Devenez un ninja avec AngularJS - extrait gratuit

Ninja Squad

Table of Contents

Devenez un ninja avec

AngularJS

ninja squad

Le chapitre qui suit vous donnera une idée du contenu et de la mise en forme du livre. Enjoy !

Chapter 1. Filtres

1.1. Principe

L'une des fonctionnalités les plus appréciables et méconnues d'Angular réside dans les filtres disponibles. Dans toute application web, il est courant de devoir transformer des valeurs à l'affichage, ou de filtrer ou réordonner une collection. Les filtres Angular répondent à ce problème et sont applicables à la fois en HTML ou en JavaScript, à un seul élément ou un tableau. En HTML, la syntaxe se rapproche du style Unix, où l'on peut chaîner les commandes à l'aide du pipe. Par exemple, on peut appliquer un filtre nommé 'uppercase' sur une expression de la façon suivante :

```
{{ expression | uppercase }}
```

On peut également chaîner plusieurs appels de filtre :

```
{{ expression | uppercase | trim }}
```

Certains filtres prennent des paramètres : c'est le cas du filtre 'number', qui nécessite de préciser le nombre de chiffres suivant la virgule. Pour passer les paramètres au filtre, il suffit de procéder comme suit :

```
{{ expression | number:2 | currency:'$'}}
```

Si un filtre prend plusieurs paramètres, alors on les sépare également avec des `:`. C'est le cas du filtre 'orderBy' qui permet de trier un tableau d'objets selon un de leur champ, et dont le deuxième paramètre permet d'inverser le tri.

```
{{ array | orderBy:'name':true }}
```

Il est également possible d'invoquer les filtres depuis le code JavaScript :

```
var toUppercase = $filter('uppercase');
toUppercase('hello'); // HELLO
```

Angular va alors retrouver la fonction de filtre correspondant à la chaîne de caractères passée en paramètre de la méthode `$filter()`, que l'on peut récupérer et appliquer comme bon nous semble.

1.2. Filtres disponibles

Angular propose par défaut certains filtres communs :

- number : permet de préciser le nombre de chiffres après la virgule à afficher (arrondi au plus proche).

```
{{ 87.67 | number:1 }} // 87.7
```

```
{{ 87.67 | number:3 }} // 87.670
```

- currency : permet de préciser la monnaie.

```
{{ 87.67 | currency:'$' }} // $87.67
```

- date : permet de formatter l'affichage des dates, en précisant un pattern. On retrouve l'écriture classique de pattern :

```
{{ today | date:'yyyy-MM-dd' }} // 2013-06-25
```

Un certain nombre de patterns sont disponibles (avec un rendu différent selon la locale) :

```
{{ today | date:'longDate' }} // June 25, 2013
```

Depuis la version 1.4, il est possible de convertir la date dans la timezone de votre choix :

```
UTC -> {{ today | date:'short':'UTC' }}
Paris -> {{ today | date:'short':'GMT+0200' }}
Auckland -> {{ today | date:'short':'GMT+1300' }}
```

- lowercase/uppercase : de façon assez évidente, ces filtres vont convertir l'expression en majuscules ou minuscules.

```
{{ "Cedric" | uppercase }} // CEDRIC
```

```
{{ "Cedric" | lowercase }} // cedric
```

- json : moins connu, ce filtre permet d'afficher l'objet au format JSON. Il est également moins utile, car, par défaut, afficher un object avec la notation '{{ }}' convertit l'objet en JSON.

```
{{ person | json }} // { name: 'Cedric', company: 'Ninja Squad'}
```

```
{{ person }} // { name: 'Cedric', company: 'Ninja Squad'}
```

- limitTo : ce filtre s'applique quant à lui à un tableau, en créant un nouveau tableau ne contenant que le nombre d'éléments passés en paramètre. Selon le signe de l'argument, les éléments sont retenus depuis le début ou la fin du tableau.

```
{{ ['a','b','c'] | limitTo:2 }} // ['a','b']
{{ ['a','b','c'] | limitTo:-2 }} // ['b','c']
```

Depuis la version 1.4, il est également possible de passer un deuxième argument pour indiquer à partir de quel élément le filtre doit s'appliquer. Si cet index est négatif, le filtre s'appliquera à partir de la fin du tableau.

```
{{ ['a','b','c'] | limitTo:2:1 }} // [b, c]
{{ ['a','b','c'] | limitTo:2:-1 }} // [c]
{{ ['a','b','c'] | limitTo:-2:-1 }} // [a, b]
```

- orderBy : là encore un filtre s'appliquant à un tableau. Celui-ci va trier le tableau selon l'accesseur passé en paramètre. L'accesseur peut être une chaîne de caractères représentant une propriété des objets à trier ou une fonction. L'accesseur sera appliqué sur chaque élément du tableau pour donner un résultat, puis le tableau de ces résultats sera trié selon l'ordre défini par les opérateurs `<`, `>`, `=`. Une propriété peut être précédée du signe `-` pour indiquer que le tri doit être inversé. A la place d'une simple propriété, il est possible de passer un tableau de propriétés ou de fonctions (chaque propriété ou fonction supplémentaire servant à affiner le tri primaire). Un second paramètre, booléen, permet quant à lui d'indiquer si le tri doit être inversé.

```
var jb = {name: 'JB', gender: 'male'};
```

```
var cyril = {name: 'Cyril', gender: 'male'};
var agnes = {name: 'Agnes', gender: 'female'};
var cedric = {name: 'cedric', gender: 'male'};
$scope.ninjas = [jb, cyril, agnes, cedric];
```

On peut ordonner la liste par genre :

```
// order by the property 'gender'
{{ ninjas | orderBy:'gender'}} // Agnes,JB,Cyril,Cédric
```

Ou par nom :

```
// order by the property 'name'
{{ ninjas | orderBy:'name'}} // Agnes,Cédric,Cyril,JB
```

Pour faire un tri plus compliqué, il est possible de définir son propre comparateur :

```
// order by a function (lowercase last)
$scope.lowercaseLast = function(elem){
  return elem.name === elem.name.toLowerCase()
};
```

```
{{ ninjas | orderBy:lowercaseLast }} // Agnes,JB,Cyril,cedric

// order by an array of properties or functions
{{ ninjas | orderBy:['-gender','name'] }} // cedric,Cyril,JB,Agnes
```

1.3. Filtre dynamique

Nous avons vu les principaux filtres, mais il manque un qui se nomme … ‘filter’. Et c’est l’un de nos préférés !

Ce filtre agit sur un tableau pour en retourner un sous-ensemble correspondant à l’expression passée en paramètre. L’expression la plus simple consiste à passer une chaîne de caractères : le filtre va alors retenir tout élément du tableau dont une propriété contient la chaîne de caractères en question.

Il devient alors très simple de faire une recherche dans un tableau dans votre application AngularJS.

Supposons que vous ayez une liste de personnes. Au hasard une équipe de ninjas.

```
$scope.ninjas = [
  { 'name': 'Agnes', 'superpower': 'Java Champion', 'skills':
 ['Java', 'JavaEE', 'BDD']},
  { 'name': 'JB', 'superpower': 'Stack Overflow Superstar', 'skills':
 ['Java', 'JavaScript', 'Gradle']},
  { 'name': 'Cyril', 'superpower': 'VAT specialist' /*I'm joking
 buddy*/, 'skills': ['Java', 'Play!']},
  { 'name': 'Cédric', 'superpower': 'Hype developer', 'skills':
 ['Java', 'JavaScript', 'Git']},
];
```

Maintenant vous voulez les afficher dans un tableau. Facile, un coup de 'ng-repeat' et l'affaire est pliée.

```
<tr ng-repeat="ninja in ninjas">
  <td>{{ ninja.name }}</td>
  <td>{{ ninja.superpower }}</td>
  <td>{{ ninja.skills.join(',') }}</td>
</tr>
```

Maintenant nous voulons ajouter notre filtre, qui s'utilise comme les autres filtres, sur notre tableau de ninjas. Pour cela nous ajoutons le pipe suivi du filtre 'filter' avec un paramètre nommé `search` qui contiendra la chaîne de caractères à rechercher dans le tableau (et ne retiendra donc que les ninjas qui satisfont la recherche).

```
<tr ng-repeat="ninja in ninjas | filter:search">
  <td>{{ ninja.name }}</td>
  <td>{{ ninja.superpower }}</td>
  <td>{{ ninja.skills.join(',') }}</td>
</tr>
```

Nous allons ajouter un input qui aura comme modèle 'search' et rendra donc la recherche dynamique.

```
<input ng-model='search' placeholder='filter...'/>
```

Et voilà! Le tableau est filtré dynamiquement ! Essayez, on vous attend ici ![1]

[1] http://embed.plnkr.co/TGuB6y/preview

C'est quand même la super classe, non ?

Il est également possible de limiter la recherche à certaines propriétés de l'objet, en passant un objet au filtre plutôt qu'une chaîne de caractères. Par exemple on peut chercher seulement le nom, en transformant l'input comme suit :

```
<input ng-model='search.name' placeholder='filter...'/>
```

Il est également possible de passer une fonction à évaluer contre chaque élément du tableau plutôt qu'une chaîne de caractères ou un objet.

Enfin, ce filtre peut prendre un deuxième paramètre, pouvant être un booléen indiquant si la recherche doit être sensible à la casse (par défaut, elle ne l'est pas) ou une fonction définissant directement comment comparer l'expression avec les objets du tableau. Si vous voulez une recherche sensible à la casse :

```
<tr ng-repeat="ninja in ninjas | filter:search:true">
  <td>{{ ninja.name }}</td>
  <td>{{ ninja.superpower }}</td>
  <td>{{ ninja.skills.join(',') }}</td>
</tr>
```

Avec une combinaison de 'filter' et de 'orderBy', vous avez toutes les cartes en main pour faire de superbes listes ou tableaux en HTML, parfaitement dynamiques, et, vous l'avez remarqué, sans écrire une ligne de JavaScript !

1.4. Créer ses propres filtres

Il est également possible de créer ses propres filtres et cela peut être parfois très utile. Il existe une fonction pour enregistrer un nouveau filtre dans votre module (comme nous l'avons vu avec les controllers). Elle se nomme 'filter' et prend comme argument le nom du filtre que vous utiliserez et une fonction. Cette fonction doit renvoyer une fonction prenant comme paramètre l'élément à filtrer. Oui, une fonction qui retourne une fonction, quand on ne fait pas souvent de JavaScript, ça fait bizarre !

Par exemple, nous utilisons souvent la librairie moment.js[2] pour la gestion des dates en JavaScript. Vous ne connaissez pas ? Elle est vraiment très bien faite, et nous est devenue indispensable dès qu'un projet veut faire des choses un peu avancée avec des dates (le filtre par défaut d'Angular sur les dates est assez limité…).

[2] http://momentjs.com/

Nous avons donc créé un filtre 'moment' pour Angular, qui nous permet d'utiliser facilement cette librairie dans nos templates.

```
app.filter('moment', function() {
  return function(input, format) {
    format = format || 'll';
    return moment(input).format(format);
  }
});
```

Notre filtre se nomme donc 'moment' et prend 2 paramètres possibles :

- un input, la date à formatter (tous les filtres ont au moins ce paramètre).
- un format, qui est optionnel, et est initialisé à 'll' (l'un des formats offerts par moment.js) si il n'est pas défini.

Nous pouvons ensuite utiliser ce filtre dans nos templates :

```
    {{ '2013-10-15' | moment }} // Oct 15 2013
    {{ '2013-10-15' | moment:'LL' }} // October 15 2013
```

L'input est formaté avec 'll' par défaut ou celui précisé, 'LL' dans le deuxième exemple.

Vous savez tout sur les filtres !

Quiz!

> Vous pouvez tester vos connaissances avec ce quiz sur les filtres[3].

[3] https://quizzie.io/quizzes/5MCEA_ciV0qsRTgr-miXQGH51425648689/