打造基于 WebSocket + CDP 的 Selenium 替代方案

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。

概述

在进入实操之前，先赘叙下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。

01 WebSocket

WebSocket 是一种网络通信协议，提供了在单个 TCP 连接上进行全双工通信（客户端和服务器可以同时发送和接收消息，而无需等待对方的响应。这使得实时数据交换变得更加高效和流畅。）的能力。

它允许客户端和服务器之间进行实时、双向的数据交换，而无需像传统的 HTTP 请求那样频繁地建立和关闭连接。

例如你去商店买东西，你问店员商品价格，店员告诉你，然后你就走了，这就是一次 HTTP 请求；而 WebSocket 可以在客户端（比如你的电脑、手机）和服务器之间建立一个持久的连接。

这个连接一旦建立，双方就可以随时互相发送数据，就像打电话，电话一直通着，我们可以随时聊天，想说啥就说啥，不用像发短信那样还得等对方回复才能再发下一条。

WebSocket 支持传输文本数据和二进制数据，这使得它可以用于各种应用场景，如实时文本聊天、在线游戏、文件传输等。

他也得到了大多数现代浏览器的支持，包括 Chrome、Firefox、Safari 和 Edge。

此外，也有许多服务器端实现，如 Node.js 的 ws 库、
Python 的 websocket-client 和 websockets 库等。

02 CDP

Chrome DevTools Protocol (CDP) 是一套开放协议，允许外部程序通过 Chrome 浏览器提供的接口与其进行交互。

CDP 提供了丰富的功能，使开发者可以远程控制 Chrome 浏览器，包括操作 DOM、监控网络请求、调试代码、截取屏幕快照等。

CDP 的核心特点有：

- 基于 **JSON-RPC**：
  CDP 协议使用 JSON 格式传输数据，简单易读，易于解析和生成。

- **双向通信：**
  不仅调试器可以发送命令，浏览器也会主动推送事件（比如断点触发、网络请求完成）。

- **模块化设计：**
  CDP 协议分为多个模块（如 DOM、Network、Runtime 等），每个模块负责不同的功能。

CDP 的工作流程如下，这也是本文介绍的方案的工作流程：

1. 建立 WebSocket 连接：通过 WebSocket 与浏览器内核建立连接。

2. 发送协议命令：客户端发送 JSON 格式的命令。

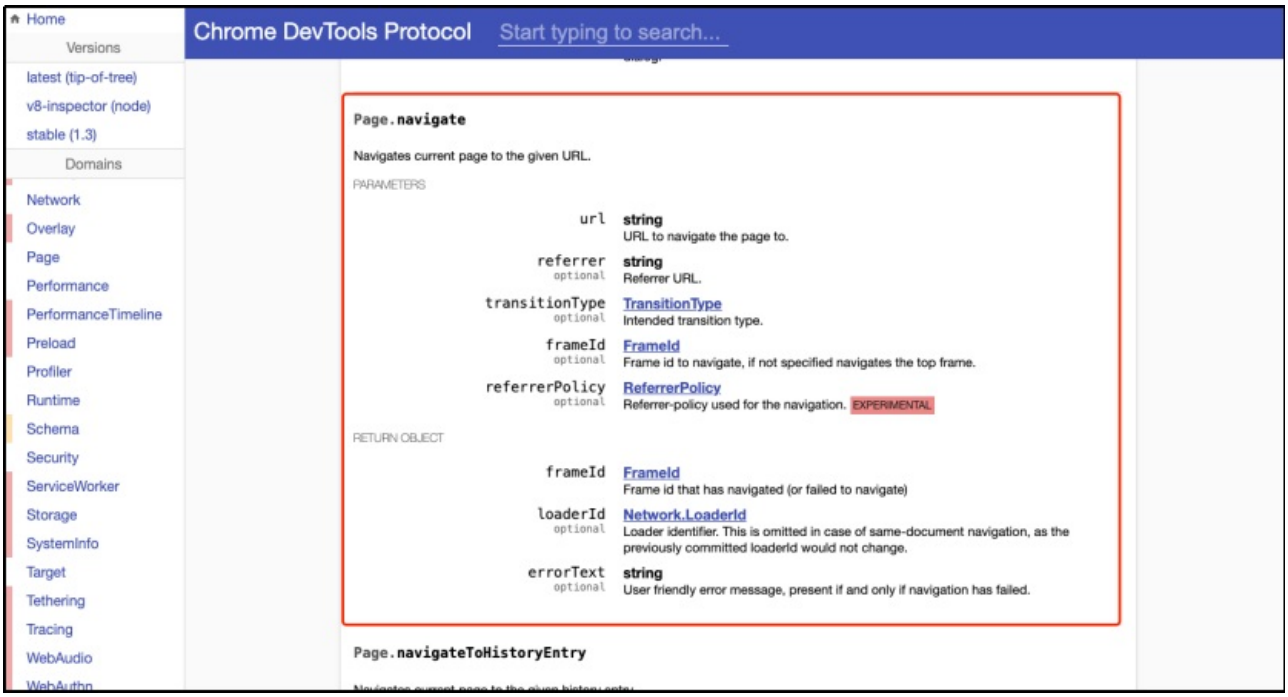
3. 执行协议命令：浏览器内核执行命令并返回结果。

4. 接收结果：客户端接收并显示结果。

在发送协议命令阶段，我们需要知道CDP都提供了哪些命令，这也是学习本文时需要重点关注的一点。

访问CDP协议网站
https://chromedevtools.github.io/devtools-protocol/ 可获取 CPD 协议支持的命令或提供的方法。

例如 Page.navigate 方法，如下图：



Page.navigate 方法用于将当前页面导航到指定的 URL。它支持多种参数，允许开发者指定导航的详细行为，如引用页、过渡类型等。

该方法返回一个对象，包含导航结果的相关信息。

提供的参数有：

- **url (string)：**
  指定浏览器将要加载的页面地址。

- **referrer (string)**：
  指定导航请求的来源页面。这在某些情况下会影响目标页面的行为，例如，某些网站会根据引用页来设置不同的内容或行为。

- **transitionType (TransitionType)**：
  指定导航的类型，这在某些情况下会影响浏览器的行为，例如，某些过渡类型可能会触发不同的缓存策略。

- **frameId (FrameId)**：
  指定要导航的具体框架。在多框架页面中，这允许开发者控制特定框架的导航行为。

- **referrerPolicy (ReferrerPolicy)**：
  指定引用页信息的发送策略，这在处理安全性和隐私问题时非常重要。

可设置的返回值有：

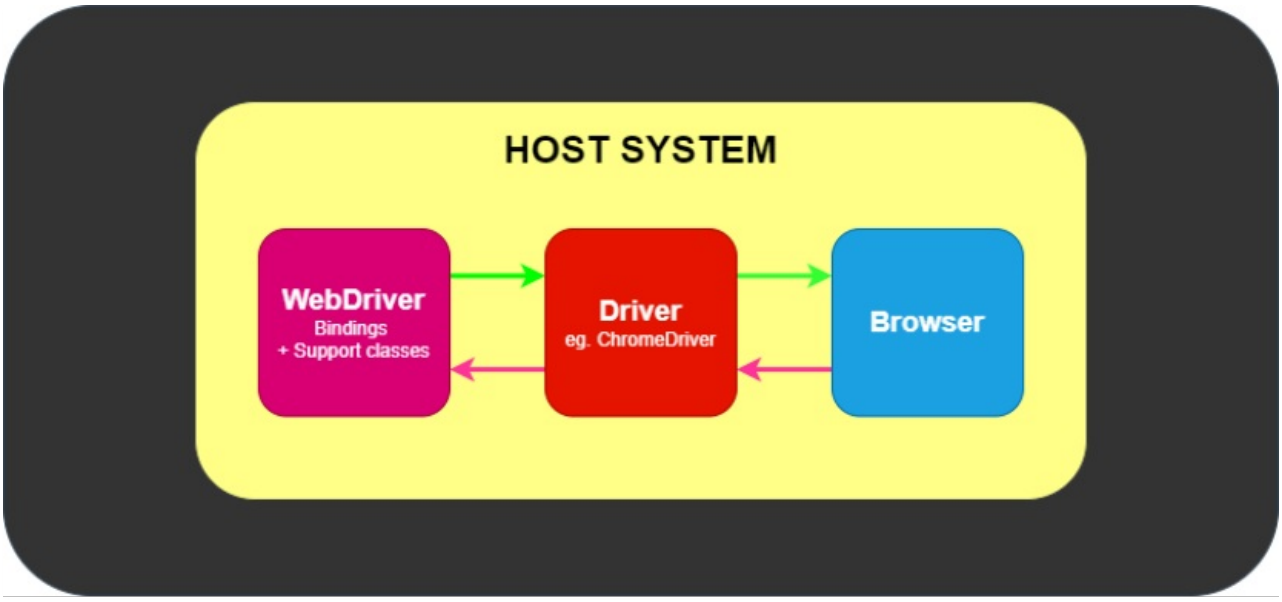
- **frameId (FrameId)**：
  返回导航操作所涉及的框架 ID，便于后续操作和调试。

- **loaderId (Network.LoaderId)**：
  加载器标识符。在同文档导航的情况下，此字段会被省略，因为之前提交的加载器 ID 不会改变。

- **errorText (string)**：
  在导航失败时，提供详细的错误信息，便于调试和问题解决。

03 Selenium 与浏览器通信流程

Selenium与浏览器通信的基本流程图如下：



图片来源：
https://www.selenium.dev/zh-cn/documentation/overview/components/

图中由三部分内容组成，WebDriver、Driver 和 Browser。

其中 WebDriver 是一个抽象接口，定义了通用的方法和属性，适用于所有支持 Selenium 的浏览器；

driver 是 WebDriver 接口的具体实现，负责与特定的浏览器进行通信。每个浏览器都有自己的驱动程序，如 ChromeDriver、GeckoDriver 等；

Browser 就是支持的浏览器，例如 Chrome、Firefox等。

基本通信流程为：

1. 客户端代码通过 Selenium WebDriver API 发送命令。

2. WebDriver 将命令序列化为 JSON 格式，并通过 HTTP 请求发送到浏览器驱动。

3. 浏览器驱动将命令转发给浏览器。

4. 浏览器执行命令，并将结果返回给浏览器驱动。

5. 浏览器驱动将结果序列化为 JSON 格式，并通过 HTTP 响应发送回客户端代码。

客户端代码接收到响应，并根据结果进行后续操作。

而本文操作流程与之类似，如下：

1. 启动 Chrome 并开启远程调试功能。

2. 建立 WebSocket 连接：通过 WebSocket 与浏览器内核建立连接。

3. 发送协议命令：客户端通过代码发送 JSON 格式的 CDP 命令。

4. 执行协议命令：浏览器内核执行命令并返回结果。

5. 接收结果：客户端接收并显示结果。

与 Selenium 通信相比，我们将直接通过 CDP 与 Chrome 的内部机制交互，绕过了中间的 WebDriver 层，从而减少了不必要的开销。

04 为什么选择 WebSocket + CDP

- **性能优势：**

  使用 WebSocket 和 CDP 的主要优势之一是性能。与 Selenium 相比，CDP 直接与 Chrome 的内部机制交互，绕过了中间的 WebDriver 层，从而减少了不必要的开销。

  这使得操作更加高效，响应时间更短，特别适合需要高效率的自动化测试和爬虫任务。

- **功能强大：**

  CDP 提供了对浏览器的全面控制，包括页面导航、DOM 操作、网络请求拦截、性能分析等。

  这些功能不仅涵盖了 Selenium 的所有功能，还提供了许多 Selenium 难以实现的高级功能。

例如，通过 CDP，你可以拦截和修改网络请求，这对于测试和开发复杂的 Web 应用非常有用。

- **轻量级：**

  与 Selenium 相比，基于 WebSocket 和 CDP 的解决方案更加轻量级。

  它不需要安装庞大的 WebDriver，减少了依赖项和资源消耗。这使得你的自动化脚本更加简洁，部署更加方便。

- **易于扩展：**

  通过 WebSocket 和 CDP，你可以轻松实现各种高级功能，如监听页面加载完成事件、进行 DOM 操作等。

  这些功能可以根据你的需求进行灵活扩展，使得你的自动化工具更加强大和适应不同的应用场景。

.....

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》
文章后续还对
“启动Chrome 并开启远程调试、及添加其他操作”
进行了详细介绍及总结
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？


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