afecarvalho@ipt.pt

AJAX

Asynchronous JavaScript and XML

AJAX

- É uma *técnica de programação* que consiste na comunicação com um servidor, *em segundo plano*, para troca de dados.
- *Asynchronous* porque estas chamadas **não interferem** com o funcionamento da página (por exemplo, não recarregam a página e não bloqueiam o funcionamento desta)
- **Não é exclusivo às aplicações web!** Esta técnica de programação aplica-se a *quase* qualquer aplicação, desde aplicações móveis (Android, iOS), como web, como aplicações desktop clássicas.

Usos (além de limpar janelas)

- Obtenção de dados em *background*, para, por exemplo, atualizar um chat, cotações de bolsa, o estado de uma caixa de correio (nota: WebSockets são mais adequadas para isto hoje em dia)
- Para obter ou enviar dados em resposta a ações do utilizador:
  - Mostrar detalhes sobre um conteúdo quando o utilizador clica nele;
  - Enviar os dados de um formulário quando o utilizador o submete, e fazer alguma coisa com o resultado, *sem recarregar a página*;
  - Enviar ou receber ficheiros (via FormData).

afecarvalho@ipt.pt

JSON

“People quickly realized that the ‘X’ in ‘AJAX’ did not stand for ‘XML’; it stood for ‘JSON’”
Douglas Crockford

JSON

- Significa “JavaScript Object Notation”.
- *Subset* do JavaScript, que é usado para definir estruturas de dados, com os seguintes tipos de dados:
  - Objectos (`{ “propriedade”: “valor” }`)
  - Arrays (`[ 1, 2, 3, 4, 5 ]`)
  - Strings (`“Olá JSON”`)
  - Números (`1.23, -5, 10, 0`)
  - Null (`null`)
  - Booleanos (`true, false`)
- JSON válido é JavaScript válido! O contrário não é sempre verdade.

JSON

- JSON é atualmente dos meios de transporte de dados mais utilizado, em detrimento do XML.
- Razões:
  - Pode ser usado para os mesmos fins (transporte de dados em APIs, bases de dados, ficheiros de configuração, etc.)
  - Mais *leve* que o XML em termos de tamanho de dados (existe muito menos *cerimónia*)
  - Muito mais fácil de ler, escrever, e usar (especialmente em JavaScript)
  - Existe “melhor” suporte para vários tipos de dados (no XML, são tudo strings, e por vezes é necessário mais *cerimónia* para indicar o tipo do dado)

afecarvalho@ipt.pt

Exemplo de JSON


```
{
  "birth_date": "1968-08-05",
  "career": [
      {
        "text": "Colin McRae began his competitive career (...) disqualified him.",
        "title": "Early Career"
      }
  ],
  "death_date": "2007-09-15",
  "id": "colinmcrae",
  "introduction": "Colin Steele McRae, (...) Scottish Sports Hall of Fame.",
  "multimedia": {
      "images": [],
      "videos": []
  },
  "name": "Colin McRae",
  "nationality": "Scottish",
  "nickname": null,
  "racing_class_ids": [ "rally", "dakar" ],
  "records": {
      "championship_victories": 1,
      "first_race_win": "1993 Rally New Zealand",
      "race_victories": 25
  }
}
```

afecarvalho@ipt.pt

Ler e escrever JSON em JavaScript

```
// Uma string de JSON, a título exemplificativo,
// com valores de texto, numéricos, e booleanos.
var jsonString = '{ "hello": "world!", "answer": 42, "isJsonAwesome": true }';


// Ler JSON com o JSON.parse
var result = JSON.parse(jsonString);

console.log(result.hello); // world!
console.log(result.answer); // 42
console.log(result.isJsonAwesome); // true

result.fibonnacciNumbers = [1, 1, 2, 3, 5, 8, 13, 21];

result.embeddedObject = {
    name: 'André',
    age: null // none of your business!
};


// JSON a partir de um objecto!
var encodedJson = JSON.stringify(result);

console.log(encodedJson);//{"hello":"world!","answer":42,"isJsonAwesome":true,"fibonnacciNumbers":[1,1
,2,3,5,8,13,21],"embeddedObject":{"name":"André","age":null}}
```

ipt

afecarvalho@ipt.pt

JSON em JavaScript: 2 funções

- Ler: **JSON.parse(texto)**
  - Interpreta o texto e devolve o resultado (pode ser um array, objecto, número, ou outra coisa qualquer).
  - O texto pode ser proveniente de qualquer fonte de dados (AJAX, ficheiros, *user input*, etc.).
- Escrever: **JSON.stringify(valor)**
  - Converte o valor para uma string de JSON.
  - O JSON resultante pode ser depois ser usado no que for preciso; incluíndo convertê-lo de volta em objectos com o JSON.parse.

afecarvalho@ipt.pt

AJAX em JavaScript

XHR, $.ajax, fetch, e Promise

ipt

Opção 1: XMLHttpRequest (XHR)

- Usado apenas quando é necessário *controlo total* sobre o ciclo pedido-resposta
  - Exemplo: update de progresso, abortar pedidos, etc..
- Serve como a base para 99.9% das bibliotecas AJAX
  - $.ajax, Axios, Superagent, browser-request, etc.
- Pouco “ergonómico”, usá-lo para tarefas simples é trabalhoso.
  - Exemplo: O código inicial da “Discoteca / SpotIPT”.
- *Deprecated*, já existem alternativas melhores
  - Não significa que não se pode usar, XHR é demasiado importante para desaparecer!

Opção 2: $.ajax (e outras bibliotecas/frameworks)

- Opção usada por quase toda a gente que tem jQuery à mão e precisa de aplicar AJAX.
- Várias funções disponíveis:
  - $.ajax - A “base”, oferece o maior nível de configuração.
  - $.getJSON - Utilitário para obter logo JSON.
  - $.load - Ler HTML de um servidor e adicionar à página.
  - $.get/post - Utilitário genérico para chamadas GET/POST.
- Ver:
  - http://api.jquery.com/category/ajax/
  - http://api.jquery.com/category/ajax/shorthand-methods/

Opção 3: fetch

- O fetch() é uma *função global* que vem substituir o XMLHttpRequest.
- 100% assíncrono - todas as operações retornam objectos Promise (Promise - representação de uma operação assíncrona)
- Pode ser usado por Service Workers
  - (Service Workers são uma nova tecnologia que permite, entre outras coisas, que sites e aplicações web funcionem offline e que disponibilizem notificações aos utilizadores, oferecendo-lhes características de aplicações nativas)

afecarvalho@ipt.pt

Promise

Representação de operações assíncronas

Promise

- Mecanismo *standard* do JavaScript para descrever o resultado (eventual) de uma operação assíncrona.
- Descreve casos de sucesso (“resolve”) e de insucesso (“reject”).
- Permite associar *callbacks* (funções) a ser executadas em caso de sucesso (.then) e insucesso (.catch).
- Análogos:
  - Java: Future<T>
  - C#/.net: Task<TResult>

afecarvalho@ipt.pt

Exemplo 1 - Divisão assíncrona

```
// Função que faz uma divisão de forma assíncrona.

function divisaoAssincrona(numerador, denominador) {
    var resultado = new Promise(function (resolve, reject) {
        if (denominador === 0) {
            reject(new Error("O denominador não pode ser zero."));
        } else {
            resolve(numerador / denominador);
        }
    });

    return resultado;
}
```

afecarvalho@ipt.pt

Exemplo 1 - Continuação

```
// Uso simples: Chamar a função, associar
// um callback de sucesso (.then), e imprimir o resultado
divisaoAssincrona(5, 10)
    .then(function (resultado) {
        console.log(resultado); // 0.5
    });


// Uso simples: Chamar a função, associar um
// callback de sucesso (.then) e outro de erro (.catch)
divisaoAssincrona(10, 0)
    .then(function (resultado) {
        console.log(resultado); // (Nunca chega aqui)
    })
    .catch(function (erro) {
        console.error(erro); // Error: O denominador não pode ser zero.
    });
```

afecarvalho@ipt.pt

Exemplo: Várias operações encadeadas

```
// Exemplo: Concatenação de várias operações (através do .then)
// para desempenhar várias tarefas, uma de cada vez, em sucessão
divisaoAssincrona(5, 10)
    .then(function (resultado) {
        return resultado * 2;
    })
    .then(function (resultado) {
        console.log(resultado); // 1 ((5 / 10) * 2 = 1)
    })
    .catch(function (erro) {
        console.error(erro); // (Nunca deve chegar aqui)
    });
```

Promise: notas

- Quando se usa uma Promise numa função, é *boa prática* retornar essa Promise (através do return).
  - Isto permite que funções dependentes possam esperar pelo resultado, mesmo que o resultado seja “void”.
- Promises são *imutáveis*! O uso do .then e do .catch *não altera* o objecto original, cria sempre um novo!
  - Não *devem* fazer .then / .catch em separado, a não ser que reassignem o valor da variável onde está a Promise!
- Lembrem-se do Event Loop!

Mais exemplos

https://github.com/ipt-ti2-2017-2018/exemplos-ajax-e-promise

Disclaimer: Esta apresentação não é um anúncio publicitário ao AJAX (produto de limpeza)

Obrigado!

André Carvalho

Email: afecarvalho@ipt.pt