



Elementos inyectables

Los servicios son sólo uno de los 5 tipos de elementos que pueden ser inyectados como dependencias

- Constantes (*constant*)
- Valores (*value*)
- Servicios (*service*)
- Factorías (*factory*)
- Proveedores (*provider*)



Objetos JavaScript



Servicios, Factorías y Providers



Objeto JavaScript (service) que nos permite obtener información.

Son funcionalidades de la aplicación **no relacionadas con el interfaz gráfico**

No interaccionan con la página (DOM), sólo con otros servicios o con un servidor de datos que pueda estar en otro Host (e.g. AJAX).

Las instancias de los servicios, para incorporarlos a la aplicación, se obtienen mediante **inyección de dependencias**.

Siempre son **Singletons**

Esto facilita el **desarrollo guiado por pruebas**
(TDD, *Test-driven development*)

Angular nos permite **decorar** los servicios, es decir interceptar su creación para añadir métodos nuevos que modifiquen su funcionamiento.



Inyección de dependencias



Angular 1

```
angular.module('appModule')
.controller('MainCtrl', ['$scope', '<Service>',
  function ($scope, <Service>) {
    ...
  }
]);
```

Angular 1.5

```
Class MainCtrl {
  constructor ($scope, <Service>) {
    'ngInject';
    this.$scope = $scope
    this.<Service> = <Service>
  }
}

angular.module('appModule')
.controller('MainCtrl', MainCtrl)
```





Servicio \$log

servicio muy simple que permite llamar a `console.log` pero abstrayendo la llamada por si no existe el objeto `console`

\$log dispone de varios métodos, incluyendo el método `debug()` que permite pasarle un mensaje para que se muestre por la consola del navegador.

```
controller('AppController',
 ['$scope', '$log', function($scope, $log){
 ...
 $log.debug("Acabamos de crear el $scope");...
```

Inyección del servicio:

- Indicamos literalmente su nombre (evita problemas de minimificación))
- lo "inyectamos en el controller para que este cree la instancia del servicio
- lo utilizamos en el controller .



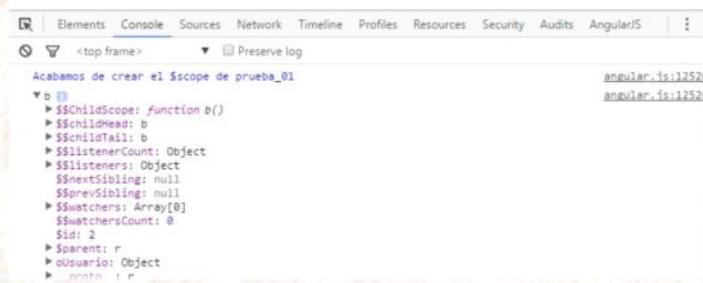
Mensajes en consola: \$log

Empleamos el servicio \$log para mostrar por consola un mensaje y el objeto \$scope

Servicio \$log

Comprueba en la consola de JS el resultado de la aplicación
Podrás conocer al famoso scope personalmente

Alejandro L. Cerezo
CLE Formación
Madrid - 2015



The screenshot shows a browser's developer tools console with the 'Console' tab selected. The log output is as follows:

```
Acabamos de crear el $scope de prueba_01
▶ b
  ► $$childScope: function b()
  ► $$childHead:
  ► $$childTail: b
  ► $$listenerCount: Object
  ► $$listeners: Object
  ► $$nextSibling: null
  ► $$prevSibling: null
  ► $$watchers: Array[0]
  ► $$watchersCount: 0
  ► $id: 2
  ► $parent: r
  ► $usuario: Object
  ► ...: r
```

On the right side of the log, there are two small lines of text: "angular.js:1252l" and "angular.js:1252l".



Servicio \$locale



Servicio responsable de la **Internacionalización** (i18n) y la **Localización** (l10n).

Da soporte a los filtros

- fecha (| date)
- numero (| number)
- moneda (| currency)

Se descarga y/o referencia el script con los elementos de configuración dependientes de la localización

```
<script src="../lib/angular-locale_es-es.js"></script>
```

<https://github.com/angular/angular.js/tree/master/src/ngLocale>

Una vez injectado, se establece el valor del atributo id

\$locale.id="es-es"





\$locale.id = "es-es"

```
<!document html>
<html ng-app>
  <head>
    <meta charset="utf-8">
    <title>locale test</title>
    <script src="../../lib/angular.js"></script>
    <script src="../../lib/i18n/angular-locale_es.js"></script>
    <script>
      function AppCtl($scope){$scope.input = 234234443432;}
    </script>
  </head>

  <body ng-controller="AppCtl">
    <input type="text" ng-model="input" value="234234443432"><br>
    date: {{input | date:"medium"}}<br>
    date: {{input | date:"longDate"}}<br>
    number: {{input | number}}<br>
    currency: {{input | currency }}
  </body>
</html>
```



Módulos de Angular



Download AngularJS

<https://code.angularjs.org/1.4.8/>

Index of /1.4.8/

File	Last Modified	Size
angular.js	20-Nov-2015 08:13	9859372
angular-animate.js	20-Nov-2015 08:13	143532
angular-animate.min.js	20-Nov-2015 08:13	25600
angular-animate.min.js.map	20-Nov-2015 08:13	69733
angular-aria.js	20-Nov-2015 08:13	14950
angular-aria.min.js	20-Nov-2015 08:13	3774
angular-aria.min.js.map	20-Nov-2015 08:13	8437
angular-cookies.js	20-Nov-2015 08:13	9729
angular-cookies.min.js	20-Nov-2015 08:13	14443
angular-cookies.min.js.map	20-Nov-2015 08:13	3399
angular-csp.css	20-Nov-2015 08:13	34
angular-loader.js	20-Nov-2015 08:13	10801
angular-loader.min.js	20-Nov-2015 08:13	1685
angular-loader.min.js.map	20-Nov-2015 08:13	3775
angular-message-format.js	20-Nov-2015 08:13	37844
angular-message-format.min.js	20-Nov-2015 08:13	10300
angular-message-format.min.js.map	20-Nov-2015 08:13	28138
angular-messages.js	20-Nov-2015 08:13	25732
angular-messages.min.js	20-Nov-2015 08:13	2734
angular-messages.min.js.map	20-Nov-2015 08:13	7419
angular-mocks.js	20-Nov-2015 08:13	82847
angular-resource.js	20-Nov-2015 08:13	2746
angular-resource.min.js	20-Nov-2015 08:13	3793
angular-resource.min.js.map	20-Nov-2015 08:13	9671
angular-route.js	20-Nov-2015 08:13	35877
angular-route.min.js	20-Nov-2015 08:13	4370
angular-route.min.js.map	20-Nov-2015 08:13	11132
angular-sanitize.js	20-Nov-2015 08:13	24546
angular-sanitize.min.js	20-Nov-2015 08:13	6021
angular-sanitize.min.js.map	20-Nov-2015 08:13	10541
angular-scenario.js	20-Nov-2015 08:13	138736
angular-touch.js	20-Nov-2015 08:13	23179
angular-touch.min.js	20-Nov-2015 08:13	3588
angular-touch.min.js.map	20-Nov-2015 08:13	9958
angular.js	20-Nov-2015 08:13	1870726
angular.min.js	20-Nov-2015 08:13	148930
angular.min.js.map	20-Nov-2015 08:13	399648
error.json	20-Nov-2015 08:13	9100
version.json	20-Nov-2015 08:13	361
version.json	20-Nov-2015 08:13	5

Branch: 1.5.x (beta) 1.4.x (stable) 1.2.x (legacy)

Build: Minified Uncompressed Zip

CDN: https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js

Bower: bower install angular#1.4.8

NPM: npm install angular@1.4.8

Extras: Browse additional modules

Previous Versions Download



Enrutamiento



En MVC se conoce como enrutamiento (*routing*) el proceso de asociar diversas rutas con sus correspondientes vista y controlador

Aplicación de una sola página (SPA)

es capaz de representar URL distintas, simulando lo que sería una navegación a través de la aplicación, pero sin salirnos nunca de la página inicial.





Rutas internas

Las rutas profundas (*deep links*) permiten crear enlaces que nos lleven a partes internas de la aplicación

<http://example.com/index.php>
<http://example.com/index.php#/seccion>
http://example.com/index.php#/pagina_interna

- permiten entradas en la aplicación alternativas a la pantalla inicial.
- facilitan el uso natural del sistema de favoritos (o marcadores) del navegador, así como el historial.
- son responsables de que AngularJS cambie la vista de acuerdo con la ruta empleada en cada momento
- permiten mantener vistas en archivos independientes, lo que reduce su complejidad y administrar los controladores que van a facilitar el procesamiento dentro de ellas.





módulo ngRoute

El módulo **ngRoute** es un paquete de utilidades para configurar el enrutado y asociar cada ruta a una vista y un controlador, que no está incluido en la distribución de base de Angular

Para usarlo es necesario

- instalarlo
- injectarlo como dependencia en el módulo principal de la aplicación
- configurar las rutas que queremos crear en nuestra aplicación de una manera declarativa.





ngRoute: instalación

Instalación: en la parte inferior de la ventana de descarga de Angular accedemos a los módulos adicionales, que pueden descargarse o utilizarse directamente a través de una CDN

A screenshot of the 'Download AngularJS' page. It shows options for Branch (1.2.x (legacy) selected, 1.3.x (latest)), Build (Minified selected, Uncompressed, Zip), CDN (https://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular.min.js selected), Bower (bower install angular#1.2.25), and Extras (Browse additional modules). A yellow box highlights the 'Extras' link. At the bottom are links for 'Previous Versions' and a large blue 'Download' button.

En cualquier caso, se incluirá en el HTML la referencia al script correspondiente, de forma local o a través de un CDN

```
<script src="angular-route.js"></script>
```





ngRoute: uso

Inyección de dependencias: utilizamos el *array* de dependencias, hasta ahora vacío en los anteriores ejemplos

```
angular.module("app", ["ngRoute"])
```

Este módulo proporciona un servicio, `$routeProvider`, que sólo puede ser inyectado en el método `config()` del módulo principal de la aplicación, donde se realizará la configuración dentro de una función en la que se ha inyectado `$routeProvider`.

```
angular.module("appModule", ["ngRoute"])
.config(function($routeProvider){
    configuración...
})
```





Configuración de las rutas

Configurar las rutas mediante los métodos de \$routeProvider

- **when()**, que permite indicar qué se debe hacer en cada ruta que se desee configurar,
- **otherwise()** que permite definir un comportamiento para cuando se intente acceder a cualquier otra ruta no declarada

Cualquiera de ellos incluye como parámetro un objeto con 2 valores

templateUrl, con la URL de la vista, plantilla o parcial, y
controller el nombre del controlador que se hará cargo de la ruta

El método *when* asocia este objeto con un primer parámetro, correspondiente a la ruta que se está definiendo



Ejemplo rutas

```
angular.module("appModule", ["ngRoute"])
.config(function($routeProvider){

$routeProvider
  .when('/', { // route for the home page
    templateUrl: 'view/view_inicio.html',
    controller: 'AppController'
  })
  .when('/about', { // route for the about page = Acerca de
    templateUrl: 'view/view_about.html',
    controller: 'AboutController'
  })
  .otherwise({ // when all else fails
    templateUrl: 'view/view_routeNotFound.html',
    controller: 'NotFoundController'
  });
});
```

Llamadas a métodos `when()` **encadenadas** sobre el `$routeProvider`, para definir cada una de las rutas





Menú y directiva ngView

Diversas rutas definidas mediante \$routeProvider significa que cada una de ellas a quedado asociada con un fichero HTML conocido como vista, plantilla o parcial, que debe incorporarse en un punto determinado del HTML principal

El papel de la directiva **ngView** es indicar el punto en el que se realizará automáticamente esta incorporación

```
<article>
    <p ng-view></p>
</article>
```

El HTML incluirá también un **menú**, con enlaces a cada una de las rutas que hemos definido

```
<li><a href="#">Inicio</a></li>
<li><a href="#/about">Acerca de</a></li>...
```



Menú: Rutas

Cle Formación

[Inicio](#) [Acerca de](#) [Servicios](#) [Clientes](#) [Contactos](#)

Página de Inicio

Alejandro L. Cerezo.
CLE Formación
Madrid - 2015





Servicio \$location

\$location permite mantener y trasladar información de la ruta actual del navegador.

Implementa una interfaz para el acceso a la propiedad nativa de JS (BOM) **window.location**, que permite para acceder a elementos de la ruta actual, conocer su estado y modificarlo

existe un enlace entre \$location y window.location en las dos direcciones. Todo cambio querealicemos en \$location también tendrá una respuesta en lo que se está mostrando en la barra de direcciones del navegador. Así como todo cambio de URL en el navegador tiene un efecto en el service \$location.





Controller para el menú

Añadimos un controlador específico para la barra de menú

en el situamos una función booleana que comprueba si el valor pasado por parámetro coincide con la url real, contenida en \$location

```
$scope.estoy = function(ruta){  
    return $location.path() == ruta;  
}
```

de esta forma detectamos en que vista estamos en cada momento y es posible resaltar la correspondiente opción del menú

```
<li ng-class="{marcado: estoy('<ruta>')}"
```



Menú: Rutas y location

Cle Formación

[Inicio](#) [Acerca de](#) [Servicios](#) [Clientes](#) [Contactos](#)

Página de Inicio

Alejandro L. Cerezo.
CLE Formación
Madrid - 2015





Rutas dinámicas

Al definir una ruta en la propiedad *when* de *\$routeProvider* es posible añadir valores variables

```
.when('/view2/:var1/:var2',{...})
```

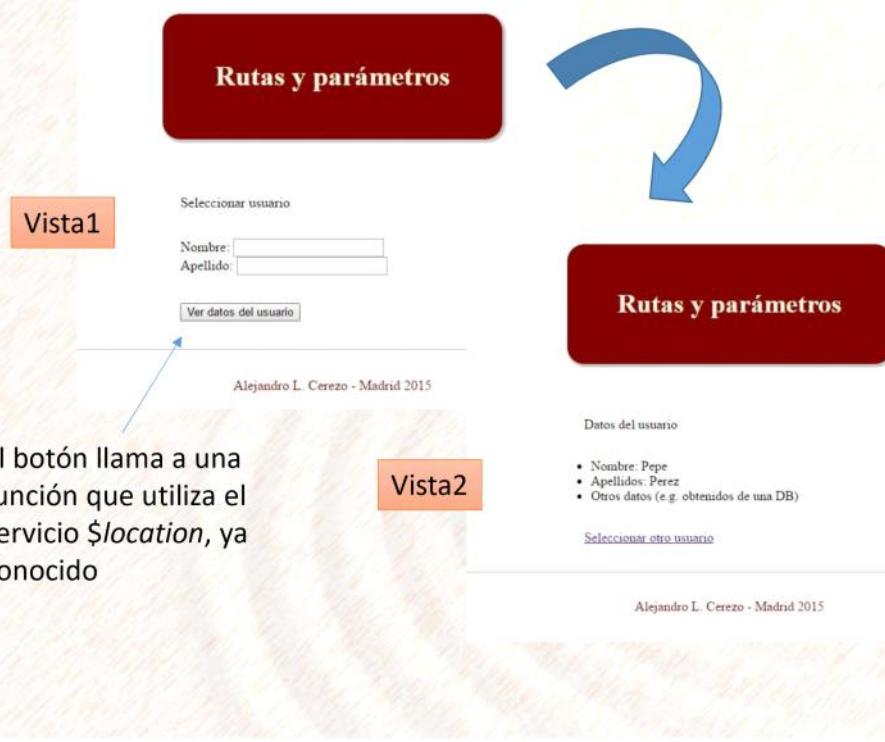
Estos valores son recibidos como parámetros por el controlador de la vista correspondiente, gracias al servicio *\$routeParams*.

```
.controller('View2Controller', ['$scope', '$routeParams',
function($scope, $routeParams){
    $scope.firstname = $routeParams.firstname;
    $scope.lastname = $routeParams.lastname;

}])
```



Usuario: Rutas con parámetros





La propiedad resolve

En la propiedad *when* de *\$routeProvider* es posible añadir al objeto configuración la propiedad **resolve**

```
.when('/view2/:var1/:var2',{
    templateUrl ... controller ...
    resolve:
})
```

Así se define un objeto cuyas propiedades serán dependencias adicionales para el controller : servicios existente o funciones, generalmente promesas e.g. el resultado de una solicitud AJAX.





Angular UI Router

- Alternativa más potente para el enrutamiento
- Admite múltiples niveles

```
bower install ui-router --save
```

```
<script type="text/javascript">  
  src="../lib/angular-ui-router/release/angular-ui-router.min.js">  
</script>
```

Inyectamos la correspondiente dependencia en al módulo de AngularJS

```
angular.module("appModule", ["ui.router"])
```



Uso de UI Router



En el punto de inserción de las vistas (parciales)
utilizamos ui-view en lugar de ng-view

<p ui-view></p>

Reconstruimos la definición de las rutas que pasan a ser "estados"

```
.config(function($stateProvider, $urlRouterProvider){  
    $stateProvider.state('home', {  
        url: '/',  
        templateUrl: 'view/view_inicio.html',  
        controller: 'AppController'  
    })
```

Reconstruimos la llamada a las distintas rutas (ahora estados)
con la directiva ui-sref

<a ui-sref="home">Inicio



Menú: Rutas con UI Router

Cle Formación

[Inicio](#) [Acerca de](#) [Servicios](#) [Clientes](#) [Contactos](#)

Página de Inicio

Alejandro L. Cerezo
CLE Formación
Madrid - 2015

Rehacemos el ejemplo
utilizando *UI Router*



Promesas y Datos REST

jueves, 10 de agosto de 2017 20:34

Una promesa representa el resultado eventual de una operación.
Se utiliza para especificar que se hará cuando esa eventual operación de un resultado de éxito o fracaso.

Promesas

JS

Un objeto promesa representa un valor que todavía no está disponible pero que lo estará en algún momento en el futuro

Permiten escribir código asíncrono de forma más similar a como se escribe el código síncrono:

- La función asíncrona retorna inmediatamente y → ese retorno se trata como un proxy cuyo valor se obtendrá en el futuro

El API de las promesas en Angular corresponde al servicio **\$q**

la biblioteca Q desarrollada por **Kris Kowal**

<https://github.com/kriskowal/q>



Promesas: \$q

JS

```
function getPromise()
```

```
    var deferred=$q.defer();
```

crea una promesa

```
    deferred.resolve()  
    deferred.reject()
```

resuelve la promesa en un sentido
u otro al cabo del tiempo

```
    return deferred.promise
```

devuelve la promesa

```
var promise = getPromise();
```

```
promise.then(successCallback,failureCallback,notifyCallback);
```

```
promise.catch(errorCallback)
```

```
promise.finally(callback)
```

promise.catch(errorCallback)

promise.finally(callback)



Promesas: ES6

JS

Implementación: new Promise

el objeto promesa recibe como parámetros dos funciones:

- La función "resolve": se ejecuta cuando queremos finalizar la promesa con éxito.
- La función "reject": se ejecuta cuando queremos finalizar una promesa informando de un caso de fracaso.

```
function hacerAlgoPromesa() {  
    return new Promise( function(resolve, reject) {  
        console.log('hacer algo que ocupa un tiempo...');  
        setTimeout(resolve, 1000);  
    })  
}
```



Promesas: ES6

JS

Utilización

a la función que retorna el objeto promesa se le encadenan dos:

- .then : la función que se ejecutará cuando la promesa haya finalizado con éxito.
- .catch : la función que se ejecutara cuando la promesa haya finalizado informando de un caso de fracaso.

```
hacerAlgoPromesa()
  .then( function() { console.log('la promesa terminó.');
})
  .catch( function() { console.log('la promesa fracasó.'));
```



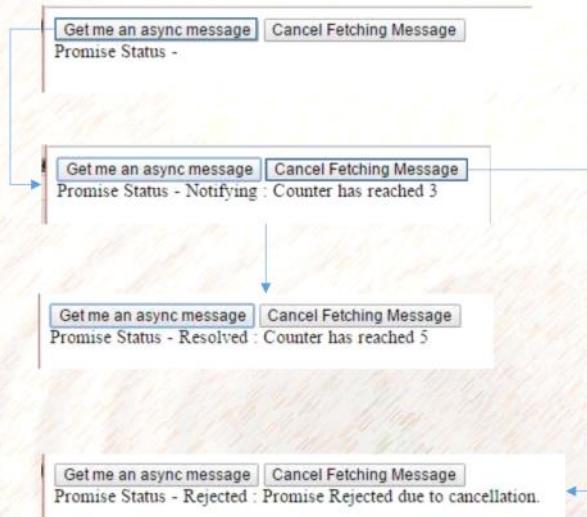
Ejemplo de promesas en ES6

```
1  function msgAfterTimeout (msg, who, timeout) {
2      return new Promise((resolve, reject) => {
3          setTimeout(
4              () => resolve(` ${msg} Hello ${who}! `),
5              timeout)
6      })
7  }
8
9  msgAfterTimeout("", "Foo", 100)
10 .then((msg) =>
11     msgAfterTimeout(msg, "Bar", 200))
12 .then((msg) => {
13     console.log(`done after 300ms:${msg}`)
14 })
```



https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

Ejemplo de promesas





Servicio \$http

servicio incluido como funcionalidad en el núcleo de AngularJS
permite comunicación asincrónica mediante el protocolo HTTP (**Ajax**)

- vía el objeto **XMLHttpRequest**
(nativo de JavaScript)
- vía JSONP.

Incluye diversos métodos
alternativos o atajos (*shortcuts*)

\$http.get()
\$http.post()
\$http.put()
\$http.delete()
\$http.jsonp()
\$http.head()
\$http.patch()

En cualquiera de los casos, devuelve una promesa.
Es por tanto muy similar al soporte de AJAX en JQuery





Uso del servicio \$http

Inyección del servicio en el controller

```
controller('AppController',
 ['$scope', '$http', function($scope, $http){
```

Con frecuencia el proceso es algo más complejo, al estar mediado por una factoría que define un servicio propio del usuario.

Uso de los métodos que proporciona el servicio

```
$http([<url> ], {<objeto con los parámetros>});
$http.get([<url> ], {<objeto con los parámetros>});
```

Ejemplo

```
$http( { method: 'POST',
url: 'memberservices/register',
data: theData} )
```



Gestión de la respuesta

\$http en cualquiera de sus variantes da lugar a un objeto "promesa" por lo que disponemos de diversos métodos para gestionarla

Fuera de uso
(deprecated)

Success (function (data, status, headers, config))
Error (function (data, status, headers, config))
Finally (function (data, status, headers, config))

En su formato actual:

```
then(<funcion_exito>,<función_error>)
```

```
$http.get("/api/my/name")
.then(
/* success */
function(response) {
  console.log("Your name is: " + response.data);
},
/* failure */
function(error) {
  console.log("The request failed: " + error);
});
```

La primera función *callback* se ejecutara en caso de la promesa resuelta de forma correcta. El parámetro *response* incluye el objeto *data*, un objeto JavaScript correspondiente a los datos JSON que ha recibido, ya procesados por *JSON.parse()*

La segunda función *callback* se ejecutará cuando esta se resuelva con un resultado de error, teniendo como parámetro el mensaje de error correspondiente

Países: consumo de un servicio

Paises del Mundo

Selecciona un continente:



Alejandro L. Cerezo
CLE Formación
Madrid - 2015



Promesas: \$http



Refactorización del código que utiliza XMLHttpRequest(AJAX)

```
angular.module('mainApp')
.controller('WeatherController', function($scope, weatherService) {
    $scope.getWeather = function() {
        weatherService.getWeather($scope.city, $scope.country)
        .then( function(data) { //success
            $scope.weatherDescription = data.weather[0].description;
        }, function() { //error
            $scope.weatherDescription = "Could not obtain data";
        });
    };
});

angular.module('mainApp').factory('weatherService', function($http) {
    return {
        getWeather: function(city, country) {
            var query = 'q=' + city + ',' + country;
            return $http.get('http://api.openweathermap.org/data/2.5/weather?' +
            query,{cache:true});
        }
    };
});
```



El servicio (*factory*) que hemos creado, una vez inyectado, proporciona el método `getWeather` que, como promesa que es puede resolverse en 2 sentidos

El inyectable de tipo factory `weatherService` devuelve (instancia) un objeto, en este caso con un solo método, `getWeather`, que devuelve una promesa, por lo que el mismo es una promesa



Otros Módulos de Angular

<https://code.angularjs.org/1.4.8/>

Index of /1.4.8/

angular.js	20-Nov-2015 08:13	-
angular-animate.js	20-Nov-2015 08:13	9836372
angular-animate.min.js	20-Nov-2015 08:13	141532
angular-animate.min.js.map	20-Nov-2015 08:13	254698
angular-aria.js	20-Nov-2015 08:13	69733
angular-aria.min.js	20-Nov-2015 08:13	14595
angular-aria.min.js.map	20-Nov-2015 08:13	3772
angular-cookies.js	20-Nov-2015 08:13	8437
angular-cookies.min.js	20-Nov-2015 08:13	9729
angular-cookies.min.js.map	20-Nov-2015 08:13	1444
angular-messages.js	20-Nov-2015 08:13	3196
angular-messages.min.js	20-Nov-2015 08:13	543
angular-messages.min.js.map	20-Nov-2015 08:13	16910
angular-loader.js	20-Nov-2015 08:13	1489
angular-loader.min.js	20-Nov-2015 08:13	3774
angular-loader.min.js.map	20-Nov-2015 08:13	37844
angular-message-format.js	20-Nov-2015 08:13	10986
angular-message-format.min.js	20-Nov-2015 08:13	28130
angular-messages.js	20-Nov-2015 08:13	25732
angular-messages.min.js	20-Nov-2015 08:13	2748
angular-messages.min.js.map	20-Nov-2015 08:13	3419
angular-mocks.js	20-Nov-2015 08:13	82847
angular-resource.js	20-Nov-2015 08:13	27407
angular-resource.min.js	20-Nov-2015 08:13	3773
angular-resource.min.js.map	20-Nov-2015 08:13	9874
angular-route.js	20-Nov-2015 08:13	35877
angular-route.min.js	20-Nov-2015 08:13	4399
angular-route.min.js.map	20-Nov-2015 08:13	11232
angular-sanitize.js	20-Nov-2015 08:13	24546
angular-sanitize.min.js	20-Nov-2015 08:13	6927
angular-sanitize.min.js.map	20-Nov-2015 08:13	18981
angular-scenario.js	20-Nov-2015 08:13	1387398
angular-touch.js	20-Nov-2015 08:13	23179
angular-touch.min.js	20-Nov-2015 08:13	3759
angular-touch.min.js.map	20-Nov-2015 08:13	9958
angular.js	20-Nov-2015 08:13	1070726
angular.min.js	20-Nov-2015 08:13	148199
angular.min.js.map	20-Nov-2015 08:13	395648
error.json	20-Nov-2015 08:13	9180
version.json	20-Nov-2015 08:13	301
version.txt	20-Nov-2015 08:13	5

- angular-animate
- angular-aria
- angular-cookies
- angular-loader
- angular-message / message-format
- **angular-resource**
- angular-route
- angular-sanitize
- angular-touch

angular-resource

API de alto nivel para las operaciones REST





Angular \$resource

API de alto nivel para las operaciones REST sobre un sistema que siga el siguiente estándar

URL	HTTP Verb	Request Body	Result
/api/entries	GET	empty	Returns all entries
/api/entries	POST	JSON String	Creates new entry
/api/entries/:id	GET	empty	Returns single entry
/api/entries/:id	PUT	JSON String	Updates existing entry
/api/entries/:id	DELETE	empty	Deletes existing entry

métodos de \$ resource

1. get()
2. query()
3. save()
4. remove()
5. delete()



Recordamos el concepto
ya conocido de
elementos inyectables

Elementos inyectables



Los servicios son sólo uno de los 5 tipos de elementos que
pueden ser inyectados como dependencias

- Constantes (*constant*)
- Valores (*value*)
- Servicios (*service*)
- Factorías (*factory*)
- Proveedores (*provider*)

↓
Objetos JavaScript

Servicios, Factorías y Providers



Objeto JavaScript (service) que nos permite obtener información.

Son funcionalidades de la aplicación **no relacionadas con el interfaz gráfico**

No interaccionan con la página (DOM), sólo con otros servicios o con
un servidor de datos que pueda estar en otro Host (e.g. AJAX).

Las instancias de los servicios, para incorporarlos a la aplicación, se
obtienen mediante **inyección de dependencias**.

Siempre son **Singletons**

Esto facilita el **desarrollo guiado por pruebas**
(TDD, *Test-driven development*)

Angular nos permite **decorar** los servicios, es decir interceptar su creación
para añadir métodos nuevos que modifiquen su funcionamiento.



Creación de un servicio



angular.module() proporciona la función **service()** que permite registrar un servicio

- Argumentos →
- el nombre del servicio
 - la **función constructora** que lo constituye.

En ella se incluyen las habituales variables o funciones propias de instancia

```
.service('pruebaService1',function(){
    this.saludar = function(){ // define un método de instancia
        alert('Hola!! Prueba de un servicio.');
    }
})
```



Servicios: Creación



Ejemplos de dos servicios muy sencillos





Creación de una factoría

angular.module() proporciona la función **factory()** que permite registrar una factoría

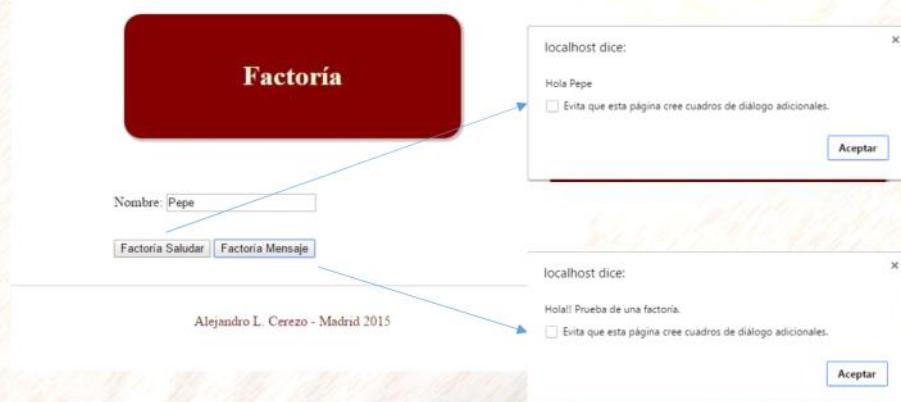
- Argumentos →
- el nombre de la factoría
 - la función que se ejecutara.

el retorno puede ser una función o un objeto, proporcionando así un conjunto de funciones

```
.factory('pruebaFactory',function(){
    return {
        mensaje : function(){
            alert('Hola!! Prueba de una factoría.');
        },
        saludar : function(name){
            alert('Hola ' + name);
        }
    }
})
```



Factorías: Creación



Ejemplo de una sencilla factoría, con dos funciones diferentes





Creación de un proveedor

angular.module() proporciona la función **provider()** que permite registrar un proveedor

- Argumentos →
- el nombre del proveedor
 - una función con un formato predefinido, que incluye this.\$get

Es como un factory pero permite que se configure antes de crear el valor del servicio.

2 partes:

- El *provider* que es una clase JavaScript de la que se crea un único objeto , el cual puede ser configurado en un bloque *config* antes de que se llame al *factory-provider*.
- El *factory-provider*, el cual crea el valor del servicio. Es prácticamente como la función *factory*.



Creación de un proveedor



```
.provider('saludar',function(){
    this....;
    this.$get = function () {
        return function(...){...}
    };
})
```

La función retornada
será la que se inyecte
al utilizar el *provider*

```
.config(function(saludarProvider) {
    saludarProvider.>funciones definidas para la configuración>;
})
```

```
.controller('AppController', ['$scope','saludar',
    function($scope, saludar){}
```



Proveedores: Creación

Proveedor

localhost dice:

Nombre: Pepe

Proveedor Saludar

Adios, Pepe

Evita que esta página cree cuadros de diálogo adicionales.

Aceptar

Alejandro L. Cerezo - Madrid 2015

Ejemplo de un proveedor, que primero se configura,
para cambiar el saludo, y luego se utiliza

