Injecció de dependències

En general, només hi ha tres maneres en què un objecte pot contenir les seves dependències:

1. Podem crear-lo internament als dependents.
2. Podem consultar-lo o referir-se a ell com una variable global.
3. Podem passar-ho on sigui necessari.

Amb la injecció de dependència, abordem la tercera (els altres dos presenten altres dificultats i reptes, com embrutar el desenvolupament global i fer que l'aïllament sigui gairebé impossible).

Dependència
La injecció és un patró de disseny que permet eliminar les dependències de codis durs, fent-ho així és possible eliminar o canviar-los en temps d'execució.
Aquesta capacitat de modificar dependències en temps d'execució ens permet crear entorns aïllats que són ideals per fer proves. Podem reemplaçar objectes reals en entorns de producció per a evitar els entorns de proves.
Funcionalment, el patró injectat dependrà dels recursos a la destinació, quan sigui necessari buscarà automàticament la dependència amb antelació i proporcionarà la destinació per a la dependència.
A mesura que escrivim components que depenguin d'altres objectes o biblioteques, descriurem les seves dependències.
En temps d'execució, un injector crearà instàncies de les dependències i les passarà al consumidor dependent.

```
// Great example from the Angular docs
function SomeClass(greeter) {
this.greeter = greeter;
}
SomeClass.prototype.greetName = function(name) {
this.greeter.greet(name)
}
```

En temps d'execució, SomeClass no es preocupa per la seva dependència de greeter, sempre que ho adquireixi. Dins del ordre per obtenir la instància greeter dins de SomeClass, el creador de SomeClass es responsable de passar les dependències de SomeClass quan es creant.

Angular utilitza $injector per gestionar cerques i la creació d'instàncies de dependències per aquest motiu. De fet, $injector és responsable del maneig de totes les instàncies dels nostres components en Angular, inclosos els nostres mòduls d'aplicacions, directives, controladors, etc. Quan algun dels nostres mòduls s'inicii en temps d'execució, l'injector és responsable de la inicialització de l'objecte i el pas de qualsevol de les seves dependències requerides. Per exemple, aquesta senzilla aplicació declara un únic mòdul i un únic controlador, així:

```
angular.module('myApp', [])
.factory('greeter', function() {
return {
greet: function(msg) { alert(msg); }
}
})
.controller('MyController',
function($scope, greeter) {
$scope.sayHello = function() {
greeter.greet("Hello!");
};
});
```

En temps d'execució, quan Angular crea una instància del nostre mòdul, es veu el greeter i simplement ho traspassa de forma natural:

```
<div ng-app="myApp">
<div ng-controller="MyController">
<button ng-click="sayHello()">Hello</button>
</div>
</div>
```

Darrere de les vistes, el procés Angular es veu així:

```
// Carreguem la App amb el injector
var injector = angular.injector(['ng', 'myApp']);
// Carreguem el service $controller amb el injector
var $controller = injector.get('$controller');
var scope = injector.get('$rootScope').$new();
// Carreguem el controller, ho passem a un scope
// que és com angular ho fa en temps d'execució
var MyController = $controller('MyController', {$scope: scope})
```

En cap altre lloc de l'exemple anterior, vam descriure com trobar el greeter; simplement funciona, com el injector s'encarrega de trobar-lo i carregar-lo per nosaltres. AngularJS utilitza una funció d'anotació per treure propietats de la matriu passada durant la creació d'instàncies. Podeu veure aquesta funció escrivint el següent a les eines del desenvolupador de Chrome:

```
> injector.annotate(function($q, greeter) {})
["$q", "greeter"]
```

A cada aplicació Angular, $injector ha estat treballant, tant si ho sabem com si no. Quan escrivim un controlador sense la notació del parèntesi [] o mitjançant una configuració explícita, el $injector interferirà les dependències en funció del nom dels arguments.

Veure ejercici2.html i ejercici2.js.

Tipus d'anotació

Angular assumeix que els noms dels paràmetres de funció són els noms de les dependències, si no especificat d'una altra manera. Per tant, cridarà toString () en la funció, analitzarà i extraurà la funció d'arguments i, a continuació, utilitzeu el $injector per injectar aquests arguments dins de la inicialització de l'objecte.

El procés d'injecció és semblant a:

```
injector.invoke(function($http, greeter) {});
```

Tingueu en compte que aquest procés només funcionarà amb un codi no minimitzat, no ofuscat, ja que Angular necessita analitzar els arguments intactes. Amb aquesta influencia de JavaScript, l'ordre no és important: Angular ho descobrirà i injectarà les propietats correctes en l'ordre "correcte".

Anotació explícita

Angular proporciona un mètode per definir explícitament les dependències que necessita una funció per realitzar la invocació. Aquest mètode permet que emprant la versió minifier, canvieu el nom dels paràmetres de la funció i encara podreu injectar els serveis adequats a la funció. El procés d'injecció utilitza la propietat $inject per anotar la funció. La propietat $inject d'una funció és un array de noms de serveis que s'injectaran com a dependències. Per utilitzar el mètode de la propietat $inject, el configurem en la funció o al nom.

```
var aControllerFactory =
function aController($scope, greeter) {
console.log("LOADED controller", greeter);
// ... Controller
};
aControllerFactory.$inject = ['$scope', 'greeter'];
// Greeter service
var greeterService = function() {
console.log("greeter service");
}
// Our app controller
angular.module('myApp', [])
.controller('MyController', aControllerFactory)
.factory('greeter', greeterService);
// Grab the injector and create a new scope
var injector = angular.injector(['ng', 'myApp']),
```

```
controller = injector.get('$controller'),
rootScope = injector.get('$rootScope'),
newScope = rootScope.$new();
// Invoke the controller
controller('MyController', {$scope: newScope});
```

Amb aquest estil d'anotació, l'ordre és important, ja que el array $inject ha de coincidir amb l'ordenació dels arguments per injectar. Aquest mètode d'injecció funciona amb minificació, perquè la informació de la anotació serà empaquetada amb la funció.

Anotació Inline

L'últim mètode d'anotació que proporciona Angular fora del quadre és l'anotació en línia. La base sintàctica funciona de la mateixa manera que el mètode $inject de les altres formes d'anotació de dalt, però permet que nosaltres fem els arguments en línia en la definició de la funció. A més, ens proporciona la capacitat de habilitar el no utilitzar una variable temporal en la definició. L'anotació en línia, ens permet passar un array d'arguments en comptes d'una funció en definir un objecte d'angular. Els elements dins d'aquest array són la llista de dependències injectables com a strings, el últim argument és la definició de funció de l'objecte.
Per exemple:

```
angular.module('myApp')
.controller('MyController',
['$scope', 'greeter',
function($scope, greeter) {
}]);
```

El mètode d'anotació en línia funciona amb minifiers, ja que passem una llista de cadenes. Sovint ens referim aquest mètode com a parèntesi o notació d'array [].

$inject API

Tot i que és relativament estrany que hàgim de treballar directament amb $injector, coneixent-ne L'API ens donarà una bona idea sobre com funciona exactament.

La funció annotate () retorna un array de noms de servei que s'injectaran a la funció quan es crea una instància. La funció annotate() la utilitza l'injector per determinar quins serveis s'injectaran a la funció en el moment de la invocació.
La funció annotate () pren un sol argument:

- fn (function o array)

L'argument fn ens dóna una funció o un array en el conjunt de anotacions d'una definició de funció. El mètode annotate () retorna una sola matriu dels noms dels serveis que s'injectaran a la funció en el moment de la invocació.

```
var injector = angular.injector(['ng', 'myApp']);
injector.annotate(function($q, greeter) {});
// ['$q', 'greeter']
```

Després de conèixer totes les bases per injectar dependències, anem a veure un ús pràctic dels mateixos.
Per això tindrem que definir alguns o repassar uns conceptes pràctics:

Que es Ajax?
Ajax es una sol·licitud HTTP realitzada de forma asíncrona amb Javascript, per obtenir dades de un servidor i mostrar-los en el client sense tenir que recarregar la pàgina sencera.

Que es Service $http?
El servei $http (Service en inglés, tal como es coneix en AngularJS) es una funcionalitat que forma part del nucli d'Angular. Serveix per realitzar comunicacions amb servidors, por mitja de HTTP, a través de Ajax i via el objecte XMLHttpRequest natiu de Javascript o via JSONP.

Després d'aquesta denominació formal, que trobem en la documentació d 'AngularJS, ens tindrem que quedar de moment en que ens serveix per realitzar sol·licituds i per això el servei $http te diversos tipus d'accions possibles. Tots es poden invocar a través dels paràmetres de la funció $http i a demés existeixen diversos mètodes alternatius (dreceres o shortcuts) que serveixen per fer coses més especifiques.

Entre els shortcuts hi ha:

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

Tant el propi $http() com les funcions de dreceres et retornen un objecte que amb el "patró promise" ens permet definir una sèrie de funcions a executar quan succeeixen coses, per exemple, que la sol·licitud HTTP s'hagi resolt amb èxit o amb fracàs.

El Service $http es bastant complexa i te moltes coses per aportar solucions a les més variades necessitats de sol·licituds HTTP asíncrones. De moment ens anem a centrar en lo més basic que serà suficient per realitzar un exemple interesant.

Lo primer que necessitarem serà injectar-lo en el controlador, o a on sigui que tinguem que utilitzar-lo. Això es similar a el que mostràvem practicant amb els controladors e injectàvem el $scope.

```
angular
    .module('apiApp', [])
    .controller('apiAppCtrl', ['$http', controladorPrincipal] );
```

Com podeu veure, en la funció del nostre controlador, anomenada controladorPrincipal, li estarem indicant que rebrà un paràmetre a on s'ha de injectar el Service $http.

Al declarar la funció rebràs aquesta dependència com a paràmetre.

```
function controladorPrincipal($http){
```

Ara, dins del codi d'aquest controlador podràs accedir al servei $http per realitzar las trucades a Ajax.

Realitzar una trucada a $http.get()

El mètode get() serveix per fer una sol·licitud de tipus GET. Rep diversos paràmetres, un obligatori, que es la URL i un altre opcional, que es la configuració de la teva sol·licitud.

```
$http.get("http://www.example.com")
```

El més interesant es el que ens retorna aquest mètode, que es un objecte "HttpPromise", sobre el que podem operar per especificar el comportament de la nostra aplicació davant diverses situacions.

Resposta en cas d'èxit

De moment, veiem què tindríem que fer per especificar-li a Angular el que te que fer quan es rebi la resposta correcta del servidor.

```
$http.get(url)
.success(function(respuesta){
//códi en cas d'èxit
});
```

Com es pot veure en aquest codi es que $http ens retorna un objecte. Sobre aquest objecte invoquem el mètode success() que serveix per indicar la funció que tenim que executar en cas d'èxit en la sol·licitud Ajax. Aquesta funció alhora rep un paràmetre que es la resposta que ens ha retornat el servidor.

Exemple complert de sol·licitud Ajax amb $http

Visualitzats aquests nous coneixements sobre el "Service" $http estem en condicions de fer una mica de Ajax per connectar-nos amb un API REST que ens ofereixi unes quantes dades. Aquestes dades son les que utilitzarem en la nostre petita aplicació per mostrar informació.

Encara que senzilla, aquesta aplicació ja conte diverses coses que per una millor comprensió convé visualitzar-ho per separat.

Aquest es el nostre HTML:

```
<div ng-app="apiApp" ng-controller="apiAppCtrl as vm">
  <h1>Probem Ajax</h1>
  <p>
      Selecciona:
      <select ng-model="vm.url" ng-change="vm.buscaEnRegion()">
       <option value="http://restcountries.eu/rest/v1/region/africa">Africa</option>
        <option value="http://restcountries.eu/rest/v1/region/europe">Europa</option>
        <option value="http://restcountries.eu/rest/v1/region/americas">America</option>
      </select>
  </p>
  <ul>
      <li ng-repeat="pais in vm.paises">
        País: <span>{{pais.name}}</span>, capital: {{pais.capital}}
      </li>
  </ul>
</div>
```

Podeu fixar-vos que tenim un camp SELECT que ens permet seleccionar una regió i per a cada un dels OPTION tenim com a value la URL del API REST que utilitzaríem per obtenir els països d'aquesta regió.

Veiem que en el camp SELECT està col·locada la directiva ngChange, que s'activa quan canviem el valor seleccionat en el combo. En aquest cas es realitza una trucada a un mètode anomenat buscaEnRegion() que veurem després escrit en el controlador.

També trobaràs una llista UL en la que tenim una sèrie d'elements LI. Aquests elements LI tenen la directiva ngRepeat per reiterar sobre un conjunt de països, de mode que tinguem un element de la llista per cada país.

Ara podem veure el exemple en el Javascript següent:

```
angular
    .module('apiApp', [])
    .controller('apiAppCtrl', ['$http', controladorPrincipal]);


function controladorPrincipal($http){
    var vm=this;


    vm.buscaEnRegion = function(){
```

```
        $http.get(vm.url).success(function(respuesta){
            //console.log("res:", respuesta);
            vm.paises = respuesta;
        });
    }
}
```

Crearem un module i després un controlador al que injectarem el $http como s'ha explicat al inicio del tema.

Després, en la funció que construeix el controlador, tenim un mètode que s'anomena buscaEnRegion() que es el que s'invoca al modificar el valor del SELECT. Aquest mètode es el que conte la trucada Ajax.

Realment el Ajax d'aquest exercici es limita al següent codi:

```
$http.get(vm.url).success(function(respuesta){
//console.log("res:", respuesta);
vm.paises = respuesta;
});
```

Utilitzem el shortcut $http.get() passant com a paràmetre la URL, que obtenim del value que hi ha marcat en el camp SELECT del HTML. Després s'especifica una funció per el cas "success" amb el patró "promise". Aquesta funció retorna en el paràmetre "respuesta" el que ens va retornar el API REST, que es un JSON amb les dades dels països d'una regió. En el exemple, en caso d'èxit, simplement bolquem en una dada del model, en el "scope", el contingut de la resposta.

En concret veiem que la resposta es volca en la variable vm.paises, que es justament la col·lecció per la que se itera en el ng-repeat del nostre HTML.