Understand promise like a progress bar.

The might not have finished yet, and you might not have access to the data to run an operation on, but you want a reference to that in progress task itself so that you could write logic around it.

Example: I want to run 3 separate tasks and then aggregate the results when all three come back and do something with the data, even though the tasks might finish at different times. This is kind of difficult to represent in callback model because you don’t have access to that in progress object.

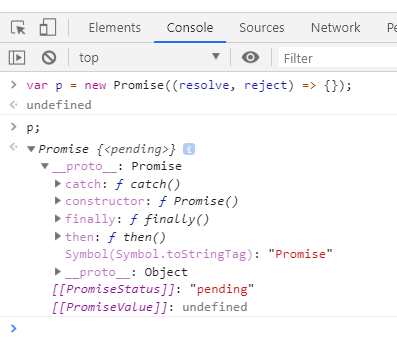
Promise object

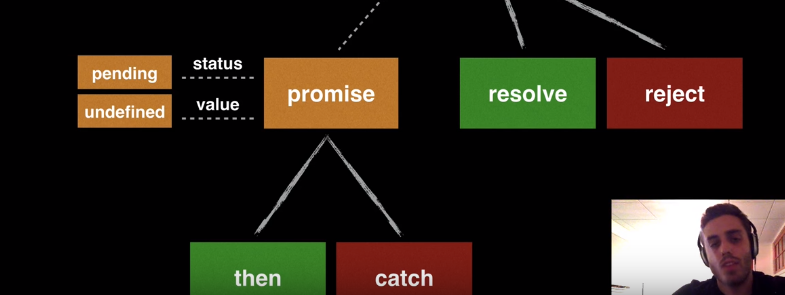
1. Two properties : *[[PromiseStatus]]* && *[[PromiseValue]]. When we create a promise, the*

*[[PromiseStatus]]*: "pending"

*[[PromiseValue]]*: undefined

1. Two functions that you can call – then and catch

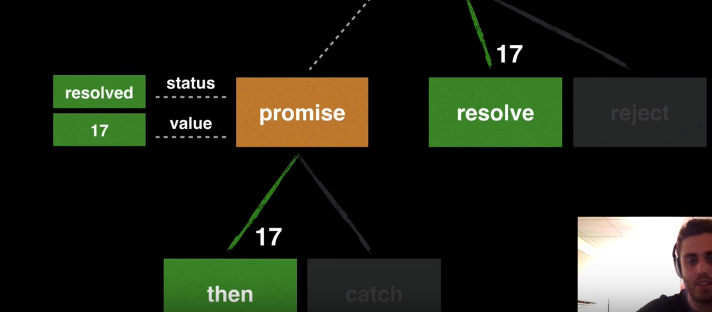




Resolve

What is resolve? If promise is progress bar then, resolve function will say this is no longer in progress, this is finished and got the value from our calculation. So it’s gonna tell everyone that was referencing that progress bar, hey! It’s not in progress anymore …. We have the value, you can go ahead with rest of your execution of your function.

When we resolve a promise, we resolve it with a value. Then that becomes value of that promise and then any function waiting for that value now uses that value to continue its execution.

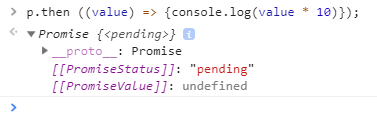


When we resolve a promise with some value say number 17, then it severed its connection with reject function. So if I call reject function after it is resolved then nothing will happen.

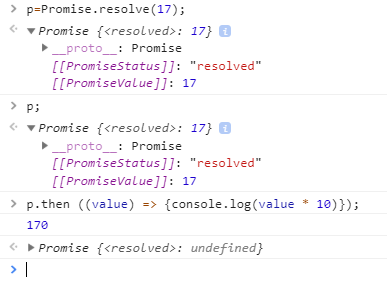
After we resolve a promise, the promise updates its status : resolved and it will update its value to 17. This will also trigger the “then” function with the argument of promise’s value passed in.

This will severe its connection with then function.

Now when I do : p.then ((value) => {console.log(value \* 10)});. Hit enter 🡪 The status is still pending, because then function is not triggered yet. Then function is only going to be triggered once promise is resolved.



When I call resolve,



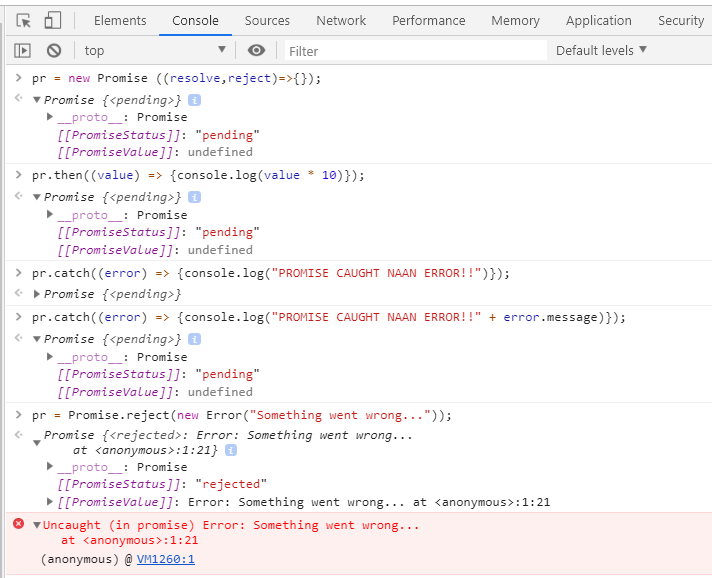
Reject

When you reject a promise, you can pass any value you want but normally you pass the JavaScript error object. Let say something went wrong … our server went down or something, we choose to reject the promise:

* Then resolve branch is severed. So, if you reject a promise then you can no longer ever resolve that promise.
* The status of promise updates to “rejected” and the value becomes that error object.
* We go down the catch and pass the error object in.

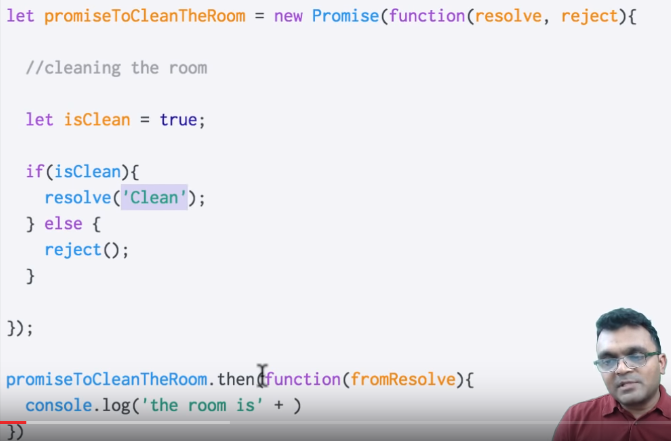
Now anything downstream that was listening to .then function knows … hey! This promise is errored out and you can no longer execute the then function you can catch that function and handle it.

1. I create a new promise object 🡪 pr = new Promise ((resolve,reject)=>{});
2. Add .then pr.then((value) => {console.log(value \* 10)}); 🡪 status still “pending”
3. Add .catch function pr.catch((error) => {console.log("PROMISE CAUGHT NAAN ERROR!!" + error.message)}); 🡪 status still “pending”
4. Reject the promise 🡪pr = Promise.reject(new Error("Something went wrong...")); Then,
5. *[[PromiseStatus]]*: "rejected"
6. *[[PromiseValue]]*: Error: Something went wrong... at <anonymous>:1:21



Promise is created with a callback function. Callback function takes two arguments --- resolve & reject. There are 3 parts in promise definition:

1. Task (which returns a token (true/false))
2. Token = true? Yes, Call resolve [ .then function is triggered]
3. Token = true? No, Call reject [ .catch function is triggered]







step1 = function() {

return new Promise(function(resolve,reject) {

resolve("Step 1. Boil Water ");

});

};

step2 = function(mes) {

return new Promise(function(resolve,reject) {

resolve(mes + ", Step 2. Add Tea & Sugar ");

});

};

step3 = function(mes) {

return new Promise(function(resolve,reject) {

resolve(mes + "Step 3. Serve hot Tea!!!");

});

};

step1().then(function(a){console.log(a); return step2(a);})

.then(function(b){console.log(b); return step3(b);}) // this "then" called with resolve message "b" from step1

.then(function(c){ console.log("finished "+c)}); // this "then" called with resolve message "c" from step2