JavaScript TUTORIAL

JavaScript Promise.withResolvers

Summary: in this tutorial, you will learn how to use the JavaScript `Promise.withResolvers()` method to create a new promise with its `resolve` and `reject` functions.

Introduction to the JavaScript Promise.withResolvers method

When creating a new Promise object, you typically pass `resolve` and `reject` functions to the promise constructor like this:

```
const promise = new Promise((resolve, reject) =>{
   // ...
});
```

This allows you to call the `resolve` and `reject` functions inside the promise constructor only.

To call these functions outside of the promise constructor, you often have to write the following boilerplate code:

```
let resolve, reject;

const promise = new Promise((res, rej) => {
    resolve = res;
    reject = rej;
});

Math.random() > 0.5 ? resolve("Success") : reject("Error");
```

In this code:

- First, declare variables that hold the `resolve` and `reject` functions of the promise.

- Second, create a new `Promise` object using the promise constructor and assign the `resolve` and `reject` functions to these variables. This makes the `resolve` and `reject` functions available outside the promise constructor.
- Third, call the `resolve` and `reject` functions outside the promise constructor.

With the `Promise.withResolvers()` function, you can simplify the code like this:

```
const { promise, resolve, reject} = Promise.withResolvers();

Math.random() > 0.5 ? resolve("Success") : reject("Error");
```

In this code, the `Promise.withResolvers()` method returns an object that contains the following properties

- `promise` : a new Promise object
- `resolve` : a function that resolves the promise.
- `reject` : a function that rejects the promise.

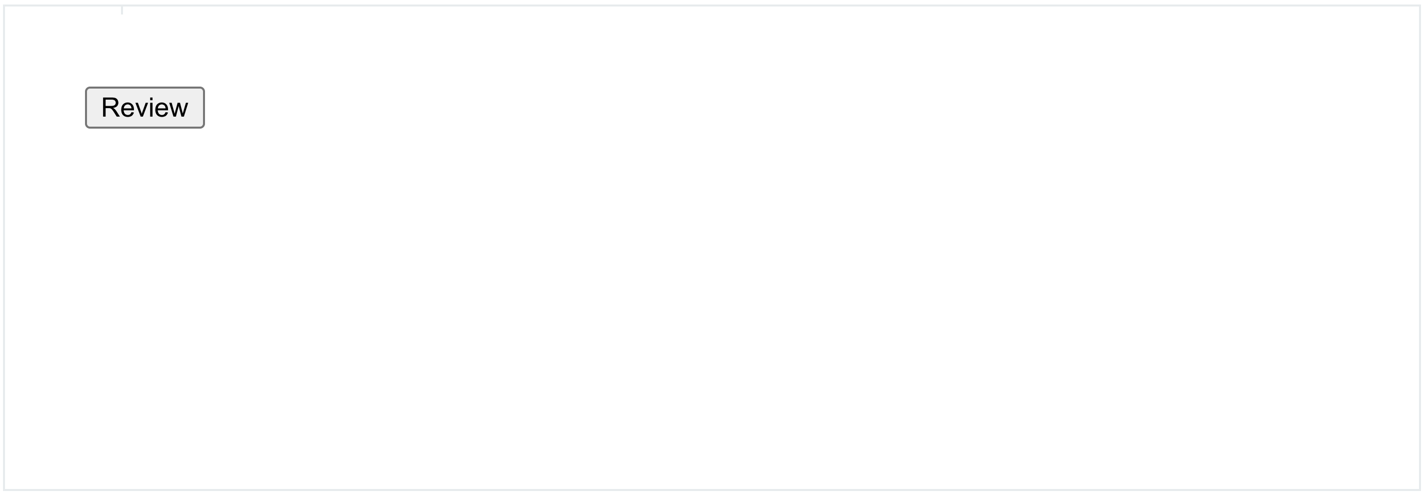
> Note that the `Promise.withResolvers()` has been available since ECMAScript 2024.

JavaScript Promise.withResolvers method example

The following example shows how to use the `Promise.withResolvers`() method to handle user input.

Suppose you have a dialog prompting a user to approve or reject a request. When the user opens the dialog, the `approve` and `reject` buttons appear.

Output



If you don’t use a promise, you can handle the approve/reject button click events like this:

```
const btnReview = document.querySelector('#btnReview');

btnReject.addEventListener('click', () => {
  // handle rejection
  dialog.close();
});

btnApprove.addEventListener('click', () => {
  // handle approval
  dialog.close();
});
```

This code should work fine but has some downsides:

- The code to handle user interaction is spread across event handlers.
- Duplicate code for closing the dialog.

To avoid these issues, you can use a `Promise.withResolvers()` method.

index.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
```

```html
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>JavaScript Promise.withResolvers() method</title>
        <script src="app.js" defer></script>
    </head>
    <body>
        <button id="btnReview">Review</button>
        <dialog>
            <p>Please approve/reject this request?</p>

            <button id="btnApprove">Approve</button>
            <button id="btnReject">Reject</button>
        </dialog>
        <p id="message" hidden></p>
    </body>
</html>
```

app.js

```javascript
const btnReview = document.querySelector('#btnReview');
const btnApprove = document.querySelector('#btnApprove');
const btnReject = document.querySelector('#btnReject');
const dialog = document.querySelector('dialog');

const { promise, resolve, reject } = Promise.withResolvers();

btnReview.addEventListener('click', () => dialog.show());
btnApprove.addEventListener('click', resolve);
btnReject.addEventListener('click', reject);

promise
  .then(() => (message.innerHTML = 'You approved it.'))
  .catch(() => (message.innerHTML = 'You rejected it.'))
  .finally(() => {
    message.hidden = false;
    dialog.close();
    btnReview.remove();
  });
```

How it works.

First, select elements of the pages including `btnReview`, `btnApprove`, `btnReject`, and `dialog`:

```
const btnReview = document.querySelector('#btnReview');
const btnApprove = document.querySelector('#btnApprove');
const btnReject = document.querySelector('#btnReject');
const dialog = document.querySelector('dialog');
```

Second, create a new promise with the resolve and reject functions:

```
const { promise, resolve, reject } = Promise.withResolvers();
```

Third, wire up the event handlers with the click events of the `btnReview`, `btnApprove`, and `btnReject` buttons:

```
btnReview.addEventListener('click', () => dialog.show());
btnApprove.addEventListener('click', resolve);
btnReject.addEventListener('click', reject);
```

Finally, call the promise object with the `then()`, `catch()`, and `finally()` methods:

```
promise
  .then(() => (message.innerHTML = 'You approved it.'))
  .catch(() => (message.innerHTML = 'You rejected it.'))
  .finally(() => {
    message.hidden = false;
    dialog.close();
    btnReview.remove();
  });
```

By using the `Promise.withResolvers()` method, we can achieve two objectives:

- Centralizing the user interaction inside a promise.
- Removing duplicate code by moving it to the `finally()` method of the promise.

Summary

- Use the `Promise.withResolvers()` method to create a new promise with its resolve and reject functions.

Quiz