The **Promise** object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.

var promise1 = new Promise(function(resolve, reject) {

setTimeout(resolve, 100, 'foo');

});

console.log(promise1);

// expected output: [object Promise]

**Methods**

[**Promise.prototype.catch(onRejected)**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/catch)

Appends a rejection handler callback to the promise, and returns a new promise resolving to the return value of the callback if it is called, or to its original fulfillment value if the promise is instead fulfilled.

[**Promise.prototype.then(onFulfilled, onRejected)**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then)

Appends fulfillment and rejection handlers to the promise, and returns a new promise resolving to the return value of the called handler, or to its original settled value if the promise was not handled (i.e. if the relevant handler onFulfilled or onRejected is not a function).

[**Promise.prototype.finally(onFinally)**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/finally)

Appends a handler to the promise, and returns a new promise which is resolved when the original promise is resolved. The handler is called when the promise is settled, whether fulfilled or rejected.

The **Promise.all(iterable)** method returns a single [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) that resolves when all of the promises in the iterable argument have resolved or when the iterable argument contains no promises. It rejects with the reason of the first promise that rejects.

var promise1 = Promise.resolve(3);

var promise2 = 42;

var promise3 = new Promise(function(resolve, reject) {

setTimeout(resolve, 100, 'foo');

});

Promise.all([promise1, promise2, promise3]).then(function(values) {

console.log(values);

});

// expected output: Array [3, 42, "foo"]

The **catch()** method returns a Promise and deals with rejected cases only. It behaves the same as calling [Promise.prototype.then(undefined, onRejected)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then" \o "The then() method returns a Promise. It takes up to two arguments: callback functions for the success and failure cases of the Promise.) (in fact, calling obj.catch(onRejected) internally calls obj.then(undefined, onRejected)).

var promise1 = new Promise(function(resolve, reject) {

throw 'Uh-oh!';

});

promise1.catch(function(error) {

console.log(error);

});

// expected output: Uh-oh!

===============================

The **then()** method returns a [Promise](https://developer.mozilla.org/en-US/docs/Web/API/Promise). It takes up to two arguments: callback functions for the success and failure cases of the Promise.

var promise1 = new Promise(function(resolve, reject) {

resolve('Success!');

});

promise1.then(function(value) {

console.log(value);

// expected output: "Success!"

});

=================================

The **Promise.race(iterable)** method returns a promise that resolves or rejects as soon as one of the promises in the iterable resolves or rejects, with the value or reason from that promise.

var promise1 = new Promise(function(resolve, reject) {

setTimeout(resolve, 500, 'one');

});

var promise2 = new Promise(function(resolve, reject) {

setTimeout(resolve, 100, 'two');

});

Promise.race([promise1, promise2]).then(function(value) {

console.log(value);

// Both resolve, but promise2 is faster

});

==============================

The **Promise.reject(reason)** method returns a Promise object that is rejected with the given reason.

function resolved(result) {

console.log('Resolved');

}

function rejected(result) {

console.log(result);

}

Promise.reject(new Error('fail')).then(resolved, rejected);

=================================

The **Promise.resolve(value)** method returns a [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) object that is resolved with the given value. If the value is a thenable (i.e. has a ["then" method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then)), the returned promise will "follow" that thenable, adopting its eventual state; if the value was a promise, that object becomes the result of the call to Promise.resolve; otherwise the returned promise will be fulfilled with the value.

var promise1 = Promise.resolve([1, 2, 3]);

promise1.then(function(value) {

console.log(value);

// expected output: Array [1, 2, 3]

});

===========================

var isMomHappy = false;

// Promise

var willIGetNewPhone = new Promise(

function (resolve, reject) {

if (isMomHappy) {

var phone = {

brand: 'Samsung',

color: 'black'

};

resolve(phone); // fulfilled

} else {

var reason = new Error('mom is not happy');

reject(reason); // reject

}

}

);

var askMom = function () {

willIGetNewPhone

.then(function (fulfilled) {

// yay, you got a new phone

console.log(fulfilled);

// output: { brand: 'Samsung', color: 'black' }

})

.catch(function (error) {

// oops, mom don't buy it

console.log(error.message);

// output: 'mom is not happy'

});

};

askMom();