Моделирование и анализ данных
2025. *15*(3), 180—185.
https://doi.org/10.17759/mda.2025150312
ISSN: 2219-3758 (печатный)
ISSN: 2311-9454 (online)

Modelling and Data Analysis
2025. *15*(3), 180—185.
https://doi.org/10.17759/mda.2025150312
ISSN: 2219-3758 (print)
ISSN: 2311-9454 (online)

Научная статья | Original paper

УДК 004.43:004.738.5

Особенности написания небуферизированного асинхронного алгоритма получения данных на языке JavaScript

С.С. Исаков
Московский государственный психолого-педагогический университет
Москва, Российская Федерация
✉ phebra@yandex.ru

Резюме

В статье рассматриваются два распространенных подхода к реализации скачивания файлов в веб-приложениях на JavaScript: использование HTML-тега `<a>` и асинхронная загрузка с помощью AJAX (fetch/XHR). Автор проводит сравнительный анализ их преимуществ и недостатков, уделяя особое внимание влиянию объема оперативной памяти пользовательского устройства на надежность работы каждого метода. Основной акцент сделан на проблему сбоев при использовании метода `response.blob()` для загрузки больших файлов в условиях ограниченной памяти. В качестве эффективного решения предлагается альтернативный метод с применением `response.bytes()`, обеспечивающий буферизированное, поэтапное скачивание и повышающий стабильность работы приложения. Материал будет полезен фронтенд-разработчикам, сталкивающимся с задачами реализации надежного и контролируемого процесса загрузки файлов.

Ключевые слова: программирование, Web-технологии, алгоритмы загрузки, оптимизация оперативной памяти, сетевая передача данных

Для цитирования: Исаков, С.С. (2025). Особенности написания небуферизированного асинхронного алгоритма получения данных на языке JavaScript. *Моделирование и анализ данных,* 15(3), 180—185. https://doi.org/10.17759/mda.2025150312

Features of writing an unbuffered asynchronous algorithm for receiving data in JavaScript

S.S. Isakov
Moscow State University of Psychology and Education, Moscow, Russian Federation
✉ phebra@yandex.ru

Abstract

The article discusses two common approaches to implementing file downloads in JavaScript web applications: using the HTML `<a>` tag and asynchronous downloading with AJAX (fetch/XHR). The author conducts a comparative analysis of their advantages and disadvantages, paying special attention to the impact of the user's device RAM on the reliability of each method. The main emphasis is on the problem of failures when using the `response.blob()` method to download large files under limited memory conditions. An alternative method using `response.bytes()` is proposed as an effective solution, providing buffered, step-by-step downloading and increasing the stability of the application. The material will be useful for front-end developers faced with the task of implementing a reliable and controlled file download process.

Keywords: programming, web technologies, loading algorithms, RAM optimization, network data transmission

For citation: Isakov, S.S. (2025). Features of writing an unbuffered asynchronous algorithm for receiving data in JavaScript. *Modelling and Data Analysis,* 15(3), 180—185. https://doi.org/10.17759/mda.2025150312

Введение

Рассматривается эффективный приём сетевой загрузки файлов, реализуемый на языке JavaScript (JS). Данный приём может быть актуален при создании как веб-ресурсов, так и десктопных приложений.

При создании страниц веб-сайтов нередко возникает потребность в предоставлении возможности скачивания файла для пользователя. Эта задача часто возникает при разработке ПО, и в большинстве случаев включает особые требования, в том числе анимированный процесс загрузки, отвечающий общему визуальному концепту разрабатываемого сайта. В большинстве случаев, при решении данной задачи используют два распространённых подхода:

- скачивание посредством использования HTML тэга `<a>`;
- асинхронное скачивание путём формирования AJAX запроса;

Однако два этих подхода будут по-разному функционировать в рамках исполнения JS кода браузером, особенно в условиях недостаточного объёма оперативной памяти компьютера.

Скачивание путём размещения ссылки на странице

При использовании данного подхода на странице размещается тэг ссылка, ведущая к URL адресу для скачивания файла, например, следующим образом:

```
<a href="https://example.com/file.pdf" download> Download
PDF </a>
```

Подобная ссылка будет открыта браузером ровно так же, как и совершается переход на новую странницу сайта. В случае исполнения традиционного перехода на новую страницу браузер получает заголовок `Content-Type: text/html; charset=UTF-8`, что указывает браузеру на то, что полезные данные, которые будут получены под данным заголовком, следует отобразить как HTML страницу. Но, в отличие от перехода на новую страницу сайта, при чтении заголовков ответа от сервера браузер получит следующие заголовки, диктующие особые инструкции по открытию данной ссылки:

```
Content-Type: application/pdf
Content-Disposition: attachment; filename="document.pdf"
Content-Length: 512000
```

В случае с заголовком `Content-Type: application/pdf` и двумя сопутствующими, браузер получает инструкцию что полезные данные в данной передаче необходимо воспринимать как файл, для последующего размещения браузером внутри файловой системы согласно настройкам браузера.

В некоторых случаях, для выполнения успешного скачивания на стороне сервера программистом формируются значения для данных заголовков, с последующей передачей в качестве ответа клиентской части в рамках организации HTTP взаимодействия. Чаще всего формирование заголовков требуется при использовании ссылки, явно не указывающей на расположение целевого файла на стороне сервера.

Преимущества и недостатки данного подхода

Весомым преимуществом данного подхода является то, что процесс получения данных с клиентской стороны и индикация процесса загрузки организована алгоритмами браузера. При скачивании файлов большого объёма браузер выполнит скачивание напрямую на жесткий диск, блоками в несколько этапов, незаметных пользователю, что делает такой подход независимым от оперативной памяти компьютера. С другой стороны, разработчик клиентской части сайта не имеет возможности контролировать, регулировать и анимировать процесс загрузки файла, что иногда бывает необходимым.

Скачивание путём исполнения асинхронного запроса

Другим подходом при скачивании файла является написание асинхронного алгоритма получения данных. Программную реализацию данного алгоритма можно обеспечить при помощи распространённых библиотек, таких как *jQuery, axios,* или же с использованием штатной функции языка JavaScript — *fetch*. Ниже представлен небольшой пример использования функции *fetch:*

```
fetch(«file.pdf»).then(response => {})
```

Более старым аналогом данной функции в рамках языка является алгоритм с использованием объекта XMLHttpRequest:

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "file.pdf", true);
xhr.responseType = "blob";
```

Особенностью таких алгоритмов является то, что процесс скачивания и получения полезных данных описывается программистом, разрабатывающим клиентскую часть веб-приложения. При необходимости данный процесс может вообще не отображаться пользователю. В силу того, что появляется возможность контролировать и отображать пользователю процесс загрузки в свободной анимированной форме, такой подход чаще используется программистами.

Согласно подавляющему большинству информационных источников в сети интернет, самым простым алгоритмом получения полезных данных является следующий набор строчек:

```
fetch(«file.pdf»)
   .then(response => {
      if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.
      status}`);
      }
      return response.blob();
   })
   .then(blob => {
   // выполнение операций с объектом BLOB
   })
   .catch(error => {
console.error("Download failed:", error);
});
```

В данном фрагменте инициатором процесса получения данных файла является строка: `return response.blob();`[1] Однако, в условиях малой оперативной

[1] Response: blob() method URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/blob (дата обращения: 10.07.2025).

памяти компьютера, на котором размещён браузер, и большого объёма скачиваемого файла, такой алгоритм остановится, получив первый блок скачиваемых данных. Максимальный допустимый объём одного фрагмента всего файла определяется оперативной памятью, выделяемой браузеру операционной системой, и может изменятся динамически, что ведёт к ошибке неполноты скачиваемых данных и делает алгоритм неработоспособным. Если же объём файла достаточно близок к максимальному допустимому объёму одного фрагмента скачивания, то со стороны пользователя ошибка становится «плавающей» и делает работу сайта нестабильной, а диагностику неисправности более трудоёмкой.

Решением данной проблемы является использование функции `response.bytes()`[2], которая выполнит буферизированное скачивание файла поэтапно, или же блоками:

```
fetch("file.pdf").then(response => {
   return response.bytes();
})
.then(bytes => {
   var blob = new Blob([bytes],{type: "application/pdf"})
   // выполнение операций с объектом BLOB
})
```

В данном примере в переменную *blob* будет записан объект класса Blob, ровно как и в примере с использованием функции `response.blob()`, рассмотренному выше, что делает эти два подхода взаимозаменяемыми.

Список источников / References

1. Response: blob() method URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/blob (дата обращения: 10.07.2025).
   Response: blob() method URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/blob (viewed: 10.07.2025).
2. Response: bytes() method URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/bytes (дата обращения: 10.07.2025).
   Response: bytes () method URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/bytes (viewed: 10.07.2025).

Информация об авторах

Исаков Сергей Сергеевич, младший научный сотрудник кафедры прикладной информатики и мультимедийных технологий, факультет информационных технологий, Московский государственный психолого-педагогический университет (ФГБОУ ВО МГППУ), Москва, Российская Федерация, ORCID: https://orcid.org/0000-0003-1719-2355, e-mail: phebra@yandex.ru

[2] Response: blob() method URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/blob (viewed: 10.07.2025).

Information about the authors

Isakov Sergey Sergeevich, Junior Researcher, Department of Applied Informatics and Multimedia Technologies, Faculty of Information Technology, Moscow State University of Psychology and Education, Moscow, Russian Federation, ORCID: https://orcid.org/0000-0003-1719-2355, e-mail: phebra@yandex.ru

Поступила в редакцию 14.07.2025
Поступила после рецензирования 28.07.2025
Принята к публикации 06.08.2025
Опубликована 30.09.2025

Received 2025.07.14
Revised 2025.07.28
Accepted 2025.08.06
Published 2025.09.30