Introduction

Dealing with the asynchronous nature of Javascript can be very challenging and frustrating. Callbacks have been for a long time the default way to do things. ES6 gave us a alternative to callbacks with promises. Promises are natively available in Node.js since version 4.

What is that?

A promise is an abstraction that allows a function to return an object called *promise*. A promise is the eventual result of an asynchronous operation. We say that a promise is **pending** when the asynchronous operation is not complete. A promise is **fulfilled** when the operation has been successfully completed. A promise is **rejected** when the operation failed.

Constructing a promise

In ES6, you can create a promise with the *Promise* constructor. It takes a function with two parameters, usually called *resolve* and *reject*. *resolve* is the function we will call when our promise is fulfilled, *reject* will be called when our promise is rejected.

Let's start with a function that returns a promise. This promise will always be fulfilled.

const myPromise = () => {

return new Promise( ( resolve, reject ) => {

console.log('I promise!')

resolve()

})

}

myPromise()

.then(() => {

console.log('I made it!')

})

myPromise returns a promise. When we call our function, the promise is pending, it's neither fulfilled or rejected. We print out I promise! and we call the resolve function. The then() method is responsible for handling a fulfilled promise. The resolve() call triggers the then() method and we print out I made it!

Let's now see a rejected promise:

const rejectedPromise = () => {

return new Promise( ( resolve, reject ) => {

console.log('I promise!')

reject('You lied to me!!')

})

}

rejectedPromise()

.then(() => {

console.log('I made it!')

})

.catch(err => {

console.log('How dare you?')

console.log(err)

})

Here, our promise calls the reject function, meaning that our promise is rejected. This triggers the catch method. It is a good practice to call reject with an error message. Note that the then() method is **NOT** called in this case.

### I promise then I promise then I promise then I promise then ...

The amazing thing about promises is the ability to chain them. If we take our previous example and add an extra then():

const rejectedPromise = () => {

return new Promise( ( resolve, reject ) => {

console.log('I promise!')

reject('You lied to me!!')

})

}

rejectedPromise()

.then(() => {

console.log('I made it!')

})

.catch(err => {

console.log('How dare you?')

console.log(err)

})

.then(() => {

console.log('I forgive you no matter what.')

})

This last then() will always run. If our promise is fulfilled, the first then will be executed, the catch will be skipped, and finally our last then will be run.

Let's create three promises and chain them:

const promiseToLove = iAmMature => {

return new Promise( ( resolve, reject ) => {

if( iAmMature ){

resolve('I love you so much!')

} else {

reject("It's not you, it's me...")

}

})

}

const promiseToProtect = iAmNice => {

return new Promise( ( resolve, reject ) => {

if( iAmNice ){

resolve('I promise I will protect you!')

} else {

reject('What? Get lost!')

}

})

}

const promiseToBeHereOnTime = hairLooksGood => {

return new Promise( ( resolve, reject ) => {

if( hairLooksGood ){

resolve('I promise I will be there!')

} else {

reject('How about tomorrow?')

}

})

}

//First promise

promiseToLove(true)

.then(statement => {

console.log(statement)

})

.catch(statement => {

console.log(statement)

})

//returns another promise

.then(() => promiseToProtect(true))

//handles our second promise

.then(statement => {

console.log(statement)

})

.catch(statement => {

console.log(statement)

})

// returns annother promise

.then(() => promiseToBeHereOnTime(true))

// handles our third promise

.then(statement => {

console.log(statement)

})

.catch(statement => {

console.log(statement)

})

// this always runs

.then(() => {

console.log('And they lived happily ever after!!!')

})

Our three functions take a single parameter ( a boolean ). If the argument is set to true, the promise will be fulfilled, otherwise, it will be rejected. Once a promise is settled, we return another one and deal with that one...

### Promises in real life

In Node.js, not all functions support promises out of the box. To solve this, you can use the promisify method in the util module. It takes a function and transforms it into a function that returns a promise.

#### Cloning a file

To clone a file, we will read its content then write it to a new file. Callback style, you would have something like this:

const fs = require('fs')

fs.readFile('myFile.js', 'utf-8', (err, data) => {

fs.writeFile('clone.js', data, err => {

if(err){

throw err

} else {

console.log('All done')

}

})

})

Ok, we can already see the gates of callback hell in the distance. Let's promisify this thing. I will even write a file first, then read it, then write in a new one, then read our new clone. Yeah, I know, I'm nuts...

const fs = require('fs')

// Get the promisify method from the util module

const { promisify } = require('util')

// Promisify our readFile and writeFile function

const readFile = promisify(fs.readFile)

const writeFile = promisify(fs.writeFile)

writeFile('original.txt', 'Promise me you will clone me!')

.then(() => readFile('original.txt', 'utf-8'))

.then(content => writeFile('clone.txt', content))

.then(() => readFile('clone.txt', 'utf-8'))

.then(cloneContent => console.log(cloneContent))

.catch(err => console.log('Error occured:', err))

// Promise me you will clone me!

Yeah, that's sexy. Why would you write with callbacks anymore? Our writeFile and readFile return either the file's content when their resolve() is called, or the error message if their reject() is called. In our example, I only wrote one catch(). But this catch() will be called if any of the promises before is rejected:

writeFile('original.txt', 'Promise me you will clone me!')

.then(() => readFile('404NOTFOUND.txt', 'utf-8')) // <= Error here

.then(content => writeFile('clone.txt', content))

.then(() => readFile('clone.txt', 'utf-8'))

.then(cloneContent => console.log(cloneContent))

.catch(err => console.log('Error occured:', err)) // <= Trigger this

//Error occured: { Error: ENOENT: no such file or directory, open //'404NOTFOUND.txt'

// errno: -2,

// code: 'ENOENT',

// syscall: 'open',

// path: '404NOTFOUND.txt' }

Alright, this should be more than enough to get you started with your own promises. Save your sanity, make your code cleaner, use promises and not callbacks :)