Closure and Callbacks

By CODEMIND Technology

Contact us 966 5044 698
966 5044 698

JavaScript Advance Stuff

- Closure
- Callbacks
  - setTimeout( ) function
  - setInterval( ) function
- Promise

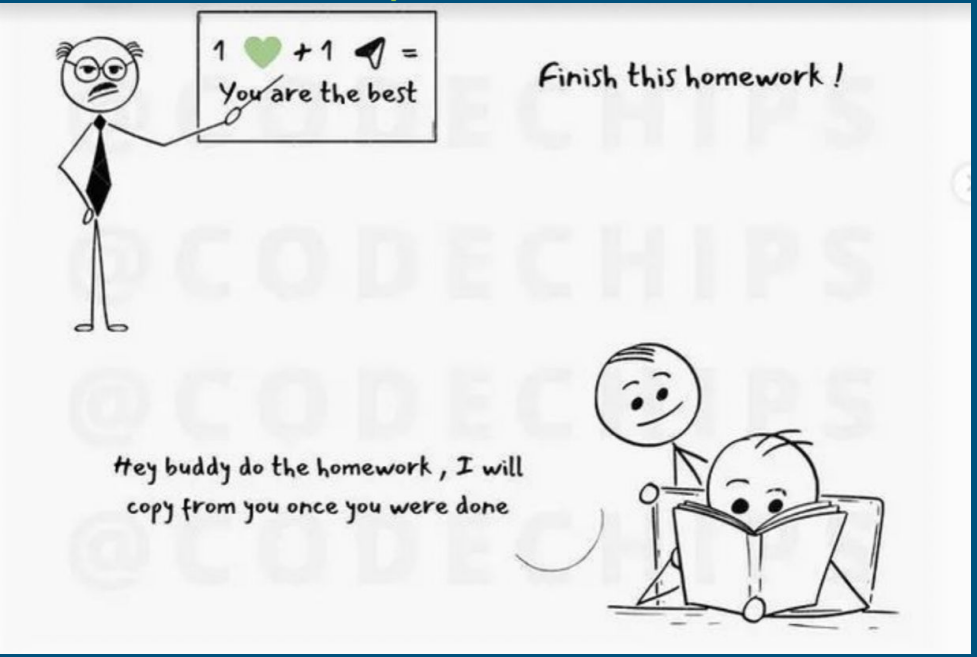
Closure: A closure is an inner function that has access to the outer functions variables

The closure has 3 scope chains:

1. It has access to its own scoped variables
2. It has access to the outer functions variable
3. It has access to the global variables.

```
var global_variable = 100;
function outer_fun() {
    let local_variable = 200;
    let inner_fun = function () {
        let own_variable = 300;
        console.log('Global variable:', global_variable);
        console.log('Local variable:', local_variable);
        console.log('Own variable:', own_variable);
    }
    return inner_fun;
}
let inner = outer_fun();
inner();
```

General Example



So you have 2 jobs to be done

- do_homework
- copy_homework

```
function do_homework( ){
    //doing math
}

function copy_homework(
    //copying homework
}

do_homework();
copy_homework();
```

Problem: Even though do_homework( ) is called before copy_homework( ), you don't know how long it will take for your friend to complete homework and copy_homework( ) will called immediately of no use

Callbacks

In JavaScript, we can also pass a function as an argument to another function.

This function that is passed as an argument to another function is called a callback

```
function do_homework(callbacks) {
    console.log("Doing home work.. Solving tricky problem");
    console.log("Finally, solved ");
    callbacks();
}

function copy_homework() {
    console.log("Copy homework from friend's notes");
}

do_homework(copy_homework);
```

setTimeout() function in JS

There is a built-in method in JavaScript called “setTimeout”, which calls a function or evaluates an expression after a given period of time (in milliseconds). So here, the “message” function is being called after 3 seconds have passed. (1 second = 1000 milliseconds)

Syntax: setTimeout(function, milliseconds);

Where → function - a function containing a block of code
milliseconds - the time after which the function is executed

```
setTimeout(showNotification, 2000);

function showNotification() {
    console.log("Showing notification...");
}
```

General Example

I Promise my friends, Pls allow me sometime.
I will share today's session notes and recording till evening

Waiting period before sharing recording and notes

Pending State

Full filled state

Sir shared notes and recording
on or before evening.
Promise Full filled

Rejected state

Sir does not shared notes and recording
on or before evening.
Promise not Full filled or Rejected

Then

You can read notes and go thru recordings

Your **trouble** starts
to prepare notes

Opps.. Yourself you have to prepare notes

Finally

Some how you will have your session notes

Promise

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 have one of the three states as:

- 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

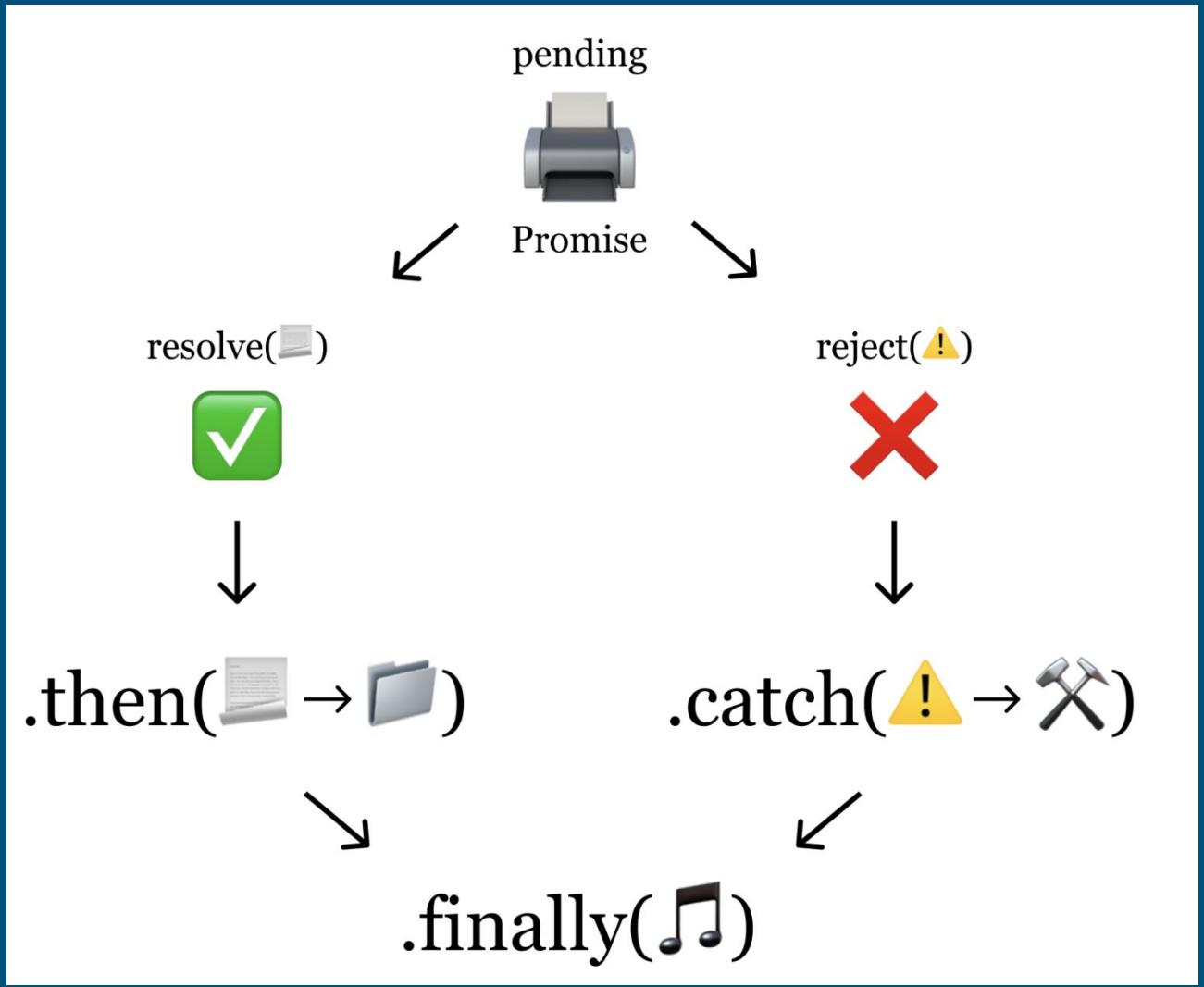
To create a promise object, we use the Promise() constructor.

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

Promise Flow

and

It’s methods



Promise methods

then( ): The then() method is called when the promise is resolved successfully.

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

finally( ): The finally() method gets executed when the promise is either resolved successfully or rejected.

```
let notes_sharing_status = true;
const promise_to_share_notes = new Promise(function (resolve, reject) {
    if (notes_sharing_status) {
        resolve("Sir.. Shared Notes and recordings !");
    } else {
        reject("Opps.. Sir did not share notes");
    }
});
promise_to_share_notes.then(function (on_success) {
    console.log(on_success);
}).catch(function (on_rejection) {
    console.log(on_rejection);
}).finally(function () {
    console.log("Finally.. Somehow you will have session notes");
});
```

Promise with arrow function

```
let notes_sharing_status = true;
const promise_to_share_notes = new Promise((resolve, reject) => {
    if (notes_sharing_status) {
        resolve("Sir.. Shared Notes and recordings !");
    } else {
        reject("Opps.. Sir did not share notes");
    }
});
promise_to_share_notes
    .then(on_success => console.log(on_success))
    .catch(on_rejection => console.log(on_rejection))
    .finally(() => console.log("Finally.. Somehow you will have session notes"));
```

Thank you