JQuery e AJAX

Requisições AJAX:

Dicionário:

Dicionários são estruturas que funcionam como Arrays, em um dicionário, o identificador pode ser qualquer valor.

Exemplo:

```
var parametros = {nome: "Paulo", idade: 33};
```

Quando acessarmos o valor usando parametros [“nome”], teremos o texto “Paulo”.

.get : requisição jQuery que aceita até três parâmetros.

→primeiro parâmetro: URL da requisição.
→segundo parâmetro(opcional): Informa os parâmetros a serem passados para o serviço.
→terceiro parâmetro(opcional): Passa um callback que recebe como parâmetro os dados que foram lidos do serviço.

Exemplo:

```
var parametros = {nome: "Caro Leitor"};
var servico = "http://livro-capitulo07.herokuapp.com/hello";
$.get(servico, parametros, function(data) {
alert(data);
}
);
```

Objeto jqXHR :

O jQuery nos oferece um objeto jqXHR, que é o modo curto que os desenvolvedores do jQuery acharam para nomear o jQuery XMLHttpRequest Object.

esse objeto pode intermediar e controlar toda a conversação entre o seu código JavaScript e o servidor que vai nos fornecer os dados.

O objeto jqXHR permite que sejam definidos callbacks para quando a requisição correu bem e para quando houve algum erro.

Usamos o evento **done** para quando tudo ocorrer bem e o evento **fail** para quando houver algum erro.

Exemplo:

```
$.get(servico, parametros)
.done(function(data) {
alert(data);
.fail(function(data) {
 alert(data.responseText);
});
```

JSON:

Existem diversas maneiras para enviarmos e recebermos listas, valores e objetos complexos, as mais comuns são XML e JSON.

JSON é uma forma de serializar, ou converter para texto, objetos JavaScript de modo que sejam legíveis.

A grande vantagem do JSON sobre qualquer outra forma de serialização é que o JavaScript lida com o formato de modo transparente, convertendo a informação em formato texto para um tipo nativo da linguagem, evitando o uso de bibliotecas e ferramentas que possam complicar o seu código.

No JSON temos 6 tipos de dados:

Textos: textos.

Números: Inteiros ou decimais.

Lógicos: True ou False.

Listas: Arrays JavaScript.

Objetos: Dicionários JavaScript.

Exemplo:

```
var Livro = {
"autor" : "Plínio Balduino",
"titulo" : "Dominando JavaScript com jQuery",
"ano" : 2013,
"editora" : "Casa do Código",
"usa_drm" : false,
"capitulos": [
        "Apresentação",
        "Refazendo uma loja virtual",
         "Adicionando JavaScript",
         "Um JavaScript diferente em cada navegador",
         "Simplifique com jQuery"
]
} ;
→Livro.titulo;
"Dominando JavaScript com jQuery"
→livro.capitulos[2];
"Adicionando JavaScript"
```

Um bom site para visualizar arquivos JSON é: JSON Editor Online: JSON editor, JSON formatter, query JSON

Juntando JSON e AJAX:

A única diferença é que usamos o método getJSON em vez de get, dessa forma o AJAX poderá ler o arquivo JSON que o servidor ira responder.

Rest:

O padrão HTTP utiliza de um **método** e um **recurso**, por métodos podemos entender o **get**, **post**, **put**, **delete** por exemplo. Existem outros métodos.

- **GET** para ler
- **POST** para atualizar
- **PUT** para incluir
- **DELETE** para excluir

O recurso se baseia em uma aplicação do servidor exemplo:

http://localhost:8080/save

http significa uma aplicação em um servidor seguindo o protocolo http.
localhost:8080 é o endereço do servidor.
save é o **recurso**.

Podemos utilizar o mesmo recurso para várias operações desde eu utilizemos métodos diferentes.

jQuery UI:

A biblioteca jQuery UI foi criada para permitir a rápida criação e utilização de componentes visuais, efeitos e temas, aproveitando toda a facilidade que o jQuery traz em relação a diferentes tipos e versões de navegadores.

A biblioteca jQuery UI deve também ser baixada e acrescentada no projeto.

https://jqueryui.com/download/

Partes do jQuery UI:

UI Core:
Contém o núcleo do jQuery UI. Todos os demais componentes dependem desta parte.

Interactions:
É a parte que dá suporte a coisas como drag-and-drop (quando você arrasta e solta algum componente pela tela), ordenar itens numa tabela usando o mouse e mesmo mudar o tamanho de algum elemento da tela se você desejar.

Widgets:
Widgets são componentes visuais que já vem com o jQuery UI. Dentre os muitos componentes disponíveis, você tem acesso a menu, abas, botões e janelas que só existem dentro da sua página.

Effects:
O jQuery já vem com alguns efeitos simples, mas o jQuery UI traz treze efeitos novos para que você possa impressionar o usuário.

Temas:

Para ajudar a deixar o seu design com aparência consistente e profissional, o jQuery UI permite o uso de temas.

Temas são conjuntos de fontes e cores que são aplicadas em todos os componentes criados pelo jQuery UI, e podem ser trocados e criados conforme a sua necessidade, fazendo com que toda a sua aplicação troque de aparência de uma vez só.

Você pode experimentar e criar temas usando uma ferramenta chamada Theme Roller, que está no endereço:

http://jqueryui.com/themeroller/

Auto-completando:

O Recurso de auto-completar um texto é muito conhecido e para ser feito através do jQuery UI deve-se informar quais atributos devem ser exibidos na lista abaixo da caixa de texto.

```
<div class=”ui-widget”>
        <label for=”linguagens”>Linguagem: </label>
        <input id=”linguagens” />
</div

var linguagens = [
```

```
 "Clojure",
 "Common LISP",
 "Erlang",
 "F#",
 "Haskell",
];
 $("#linguagens").autocomplete({source: linguagens});
```

Exibindo dialogs:

É possível exibir pequenas janelas dentro do browser, chamadas de Dialogs, ou caixas de diálogo.

```
<div id=”dialog” title=”Aviso do sistema”>
    <p>Você acha que esse é o melhor
    Livro de JavaScript que já existiu?</p>
</div>
```

Em seguida, vamos usar o método dialog para configurar a Dialog, passando um objeto contendo as configurações.

O nosso código vai ficar assim:

```
$(function () {

    function onSimClick() {
    alert("Uhu! Você escolheu 'Sim'!");
    $(this).dialog("close");
    };

    function onClaroClick() {
    alert("\\o/");
    $(this).dialog("close");
    };

    var opcoes = {

    buttons: {
    "Sim": onSimClick,
    "Claro!": onClaroClick
    }

   };

    $("#dialog").dialog(opcoes);
});
```

Problemas com o jQuery:

O maior problema do jQuery é seu tamanho, por isso devemos selecionar as bibliotecas de forma correta na hora de baixar.

jQuery mobile:

Assim como o jQuery UI o jQuery mobile também é uma bliblioteca do jQuery porém com foco em dispositivos moveis,

O jQuery Mobile não garante que sua aplicação ou site vão ser executados corretamente em todos os dispositivos. Você pode acessar o endereço abaixo para verificar quais dispositivos são compatíveis com a biblioteca:

http://jquerymobile.com/gbs/

A página inicial do jQuery Mobile (http://jquerymobile.com/) oferece uma ferramenta para que você possa experimentar os diversos componentes num simulador de dispositivo móvel.

O Theme Roller para jQuery Mobile está em

http://jquerymobile.com/themeroller/

Orientação a objetos no JavaScript

Objetos:

Os conceitos de objetos no JavaScript são um pouco diferentes.

Primeiramente, não existe o conceito de classes no JavaScript tal como o conhecemos nas linguagens, então você não precisa de uma classe para criar seus objetos.

Em segundo lugar, o que normalmente tratamos como objetos no JavaScript não são necessariamente objetos, mas funções com propriedades. E os objetos que são realmente objetos são diferentes dos objetos de outros linguagens.

Objeto simples:

O jeito mais simples de criarmos um objeto em JavaScript é utilizando um dicionário, ou se preferir, mapa ou hashtable.

O objeto mais simples que podemos criar é simplesmente o { }. Ele não faz nada, nem guarda valor nenhum, mas podemos adicionar funcionalidades nele conforme as nossas necessidades.

```
var simples = { };
simples.nome = "Plínio";
simples.oi = function() {
 console.log("Olá, " + this.nome + "!");
}

simples.oi();
```

Resultado: Olá, Plínio!

Funções em vez de classes:

Em JavaScript não existe o conceito de classes, mas as funções podem muito bem assumir esse papel, o que significa que podemos criar objetos a partir de funções.

Exemplo:

```
function Funcionario(nome, cargo, salario) {
this.nome = nome;
this.cargo = cargo;
 this.salario = salario;
}

var paulo = new Funcionario("Paulo", "Analista de sistemas", 5000);
var pedro = new Funcionario("Pedro", "Gerente de contas", 4500);
```

Prototipação:

O JavaScript é uma linguagem que faz uso de prototipação, o que significa que, ao invés de termos uma classe herdando características de outra classe, os objetos herdam suas características de outros objetos.

Exemplo:

```
function Funcionario(){
}

var paulo = new Funcionario();
Funcionario.prototype.nome = “<sem nome>”;
Funcionario.prototype.cargo = “<sem cargo>”;
Funcionario.prototype.salario = NaN;
```

Aqui você deve estar se perguntando qual a vantagem dessa tal prototipação. Uma das principais vantagens é a possibilidade de poder adicionar características a classes já existentes, fazendo com que seu código fique mais limpo e reutilizável.

Herança:

A herança no Javascript funciona através da prototipação de objetos.

Exemplo:

```
function Animal() {

this.comer = function() {
 console.log("Eu como");
 };

 this.respirar = function() {
 console.log("Eu respiro");
 };

 }


function Mamifero() {
 this.mamar = function() {
 console.log("Eu mamo");
 }
}

Mamifero.prototype = new Animal();
mamifero = new Mamifero();
```

Mamífero irá herdar as características de animal

Mixin no JavaScript:

Mixin é uma forma de adicionar comportamentos e características a uma classe já existente, evitando que você tenha que reescrever o mesmo código várias vezes em lugares diferentes. Mixin também pode ser considerado como uma forma de se implementar herança múltipla.

A herança múltipla no Javascript pode ser usada através do método **.includes** em vez de **.prototype.**

Exemplo:

```
function Peixe() {
 this.respirar = function() {
 console.log("Eu respiro embaixo d'água");
 };
 }

Peixe.prototype = new Animal();
 var peixe = new Peixe();

function Nadador() {
 this.nadar = function() {
 console.log("Eu nado");
 }
}

Peixe.includes(Nadador);
```

Porem temos que criar uma função includes:

```
Object.prototype.includes = function (constructor) {

 var objeto = new constructor();
 for (var propriedade in objeto) {
 if (objeto.hasOwnProperty(propriedade)) {
 this.prototype[propriedade] = objeto[propriedade];
 }
}
};
```

Programação Funcional:

A programação funcional é um paradigma como os outros paradigmas na programação.

Na programação funcional a menor parte do seu sistema é uma função, isso implica que você pode atribuir funções a variáveis, pode passá-las por parâmetro e mesmo fazer com que uma função retorne outra função.

Escopo:

O Escopo no JavaScipt nos diz que variáveis globais podem ser usadas por todas as funções e variáveis, e as variáveis locais presentes nas funções só podem ser usadas no seu escopo local no caso a função em que está.

Plugins:

Os plugins são métodos a parte do jQuery e são tratados quase como se fossem parte da biblioteca.

Anatomia dos Plugins:

Um plugin é um objeto que você agrega ao objeto jQuery, e que deve seguir algumas boas práticas para manter uma boa relação com os outros milhares de plugins existentes.

Algumas dessas boas práticas são tidas como padrão pelos desenvolvedores e usuários mais experientes. Outras são apenas questão de bom senso e surgem depois de você cansar de ter plugins dando problema em produção e entrando em conflito com outros plugins ou bibliotecas.

Todas as chamadas, consultas e configurações devem ser feitas através de um único método.

Criando um plugin:

A diferença aqui é que, ao invés de adicionar o método em prototype, vamos adicionar em fn.

```
jQuery.fn.nome = function(options) {
// esse é o menor plugin que existe }
```

Nota-se que usamos jQuery em vez de $, fazemos isso pois outras bibliotecas utilizam de $ e isso gera conflitos.

Uma forma de contornarmos isso é criando uma variável $ que receba a palavra jQuery.

```
;(function($) {
 $.fn.validador = function(options) {
 // esse é o segundo menor plugin que existe
 }
})(jQuery);
```

Outros Plugins:

MáskMoney:

É basicamente um plugin que permite a formatação de campos de texto para permitir a digitação de valores monetários.

```
$("#valor").maskMoney({symbol: "R$",
 thousands: ".",
 decimal: ","
});
```

O repositório do GitHub fica no endereço abaixo:
https://github.com/plentz/jquery-maskmoney

Gráficos:

O jqPlot é uma ferramenta de geração de gráficos que utiliza o componente Canvas do HTML5.

```
$.jqplot('chartdiv',
 [[[1, 2],
[3, 5.12],
 [5, 13.1],
 [7, 33.6],
 [9, 85.9],
 [11, 219.9]]]);
```