The ES6 Promises

codeburst.io/the-es6-promises-87a979ab27e4

Promises introduced in ES6 to improve handling of async operations. They are around for a long time but with ES6 they became part of vanilla JavaScript. Now you can use Promises in your JavaScript code without any additional library.

This post is a very basic explanation of promises to get you started using them in your code.

So, What exactly is a Promise?

Promise gives us control over our async function and allows us to either resolve or reject an operation request.

Let's understand them by example. Here I am creating a Promise

```
const isSmallThenTen = (num) => {
    return new Promise((resolve, reject) => {
        if(num < 10) {
            resolve(true)
        } else {
            reject(false)
        }
    })
}
```

We created a function which takes a number, if `num < 10` evaluates to `true` then the promise is resolved and `true` is returned. If the condition fails the Promise gets rejected and `false` is returned.

Let's take a look how we are going to use this function in our code

```
isSmallThenTen(9)
    .then(res => console.log('The number is smaller then 10'))
    .catch(err => console.log('The number is not smaller then 10'))
```

When we resolve a Promise the handler registered on `then` will get fired and if we reject the request then handler registered on `catch` will be called.

Let's look at an example which run an async operation

```
const timeoutIn = (time) => {
   return new Promise((resolve, reject) => {
      setTimeout(() => resolve(time), time)
   })
}

timeoutIn(2000)
   .then(res => console.log(`Resloved in ${res/1000} seconds`))
```

We defined a function `timeoutIn` which takes an argument time and resolves the promise after the time passed in using setTimeout. If you run this code You will see that our message will show after 2 seconds. That is because we passed in 2000 milliseconds as the argument and because of that promise resolved after 2000 milliseconds.

Promise.all

Sometimes we are in a situation where we have to do multiple async operations and wait till all the operations are completed before executing any other code. In the past most of the time we use the callback to handle async code and with callback, it is very difficult to do something like this.

Promises have a new method `.all` which is created just to handle this problem.

Promise.all takes an array of unresolved promises and resolve them one by one. If all the promises resolved correctly Promise.all trigger its `then` handler with an array of resolved values in the same order as promises passed in. If any of the promises get rejected then Promise.all call its `catch` handler and stops the resolving.

Here is an example of Promise.all

```
const timeoutIn = (time) => {
   return new Promise((resolve, reject) => {
      setTimeout(() => resolve(time), time)
   })
}
```

```
const timeoutInArr = [
    timeoutIn(2000),
    timeoutIn(4000),
    timeoutIn(8000)
]

Promise.all(timeoutInArr)
    .then(values => {
        console.log('All the promises are resolved now you can run your code', values
        // your code
    })
```

We created an array of unresolved promises and passed it into the Promise.all method once all the promises get resolved we output the values on console.

Here is a basic explanation of ES6 Promises. There is a lot more about Promises you should learn about. There are lot of articles out there just google them.