

## 目录

|             |       |
|-------------|-------|
| 前言          | 1.1   |
| puppeteer概述 | 1.2   |
| 初始化环境       | 1.3   |
| 基本操作        | 1.4   |
| 查找定位元素      | 1.4.1 |
| 输入文字        | 1.4.2 |
| 模拟按键        | 1.4.3 |
| 等待元素出现      | 1.4.4 |
| 获取元素属性      | 1.4.5 |
| 举例          | 1.5   |
| 百度搜索自动化     | 1.5.1 |
| 常见问题        | 1.6   |
| 附录          | 1.7   |
| 相关资料        | 1.7.1 |
| 参考资料        | 1.7.2 |

## Web前端自动化利器：puppeteer

- 最新版本： v1.0
- 更新时间： 20210628

### 简介

介绍Web前端自动化利器puppeteer。如何用puppeteer实现模拟用户自动操作Web页面。

### 源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

#### Gitbook源码

- [crifan/web\\_automation\\_tool\\_puppeteer](#): Web前端自动化利器：puppeteer

#### 如何使用此Gitbook源码去生成发布为电子书

详见：[crifan/gitbook\\_template](#): demo how to use crifan gitbook template and demo

#### 在线浏览

- Web前端自动化利器：[puppeteer book.crifan.com](#)
- Web前端自动化利器：[puppeteer crifan.github.io](#)

#### 离线下载阅读

- Web前端自动化利器：[puppeteer PDF](#)
- Web前端自动化利器：[puppeteer ePub](#)
- Web前端自动化利器：[puppeteer Mobi](#)

### 版权说明

此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您的版权，请通过邮箱联系我 [admin](#) 艾特 [crifan.com](#)，我会尽快删除。谢谢合作。

## 鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 crifan 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

## 更多其他电子书

本人 crifan 还写了其他 100+ 本电子书教程，感兴趣可移步至：

[crifan/crifan\\_ebook\\_readme](#): Crifan的电子书的使用说明

crifan.com，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新： 2021-07-17 17:30:06

## puppeteer概述

一句话描述：

用于替代 PhantomJS 和 Selenium 的前端自动化测试工具。特点是简单易用、速度快、支持屏幕截图和生成pdf文件。

## 背景

前端就有了对 headless 浏览器的需求，最多的应用场景有两个

1. UI自动化测试：摆脱手工浏览点击页面确认功能模式
2. 爬虫：解决页面内容异步加载等问题

前端经常使用的莫过于

- PhantomJS
  - <http://phantomjs.org/>
- selenium + webdriver
  - <http://seleniumhq.github.io/selenium/docs/api/javascript/>

但两个库有一个共性：

- 难用
  - 环境安装复杂
  - API 调用不友好

2017 年 Chrome 团队连续放了两个大招

- Headless Chromium
  - <https://chromium.googlesource.com/chromium/src/+lkgr/headless/README.md>
- NodeJS API Puppeteer
  - <https://github.com/GoogleChrome/puppeteer>

-> 直接让 PhantomJS 和 Selenium IDE for Firefox 作者悬宣布没必要继续维护其产品

我们手工可以在浏览器上做的事情 Puppeteer 都能胜任

1. 生成网页截图或者 PDF
2. 爬取大量异步渲染内容的网页，基本就是人肉爬虫
3. 模拟键盘输入、表单自动提交、UI 自动化测试

## 概述

- puppeteer

- 概述：微软开发的基于 Javascript 的web自动化的库
- 官网
  - github
    - puppeteer/puppeteer: Headless Chrome Node.js API
    - <https://github.com/puppeteer/puppeteer>
  - google
    - Puppeteer | Tools for Web Developers | Google Developers
    - <https://developers.google.com/web/tools/puppeteer>
- 优势
  - 可以用TypeScript编写测试
  - Devs还可以在运行测试时连接Chrome DevTools

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:22:29

## 初始化环境

此处介绍如何（在Mac中）初始化 `pyppeteer` 开发环境。

## 下载和安装 `pyppeteer`

- Mac中安装 `pyppeteer`

```
pip install pyppeteer
```

- 用 `pyppeteer-install` 去下载浏览器内核

```
pyppeteer-install
```

- 可以看到下载了chrome
  - 此处位置是： `/Users/crifan/Library/Application Support/pyppeteer/local-chromium/588429`

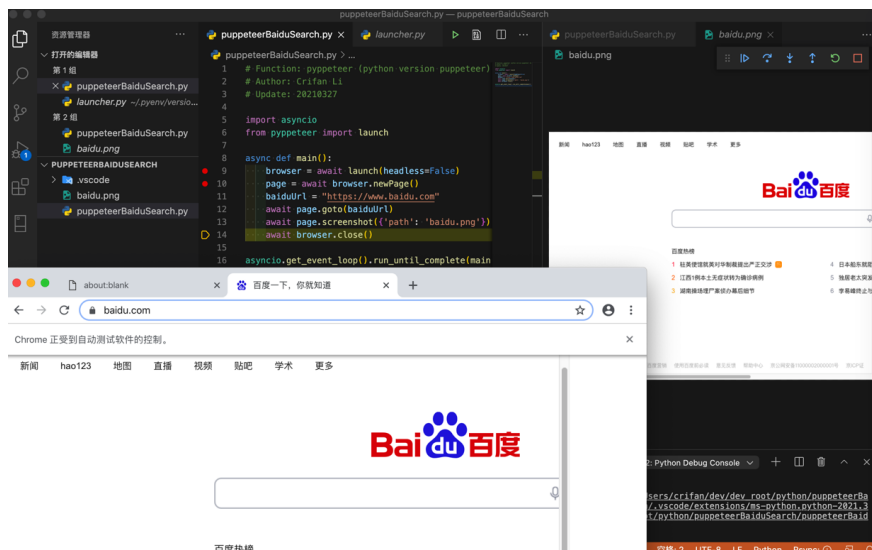
## 测试代码

```
import asyncio
from pyppeteer import launch

async def main():
    browser = await launch(headless=False)
    page = await browser.newPage()
    baiduUrl = "https://www.baidu.com"
    await page.goto(baiduUrl)
    await page.screenshot({'path': 'baidu.png'})
    await browser.close()

asyncio.get_event_loop().run_until_complete(main())
```

即可，启动Chromium浏览器，并打开百度，和本地截图：



## 常见问题

### pyppeteer代码正常运行，但没有启动Chrome浏览器

现象：pyppeteer 代码

```
browser = await launch()
```

是正常运行了，但是没看到Chrome浏览器启动

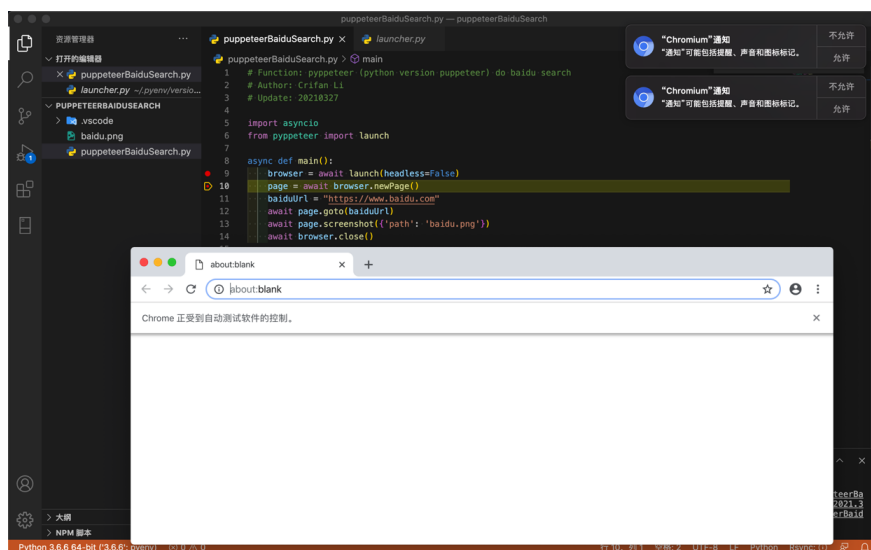
原因：puppeteer (pyppeteer) 默认是启动 无头模式，所以内部其实启动了，只是没有界面显示，即看不到Chrome浏览器启动而已。

解决办法：加上参数，取消无头模式

代码：

```
browser = await launch(headless=False)
```

即可看到Chrome浏览器



## 参数传递方式也可以用dict字典方式

也可以写成dict字典的方式传参

```
browser = await launch({'headless': False})
```

效果是一样的

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:27:13

## 基本操作

此处介绍pyppeteer的常见的基本操作。

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:23:28

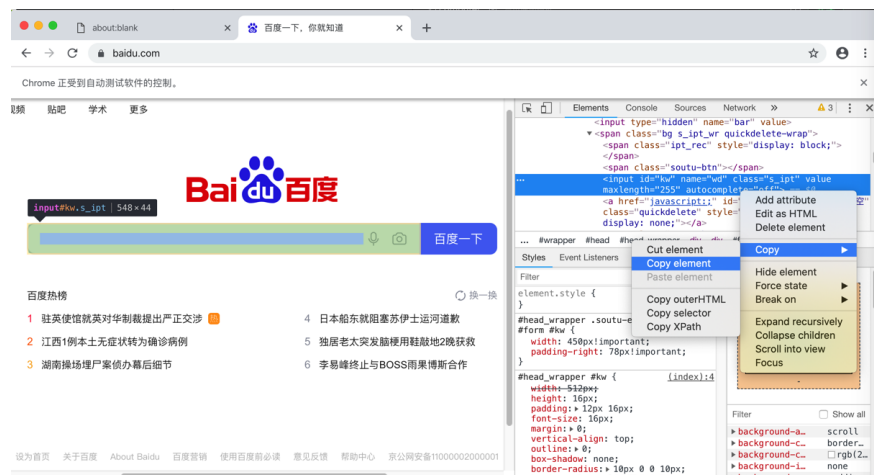
## 查找定位元素

查找元素相关函数：

- `pyppeteer`
  - `Page.querySelector()`
    - 别名: `Page.J()`
  - `Page.querySelectorAll()`
    - 别名: `Page.JJ()`
  - `Page.xpath()`
    - 别名: `Page.Jx()`

## 单个查找 `xpath`

对于页面：



对应html：

```
<input id="kw" name="wd" class="s_ipt" value="" maxlength="
```

代码：

```
SearchButtonXPath = "//input[@id='kw']"
searchButtonElem = page.xpath(SearchButtonXPath)
print("searchButtonElem=%s" % searchButtonElem)
```

输出：

```
searchButtonElem=<coroutine object Page.xpath at 0x10f15cd1
```

调试效果：

The screenshot shows a Windows IDE with a Python script named `puppeteerBaiduSearch.py`. The script is designed to automate a search on Baidu using Puppeteer. It defines a search button path, a URL, a browser, a page, and a search button element. The script then uses `asyncio` to click the button and print the search results.

```

1 # Function: puppeteer (python version puppeteer) do baidu search
2 # Author: Crifan Li
3 # Update: 20210327
4
5 import asyncio
6 from
7
8 > special variables
9 > function variables
10 cr_email: None
11 p
12 > cr.code: <code object xpath at 0x10f9c91>
13 > cr.frame: <frame object at 0x10f1c748>
14 > cr.running: False
15
16 5 按住 Option 键可切换到编辑器最右侧
17 searchButtonElem = page.xpath(SearchButtonXPath)
18 print("searchButtonElem-%s" % searchButtonElem)
19 await browser.close()
20
21 asyncio.get_event_loop().run_until_complete(main())
22

```

The IDE interface includes a file explorer on the left, a console at the bottom, and a debug console at the bottom right. The console shows the output of the script, which is the search button element's path. The debug console shows the current state of the script, including the search button element's path.

对于html

想要查找=定位 (所有的) 元素 a

即可找到元素：

```

puppeteerBaiduSearch.py — puppeteerBaiduSearch
运行和调试 Python: 当前文件
puppeteerBaiduSearch.py x page.py
变量 puppeteerBaiduSearch.py x main
Local
  searchInputSelector: 'input[id="kw"]'
  allElemList: [ppypeteer_element_h_...107f485b..., ...]
  browserUrl: 'https://www.baidu.com'
  browser: <ppypeteer.browser.Browser object>
  h3ASelector: 'h3[class="ti"] a'
  page: <ppypeteer.page.Page object at 0x107...>
  searchStr: 'orifan'
Global
  监视
  调用堆栈
  主模块 puppeteerBaiduSearch.py 59:1
  <module> puppeteerBaiduSearch.py 63:3
  加载所有模块
  57 h3Ase1 按住 Option 键可切换到英语语言系统
  58 aElemList = await page.querySelectorAll(h3ASelector)
  59 print("aElemList=" + aElemList)
  60
  61 await browser.close()
```

查找定位元素

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:23:03

## 输入文字

对于页面元素：

```
<input id="kw" name="wd" class="s_ip" value="" maxlength="
```

想要输入文字

## 不支持定位元素再输入

此处，无需，也没法实现：先定位元素，再输入

而是，只支持：直接定位并输入

## 多种实现方式

### 方式1：先 Selector 定位，再 type 输入

定位此处只适合用：Selector，具体写法是：

```
"input[id='kw']"
```

然后输入是用 type 函数

完整代码：

```
searchStr = "crifan"
SearchInputSelector = "input[id='kw']"

await page.type(SearchInputSelector, searchStr, delay=20)
```

### 方式2：先 focus，再 keyboard 的 type

先（通过selector）focus，再（用keyboard）type

完整代码：

```
searchStr = "crifan"
SearchInputSelector = "input[id='kw']"

await page.focus(SearchInputSelector)
await page.keyboard.type(searchStr)
```

## 方式3：先 focus ，再 click ，最后用 keyboard 的 type

先 selector ，再click（类似于focus），最后用keyboard输入type

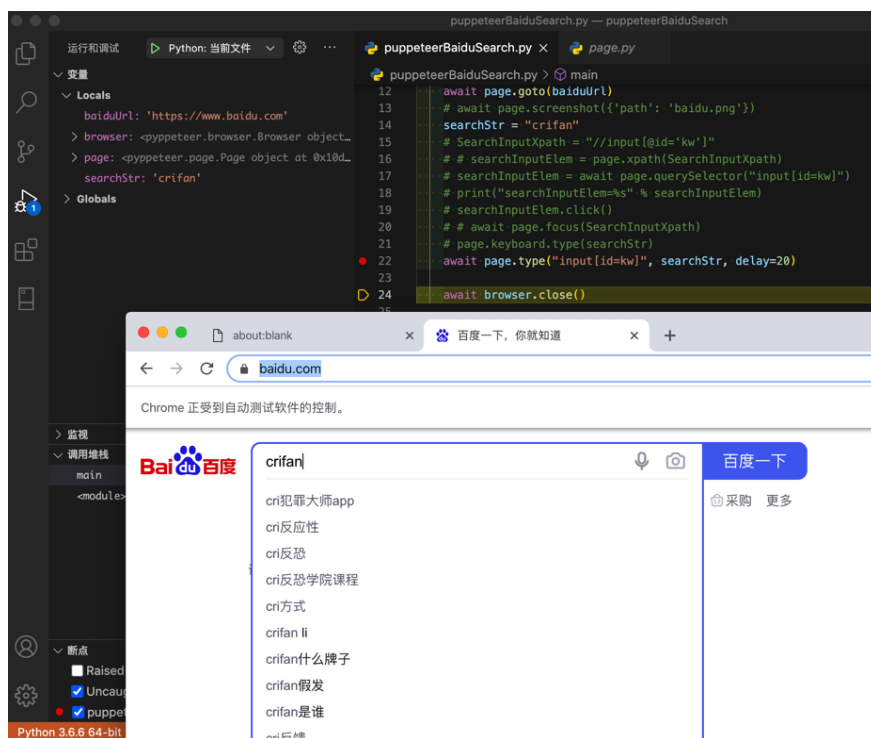
完整代码：

```
searchStr = "crifan"
SearchInputSelector = "input[id='kw']"

searchInputElement = await page.querySelector(SearchInputSelector)
await searchInputElement.click()
await page.keyboard.type(searchStr)
```

## 效果

百度首页的输入框中，输入了字符串 crifan 后的效果：



## 注意事项

### 加记得加 await

注意：一定要加await，否则：代码运行无效果。

且还会报警告：

```
/Users/crifan/dev/dev_root/python/puppeteerBaiduSearch/pup  
searchInputElem.click()
```

## 相关文档

- `coroutine type(selector: str, text: str, options: dict = None, **kwargs)`  
→ `None[source]` API Reference — Pyppeteer 0.0.25 documentation
  - [https://miyakogi.github.io/pyppeteer/reference.html#pyppeteer.p  
age.Page.type](https://miyakogi.github.io/pyppeteer/reference.html#pyppeteer.page.Page.type)

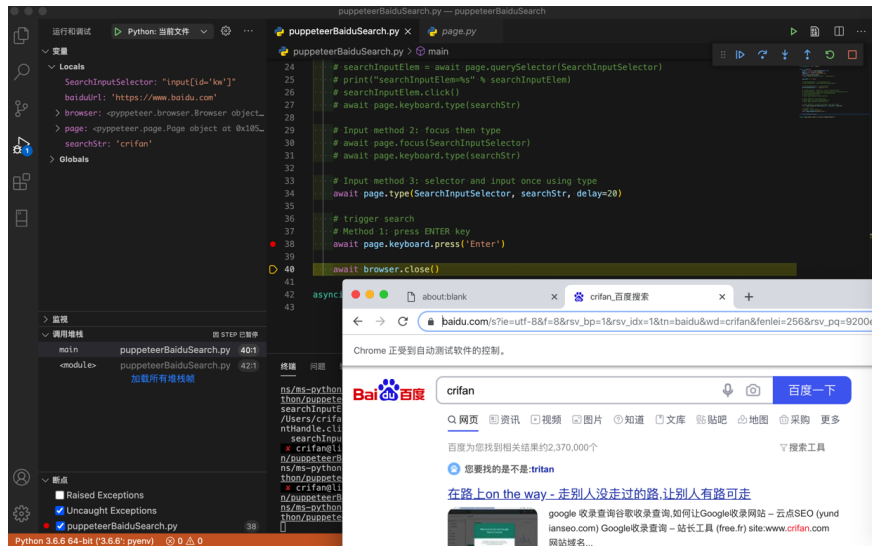
crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:25:47

## 模拟按键

模拟 回车键：

```
await page.keyboard.press('Enter')
```

此处效果：可以触发百度搜索，显示搜索结果：



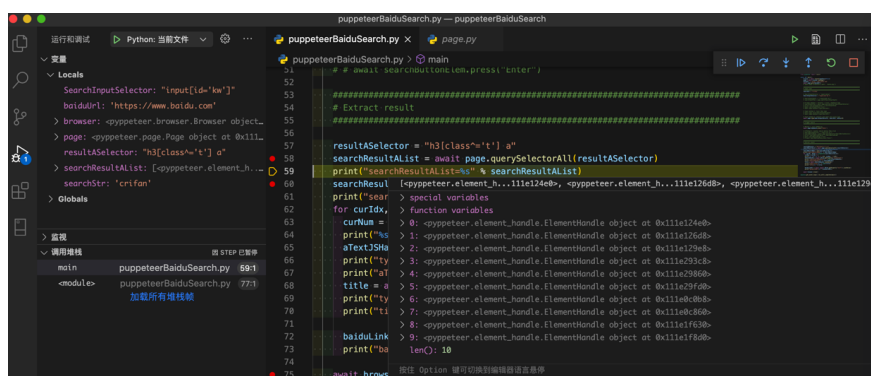
crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新： 2021-07-17 17:27:07

## 等待元素出现

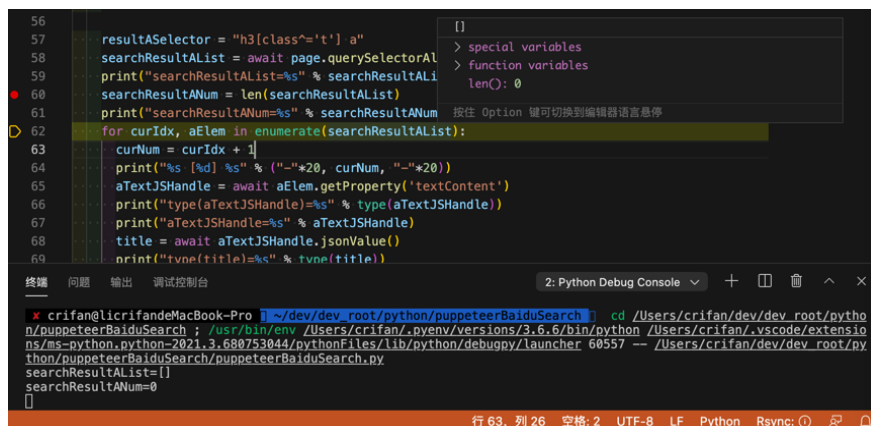
现象：对于代码：

```
resultASelector = "h3[class^='t'] a"
searchResultAList = await page.querySelectorAll(resultASelector)
```

调试时可以正常运行，可以找到元素：



直接运行时，却找不到元素了：



原因：页面重新加载了，但是内容还没显示出来。所以找不到元素。

解决办法：等待页面加载完毕。再去查找元素，就可以找到了。

## 如何确保页面加载完毕？

核心逻辑：找到页面加载完毕，一定会显示（出现）的元素，去等待其出现，即可。

此处，百度搜索后，一定会出现（显示）的元素是：

```
<span class="nums_text">百度为您找到相关结果约2,370,000个</span>
```

对应等待元素出现

- （好的）方法1: `querySelector + sleep`
  - 代码

```
SearchFoundWordsSelector = 'span.nums_text'
SearchFoundWordsXpath = "//span[@class='nums_te

# Method 2: wait element showing
SingleWaitSeconds = 1
while not await page.querySelector(SearchFoundW
    print("Still not found %s, wait %s seconds" %
    await asyncio.sleep(SingleWaitSeconds)
```

- （不够好的）方法2: 直接wait等待
  - 代码

```
# # Method 1: just wait
await page.waitFor(2000) # millisecond
```

- 评价: 不够好, 不能精确判断元素是否出现

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:22:46

## 获取元素属性

获取元素属性可以用：`someElement.getProperty('propertyName')`

举例：

```
<h3 class="t"><a data-click="{
  'F': '778317EA',
  'F1': '9D73F1E4',
  'F2': '4CA6DE6B',
  'F3': '54E5243F',
  'T': '1616767238',
  'y': 'EFBCEFB'E
}" href="https://www.baidu.com/link?url=nDSbU91
```

中的 `a` 元素中的 `href` 和文本值

对于已经找到元素的列表：

```
resultASelector = "h3[class^='t'] a"
searchResultAList = await page.querySelectorAll(resultASelector)
# print("searchResultAList=%s" % searchResultAList)
searchResultANum = len(searchResultAList)
print("Found %s search result:" % searchResultANum)
```

后去获取文本值 `text` 和属性值 `href`：

```
for curIdx, aElem in enumerate(searchResultAList):
    curNum = curIdx + 1
    print("%s [%d] %s" % ("-"*20, curNum, "-"*20))
    aTextJSHandle = await aElem.getProperty('textContent')
    # print("type(aTextJSHandle)=%s" % type(aTextJSHandle))
    # type(aTextJSHandle)=<class 'pyppeteer.execution_context.JSHandle'>
    # print("aTextJSHandle=%s" % aTextJSHandle)
    # aTextJSHandle=pyppeteer.execution_context.JSHandle
    title = await aTextJSHandle.jsonValue()
    # print("type(title)=%s" % type(title))
    # type(title)=<class 'str'>
    print("title=%s" % title)

    baiduLinkUrl = await (await aElem.getProperty("href"))
    print("baiduLinkUrl=%s" % baiduLinkUrl)
```

输出：

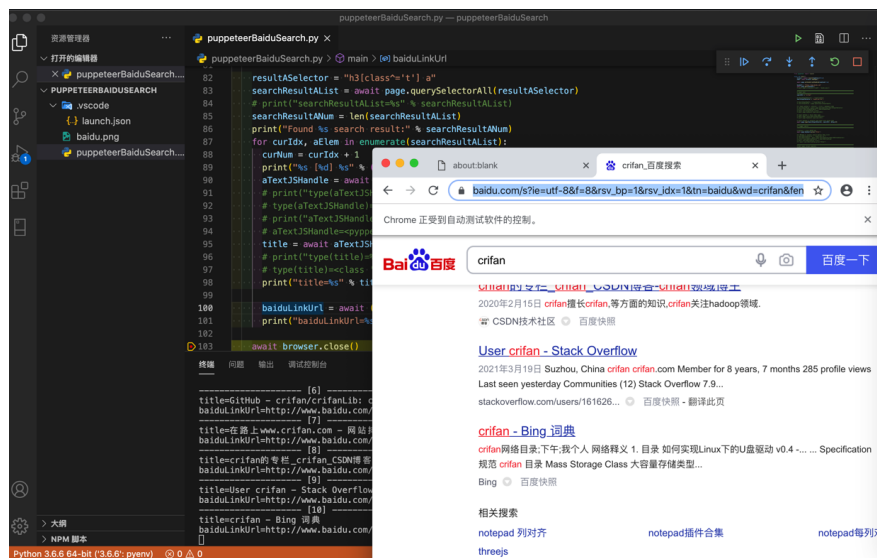
## 查找定位元素

Found 10 search result:

```
----- [1] -----
title=在路上on the way - 走别人没走过的路,让别人有路可走
baiduLinkUrl=http://www.baidu.com/link?url=eGTzEXXlMw-hnvX
----- [2] -----
title=crifan - 在路上
baiduLinkUrl=http://www.baidu.com/link?url=l6jXejlgARWj34C
...

```

效果:



crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:21:45

## 举例

下面给出具体的puppeteer的实例案例供参考。

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:28:08

## 百度搜索自动化

此处给出用 `puppeteer` 模拟百度搜索，即百度搜索自动化的完整例子。

### 代码

- 文件下载：[puppeteerDemoBaiduSearch.py](#)
- 贴出代码

```

# Function: pyppeteer (python version puppeteer) do baidu search
# Author: Crifan Li
# Update: 20210330

import asyncio
from pyppeteer import launch

async def main():
    browser = await launch(headless=False)
    page = await browser.newPage()

    await page.setJavaScriptEnabled(enabled=True)

    baiduUrl = "https://www.baidu.com"
    await page.goto(baiduUrl)
    # await page.screenshot({'path': 'baidu.png'})

    #####
    # Input text
    #####
    searchStr = "crifan"

    # SearchInputSelector = "input[id=kw]"
    SearchInputSelector = "input[id='kw']"

    # SearchInputXpath = "//input[@id='kw']"
    # searchInputElem = page.xpath(SearchInputXpath)

    # # Input method 1: selector + click + keyboard type
    # searchInputElem = await page.querySelector(SearchInputSelector)
    # print("searchInputElem=%s" % searchInputElem)
    # await searchInputElem.click()
    # await page.keyboard.type(searchStr)

    # Input method 2: focus then type
    # await page.focus(SearchInputSelector)
    # await page.keyboard.type(searchStr)

    # Input method 3: selector and input once using type
    await page.type(SearchInputSelector, searchStr, delay=100)

    #####
    # Trigger search
    #####

    # Method 1: press ENTER key
    await page.keyboard.press('Enter')

    # # Method 2: locator search button then click
    # SearchButtonSelector = "input[id='su']"

```

```

# searchButtonElem = await page.querySelector(SearchBut
# print("searchButtonElem=%s" % searchButtonElem)
# await searchButtonElem.click()
# # await searchButtonElem.press("Enter")

#####

# Wait page reload complete
#####

SearchFoundWordsSelector = 'span.nums_text'
SearchFoundWordsXpath = "//span[@class='nums_text']"

# await page.waitForSelector(SearchFoundWordsSelector)
# await page.waitFor(SearchFoundWordsSelector)
# await page.waitForXPath(SearchFoundWordsXpath)
# Note: all above exception: 发生异常: ElementHandleError
# so change to following

# # Method 1: just wait
# await page.waitFor(2000) # millisecond

# Method 2: wait element showing
SingleWaitSeconds = 1
while not await page.querySelector(SearchFoundWordsSele
    print("Still not found %s, wait %s seconds" % (Search
    await asyncio.sleep(SingleWaitSeconds)
    # pass

#####

# Extract result
#####

resultASelector = "h3[class^='t'] a"
searchResultAList = await page.querySelectorAll(resultA
# print("searchResultAList=%s" % searchResultAList)
searchResultANum = len(searchResultAList)
print("Found %s search result:" % searchResultANum)
for curIdx, aElem in enumerate(searchResultAList):
    curNum = curIdx + 1
    print("%s [%d] %s" % ("-"*20, curNum, "-"*20))
    aTextJSHandle = await aElem.getProperty('textContent')
    # print("type(aTextJSHandle)=%s" % type(aTextJSHandle))
    # type(aTextJSHandle)=<class 'pyppeteer.execution_con
    # print("aTextJSHandle=%s" % aTextJSHandle)
    # aTextJSHandle=<pyppeteer.execution_context.JSHandle
    title = await aTextJSHandle.jsonValue()
    # print("type(title)=%s" % type(title))
    # type(title)=<class 'str'>
    print("title=%s" % title)

    baiduLinkUrl = await (await aElem.getProperty("href"))
    print("baiduLinkUrl=%s" % baiduLinkUrl)

```

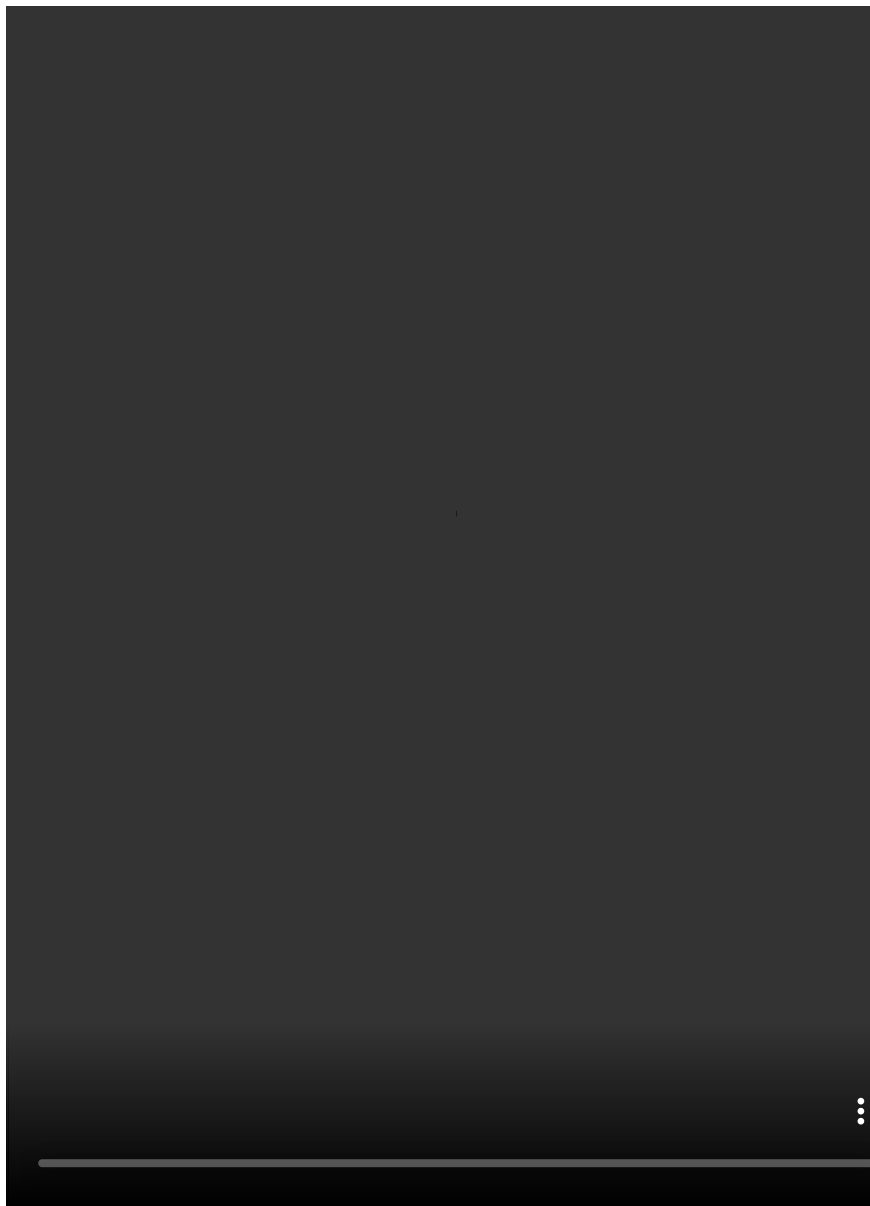
查找定位元素

```
await browser.close()

asyncio.get_event_loop().run_until_complete(main())
```

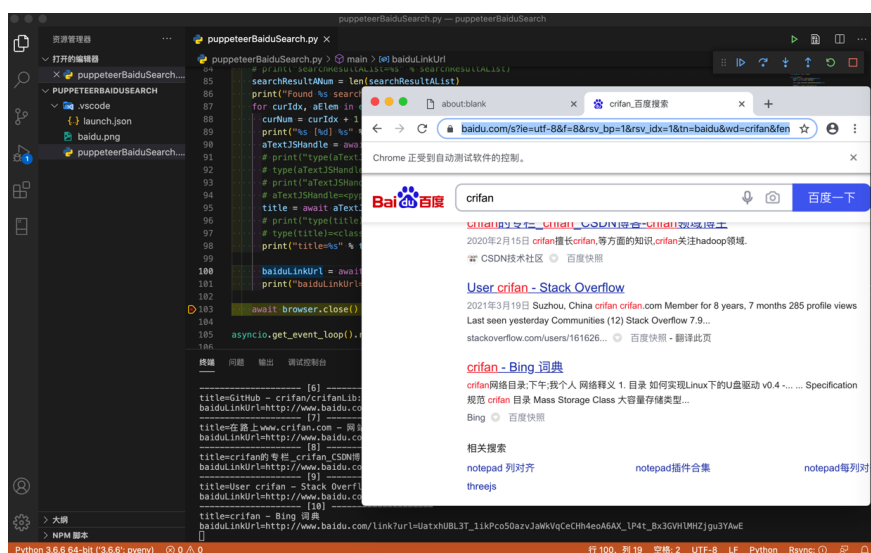
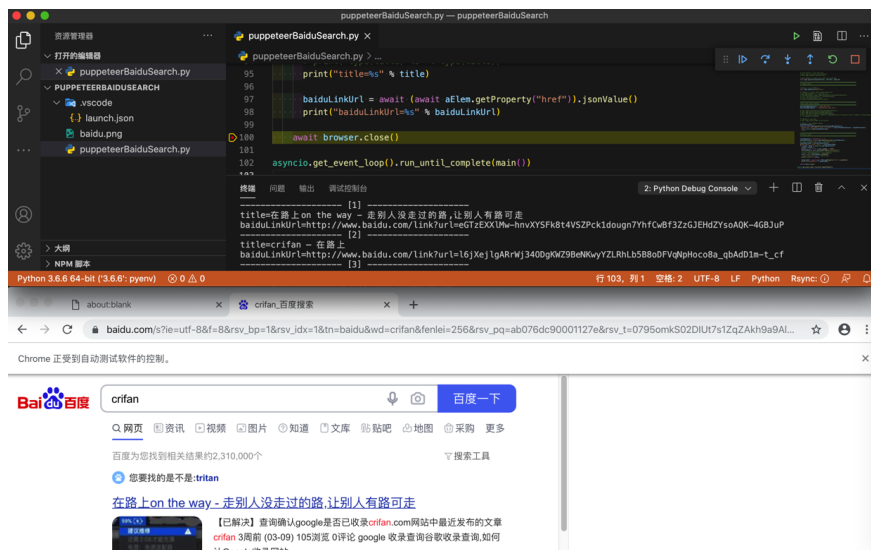
效果

视频



图

## 查找定位元素



## 输出

```
Still not found span.nums_text, wait 1 seconds
Found 10 search result:
----- [1] -----
title=在路上on the way - 走别人没走过的路,让别人有路可走
baiduLinkUrl=http://www.baidu.com/link?url=eGTzEXXlMw-hnvX
----- [2] -----
title=crifan - 在路上
baiduLinkUrl=http://www.baidu.com/link?url=l6jXejlgARrWj340
----- [3] -----
title=crifan简介_crifan的专栏-CSDN博客_crifan
baiduLinkUrl=http://www.baidu.com/link?url=IIqPM5wuVE_QP7S
----- [4] -----
title=crifan的微博_微博
baiduLinkUrl=http://www.baidu.com/link?url=NnqeMlu4Jr_Ld-zc
----- [5] -----
title=Crifan的电子书大全 | crifan.github.io
baiduLinkUrl=http://www.baidu.com/link?url=u0Z-AmgNBNr3mGd
----- [6] -----
title=GitHub - crifan/crifanLib: crifan's library
baiduLinkUrl=http://www.baidu.com/link?url=t42I1rYfn32DGw9
----- [7] -----
title=在路上www.crifan.com - 网站排行榜
baiduLinkUrl=http://www.baidu.com/link?url=WwLwfxA72vK080by
----- [8] -----
title=crifan的专栏_crifan_CSDN博客-crifan领域博主
baiduLinkUrl=http://www.baidu.com/link?url=Cmcn2mXwiZr87FB
----- [9] -----
title=User crifan - Stack Overflow
baiduLinkUrl=http://www.baidu.com/link?url=yGgsq1z2vNDAAeW
----- [10] -----
title=crifan - Bing 词典
baiduLinkUrl=http://www.baidu.com/link?url=UatxhUBL3T_1ikP
```

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:28:54

## 常见问题

### waitFor系列函数全部不可用

#### waitFor系列的所有函数都无效

经过实际测试，`waitFor` 系列的各个函数，此处都无效

```
# await page.waitForSelector(SearchFoundWordsSelector)
# await page.waitFor(SearchFoundWordsSelector)
# await page.waitForXPath(SearchFoundWordsXPath)
# Note: all above exception: 发生异常: ElementHandleError
```

都会报错: `ElementHandleError Evaluation failed: TypeError: MutationObserver is not a constructor`

所以，如果想要实现，等待元素出现，只能 `while` + `querySelector` 去不断检测去实现

比如

```
waitSeconds = 1
while not await page.querySelector(xxxSelector):
    await asyncio.sleep(waitSeconds)
```

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:22:23

## 附录

下面列出相关参考资料。

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新: 2021-07-17 17:21:08

## puppeteer资料

- 官网
  - github
    - puppeteer/puppeteer: Headless Chrome Node.js API
      - <https://github.com/puppeteer/puppeteer>
  - google
    - Puppeteer | Tools for Web Developers | Google Developers
      - <https://developers.google.com/web/tools/puppeteer>
    - Quick start | Tools for Web Developers | Google Developers
      - <https://developers.google.com/web/tools/puppeteer/get-started>

**Python 版 puppeteer : pyppeteer**

puppeteer 是基于 (NodeJS的) js 语言的

对应的Python版本的库是: pyppeteer

- PyPI
  - pyppeteer · PyPI
    - <https://pypi.org/project/pyppeteer/>
- Github
  - 旧版 = miyakogi版
    - miyakogi/pyppeteer: Headless chrome/chromium automation library (unofficial port of puppeteer)
      - <https://github.com/miyakogi/pyppeteer>
        - 已经archive了
          - 最后更新: 8 May 2020
    - 对应文档
      - API Reference — Pyppeteer 0.0.25 documentation
        - <https://miyakogi.github.io/pyppeteer/reference.html>
      - Pyppeteer's documentation — Pyppeteer 0.0.25 documentation
        - <https://miyakogi.github.io/pyppeteer/index.html>
      - pyppeteer.page — Pyppeteer 0.0.25 documentation
        - [https://miyakogi.github.io/pyppeteer/\\_modules/pyppeteer/page.html](https://miyakogi.github.io/pyppeteer/_modules/pyppeteer/page.html)
  - 新版 = pyppeteer版
    - puppeteer/pyppeteer: Headless chrome/chromium automation library (unofficial port of puppeteer)

- <https://github.com/pyppeteer/pyppeteer>
  - 但是是非官方的
    - 最后更新：2021 9 Jan
- 对应文档
  - Pyppeteer's documentation — Pyppeteer 0.0.25 documentation
    - <https://pyppeteer.github.io/pyppeteer/>
  - API Reference — Pyppeteer 0.0.25 documentation
    - <https://pyppeteer.github.io/pyppeteer/reference.html>

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新： 2021-07-17 17:27:15

## 参考资料

- **【已解决】** pyppeteer中提取百度搜索结果中的信息
- **【已解决】** pyppeteer中page.querySelectorAll运行时无法获取到结果
- **【规避解决】** pyppeteer不调试直接运行waitForSelector报错：  
ElementHandleError Evaluation failed TypeError MutationObserver  
is not a constructor at pollMutation
- **【已解决】** Mac中初始化搭建Python版puppeteer的pyppeteer的开发环境
- **【已解决】** pyppeteer如何给输入框中输入文字
- 
- [网络爬虫之使用pyppeteer替代selenium完美绕过webdriver检测 阅读目录 - 知乎](#)
- [爬虫神器pyppeteer，对js加密降维打击 - 掘金](#)
- [pyppeteer\(python版puppeteer\)基本使用 - 白灰 - 博客园](#)
- [Selenium 凭什么成为 Web 自动化测试的首选？（内附图谱）| 极客时间](#)
- [为什么puppeteer比selenium好？ - 掘金](#)
- [Selenium vs Puppeteer: testing the testing tools](#)
- [Selenium vs. Puppeteer for Test Automation: Is a New Leader Emerging? - Flood](#)
- [Selenium vs. Puppeteer - When to Choose What? | TestProject](#)
- [Puppeteer: 更友好的 Headless Chrome Node API - 知乎](#)
- [无头浏览器 Puppeteer 初探 - 掘金](#)
- [Puppeteer-无头浏览器简介 - 知乎](#)
- [Puppeteer浏览器自动化 - HelpDocs](#)
- [Puppeteer 配置浏览器属性 | LFhacks.com](#)
- [Puppeteer 获取和修改 元素节点的属性 | LFhacks.com](#)
- [Any method to access full html content? · Issue #331 · puppeteer/puppeteer](#)
- [Pyppeteer 使用笔记 - 拐弯 - 博客园](#)
- [pyppeteer\(python版puppeteer\)基本使用 - 白灰 - 博客园](#)
- [Pyppeteer's documentation — Pyppeteer 0.0.25 documentation](#)
- [miyakogi/pyppeteer: Headless chrome/chromium automation library \(unofficial port of puppeteer\)](#)
- 

crifan.com，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,  
powered by Gitbook最后更新： 2021-07-17 17:26:02