Асинхронный JavaScript

Цели лекции

- ❑ Синхронность/Асинхронность
- ❑ Promise
- ❑ Работа с Promise (fetch)
- ❑ Обработка ошибок Promise
- ❑ Async/Await

Немного истории

Брендан Эйх и его команда, работая над скриптовым языком для браузера Netscape Navigation, не планировали что он прибавит в своей сложности, ведь он был больше нацелен на начинающих программистов и веб-дизайнеров. Тогда никто не думал, что с ходом времени всё станет иначе.

2005 - Google Maps, YouTube

2006 - Google Docs

2011 - Twitch

2013 - Netflix

Синхронность

Код

```
const btn = document.querySelector('button');

btn.addEventListener('click', () => {

  alert('You clicked me!');

  let pElem = document.createElement('p');

  pElem.textContent = 'This is a newly-added paragraph.';

  document.body.appendChild(pElem);

});
```

Проблема

Мы вынуждены ждать завершения работы предыдущих задач. Если функция зависит от результата выполнения другой функции, то она должна дождаться пока нужная ей функция не завершит свою работу и не вернёт результат и до тех пор пока это не произойдёт, выполнение программы, по сути, будет остановлено с точки зрения пользователя.

Пример: тык

Потоки

Что это?

Обычно, понимают одиночный процесс, который может использовать программа, для выполнения своих нужд. Каждый поток может выполнять только одну задачу в текущий момент времени

Task A --> Task B --> Task C

И сколько JS может выполнять потоков?

JavaScript, традиционно для скриптовых языков, однопоточный

Main thread: Render circles to canvas --> Display alert()

Асинхронность

Пример

```
console.log(1);

setTimeout(function(){console.log(2)}, 3000);

console.log(3);

setTimeout(function(){console.log(4)}, 1000);
```

Что расширяет возможности JS?

Web Workers

Он позволяют вам обработать некоторый JavaScript код **в отдельном асинхронном потоке.**В основном, вы будете использовать воркеры, чтобы запустить ресурсоёмкий процесс.

Пример

Main thread: Task A --> Task C

Worker thread: Expensive task B

Promise

Они позволяют запустить некоторую операцию (например, получение картинки с сервера), и затем **подождать** пока операция не вернёт результат, перед тем как начать выполнение другой задачи

Main thread: Task A Task B

Promise: |__async operation__|

Асинхронный callback

Что это?

Это **функция**, которая определяется **как аргумент** при вызове какой-либо функции, чьё выполнение кода начнется на заднем фоне. Когда код на заднем фоне завершает свою работу, он **вызывает колбэк-функцию,** оповещающую, что работа сделана, либо оповещающую о трудностях в завершении работы.

Пример из жизни

Поход магазин со списком продуктов. Вас попросили купить определенные продукты, и если все хорошо, вы их покупаете, если нет - пишите СМС или позвоните чтобы узнать что купить взамен.

Псевдо-код

```
function goShooping(callback) {
 ...
 callback(err, vegetables);
}
goShopping(function(err, vegetables) {
 if (err) throw err;
 switchToCooking(vegetables);
});
```

Проблема

```
loadScipt(“1.js”, function(err, sctipt){
  if (err){
    handleError(err);
  } else {
   loadScipt(“2.js”, function(err, sctipt){
    if (err){
      handleError(err);
    } else {
      loadScipt(“3.js”, function(err, sctipt){
       if (err){
         handleError(err);
       } else {...}
})
```

Promise

Что это?

Это объект, представляющий окончательное завершение или сбой асинхронной операции. По сути, промис — это возвращаемый объект, к которому прикрепляется колбэк, вместо его передачи в функцию (в отличии от асинхронного колбэка).

Пример из жизни

Работа на кассе и на выдаче в McDonald’s.

Псевдо-код

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

    // Делаем, что-то, возможно асинхронное, тогда…

    if (/* Всё прошло отлично */) {

        resolve('Сработало!');

    }

    else {

        reject(Error('Сломалось'));

    }

});
```

Псевдо-код

```
promise.then((result) => {

    console.log('Промис сработал');

},(err) => {

    console.log('Что-то сломалось');

});
```

Состояния

Если промис прошёл успешно, будет выполнен **resolve**, и консоль выведет Промис сработал, в противном случае выведется Что-то сломалось. Это состояние до получения **resolve** или **reject** называется состоянием ожидания, **pending**.

Итого

1. Ожидание ответа: **pending**.
2. Успешное выполнение: **resolve**.
3. Выход ошибкой: **reject**.

Цепочка промисов

———

```
loadImage('1.img')
  .then(response => response.json())
  .then(json => initialize(json))
  .then((res => console.log(res));
```

fetch()

Хорошим примером промиса является fetch() API, который современнее и эффективнее чем XMLHttpRequest.

Пример

```
fetch('products.json')
  .then(response => response.json())
  .then(json => initialize(json));
```

Promise API

Promise.all

Promise.allSettled

Promise.race

Promise.resolve/reject

Обработка ошибок

.catch

Для того, чтобы поставить обработчик на ошибку в промисе нужно написать **.catch(onRejected)**

Пример

```
fetch('products.json')
  .then(response => response.json())
  .then(json => initialize(json))
  .catch(err => showError(err));
```

Async/await

Зачем нужно?

Проще говоря - писать асинхронный код в виде синхронного

Пример

```
async function showAvatar() {

  // запрашиваем JSON с данными пользователя
  let response = await fetch('/article/promise-chaining/user.json');
  let user = await response.json();

  return user;
}

showAvatar()
```

Обработка ошибок Async/Await

```
async function getUser() {

  try {
    let response = await fetch('/no-user-here');
    let user = await response.json();
  } catch(err) {
    // перехватит любую ошибку в блоке try: и в fetch, и в
response.json
    alert(err);
  }
}

getUser();
```

Задача

“Комментарии”

Отобразить всю информацию о комментариях, которые оставили пользователи в сети

Данные можно отобразить в каком угодно виде, на ваше усмотрение (... , <table>...</table>)

Обязательно использовать fetch(), сделать обработку ошибок, использовать API - https://jsonplaceholder.typicode.com/comments

*Сделать возможность фильтровать пользователей по email используя input, использовать async/await

*Рабочий код выложить сюда, сохранить и получить публичную ссылку stackblitz.com (сохранить списком в https)

Материалы для изучения

1. Learn JavaScript Промисы
2. Learn Javascript Цепочка промисов
3. Learn Javascript Fetch
4. Learn JavaScript ASync/Await
5. MDN EventLoop
6. MDN Введение в асинхронный JS
7. MDN Основные понятия
8. MDN Асинхронный JS