Cómo programar orientado a Componentes con Directivas de Angular 1.x

Introducción

Normalmente si sigues el tutorial de la página de AngularJS o los cursos online de CodeSchool y CodeAcademy para aprender a usar AngularJS, encontrarás que todos los ejemplos los hacen utilizando $scope.

$scope es un servicio de AngularJS que nos permite comunicar la vista (HTML) con el código de controlador (JavaScript).

El problema es que en la nueva versión de Angular (La 2.0) el $scope ya no existe y hay varias guías de buenas prácticas dónde nos recomiendan no utilizarlo y emplear en su lugar el objeto this en el controlador y la directiva controller as para utilizar un alias en la vista.

Desde hace tiempo, a la hora de programar en AngularJS utilizo estas buenas prácticas. En concreto las de John Papa, un Google Developer Expert. Si las sigues, tu código de AngularJS será más limpio, legible y mantenible. Además lo tendrás preparado para cuando quieras dar el salto a Angular 2.0, ya que la nueva versión sigue esta metodología en cierta medida.

Otra buena guía de buenas prácticas en AngularJS es la de Todd Motto, otro Google Developer Expert. Ambas son grandes guías y pueden complementarse.

Forma tradicional

Imaginemos una pequeña aplicación web con AngularJS que pinte una lista de productos. Simplemente tenemos nuestro index.html y nuestro código javascript en app.js donde estará el controlador de la vista y un servicio que simula la entrega de los datos.

Siguiendo la forma tradicional, utilizamos $scope para acoplar la vista al código.

Esto sería el HTML

```
<div ng-app="myApp" ng-controller="MyController">
 <h1>Listado de Productos</h1>
  <ul>
    <li ng-repeat="producto in productos">
      <strong>{{ producto.titulo }}</strong>: {{ producto.precio | currency }}
    </li>
  </ul>
</div>
```

Y esto nuestro fichero app.js

```
angular
  .module('myApp',[])
  .controller('MyController', MyController)
  .service('MyService', MyService);

function MyController ($scope, MyService) {
  $scope.productos = MyService.getData();
```

```
}

function MyService () {
  return {
    getData: getData
  }

  function getData () {
      var datos = [
        { titulo: "Producto 1", precio: 2 },
        { titulo: "Producto 2", precio: 1.5 },
        { titulo: "Producto 3", precio: 4.2 },
        { titulo: "Producto 4", precio: 3 },
        { titulo: "Producto 5", precio: 2.5 }
      ];
      return datos;
  }
}
```

Prescindiendo de $scope

La alternativa a $scope es utilizar un alias en el controlador con Controller As y emplear el objeto this en el controlador para acceder a los elementos de la vista.

Este sería el cambio en la vista HTML:

```
<div ng-app="myApp" ng-controller="MyController as my">
 <h1>Listado de Productos</h1>
  <ul>
    <li ng-repeat="producto in my.productos">
      <strong>{{ producto.titulo }}</strong>: {{ producto.precio | currency }}
    </li>
  </ul>
</div>
```

Y en el código de app.js vemos el cambio de $scope por this.

```
angular
  .module('myApp',[])
  .controller('MyController', MyController)
  .service('MyService', MyService);

function MyController (MyService) {
  this.productos = MyService.getData();
}

function MyService () {
  return {
    getData: getData
  }

  function getData () {
      var datos = [
        { titulo: "Producto 1", precio: 2 },
        { titulo: "Producto 2", precio: 1.5 },
        { titulo: "Producto 3", precio: 4.2 },
        { titulo: "Producto 4", precio: 3 },
```

```
        { titulo: "Producto 5", precio: 2.5 }
      ];
      return datos;
  }
}
```

Esto es una buena práctica porque nos permite tener controladores anidados y que no entren en conflicto.

Además el código es más legible y te prepara para poder usar los controladores como clases de ECMAScript 6 que será lo que utilicemos en el futuro con Angular 2.0.

Usando rutas y vistas

Cuando nuestra aplicación se va volviendo más compleja necesitamos echar mano de rutas. Para ello en nuestro HTML debemos añadir la directiva ng-view para indicar a Angular dónde se insertarán las vistas asociadas a las rutas.

Aparte debemos incluir entre los scripts la librería de angular-route.js ya que éste módulo no forma parte del core de Angular.

```
<div ng-app="myApp">
  <div ng-view></div>
</div>
```

En nuestro código JavaScript debemos incluir el módulo ngRoute dentro de las dependencias de nuestro módulo principal myApp, para poder acceder al servicio $routeProvider

Las rutas se configuran dentro del config de módulo, y cada ruta lleva su URL al template HTML que se cargará con determinada ruta, su controlador y su alias.

El código para utilizar el controlador y plantilla que estábamos usando hasta ahora, pero con rutas, sería el siguiente:

```
angular
  .module('myApp',['ngRoute'])
  .config(appConfig)
  .controller('MyController', MyController)
  .service('MyService', MyService);

function appConfig ($routeProvider) {
  $routeProvider
    .when('/', {
      templateUrl: 'tpl/listado.html',
      controller: 'MyController',
      controllerAs: 'my'
    });
}

function MyController (MyService) {
  this.productos = MyService.getData();
}

function MyService () {
  return {
    getData: getData
  }

  function getData () {
      var datos = [
        { titulo: "Producto 1", precio: 2 },
```

```
        { titulo: "Producto 2", precio: 1.5 },
        { titulo: "Producto 3", precio: 4.2 },
        { titulo: "Producto 4", precio: 3 },
        { titulo: "Producto 5", precio: 2.5 }
      ];
      return datos;
  }
}
```

Y el HTML de la plantilla, sería tal que así en un fichero tpl/listado.html como hemos definido en el $routeProvider.

```
<h1>Listado de Productos</h1>
<ul>
  <li ng-repeat="producto in my.productos">
    <strong>{{ producto.titulo }}</strong>: {{ producto.precio | currency }}
  </li>
</ul>
```

Creando una directiva

En Angular tenemos directivas, que actualmente son lo que conocemos como WebComponents, pero en el momento que se creó Angular esto aún no existía, es por ello que en Angular 2 se utilizarán los WebComponents de forma nativa para ello.

Creando directivas, podemos reutilizar controladores y vistas, creando de esta forma componentes reutilizables. Es el método de programación orientada a componentes que siguen librerías como Polymer o React

Con AngularJS podemos seguir esta metodología utilizando directivas.

Si asociamos el controlador anterior y el HTML de la plantilla en una directiva, el código sería el siguiente:

```
function miDirectiva () {
  return {
    scope: {},
    templateUrl: 'tpl/listado.html',
    controller: 'MyController',
    controllerAs: 'my'
  }
}
```

De esta manera, tendríamos una directiva o tag HTML así: , que nada nos impide utilizarlo en la configuración de las rutas en $routeProvider con la propiedad template, quedando más legible esta configuración.

```
function appConfig ($routeProvider) {
  $routeProvider
    .when('/', {
      template: '<mi-directiva></mi-directiva>'
    });
}
```

Simplificando la directiva como componente

Aunque hemos creado una directiva seguimos utilizando una plantilla HTML externa y un controlador.

En las directivas de Angular podemos englobar todo dentro de la misma función utilizando los atributos controller, controller as y template.

El app.js complto con la configuración, el servicio y la directiva sería el siguiente:

```
angular
  .module('myApp',['ngRoute'])
  .config(appConfig)
  .service('MyService', MyService)
  .directive('miDirectiva', miDirectiva);

function appConfig ($routeProvider) {
  $routeProvider
    .when('/', {
      template: '<mi-directiva></mi-directiva>'
    });
}

function miDirectiva () {
  return {
    scope: {},
    controller: function(MyService) {
      this.productos = MyService.getData();
    },
    controllerAs: 'vm',
    template: [
      '<h1>Listado de Productos</h1>',
      '<ul>',
        '<li ng-repeat="producto in vm.productos">',
          '<strong>{{ producto.titulo }}</strong>: {{ producto.precio | currency }}',
        '</li>',
      '</ul>'
    ].join('')
  }
}

function MyService () {
  return {
    getData: getData
  }

  function getData () {
      var datos = [
        { titulo: "Producto 1", precio: 2 },
        { titulo: "Producto 2", precio: 1.5 },
        { titulo: "Producto 3", precio: 4.2 },
        { titulo: "Producto 4", precio: 3 },
        { titulo: "Producto 5", precio: 2.5 }
      ];
      return datos;
  }
}
```

Anidando directivas y usando atributos para pasar datos

Vale, esto está muy bien pero que pasa si quieros usar directivas dentro de otras, y poder pasarle parámetros de una a otra.

Muy sencillo, imaginemos que pintará una lista de componentes que ahora definiremos, pero estos tienen un atributo data donde le pasamos el objeto que queramos que pinte.

Nuestra directiva será así:

```
function miDirectiva () {
  return {
    scope: {},
    controller: function(MyService) {
      this.productos = MyService.getData();
    },
    controllerAs: 'vm',
    template: [
      '<h1>Listado de Productos</h1>',
      '<ul>',
        '<mi-item ng-repeat="producto in vm.productos" data="producto">',
        '</mi-item>',
      '</ul>'
    ].join('')
  }
}
```

Ahora debemos implementar teniendo en cuenta el atributo data.

Ésta directiva no necesita controlador porque ya le pasamos los datos dentro del scope interno.

Para pasar datos entre directivas ya escribí un artículo anterior que te invito a que le des un vistazo para saber cuándo utilizar =, @ o &. En este caso asocio el atributo data con = porque no le paso los datos desde una expresión con doble llave {{ ... }}, si no como variable.

Dentro del template de la directiva puedo usar ese data para acceder a los datos.

```
function miItem () {
  return {
    scope: {
      data: '='
    },
    template: [
      '<li>',
        '<strong>{{ data.titulo }}</strong>: ',
        '{{ data.precio | currency }}',
      '</li>'
    ].join('')
  }
}
```

El código completo del app.js con las dos directivas será:

```
angular
  .module('myApp',['ngRoute'])
  .config(appConfig)
  .service('MyService', MyService)
  .directive('miItem', miItem)
  .directive('miDirectiva', miDirectiva);
```

```
function appConfig ($routeProvider) {
  $routeProvider
    .when('/', {
      template: '<mi-directiva></mi-directiva>'
    });
}

function miDirectiva () {
  return {
    scope: {},
    controller: function(MyService) {
      this.productos = MyService.getData();
    },
    controllerAs: 'vm',
    template: [
      '<h1>Listado de Productos</h1>',
      '<ul>',
        '<mi-item ng-repeat="producto in vm.productos" data="producto">',
        '</mi-item>',
      '</ul>'
    ].join('')
  }
}

function miItem () {
  return {
    scope: {
      data: '='
    },
    template: [
      '<li>',
        '<strong>{{ data.titulo }}</strong>: ',
        '{{ data.precio | currency }}',
      '</li>'
    ].join('')
  }
}

function MyService () {
  return {
    getData: getData
  }

  function getData () {
      var datos = [
        { titulo: "Producto 1", precio: 2 },
        { titulo: "Producto 2", precio: 1.5 },
        { titulo: "Producto 3", precio: 4.2 },
        { titulo: "Producto 4", precio: 3 },
        { titulo: "Producto 5", precio: 2.5 }
      ];
      return datos;
  }
}
```

Por qué seguimos esta forma de programar?

En las próximas versiones de Angular 1.x (la 1.5 y 1.6 al menos) tendremos novedades que nos permitirán adaptarnos a la versión 2.

Una de ellas es el nuevo router. Actualmente está como una librería aparte que surgió cuando la versión 1.4, pero se prevé que se incluya en la versión 1.5 (actualmente en beta)

Este nuevo router funciona orientado a componentes, por lo que tendríamos un código de configuración similar a éste:

```
angular
  .module('myApp', ['ngNewRouter'])
  .controller('AppController', AppController);

function AppController($router) {
  $router.config([
    { path: '/', component: 'miDirectiva' }
  ])
}
```

Y también en Angular 1.5 tendremos el método component que sustituye de alguna forma a directive como vemos en éste artículo de Todd Motto

En forma de directiva

```
angular
  .module('myApp')
  .directive('miDirectiva', miDirectiva);

function miDirectiva () {
  return {
    scope: {},
    controller: function(MyService) {
      this.productos = MyService.getData();
    },
    controllerAs: 'vm',
    template: [
      '<h1>Listado de Productos</h1>',
      '<ul>',
        '<mi-item ng-repeat="producto in vm.productos" data="producto">',
        '</mi-item>',
      '</ul>'
    ].join('')
  }
}
```

En forma de componente

```
angular
  .module('myApp')
  .component('miDirectiva', {
    controller: function(MyService) {
      this.productos = MyService.getData();
    },
    template: [
      '<h1>Listado de Productos</h1>',
      '<ul>',
        '<mi-item ng-repeat="producto in vm.productos" data="producto">',
        '</mi-item>',
      '</ul>'
    ].join('')
  })
```

Por tanto, programar orientado a directivas (o componentes) aparte de ser una buena práctica nos permite prepararnos para las nuevas

versiones de AngularJS (1.5 y 1.6 al menos) y la nueva 2.0, además de utilizar la nueva metodología de componentizar nuestras aplicaciones.