

automation

**深 圳 市 门 道 信 息 咨 询 有 限 公 司**

Shenzhen MT Information Consulting Co . , LTD

**版权所有.侵权必究**

目录

[1. 前言 3](#_Toc27839099)

[2. 概述 3](#_Toc27839100)

[2.1 基本概念 3](#_Toc27839103)

[2.2 误区 3](#_Toc27839104)

[2.3 使用场景 3](#_Toc27839105)

[2.4 高阶理解 3](#_Toc27839106)

[2.5 工具 4](#_Toc27839107)

[3. selenium简介 4](#_Toc27839108)

[3.1 简介 4](#_Toc27839110)

[3.2 selenium和webdriver的那些事 4](#_Toc27839111)

[3.3 selenium2.x&3.x 5](#_Toc27839112)

[3.4 优势 5](#_Toc27839113)

[4. 第一个selenium 5](#_Toc27839114)

[5. 元素定位 5](#_Toc27839115)

[5.1 id定位 6](#_Toc27839120)

[5.2 其他定位 6](#_Toc27839121)

[6. 控制浏览器 7](#_Toc27839122)

[7. 鼠标操作 &键盘操作 8](#_Toc27839123)

[8. 下拉框 8](#_Toc27839124)

[9. 元素等待 8](#_Toc27839125)

[10. 内帧 10](#_Toc27839126)

[11. 警告框 10](#_Toc27839127)

[12. 上传 10](#_Toc27839128)

[13. 下载 11](#_Toc27839129)

# 前言

必须具备起码的命令知识

# 概述



## 基本概念

* 利用测试工具实现自动测试
* 自动化测试是软件测试的一个重要组成部分，它能完成许多手工测试无法实现或难以实现的测试

## 误区

* 自动化的主要目的并不是为了找Bug，仅仅是抽取出人做起来耗时好力那个部分
* 并不是所有的功能都可以自动化（比如：硬件）
* 并不是所有的项目都可以使用自动化（比如:selenium只能使用bs项目,小项目不适合使用自动化。）

## 使用场景

* 任务测试明确、版本不会频繁变动
* 软件需求变更少
* 项目周期长，迭代频繁的项目
* **在软件版本还没有稳定的情况下，千万不要开展自动化测试**
* **主要用在回归测试和接口测试**

## 高阶理解

**自动化测试的问题:** 昂贵，不可靠，不是单靠完全的自动化测试就能完成所有测试任务

**自动化测试的收益=自动化测试的运行次数**

**自动化实施成本 = 前期开发成本+后期维护成本**

前期开发成本：人力，时间，金钱

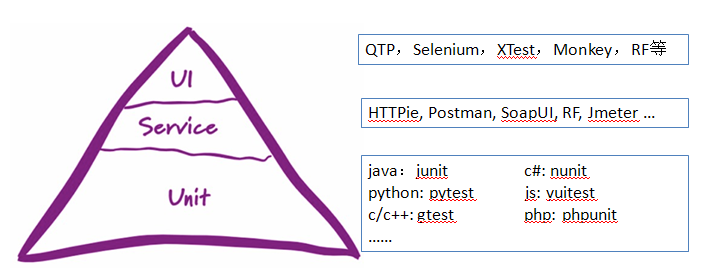
后期维护成本： 产品变更后脚本的维护成本脚本健壮性及可靠性维护成本

**自动化测试实施成本=（手工执行所有用例所花费的时间成本\*自动化用例执行次数）/自动化实施成本**

**所以，一家公司项目要不要做自动化，如何做，做哪部分都是要综合评估的？**

**思考一个问题：一个有自动化测试的公司一定比一个没有自动化（全是手工的公司）好？**

## 工具



在门道，我们重点学习service层和ui层的测试，unit层需要具体业务，需要掌握太多的开发知识和对应框架，新手不适合

# selenium简介



## 简介

|  |
| --- |
| *Selenium automates browsers*. That's it! What you do with that power is entirely up to you. Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) be automated as well.  Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks. |

Selenium使浏览器自动化。就是这样！您如何使用这种功能完全取决于您。首先，它用于自动化Web应用程序以进行测试，但是当然不限于此。无聊的基于Web的管理任务也可以（并且应该！）也可以自动化。

Selenium得到了一些最大的浏览器供应商的支持，这些供应商已采取（或正在采取步骤）使Selenium成为其浏览器的本机部分。它也是无数其他浏览器自动化工具，API和框架中的核心技术。

## selenium和webdriver的那些事

最开始的selenium是1.x，和他同台的另外一个web自动化框架webdriver后来居上更优秀，于是selenium就“决定”2者整合，于是就推出了selenium2.x

## selenium2.x&3.x

Selenium2.0集成了RC和webdriver来提供web UI级自动化测试能力。selenium2.0默认支持firefox浏览器，还是比较方便的，但是selenium2.0对firefox浏览器支持最高只支持46及以下版本，

selenium 3.0有了更新的特性加入，尤其是对Edge和safari原生驱动的支持，Edge驱动由MS提供，safari原生驱动由Apple提供。

在最新的Firefox方面，开始支持Mlzilla的geckodriver驱动，来驱动Firefox的控制。selenium3.0可以支持firefox47以上版本，但是许要下载geckodriver.exe驱动，并添加到环境变量path下

## 优势

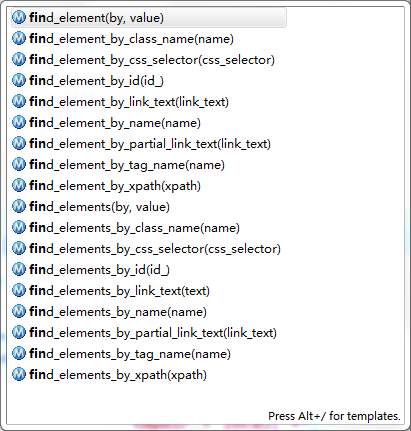
1. 通过编写模仿用户操作的 Selenium测试脚本，可以从终端用户的角度来测试应用程序。
2. 通过在不同浏览器中运行测试，更容易发现浏览器的不兼容性。
3. 支持多平台-windows、linux、MAC
4. 可以把测试用例分布到不同的测试机器上执行、相当于分发机的功能
5. 支持多语言-java、ruby、python、c#
6. 免费开源，对商业用户也没有任何限制

# 第一个selenium

|  |
| --- |
| # -\*-coding:UTF-8 -\*-  from selenium import webdriver  driver = webdriver.Chrome(r'd:\chromedriver.exe')  driver.get('httP://192.168.1.43:8082/mt')  driver.maximize\_window() |

# 元素定位

selenium有丰富的库，分别有单一定位和多元素定位



从方法看，是多元素定位

从方法看，是单一元素定位



## id定位

|  |
| --- |
| driver.find\_element\_by\_id("username") |

## 其他定位

|  |
| --- |
| # -\*-coding:UTF-8 -\*-  from selenium import webdriver  from selenium.webdriver.common.by import By  driver = webdriver.Chrome(r'd:\chromedriver.exe')  driver.get('httP://192.168.1.43:8082/mt')  driver.maximize\_window()  driver.find\_element(By.ID, "value")  driver.find\_element\_by\_id("username")  driver.find\_element\_by\_class\_name("class")  driver.find\_element\_by\_css\_selector("css\_selector")  driver.find\_element\_by\_link\_text("link\_text")  driver.find\_element\_by\_name("name")  driver.find\_element\_by\_partial\_link\_text("link\_text")  driver.find\_element\_by\_tag\_name("name")  driver.find\_element\_by\_xpath("xpath") |

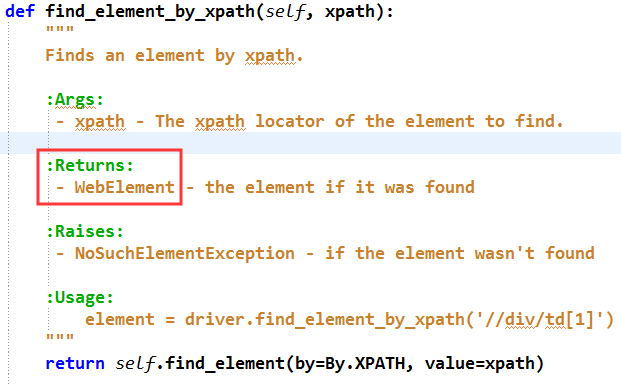
问题有3：

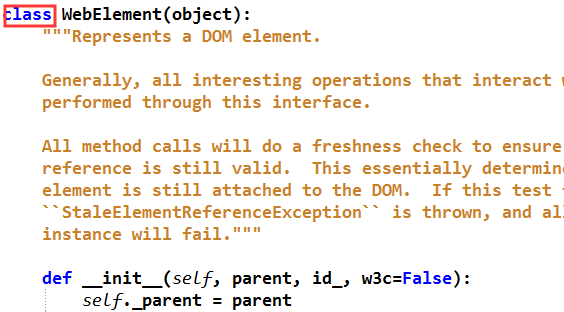
1、什么时候单一定位，什么时候多元素定位？举例说明

2、driver.find\_element(By.ID, "value") 的写法中，By.id你是怎么知道这样写的？？

3、这些定位都返回的是WebElement，这个WebElement是个类，那么代码就还可以这样写，为什么？

|  |
| --- |
| driver.find\_element\_by\_id("username").send\_keys("caichang") |





# 控制浏览器

|  |
| --- |
| driver.minimize\_window()  driver.maximize\_window()  driver.back()  driver.forward()  driver.refresh() |

# 鼠标操作 &键盘操作

请跟踪ActionChains和Keys的api就应该能明白怎么写

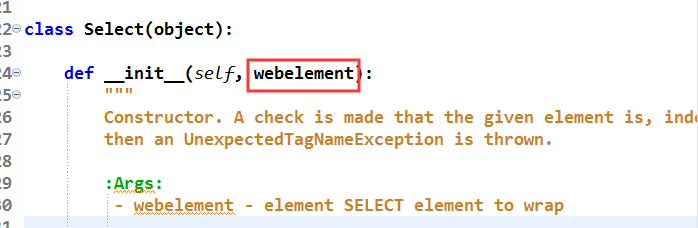
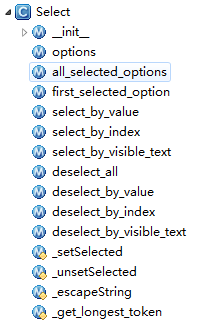
|  |
| --- |
| ActionChains(driver).double\_click(driver.find\_element\_by\_id("username")).perform()  driver.find\_element\_by\_id("username").send\_keys(Keys.ENTER) |

# 下拉框

|  |
| --- |
| Select(driver.find\_element\_by\_id("id")).select\_by\_value("20") |

初始化时候，必须传一个webelement

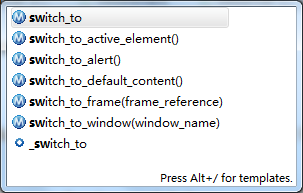
返回的是对象，然后对象.方法使用也可以



# 元素等待

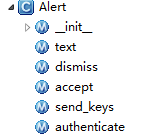
|  |
| --- |
| #强制等待  '''  优点：代码简洁，不啰嗦。  缺点：假如你设置sleep等待时间太短，元素没加载出来，程序照样报错。倘若你设置sleep等待时间设置太长，元素可能已经加载出来了，但是由于你设置过长，程序会一直等待你设置的时间过完才执行，这样会影响代码整体的运行效率。  '''  sleep(1)  #显式等待  '''  优点：  1.代码简洁。  2.在代码前部分加implicitly\_wait() ，整个的程序运行过程中都会有效，都会等待元素加载完成，不需要像sleep一样每次要设置一遍。  3.在设置的时间内没有加载到整个页面，则会报NosuchElementError。如果元素在第20s被加载出来，自动执行下面的脚本，不会一直等待20s。  缺点：非要加载到整个页面才执行代码，这样影响代码的执行效率。一般情况下，我们想要的结果是只需加载到了我要定位的元素就执行代码，不需要等待整个页面的完全加载出来再执行代码。  '''  driver.implicitly\_wait(2)  #隐式等待  '''  优点：代码执行效率快。无需等待整个页面加载完成，只需加载到你要定位的元素就可以执行代码。是最智能的设置元素等待的方式。  缺点：  1.要导入  from selenium.webdriver.support.expected\_conditions import visibility\_of\_element\_located  from selenium.webdriver.support.ui import WebDriverWait  from selenium.webdriver.common.by import By  必须要导入以上3个包，导包路径相当的复杂，啰嗦而且麻烦  2.写等待时间的代码也是复杂。步骤稍微有点多。  element=WebDriverWait(dr,10).until(EC.presence\_of\_element\_located((By.ID,‘kw’)))  element.send\_keys(‘123’)  '''  WebDriverWait(driver,15).until(EC.visibility\_of\_element\_located((By.XPATH,'//span[@id="username"]'))) |

# 内帧



|  |
| --- |
| driver.switch\_to.default\_content()  driver.switch\_to\_default\_content() |

# 警告框



|  |
| --- |
| driver.switch\_to\_alert().accept() |

# 上传

|  |
| --- |
| #input型  #<input id="txt\_userfile" class="file" type="file" value="" name="txt\_userfile">  driver.find\_element\_by\_name('txt\_userfile').send\_keys(r"E:\test.jpg")  #非input型  #pip install SendKeys  driver.find\_element\_by\_id("id").click()  #输入文件地址，绝对路径  SendKeys.SendKeys(r"x")  #输入回车键  SendKeys.SendKeys("{ENTER}")  sleep(2)  #判断文件是否上传成功,在文件路径展示的地方，使用firebug邮件查看位置  new=driver.find\_elements\_by\_xpath("x")  if len(new)==1:  print "上传成功"  else:  print "上传失败" |

# 下载

|  |
| --- |
| profile=webdriver.FirefoxProfile().set\_preference('browser.download.dir','d:\\iDownload')  webdriver.Firefox(firefox\_profile=profile)  webdriver.Chrome(browser\_profile=profile)  url = "https://github.com/mozilla/geckodriver/releases"  driver.get(url)  #选择下载ZIP类型文件，使用application/zip指代此类型文件  driver.find\_element\_by\_xpath("//strong[. = 'geckodriver-v0.16.1-win64.zip']").click() |

chrome\_options = Options()  
prefs = {  
    "download.default\_directory": "E:\\",   
    "download.prompt\_for\_download": False,   
}  
chrome\_options.add\_experimental\_option("prefs", prefs)  
  
driver = webdriver.Chrome(options=chrome\_options)