Revisões:

Vantagens da utilização de jQuery

Separação entre o Javascript e o HTML

Ao invés de usar atributos HTML para identificar as funções para manipulação de eventos, o jQuery lida com eventos puramente em JavaScript. Deste modo, as tags HTML e o código Javascript são completamente separados.

Elimina incompatibilidades entre navegadores:

Os motores de Javascript dos diferentes navegadores diferem ligeiramente, de modo que o código Javascript que funciona para um navegador pode não funcionar em outro.

O jQuery lida com todas essas inconsistências entre browsers e fornece uma interface consistente que funciona nos diferentes navegadores.

Extensível:

O jQuery é muito extensível – através da adição de novas livrarias ao projeto.

Novos eventos, elementos e métodos podem ser facilmente adicionados e depois reutilizados como um plugin.

©2014-17, JOAQUIM SOUSA PINTO

Revisões:

JSON Objects & Arrays

Os objetos JSON são escritos dentro de chavetas {} e podem conter vários pares nome / valor, separados por vírgulas:

```
{'name': 'Noé Elisabete Ferreiro',
    'email' : 'noe.ferreiro@nowhere.com',
    'address' : 'Street name & number\nCounty\nState',
    'birthDate' : '1990/11/24',
    'sex' : 'Male',
    'course' : {
        'id' : 1234,
        'name' : 'Course name'}
}
```

Nota: Os valores do tipo texto são escritos entre aspas (simples '...' ou duplas "..."). Os valores lógicos ou numéricos são escritos diretamente.

Os objetos JSON podem ser agrupados em arrays que são escritos entre colchetes [] e separados por vírgulas:

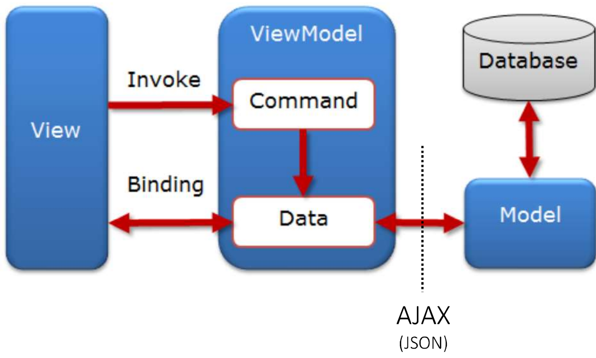
```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
]
```


©2014-17, JOAQUIM SOUSA PINTO

Antes de utilizar knockout

Model-View-View Model (MVVM)

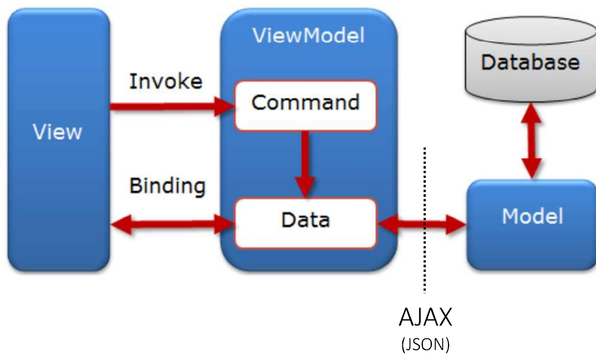
Model-View-View Model (MVVM) é um padrão de design para criar interfaces
Descreve como manter uma interface de utilizador dividindo-a em três partes: um Model, um ViewModel e uma View



©2014-17, JOAQUIM SOUSA PINTO

Componentes do Model-View-View Model (MVVM)

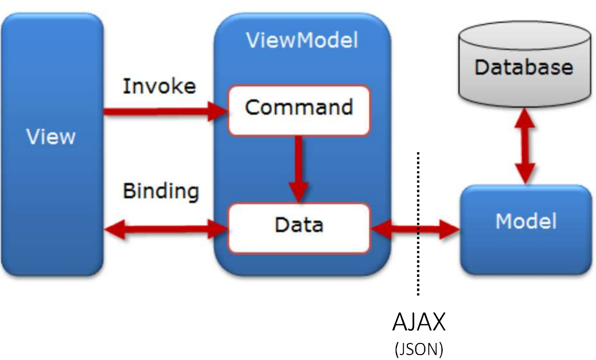
Um model contém os dados armazenados da aplicação. Esses dados representam objetos e operações referentes à logica de negócio (*business logic*) e são independentes de qualquer interface - normalmente, o acesso ao modelo faz-se através de chamadas Ajax para algum código do lado do servidor para ler e gravar os dados do modelo armazenado.



©2014-17, JOAQUIM SOUSA PINTO

Componentes do Model-View-View Model (MVVM)

Um view model contém uma representação em código dos dados do modelo e operações da interface. Note que esta não é a interface com o utilizador em si: não tem qualquer conceito de botões ou estilos de exibição. Também não é o modelo de dados persistentes que estão numa base de dados - ele contém os dados não salvos com que o utilizador está a trabalhar.

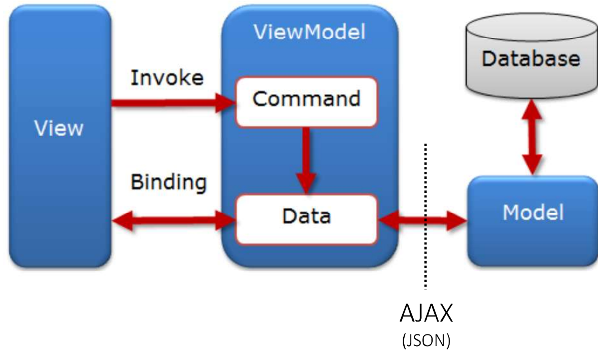


AJAX
(JSON)

©2014-17, JOAQUIM SOUSA PINTO

Componentes do Model-View-View Model (MVVM)

Uma view contém uma interface visível e interativa representando o estado do *view model*. Ele exibe informações do *view model*, envia comandos para o *view model* (por exemplo, quando o utilizador clica nos botões) e atualiza-se automaticamente sempre que o estado do *view model* é alterado. É normalmente um documento HTML com ligações declarativas (**data bindings**) que permitem a ligação com o *view model*.



AJAX
(JSON)

©2014-17, JOAQUIM SOUSA PINTO

A livraria KnockoutJS

Knockout é uma biblioteca JavaScript que ajuda a criar interfaces de utilizador de exibição e edição ricas e responsivas com um modelo de dados subjacente limpo.

Sempre que há seções da interface de utilizador que necessitam de atualização dinâmica (por exemplo, devido às ações do utilizador ou quando uma fonte de dados externa é alterada), o KO, acrónimo do Knockout, pode ajudar nessa implementação de forma mais simples e mais eficiente que utilizando apenas javascript ou mesmo jQuery.

©2014-17, JOAQUIM SOUSA PINTO

A livraria KnockoutJS

Principais características:

- Vinculações declarativas
  - Associa elementos do DOM a um modelo de dados através de uma sintaxe concisa e legível
- Atualização automática da interface com o utilizador
  - Quando o estado do modelo de dados é alterado, a interface com o utilizador é atualizada automaticamente
- Acompanhamento de dependências
  - Implicitamente estabelece cadeias de relações entre os dados do modelo de modo a transformá-los e combiná-los
- Templating
  - Gera rapidamente intefaces de utilizador sofisticadas como uma função dos dados do modelo

©2014-17, JOAQUIM SOUSA PINTO

A livraria KnockoutJS

Outras características:

- Livre, código aberto (licença MIT)
- JavaScript puro - funciona com qualquer framework web
  - Sem dependências
- Pequeno e leve - 54kb minified
- Suporta todos os navegadores habituais, mesmo os antigos
  - IE 6+, Firefox 3.5+, Chrome, Opera, Safari (desktop / mobile)
- Totalmente documentado
  - Disponibiliza documentos da API, exemplos e tutoriais interativos

©2014-17, JOAQUIM SOUSA PINTO

Como usar o knockout? (1)

Para criar um *view model* com KO, basta declarar qualquer objeto JavaScript (JSON). Por exemplo:

```
var myViewModel = {
    personName: 'Zé Maria',
    personAge: 45
};
```

Pode criar-se uma *view* deste *view model* usando uma vinculação declarativa.

```
O meu nome é <span data-bind="text: personName"></span>
```

Para que tudo funcione, é preciso preciso ativar o knockout:

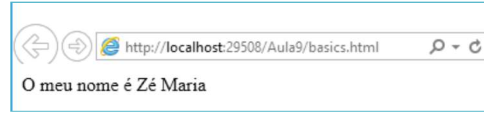
```
ko.applyBindings(myViewModel);
```


©2014-17, JOAQUIM SOUSA PINTO

Como usar o knockout? (2)

```
<!DOCTYPE html>
<html>
<head>
    <title>o meu primeiro teste knockout</title>
    <meta charset="utf-8" />
</head>
<body>
    O meu nome é <span data-bind="text: personName"></span>
    <script src="../Scripts/knockout-3.4.0.js"></script>
    <script>
        var myViewModel = {
            personName: 'Zé Maria',
            personAge: 45
        };

        ko.applyBindings(myViewModel);
    </script>
</body>
</html>
```



Observáveis e dependências (`ko.observable()`)(1)

http://knockoutjs.com/documentation/observables.html

Já vimos como criar um *view model* básico e como exibir uma das suas propriedades (text) usando uma ligação.

Um dos principais benefícios do KO é que ele atualiza a interface do utilizador automaticamente quando o *view model* muda.

- Como é que o KO pode saber quando as partes do *view model* mudam?
- Resposta: é preciso declarar as propriedades do seu modelo como observáveis
  - *Os observáveis são objetos JavaScript especiais que podem notificar os assinantes sobre as alterações e podem detectar dependências automaticamente.*

Observáveis e dependências (`ko.observable()`)((2)

Para utilizar variáveis observáveis, reescreve-se o *view model* anterior da seguinte maneira:

```
var myViewModel = {
    personName: ko.observable('Zé Maria'),
    personAge: ko.observable(45)
};
```

Não é preciso alterar a *view* - a sintaxe de ligação de dados é a mesma.

- A diferença é que agora a *view* é capaz de detectar alterações do *view model* e, quando isso acontecer, atualizará a exibição na *view* automaticamente.

©2014-17, JOAQUIM SOUSA PINTO

Observáveis e dependências (`ko.observable()`)((3)

Problema:

- Nem todos os browser suportam suportam operações de leitura (get) e escrita (set) de JavaScript (IE), portanto, por questões de compatibilidade, os objetos `ko.observable` são funções.

Para ler o valor atual do observável, basta chamar o observável sem parâmetros.

- Do exemplo, `myViewModel.personName()` retornará `'Zé Maria'`, e `myViewModel.personAge()` retornará `45`.

Para escrever um novo valor no observável, invoca-se o observável e passa-se o novo valor como parâmetro.

- Por exemplo, `myViewModel.personName('Maria')` irá alterar o valor de nome para 'Maria'.

Para gravar valores em várias propriedades observáveis num *view model*, pode usar a sintaxe de encadeamento.

- Por exemplo, `myViewModel.personName('Maria').PersonAge(50)`
- Isto mudará, simultaneamente, `myViewModel.personName` e `myViewModel.PersonAge`.

©2014-17, JOAQUIM SOUSA PINTO

Arrays de observáveis (`ko.observableArray([])`)

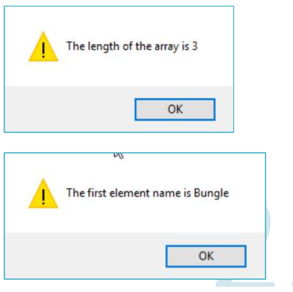
http://knockoutjs.com/documentation/observableArrays.html

Já vimos que, caso se pretenda detectar e responder a alterações num objeto, usamos observáveis.

Se pretendermos detectar e responder a alterações numa coleção de objetos, deveremos utilizar um `observableArray`.

Esta possibilidade é particularmente útil em cenários em que se exibem ou editam vários valores e são necessárias seções repetidas da interface para fazer aparecer e desaparecer à medida que os itens são adicionados e/ou removidos.

```
// This observable array initially contains three objects
       var myObservableArray = ko.observableArray([
           { name: "Bungle", type: "Bear" },
           { name: "George", type: "Hippo" },
           { name: "Zippy", type: "Unknown" }
       ]);
       alert('The length of the array is ' + myObservableArray().length);
       alert('The first element name is ' + myObservableArray()[0].name);
```



©2014-17, JOAQUIM SOUSA PINTO

Observáveis calculadas(`ko.computed`)

Suponha que já tem um observável para firstName, e outro para lastName, e deseja exibir o nome completo?

É aí que os `observáveis calculados` são úteis - são funções que dependem de um ou mais observáveis e serão atualizados automaticamente sempre que alguma das suas dependências mudarem.

```
O meu nome é <span data-bind="text: fullName"></span>
```

```
function AppViewModel() {
    var self = this;

    self.firstName = ko.observable('Bob');
    self.lastName = ko.observable('Smith');
    self.fullName = ko.computed(function () {
        return self.firstName() + " " + self.lastName();
    });
}
```


©2014-17, JOAQUIM SOUSA PINTO

KO bindings (1)

`text:` – o binding com **`text`** faz com que o elemento DOM associado exiba o valor de texto do seu parâmetro.

Normalmente, esta propriedade é útil com elementos que tradicionalmente exibem texto, como por exemplo o <span> ou o <em>, mas tecnicamente pode usá-lo com qualquer elemento.

`html:` – o binding com **`html`** faz com que o elemento DOM associado exiba o valor do seu parâmetro como HTML.

Normalmente, isso é útil quando os valores no *view model* contêm marcação HTML.

KO bindings (1)

`text:` – o binding com **`text`** faz com que o elemento DOM associado exiba o valor de texto do seu parâmetro.

Normalmente, esta propriedade é útil com elementos que tradicionalmente exibem texto, como por exemplo o <span> ou o <em>, mas tecnicamente pode usá-lo com qualquer elemento.

`html:` – o binding com **`html`** faz com que o elemento DOM associado exiba o valor do seu parâmetro como HTML.

Normalmente, isso é útil quando os valores no *view model* contêm marcação HTML.

Exemplo:

View: <span data-bind="text: fullName">

ViewModel: return “” + firstName + “ “ + lastName + “"

KO bindings (2)

`css:` – o binding **`css`** adiciona ou remove uma ou mais classes CSS ao elemento DOM associado. Permite aplicar classes de acordo com valores do *view model*.

(Nota: Se não quiser aplicar uma classe CSS, mas preferir atribuir um valor de atributo de estilo diretamente, consulte o binding `style`.)

```
<div data-bind="css: profitStatus">Profit Information</div>
```

`Style:` – o binding **`style`** adiciona ou remove um ou mais valores de estilo ao elemento DOM associado.

```
<div data-bind="style: { color: currentProfit() < 0 ? 'red' : 'black' }">Profit Information</div>
```


©2014-17, JOAQUIM SOUSA PINTO

KO bindings (3)

`attr:` – O binding **`attr`** fornece uma maneira genérica de definir o valor de qualquer atributo para o elemento DOM associado.

Isso é útil, por exemplo, quando precisa definir o atributo de título de um elemento, o src de uma tag img ou o href de um link com base em valores no seu *view model*, com o valor do atributo sendo atualizado automaticamente sempre que a propriedade correspondente no *view model* muda.

```
<a data-bind="attr: { href: url, title: details }">Relatório</a>

<script type="text/javascript">
   var viewModel = {
        url: ko.observable("yearReport.html"),
        details: ko.observable("relatório e contas referente ao corrente ano")
    };
</script>
```


©2014-17, JOAQUIM SOUSA PINTO

KO bindings (4)

`visible:` – permite fazer o binding da propriedade visível a um elemento DOM que ficará visível sempre que a variável de controlo do *view model* tomar um valor `true`.

©2014-17, JOAQUIM SOUSA PINTO

KO – controlo de fluxo

http://knockoutjs.com/documentation/foreach-binding.html

`foreach:` – o binding **`foreach`** duplica uma seção de marcação para cada entrada de uma matriz e associa cada cópia dessa marcação ao item correspondente da matriz. Isso é especialmente útil para renderizar listas ou tabelas.

Assumindo que a matriz é um array de observáveis, sempre que adicionar, remover ou reordenar as entradas da matriz, a ligação atualizará eficientemente a UI mantendo o sincronismo entre elas - inserindo ou removendo mais cópias da marcação ou reordenando elementos DOM existentes, sem afetar quaisquer outros elementos DOM.

Isso é muito mais rápido do que regenerar a saída foreach por inteiro após cada alteração de matriz.

Pode colocar qualquer número de bindings `foreach` junto com outras ligações de controle-fluxo, como `if` ou `with`.

©2014-17, JOAQUIM SOUSA PINTO

Exemplo de binding com foreach

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo foreach knockout</title>
    <link href="../Content/bootstrap.min.css" rel="stylesheet" />
    <meta charset="utf-8" />
</head>
<body>
    <table class="table table-striped table-condensed">
        <thead>
            <tr><th>First name</th><th>Last name</th></tr>
        </thead>
        <tbody data-bind="foreach: people">
            <tr>
                <td data-bind="text: firstName"></td>
                <td data-bind="text: lastName"></td>
            </tr>
        </tbody>
    </table>
```

```
    <script src="../Scripts/jquery-3.1.1.min.js"></script>
    <script src="../Scripts/bootstrap.min.js"></script>
    <script src="../Scripts/knockout-3.4.2.js"></script>
    <script type="text/javascript">
    ko.applyBindings({
        people: [
            { firstName: 'Bert', lastName: 'Bertington' },
            { firstName: 'Charles', lastName: 'Charlesforth' },
            { firstName: 'Denise', lastName: 'Dentiste' }
        ]
    });
    </script>
</body>
</html>
```

First name	Last name
Bert	Bertington
Charles	Charlesforth
Denise	Dentiste

©2014-17, JOAQUIM SOUSA PINTO

KO – controlo de fluxo

http://knockoutjs.com/documentation/if-binding.html
http://knockoutjs.com/documentation/ifnot-binding.html
http://knockoutjs.com/documentation/with-binding.html

if: – o binding if faz com que uma seção de marcação apareça no documento somente se a variável de controlo especificada for avaliada como verdadeira.

ifnot: – é igual ao binding `if` somente inverte o valor da expressão de avaliação especificada – isto porque não existe um "else binding"

with: - o binding com `with` cria um novo contexto de vinculação, de modo que os elementos descendentes são vinculados no contexto de um objeto especificado.

©2014-17, JOAQUIM SOUSA PINTO

KO – binding eventos

click: – O binding do evento clique permite associar um gestor de eventos cuja função JavaScript é chamada quando o elemento DOM associado for clicado.

Isso é mais comumente usado com elementos como botões, input e hiperligações, mas na verdade funciona com qualquer elemento DOM visível.

©2014-17, JOAQUIM SOUSA PINTO