Документация

JS Fetch API

`Fetch API` — интерфейс для работы с HTTP-запросами и ответами, предоставляет возможность взаимодействия с веб-сервисами, ресурсами сети непосредственно из узла `JavaScript`.

Интерфейс `Fetch API` включает в себя следующие объекты:

- Headers — заголовки запроса/ответа;
- Request — запрос ресурса сети;
- Response — ответ на запрос;
- AbortController — управление отменой запроса;
- fetch() — функция, используемая для получения ресурсов сети.

Перед использованием вышеуказанные объекты должны быть импортированы из встроенного модуля `builtIn/Fetch`:

```
import {fetch, Request, Response, Headers, AbortController} from
"builtIn/Fetch";
```

Также доступен объект `AbortSignal`. Он предоставляется через свойство `signal` экземпляра `AbortController` и отдельно не импортируется.

> **Примечание**: у объектов `Request` и `Response` отсутствует часть методов и свойств, стандартных для браузеров. Полный перечень см. в Полное описание API, а также в описании ниже.

Описание объектов

Headers

Представляет итерируемую коллекцию HTTP-заголовков. Методы объекта позволяют получать, устанавливать, добавлять и удалять заголовки из коллекции заголовков запроса.

Конструктор

```
let headers = new Headers([init]);
```

где:

- `init` принимает объект типа `HeadersInit` (см. Полное описание API). Необязательный параметр.

Создает объект `Headers` (см. Полное описание API).

Методы Headers

append

append(name, value)

- name — имя добавляемого заголовка. Принимает значение типа `string`. Обязательный параметр.
- value — значение заголовка. Принимает значение типа `string`. Обязательный параметр.

Добавляет новое значение к существующему заголовку внутри Headers-объекта или добавляет заголовок, если он еще не существует. Метод возвращает `undefined`.

delete

delete(name)

- name — имя удаляемого заголовка. Принимает значение типа `string`. Обязательный параметр.

Удаляет заголовок из текущего Headers-объекта. Метод возвращает `undefined`.

entries

entries()

Метод возвращает итерируемую коллекцию пар имя/значение заголовков, содержащихся в Headers-объекте. Имя и значение являются строками.

forEach

forEach(callbackfn(value, key, parent)[, thisArg])

- callbackfn — функция, применяемая к каждому заголовку. Обязательный параметр. В `callbackfn` в указанном порядке передаются параметры:
  - value — значение заголовка. Принимает значение типа `string`.
  - key — имя заголовка. Принимает значение типа `string`.
  - parent — Headers-объект.
- thisArg — значение, используемое как `this` при вызове `callbackfn`. Необязательный параметр.

Метод перебирает коллекцию заголовков в Headers-объекте и выполняет предоставленную функцию для каждого заголовка.

get

get(name)

- name — имя заголовка. Принимает значение типа `string`. Обязательный параметр.

Возвращает строку, представляющую значение заголовка или `null`, если этот заголовок не установлен.

set

set(name, value)

- name — имя заголовка. Принимает значение типа `string`. Обязательный параметр.
- value — значение заголовка. Принимает значение типа `string`. Обязательный параметр.

Устанавливает новое значение для существующего заголовка внутри Headers-объекта или добавляет заголовок, если он еще не существует. Метод возвращает `undefined`.

has

has(name)

- name — имя заголовка. Принимает значение типа `string`. Обязательный параметр.

Возвращает `true` или `false` в зависимости от того, содержит ли Headers-объект заголовок с указанным именем.

keys

keys()

Возвращает итерируемую коллекцию имен заголовков Headers-объекта.

values

values()

Возвращает итерируемую коллекцию значений заголовков Headers-объекта.

Request

Представляет собой HTTP-запрос.

Конструктор

```
let request = new Request(input[, init]);
```

где:

- `input` — объект типа `RequestInfo` (см. Полное описание API). Обязательный параметр. Принимает URL-адрес запрашиваемого ресурса или объект, реализующий интерфейс `Request`.
- `init` — объект, реализующий интерфейс `RequestInit` (см. Полное описание API). Необязательный параметр. Принимает параметры HTTP-запроса. `init`-объект может содержать следующие параметры:
  - `body` — тело HTTP-запроса. Объект типа `BodyInit` (см. Полное описание API). `body` может быть строкой или объектом типов `ArrayBuffer`, `ArrayBufferView`.
  - `headers` — заголовки HTTP-запроса. headers-объект (см. также Полное описание API).
  - `method` — строка, содержащая метод HTTP-запроса (`get`, `post` и т.д.).
  - `redirect` — строка, содержащая режим обработки перенаправлений (`follow`, `error`, `manual`).
  - `signal` — объект `AbortSignal`, используемый для досрочной отмены запроса.

Создает объект `Request` (см. Полное описание API).

Особенности реализации

Разрешена установка HTTP-заголовков `Cookie` и `Cookie2`.

Свойства Request

▼ bodyUsed

bodyUsed

Содержит логическое значение, указывающее, было ли считано тело запроса (тело запроса может быть считано только один раз). Возвращает `true` или `false`. Доступно только для чтения.

▼ headers

headers

Содержит HTTP-заголовки запроса (объект `Headers`). Доступно только для чтения.

▼ method

method

Содержит метод запроса (GET, POST и т.д.). Возвращает значение типа `string`. Доступно только для чтения.

redirect

redirect

Содержит режим обработки перенаправлений. Возвращает одно из значений типа `string`:

- `follow`
- `error`
- `manual`

Если свойство не указано при создании запроса, принимает значение по умолчанию `follow`. Доступно только для чтения.

signal

signal

Содержит объект `AbortSignal`, связанный с запросом. Доступно только для чтения.

url

url

Содержит URL-адрес запроса. Возвращает значение типа `string`. Доступно только для чтения.

Методы Request

arrayBuffer

arrayBuffer()

Считывает тело запроса и возвращает promise значения типа `ArrayBuffer`.

json

json()

Возвращает promise с объектом, полученным в результате разбора тела запроса как текста в формате JSON.

text

text()

> Возвращает promise со строкой, полученной в результате интерпретации тела запроса как текста в кодировке UTF-8.

▼ clone

> **clone()**
>
> Создает копию текущего Request-объекта.

Response

Представляет собой ответ на HTTP-запрос.

Конструктор

Можно создать новый Response-объект с помощью конструктора, но на практике скорее всего встретится объект, возвращаемый функцией fetch().

```
let response = new Response([body][, init]);
```

где:

- `body` принимает объект типа `BodyInit` (см. Полное описание API) или `null`. Необязательный параметр.
- `init` принимает объект, реализующий интерфейс `ResponseInit` (см. Полное описание API). Необязательный параметр.

Создает объект `Response` (см. Полное описание API).

Особенности реализации

Разрешены заголовки `Set-Cookie` и `Set-Cookie2` для использования совместно с заголовками `Cookie` и `Cookie2` запроса.

Свойства Response

▼ bodyUsed

> **bodyUsed**
>
> Содержит логическое значение, указывающее, было ли считано тело ответа (тело ответа может быть считано только один раз). Возвращает `true` или `false`. Доступно только для чтения.

▼ headers

headers

Содержит HTTP-заголовки ответа (объект `Headers`). Доступно только для чтения.

ok

ok

Содержит логическое значение, указывающее, был ли ответ успешным (статус в диапазоне 200–299) или нет. Возвращает `true` или `false`. Доступно только для чтения.

redirected

redirected

Содержит логическое значение, указывающее, является ли ответ результатом перенаправленного запроса. Возвращает `true` или `false`. Доступно только для чтения.

status

status

Содержит код состояния HTTP-ответа. Возвращает значение типа `number`. Доступно только для чтения.

statusText

statusText

Содержит сообщение, соответствующее коду состояния HTTP-ответа. Например, `OK` соответствует коду состояния 200, `Continue` — 100, `Not Found` — 404. Возвращает значение типа `string`. Доступно только для чтения.

url

url

Содержит URL-адрес ответа. Значение `url` свойства будет конечным URL-адресом, полученным после любых перенаправлений. Возвращает значение типа `string`. Доступно только для чтения.

Методы Response

arrayBuffer

arrayBuffer()

Считывает тело ответа и возвращает promise значения типа `ArrayBuffer`.

▼ json

json()

Возвращает promise с объектом, полученным в результате разбора тела ответа как текста в формате JSON.

▼ text

text()

Возвращает promise со строкой, полученной в результате интерпретации тела запроса как текста в кодировке UTF-8.

▼ clone

clone()

Создает копию текущего Response-объекта.

AbortController

Предназначен для досрочной отмены HTTP-запроса, выполняемого функцией `fetch()`.

Конструктор

```
let controller = new AbortController();
```

Создает объект `AbortController` и связанный с ним сигнал отмены.

Свойства AbortController

▼ signal

signal

Содержит объект `AbortSignal`, который передается в параметры запроса. Доступно только для чтения.

Методы AbortController

▼ abort

abort([reason])

- reason — произвольное значение, содержащее причину отмены. Необязательный параметр.

Переводит связанный объект `AbortSignal` в состояние отмены и отменяет операции, которым был передан этот сигнал.

Если параметр `reason` не указан, причиной отмены становится объект ошибки со следующими свойствами:

```
{
    name: "AbortError",
    message: "signal is aborted without reason"
}
```

Повторный вызов метода не изменяет состояние и причину отмены и не вызывает событие `abort` повторно.

AbortSignal

Содержит состояние и причину отмены запроса. Объект `AbortSignal` получают через свойство `signal` экземпляра `AbortController`.

Свойства AbortSignal

▼ aborted

aborted

Содержит логическое значение, указывающее, был ли сигнал отменен. До отмены возвращает `false`, после отмены — `true`. Доступно только для чтения.

▼ reason

reason

Содержит причину отмены, переданную методу `AbortController.abort()`. До отмены возвращает `undefined`. Доступно только для чтения.

▼ onabort

onabort

Содержит функцию-обработчик события `abort`. При возникновении события обработчику передается объект со свойствами:

- `type` — строка `"abort"`;
- `reason` — причина отмены.

В качестве `this` при вызове обработчика используется объект `AbortSignal`.

Методы AbortSignal

▼ addEventListener

addEventListener(type, listener)

- type — тип события. Принимает значение `"abort"`. Обязательный параметр.
- listener — функция-обработчик события. Обязательный параметр.

Добавляет обработчик события `abort`. Обработчику передается объект события со свойствами `type` и `reason`. В качестве `this` используется объект `AbortSignal`.

▼ removeEventListener

removeEventListener(type, listener)

- type — тип события. Принимает значение `"abort"`. Обязательный параметр.
- listener — ранее добавленная функция-обработчик. Обязательный параметр.

Удаляет обработчик события `abort`.

fetch()

`fetch(resource[, init])`, где

- resource — принимает объект, реализующий интерфейс `Request`, или строку, содержащую URL запроса. Обязательный параметр.
- init — принимает объект, реализующий интерфейс `RequestInit` (см. Полное описание API). Необязательный параметр.

Асинхронная функция `fetch` запускает процесс извлечения ресурса из сети, возвращая promise объекта `Response` (см. Полное описание API).

В параметре `init.signal` можно передать объект `AbortSignal`. Если сигнал уже отменен или отменяется во время выполнения запроса, promise функции `fetch()` переходит в отклоненное состояние с причиной, содержащейся в свойстве `signal.reason`.

Функция `fetch()` не ограничивает ожидание ответа собственным фиксированным тайм-аутом. Для ограничения времени запроса используется `AbortController` совместно с функцией setTimeout() (см. пример Ограничение времени выполнения запроса).

Примеры

Использование Fetch API

```
import {fetch, Request, Headers} from "builtIn/Fetch";

// Создание объекта заголовков запроса:
let headers = new Headers({"Content-Type": "text/html", "Custom-Header":
"delete me"})
// Вывод значения заголовка
console.log("Custom-Header: ", headers.get("Custom-Header"))
// Удаление заголовка
headers.delete("Custom-Header")
// Проверка существования заголовка
console.log(headers.has('Custom-Header'));
// Добавление нового заголовка
headers.append("Accept-Charset", "utf-8")
// Изменение значения заголовка
headers.set("Content-Type", "application/json")

// Создание объекта запроса:
let request = new Request("http://httpbin.org/post", {
    method: "post",
    headers: headers,
    body: "{ \"str\": message }",
    redirect: "follow"
});

// Вывод параметров запроса:
console.log("url: " + request.url);
console.log("bodyUsed: " + request.bodyUsed);
console.log("redirect: " + request.redirect);
console.log("method: " + request.method);
for (let header of headers.entries()) {
    console.log(header[0]+ ': '+ header[1]);
}

// Вызов сервиса httpbin.org и вывод параметров ответа:
fetch(request)
    .then(response => {
        console.log("ok: " + response.ok);
```

```
        console.log("status: " + response.status);
        console.log("statusText: " + response.statusText);
        console.log("redirected: " + response.redirected);
        console.log("url: " + response.url);
        headers.forEach(function(value, key) {
            console.log("Name : ", key, " Value : ", value)
        });
        response.text().then(text => console.log(text));
    }).catch(e => console.log(e));
```

Получение курсов валют ЦБ РФ

```
import {fetch} from "builtIn/Fetch";

// Запрос сервиса ЦБ:
fetch("https://www.cbr-xml-daily.ru/daily_json.js")
    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status},
                            statusText: ${response.statusText}`);
        }
        return response.json();
    })
    .then(json => console.log(JSON.stringify(json)))
    .catch(e => console.log(e));
```

Последовательное выполнение запросов

```
import {fetch, Headers} from "builtIn/Fetch";

(async() => {
    try {
        // Запрос 1-ого сервиса
        let response1 = await
fetch('https://jsonplaceholder.typicode.com/posts', {
            method: 'POST',
            body: JSON.stringify({
                title: "foo",
                body: "bar",
                userId: 1,
            }),
            headers: new Headers({"Content-type": "application/json;
charset=UTF-8"}),
        });
        let payload = await response1.arrayBuffer();

        // Запрос 2-ого сервиса
        let response2 = await fetch('http://httpbin.org/post', {
            method: 'POST',
            headers: new Headers({"Content-Type": "application/json",
                                  "Accept-Charset": "utf-8"}),
            body: payload,
        });
        console.log(await response2.text())
    } catch(e) {
        console.log(e);
    }
})()
```

Ограничение времени выполнения запроса

В следующем примере запрос к сервису с задержкой ответа отменяется через три секунды:

```
import {fetch, AbortController} from "builtIn/Fetch";

(async() => {
    let controller = new AbortController();

    // По истечении трех секунд переводим сигнал в состояние отмены.
    let timeoutId = setTimeout(function() {
        controller.abort();
    }, 3000);

    try {
        let response = await fetch("https://httpbin.org/delay/10", {
            signal: controller.signal
        });

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status},
                             statusText: ${response.statusText}`);
        }

        console.log(await response.text());
    } catch(error) {
        // Проверяем, что ошибка вызвана отменой именно этого запроса.
        if (error && error.name === "AbortError") {
            console.log("Превышено допустимое время выполнения запроса");
        } else {
            throw error;
        }
    } finally {
        // Если запрос завершился раньше, запланированная отмена не нужна.
        clearTimeout(timeoutId);
    }
})()
```