Сетевые запросы

**Сетевой запрос** – это процесс общения клиентской стороны (страницы в браузере) с каким-либо сервером. Общение бывает двух типов:

**GET запрос** – это запрос, который получает данные с сервера, например пользователь получает сообщение.

**POST запрос** – это запрос, который отправляет данные с сервера, например авторизация пользователя.

**Протокол HTTP** – это протокол, позволяющий получить какие-либо данные, т.е. совершить запрос. Он может работать вне зависимости от типа данных.

По http у каждого запроса есть **заголовки** – это дополнительные данные, например hash сайта, естественный язык и т.д. Делятся на *общие* (general headers), *post* (response headers) и *get* (request headers).

В Native JS реализованы сетевые запросы через класс XMLHttpRequest (короче Xhr) и функцию fetch. Первый многословный и не представляет удобных методов для преобразования, а также он не промисицированный, т.е. не работает, как класс Promise.

Второй более удобен во всех планах, и он возвращает Promise, что намного удобнее. Оба способа, как понятно асинхронны.

В примерах ниже будет работа с сервисом [jsonplaceholder](https://jsonplaceholder.typicode.com/), который будет возвращать рандомные ответы.

Xhr

const URL = 'https://jsonplaceholder.typicode.com/users';

let xhr = new XMLHttpRequest();

xhr.open ('GET', URL);

xhr.send ();

Создаем объект xhr с классом XMLHttpRequest. Далее вызываем у него метод open(), который создает запрос, а метод send отправляет запрос на сервер. В метод open передается тип запроса и URL к странице серверу, с которым будем работать.

В браузере во вкладке Network в окне XHR и Fetch можно увидеть отправленный запрос.

* headers – содержит заголовки
* response – то что получили
* preview - результат для «глаз»
* timing – справка о времени.

Для получения результата используется свойство response, но так как запрос асинхронный, то надо использовать обработчик onload.

Свойство response по-умолчанию вернет строку, но в свойство responseType можно указать тип, который мы хотим.

Также есть обработчик onerror, который обрабатывает ошибки. Но для того чтоб обрабатывать ошибки в ответе, т.е. статус используется onload Обновленный пример:

const URL = 'https://jsonplaceholder.typicode.com/users';

let xhr = new XMLHttpRequest();

xhr.open ('GET', URL);

xhr.responseType = 'json';

xhr.onload = () => {

if (xhr.status >= 400) {

console.error (xhr.response);

} else {

console.log (xhr.response);

}

}

xhr.onerror = () => {

console.log ('error', xhr);

}

xhr.send ();

Полученный код следует вынести в отдельную функцию и добавить ради удобства промисы:

function sendRequest(method, url) {

return new Promise((resolve, reject) => {

let xhr = new XMLHttpRequest();

xhr.open (method, url);

xhr.responseType = 'json';

xhr.onload = () => {

if (xhr.status >= 400) {

reject (xhr.response);

} else {

resolve (xhr.response);

}

}

xhr.onerror = () => {

reject (xhr.response);

}

xhr.send ();

});

}

Работаем с функцией:

const URL = 'https://jsonplaceholder.typicode.com/users';

sendRequest ('GET', URL)

.then (value => {

console.log (value);

})

.catch (err => {

console.error(err);

});

Для работы с POST нужно еще **body** – это то, что передаем серверу. Сам body передается в метод send, притом если запрос GET, то метод не обратит внимания на body. Стоит сказать, что body должна быть строка в любой случаи.

function sendRequest(method, url, body = null) {

return new Promise((resolve, reject) => {

let xhr = new XMLHttpRequest();

xhr.open (method, url);

xhr.responseType = 'json';

xhr.onload = () => {

if (xhr.status >= 400) {

reject (xhr.response);

} else {

resolve (xhr.response);

}

}

xhr.onerror = () => {

reject (xhr.response);

}

xhr.send (JSON.stringify (body));

});

}

Воспользуемся функцией:

const body = {

name: 'alex',

old: 18

}

const URL = 'https://jsonplaceholder.typicode.com/users';

sendRequest('POST', URL, body)

.then (value => {

console.log (value);

})

.catch (err => {

console.error(err);

});

Нам вернется объект, в котором будет указан id нового пользователя. Также в Network поле Statues Code будет 201, т.е. объект создан.

Но на деле сервер обработал body, как строку, именно поэтому мы получает просто объект с id, а не с объектом плюс id. Для этого нужно заголовку Content-Type задать значение application/json.

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

Все! Полученная функция делает запросы на сервер, без библиотек. Она может пригодиться для мелких и простых проектов, которые должны работать в старых браузерах.

Функция fetch

Все-то же самое способна сделать функция fetch. Ее параметры:

let promise = fetch (url, opinion)

Здесь объект promise со значением класс Response. У класса Response есть поле ok, которое означает то, что запрос успешно отправлен, а также поле status код статуса запроса.

Также есть методы для получения результата в нужном нам формате:

* response.json() – возвращает ответ в формате JSON
* response.text() – возвращает ответ в строке
* response.blob() – возвращает ответ объекте Blob
* response.formData() – возвращает ответ объекте FormData
* response.arrayBuffer() – возвращает ответ объекте ArrayBuffer

Пример простого GET запроса:

fetch ('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits')

.then (response => response.json ())

.then (commits => alert (commits [0].author.login));

Можно переписать через async-await:

async function main() {

let url = 'https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits';;

let response = await fetch(url);

let commits = await response.json();

console.log(commits[0].author.login);

}

main();

Свойство response.headers содержит все заголовки запроса, это псевдо Map:

// Получение заголовка по имени

console.log (response.headers.get ('Content-Type'));

// Перебор всех заголовков

for (let [key, value] of response.headers) {

console.log (`${key}: ${value}`);

}

Для того чтоб сделать POST нужно передать объект opinion. В нем нужно в method значение GET, передать body и установить нужный заголовок. Пример отправки:

async function main() {

let url = 'https://jsonplaceholder.typicode.com/users';

let body = {

name: 'Robert',

old: 20

}

let response = await fetch(url, {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(body)

});

if (response.ok) {

return await response.json();

}

}

main()

.then (response => console.log (response));

Для более удобной работы с формами есть класс FormData, упомянутый ранее. Его удобство в том, что fetch позволяет указать его как в body.

let formData = new FormData (*form*);

Его методы:

* delete(name) удаляет элемент со значением name
* get(name) возвращает значение name
* has(name) проверяет существует ли поле с name
* set(name, value) устанавливает значение name на value
* set(name, blob, fileType) устанавливает у input[type="file"] c name значение файл blob с типом fileType

Пример отправки простой формы:

formElem.onsubmit = async(e) => {

e.preventDefault ();

let response = await fetch('https://jsonplaceholder.typicode.com/users', {

method: 'POST',

body: new FormData(formElem)

});

let res = await response.json();

console.log(res);

}