AngularJS
na prática

Daniel Schmitz
Douglas Lira

[PT_BR] AngularJS na prática

Crie aplicações web com AngularJS

Daniel Schmitz e Douglas Lira

Esse livro está à venda em http://leanpub.com/livro-angularJS

Essa versão foi publicada em 2015-07-06

Tweet Sobre Esse Livro!

Por favor ajude Daniel Schmitz e Douglas Lira a divulgar esse livro no Twitter!

O tweet sugerido para esse livro é:

Comprei o livro AngularJS na prática do autor Daniel Schmitz. Mais informações em: https://leanpub.com/livro-angularJS

A hashtag sugerida para esse livro é #livroangularjs.

Descubra o que as outras pessoas estão falando sobre esse livro clicando nesse link para buscar a hashtag no Twitter:

https://twitter.com/search?q=#livroangularjs

Conteúdo

Errata

Esta é uma obra em constante evolução, e erros podem aparecer. Caso encontre algo errado, por favor envie um email para danieljfa@gmail.com.

Todos os erros corrigidos serão atualizados automaticamente, e publicados em futuras versões da obra. Você poderá a qualquer momento obter uma nova versão, visitando o site leanpub.com

Introdução

A linguagem JavaScript vem tornando-se uma das mais usadas em todas as áreas de programação que estão ligadas a Web. É uma linguagem que praticamente nasceu com a Web, criada inicialmente para os navegadores *Netscape*, por volta de 1995.

Mais de 15 anos depois, javascript evolui de forma significativa com a criação de novos frameworks, tornando-se a cada dia mais poderosa e utilizada. Um deles é o jQuery, amplamente conhecido e praticamente obrigatório se você deseja controlar os elementos de uma página HTML, também chamado de DOM.

Nesta "onda", diversos frameworks de qualidade surgem a cada dia, e dentre eles temos o AngularJS, que será o nosso principal objeto de estudo.

O que é AngularJS?

Este framework é mantido pelo Google e possui algumas particularidades interessantes, que o fazem um framework muito poderoso.

Uma dessas particularidades é que ele funciona como uma extensão ao documento HTML, adicionando novos parâmetros e interagindo de forma dinâmica com vários elementos. Ou seja, com AngularJS podemos adicionar novos atributos no html para conseguir adicionar funcionalidades extras, sem a necessidade de programar em javascript.

AngularJS é quase uma linguagem declarativa, ou seja, você usa novos parâmetros na linguagem html para alterar o comportamento padrão do html. Estes parâmetros (ou propriedades) são chamados de diretivas, na qual iremos conhecer cada uma ao longo desta obra.

Além disso, é fornecido também um conjunto de funcionalidades que tornam o desenvolvimento web muito mais fácil e empolgante, tais como o DataBinding, templates e fácil uso do Ajax. Todas essas funcionalidades serão abordadas ao longo desta obra.

Código fonte

O código fonte desta obra encontra-se em:

https://github.com/danielps/livro-angular

Uma nota sobre pirataria

Todos nós sabemos que a pirataria existe, seja ela de filmes, músicas e livros. Eu, como autor, entendo o porquê disso acontecer, pois a maioria das distribuidoras cobra um preço além do valor, pois eles precisam obter uma margem de lucro satisfatória. Não é atoa que existem ebooks pelo preco de 50 reais ou até mais.

Como autor independente, eu consigo retirar a maioria dos custos envolvidos na produção do livro. Retiro custos de produção, design e até impostos (sim, eu posso retirar os impostos por estar produzindo conhecimento).

Tudo isso é retirado e o que você paga nos meus ebooks é o preço do meu desenvolvimento, que nunca vai passar de 10 dólares.

Se você comprou o livro, mais uma vez *obrigado!!*, mas se você obteve esta cópia por outros meios, eu faço um convite para que leia a obra e, caso goste, visite https://leanpub.com/livro-angularJS e compre o nosso livro.

Ajude o autor a produzir mais livros de qualidade por um preço justo, contribua para que você possa encorajar outros autores a produzirem conteúdo nesse formato.

Parte 1 - AngularJS

Preparando o ambiente

É preciso muito pouco para começar a aprender AngularJS. Em um nível mais básico, você precisa de um editor de textos e de um navegador web.

Como o nosso objeto é criar sistemas, iremos adicionar como requisito o uso de um servidor web capaz de processar páginas em PHP.

Editor de textos

Aqui deixamos livre a sua escolha por um editor de textos ou uma IDE. Lembre-se que todo o nosso desenvolvimento é focado em HTML e JavaScript, ou seja, você não precisará de algo "poderoso" para aprender AngularJS, apenas algo que complemente o código HTML já está ótimo.

> Nesta obra estamos usando extensivamente o *Sublime Text 2*, inclusive para escrever o próprio livro, então nos sentimos confortáveis em recomendá-lo.

Servidor Web

O Servidor Web é necessário para processar as páginas PHP dos sistemas que iremos construir. O AngularJS é um framework para visualização de dados e informações, ele não possui a funcionalidade de prover dados dinâmicos, ou persistir informações em um banco de dados. Estas características são provenientes de um servidor Web.

Um dos servidores mais comuns nos dias de hoje é o Apache. O banco de dados que utilizaremos é o MySql. Para cada sistema operacional, existe uma forma bastante fácil de possuir estes programas.

Windows

Se você utiliza Windows, poderá instalar o **Wamp Server**, disponível neste link[1]. Faça o download da versão mais recente e instale o Wamp na configuração padrão.

Após instalado, você poderá incluir arquivos na seguinte pasta **C:\wamp\www**, e poderá utilizar o Apache, bem como o MySql e outros utilitários acessando http://localhost/.

[1] http://www.wampserver.com/en/

Mac Assim como existe o Wamp para o Windows, existe o Mamp[2] para o Mac, e você pode instalá-lo caso julgue necessário.

Linux
Se usa Linux, acredito que instalar Apache e MySql no sistema é uma tarefa extremamente simples para você :)

> Os exemplos desta obra foram criados utilizando o Wamp Server como servidor Web.

Instalando o AngularJS

É preciso apenas duas alterações na estrutura de um documento HTML para que possamos ter o AngularJS instalado. A primeira, e mais óbvia, é incluir a biblioteca javascript no cabeçalho do documento. A segunda, e aqui temos uma novidade, é incluir a propriedade **ng-app** no elemento html em que queremos "ativar" o angularJS. Neste caso, começamos inserindo na tag `<html>` do documento. O exemplo a seguir ilustra este processo.

HTML default para o AngularJS

```
1 <html ng-app>
2     <head>
3         <title>Lista de compras</title>
4         <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angul\
5 ar.min.js"></script>
6     </head>
7     <body>
8     </body>
9 </html>
```

Na linha 1, temos o uso do **ng-app** que habilita o AngularJS a alterar o comportamento das tags html abaixo dele, permitindo assim que possamos utilizar mais propriedades do AngularJS.

Na linha 4 incluímos a biblioteca angularJS diretamente, através de um endereço CDN. Pode-se também realizar o download da biblioteca e inserir no projeto. Você pode usar tanto o arquivo minificado quando o arquivo normal.

[2] http://www.mamp.info/en/index.html

Principais características do AngularJS

Agora que temos o básico em funcionamento, vamos aprender as principais regras do AngularJS. Através delas será possível realizar diferentes tarefas que irão tornar o desenvolvimento web muito mais simples e prazeroso.

DataBind

Uma das principais vantagens do AngularJS é o seu DataBind. Este termo é compreendido como uma forma de ligar automaticamente uma variável qualquer a uma outra. Geralmente, o DataBind é usado para ligar uma variável do JavaScript (ou um objeto) a algum elemento do documento HTML.

No exemplo a seguir, estamos usando o AngularJS para ligar uma caixa de texto (o elemento `input` do html) à um cabeçalho.

Exemplo de DataBind

```
1  <html ng-app>
2      <head>
3          <title>Hello World</title>
4          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angul\
5  ar.min.js"></script>
6      </head>
7      <body>
8              Hello <input type="text" ng-model="yourName"/>
9              <hr/>
10             <h1>Hello {{yourName}}</h1>
11     </body>
12 </html>
```

Além da propriedade **ng-app** (linha 1), utilizamos para DataBind a propriedade **ng-model**, para informar que este elemento estará ligado a uma variável do AngularJS, através da variável `yourName`, na linha 8. Isso significa que qualquer alteração na caixa de texto irá atualizar o valor da variável.

Na linha 10, temos a chamada à variável através do comando `{{yourName}}`, que imprime o valor da variável. Como o DataBind é dinâmico, ao mesmo tempo que algo é escrito na caixa de texto, o seu referido bind é realizado, atualizando instantaneamente o seu valor.

Bind também pode ser realizado em objetos, mas antes de começar a aumentar a complexidade do código, vamos criar um *controller* para melhorar a organização do código.

Controller

Um controller é, na maioria das vezes, um arquivo JavaScript que contém funcionalidades pertinentes à alguma parte do documento HTML. Não existe uma regra para o controller, como por exemplo ter um controller por arquivo HTML, mas sim uma forma de sintetizar as regras de negócio (funções javascript) em um lugar separado ao documento HTML.

Inicialmente, vamos criar um exemplo simples apenas para ilustrar como usar o controller.

Uso do controller

```
1  <html ng-app="app">
2      <head>
3          <title>Hello World</title>
4          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angul\
5  ar.min.js"></script>
6          <script src="simpleController.js"></script>
7      </head>
8      <body ng-controller="simpleController">
9              Hello <input type="text" ng-model="user.name"/>
10             <hr/>
11             <h1>Hello {{user.name}}</h1>
12     </body>
13 </html>
```

Neste exemplo, após incluir o arquivo `simpleController.js`, usamos a propriedade **ng-controller** (linha 8) para dizer que, todo elemento abaixo do `<body>` será gerenciado pelo controller.

Na linha 9, adicionamos uma caixa de texto utilizando a propriedade **ng-model** e atribuímos o valor `user.name`. Neste caso, `user` é um objeto e `name` é uma propriedade a este objeto. Na linha 11 temos o uso do bind no objeto `user`.

simpleController.js

```
1  var app = angular.module('app', []);
2
3  app.controller('simpleController', function ($scope) {
4          $scope.user = {name:"Daniel"}
5  });
```

O arquivo `simpleController.js` contém a criação da app e a indicação do controller, que é criado de forma modularizada. O nome do arquivo javaScript não precisa ser o mesmo, e com isso pode-se ter vários controllers em somente um arquivo javaScript, bastando apenas registrá-lo na aplicação.

Perceba que, na linha 1, criamos um módulo do angular na qual chamamos de ´app´, e depois registramos um controller à este módulo.

Neste controller, temos o parâmetro `$scope` que é um "ponteiro" para a aplicação em si, ou seja, `$scope` significa a própria página html. Como o controller foi declarado no elemento `<body>`, `$scope` é usado para todo este elemento. Usa-se o `$scope` para criar uma conexão entre o model e a view, como foi feito no exemplo utilizando o objeto `user`.

Métodos do controller

O controller é usado também para manipular regras de negócio que podem ou não alterar os models. No exemplo a seguir, usamos o controller para definir uma variável e um método para incrementar em 1 esta variável.

Simples exemplo de contador

```
1  <html ng-app="app">
2      <head>
3          <title>Hello Counter</title>
4          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angul\
5  ar.min.js"></script>
6          <script src="countController.js"></script>
7      </head>
8      <body ng-controller="countController">
9              <a href="#" ng-click="addOne()">Add 1</a>
10             <hr/>
11             <p>Counter value: {{counter}}</p>
12     </body>
13 </html>
```

Neste exemplo, usamos a propriedade **ng-click** que irá chamar um método dentro do seu respectivo controller.

countController.js

```
1  var app = angular.module('app', []);
2
3  app.controller('countController', function ($scope) {
4          $scope.counter = 0;
5
6          $scope.addOne = function(){
7                  $scope.counter++;
8          }
9  });
```

No controller, criamos a variável `counter` e também o método `addOne`, que manipula a variável, de forma que o seu valor é refletido automaticamente na view (html) através do dataBind.

Loops

Outra característica do AngularJS é utilizar templates para que se possa adicionar conteúdo dinâmico. Um loop é sempre realizado através da propriedade **ng-repeat** e obedece a uma variável que geralmente é um array de dados.

O exemplo a seguir ilustra este processo, utilizando a tag `li` para exibir uma lista qualquer.

Usando loops

```
<html ng-app="app">
    <head>
        <title>Hello Counter</title>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angul\
ar.min.js"></script>
        <script type="text/javaScript">
            var app = angular.module('app', []);
            app.controller('loopController', function ($scope) {
                $scope.fruits = ['banana','apple','orange'];
            });
        </script>
    </head>
    <body ng-controller="loopController">
        <ul>
            <li ng-repeat="fruit in fruits">{{fruit}}</li>
        </ul>
    </body>
</html>
```

> Como visto neste exemplo, pode-se adicionar JavaScript no arquivo HTML, mas não recomenda-se esta prática. É uma boa prática de programação utilizar o arquivo HTML apenas como a camada view, e o arquivo JavaScript como a camada de controller, de acordo com os padrões MVC.

Formulários

Existem diversas características que um formulário contém, tais como validação, mensagens de erro, formato dos campos, entre outros. Neste caso, usamos o AngularJS de diferentes formas, e usamos

vários parâmetros **ng** para controlar todo o processo.

O exemplo a seguir exibe apenas algumas dessas propriedades, para que você possa entender como o processo funciona, mas durante a obra iremos verificar todos os detalhes necessários para construir um formulário por completo.

Formulário com validação

```
<html ng-app>
<head>
    <title>Simple Form</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.m\
in.js"></script>
</head>
<body>

    <form name="myForm">
        <span ng-show="myForm.$invalid">
            Found erros in the form!
        </span>
        <input type="text" ng-model="name" name="Name" value="Your Name" require\
d/>
        <button ng-disabled="myForm.$invalid"/>Save</button>
    </form>

</body>
</html>
```

Neste formulário, usamos mais algumas propriedades, como por exemplo **ng-show** que irá exibir ou não a tag `<span>` contendo a mensagem de erro do formulário, e **ng-disabled** que desativa o botão de submissão do formulário.

O uso do `myForm.$invalid` é um recurso do AngularJS que define se um formulário está inválido ou não. Como usamos uma caixa de texto com a propriedade `required`, se o campo não estiver preenchido, o formulário ficará inválido.

Rotas e Deep linking

O AngularJS possui um recurso chamado Deep Linking, que consiste em criar rotas na URI do documento HTML para manipular partes do código HTML de forma independente, podendo assim separar ainda mais as camadas da sua aplicação. No caso mais simples, suponha que exista uma lista de dados que são exibidos em uma tabela, e que ao clicar em um item desta lista, deseja-se exibir um formulário com os dados daquela linha.

No documento HTML criado, existem dois componentes bem definidos pela aplicação. O primeiro é a tabela com as informações, e o segundo, o formulário para edição dos dados.

> O uso de DeepLinking usa Ajax para carregar templates de forma dinâmica, então é necessário que todo o exemplo seja testado em um servidor web.

Se organizarmos esta pequena aplicação em arquivos, teremos:

index.html
: O arquivo principal da aplicação, que contém o código html, o **ng-app**, a inclusão do AngularJS, entre outras propriedades.

app.js
: Contém todo o código javascript que define as regras de negócio da aplicação.

list.html
: Contém a tabela que lista os dados.

form.html
: Contém o formulário de edição e criação de um novo registro.

index.html

```
1  <html ng-app="app">
2  <head>
3          <title>DeepLinking Example</title>
4          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.js">\
5  </script>
6          <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular-rout\
7  e.js"></script>
8          <script src="app.js"></script>
9  </head>
10 <body>
11         <h1>DeepLink Example</h1>
12         <div ng-view></div>
13
14 </body>
15 </html>
```

Inicialmente criamos o arquivo `index.html`, que contém a chamada aos arquivos javascript da aplicação. Além dos arquivos javaScript, também usamos a propriedade **ng-app**, que já aprendemos a usar em qualquer aplicação que use o framework.

Também adicionamos um segundo arquivo javascript, que é responsável pelo gerenciamento da rota, chamado de ´angular-route.js´.

Atenção

Se você olhar bem a inclusão do arquivo javscript, irá perceber que incluímos o arquivo `angular.js` e não `angular.min.js`.

Isso foi feito para mostrar que ambas os arquivos funcionam, mas o arquivo "minificado" com a extensão `.min.js` é incompreensível para que possamos detectar erros de javascript.

Já o arquivo `angular.js` é compreensível para nossa leitura e os erros de javascript que por ventura cometemos poderão ser analisados com mais eficiência.

Resumindo, use o arquivo `angular.min.js` somente quando o seu sistema está pronto e em produção. Caso contrário, use o arquivo `angular.js`

Este módulo é criado através da definição de um nome para o **ng app**, ficando desta forma: `ng-app="app"`. Assim, estamos criando um módulo chamado `App` que deve estar definido pela aplicação.

Como podemos ver, o arquivo `index.html` não tem nenhum conteúdo, apenas o cabeçalho e uma `div` que possui a propriedade **ng view**. Esta propriedade configura o AngularJS para que toda a geração de código seja renderizada dentro desta tag.

Esta definição é realizada no arquivo `app.js`, cuja parte inicial está descrita a seguir.

app.js

```
1  var app = angular.module('app',['ngRoute']);
2
3  app.config(['$routeProvider',function($routeProvider){
4          $routeProvider.
5          when('/',{controller:'listController', templateUrl:'list.html'}).
6          when('/edit/:name',{controller:'editController',templateUrl:'form.html'}).
7          when('/new',{controller:'newController', templateUrl:'form.html'}).
8          otherwise({redirectTo:'/'});
9  ]);
10
11 app.run(['$rootScope',function($rootScope){
12         $rootScope.fruits = ["banana","apple","orange"];
13         console.log('app.run');
14 }]);
```

Nesta primeira parte, usamos o método `angular.module` para criar um módulo, cujo o nome é `app`.

O segundo parâmetro é a referência ao módulo `ngRoute`, que é usado para criar as rotas (lembre-se que ele deve ser incuído, conforme cisto na linha 6 do arquivo index.html).

Após criar o módulo e atribuí-lo a variável `app`, usamos o método `config` para configurar o módulo, neste caso estamos configurando uma funcionalidade chamada *Router*, que possui a função de carregar templates e controllers de acordo com uma URI, ou seja, um endereço repassado pelo navegador.

Na linha 5 temos a primeira configuração através do método `when`, que informa ao *Router* que, ao acessar a raiz do endereço web que o arquivo `index.html` está, deve ser carregado o controller `listController` e o template `list.html`.

> Tanto o template quanto o controller serão carregados no elemento html que contém a propriedade **ng-view** do index.html.

Na linha 6 adicionamos mais uma rota, e agora configuramos que quando a URI for `/edit/:name`, o controller `editController` e o template `form.html` serão carregados. O atributo `:name` será uma variável que poderá ser obtida no controller.

Tanto na linha 6 quanto na linha 7 usamos o mesmo template `form.html` que contém um formulário para edição ou inserção de um registro.

Na linha 8, configuramos a rota padrão da URI, que é ativada quando nenhuma rota configurada é encontrada.

Na linha 11 usamos o método `app.run` para configurar a variável `$scope` da aplicação, em um contexto global ao módulo. Neste método criamos a variável `fruits` que possui um contexto global à aplicação.

Continuando no arquivo app.js, temos:

app.js

```
14 app.controller('listController',function ($scope) {
15         console.log('listController');
16 });
17
18 app.controller('editController', function ($scope,$location,$routeParams) {
19         $scope.title = "Editar Fruta";
20         $scope.fruit  = $routeParams.name;
21
22         $scope.fruitIndex = $scope.fruits.indexOf($scope.fruit);
23
24         $scope.save = function(){
25                 $scope.fruits[$scope.fruitIndex]=$scope.fruit;
```

```
26                  $location.path('/');
27          }
28  });
29
30  app.controller('newController', function ($scope,$location,$routeParams) {
31
32          $scope.title = "Nova Fruta";
33          $scope.fruit  = "";
34
35          $scope.save = function(){
36                  $scope.fruits.push($scope.fruit);
37                  $location.path('/');
38          }
39  });
```

Criamos três controllers para a aplicação, sendo que o primeiro, `listController`, ainda não é utilizado, mas pode ser útil em um momento futuro.

Na linha 18 temos o controller `editController` que possui três parâmetros:

- **scope** É o escopo da aplicação que pode ser utilizada no template do controller criado.
- **location** Usada para realizar redirecionamentos entre as rotas
- **routeParams** São os parâmetros repassados pela URI

Na linha 19 preenchemos a variável `$scope.title`, para que o título do formulário mude, lembrando que o formulário é usado tanto para criar um novo registro quando editá-lo.

Na linha 20 pegamos como parâmetro o nome da fruta que foi repassada pela URI. Este valor é pego de acordo com o parâmetro `:name` criado pela rota, na linha 6.

Na linha 22 obtemos o índice do item que está para ser editado. Usamos isso para poder editar o item no método `save` criado logo a seguir.

Na linha 24 temos o método `save` que é usado para "salvar" o registro no array global. Em uma aplicação real estaríamos utilizando ajax para que o servidor persistisse o dado. Na linha 26 redirecionamos a aplicação e com isso outro template será carregado.

Na linha 30, criamos o controller `newController`, que é semelhante ao `editController` e possui o método `save` onde um novo registro é inserido no array `fruits`.

Vamos agora analisar o arquivo `list.html` que é um template e carregado diretamente pelo roteamento do módulo (app.js, linha 5).

list.html

```
1 <h2>Fruits ({{fruits.length}})</h2>
2 <ul>
3         <li ng-repeat="fruit in fruits"><a href="#/edit/{{fruit}}">{{fruit}}</a></li>
4 </ul>
5 <a href="#/new">New</a>
```

O template não necessita informar o seu controller, pois isso já foi feito pelo módulo do AngularJS (app.js, linha 5). Como a variável `fruits` possui um escopo global, ela pode ser usada pelo template e na linha 1, contamos quantos itens existem no array.

Na linha 3 iniciamos a repetição dos elementos que pertencem ao array `fruits` e incluímos na repetição um link para `#/edit/`. Esta é a forma com que o roteamento do AngularJS funciona, iniciando com **#** e repassando a URI logo a seguir. Na linha 5, criamos outro link, para incluir um novo registro. Novamente usamos a URI que será utilizada pelo roteamento do AngularJS.

O último arquivo deste pequeno exemplo é o formulário que irá editar ou inserir um novo registro.

form.html

```
1 <h2> {{title}} </h2>
2 <form name="myForm">
3         <input type="text" ng-model="fruit" name="fruit" required>
4         <button ng-click="save()" ng-disabled="myForm.$invalid">Save</button>
5 </form>
6 <a href="#/">Cancel</a>
```

Na linha 1 usamos o `{{title}}` para inserir um título que é criado pelo controller. O formulário possui apenas um campo cujo **ng-model** é `fruit`, que será utilizado pelo controller (app.js, linhas 25 e 35). Neste formulário também utilizamos **ng-disabled** para que o botão seja ativado somente se houver algum texto digitado na caixa de texto. O botão *save* possui a propriedade **ng-click**, que irá chamar o método `save()` do controller.

Conectando AngularJS ao servidor

Agora que conhecemos um pouco sobre o AngularJS, podemos entender como funciona a sua comunicação com o servidor. Assim como é feito com jQuery e até com javascript puro, a melhor forma de obter e enviar dados para o servidor é através de Ajax e o formato de dados para se usar nesta comunicação é JSON.

Existem diversas formas de conexão entre cliente (neste caso, AngularJS) e servidor, e nesta obra estaremos utilizando um conceito chamado RESTful, que é uma comunicação HTTP que segue um padrão bastante simples, utilizando cabeçalhos HTTP como POST, GET, PUT, DELETE.

Na forma mais simples de comunicação de dados, onde temos um objeto e as ações de criar, editar, listar e deletar objetos, resumimos o padrão RESTful às seguintes ações:

Método	http://site.com/produtos
GET	Listar todos os produtos
POST	Editar uma lista de produtos
PUT	Criar um novo produto na lista de produtos
DELETE	Excluir uma lista de produtos

Método	http://site.com/produto/1
GET	Obter o produto cujo id é 1
POST	Em teoria nao possui funcionalidade
PUT	Edita ou cria um novo produto
DELETE	Exclui um produto cujo o id é 1

As tabelas acima são uma sugestão que pode ser seguido ou não. Mesmo se simplificarmos os métodos acima, o mínimo que podemos estabelecer é que métodos GET não alterem dados, pois um método GET pode ser facilmente acessado através do navegador.

O AgularJS fornece duas formas distintas de trabalhar com estas conexões. A primeira delas, e mais simples, é através do serviço `$http`, que pode ser injetado em um controller. A segunda forma é através do serviço `$resource` que é uma abstração RESTful, funcionando como um *data source*.

Uso do $http

O uso do $http não deve ser ignorado, mesmo que o `$resource` seja mais poderoso. Em aplicações simples, ou quando deseja obter dados de uma forma rápida, deve-se utilizar $http.

> $http é uma implementação ajax através do XMLHttpRequest utilizando JSONP. Iremos sempre usar JSON para troca de dados entre cliente e servidor.

A forma mais simples de uso do $http está descrito no exemplo a seguir:

$http na sua forma mais simples

```
1 $http({method:'GET', url:'/someUrl'}).success(function(data){
2
3 });
```

O método get. pode ser generalizado para:

$http.get

```
1 $http.get('/someUrl').success(function(data){
2
3 });
```

Assim como existe o get, existem os outros também, conforme a lista a seguir:

- $http.get
- $http.head
- $http.post
- $http.put
- $http.delete
- $http.jsonp

Para todos estes métodos, o AngularJS configura automaticamente o cabeçalho da requisição HTTP. Por exemplo, em uma requisição POST os cabeçalhos preenchidos são:

- Accept: application/json, text/plain, * / *
- X-Requested-With: XMLHttpRequest
- Content-Type: application/json

Além dos cabeçalhos, o AngularJS também serializa o objeto JSON que é repassado entre as requisições. Se um objeto é enviado para o servidor, ele é convertido para JSON. Se uma *string JSON* retorna do servidor, ela é convertida em objeto utilizando um parser JSON.

Exemplo com $http

Vamos criar um exemplo utilizando o serviço $http, para obter uma lista de dados e preencher uma tabela. Inicialmente, criamos um arquivo simples no servidor que, em teoria, retornaria com informações de uma lista de objetos em JSON. Como ainda não estamos trabalhando com banco de dados, esta lista será criada diretamente no arquivo, em formato json.

listFruits.html

```
{ "fruits":
        [
                {
                "id": 1,
                "name": "Apple",
                "color": "Red"
                },
                {
                "id": 2,
                "name": "Banana",
                "color": "Yellow"
                },
                {
                "id": 3,
                "name": "watermelon",
                "color": "Green"
                },
                {
                "id": 4,
                "name": "Orange",
                "color": "Orange"
                }
        ]
}
```

Neste exemplo, estamos inserindo o arquivo `listFruits.html` na pasta `c:\wamp\www\http-example\` e podemos acessá-lo através da url `http://localhost/http-example/listFruits.html`. O próximo passo é criar o arquivo index.html, que contém a camada view do exemplo.

index.html

```
<html ng-app="app">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.j\
s"></script>
    <script src="app.js"></script>
  </head>
  <body>
    <div ng-controller="appController">
      <button ng-click="getData()">Get Data</button>
      <h2 ng-show="fruits.length>0">Fruits</h2>
      <ul>
          <li ng-repeat="fruit in fruits" >
             {{fruit.id}} - {{fruit.name}} ({{fruit.color}})
          </li>
      </ul>
    </div>
  </body>
</html>
```

No arquivo `index.html` não temos nenhuma novidade, pois a regra de negócio está no seu controller, onde realizamos o Ajax.

app.js

```
var app = angular.module('app',[]);

app.controller('appController',function ($scope,$http){
        $scope.fruits = Array();

        $scope.getData = function(){
                $http.get("listFruits.html").success(function(data){
                        $scope.fruits = data.fruits;
                        console.log($scope.fruits);
                }).error(function(data){
                        alert("Error...");
                        console.log(data);
                });
        }
});
```

No controller da aplicação, criamos o método `getData`, que é executado quando clicamos no botão "GetData" da view (index.html). Neste método, usamos a variável `$http` para as requisições Ajax. Repare que ela é repassada pelo parâmetro do controller, após o `$scope`. Neste caso, o AngularJS encarrega-se de injetar o serviço *http* nesta variável.

Na linha 6 temos o método `$http.get` onde estamos realizando uma requisição Ajax acessando o arquivo `listFruits.html`, que contém a resposta Json. Neste método, podemos concatenar outro método chamado `success`, que é executado se a requisição HTTP GET for realizada com sucesso. Neste caso, a reposta do servidor estará armazenada na variável `data`, e poderemos acessar a variável `data.fruits` que contém o array de objetos que serão usados no loop da view.

> Na linha 8 temos o uso do `console.log` que pode ser usado em conjunto com o Firebug (Firefox) ou com o Google Chrome, para verificar resultados no console da janela "developer tools". Pode-se usar o *developer tools* para analisar as chamadas ajax também, geralmente na aba *Network*.

Com este simples exemplo conseguimos mostrar como é fácil realizar uma requisição Ajax para obter dados do servidor. Pode-se usar o serviço `$http` para toda a sua aplicação, mas quando estamos utilizando RESTfull, existe outro serviço que torna o acesso ao servidor mais abstrato, chamado de resource, no qual veremos a seguir.

Uso do $resource

Aprendemos a realizar chamadas Ajax através do `$http` e caso haja necessidade, podemos abstrair ainda mais a forma como o AngularJS acessa o servidor.

Neste contexto entra o `$resource` que estabelece um padrão de comunicação RESTfull entre a aplicação e o servidor.

Para que possamos usar esta biblioteca é preciso adicionar o arquivo `angular-resource.js` no documento HTML. Ou seja, além da biblioteca padrão também incluímos a biblioteca `resource`, conforme o exemplo a seguir.

Adicionando a biblioteca angular-resource

```
1 <html ng-app>
2     <head>
3         <title>Lista de compras</title>
4           <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/ang\
5 ular.js"></script>
6         <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angul\
7 ar-resource.js"></script>
8     </head>
```

```
9       <body>
10      </body>
11  </html>
```

Exemplo simples com $resource

Com a biblioteca devidamente instalada, devemos carregá-la, através de um parâmetro na criação do módulo da aplicação. Para acompanhar este processo, vamos criar um exemplo simples, utilizando operações CRUD com o `$resource`.

index.html

```
1   <html ng-app="app">
2   <head>
3           <title>Lista de compras</title>
4           <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.min.\
5   js"></script>
6           <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular-reso\
7   urce.min.js"></script>
8           <script src="app.js"></script>
9   </head>
10  <body ng-controller="phoneController">
11          <input type="text" ng-model="idPhone" value="1"/>
12          <button ng-click="getPhoneById()">GetPhone By Id</button>
13          <hr/>
14          <button ng-click="getPhones()">GetPhones</button>
15          <hr/>
16          <button ng-click="savePhone()">Save Phone</button>
17          <hr/>
18          <button ng-click="deletePhone()">Delete Phone</button>
19  </body>
20  </html>
```

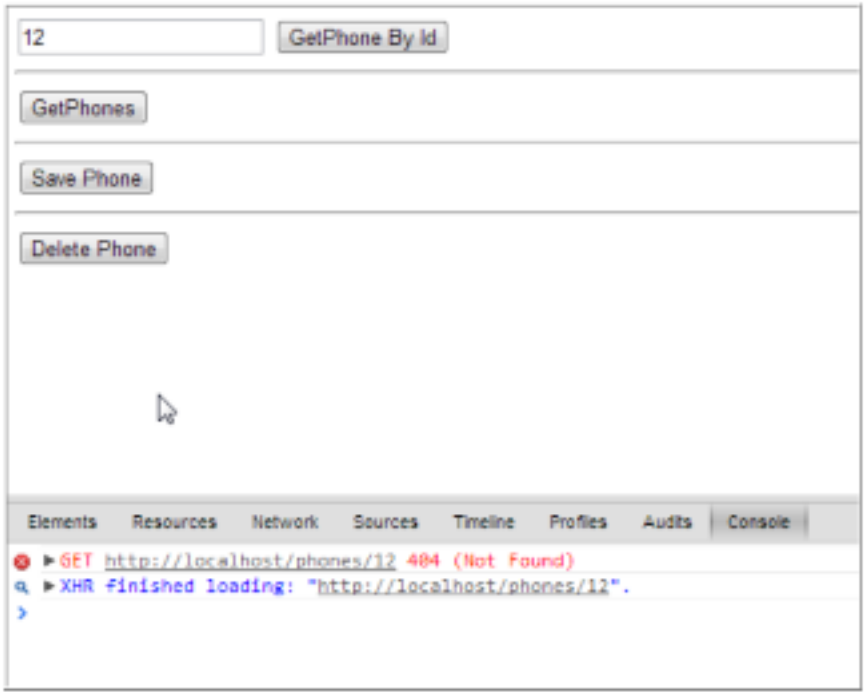
Este código não apresenta nenhuma novidade. Estamos criando o *module* do AngularJS chamado `app` e criamos um formulário com alguns botões no controller `PhoneController`.

app.js

```
1  var $app = angular.module('app',['ngResource']);
2
3  $app.controller("phoneController",function ($scope,$resource){
4
5          var Phone = $resource("/phones/:phoneId");
6
7          $scope.getPhoneById = function(){
8                  Phone.get({phoneId:$scope.idPhone},function(data){
9                          $scope.phone=data;
10                 });
11         }
12
13         $scope.getPhones = function(){
14                 Phone.query(function (data){
15                         $scope.phones = data;
16                 });
17         }
18
19
20         $scope.savePhone =  function(){
21                 p = new Phone();
22                 p.number="1111-2222"
23                 p.save();
24         }
25
26         $scope.deletePhone = function(){
27                 Phone.delete({phoneId:10});
28         }
29
30 });
```

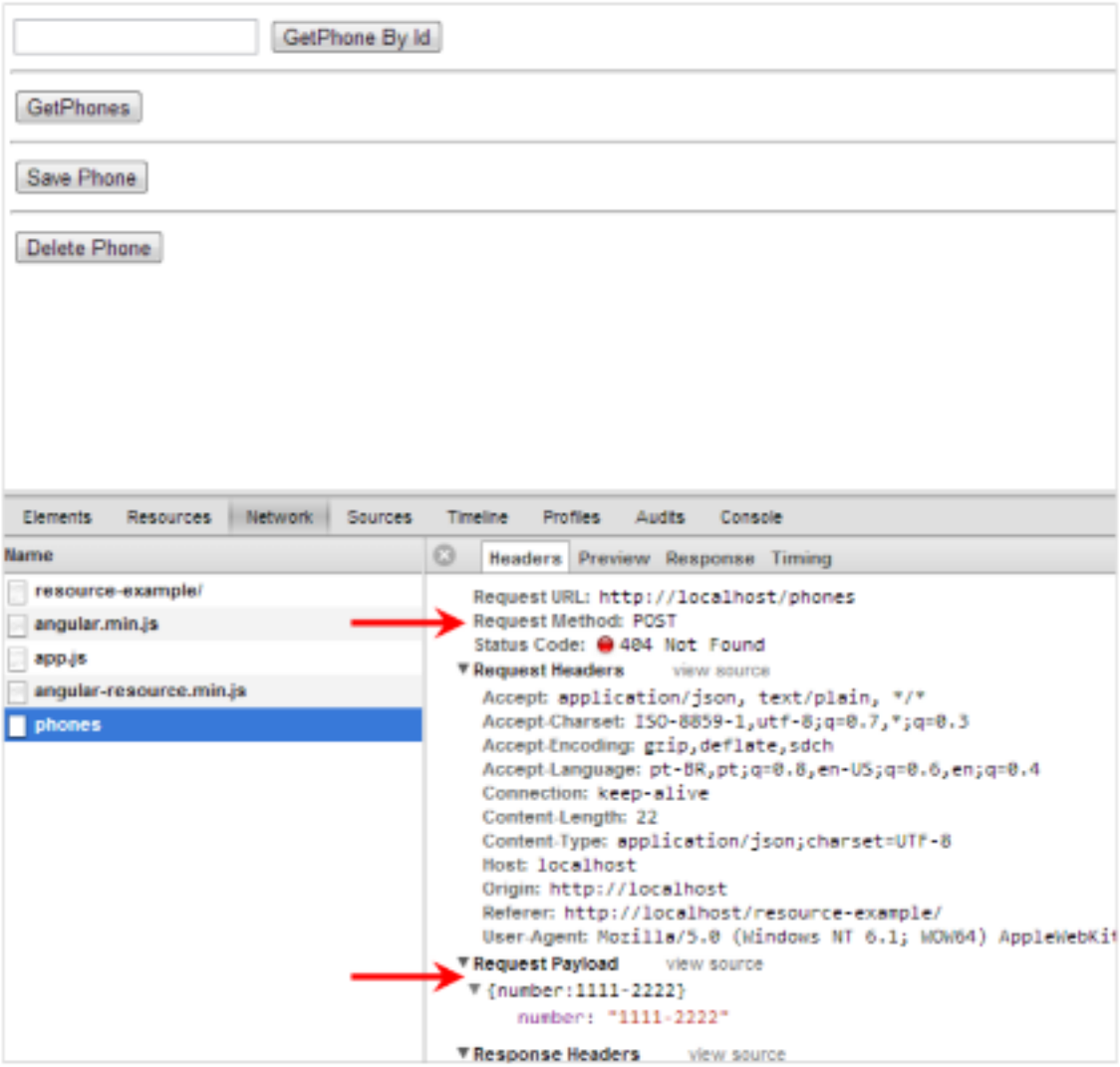
Na definição do `phoneController` adicionamos a variável `$resource` como parâmetro da função, que será injetada pelo AngularJS. Usamos `$resource` para definir a criação da variável `Phone`. Esta variável é criada e configurada como um resource, sendo o primeiro parâmetro a url de acesso ao servidor.

Na linha 7 temos o método `getPhoneById` que é chamado pela view e usa `Phone.get` para realizar uma chamada Ajax ao servidor, conforme a figura a seguir.



Uso do Ajax pelo resources

O mesmo acontece com os outros métodos, e como podemos ver na figura a seguir, quando realizamos um save é realizado um POST no servidor, seguindo as especificações do RESTfull.



Uso do Ajax pelo resources

Nesta figura, temos em detalhe o POST e a variável `number` sendo enviada como json (e não na URL do POST).

> Lembre-se que, para testar o `resources`, você precisa utilizar um servidor web.

Este exemplo está em sua forma mais simples e possivelmente o resource não sera criado em um controller da aplicação.