



第一章 环境与简介 (上海-悠悠)

第一章 环境与简介 (上海-悠悠)

1.1 Selenium 发展史

- 1.1.1 Selenium 1.0
- 1.1.2 Selenium 2.0
- 1.1.3 Selenium 3.0
- 1.1.4 Selenium 4.0

1.2 后 Selenium 时代

- 1.2.1 Cypress 简介
- 1.2.2 Puppeteer
- 1.2.3 TestCafe
- 1.2.4 为什么要学 Playwright ?

1.3 playwright 环境准备

- 1.3.1 环境准备
- 1.3.2 运行代码快速体验

1.4 快速开始

- 1.4.1 简单使用
- 1.4.2 headless 模式
- 1.4.3 关于等待

1.5 录制生成脚本

- 1.5.1 启动运行
- 1.5.2 录制相关命令操作
- 1.5.3 page.pause() 断点调试

1.6 pause断点 inspector 使用

- 1.6.1 page.pause() 断点
- 1.6.2 console 调试定位
- 1.6.3 playwright inspector 使用

1.7 Playwright+Pytest+Allure POM项目

- 1.7.1 POM 设计模式
- 1.7.2 运行用例

1.8 窗口最大化

- 1.8.1 no_viewport 禁用窗口大小
- 1.8.2 viewport 指定窗口大小

- 1.8.3 第2种方式
- 1.9 启动 Chrome 和 Edge浏览器
 - 1.9.1 命令行下载
 - 1.9.2 指定 channel 打开浏览器
- 1.10 启动本地 Chrome 加载用户缓存
 - 1.10.1 selenium 加载 Google Chrome 插件
 - 1.10.2 如何查看 `--user-data-dir` 用户数据目录
 - 1.10.3 playwright 启动本地 Google Chrome
 - 1.10.4 配置args 参数加载插件
 - 1.10.5 使用代理
- 1.11 操作已打开浏览器，绕过登录验证码
 - 1.11.1 环境准备
 - 1.11.2 参数配置
 - 1.11.3 playwright 接管页面
- 1.12 登录页面滑动解锁
 - 1.12.1 滑动解锁场景
 - 1.12.2 操作滑块
- 1.13 离线安装 playwright 环境
 - 1.13.1 playwright 本地下载
 - 1.13.2 多个包批量下载
 - 1.13.3 离线安装
 - 1.13.4 启动本地 chrome 浏览器
 - 1.13.5 离线安装chromium,firefox 和 webkit
- 1.14 Pyinstaller 打包生成独立的可执行文件
 - 1.14.1 打成一个exe的独立包
 - 1.14.2 chromium 本地化安装
 - 1.14.3 Pyinstaller打包
 - 1.14.4 icon 制作
- 1.15 登录-验证码识别
 - 1.15.1 登录验证码
 - 1.15.2 代码示例
- 1.16 登录-滑块拼图验证码
 - 1.16.1 滑块示例
 - 1.16.2 计算缺口位置
 - 1.16.3 定位滑动的按钮

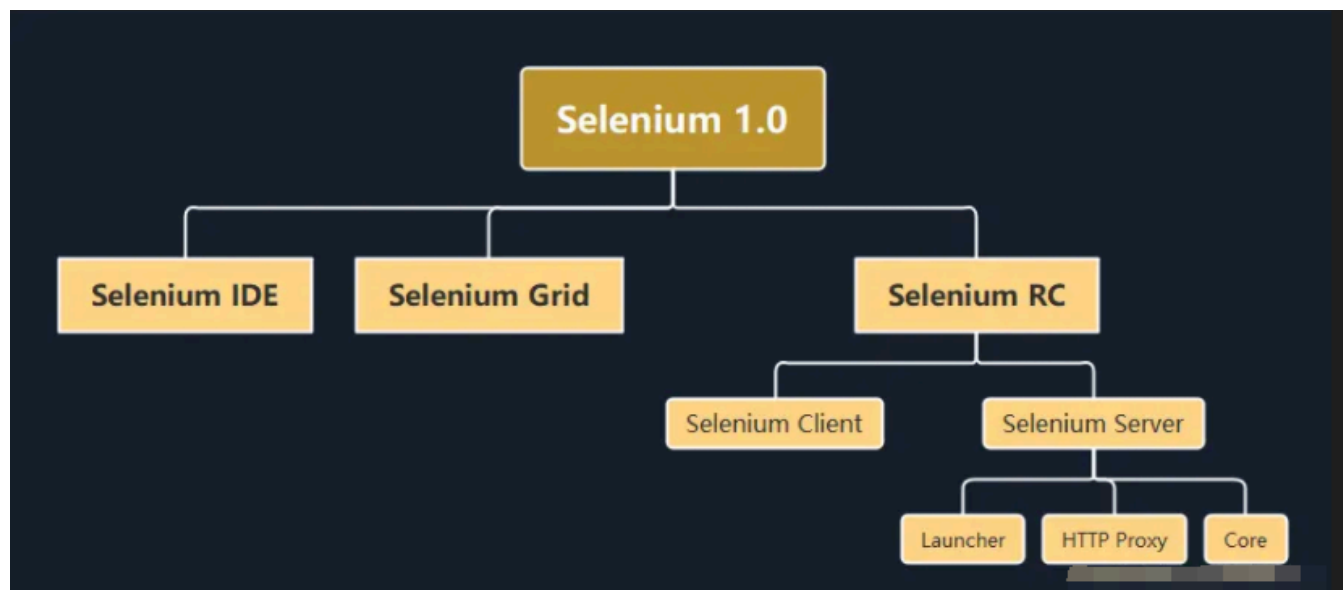
- 1.16.4 page.mouse 鼠标操作
- 1.16.5 代码示例
- 1.17 非无痕模式启动 launch_persistent_context
 - 1.17.1 无痕模式启动浏览器
 - 1.17.2 非无痕模式 launch_persistent_context
- 1.18 highlight 调试定位时高亮显示元素
 - 1.18.1 遇到的问题
 - 1.18.2 highlight 高亮调试

▪[网易云视频课程](#)

1.1 Selenium 发展史

1.1.1 Selenium 1.0

在 Selenium 1.0 发布的 2006 年，Google 工程师 Simon Stewart 发起了一个名为 WebDriver 的项目。它也是一个自动化测试工具，彼时刚刚起步，后来它也将成为 Selenium 的竞品之一。

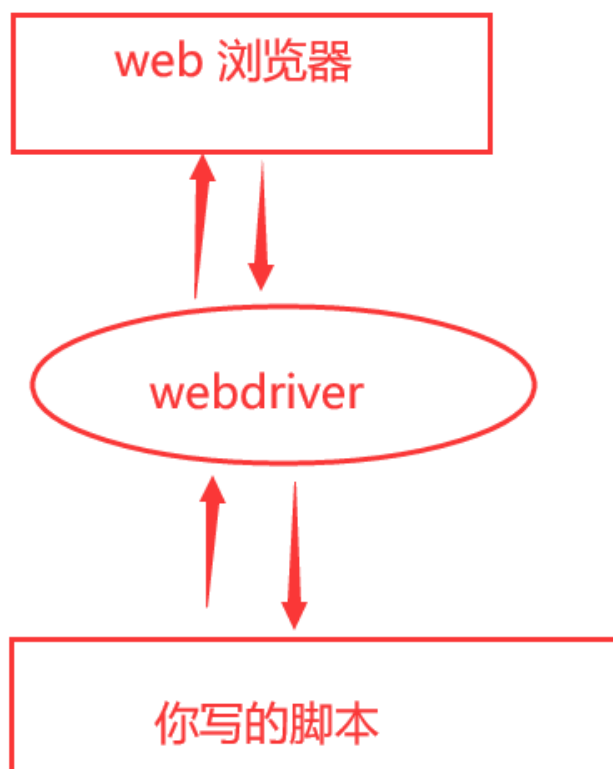


1.1.2 Selenium 2.0

2009 年，在 Google 测试自动化会议上，两个团队的开发人员在沟通后决定合并这两个项目，新项目被命名为 Selenium Web Driver，也就是 Selenium 2.0。

WebDriver 的作者是这样解释二者合并的原因的：“一方面 WebDriver 解决了 Selenium 存在的缺点（例如可以绕过 JavaScript 沙箱，WebDriver 有出色的API），另一方面是 Selenium 解决了 WebDriver 存在的问题，还有就是 Selenium 的主要贡献者和我都以为合并项目是为用户提供最优秀框架的最佳途径。”

Selenium 2.x (WebDriver) 真正的兴起是在 2014 年开始，到 2016 年左右成为 web 自动化最热门的框架。几乎只要一说 web 自动化，那就是 selenium，它不仅仅在 web 自动化测试领域很火，在爬虫领域也是非常热门的，可以说全国不管哪行的都开始学 selenium 了。



Selenium Remote Control (RC)：支持多种平台(Windows, Linux, Solaris)和多种浏览器(IE, Firefox, Opera, Safari)，可以用多种语言(Java, Ruby, Python, Perl, PHP, C#)

1.1.3 Selenium 3.0

2016 年，Selenium 3 发布。这个版本并没有引入新的工具，主要加强了对浏览器的支持。

相较 Selenium 2 的主要的变动有：

完全移除了 Selenium RC。

WebDriver 暴露一个供浏览器接入的 API，通过各浏览器厂商提供的 Driver 来接入。

将 Firefox Driver 剔除（之前 Firefox Driver 是内置的）。

支持 Firefox 通过 GECKO Driver 来接入 Selenium。

通过 Apple 提供的 Safari Driver，Selenium 可以支持 Safari 接入。

通过 Edge Driver 支持 IE 接入。

1.1.4 Selenium 4.0

2021 年，Selenium 发布 Selenium 4。

在 Selenium 3 中，与浏览器的通信基于 JSON-wire 协议，因此 Selenium 需要对 API 进行编解码。而 Selenium 4 遵循 W3C 标准协议，Driver 与浏览器之间通信的标准化使得他们可以直接通信。

除此之外，Selenium 4 还做了很多改动。包括：

优化了对浏览器的支持。

使用新的设计优化了 Selenium Grid。

标准化了 Selenium 的文档（你敢信从 Selenium 2.0 开始，文档就没更新过...）。

IDE 中的 CLI Runner 变更为基于 NodeJS（之前是 HTML Runner）。

Client 和 Driver 支持了新的元素定位 API。

支持屏幕截图。

改进了 Chrome Dev Tools。之前 Chrome Driver 直接继承自 Remote Web Driver 类，现在继承自 Chromium Driver 类，这个改动使得 IDE 开发可以使用更多的 API。

1.2 后 Selenium 时代

Selenium 的辉煌时代源于 webdriver 的引入，在2020年逐步走向没落，

最终成也 webdriver 败也webdriver!

在2020年左右，Cypress、TestCafe、Puppeteer在技术雷达中被誉为后Selenium时代Web UI测试的三驾马车。

web 自动化

selenium 统一天下

末年，三分天下

cypress

puppeteer

testcafe

天下一统

playwright

1.2.1 Cypress 简介

Cypress 的第一版于2015年发布，_并于2018年开始真正流行，2020年,Cypress正式开源
Cypress官网地址：<https://docs.cypress.io>

随便着Web应用项目的不断发展，测试也要不断发展，Cypress是为现代网络打造的，基于JavaScript语言的一种前端自动化测试工具，测试也是如此，对浏览器中运行的所有内容进行快速，轻松和可靠的测试。Cypress是自集成的，这类测试框架统称为e2e测试，即end to

end（端到端）测试。理论上前端页面由前端框架来测试确实更为合适。它提供了一套完整的端到端的测试体验，正如官网上所说：

1. 安装Cypress很简单,无需依赖项,无需额外下载或更改代码。
2. 轻松，快速地编写测试，并在构建Web应用程序时观察它们的实时执行。
3. 在CI中调试测试就像在本地运行测试一样容易。具有内置的并行化和负载平衡功能。
4. 记录CI测试数据，屏幕截图和视频-并在“仪表盘”中查看汇总的下一级见解。

八大功能：

1. 时间穿梭：Cypress会在测试运行时拍摄快照。只需将鼠标悬停在“命令日志”中的命令上，即可准确查看每一步都发生了什么
2. 可调试性：停止猜测你的测试失败的原因。直接从熟悉的工具如Chrome DevTools进行调试。我们可读的错误和堆栈跟踪使调试更加快速便捷。
3. 实时重载：每当你对测试进行更改时，Cypress都会自动重新加载。查看应用程序中实时执行的命令。
4. 自动等待：不要在测试中添加等待或休眠。Cypress在继续下一步之前会自动等待元素至可操作状态时才会执行命令或断言。异步操作不再是噩梦。
5. 间谍，存根和时钟：Cypress允许你验证并控制函数行为，Mock服务器响应或更改系统时间，你喜欢的单元测试就在眼前。
6. 网络流量控制：Cypress可以Mock服务器返回结果，无须连接后端服务器即可实现轻松控制，模拟网络请求。
7. 运行结果一致性：Cypress架构不使用Selenium或Webdriver,在运行速度，可靠性测试，测试结果一致性上均有良好的保障。
8. 截图和视频：Cypress在测试运行失败时自动截图，在无头运行时录制整个测试套件的视频，使你轻松掌握测试运行情况。

1.2.2 Puppeteer

Puppeteer 是 Chrome 开发团队在 2017 年发布的一个 Node.js 包, puppeteer 是google公司出品 对于运行无界面chrome (学名headless)。

Puppeteer 是一个 Node 库，它提供了一个高级 API 来通过 DevTools 协议控制 Chromium 或 Chrome。

Puppeteer 默认以无头模式（headless）运行，也就是运行一个无界面的 Chrome 浏览器。

2 应用场景

2.1 页面生成 PDF

Puppeteer 提供了页面生成 PDF 的方法，我们可以利用这个方法将页面导出为 PDF，导出的 PDF 效果和 Chrome 浏览器打印功能导出的 PDF 一致。

具体的应用场景有：

- 报表导出 PDF
- 在线文档导出 PDF

2.2 页面截图

Puppeteer 提供了截图的方法，我们可以利用这个方法将页面的指定区域导出为 jpeg 或 png 图片。

具体的应用场景有：

- 报表导出图片
- 在线文档导出图片
- 营销页面生成预览图、封面图
- 生成骨架屏图片

2.3 服务端渲染

单页应用（SPA）的主要内容是在 JavaScript 向服务端请求数据后渲染的，存在爬虫难以抓取主要内容、首屏加载慢等问题，而使用 Next.js、Nuxt.js 等服务端渲染框架改造的成本较高。

如果只是为了搜索引擎优化，我们可以考虑利用 Puppeteer 来实现。我们可以在网关层判断请求的来源，如果是爬虫，直接返回由 Puppeteer 服务端渲染的 html 文件。

2.4 自动化UI测试

使用 Puppeteer 可以模拟 Chrome 浏览器环境，结合 JavaScript 测试框架（如 Jest）可以实现自动化 UI 测试。

Puppeteer 提供了 Mouse 类来模拟鼠标操作，提供了 Keyboard 类来模拟键盘操作，提供了 Touchscreen 类来模拟触屏操作，并且 Puppeteer 提供的 Page 类里有很多方法可以用来操作元素，比如点击元素、聚焦元素等操作

1.2.3 TestCafe

2018 年11月TestCafe首次进入技术雷达_,处于评估阶段,2019年4月的技术雷达里,它的状态更新为试验阶段

TestCafe 是一款 Node.js 库, 用来实现 web页面的自动化测试, 支持多种浏览器, 将来还有接口测试的计划。

下面通过和 **Puppeteer** 的对比来总结一些 TestCafe 的特点 (顺便带上 selenium) :

对比	Puppeteer	TestCafe	Selenium
语言	Javascript	Javascript/TypeScript	多种 binding 语言
兼容浏览器	仅Chrome系列	多种浏览器 (包括 IE)	多种浏览器 (包括 IE)
测试框架	不面向测试, 无框架	本身就是测试框架	无框架
工作原理	通过WebSocket 协议与 chrome 的通信	通过设置代理, 将用户的行为传递给被测页面	通过 wire protocol 和 浏览器 通信

1.2.4 为什么要学 Playwright ?

2020年微软开源一个 UI 自动化测试工具 Playwright, 支持 Node.js、Python、C# 和 Java 语言。

当 Microsoft 于2020 年 1 月 31 日发布Playwright的第一个公共版本时, 我们获得了一个新选项。

如果您比较Playwright和 Puppeteer的贡献者页面, 您会注意到Puppeteer的前两个贡献者现在在 Playwright 上工作。Puppeteer 团队实质上是从 Google 转移到 Microsoft 并成为 Playwright 团队。

因此, Playwright 在很多方面与 Puppeteer 非常相似。API 方法在大多数情况下是相同的, 并且默认情况下 Playwright 还捆绑了兼容的浏览器。

Playwright 最大的区别在于跨浏览器支持。它可以驱动 Chromium、WebKit (Safari 的浏览器引擎) 和 Firefox。

那么现在微软推出的 Playwright 到底有没必要去学呢? 先看下官方介绍<https://playwright.dev/python/>

跨浏览器和平台

- 跨浏览器。Playwright 支持所有现代渲染引擎，包括 Chromium、WebKit（Safari 的浏览器引擎）和 Firefox。
- 跨平台。在 Windows、Linux 和 macOS 上进行本地测试或在 CI 上进行无头或有头测试。
- 跨语言。在 TypeScript、JavaScript、Python、.NET、Java 中使用 Playwright API。
- 测试移动网络。适用于 Android 和 Mobile Safari 的 Google Chrome 浏览器的本机移动仿真。相同的渲染引擎适用于您的桌面和云端。

稳定性

- 自动等待。Playwright 在执行动作之前等待元素可操作。它还具有一组丰富的内省事件。两者的结合消除了人为超时的需要——这是不稳定测试的主要原因。
- Web 优先断言。Playwright 断言是专门为动态网络创建的。检查会自动重试，直到满足必要的条件。
- 追踪。配置测试重试策略，捕获执行跟踪、视频、屏幕截图以定位问题。

运行机制

浏览器在不同进程中运行属于不同来源的 Web 内容。Playwright 与现代浏览器架构保持一致，并在进程外运行测试。这使得 Playwright 摆脱了典型的进程内测试运行器的限制。

- 多重一切。测试跨越多个选项卡、多个来源和多个用户的场景。为不同的用户创建具有不同上下文的场景，并在您的服务器上运行它们，所有这些都在一次测试中完成。
- 可信事件。悬停元素，与动态控件交互，产生可信事件。Playwright 使用与真实用户无法区分的真实浏览器输入管道。
- 测试框架，穿透 Shadow DOM。Playwright 选择器穿透影子 DOM 并允许无缝地输入帧。

完全隔离-快速执行

- 浏览器上下文。Playwright 为每个测试创建一个浏览器上下文。浏览器上下文相当于一个全新的浏览器配置文件。这提供了零开销的完全测试隔离。创建一个新的浏览器上下文只需要几毫秒。
- 登录一次。保存上下文的身份验证状态并在所有测试中重用它。这绕过了每个测试

中的重复登录操作，但提供了独立测试的完全隔离。

强大的工具

- 代码生成器。通过记录您的操作来生成测试。将它们保存为任何语言。
- 调试。检查页面、生成选择器、逐步执行测试、查看点击点、探索执行日志。
- 跟踪查看器。捕获所有信息以调查测试失败。Playwright 跟踪包含测试执行截屏、实时 DOM 快照、动作资源管理器、测试源等等。

1.3 playwright 环境准备

版本要求

python3.7+ 版本 (课程推荐python3.8)版本

playwright 最新版 1.31.1

1.3.1 环境准备

Playwright 是专门为满足端到端测试的需要而创建的。Playwright 支持所有现代渲染引擎，包括 Chromium、WebKit（Safari 的浏览器引擎）和 Firefox。

在 Windows、Linux 和 macOS 上进行本地测试或在 CI 上进行测试。

python 版本要求 python3.7+ 版本。

安装 playwright :

```
pip install playwright
```

安装所需的浏览器 chromium,firefox 和 webkit :

```
playwright install
```

仅需这一步即可安装所需的浏览器，并且不需要安装驱动包了（解决了selenium启动浏览器，总是要找对应驱动包的痛点）

整个下载过程会比较慢，大概需要3-5分钟，它会下载3个内置浏览器，放到你电脑本地。

如果安装报错，提示缺少Visual C++， 解决办法：

安装Microsoft Visual C++ Redistributable 2019

```
https://aka.ms/vs/16/release/VC_redist.x64.exe
```

直接点击就可以下载了，下载后直接安装即可。

或者运行代码报错 `Dll load failed while importing _greenlet`，也可以用上面方法解决

```
Traceback (most recent call last):
  File "E:\LESSON\PythonBaseDocument\WEB_ui\UI\lesson1.py", line 1, in <module>
    from playwright.sync_api import sync_playwright
  File "E:\LESSON\PythonBaseDocument\WEB_ui\venv\lib\site-packages\playwright\sync_api\__init__.py", line 25, in <module>
    import playwright.sync_api._generated
  File "E:\LESSON\PythonBaseDocument\WEB_ui\venv\lib\site-packages\playwright\sync_api\_generated.py", line 25, in <module>
    from playwright._impl._accessibility import Accessibility as AccessibilityImpl
  File "E:\LESSON\PythonBaseDocument\WEB_ui\venv\lib\site-packages\playwright\_impl\_accessibility.py", line 17, in <module>
    from playwright._impl._connection import Channel
  File "E:\LESSON\PythonBaseDocument\WEB_ui\venv\lib\site-packages\playwright\_impl\_connection.py", line 34, in <module>
    from greenlet import greenlet
  File "E:\LESSON\PythonBaseDocument\WEB_ui\venv\lib\site-packages\greenlet\__init__.py", line 29, in <module>
    from _greenlet import _C_API # pylint:disable=no-name-in-module
ImportError: DLL load failed while importing _greenlet: 找不到指定的模块。
```

1.3.2 运行代码快速体验

启动浏览器并打开百度页面

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False) # 启动 chromium 浏览器
    page = browser.new_page() # 打开一个标签页
    page.goto("https://www.baidu.com") # 打开百度地址
    print(page.title()) # 打印当前页面title
    browser.close() # 关闭浏览器对象
```

1.4 快速开始

1.4.1 简单使用

官方介绍<https://playwright.dev/python/>

安装后，您可以在 Python 脚本中使用 Playwright，并启动 3 种浏览器中的任何一种（chromium,firefox和webkit）。

启动浏览器并打开百度页面

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False) # 启动 chromium 浏览器
    page = browser.new_page() # 打开一个标签页
    page.goto("https://www.baidu.com") # 打开百度地址
    print(page.title()) # 打印当前页面title
    browser.close() # 关闭浏览器对象
```

Playwright 支持2种运行方式：同步和异步。如果您的现代项目使用asyncio，您应该使用 async API：

以下是异步运行方式

```
import asyncio
from playwright.async_api import async_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

async def main():
    async with async_playwright() as p:
        browser = await p.chromium.launch(headless=False)
        page = await browser.new_page()
        await page.goto("https://www.baidu.com")
        print(await page.title())
        await browser.close()

asyncio.run(main())
```

如果你不习惯with语句，也可以用start() 和stop() 的方式

```
from playwright.sync_api import sync_playwright

playwright = sync_playwright().start()

browser = playwright.chromium.launch(headless=False)
page = browser.new_page()
page.goto("https://www.baidu.com/")

browser.close()
playwright.stop()
```

1.4.2 headless 模式

默认情况下，Playwright 以无头模式运行浏览器。要查看浏览器 UI，请`headless=False`在启动浏览器时传递标志。

headless 无头模式运行浏览器示例：

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch()           # 启动 chromium 浏览器
    page = browser.new_page()               # 打开一个标签页
    page.goto("https://www.baidu.com")     # 打开百度地址
    print(page.title())                    # 打印当前页面title
    browser.close()                         # 关闭浏览器对象
```

1.4.3 关于等待

Playwright 打开浏览器运行脚本的速度那就是一个字：快！

您还可以用来`slow_mo`（单位是毫秒）减慢执行速度。它的作用范围是全局的，从启动浏览器到操作元素每个动作都会有等待间隔，方便在出现问题的时候看到页面操作情况。

```
chromium.launch(headless=False, slow_mo=50)
```

使用示例

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://www.baidu.com")
    print(page.title())
    page.fill('#kw', "上海-悠悠博客")
    page.click('#su')
    browser.close()
```

运行后会发现每个操作都会有间隔时间。

time.sleep() 不再使用

Playwright 在查找元素的时候具有自动等待功能，如果你在调试的时候需要使用等待，你应该使用 `page.wait_for_timeout(5000)` 代替 `time.sleep(5)` 并且最好不要等待超时。

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    page = browser.new_page()
    page.goto("https://www.baidu.com")
    print(page.title())
    # 等待5秒
    page.wait_for_timeout(5000)
    page.fill('#kw', "上海-悠悠博客")
    page.click('#su')
    browser.close()
```

请使用 `wait(wait_for_timeout)` 方法而不是 `time` 模块。这是因为我们内部依赖于异步操作，并且在使用时 `time.sleep(5)` 无法正确处理它们。

playwright 可以支持自动录制生成脚本，也就是说只需要在页面上点点点，就可以自动生成对应的脚本了。

1.5 录制生成脚本

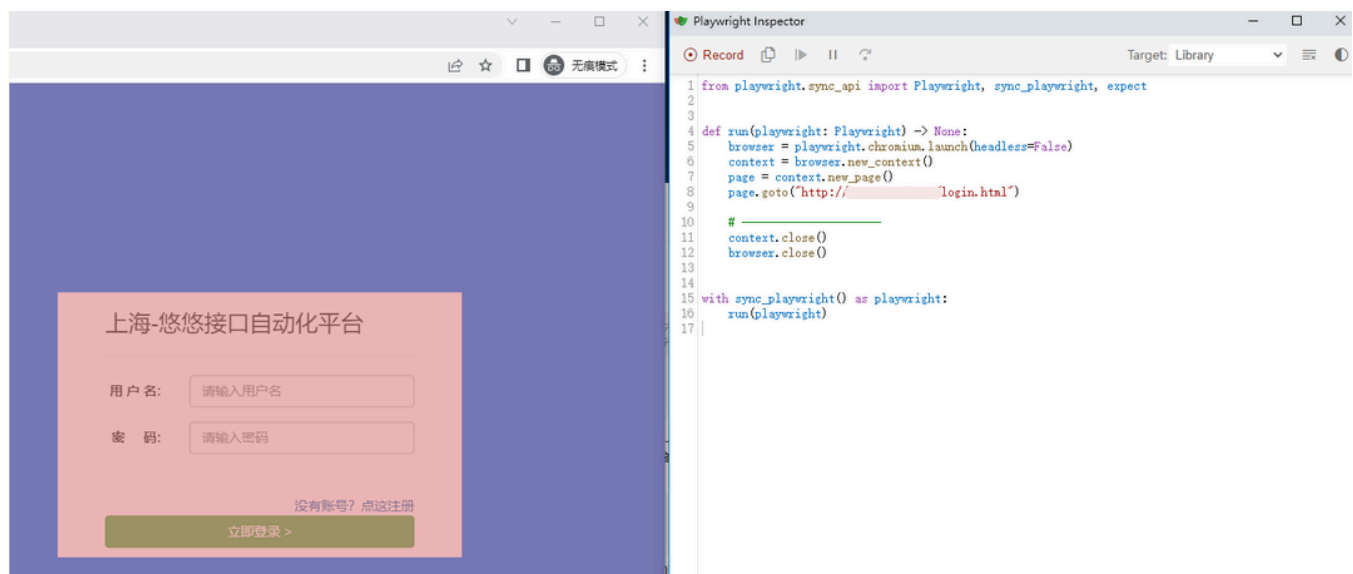
1.5.1 启动运行

Playwright 具有开箱即用的生成测试的能力，是快速开始测试的好方法。它将打开两个窗口，一个是浏览器窗口，您可以在其中与要测试的网站进行交互，另一个是 Playwright Inspector 窗口，您可以在其中记录测试、复制测试、清除测试以及更改测试语言。

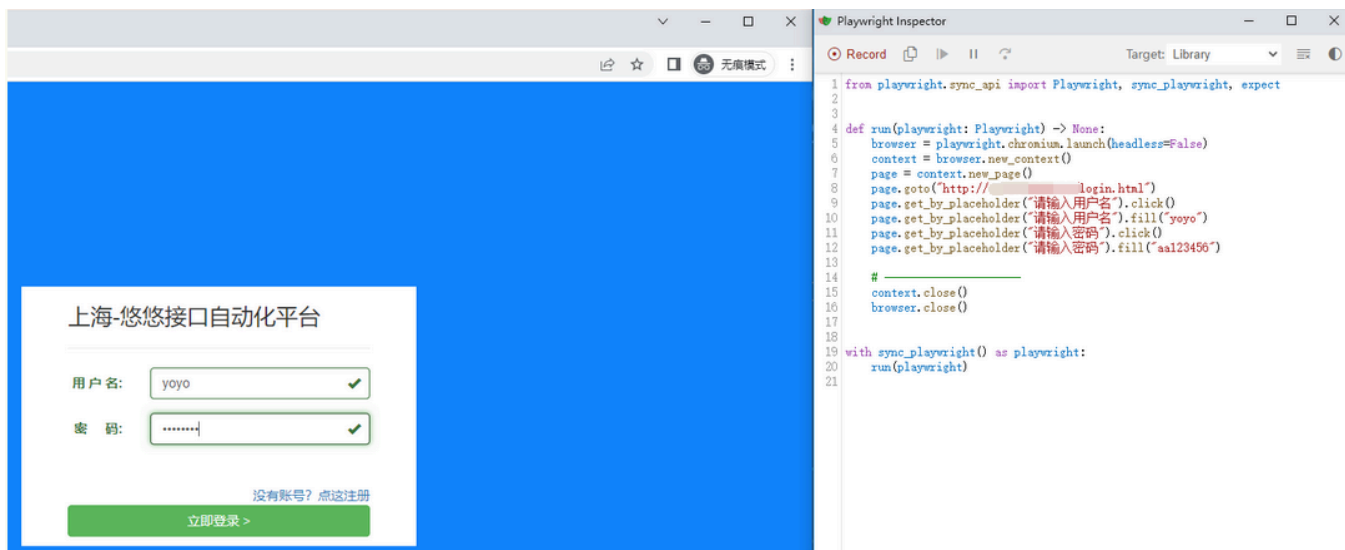
使用命令行启动

```
playwright codegen http://网站地址
```

启动后，电脑上出现2个窗口，左边是浏览器打开网站可以操作，右边是inspector 可以生成对应的脚本



登录框输入账号和密码点登录为例



一个完整的登录流程代码生成如下

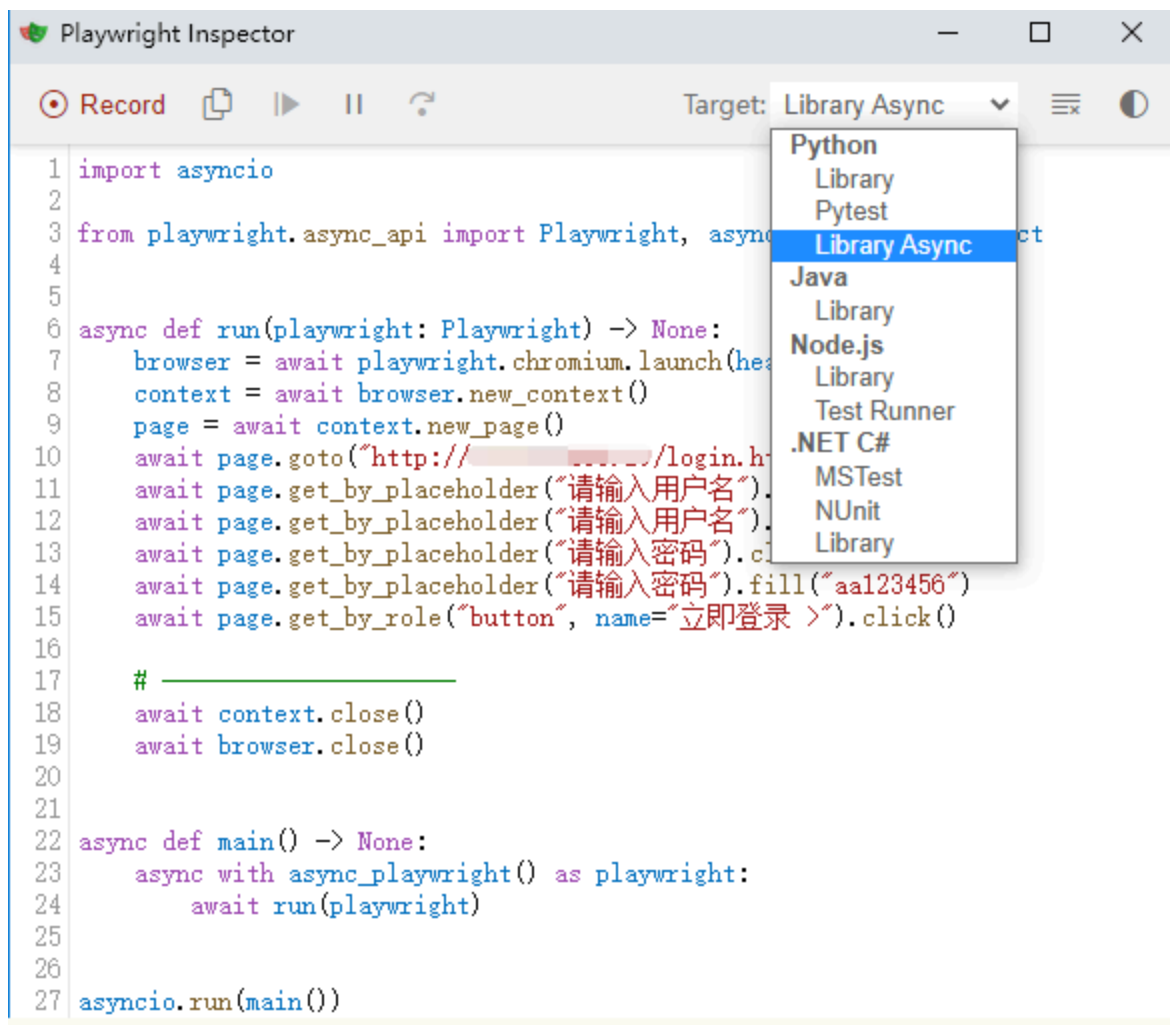
```
from playwright.sync_api import Playwright, sync_playwright, expect
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def run(playwright: Playwright) -> None:
    browser = playwright.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto("http://127.0.0.1:8000/login.html")
    page.get_by_placeholder("请输入用户名").click()
    page.get_by_placeholder("请输入用户名").fill("yoyo")
    page.get_by_placeholder("请输入密码").click()
    page.get_by_placeholder("请输入密码").fill("aa123456")
    page.get_by_role("button", name="立即登录 >").click()

    # -----
    context.close()
    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```

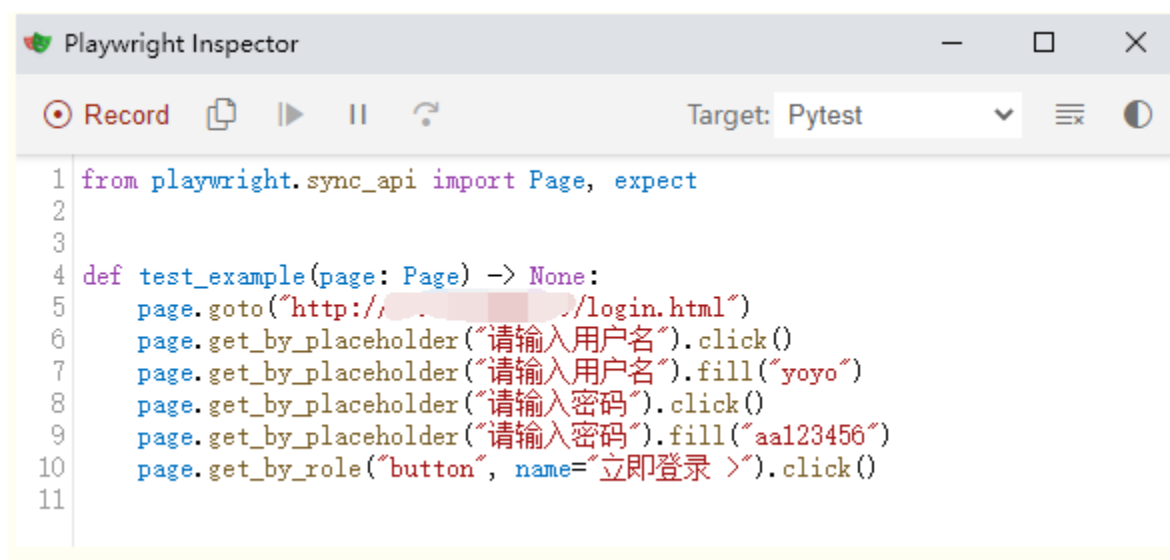
还可以选择生成异步代码



The image shows the Playwright Inspector application. The top bar includes a 'Record' button and a 'Target' dropdown menu currently set to 'Library Async'. A dropdown menu is open, showing a list of programming languages and their associated libraries: Python (Library, Pytest, Library Async), Java (Library), Node.js (Library, Test Runner), .NET C# (MSTest, NUnit, Library). The main code editor displays Python code for an asynchronous login test. The code imports 'asyncio' and 'Playwright' from 'playwright.async_api'. It defines an 'async def run' function that launches a browser, creates a context, and navigates to a login page. It then performs actions like clicking on placeholders and filling in a username 'yoyo' and password 'aa123456'. Finally, it clicks a button with the role 'button' and name '立即登录 >'. A 'main' function is also defined to run the 'run' function, and 'asyncio.run(main())' is called at the bottom.

```
1 import asyncio
2
3 from playwright.async_api import Playwright, async_playwright
4
5
6 async def run(playwright: Playwright) -> None:
7     browser = await playwright.chromium.launch(headless=True)
8     context = await browser.new_context()
9     page = await context.new_page()
10    await page.goto("http://localhost:3000/login.html")
11    await page.get_by_placeholder("请输入用户名").click()
12    await page.get_by_placeholder("请输入用户名").fill("yoyo")
13    await page.get_by_placeholder("请输入密码").click()
14    await page.get_by_placeholder("请输入密码").fill("aa123456")
15    await page.get_by_role("button", name="立即登录 >").click()
16
17    # ...
18    await context.close()
19    await browser.close()
20
21
22 async def main() -> None:
23     async with async_playwright() as playwright:
24         await run(playwright)
25
26
27 asyncio.run(main())
```

如果你是写自动化测试用例，还能自动生成 pytest 框架的代码，简直了！



The image shows the Playwright Inspector application with the 'Target' dropdown menu set to 'Pytest'. The main code editor displays Python code for a login test using the pytest framework. The code imports 'Page' and 'expect' from 'playwright.sync_api'. It defines a 'def test_example' function that takes a 'Page' object as an argument. Inside the function, it performs the same login actions as the previous code: navigating to the login page, clicking on the username placeholder, filling in 'yoyo', clicking on the password placeholder, filling in 'aa123456', and clicking the '立即登录 >' button. The code ends with a blank line.

```
1 from playwright.sync_api import Page, expect
2
3
4 def test_example(page: Page) -> None:
5     page.goto("http://localhost:3000/login.html")
6     page.get_by_placeholder("请输入用户名").click()
7     page.get_by_placeholder("请输入用户名").fill("yoyo")
8     page.get_by_placeholder("请输入密码").click()
9     page.get_by_placeholder("请输入密码").fill("aa123456")
10    page.get_by_role("button", name="立即登录 >").click()
11
```

```
from playwright.sync_api import Page, expect
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def test_example(page: Page) -> None:
    page.goto("http://127.0.0.1:8000/login.html")
    page.get_by_placeholder("请输入用户名").click()
    page.get_by_placeholder("请输入用户名").fill("yoyo")
    page.get_by_placeholder("请输入密码").click()
    page.get_by_placeholder("请输入密码").fill("aa123456")
    page.get_by_role("button", name="立即登录 >").click()
```

1.5.2 录制相关命令操作

相关命令参数：

1.codegen在浏览器中运行并执行操作

```
playwright codegen playwright.dev
```

2.Playwright 打开一个浏览器窗口，其视口设置为特定的宽度和高度，并且没有响应，因为需要在相同条件下运行测试。

使用该--viewport选项生成具有不同视口大小的测试。

```
playwright codegen --viewport-size=800,600 playwright.dev
```

3. --device 使用设置视口大小和用户代理等选项模拟移动设备时记录脚本和测试。

模拟移动设备iPhone11，注意：device的值必须用双引号，并且区分大小写

```
playwright codegen --device="iPhone 11" playwright.dev
```

4.模拟配色

```
playwright codegen --color-scheme=dark playwright.dev
```

5.模拟地理位置、语言和时区

```
playwright codegen --timezone="Europe/Rome" --geolocation="41.890221,12.492348" --lang="it-IT" maps.google.com
```

6.保留经过身份验证的状态

运行codegen以在会话结束时--save-storage保存cookie和localStorage。这对于单独记录身份验证步骤并在稍后的测试中重用它很有用。

执行身份验证并关闭浏览器后，auth.json将包含存储状态。

```
playwright codegen --save-storage=auth.json
```

运行--load-storage以消耗先前加载的存储。这样，所有的cookie和localStorage都将被恢复，使大多数网络应用程序进入身份验证状态。

```
playwright open --load-storage=auth.json my.web.app
playwright codegen --load-storage=auth.json my.web.app
# Perform actions in authenticated state.
```

1.5.3 page.pause() 断点调试

如果您想在某些非标准设置中使用codegen（例如，使用browser_context.route()），可以调用page.pause()，这将打开一个带有codegen控件的单独窗口。

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    # Make sure to run headed.
    browser = p.chromium.launch(headless=False)

    # Setup context however you like.
    context = browser.new_context() # Pass any options
    context.route('**/*', lambda route: route.continue_())

    # Pause the page, and start recording manually.
    page = context.new_page()
    page.pause()
```

1.6 pause断点 inspector 使用

在运行selenium脚本的时候，我们通常习惯用sleep去让页面暂停，打开console 输入

`$(selector)` 或者 `$x(xpath)` 去调试定位页面的元素。

有时候明明页面能找到元素，代码运行却找不到，很是郁闷！

playwright 的 `page.pause()` 断点功能出现, 让打开可以愉快的在页面上调试了, 我们甚至可以直接使用 `playwright.$(selector)` 直接支持playwright选择器的方法。

1.6.1 page.pause() 断点

在代码中加入 `page.pause()` 进入断点状态

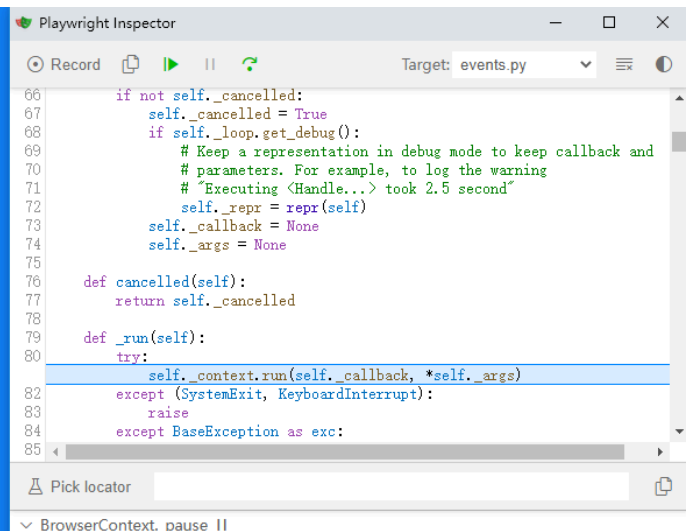
```
from playwright.sync_api import Playwright, sync_playwright, expect
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def run(playwright: Playwright) -> None:
    browser = playwright.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto("http://127.0.0.1:8000/login.html")
    page.get_by_placeholder("请输入用户名").click()
    page.get_by_placeholder("请输入用户名").fill("yoyo")
    page.get_by_placeholder("请输入密码").click()
    page.get_by_placeholder("请输入密码").fill("aa123456")
    page.pause() # 断点
    page.get_by_role("button", name="立即登录 >").click()

    # -----
    context.close()
    browser.close()

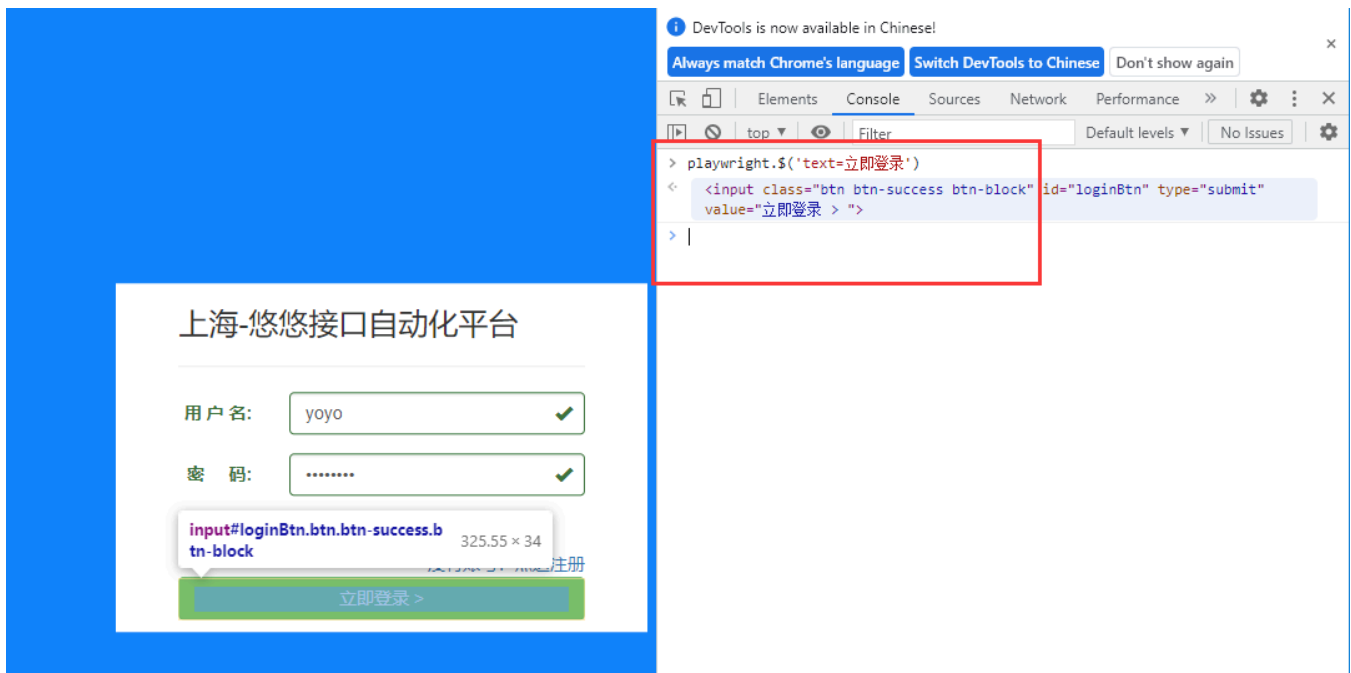
with sync_playwright() as playwright:
    run(playwright)
```

运行后会弹出 `playwright inspector` 工具



1.6.2 console 调试定位

我们可以在 用代码打开的浏览器上f12 打开console页面，输入 `playwright.$(selector)` 调试定位



selector 语法可以支持 playwright 的selector 定位的语法。

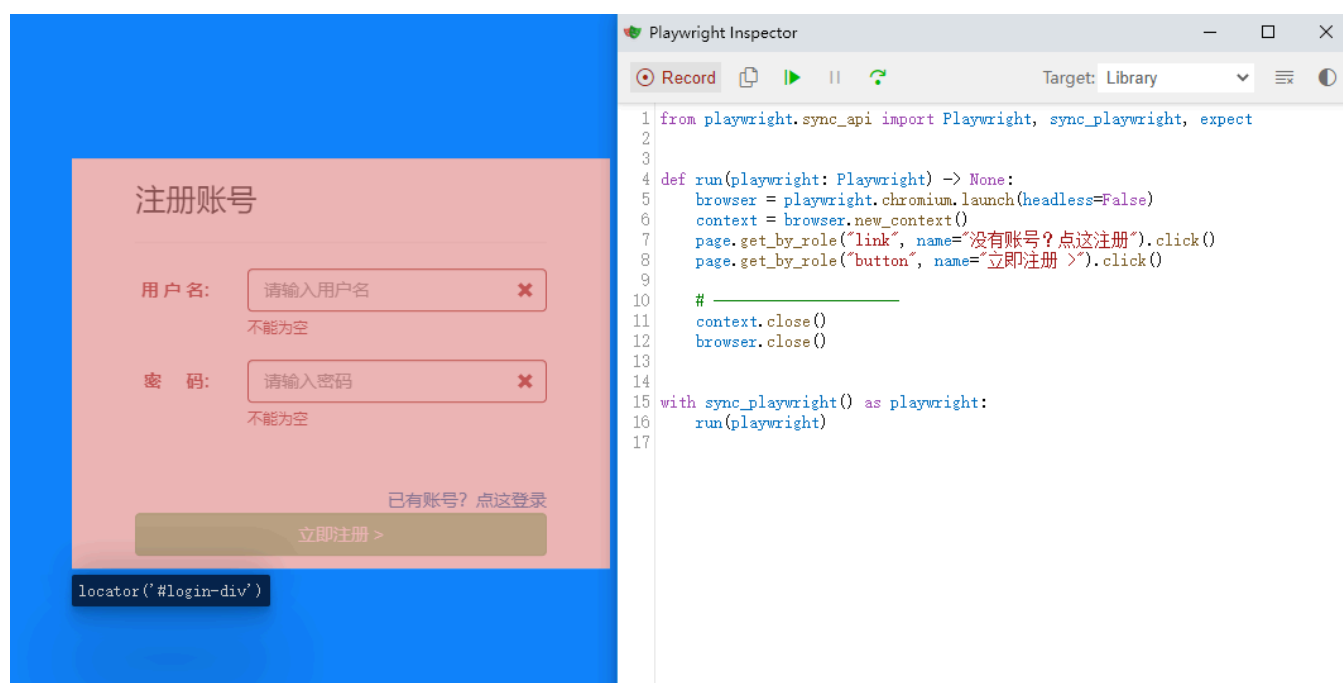
其它相关操作

- `playwright.$(selector)` 使用实际的Playwright查询引擎查询Playwright选择器

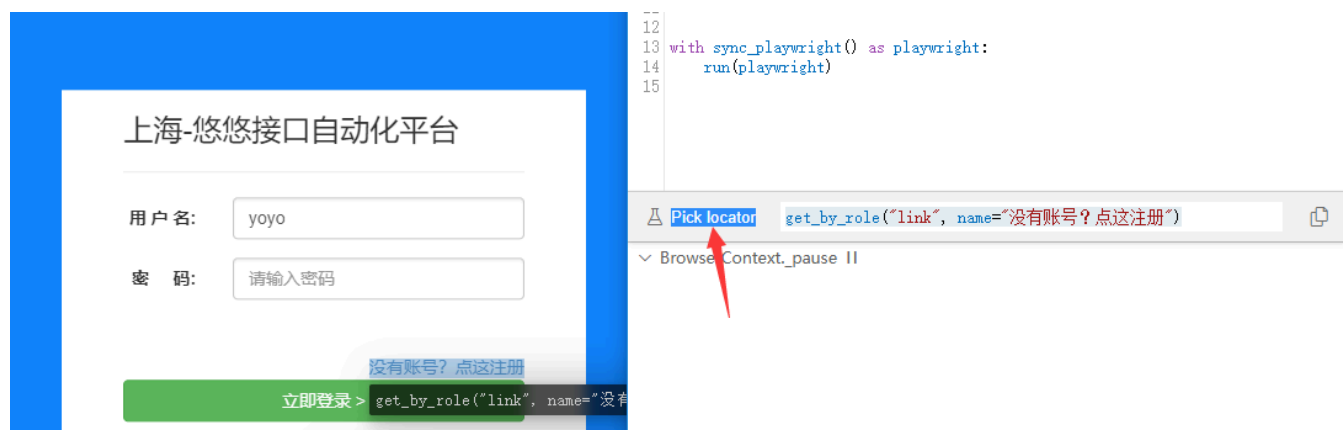
- `playwright.$$selector` 类似于 `playwright.$` ,但是返回全部的匹配元素
- `playwright.inspect(selector)` 在元素面板中显示元素（如果相应浏览器的 DevTools支持）。
- `playwright.locator(selector)` 使用实际的Playwright查询引擎查询Playwright元素
- `playwright.selector(element)` 为给定元素生成选择器。

1.6.3 playwright inspector 使用

还可以在playwright inspector 工具上点开启录制按钮，在页面上点点点，就可以生成对应的元素和操作



Pick locator 使用



点击 Pick locator 后在浏览器上选择需要定位的元素，即可生成对应的 locator

1.7 Playwright+Pytest+Allure POM项目

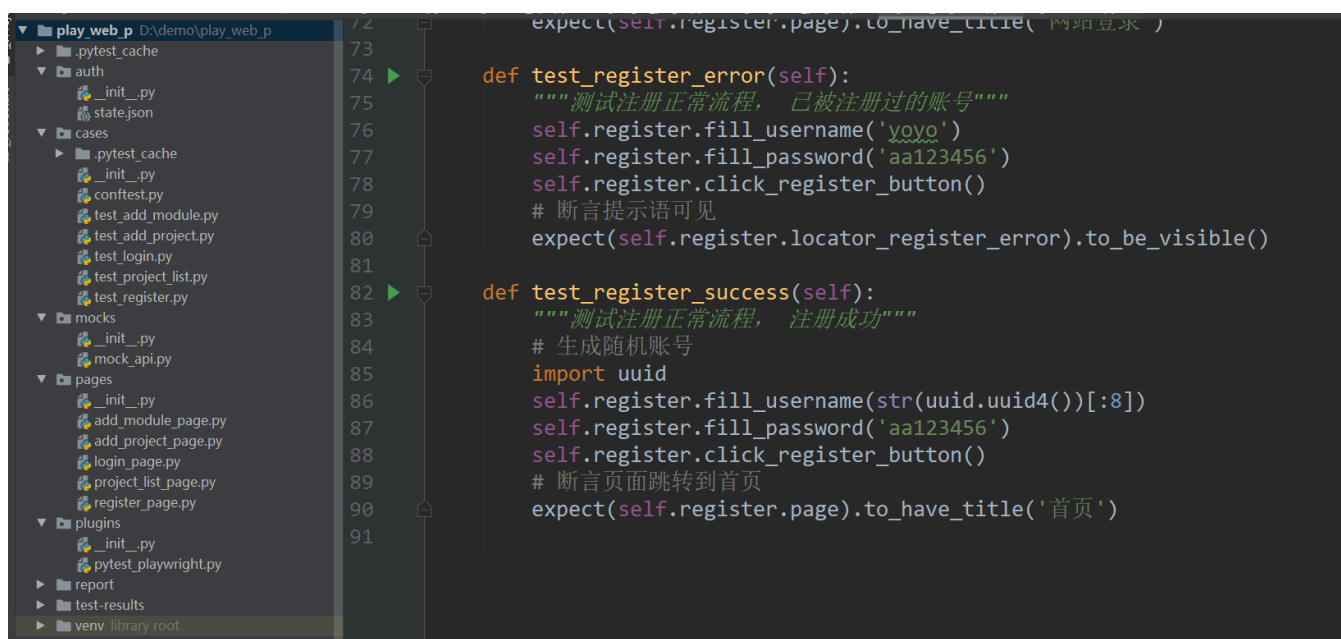
1.7.1 POM 设计模式

POM(Page Object Models) 页面对象模型已经成了写 web 自动化的一个标准模型。

页面对象代表 Web 应用程序的一部分。电子商务 Web 应用程序可能有一个主页、一个列表页面和一个结帐页面。它们中的每一个都可以由页面对象模型表示。

页面对象通过创建适合您的应用程序的更高级别的 API 来简化创作，并通过在一个地方捕获元素选择器和创建可重用代码来避免重复来简化维护。

采用传统的pom设计模型



```
72 expect(self.register.page).to_have_title('网站登录')
73
74 def test_register_error(self):
75     """测试注册正常流程， 已被注册过的账号"""
76     self.register.fill_username('yoyo')
77     self.register.fill_password('aa123456')
78     self.register.click_register_button()
79     # 断言提示语可见
80     expect(self.register.locator_register_error).to_be_visible()
81
82 def test_register_success(self):
83     """测试注册正常流程， 注册成功"""
84     # 生成随机账号
85     import uuid
86     self.register.fill_username(str(uuid.uuid4())[:8])
87     self.register.fill_password('aa123456')
88     self.register.click_register_button()
89     # 断言页面跳转到首页
90     expect(self.register.page).to_have_title('首页')
91
```

页面封装元素定位和操作方法


```

class RegisterPage:

    def __init__(self, page: Page):
        self.page = page
        self.locator_username = page.get_by_label("用户名:")
        self.locator_password = page.get_by_label("密码:")
        self.locator_register_btn = page.locator('text=立即注册')
        self.locator_login_link = page.locator('text=已有账号? 点这登录')
        # 用户名输入框提示语
        self.locator_username_tip1 = page.locator('[data-fv-validator="notEmpty"][data-')
        self.locator_username_tip2 = page.locator('[data-fv-validator="stringLength"][data-')
        self.locator_username_tip3 = page.locator('[data-fv-validator="regex"][data-')
        # 密码输入框提示语
        self.locator_password_tip1 = page.locator('[data-fv-validator="notEmpty"][data-')
        self.locator_password_tip2 = page.locator('[data-fv-validator="stringLength"][data-')
        self.locator_password_tip3 = page.locator('[data-fv-validator="regex"][data-')
        # 账号或密码不正确!
        self.locator_register_error = page.locator('text=用户名已存在或不合法! ')

```

```

def navigate(self):
    with allure.step("导航到注册页"):
        self.page.goto("/register.html")

def fill_username(self, username):
    with allure.step(f"输入用户名:{username}"):
        self.locator_username.fill(username)

def fill_password(self, password):
    with allure.step(f"输入密码:{password}"):
        self.locator_password.fill(password)

def click_register_button(self):
    with allure.step(f"点注册按钮"):
        self.locator_register_btn.click()

def click_login_link(self):
    with allure.step(f"点登录链接"):
        self.locator_login_link.click()

```

用例设计

```

2   from playwright.sync_api import expect
3   import pytest
4
5
6   class TestRegister:
7       """注册功能"""
8
9       @pytest.fixture(autouse=True)
10      def start_for_each(self, page):
11          print("for each--start: 打开新页面访问注册页")
12          self.register = RegisterPage(page)
13          self.register.navigate()
14          yield
15          print("for each--end: 后置操作")
16
17      def test_register_1(self):
18          """用户名为空, 点注册"""
19          self.register.fill_username('')
20          self.register.fill_password('123456')
21          self.register.click_register_button()
22          # 断言
23          expect(self.register.locator_username_tip1).to_be_visible()
24          expect(self.register.locator_username_tip1).to_contain_text("不能为空")

```

```

def test_register_error(self):
    """测试注册正常流程, 已被注册过的账号"""
    self.register.fill_username('yoyo')
    self.register.fill_password('aa123456')
    self.register.click_register_button()
    # 断言提示语可见
    expect(self.register.locator_register_error).to_be_visible()

def test_register_success(self):
    """测试注册正常流程, 注册成功"""
    # 生成随机账号
    import uuid
    self.register.fill_username(str(uuid.uuid4())[:8])
    self.register.fill_password('aa123456')
    self.register.click_register_button()
    # 断言页面跳转到首页
    expect(self.register.page).to_have_title('首页')

```

1.7.2 运行用例

使用pytest 命令行运行用例

```
>pytest --alluredir ./report

===== test session starts =====
platform win32 -- Python 3.8.5, pytest-7.2.1, pluggy-1.0.0
baseurl: http://47.108.155.10
rootdir: D:\demo\play_web_p, configfile: pytest.ini
plugins: allure-pytest-2.13.0, base-url-2.0.0
collected 41 items
```

查看报告

```
allure serve ./report
```

Allure

总览

类别

测试套

图表

时间刻度

功能

包

功能

order名称用时状态通过用例状态过滤: 104000Marks: 104000

注册功能

添加模块

添加项目

登录功能

项目列表

注册功能

添加模块

添加项目

登录功能

项目列表

cases.test_add_project.TestAddProject#test_add_proj

通过 异常场景-项目名称, 无效等价: 特殊

总览 历史 重试次数

优先级: normal

耗时: 179ms

描述

异常场景-项目名称, 无效等价: 特殊字符/大于30个字符

参数

app: 'aal@'

browser_name: 'chromium'

desc: ''

name: 'abc'

执行

前置

测试步骤

后置

用例失败截图和视频

order名称用时状态通过用例状态过滤104000Marks:10A

> 注册功能10

> 添加模块4

> 添加项目7

✓ #2 异常场景-项目名称不能为空'chromium'247ms

✓ #1 异常场景-项目名称, 无效等价: 特殊字符/大于30个字符", 'chromium', ", 'abcl@'171ms

✓ #6 异常场景-项目名称, 无效等价: 特殊字符/大于30个字符", 'chromium', ", 'aaaaabbbbbbccccddddddeeee... 241ms

✓ #7 异常场景-项目名称, 无效等价: 特殊字符/大于30个字符'aal@', 'chromium', ", 'abc'179ms

✓ #3 提交成功, 跳转到项目列表'chromium'532ms

✓ #5 服务器异常 500 状态码'chromium'305ms

✓ #4 项目名称重复, 弹出模态框'chromium'331ms

> 登录功能16

✗ #1 用户名为空, 点注册'chromium'186ms

✓ #2 用户名大于30字符'chromium'134ms

✓ #3 用户名有特殊字符'chromium'125ms

✓ #4 登录失败场景'chromium', '12345678', '输入错误的密码', 'yoyo'560ms

✓ #7 登录失败场景'chromium', '12345678', '输入错误的账号', 'yoyox1x2x3'580ms

✓ #6 登录正常流程, 登录成功'chromium'609ms

✓ #5 登录正常流程, 获取异步ajax 请求'chromium'413ms

> 项目列表12

✓ #1 用例描述0s

stdout159 B

AssertionError: assert 1 == 2

后置page::00scontext::0 2个附件525ms

test_login_1[chromium]-failed-120.6 KB

test_login_1[chromium]-failed-122.9 KB

测试步骤stdout159 B

后置page::00scontext::0 2个附件525ms

test_login_1[chromium]-failed-120.6 KB

test_login_1[chromium]-failed-122.9 KB

playwright::034msstart_for_each::01msbrowser::0

激活 Windows

order名称用时状态通过用例状态过滤104000Marks:10A

> 注册功能10

> 添加模块4

> 添加项目7

✓ #2 异常场景-项目名称不能为空'chromium'247ms

✓ #1 异常场景-项目名称, 无效等价: 特殊字符/大于30个字符", 'chromium', ", 'abcl@'171ms

✓ #6 异常场景-项目名称, 无效等价: 特殊字符/大于30个字符", 'chromium', ", 'aaaaabbbbbbccccddddddeeee... 241ms

✓ #7 异常场景-项目名称, 无效等价: 特殊字符/大于30个字符'aal@', 'chromium', ", 'abc'179ms

✓ #3 提交成功, 跳转到项目列表'chromium'532ms

✓ #5 服务器异常 500 状态码'chromium'305ms

✓ #4 项目名称重复, 弹出模态框'chromium'331ms

> 登录功能16

✗ #1 用户名为空, 点注册'chromium'186ms

✓ #2 用户名大于30字符'chromium'134ms

✓ #3 用户名有特殊字符'chromium'125ms

✓ #4 登录失败场景'chromium', '12345678', '输入错误的密码', 'yoyo'560ms

✓ #7 登录失败场景'chromium', '12345678', '输入错误的账号', 'yoyox1x2x3'580ms

✓ #6 登录正常流程, 登录成功'chromium'609ms

✓ #5 登录正常流程, 获取异步ajax 请求'chromium'413ms

> 项目列表12

✓ #1 用例描述0s

stdout159 B

AssertionError: assert 1 == 2

后置page::00scontext::0 2个附件525ms

test_login_1[chromium]-failed-120.6 KB

test_login_1[chromium]-failed-122.9 KB

0.01 / 0.01

playwright::034msstart_for_each::01msbrowser::0

激活 Windows

1.8 窗口最大化

playwright 默认启动的浏览器窗口大小是1280x720， 我们可以通过设置no_viewport参数来禁用固定的窗口大小

1.8.1 no_viewport 禁用窗口大小

设置args参数 `--start-maximized` 并且设置 `no_viewport=True`

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(
        headless=False,
        args=['--start-maximized']
    )
    context = browser.new_context(no_viewport=True)
    page = context.new_page()

    page.goto("https://www.cnblogs.com/yoyoketang/")
    page.pause()
```

1.8.2 viewport 指定窗口大小

viewport 参数可以设置固定的窗口

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(
        headless=False,
    )
    context = browser.new_context(
        viewport={'width': 1920, 'height': 1080}
    )
    page = context.new_page()

    page.goto("https://www.cnblogs.com/yoyoketang/")
    page.pause()
```

1.8.3 第2种方式

如果你不是通过context 上下文创建的page对象， 直接通过browser创建的page， 那么参数直接写到new_page位置

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch( headless=False)

    page = browser.new_page(
        viewport={'width': 1920, 'height': 1080}
    )

    page.goto("https://www.cnblogs.com/yoyoketang/")
    page.pause()
```

最大化

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(
        headless=False,
        args=['--start-maximized']
    )

    page = browser.new_page(no_viewport=True)

    page.goto("https://www.cnblogs.com/yoyoketang/")
    page.pause()
```

官方推荐大家使用context 上下文的方式来创建page对象。

1.9 启动 Chrome 和 Edge浏览器

playwright 默认会下载 chromium, firefox 和 webkit 三个浏览器，目前支持通过命令下载的浏览器有：chromium、chrome、chrome-beta、msedge、msedge-beta、msedge-dev、firefox、webkit

1.9.1 命令行下载

使用 `playwright install` 命令默认会安装chromium,firefox 和 webkit 三个浏览器。
可以通过 `playwright -h` 命令查看目前支持的浏览器

```
>playwright install --help
```

```
Usage: playwright install [options] [browser...]
```

ensure browsers necessary for this version of Playwright are installed

Options:

- with-deps install system dependencies for browsers
- dry-run do not execute installation, only print information
- force force reinstall of stable browser channels
- h, --help display help for command

Examples:

- \$ install
Install default browsers.

- \$ install chrome firefox

Install custom browsers, supports chromium, chrome, chrome-beta, msedge, msedge-beta, msedge-d

从命令行帮助信息中可以看到支持的浏览器有:chromium, chrome, chrome-beta, msedge, msedge-beta, msedge-dev, firefox, webkit

安装指定的浏览器, 如果本机已经安装过了, 就不会安装了

```
playwright install chrome
```

```
playwright install msedge
```

Google Chrome 或 Microsoft Edge 安装不会被隔离。它们将安装在默认的全局位置, 具体取决于您的操作系统。

如果提示已经在系统里面安装了chrome

Failed to install browsers
Error:

```
ATTENTION: "chrome" is already installed on the system!

"chrome" installation is not hermetic; installing newer version
requires *removal* of a current installation first.

To *uninstall* current version and re-install latest "chrome":

- Close all running instances of "chrome", if any
- Use "--force" to install browser:

    playwright install --force chrome

<3 Playwright Team
```

可以先关闭正在运行的chrome 浏览器，使用以下命令安装到最新版

```
playwright install --force chrome
```

安装完成会显示版本号，以及安装的位置

```
>playwright install --force chrome
```

Downloading Google Chrome

Installing Google Chrome

ProductVersion	FileVersion	FileName
-----	-----	-----
111.0.5563.65	111.0.5563.65	C:\Program Files\Google\Chrome\Application\chrome.exe

1.9.2 指定 channel 打开浏览器

默认情况下，chromium.launch() 不带 channel 参数打开的是 chromium 浏览器


```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as pw:
    browser = pw.chromium.launch(headless=False)
    page = browser.new_page()

    page.goto("https://www.baidu.com/")
```

Google Chrome 和 Microsoft Edge浏览器都是用的 chromium 内核，所以只需加一个 `channel="chrome"` 即可打开谷歌浏览器

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as pw:
    browser = pw.chromium.launch(headless=False, channel="chrome")
    page = browser.new_page()

    page.goto("https://www.baidu.com/")
```

添加 `channel="msedge"` 即可打开Microsoft Edge浏览器

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as pw:
    browser = pw.chromium.launch(headless=False, channel="msedge")
    page = browser.new_page()

    page.goto("https://www.baidu.com/")
```

1.10 启动本地 Chrome 加载用户缓存

playwright 启动Google Chrome 浏览器的时候默认用的是无痕模式，不加载本地的数据，这对于测试人员运行一个干净的浏览器是没问题的。

大家在学selenium的时候，知道 selenium 可以启动本地的 Google Chrome 浏览器并加载本地

数据，这样可以在本地已经登录过网站的情况下，下次打开网站不需要登录继续操作，对于一些爬虫用户是非常重要的功能。

playwright 可以使用`launch_persistent_context`方法启动本地的chrome 浏览器

1.10.1 selenium 加载 Google Chrome 插件

在启动浏览器的时候添加 `--user-data-dir` 用户数据目录，即可启动带插件的浏览器，并且会记住用户的cookies数据

```
from selenium import webdriver
import getpass
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

# 启用带插件的浏览器
option = webdriver.ChromeOptions()
option.add_argument(f"--user-data-dir=C:\\Users\\{getpass.getuser()}\\AppData\\Local\\Google\\Chrome\\")
driver = webdriver.Chrome(chrome_options=option) # 打开chrome浏览器
driver.get("https://www.cnblogs.com/yoyoketang/")
```

1.10.2 如何查看 `--user-data-dir` 用户数据目录

有些同学不是windows系统的电脑（有同学用的mac电脑不知道如何查看用户数据目录），`--user-data-dir` 用户数据目录可以在你的google浏览器地址栏输入

```
chrome://version
```

查看用户数据目录和安装路径

Google Chrome: 112.0.5615.121 (正式版本) (64 位) (cohort: Stable Installs & Version Pins) 

修订版本: 39cc4e45904ae9f1741c4fbba866e629c96f2268-refs/branch-heads/5615_51@{#8}

操作系统: Windows 10 Version 1803 (Build 17134.1130)

JavaScript: V8 11.2.214.14

用户代理: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36

命令行: "C:\Program Files\Google\Chrome\Application\chrome.exe" --flag-switches-begin --flag-switches-end

可执行文件路径: C:\Program Files\Google\Chrome\Application\chrome.exe

个人资料路径: C:\Users\de11\AppData\Local\Google\Chrome\User Data\Default

使用中的变体: 272b4158-ca7d8d80
f6ae519e-ca7d8d80
df319cb2-33c3eba5
857feb0-ca7d8d80
3e7d7783-377be55a
2fcbfa14-377be55a
e5726113-377be55a
3e6ca8ad-ca7d8d80
ce1fd4ce-ca7d8d80
aa21b4f1-a2746ce8
e1995fc9-377be55a
d327bc9c-ca7d8d80
9d165456-ca7d8d80
eddd0d82-7723ede9
1918042f-8046c681
c6858e09-377be55a
89a16501-ca7d8d80
42f0e0ea-ca7d8d80
29ea62bc-ca7d8d80
9785b8cf-12ede6a2
5576b741-ca7d8d80
11205616-7723ede9

1.10.3 playwright 启动本地 Google Chrome

使用 `launch_persistent_context` 方法启动本地的chrome 浏览器，并且设置 `channel="chrome"`

```

import getpass
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

# 获取 google chrome 的本地缓存文件
USER_DIR_PATH = f"C:\\Users\\{getpass.getuser()}\\AppData\\Local\\Google\\Chrome\\User Data"

with sync_playwright() as p:
    browser = p.chromium.launch_persistent_context(
        # 指定本机用户缓存地址
        user_data_dir=USER_DIR_PATH,
        # 接收下载事件
        accept_downloads=True,
        # 设置 GUI 模式
        headless=False,
        bypass_csp=True,
        slow_mo=1000,
        channel="chrome",
    )

    page = browser.new_page()
    page.goto("https://www.cnblogs.com/yoyoketang")

    page.pause()

```

在运行的时候，先关闭本地的chrome 浏览器，再执行代码，就可以看到启动的浏览器，打开网站不需要登录了，但是插件并没有加载进去。

1.10.4 配置args 参数加载插件

Google Chrome 插件加载需配置args 参数，所有的 args 参数列表可以在这个地址查询 <https://peter.sh/experiments/chromium-command-line-switches/>

<code>--disable-dev-shm-usage</code> ^[19]	The /dev/shm partition is too small in certain VM environments, causing Chrome to fail or crash (see http://crbug.com/715363). Use this flag to work-around this issue (a temporary directory will always be used to create anonymous shared memory files).
<code>--disable-device-disabling</code>	If this switch is set, the device cannot be remotely disabled by its owner.
<code>--disable-dinosaur-easter-egg</code>	Disables the dinosaur easter egg on the offline interstitial.
<code>--disable-domain-blocking-for-3d-apis</code>	Disable the per-domain blocking for 3D APIs after GPU reset. This switch is intended only for tests.
<code>--disable-domain-reliability</code>	Disables Domain Reliability Monitoring.
<code>--disable-explicit-dma-fences</code>	Disable explicit DMA-fences
<code>--disable-extensions</code>	Disable extensions.
<code>--disable-extensions-except</code>	Disable extensions except those specified in a comma-separated list.
<code>--disable-extensions-file-access-check</code>	Disable checking for user opt-in for extensions that want to inject script into file URLs (ie, always allow it). This is used during automated testing.
<code>--disable-extensions-http-throttling</code>	Disable the net::URLRequestThrottlerManager functionality for requests originating from extensions.
<code>--disable-features</code>	Comma-separated list of feature names to disable. See also <code>kEnableFeatures</code> .
<code>--disable-fetching-hints-at-navigation-start</code>	No description
<code>--disable-field-trial-config</code>	Disable field trial tests configured in <code>fieldtrial_testing_config.json</code> .
<code>--disable-file-system</code>	Disable FileSystem API.
<code>--disable-fine-grained-time-zone-detection</code>	Disables fine grained time zone detection.

在使用脚本加载扩展插件时，一定要解压crx文件，不要直接安装crx

```

import getpass
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

# 获取 google chrome 的本地缓存文件
USER_DIR_PATH = f"C:\\Users\\{getpass.getuser()}\\AppData\\Local\\Google\\Chrome\\User Data"
# chrome.exe指定可执行文件路径
# executable_path = "chromium-111/chrome-win/chrome.exe"

# chrome插件目录, 下载后解压crx
path_to_extension = f"\\Users\\Downloads\\extension\\xxx"

with sync_playwright() as p:
    browser = p.chromium.launch_persistent_context(
        # 指定本机用户缓存地址
        user_data_dir=USER_DIR_PATH,
        # executable_path=executable_path, # 如果有需要可以下载chrome.exe到指定目录
        # 接收下载事件
        accept_downloads=True,
        # 设置 GUI 模式
        headless=False,
        bypass_csp=True,
        slow_mo=1000,
        channel="chrome",
        args=[
            f"--disable-extensions-except={path_to_extension}",
            f"--load-extension={path_to_extension}"
        ], # 加载扩展插件

    )

```

扩展程序仅在 Chrome / Chromium GUI模式中使用。

1.10.5 使用代理

使用代理可以在launch_persistent_context 加上proxy 参数

```

proxy=ProxySettings(server="http://xxx.xxx.xxx.xxx:xxxx"),

```

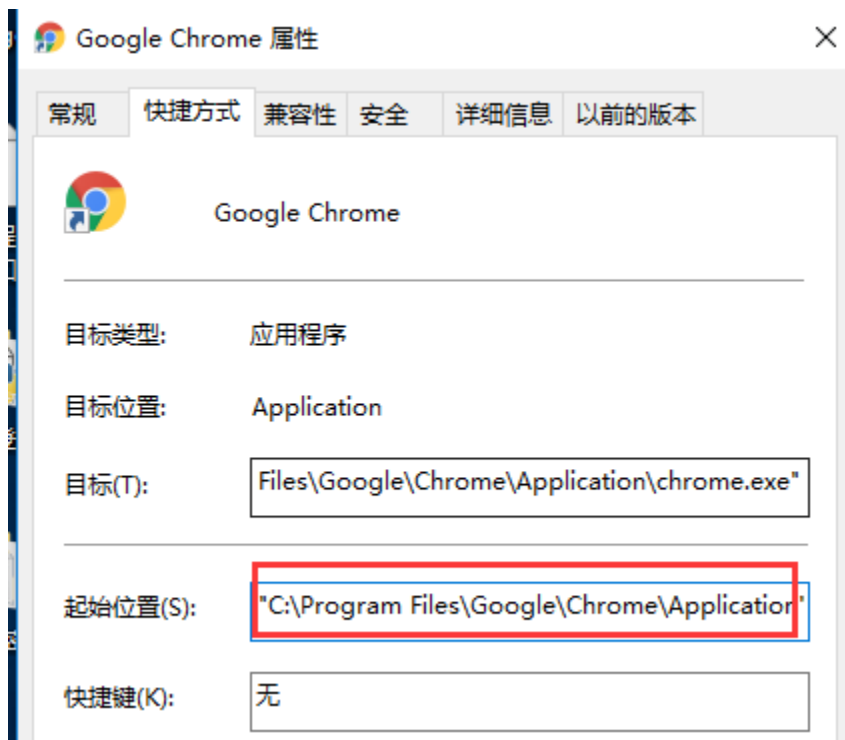
1.11 操作已打开浏览器，绕过登录验证码

有些网站的安全级别比较高，你可能想绕过登录验证，先手工打开浏览器操作登录，让 playwright 继续在你已经打开的浏览器上操作。

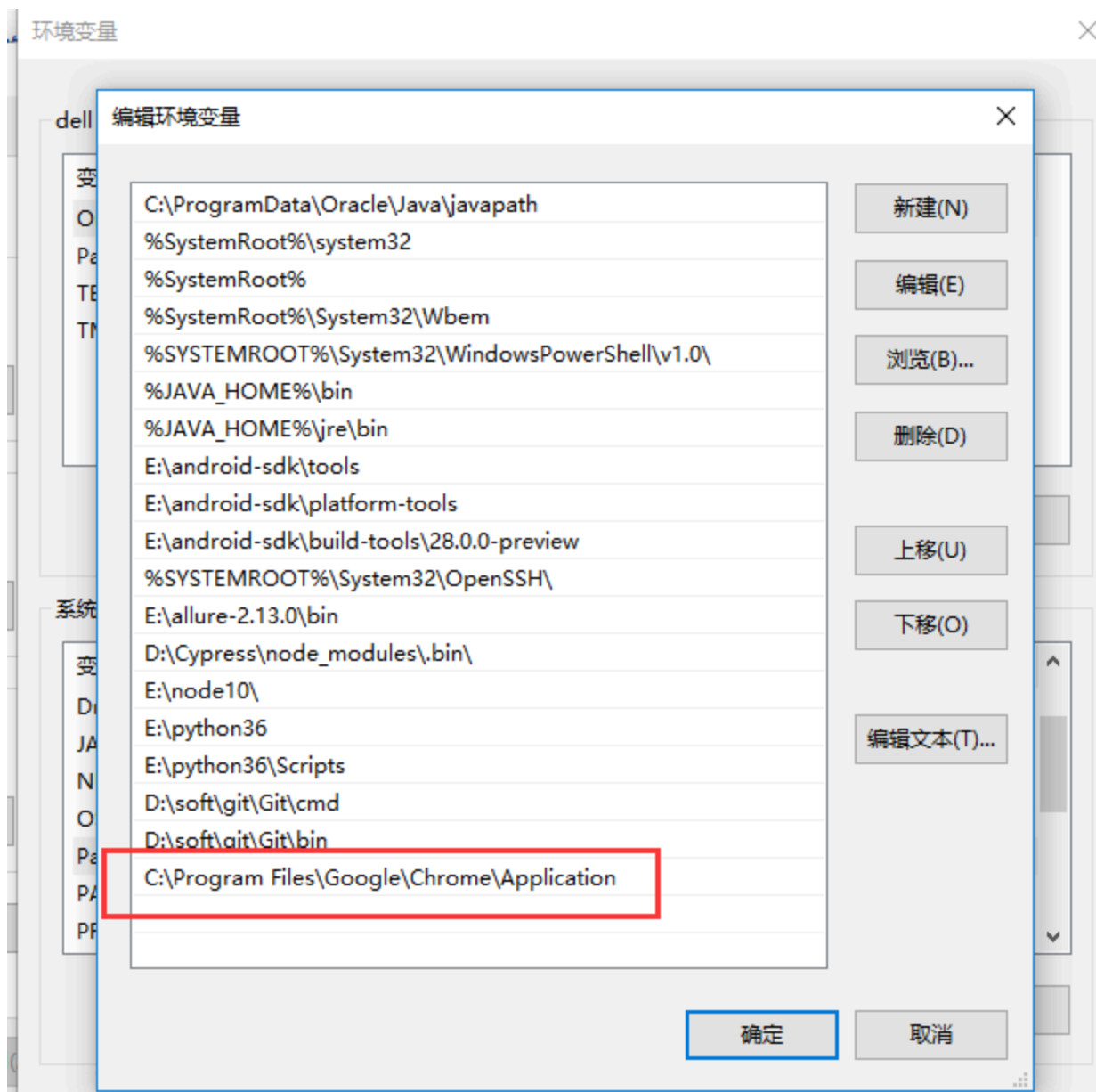
selenium 是可以做到这点，那么 playwright 也可以实现此功能。

1.11.1 环境准备

首先右键 Chrome 浏览器桌面图标，找到 chrome.exe 的安装路径



复制地址 `C:\Program Files\Google\Chrome\Application` 添加到环境变量Path下



打开cmd输入命令启动chrome浏览器

- --remote-debugging-port 是指定运行端口，只要没被占用就行
- --user-data-dir 指定运行浏览器的运行数据，新建一个干净目录，不影响系统原来的数据

```
> chrome.exe --remote-debugging-port=12345 --user-data-dir="D:\playwright_chrome"
```

执行后会启动chrome浏览器



1.11.2 参数配置

在启动浏览器的时候，我们还可以带上一些其它参数

- `--incognito` 隐私模式打开
- `--start-maximized` : 窗口最大化
- `--new-window` : 直接打开网址

使用示例，用隐私模式打开我的博客

```
chrome.exe --remote-debugging-port=12345 --incognito --start-maximized --user-data-dir="D:\demo" --new-window https
```



在你已经打开的浏览器页面，手工操作登录，登录成功后，让playwright 继续操作。

1.11.3 playwright 接管页面

当页面打开后，可以使用connect_over_cdp()方法接管前面已经打开的浏览器，获取到context上下文，通过上下文再获取到page对象

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.connect_over_cdp('http://localhost:12345/')
    # 获取page对象
    page = browser.contexts[0].pages[0]
    print(page.url)
    print(page.title())
    page.get_by_text('新随笔').click()
```

后面的操作，就跟操作page方法一样了

1.12 登录页面滑动解锁

登录页面会遇到滑块解锁，滑动解锁的目的就是为了防止别人用代码登录（也就是为了防止你自动化登录），有些滑动解锁是需要去拼图这种会难一点。

有些直接拖到最最右侧就可以了，本篇讲下最简单的直接滑动最右侧的滑块解锁。

1.12.1 滑动解锁场景

看下图，是我本地写的一个 slider.html 网页



除了输入账号和密码，还需将滑块拖动到最右端才能解锁



最后才去点登陆按钮

1.12.2 操作滑块

操作滑块，需用到mouse 方法

- mouse.move() 起点或终点坐标位置
- mouse.down() 按住鼠标

- `mouse.up()` 释放鼠标

示例代码

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    context = browser.new_context()
    page = context.new_page()

    page.goto('file:///C:/Users/dell/Desktop/slider.html')

    # 滑块
    slider = page.locator('.slider').bounding_box()
    page.mouse.move(x=slider['x'], y=slider['y']+slider['height']/2)
    page.mouse.down() # 按住
    page.mouse.move(x=slider['x']+240, y=slider['y']+slider['height']/2)
    page.mouse.up() # 释放
    page.pause()
```

1.13 离线安装 playwright 环境

有些同学可能是在公司局域网办公，无法连到外网去在线下载，本篇教大家在本地图域网部署好 playwright 环境

1.13.1 playwright 本地下载

先找个有网络的电脑，下载 playwright，不要去 pypi 库单独下载这一个包，它在安装过程中还会下载其他依赖包。

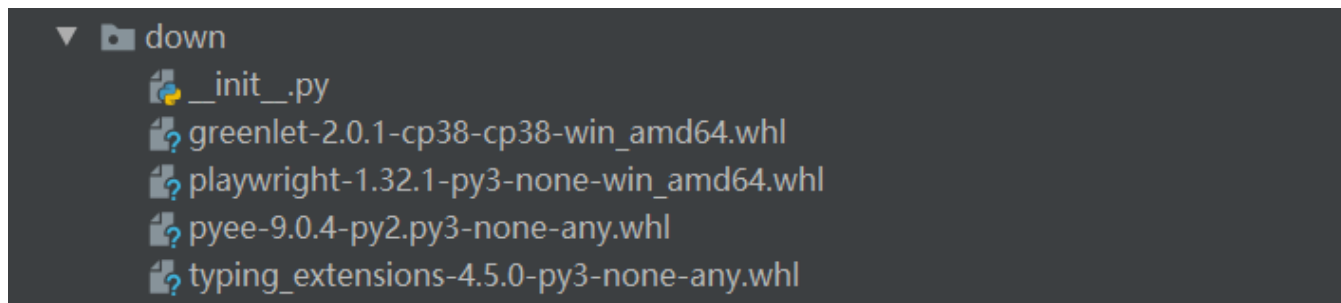
如果你只下载一个 `playwright-1.32.1-py3-none-win_amd64.whl`，然后 pip 安装肯定会失败。

正确的方法是通过 `pip download playwright` 下载安装包

```
pip download playwright -d ./down
```

-d 参数是指定下载安装到本地 down 目录

正常情况下会下载以下四个包



如果你还需要有其他的包需要下载，也可以按上面方式下载，比如pytest，allure-pytest 等包

1.13.2 多个包批量下载

如果你本地已经安装了一些依赖包了，可以通过 `pip freeze > requirements.txt`，导出本地的全部依赖包到requirements.txt文件

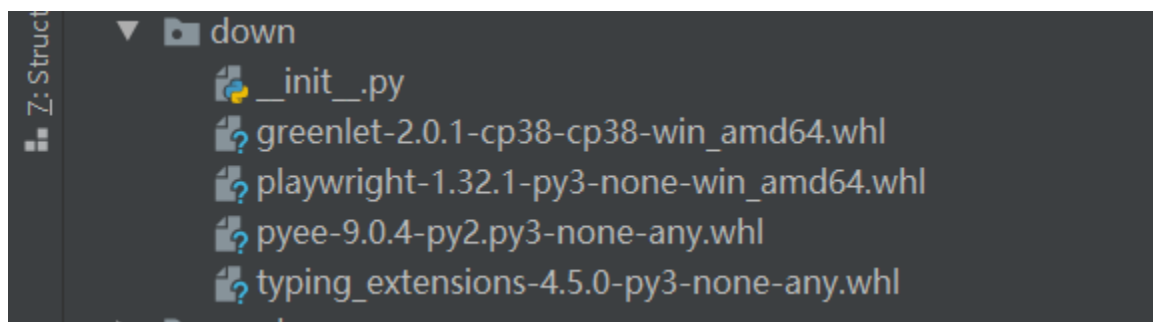
```
pip freeze > requirements.txt
```

requirements.txt 内容格式如下

```
greenlet==2.0.1
playwright==1.32.1
pyee==9.0.4
typing-extensions==4.5.0
```

再通过download命令下载全部

```
pip download -r requirements.txt -d ./down
```

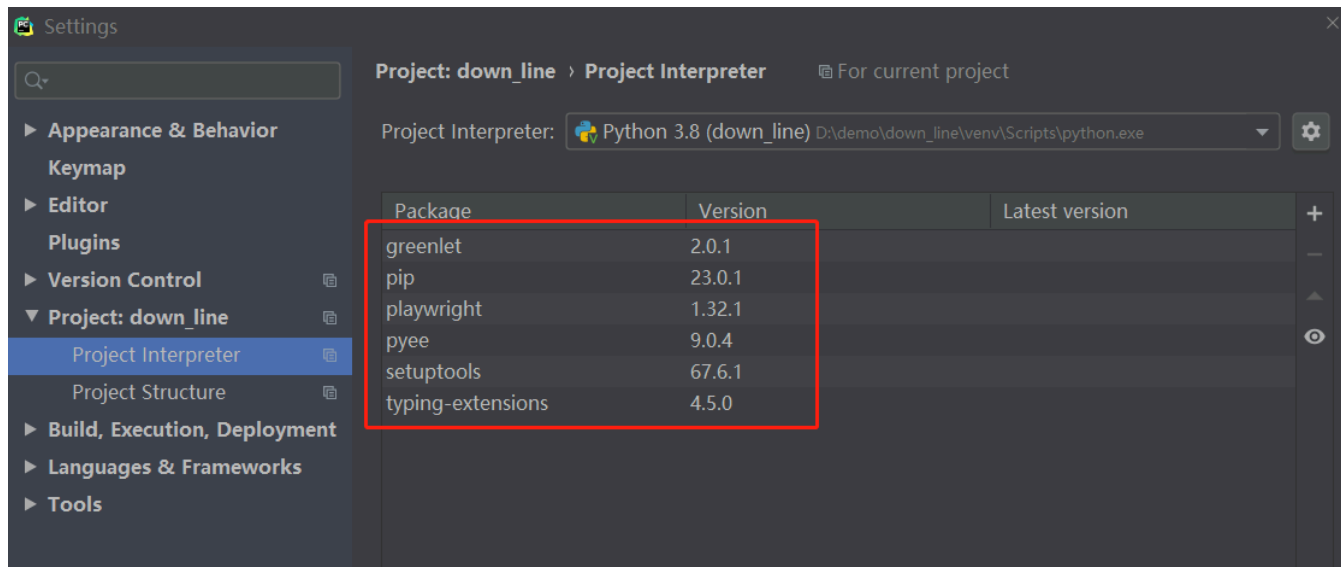


1.13.3 离线安装

你把前面下载的down文件下全部包，以及requirements.txt文件上传到你局域网的电脑上。
本地离线批量安装依赖包

```
pip install --no-index --find-links=./ -r requirements.txt
```

只要你 requirements.txt 文件的包和下载的包是一一对应的，上面的安装就不会报错了。



到这一步playwright 的基本环境就已经安装完成了。

1.13.4 启动本地 chrome 浏览器

如果你仅仅只需要在chrome浏览器上运行你的代码，那么是不需要执行 `playwright install` 下载 chromium,firefox 和 webkit。

首先你确保在你自己本机电脑上安装了chrome浏览器，并且安装是按默认的安装路径
那么在启动的时候，只需指定 `channel='chrome'` 就可以启动本地chrome 浏览器了。

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(channel='chrome', headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto('http://xxx.xx.com')
```

如果遇到以下报错，说明你浏览器没正确安装,重新安装一次chrome浏览器，按默认的路径安装即可。

```
playwright._impl._api_types.Error: Chromium distribution 'chrome' is not found at C:\Users\dell\AppData\Local\Google\Chrome\Application\chrome.exe
Run "playwright install chrome"
```

1.13.5 离线安装chromium,firefox 和 webkit

如果你有安装chromium,firefox 和 webkit 这3个官方提供的内置浏览器的需求，那么接着往下看。

我们先看去哪里下载到这3个浏览器，在终端执行以下命令

```
>playwright install --dry-run
```

它会根据你当前安装的playwright（我当前版本是1.32.1）版本，给出对应的浏览器最近匹配版本，以及下载地址

```
(venv) D:\demo\down_line\down>playwright install --dry-run
browser: chromium version 112.0.5615.29
  Install location:      C:\Users\dell\AppData\Local\ms-playwright\chromium-1055
  Download url:          https://playwright.azureedge.net/builds/chromium/1055/chromium-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/chromium/1055/chromium-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/chromium/1055/chromium-win64.zip

browser: firefox version 111.0
  Install location:      C:\Users\dell\AppData\Local\ms-playwright\firefox-1391
  Download url:          https://playwright.azureedge.net/builds/firefox/1391/firefox-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/firefox/1391/firefox-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/firefox/1391/firefox-win64.zip

browser: webkit version 16.4
  Install location:      C:\Users\dell\AppData\Local\ms-playwright\webkit-1811
  Download url:          https://playwright.azureedge.net/builds/webkit/1811/webkit-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/webkit/1811/webkit-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/webkit/1811/webkit-win64.zip

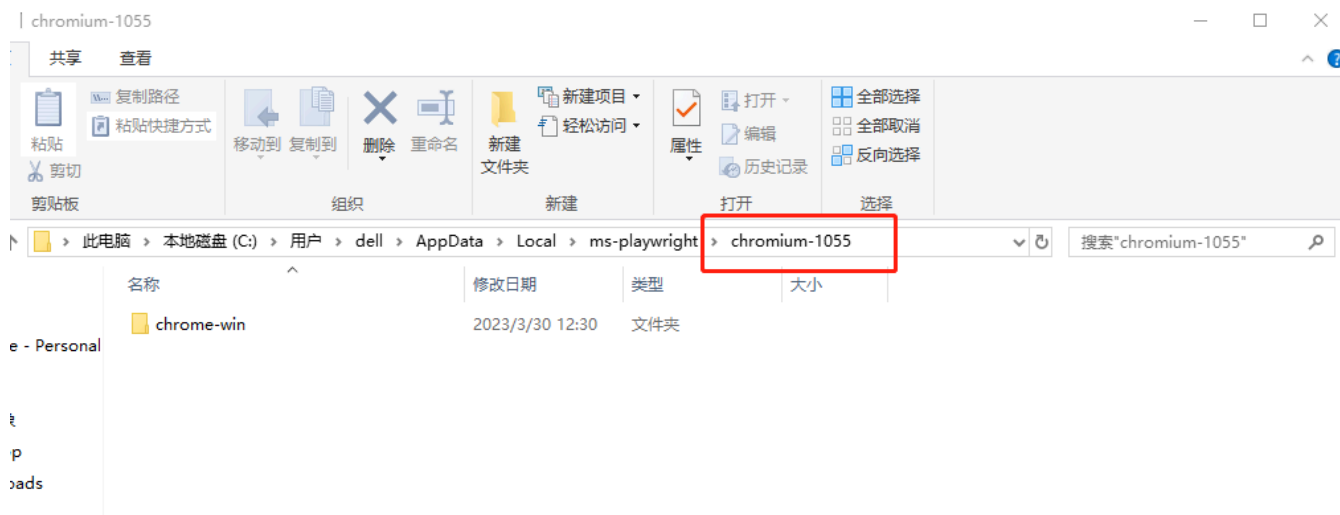
browser: ffmpeg
  Install location:      C:\Users\dell\AppData\Local\ms-playwright\ffmpeg-1008
  Download url:          https://playwright.azureedge.net/builds/ffmpeg/1008/ffmpeg-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/ffmpeg/1008/ffmpeg-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/ffmpeg/1008/ffmpeg-win64.zip
```

以 chromium 安装为例，先下载<https://playwright.azureedge.net/builds/chromium/1055/chromium-win64.zip>

下载后是一个chromium-win64.zip压缩包。

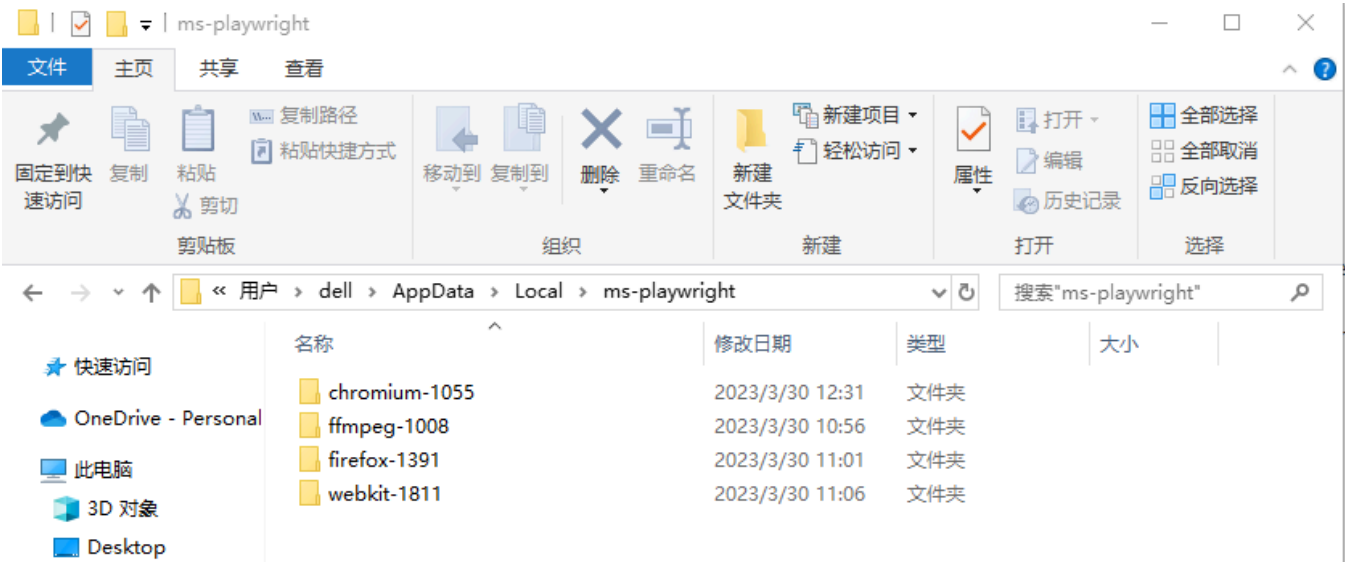
接着看Install location 安装位置：`C:\Users\dell\AppData\Local\ms-playwright\chromium-1055`

按照这个路径依次创建文件夹，把压缩包放到chromium-1055下解压即可



还有个 ffmpeg 包也需要按上面的路径正确解压，此包跟录制视频有关。
这样你本地就有了对应的chromium,firefox 和 webkit 环境。

在ms-playwright 目录下有以下四个文件



1.14 Pyinstaller 打包生成独立的可执行文件

playwright 与 Pyinstaller 结合使用来创建独立的可执行文件。

1.14.1 打成一个exe的独立包

有同学提到说想打成一个exe的独立包，但是执行 `playwright install` 会默认把 chromium,firefox 和 webkit 三个浏览器安装到系统目录。
这样打包的时候就找不到启动的浏览器文件。于是就想到把浏览器文件下载到我们的代码的项目目录，打到一起。

在playwright 官方文档中有提到相关资料：
您可以将 Playwright 与Pyinstaller结合使用来创建独立的可执行文件。

```
# main.py
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
    page = browser.new_page()
    page.goto("http://whatsmyuseragent.org/")
    page.screenshot(path="example.png")
    browser.close()
```

如果你想将浏览器与可执行文件捆绑在一起：

```
set PLAYWRIGHT_BROWSERS_PATH=0
playwright install chromium
pyinstaller -F main.py
```

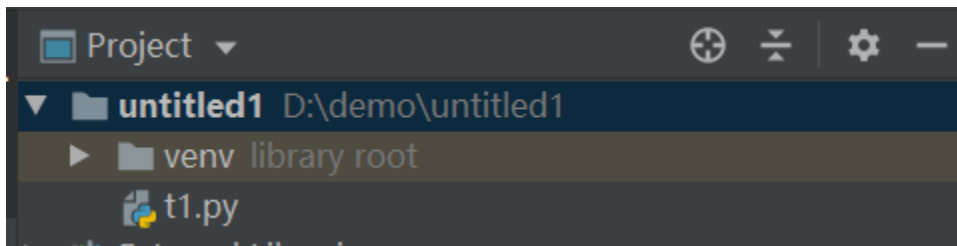
将浏览器与可执行文件捆绑在一起将生成更大的二进制文件。建议只捆绑您使用的浏览器。

上面这段就是官方文档提到的，资料比较少，接下来详细讲解

下 `set PLAYWRIGHT_BROWSERS_PATH=0` 的作用

1.14.2 chromium 本地化安装

我们可以先创建一个虚拟环境



在虚拟环境下，安装你需要的包

```
pip install playwright
```

先设置环境变量

```
set PLAYWRIGHT_BROWSERS_PATH=0
```

设置参数 `PLAYWRIGHT_BROWSERS_PATH` 默认设置为 0，意思是安装到site-packages下的

playwright目录

当然你也可以设置成其他位置，比如你指定安装到你代码的跟目录，得使用路径的绝对路径

设置完成后使用 `playwright install --dry-run` 查看浏览器安装路径

```
(venv) D:\demo\untitled1>playwright install --dry-run
browser: chromium version 112.0.5615.29
  Install location:      d:\demo\untitled1\venv\Lib\site-packages\playwright\driver\package\.local-browsers\chromium-112.0.5615.29-win64
  Download url:          https://playwright.azureedge.net/builds/chromium/1055/chromium-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/chromium/1055/chromium-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/chromium/1055/chromium-win64.zip

browser: firefox version 111.0
  Install location:      d:\demo\untitled1\venv\Lib\site-packages\playwright\driver\package\.local-browsers\firefox-111.0-win64
  Download url:          https://playwright.azureedge.net/builds/firefox/1391/firefox-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/firefox/1391/firefox-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/firefox/1391/firefox-win64.zip

browser: webkit version 16.4
  Install location:      d:\demo\untitled1\venv\Lib\site-packages\playwright\driver\package\.local-browsers\webkit-1811-win64
  Download url:          https://playwright.azureedge.net/builds/webkit/1811/webkit-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/webkit/1811/webkit-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/webkit/1811/webkit-win64.zip

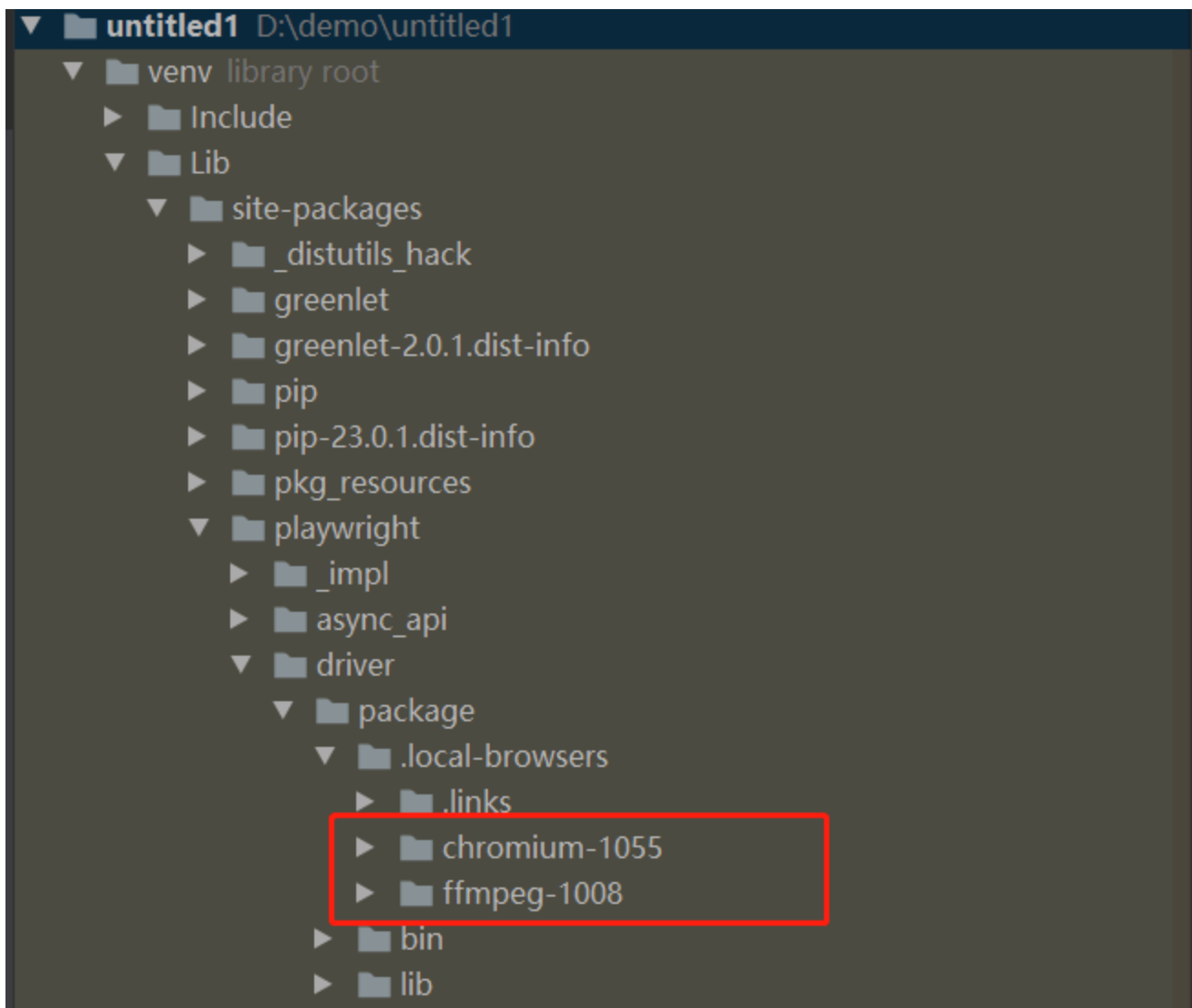
browser: ffmpeg
  Install location:      d:\demo\untitled1\venv\Lib\site-packages\playwright\driver\package\.local-browsers\ffmpeg-1008-win64
  Download url:          https://playwright.azureedge.net/builds/ffmpeg/1008/ffmpeg-win64.zip
  Download fallback 1:   https://playwright-akamai.azureedge.net/builds/ffmpeg/1008/ffmpeg-win64.zip
  Download fallback 2:   https://playwright-verizon.azureedge.net/builds/ffmpeg/1008/ffmpeg-win64.zip
```

如果你只用一个浏览器，那么你就只下载一个，以chromium 为例，执行

```
playwright install chromium
```

于是你可以在此目录找

到 `d:\demo\untitled1\venv\Lib\site-packages\playwright\driver\package\.local-browsers\webkit-1811`



顺着路径，你就会找到chromium-1055 和 ffmpeg 这2个文件。

需注意的是chromium-1055后面的1055 是版本号，此数字不能随便改，让它自己下载匹配的版本，不能直接复制别人下载的包。

并且安装包的位置不能随便乱放，一定要是先设置环境变量 `PLAYWRIGHT_BROWSERS_PATH=0`，再去安装让它自己去下载后放到指定的目录

1.14.3 Pyinstaller打包

安装Pyinstaller

```
pip install pyinstaller
```

安装完成后执行打包

```
pyinstaller -F main.py
```

1.14.4 icon 制作

-i 参数打包的时候可以自定义 icon 图标

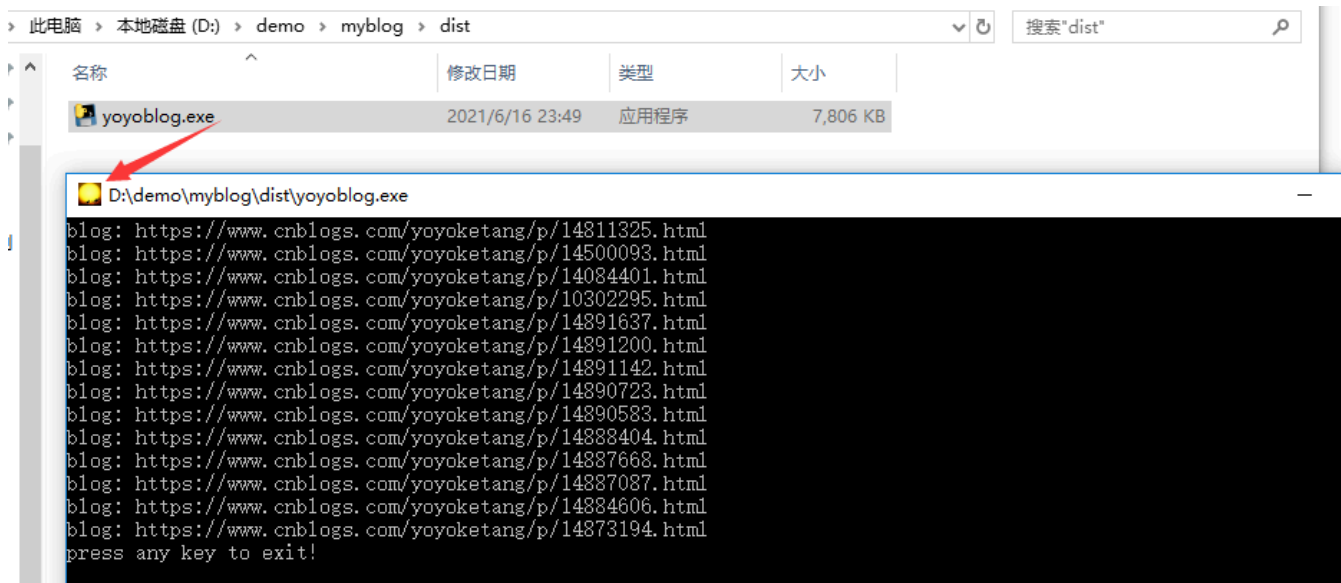
```
-i <FILE.ico or FILE.exe,ID or FILE.icns or "NONE">, --icon <FILE.ico or FILE.exe,ID or FILE.icns or "NONE">  
FILE.ico: apply that icon to a Windows executable.  
FILE.exe,ID, extract the icon with ID from an exe.  
FILE.icns: apply the icon to the .app bundle on Mac OS  
X. Use "NONE" to not apply any icon, thereby making  
the OS to show some default (default: apply  
PyInstaller's icon)
```

先找一张icon图片放到项目跟目录（注意并不是每个图片格式都可以，必须是icon格式）

加 **-i** 参数打包

```
pyinstaller -F yoyoblog.py -i favicon.ico
```

打包完成重新双击运行，会看到左上角有自己的icon了



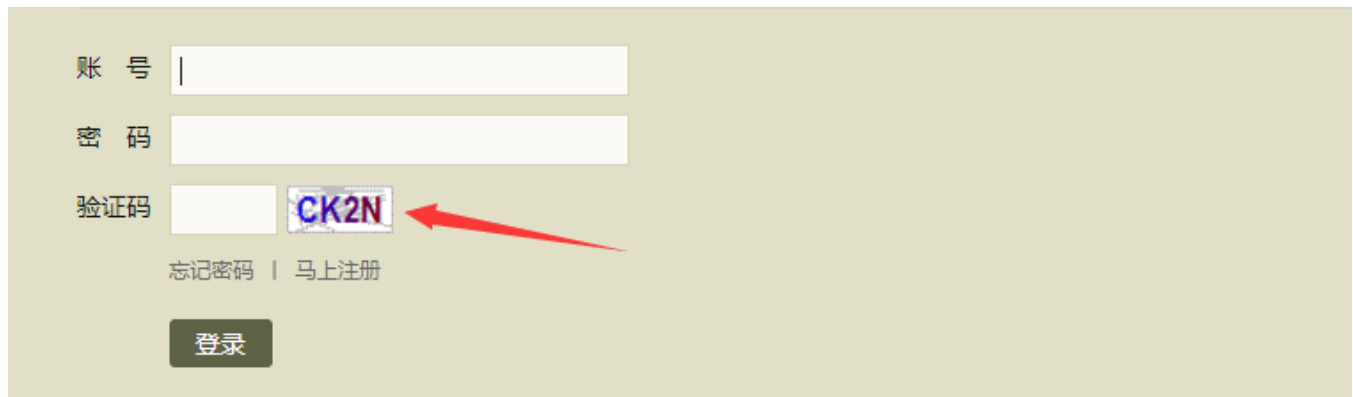
icon在线制作<https://www.bitbug.net/>

1.15 登录-验证码识别

简单的登录验证码，数字和英文组合的，可以轻松识别

1.15.1 登录验证码

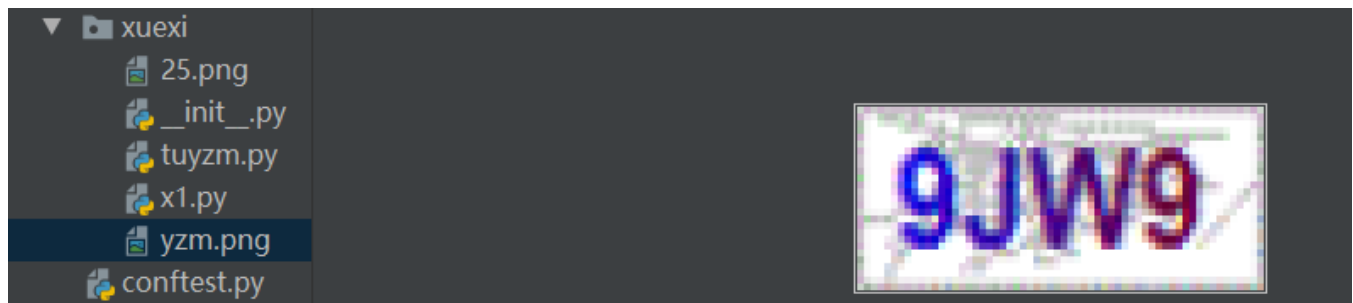
如下图登录验证码



验证码是一个图片链接，每次打开页面它会自动刷新

解决思路是先获取到验证码图片，获取验证码图片的方式，可以直接定位到img元素，对元素截图即可

```
# 保存验证码
page.locator('#imgCode').screenshot(path='yzm.png')
```



最后使用ddddocr 快速识别

```
import dddocr

# 识别验证码
ocr = dddocr.DdddOcr(show_ad=False) # 实例化
with open('yzm.png', 'rb') as f: # 打开图片
    img_bytes = f.read() # 读取图片
yzm = ocr.classification(img_bytes) # 识别
print(f'识别到的验证码: {yzm}')
```

1.15.2 代码示例

先安装dddocr

```
pip install dddocr -i https://pypi.douban.com/simple
```

完整代码

```

"""
简单的图像验证码
"""

from playwright.sync_api import sync_playwright
import ddddocr

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()

    page.goto('https://so.gushiwen.cn/user/login.aspx?from=http://so.gushiwen.cn/user/collect.aspx')
    page.locator("#email").fill('123@qq.com')
    page.locator('#pwd').fill('111111')
    # 保存验证码
    page.locator('#imgCode').screenshot(path='yzm.png')

    # 识别验证码
    ocr = ddddocr.DdddOcr(show_ad=False) # 实例化
    with open('yzm.png', 'rb') as f: # 打开图片
        img_bytes = f.read() # 读取图片
    yzm = ocr.classification(img_bytes) # 识别
    print(f'识别到的验证码: {yzm}')

    # 输入验证码
    page.locator('#code').fill(yzm)

    page.pause()

```

1.16 登录-滑块拼图验证码

有些登录页面经常会遇到滑块验证码，滑块的操作思路基本都差不多，先确定缺口的位置，再滑动过去。

一般在滑动过去的时候，会有人机识别机制，有时候你准确的滑动位置了，但不一定会解锁成功。

整体来说滑块的解决需解决以下3个问题：

- 1.得到滑块背景图
- 2.计算缺口位置
- 3.滑动轨迹（避开防爬机制）

不同的网站滑块也有区别，没有统一固定的答案

1.16.1 滑块示例

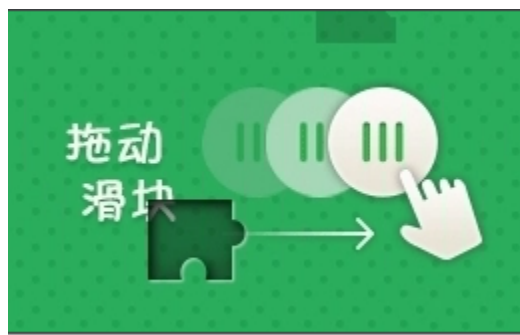
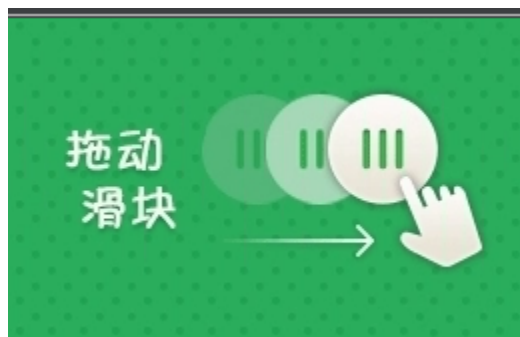
以下滑块为例



需先计算出滑块的缺口位置，也就是我们需要滑动的距离

1.16.2 计算缺口位置

计算缺口位置的方法，网上都有现成的解决方案，我们只需要得到2张图，一个是背景图，另外一个缺口图



如何得到这2个图是解决问题的关键



canvas 元素获取图片

```
# 获取 canvas 元素背景图
bg_src = page.evaluate(
    "document.getElementsByClassName('geetest_canvas_bg geetest_absolute')[0].toDataURL('image/png')")

full_src = page.evaluate(
    "document.getElementsByClassName('geetest_canvas_fullbg geetest_fade geetest_absolute')[0].toDataURL('image/png')")

# 从base64 编码中读取图片
bg_base = bg_src.split(',')[1]
bg_base = base64.b64decode(bg_base)
with open('bg.jpg', 'wb') as f:
    f.write(bg_base)

full_base = full_src.split(',')[1]
full_base = base64.b64decode(full_base)
with open('full.jpg', 'wb') as f:
    f.write(full_base)
```

通过上面2张图，我们就可以计算出缺口的位置了

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

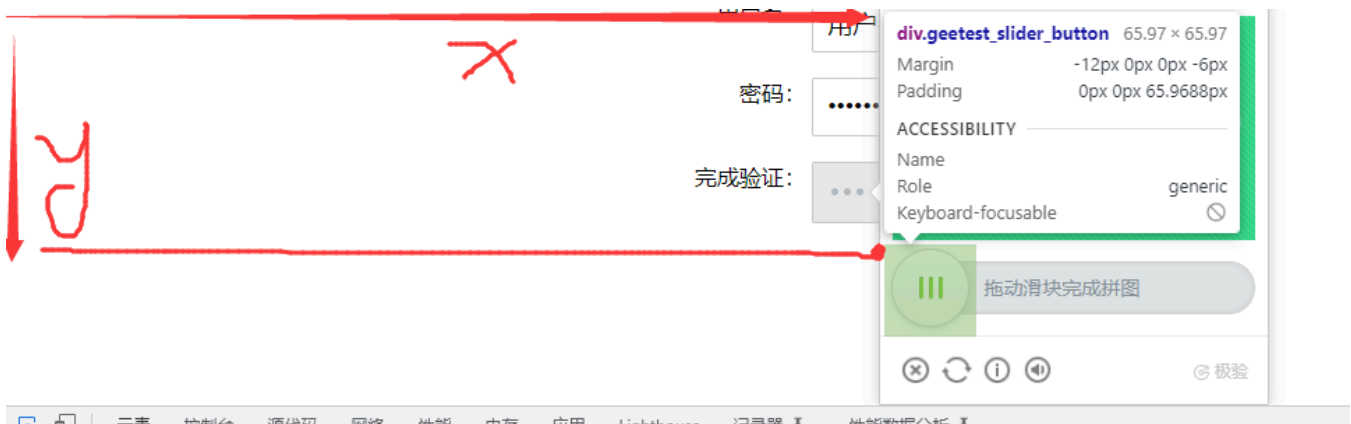
def target_position() -> int:
    """
    进行缺口位置计算识别 可以找我wx:283340479有偿解决
    :return: 缺口位置
    """
    slide = dddocr.DdddOcr(show_ad=False, det=False, ocr=False)
    with open('bg.jpg', 'rb') as f:
        target_bytes = f.read()
    with open('full.jpg', 'rb') as f:
        background_bytes = f.read()
    res = slide.slide_comparison(target_bytes, background_bytes)
    return res.get('target')[0]

x = target_position()
print(x) #
```

也就是说，我们只要从网页上获取到上面2张图，也就得到了缺口位置

1.16.3 定位滑动的按钮

定位滑动操作的按钮，计算按钮的x和y轴距离



x 和 y轴距离是以浏览器左上角的位置，横向是x，纵向是y

通过定位到元素，调用bounding_box() 方法，得到x和y，以及图片的宽高

```
# 滑动按钮
slider = page.locator('div.geetest_slider_button').bounding_box()
print(slider)
```

返回结果

```
{'x': 811.53125, 'y': 489.875, 'width': 55, 'height': 55}
```

1.16.4 page.mouse 鼠标操作

page.mouse 鼠标操作的几个方法

- page.mouse.move 是先把鼠标放到指定的坐标位置，x和y是坐标位置
- page.mouse.down 是按下鼠标，button="middle" 参数是按住鼠标的中间位置
- page.wait_for_timeout 是滑动的时候加一些等待时间，防止操作太快会被识别到不是人是手动操作
- page.mouse.up 是释放鼠标

需注意是是page.mouse.move 传的参数是x和y的绝对坐标位置，跟selenium操作不一样，selenium是先定位某个元素，根据该元素的位置坐偏移的计算（相对位置）

1.16.5 代码示例

完整代码示例

```

from playwright.sync_api import sync_playwright
import base64
import ddddocr
import random

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, args=['--start-maximized'])
    context = browser.new_context(no_viewport=True, record_video_dir="videos/")
    page = context.new_page()
    page.goto('https://www.geetest.com/demo/slide-float.html')

    # 2、填写信息
    page.locator('#username').clear()
    page.locator('#username').type('shanghaiyoyo')
    page.locator('#password').clear()
    page.locator('#password').type('12312313')

    # 点滑块
    page.locator('.geetest_radar_tip').click()
    page.wait_for_timeout(3000)

    # 获取 canvas 元素背景图
    bg_src = page.evaluate(
        "document.getElementsByClassName('geetest_canvas_bg geetest_absolute')[0].toDataURL('image/png')"

    full_src = page.evaluate(
        "document.getElementsByClassName('geetest_canvas_fullbg geetest_fade geetest_absolute')[0].toDataURL('image/png')"

    # 从base64 编码中读取图片
    bg_base = bg_src.split(',')[1]
    bg_base = base64.b64decode(bg_base)
    with open('bg.jpg', 'wb') as f:
        f.write(bg_base)

    full_base = full_src.split(',')[1]
    full_base = base64.b64decode(full_base)
    with open('full.jpg', 'wb') as f:
        f.write(full_base)

    # 进行缺口识别

def read_img():
    slide = ddddocr.DdddOcr(show_ad=False, det=False, ocr=False)
    with open('bg.jpg', 'rb') as fp:

```

```

        target_bytes = fp.read()
    with open('full.jpg', 'rb') as fp:
        background_bytes = fp.read()
    res = slide.slide_comparison(target_bytes, background_bytes)
    return res.get('target')[0]

x = read_img()
print(x)

# 选中拖动的按钮
slider = page.locator('div.geetest_slider_button').bounding_box()
print(slider)

# 拖动鼠标位置需要去调试, 找到合适的等待时间以及移动距离
page.mouse.move(x=int(slider['x']), y=slider['y'] + slider['height'] / 2)
page.mouse.down(button="middle")
page.wait_for_timeout(300)
page.mouse.move(x=int(slider['x']) + x + random.randint(2, 8), y=slider['y'] + slider['height'] / 2)
page.wait_for_timeout(500)
page.mouse.move(x=int(slider['x']) + x - 2, y=slider['y'] + slider['height'] / 2)
page.mouse.move(x=int(slider['x']) + x - 6, y=slider['y'] + slider['height'] / 2)
page.wait_for_timeout(300)
page.mouse.move(x=int(slider['x']) + x - 8, y=slider['y'] + slider['height'] / 2)

page.mouse.up(button="middle")
page.wait_for_timeout(6000)

context.close()

```

1.17 非无痕模式启动 launch_persistent_context

最近有一些爬虫用户在使用 playwright 的时候, 提到 playwright 默认是用无痕模式打开的浏览器, 很多网站会有反爬机制, 使用无痕模式打开的时候功能无法正常使用。

playwright 提供了 launch_persistent_context 启动浏览器的方法, 可以非无痕模式启动浏览器。

- 无痕模式启动浏览器适合做自动化测试的人员
- 非无痕模式启动浏览器适合一些爬虫用户人员

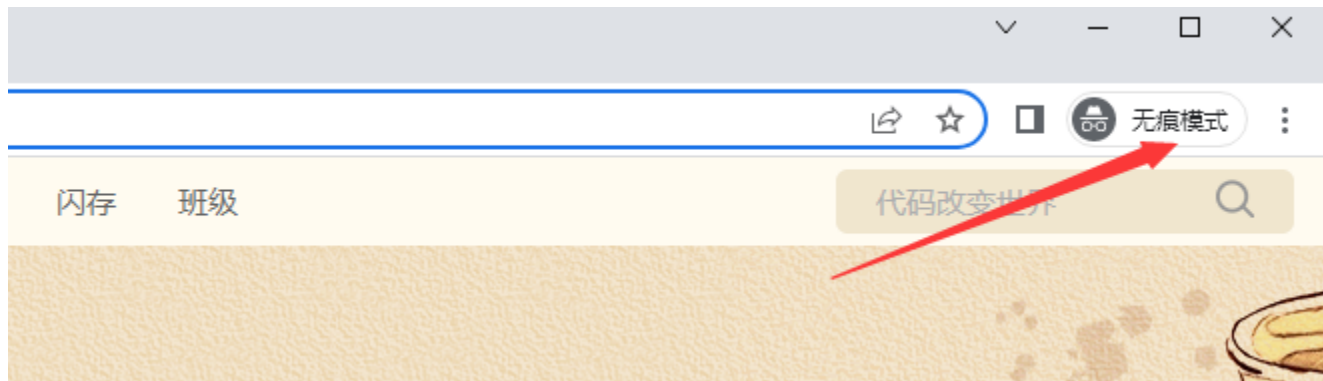
1.17.1 无痕模式启动浏览器

launch 方法是无痕模式启动浏览器

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto("https://www.cnblogs.com/yoyoketang")

    # do ....
    context.close()
    browser.close()
```



1.17.2 非无痕模式 launch_persistent_context

如果网站被识别无痕模式不能使用，那么可以用 launch_persistent_context 非无痕模式启动浏览器

相关参数说明：

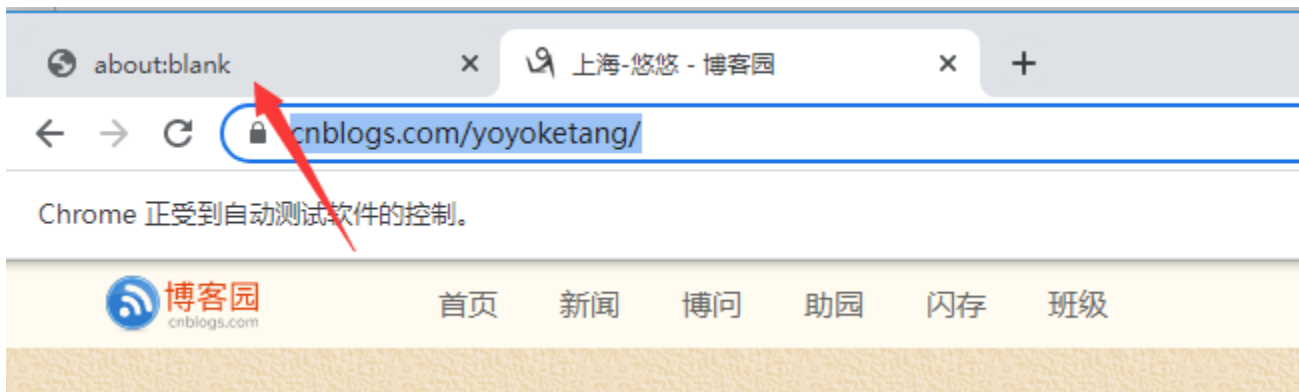
- user_data_dir：用户数据目录，此参数是必须的，可以自定义一个目录
- accept_downloads：接收下载事件
- headless：是否设置无头模式
- channel：指定浏览器类型，默认 chromium

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

p = sync_playwright().start()
browser = p.chromium.launch_persistent_context(
    # 指定本机用户缓存地址
    user_data_dir=f"D:\\chrome_userx\\yoyo",
    # 接收下载事件
    accept_downloads=True,
    # 设置 GUI 模式
    headless=False,
    bypass_csp=True,
    slow_mo=1000,
    channel="chrome"
)
page = browser.new_page()
page.goto("https://www.cnblogs.com/yoyoketang/")

# do ...
browser.close()
```

问题1：以上代码执行时，会发现多一个空白页？



因为使用 `launch_persistent_context` 方法会自动打开一个 tab 标签页，后面代码 `browser.new_page()` 重新打开了一个新的page 对象。

解决办法很简单，去掉 `browser.new_page()` 代码即可。直接用默认打开发tab 标签页对象。


```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

p = sync_playwright().start()
browser = p.chromium.launch_persistent_context(
    # 指定本机用户缓存地址
    user_data_dir=f"D:\chrome_userx\yoyo",
    # 接收下载事件
    accept_downloads=True,
    # 设置 GUI 模式
    headless=False,
    bypass_csp=True,
    slow_mo=1000,
    channel="chrome"
)
page = browser.pages[0]
page.goto("https://www.cnblogs.com/yoyoketang/")

# do ...

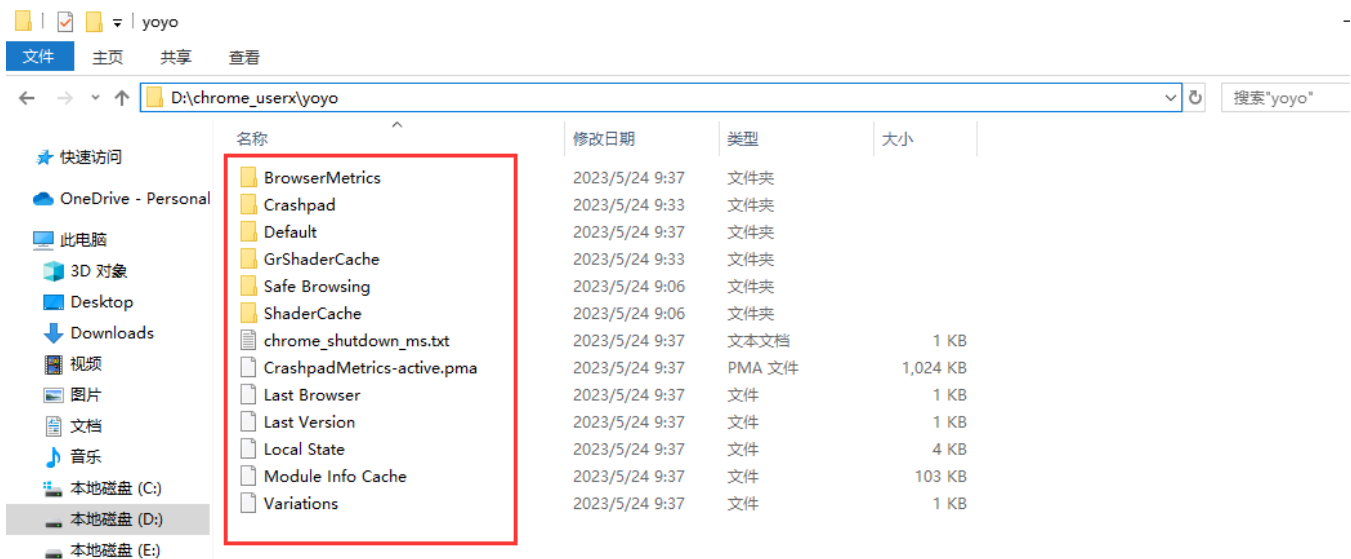
browser.close()
```

问题2：launch_persistent_context 创建的浏览器对象，无法使用 browser.new_context() 创建上下文？

因为 launch_persistent_context 字面上意思就已经是一个 context 上下文对象了，所以无法创建上下文，只能创建 page 对象。

问题3：user_data_dir 路径参数是干什么的

user_data_dir 是指定浏览器启动的用户数据缓存目录，当指定一个新的目录时，启动浏览器会发现自动生成缓存文件



打开 `D:\chrome_userx\yoyo` 目录会看到加载的浏览器缓存文件。

问题4：user_data_dir 能不能记住用户登录的状态

user_data_dir 就是你自己定义的打开浏览器保存的用户数据，包含了用户的 cookies，所以你只要登录过，就会自动保存。

所以你只要代码打开网站，如果不能通过代码自动登录（可能有一些验证码什么的），你可以断点后手工去登录一次，也会记住 cookies。下次代码再打开就不需要登录了。

问题5：为什么按你的教程，我这个网站就无法保持登录？

能不能保持登录状态，主要看你网站的 cookies 有效期，有些网站关闭浏览器后就失效了，比如一些银行的网站，你只要关闭浏览器窗口，下次就需要再次登录。

简单来说一句话：你手工去操作一次，关闭浏览器，再打开还要不要登录，如果关闭浏览器需要再次登录，那代码也没法做到保持登录。

有些博客网站，你登录一次，cookies 几个月都有效，这种就可以利用缓存的 cookies 保持登录。

问题6：为什么网上其他教程 user_data_dir 写chrome 的安装目录？

其实没必要非要写 chrome 的安装目录"`C:\Users\{getpass.getuser()}\AppData\Local\Google\Chrome\User Data`"。

如果你写的是系统默认安装目录的用户数据，那你本地浏览器打开后，执行代码就会报错。所以不推荐！

问题7：默认启动的是 chromium 浏览器，能不能换成其他的？

可以通过 "channel" 参数指定浏览器, 可以支持 chromium 系列 : chromium、chrome、chrome-beta、msedge

问题8 : 如何设置窗口最大化?

添加 `args=['--start-maximized']` 和 `no_viewport=True` 两个参数设置窗口最大化

```
browser = p.chromium.launch_persistent_context(  
    # 指定本机用户缓存地址  
    user_data_dir=f"D:\\chrome_userx\\yoyo",  
    # 接收下载事件  
    accept_downloads=True,  
    # 设置 GUI 模式  
    headless=False,  
    bypass_csp=True,  
    slow_mo=1000,  
    channel="chrome",  
    args=['--start-maximized'],  
    no_viewport=True  
)
```

或者使用 `viewport={'width': 1920, 'height': 1080}` 设置屏幕分辨率

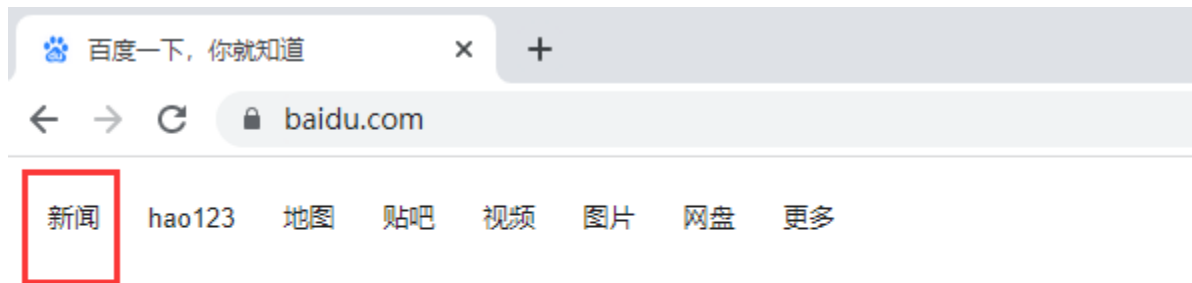
```
browser = p.chromium.launch_persistent_context(  
    # 指定本机用户缓存地址  
    user_data_dir=f"D:\\chrome_userx\\yoyo",  
    # 接收下载事件  
    accept_downloads=True,  
    # 设置 GUI 模式  
    headless=False,  
    bypass_csp=True,  
    slow_mo=1000,  
    channel="chrome",  
    viewport={'width': 1920, 'height': 1080}  
)
```

1.18 highlight 调试定位时高亮显示元素

`highlight()` 方法是通过高亮显示元素,在调试中有很大的优势,可以清楚看到定位的元素所在的位置

1.18.1 遇到的问题

使用示例：点百度页面，定位文本元素“新闻”后点击



```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=2000)
    context = browser.new_context()
    page = context.new_page()
    page.goto('https://www.baidu.com')
    page.get_by_text("新闻").click()

    page.wait_for_timeout(200000)
```

运行时会报错

```
return await self._connection.wrap_api_call(
File "D:\demo\untitled1\venv\lib\site-packages\playwright\impl\connection.py", line 461, in wrap_api_call
    return await cb()
File "D:\demo\untitled1\venv\lib\site-packages\playwright\impl\connection.py", line 96, in inner_send
    result = next(iter(done)).result()
playwright.impl.api_types.Error: Error: strict mode violation: get_by_text("新闻") resolved to 3 elements:
1) <a target="_blank" href="http://news.baidu.com" cla...>新闻</a> aka get_by_role("link", name="新闻")
2) <span class="text-color">互联网新闻信息服务许可证11220180008</span> aka get_by_role("paragraph").filter(
3) <span class="text-color">互联网新闻信息服务许可证11220180008</span> aka locator("div").filter(has_text=

===== logs =====
waiting for get_by_text("新闻")
=====
```

从报错日志中你会看到定位到 3 个元素，导致点击报错。

1.18.2 highlight 高亮调试

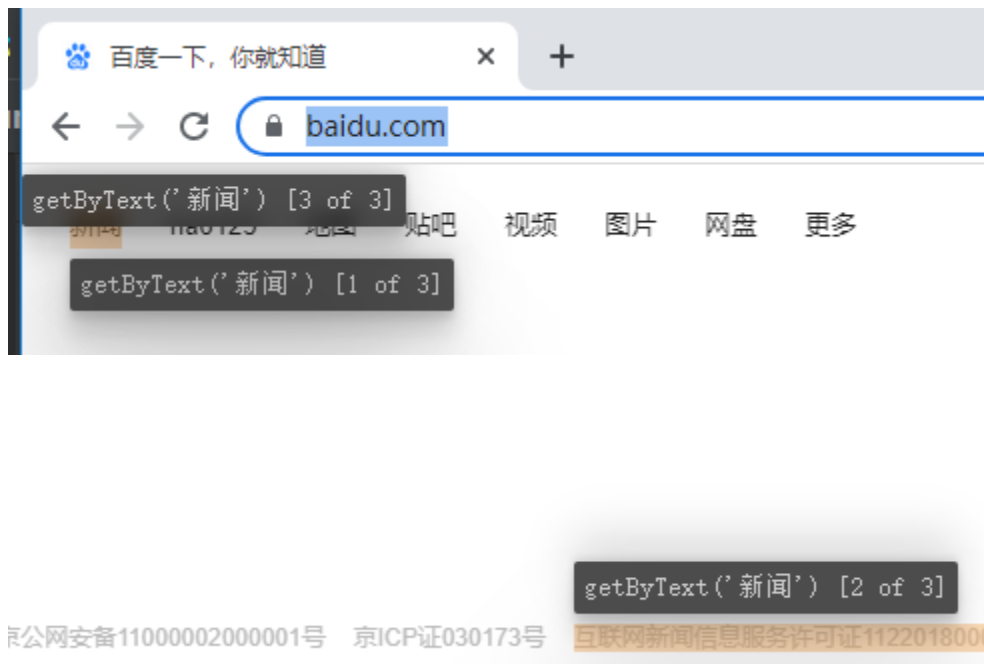
为了更清楚的知道当前定位的方式，在页面上找到哪些元素，可以用到highlight() 方法，方便达到调试的目的（仅仅是调试的时候用）。

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=2000)
    context = browser.new_context()
    page = context.new_page()
    page.goto('https://www.baidu.com')
    page.get_by_text("新闻").highlight() # 高亮

    page.wait_for_timeout(200000)
```

让定位的元素处于高亮状态



这样会发现在屏幕上定位到 3 个元素（其中有一个是隐藏的元素，也被定位到了）

网易云视频课程

网易云视频完整课程地址<https://study.163.com/course/>

[courseMain.htm?courseId=1213382811&share=2&shareId=480000002230338](https://www.udacity.com/courseMain.htm?courseId=1213382811&share=2&shareId=480000002230338)

