david.webdev

5 Awesome Tips & Tricks For Promises In JavaScript

Follow For More

Web Development | Design | Coding

Swipe

1 Using Promise.race()

A practical way to use the Promise.race() method is to set a time limit for a given task! By using the code below, we can run a race between our timer Promise and the actual request we want to make.

If our request promise is settled before the timer ends, it will be considered a success, otherwise, an error will be thrown!

```
const fakeFetch = () => new Promise((resolve, reject) => {
    setTimeout(resolve, 6000)
})

const timer = timeout => new Promise((_resolve, reject) =>
    setTimeout(() => reject(new Error("Time limit exceeded!")), timeout)
)

Promise.race([fakeFetch(), timer(7000)])
    .then(() => console.log("✅ Successful within time limit"))
    .catch(console.error);
```

Next

2 Avoiding Nested Promises

Instead of nesting promises and creating some really bad looking code, you an use async/await to accomplish the same set of tasks, in a much cleaner way!

```
const sayWord = (word) => Promise.resolve((word));

sayWord('Hello')
    .then(word1 => {
        console.log(word1);
        sayWord('There')
            .then(word2 => {
                console.log(word2);
                sayWord('Everyone')
                    .then(word3 => {
                        console.log(word3)
                        // ...
                    })
                    .catch(console.error);
            })
            .catch(console.error)
    }).catch(console.error);
```

Bad

Good

```
const sayWords = async () => {
    try {
        console.log(await sayWord('Hello'));
        console.log(await sayWord('There'));
        console.log(await sayWord('Everyone'));
    } catch (err) {
        console.error(err);
    }
}
```

Next

3 Using Promise.all()

You can easily use Promise.all() when you want to run multiple promises and wait for them all to be resolved successfully. Keep in mind that if one of them fails, the whole operation will be rejected!

```
const sayName = () => Promise.resolve('David');
const sayJavaScript = () => Promise.resolve('JavaScript');
const sayHello = () => new Promise(resolve => resolve('Hello!'));

Promise.all([sayHello(), sayName(), sayJavaScript()])
    .then(([greeting, firstname, language]) => {
        console.log(`${greeting} ${firstname} learns ${language}!`)
    }) // Hello! David learns JavaScript!
```

If you want to avoid having your promise rejected just because one of the passed promises fail, you can use Promise.allSettled() instead!

Next

4 Using Two Callbacks In a .then Block

The .then block of a promise can actually accept two callbacks as its arguments.

```
const failOnPurpose = () => new Promise((resolve, reject) => {
    reject('Yikes!');
});

failOnPurpose().then(
    () => 'Promise Success!', // first callback (onfulfilled)
    (err) => console.error(err) // second callback (onrejected)
); // Yikes!
```

You can actually use the second callback for error handling purposes instead of adding a .catch block to your promise! Pretty cool!

Next

5 Using Promise.any()

Promise.any() is similar to Promise.race(), the key difference being, Promise.race() will reject if any of the passed promises fail, while Promise.any() stores the first resolved promise regardless if other promises have failed!

```
const p1 = new Promise((res, rej) => setTimeout(() => res('Hello'), 500));
const p2 = new Promise((res, rej) => setTimeout(() => res('World'), 100));
const p3 = new Promise((res, rej) => rej('P3 Fail!'));

const promises = [p1, p2, p3];

Promise.any(promises)
    .then(firstResult => { console.log(firstResult) }) // World
    .catch(errors => { console.error(errors) })
```

If we replace .any() with .race(), then the error in the third promise would not allow the other promises to finish executing, but with .any(), the first successful promise is returned! (p2)

Next