ES2015 Promises

Monday, October 12, 2020 3:19 PM

- It is not a NodeJS feature but available in general in all javascript programming
- **Evolved much later**
- **NodeJS was already using its own model of programming**

- Many Nodejs libraries are now slowly moving to Promise rather than node callbacks

A Promise

- It's a bulit in ES2015 (Javascript feature)
- **Promise is an object that promises to get some result in future**
  - **Promise also take a callback with two parameters**
    - **These two parameters are again call backs**
      1. **To call when success**
      2. **To call when failed**

- Promise to get a result asynchronously by calling another function

Promise says let me run this code and I will let you know when we are ready

```
let promise= new Promise(   function_that_will_give_you_a_result );
```

```
function  function_that_will_give_you_a_result(   fnResolve, fnReject ){

      ...
      if(success)
            fnResolve( result);   //call when you completed successfully
      else
            fnReject(err_details); //call this function when you fail


}
```

This is your business logic

Creating an api — callback vs Promise

```
function findPrimes(   min, max, cb ){
      //business logic
      ...
      if(success)
            cb( null, result);
      else
            cb(err_details);


      //This function returns nothing
}
```

No callback passed.

```
function findPrimes(   min, max ){

      let promise=new Promise( (resolve,reject) =>{
            //business logic here
            if(success)
                  resolve(  result);
            else
                  reject(err_details);

      }
      return promise;
}
```

We handle promise once returned

Consuming The Asynchronous operations

```
//callback example
findPrimes( 2, 100 ,   (err,primes) =>{

      if(err){
            console.log('err',err); //on failure
      } else{
            Console.log('primes', primes.length);  //on success
      }

});

//we are free to do whatever we want
//the callback will be called sometimes in future
//same callback will get both err and result
```

```
//promise based design

//function doesn't return result. It returns a future promise
let promise= findPrimes(2,100);

//we can set for future when it completes
//if promise is resolved successfully
promise. then( primes=> console.log('primes', promes.length);

//if promise is rejected because of error
promise.catch( err => console.log( 'err', err);


//we can do whatever we want to do. then() and catch() will
```

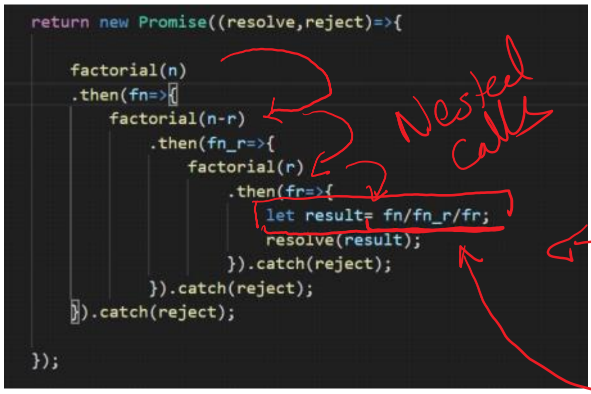
execute asynchrnously when promise is resolved/rejected in future.

//this code will execute immediately.

Promises can
Be chained

```
findPrimes(2,100)
        .then(primes=> console.log(primes))
        .catch(err=>console.log(err);
```

Nested Promise Problem



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

    factorial(n)
    .then(fn=>{
        factorial(n-r)
        .then(fn_r=>{
            factorial(r)
            .then(fr=>{
                let result= fn/fn_r/fr;
                resolve(result);
            }).catch(reject);
        }).catch(reject);
    }).catch(reject);

});
```

1. Calculate factorial n
2. Calculate factorial of n-r
3. Calculate factorial of r
4. Use the first 3 calculation to calculate combination

- Can you see the sequence in nested promise?

This calculation depends on all the three

Async - Await Keywords

- Since Promise is a javascript feature, javascript has defined a set of keywords that makes working with Promise easy and straight forward.
- **await** is a javascript keyword that automatically resolves the promise and give you resolved result rather than promise
    - Remember this result will not come immediately but sometimes in future
- When you use **await**, the rejection is thrown as an exception that can be handled using standard **catch** keyword
- The function is actually waiting for resolved/rejected, but will finish immediately asynchrnously
    - It will execute the code later.

Manual Promise Resolution

```
function testFactorial(n){

    let p = factorial(n); //it returns a promise

     //wait for promise to complete and get resolved result
    p.then( fn => console.log(fn));

    //if promise is rejected you get rejection message
    p.catch( err=> console.log('err',err) );


}
```

Using await

- Function having await must be marked **async**
- An async function always returns a Promise implicilty

```
async function  testFactorial(n){

    try{
        //await will wait for promise resolution.
        let fn = await factorial(n);  //taking n*100ms

        //next piece of code is what you would write in then, to be executed in
        future
        console.log('result is ',fn);    //typically what you write in then

    }
    catch(err){ //rejection is handled in catch
        console.log('err',err) ; //what you write in .catch()
    }


}
```

1. Looks like this code is synchrnous. But actually there may be long gap between these two lines
2. This code may run in future but the function will return immediately

Anything that follows await will be executed later and therefore this function creates a Promise and returns immediately