Seção 08 Javascript assíncrono – Promises, Ajax, Fecth e Async/Await

* Aula 86 Promises

function rand(min, max) {

    min \*= 1000;

    max \*= 1000;

    return Math.floor(Math.random() \* (max - min) + min);

}

function esperaAi(msg, tempo){

    setTimeout( () => {

        console.log(msg);

    }, tempo);

}

esperaAi('Frase 1', rand(1, 3));

esperaAi('Frase 2', rand(1, 3));

esperaAi('Frase 3', rand(1, 3));

aqui cada função esperai() levará um tempo para ser executada.

Para resolver isso vamos utilizar as promises.

Uma ação so será utilizada depois que a anterior tiver acabado.

function esperaAi(msg, tempo){

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

        setTimeout( () => {

            resolve(msg);

        }, tempo);

    });

}

Essa e uma função que retorna uma promessa.

Para que isso ocorra e nescessario chamar o new Promise(res,rej).

Resolve -> tente fazer isso, caso de certo use resolve

Reject -> tente fazer isso caso de errado use reject

esperaAi('Conexão com o BD', rand(1, 3))

.then(resposta => {

    console.log(resposta);

    return esperaAi('Buscando dado da base', rand(1, 3));

})

.then(resposta => {

    console.log(resposta);

    return esperaAi('Tratando os dados da base', rand(1, 3));

})

.then(resposta => {

    console.log(resposta);

})

.then(() => {

    console.log('Exibi os dados na tela');

})

.catch();

Aqui fica o complemento do código.

O uso do .then, indica uma promessa para esperar que deu certo.

O uso do .catch, indica que uma promessa que deu erro, se na cadeira ocorre um erro você vai direto pro catch.

Aqui vamos ver caso o tipo de msg for string ele ira cair no reject

function esperaAi(msg, tempo){

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

        if(typeof msg !== 'string'){

reject('BAD VALUE');

return;

};

        setTimeout( () => {

            resolve(msg);

        }, tempo);

    });

}

esperaAi('Conexão com o BD', rand(1, 3))

.then(resposta => {

    console.log(resposta);

    return esperaAi('Buscando dado da base', rand(1, 3));

})

.then(resposta => {

    console.log(resposta);

    return esperaAi(222222222222, rand(1, 3));

}).then(resposta => {

    console.log(resposta);

})

.then(() => {

    console.log('Exibi os dados na tela');

})

.catch(e => {

    console.log('ERRO', e);

});

Todo o resolve vai cair no then.

Todo o reject cai no catch.

* Aula 87 Metodos uteis para promises

function rand(min, max) {

    min \*= 1000;

    max \*= 1000;

    return Math.floor(Math.random() \* (max - min) + min);

}

function esperaAi(msg, tempo){

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

        if(typeof msg !== 'string') {

reject('BAD VALUE');

return;

}

        setTimeout( () => {

            resolve(msg.toUpperCase() + ' - Passei na promise');

        }, tempo);

    });

}

// Promise.all Promise.race Promise.resolve Promise.reject

const promises = [

    'Primeiro valor',

    esperaAi('Promise 1', 3000),

    esperaAi('Promise 2', 500),

    esperaAi('Promise 3', 1000),

    'Outro Valor'

];

Promise.all(promises)

.then(function(valor){

    console.log(valor);

})

.catch(function (erro){

    console.log(erro);

});

Ele vai resolver todas as promesas que tem dentro de promises,se uma de errado ele rejeita todas

Promise.race(promises)

.then(function(valor){

    console.log(valor);

})

.catch(function (erro){

    console.log(erro);

});

Nesse ele vai retorna sempre o primeiro valor. Como numa corrida.

Ele ira executar todas as tarefas, porem so ira retornar o primeiro valor o valor mais rápido

function baixaPagina() {

    const emChache = true;

    if(emChache) {

        return Promise.resolve('Pagina em chache');

    } else {

        return esperaAi('Baixei a pagina', 3000);

    }

}

Retorna uma promessa resolvida ou rejeitada de cara.

* Aula 88 Async/Await

function esperaAi(msg, tempo) {

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

        setTimeout(() => {

            if(typeof msg !== 'string') {

                reject('CAI NO ERRO');

                return;

            }

            resolve(msg + ' - Passei na promise');

            return;

        }, tempo);

    });

}

esperaAi('Fase 1',rand())

.then(fase => console.log(fase));

Ao usar uma função que retorna uma promessa, ao chamar ela e nescessario utilizar a palavrinha .then().

A função acima esperaAi(), esta retornando uma new Promise, logo quando ela e chamada e nescessario utilizar a palavra .then()

esperaAi('FASE 1', rand())

.then(valor => {

    console.log(valor);

    return esperaAi('FASE 2', rand());

})

.then(fase => {

    console.log(fase);

    return esperaAi('FASE 3', rand())

})

.then(fase => {

    console.log(fase);

})

.catch(e => console.log(e))

Aqui esta função esta encadeada, ela retorna uma nova função que também e uma promissa, para cada new Promise e nescessario utilizar a palavra .then(valor => {console.log(valor)}), para que seja mostrado o que foi retornado, caso haja algum erro em alguma promessa o código cairá no catch.

esperaAi('FASE 1', rand())

.then(valor => {

    console.log(valor);

    return esperaAi('FASE 2', rand());

})

.then(fase => {

    console.log(fase);

    return esperaAi('FASE 3', rand())

})

.then(fase => {

    console.log(fase);

    return fase;

})

.then(fase => {

    console.log('Terminamos na fase', fase);

})

.catch(e => console.log(e))

Porem esse código fica muito verboso para ler

Para resolver essa questão de verbosidade ultilizamos as palavras Async/Await

async function executa() {

    await esperaAi('FASE 1', rand())

}

Basta criar uma função async(assincrona), e dentro da função colocar a função que deverá ser executada, que irá esperar.

async function executa() {

    const fase1 = await esperaAi('FASE 1', rand());

    console.log(fase1);

}

executa();

para pegar o valor desse await basta jogala dentro de uma variável.

E ao final executar a função async

async function executa() {

    const fase1 = await esperaAi('FASE 1', rand());

    console.log(fase1);

    const fase2 = await esperaAi('FASE 2', rand());

    console.log(fase2);

    const fase3 = await esperaAi('FASE 3', rand());

    console.log(fase3);

console.log('Terminamos na:', fase3);

}

executa();

a função executa(), e uma função async que espera a primeira função esperaAi() ser completamente executada, para executar a segunda função esperaAi(), que tem que ser completamente executada para por fim ir para a ultima função esperaAi().

Cada função so e executada quando a anterior e completamente finalizada.

Nesse caso caso queira usar o cacth, basta englobar todas em um try/catch

async function executa() {

    try {

        const fase1 = await esperaAi('FASE 1', rand());

        console.log(fase1);

        const fase2 = await esperaAi(333, rand()); <- ERRO

        console.log(fase2);

        const fase3 = await esperaAi('FASE 3', rand());

        console.log(fase3);

        console.log('Terminamos na:', fase3);

    } catch(e) {

        console.log(e);

    }

}

executa();

// FASE 1 - Passei na promise

// CAI NO ERRO

ele vai exetutar todas as promessas, ele vai executar ate acha um erro, caso alguma de errado seja achado, ele ira cair no catch.

As promisses possuiem três estados:

Pendente – quando uma função está sendo executada e ainda não retornou o valor.

const fase2 = esperaAi('FASE 2', rand());

console.log(fase2);

// Promise { <pending> }

Essa função esperaAi(), retorna uma new Promise, e logo em sequida ela e chamada no log, não dando tempo de ser completada, tornando assim uma promessa pendente.

Fullfill – quando está completamente finalizada

const fase2 = await esperaAi('FASE 2', rand());

console.log(fase2);

// FASE 2 - Passei na promise

Rejected – quando a promessa e rejeitada

const fase2 = await esperaAi(9876, rand());

console.log(fase2);

// CAI NO ERRO

A função esperaAi recebe como argumentos(mensagem do tipo string, e tempo).

function rand(min = 0, max = 3) {

    min \*= 1000;

    max \*= 1000;

    return Math.floor(Math.random() \* (max - min) + min);

}

function esperaAi(msg, tempo) {

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

        setTimeout(() => {

            if(typeof msg !== 'string') {

                reject('CAI NO ERRO');

                return;

            }

            resolve(msg + ' - Passei na promise');

            return;

        }, tempo);

    });

}

async function executa() {

    try {

        const fase1 = await esperaAi('FASE 1', rand());

        console.log(fase1);

        const fase2 = await esperaAi(9876, rand());

        console.log(fase2);

        const fase3 = await esperaAi(3333, rand()); // <- ERRO

        console.log(fase3);

        console.log('Terminamos na:', fase3);

    } catch(e) {

        console.log(e);

    }

}

executa();

resumo, a função executar, e uma função async, que tenta executar a função esperaAi, que e uma função de espera, uma função await, que caso de algum erro e direcionado ao catch da função.