

Раздел

Сетевые запросы

Навигация по уроку

Заголовки ответа

Заголовки запроса

POST-запросы

Отправка изображения

Итого



Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub

 → Сетевые запросы 23-го июля 2020

Fetch

JavaScript может отправлять сетевые запросы на сервер и подгружать новую информацию по мере необходимости.

Например, мы можем использовать сетевой запрос, чтобы:

- Отправить заказ,
- Загрузить информацию о пользователе,
- Запросить последние обновления с сервера,
- ...и т.п.

Для сетевых запросов из JavaScript есть широко известный термин «AJAX» (аббревиатура от **A**synchronous **J**avascript **A**nd **X**ML). XML мы использовать не обязаны, просто термин старый, поэтому в нём есть это слово. Возможно, вы его уже где-то слышали.

Есть несколько способов делать сетевые запросы и получать информацию с сервера.

Метод `fetch()` — современный и очень мощный, поэтому начнём с него. Он не поддерживается старыми (можно использовать полифил), но поддерживается всеми современными браузерами.

Базовый синтаксис:

```
1 let promise = fetch(url, [options])
```

- **url** — URL для отправки запроса.
- **options** — дополнительные параметры: метод, заголовки и так далее.

Без `options` это простой GET-запрос, скачивающий содержимое по адресу `url`.

Браузер сразу же начинает запрос и возвращает промис, который внешний код использует для получения результата.

Процесс получения ответа обычно происходит в два этапа.

Во-первых, `promise` выполняется с объектом встроенного класса `Response` в качестве результата, как только сервер пришлёт заголовки ответа.

На этом этапе мы можем проверить статус HTTP-запроса и определить, выполнен ли он успешно, а также посмотреть заголовки, но пока без тела ответа.

Промис завершается с ошибкой, если `fetch` не смог выполнить HTTP-запрос, например при ошибке сети или если нет такого сайта. HTTP-статусы 404 и 500 не являются ошибкой.

Мы можем увидеть HTTP-статус в свойствах ответа:

- **status** — код статуса HTTP-запроса, например 200.
- **ok** — логическое значение: будет `true`, если код HTTP-статуса в диапазоне 200-299.

Например:

```
1 let response = await fetch(url);
2
3 if (response.ok) { // если HTTP-статус в диапазоне 200-
4   // получаем тело ответа (см. про этот метод ниже)
5   let json = await response.json();
6 } else {
7
```

```
8   alert("Ошибка HTTP: " + response.status);
}
```

Раздел

[Сетевые запросы](#)

Навигация по уроку

Заголовки ответа

Заголовки запроса

POST-запросы

Отправка изображения

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



Во-вторых, для получения тела ответа нам нужно использовать дополнительный вызов метода.

Response предоставляет несколько методов, основанных на промисах, для доступа к телу ответа в различных форматах:

- **response.text()** – читает ответ и возвращает как обычный текст,
- **response.json()** – декодирует ответ в формате JSON,
- **response.formData()** – возвращает ответ как объект FormData (разберём его в [следующей главе](#)),
- **response.blob()** – возвращает объект как Blob (бинарные данные с типом),
- **response.arrayBuffer()** – возвращает ответ как ArrayBuffer (низкоуровневое представление бинарных данных),
- помимо этого, response.body – это объект ReadableStream, с помощью которого можно считывать тело запроса по частям. Мы рассмотрим и такой пример несколько позже.

Например, получим JSON-объект с последними коммитами из репозитория на GitHub:

```
1 let url = 'https://api.github.com/repos/javascript-tuto
2 let response = await fetch(url);
3
4 let commits = await response.json(); // читаем ответ в
5
6 alert(commits[0].author.login);
```

То же самое без await, с использованием промисов:

```
1 fetch('https://api.github.com/repos/javascript-tutorial
2   .then(response => response.json())
3   .then(commits => alert(commits[0].author.login));
```

Для получения ответа в виде текста используем await response.text() вместо .json():

```
1 let response = await fetch('https://api.github.com/repos
2
3 let text = await response.text(); // прочитать тело отв
4
5 alert(text.slice(0, 80) + '...');
```

В качестве примера работы с бинарными данными, давайте запросим и выведем на экран логотип [спецификации «fetch»](#) (см. главу [Blob](#), чтобы узнать про операции с Blob):

```
1 let response = await fetch('/article/fetch/logo-fetch.s
2
3 let blob = await response.blob(); // скачиваем как Blob
4
5 // создаём <img>
6 let img = document.createElement('img');
7 img.style = 'position:fixed;top:10px;left:10px;width:10
8 document.body.append(img);
9
10 // выводим на экран
11 img.src = URL.createObjectURL(blob);
12
13 setTimeout(() => { // прячем через три секунды
14   img.remove();
```

```
15 URL.revokeObjectURL(img.src);
16 }, 3000);
```

Раздел

[Сетевые запросы](#)

Навигация по уроку

Заголовки ответа

Заголовки запроса

POST-запросы

Отправка изображения

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



⚠ Важно:

Мы можем выбрать только один метод чтения ответа.

Если мы уже получили ответ с `response.text()`, тогда `response.json()` не сработает, так как данные уже были обработаны.

```
1 let text = await response.text(); // тело ответа с
2 let parsed = await response.json(); // ошибка (да
```

Заголовки ответа

Заголовки ответа хранятся в похожем на `Map` объекте `response.headers`.

Это не совсем `Map`, но мы можем использовать такие же методы, как с `Map`, чтобы получить заголовок по его имени или перебрать заголовки в цикле:

```
1 let response = await fetch('https://api.github.com/repo
2
3 // получить один заголовок
4 alert(response.headers.get('Content-Type')); // applica
5
6 // перебрать все заголовки
7 for (let [key, value] of response.headers) {
8   alert(`${key} = ${value}`);
9 }
```

Заголовки запроса

Для установки заголовка запроса в `fetch` мы можем использовать опцию `headers`. Она содержит объект с исходящими заголовками, например:

```
1 let response = fetch(protectedUrl, {
2   headers: {
3     Authentication: 'secret'
4   }
5 });
```

Есть список [запрещённых HTTP-заголовков](#), которые мы не можем установить:

- `Accept-Charset`, `Accept-Encoding`
- `Access-Control-Request-Headers`
- `Access-Control-Request-Method`
- `Connection`
- `Content-Length`
- `Cookie`, `Cookie2`
- `Date`
- `DNT`
- `Expect`
- `Host`
- `Keep-Alive`
- `Origin`
- `Referer`
- `TE`
- `Trailer`

Раздел

[Сетевые запросы](#)

Навигация по уроку

Заголовки ответа

Заголовки запроса

POST-запросы

Отправка изображения

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



- Transfer-Encoding
- Upgrade
- Via
- Proxy-*
- Sec-*

Эти заголовки обеспечивают достоверность данных и корректную работу протокола HTTP, поэтому они контролируются исключительно браузером.

POST-запросы

Для отправки POST -запроса или запроса с другим методом, нам необходимо использовать `fetch` параметры:

- **method** – HTTP метод, например `POST`,
- **body** – тело запроса, одно из списка:
 - строка (например, в формате `JSON`),
 - объект `FormData` для отправки данных как `form/multipart`,
 - `Blob / BufferSource` для отправки бинарных данных,
 - [URLSearchParams](#) для отправки данных в кодировке `x-www-form-urlencoded`, используется редко.

Чаще всего используется `JSON`.

Например, этот код отправляет объект `user` как `JSON`:

```
1 let user = {
2   name: 'John',
3   surname: 'Smith'
4 };
5
6 let response = await fetch('/article/fetch/post/user',
7   method: 'POST',
8   headers: {
9     'Content-Type': 'application/json;charset=utf-8'
10  },
11  body: JSON.stringify(user)
12 });
13
14 let result = await response.json();
15 alert(result.message);
```

Заметим, что так как тело запроса `body` – строка, то заголовок `Content-Type` по умолчанию будет `text/plain;charset=UTF-8`.

Но, так как мы посылаем `JSON`, то используем параметр `headers` для отправки вместо этого `application/json`, правильный `Content-Type` для `JSON`.

Отправка изображения

Мы можем отправить бинарные данные при помощи `fetch`, используя объекты `Blob` или `BufferSource`.

В этом примере есть элемент `<canvas>`, на котором мы можем рисовать движением мыши. При нажатии на кнопку «Отправить» изображение отправляется на сервер:

```
1 <body style="margin:0">
2   <canvas id="canvasElem" width="100" height="80" style=
3
4   <input type="button" value="Отправить" onclick="submi
5
6   <script>
7     canvasElem.onmousemove = function(e) {
8       let ctx = canvasElem.getContext('2d');
9       ctx.lineTo(e.clientX, e.clientY);
10      ctx.stroke();
```

Раздел

Сетевые запросы

Навигация по уроку

Заголовки ответа

Заголовки запроса

POST-запросы

Отправка изображения

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



```
11     };
12
13     async function submit() {
14         let blob = await new Promise(resolve => canvasElem
15         let response = await fetch('/article/fetch/post/i
16             method: 'POST',
17             body: blob
18         });
19
20         // сервер ответит подтверждением и размером изобра
21         let result = await response.json();
22         alert(result.message);
23     }
24
25     </script>
26 </body>
```

Отправить

Заметим, что здесь нам не нужно вручную устанавливать заголовок Content-Type, потому что объект Blob имеет встроенный тип (image/png, заданный в toBlob). При отправке объектов Blob он автоматически становится значением Content-Type.

Функция submit() может быть переписана без async/await, например, так:

```
1 function submit() {
2     canvasElem.toBlob(function(blob) {
3         fetch('/article/fetch/post/image', {
4             method: 'POST',
5             body: blob
6         })
7         .then(response => response.json())
8         .then(result => alert(JSON.stringify(result, null
9     }, 'image/png'));
10 }
```

Итого

Типичный запрос с помощью fetch состоит из двух операторов await:

```
1 let response = await fetch(url, options); // завершается
2 let result = await response.json(); // читать тело отве
```

Или, без await:

```
1 fetch(url, options)
2     .then(response => response.json())
3     .then(result => /* обрабатываем результат */)
```

Параметры ответа:

- response.status – HTTP-код ответа,
- response.ok – true, если статус ответа в диапазоне 200-299.
- response.headers – похожий на Map объект с HTTP-заголовками.

Методы для получения тела ответа:

- response.text() – возвращает ответ как обычный текст,
- response.json() – преобразовывает ответ в JSON-объект,
- response.formData() – возвращает ответ как объект FormData (кодировка form/multipart, см. следующую главу),

Раздел

[Сетевые запросы](#)

Навигация по уроку

Заголовки ответа

Заголовки запроса

POST-запросы

Отправка изображения

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



- `response.blob()` – возвращает объект как [Blob](#) (бинарные данные с типом),
- `response.arrayBuffer()` – возвращает ответ как [ArrayBuffer](#) (низкоуровневые бинарные данные),

Опции `fetch`, которые мы изучили на данный момент:

- `method` – HTTP-метод,
- `headers` – объект с запрашиваемыми заголовками (не все заголовки разрешены),
- `body` – данные для отправки (тело запроса) в виде текста, `FormData`, `BufferSource`, `Blob` или `UrlSearchParams`.

В следующих главах мы рассмотрим больше параметров и вариантов использования `fetch`.

✓ Задачи

Получите данные о пользователях GitHub

Создайте асинхронную функцию `getUsers(names)`, которая получает на вход массив логинов пользователей GitHub, запрашивает у GitHub информацию о них и возвращает массив объектов-пользователей.

Информация о пользователе GitHub с логином `USERNAME` доступна по ссылке: `https://api.github.com/users/USERNAME`.

В песочнице есть тестовый пример.

Важные детали:

- На каждого пользователя должен приходиться один запрос `fetch`.
- Запросы не должны ожидать завершения друг друга. Надо, чтобы данные приходили как можно быстрее.
- Если какой-то запрос завершается ошибкой или оказалось, что данных о запрашиваемом пользователе нет, то функция должна возвращать `null` в массиве результатов.

[Открыть песочницу с тестами для задачи.](#)

решение

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

💬 Комментарии

перед тем как писать...