網路爬蟲與資料分析
動態網頁解析

Instructor: 馬豪尚

Cloudflare 反爬蟲技術

- Cloudflare採取多層次的防護機制
  - IP封鎖和速率限制：監控訪問頻率和模式，識別異常行為，對可疑IP地址進行速率限制或封鎖。
  - JavaScript挑戰：要求訪問者執行特定的JavaScript代碼，以驗證其為真實使用者。
  - 設備指紋識別：收集並分析訪問設備的特徵資訊，區分自動化爬蟲和真實使用者。
  - CAPTCHA驗證：當檢測到可疑行為時，觸發CAPTCHA驗證，阻止自動化腳本的操作。

如何判定是否有Cloudflare

- 對網頁做request，回傳的網頁內容含有一些和challenge相關的標籤
  - <form id="challenge-form" ....>
  - a.src='/cdn-cgi/challenge-platform/scripts/jsd/main.js'
  - <script src="https://challenges.cloudflare.com/turnstile/v0/b/22755d9a86c9/api.js?onload=clJo2&render=explicit" ...

Pyppeteer 控制瀏覽器套件

- 安裝套件
  - pip install pyppeteer
- 啟動瀏覽器: launch(options)
  - options為傳入瀏覽器的參數，主要有headless、args
  - headless: True/False，控制瀏覽器是否以無頭模式啟動
  - args: list[str]，傳遞給瀏覽器的啟動參數
    - --no-sandbox: 停用sandbox模式，提升兼容性（在某些系統環境下必須）
    - --disable-infobars: 停用「Chrome 正由自動測試軟體控制」提示
    - --start-maximized: 啟動時視窗最大化

Example:

```
browser = await launch(headless=True, args=['--no-sandbox'])
```

Pyppeteer 瀏覽器物件主要功能與方法

- browser.newPage(): 創建並返回一個新的瀏覽器分頁
  - page = await browser.newPage()
- browser.pages(): 返回當前所有分頁的列表
  - pages = await browser.pages()
- browser.close(): 關閉瀏覽器
  - await browser.close()

Pyppeteer 頁面物件主要功能與方法

- page 代表瀏覽器中的一個分頁
  - 有許多API提供瀏覽網頁、模擬使用者操作、截圖、獲取內容等功能
- 前往網頁
  - page.goto(url)
- 重新載入頁面
  - page.reload()
- 網頁截圖
  - page.screenshot(options)
    - path：保存文件的路徑。
    - fullPage：是否截取完整頁面。
    - type：圖片類型（'png' 或 'jpeg'）

Pyppeteer 頁面物件獲取網頁內容方法

- 返回頁面的完整 HTML
  - html = await page.content()
- 返回頁面的標題
  - title = await page.title()
- 在頁面上下文中執行 JavaScript 函數
  - await page.evaluate(pageFunction, *args)

Pyppeteer 頁面物件模擬使用者操作

- 模擬點擊指定的 DOM 元素
  - page.click(selector, options)
    - options常用參數
      - button：按鍵類型（'left', 'right', 'middle'）
      - clickCount：點擊次數
    - Example: await page.click('#submit-button')
- 模擬在輸入框中輸入文字
  - page.type(selector, text)
    - Example: await page.type('#username', 'example_user')

Pyppeteer 頁面物件模擬使用者操作

- 模擬將滑鼠鼠標停在指定元素上
  - page.hover(selector)
  - Example: await page.hover('#menu-item’)
- 模擬鼠標控制，用於鼠標移動和點擊操作
  - Example: await page.mouse.click(100, 200)
- 模擬鍵盤控制，用於鍵盤輸入操作
  - Example: await page.keyboard.type('Hello World!')

Pyppeteer 頁面物件獲取頁內元素

- page.$(selector) : 選擇第一個符合 CSS 選擇器的元素
  - Example: element = await page.$('#my-element')
- page.$$(selector):選擇所有符合 CSS 選擇器的元素，返回列表
  - Example: elements = await page.$$('.list-item')
- page.waitForSelector(selector): 等待指定的DOM元素出現
  - Example: await page.waitForSelector('#submit-button')
- page.waitForXPath(xpath): 等待指定的 XPath 元素出現
  - Example: await page.waitForXPath('//button[text()="Submit"]')

Python asyncio 模組

- Python中專門用於支援非同步執行的模組
- await
  - await 用於等待一個可等待對象（awaitable object）完成
- Event Loop
  - 事件循環 是 asyncio 的核心，用於調度和執行非同步任務
  - 事件循環會管理所有的非同步任務，並在適當的時機執行它們。任務遇到 I/O 操作（如網絡請求、文件讀取）時會暫停，事件循環會切換執行其他任務，直到操作完成。

asyncio 的主要功能與方法

- asyncio.run(coroutine)
  - 最常用的方式，啟動協程並運行事件循環。適合在頂層執行非同步任務。
  - Example: asyncio.run(main())
- asyncio.get_event_loop()
  - 獲取當前的事件循環
  - Example: loop = asyncio.get_event_loop()
- loop.run_until_complete(future)
  - 手動啟動事件循環，運行直到給定的function或未來對象完成
  - Example: loop.run_until_complete(scrape_website(url))

pyppeteer_stealth

- pyppeteer_stealth 是一個第三方工具，專為 Pyppeteer 設計，用於繞過目標網站的反爬蟲機制
- 對瀏覽器的一些屬性進行修改或偽裝，使瀏覽器更像是普通使用者操作的真實環境，而非機器操作
- 安裝 pyppeteer_stealth
  - pip install pyppeteer-stealth
- 使用stealth偽裝瀏覽器
  - await stealth(page)

stealth 方法的功能

- stealth 方法的功能stealth 方法會對頁面（Page 對象）進行以下修改：
  - 偽裝 navigator.webdriver：設置 navigator.webdriver = false，讓瀏覽器看起來不是自動化工具。
  - 修改插件和 MimeTypes：模擬普通瀏覽器安裝的插件和支援的文件類型。
  - 修改 User-Agent：使其更接近於真實的使用者瀏覽器。
  - 偽裝堆疊追蹤（Stack Trace）：防止網站通過腳本檢測自動化工具的異常堆疊。
  - 處理 Chrome 特性：修改一些與 Chrome 自動化相關的特性，如 window.chrome。
  - 處理 iframe 偵測：修正 iframe 的 contentWindow，避免被檢測。

練習

- 挑戰openai api介紹網頁的cloudflare防護
  - https://openai.com/index/introducing-structured-outputs-in-the-api/
- 爬取網頁內的python程式碼範例
  - 存成文字檔案