Frontender[1.0] JavaScript - Promise, цепочка вызовов, then, catch, finally, allSettled, race, any

YouTube	https://youtu.be/uz0FDBT9tp0
Telegram	https://t.me/Dmitry_Kolotilshikov
Github	https://github.com/DmitryKolotilshikov/
Boosty	https://boosty.to/dmitry_ko
# Номер урока	48

💡 Полезные ссылки:

https://learn.javascript.ru/promise-basics

https://learn.javascript.ru/promise-api

💡 Есть «**создающий**» код, который делает что-то, что занимает время. Например, загружает данные по сети. Есть «
потребляющий» код, который хочет получить результат «создающего» кода, когда он будет готов

`Promise` (обещание) – это специальный объект в JavaScript, который связывает «**создающий**» и «**потребляющий**» коды вместе.

`Promise` используется для обработки асинхронных операций и позволяет выполнять код **асинхронно** и управлять состоянием этой операции, предоставляя понятный способ обработки.

💡 Объект `Promise` имеет три состояния:

1. **Pending (ожидание):** Начальное состояние, операция еще не завершена.
2. **Fulfilled (выполнено):** Операция завершена успешно, и `Promise` вернул результат.
3. **Rejected (отклонено):** Операция завершена с ошибкой.

💡

- `resolve` — функция, вызываемая при успешном выполнении.
- `reject` — функция, вызываемая при ошибке.

Методы:

- `.then(callback)` — используется для обработки успешного выполнения `Promise`.
- `.catch(callback)` — используется для обработки ошибок.
- `.finally(callback)` — выполняется в любом случае, после завершения `Promise`.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="script.js" defer></script>
    <title>promise</title>
</head>

<body>
</body>
</html>
```

```
// promise

// -------------- promise (база) --------------

// const promise = new Promise((resolve, reject) => resolve("Hello"));
// promise.then((message) => console.log(message));

// const promise = new Promise((resolve, reject) => reject("Error!!!"));
// promise.then((message) => console.log(message), (errorMessage) => console.log(errorMessag
e));

// const promise = new Promise((resolve, reject) => reject(new Error("Error!!!")));
// promise.then((message) => console.log(message), (error) => console.log(error.message));


// -------------- then, catch, finally --------------

const myPromise = new Promise((resolve, reject) => {
    // Асинхронная операция
    setTimeout(() => {
        const success = true;

        if (success) {
            resolve("Операция выполнена успешно!");
        } else {
            reject("Произошла ошибка!");
        }
    }, 2000);
});

myPromise
```

```
    .then((result) => {
        console.log(result); // success (если resolve)
    })
    .catch((error) => {
        console.error(error); // error (если reject)
    })
    .finally(() => {
        console.log("Завершение операции."); // будет выполнен всегда
    });

// -------------- Асинхронный запрос к серверу (симуляция) --------------

const fetchData = () => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const data = { id: 1, name: "Dmitry" };
            resolve(data);
        }, 1000);
    });
}

fetchData()
    .then((data) => {
        console.log("Полученные данные:", data);
    })
    .catch((error) => {
        console.error("Ошибка загрузки данных:", error);
    })

// -------------- Цепочка Promise --------------

const step1 = new Promise((res) => setTimeout(() => res("Шаг 1 выполнен"), 1000))
const step2 = new Promise((res) => setTimeout(() => res("Шаг 2 выполнен"), 1000))
const step3 = new Promise((res) => setTimeout(() => res("Шаг 3 выполнен"), 1000))

step1
    .then((result) => {
        console.log(result);
        return step2;
    })
    .then((result) => {
        console.log(result);
        return step3;
    })
    .then((result) => {
        console.log(result);
        console.log("Все шаги выполнены!");
    });

// -------------- 🍏 МЕТОДЫ 🍏--------------
// -------------- Promise.all --------------

// Promise.all - запускается множество промисов параллельно и дожидаемся, пока все они выполн
ятся.

const step4 = new Promise((res, rej) => setTimeout(() => rej("Шаг 4 сломался"), 1000))

Promise.all([step1, step2, step3])
```

```
    .then((results) => {
        console.log("[Promise.all] Все выполнены:", results);
    })
    .catch((err) => {
        console.error("[Promise.all] Ошибка в одном из обещаний:", err);
    });

// -------------- Promise.allSettled --------------

/*
Метод Promise.allSettled всегда ждёт завершения всех промисов.

В массиве результатов будет:
{status:"fulfilled", value:результат} для успешных завершений,
{status:"rejected", reason:ошибка} для ошибок.
*/

Promise.allSettled([step1, step2, step3, step4])
    .then((results) => {
        console.log("[Promise.allSettled] Все выполнены:", results);
    })
    .catch((err) => {
        console.error("[Promise.allSettled] Ошибка в одном из обещаний:", err);
    });

// -------------- Promise.any --------------
// Метод Promise.any всегда ждёт завершения первого промиса который выполнился успешно или
// когда все промисы reject

Promise.any([step1, step2, step3, step4])
    .then((results) => {
        console.log("[Promise.any] Выполнен первый:", results);
    })
    .catch((err) => {
        console.error("[Promise.any] Ошибка в одном из обещаний:", err);
    });

// -------------- Promise.race --------------
// Метод Promise.race всегда ждёт завершения самого первого промиса будет ли он resolve или r
eject.

Promise.race([step1, step2, step3, step4])
    .then((results) => {
        console.log("[Promise.race] Выполнен первый:", results);
    })
    .catch((err) => {
        console.error("[Promise.race] Ошибка в одном из обещаний:", err);
    });
48
```