Factory

Introducció al Factory

Què són les factories de AngularJS, factory en la terminologia Angular. Per a què serveixen, quins trets les caracteritzen i com crear-les.

Fins al moment en el Manual d'AngularJS nostre codi Javascript ho hem col·locat principalment dins de controladors. Ens han servit perfectament per a les nostres aplicacions, però ja hem pogut observar que en certes ocasions els controladors no es comporten com nosaltres podríem desitjar.

Aspectes d'aquesta problemàtica deuen haver quedar clars si consulteu anteriors temes, on vam aprendre a crear controladors en paral·lel i on vam introduir $ location. En aquest exercici vam veure que els controladors s'invoquen amb cada vista on els estiguem utilitzant, executant la funció que els defineix cada vegada que es carrega la vista. Per aquest motiu totes les dades que s'inicialitzen en els controladors es tornen a posar als valors per defecte quan carreguem qualsevol vista que treballi amb aquest controlador.

Una gran funcionalitat de les factories, es solucionar la pèrdua de dades dels controladors quan canviem la vista. Així que anem a aprendre com crear-les i emprar-les.

Però aquest no és l'únic cas on trobaràs utilitat en les factories. Per exemple, algunes de les necessitats que podríem tenir i que els controladors no ens resolen són:

1) Compartir dades entre diversos controladors, el que permet tenir aplicacions de veritat, capaços de memoritzar estats entre diverses de les seves pantalles.

2) Compartir dades entre diverses vistes diferents. Per descomptat, sense usar les temudes variables globals.

Això és justament el que vam veure en tems anteriors, que ja tinguis un o diversos controladors, no comparteixen ni es memoritzen estats en passar d'una vista a una altra.

A més, potser el més representatiu dels usos de les factories és:

3) Empaquetar operacions comuns a diversos controladors (per exemple en una aplicació de facturació podríem necessitar calcular l'IVA o accedir als diferents tipus d'IVA en diversos punts del sistema). Per descomptat, no volem col·locar el codi d'aquests càlculs o operacions repetit en tots els controladors que han de utilitzar-los i tampoc volem crear funcions amb àmbit global.

Què son les factories

Les factories són com a contenidors de codi que podem fer servir en els nostres llocs desenvolupats amb AngularJS. Són un tipus de servei, "Service" en Angular, amb el qual podem implementar llibreries de funcions o emmagatzemar dades.

Quan les fem servir tenen la particularitat de retornar una dada, de qualsevol tipus. El comú és que ens tornin un objecte de Javascript on podrem trobar dades (propietats) i operacions (mètodes). Amb diferència dels controladors, les factories tenen la característica de ser instanciats una única vegada dins de les aplicacions, de manera que no perden el seu estat. Per tant, són un bon candidat per emmagatzemar dades a la nostra aplicació que vulguem fer servir al llarg de diversos controladors, sense que s'inicialitzin de nou quan es canvia de vista.

Angular aconsegueix aquest comportament usant el patró "Singleton" que bàsicament vol dir que, cada vegada que es necessiti un objecte d'aquest tipus, s'enviarà la mateixa instància d'aquest objecte en lloc de tornar a instanciar un exemplar.

Nota: El patró "Singleton" no és una cosa específic de AngularJS, en realitat és un patró general de programació orientada a objectes. Així com les factories en línies generals també són un conegut patró de disseny de programari que es fa servir en el desenvolupament d'aplicacions web i aplicacions tradicionals orientades a objectes.

Notes sobre els "services" en Angular

Els "services" en AngularJS inclouen tant factories com serveis. El que és vol definir és que aquests contenidors de codi ja els hem fet servir en diverses ocasions i potser és pot entendre millor la seva utilitat si analitzem coses que ja coneixem.

Per exemple, quan estem fent Ajax, pels mètodes que hem conegut fins ara, fem servir $http. Aquest no és més que un service de Angular que engloba tota la funcionalitat necessària per a realitzar sol·licituds asíncrones a un servidor.

Per tant, alguna cosa com Ajax, que se suposa que es pot desitjar realitzar al llarg de la aplicació en diverses parts del codi, s'ha separat a un "servei". Això vol dir que, quan es vulgui emprar el Ajax, s'hauran de fer servir el codi del "service" $http.

Els serveis i factories que es desitgin fer servir en els controladors o mòduls s'hauran injectar com s'ha vist fer en diverses parts dels nostres exemples. No us preocupeu si no ho recordeu be perquè a continuació veurem exemples.

Exemple de factoria en AngularJS

Ara ens posarem mans a l'obra creant la nostra primera factoria. Com veureu, volem implementar un sistema que ens memoritzi certa informació de la nostra aplicació al llarg de diverses vistes. Implementem factories amb el mètode factory () que depèn del mòdul (objecte module) de la nostra aplicació. O sigui, com qualsevol aplicació de Angular, el primer pas serà crear el "module" principal:

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

I sobre l'objecte que retorna aquesta operació crearem les factories.

Nota: Com observaràs, el mecanisme per crear la factoria és el mateix que fem per crear els controladors. Per crear el controlador fas servir el mètode controller () i per a la factoria el mètode factory ().

```
.factory("descargasFactory", function(){
    var descargasRealizadas = ["Manual de Javascript", "Manual de jQuery", "Manual de
AngularJS"];

    var interfaz = {
        nombre: "Manolo",
        getDescargas: function(){
            return descargasRealizadas;
        },
        nuevaDescarga: function(descarga){
            descargasRealizadas.push(descarga);
```

```
        }
    }
    return interfaz;
})
```

Aquesta factoria es diu "descargasFactory". El nom l'hem definit en la crida al mètode factory. Recordeu-vos d'aquest nom, ja que després s'haurà de fer servir per injectar la dependència d'aquesta factoria en els controladors. Aquest codi té una sèrie de detalls interessants, des del punt de vista de Angular i també des del de Javascript en general, perquè entren en joc diversos conceptes de la programació orientada a objectes en Javascript. De totes maneres, et anem a resumir una mica el que trobem.

• El més destacat sobre les factories en AngularJS el trobes en l'última línia: "return interfaz;" Totes les factories han de tornar alguna cosa. El que sigui, encara que l'habitual com vam dir és retornar un objecte. Per definició te que ser així en AngularJS.

• Allò que tornes és el que es coneix des de fora de la factoria. Per dir-ho d'una altra manera, és la interfície pública d'ús d'aquesta factoria. Per això hem cridat a la variable que tornem al return "interfaz", perquè és la sèrie de propietats i mètodes que estàs fent públic per a tothom que utilitzi aquesta factoria. Lògicament, aquesta "interfície" no és més que una manera nostra de cridar a la variable i tu utilitzaràs la que vulguis.

• Però fixa't que la variable "descargasRealizadas" és privada a la factoria, ja que no es torna a la interfície. Per tant, aquest array no podrà ser accessible des de fora de la factoria. Podem entendre-ho com una propietat privada.

• Per accedir al array "descargasRealizadas" es farà ús dels mètodes definits en "interfaz": getDescargas () i nuevaDescarga (). Aquests mètodes són públics, per haver-los definit en la interfície que tornem a la funció de la factoria i es podran accedir des de qualsevol lloc on tinguem disponible la factoria.

• No obstant això no totes les dades que anem a fer servir en les factories necessitem fer-les privades. En concret trobaràs, la propietat "nom" que està dins la nostra interfície i per tant és pública i podrà accedir-se tal qual des de fora de la factoria.

Nota: Per entendre els avantatges fer les coses públiques o privades hauràs de conèixer alguns dels conceptes bàsics de programació en general i programació orientada a objectes en particular, com són l'abstracció i l'encapsulament. És una decisió de disseny de programari, que ha de aprendre el desenvolupador i tant Javascript com per extensió AngularJS tenen mecanismes per implementar elements públics o privats. Així, com a regla global en la programació orientada a objectes, tot dada hauria de ser definit com privat, tret que algú de fora et demani que ho exposis públicament.

Veure ejercici3.html i ejercici3.js.

Utilitzar una factoria

Ara podem veure com fer servir la factoria que acabem de realitzar. El procediment és tan simple com, un cop definida, injectar al controlador on la volem utilitzar. Fem servir el sistema d'injecció de dependències que ja coneixes.
En crear la funció del controlador hem de definir un paràmetre que es digui exactament igual al nom que li has donat a la factoria. En aquest cas el paràmetre que injectem al controlador es diu "descargasFactory" doncs així havíem cridat a la factoria al crear-la. Ara veurem un controlador que fa servir aquesta factoria.

```
.controller("appCtrl", function(descargasFactory){
    var vm = this;

    vm.nombre = descargasFactory.nombre;
    vm.descargas = descargasFactory.getDescargas();
    vm.funciones = {
        guardarNombre: function(){
            descargasFactory.nombre = vm.nombre;
        },
        agregarDescarga: function(){
            descargasFactory.nuevaDescarga(vm.nombreNuevaDescarga);
            vm.mensaje = "Descarga agregada";
        },
        borrarMensaje: function(){
            vm.mensaje = "";
        }
    }
});
```

Dins del nostre controlador la variable descargasFactory que rebem com a paràmetre conté totes les dades i funcions que hem definit en la interfície pública de la nostra factoria.

Per tant:

- descargasFactory.nombre contindrà la propietat "nombre" definida en la factory.
- descargasFactory.nuevaDescarga() o descargasFactory.getDescargas() seran trucades a els mètodes que havíem definit en la factoria.

Nota: Compte, des de la teva vista, en l'HTML, no seràs capaç d'accedir a la factoria. És una cosa que ara pertany al controlador i mitjançant el scope creat per aquest controlador no pots accedir a les dades que tenen les seves dependències. Si vols accedir a una dada d'aquesta factoria el mecanisme és bolcar aquesta dada en el scope. Això ho fem al principi d'aquest controlador en les línies:

```
vm.nombre = descargasFactory.nombre;
vm.descargas = descargasFactory.getDescargas();
```

Injectant dependències

Igual que en el service es podien injectar dependències al constructor, en un factory es poden injectar dependències en la funció de factoria.
Anem a veure un exemple addicional, per deixar clar aquest concepte:

```
1   var app=angular.module("app",[]);
2
3   app.value("tamanyoInicialRectangulo",{
4     ancho:2,
5     alto:3
6   });
7
8   function Rectangulo(tamanyoInicial) {
9     this.ancho=tamanyoInicial.ancho;
10    this.alto=tamanyoInicial.alto;
11
12    this.setAncho=function(ancho) {
13      this.ancho=ancho;
14    }
15
16    this.setAlto=function(alto) {
17      this.alto=alto;
18    }
19
20    this.getArea=function() {
21      return this.ancho * this.alto;
22    }
23  }
24
25  app.factory("rectangulo",['tamanyoInicialRectangulo',function(tamanyoInicialRectangulo) {
    var rectangulo=new Rectangulo(tamanyoInicialRectangulo);
```

```
26
27      return rectangulo;
28  }]);
29
30
31  app.controller("PruebaController",["$scope","rectangulo",function($scope,rectangulo) {
32      $scope.area=rectangulo.getArea();
33  }]);
34
```

- Línia 25: Com ja hem vist en les funcions dels controladors, injectem la dependència tamanyoInicialRectangulo simplement afegint l'array amb el nom i posant la variable com a argument de la funció.
- Línia 26: Aquí és quan es veu realment què és una funció de factoria. Estem creant el valor del servei en crear l'objecte rectangle. Veiem com li estem passant al constructor, el valor que injectem a la funció.
- Línia 28: com ja sabem s'ha de retornar el valor del servei.

La necessitat del factory

Anem a explicar ara un exemple en què necessitem la funció factory ja que la creació del valor del servei va a ser una mica complexa.

Anem a suposar que necessitem enviar el hash d'una contrasenya a un servidor. Tindrem un servei de AngularJS anomenat hash que és una funció. Aquesta funció acceptarà com a paràmetre un String i ens retornarà el hash en format Base64. Però volem que aquest servei sigui reutilitzable pel que va a suportar diversos tipus de funcions de Hash: MD5, SHA1, SHA2-256 i SHA-2-512.

Per implementar les diferents funcions de Hash farem servir la llibreria CryptoJS.

Les diferents funcions de Hash que té són:

https://code.google.com/p/crypto-js/#MD5: CryptoJS.MD5

https://code.google.com/p/crypto-js/#SHA-1: CryptoJS.SHA1

https://code.google.com/p/crypto-js/#SHA-2: CryptoJS.SHA256 i CryptoJS.SHA512

Important!!! Actualment aquest aquest projecte esta en StandBy y nomes es pot descarregar la llibreria en local i com a conseqüència fer la crida dins d'aquest àmbit. Podeu obtenir-la en el següent enllaç:

https://code.google.com/archive/p/crypto-js/downloads

I la forma de generar el resultat en Base64 és mitjançant The Hasher Output usant la funció: hash.toString (CryptoJS.enc.Base64)

Un exemple de tot això és el següent:

```
<Script src = "sha256.js"> </ script>

<Script src = "enc-base64-min.js"> </ script>

<Script>

   var hash = CryptoJS.SHA256 ( "Message");

   alert (hash.toString (CryptoJS.enc.Base64)); // L3dmip37 + NWEi57rSnFFypTG7ZI25Kdz9tyvpRMrL5E =

</ Script>
```

Per configurar el servei és necessari crear un value anomenat "algoritmo" amb l'algoritme que volem utilitzar.

```
1   app.value("algoritmo","SHA-1");
```

És aquest cas hem configurat l'algoritme perquè sigui "SHA-1".
Consumir el servei hash és tan senzill com passar el String i ens retorna altre.

```
1   app.controller("PruebaController",["$scope","hash",function($scope,hash) {
2     $scope.password="s3cret";
3     $scope.getHash=function(message) {
4       var hashResult=hash(message);
5       return hashResult;
6     }
7   }]);
```

- Línia 1: Al controlador injectem la funció de hash anomenada hash.

- Línia 3: S'ha creat al $ scope una funció anomenada getHash que accepta un String com a paràmetre i retorna el resultat de cridar a la funció de Hash.

El codi HTML és el següent:

index.html

```
1   <!DOCTYPE html>
2   <html ng-app="app">
3
4     <head>
5       <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.min.js"></script
6       <script src="script.js"></script>
7       <script src="md5.js"></script>
8       <script src="sha1.js"></script>
9       <script src="sha256.js"></script>
10      <script src="sha512.js"></script>
11      <script src="enc-base64-min.js"></script>
12    </head>
13
14    <body ng-controller="PruebaController">
15      Contrase&ntilde;a:<input ng-model="password" />
16      <br>
17      El hash de la contrase&ntilde;a es:
18      <br>
19      {{getHash(password)}}
20    </body>
21  </html>
```

- Línies 7-11: Carreguem la llibreria amb els 4 algoritmes de Hash a usar i la codificació en Base64.
- Línia 15: Tenim un tag <input> perquè l'usuari escrigui la contrasenya.
- Línia 18: Es crida a la funció getHash perquè mostri el hash. Ara ve la part interessant, hem de crear un servei mitjançant un factory que segons el valor del value "algoritmo", utilitzi una funció de Hash o una altra però a més volem que el resultat de cridar a aquesta funció sempre estigui codificat en Base64.

El factory

Com ja hem dit el nostre servei anomenat hash ens retornarà una funció que realitza el hash però amb l'algoritme configura ble i el resultat codificat en Base64.

Vegem com queda la funció:

```
app.factory("hash",['algoritmo',function(algoritmo) {
  var hashFunction;

  if (algoritmo==="MD5") {
    hashFunction=CryptoJS.MD5;
  } else if (algoritmo==="SHA-1") {
    hashFunction=CryptoJS.SHA1;
  } else if (algoritmo==="SHA-2-256") {
    hashFunction=CryptoJS.SHA256;
  } else if (algoritmo==="SHA-2-512") {
    hashFunction=CryptoJS.SHA512;
  } else {
    throw Error("El tipo de algoritmo no es válido:"+algoritmo);
  }

  var hash=function(message) {
    var objHashResult=hashFunction(message);

    var strHashResult=objHashResult.toString(CryptoJS.enc.Base64);

    return strHashResult;
  }

  return hash;

```

```
}]);
```

- Línia 1: Injectem el valor de l'algoritme a utilitzar.
- Línies 4 a 14: En funció del valor de l'algoritme fem servir una funció o una altra de Hash. Però aquesta no serà la veritable funció que anem a retornar ja que el resultat d'aquesta funció és un Objecte però nosaltres volem que sigui un String codificat en Base64.
- Línia 16: Ara anem a crear la veritable funció, la qual és la que retornarà el servicio. I que és la que realment serà trucada des del controlador. Veiem que accepta com a únic argument el missatge.
- Línia 17: Fem una crida a la funció que veritablement genera el Hash.
- Línia 19: Ara transformem l'objecte en un String en Base64 cridant al mètode toString.
- Línia 24: Aquí és on la funció factory retorna la funció de hash que hem creat.

Veure ejercici5.html, ejercici5.js, sha512,js, md5.js i sha1.js.

value vs service vs factory

Aquesta és la típica pregunta que apareix sempre en els fòrums de AngularJS i la resposta és bastant senzilla. Realment és igual quin facis servir perquè tots acaben sent un provider per AngularJS. Però així i quin és més recomanable?

Si tens una classe de la qual és necessari crear una instància millor fa servir un service: Només hauràs de passar-li el nom de la classe i ja està.

Si tens directament el valor, millor fer servir el value: Només hauràs de passar-li aquest valor i ja està.

Si no queda més remei llavors usa el factory: És el més complex d'usar així que hauria de ser sempre l'última opció.

En el següent tema per fi veurem el provider el qual té una funcionalitat extra al factory que si la necessitem estarem obligats a usar-lo.

En aquesta discussió no hem parlat del constant ja que aquest és diferent als 3 anteriors, ja que permet ser injectat en un bloc config i en un provider, cosa que cap dels 3 anteriors permet, de manera que si necessitem aquesta funcionalitat haurem utilitzar obligatòriament el constant.