告别 Selenium！用 WebSocket + CDP 打造超高效自动化工具

在自动化测试和爬虫开发领域，Selenium 是一个广泛使用的工具，但随着技术的发展，Chrome DevTools Protocol（CDP）提供了更高效、更强大的功能。

本文将介绍如何结合 WebSocket 和 CDP，打造一个类似 Selenium 的自动化工具，实现对 Chrome 浏览器的精细控制。

这种方法不仅性能更高，功能更强大，而且更加轻量级。

上篇文章为大家介绍了**WebSocket、CDP和Selenium与浏览器通信流程三个概念，以及为什么选择 WebSocket + CDP**。接下来将为大家介绍如何**启动Chrome 并开启远程调试。**

启动 Chrome 并开启远程调试

01 准备工作

在开始之前，你需要确保已经安装了以下工具和库：

- **Chrome 浏览器**：
  确保你的 Chrome 浏览器版本支持 CDP。

- **Python**：
  用于编写自动化脚本。版本最好在 Python3.6 以上。

- **websocket-client**：
  用于与 Chrome 的 WebSocket 调试端口通信。

  websocket-client 是 Python 的一个第三方库，需要提前安装。

  安装命令：pip install websocket-client

- **requests**：
  一个非常流行的 Python HTTP 库，用于发送 HTTP 请求，支持多种 HTTP 方法（如 GET、POST、PUT、DELETE 等），并提供了丰富的功能等。

02 启动 Chrome 并开启远程调试

不同操作系统，启动 Chrome 并开启远程调试的命令稍微有所区别，下面对主流的 Windows、Linux 和 MacOS 系统进行介绍。

在 Windows 系统中，可以通过以下命令启动 Chrome 并开启远程调试：

```
"C:\Program Files\Google\Chrome\Application\chrome.exe" --remote-debugging-port=9222
```

在 Linux 系统中，可以通过以下命令启动 Chrome 并开启远程调试：

```
google-chrome
--remote-debugging-port=9222
```

在 MacOS 系统中，可以通过以下命令启动 Chrome 并开启远程调试：

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --remote-debugging-port=9222
```

在启动时，我们还可以添加一些参数来指定调试端口、用户数据目录、代理服务器等。以下是一些常用的参数及其说明：

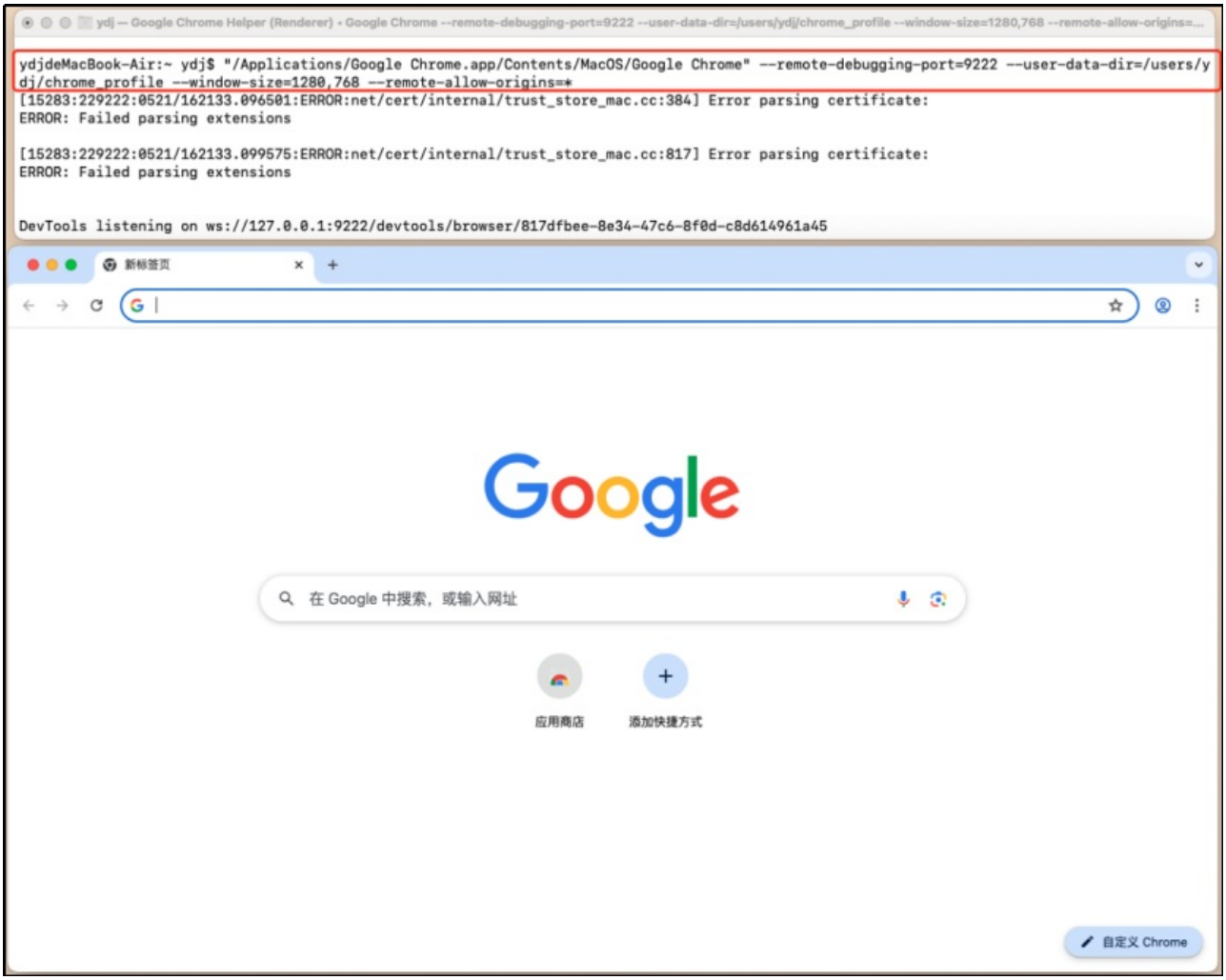
参数	说明
--remote-debugging-port=<port>	指定远程调试端口，默认为 9222
--user-data-dir=<path>	指定用户数据目录，避免与默认的 Chrome 配置冲突
--proxy-server=<proxy>	设置代理服务器，格式为protocol://username:password@host:port
--headless=new	无头模式运行，适合自动化测试
--window-size=WIDTH,HEIGHT	设置初始窗口尺寸
--lang=<language>	设置浏览器语言
--disable-extensions	禁用扩展程序
--incognito	启动隐身模式
--remote-allow-origins=<origins>	限制访问范围，* 表示所有人都可以访问

例如我们启动一个调试端口为 9222、用户数据目录为 /users/ydj/chrome_profile、初始窗口大小为 1280x768 、限制访问范围为所有人都可以访问的 Chrome。

命令就可以写成：

```
"/Applications/Google Chrome.app/Contents/MacOS/Google Chrome" --remote-debugging-port=9222 --user-data-dir=/users/ydj/chrome_profile --window-size=1280,768 --remote-allow-origins=*
```

在命令行窗口执行上面命令后，会按照参数设置的启动浏览器，如下图所示：



在命令行工具中可以看到
DevTools listening on ws:
//127.0.0.1:9222/devtools/browser/817dfbee-8e34-47c6-8f0d-c8d614961a45信息。

表示Chrome 已经成功启动，并且 DevTools（开发者工具）正在监听
WebSocket 地址 ws:
//127.0.0.1:9222/devtools/browser/817dfbee-8e34-47c6-8f0d-c8d614961a45。

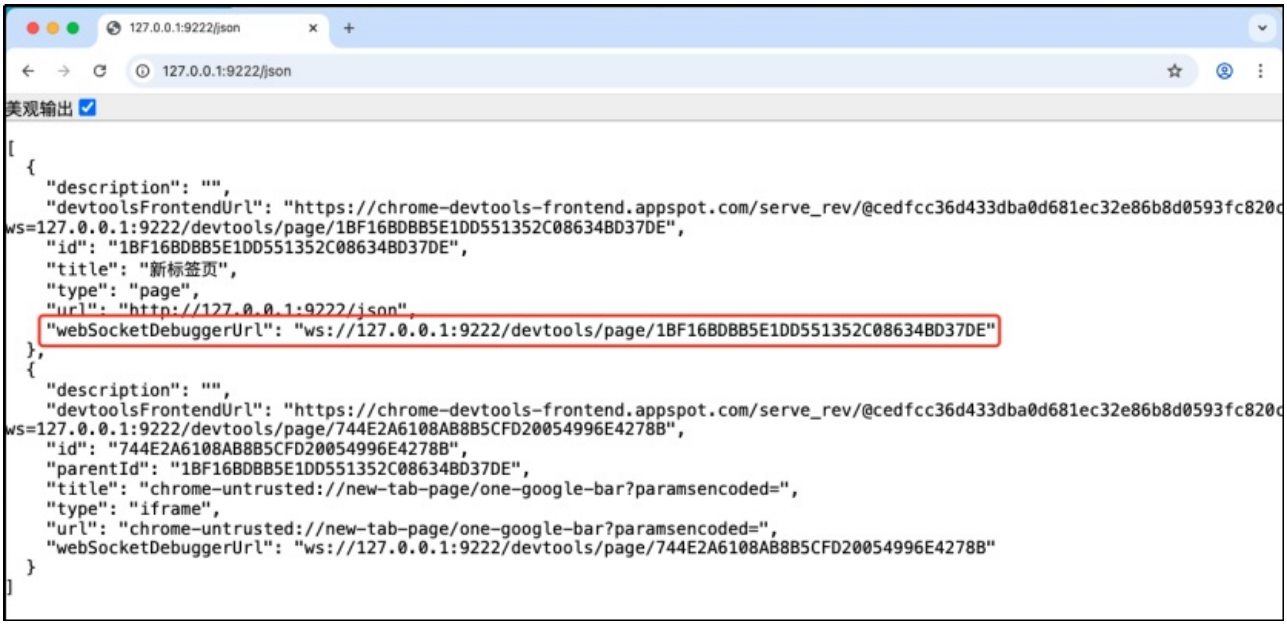
这意味着我们可以通过这个地址连接到 Chrome 的远程调试接口。

03 获取 WebSocket 调试 URL

启动 Chrome 并开启远程调试后，输入地址`http://127.0.0.1:9222/json`将得到一个 JSON 格式的列表，包含所有正在运行的 Chrome 实例及其调试信息。

我们可以从中找到目标实例，并通过其 WebSocket URL 连接到 DevTools。

如下图标记处，就是新打开 Page 页面的 WebSocket URL：



04

代码封装

前面几步介绍了手动操作，从打开启动 Chrome 并开启远程调试到获取 WebSocket 调试 URL。

接下来我们需要将上面内容使用 Python 进行封装。

本次代码封装我们做到两个功能，第一需要一个命令来启动 Chrome 浏览器，并通过远程调试功能连接到特定页面。第二需要支持无头模式、自定义窗口大小、用户数据目录等参数。

在类结构上，我们需要设计一个 Browser 类和 Page类，Browser 类负责启动和管理 Chrome 浏览器；Page 类继承自 Browser 类，负责连接到特定页面并进行 WebSocket 通信。

下面对这两个类中方法或变量进行说明：

1. Browser 类：

- **默认值：**
  定义一些默认值，如 Chrome 的路径、远程调试端口等。
- **初始化方法：**
  允许用户在实例化时覆盖默认值。
- **设置方法：**
  提供动态设置参数的功能。
- **启动 Chrome：**
  构造启动参数并使用 subprocess.Popen 启动 Chrome。
- **关闭 Chrome：**
  确保在程序退出时关闭 Chrome。
- **获取页面 WebSocket URL：**

发送 HTTP 请求到远程调试端口，解析返回的 JSON 数据，找到对应的 WebSocket URL。

2. Page 类：

- **初始化方法：**
  调用父类的初始化方法，设置 WebSocket URL，默认为第一个页面的 WebSocket URL。

- **连接到页面：**
  使用 websocket.create_connection 方法连接到指定的 WebSocket URL。

- **获取默认页面 WebSocket URL：**
  获取第一个页面的 WebSocket URL。

根据上面逻辑实现的代码如下：

```
import atexit
import subprocess
import time
import requests
from websocket import create_connection

class Browser:
    # 设置默认值
    _chrome_path = "/Applications/Google Chrome.app/Contents/MacOS/Google Chrome"
    _remote_debugging_port = 9222
    _user_data_dir = "/users/ydj/chrome_profile"
    _remote_allow_origins = "*"
    _headless = False
    _window_size = ("1280", "768")

    def __init__(self,
                 chrome_path=None,
                 remote_debugging_port=None,
                 user_data_dir=None,
                 remote_allow_origins=None,
                 headless=None,
                 window_size=None,
                 ):
        self.chrome_path = chrome_path or self._chrome_path
        self.remote_debugging_port = remote_debugging_port or self._remote_debugging_port
        self.user_data_dir = user_data_dir or self._user_data_dir
        self.remote_allow_origins = remote_allow_origins or self._remote_allow_origins
        self.headless = headless or self._headless
        self.window_size = window_size or self._window_size
        self.__process = None
        # 注册清理函数
        atexit.register(self.close_chrome)

    @property
    def chrome_path(self):
        return Browser._chrome_path

    @chrome_path.setter
    def chrome_path(self, value):
        Browser._chrome_path = value

    @property
    def remote_debugging_port(self):
        return Browser._remote_debugging_port

    @remote_debugging_port.setter
    def remote_debugging_port(self, value):
```

```
    def remote_debugging_port(self, value):
        Browser._remote_debugging_port = value

    @property
    def user_data_dir(self):
        return Browser._user_data_dir

    @user_data_dir.setter
    def user_data_dir(self, value):
        Browser._user_data_dir = value

    @property
    def remote_allow_origins(self):
        return Browser._remote_allow_origins

    @remote_allow_origins.setter
    def remote_allow_origins(self, value):
        Browser._remote_allow_origins = value

    @property
    def headless(self):
        return Browser._headless

    @headless.setter
    def headless(self, value):
        Browser._headless = value

    @property
    def window_size(self):
        return Browser._window_size

    @window_size.setter
    def window_size(self, value):
        Browser._window_size = value

    def start_chrome(self):
        # 定义Chrome的完整路径和启动参数
        chrome_args = [
            self.chrome_path,
            f"--remote-debugging-port={self._remote_debugging_port}",
            f"--user-data-dir={self._user_data_dir}",
            f"--remote-allow-origins={self._remote_allow_origins}",
            f"{'--headless=new' if self._headless else ''}",
            f"--window-size={self._window_size}",
        ]

        # 启动Chrome浏览器
        self.__process = subprocess.Popen(chrome_args)
        # 等待一段时间，确保Chrome已经启动
        time.sleep(5)

    def close_chrome(self):
        # 关闭Chrome浏览器
        if self.__process:
            self.__process.terminate()
            self.__process.wait()

    def __del__(self):
        self.close_chrome()

    def get_page_ws_url(self, page_title=None, page_index: int = None):
        # 获取 Chrome 的 WebSocket 调试 URL
        resp = requests.get(f'http://127.0.0.1:{self.remote_debugging_port}/json')
        assert resp.status_code == 200, "Failed to get page information from Chrome"
        resp_json = resp.json()
        if page_index is not None:
            ws_url = resp_json[page_index].get('webSocketDebuggerUrl')
        elif page_title is not None:
            ws_urls = [item['webSocketDebuggerUrl'] for item in resp_json if item.get('title') == pa
            if ws_urls:
                ws_url = ws_urls[0]
            else:
```

```
                raise ValueError(f"No page found with title: {page_title}")
        else:
            ws_url = resp_json[0].get('webSocketDebuggerUrl')
        return ws_url

class Page(Browser):
    def __init__(self, ws_url=None):
        super().__init__()
        self.ws_url = ws_url or self.get_default_page_ws_url()
        self.connection = self.connect_to_page(self.ws_url)

    def __repr__(self):
        return f"Page(ws_url={self.ws_url!r})"

    def connect_to_page(self, ws_url):
        # 连接页面
        return create_connection(ws_url)

    def get_default_page_ws_url(self):
        # 获取第一个页面 ws url
        return super().get_page_ws_url()

# 示例用法
if __name__ == '__main__':
    # 实例化时给值
    browser = Browser(remote_debugging_port=9223)
    # 单独赋值
    browser.remote_debugging_port = 9224
    browser.start_chrome()
    page = Page()
    print(f"Connected to page with WebSocket URL: {page.ws_url}")
```

```
145
146
147
148
149
```

上面代码中：

1. Browser 类

Browser 类用于启动和管理一个 Chrome 浏览器实例，支持远程调试功能。以下是其主要功能和方法：

- **初始化方法 (__init__)**：
  - 设置 Chrome 的路径、远程调试端口、用户数据目录、允许的远程来源、是否无头模式以及窗口大小。
  - 注册 close_chrome 方法，确保在程序退出时关闭 Chrome 浏览器。
- **属性装饰器**：
  - @property 和 @<property>.setter 装饰器用于管理类变量的访问和修改。
- **启动 Chrome (start_chrome)**：
  - 构造 Chrome 的启动参数列表。
  - 使用 subprocess.Popen 启动 Chrome 浏览器。
  - 等待 5 秒，确保 Chrome 已经启动。
- **关闭 Chrome (close_chrome)**：
  - 如果浏览器进程存在，则终止并等待其退出。
- **获取页面 WebSocket URL (get_page_ws_url)**：
  - 发送 HTTP 请求到 Chrome 的远程调试端口，获取页面信息。
  - 根据提供的 page_title 或 page_index，找到对应的 WebSocket URL。
  - 如果没有找到匹配的页面，抛出 ValueError。

2. Page 类

Page 类继承自 Browser 类，用于连接到一个特定的 Chrome 页面，并通过 WebSocket 进行通信。以下是其主要功能和方法：

- **初始化方法 (__init__)**：
  - 调用父类的初始化方法。
  - 设置 WebSocket URL，默认为第一个页面的 WebSocket URL。
  - 连接到页面。
- **连接到页面 (connect_to_page)**：

- 使用 websocket.create_connection 方法连接到指定的 WebSocket URL。

- **获取默认页面 WebSocket URL (get_default_page_ws_url)：**
  - 调用父类的 get_page_ws_url 方法，获取第一个页面的 WebSocket URL。

运行上面代码，首先实例化 Browser 类，并设置远程调试端口为 9223，然后修改为 9224，接着启动 Chrome 浏览器；

最后创建 Page 实例，连接到第一个页面，并打印 WebSocket URL。

代码运行成功后，控制台会成功输出第一个页面的 WebSocket URL。

如下图：

```
DevTools listening on ws://127.0.0.1:9224/devtools/browser/0a95109e-4977-4f1c-8856-54e741f682bd
Connected to page with WebSocket URL: ws://127.0.0.1:9224/devtools/page/6A76AA1A4867BBA67FF7CEE1EB85ED58
```

.....

本文节选自第八十六期《51测试天地》
原创文章《打造基于 WebSocket + CDP 的 Selenium 替代方案》
文章后续将继续为大家介绍
“如何发送 CDP 命令及添加其他操作”
想继续阅读全文或查看更多《51测试天地》的原创文章
请点击下方 阅读原文或扫描二维码 查看

每日有奖互动

你在项目中用过其他自动化测试方案吗？
和本文的方案相比，体验如何？


51Testing软件测试网
上海博为峰软件技术股份有限公司
上海
已关注公众号
发消息
51Testing软件测试网
进店 >
2.
设为星标
推荐给朋友
投诉


填问卷
100%中奖
免费参与抽奖
公众号 · 51Testing软件测试网