如何利用 Puppeteer 爬取数据？

作为一个程序员，我跟大多数人一样，都不善言辞，以前也尝试过录制视频，很不理想，无意中发现了 GitChat 这个功能，对我们这些不太会录视频的人简直是福音，在这个人人自媒体的时代，大家都可以发光发热，把自己所了解的跟大家共同分享，真是一件大好事儿。

我们先来看下文章的整体结构。

GitChat

引言

今天给大家分享下 Puppeteer 的部分功能——抓取网页内容。本文所演示的环境是在 MacOS，其他平台类似，如有问题，请提出，大家共同讨论解决。

介绍 Puppeteer

Puppeteer 是一个提供强大的 API 用来控制操作 Chrome 的类库。通俗点儿说，你可以通过代码的方式模拟人在 Chrome 中的各种操作，打开网址、开启多个 Tab、填写输入框，模拟鼠标轨迹、滚动滚动条，甚至截屏某个元素都可以。具体详细的内容可以参看 Puppeteer 官方和 API 文档。

安装

安装 Node.js 和 npm

npm 是和 Node.js 一起发布的，所以下载安装 Node.js 之后，就可以使用 npm 了，可以点击这里下载安装对应的版本。执行下面的内容，显示出版本信息就证明安装成功了。

```
/tmp# node -v
v9.3.0
/tmp# npm -v
5.6.0
```

初始化项目

（1）新建文件夹 xxx。

```
/tmp# mkdir spider-puppeteer
/tmp# cd spider-puppeteer
```

GitChat

（2）初始化 npm 项目。

会生成一个配置文件 package.json，这个文件记录了我们项目的依赖，移植起来会很方便。

```
/tmp/spider-puppeteer# npm init -y
/tmp/spider-puppeteer# ll
-rw-r--r--  1 ritoyan  wheel  230  3 11 09:58 package.json
```

安装 Puppeteer

```
/tmp/spider-puppeteer# npm install --save puppeteer
```

因为要下载 Chrome 内核，直接用上面的代码，国内大部分时候会失败，原因你懂得。如果安装的时候出现类似 `ERROR: Failed to download Chromium r536395! Set "PUPPETEER_SKIP_CHROMIUM_DOWNLOAD" env variable to skip download.{ Error: connect ETIMEDOUT` 的字样，这意味着你被无情的挡在了墙里面。

安装失败，就需要手动去下载了，墙再高也挡不住大家热爱知识的热情。分两步，第一步安装 Puppeteer 代码，第二部手动下载他所依赖的 Chromuim 内核。

（1）安装 Puppeteer 代码。

```
/tmp/spider-puppeteer# npm install --save puppeteer --ignore-
scripts
```

（2）手动下载 Chromuim。

点击这里下载，点进去之后就是对应平台的版本，直接下载就行。下载成功后解压到我们的项目目录 spider-puppeteer 下面 `mv ~/Downloads/chrome-mac/Chromium.app ./Chromium/Chromium.app`，Chromium 文件夹需要创建。

基本使用

小试牛刀

如果是自己手动下载的 Chromuim，需要在 puppeteer.launch 的时候手动配置 executablePath 参数，对应可执行文件位置。

- 参数 headless 表示是否显示界面，false 显示界面。
- 参数 args 用来设置启动浏览器时候的一些参数，其中 `--window-size=1360,768` 表示设置参浏览器启动的时候，浏览器窗口的大小，还有很多类似的参数，详细内容请看传送门，我们这里只讲用到的，因为参数实在太多了。

GitChat

```
// 包含Puppeteer类库
const puppeteer = require("puppeteer");
const firstTest = async function() {
    // 下面异步相当于打开浏览器
    const browser = await puppeteer.launch({
        executablePath:
"./Chromium/Chromium.app/Contents/MacOS/Chromium",
        headless: false,
        args: [
            "--window-size=1360,768"
        ]
    });
    // 开一个新的页签
    const pageGitchat = await browser.newPage();
    // 设置浏览器视图大小为1360x768，就是网页所占的区域
    await pageGitchat.setViewport({width:1360, height:768});
    // 输入gitchat网址并回车
    await pageGitchat.goto("http://gitbook.cn");
    // 再开一个新的页签
    const pageWeibo = await browser.newPage();
    // 输入我的微博地址
    await pageWeibo.goto("https://weibo.com/u/5824742984");
```

```
    // 等待1s
    await pageWeibo.waitFor(1000);
    await browser.close();
};
firstTest();
```

效果如下所示，为了让图片减小，截取了一部分，但还是有好几MB。

GitChat

进阶使用

注入 JavaScript

注入 JavaScript，就是向目标页面注入我们自己的 JavaScript，方便我们操作、处理页面元素。比如我们这里想要获取到微博用户主页的用户名，习惯用 jQuery 的话就可以主动注入，提高自己的效率。

这里面又有了下面几个概念：

- waitFor，顾名思义，就是等待的意思，用它可以做很多操作，比如等待页面元素，页面中某个元素不确定什么时候加载出来，但是我们要在元素存在的时候处理，就可以使用这个方法。还可以用它来 sleep，在开启的页签中等待。
- evaluate，向页面中注入代码执行，并返回结果。

```
const puppeteer = require("puppeteer");
const firstTest = async function() {
    // 打开浏览器
    const browser = await puppeteer.launch({
        headless: false,
```

```
            args: [
                "--window-size=1360,768"
            ]
        });

        // 开一个新的页签
        const pageWeibo = await browser.newPage();
        await pageWeibo.setViewport({width:1360, height:768});
        // 输入我的微博地址
        await pageWeibo.goto("https://weibo.com/u/5824742984");

        // 因为页面要跳转，第一次解析完之后是一个中间页面没有我们要的东西，所以等待页解析，知道存在<h1 class="username">元素
        await pageWeibo.waitFor("h1.username");

        // 第一次执行
        let result = await pageWeibo.evaluate(() => {
            let ret;
            try {
                ret = $("h1.username").html();
            } catch (err) {
                ret = err.message;
            }
            return ret;
        });
        console.log("未注入jquery之前的结果："+result);

        // 注入jquery
        await pageWeibo.addScriptTag({
            url: "https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"
        });
        // 从页面中获取信息
        result = await pageWeibo.evaluate(() => {
            return $("h1.username").html();
        });

        console.log("注入jquery之后的结果："+result);

        await browser.close();
    };
    firstTest();
```

返回结果如下所示：

```
/tmp/spider-puppeteer# node inject.js
```

- 未注入 jQuery 之前的结果：$ is not defined。
- 注入 jQuery 之后的结果：燕睿涛Liam。

GitChat

获取 Google 翻译结果

目前为止，我们对 Puppeteer 的基本使用算是有个了解了，接下来我们来做个小工具，抓取 Google 翻译的结果（Tips：需要翻过高高的围墙）。

为了让工具体验更好我们需要另外一个类库 commander，用来处理命令行参数。

通过 `npm install --save commander` 安装参数处理类库，下面我们会用到。

获取谷歌翻译，我们主要用到了 Puppeteer 的下面几个知识点儿：

- 等待页面元素
- 模拟聚焦页面元素输入框
- 模拟键盘输入内容
- 模拟点击页面按钮
- 获取页面元素内容

代码如下：

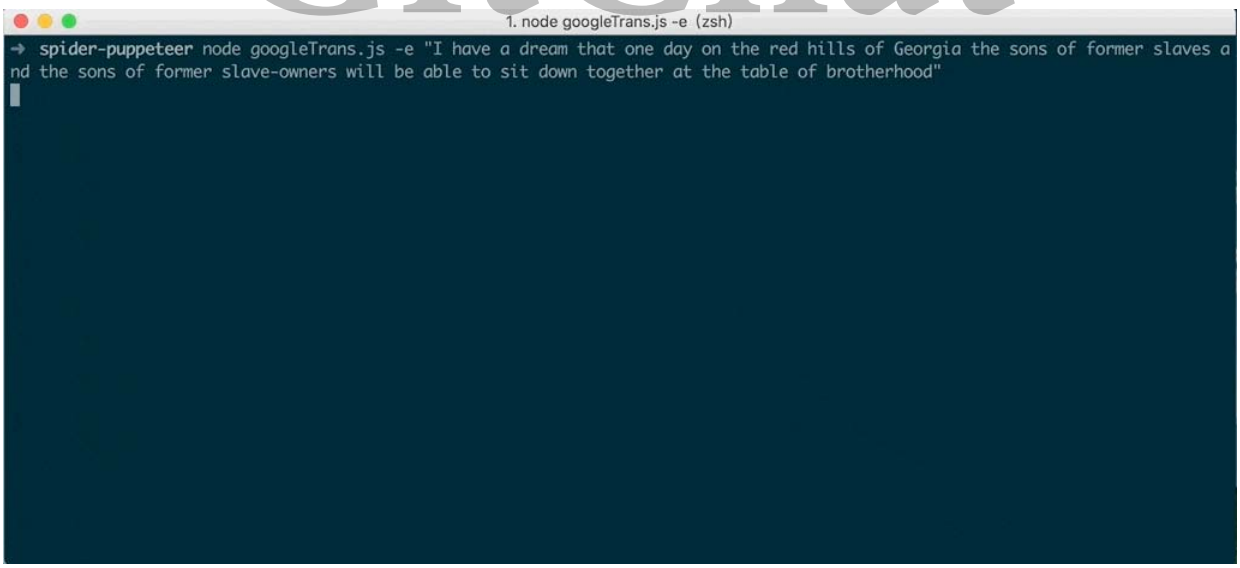
```
// 引入需要用到的类库
const puppeteer = require('puppeteer');
const program = require('commander');
// 处理参数，接受一个-e|--english的参数
program
    .option('-e, --english [value]', '待翻译的英文')
    .parse(process.argv);

// 获取传入的参数
let english = program.english;
(async () => {
    const browser = await puppeteer.launch({
        executablePath:
'./Chromium/Chromium.app/Contents/MacOS/Chromium',
    });
    const page = await browser.newPage();
    await page.goto("https://translate.google.cn/#auto/zh-CN", {
        // 由于要绕过高墙，可能比较耗时，这里超时设为0，也就是没有超时时间
        timeout: 0
    });
    // 等待页面加载出id="source"的元素
    await page.waitFor("#source");
    // 获取id="source"的元素
    let englishObj = await page.$("#source");
    //  光标聚焦到id="source"的元素
    englishObj.focus();
    // 聚焦之后输入我们要翻译的英文，输入的时候每次按键之间间隔1ms
    await page.keyboard.type(english, {delay: 1});
    // 等待“翻译”按钮展示出来
    await page.waitFor("#gt-submit");
```

GitChat

```
    // 获取“翻译”按钮对象
    let gtBtn = await page.$("#gt-submit");
    // 模拟鼠标点击“翻译”按钮对象
    await gtBtn.click();
    // 等待500ms
    await page.waitFor(500);
    // 等待翻译完毕，翻译前后下面这个元素会改变他的属性，我们等待他翻译完毕
    await page.waitFor("#gt-swap[aria-disabled=false]");
    // 等待翻译结果页面渲染完毕
    await page.waitFor("#result_box span");
    // 将函数传递给页面执行，获取页面中翻译的结果
    let chinese = await page.evaluate(()=>{
        elements = document.querySelectorAll("#result_box span");
        var chinese = "";
        for (var i=0; i<elements.length; i++) {
            chinese += elements[i].innerHTML;
        }
        return chinese;
    });
    // 打印结果
    console.log(chinese);
    // 关闭浏览器
    browser.close();
})();
```

GitChat

执行结果如下所示：



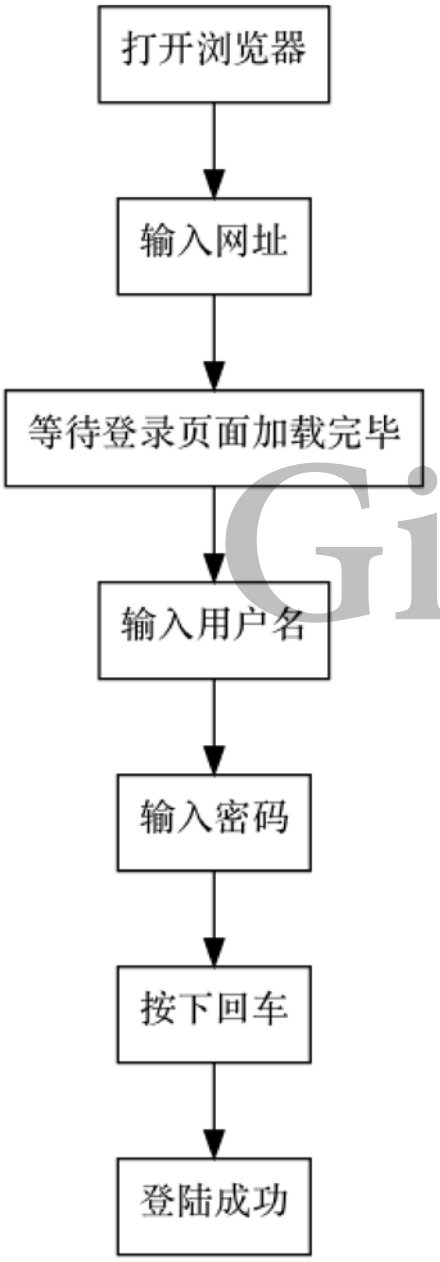

```
/tmp/spider-puppeteer#  node googleTrans.js -e "hello world"
你好，世界
/tmp/spider-puppeteer#  node googleTrans.js -e "hello world, I am
Liam Yan"
你好，我是利亚姆
/tmp/spider-puppeteer#  node googleTrans.js -e "I have a dream
that one day on the red hills of Georgia the sons of former
slaves and the sons of former slave-owners will be able to sit
down together at the table of brotherhood"
```

> 我有一个梦想，有一天，在格鲁吉亚红山丘上，前奴隶的儿子和前奴隶主的儿子们将能够一起坐在兄弟会的桌子上

登录微博

未登录状态只可以看部分微博，所以，获取微博内容之前，需要我们模拟登陆微博。

使用 Puppeteer 模拟登录微博的时候，跟我们抓取静态页面的模拟登录不太一样，我们这里完全模拟人机交互登录微博。



GitChat

登陆的时候 page.waitForNavigation(options) 很重要，一定要等到页面完全加载完毕，再输入用户名密码，不然会出现输入中失去焦点的情况，导致登录失败。

下面的代码只是部分代码，拿出来介绍登录流程。

```
// 开一个页签，模拟登陆微博
const pageWeibo = await browser.newPage();
await pageWeibo.setViewport({
    width:1280,
    height:768
});
console.log("开始登陆...");
await pageWeibo.goto("https://www.weibo.com", {
    timeout: 0
});
// 等待浏览器加载完毕
await pageWeibo.waitForNavigation({
    waitUntil: ["load"],
    timeout: 0
});

// 模拟输入用户名
console.log("输入用户名...");
await pageWeibo.waitForSelector("#loginname");
await pageWeibo.focus("#loginname");
await pageWeibo.keyboard.type(userName, {
    delay: 10
});

// 模拟输入密码
console.log("输入密码...");
// 等待浏览器中出现元素`input[name=password]`
await pageWeibo.waitForSelector("input[name=password]");
// 鼠标聚焦元素`input[name=password]`
await pageWeibo.focus("input[name=password]");
// 键盘输入密码，每次按键间隔10ms
await pageWeibo.keyboard.type(passWord, {
    delay: 10
});

// 模拟点击登录
console.log("登录中...");
await pageWeibo.click("a[action-type=btn_submit]", {
    delay: 500
});
await pageWeibo.waitForNavigation({
    waitUntil: ["load"],
    timeout: 0
});
console.log("登录成功\n\n");
```

GitChat

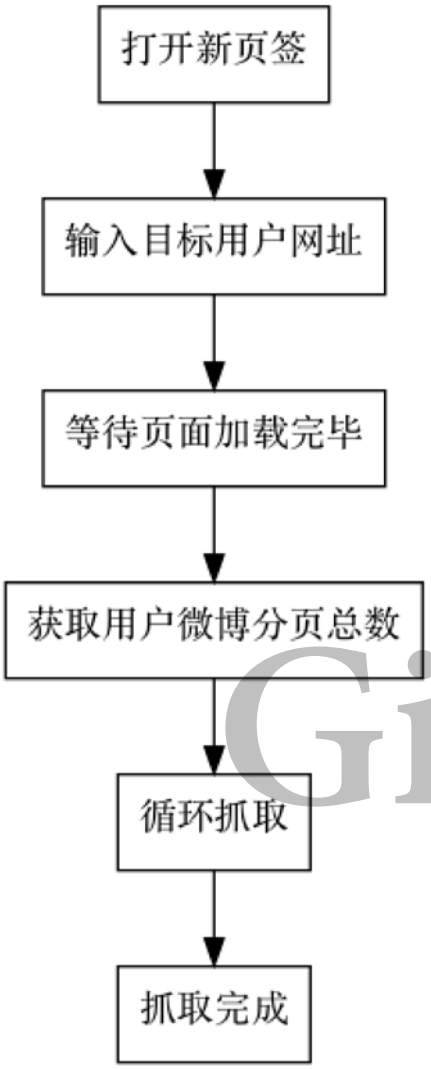
爬取微博用户微博

爬取微博用户微博，用到了 querystringify 组件，用来解析 query 参数。

安装过程如下：

```
npm install --save querystringify
```

爬取用户微博截图整体流程：



GitChat

由于微博是 Lazy Load 模式，不会一次将分页的所有内容展示出来，需要通过用户滚动鼠标滑轮触发加载事件，我们这里通过 Puppeteer 发送 JavaScript 命令给页面，让页面滚动加载，详情见下面的 scrollToPageBar。

抓取微博内容是通过 elementHandle.screenshot([options]) 函数来截取元素快照的。但到目前为止，还有些瑕疵，比如抓取微博内容的时候要对微博列表做各种处理，先隐藏所有，再显示要截屏的，不然会出现文字被背景覆盖的情况，不知道是不是 Puppeteer 的 Bug。另外还有它的 Headless 模式，也就是不显示浏览器的模式，抓取内容抓取几页之后，可能会随机的出现“假死”的情况，就卡在截屏操作那里，也不报错，这个问题目前还没找到原因。因此我们这里一般都将“headless”设置为 false，显示浏览器。请看下面的代码。

```
const puppeteer = require("puppeteer");
const qs = require('querystringify');
```

```
const userName = "你的微博名";
const passWord = "你的微博密码";
// 等待[ms]ms函数
function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms))
}

const firstTest = async function() {
    // 下面异步相当于打开浏览器
    const browser = await puppeteer.launch({
        executablePath:
'./Chromium/Chromium.app/Contents/MacOS/Chromium',
        headless: false,
        args: [
            "--window-size=1360,768"
        ]
    });
    // 模拟登陆
    // 开一个新的页签，准备抓取微博数据
    const pageWeiboUser = await browser.newPage();
    await pageWeiboUser.setViewport({
        width:1360,
        height:768
    });

    /**
     * 滚动加在到分页bar出来
     * @return {[type]} [description]
     */
    let scrollToPageBar = async function() {
        let pageBar = await pageWeiboUser.$("div[node-
type=feed_list_page]");
        while (!pageBar) {
            // 传递命令给浏览器，让浏览器执行滚动
            await pageWeiboUser.evaluate((scrollStep)=>{
                let scrollTop =
document.scrollingElement.scrollTop;
                document.scrollingElement.scrollTop = scrollTop +
scrollStep;
            }, 1000);
            await sleep(100);
            pageBar = await pageWeiboUser.$("div[node-
type=feed_list_page]")
        }
    };

    /**
     * 点击下一页面按钮
     * @return {[type]} [description]
     */
    let gotoNextPage = async function(pageNum) {
        await pageWeiboUser.goto("https://weibo.com/lixiaolu?
```

GitChat

```
is_search=0&visible=0&is_ori=1&is_tag=0&profile_ftype=1&page="+pa
geNum+"#feedtop");
        await pageWeiboUser.addScriptTag({
            url:
"https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"
        });
    };

    /**
     * 获取带抓取微博的总页数
     * @return {[type]} [description]
     */
    let getTotalPage = async function() {
        await scrollToPageBar();
        // 发送命令获取总页数
        let pageInfo = await pageWeiboUser.evaluate(() => {
            let pageMore = $("a[action-
type=feed_list_page_more]");
            let pageInfo = pageMore.attr("action-data");
            return pageInfo;
        });
        let pageInfoObj = qs.parse(pageInfo);
        return pageInfoObj.countPage;
    };

    /**
     * 抓取当前页面的微博
     * @return {[type]} [description]
     */
    let getWeiboScreenshots = async function(pageNum, countPage)
{
        await scrollToPageBar();
        await pageWeiboUser.evaluate(() => {
            document.scrollingElement.scrollTop = 300;
        });
        // 获取微博个数
        let count = await pageWeiboUser.evaluate(() => {
            return $("div[action-type=feed_list_item]").length;
        })
        // 循环截图当前页面每一条微博
        for (let i=0; i< count; i++) {
            // puppeteer可能有bug，试了好多次，折中方案，每次把要截屏的
微博显示到最前面，不然截取有bug
            await pageWeiboUser.evaluate((index) => {
                $("div[action-type=feed_list_item]").css({
                    "display": "none"
                });
                $("div[action-
type=feed_list_item]:eq("+index+")").css({
                    "display": "block",
                }).attr("id", "spider_"+index);
            }, i);
```

GitChat

```
            let weibo = await pageWeiboUser.$("#spider_"+i);
            await weibo.screenshot({
                path: "./screenshots/"+pageNum+"_"+(i+1)+".png"
            });
            process.stdout.write(".");
            await sleep(50);
        }
        process.stdout.write("\n");
    };

    let pageNum = 1;
    await pageWeiboUser.goto("https://weibo.com/lixiaolu?
profile_ftype=1&is_ori=1");
    await pageWeiboUser.addScriptTag({
        url: "https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"
    });
    let countPage = await getTotalPage();

    while (countPage >= pageNum) {
        console.log("开始抓取第["+pageNum+"]页数据...");
        await getWeiboScreenshots(pageNum, countPage);
        console.log("第["+pageNum+"]页数据抓取结束");
        pageNum++;
        await gotoNextPage(pageNum);
    }
    console.log("\n\n抓取结束");

    await browser.close();
};
firstTest();
```

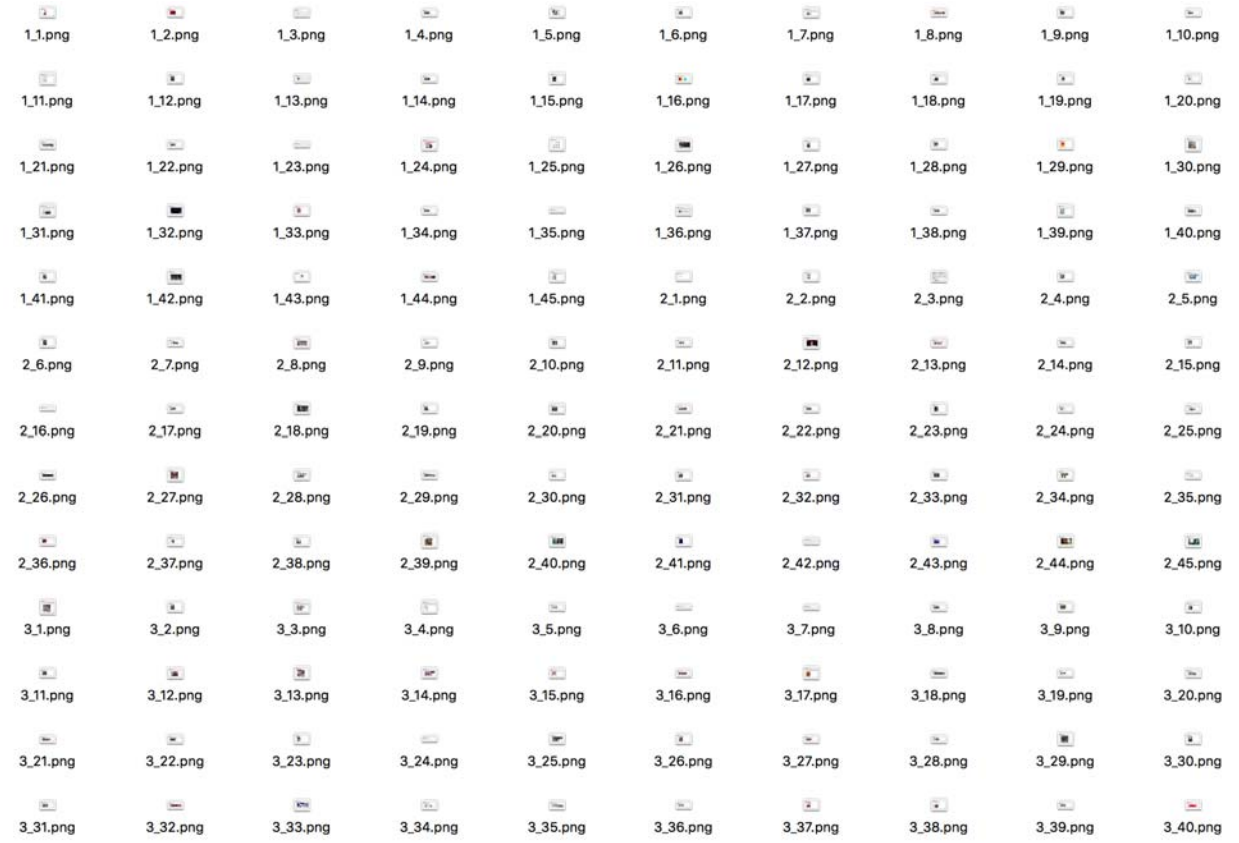
GitChat

下面是命令行的输出情况，大家可以试试效果，我这边抓取了李小璐的所有原创微博，一共970多条，图片命名方式是 [{页码}_{页中第几条}.png]。

命令行会记录脚本的状态，抓取到第几页，每抓取一个就显示一个”.”。

如果你想要抓取自己感兴趣的用户，需要将 weiboSpider.js 中的 91 、 145 行代码替换为自己感兴趣的微博用户对应链接。

```
/tmp/spider-puppeteer# node spiderWeibo.js
开始登陆...
输入用户名...
输入密码...
登录中...
登录成功
开始抓取第[1]页数据...
.........................................
第[1]页数据抓取结束
```



总结

GitChat

听我说了这么多，你可能早已经跃跃欲试了，这里把本文相关的源代码放出来，感兴趣的可以自己试下。另外，这里面需要使用用户名密码的地方需要替换为你自己的。如果微博出现让输入验证码的情况，可以等一等再试，验证码这里目前还没有完全自动化。

对 Puppeteer，我个人的愚见是：网页中只要你能看到的地方，它几乎都可以帮你处理，对于爬取某些加密的地方太有用了。它也不用我们去模拟登陆收集各种 Cookie 信息了。还有让输入验证码的地方也会有相对更友好的方式处理了，做成半自动化的形式，这个后面有机会了再尝试。

源代码地址：https://pan.baidu.com/s/1fBfhbXtLRxZbfUUYEWzJog，密码：759k。

下载下来解压之后，执行 npm install 就可以了，如果安装失败，看看是不是下载 Chromuim 失败，如果失败，可以执行 npm install –ignore-scripts，然后按照上面的步骤手动下载就可以了。

谢谢大家的支持。