

AJAX

ASYNCHRONOUS JAVASCRIPT AND XML



Что такое AJAX?

AJAX – технология обращения к серверу без перезагрузки страницы.

Что я могу сделать с помощью AJAX?

- Взаимодействие с элементами интерфейса
- Динамическая подгрузка данных
- Живой поиск

AJAX

```
var xhr = new XMLHttpRequest();  
xhr.open(method, URL, async, user?, password?);  
xhr.send(body?);  
//xhr.abort(); Прерывает выполнение запроса
```

AJAX

```
xhr.open(method, URL, async?, user?, password?);
```

method – HTTP-метод. Как правило, используется GET либо POST, хотя доступны и более экзотические, вроде DELETE/PUT и т.п.

URL – адрес запроса.

async – если установлено в false, то запрос производится синхронно, если true – асинхронно.

«Синхронный запрос» означает, что после вызова `xhr.send()` и до ответа сервера главный поток будет «заморожен»: посетитель не сможет взаимодействовать со страницей – прокручивать, нажимать на кнопки и т.п. После получения ответа выполнение продолжится со следующей строки.

«Асинхронный запрос» означает, что браузер отправит запрос, а далее результат нужно будет получить через обработчики событий, которые мы рассмотрим далее.

user, password – логин и пароль для HTTP-авторизации, если нужны.

AJAX

```
xhr.send(body?);
```

Именно этот метод открывает соединение и отправляет запрос на сервер.

В `body` находится *тело* запроса. Не у всякого запроса есть тело, например у GET-запросов тела нет, а у POST – основные данные как раз передаются через `body`.

Ответ: `status`, `statusText`, `responseText`

`status`

HTTP-код ответа: 200, 404, 403 и так далее. Может быть также равен 0, если сервер не ответил или при запросе на другой домен.

`statusText`

Текстовое описание статуса от сервера: *OK*, *Not Found*, *Forbidden* и так далее.

`responseText`

Текст ответа сервера.

Есть и ещё одно свойство, которое используется гораздо реже:

`responseXML`

Если сервер вернул XML, снабдив его правильным заголовком `Content-type: text/xml`, то браузер создаст из него XML-документ. По нему можно будет делать запросы `xhr.responseXML.querySelector("...")` и другие. Оно используется редко, так как обычно используют не XML, а JSON. То есть, сервер возвращает JSON в виде текста, который браузер превращает в объект вызовом `JSON.parse(xhr.responseText)`.

AJAX

```
"{"id":21262,"group":1610,"name":"Seitmagambet Olzhas"}"
```

```
JSON.parse(xhr.responseText);
```

```
{  
  id: 21262,  
  group: 1610,  
  name: "Seitmagambet Olzhas",  
}
```


Событие `readystatechange`

Событие **`readystatechange`** происходит несколько раз в процессе отсылки и получения ответа. При этом можно посмотреть «текущее состояние запроса» в свойстве **`xhr.readyState`**.

- 0 // начальное состояние
- 1 // вызван `open`
- 2 // получены заголовки
- 3 // загружается тело (получен очередной пакет данных)
- 4 // запрос завершён

Запрос проходит их в порядке $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow 3 \rightarrow 4$, состояние 3 повторяется при каждом получении очередного пакета данных по сети

AJAX

```
var xhr = new XMLHttpRequest();
xhr.open(method, URL, async, user, password);
xhr.send(body);

xhr.onreadystatechange = function() {
    if (xhr.readyState !== 4) return;

    if (xhr.status !== 200) {
        console.log(xhr.status + ': ' + xhr.statusText);
    } else {
        console.log(JSON.parse(xhr.responseText));
    }
};
```

Полный список событий

Современная спецификация предусматривает следующие события по ходу обработки запроса:

`loadstart` – запрос начат.

`progress` – браузер получил очередной пакет данных, можно прочитать текущие полученные данные в `responseText`.

`abort` – запрос был отменён вызовом `xhr.abort()`.

`error` – произошла ошибка.

`load` – запрос был успешно (без ошибок) завершён.

`timeout` – запрос был прекращён по таймауту.

`loadend` – запрос был завершён (успешно или неуспешно)

Используя эти события можно более удобно отслеживать загрузку (`onload`) и ошибку (`onerror`), а также количество загруженных данных (`onprogress`).



Метод `fetch`: замена `XMLHttpRequest`

Метод **`fetch`** – это `XMLHttpRequest` нового поколения. Он предоставляет улучшенный интерфейс для осуществления запросов к серверу: как по части возможностей и контроля над происходящим, так и по синтаксису, так как построен на промисах.

Синтаксис

Синтаксис метода `fetch`:

```
var promise = fetch(url, options?);
```

`url` – URL, на который сделать запрос,

`options` – необязательный объект с настройками запроса.

Свойства `options`:

`method` – метод запроса,

`headers` – заголовки запроса (объект),

`body` – тело запроса: `FormData`, `Blob`, строка и т.п.

...

AJAX

Использование

При вызове `fetch` возвращает промис, который, когда получен ответ, выполняет коллбэки с объектом `Response` или с ошибкой, если запрос не удался.

Пример использования:

```
fetch( '/offers/1' )
  .then( function( response ) {
    return response.json();
  })
  .then( function( offer ) {
    console.log(offer.offer_title); // Акция 1
  })
  .catch( console.log );
```

AJAX

+ fetch (современная замена XMLHttpRequest)

 **Сторонние библиотеки**

+ jQuery AJAX

+ axios

AXIOS

```
axios({  
  method: 'post',  
  url: '/offers/create',  
  data: {  
    offer_title: "AJAX бесплатно! без регистрации!",  
    offer_description: "Изучи AJAX и делай запросы без перезагрузки»,  
    offer_price: 100500  
  }  
});
```

HTTP Method Aliases

axios.post(url[, data[, config]])

axios.get(url[, config])

...



Результат запроса

```
{  
  // `data` is the response that was provided by the server  
  data: {},  
  
  // `status` is the HTTP status code from the server response  
  status: 200,  
  
  // `statusText` is the HTTP status message from the server response  
  statusText: 'OK',  
  
  // `headers` the headers that the server responded with  
  // All header names are lower cased  
  headers: {},  
  
  // `config` is the config that was provided to `axios` for the request  
  config: {},  
  
  // `request` is the request that generated this response  
  // It is the last ClientRequest instance in node.js (in redirects)  
  // and an XMLHttpRequest instance the browser  
  request: {}  
}
```


AXIOS

```
axios.get('/offers/1')  
  .then(response => {  
    console.log(response.data);  
  });
```

```
/*
```

```
...
```

```
*/
```

```
var offer = {  
  title: "AJAX бесплатно! без регистрации!",  
  description: "Изучи AJAX и делай запросы без перезагрузки",  
  price: 100500  
};
```

```
axios.post('/offers/create', offer)  
  .then(function(response) {  
    console.log(response.data);  
  });
```

AJAX

Материалы

Документация

<https://learn.javascript.ru/ajax>

<https://github.com/axios/axios>

<https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest>

Код

<https://github.com/winexy/laravel-ajax>