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

点击蓝字，立即关注

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。

前面两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，如何启动Chrome 并开启远程调试**。

接下来继续为大家介绍如何**发送 CDP 命令及添加其他操作**。

发送 CDP 命令

CDP 连接成功后，我们就可以发送 CDP 命令来操作浏览器。

步骤一：

首先我们先在 Page 类下创建一个 send_command 方法，用于向 Chrome 的远程调试接口发送命令，并接收响应。

这个方法利用了 WebSocket 连接来与 Chrome 通信，发送 JSON 格式的命令，并接收 JSON 格式的响应。

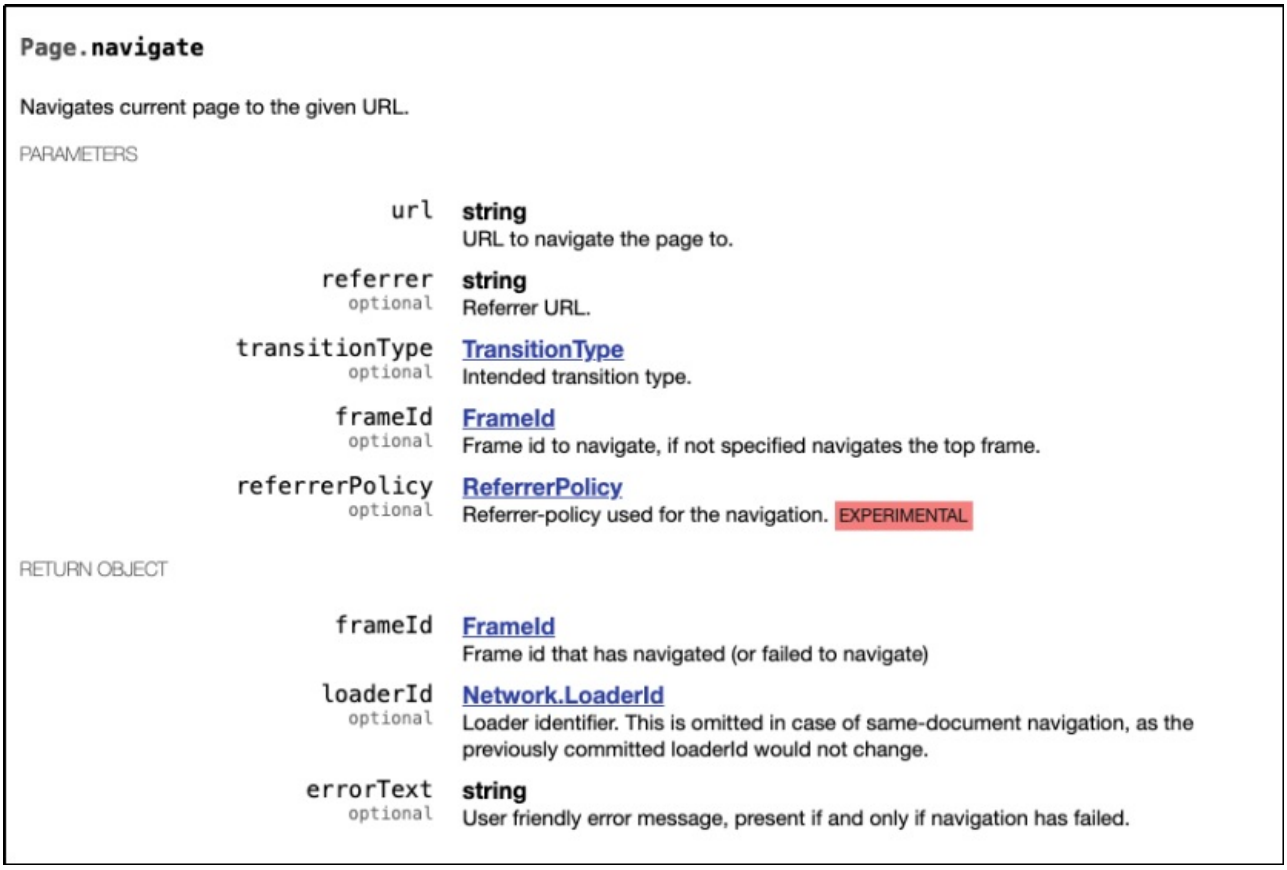
代码如下：

```
class Page(Browser):
    """
    省略其他代码
    """

    def send_command(self, command):
        # 发送 命令
        self.connection.send(json.dumps(command))
        return json.loads(self.connection.recv())

```

步骤二：

然后我们添加打开网页的操作。

首先在 CDP 协议网站
https://chromedevtools.github.io/devtools-protocol/ 找到打开网页操作的命令，如下截图所示：



Page.navigate

Navigates current page to the given URL.

PARAMETERS

url		**string** URL to navigate the page to.
referrer	optional	**string** Referrer URL.
transitionType	optional	TransitionType Intended transition type.
frameId	optional	FrameId Frame id to navigate, if not specified navigates the top frame.
referrerPolicy	optional	ReferrerPolicy Referrer-policy used for the navigation. EXPERIMENTAL

RETURN OBJECT

frameId		FrameId Frame id that has navigated (or failed to navigate)
loaderId	optional	Network.LoaderId Loader identifier. This is omitted in case of same-document navigation, as the previously committed loaderId would not change.
errorText	optional	**string** User friendly error message, present if and only if navigation has failed.

然后根据如下格式进行编辑命令：

```
command = {
    "id": 1,
    "method": method,
    "params": {
        "key": value
    }
}
```

例如打开网页的操作命令就可写成：

```
command = {
    "id": 1,
    "method": "Page.navigate",
    "params": {
        "url": url
    }
}
```

步骤三：

下面我们就可以在 Page 类下添加对应的操作方法。

例如在 Page 类下封装 navigate_to 方法，用于让 Chrome 浏览器导航到指定的 URL。

它通过发送一个 Page.navigate 命令到 Chrome 的远程调试接口来实现这一功能。

此方法利用了刚才封装的 send_command 方法来发送命令并处理响应。

代码如下：

```
class Page(Browser):
    """
    省略其他代码
    """

    def navigate_to(self, url):
        command = {
            "id": 1,
            "method": "Page.navigate",
            "params": {
                "url": url
            }
        }
        self.send_command(command)


```

步骤四：

接下来进行验证，如下代码：

```
if __name__ == '__main__':
    browser = Browser()
    browser.start_chrome()
    page = Page()
    page.navigate_to("http://www.cnblogs.com/tynam")
    time.sleep(10)
```

运行上面代码可以看到，启动浏览器后会访问 “http://www.cnblogs.com/tynam” 网页。

添加其他操作

有了上面添加浏览器操作的经验后，我们便同理添加其他操作。如下：

1. 浏览器打开新的 Tab 页面

```
def new_page(self, new_window: bool = False):
    # 打开新页面
    # new_window：是否在当前浏览器窗口打开
    # 发送命令以创建新标签
    command = {
        "id": 1,
        "method": "Target.createTarget",
        "params": {
            "url": "",
            "newWindow": new_window
        }
```

```
    }
    # 接收响应，获取新页面的 targetId
    response = self.send_command(command)
```

2. 切换 Tab 页面

```
def switch_page(self, page_title=None, page_index: int = None):
    # 切换页面
    ws_url = self.get_page_ws_url(page_title, page_index)
    self.connection = self.connect_to_page(ws_url)
```

3. 页面刷新

```
def page_refresh(self):
    command = {
        "id": 1,
        "method": "Page.reload",
        "params": {}
    }

    self.send_command(command)
```

4. 获取页面 title

```
def get_title(self):
    command = {
        "id": 1,
        "method": "Page.getNavigationHistory",
        "params": {}
    }

    response = self.send_command(command)
    if 'result' in response and 'currentIndex' in response['result']:
        result = response['result']
        page_info = result["entries"][result['currentIndex']]
        return page_info['title']
    else:
        raise ValueError("Failed to get page title")
```

5. 获取页面 url

```
def get_url(self):
```

```
    command = {
        "id": 1,
        "method": "Page.getNavigationHistory",
        "params": {}
    }

    response = self.send_command(command)
    if 'result' in response and 'currentIndex' in response['result']:
        result = response['result']
        page_info = result["entries"][result['currentIndex']]
        return page_info['url']
    else:
        raise ValueError("Failed to get page url")
```

上面只是基本的页面操作，更多内容可参考 CDP 命令的 Page 操作。

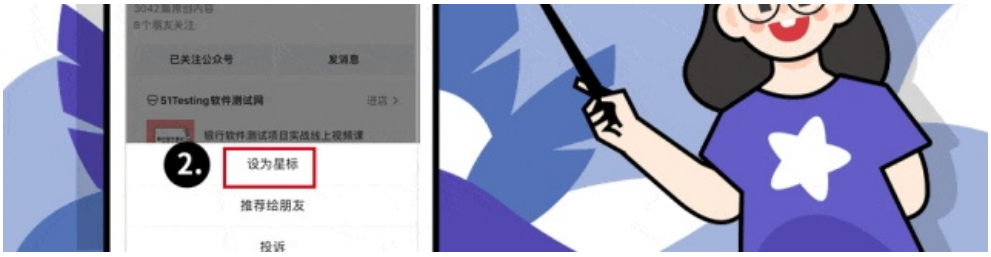
由于篇幅问题，在此就不再往下补充了。感兴趣的同学可继续往下探索，例如探索如何获取页面元素并进行点击。

.....

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》
文章后续还为大家进行了总结
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

你认为 CDP + WebSocket 是未来浏览器自动化的趋势吗？
为什么？


已关注公众号
发消息
51Testing软件测试网
进店 >
银行软件测试项目实战线上视频课
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