

Пробная версия



Как Тос

Парсинг веб-страниц с помощью Puppeteer

Из этого пошагового руководства вы узнаете, как парсить статические и динамические сайты с помощью Puppeteer.

6 минут чтения



Гинтс Дрейманис

Puppeteer — это библиотека для тестирования и автоматизации браузера, которая также хороша для парсинга веб-страниц. По сравнению с более простыми инструментами, такими как [Axios](#) и [Cheerio](#), Puppeteer позволяет разработчикам очищать динамический контент (т. е. контент, который меняется в зависимости от действий пользователя).

меняется в зависимости от действия пользователя).

То есть вы можете использовать ее для парсинга веб-приложений (одностраничных приложений), которые загружают контент с помощью JavaScript. Это вы и будете делать здесь.

Парсинг веб-страниц с использованием Puppeteer

В этом уроке вы узнаете, как парсить статические и динамические данные (например, заголовки публикаций и ссылки из [блога Bright Data](#)) с помощью Puppeteer.

Пробная версия

Затем создайте новый каталог для проекта и перейдите к нему с помощью следующих команд:

```
mkdir puppeteer_tutorial
cd puppeteer_tutorial
npm init -y
```

Копировать

Далее установите Puppeteer с помощью этой команды:

```
npm i puppeteer --save
```

Копировать

Эта команда также загружает специальный браузер, который будет использоваться библиотекой.

Парсинг статического сайта

Как и все инструменты для парсинга, Puppeteer позволяет соскабливать HTML-код веб-страниц.

Ниже мы описали шаги, которые вы можете выполнить, чтобы использовать Puppeteer для парсинга первой страницы статей блога Bright Data:

Создайте `index.js` файл и импортируйте Puppeteer:

```
const puppeteer = require('puppeteer');
```

Копировать

Затем вставьте шаблон, который нужен для запуска Puppeteer:

```
(async () => {
  const browser = await puppeteer.launch({
    headless: false,
    defaultViewport: null
  });

  const page = await browser.newPage();
  await page.goto('https://brightdata.com/blog');
```

Копировать

```
// all the web scraping will happen here

await browser.close();

})();
```

Эта функция открывает браузер, переходит на страницу для ее парсинга и закрывает браузер.

Все, что осталось сделать, это собрать данные со страницы.

Пробная версия

В блоге **Bright Data** все данные сообщений блога заключены в `<a>` тег с классом `brd_post_entry`. Заголовок сообщения находится в `<h3>` элементе класса `brd_post_title`. Ссылка на пост имеет `href` значение `brd_post_entry`.

Вот как `page.evaluate` выглядит функция, извлекающая эти значения:

```
const data = await page.evaluate( () => {

  let data = [];
  const titles = document.querySelectorAll('.brd_post_entry');

  for (const title of titles) {
    const titleText = title.querySelector('.brd_post_title').textContent;
    const titleLink = title.href;

    const article = { title: titleText, link: titleLink };
    data.push(article);
  }

  return data;

})
```

Копировать

Теперь вы можете вывести данные в консоль:

```
console.log(data);
```

Копировать

Полный скрипт выглядит так:

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch({
    headless: false,
    defaultViewport: null
  });
```

Копировать

```
const page = await browser.newPage();
await page.goto('https://brightdata.com/blog');

const data = await page.evaluate(() => {

  let data = [];
  const titles = document.querySelectorAll('.brd_post_entry');

  for (const title of titles) {
    const titleText = title.querySelector('.brd_post_title').textContent;

    data.push({
      title: titleText,
      link: title.querySelector('a').href
    });
  }

  return data;
})

console.log(data);

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

Пробная версия

Запустите его, вызвав `node index.js` терминал. Скрипт должен вернуть список заголовков сообщений и ссылок:

```
[
  {
    title: 'APIs for Dummies: Learning About APIs',
    link: 'https://brightdata.com/blog/web-data/apis-for-dummies'
  },
  {
    title: 'Guide to Using cURL with Python',
    link: 'https://brightdata.com/blog/how-tos/curl-with-python'
  },
  {
    title: 'Guide to Scraping Walmart',
    link: 'https://brightdata.com/blog/how-tos/guide-to-scraping-walmart'
  },
  ...
]
```

Копировать

Парсинг динамического контента

Парсинг статического контента — простая задача, которую легко решить с помощью несложных инструментов. К счастью, Puppeteer можно использовать и для выполнения более широкого спектра действий, например, клик, ввод текста и прокрутка. Вы можете использовать все это для взаимодействия с динамическими страницами и имитирования действий пользователя.

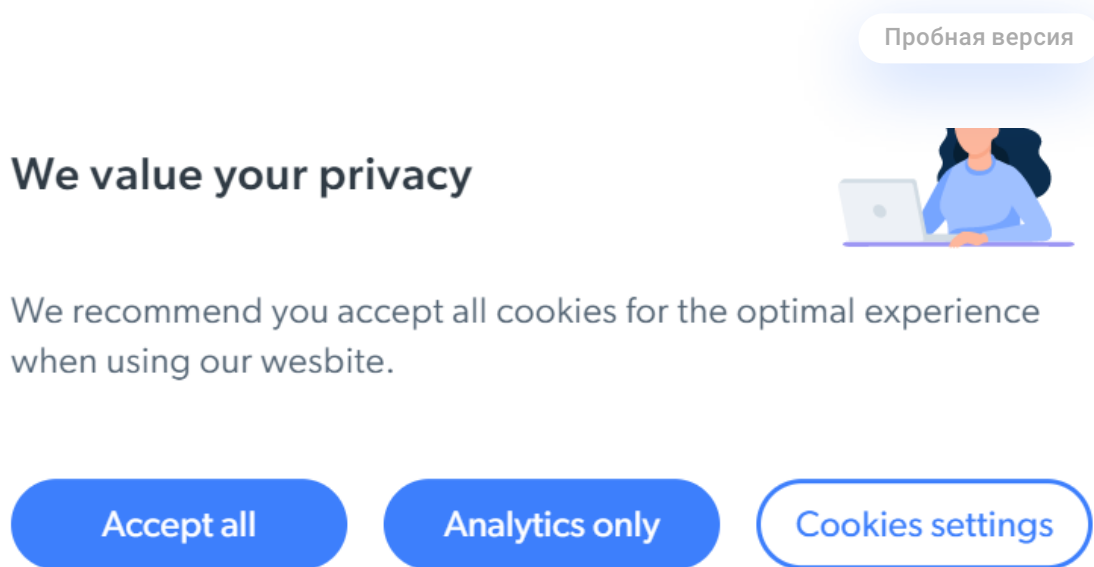
Обычная задача парсинга веб-страниц с помощью такой библиотеки — поиск определенного набора

данных на странице. Например, вы можете использовать Puppeteer для поиска всех статей Bright Data про Puppeteer.

Вот как это сделать:

Шаг 1. Примите файлы cookie

Когда человек заходит в блог Bright Data, иногда появляется баннер cookie:



Чтобы справиться с этим, вам нужно нажать кнопку **«Принять все»** со следующим кодом:

```
await page.waitForSelector('#brd_cookies_bar_accept', {timeout: 5000})
  .then(element => element.click())
  .catch(error => console.log(error));
```

Копировать

Первая строка кода ожидает `#brd_cookies_bar_accept` появления элемента с в течение 5 секунд. Вторая строка нажимает на этот элемент. Третья строка гарантирует, что скрипт не выйдет из строя, если панель cookie не появится.

Обратите внимание, что ожидание в Puppeteer происходит путем предоставления некоторого условия, которого вы хотите дождаться, а не путем установки определенного времени ожидания для выполнения предыдущего действия. Первое называется неявным ожиданием, а второе — явным.

Явное ожидание в Puppeteer не рекомендуется, поскольку приводит к проблемам с выполнением. Если вы укажете явное время ожидания, оно будет либо слишком длинным (что неэффективно), либо слишком коротким (скрипт будет выполняться некорректно).

Шаг 2. Поиск сообщений

После этого в скрипте нужно нажать на иконку поиска. Ввести Puppeteer и снова нажать значок поиска, чтобы запустить его:

Это можно сделать с помощью следующего кода:

Пробная версия

```
await page.waitForSelector( '.search_container.active' );
const search_form = await page.waitForSelector('#blog_search');
await search_form.type('puppeteer');

await page.click('.search_icon');

await new Promise(r => setTimeout(r, 2000));
```

Этот пример работает аналогично примеру баннера cookie. После нажатия кнопки вам нужно дождаться появления контейнера поиска, поэтому код ожидает элемент, соответствующий CSS-селектору `.search_container.active`.

Кроме того, в конце вам нужно добавить 2-секундную остановку для загрузки элементов. Хотя явное ожидание не рекомендуется в Puppeteer, сейчас здесь нет других хороших вариантов.

На большинстве веб-сайтов, если произойдет изменение URL-адреса, вы можете использовать этот `waitForNavigation` метод. Если появится новый элемент, вы можете использовать `waitForSelector` метод. Выяснить, обновляются ли некоторые элементы, немного сложнее и выходит за рамки этой статьи.

Если вы хотите попробовать это самостоятельно, [этот ответ на вопрос «Переполнение стека»](#) может вам помочь.

Шаг 3: Соберите сообщения

После поиска постов вы можете взять код, который вы уже использовали для парсинга статической страницы, чтобы получить заголовки текстов в блоге:

Копировать

```
const data = await page.evaluate( () => {

  let data = [];
  const titles = document.querySelectorAll('.brd_post_entry');

  for (const title of titles) {
    const titleText = title.querySelector('.brd_post_title').textContent;
    const titleLink = title.href;

    const article = { title: titleText, link: titleLink };
    data.push(article);
  }
});
```

```

        data.push(article);
    }

    return data;
})

console.log(data);

```

Вот полный код скрипта:

Пробная версия

```

(async () => {
    const browser = await puppeteer.launch({
        headless: false,
        defaultViewport: null
    });

    const page = await browser.newPage();
    await page.goto('https://brightdata.com/blog');

    const cookie_bar_accept = await
page.waitForSelector('#brd_cookies_bar_accept');
    await cookie_bar_accept.click();
    await new Promise(r => setTimeout(r, 500));

    await page.click('.search_icon');

    await page.waitForSelector('.search_container.active');
    const search_form = await page.waitForSelector('#blog_search');
    await search_form.type('puppeteer');

    await page.click('.search_icon');

    await new Promise(r => setTimeout(r, 2000));

    const data = await page.evaluate(() => {

        let data = [];
        const titles = document.querySelectorAll('.brd_post_entry');

        for (const title of titles) {
            const titleText = title.querySelector('.brd_post_title').textContent;
            const titleLink = title.href;

            const article = { title: titleText, link: titleLink };
            data.push(article);
        }

        return data;
    })

    console.log(data);

    await browser.close();

```

```
})(();
```

Можете ли вы сделать лучше?

Веб-скрапинг с помощью Puppeteer возможен, однако он не идеален. Библиотека создана для автоматизации тестирования, что делает ее немного неудобной для парсинга веб-страниц.

Пробная версия

вам необходимо самостоятельно найти [прокси-сеть](#) и заключить с ней контракт ([узнайте больше об интеграции прокси-сервера Puppeteer с Bright Data](#)).

Бесплатная пробная версия прокси-сети

Кроме того, оптимизировать Puppeteer для параллельного использования непросто. Если вы хотите собрать много данных, вам придется много работать, чтобы добиться оптимальной производительности.

Эти недостатки означают, что Puppeteer — хороший выбор для небольших скриптов для хобби, но для масштабирования операции вам может потребоваться много времени.

Если вам нужно что-то более простое в использовании, вы можете выбрать платформу веб-данных, например [Bright Data](#). Он позволяет компаниям собирать огромные объемы структурированных данных из Интернета с помощью простых в использовании инструментов, таких как [Scraping Browser](#) (совместимый с Puppeteer/Playwright), специально созданный для парсинга.

Бесплатная пробная версия Scraping Browser

Заключение

В этой статье вы узнали, как использовать Puppeteer для парсинга статических и динамических веб-страниц.

Puppeteer может выполнять многие действия как браузер, включая нажатие элементов, ввод текста и выполнение JavaScript. А благодаря использованию неявных ожиданий скрипты, написанные в Puppeteer, пишутся быстро и легко.

Однако могут быть и некоторые проблемы: Puppeteer — не самый эффективный инструмент для

парсинга веб-страниц, а его документация не совсем подходит для новичков. Кроме того, если вы не имеете достаточно опыта работы с Puppeteer, вам будет трудно масштабировать операции парсинга с его помощью.

Устали собирать данные самостоятельно? Получите предварительно собранные или пользовательские [наборы данных](#) .

Пробная версия

Вас также может заинтересовать

Как Тос

Как использовать прокси-серверы в Node.js

4 минуты чтения



Джакки Кукемоер

How to Set Proxy in AIOHTTP

```
async with aiohttp.ClientSession() as session:
    async with session.get(
        'https://example.com',
        proxy='http://190.6.23.219:999'
```

```
) as response:  
    print(response.status)
```

Как Тос

Как настроить прокси-сервер в AIOHTTP

4 минуты чтения

Пробная версия

Invoke-Webrequest With a Proxy



Как Тос

Как использовать PowerShell Invoke-WebRequest с прокси-сервером

4 минуты чтения

A

Антонелло Занини

data

ПРОДУКТЫ



ТИПЫ ПРОКСИ



ЦЕНЫ



ПРОГРАММЫ



ОБУЧЕНИЕ



ЮРИДИЧЕСКАЯ ИНФОРМАЦИЯ



bright data

Пробная версия

ПАРТНЕРСТВО ПО УЛУЧШЕНИЮ КАЧЕСТВА ОБСЛУЖИВАНИЯ КЛИЕНТОВ



ПАРТНЕРСКИЕ ОТНОШЕНИЯ



Пробная версия >

СВЯЖИТЕСЬ С НАМИ



ПРИСОЕДИНЯЙТЕСЬ

© Copyright 2024 Bright Data Ltd. | Все права защищены

Bright Data Ltd. (штаб-квартира), ул. Хамахшева, 4, Нетания, 4250714, Израиль (а/я 8025).
Bright Data, Inc., 500 7th Ave, 9th Floor Office 9A1234, Нью-Йорк, Нью-Йорк 10018, США.