All About JavaScript Promises

@adityauke

A Promise is an object which is used to find out if the asynchronous operation is completed or not. "The code either executes or fails" in both cases user will be notified.

The Promise object has two properties: state and result.

State:

- **Pending:** initial state, neither fulfilled nor rejected.
- **Fulfilled:** meaning that the operation was completed successfully.
- **Rejected:** meaning that the operation failed.

Result:

- **Undefined:** we get this value when the promise is pending.
- **A result value:** we get this value when the promise is fulfilled.
- **An error object:** we get this value when the promise is rejected.

A promise is said to be settled if it is either fulfilled or rejected, but not pending.

 @adityauke

Creating a Promise:

A Promise object is created using the new keyword and its constructor. This constructor takes a function, called the executor function, as its parameter.

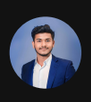
This executor function takes two functions as parameters.

- resolve - It is called when the task completes successfully and returns the results of the task as a value.
- reject - It is called when the task fails and returns the reason for failure, which is typically an error object.

script.js > ...

```
let p1 = new Promise((resolve, reject) => {
    // Code
});
```

resolve & reject are built-in javascript callbacks. We can name them differently also like myResolve & myReject.



@adityauke

How to use a Promise:

When a promise is fulfilled, you can access the resolved data in the .then method of the promise.

script.js > ...

```
promise.then((value) => {
  // use value for something
});
```

When a promise is rejected (that is, the promise fails), you can access the error information returned in the .catch method of the promise

script.js > p1 > <function> > setTimeout() callback

```
promise.catch((error) => {
  // interpret error or display something
});
```

- .then(): It can deal with the resolved case and rejected case also.
- **catch()**: It deals with the rejected case only.

 **@adityauke**

e.g :

script.js > ...

```js
const onload = (src) => {
  return new Promise((resolve, reject) => {
    let script = document.createElement("script");
    script.src = src;
    document.body.appendChild(script);

    script.onload = setTimeout(() => {
      resolve(1);
    }, 5000);

    script.onerror = () => {
      reject(new Error("Something went wrong!!!! plz try again...."));
    }; (script.onerror, 5000);
  });
};

onload("https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js")
  .then((value) => {
    console.log(value);
  }).catch((error) => {
  console.log(error);
});
```

Promise Chaining: You can attach multiple handlers to a promise with the help of promise chaining which solves the problem of Callback() Hell.

```js
Promise.then((value) => {
  (value);
}).then(() => {
  console.log("start executing second script")
}).then(()=>{
  /// so on.....................
})
```

@adityauke