<https://jssaini07.medium.com/understanding-javascript-promises-by-writing-a-polyfill-69c8d51c23b4>

# Understanding Javascript Promises by writing a polyfill

Javascript is a single threaded language, which essentially means that it has no native support for performing multiple tasks simultaneously. This poses a problem because on UI it is crucial to have an async behaviour. Javascript solves this problem using the event loop. In order to adapt to this async behaviour the simplest option we can opt for are **callbacks** which have their own issues. Another option available to us is **Promises**

As per MDN web docs, Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value. A promise can be in three states **Pending**, **Fulfilled**or**Rejected**. It provides us methods like then and catch to pass in the appropriate handlers.

## Let’s try to understand this concept by implementing our own Custom ****Promise**** class.

Logo

Description automatically generated

Image courtesy <https://blog.alexdevero.com/>

# ****Setting the base****

Lets start by defining the problem statement and the expectations.

We will start by implementing a function named **getNumber** which generates a random number (let’s call it **randomNumber**), if randomNumber is divisible by 5 it will reject the promise else it will resolve the promise. Let’s also keep the promise resolution/rejection time as a variable.

Following is one way to implement this method

Text

Description automatically generated

getNumber when called will return a Promise which will either resolve or reject with a string that contains the random number. The promise returned will resolve/reject in randomNumber\*10 ms (which gives us a range of 0-1s)

We can quickly create a button and a div to show the content returned, This is how it will all look like

Text

Description automatically generated

I have added a button and a div to show the result of the promise. This is just for visualization purposes. Following is how the complete interaction looks like

Graphical user interface, text, application, chat or text message

Description automatically generated

Now that everything works with ES6's Promise, lets define a class named **CustomPromise** and replace **Promise** with it.

Text

Description automatically generated

**Our goal now is to implement the CustomPromise class so that it provides some of the capabilities that are provided by the inbuilt ES6 promises**

# Implementing Custom Promise

We initialize a Promise by passing it the action we want to perform. This method receives resolve and reject methods in its arguments. Depending on our logic we call one of these methods which decides the final state of the Promise.

*CustomPromise should support initialization in the following way  
new CustomPromise((res, rej) => {/\* \*/})*

We need CustomPromise’s constructor to call the function passed to it with a context of resolver and rejector methods

Text

Description automatically generated

At this point we have empty resolver and rejector methods that don’t do anything with the result. CustomPromise on resolution should call the method it was passed using then and on rejection should call the function passed via catch.

To simplify the implementation for this phase let’s not consider [**then chaining**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises#Chaining)(We will take a look at then chaining in the next phase). We will simply create two methods then and catchthat will register the handlers so that upon resolution/rejection of the promise these methods can be called.

Text

Description automatically generated

**Note that then and catch will just update the thenFn and catchFn methods to point to the handlers passed during initialization. These methods are called when resolver/rejector are called.**

Let’s keep a track of our Custom Promise’s state by introducing a variable name CustomPromiseState. CustomPromiseState can be in three states PENDING RESOLVED or REJECTED

Also, Once a promise is resolved, rejector method should have no effect on it i.e it shouldn’t be executed (same is true vice versa). Let’s add a check such that we call thenFn only if our Promise is in PENDING state

Text

Description automatically generated

🎉 Voilà, With this code we have written the simplest polyfill for Promise. This can be seen in the demo below. We are getting similar behaviour with CustomPromise as we were getting with ES6’s Promise.

Text

Description automatically generated

Next, Let’s handle **then chaining**

# Updating our CustomPromise class to support then chaining

Let’s start by setting up the base again. Replace CustomPromise by ES6’s Promise.

We will set the randomNumber to some known quantity (say 10) and create a method called incrementBy which will receive two arguments (original value and increment quantity), it returns the sum of both. Next, we update our clickHandler and chain three thens such that we increment the original resolved value i.e 10 by 10, 20 and 30. Hence, the final value that we see becomes 70.

Text

Description automatically generated

Now let’s replace Promise by CustomPromise and update our class to support this chaining functionality.

With current implementation, our code will show 10 on UI because we are not keeping track of all the then function handlers. Then last registered method i.e display in this case is the only method that gets called when the promise resolves. Let’s change this behaviour by introducing a list to store all the registered then handlers in sequence.

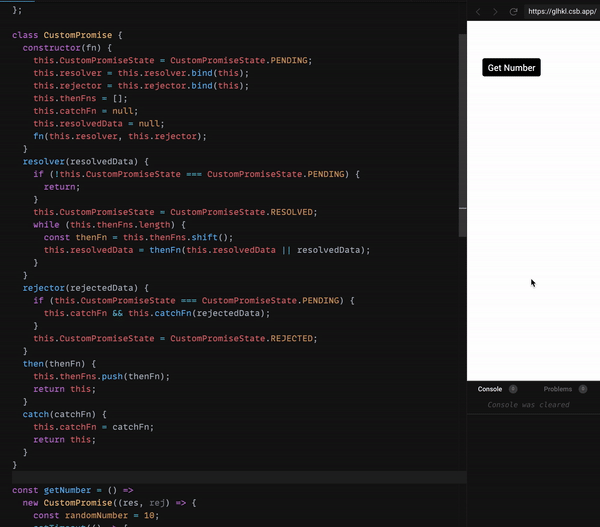
Text

Description automatically generated

Following are the changes made,

* We updated thenFns such that it is now initialized to an empty list
* Inside our then method, we are now pushing each then handler onto the list
* In our resolver, instead of calling the last then handler, we now loop through entire thenFns queue and call each method. The first then handler receives resolvedData as the argument following which each method receives the value returned by previous handler.

That’s it, this makes our class compatible with the changes we did (i.e chaining)



Note that I have not done chaining for the catch functions. In order to do that you’ll need to know the following rules

* then functions will be called in sequence as long as none of them throw an error
* If at any point in the chain an error is thrown it goes to the next catch method in sequence
* After calling the catch method, if it throws an error the next catch method in sequence is called. If the current catch method doesn’t throw any error, it calls the next then method in sequence

Before I end this article, here is an example where I use our Custom Promise class to fetch the status of 3 websites sequentially and display them on UI

Text

Description automatically generated

With that I will mark the end of this article. There is still a lot that can be implemented in the Custom Promise class that we created, For instance you can implement finally method, catch chaining, allSettled and many more. You can find more methods to implement [here](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise). Maybe I’ll write a part two and cover these methods there (no promises 😉).

If you find any mistake in the article or if you want to discuss something I encourage you to leave a comment. If you got something to learn do consider sharing this article. Thank you and keep learning.