BEEVA - CURSO DE ANGULAR 2015
TEORÍA (KATA NIVEL AVANZADO)
Fernando Castro / Juan Ferrer

FILTROS

DEFINICIÓN

Se trata de una forma de modificar los datos que vamos a mostrar en pantalla. Se caracterizan por la **rapidez y sencillez** con que se pueden aplicar a diferentes tipos de valores.

Podremos usar los que proporciona angular como nativos, pero también podremos crear nuestros propios filtros personalizados.

TIPOS DE FILTROS NATIVOS

Vamos a ver los diferentes tipos de filtros nativos, su descripción y ejemplos para entender perfectamente su funcionamiento.

FILTRAR ARRAYS
FILTER

Se trata de obtener un subconjunto de elementos de un array, devolviendo un nuevo array.
El array podrá contener cadenas, objetos simples, objetos a varios niveles…
Se podrá usar el símbolo de negación: !.

Bastará con añadirlo a un array determinado, utilizando para ello el símbolo | como separación. Veamos su estructura:

```
Html: {{array | filter : expression : comparator}}
Js: $filter('filter')(array, expression, comparator)
```

- La propiedad **expression** puede ser
  - **String**:
    - se filtrará por una cadena simple

      ```
      Js: var lugares = ['casa', 'escuela', 'parque', 'casa_1', 'casa_2'];
      Html: <div>{{lugares | filter: 'casa'}}</div>
      Salida: 'casa', 'salon_casa', 'entrada_casa'
      ```

  - **Propiedad/es de un objeto**:
    - se filtrará por las propiedades
    - se puede combinar la búsqueda de varias propiedades (AND)
      filter: {prop1: tx, prop2: tx}
    - se puede utilizar el símbolo $ para recuperar cualquier campo (pudiendo usarlo en objetos a varios niveles), y combinarlo con búsqueda por cadena.

      ```
      Js: var per = [{nom: 'Juan'}, {nom: 'Jon'}, {nom: 'Jonathan'}];
      Html: <div>{{per | filter: {nom: 'Jon'}}</div>
      Salida: {nom: 'Jon'}, {nom: 'Jonathan'}
      ```

- **Función** (value, index, array):
  - Permite crear nuestra propia expresión, llamando a la función por cada elemento del array, pasando como argumentos el valor a evaluar, el índice y el propio array
  - Nos permite hacer combinación de propiedades utilizando no sólo AND, sino también OR (que hasta ahora no habíamos podido utilizar)

```
Js: var per = [{nom: 'Juan'}, {nom: 'Jon'}, {nom: 'Jonathan'}];

      $scope.findNames = function(value, index, array) {
            if (value.nom === 'Juan' || value.nom === 'Jon') {
                  return true;
            } else {
                  return false;
            }
      }

Html: <div>{{per | filter: findNames}}</div>
Salida: {nom: 'Juan'}, {nom: 'Jon'}
```

- La propiedad **comparator** permite evaluar de forma estricta la comparación entre el valor actual y el valor evaluado. Puede tomar los siguientes valores:
  - **true:** el valor a buscar tiene que ser idéntico
  - **false (es el valor por defecto):** tiene que contener el valor a buscar
  - **function (valor actual, valor esperado):** la función devolverá true o false

ORDER BY

Permite ordenar un array determinado siguiendo unas directrices especificadas.

```
Html: {{array | orderBy : expression : reverse}}
Js: $filter('orderBy')(array, expression, reverse)
```

- La propiedad **expression** especifica el campo por el que se quiere hacer la ordenación
  - podrá ser una cadena, un array de cadenas si queremos ordenar por varios campos o el resultado de una función
  - utilizaremos el signo – si queremos que la ordenación sea descendente
  - no tiene en cuenta los acentos (orden ASCII)

```
Js: var players = [{name: 'Finidi'}, {name: 'Batista'}, {name: 'Guti'}];
Html: <div>{{players | orderBy: 'name' }</div>
Salida: 'Batista', 'Finidi', 'Guti'
```

- Especificando la propiedad **reverse** conseguiremos que se invierta el orden establecido.

```
Js: var players = [{name: 'Finidi'}, {name: 'Batista'}, {name: 'Guti'}];
Html: <div>{{players | orderBy: 'name': reverse }</div>
Salida: 'Guti ', 'Finidi', ' Batista'
```

LIMIT TO

Partiendo de un array, número o cadena, permite crear un nuevo array o cadena con el número de elementos especificados.

```
Html: {{array | limitTo : limit : begin}}
Js: $filter('limitTo')(array, limit, begin)
```

- Mediante la propiedad **limit** podremos especificar la longitud del resultado a devolver
  - filtrará desde el principio si el valor es positivo, haciéndolo desde el final en caso de ser negativo
  - si el número sobrepasa la longitud total devolverá el total, al igual que si se trata de un valor sin definir

```
Js: var numbers = [1,2,3,4,5,6];
Html: <div>{{numbers | limitTo: 4}</div>
Salida: '1', '2', '3', '4'
```

- Con la propiedad **begin** se indicará el índice desde donde se empezará a filtrar
  - Si es positivo contará desde el principio, haciéndolo desde el final si se trata de un número negativo.
  - Valor por defecto será 0

```
Js: var numbers = [1,2,3,4,5,6];
Html: <div>{{numbers | limitTo: 4: -1}</div>
Salida: [2,3,4,5]
```

FORMATEAR DATOS

NUMBER

Se limita el número de decimales, devolviendo un tipo cadena.
Si no se indica el número de limitación, por defecto se mostrará con tres decimales.
También se encarga de adaptar el símbolo de separación según el sistema usado.

```
Html: {{value | number : fractionSize}}
Js: $filter('number')(value, fractionSize)
```

- El **fractionSize** será un entero

```
Js: money  = 12,98754;
Html: <div>{{money | number:3}}</div>
Salida: '12,987'
```

DATE

Se formatea una fecha para mostrarla según las necesidades de visualización.

```
Html: {{value | date : format : timezone}}
Js: $filter('date')(value, format, timezone)
```

- Veamos algunos de los valores (se puede ver el total de posibilidades en la página de Angular: date) que puede tomar la propiedad **format**

    - Para representar el año
        - '**yyyy**': completo
        - '**yy**': dos últimos dígitos

    - Para representar el mes
        - '**MMMM**': nombre completo del mes,
        - '**MMM**': tres primeras letras del mes,
        - '**MM**': 01-12,
        - '**M**': 1-12

    - Para representar el día
        - '**dd**': 01-31
        - '**d**': 1-31

    - Formatos predefinidos (de acuerdo al idioma correspondiente)
        - '**short**': 'M/d/yy h:mm a' (US)
        - **'longDate'**: 'MMMM d, y' (US)

```
Js: date_today  = 1288323623006;
Html: <div>{{ date_today  | date: 'dd/MM/yyyy'}}</div>
Salida: '29/10/2010'
```

- La propiedad **timezone** permite especificar la zona horaria. Si no se indica, se utiliza la actual por defecto.

CURRENCY

Se formatea un número para devolver el símbolo de la moneda y el número de decimales correspondientes al idioma por defecto.

```
Html: {{value | currency : symbol : fractionSize}}
Js: $filter('currency')(value, symbol, fractionSize)
```

- Se pueden incluir las propiedades **symbol** y **fractionSize** para obtener un dato adaptado a nuestras necesidades

```
Js: number  = 12,45674;
Html: <div>{{ number  | currency: $ : 2}}</div>
Salida: '12,45$'
```

LOWERCASE / UPPERCASE

Nos permiten transformar cadenas a minúsculas y mayúsculas respectivamente.

```
Html: {{value | lowercase}}
Js: $filter('lowercase')(value)
```

```
Html: {{value | uppercase}}
Js: $filter('uppercase')(value)
```

```
Js: nameUPP = 'JAMES'; nameLOW = 'james';
Html: <div>{{ nameUPP  | lowercase}}</div>
      <div>{{ nameLOW  | uppercase}}</div>
Salida: 'james'
        'JAMES'
```

PRUEBAS

JSON

Se encargar de transformar un objeto a formato JSON. Este filtro se suele utilizar para realizar pruebas.

```
Html: {{obj | json: spacing}}
Js: $filter('json')(obj, spacing)
```

- La propiedad **spacing** nos permite especificar las tabulaciones del código a mostrar

```
Js: obj = {'name': 'value'}
Html: <div>{{ obj  | json : 3}}</div>
Salida: {
           'name' : 'value'
        }
```

FILTROS PERSONALIZADOS

Para crear nuestros propios filtros definiremos una función de factoría, que devolverá nuestra función al **método filter** de un módulo determinado (que puede ser el principal, uno específico que podemos crear para albergar los diferentes filtros a definir…).

Habrá que comprobar qué tipo de dato es el que nos llega para poder realizar las operaciones correspondientes.

Veamos la estructura principal:

```
angular.module('myFiltersApp', [])

        .filter('nameFilter', function() {

                return function(input, param1, param2, param3) {
                        // code
                }

        });
```

```
Js: obj = {'name': 'value'}
Html: <div>{{ obj  | nameFilter : param1 : param2 : pream3}}</div>
```

BIBLIOGRAFÍA

Se ha recurrido al abundante material disponible en la red para explicar de la mejor manera posible los diferentes conceptos que aparecen en esta documentación:

Página principal de Angular: https://angularjs.org
Filtros: http://cursoangularjs.es/doku.php?id=unidades:05_filtros:00_start