JavaScript + PHP

Cliente JS

Tiago Lopes Telecken
telecken@gmail.com

JS - Assíncrono

- Normalmente o código java script é síncrono onde uma linha é executada por vez e uma linha só é executada quando a anterior termina
- Porém, para alguns comandos como requisição a rede e arquivos, o js não espera a resposta vir. Ele dispara a requisição segue os comandos seguintes e quando a requisição volta ele a trata (normalmente como um evento)
- Este comportamento, em algumas situações pode causar erros ou impedir a execução de tarefas. Para solucionar este problema o java script permite alguns controles/facilidades para gerenciar códigos assíncronos. Por exemplo pode-se forçar que uma requisição termine para que outro comando seja executado…(usando-se o async, await)

Fetch, async, await

```
// Ver arquivo fetchArquivoTexto.html
async function buscaTexto() {
        const painel = document.querySelector('#mostra');
        const url="./texto.txt";
        let resposta=  await fetch(url);
        let texto = await resposta.text();
        painel.innerHTML = texto;
    }
buscaTexto();
```

Fetch()

- A função buscaTexto é chamada. O comando **async** informa que esta é uma função assincrona. Ela será executada numa nova thread enquanto a thread inicial segue rodando
- O **fech** faz uma requisição para um servidor. Seu parâmetro é a url da requisição
- Ao terminar a requisição o fech retorna a resposta da requisição. Esta resposta foi armazenada no objeto resposta.
- O **await** faz com que o processamento desta thread só prossiga depois que a resposta chegar
- Depois a resposta da requisição é transformada em texto pelo método text(). O texto é armazenado na variável texto
- O segundo await faz com que o processamento desta thread só prossiga depois que a transformação terminar
- O await só funciona em uma função com async

Fetch – tratando erros com try, catch

```
// Ver arquivo fechCatchTexto.html
url=”./texto.txt”;
try{
        let resposta=  await fetch(url);
        let texto = await resposta.json(); //vai gerar um erro
        painel.innerHTML = texto;
    } catch (err){
        console.log(err);
    }
```

- O fetch pode ser colocado em um try. Se algum erro ocorrer no try o fluxo é interrompido e o que está no catch é executado. Ao chamar o catch é enviado um objeto com as informações do erro

JSON

- JSON (JavaScript Object Notation) é a notação de objetos e arrays do JavaScript.
- Informações podem ser armazenadas neste formato em arquivos, BD ou criadas e transferidas pela rede.
- Quando armazenadas ou transferidas os objetos/informações estão no formato de uma string.
- No caso de pgs html com JavaScript estas strings são requisitadas e ao chegarem são transformadas em objetos literais javascript (e/ou arrays, objetos com arrays, arrays de objetos…)
- O contrário também ocorre. Objetos JS são transformados em uma string para serem armazenados, transferidos ou mostrados (utiliza-se a função JSON.stringify)

JSON

```
{
  "Nome": "Jose",
  “Idade": 32,
  "Empregado": true,
  “Endereco": {
    “Rua": "701 Av. Brasil.",
    "Cidade": “Rio Grande, RS",
    “Pais": “Brasil"
  },
  “Filhos": [
    {
      "Nome": "Ricardo",
      " Idade ": 7
    },
    {
      "Name": "Susan",
      " Idade ": 4
    },
  ]
}
```

Fetch() - Json

```
//Ver arquivo fechJson.html

    let resposta=  await fetch(url);
    let js = await resposta.json();
    painel.innerHTML = JSON.stringify(js);
```

- Aqui a resposta (um arquivo com uma string Json) é transformada em um objeto Javascript pelo método json()

  Depois o objeto é transformado em uma string e colocado na tela com o método JSON.stringify();

Fetch() - Json

```
//Ver arquivo fechJson2.html
let resposta=  await fetch(url);
    let js = await resposta.json();

    let texto="<h1>Cidades de Santa Catarina</h1>";
    js.SC.forEach(cidade => {
            texto=texto + cidade.nome +"<br>";
        });
    texto= texto + "<h1>Capital de Santa Catarina</h1>";
    texto=texto + js.SC[0].nome;
    painel.innerHTML = texto;
```

- Aqui o objeto json obtido é armazenado em js e acessado para se colocar dados na tela.
- Para percorrer os dados de js são utilizadas as técnicas para manipulação de objetos literais (visto na aula de objetos literais)

Fetch() - Buscando imagem - blob

```
    // Ver arquivo fetchImagem.html
    const url="./nave.jpg";
    let resposta=  await fetch(url);
    let imagem = await resposta.blob();
    const imageObjectURL = URL.createObjectURL(imagem);
    let img = document.createElement('img');
    img.src = imageObjectURL;
    painel.appendChild(img);
     });
});
```

- Aqui o fech chama uma imagem. Quando a imagem chega a função blob transforma a requisição em formato binário e o armazena em imagem
- Para mostrar a imagem cria-se uma url temporária e depois insere-se uma tag img (com o atributo src apontando para a url temporária)no html.

Fetch()

- O fech pode fazer uma requisição (request) para qualquer tipo de item (texto, json, imagem, vídeo,…)
- Após fazer a requisição vem uma resposta (response ou promisse) que pode conter qualquer tipo de dado (texto, json, imagem, vídeo,…) ou um erro
- Ao chegar uma resposta, deve-se tratá-la adequadamente
  - Primeiro converter a resposta para o correspondente formato (.text(),.json(),.blob(),…)
  - Depois manipular o item obtido adequadamente. Manipular imagem como imagem, json como json, etc…
- Assim antes de requisitar um item você já deve saber o que está requisitando, o que que vai chegar e como tratá-lo

Fetch()

- Nos exemplos até aqui o **fetch** fez requisições para itens no mesmo servidor da página. Informamos na url só caminhos relativos locais (./nome,../../nome)
- Porém o fetch também pode fazer requisições para itens que estão em outros servidores. A url pode apontar para qualquer item que esteja na internet (por ex: http://ifrs.edu.br/rg/notas.txt)
- **O fetch busca os itens nos servidores externos e o seu tratamento a partir dai é o mesmo dos itens buscados localmente**
- É importante saber que quando busca-se itens externos mais erros podem ocorrer (erros na rede, tempo de busca, firewalls…) assim mais situações devem ser previstas
- O servidor externo pode exigir mais informações na requisição para enviar a correta resposta (senhas, headers específicos, etc,…). Ou servidor pode disponibilizar o item e não compartilhá-lo para requisições de JS.

Fetch()- php,java,API, WS

- O fech também pode fazer requisições a páginas ou programas de um servidor (php, java, node…)
- Estas páginas criam um item (json, um txt, uma imagem…) dinamicamente (no momento em que é requisitada) e enviam este item para quem o requisitou...a partir dai pode-se tratar as requisições do mesmo modo que um item estático local
- Estas páginas são preparadas para receber fetchs (e outros tipos de requisições) e respondê-las adequadamente. Conforme sua estrutura podem ser chamados de API, Web Services...

Fetch() - API

- Para requisitar dados de uma api deve-se estudar, conhecer a API para saber como solicitar dados e como eles serão enviados
- A API dos correios por exemplo funciona da seguinte maneira: Faça uma fetch para o seguinte endereço
- https://viacep.com.br/ws/96201460/json/
- E o servidor vai enviar de volta os seguinte json com os dados sobre o cep solicitado (96201460)

{cep: "96201-460", logradouro: "Rua Engenheiro Alfredo Huch", complemento: "", bairro: "Parque Residencial Salgado Filho", localidade: "Rio Grande", …}

- Troque o numero do cep na url e dados deste outro cep serão enviados

Fetch() - API

```
//Ver arquivo fetchApi.html
const url='https://viacep.com.br/ws/96201460/json/';
let resposta=  await fetch(url);
let js = await resposta.json();
console.log(js);
```

- Aqui o fetch traz dados sobre o cep 96201460.
- Ao chegarem os dados são convertidos para json e na sequência são mostrados no console do JavaScript
- O arquivo fetchApi2.html mostra uma versão mais elaborada

Fetch(url,dados)

- Além da url o fetch pode ter outro parâmetro com dados da requisição
- Muitos itens da internet podem ser acessados com a configuração padrão destes dados. Entretanto alguns itens ou aplicações exigem configurações específicas
- Nestes casos deve-se colocar as novas configurações no segundo parâmetro

Fetch(url, dados)

```
const painel = document.querySelector('#mostra');
const url="./texto.txt";
const dados={method:’GET’, mode:’cors’, cache:’default’};

let resposta=  await fetch(url,dados);
let texto = await resposta.json(); //vai gerar um erro
painel.innerHTML = texto;
```

- Aqui tem um fetch que utiliza o segundo parâmetro para requisitar um arquivo texto

Fetch(url, dados)

```
// as opções default estão marcadas com *
Dados = {
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *cors, same-origin
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    credentials: 'same-origin', // include, *same-origin, omit
    headers: {
      'Content-Type': 'application/json'
      // 'Content-Type': 'application/x-www-form-urlencoded',
    },
    redirect: 'follow', // manual, *follow, error
    referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-downgrade, origin, origin-when-cross-origin, ...
    body: JSON.stringify(data) // body data type must match "Content-Type" header
  });
```

- Aqui tem um objeto Dados mais complexo com várias configurações bem como algumas alternativas de configurações.
- O uso e especificação deste objeto depende de cada requisição.
- Algumas API, por exemplo, informam como deve ser este objeto para que a requisição seja correta

Alternativa - then

```
// Ver arquivo fetchThenTexto.html
 const url="./texto.txt";
 fetch(url).then(function(response) {
  response.text().then(function(texto) {
      painel.innerHTML = texto;
  });
});
```

- Uma alternativa ao async await é o then
- Neste exemplo o Fech faz uma requisição para um servidor.
- Ao terminar a requisição, é executada a função que estiver no then(). Esta função recebe um parâmetro. É a resposta (***response***) da requisição. Neste exemplo a response é transformada em texto pelo método .text()
- Se a transformação deu certo é executada a função do segundo then(). Esta função recebe um parâmetro que é o texto da requisição.

Alternativa - Async, await ou then

- O then é uma alternativa mais compacta e boa para quem gosta de programar em “pipelines” (vários comandos em uma linha)
- Entretanto quando se tem muitos then com catch encadeados em um pipeline o código pode ficar ilegível
- Usando async await a sequência de comandos que serão executadas são colocadas na forma linha após linha (mais fácil de entender)
- Já com o then deve-se seguir a sequencia de um pipe, pulando de função para função dentro de um único comando (uma forma mais complexa de se entender o código)

Alternativa - XMLhttpRequest

- Também pode-se fazer requisições através do XMLhttpRequest (usado no ajax),

- Porém esta alternativa tende a se tornar obsoleta dando lugar ao fetch

JavaScript

Tiago Lopes Telecken
telecken@gmail.com