PROMISSES

// создадим новый промис

const promise = new Promise((resolve, reject) => {

// оценку, которые мы получим определим случайным образом спустя некоторое время (например, 5 секунд)

setTimeout(() => {

// сгенерируем оценку от 2 до 5

const mark = Math.floor(Math.random() \* 4) + 2;

// если оценка больше 3, то...

if (mark > 3) {

// завершим промис успешно: для этого вызовем функцию resolve() и передадим ей в скобках полученную оценку (это нужно для того, чтобы мы затем её могли получить в методе then())

resolve(mark);

}

// завершим промис с ошибкой

reject(mark);

}, 5000);

});

// выполнение действий после завершения промиса выполняется с помощью методов: then (в случае успеха) и catch (при ошибке)

promise

.then(result => console.log(`Ура! Я сдал экзамен на ${result}! Папа, как и обещал дал мне 100$.`))

.catch(result => console.log(`Увы, я получил оценку ${result}! Папа мне не дал 100$`));

// сдал ли экзамен

const passexam = true;

// промис

const result = new Promise((resolve, reject) => {

setTimeout(() => {

passexam ? resolve('Папа подарил 100$.') : reject('Папа не подарил 100$.');

}, 5000);

});

result

.then(value => {

console.log(result);

console.log(value);

})

.catch(value => {

console.log(result);

console.error(value);

});

<div><button id="run">Новая попытка</button></div>

<div id="result"></div>

<script>

let isProcess = false;

elResult = document.querySelector('#result');

document.querySelector('#run').onclick = () => {

if (isProcess) {

elResult.textContent = 'Подождите! Задача ещё выполняется!';

return;

}

isProcess = true;

elResult.textContent = 'Задача в процессе...';

const promise = new Promise((resolve, reject) => {

setTimeout(() => {

const mark = Math.floor(Math.random() \* 4) + 2;

mark > 3 ? resolve(mark) : reject(mark);

}, 5000);

});

promise

.then(value => {

elResult.textContent = `Ура! Вы сдали экзамен на ${value}! Папа, как и обещал дал вам 100$.`;

})

.catch(value => {

elResult.textContent = `Увы, вы получили оценку ${value}! Папа не дал вам 100$`;

})

.finally(() => {

isProcess = false;

});

}

</script>

Пример цепочки промисов:

const promise = new Promise(resolve => {

setTimeout(() => resolve(2), 3000);

});

promise

.then(value => {

console.log(value);

return new Promise(resolve => {

setTimeout(() => resolve(value \* 2), 3000);

});

})

.then(value => {

console.log(value);

return new Promise(resolve => {

setTimeout(() => resolve(value \* 2), 3000);

});

})

.then(value => console.log(value))

Promice.all() и Promise.race() – это статические методы Promice, которые принимают на вход массив промисов и возвращают новый промис.

В случае с Promice.all() промис завершится когда завершатся все промисы в массиве.

const promise1 = new Promise(resolve => setTimeout(resolve, 1000, 'one'));

const promise2 = new Promise(resolve => setTimeout(resolve, 2000, 'two'));

const promise3 = new Promise(resolve => setTimeout(resolve, 3000, 'three'));

Promise.all([promise1, promise2, promise3]).then(value => console.log(value)); // ['one', 'two', 'three']

При использовании Promise.race() промис завершится, как только выполнится первый промис из массива.

const promise1 = new Promise(resolve => setTimeout(resolve, 1000, 'one'));

const promise2 = new Promise(resolve => setTimeout(resolve, 2000, 'two'));

const promise3 = new Promise(resolve => setTimeout(resolve, 3000, 'three'));

Promise.race([promise1, promise2, promise3]).then(value => console.log(value)); // 'one'

<button type="button" id="get-text">Получить текст с сервера</button>

<div id="result"></div>

<script>

const request = (url, success, error) => {

const xhr = new XMLHttpRequest();

xhr.open('GET', url);

xhr.onload = () => {

xhr.status === 200 ? success(xhr.response) : error(xhr.statusText);

}

xhr.onerror = () => error(xhr.statusText);

xhr.send();

}

document.querySelector('#get-text').onclick = () => {

request('/examples/ajax/01.html', data => {

document.querySelector('#result').innerHTML = data;

}, error => {

console.error(error);

});

}

</script>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<button type="button" id="get-text">Получить текст с сервера</button>

<div id="result"></div>

<script>

const request = (url) => {

return new Promise((resolve, reject) => {

const xhr = new XMLHttpRequest();

xhr.open('GET', url);

xhr.onload = () => {

xhr.status === 200 ? resolve(xhr.response) : reject(xhr.statusText);

}

xhr.onerror = () => reject(xhr.statusText);

xhr.send();

})

}

document.querySelector('#get-text').onclick = () => {

request('/examples/ajax/01.html')

.then(data => {

document.querySelector('#result').innerHTML = data;

})

.catch(error => {

console.error(error);

});

}

</script>

**CALLBACK**

function loadScript(src, callback) {

let script = document.createElement('script');

script.src = src;

script.onload = () => callback(script);

document.head.append(script);

}

Теперь, если мы хотим вызвать функцию из скрипта, нужно делать это в колбэке:

loadScript('/my/script.js', function() {

// эта функция вызовется после того, как загрузится скрипт

newFunction(); // теперь всё работает

...

});