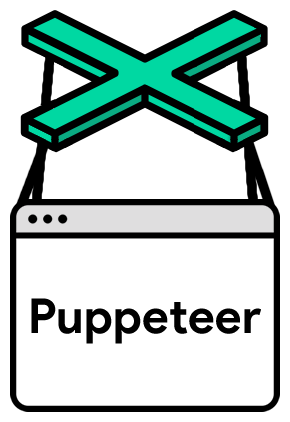
**Puppeteer之爬虫入门**

**译者按：** 本文通过简单的例子介绍如何使用Puppeteer来爬取网页数据，特别是用谷歌开发者工具获取元素选择器值得学习。

* 原文: A Guide to Automating & Scraping the Web with JavaScript (Chrome + Puppeteer + Node JS)
* 译者: [Fundebug](https://fundebug.com/)

**为了保证可读性，本文采用意译而非直译。另外，本文版权归原作者所有，翻译仅用于学习。**

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/puppeteer.png)

**我们将会学到什么？**

在这篇文章，你讲会学到如何使用JavaScript自动化抓取网页里面感兴趣的内容。我们将会使用Puppeteer，Puppeteer是一个Node库，提供接口来控制headless Chrome。Headless Chrome是一种不使用Chrome来运行Chrome浏览器的方式。

如果你不知道Puppeteer，也不了解headless Chrome，那么你只要知道我们将要编写JavaScript代码来自动化控制Chrome就行。

**准备工作**

你需要安装版本8以上的Node，你可以在这里找到安装方法。确保选择**Current**版本，因为它是8+。

当你将Node安装好以后，创建一个新的文件夹，将Puppeteer安装在该文件夹下。

*npm install –save puppeteer*

**例1：截屏**

当你把Puppeteer安装好了以后，我们来尝试第一个简单的例子。这个例子来自于Puppeteer文档（稍微改动）。我们编写的代码将会把你要访问的网页截屏并保存为png文件。

首先，创建一个test.js文件，并编写如下代码。

|  |
| --- |
| const puppeteer = require('puppeteer');  async function getPic() {  const browser = await puppeteer.launch();  const page = await browser.newPage();  await page.goto('https://google.com');  await page.screenshot({path: 'google.png'});  await browser.close();  }  getPic(); |

我们来一行一行理解一下代码的含义。

* 第1行：引入我们需要的库Puppeteer；
* 第3-10行：主函数getPic()包含了所有的自动化代码；
* 第12行：调用getPic()函数。

这里需要提醒注意getPic()函数是一个async函数，使用了ES 2017 async/await特性。该函数是一个异步函数，会返回一个Promise。如果async最终顺利返回值，Promise则可以顺利reslove，得到结果；否则将会reject一个错误。

因为我们使用了async函数，我们使用await来暂停函数的执行，直到Promise返回。

接下来我们深入理解一下getPic()：

* 第4行：

|  |
| --- |
| const broswer = await puppeteer.launch(); |

* 这行代码启动puppeteer，我们实际上启动了一个Chrome实例，并且和我们声明的browser变量绑定起来。因为我们使用了await关键字，该函数会暂停直到Promise完全被解析。也就是说成功创建Chrome实例或则报错。
* 第5行：

|  |
| --- |
| const page = await browser.newPage(); |

* 我们在浏览器中创建一个新的页面，通过使用await关键字来等待页面成功创建。
* 第6行：

|  |
| --- |
| await page.goto('https://google.com'); |

* 使用page.goto()打开谷歌首页。
* 第7行：

|  |
| --- |
| await page.screenshot({path: 'google.png'}); |

* 调用screenshot()函数将当前页面截屏。
* 第9行：

|  |
| --- |
| await browser.close(); |

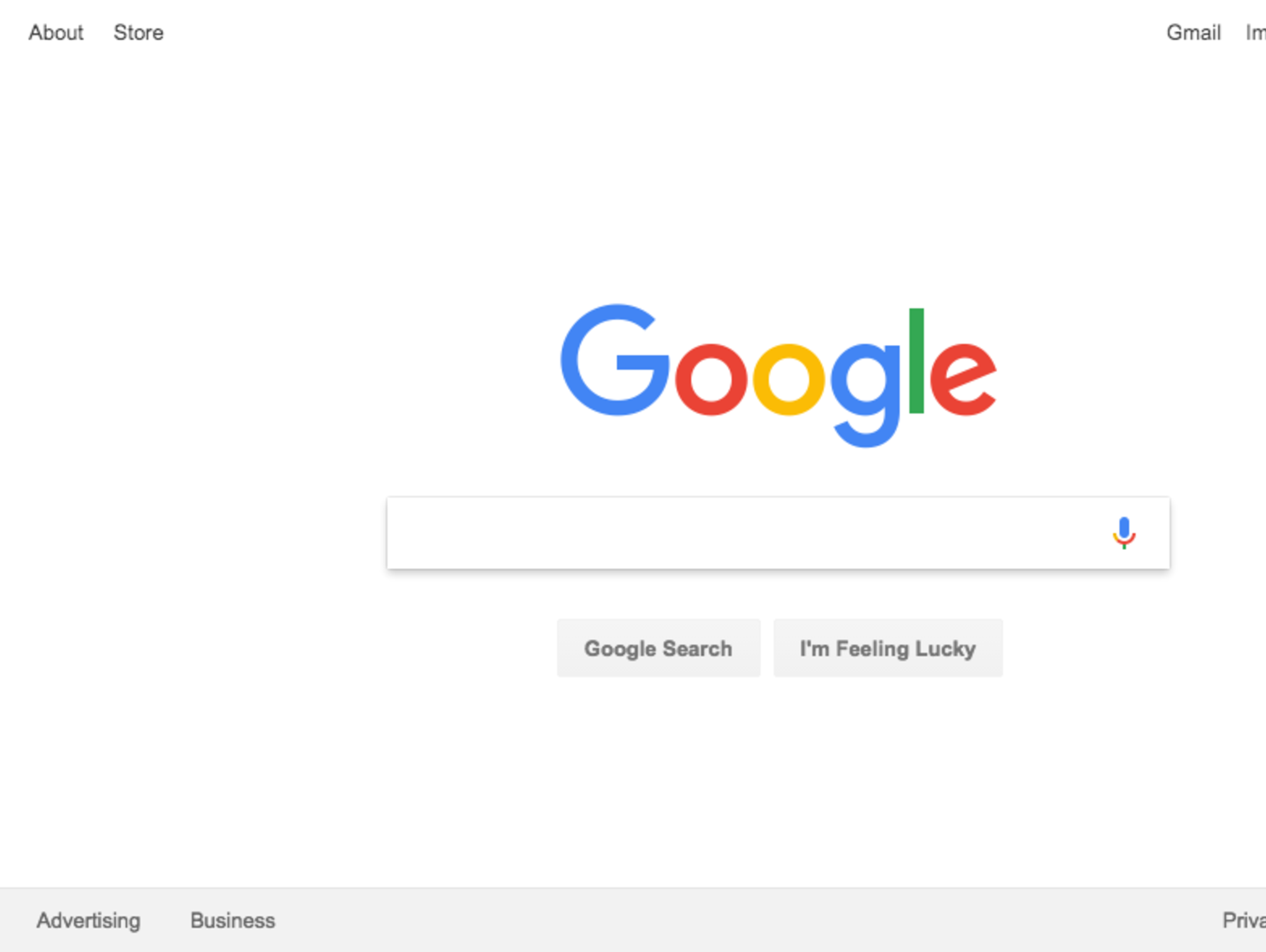
* 将浏览器关闭。

**执行实例**

使用Node执行：

|  |
| --- |
| node test.js |

下面截取的图片google.png：

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/google.png)

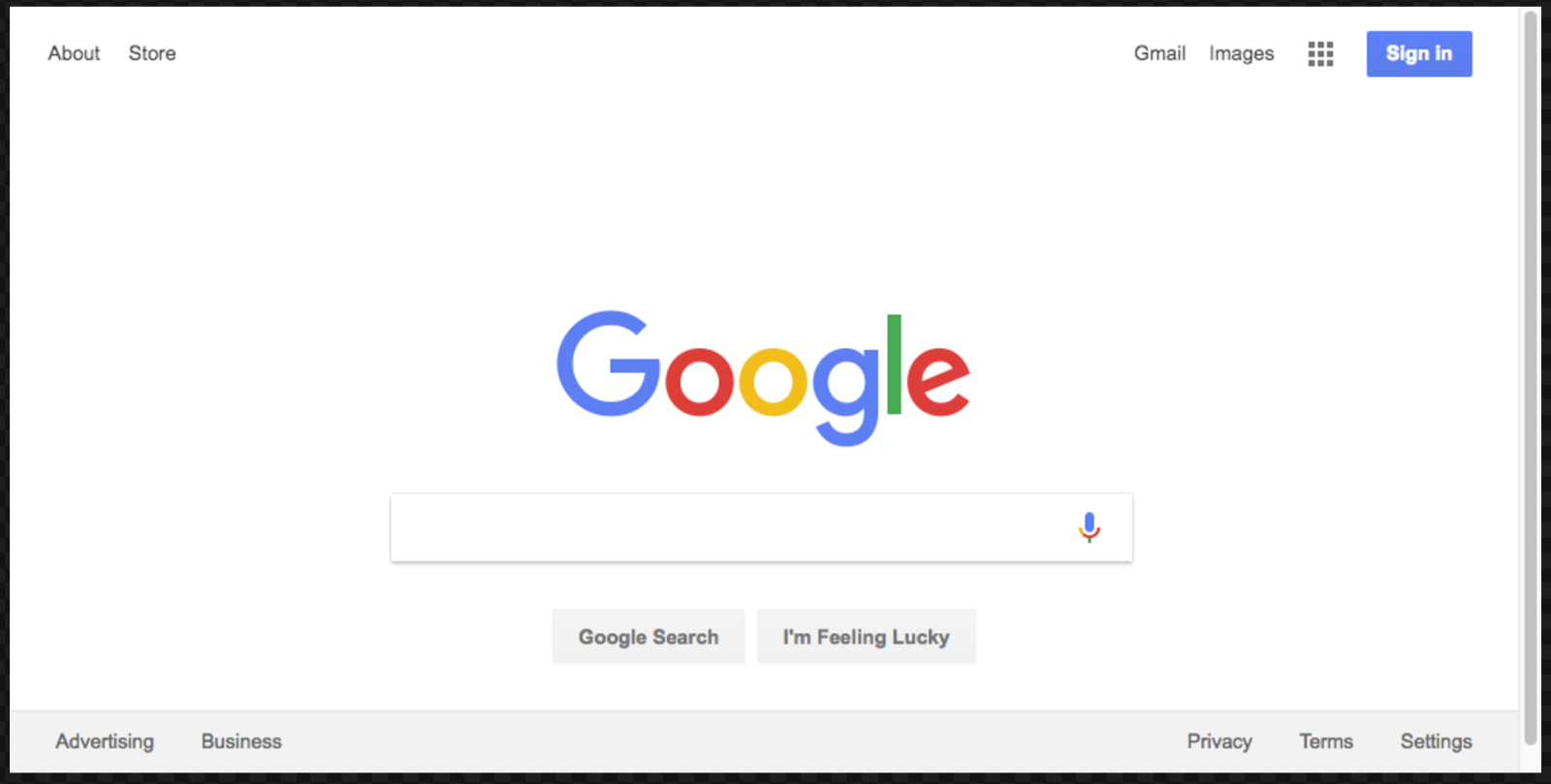
现在我们来使用non-headless模式试试。将第4行代码改为：

|  |
| --- |
| const browser = await puppeteer.launch({headless: false}); |

然后运行试试。你会发现谷歌浏览器打开了，并且导航到了谷歌搜索页面。但是截屏没有居中，我们可以调节一下页面的大小配置。

|  |
| --- |
| await page.setViewport({width: 1000, height: 500}); |

截屏的效果会更加漂亮。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/google1.png)

下面是最终版本的代码：

|  |
| --- |
| const puppeteer = require('puppeteer');  async function getPic() {  const browser = await puppeteer.launch({headless: false});  const page = await browser.newPage();  await page.goto('https://google.com');  await page.setViewport({width: 1000, height: 500})  await page.screenshot({path: 'google.png'});  await browser.close();  }  getPic(); |

**例2：爬取数据**

首先，了解一下Puppeteer的[API](https://github.com/GoogleChrome/puppeteer/blob/master/docs/api.md" \t "_blank)。文档提供了非常丰富的方法不仅支持在网页上点击，而且可以填写表单，读取数据。

接下来我们会爬取[Books to Scrape](http://books.toscrape.com/)，这是一个伪造的网上书店专门用来练习爬取数据。

在当前目录下，我们创建一个scrape.js文件，输入如下代码：

|  |
| --- |
| const puppeteer = require('puppeteer');  let scrape = async () => {  // 爬取数据的代码    // 返回数据  };  scrape().then((value) => {  console.log(value); // 成功！  }); |

**第一步：基本配置**

我们首先创建一个浏览器实例，打开一个新页面，并且导航到要爬取数据的页面。

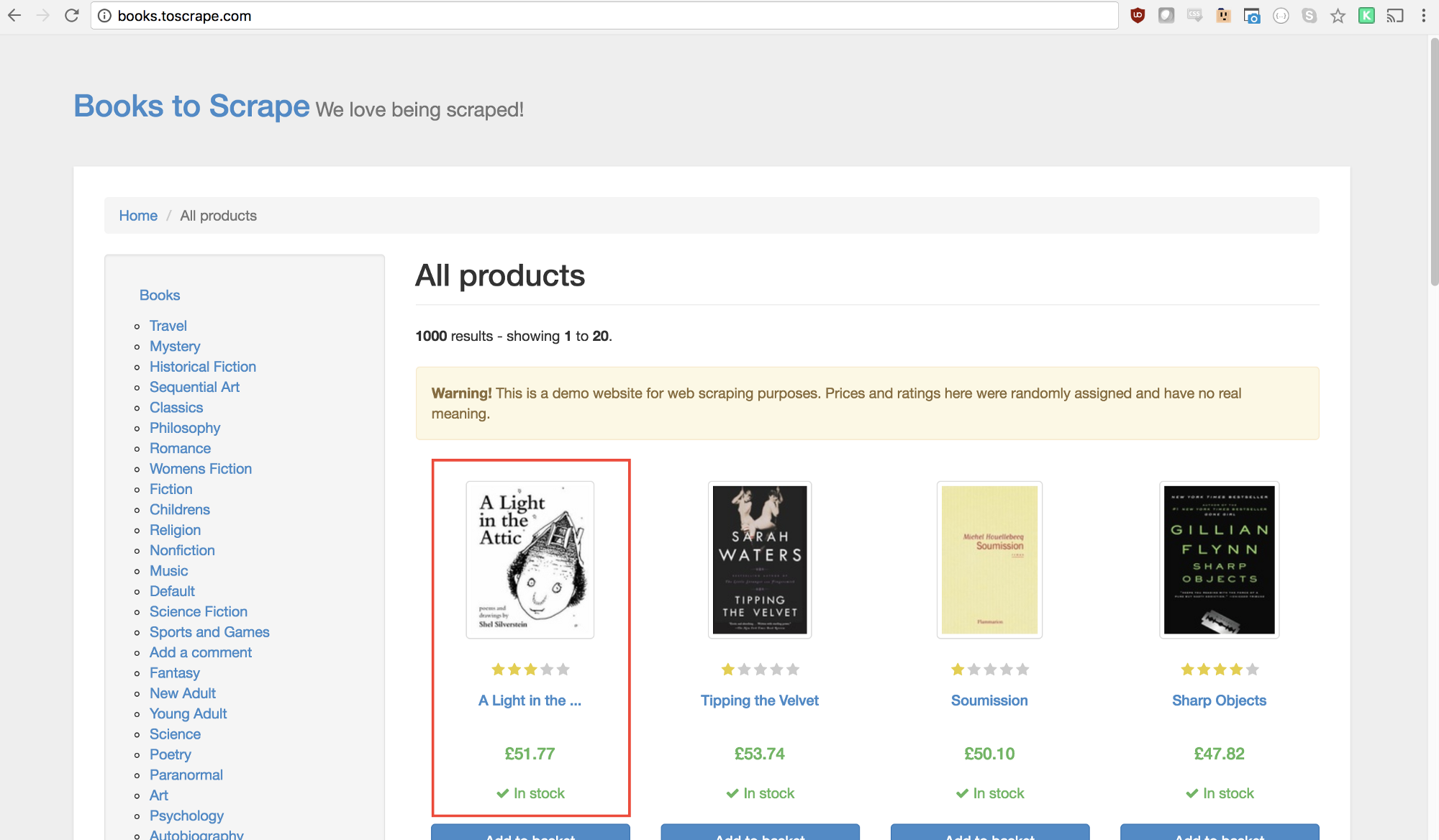
|  |
| --- |
| let scrape = async () => {  const browser = await puppeteer.launch({headless: false});  const page = await browser.newPage();  await page.goto('http://books.toscrape.com/');  await page.waitFor(1000);  // Scrape  browser.close();  return result;  }; |

注意其中有一行代码让浏览器延时关闭。这行代码本来是不需要的，主要是方便查看页面是否完全加载。

|  |
| --- |
| await page.waitFor(1000); |

**第二步：抓取数据**

我们接下来要选择页面上的第一本书，然后获取它的标题和价格。

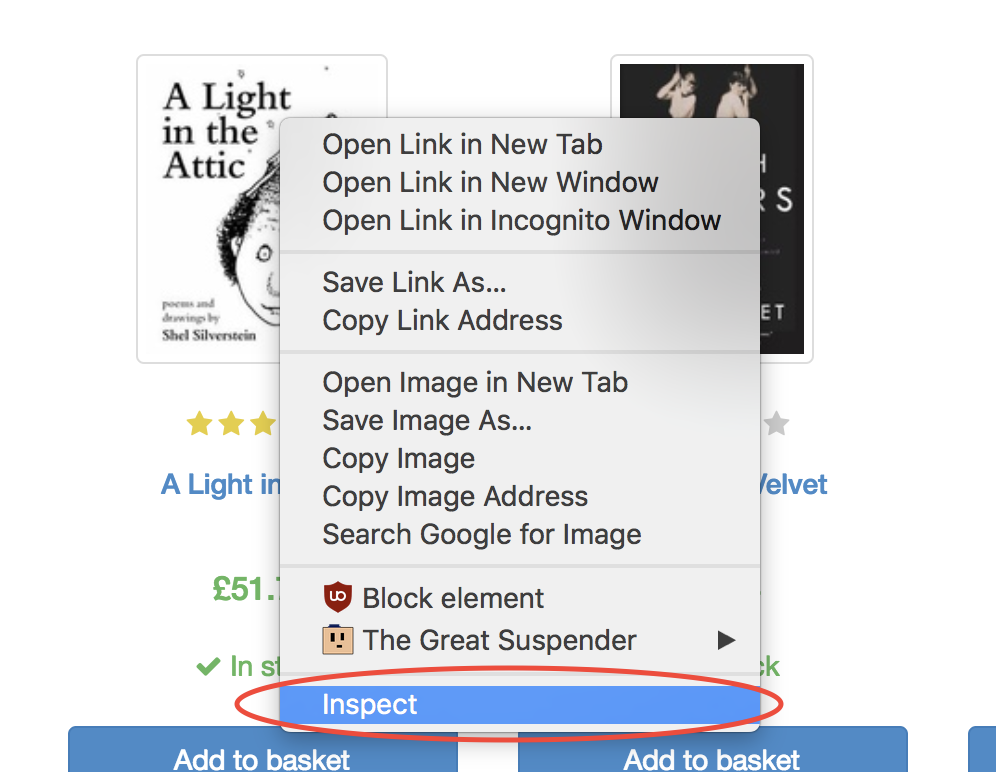
[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/first_book.png)

查看Puppeteer API，可以找到定义点击的函数：

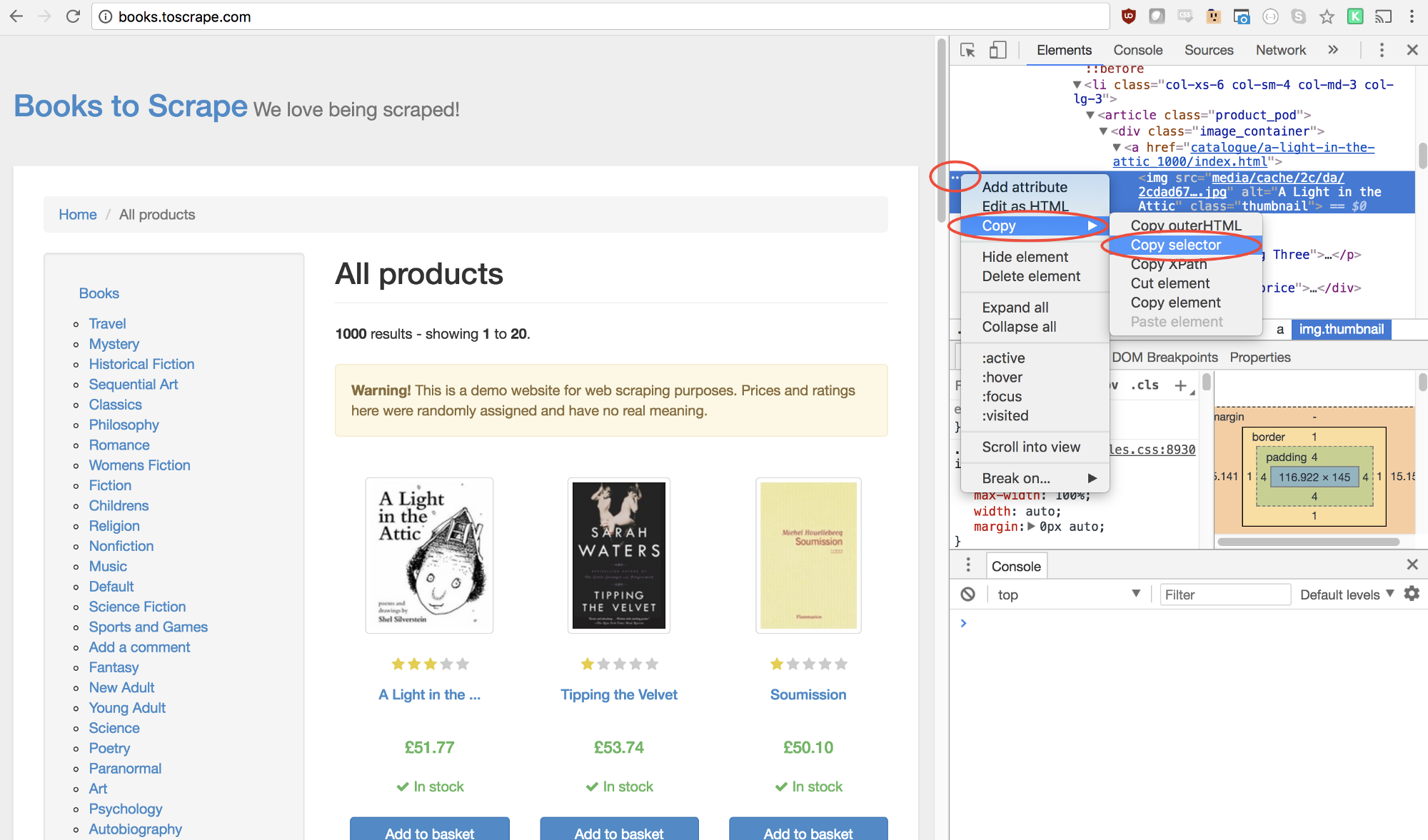
**page.click(selector[, options])**

* selector 一个选择器来指定要点击的元素。如果多个元素满足，那么默认选择第一个。

幸运的是，谷歌开发者工具提供一个可以快速找到选择器元素的方法。在图片上方右击，选择*检查(Inspect)*选项。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/first_book_inspect.png)

谷歌开发者工具的Elements界面会打开，并且选定部分对应的代码会高亮。右击左侧的三个点，选择拷贝(Copy)，然后选择拷贝选择器(Copy selector)。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/first_book_inspect_copy.png)

接下来将拷贝的选择器插入到函数中。

|  |
| --- |
| await page.click('#default > div > div > div > div > section > div:nth-child(2) > ol > li:nth-child(1) > article > div.image\_container > a > img'); |

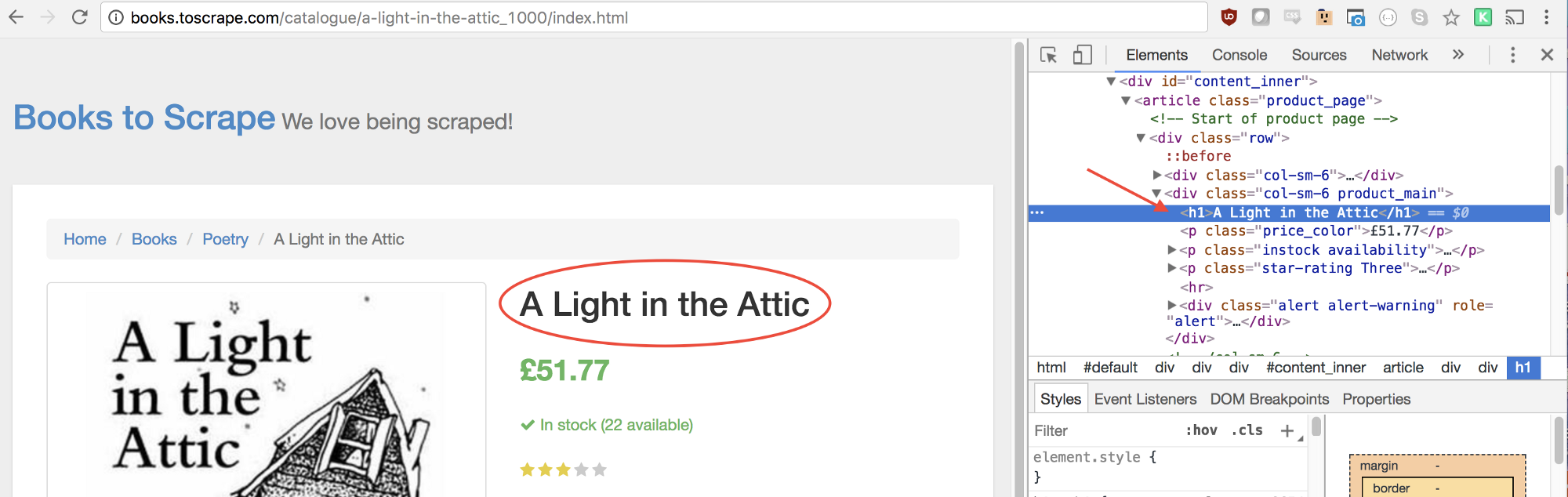
加入了点击事件的代码执行后会直接跳转到详细介绍这本书的页面。而我们则关心它的标题和价格部分。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/first_book_detail.png)

为了获取它们，我们首选需要使用page.evaluate()函数。该函数可以让我们使用内置的DOM选择器，比如querySelector()。

|  |
| --- |
| const result = await page.evaluate(() => {  // return something  }); |

然后，我们使用类似的手段获取标题的选择器。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/first_book_detail_title.png)

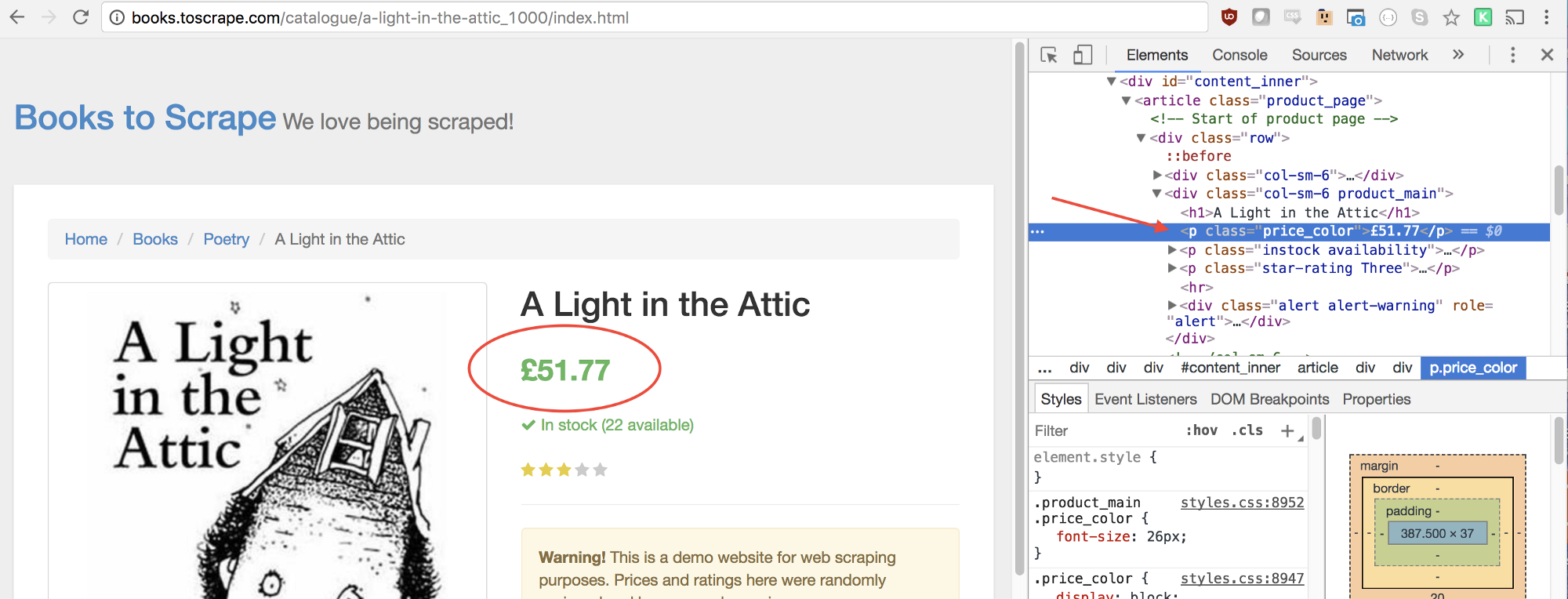
使用如下代码可以获取该元素：

|  |
| --- |
| let title = document.querySelector('h1'); |

但是，我们真正想要的是里面的文本文字。因此，通过.innerText来获取。

|  |
| --- |
| let title = document.querySelector('h1').innerText; |

价格也可以用相同的方法获取。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/first_book_detail_price.png)

|  |
| --- |
| let price = document.querySelector('.price\_color').innerText; |

最终，将它们一起返回，完整代码如下：

|  |
| --- |
| const result = await page.evaluate(() => {  let title = document.querySelector('h1').innerText;  let price = document.querySelector('.price\_color').innerText;  return {  title,  price  }  }); |

所有的代码整合到一起，如下：

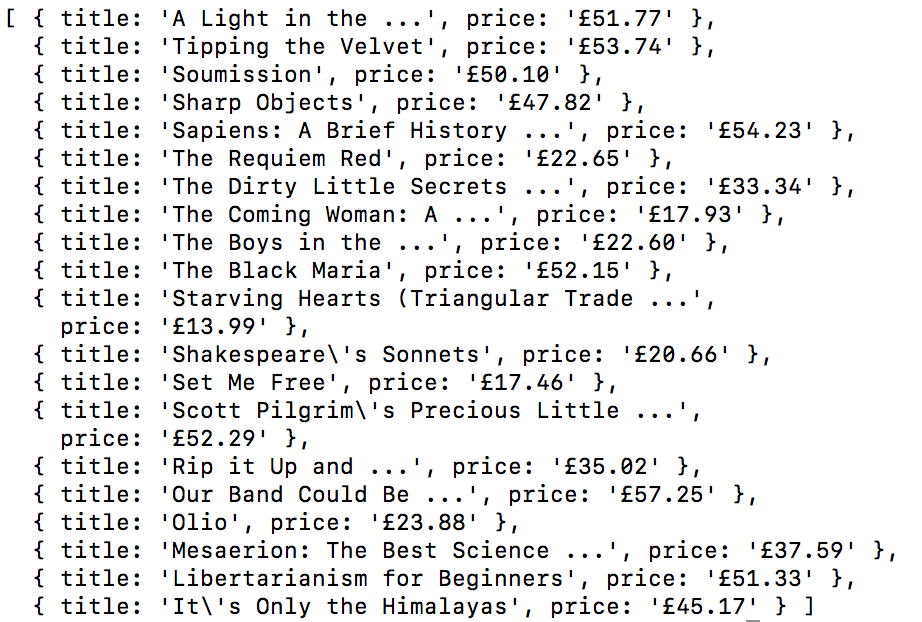
|  |
| --- |
| const puppeteer = require('puppeteer');  let scrape = async () => {  const browser = await puppeteer.launch({headless: false});  const page = await browser.newPage();  await page.goto('http://books.toscrape.com/');  await page.click('#default > div > div > div > div > section > div:nth-child(2) > ol > li:nth-child(1) > article > div.image\_container > a > img');  await page.waitFor(1000);  const result = await page.evaluate(() => {  let title = document.querySelector('h1').innerText;  let price = document.querySelector('.price\_color').innerText;  return {  title,  price  }  });  browser.close();  return result;  };  scrape().then((value) => {  console.log(value); // Success!  }); |

运行node scrape.js即可返回数据

|  |
| --- |
| { title: 'A Light in the Attic', price: '£51.77' } |

**例3：进一步优化**

从主页获取所有书籍的标题和价格，然后将它们返回。

[](https://blog.fundebug.com/2017/11/01/guide-to-automating-scraping-the-web-with-js/booklist.png)

**提示**

和例2的区别在于我们需要用一个循环来获取所有书籍的信息。

|  |
| --- |
| const result = await page.evaluate(() => {  let data = []; // Create an empty array  let elements = document.querySelectorAll('xxx'); // 获取所有书籍元素  // 循环处理每一个元素  // 获取标题  // 获取价格  data.push({title, price}); // 将结果存入数组  return data; // 返回数据  }); |

**解法**

|  |
| --- |
| const puppeteer = require('puppeteer');  let scrape = async () => {  const browser = await puppeteer.launch({headless: false});  const page = await browser.newPage();  await page.goto('http://books.toscrape.com/');  const result = await page.evaluate(() => {  let data = []; // 初始化空数组来存储数据  let elements = document.querySelectorAll('.product\_pod'); // 获取所有书籍元素  for (var element of elements){ // 循环  let title = element.childNodes[5].innerText; // 获取标题  let price = element.childNodes[7].children[0].innerText; // 获取价格  data.push({title, price}); // 存入数组  }  return data; // 返回数据  });  browser.close();  return result;  };  scrape().then((value) => {  console.log(value); // Success!  }); |

[更多干货](http://www.hclcm.cn)：http://www.hclcm.cn