Node 使用 Selenium 进行前端自动化操作

饭等米

前言：

最近项目中有类似的需求：需要对前端项目中某一个用户下的产品数据进行批量的处理。手动处理的流程大概是首先登录系统，获取到当前用户下的产品列表，点击产品列表的中产品项进入详情页，对该产品进行一系列的操作，然后保存退出。因为当前有 20 多万条数据，手动一条一条的处理不太现实，所以希望通过写脚本的方式来进行处理。

需求分析

其实这个需求还算比较简单，需要实现的点主要有三个，**一是**如何进行登录，获取登录信息，查询当前用户下的产品数据；二是如何知道当前数据是否处理完，然后退出当前的处理流程；三是如何异步的处理一批数据。

所以需要做的工作就是模拟登录，调用产品列表的查询接口获取产品 ID 集合，然后循环遍历当前的集合，通过产品 ID 跳转产品详情页面，模拟页面按钮的点击操作，监听处理完成的动作，退出当前的流程。

Selenium 介绍

What is Selenium?

Selenium automates browsers. That's it! What you do with that power is entirely up to you. Primarily, it is for automating web applications for testing

purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) be automated as well.

Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

翻译过来大致意思就是： Selenium 可以自动化操作浏览器。怎么去使用 Selenium 的功能完全取决于我们自己。它主要还是使用在 web 应用的自动化测试上。但是他的功能并不仅限于此。那些枯燥的基于 web 的管理任务也可以自动化。很多流行的浏览器都采取了一些措施来支持 Selenium 实现本地化。它也是很多浏览器自动化工具、API 自动化以及框架的核心技术。

Selenium 主要分 Selenium WebDriver 以及 Selenium IDE。我主要结合 Node 来介绍 Selenium WebDriver 的安装使用。本文主要介绍 Selenium 结合 Node 的安装使用。需要进行深入研究的同学请自行查看官网文档。

Node 环境搭建

1. node 的安装在此不再赘述。点击链接查看官网下载安装方法。

2. express 安装

```
$ npx express-generator
```

或者

```
$ npm install -g express-generator
```

创建项目：

```
$ express --view=ejs selenium-start

$ cd selenium-start

$ yarn
```

启动项目：

```
$ DEBUG=myapp:* yarn start
```

至此，Node 项目创建完毕。接下来我们就可以在项目中集成 Selenium WebDriver

Selenium WebDriver 集成

1. 安装 selenium-webdriver

```
yarn add selenium-webdriver
```

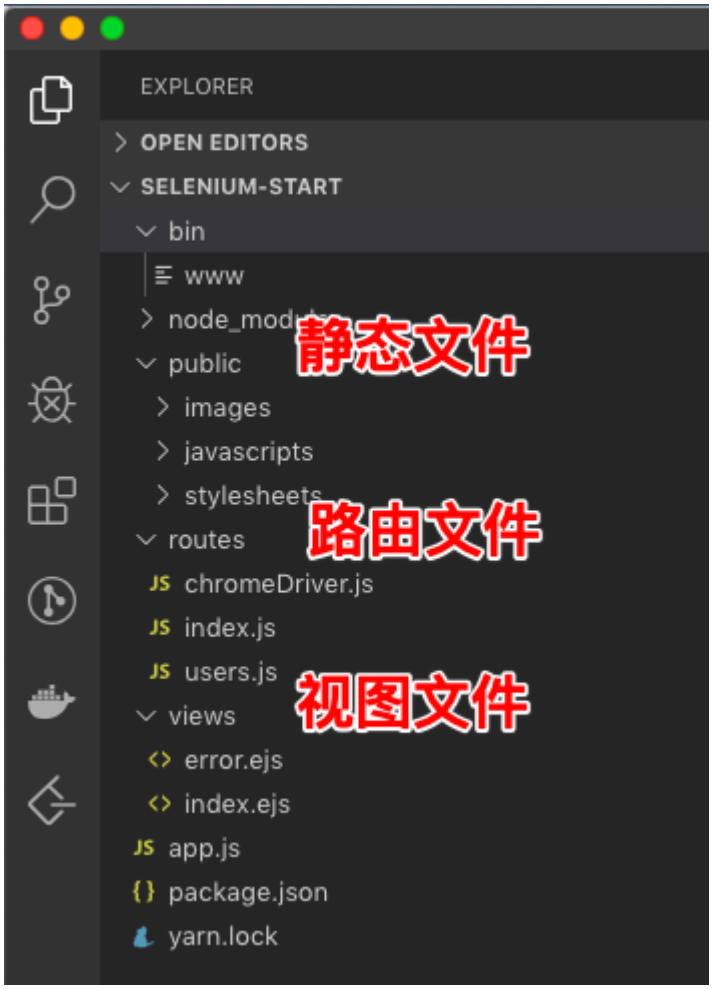
2. 下载安装支持不同浏览器的驱动。（此处只介绍 Chrome 驱动）

```
[ChromeDriver][3]
```

下载并解压文件，同时把解压的执行文件放置到 /usr/bin 目录下。或者设置相应的 PATH 路径，确保可执行文件在 PATH 路径中。

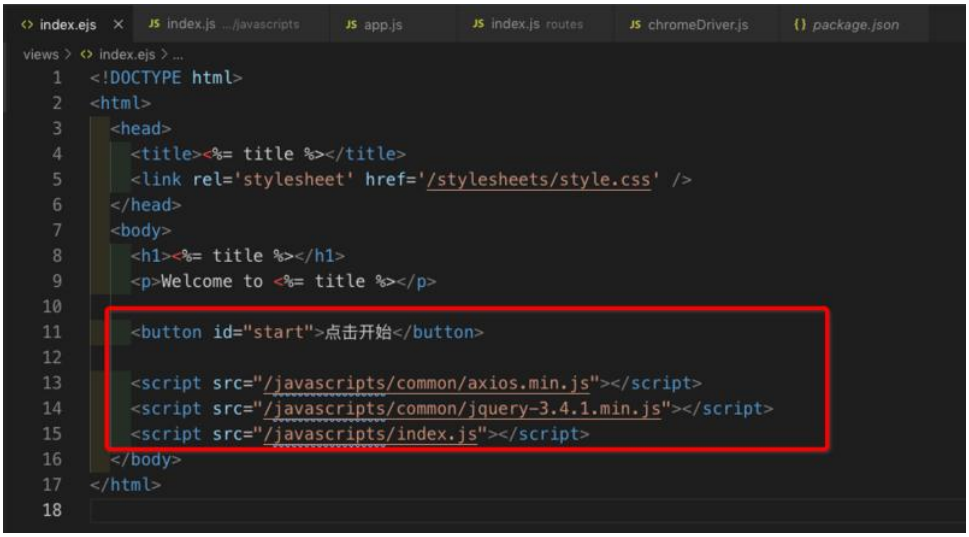
开始使用

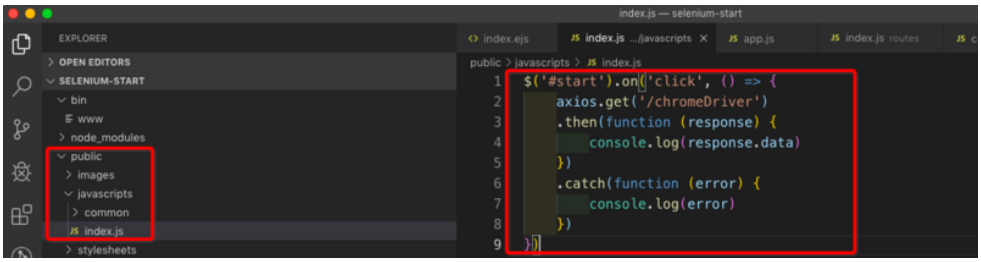
进入我们刚才创建的项目文件夹，目录如下：



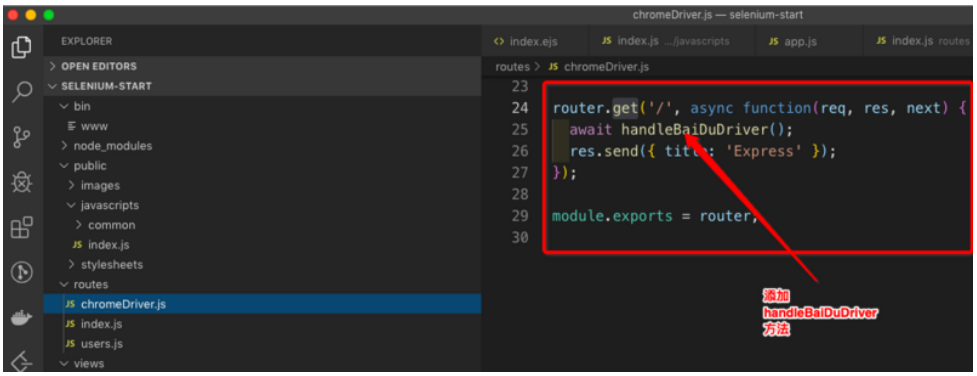
1. 页面添加一个开始按钮，以及给按钮添加事件。

   找到 views/index.ejs，添加如下代码：(为了方便操作，引入了 jquery, axios，所以需要下载准备好)





2. 添加对应的路由



在 app.js 文件中，引入路由 chromeDriver

```
var chromeDriverRouter = require('./routes/chromeDriver');

app.use('/chromeDriver', chromeDriverRouter);
```

3. 引入 selenium-webdriver

在 routes/chromeDirver.js 文件中，我们添加了一个方法

handleBaiDuDriver，这个方法用于处理模拟百度搜索自动化的一些测试。

首先我们需要在文件顶部引入 selenium-webdriver

```
const {Builder, By, Key, until} = require('selenium-webdriver');



// Builder: 用于创建一个 WebDriver 实例。

// By: 表示通过什么方式来查找页面的元素。

    // By.className( name ) → By

    // By.css( selector ) → By
```

```
    // By.id( id ) → By

    // By.js( script, ...var_args ) → function(WebDriver): Promi
se

    // By.linkText( text ) → By

    // By.name( name ) → By

    // By.partialLinkText( text ) → By

// Key: 表示键盘上一系列的按键。

// until:  定义了一些工具类的方法。
```

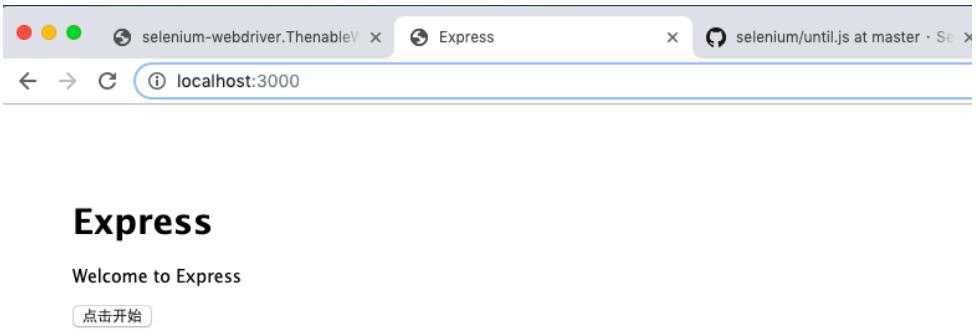
然后书写我们的方法体里的内容。

```
const handleBaiDuDriver = async () => {

    let driver = await new Builder().forBrowser('chrome').build
();

    try {

        await driver.get('http://www.baidu.com');



        await driver.findElement(By.id('kw')).sendKeys('webdriver
', Key.RETURN);//正常使用



        await driver.findElement(By.id('su')).click();



        await driver.wait(until.titleIs('百度一下，你就知道'), 100
0);
```

```
    } catch (error) {
        console.log(error)
    } finally {
        await driver.sleep(2000);
        await driver.quit();
    }
}
```

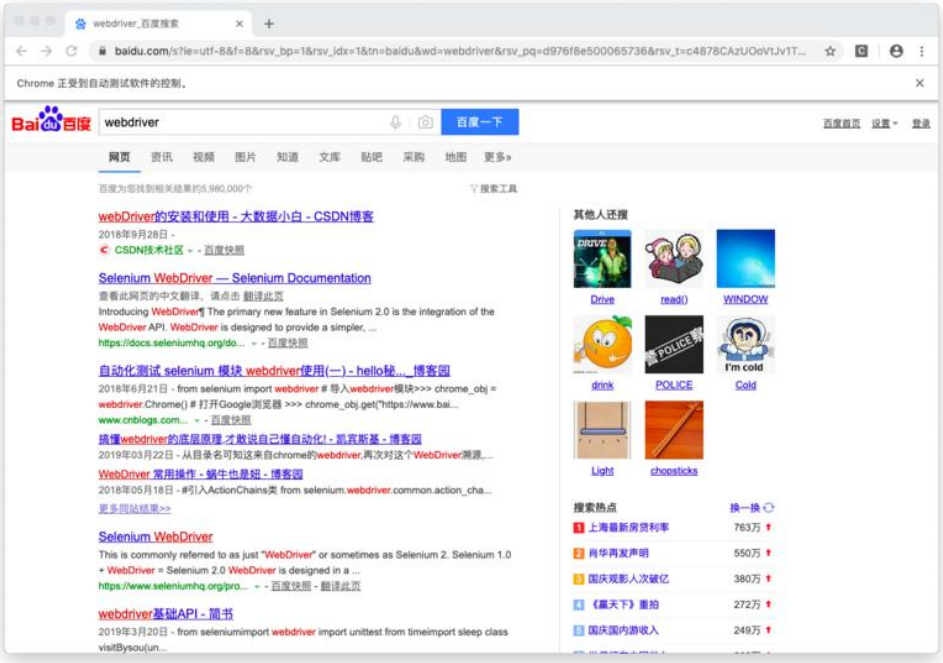
4. 启动服务，查看效果。

   启动服务之后，我梦能看到如下的界面。



   点击页面中的【点击开始】按钮，最终能够看到如下的界面，为了演示我做了两秒的延迟。生成的 gif 图有 9M 多，无法上传。后续可以下载源码运

行看效果。



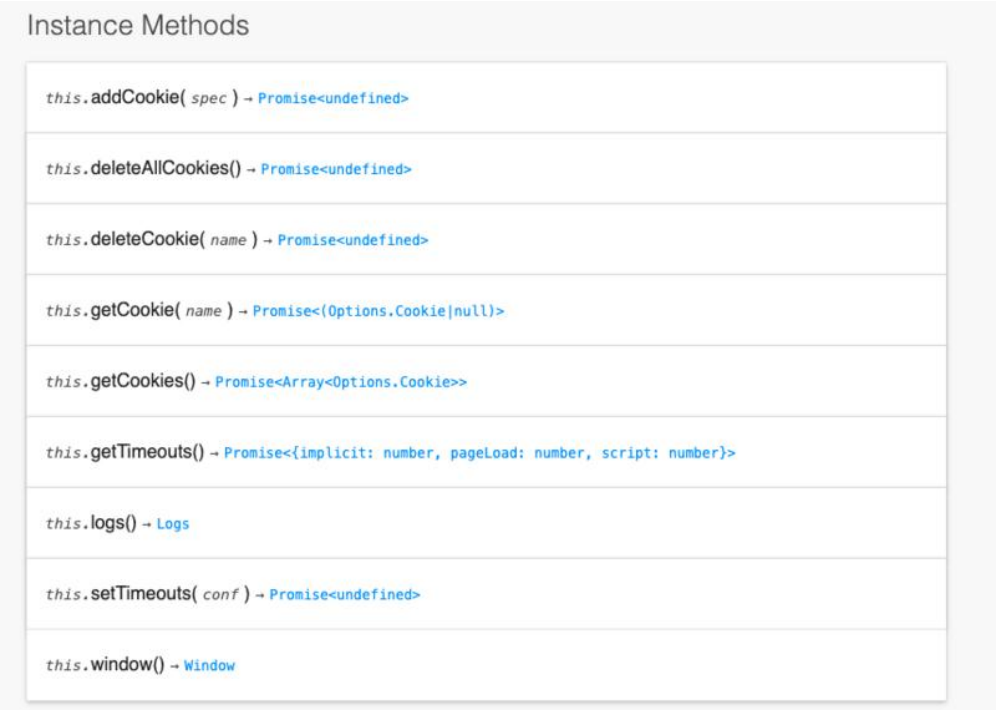
5. 获取登录信息

以上是 selenium-webdriver 的简单集成。在之前我们提到过实际需求中如何获取登录信息的问题。在访问产品列表页面的时候需要进行登录校验。如果没有登录则会跳转界面。由于我们的登录页是通过 iframe 来嵌套引入的。由于暂时还没有了解如何处理 iframe 里的操作，所以没法去模拟用户名密码的输入。

查看 API 文档，WebDriver 会有一个 manage 方法：

```
this.manage() → Options
```

该方法会返回一个 Options 实例，具有如下的方法：



其中有对 cookie 的操作方法。所以可以通过首次输入用户信息并进行缓存的方式来实现登录态的保存。在下一次再打开页面的时候直接从缓存里获取 cookie 信息，并通过 addCookie 方法进行 cookie 的设置。但是由于我不知道什么时候、多长时间登录才会成功，所以在获取 cookie 的时候需要通过不断循环的方式去获取，直到拿到 cookie。当然可以设置一个超时时间。超时之后就退出当前 driver。

```
// 缓存 cookie
async function setCookies(driver) {
    const manage = driver.manage();
    let sleepTime = 6000;
    await driver.sleep(sleepTime);
```

```
let cookies = null
try {
    cookies = await manage.getCookies();
} catch (error) {
}
while (!cookies || !findSessionIdFromCookies(cookies)) {
    await driver.sleep(2000)
    sleepTime += 2000;
    try {
        cookies = await manage.getCookies();
    } catch (error) {
    }
}
if (cookies && findSessionIdFromCookies(cookies)) {
    cache.cookies = cookies; // cache 是全局用于缓存 cookie 的对
象

    cache.cookiesStr = cache.cookies.map((cookie) => {
        return `${cookie.name}=${cookie.value}`
    }).join(';');
}
return cookies;
```

```
}



// 设置 cookie
async function initCookies(driver) {
    const cookies = cache.cookies;
    if (cookies && cookies.length > 0) {
        await driver.manage().deleteAllCookies();
        for (let i = 0 ; i < cookies.length; i++) {
            cookie = cookies[i];
            await driver.manage().addCookie(cookie);
        };
    }
}
```

获取到 cookie 信息之后就可以请求产品列表以及通过产品 ID 进入产品详情页。

然后再模拟页面按钮点击操作即可。