**Promises**

1. <https://learn.javascript.ru/promise-basics>
2. [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Promise?retiredLocale=uk](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise?retiredLocale=ukhttps://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise?retiredLocale=uk)
3. <https://stasonmars.ru/javascript/promisy-v-javascript-dlya-chainikov/>
4. <https://stasonmars.ru/javascript/vse-chto-nuzhno-znat-ob-async-await-tsikly-kontrol-potokov-ogranicheniya/>

**Promise** або обіцянку - це об'єкт, який гарантує виконання деякого коду при настанні очікуваної події; в разі помилки промис-об'єкт також викличе обробник для цієї помилки.

**Promise** обіцяє, що та чи інша функція буде виконана в разі успішного наступу події або невдачі.

**Promise** зручні в обробці асинхронного коду. Вони додають трохи синхронності асинхронним функцій і дозволяють позбутися від ланцюжка колбеков (колбек в колбеке, всередині якого колбек

Об’єкт який огортає асинхронний код, завдяки чому можна вистроїти синхронність при декількох запитах, створивши ланцюжок, опрацьовуючи помилки на кожному елементі ланцюга, і на кожному елементі опрацьовуємо вихідні дані попереднього ланцюга, завдяки callbacks, які повертають помилку або результат виклику

**Проміс дозволяють будувати ланцюжки.**

Якщо функція всередині then повертає деяке значення, то наступний в ланцюжку then буде викликаний з цим значенням (з попереднього then)

Якщо функція всередині then повертає інший промис, то наступний в ланцюжку then дочекається виконання нового Проміс, і буде викликаний зі значенням, яке містить новий промис:

then - запускає колбек, який ви передали, коли промис завершено.

catch - запускає колбек, який ви передали, коли щось йде не так, що викликає reject замість resolve

Для **виклику асинхронних методів в циклі** який їх розуміє – використовують **for of**

**Методи:**

Кожен з методів приймає функцію-аргумент.

Функція буде викликана c одним аргументом - значенням, з яким завершився промис

1. **promise**.**then ()** - при успішному завершенні Проміс;

2. **promise.catch ()** - якщо промис завершився невдачею (з помилкою).

3. **promise.finally ()** - цей метод спрацює незалежно від результату Проміс.

1. **Promise.all()** –получає масив (чи інший ітеріруємий тип дпних) промісів і повертає проміс, який чекає, поки всі дані проміса завершаться, і переходить в стан виконано із масивом результатів.

Promise.all([

httpGet('/article/promise/user.json'),

httpGet('/article/promise/guest.json'),

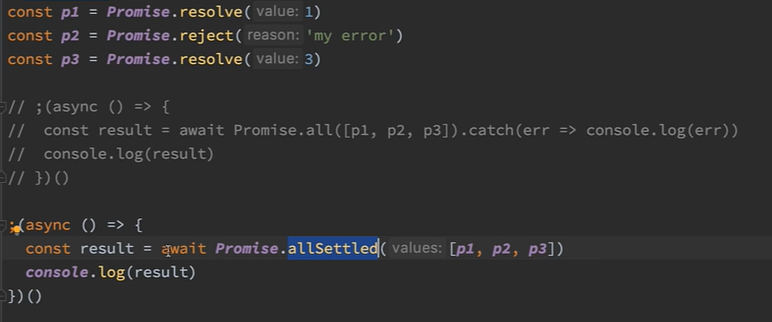
httpGet('/article/promise/no-such-page.json') // (нет такой страницы)

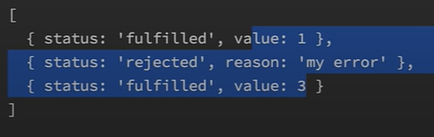
]).then(

result => alert("не сработает"),

error => alert("Ошибка: " + error.message)

1. **Promise**.**allSettled**([prom, prom,…]) – очікує завершення усіх промісів. На відміну від Promise.all – поверне результат навіть при виникнені помилки в якомусь із промісів. Результат – масив об’єктів із полями status (fulfilled || rejected) і value із результатом

****

****

1. **Promise.race**, як і  Promise.all, получає ітеруємий тип даних з промісами, які потрібно виконати, і повертає новий проміс.

Але відмінністю від Promise.all, результатом буде лише перший успіщно виконаний проміс із списка, всі решта будуть проігнорован.

Promise.race([

httpGet('/article/promise/user.json'),

httpGet('/article/promise/guest.json')

]).then(firstResult => {

firstResult = JSON.parse(firstResult);

alert( firstResult.name ); // iliakan или guest, смотря что загрузится раньше

});

3. **Promise.resolve(value)** створює успішно вконаний проміс із рещультатом value. **Promise.resolve** використовують, коли хочуть построїти ланцюг, і початковий результат уже є.

4. **Promise**.**any**([pr, pr, pr…]) – з усього масиву промісів, повертає резкльтат виконання найшвидшого промісу. Якщо серед усіх промісів декілька вибило помилку, то повертається масив помилок

5. **Promise.reject(error)** створює уже виконаний проміс, но не з успішним результатом, а з помилкою.

**New features 2020 - 2021**

Спосіб перебору промісів циклом в новому синтаксисі. Синхронізуємо виконання асинхронного коду в циклі

