WebDriver元素等待有几种类型呢？

1.3 元素等待类型

1. 显式等待
2. 隐式等待

2. 显式等待【了解】

概念：使WebDriver等待指定元素条件成立时继续执行，否则在达到最大时长时抛出超时异常(TimeoutException)

提示：

1). 在WebDriver中把显式等待的相关方法封装在WebDriverWait类中

2). 等待是判定条件成立时，那如何判断条件成立？相关判断的方法封装在expected_conditions类中

2.1 案例-1 注册页面A

需求：
    1. 如果用户名文本框存在，就输入admin

2.2 实现难点分析

```
1. 导包 等待类          --> from selenium.webdriver.support.wait import WebDriverWait
2. 导包 判断条件      --> from selenium.webdriver.support import expected_conditions as E
C
                          (将expected_conditions 通过as关键字起个别名：EC)
3. WebDriverWait(driver, timeout, poll_frequency=0.5)
        1). driver：浏览器对象
        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
4. 调用方法 until(method)：直到..时
        1). method：调用EC.presence_of_element_located(element)
                    element：调用By类方法进行定位
```

2.3 案例-1 代码示例

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait
url = r'E:\测试\课件\Web自动化\Web自动化课件\02img\注册A.html'
driver = webdriver.Firefox()
driver.get(url)
element = WebDriverWait(driver, 5).until(EC.presence_of_element_located((By.ID, 'userA'
)))
element.send_keys("admin")
```

3. 隐式等待【掌握】

说明：等待元素加载指定的时长，超出抛出NoSuchElementException异常，实际工作中，一般都使用隐式等待；

显式与隐式区别：
    1. 作用域：显式等待为单个元素有效，隐式为全局元素
    2. 方法：显式等待方法封装在WebDriverWait类中，而隐式等待则直接通过浏览器实例化对象调用

3.1 隐式等待调用方法

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

调用：driver.implicitly_wait(10)
　　(driver：为浏览器实例化对象名称)

3.2 隐式等待执行-说明

如果定位某一元素定位失败，那么就会触发隐式等待有效时长，如果在指定时长内加载完毕，则继续执行，否则
抛出NoSuchElementException异常，如果元素在第一次就定位到则不会触发隐式等待时长；

---

4. 元素等待-总结

1. 为什么要设置元素等待
2. 显式等待与隐式等待区别
3. 掌握隐式等待

下拉选择框、警告框、滚动条操作

目标

1. 掌握下拉选择框的操作方法
2. 掌握处理警告框的方法
3. 掌握调用JavaScript方法

1. 什么是下拉选择框

说明：下拉框就是HTML中<select>元素；

1.1 为什么学习下拉选择框？

需求：

案例-1 注册页面A，城市选项-暂停2秒后选择上海A，暂停2秒后选择重庆，暂停2秒后选择广州

案例-1 实现方式

1. 定位option选项
2. 定位方式不限

问题

1. 经过刚才代码演示，只能定位根据option选项的值来判断；
2. 如果有多个选项原有的定位方式处理起来比较繁琐；

1.2 Select类

说明：Select类是WebDriver为解决select标签定位诞生的，此类定位的是select标签

select类有哪些方法？

select方法：

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

1.3 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(WebElemet)
                    (WebElement):driver.find_element_by_id("selectA"))
3. 调用方法：select.select_by_index(index)
                (index：为列表索引，从0开始)
```

1.4 Select实现代码 总结

```
#导包
from selenium.webdriver.support.select import Select
...
# 1. 根据索引实现
select.select_by_index(1)
select.select_by_index(3)
select.select_by_index(2)
# 2. 根据文本值实现
select.select_by_visible_text("A上海")
select.select_by_visible_text("A重庆")
select.select_by_visible_text("A广州")
# 3. 根据value属性实现
select.select_by_value("sh")
select.select_by_value("cq")
select.select_by_value("gz")
...
```

需求

```
对案例-1 注册页面A，首先点击alerta按钮，其次输入用户名：admin
```

问题

```
1. 按钮被点击后弹出对话框，而接下来输入用户名的语句没有生效
```

---

2. 警告框处理

说明：WebDriver中对处理警告框的操作，有专用的处理方法；

提示：
    HTML中常用的对话框有三种，处理的方法都一样
        1). alert
        2). confirm
        3). prompt

2.1 警告框处理方法

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

2.2 调用方法

```
1. 获取警告框
        alert=driver.switch_to.alert
2. 调用
        alert.text
        alert.accept()
        alert.dismiss()
```

2.3 处理警告框-总结

```
...
# 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()
...
```

3. 滚动条操作

说明：WebDriver类库中并没有直接提供对滚动条进行操作方法，但是它提供了可调用JavaScript脚本的方法，所
以我们可以通过JavaScript脚本来达到操作滚动条的目的；

备注：
1). 滚动条：一种可控制程序显示范围的组件
2). JavaScript：一种流行脚本语言，可以操作HTML标签；
JavaScript学习资料：http://www.w3school.com.cn/js/js_intro.asp

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

需求

案例-1 注册页面A，打开页面2秒后，滚动条拉倒最底层

需求实现分析：

1. 设置JavaScritp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. WebDriver调用js脚本方法 driver.execute_script(js)

3.2 控制滚动条实现代码

```
...
# 最底层
js1="window.scrollTo(0,1000)"
# 最顶层
js2="window.scrollTo(0,0)"
# 执行最底层
driver.execute_script(js1)
# 执行最顶层
driver.execute_script(js2)
...
```

3.3 滚动条总结

1. WebDriver控制滚动方法
2. JavaScript控制滚动条语句

备注：js控制滚动条语句有很多种，如：js=document.documentElement.scrollTop=1000;但是推荐使用JS调用

window句柄去控制：

frame表单切换、多窗口切换

目标

1. 掌握WebDriver切换frame表单方法
2. 掌握WebDriver多窗口切换的技巧

1. frame表单

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；
(HTML语言中，frame/iframe标签为表单框架)

1.1 为什么要学习frame表单切换

需求：案例-2 注册实例.html
1. 此页面有三个注册界面，先填写最上边注册信息，其次填写注册A页面注册信息，最后填写注册B页面信息
2. 定位方式不限

问题

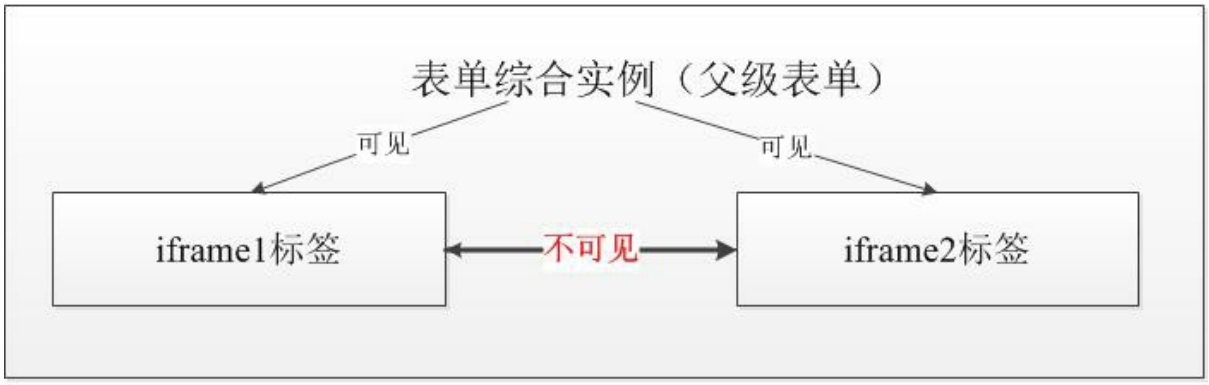
1. 当前页面内无法定位注册页面A和注册页面B

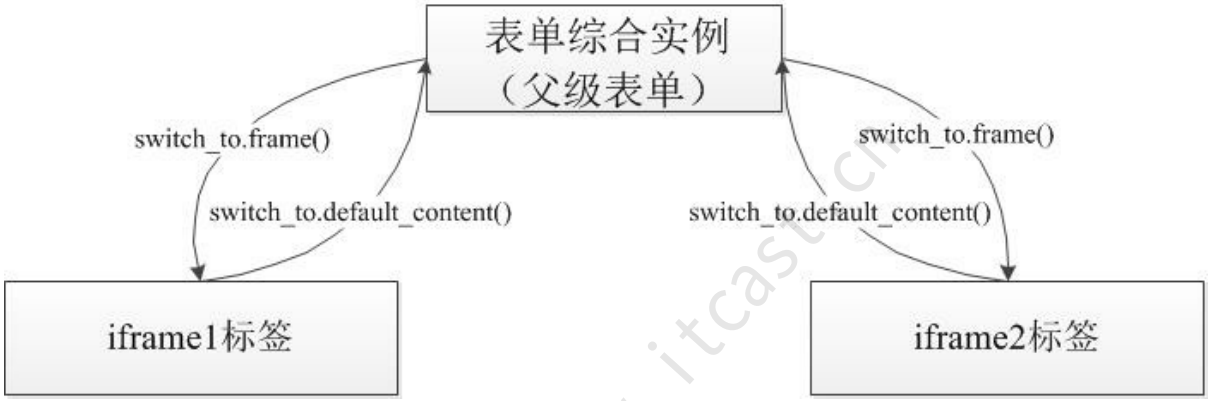
1.2 frame表单切换

说明：在WebDriver类库中封装了HTML页面中使用frame表单框架技术定位的方法

方法：
1). driver.switch_to.frame("myframe1") --> 切换表单方法
(myframe1：为frame表单的name或id)
2). driver.switch_to.default_content() --> 恢复默认页面方法
(在frame表单中操作其他页面，必须先回到默认页面，才能进一步操作)

表单切换的示意图：





1.3 案例-2解决方案

1. 完成当前页面注册信息；
2. 调用表单切换方法(switch_to.frame("myframe1"))切换到注册用户A表单中
3. 调用恢复默认页面方法(switch_to.default_content())
4. 调用表单切换方法(switch_to.frame("myframe2"))切换到注册用户B表单中

1.4 表单切换-总结

1. HTML中常用的表单框架
2. 切换表单方法
3. 为什么要恢复默认页面？
4. 恢复默认页面方法

2. 多窗口

说明：在HTML页面中，经常有a标签也就是超链接，这些链接在被执行时，有的会在新的窗口打开链接；

2.1 为什么要切换窗口？

案例-2 注册实例.html

需求：
1). 点击注册A页面链接，在打开的页面，填写A页面注册信息；

问题

1). 无法定位注册A页面

2.2 多窗口切换

说明：在WebDriver中封装了获取当前窗口句柄方法和获取所有窗口句柄的方法以及切换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-2 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. 遍历判断窗口句柄并切换到注册A页面
5. 操作注册A页面元素，注册信息

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WebDriver屏幕截图方法
2. 了解验证码处理的方式

1. 截图

说明：把当前操作页面，截图保存到指定位置

1.1 为什么要窗口截图？

说明：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

说明：在WebDriver类库中，提供了截图方法，我们只需要调用即可；

方法：

```
1). get_screenshot_as_file(imgpath)            --> 截取当前窗口
    (imgpath：图片保存路径)
```

1.3 案例-2 注册实例.html

需求：

1. 填写注册A页面注册信息，填写完毕，截图保存；

1.4 案例2-解决方案步骤

1. 打开注册实例.html
2. 切换注册A页面frame表单   --> driver.switch_to.frame(myframe1)

```
3. 输入注册信息
4. 调用截屏方法                          --> driver.get_screenshot_as_file("../Image/Image01.jpg")
```

---

2. 验证码【了解】

说明：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。

2.1 为什么要学习验证码？

说明：在Web应用中，大部分系统在用户登陆的时候都要求输入验证码，而我们在设计自动化脚本时候，就需要面
临这验证码的问题。

2.2 验证码的处理方式

说明：WebDriver类库中没有对验证码处理的方法，但是在这里可以叙说下针对验证码的几种常用处理方式；

方式：

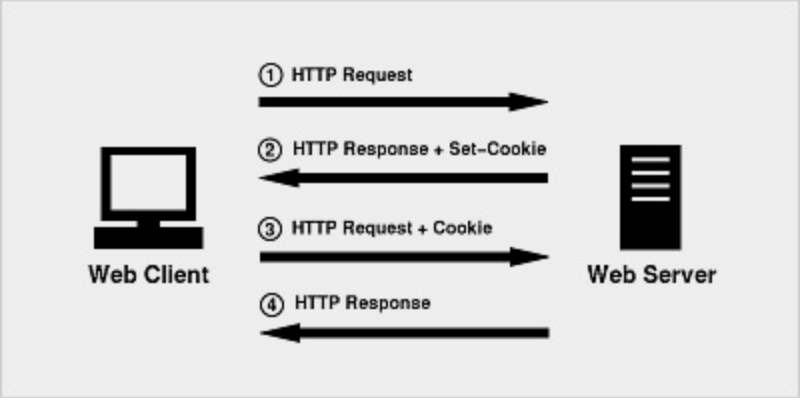
1). 去掉验证码
(测试环境下-采用)
2). 设置万能验证码
(生产环境-采用)
3). 验证码识别技术
(通过Python-tesseract来识别图片类型验证码；识别率很难达到100%)
4). 记录cookie
(通过记录cookie进行登录-推荐)

提示：

```
1. 去掉验证码、设置万能验证码：太简单都是开发来完成，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？



cookie：

1. Cookie是一小段的文本信息；格式：python中的字典（键值对组成）
2. 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格式
3. Cookie使用：当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器，服务器检查该Cookie，以此来辨认用户状态。

3.2 为什么记录cookie？

说明：

1. 用户第一次登陆时，勾选下次直接登陆或者记住密码，就是采用记录cookie实现的
2. cookie内记录用户名和密码(加密)信息，只要请求时服务器收到cookie，就识别成功，默认为已登陆。

3.3 记录cookie

说明：

1. WebDriver中对cookie操作提供相应的方法

方法：

1. get_cookie(name) --> 获取指定cookie
   (name:为健名)
2. get_cookies() --> 获取本网站所有本地cookies
3. add_cookie(str) --> 添加cookie
   (str：为python中的字典格式)

案例-3 访问百度

需求：

1. 登陆百度，获取cookie
2. 使用获取的cookie，在WebDriver中，添加Cookie，达到登陆目的

案例**3-**实现步骤分析

1. 登陆baidu，登陆的时候抓取 (BAIDUID,BDUSS)
2. 使用add_cookie()方法，添加 (BAIDUID,BDUSS)键和值
3. 调用刷新方法 driver.refresh()

3.4 代码示例

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

---

4. 窗口截屏、验证码总结

1. 截屏方法
2. 验证码常用的处理方式
3. Cookie的作用