

Selenium爬虫

第01节 Selenium

1. Selenium概述

[Selenium](#)是一个Web的自动化测试工具，最初是为网站自动化测试而开发的，Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器。

因为Selenium可以控制浏览器发送请求，并获取网页数据，因此可以应用于爬虫领域。

Selenium 可以根据我们的指令，让浏览器自动加载页面，获取需要的数据，甚至页面截屏，或者判断网站上某些动作是否发生。

Selenium 自己不带浏览器，不支持浏览器的功能，它需要与第三方浏览器结合在一起才能使用。

官方文档: <http://selenium-python.readthedocs.io/index.html>

2. 浏览器驱动

浏览器驱动用于使用selenium操控本地浏览器执行自动化操作。

驱动网站: <https://npmmirror.com/>

课程中使用Chrome浏览器作为演示工具，因此下载ChromeDriver

- 当前 [npmmirror.com](#) 运行版本是: cnpmcore@1.0.0-rc.1
- 本系统运行在 [Node.js@v16.13.2](#) 上
- 开源镜像: <https://npmmirror.com/mirrors/>
- Node.js 镜像: <https://npmmirror.com/mirrors/node/>
- alinode 镜像: <https://npmmirror.com/mirrors/alinode/>
- **ChromeDriver 镜像: <https://npmmirror.com/mirrors/chromedriver/>**
- OperaDriver 镜像: <https://npmmirror.com/mirrors/operadriver/>
- Selenium 镜像: <https://npmmirror.com/mirrors/selenium/>
- electron 镜像: <https://npmmirror.com/mirrors/electron/>







注意需要根据本地电脑Chrome的版本选择对应的驱动包，否则无法操控浏览器

 Google Chrome






 Chrome 已是最新版本
版本 101.0.4951.67 (正式版本) (64 位)

获取有关 Chrome 的帮助 

报告问题 

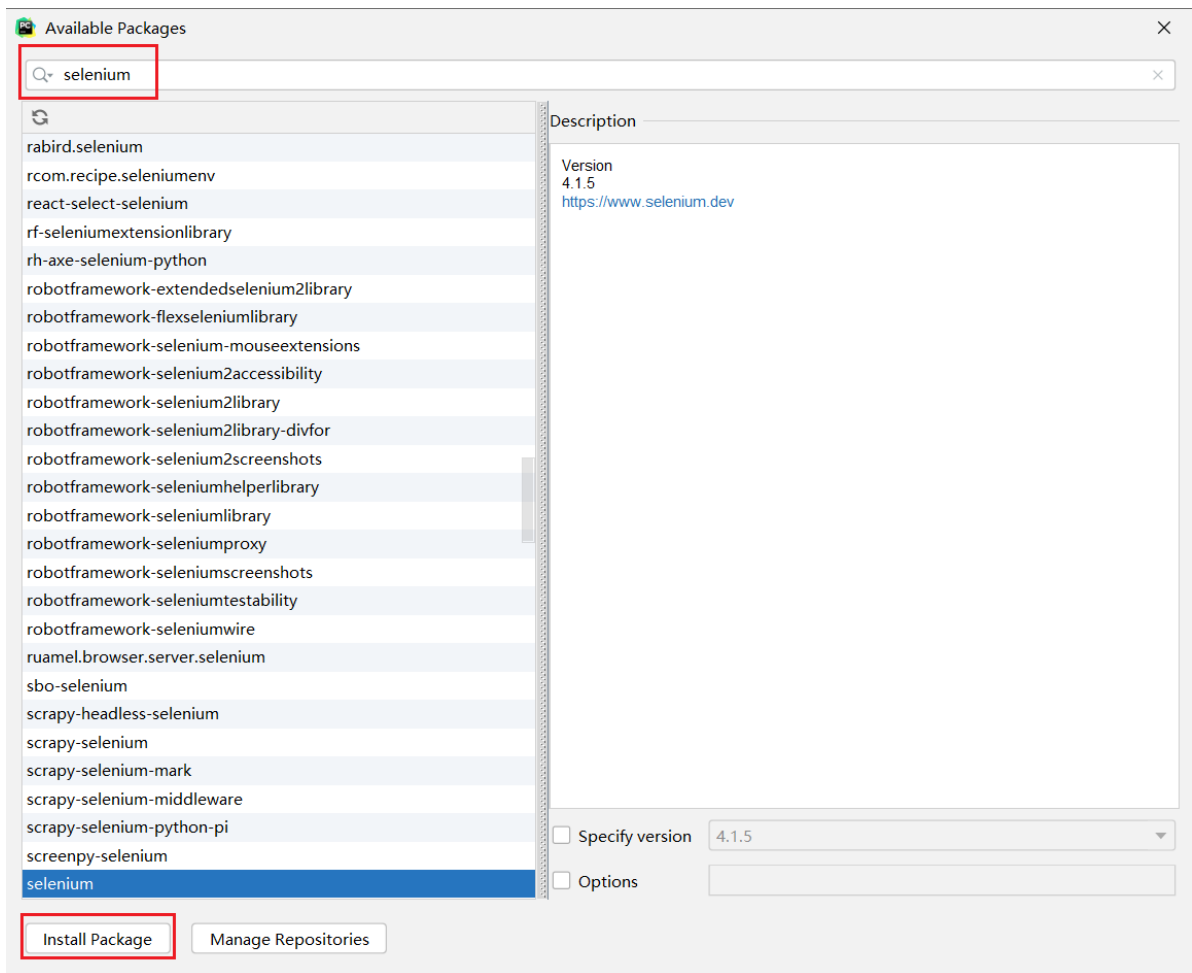
	Parent Directory	
	100.0.4896.20/	2022-05-24T02:00:00Z
	100.0.4896.60/	2022-05-24T02:00:00Z
	101.0.4951.15/	2022-05-24T02:00:00Z
	101.0.4951.41/	2022-05-24T02:00:00Z
	102.0.5005.27/	2022-05-24T02:00:00Z

再根据操作系统选择不同的驱动包即可，驱动包不需要安装，只需要解压到项目目录，后续会在代码中调用。

	Name	Last modified	Size
	Parent Directory		-
	chromedriver_linux64.zip	2022-04-01T07:53:29.832Z	6.28MB
	chromedriver_mac64.zip	2022-04-01T07:53:32.347Z	7.96MB
	chromedriver_mac64_m1.zip	2022-04-01T07:53:34.915Z	7.19MB
	chromedriver_win32.zip	2022-04-01T07:53:37.327Z	6.05MB
	notes.txt	2022-04-01T07:53:42.624Z	255

3. 基本使用

安装selenium库

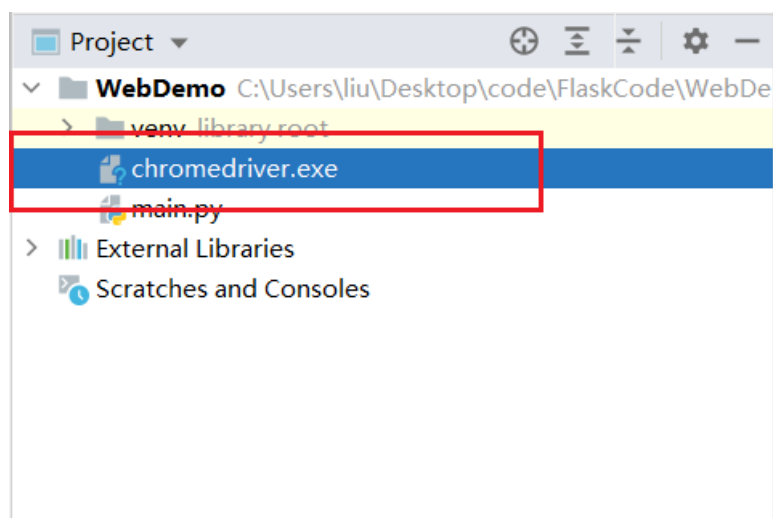


或使用命令

```
pip install selenium
```

控制浏览器自动网页

将驱动包导入到项目



调用Chrome驱动，控制浏览器打开网页

```
# 导入 webdriver
from selenium import webdriver
from selenium.webdriver.common.by import By
# 调用键盘按键操作时需要引入的Keys包
from selenium.webdriver.common.keys import Keys

# 调用环境变量指定的PhantomJS浏览器创建浏览器对象
driver = webdriver.Chrome("./chromedriver.exe")
# get方法会一直等到页面被完全加载，然后才会继续程序，通常测试会在这里选择 time.sleep(2)
driver.get("http://www.baidu.com/")
```

控制页面元素

```
# id="kw"是百度搜索输入框，输入字符串"长城"
driver.find_element(By.CSS_SELECTOR, "#kw").send_keys("长城")
# id="su"是百度搜索按钮，click() 是模拟点击
driver.find_element(By.CSS_SELECTOR, "#su").click()

# 关闭浏览器
driver.quit()
```

第03节 常用操作

1. 元素定位

获取单个元素

```
driver.find_element(By.ID, "inputOriginal")
driver.find_element(By.CSS_SELECTOR, "#inputOriginal")
driver.find_element(By.TAG_NAME, "div")
driver.find_element(By.NAME, "username")
driver.find_element(By.LINK_TEXT, "下一页")
```

如果找不到相应的元素会报错

```
selenium.common.exceptions.NoSuchElementException: Message: no such element:
Unable to locate element: xx
```

获取多个元素

```
driver.find_elements(By.ID, "inputOriginal")
driver.find_elements(By.CSS_SELECTOR, "#inputOriginal")
driver.find_elements(By.TAG_NAME, "div")
driver.find_elements(By.NAME, "username")
driver.find_elements(By.LINK_TEXT, "下一页")
```

访问有道翻译网站，输入单词，并获取翻译后的内容

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
```

```

driver = webdriver.Chrome("./chromedriver.exe")
# 加载有道翻译页面
driver.get("https://fanyi.youdao.com/")
# 获取输入框
input = driver.find_element(By.ID, "inputOriginal")
# 输入内容
input.send_keys("hello")
# 获取翻译按钮
tbtn = driver.find_element(By.ID, "transMachine")
# 发现页面被遮挡, 此时无法点击
# tbtn.click()

# 先获取遮挡的广告条, 点击关闭按钮
close_btn = driver.find_element(By.CSS_SELECTOR, ".guide-con .guide-close")
close_btn.click()

# 点击翻译
tbtn.click()

# 获取翻译后的内容
transTarget = driver.find_element(By.ID, "transTarget")
print(transTarget.text)

```

2. 内容获取

1. size	返回元素大小
2. text	获取元素的文本 <div>hello</div>
3. title	获取页面title
4. current_url	获取当前页面URL
5. get_attribute()	获取属性值 百度
6. is_displayed()	判断元素是否可见
7. is_enabled()	判断元素是否可用

```

driver = webdriver.Chrome("./chromedriver.exe")
# 加载有道翻译页面
driver.get("https://fanyi.youdao.com/")

print(driver.title)
print(driver.current_url)

# 获取输入框
transMachine = driver.find_element(By.ID, "transMachine")

print(transMachine.size)
print(transMachine.text)
print(transMachine.get_attribute("href"))
print(transMachine.is_displayed())
print(transMachine.is_enabled())

```

3. 窗口操作

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

4. 元素等待

翻页获取每页元素

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome("./chromedriver.exe")
# 加载当当网
driver.get("https://www.dangdang.com/")

# 获取输入框
key = driver.find_element(By.ID, "key_s")
key.send_keys("科幻")
# 获取搜索框, 点击搜索
search = driver.find_element(By.CSS_SELECTOR, ".search .button")
search.click()
# 获取商品标题及价格
for i in range(5):
    shoplist = driver.find_elements(By.CSS_SELECTOR, ".shoplist li")
    for li in shoplist:
        print(li.find_element(By.CSS_SELECTOR, "a").get_attribute("title"))
# 获取下一页按钮
next = driver.find_element(By.LINK_TEXT, "下一页")
next.click()
```

现在的网页越来越多采用了 Ajax 技术, 这样程序便不能确定何时某个元素完全加载出来了。如果实际页面等待时间过长导致某个dom元素还没出来, 但是你的代码直接使用了这个WebElement, 那么就会抛出NullPointerException的异常。

为了避免这种元素定位困难而且会提高产生 ElementNotVisibleException 的概率。所以 Selenium 提供了两种等待方式, 一种是隐式等待, 一种是显式等待。

隐式等待是等待特定的时间, 显式等待是指定某一条件直到这个条件成立时继续执行。

显式等待

显式等待指定某个条件, 然后设置最长等待时间。如果在这个时间还没有找到元素, 那么便会抛出异常了。

显示等待使用WebDriverWait完成

```
WebDriverWait(driver, timeout, poll_frequency=POLL_FREQUENCY,  
ignored_exceptions=None)
```

- driver 所创建的浏览器driver
- timeout 最长时间长度（默认单位：秒）
- poll_frequency 间隔检测时长（每）默认0.5秒
- ignored_exceptions 方法调用中忽略的异常，默认只抛出：找不到元素的异常

基础格式（WebDriverWait+until+（判断条件））

until

直到调用的方法返回值为True

```
until(method, message='')
```

method: expected_conditions库中定义的方法

message: 自定义报错信息

判断条件

判断当前页面标题是否为title title_is(title)

判断当前页面标题是否包含title title_contains(title)

判断此定位的元素是否存在，presence_of_element_located(locator)

判断页面网址中是否包含url url_contains(url)

判断此定位的元素是否可见 visibility_of_element_located(locator)

判断此元素是否可见 visibility_of(element) element: 所获得的元素

判断此定位的一组元素是否至少存在一个 presence_of_all_elements_located(locator)

判断此定位的一组元素至少有一个可见，visibility_of_any_elements_located(locator)

判断此定位的一组元素全部可见visibility_of_all_elements_located(locator)

判断此定位中是否包含text_的内容，text_to_be_present_in_element(locator, text_)

locator: 元素的定位信息

text_: 期望的文本信息

判断此定位中的value属性中是否包含text_的内容

text_to_be_present_in_element_value(locator, text_)

locator: 元素的定位信息

text_: 期望的文本信息

判断定位的元素是否为frame，并直接切换到这个frame中

frame_to_be_available_and_switch_to_it(locator)

locator: 元素的定位信息

判断定位的元素是否不可见 invisibility_of_element_located(locator)

locator: 元素的定位信息

判断此元素是否不可见 `invisibility_of_element(element)`

`element`: 所获得的元素

判断所定位的元素是否可见且可点击 `element_to_be_clickable(locator)`

`locator`: 元素的定位信息

判断此元素是否不可用 `staleness_of(element)`

`element`: 所获得的元素

判断该元素是否被选中 `element_to_be_selected(element)`

`element`: 所获得的元素

判断定位的元素是否被选中 `element_located_to_be_selected(locator)`

`locator`: 元素的定位信息

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
# 程序每0.5秒检查，是否满足：标题包含“百度一下”这个条件，检查是否满足条件的最长时间为：15秒，超过15秒仍未满足条件则抛出异常
WebDriverWait(driver, 15, 0.5).until(EC.title_contains("百度一下"))

# 程序每0.5秒检查，是否满足：某定位的元素出现，检查是否满足条件的最长时间为：15秒，超过15秒仍未满足条件则抛出异常
WebDriverWait(driver, 15, 0.5).until(EC.visibility_of_element_located(By.CSS_SELECTOR, "xx"))
```

隐式等待

隐式等待比较简单，就是设置全局元素查找的超时时间

```
implicitly_wait(time_to_wait)
```

设置的时间单位为秒，例如`implicitly_wait(30)`，意思是超过30秒没有定位到一个元素，程序就会报错抛出异常，期间会一直轮询查找定位元素。

第04节 页面操作(动作链)

1. 鼠标及键盘操作

鼠标操作

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

```
driver.get("https://www.baidu.com/")
more = driver.find_element(By.LINK_TEXT, "更多")
#将鼠标移动到更多按钮
ActionChains(driver).move_to_element(more).perform()
```


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)

```
from selenium.webdriver.common.keys import Keys
```

```
driver.get("https://www.baidu.com/")

element = driver.find_element(By.ID, "kw")

# 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
# element.send_keys(Keys.CONTROL, 'v')
```

2. 滚动条

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

```
# 1. 设置JavaScript脚本控制滚动条，(0:左边距；1000: 上边距；单位像素)
js="window.scrollTo(0,1000)"

# 2. WebDriver调用js脚本方法
driver.execute_script(js)
```

3. 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。

```
# 截取当前窗口
driver.get_screenshot_as_file("./demo.png")
```

第05节 存储数据

1. 将数据写入CSV文件

```
# 读写CSV文件
import csv
# 以写入方式打开文件，如果文件不存在则自动创建
f = open("d:/test.csv", 'w')
# 获取csv writer，用于写入csv格式数据
writer = csv.writer(f)
# 写入数据
writer.writerow(["张三", "男", "1.7"])
# 关闭文件
f.close()
```

2. 将数据写入至MySQL

安装模块

```
pip install pymysql
```

```
import pymysql

# 创建连接
conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='',
db='tkql', charset='utf8')
# 创建游标
cursor = conn.cursor()

# 执行SQL，并返回回收影响行数
effect_row = cursor.execute("select * from tb7")

# 执行SQL，并返回受影响行数
#effect_row = cursor.execute("update tb7 set pass = '123' where nid = %s",
(11,))

# 执行SQL，并返回受影响行数,执行多次
#effect_row = cursor.executemany("insert into
tb7(user,pass,licnese)values(%s,%s,%s)", [("u1","u1pass","11111"),
("u2","u2pass","22222")])

# 提交，不然无法保存新建或者修改的数据
conn.commit()

# 关闭游标
cursor.close()
# 关闭连接
conn.close()
```

参考: <https://www.jb51.net/article/92516.htm>

注意事项

因为爬虫技术涉及面广，且技术复杂，推荐直接在网络上寻找已有数据进行处理分析，因此爬虫阶段可以只做参考。

数据网站参考：

- 和鲸社区：<https://www.heywhale.com/home/dataset>
- Kaggle：<https://www.kaggle.com/datasets>
- GitHub：<https://github.com/awesomedata/awesome-public-datasets>