# Promise

Support **chained calls and no Callback hell**

两个函数类型的参数

//resolve func(success)

//reject func(fail)

const p = new Promise((resolve, reject) => {

    setTimeout(() => {

        let n = rand(1, 100);

        if (n <= 30) {

            resolve(n);//set the state inside the Promise Object to [successful]

            //可以传参数(value)进去resolve(), then is able to use this parameter(1st func parameter in the p.then(...))

        }else {

            reject(n); //set the state inside the Promise Object to [fail]

            //可以传参数(reason)进去resolve(), then is able to use this parameter(2nd func parameter in the p.then(...))

        }

    }, 1000)

})

p.then((value) => {

    alert("ok with num" + value); //成功调用第一个函数, set status of promise obj to success

}, (reason) => {

    alert("good luck next time" + reason); //失败的话调用第二个函数, set status of promise obj to fail

})

# Promise states

A property in the instance promise which is obj.PromiseStata

* pending: initial state, neither fulfilled nor rejected.
* fulfilled: meaning that the operation was completed successfully.(resolved)
* rejected: meaning that the operation failed.

Its either pending -> fulfilled(result is normally value) or pending -> rejected(result is normally reason) and can only change one time.

# Promise result

Another property called PromiseResult which contains the result of fulfilled or rejected for a async task.

# Flow of Promise

Diagram

Description automatically generated

Diagram

Description automatically generated

# How to change the state of promise

* resolve() // pending -> fulfilled
* reject() // pending -> rejected
* throw error // pending -> rejected

# when there is more than one callback functions(ok or rejected) for the promise(ie. Serval then(…)) , will it all been called??

Yes, it will call all callback functions once there is a state change.

# Order of change of promise state and specify callback functios (for success or reject)

* any order is possible.

If sync function, change of state first and then specify callback.(1.executor directly call resolve()/reject() 2. Delay more time to call then(…))

If async with delay, probably specify call back then change state.

* When to get data?(when to call specified callback functions)

1. If specify callback function first(then you call p.then(…) state in promise object not changed yet), there it will call specified callback functions once the state of promise changed.
2. If the state of promise changed first, it will call specified callback functions when specifying.(when u define p.then(…))

//Promise API

//1. constructor

//Promise(executor){}

const { resolve } = require("path")

//executor function : (resolve, reject) =>{...}

let p = new Promise((resolve, reject) =>{

    //called when define

    console.log("同步调用")

})

//2. .then() method with 2 callbacks

// (onResolved, onRejected) => { }

//onResolved function 成功时回调(value) => {...}

//onRejected function 失败时回调(reason)=> {...}

//3. catch with 1 callback

//(onRejected) => {...}

//onRejected function 失败时回调(reason)=> {...}

//Promise.resolve() method belong Promise function object instead of instance

let p1 = Promise.resolve(521)

//if the parameter is a non-promise type object, then the returned result is a successful promise object

// whose content is that parameter that we input earlier on

let p2 = Promise.resolve(new Promise((resolve, reject) => {

    resolve("ok");

    //reject("Error")

}))

// if the parameter is a promise type object, then the result of parameter determined the result of the resolve

//Promise.reject method belong Promise function object instead of instance

//返回一个失败的Promise forever

let p = Promise.reject(521)

let p2 = Promise.reject("123")

let p3 = Promise.reject(new Promise((resolve, reject) => {

    resolve("ok")

}))

console.log(p3)// will return a failed promise whose result is the successful Promise object(parameter)

**//Promise.all**

//include an arrary containing n promises object

//will return a successful promise which contains the result of all promises in the array when all promises are successful

//will return a failed promise object when there is a promise in the array failed whose returned result will be the final result of promise.all

let p1 = new Promise((resolve, reject) => {

    resolve("ok")

})

let p2 = Promise.resolve("success")

let p3 = Promise.resolve("success2")

const result = Promise.all([p1, p2, p3])

console.log(result)//

result\_ok = {

    PromiseState:"fulfilled",

    PromiseResult:[

        "OK",

        "success",

        "success2"

    ]

}

// if p2 = Promise.reject("Error")

result\_err = {

    PromiseState:"rejected",

    PromiseResult:"Error"

}

//**Promise.race**

//same as Promise.add whose parameter is an array which contains n promise objects

//it will return a result of a promise who first returned the value

result = Promise.race([p1, p2, p3])

console.log(result) //p1 result

//if set a SetTimeOut for p1

//then it will return the result of p2

# The returned result of then(…)

let p = new Promise((resolve, reject) => {

    return resolve("ok")

})

let result = p.then(value => {

    console.log(value)//fullfilled + ok

    //1. throw error

    throw 'error' // rejected  + error

    //2. return non-promise object

    return 521 // fulfilled + 521

    //3. return promise object

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

        //resolve('success') -> fullfilled + success

        reject('error') // -> rejected + error

    })

}, reason => {

    console.warn(reason)

})

console.log(result) // result is a new promise object

then(…) 返回一个promise对象， 其结果由then(…)指定的回调函数的执行结果决定()

* If error thrown, new promise will become rejected with reason of that error
* If a non-promise object is returned by callbacks, new promise become resolved + value returned
* If return a new promise, the result of that promise will be our final result

# 多个任务的串联

//chained callbacks

let p = new Promise((resolve, reject) => {

    setTimeout(() =>{

        resolve("ok")

    }, 1000)

})

p.then(value => {

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

        resolve('success')

    })

}).then(value => {

    console.log(value) //success

}).then(value => {

    console.log(value) //undefined

})

原因then(...)的返回结果是一个promise， the state of promise determined by the result of its specified callback functions

因为上一个then(...) 没有返回值，所以其结果为undefined

# 异常穿透

You can specify .catch(reason => {}) in the end when using the chained then(…), it will catch the first rejected promise and call the callback functions defined within catch(…)

最后指定一个catch

# Stop/interrupt (中断) Chained Promise

The only way: Return a promise object which state is pending

Return new Promise(() => {})

# Async function

返回的结果是一个promise 对象， 对象的状态以及result跟then(…)的规则一致

A screenshot of a computer

Description automatically generated with medium confidence

# Await

Text, letter

Description automatically generated

