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.

* Aula 89 XMLHttpRequest(GET) + Promises

const request = obj => {

  const xhr = new XMLHttpRequest();

  xhr.open('GET', 'U', true);

}

GET – Verbo/metodo http que queremos, get ou set, get representa você buscar algum conteúdo da internet.

U – Url, http:algumacoisa.com

Síncrono ou assíncrono – para ser assíncrono sempre true.

Ou você pode pegar os parâmetros desse objeto.

const request = obj => {

  const xhr = new XMLHttpRequest();

  xhr.open(obj.method, obj.url, true);

}

Ficando assim.

Se fosse post no verbo, deveria ter o .send(), contendo os dados que iriam ser enviados.

const request = obj => {

  const xhr = new XMLHttpRequest();

  xhr.open(obj.method, obj.url, true);

xhr.send();

}

Agora precisamos verificar quando essa requisição terminol.

Para isso iremos fazer

 xhr.addEventListener('load', () => {

  })

Criamos um evento de load no xhr.

xhr.addEventListener('load', () => {

    // Lista de código http

    // código entre 200 e 300 = requisição de sucesso

    // de 300 para cima = consideramos um erro

    if(xhr.status >= 200 && xhr.status < 300) {

      obj.success(xhr.responseText);

    } else {

      obj.error(xhr.statusText);

    }

  })

Aqui quando carregado verifica qual o código http recebeu

document.addEventListener('click', e => {

  const el = e.target;

  const tag = el.tagName.toLowerCase();

  if (tag === 'a') {

    e.preventDefault();

    carregaPagina(el);

  }

});

Aqui esta monitorando cada ação de clique na página, caso sea um link, executa uma ação.

function carregaPagina(el) {

  const href = el.getAttribute('href');

  request({

    method: 'GET',

    url: href,

    success(response) {

      carregaResultado(response)

    },

    error(errorText) {

      console.log(errorText);

    }

  })

};

const request = obj => {

  const xhr = new XMLHttpRequest();

  xhr.open(obj.method, obj.url, true);

  xhr.send();

  xhr.addEventListener('load', () => {

    if(xhr.status >= 200 && xhr.status < 300) {

      obj.success(xhr.responseText);

    } else {

      obj.error(xhr.statusText);

    }

  })

};

document.addEventListener('click', e => {

  const el = e.target;

  const tag = el.tagName.toLowerCase();

  if (tag === 'a') {

    e.preventDefault();

    carregaPagina(el);

  }

});

function carregaPagina(el) {

  const href = el.getAttribute('href');

  request({

    method: 'GET',

    url: href,

    success(response) {

      carregaResultado(response)

    },

    error(errorText) {

      console.log(errorText);

    }

  })

};

function carregaResultado(response) {

  const resultado = document.querySelector('.resultado');

  resultado.innerHTML = response;

}

Esse e o modelo de callback para fazer uma requisição XML

Outra forma de fazer:

const request = obj => {

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

    const xhr = new XMLHttpRequest();

    xhr.open(obj.method, obj.url, true);

    xhr.send();

    xhr.addEventListener('load', () => {

      if(xhr.status >= 200 && xhr.status < 300) {

        resolve(xhr.responseText);

      } else {

        reject(xhr.statusText);

      }

    })

  });

};

document.addEventListener('click', e => {

  const el = e.target;

  const tag = el.tagName.toLowerCase();

  if (tag === 'a') {

    e.preventDefault();

    carregaPagina(el);

  }

});

function carregaPagina(el) {

  const href = el.getAttribute('href');

  const objConfig = {

    method: 'GET',

    url: href

  };

  request(objConfig).then(response => {

    carregaResultado(response);

  }).catch(error => console.log(error))

}

function carregaResultado(response) {

  const resultado = document.querySelector('.resultado');

  resultado.innerHTML = response;

}

Que pode ser melhorada dessa maneira a seguir:

const request = obj => {

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

    const xhr = new XMLHttpRequest();

    xhr.open(obj.method, obj.url, true);

    xhr.send();

    xhr.addEventListener('load', () => {

      if(xhr.status >= 200 && xhr.status < 300) {

        resolve(xhr.responseText);

      } else {

        reject(xhr.statusText);

      }

    })

  });

};

document.addEventListener('click', e => {

  const el = e.target;

  const tag = el.tagName.toLowerCase();

  if (tag === 'a') {

    e.preventDefault();

    carregaPagina(el);

  }

});

async function carregaPagina(el) {

  const href = el.getAttribute('href');

  const objConfig = {

    method: 'GET',

    url: href

  };

  try {

    const response = await request(objConfig);

    carregaResultado(response);

  } catch(e) {

    console.log(e);

  }

}

function carregaResultado(response) {

  const resultado = document.querySelector('.resultado');

  resultado.innerHTML = response;

}

Finalizando o código, porem esse tipo de código não e mais tao utilizado hoje me dia por ser mais verbosso, para contornar isso pode ser utilizar o fetch API.

* Aula 90 Fetch API (GET)

Fetch(caminho\_do\_aquivo/ou\_url) que quer carregar

fetch('pagina1.html');

// essa fetch, já retorna por padrão uma promisse

// essa fetch, já retorna por padrão uma promisse

fetch('pagina1.html')

.then(resposta => {

  if(resposta.status !== 200) throw new Error('ERRO 404 NOSSO');

  return resposta.text();

})

.then(html => console.log(html))

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

document.addEventListener('click', e => {

  const el = e.target;

  const tag = el.tagName.toLowerCase();

  if (tag === 'a') {

    e.preventDefault();

    carregaPagina(el);

  }

});

async function carregaPagina(el) {

  try {

    const href = el.getAttribute('href');

    const response = await fetch(href);;

    if(response.status !== 200) throw new Error('ERRO!!!');

    const html = await response.text();

    carregaResultado(html);

  } catch(e) {

    console.log(e);

  }

}

function carregaResultado(response) {

  const resultado = document.querySelector('.resultado');

  resultado.innerHTML = response;

}

* Aula 91 Fetch APIe Axion (JSON)

fetch('pessoas.json')

  .then(resposta => resposta.json())

  .then(json => carregaElementosNaPagina(json));

// axios('pessoas.json')

//   .then(resposta => carregaElementosNaPagina(resposta.data));

function carregaElementosNaPagina(json) {

  const table = document.createElement('table');

  for(let pessoa of json) {

    const tr = document.createElement('tr');

    let td1 = document.createElement('td');

    td1.innerHTML = pessoa.nome;

    tr.appendChild(td1);

    let td2 = document.createElement('td');

    td2.innerHTML = pessoa.idade;

    tr.appendChild(td2);

    table.appendChild(tr);

  }

  const resultado = document.querySelector('.resultado');

  resultado.appendChild(table);

}