pyppeteer 的基本使用

作者：kanadeblisst
原文链接：https://ld246.com/article/1566221786951
来源网站：链滴

<h2 id="前言">前言</h2>
前段时间因为有任务，需要四个电商(京东、淘宝、天猫、拼多多)的数据。而其中京东的没什么爬，基本是随便抓。拼多多的加密参数有点复杂，而且变化也挺频繁的，用的是 selenium，也没什可提的。抓淘宝和天猫因为用 selenium 滑块过不了，于是就改成了 pyppeteer。现在我说一下使用心得。有些细节因为当时没有记录，现在也记不起来了。
首先 <code>pip install pyppeteer</code>，这时候我们先看一下 websockets 这个库的版本因为必须保证这个库是 7.0 的版本。8.0 和 6.0 虽然可以使用，但是在访问多个网页会出现一些无法决的异常。现在安装 pyppeteer，websockets 的版本默认是 8.0，我们需要先 <code>pip uninstall websockets</code> 卸载 8.0，然后 <code>pip install websockets==7.0</code>。然后我们照 https://github.com/miakogi/pyppeteer/pull/160/files 这个网页修改 pyppeteer 的源码(如果是 anaconda，则路径 anaconda 安装路径\Lib\site-packages\pyppeteer\connection.py 这个文件)。这样准备工作就完了，这样操作之后就可以避免很多异常。
需要这么费劲的折腾，是因为 pyppeteer 这个库已经很久没有做大的更新了，虽然有点小的改，但是基本的 bug 还是没有解决。所以如果有 js 基础，最好还是使用 JavaScript 的 puppeteer 这工具，这个是谷歌出品的，一直在更新维护，基本没有 bug。
补充：有时候会出现明明网页加载完成了，但是还是会出现超时的情况。不知道是什么bug，解决方法也很简单，只要捕获这个异常，然后该做啥做啥。
以下有的方法前面带 coroutine 这个关键字，有点没有，这是因为带这个参数的方法是异步方法，这种方法必须被 async 关键字修饰，调用的时候也必须 await 关键字修饰。不过我们然使用浏览器爬虫了，肯定已经不在乎效率了，异步等于在等网站封，所以只需要遵守某些规则即可(懂的请百度 asyncio 的使用)，实际上还是同步运行。另外，虽然 page.on（）这个方法是存在的，是官方文档中并未给出详细的说明，甚至我查看 pyppeteer 的源码也未找到这个函数，不知道是怎回事。但是参考 puppeteer 的文档也是可以用的，不过报错居多，我也就没有深究了。
<h2 id="API参考">API 参考</h2>
<h2 id="启动器">启动器</h2>

pyppeteer.launcher.launch()
启动 chrome 进程并返回浏览器实例

参数：

<code>ignoreHTTPSErrors</code>（bool）：是否忽略 HTTPS 错误。默认为 <code>False/code>
<code>headless</code>（bool）：是否在无头模式下运行浏览器。默认为 <code>True</code> 除非 <code>appMode</code> 或 <code>devtools</code> 选项 <code>True</code></i>
<code>executablePath</code> （str）：运行 Chromium 或 Chrome 可执行文件的路径，不是默认捆绑的 Chromium
<code>slowMo</code> （int | float）：按指定的毫秒数减慢 pyppeteer 操作。
<code>args</code> （List [str]）：传递给浏览器进程的附加参数（标志）。
<code>dumpio</code>（bool）：是否管道浏览器进程 stdout 和 stderr 进入 <code>process.stdout</code> 和 <code>process.stderr</code>。默认为 <code>False</code>。
<code>userDataDir</code> （str）：用户数据目录的路径。
<code>env</code>（dict）：指定浏览器可见的环境变量。默认与 python 进程相同。
<code>devtools</code>（bool）：是否为每个选项卡自动打开 DevTools 面板。如果是此项 <code>True</code>，<code>headless</code> 则将设置该选项 <code>False</code>。</i>
<code>logLevel</code>（int | str）：用于打印日志的日志级别。默认值与根记录器相同。</l>
<code>autoClose</code>（bool）：脚本完成时自动关闭浏览器进程。默认为 <code>True/code>。

<code>loop</code>（asyncio.AbstractEventLoop）：事件循环（实验）。

<hr>

ignoreHTTPSErrors：意思很明显，不解释
headless：不解释
executablePath：这个一般是默认的路径就行
slowMo：我一般会加这个参数，至于效果吗，好像有点。
args：常用的有['--no-sandbox','--disable-gpu', '--disable-setuid-sandbox','--window-size 1440x900']
dumpio: 不知道为什么，如果不加 dumpio=True 有时会出现浏览器卡顿
userDataDir：selenium 的 userdata 可以保留 cookie，下次启动可以达到免登录，而这个参没什么用
env：看需求
devtools：用的不深，没用过
logLevel：没什么用，在 Linux 下还是打印大量的日志
autoClose：默认就好，不过如果你需要保持浏览器状态，可以不关闭，下次直接连接这个已存的浏览器
loop：暂时不知道有什么用

<ol start="2">
pyppeteer.launcher.connect()
连接到现有的 chrome

参数：

<code>browserWSEndpoint</code>（str）：要连接的浏览器 websocket 端点。（必填）
<code>ignoreHTTPSErrors</code>（bool）：是否忽略 HTTPS 错误。默认为 <code>False</code>。
<code>slowMo</code> （int | float）：按指定的毫秒数减慢 pyppeteer 的速度。
<code>logLevel</code>（int | str）：用于打印日志的日志级别。默认值与根记录器相同。
<code>loop</code>（asyncio.AbstractEventLoop）：事件循环（实验）。

browserWSEndpoint：格式是 <code>ws://${host}:${port}/devtools/browser/<id></ode>。以前浏览器的 wsEndpoint 这个值，可以通过 <code> brower = pyppeteer.launcher.launh()</code> <code>dev = brower. wsEndpoint</code> 保存这个值就可以下次启动了
<ol start="3">
pyppeteer.launcher.executablePath()
获取默认 chrome 的可执行路径

<h2 id="浏览器类">浏览器类</h2>
pyppeteer.browser.Browser()
这个类是由 launch()返回的实例

browserContexts： 返回所有打开的浏览器上下文的列表，在新创建的浏览器中，这将返回单个例
coroutine close()： 关闭连接并终止浏览器进程
coroutine createIncognitoBrowserContext(): 创建一个新的隐身浏览器上下文。这不会与其他览器上下文共享 cookie /缓存
coroutine disconnect(): 断开浏览器

coroutine newPage(): 在此浏览器上创建新页面并返回其对象
coroutine pages(): 获取此浏览器的所有页面。返回格式为列表，包含所有页面
process：返回此浏览器的进程
target(): 获取浏览器中所有活动目标的列表
coroutine userAgent()：返回浏览器的原始 UA
coroutine version(): 获取浏览器的版本
wsEndpoint: 返回 websocket 端点 url

<h2 id="页面类">页面类</h2>
pyppeteer.page.Page()
由 brower.newPage()或者 brower.pages()得到

coroutine J()：别名 querySelector(),看名字就知道通过 css 选择器来选出元素

coroutine JJ(): 别名 querySelectorAll()

coroutine Jeval(): 功能比 page.J()功能多一点，可以选出网页文本或者属性指

coroutine JJeval(): querySelectorAllEval()

coroutine Jx(): 别名 xpath()

coroutine addScriptTag(): 将脚本标记添加到此页面, 返回 ElementHandle
其中一个 url，path 或 content 选择是必要的。

<code>url</code> （字符串）：要添加的脚本的 URL。
<code>path</code> （字符串）：要添加的本地 JavaScript 文件的路径。
<code>content</code> （字符串）：要添加的 JavaScript 字符串。
<code>type</code>（字符串）：脚本类型。使用 <code>module</code> 以加载一个 JavaScript ES6 模块。

coroutine addStyleTag(): 将样式或链接标记添加到此页面, 返回 ElementHandle
其中一个 <code>url</code>，<code>path</code> 或 <code>content</code> 选择是必要。

<code>url</code> （字符串）：要添加的链接标记的 URL。
<code>path</code> （字符串）：要添加的本地 CSS 文件的路径。
<code>content</code> （字符串）：要添加的 CSS 字符串。

coroutine authenticate(): 提供 http 身份验证的凭据

coroutine bringToFront(): 将页面置于前面

browser: 获取该页面所属的浏览器

coroutine click(): 相当于鼠标左键单击匹配的元素,参数是 css 选择器的字符串。如果没有匹配元素，则会引发 PageError
可选参数：

<code>button</code>（str）：，left，right 或 middle，默认为 left。
<code>clickCount</code>（int）：默认为 1。
<code>delay</code>（int | float）：等待时间 mousedown 和 mouseup 以毫秒为单位的间。默认为 0。

注意
如果此方法触发导航事件并且存在单独的事件，则 waitForNavigation()最终可能会出现产生意外结的竞争条件。单击并等待导航的正确模式如下：
<pre><code class="highlight-chroma">```
await asyncio.gat er(

    page.waitForNa igation(waitOptions),

    page.click(selec or, clickOptions),

)
<span class="highlight-line"><span class="highlight-cl">```
</code></pre>
大概意思是 click()触发如果会跳转网页，则需要 page.waitForNavigation()这个函数来等待。点击事件和触发事件是同时的，所以需要将这两个事件绑定，代码如上。

coroutine close(): 关闭此页面

coroutine content(): 获取页面的完整 HTML 内容

coroutine cookies(): 获取 cookie。参数为 URL，如果未指定 URL，则此方法返回当前页面 URL 的 cookie。如果指定了 URL，则仅返回这些 URL 的 cookie

coverage： 返回 coverage

coroutine deleteCookie(): 删除 cookie，参数为 cookie 字段参数，比如 name、url、domai 、path 等

coroutine emulate(): 模拟给定的设备指标和用户代理。相当于同时设置了 setUserAgent()、se

Viewport()

coroutine emulateMedia(): 模拟页面的 css 媒体类型。参数为：screen、print、None。

coroutine evaluate(): 在浏览器上执行 js 并获取结果

coroutine evaluateHandle(): 同上，只是会返回一个值。至于是什么，没用过

coroutine evaluateOnNewDocument(): 也是执行 js，至于区别。好像是在页面跳转时执行</
>

coroutine exposeFunction(): 将 python 函数添加到浏览器的 window 对象中 name。不知道
什么用

coroutine focus(): 聚焦匹配的元素。如果不存在，抛出 PageError

frames：获取此页面的所有 frame

coroutine goBack()：返回上一个网页。如果不存在，则返回 None

coroutine goForward(): 跳转到下一页。如果不存在，则返回 None

coroutine goto(): 访问某个 URL。
参数：

url：。。。
timeout：。。。
waitUntil：什么情况下算成功访问。没看懂参数的几个值有什么用

coroutine hover(): 鼠标悬停匹配的元素

isClosed(): 判断页面是否关闭

keyboard： 获取 keyboard 对象

mainFrame： 获取 frame 页面的主要内容

coroutine metrics(): 好像是获取页面中的信息

mouse： 获取 mouse 对象

coroutine reload(): 刷新网页

coroutine screenshot(): 截图
参数：

<code>path</code>（str）：保存图像的文件路径。屏幕截图类型将从文件扩展名中推断出。
<code>type</code>（str）：指定屏幕截图类型，可以是 <code>jpeg</code> 或 <code>ng</code>。默认为 <code>png</code>。
<code>quality</code>（int）：图像的质量，在 0-100 之间。不适用于 <code>png</code> 图像。
<code>fullPage</code>（bool）：如果为 true，请截取完整的可滚动页面。默认为 <code>alse</code>。
<code>clip</code>（字典）：指定页面剪切区域的对象。此选项应包含以下字段：

<code>x</code> （int）：剪辑区域左上角的 x 坐标。
<code>y</code> （int）：剪辑区域左上角的 y 坐标。
<code>width</code> （int）：剪切区域的宽度。
<code>height</code> （int）：剪切区域的高度。

<code>omitBackground</code> （bool）：隐藏默认的白色背景并允许捕获具有透明度的幕截图。
<code>encoding</code>（str）：图像的编码可以是 <code>'base64'</code> 或 <code>binary'</code>。默认为 <code>'binary'</code>。

coroutine select(): 选择选项并返回所选值。如果未找到元素，抛出 ElementHandleError。

coroutine setCacheEnabled(): 为每个请求启用/禁用缓存。默认启用

coroutine setContent(): 将 html 加到网页

coroutine setCookie(): 设置 cookie

setDefaultNavigationTimeout(): 更改默认的最大导航超时。默认 30 秒

coroutine setJavaScriptEnabled(): 设置 JavaScript 启用/禁用

coroutine setExtraHTTPHeaders(): 设置默认 headers 头，值为一个字典

coroutine setRequestInterception(): 启用/禁用请求拦截。配合 page.on()

coroutine setUserAgent(): 设置要在此页面中使用的 UA

coroutine setViewport(): 设置窗口大小。
参数：

<code>width</code> （int）：以像素为单位的页面宽度。
<code>height</code> （int）：以像素为单位的页面高度。
<code>deviceScaleFactor</code> （float）：默认为 1.0。
<code>isMobile</code>（bool）：默认为 <code>False</code>。
<code>hasTouch</code>（bool）：默认为 <code>False</code>。
<code>isLandscape</code>（bool）：默认为 <code>False</code>。

coroutine tap(): 点击与之匹配的元素

coroutine title(): 获取页面标签

tracing：获取 tracing 对象

coroutine type(): 往输入框中输入内容，第一个参数为 css 选择器，第二个为文本内容

url：获取此页面的 url

waitFor(): 等待页面上匹配的函数，超时或元素。参数可以是 int 或者 float，单位毫秒。也可以 css 选择器

waitForFunction():

coroutine waitForNavigation(): 等到网页跳转

coroutine waitForRequest(): 等待请求

coroutine waitForResponse(): 等待回应

waitForSelector(): 等到页面上出现匹配的元素。其实 waitFor()也有同样功能。

waitForFunction(): 等到函数完成并返回一个值

coroutine waitForXPath(): 等到 xpath 页面上出现匹配的元素

workers: 获取页面的所有 worker。至于是什么，我不知道

coroutine xpath(): ..

<h2 id="worker类">worker 类</h2>
page.on('workercreated', 函数) 这个函数传入的参数就是 worker 类

coroutine evaluate(): 同上
coroutine evaluateHandle():同上
coroutine executionContext():同上
url: 同上

<h2 id="键盘类">键盘类</h2>

coroutine down(): 如果没有参数，则是按下鼠标左键。如果是键盘的某个值比如 shift、A 等键则是相当于按下这些键。
coroutine press(): 同 down()，但是上面是不会释放鼠标或者键盘，需要调用 up 方法释放鼠标
coroutine sendCharacter：将字符发送到页面。没用过
coroutine type(): 同上
coroutine up(): 释放由 down 按下的键或者鼠标

<h2 id="鼠标类">鼠标类</h2>

coroutine click(): 按下(x, y)处的按钮
参数：

<code>button</code>（STR）： ，left（左键）、right（右键）或 middle（中键），默认为 <code>left</code>。
<code>clickCount</code> （int）：默认为 1。
<code>delay</code>（int | float）：等待时间 <code>mousedown</code> 和 <code>mouseup</code> 以毫秒为单位的时间。默认为 0。

coroutine down(): 同上

coroutine move(): 移动鼠标光标。

coroutine up()： 同上

<h2 id="tracing类">tracing 类</h2>

创建可在 Chrome DevTools 或时间线查看器打开的跟踪文件，使用如下
<pre><code class="highlight-chroma"><span class="highlight
cl">await page.tracing.start({'path': 'trace.json'})

await page.goto('
ttps://www.google.com')

await page.tracing
stop()
</code></pre>

coroutine start(): 开始跟踪
coroutine stop(): 停止跟踪

不好用，可能是我不知道用
<h2 id="Dialog类">Dialog 类</h2>
<code>page.on('dialog', 函数)</code>，函数的参数就是这个类

coroutine accept(): 接受对话框
defaultValue： 如果对话框提示，则获取默认提示值
coroutine dismiss(): 关闭对话框
message: 获取对话框消息
type： 获取对话框类型。类型有：<code>alert</code>，<code>beforeunload</code>，
code>confirm</code>，或 <code>prompt</code>

<h2 id="frame类">frame 类</h2>

coroutine J():同上
coroutine JJ():同上
coroutine JJeval(): 同上
coroutine Jeval(): 同上
coroutine Jx(): 同上
coroutine addScriptTag(): 同上
coroutine addStyleTag(): 同上
childFrames: 获取子框架
coroutine click(): 同上
coroutine content(): 同上
coroutine evaluate(): 同上
coroutine evaluateHandle(): 同上
coroutine executionContext(): 同上
coroutine focus(): 同上
coroutine hover(): 同上
isDetached()： 如果此框架已分离，则返回 True
name：获取 frame 的名称，如果没有则返回 ID
parentFrame：获取父框架
后面这些属性和 page 类一模一样

<h2 id="ElementHandle类">ElementHandle 类</h2>

coroutine J()
coroutine JJ()
coroutine JJeval()
coroutine Jeval()
asElement

coroutine boundingBox(): 返回此元素的边界框,如果元素不可见，则返回 None
coroutine boxModel():返回元素框
coroutine click()
coroutine contentFrame(): 返回元素句柄的 frame
coroutine focus()
coroutine hover()
coroutine isIntersectingViewport(): 如果元素在视口中可见，则返回 True
coroutine press()
coroutine screenshot()
coroutine tap()
coroutine type()
coroutine uploadFile(): 上传文件

<h2 id="request类">request 类</h2>
page.on('request', 函数) 这个函数传入的参数就是 request 类

coroutine abort(): 中断请求
coroutine continue_(): 使用可选的请求覆盖继续请求
参数:

<code>url</code> （str）：如果设置，请求 URL 将被更改。
<code>method</code>（str）：如果设置，则更改请求方法（例如 <code>GET</code>）

<code>postData</code> （str）：如果设置，则更改发布数据或请求。
<code>headers</code> （dict）：如果设置，则更改请求 HTTP 标头

failure(): 返回错误文本
frame : 返回匹配的 frame 对象
headers: 返回此请求的 HTTP 标头字典
isNavigationRequest(): ？？
method：返回此请求的方法
postData： 返回此请求的帖子正文
redirectChain： request 类重定向的所有请求
resourceType：渲染引擎感知的此请求的资源类型
coroutine respond(): 通过给定的参数修改请求
参数:

<code>status</code> （int）：响应状态代码，默认为 200。
<code>headers</code> （dict）：可选的响应头。
<code>contentType</code>（str）：如果设置，则等于设置 <code>Content-Type</code
响应头。
<code>body</code> （str | bytes）：可选的响应主体。

response(): 返回匹配的 response 对象
url： 此请求的 URL

<h2 id="response类">response 类</h2>

buffer(): 返回相应的字节内容
fromCache： 如果响应是从缓存提供的，则返回 True。缓存可能是浏览器的磁盘缓存或者内存
而不是服务器响应的
fromServiceWorker： 如果响应由服务器提供，则返回 True

headers： 返回此响应的 HTTP 标头字典
coroutine json(): 获取响应正文的 JSON
ok： 返回 bool 此请求是否成功
request： 获取匹配的 request 对象
securityDetails： 返回与此响应关联的安全详细信息
status： 响应的状态代码
coroutine text(): 获取响应正文的文本
url : 响应的 URL

<h2 id="target类">target 类</h2>
我到现在还没明白这是干什么的

browser： 获取目标所属的浏览器
browserContext： 返回目标所属的浏览器上下文
coroutine createCDPSession(): 创建附加到目标的 Chrome Devtools 协议会话
opener: 获取打开此目标的目标
coroutine page(): 获取此目标的页面
type: 获取此目标的类型
url: 获取此目标的网址

<h2 id="CDPSession类">CDPSession 类</h2>

coroutine detach(): 从目标分离会话
send() :

<h2 id="Debugging">Debugging</h2>
对于调试，您可以设置 for 和 功能 <code>logLevel</code> 选项。但是，此选项会打印太多志，包括 pyppeteer 的 SEND / RECV 消息。为了只显示被抑制的错误消息，您应该设置 logging.DBUG 为 True
<pre><code class="highlight-chroma">import asyncio
import pyppeteer
from pyppeteer i port launch

pyppeteer.DEBUG = True # print suppressed errors as error log

async def main():

    browser = await launch()

    ... # do someth ng

asyncio.get_event_ oop().run_until_complete(main())
</code></pre>