Асинхронный JS
Promises и сетевые запросы

Кирилл Талецкий

TeachMeSkills
11 сентября 2023

Callback Hell

Callback Hell 👹

Последовательно загружаем картинки

- Пробрасывать колбэк на успешное выполнение/ошибку бывает не очень удобно
- Это особенно неудобно, когда нам нужно выполнить последовательную цепочку вызовов

```
img.addEventListener('load', () => {
  const img = new Image();
  img.src = 'http://netgeek.biz/funnycat1.jpg';
  document.body.append(img);

  img.addEventListener('load', () => {
    const img = new Image();
    img.src = 'http://netgeek.biz/funnycat2.jpg';
    document.body.append(img);

    img.addEventListener('load', () => {
      const img = new Image();
      img.src = 'http://netgeek.biz/funnycat3.jpg';
      document.body.append(img);

      img.addEventListener('load', () => {
        const img = new Image();
        img.src = 'http://netgeek.biz/funnycat4.jpg';
        document.body.append(img);


      });
    });
  });
});
```

Promise

Promise

Аналогия из жизни

- Вы делаете свой проект, и тут у вас возникает проблема
- У вас не получается её решить, и вы пишете мне — преподу
- Я не могу сразу ответить — проблема оказалась трудной и мне нужно время чтобы разобраться
- Я говорю, что вернусь к вам когда решу проблему (даю обещание — Promise)
- Вы продолжаете работать над другими задачами
- В какой-то момент я пишу в чат решение проблемы
- Вы это решение обрабатываете — разбираетесь, добавляете в свой проект
- Вы продолжаете работать над проектом

Promise

Как давать обещания в JS

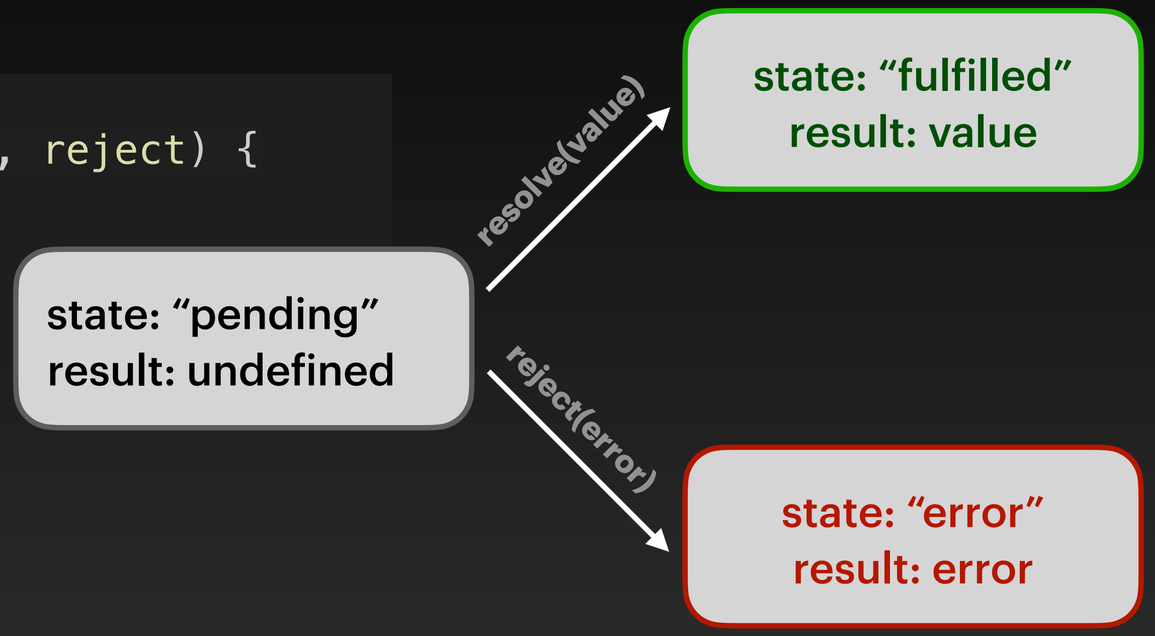
```
const promise = new Promise(function (resolve, reject) {
  // функция-исполнитель (executor)
  // “препод" из примера
});
```

- resolve(value) — если работа завершилась успешно, с результатом value
- reject(error) — если произошла ошибка, error – объект ошибки

Promise

Как давать обещания в JS

```
const promise = new Promise(function (resolve, reject) {
  // функция-исполнитель (executor)
  // “препод" из примера
});
```



- state («состояние») — вначале "pending" («ожидание»), потом меняется на "fulfilled" («выполнено успешно») при вызове resolve или на "rejected" («выполнено с ошибкой») при вызове reject.
- result («результат») — вначале undefined, далее изменяется на value при вызове resolve(value) или на error при вызове reject(error).

Promise
Потребление — then

```
promise.then(
  function(result) { /* обработает успешное выполнение */ },
  function(error) { /* обработает ошибку */ }
);
```

- Первый аргумент — выполнится когда Promise успешно завершится (resolve)
- Второй аргумент — выполнится когда Promise упадёт с ошибкой (reject)

Promise

Потребление — catch и finally

```
promise.catch(
  function(error) { /* обработает ошибку */ }
);
```

```
promise.finally(function () {
  /* "подчищаем" вне зависимости от результата */
});
```

Цепочка промисов

Цепочка промисов

```
let loading = true;

fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(() => fetch('https://jsonplaceholder.typicode.com/todos/2'))
  .then(() => fetch('https://jsonplaceholder.typicode.com/todos/3'))
  .catch(() => {
    console.error('failed to load first part');
  })
  .then(() => fetch('https://jsonplaceholder.typicode.com/todos/4'))
  .then(() => fetch('https://jsonplaceholder.typicode.com/todos/5'))
  .then(() => fetch('https://jsonplaceholder.typicode.com/todos/6'))
  .catch(() => {
    console.error('failed to load second part');
  })
  .finally(() => {
    loading = false;
  });
```

Promise API

Promise.all

- Принимает массив промисов
- Возвращает промис, который выполнится в момент, когда завершатся *все* промисы из массива

```
Promise.all([
  fetch('https://jsonplaceholder.typicode.com/todos/1'),
  fetch('https://jsonplaceholder.typicode.com/todos/2'),
  fetch('https://jsonplaceholder.typicode.com/todos/3'),
]).then(console.log);
```

Promise API

- Promise.allSettled([...promises]) – работает похоже на Promise.all, но возвращает статусы всех промисов по их завершении. Не падает, если один из промисов упал
- Promise.race([...promises]) – ждёт только первый *выполненный* промис
- Promise.any([...promises]) – ждёт только первый *успешно выполненный* промис

- Promise.resolve() – на месте создаёт успешно выполненный промис
- Promise.reject() – на месте создаёт промис завершённый с ошибкой

Async/Await

Async функции

- async — функция всегда возвращает промис

```
async function getData() {
  return 'kek';
}

console.log(getData()); // Promise {<fulfilled>: 'kek'}
```

Await

- await — ключевое слово, которое заставляет дожидаться выполнения промиса

```
// работает только внутри async-функций
const value = await promise;
```

then vs await

```
let loading = true;

const getAll = () => {
  return fetch('https://...')
    .then(() => fetch('https://...'))
    .then(() => fetch('https://...'))
    .catch(() => {
      console.error('failed to load data');
    })
    .finally(() => {
      loading = false;
    });
}
```

```
let loading = true;

const getAll = async () => {
  try {
    await fetch('https://...');
    await fetch('https://...');
    await fetch('https://...');
  } catch {
    console.error('failed to load data');
  } finally {
    loading = false;
  }
}
```

Микрозадачи

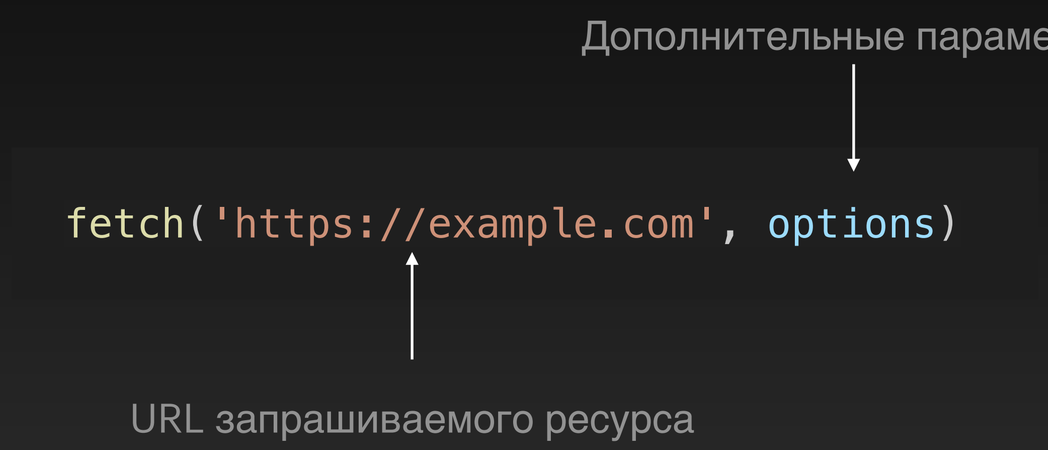
Сетевые запросы
Fetch API

Fetch API

- Предоставляет набор абстракций для удобной работы с сетевыми запросами

fetch()	Headers	Request	Response
Метод который позволяет выбрать необходимый ресурс	Предоставляет заголовки ответа / запроса	Предоставляет запрос ресурса	Предоставляет ответ на запрос

Метод fetch()


Дополнительные параметры
fetch('https://example.com', options)
URL запрашиваемого ресурса

Пример

```
fetch('https://example.com')
  .then(response => console.log(response.json()))
  .then(data => ...)
```

Пример

```
const options = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ key: ‘value’ }),
};

fetch('https://example.com', options)
  .then(response => console.log(response.json()))
  .then(data => ...)
```

Конструкторы Headers() и Request()

```
const URL = 'https://jsonplaceholder.typicode.com/todos/1'

const headers = new Headers();
headers.append('Content-Type', 'application/json');
headers.append('Kek-Header', 'some value');

const options = {
  method: 'GET',
  headers,
}

const request = new Request(URL, options);

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

Объект Response

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(console.log)

/**
 * body: ReadableStream
 * bodyUsed: false
 * headers: Headers {}
 * ok: true
 * redirected: false
 * status: 200
 * statusText: ""
 * type: "cors"
 * url: "https://jsonplaceholder.typicode.com/todos/1"
 */
```