УДК 004

Создание парсера с имитацией пользователя на Node.js

Вихляев Дмитрий Романович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается создание парсера скачиваемых данных с сайта rp5. Реализация программы написана на JavaScript с помощью Node.js. В результате исследования будет проведён анализ взаимодействия между клиентом и сервером и описан процесс написания парсера.
Ключевые слова: парсинг, Node.js, POST-запрос, Coockie, сессия.

Creating a parser with a simulated user on Node.js

Vikhlyaev Dmitry Romanovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses the creation of a parser for downloaded data from the rp5 website. The program implementation is written in JavaScript using Node.js . As a result of the research, an analysis of the interaction between the client and the server will be carried out and the process of writing the parser will be described.
Keywords: parsing, Node.js, POST request, Cookie, session.

1 Введение

1.1 Актуальность

Создание парсера с имитацией пользователя помогает обойти защиту от ботов и капчи. Многие сайты защищены от автоматического парсинга различными способами, такими как использование CAPTCHA или ограничение частоты запросов. Получение и использование данных таких как Coockie и session ID позволяет парсеру работать так, как будто это действия реального пользователя, что может помочь обойти некоторые из этих защитных мер. Также это необходимо для работы с авторизованным контентом. Некоторый контент на веб-сайтах доступен только для зарегистрированных пользователей. В таком случае получение session ID после авторизации (ввода логина и пароля) позволит парсеру получать доступ к закрытым разделам сайта. Сессии могут сохранять информацию о состоянии пользователя, такие как предпочтения, содержимое корзины покупок и другие данные. Имитируя пользователя, парсер может сохранять и использовать эту информацию для более точного и эффективного взаимодействия с сервером. Имитируя реального пользователя, парсер может

лучше взаимодействовать с динамическими элементами страницы, такими как AJAX-запросы или элементы, загружаемые при прокрутке страницы. Это делает процесс парсинга более надежным и полным.

1.2Обзор исследований

В.Н.Гринкин описал создание универсального web-парсера [1]. Е.И.Корнилов, Г.О.Рожков разработал web-сервис для автоматизации сбора данных [2]. А.С.Кучерина написала обзор методов и систем обработки данных с веб-страниц [3]. Р.Ю.Демина, Д.Э.Шукралиева иследовали методы защиты web-ресурсов от вредоносных парсеров [4]. В.Ю.Чистяков в своей статье обработал результаты парсинга [5]. Ж.С.Жукова, В.В.Ерофеева Д.А.Кулагин, К.С.Шварцман разработали комплекс программ парсеров для сбора климатических данных [6]. Т.Е.Фомина использовал Node.js для сбора и статистического анализа корпуса текстов [7]. А.А.Корепанова, Ф.В.Бушмелев, А.А.Сабреков в своей работе проведи иследование технологии парсинга Node.js в задаче агрегации сведений и оценки параметров грузовых маршрутов посредством извлечения данных из открытых источников [8].

1.3Цель исследования

Цель исследования – реализовать парсер для скачивания файла с сайта прогноза погоды rp5.

2 Материалы и методы

Для работы используется язык программирования JavaScript через Node.js. Проведение анализа взаимодействия клиента сервера осуществляется через инструменты браузера для разработчиков.

3 Результаты и обсуждения

Сайт rp5 предоставляет прогнозы погоды и информацию о фактической погоде, которая наблюдается на наземных станциях, всех стран мира. Дополнительно имеется раздел для просмотра и скачивания архивных данных метеостанций.

Чтобы скачать архив с данными нужно заполнить и отправить форму, в следствии создаётся ссылка на архив (рис.1).

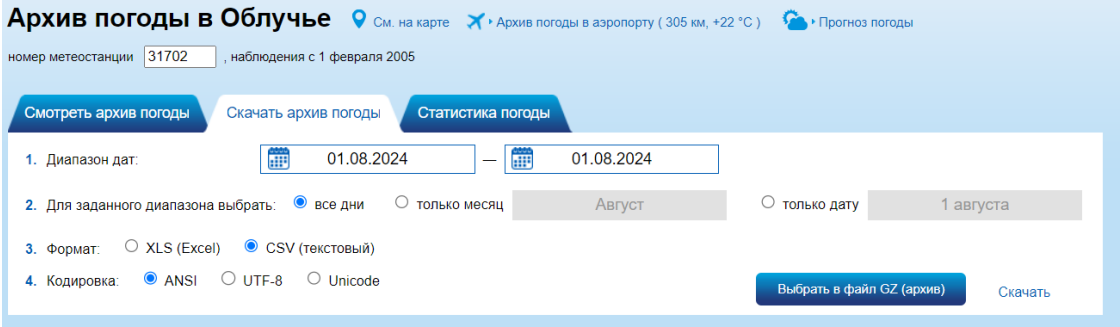

Рис. 1. Форма для скачивания архивных данных метеостанции

С помощью инструментов браузера для разработчиков, можно определить что происходит, когда пользователь нажимает на кнопку.

Исследуя заголовок запроса, становится ясно, что генерируется POST запрос по указанному URL адресу (рис.2).

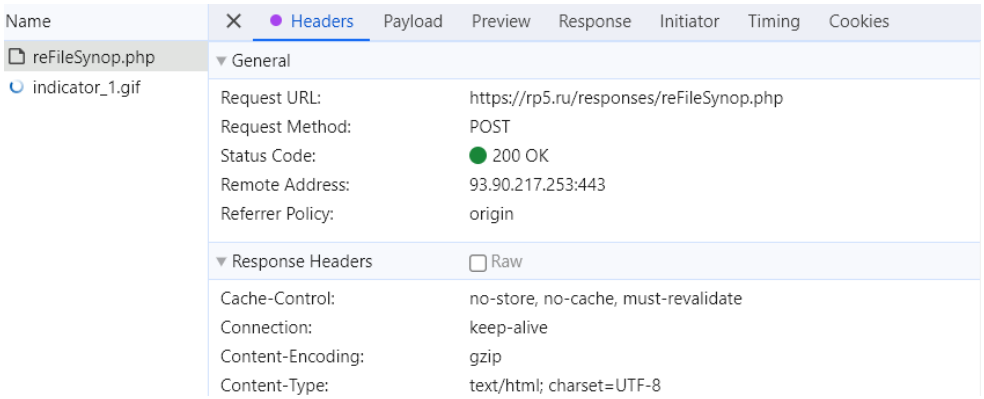

Рис. 2. Заголовок запроса на отправку формы

В разделе Payload для браузера Google Chrome, можно увидеть какие данные отправляются. Описание параметров и их расшифровка представлены на рисунке 3.

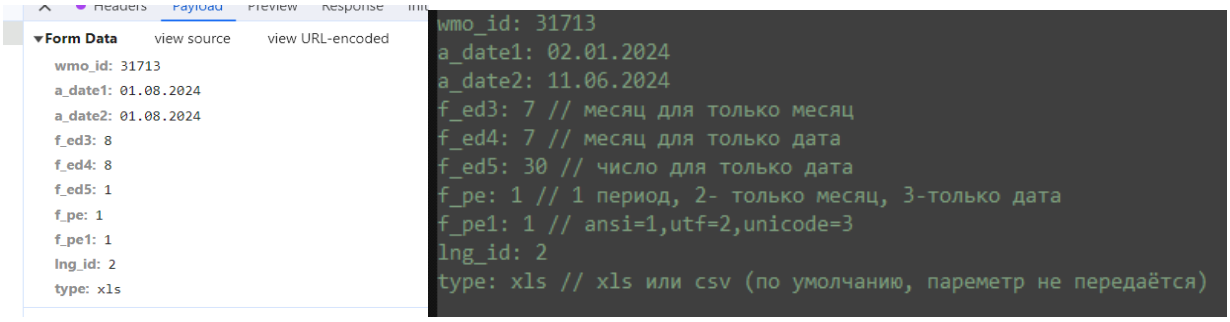

Рис. 3. Данные отправляемые через POST запрос

В качестве ответа сервер отправляет JavaScript код с добавлением ссылки на архив. Сама ссылка создаётся динамически, запрашиваемые данные лишь небольшой промежуток времени хранятся на сервере, после истечения периода ссылка становится недействительной, поэтому делать парсер на её основе нет смысла (рис.4).

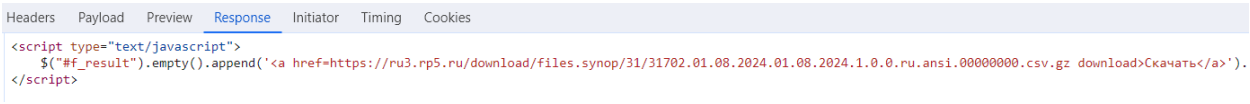

Рис. 4. Ответ сервера на запрос

Несмотря на то, что уже известны все шаги для реализации парсинга, обычный POST запрос по указанной в заголовке ссылке и отправкой данных ничего не даст. Следовательно, сервер требует ещё каких-то данных. Дополнительные данные могут храниться лишь в заголовке запроса. Одним из самых требовательных параметров являются данные Coockie.

Проанализировав их можно найти параметр PHPSESSID, что как нетрудно догадается, является идентификатором сессии. Для подтверждения, что идентификатор сессии действительно необходим, последовало две проверки. В первой, сразу после запроса браузера, были скопированы данные запроса и воспроизведены другим приложением. Какое-то время сервер корректно обрабатывал запрос, но потом перестал. Во второй проверке в браузере были отключены Coockie, вследствие чего при нажатии на кнопку отправки формы ссылка не появлялась, а в консоли разработчика выводилась ошибка JavaScript кода, на строку с проверкой вариации сессии (рис.5).

```
ClearErrors: function(){
    this.errors = this.fatal_errors = '';
    if(this.aErr_fields.indexOf(0) != -1) $(this.t_first).
    if(this.aErr_fields.indexOf(1) != -1) $(this.t_last).re
    this.aErr_fields = [];
    sessionStorage.setItem('archErrFields',null);
    this.bAccept = true;
```

Рис. 5. Ошибка при отсутствии идентификатора сессии

Однако одних данных сессии было мало для того чтобы сервер начал общение. Поэтому все данные передаваемые браузером были скопированы, в форме fetch запроса для Node.js (рис.6).

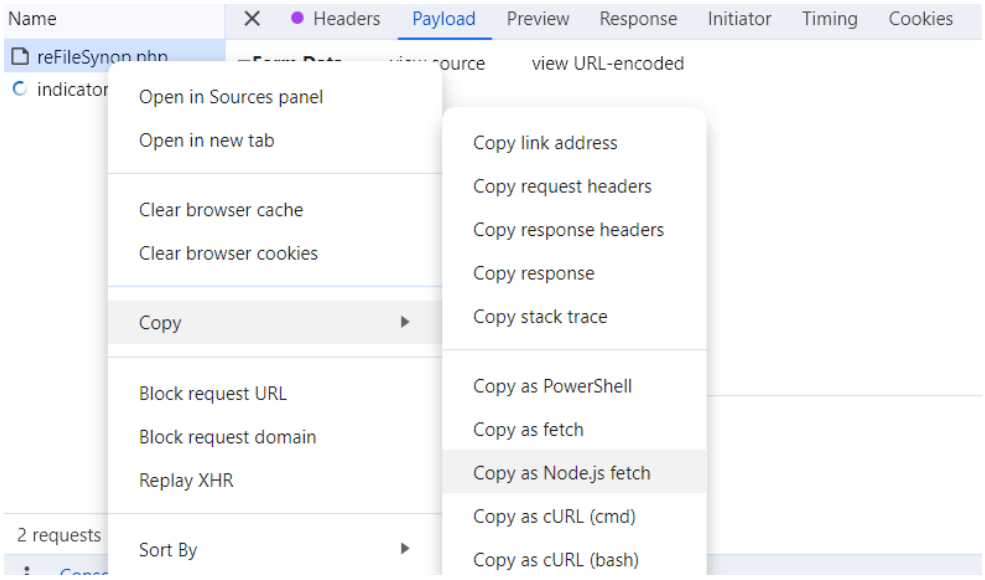

Рис. 6. Копирование всех данных запроса в браузере

Единственное, что нужно заменить это данные сессии, хранящиеся в Coockie. Сначала их нужно получить, сымитировав обычный GET запрос от браузера. Из полученного ответа получить данные Coockie (рис.7).

```
async function getCoockie()
{

    try {
        // URL для получения cookies
        const initialUrl = 'https://rp5.ru/';

        // Запрос для получения cookies
        const initialResponse = await fetch(initialUrl, {
          method: 'GET',
          headers: {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'
          }
        });

        // Проверка успешности запроса
        if (!initialResponse.ok) {
          throw new Error(`HTTP error! status: ${initialResponse.status}`);
        }

        // Получение cookies
        const cookies = initialResponse.headers.raw()['set-cookie'];
        console.log('Cookies:', cookies);

        // Преобразование массива cookies в строку
        const cookieHeader = cookies.map(cookie => cookie.split(';')[0]).join('; ');

        console.log(cookieHeader);
        return cookieHeader;
    } catch (error) {
        console.error('Error:', error);
        return null;
    }
}
```

Рис. 7. Имитация запроса от браузера на сервер и получение Coockie

Далее в скопированном запросе браузера в заголовок подставляются полученные данные Coockie, а в тело передаётся строка с параметрами. В переменную «params» помещены нужные значения параметров, затем они преобразовываются в URL строку и передаются переменной «body».

Ответ от сервера подвергается парсингу через регулярные выражения. В результате вытаскивается ссылка на файл для скачивания (рис.7).

```
  // Преобразование параметров в строку для тела запроса
  const body = new URLSearchParams(params).toString();

  // Выполнение POST-запроса с переданными cookies
  const response = await fetch(postUrl, {
    method: 'POST',
    headers: {
      'accept': 'text/html, */*; q=0.01',
      'accept-language': 'ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7',
      'content-type': 'application/x-www-form-urlencoded',
      'sec-ch-ua': '"Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"',
      'sec-ch-ua-mobile': '?1',
      'sec-ch-ua-platform': '"Windows"',
      'sec-fetch-dest': 'empty',
      'sec-fetch-mode': 'cors',
      'sec-fetch-site': 'same-origin',
      'x-requested-with': 'XMLHttpRequest',
      'cookie': cookieHeader, // Вставляем куки
      'Referer': 'https://rp5.ru/',
      'Referrer-Policy': 'origin'
    },
    body: body
  });

  // Проверка успешности запроса
  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }

  // Парсинг HTML ответа для извлечения ссылки на скачивание
  const html = await response.text();
  console.log(html);
  const $ = cheerio.load(html);
  const scriptContent = $('script').html();
  const linkMatch = scriptContent.match(/<a href=(https:\/\/[\w.\/\-]+) download>/);
  if (linkMatch && linkMatch[1]) {
    const downloadLink = linkMatch[1];
    console.log('Download Link:', downloadLink);
    return downloadLink;
  } else {
    throw new Error('Download link not found in response');
  }
} catch (error) {
      console.error('Error:', error);
      return;
}
```

Рис. 8. Реализация POST запроса и получение ссылки из JavaScript кода

Последними и самым простым этапом является отправка запроса и запись данных в локальный файл сервера (рис.9).

```
async function downloadFile(url, dest) {
  const writer = fs.createWriteStream(dest);

  const response = await axios({
    url,
    method: 'GET',
    responseType: 'stream'
  });

  response.data.pipe(writer);

  return new Promise((resolve, reject) => {
    writer.on('finish', () => {
      writer.close(() => {
        console.log('File successfully closed.');
        resolve();
      });
    });
    writer.on('error', (err) => {
      fs.unlink(dest, () => reject(err));
    });
  });
}
```

Рис. 9. Реализация функции запроса на скачивание файла и сохранение в локальном хранилище

В результате исследования был рассмотрен пример парсинга сайта с имитацией пользователя для получения идентификатора сессии через данные Coockie. Описаны методы и инструменты для анализа внутреннего взаимодействия пользователя с сервером. Представлена работа с инструментами разработчика в браузере. Реализована программа парсер написанная на JavaScript с помощью Node.js.

Библиографический список

1. Гринкин В.Н. Универсальный web-парсер // В сборнике: Студенческая научно-исследовательская лаборатория: современное состояние и перспективы. Сборник научных статей II Международной студенческой междисциплинарной научно-практической конференции. Краснодар, 2023. С. 133-141.
2. Корнилов Е.И., Рожков Г.О. Разработка web-сервиса для автоматизации сбора данных // В сборнике: Молодёжная наука 2024: технологии, инновации. Материалы Всероссийской научно-практической конференции, молодых учёных, аспирантов и студентов, посвящённой Десятилетию науки и технологий в Российской Федерации. В 4-х частях. Пермь, 2024. С. 73-78.
3. Кучерина А.С. Обзор методов и систем обработки данных с веб-страниц // В сборнике: Проблемы управления в социально-экономических и технических системах. Материалы XIX Международной научно-

практической конференции. Саратов, 2023. С. 390-394.
4. Демина Р.Ю., Шукралиева Д.Э. Методы защиты web-ресурсов от вредоносных парсеров // В сборнике: Проблемы повышения эффективности научной работы в оборонно-промышленном комплексе России. Материалы VI Всероссийской научно-практической конференции. Составитель С.Н. Бориско. Астрахань, 2023. С. 96-100.
5. Чистяков В.Ю. Обработка результата парсинга // RSDN Magazine. 2014. № 1. С. 47-54.
6. Жукова Ж.С., Ерофеева В.В., Кулагин Д.А., Шварцман К.С., Яблочников С.Л. Разработка комплекса программ парсеров для сбора климатических данных // Вестник Кыргызско-Российского Славянского университета. 2024. Т. 24. № 4. С. 18-25.
7. Фомина Т.Е. Использование Node.js для сбора и статистического анализа корпуса текстов // В сборнике: Четвертая зимняя школа по гуманитарной информатике. сборник статей. Калининград, 2020. С. 74-78.
8. Корепанова А.А., Бушмелев Ф.В., Сабреков А.А. Технологии парсинга Node.js в задаче агрегации сведений и оценки параметров грузовых маршрутов посредством извлечения данных из открытых источников // Компьютерные инструменты в образовании. 2021. № 3. С. 41-56.