


Раздел

[Сетевые запросы](#)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Сетевые запросы](#) 24-го августа 2019

Fetch: ход загрузки

Метод `fetch` позволяет отслеживать процесс *получения* данных.

Заметим, на данный момент в `fetch` нет способа отслеживать процесс *отправки*. Для этого используйте [XMLHttpRequest](#), позже мы его рассмотрим.

Чтобы отслеживать ход загрузки данных с сервера, можно использовать свойство `response.body`. Это `ReadableStream` («поток для чтения») – особый объект, который предоставляет тело ответа по частям, по мере поступления. Потоки для чтения описаны в спецификации [Streams API](#).

В отличие от `response.text()`, `response.json()` и других методов, `response.body` даёт полный контроль над процессом чтения, и мы можем подсчитать, сколько данных получено на каждый момент.

Вот примерный код, который читает ответ из `response.body`:

```
1 // вместо response.json() и других методов
2 const reader = response.body.getReader();
3
4 // бесконечный цикл, пока идёт загрузка
5 while(true) {
6   // done становится true в последнем фрагменте
7   // value - Uint8Array из байтов каждого фрагмента
8   const {done, value} = await reader.read();
9
10  if (done) {
11    break;
12  }
13
14  console.log(`Получено ${value.length} байт`)
15 }
```

Результат вызова `await reader.read()` – это объект с двумя свойствами:

- **done** – `true`, когда чтение закончено, иначе `false`.
- **value** – типизированный массив данных ответа `Uint8Array`.


На заметку:

`Streams API` также описывает асинхронный перебор по `ReadableStream`, при помощи цикла `for await...of`, но он пока слабо поддерживается (см. [задачи для браузеров](#)), поэтому используем цикл `while`.

Мы получаем новые фрагменты данных в цикле, пока загрузка не завершится, то есть пока `done` не станет `true`.

Чтобы отслеживать процесс загрузки, нам нужно при получении очередного фрагмента прибавлять его длину `value` к счётчику.

Вот полный рабочий пример, который получает ответ сервера и в процессе получения выводит в консоли длину полученных данных:

```
1 // Шаг 1: начинаем загрузку fetch, получаем поток  для ч
2 let response = await fetch('https://api.github.com/repo
3
4 const reader = response.body.getReader();
5
6 // Шаг 2: получаем длину содержимого ответа
```

Раздел

Сетевые запросы

Комментарии

Поделиться



Редактировать на GitHub



```
7 const contentLength = +response.headers.get('Content-Le
8
9 // Шаг 3: считываем данные:
10 let receivedLength = 0; // количество байт, полученных
11 let chunks = []; // массив полученных двоичных фрагмент
12 while(true) {
13     const {done, value} = await reader.read();
14
15     if (done) {
16         break;
17     }
18
19     chunks.push(value);
20     receivedLength += value.length;
21
22     console.log(`Получено ${receivedLength} из ${contentL
23 }
24
25 // Шаг 4: соединим фрагменты в общий типизированный мас
26 let chunksAll = new Uint8Array(receivedLength); // (4.1
27 let position = 0;
28 for(let chunk of chunks) {
29     chunksAll.set(chunk, position); // (4.2)
30     position += chunk.length;
31 }
32
33 // Шаг 5: декодируем Uint8Array обратно в строку
34 let result = new TextDecoder("utf-8").decode(chunksAll)
35
36 // Готово!
37 let commits = JSON.parse(result);
38 alert(commits[0].author.login);
```

Разберёмся, что здесь произошло:



1. Мы обращаемся к `fetch` как обычно, но вместо вызова `response.json()` мы получаем доступ к потоку чтения `response.body.getReader()`.

Обратите внимание, что мы не можем использовать одновременно оба эти метода для чтения одного и того же ответа: либо обычный метод `response.json()`, либо чтение потока `response.body`.

2. Ещё до чтения потока мы можем вычислить полную длину ответа из заголовка `Content-Length`.

Он может быть нечитаемым при запросах на другой источник (подробнее в разделе [Fetch: запросы на другие сайты](#)) и, в общем-то, серверу необязательно его устанавливать. Тем не менее, обычно длина указана.

3. Вызываем `await reader.read()` до окончания загрузки.

Всё, что получили, мы складываем по «кусочкам» в массив `chunks`. Это важно, потому что после того, как ответ получен, мы уже не сможем «перечитать» его, используя `response.json()` или любой другой способ (попробуйте – будет ошибка).

4. В самом конце у нас типизированный массив – `Uint8Array`. В нём находятся фрагменты данных. Нам нужно их склеить, чтобы получить строку. К сожалению, для этого нет специального метода, но можно сделать, например, так:

1. Создаём `chunksAll = new Uint8Array(receivedLength)` – массив того же типа заданной длины.
2. Используем `.set(chunk, position)` для копирования каждого фрагмента друг за другом в него.

5. Наш результат теперь хранится в `chunksAll`. Это не строка, а байтовый массив.

Чтобы получить именно строку, надо декодировать байты. Встроенный объект `TextDecoder` как раз этим и занимается. Потом мы можем, если необходимо, преобразовать строку в данные с помощью `JSON.parse`.

Раздел

[Сетевые запросы](#)

Комментарии

Поделиться



[Редактировать на GitHub](#)



Что если результат нам нужен в бинарном виде вместо строки? Это ещё проще. Замените шаги 4 и 5 на создание единого `Blob` из всех фрагментов:

```
1 let blob = new Blob(chunks);
```

В итоге у нас есть результат (строки или `Blob`, смотря что удобно) и отслеживание прогресса получения.

На всякий случай повторимся, что здесь мы рассмотрели, как отслеживать процесс получения данных с сервера, а не их отправки на сервер. Для отслеживания отправки у `fetch` пока нет способа.

Проводим [курсы по JavaScript и фреймворкам](#).

Комментарии

перед тем как писать...

