Promises

JavaScript

JavaScript Promise

Promise is a good way to handle asynchronous operations.

It is used to find out if the asynchronous operation is successfully completed or not.

A promise may have one of three states.

- Pending - process is not complete
- Fulfilled - operation is successful
- Rejected - an error occurs

Create A Promise

To create a promise object, we use the Promise() constructor.

```
let promise = new Promise(function(resolve, reject){
    //do something
});
```

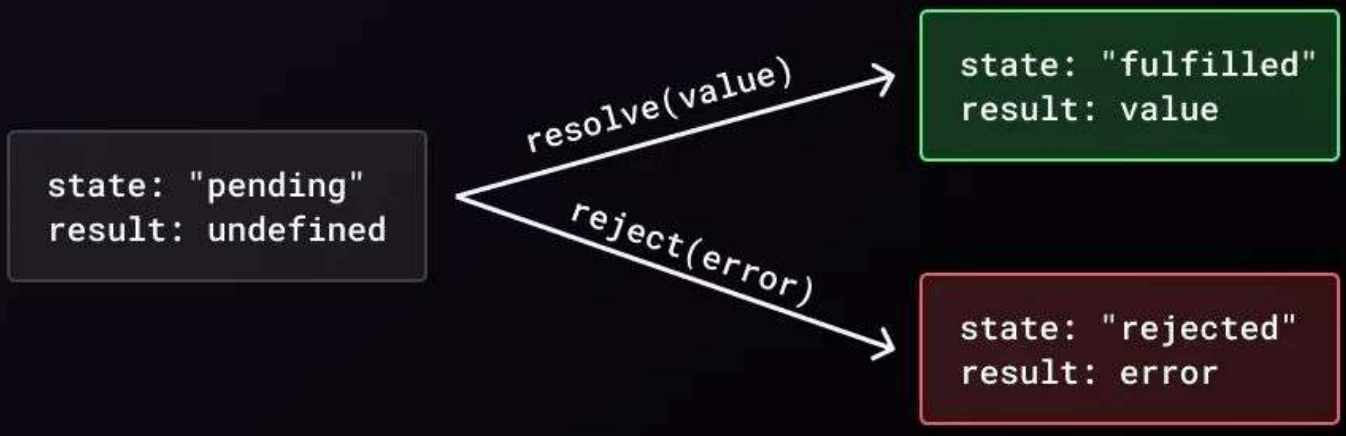
If the promise returns successfully, the resolve() function is called.

And, if an error occurs, the reject() function is called.

Example

Let's suppose that the program below is an asynchronous program.

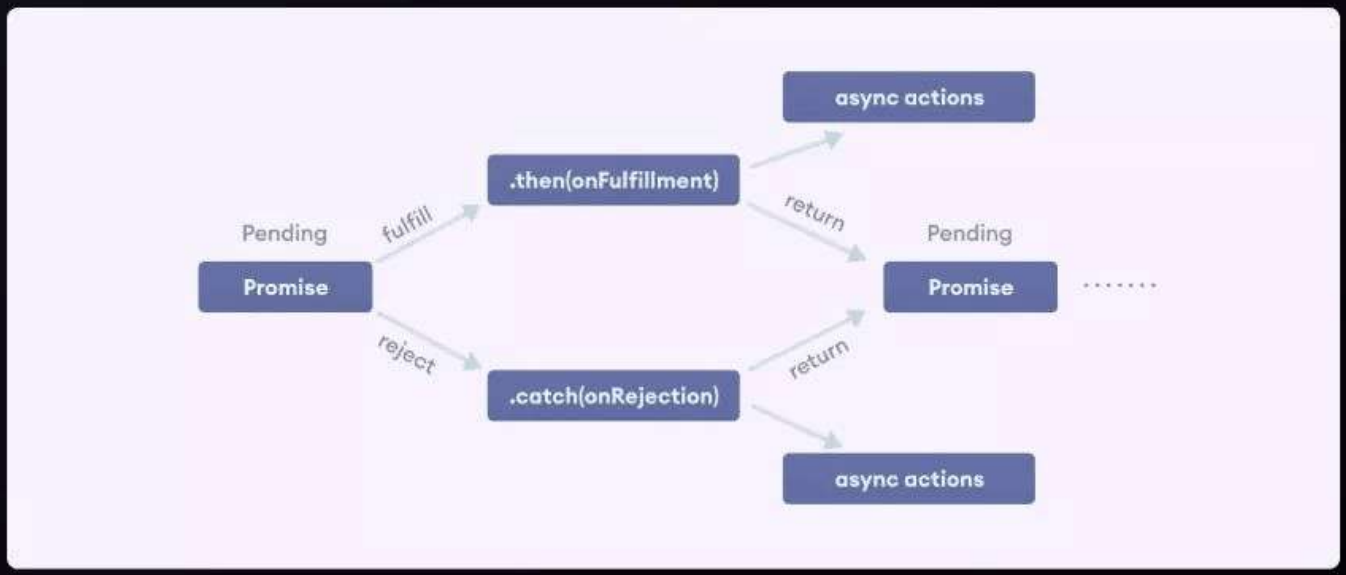
```
const count = true;

let countValue = new Promise(function (resolve, reject) {
    if (count) {
        resolve("There is a count value.");
    } else {
        reject("There is no count value");
    }
});

console.log(countValue);

// Promise {<resolved>: "There is a count value."}
```



Promise Chaining

Promises are useful when you have to handle more than one asynchronous task, one after another.
For that, we use promise chaining.

You can perform an operation after a promise is resolved using methods then(), catch() and finally().



then() Method

The then() method is used with the callback when the promise is successfully fulfilled or resolved.

```
promiseObject.then(onFulfilled, onRejected);
```

You can chain multiple then() methods with the promise.

```
// returns a promise
let countValue = new Promise(function (resolve, reject) {
  resolve("Promise resolved");
});

// executes when promise is resolved successfully
countValue
  .then(function successValue(result) {
    console.log(result);
  })
  .then(function successValue1() {
    console.log("You can call multiple functions this way.");
  });

// Promise resolved
// You can call multiple functions this way.
```

catch() Method

The catch() method is used with the callback when the promise is rejected or if an error occurs.

```
// returns a promise
let countValue = new Promise(function (resolve, reject) {
   reject('Promise rejected');
});

// executes when promise is resolved successfully
countValue
  .then(
    function successValue(result) {
        console.log(result);
    },
 )
// executes if there is an error
  .catch(
    function errorValue(result) {
        console.log(result);
    }
  );

// Promise rejected
```

Promises Vs Callback

JavaScript Promise

- The syntax is user-friendly and easy to read.
- Error handling is easier to manage.

```
api().then(function(result) {
    return api2() ;
}).then(function(result2) {
    return api3();
}).then(function(result3) {
    // do work
}).catch(function(error) {
    //handle any error that
    //may occur before this point
});
```

JavaScript Callback

- The syntax is difficult to understand
- Error handling may be hard to manage

```
api(function(result){
    api2(function(result2){
        api3(function(result3){
            // do work
            if(error) {
                // do something
            }
            else {
                // do something
            }
        });
    });
});
```