Day 13

Wednesday, 24 January 2024 12:26 AM

Asynchronous Java Script

Script that runs parallelly scripts in the Web Api's

Will execute only when the call stack is empty means at the end or after the compilation of JS code

Methods → Promises → Manage callbacks more efficiently using resolve & reject

Set Interval
↓
runs a single code multi times after a given time interval

setInterval (anyfunction , interval)

Clear Interval → to clear the reoccurring caused by the SetInterval.

Set Timeout
↓
executes a particular script after a given interval of time

set Stoppage (function/any, interval)

```
const myPromise = new Promise ( function
    ( response, reject ) {
    if condition {
        response ("Your Output")
    }
    else { reject ("Your output in rejection")
    }
```

in case of True event response ↑

```
myPromise.then ( function ( value ) {
    console.log (value) }
  .catch ( function (value) {
    console.log (value) }
```

↓ in case of reject

Call back Hell & Pyramid of Doom concepts

occurs in case of Multiple Callbacks scenario (occurs wherever there are multiple set methods or callback were used in a single process)

```
setTimeout( function (){
    setTimeout (function() {
        setTimeout( function ()
        {
            setTimeout( function(){
            },1000)
        },1000)
    },1000)
},1000)
```

This type of function causes difficulties in the complex web page

difficulties
web page

}, 1000)
}, 1000)

Conversion of Pyramid of Doom Concept to an optimize code

Some Key Points:

- everytime Event Loop will first execute the Micro Callback queue (Promises) then the normal callback queue (setTimeout) (Promises > setTimeouts)
- .then method will return the entire promise itself. (& You can also chain it when you use the return statement with it)

CHAINING OF PROMISES

```
example → const MyPromise1 = Promise.response ("completed")      OP →
MyPromise1.then (value => { console.log (value)                  → completed
                  value = value + 'task')
                  return value
                                                                 → completed task
  .then (value => { console.log (value)
                  value = value + 'task'
                  return value}
                                                                 → completed task task
  .then (value => { console.log (value)
                  }
```

Solving Pyramid of Doom Concept

```
function RidofCallback ( ) {
  return new Promise ( function(any,
                      interval }

RidofCallback . then ( function (value) {
    return RidofCallback()
  }
  ... function (value) {
          [illegible]                  }

. then ( function (value) {
            return Rid of (callback)
                 }
   . then ( function (value) {
           return Ridof(callback1)
                    }
```