# **Promises**

In JavaScript, a 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
* Fulfilled
* Rejected

A promise starts in a pending state. That means the process is not complete. If the operation is successful, the process ends in a fulfilled state. And, if an error occurs, the process ends in a rejected state.

For example, when you request data from the server by using a promise, it will be in a pending state. When the data arrives successfully, it will be in a fulfilled state. If an error occurs, then it will be in a rejected state.

## **Create a Promise**

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

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

The Promise() constructor takes a function as an argument. The function also accepts two functions resolve() and reject().

If the promise returns successfully, the resolve() function is called. And, if an error occurs, the reject() function is called.

Let's suppose that the program below is an asynchronous program. Then the program can be handled by using a promise.

### **Example 1: Program with a Promise**

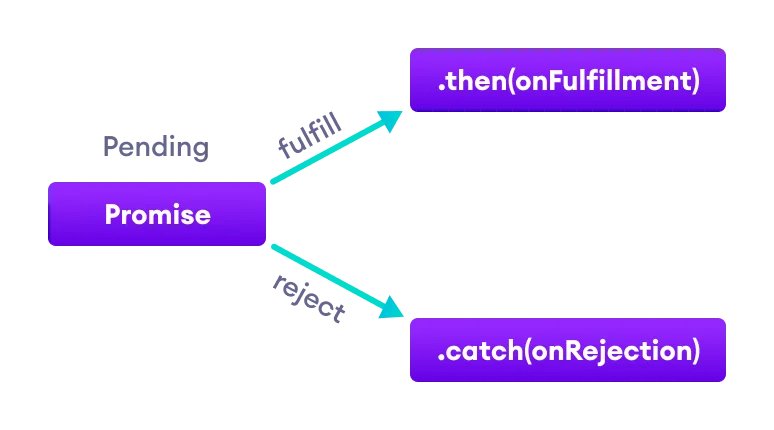
| const count = true;  let countValue = new Promise(function (resolve, reject) {  if (count) {  resolve("There is a count value.");  } else {  reject("There is no count value");  } });  console.log(countValue); |
| --- |

Output

| Promise {<resolved>: "There is a count value."} |
| --- |

In the above program, a Promise object is created that takes two functions: resolve() and reject(). resolve() is used if the process is successful and reject() is used when an error occurs in the promise.

The promise is resolved if the value of count is true.



## **JavaScript 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().

### **JavaScript then() method**

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

The syntax of then() method is:

| promiseObject.then(onFulfilled, onRejected); |
| --- |

| *// 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.");  }); |
| --- |

In the above program, the then() method is used to chain the functions to the promise. The then() method is called when the promise is resolved successfully.

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

### **JavaScript catch() method**

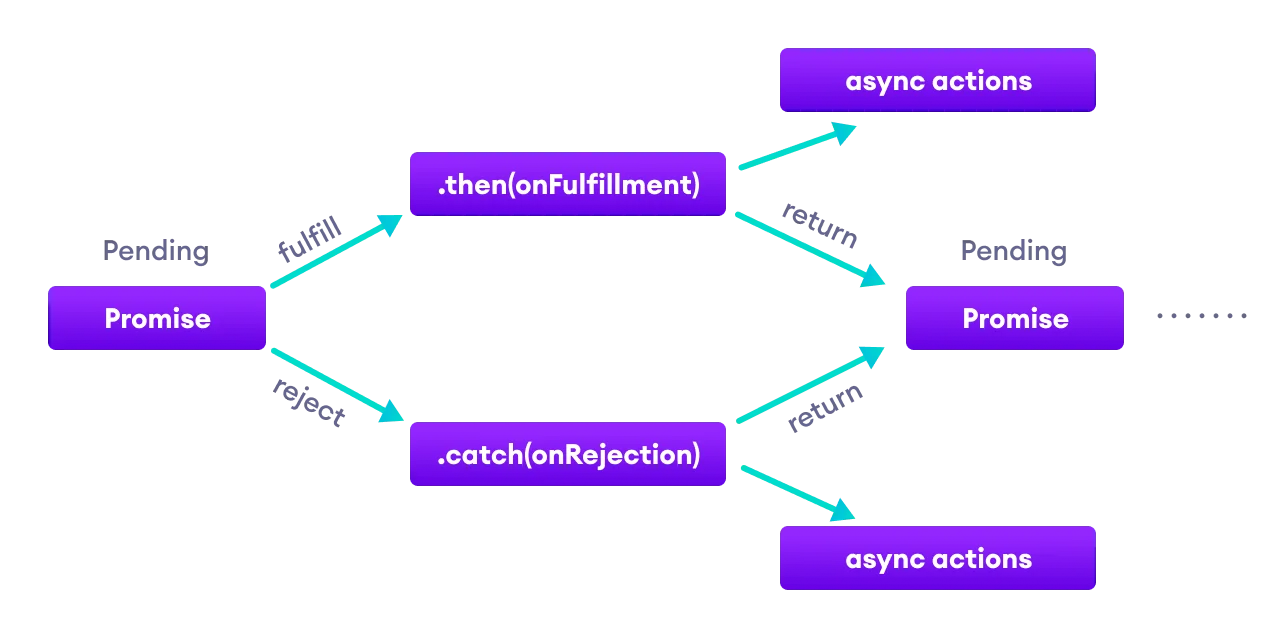
The catch() method is used with the callback when the promise is rejected or if an error occurs. For example.

| *// 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);  } ); |
| --- |

Output

| Promise rejected |
| --- |

In the above program, the promise is rejected. And the catch() method is used with a promise to handle the error.



### **JavaScript finally() method**

You can also use the finally() method with promises. The finally() method gets executed when the promise is either resolved successfully or rejected. For example:

| *// returns a promise* let countValue = new Promise(function (resolve, reject) {  *// could be resolved or rejected*   resolve('Promise resolved');  });  *// add other blocks of code* countValue.finally(  function greet() {  console.log('This code is executed.');  } ); |
| --- |

## **Promise Methods**

| Method | Description |
| --- | --- |
| all(iterable) | Waits for all promises to be resolved or any one to be rejected |
| allSettled(iterable) | Waits until all promises are either resolved or rejected |
| any(iterable) | Returns the promise value as soon as any one of the promises is fulfilled |
| race(iterable) | Wait until any of the promises is resolved or rejected |
| reject(reason) | Returns a new Promise object that is rejected for the given reason |
| resolve(value) | Returns a new Promise object that is resolved with the given value |
| catch() | Appends the rejection handler callback |
| then() | Appends the resolved handler callback |
| finally() | Appends a handler to the promise |