

Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник


Итого

Комментарии

Поделиться



Редактировать на GitHub

 → Сетевые запросы 6-го сентября 2020

# XMLHttpRequest

XMLHttpRequest – это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.

Несмотря на наличие слова «XML» в названии, XMLHttpRequest может работать с любыми данными, а не только с XML. Мы можем загружать/скачивать файлы, отслеживать прогресс и многое другое.

На сегодняшний день не обязательно использовать XMLHttpRequest, так как существует другой, более современный метод `fetch`.

В современной веб-разработке XMLHttpRequest используется по трём причинам:

1. По историческим причинам: существует много кода, использующего XMLHttpRequest, который нужно поддерживать.
2. Необходимость поддерживать старые браузеры и нежелание использовать полифилы (например, чтобы уменьшить количество кода).
3. Потребность в функциональности, которую `fetch` пока что не может предоставить, к примеру, отслеживание прогресса отправки на сервер.

Что-то из этого списка звучит знакомо? Если да, тогда вперёд, приятного знакомства с XMLHttpRequest. Если же нет, возможно, имеет смысл изучать сразу [Fetch](#).

## Основы

XMLHttpRequest имеет два режима работы: синхронный и асинхронный.

Сначала рассмотрим асинхронный, так как в большинстве случаев используется именно он.

Чтобы сделать запрос, нам нужно выполнить три шага:

1. Создать XMLHttpRequest.

```
1 let xhr = new XMLHttpRequest(); // у конструктора нет
```

2. Инициализировать его.

```
1 xhr.open(method, URL, [async, user, password])
```

Этот метод обычно вызывается сразу после `new XMLHttpRequest`. В него передаются основные параметры запроса:

- `method` – HTTP-метод. Обычно это "GET" или "POST".
- `URL` – URL, куда отправляется запрос: строка, может быть и объект [URL](#).
- `async` – если указать `false`, тогда запрос будет выполнен синхронно, это мы рассмотрим чуть позже.
- `user`, `password` – логин и пароль для базовой HTTP-авторизации (если требуется).

Заметим, что вызов `open`, вопреки своему названию, не открывает соединение. Он лишь конфигурирует запрос, но непосредственно отсылается запрос только лишь после вызова `send`.

3. Послать запрос.

```
1 xhr.send([body])
```

Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



Редактировать на GitHub



Этот метод устанавливает соединение и отправляет запрос к серверу. Необязательный параметр `body` содержит тело запроса.

Некоторые типы запросов, такие как `GET`, не имеют тела. А некоторые, как, например, `POST`, используют `body`, чтобы отправлять данные на сервер. Мы позже увидим примеры.

#### 4. Слушать события на `xhr`, чтобы получить ответ.

Три наиболее используемых события:

- `load` – происходит, когда получен какой-либо ответ, включая ответы с HTTP-ошибкой, например 404.
- `error` – когда запрос не может быть выполнен, например, нет соединения или невалидный URL.
- `progress` – происходит периодически во время загрузки ответа, сообщает о прогрессе.

```
1 xhr.onload = function() {
2   alert(`Загружено: ${xhr.status} ${xhr.response}`);
3 };
4
5 xhr.onerror = function() { // происходит, только когд
6   alert(`Ошибка соединения`);
7 };
8
9 xhr.onprogress = function(event) { // запускается пер
10  // event.loaded - количество загруженных байт
11  // event.lengthComputable = равно true, если сервер
12  // event.total - количество байт всего (только если
13  alert(`Загружено ${event.loaded} из ${event.total}`
14  };
```

Вот полный пример. Код ниже загружает `/article/xmlhttprequest/example/load` с сервера и сообщает о прогрессе:

```
1 // 1. Создаём новый XMLHttpRequest-объект
2 let xhr = new XMLHttpRequest();
3
4 // 2. Настраиваем его: GET-запрос по URL /article/.../l
5 xhr.open('GET', '/article/xmlhttprequest/example/load')
6
7 // 3. Отсылаем запрос
8 xhr.send();
9
10 // 4. Этот код сработает после того, как мы получим отв
11 xhr.onload = function() {
12   if (xhr.status !== 200) { // анализируем HTTP-статус о
13     alert(`Ошибка ${xhr.status}: ${xhr.statusText}`); /
14   } else { // если всё прошло гладко, выводим результат
15     alert(`Готово, получили ${xhr.response.length} байт
16   }
17 };
18
19 xhr.onprogress = function(event) {
20   if (event.lengthComputable) {
21     alert(`Получено ${event.loaded} из ${event.total} б
22   } else {
23     alert(`Получено ${event.loaded} байт`); // если в о
24   }
25
26 };
27
28 xhr.onerror = function() {
29   alert("Запрос не удался");
30 };
```

Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



Редактировать на GitHub



После ответа сервера мы можем получить результат запроса в следующих свойствах `xhr` :

### status

Код состояния HTTP (число): 200 , 404 , 403 и так далее, может быть 0 в случае, если ошибка не связана с HTTP.

### statusText

Сообщение о состоянии ответа HTTP (строка): обычно OK для 200 , Not Found для 404 , Forbidden для 403 , и так далее.

### response (в старом коде может встречаться как responseText)

Тело ответа сервера.

Мы можем также указать таймаут – промежуток времени, который мы готовы ждать ответ:

```
1  xhr.timeout = 10000; // таймаут указывается в миллисекундах
```

Если запрос не успевает выполниться в установленное время, то он прерывается, и происходит событие `timeout` .

#### URL с параметрами

Чтобы добавить к URL параметры, вида `?name=value` , и корректно закодировать их, можно использовать объект [URL](#):

```
1  let url = new URL('https://google.com/search');
2  url.searchParams.set('q', 'test me!');
3
4  // параметр 'q' закодирован
5  xhr.open('GET', url); // https://google.com/search
```

## Тип ответа

Мы можем использовать свойство `xhr.responseType` , чтобы указать ожидаемый тип ответа:

- "" (по умолчанию) – строка,
- "text" – строка,
- "arraybuffer" – `ArrayBuffer` (для бинарных данных, смотрите в [ArrayBuffer](#), бинарные массивы),
- "blob" – `Blob` (для бинарных данных, смотрите в [Blob](#)),
- "document" – XML-документ (может использовать XPath и другие XML-методы),
- "json" – JSON (парсится автоматически).

К примеру, давайте получим ответ в формате JSON:

```
1  let xhr = new XMLHttpRequest();
2
3  xhr.open('GET', '/article/xmlhttprequest/example/json');
4
5  xhr.responseType = 'json';
6
7  xhr.send();
8
9  // тело ответа {"сообщение": "Привет, мир!"}
10 xhr.onload = function() {
11   let responseObj = xhr.response;
12   alert(responseObj.message); // Привет, мир!
13 };
```

Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



Редактировать на GitHub



#### На заметку:

В старом коде вы можете встретить свойства `xhr.responseText` и даже `xhr.responseXML`.

Они существуют по историческим причинам, раньше с их помощью получали строки или XML-документы. Сегодня следует устанавливать желаемый тип объекта в `xhr.responseType` и получать `xhr.response`, как показано выше.

## Состояния запроса

У `XMLHttpRequest` есть состояния, которые меняются по мере выполнения запроса. Текущее состояние можно посмотреть в свойстве `xhr.readyState`.

Список всех состояний, указанных в [спецификации](#):

```
1 UNSENT = 0; // исходное состояние
2 OPENED = 1; // вызван метод open
3 HEADERS_RECEIVED = 2; // получены заголовки ответа
4 LOADING = 3; // ответ в процессе передачи (данные части
5 DONE = 4; // запрос завершён
```

Состояния объекта `XMLHttpRequest` меняются в таком порядке:  $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow 3 \rightarrow 4$ . Состояние 3 повторяется каждый раз, когда получена часть данных.

Изменения в состоянии объекта запроса генерируют событие `readystatechange`:

```
1 xhr.onreadystatechange = function() {
2   if (xhr.readyState == 3) {
3     // загрузка
4   }
5   if (xhr.readyState == 4) {
6     // запрос завершён
7   }
8 };
```

Вы можете наткнуться на обработчики события `readystatechange` в очень старом коде, так уж сложилось исторически, когда-то не было событий `load` и других. Сегодня из-за существования событий `load/error/progress` можно сказать, что событие `readystatechange` «морально устарело».

## Отмена запроса

Если мы передумали делать запрос, можно отменить его вызовом `xhr.abort()`:

```
1 xhr.abort(); // завершить запрос
```

При этом генерируется событие `abort`, а `xhr.status` устанавливается в 0.

## Синхронные запросы

Если в методе `open` третий параметр `async` установлен на `false`, запрос выполняется синхронно.

Другими словами, выполнение JavaScript останавливается на `send()` и возобновляется после получения ответа. Так ведут себя, например, функции `alert` или `prompt`.

Вот переписанный пример с параметром `async`, равным `false`:

Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
1 let xhr = new XMLHttpRequest();
2
3 xhr.open('GET', '/article/xmlhttprequest/hello.txt', fa
4
5 try {
6     xhr.send();
7     if (xhr.status !== 200) {
8         alert(`Ошибка ${xhr.status}: ${xhr.statusText}`);
9     } else {
10         alert(xhr.response);
11     }
12 } catch(err) { // для отлова ошибок используем констук
13     alert("Запрос не удался");
14 }
```

Выглядит, может быть, и неплохо, но синхронные запросы используются редко, так как они блокируют выполнение JavaScript до тех пор, пока загрузка не завершена. В некоторых браузерах нельзя прокручивать страницу, пока идёт синхронный запрос. Ну а если же синхронный запрос по какой-то причине выполняется слишком долго, браузер предложит закрыть «зависшую» страницу.

Многие продвинутые возможности XMLHttpRequest, такие как выполнение запроса на другой домен или установка таймаута, недоступны для синхронных запросов. Также, как вы могли заметить, ни о какой индикации прогресса речь тут не идёт.

Из-за всего этого синхронные запросы используют очень редко. Мы более не будем рассматривать их.

## HTTP-заголовки

XMLHttpRequest умеет как указывать свои заголовки в запросе, так и читать присланные в ответ.



Для работы с HTTP-заголовками есть 3 метода:



### setRequestHeader(name, value)

Устанавливает заголовок запроса с именем name и значением value.

Например:

```
1 xhr.setRequestHeader('Content-Type', 'application/json')
```

#### ⚠ Ограничения на заголовки

Некоторые заголовки управляются исключительно браузером, например Referer или Host, а также ряд других. Полный список [тут](#).

XMLHttpRequest не разрешено изменять их ради безопасности пользователей и для обеспечения корректности HTTP-запроса.

Раздел

[Сетевые запросы](#)

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



### ⚠ Поставленный заголовок нельзя снять

Ещё одной особенностью XMLHttpRequest является то, что отменить `setRequestHeader` невозможно.

Если заголовок определён, то его нельзя снять. Повторные вызовы лишь добавляют информацию к заголовку, а не перезаписывают его.

Например:

```
1 xhr.setRequestHeader('X-Auth', '123');
2 xhr.setRequestHeader('X-Auth', '456');
3
4 // заголовок получится такой:
5 // X-Auth: 123, 456
```

### `getResponseHeader(name)`

Возвращает значение заголовка ответа `name` (кроме `Set-Cookie` и `Set-Cookie2`).

Например:

```
1 xhr.getResponseHeader('Content-Type')
```

### `getAllResponseHeaders()`

Возвращает все заголовки ответа, кроме `Set-Cookie` и `Set-Cookie2`.

Заголовки возвращаются в виде единой строки, например:

```
1 Cache-Control: max-age=31536000
2 Content-Length: 4260
3 Content-Type: image/png
4 Date: Sat, 08 Sep 2012 16:53:16 GMT
```

Между заголовками всегда стоит перевод строки в два символа `"\r\n"` (независимо от ОС), так что мы можем легко разделить их на отдельные заголовки. Значение заголовка всегда отделено двоеточием с пробелом `": "`. Этот формат задан стандартом.

Таким образом, если хочется получить объект с парами заголовок-значение, нам нужно задействовать немного JS.

Вот так (предполагается, что если два заголовка имеют одинаковое имя, то последний перезаписывает предыдущий):

```
1 let headers = xhr
2   .getAllResponseHeaders()
3   .split('\r\n')
4   .reduce((result, current) => {
5     let [name, value] = current.split(': ');
6     result[name] = value;
7     return result;
8   }, {});
9
10 // headers['Content-Type'] = 'image/png'
```

## POST, FormData

Чтобы сделать POST-запрос, мы можем использовать встроенный объект [FormData](#).

Синтаксис:

```
1 let formData = new FormData([form]); // создаём объект,
```

```
2 formData.append(name, value); // добавляем поле
```

Мы создаём объект, при желании указываем, из какой формы form взять данные, затем, если нужно, с помощью метода append добавляем дополнительные поля, после чего:

1. `xhr.open('POST', ...)` – создаём POST-запрос.
2. `xhr.send(formData)` – отправляем форму серверу.

Например:

```
1 <form name="person">
2   <input name="name" value="Петя">
3   <input name="surname" value="Васечкин">
4 </form>
5
6 <script>
7   // заполним FormData данными из формы
8   let formData = new FormData(document.forms.person);
9
10  // добавим ещё одно поле
11  formData.append("middle", "Иванович");
12
13  // отправим данные
14  let xhr = new XMLHttpRequest();
15  xhr.open("POST", "/article/xmlhttprequest/post/user");
16  xhr.send(formData);
17
18  xhr.onload = () => alert(xhr.response);
19 </script>
```

Обычно форма отправляется в кодировке `multipart/form-data`.

Если нам больше нравится формат JSON, то используем `JSON.stringify` и отправляем данные как строку.

Важно не забыть поставить соответствующий заголовок `Content-Type`: `application/json`, многие серверные фреймворки автоматически декодируют JSON при его наличии:

```
1 let xhr = new XMLHttpRequest();
2
3 let json = JSON.stringify({
4   name: "Вася",
5   surname: "Петров"
6 });
7
8 xhr.open("POST", '/submit')
9 xhr.setRequestHeader('Content-type', 'application/json;
10
11 xhr.send(json);
```

Метод `.send(body)` весьма всеяден. Он может отправить практически что угодно в `body`, включая объекты типа `Blob` и `BufferSource`.

## Прогресс отправки

Событие `progress` срабатывает только на стадии загрузки ответа с сервера.

А именно: если мы отправляем что-то через POST-запрос, `XMLHttpRequest` сперва отправит наши данные (тело запроса) на сервер, а потом загрузит ответ сервера. И событие `progress` будет срабатывать только во время загрузки ответа.

Если мы отправляем что-то большое, то нас гораздо больше интересует прогресс отправки данных на сервер. Но `xhr.onprogress` тут не поможет.

Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



Редактировать на GitHub

Раздел

[Сетевые запросы](#)

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



Существует другой объект, без методов, только для отслеживания событий отправки: `xhr.upload`.

Он генерирует события, похожие на события `xhr`, но только во время отправки данных на сервер:

- `loadstart` – начало загрузки данных.
- `progress` – генерируется периодически во время отправки на сервер.
- `abort` – загрузка прервана.
- `error` – ошибка, не связанная с HTTP.
- `load` – загрузка успешно завершена.
- `timeout` – вышло время, отведённое на загрузку (при установленном свойстве `timeout`).
- `loadend` – загрузка завершена, вне зависимости от того, как – успешно или нет.

Примеры обработчиков для этих событий:

```
1 xhr.upload.onprogress = function(event) {
2   alert(`Отправлено ${event.loaded} из ${event.total} б
3 };
4
5 xhr.upload.onload = function() {
6   alert(`Данные успешно отправлены.`);
7 };
8
9 xhr.upload.onerror = function() {
10  alert(`Произошла ошибка во время отправки: ${xhr.stat
11 };
```

Пример из реальной жизни: загрузка файла на сервер с индикацией прогресса:



```
1 <input type="file" onchange="upload(this.files[0])">
2
3 <script>
4 function upload(file) {
5   let xhr = new XMLHttpRequest();
6
7   // отслеживаем процесс отправки
8   xhr.upload.onprogress = function(event) {
9     console.log(`Отправлено ${event.loaded} из ${event.t
10  };
11
12   // Ждём завершения: неважно, успешного или нет
13   xhr.onloadend = function() {
14     if (xhr.status == 200) {
15       console.log("Успех");
16     } else {
17       console.log("Ошибка " + this.status);
18     }
19   };
20
21   xhr.open("POST", "/article/xmlhttprequest/post/upload
22   xhr.send(file);
23 }
24 </script>
```



## Запросы на другой источник

`XMLHttpRequest` может осуществлять запросы на другие сайты, используя ту же политику CORS, что и `fetch`.

Точно так же, как и при работе с `fetch`, по умолчанию на другой источник не отсылаются куки и заголовки HTTP-авторизации. Чтобы это изменить, установите `xhr.withCredentials` в `true`:



Раздел

Сетевые запросы

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
1 let xhr = new XMLHttpRequest();
2 xhr.withCredentials = true;
3
4 xhr.open('POST', 'http://anywhere.com/request');
5 ...
```

Детали по заголовкам, которые при этом необходимы, смотрите в главе [fetch](#).

## Итого

Типичный код GET-запроса с использованием XMLHttpRequest :

```
1 let xhr = new XMLHttpRequest();
2
3 xhr.open('GET', '/my/url');
4
5 xhr.send();
6
7 xhr.onload = function() {
8   if (xhr.status !== 200) { // HTTP ошибка?
9     // обработаем ошибку
10    alert( 'Ошибка: ' + xhr.status);
11    return;
12   }
13
14   // получим ответ из xhr.response
15 };
16
17 xhr.onprogress = function(event) {
18   // выведем прогресс
19   alert( `Загружено ${event.loaded} из ${event.total}` );
20 };
21
22 xhr.onerror = function() {
23   // обработаем ошибку, не связанную с HTTP (например,
24   // таймаут)
```

Событий на самом деле больше, в [современной спецификации](#) они все перечислены в том порядке, в каком генерируются во время запроса:

- `loadstart` – начало запроса.
- `progress` – прибыла часть данных ответа, тело ответа полностью на данный момент можно получить из свойства `responseText`.
- `abort` – запрос был прерван вызовом `xhr.abort()`.
- `error` – произошла ошибка соединения, например неправильное доменное имя. Событие не генерируется для HTTP-ошибок как, например, 404.
- `load` – запрос успешно завершён.
- `timeout` – запрос был отменён по причине истечения отведённого для него времени (происходит, только если был установлен таймаут).
- `loadend` – срабатывает после `load`, `error`, `timeout` или `abort`.

События `error`, `abort`, `timeout` и `load` взаимно исключают друг друга – может произойти только одно из них.

Наиболее часто используют события завершения загрузки ( `load` ), ошибки загрузки ( `error` ), или мы можем использовать единый обработчик `loadend` для всего и смотреть в свойствах объекта запроса `xhr` детали произошедшего.

Также мы уже видели событие: `readystatechange`. Исторически оно появилось одним из первых, даже раньше, чем была составлена спецификация. Сегодня нет необходимости использовать его, так как оно может быть заменено современными событиями, но на него можно часто наткнуться в старом коде.

Если же нам нужно следить именно за процессом отправки данных на сервер, тогда можно использовать те же события, но для объекта

xhr.upload.

Раздел

[Сетевые запросы](#)

Навигация по уроку

Основы

Тип ответа

Состояния запроса

Отмена запроса

Синхронные запросы

HTTP-заголовки

POST, FormData

Прогресс отправки

Запросы на другой источник

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



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



## Комментарии

перед тем как писать...

© 2007—2020 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфи](#)

