

Angular Js 2016

Nivel Avanzado

BEE PART OF THE CHANGE

Avenida de Burgos 16 D, 28036 Madrid

hablemos@beeva.com

www.beeva.com



DOCUMENTACIÓN

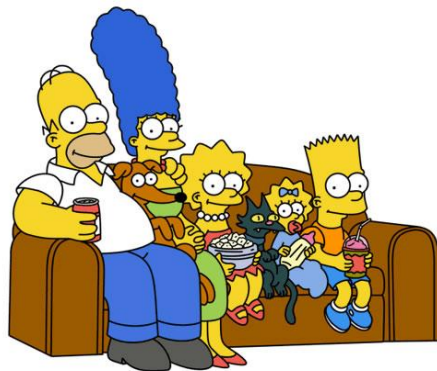
NIVEL AVANZADO



DIRECTIVAS PERSONALIZADAS
SERVICIOS
FILTROS
TRANSCLUSIÓN
HERENCIA DE DIRECTIVAS
ANGULAR 2



DIRECTIVAS PERSONALIZADAS



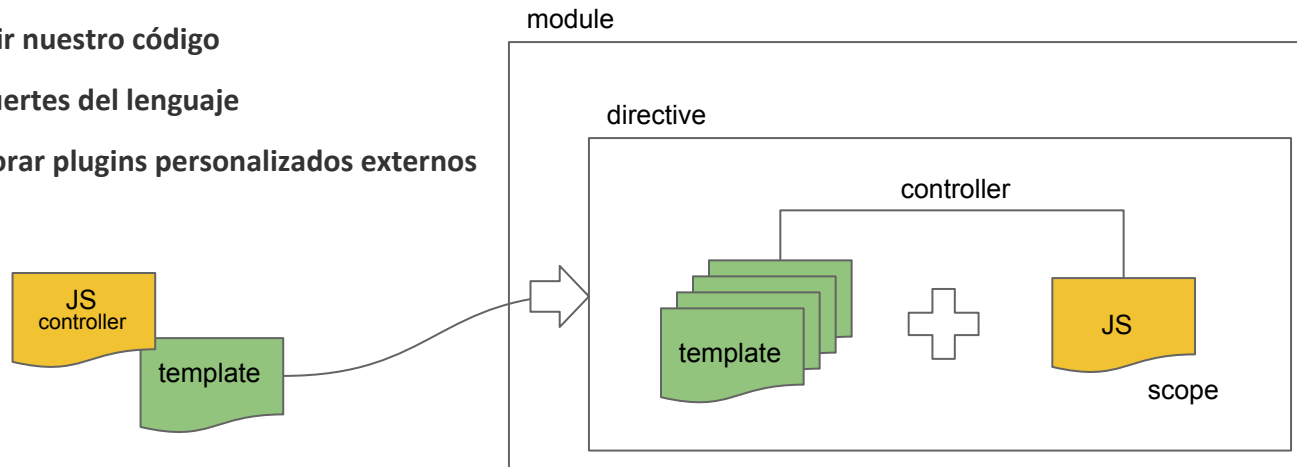
MATT GROENING

DIRECTIVAS PERSONALIZADAS I

Gran libertad a la hora de definir nuestro código

Se trata de uno de los puntos fuertes del lenguaje

La forma más común de incorporar plugins personalizados externos



DIRECTIVAS PERSONALIZADAS II

NOMENCLATURA

Seguiremos las siguientes recomendaciones:

- Asegurarse de utilizar un nombre no existente
- En la definición (js): **Estándar CamelCase** *filterDirective*
- En la llamada (HTML):



general



caso especial

<data-filter-directive></data-filter-directive>

<filter:directive></filter:directive>

<filter_directive></filter_directive>

<filter-directive></filter-directive>

<x-filter-directive></x-filter-directive>

DIRECTIVAS PERSONALIZADAS III

DEFINICIÓN DESDE LA PLANTILLA (HTML)

Cuatro formas posibles:

- Como etiqueta

```
<data-filter-directive></data-filter-directive>
```

- Como clase

```
<div class="data-filter-directive"></div>
```

- Como atributo

```
<div data-filter-directive></div>
```

- Como comentario

```
<!-- directive: data-filter-directive -->
```

DIRECTIVAS PERSONALIZADAS IV

PROPIEDADES DE CONFIGURACIÓN

Vamos a ver las principales propiedades que nos permitirán configurar nuestra directiva

restrict

scope

template

templateUrl

compile / link

replace

DIRECTIVAS PERSONALIZADAS V

PROPIEDADES DE CONFIGURACIÓN

restrict

Llamada a la directiva desde el documento HTML



restrict: 'A'



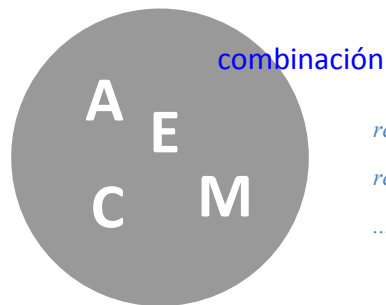
restrict: 'E'



restrict: 'C'



restrict: 'M'



restrict: 'EA'

restrict: 'CM'

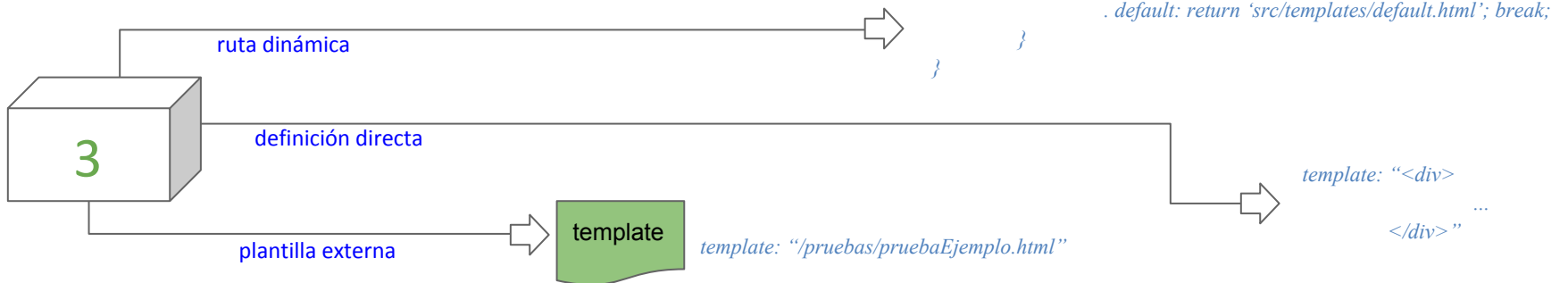
...

DIRECTIVAS PERSONALIZADAS VI

PROPIEDADES DE CONFIGURACIÓN

template

La plantilla asociada a la vista de la directive



DIRECTIVAS PERSONALIZADAS VII

PROPIEDADES DE CONFIGURACIÓN

scope

Espacio de variables a utilizar en nuestra directiva, que se relacionará con el entorno de su padre

- **false:** se usará el scope del padre (funcionamiento por defecto)
- **true:** se crea un nuevo scope, pero heredando del scope del padre
- **{..}:** se crea un nuevo scope, pero cerrado y únicamente heredando del \$rootScope

DIRECTIVAS PERSONALIZADAS VIII

PROPIEDADES DE CONFIGURACIÓN

scope

Dentro de {...} podremos definir variables para que pueda haber comunicación con el padre. Los tipos serán los siguientes:

- **@**: String (unidireccional)
- **=**: variables bidireccionales (? obligatoriedad)
- **&**: funciones

HTML Code

```
<div ng-controller="Controller">  
  <my-customer same-name="naomi" same-name2="{{prueba}}" txt="paco"  
    op-focus=exec_focus() variable="prueba" variable2="texto">  
  </my-customer>  
</div>
```

JavaScript Code

```
scope: {  
  sameName: '@',  
  sameName2: '@',  
  sameName3: '@txt',  
  opFocus: '&',  
  variable: '=',  
  variable2: '='  
}
```

DIRECTIVAS PERSONALIZADAS IX

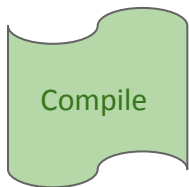
scope	Unidireccional	Bidireccional
Carácter usado	@	=
Valores literales	valor	'valor'
Propiedades padre	{{nombre propiedad}}	nombre propiedad
Si se modifica la propiedad del padre	SI (¿se modifica la propiedad en la directiva?)	SI (¿se modifica la propiedad en la directiva?)
Si se modifica la propiedad de la directiva	NO (¿se modifica la propiedad en e padre?)	SI (¿se modifica la propiedad en el padre?)

DIRECTIVAS PERSONALIZADAS X

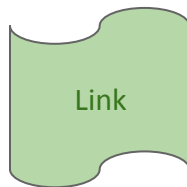
PROPIEDADES DE CONFIGURACIÓN

compile / link

Funciones que utilizaremos para inicializar la directiva, así como para incluir la funcionalidad asociada



- Se ejecuta una sólo vez por directiva
- Para ejecutar código antes de compilar
- Parámetros: elemento que instancia la directiva, sus atributos...



- Se ejecuta una vez por cada instancia de la directiva
- Para ejecutar código después de compilar
- Parámetros: elemento que instancia la directiva, sus parámetros, el scope...

compile: function(elemArr, attrArr, controller, transclude){...}

link: function(scope, elem, attrs, controller, transclude){...}

DIRECTIVAS PERSONALIZADAS XI

PROPIEDADES DE CONFIGURACIÓN

replace

Determina si el contenido de la plantilla se añadirá dentro del tag de la propia directiva

HTML Code

```
<my-customer></my-customer>
```

JavaScript Code

```
replace: false
```

Developer View

```
<my-customer>  
  // content  
</my-customer>
```

HTML Code

```
<my-customer></my-customer>
```

JavaScript Code

```
replace: true
```

Developer View

```
// content
```

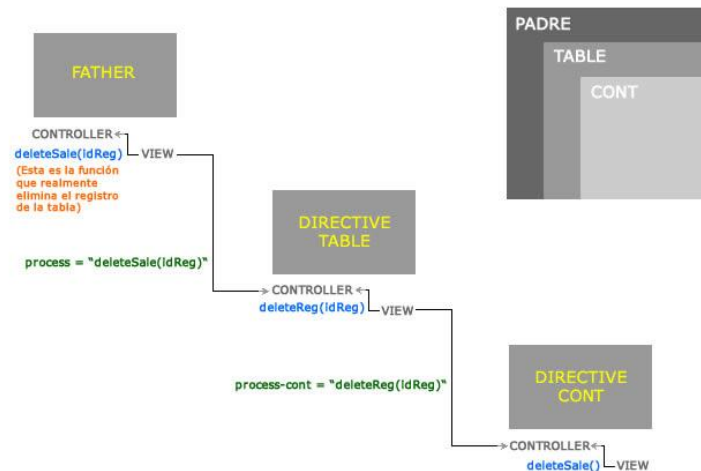
DIRECTIVAS PERSONALIZADAS XII

EJEMPLO DIRECTIVAS A VARIOS NIVELES

<https://www.beeva.com/beeva-view/metodologiasagiles/angular-en-nimble-directivas-a-varios-niveles>

ESQUELETO BÁSICO DIRECTIVA SIMPLE

```
angular.module('nombreModulo').directive('nombreDirectiva', function() {  
  
    return {  
  
        restrict: "",  
        templateUrl: "",  
        scope: {},  
        link: function(scope, element) {  
            // inicialización  
            // funciones  
        }  
    }  
};
```



SERVICIOS




SERVICIOS (I)

DEFINICIÓN

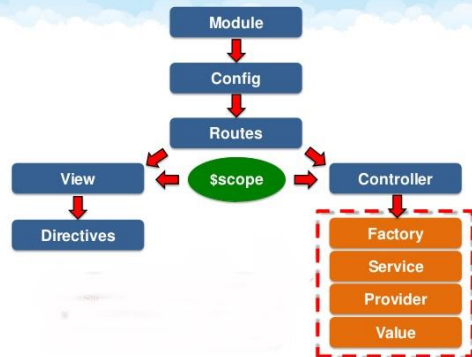
Elementos utilizados para organizar y compartir información entre las diferentes partes de la aplicación

Se usan mediante la inyección de dependencias

Sus características principales:

- Una única instancia en cada dependencia (Singleton) aunque los usemos más de una vez
- El nombre del servicio es independiente del módulo donde se crea
- No interacciona con la propia página (sólo con otros servicios o servidor de datos)
- Nomenclatura CamelCase
- Los servicios nativos siempre empiezan con  (\$http, \$timeout, ..)
- Facilitan el TDD

ANATOMÍA DE UNA APLICACIÓN ANGULAR



SERVICIOS (II)

NATIVOS

Servicios proporcionados por Angular que nos permiten agilizar algunos procesos:

\$http	\$filter	\$interval
\$timeout	\$location	\$animate
\$anchorScroll	\$q	

SERVICIOS (III)

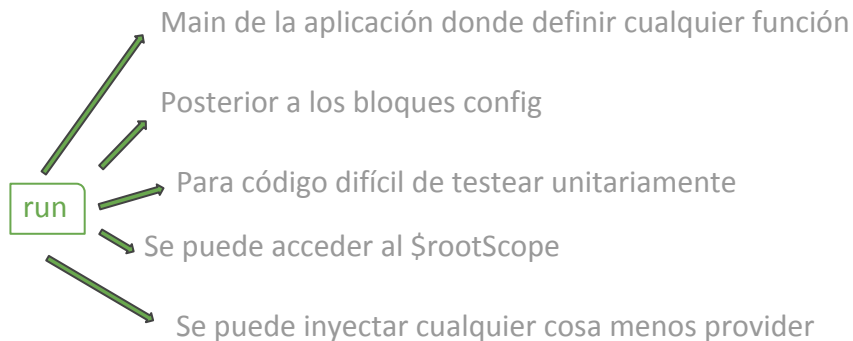
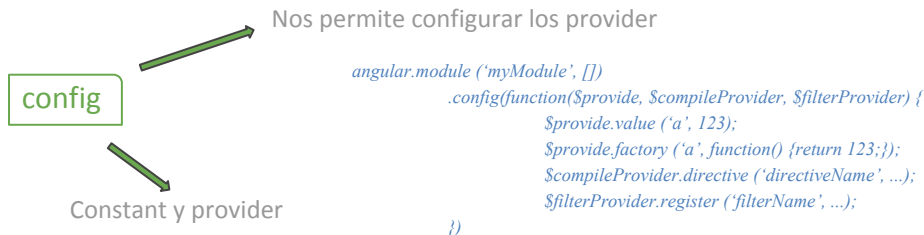
DOS NUEVOS ARTEFACTOS: BLOQUES

Se usan a través de sus métodos, del mismo nombre

Se definen dentro de un módulo, siendo generales para toda la aplicación

Se ejecutarán al inicio del programa

Se pueden crear tantos bloques como sean necesarios



SERVICIOS (IV)

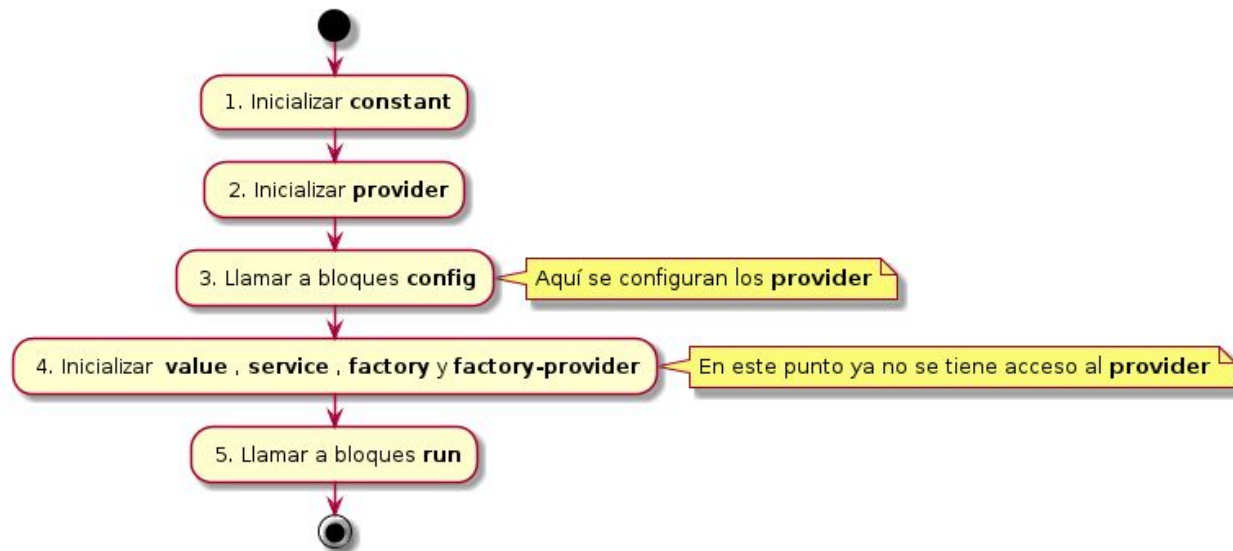
PERSONALIZADOS

Veamos los tipos de servicios personalizados que Angular nos permite definir:

VALUE	SERVICE	PROVIDER
CONSTANT	FACTORY	

SERVICIOS (V)

PASOS AL INICIAR LA APLICACIÓN



SERVICIOS (VI)

VALUE

Nos permite crear valores que podremos inyectar en cualquier parte de la aplicación, excepto config y provider

Se define llamando al método value del módulo

Se pasa directamente el valor que debe tener el servicio

Se pueden definir objetos, funciones, cadenas, números...

```
angular.module('myModule', [])  
  
  .value('miServicioValor', {  
  
    VARIABLE1: 'dato 1',  
    VARIABLE2: 'dato 2'  
  
  });
```

SERVICIOS (VII)

CONSTANT

Misma funcionalidad que `value`, pero se puede utilizar en cualquier sitio

```
angular.module('myModule', [])  
  
  .constant('miServicioConstante', {  
  
    VARIABLE1: 'dato 1',  
    VARIABLE2: 'dato 2'  
  
  });
```

SERVICIOS (VIII)

SERVICE

Se define una clase con su constructor, siendo Angular el encargado de instanciarla internamente

Se pueden definir métodos, variables...

Se pueden inyectar dependencias

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

.value('miValorInit', { VALUE: 2 })

.service('miServicio', [miValorInit, function(miValorInit) {
    this.value = miValorInit.VALUE;

    this.setValue = function(value) {
        this.value = value;
    };

    this.getValue = function() {
        return this.value;
    }
}]);
```


SERVICIOS (IX)

FACTORY

Más adecuado para la creación de objetos complejos

Se puede crear una única instancia dentro de la factoría (con respecto a un constructor)

Se diferencia del tipo 'service' en que devolveremos lo creado dentro de la factoría

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

.factory('miFactoria', function() {

    var data = {

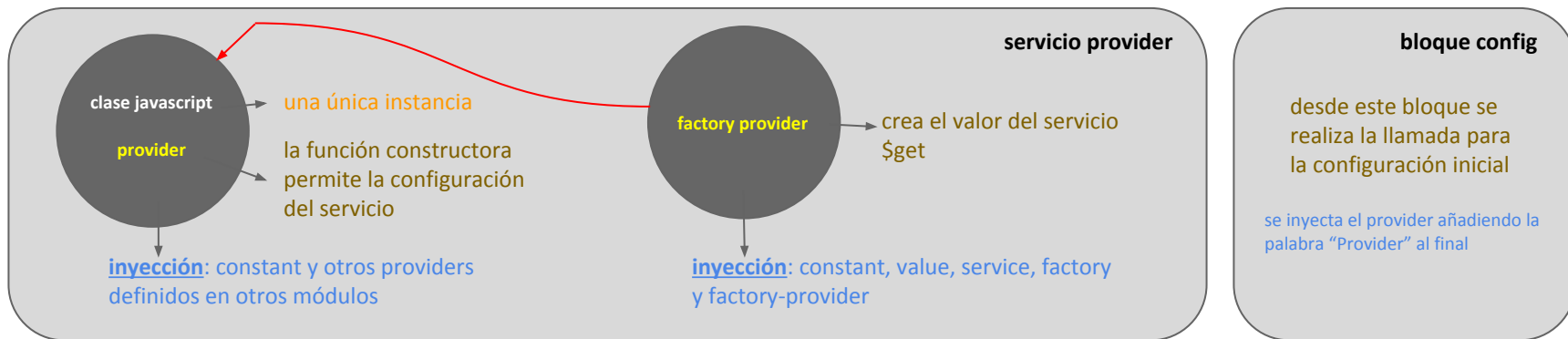
        name: '',
        postal_code: '',
        address: function() {
            // code
        }
    }

    return data;
});
```

SERVICIOS (X)

PROVIDER (I)

Parecido al 'factory', pero permitiendo configuración inicial antes de crear el valor del servicio



SERVICIOS (XI)

PROVIDER (II)

```
app.provider('prueba', function() {  
  var _type = '';  
  constructor → this.setType = function(value) {_type = value;}  
  factory-provider → this.$get = function() {  
    var op;  
  
    if(_type === 'suma') { op = SUMA; }  
    else if(_type === 'resta') { op = RESTA; }  
  
    var op_result = function(value1, value 2) {  
      return op(value1, value2);  
    }  
  
    return op_result;  
  }  
});
```

```
app.constant('operation', 'suma');  
app.config([inyección de una constante → 'pruebaProvider', 'operation', function(pruebaProvider, operation) {  
  pruebaProvider.setType(operation);  
}]);  
  
configuración del provider →
```

valor del servicio →

FILTROS



FILTROS (I)

Es una forma de modificar los datos que veremos en pantalla

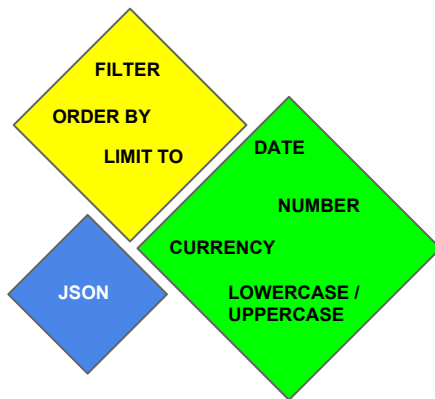
Se caracterizan por la rapidez y sencillez en su aplicación

Se utilizar el símbolo de la tubería (|)

Se pueden aplicar varios filtros distintos concatenando más tuberías y el nombre del siguiente filtro

NATIVOS

- ◆ Filtrar Arrays (listas)
- ◆ Formatear datos (escalares)
- ◆ Pruebas



FILTROS (II)

FILTER

- Subconjunto de elementos de un array
- Array de cadenas, números, objetos....
- Se puede utilizar el símbolo de negación !

expression

- ☐ **String:** cadena simple 'txt'
- ☐ **Propiedad/es** de un objeto
 - se pueden combinar varias propiedades tipo AND: {prop1: txt, prop2: txt}
 - se puede utilizar el símbolo \$ para recuperar cualquier campo, y combinarlo con búsqueda por cadena
- ☐ **Función:** definimos nuestra propia expresión, con argumentos value, index, array
 - se pueden hacer combinaciones tipo AND y OR

HTML `<div>{{array | filter: expression : comparator }}</div>`

JS `$filter('filter')(array, expression : comparator)`

```
<div>{{(nom:'Juan', nom:'Paco' ] | filter: {nom: 'Paco': true }}</div>
```

```
[nom:'Paco' ]
```

comparator: permite evaluar de forma estricta la comparación entre valor actual y el evaluado

- ☐ **true** (el valor debe ser igual), **false** (es como un like), **function** (valor de la propiedad, valor a buscar)

FILTROS (III)

◆ ORDER BY

- Permite ordenar una array determinado

expression: especifica el campo por el que hacer la ordenación

- ☐ **String, array** de cadenas, resultado de una **función**
- ☐ Utilizaremos el signo - si queremos que sea descendente
- ☐ No tiene en cuenta los acentos (orden ASCII)

reverse: conseguiremos que se invierta el orden establecido

HTML `<div>{{array | orderBy: expression : reverse }}</div>`

JS `$filter('orderBy')(array, expression : reverse)`

```
var players = [{name: 'Finidi', name: 'Batista', name: 'Guti'}]
```

```
<div>{{players | orderBy: 'name' : reverse }}</div>
```

```
[[{name: 'Guti', name: 'Finidi', name: 'Batista'}]]
```

FILTROS (IV)

◆ LIMIT TO

- Permite crear un nuevo array con el número de elementos especificados
- También se puede trabajar con números o cadenas

limit: longitud del resultado a devolver

- ☐ Filtrará desde el principio si el valor es positivo
- ☐ Filtrará desde el final si el valor es negativo
- ☐ Devolverá el total si sobrepasa la longitud, o con un valor sin definir

begin: indica el índice desde donde se empezará a filtrar

- ☐ Desde el principio si es positivo
- ☐ Desde el final si es negativo
- ☐ Valor por defecto: 0

HTML `<div>{{array | limitTo: limit : begin }}</div>`

JS `$filter('limitTo')(array, limit : begin)`

```
var numbers = [1, 2, 3, 4, 5, 6]
```

```
<div>{{numbers | limitTo: 4 : -1 }}</div>
```

```
[2, 3, 4, 5]
```


FILTROS (V)

◆ NUMBER

- Permite limitar el número de decimales
- Devolución en formato cadena, con tres decimales por defecto
- Ajusta la puntuación según el sistema

fractionSize: número entero que indica el número de decimales

HTML `<div>{{value | number: fractionSize }}</div>`

JS `$filter('number')(value, fractionSize)`

```
var money = 12,98754
```

```
<div>{{money | number: 4 }}</div>
```

```
'12,9875'
```

FILTROS (VI)

◆ DATE

- Se formatea una fecha según las necesidades de visualización

format: para indicar el formato de salida de la fecha

- ☐ Año: 'yyyy', 'yy'
- ☐ Mes: 'MMMM', 'MMM', 'MM', 'M'
- ☐ Día: 'dd', 'd'
- ☐ Formatos predefinidos: 'short', 'longDate'

timezone: permite especificar la zona horaria

- ☐ Si no se indica, se utiliza la actual por defecto

HTML `<div>{{value | date: format : timezone }}</div>`

JS `$filter('date')(value, format : timezone)`

```
var date_today = 1288323623006
```

```
<div>{{date_today | date: 'dd/MM/yyyy' }}</div>
```

```
'29/10/2010'
```

FILTROS (VII)

◆ CURRENCY

- Se formatea un número para devolver símbolo de la moneda y número de decimales
- Se adapta al idioma por defecto

symbol: permite especificar el tipo de moneda a mostrar

fractionSize: ajusta el número de decimales

HTML `<div>{{value | currency: symbol : fractionSize }}</div>`

JS `$filter('currency')(value, symbol : fractionSize)`

```
var number = 12,45674
```

```
<div>{{number | currency: $ : 2 }}</div>
```

```
'12,45$'
```

FILTROS (VIII)

◆ LOWERCASE / UPPERCASE

- Nos permiten transformar cadenas a minúsculas o mayúsculas

HTML `<div>{{value | lowercase }}</div>`

JS `$filter('lowercase')(value)`

```
var nameUpp = 'JAMES'
```

```
<div>{{nameUpp | lowercase }}</div>
```

```
'james'
```

HTML `<div>{{value | uppercase }}</div>`

JS `$filter('uppercase')(value)`

```
var nameLow = 'james'
```

```
<div>{{nameLow | uppercase }}</div>
```

```
'JAMES'
```

FILTROS (IX)

◆ JSON

- Transforma un objeto a formato JSON
- Se utiliza para realizar pruebas

spacing: permite especificar las tabulaciones del código a mostrar

HTML `<div>{{obj | json: spacing }}</div>`

JS `$filter('json')(obj, spacing)`

```
var obj = {'name' : 'value'}
```

```
<div>{{obj | json: 3 }}</div>
```

```
{  
  'name' : 'value'  
}
```

FILTROS (X)

PERSONALIZADOS

A través de una función de factoría

Devolverá nuestra función al método filter de un determinado módulo

Se comprobará el tipo de dato de entrada para realizar la operación correspondiente

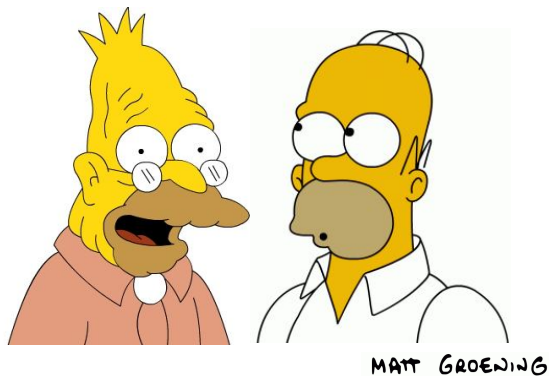
HTML Code

```
<div>{{ obj | nameFilter : param1 : param2 : param3 }}</div>
```

JavaScript Code

```
angular.module('myFiltersApp', [])  
  .filter('nameFilter', function() {  
    return function (input, param1, param2, param3) { // code }  
  })  
var obj = {'name' : 'value'}
```

TRANSCLUSIÓN Y HERENCIA DE DIRECTIVAS



TRANSCLUSIÓN (I)

DEFINICIÓN

Propiedad exclusiva de las directivas personalizadas

Nos permite insertar contenido dentro de la plantilla de una directiva

Envolveremos el código insertado dentro del elemento (básica) o el elemento completo (avanzada)

TRANSLUCIÓN (II)

CONFIGURACIÓN Y MÉTODOS

Nueva propiedad > **transclude: true**

Dos posibles métodos para poder usar la propiedad

ng-transclude

HTML Code

```
<div foo>Contenido</div>
```

JavaScript Code

```
.directive('foo', function() {  
  return {  
    transclude: true,  
    template: "<div>plantilla</div>  
      <div ng-transclude></div>"  
  }  
})
```

lo usaremos en la parte de la plantilla
donde queramos incluir el contenido del
HTML

transclude (fn)

JavaScript Code

```
.directive('foo', function() {  
  return {  
    transclude: true,  
    template: "<div>plantilla</div>",  
    link: function (scope, element, attrs, transclude) {  
      transclude(function (clone) {  
        element.append(clone);  
        element.find('...').prepend(clone);  
      });  
    }  
  }  
})
```

usaremos la función que vendrá como
propiedad dentro de link, añadiendo el
código en el DOM

TRANSLUCSIÓN (III)

MULTI-TRANSLUCSIÓN

Podemos pasar diferentes bloques a la directiva

Cada bloque tiene un nombre distintivo, pudiendo colocarlos de forma libre

HTML Code

```
<ejemplo>
  <ejemplo-titulo>{{titulo}}</ejemplo-titulo>
  <ejemplo-cont>contenido de la directiva</ejemplo-cont>
  <ejemplo-reglas>reglas a mostrar</ejemplo-reglas>
</ejemplo>
```

JavaScript Code

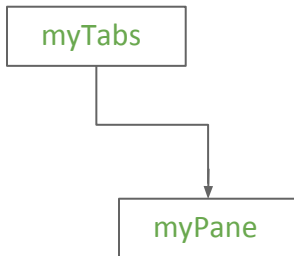
```
.directive('ejemplo', function(){
  return {
    restrict: 'A',
    transclude: {
      'titulo': '?ejemploTitulo',
      'body': 'ejemploCont',
      'footer': 'ejemploReglas'
    },
    template: '<div ng-transclude="body"></div>'
  }
})
```

HERENCIA DE DIRECTIVAS (I)

DEFINICIÓN Y EJEMPLO

Vamos a recibir instancias de otras directivas, pudiendo utilizar métodos y variables

String o array de String



JavaScript Code

```
.directive('myPane', function() {  
  return {  
    require: '^myTabs',  
    restrict: 'E',  
    link: function (scope, element, attrs, tabsCtrl) {  
      tabsCtrl.addPane(scope);  
    },  
    templateUrl: 'my-pane.html'  
  }  
})
```

```
.directive('myTabs', function() {  
  return {  
    restrict: 'E',  
    controller: ['$scope', function($scope) {  
      var panes = $scope.panes = [];  
      $scope.select = function (pane) {  
        // code  
      };  
      this.addPane = function (pane) {  
        // code  
      }  
    }],  
    templateUrl: 'my-tabs.html'  
  }  
})
```

HERENCIA DE DIRECTIVAS (II)

PROPIEDADES

Para heredar una directiva se dispone de las siguientes opciones en la definición del require:



optativa en el mismo elemento

require: '?myTabs',



optativa en algún elemento padre

require: '^myTabs',



obligatoria en algún elemento padre

require: '^myTabs',

obligatoria en el mismo elemento

require: 'myTabs',

ANGULAR 2



MATT GROENING

ANGULAR 2 (I)

DEFINICIÓN Y CARACTERÍSTICAS PRINCIPALES

Plataforma para la creación de aplicaciones web y móviles

Soporta TypeScript, Javascript y Dart



ANGULAR 2 (II)

COMPONENT-BASED UI

Desaparecen los conceptos de controlador y directiva personalizada

Vamos a utilizar componentes (similar a como se utiliza en ReactJs)

Angular 2

```
var AppComponent = ng.Component({  
  selector: 'my-app',  
  template: '<h1>Hola Mundo</h1>'  
})  
  
.Class ({  
  constructor: function() {  
    .....  
  }  
});
```

Angular 1.x

```
angular.module('ejemplo')  
  .controller('ejemploCtrl', function() {  
    .....  
  });
```



ANGULAR 2 (III)

TYPESCRIPT

El archivo `app.js` lo crearemos con `typeScript > app.ts`

Angular 2 se ha creado mediante este lenguaje de código abierto

Permite transformar su código en Javascript (compatible con ECMAScript 5 y 6)

```
import {Component} from 'angular2/core';

@Component({
  selector: 'prueba'
})
@view({
  templateUrl: 'app/views/prueba.html'
})
class Prueba {
  constructor() {}
  funcionEjemplo() {}
}
```

- Variables y funciones tipadas (tipos de datos)
- Clases (atributos, métodos, constructores) y objetos
- Herencia



Imagen: <https://flipboard.com/topic/typescript>

Imagen: <http://blog.tercerplaneta.com/2016/02/un-recorrido-por-typescript.html>

ANGULAR 2 (IV)

CAMBIOS EN LA SINTAXIS

Eventos

Angular 2

```
<button (click)="lanzarFuncion(item)" type="submit">
```

Angular 1.x

```
<button ng-click="lanzarFuncion(item)" type="submit">
```

Directivas nativas

Angular 2

```
<li *ngFor="#vehicle of vehicles"></li>
```

```
<li *ngIf="#vehicles.length"></li>
```

Angular 1.x

```
<li ng-repeat="vehicle in vehicles"></li>
```

```
<li ng-if="vehicles.length"></li>
```



ANGULAR 2 (V)

UN ADIÓS INESPERADO

Desaparece el \$scope

Los elementos se definen en this (dentro de la clase correspondiente)

Angular 2

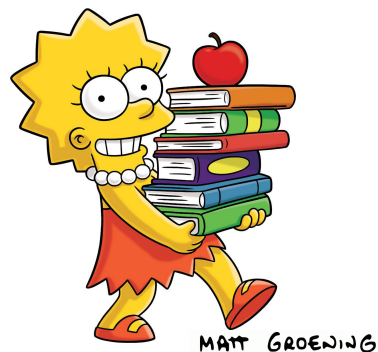
```
.class({  
  constructor: function() {  
    this.variable = "paco"  
  }  
})
```

Angular 1.x

```
.controller('EjemploCtrl', function($scope) {  
  $scope.variable = "paco"  
})
```



BIBLIOGRAFÍA



BIBLIOGRAFÍA (I)

- Página principal de Angular: <https://angularjs.org/>
- API de Angular: <https://docs.angularjs.org/api/>
- Filtros: http://cursoangularjs.es/doku.php?id=unidades:05_filtros:00_start
- Servicios:
http://cursoangularjs.es/doku.php?id=unidades:03_servicios:00_start
<http://www.slideshare.net/DreamFactorySoftware/angularjs-and-rest-made-simple>
- Transclusión:
<https://www.accelebrate.com/blog/angularjs-transclusion-part-1/>
<https://www.accelebrate.com/blog/transclusion-angularjs-part-2-2/>

BIBLIOGRAFÍA (II)

- Herencia: <http://prendedor.es/creacion-directivas-personalizadas-angularjs/>
- Angular 2:
 - <http://www.sebastian-gomez.com/desarrollo-web/angular-1-vs-angular-2-parte-2/>
 - <http://www.flaviocorpa.com/encuentra-las-7-diferencias-angular-1-vs-angular-2/>
- Blog de Beeva:
 - <https://www.beeva.com/beeva-view/tecnologia/buenas-practicas-de-angular-en-nimble/>
 - <https://www.beeva.com/beeva-view/tecnologia/angular-en-nimble-trabajando-con-clases-objetos-e-instancias/>
 - <https://www.beeva.com/beeva-view/metodologiasagiles/angular-en-nimble-directivas-a-varios-niveles/>
 - <https://www.beeva.com/beeva-view/ahora-en-beeva/angular-nimble-transclusion/>



hablemos@beeva.com

www.beeva.com

Fernando Castro García

Software Analyst in BEEVA

fernando.castro@beeva.com



hablemos@beeva.com

www.beeva.com