Etapa 03- Recebendo e filtrando dados

Expressões

Na etapa 1, as expressões já foram apresentadas informalmente ao leitor. Agora serão mostrados alguns detalhes sobre esse recurso do framework.

Se não se recorda da apresentação inicial, reveja esse exemplo.
<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/uajvzmat/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Com o conceito fresco e focando o pensamento nele, certamente surge a pergunta: o que mais pode ser *interpolado* nessa expressão?

Então veja algumas características funcionais das expressões.

- **Contexto**: o contexto de execução da expressão será sempre o escopo
- **Perdão**: o framework perdoa se o resultado interpolado for null ou undefined escondendo-o do usuário
- **Filtragem:** é possível filtrar dados em expressões como foi visto nesse exemplo
- **Invocação de funções**: é possível invocar funções dentro das expressões

Muito útil ter esses recursos à mão, no entanto existem restrições de uso, veja.

- **Instruções de controle de fluxo, loops ou exceções**: ou seja, *if-else, switch, while, for* e *throw-catch* são proibidos
- **Declaração de funções**: não se pode declarar funções quando interpolando dados
- **Criação de RegEx**: é proibido também criar expressões regulares e
- **Criação de objetos com o operador *new***

Portanto, apesar das restrições, ainda é possível criar expressões com operações aritméticas ou relacionais, acessar objetos, suas propriedades ou vetores, tornando o recurso um facilitador nas declarações dos templates HTML.

Para encerrar o assunto uma nota a respeito de eficiência: sempre que possível, utilize a notação *one time binding* (amarração apenas uma vez).

```
{{ ::qualquer_expressão_javascript }}
```

Nesse caso o ciclo de digestão, abordado na etapa 2, fica aliviado com menos uma variável a ser checada. Essa otimização é possível toda vez que se sabe que o valor da variável não será mudado durante o seu uso, por exemplo: títulos de páginas. Veja o exemplo para esclarecer a diferença.
<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/p36jtoxd/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Observe que uma das amarrações da variável *name* ela deixa de ser observada nos ciclos de digestão e a outra não.

Filtros

Mais uma técnica que foi rapidamente introduzida foi a de filtragem. Os filtros são aliados poderos no quesito declaratividade, isto é, eles nos empoderam com capacidade de formatar as views escrevendo muito pouco, sempre é claro de maneira declarativa. Reveja esse exemplo do filtro UPPERCASE.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/yo71y7wy/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

O acabou de ser visto foi um exemplo de filtro de DOM, que usa o caractere **|** após a variável a ser filtrada e em seguida o nome do filtro, que podem ser encadeados.

```
{ expression | filter }}

{{ expression | filter1 | filter2 | ... }}
```

Filtros podem passar parâmetros de filtragem como visto no exemplo acima, basta que após o filtro os parâmetros sejam separados por ' **:** '. Assim:

```
{{ expression | filter:parâmetro1:parâmetro2:... }}
```

No exemplo, o filtro usado é o *filter* (isso mesmo existe um filtro de nome filter) e seu parâmetro é *{ details: { username: 'M' } }*, que é o que deve ser buscado.

```
<div ng-repeat="user in users | filter:{details:{username:'M'}}">
```

Agora serão apresentados os filtros padrões disponíveis no framework.

UPPERCASE e lowercase

Esses filtros dispensam maiores comentários. Eles são usados para converter a string de entrada para o formato de CAIXA ALTA ou caixa baixa, respectivamente.

```
{{ expression | uppercase}}

{{ expression | lowercase}}
```

limitTo

Outro filtro útil é o limitTo, que consegue reduzir as opções disponíveis a um número desejado para a apresentação. Essa redução pode começar de qualquer ponto da lista, inclusive em ordem reversa, ou seja, começando a escolha do ponto inicial de amostras a partir do fim da lista.

```
{{ expressão | limitTo : quantidadeAmostras : índiceInício}}
índiceInício é opcional
```

Dessa sintaxe pode-se apreender os seguintes exemplos:

```
<div ng-repeat="user in users | limitTo:3">
Limita a lista a apenas 3 usuários, começando pelo 1o elemento da lista ou índice 0
```

```
<div ng-repeat="user in users | limitTo:3:5">
Limita a lista a apenas 3 usuários, começando pelo índice 5, 6o elemento da lista
ou
```

```
<div ng-repeat="user in users | limitTo:1:-1">
Limita a lista a apenas o último usuário
```

```
<div ng-repeat="user in users | limitTo:3:-3">
Limita a lista a apenas 3 usuários, começando pelo antepenúltimo (3 últimos)
```

Veja os exemplos.
<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/6yo2ohcr/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

currency

O filtro currency é usado para ajustar valores monetários, isto é, corrigir o formato de apresentação de números que são modelos para valores. Seu uso segue a sintaxe:

```
{{ currency_expression | currency : símbolo : quantidadeCasasDecimais}}
```

Veja o exemplo.
<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/67j0qd6a/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Observe a inclusão de um script i18n (locale pt_br) para adequar o formato de representação do americano para o brasileiro.

json

O filtro json converte objetos para a especificação de um objeto json. JSON significa *Javascript Object Notation*, no entanto, durante a escrita de software em JS, os objetos javascript (POJO - *Plain Old Javascript Object*) possuem uma notação mais relaxada do que o formato JSON. Esse formato é usado por muitas aplicações como formato de serialização para transporte de dados.

A sintaxe usada é:

```
{{ expressão | json : quantidadeEspaçosParaTabulação }}
```

Quando um objeto javascript é colocado em um template usando a expressão com dupla chave, o uso do filtro json é automático e implícito. Por padrão, o filtro usa tabulação de 2 espaços.

```
{{ usuários }} //Filtro json aplicado por padrão.
```

number

O filtro number tem aplicação muito parecida com o filtro currency, formatando números. Veja o exemplo e repare no resultado da divisão por zero, considerada um erro em javascript.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/jcttpg55/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Observe a inclusão de um script i18n (locale pt_br) para adequar o formato de representação americano para o brasileiro.

date

Para formatar datas existe o filtro *date*. Com ele é possível formatar a saída de datas nas *views*. Cuidado com os formatos locais.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/sLmnajts/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Observe a inclusão de um script i18n (locale pt_br) para adequar o formato de representação americano para o brasileiro.

filter

O nome causa estranheza mas está correto. O filtro *filter* restringe as opções disponíveis em uma lista de acordo com um padrão de pesquisa dado. A sintaxe que será usada é a seguinte:

```
{{ expression | filter : padrãoDeBusca}}
```

Esse filtro possui uma um funcionamento mais complexo, o que o torna muito versátil. O padrão de busca é uma string, objeto ou função de filtragem que determinará se um elemento da lista é de interesse do filtro ou não, por meio de comparação.

Quando se utiliza uma string como padrão de busca, o termo é procurado em qualquer propriedade dos itens da lista. Já quando se usa um objeto, deseja-se especificar melhor em que propriedade buscar uma dada informação. Caso se deseje obter, em uma busca usando objeto, o mesmo comportamento da busca por string basta usar a propriedade **$** para busca. Veja no exemplo.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/kj8r9tzg/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

orderBy

O filtro orderBy é outro filtro muito versátil e complexo que permite a ordenação de itens de uma lista. A sintaxe a ser usada é:

```
{{ expressão | orderBy : predicadoOrdenação : inverterOrdem}}
```

A ordenação de de um vetor de números ou de um vetor de strings é trivial e não precisa de um predicado. Quando é realizada a ordenação de objetos é preciso escolher quais predicados (propriedades) definem a ordenação e escolher se a ordem será crescente ou decrescente.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/24bLno0y/embedded/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Formulários

Formulários são uma necessidade em qualquer aplicação que precise de entrada de dados do usuário. Veja alguns detalhes sobre o uso desse recurso a seguir.

form

A diretiva *form* define formulários, quando acompanhado do atributo *name* ele é publicado dentro do escopo, conforme valor determinado no atributo.

Um formulário pode assumir os estados:

- válido/inválido: o formulário é válido se todos os modelos pertencentes a ele são válidos, se um deles for inválido o formulário também será inválido
- dirty/pristine: o formulário quando inicia é sempre pristine (limpo) pois todos os controles no formulário são pristines, se algum controle do formulário sofrer interação ele fica dirty (sujo) e o formulário também ficará
- touched/untouched: o formulário inicia untouched (não tocado) pois seus componentes não foram tocados, a partir do momento que algum momento perde o foco do cursor do usuário ele, e o formulário, passa ao estado de tocado
- empty/not-empty: o formulário ficará no estado empty (vazio) se todos os seus componentes estiverem vazios, caso algum formulário for não vazio, o formulário também será

Quando ele assume esses estados, uma classe CSS3 é associada *ng-valid, ng-invalid, ng-dirty, ng-pristine, ng-touched, ng-untouched, ng-empty* e *ng-not-empty*

ngSubmit

O comportamento padrão de um form é submeter os dados de formulário usando um http/get ou http/post dependendo do atributo *action*. Portanto para anular esse comportamento não deve ser usado, em hipótese alguma, o atributo *action*. Para processar um formulário, deve-se usar o *ng-submit* no elemento form ou o *ng-click* em um input[type=submit], mas é importante usar apenas um deles para que não haja uma dupla submissão. Veja o exemplo.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/L2xgusr7/embedded/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Input, textarea e select

As diretivas input, textarea e select são os controles disponíveis em HTML5. Com eles pode-se criar aplicações à vontade, lembrando que input pode assumir diversos tipos diferentes:

- checkbox
- date
- datetime-local
- email
- month
- number
- radio
- range
- text
- time
- url
- week

A diretiva textarea cria caixas de texto com múltiplas linhas.

Com a diretiva select é possível criar caixas de opções e populá-la dinamicamente com a diretiva *ng-options*.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/9rspndnq/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

ngChange

Essa diretiva é muito interessante está relacionada ao evento *onchange*. Ela é disparada imediatamente, quando ocorre a mudança do modelo, diferentemente do evento nativo javascript

que só é ativada quando o elemento HTML perde o foco ou quando o usuário pressiona o botão *enter*.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/vqkjf8en/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

ngDisabled

A diretiva *ng-disabled* permite mudar dinamicamente o atributo *disabled* em elementos HTML. Veja o exemplo.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/c7615wmx/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

ngMaxlength e ngMinlength

As diretivas *ng-maxlength* e *ng-minlength* adicionam ao ngModel que pertencem validadores da quantidade máxima e mínima de caracteres permitida no campo, respectivamente. Veja o exemplo.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/mk0mdebx/embedded/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

ngModel e ngModelOptions

A diretiva ng-model já é velha conhecida, no entanto está na hora de conhecer as opções disponíveis para o gerenciamento de modelos. ngModel amarra um input, select ou textarea a uma propriedade do escopo. Além disso ng-model também é responsável por:

- Prover validação quando atributos como *required, number, email* ou *url* são utilizados
- Manter o estado do controle (elemento HTML) que podem ser:
    - válido/inválido: definido pelos validadores aplicados no controle
    - dirty/pristine: controle teve valor inicial alterado ou não
    - touched/untouched: controle teve foco e perdeu (tocado) ou não
- Setar ou remover classes css: *ng-valid, ng-invalid, ng-dirty, ng-pristine, ng-touched, ng-untouched, ng-empty* e *ng-not-empty*
- Registrar o controle com o formulário pai

Veja a transição desses valores em um formulário.
<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/ha6w1fmj/embedded/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Para complementar ou modificar o comportamento padrão de ngModel, existem opções que podem ser usadas na diretiva auxiliar *ng-model-options*, duas delas são:

- updateOn: string que determina a quais eventos o controle deve responder, pode ser fornecido mais de um evento ou o evento fictício *default* que representa os eventos padrões do controle
- debounce: inteiro que define o intervalo a ser esperado pelo controle para atualizar o modelo. Os diferentes eventos do controle podem ser configurados com intervalos de atualização diferentes, nesse caso utiliza-se um objeto com o tempo pra cada propriedade (a propriedade terá o nome do evento)

Pode se usar ngModelOptions em qualquer elemento HTML, elementos com a diretiva ngModel irão buscar ngModelOptions do parente mais próximo, caso não possua a sua própria.

Veja no exemplo o uso das duas.
<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/7Lg46x5o/embedded/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

ngReadonly

Esta diretiva habilita o atributo *readonly* no elemento onde a diretiva foi aplicada, passando a permitir apenas leitura no elemento. Ela deve ser aplicada em elementos input que não possam ser alteradas pelo usuário. Veja o exemplo.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/r766Lroy/embedded/html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

ngRequired

A diretiva *ng-required* adiciona ao ng-model do elemento o validador required. Esse validador exige que seja fornecido algum valor para o controle para que ele seja considerado válido. Veja o exemplo.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/eL1utzjg/embedded/html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Serviços e API

" Serviços em AngularJS são objetos substituíveis que são conectados à aplicação por meio do mecanismo de injeção de dependências."

Fonte: *https://docs.angularjs.org/guide/services*

Você deve usar serviços para organizar e compartilhar seu código pela aplicação. Serviços são:

- Lazy instantiated: possuem um carregamento por demanda, portanto um objeto serviço só é criado quando precisa ser injetado em algum outro componente
- Singletons: Todos os componentes que dependem do serviço recebem a mesma referência, isto é, um serviço possui apenas uma instância em toda a aplicação

Para criar serviços é preciso registrar o nome do serviço e a função de fabricação do serviço. Essa função retorna um objeto ou uma função que é injetada em componentes (controladoras, filtros, diretivas ou outros serviços).

```
var myModule = angular.module('myApp', []);
myModule.factory('serviceId', function() {
  var novaInstânciaDoServiço;
  // aqui se contrói a novaInstânciaDoServiço
  return novaInstânciaDoServiço;
});
```

É importante notar nesse exemplo que não se registra o objeto do serviço e sim a função fábrica do serviço, logo a função fábrica é única mas ela pode criar vários exemplares.

Uma das grandes forças do framework está em poder construir serviços, no entanto nesta etapa apenas será apresentado ao leitor alguns dos inúmeros serviços fornecidos pelo framework. O objetivo é que após a leitura você seja capaz de usá-los com fluência.

$document e $window

Esses serviços são embalagens criadas pelo framework para disponibilizar com segurança, as variáveis globais *document* e *window*, respectivamente. A documentação do framework recomenda nunca acessar objetos globais de dentro do framework diretamente.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/nzsdcq24/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

$log

O serviço *$log* provê acesso seguro à API do console do navegador. Com ele é possível imprimir no console mensagens do tipo: *debug, log, info, warning* e *error*.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/82xjxjea/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

$interval e $timeout

Esses serviços permitem a execução adiada de uma função (com um delay de tempo). O $timeout registra um temporizador, executa uma dada função quando termina o delay desejado e encerra suas atividades.

O serviço $interval por sua vez, adiciona o temporizador, executa a função no término do tempo e adiciona outro intervalo, repetidamente, até que seja cancelado.

Cuidado ao usar o serviço $interval. Mesmo quando uma controladora que utiliza esse serviço é destruída, o serviço ainda continua registrado e sendo executado.

É importante lembrar que o intervalo de tempo não é garantido em javascript, no término do temporizador, a execução da função desejada vai entrar na fila, podendo haver um delay entre o estouro e o início da execução.

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/f8bnyjn9/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

Fonte: http://ejohn.org/blog/how-javascript-timers-work/

$http

"O serviço $http é um serviço core do AngularJS que facilita a comunicação com servidores HTTP remotos por meio da API XMLHttpRequest do navegador."

Fonte: https://docs.angularjs.org/api/ng/service/$http

O serviço é baseada na API de promessas do framework. A chamada para do serviço $http recebe apenas um parâmetro, um objeto de configuração.

```
$http({
  method: 'GET',
  url: '/algumaUrl'
}).then((response) => {
    // essa função vai ser chamada com a resposta http, assim que ela chegar
  })
  .catch((error) => {
    // essa função é chamada quando ocorre um erro
```

```
  });
```

Existem alguns métodos no objeto $http que facilitam o seu uso, que são:

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

O exemplo acima, como é uma chamada que usa o método GET, ficaria assim.

```
$http.get('/algumaUrl')
  .then((response) => {
    // essa função vai ser chamada com a resposta http, assim que ela chegar
  })
  .catch((error) => {
    // essa função é chamada quando ocorre um erro
  });
```

Veja nesse exemplo como buscar usuários nesse exemplo.

- HTTP/GET para '/users' retorna um vetor de usuários
- HTTP/POST para '/user' passando o nome do usuário, retorna status 200 se der certo

<iframe width="100%" height="300" src="//jsfiddle.net/design_educacional/ycp2z5r1/embedded/js,html,result/" allowfullscreen="allowfullscreen" frameborder="0"></iframe>

$httpBackend (ngMockE2E)

O serviço $httpBackend fornece um backend HTTP falso, isto é, ele recebe todas as requisições HTTP e responde para o frontend.
Não é objetivo deste curso ensinar o leitor a utilizar esse serviço, mas nos trabalhos será provido um backend forjado, usando esse serviço, para que você aluno possa trabalhar exclusivamente com o frontend.