Сетевые запросы

- Fetch API
- new Promise

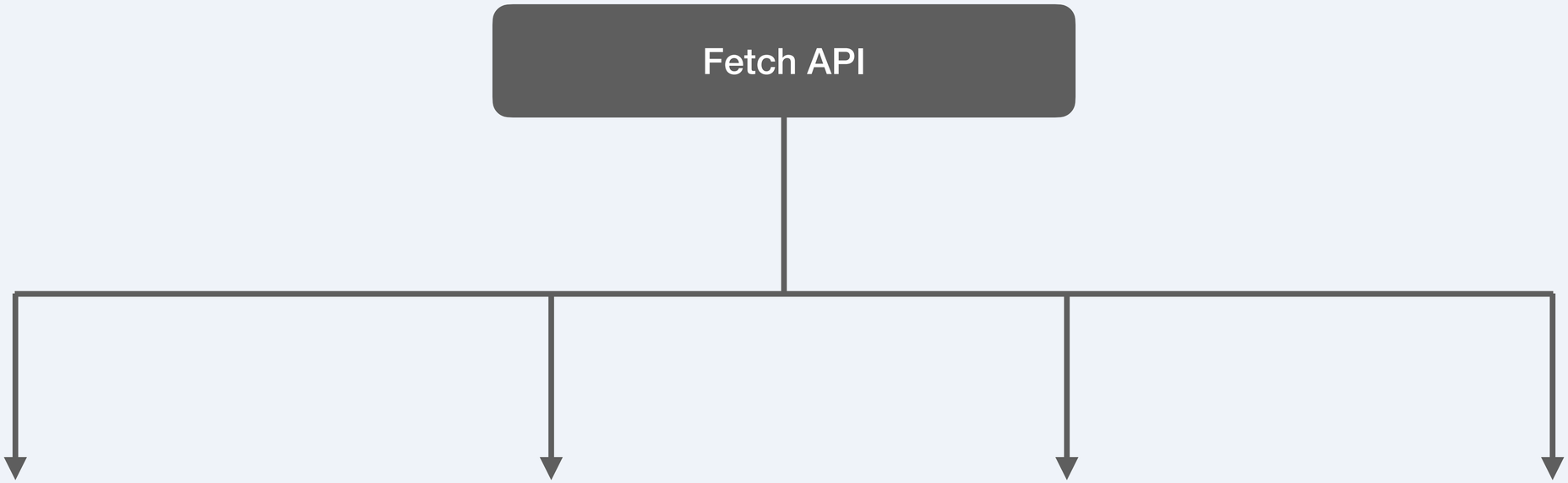
< Teach
Me
Skills / >

Урок

Fetch API

Fetch API предоставляет нам набор функций и объектов, которые помогают нам гибко и удобно делать запросы на сервер и работать с полученным ответом.

Ранее использовался XMLHttpRequest, но на его замену пришел fetch, который представляет собой лучшую альтернативу.



`fetch()`

- Метод который позволяет выбрать необходимый ресурс

`Headers`

- Предоставляет заголовки ответа / запроса.

`Request`

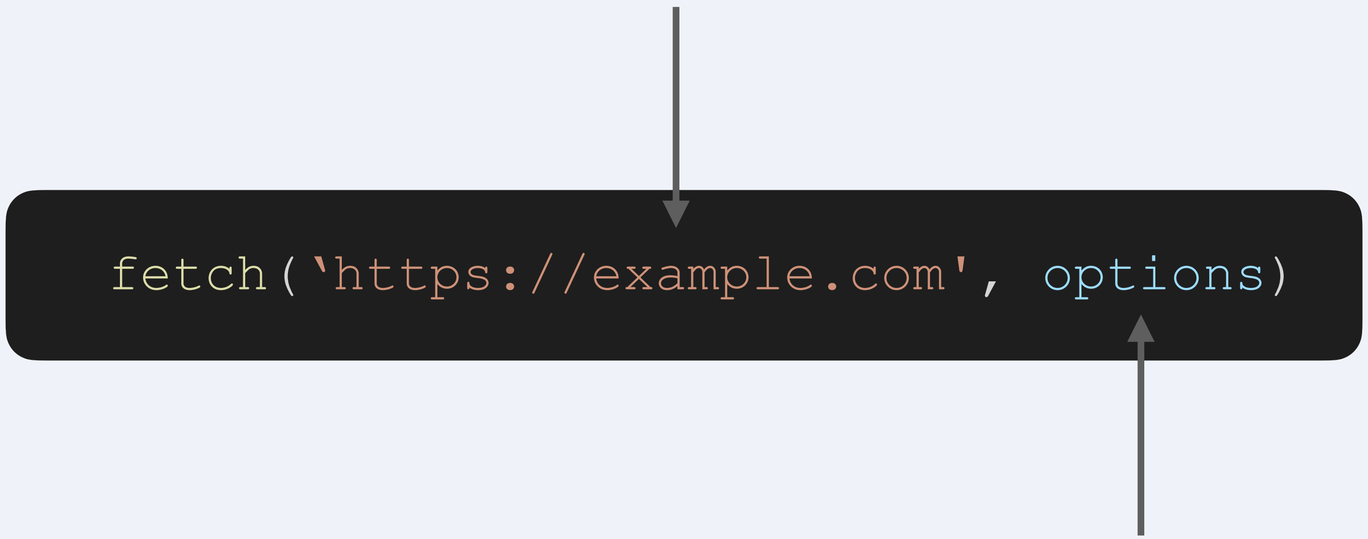
- Предоставляет запрос ресурса

`Response`

- Предоставляет ответ на запрос

Метод Fetch

URL для отправки запроса



Дополнительные параметры

< Teach
Me
Skills / >

Рассмотрим простой пример запроса

Метод then выполняет функцию после завершения вызова функции fetch

Передаем URL запроса

```
fetch('https://jsonplaceholder.typicode.com/todos')
  .then(response => response.json())
  .then(response => . . .)
```

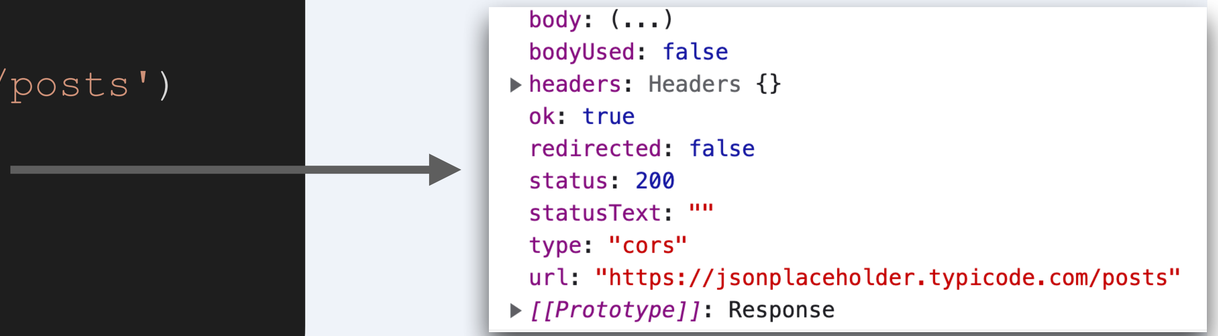
Метод fetch возвращает объект response

Объект ответа содержит методы для доступа к полученным данным в различных форматах

Объект Response

Объект Response представляет собой ответ на запрос.

```
fetch('https://jsonplaceholder.typicode.com/posts')
    .then(response => {
        console.log(response);
})
```



Тело ответа (body) имеет различные методы - например:

- `json()` - **преобразует ответ в javascript объект**
- `text()` - **преобразует ответ в текст**

Объект Request

Request представляет собой конструктор запроса.

```
const URL = 'https://616e893c715a630017b39xxx.mockapi.io/users'
const options = {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({name: 'John'}),
}

const request = new Request(URL, options)

fetch(request)
    .then(response => response.json())
    .then(data => console.log(data))
```

Принимает два параметра
URL по которому необходимо сделать запрос и параметры отправки запроса

Promise

Объект Promise служит удобной оберткой для обслуживания асинхронного функционала.

Основное назначение Promise создавать цепочки вызовов асинхронных функций.

Синтаксис создания Promise

Создание объекта Promise

```
let examplePromise = new Promise ( ( resolve, reject ) => {} )
```

Promise принимает в себя функцию executor с двумя аргументами

Executor описывает выполнение какой-то асинхронной работы, по завершении которой необходимо вызвать функцию `resolve` или `reject`. Обратите внимание, что возвращаемое значение функции `executor` игнорируется

Синтаксис создания Promise

Первый аргумент (`resolve`) вызывает успешное исполнение promise, второй (`reject`) отклоняет его.

Возвращаем результат успешной работы

Возвращаем результат ошибки

```
let examplePromise = new Promise(( resolve, reject ) => {

    // ...

    resolve('Success')

    reject(new Error('error'))

})
```

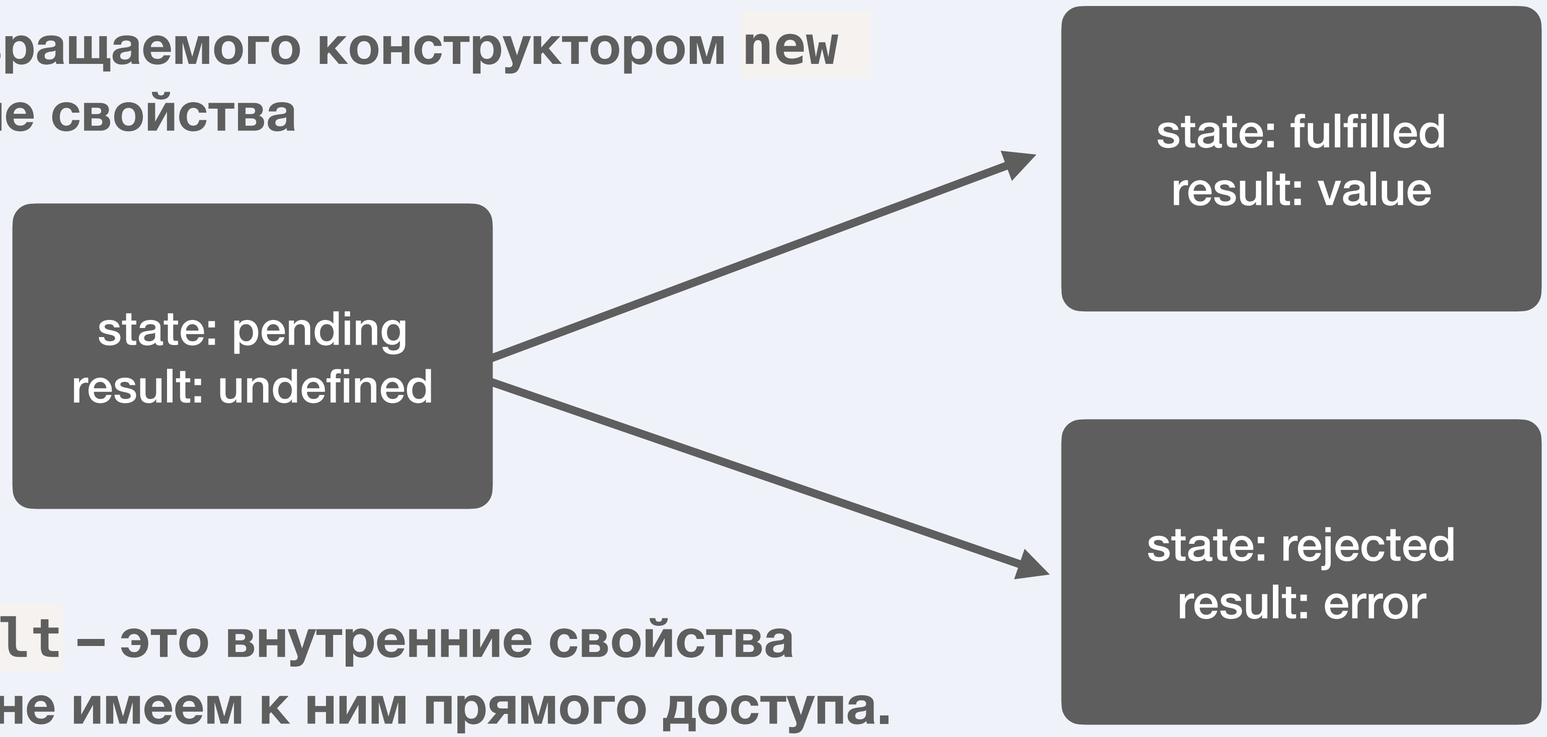
Executor запускается автоматически, он должен выполнить работу, а затем вызвать resolve или reject

Важно знать !

- Executor function должен вызвать что-то одно: `resolve` или `reject`. Состояние promise может быть изменено только один раз.
- Чтобы снабдить функцию функционалом promise, нужно просто вернуть в ней объект `Promise`
- Вызывайте `reject` с объектом `Error`
- Обычно executor function делает что-то асинхронное и после этого вызывает `resolve/reject`, то есть через какое-то время. Но это не обязательно, `resolve` или `reject` могут быть вызваны сразу

Promise state

У объекта `promise`, возвращаемого конструктором `new Promise`, есть внутренние свойства



- **Свойства `state` и `result` – это внутренние свойства объекта `Promise` и мы не имеем к ним прямого доступа. Для обработки результата следует использовать методы `.then/.catch/.finally` с которым познакомимся чуть позже**

Fetch & new Promise

```
const URL = 'https://jsonplaceholder.typicode.com/comments'

function getComments(){
    return new Promise((resolve, reject) => {
        fetch(URL).then(response => {
            if(response.ok){
                resolve(response.json())
            } else {
                reject(new Error('Some Error here 🦉'))
            }
        })
    })
}

getComments().then(comments => {
    console.log(comments);
})
```