Taller de AngularJS

ANGULARJS

by Google

Miguel Angel Alvarez

miguel@desarrolloweb.com
@midesweb

Taller de AngularJS

1.- Introducción a AngularJS
2.- Trabajando Scope y directivas
3.- Trabajando con Vistas
4.- Trabajando con Factorías
5.- Acceso a datos remotos: Ajax / API

1.1 Conceptos generales sobre el desarrollo web

Javascript

Es un lenguaje de programación

- Originario como scripting en el navegador
- Actualmente de propósito general (NodeJS)
- Nativo en diversas plataformas de ordenadores o dispositivos (Windows, Firefox OS…)

Cliente / Servidor

En el desarrollo web la programación se puede dar tanto en el cliente como el servidor.

- Cliente, se ejecuta en el navegador
- Servidor, se ejecuta en el servidor web
- Frontend / Backend

Ajax

Operación de solicitud al servidor sin necesidad de recargar toda la página.

- Se hace con programación del lado del cliente, Javascript.
- Se usa comúnmente una librería JS, como jQuery.
- Asíncrono

Single Page Application (SPA)

Es un modelo de aplicación web que sólamente tiene un punto de entrada y donde toda la operativa se desarrolla en la misma página.

- Una página, pero puede haber varias vistas
- Aplicaciones que se parecen a las de escritorio
- Uso intensivo de Ajax
- A menudo se encuentran aplicaciones híbridas

MVC

Patrón de arquitectura de software.

- Usado en el desarrollo en general
- Modelo / Vista / Controlador
- Separación del código por sus intereses o responsabilidades

Framework

Definición de arquitectura y librería para ordenar y acelerar el desarrollo.

- Funcionalidades comunes de las aplicaciones ya implementadas
- Definen un estándar en la codificación de aplicaciones, usan patrones variados, como MVC

Framework Javascript (frontend)

Un framework que se ejecuta en el lado del cliente.

- Enfocados en la realización de aplicaciones web
- Se especializa en la parte visual, interfaces
- Usan datos ofrecidos por algún servidor
- AngularJS es un ejemplo, EmberJS...

Binding

Binding es un enlace. Es el enlace que hay entre elementos de una aplicación, de modo que los cambios o acciones en uno, repercuten en cambios o acciones en otro.

- Suscripción a eventos

AngularJS

Framework Javascript del lado del cliente.

- Compatible tanto con navegadores de escritorio como navegadores móviles
- Usa el patrón MVC (o MVVM o MV*)
- Especialmente indicado para SPA
- Caracterizado por el **doble binding**

1.2

Práctica: "Hola Mundo" en AngularJS

Directivas

Enriquecen el HTML, permitiendo de una forma declarativa, indicar la función que tienen ciertos elementos dentro de una aplicación web.

- ngApp => ng-app
- ngModel => ng-model
- ngClick => ng-click

Expresiones

Se colocan entre dos llaves {{dobles}}. Sirven para volcar datos en la página y admiten un pequeño conjunto de operaciones Javascript.

- Operadores diversos
- Operador ternario
- Nunca expresiones de control

Binding & Two way Binding

Enlace que hemos apreciado, cambiando los datos en un lugar, repercute en el otro.

- Expresiones: "one way binding"
- ng-model: "two way binding"

1.3

Desarrollo declarativo

Directiva ngInit

Inicializar datos de la aplicación.

- Ojo a usos inadecuados.

Directiva ngRepeat

Realizar repeticiones, iterando sobre los elementos de un array o las propiedades de un objeto.

```
<ul>
    <li ng-repeat="pelicula in peliculas track by $index">
        {{pelicula}}
    </li>
</ul>
```

1.4

Module Controller

angular.module()

Los "module" son contenedores de código aislados.

- Una aplicación puede tener varios módulos (Al menos 1)
- Permiten encapsular código, exponiendo hacia fuera solo lo que sea necesario
- Facilitan la reutilización

- Ojo, no confundir con lo que sería un módulo del MVC

angular.module()

Definimos módulos en AngularJS mediante el método module() del objeto angular

- angular es el único objeto que expone AngularJS al ámbito global. (variable global)
- Indicamos el nombre del módulo y el array de dependencias
- Devuelve un objeto module sobre el que podremos registrar diversas piezas de código de la aplicación.

```
var app = angular.module("miApp", []);
```

ng-app="miModulo"

En el HTML definimos el módulo que va a contener el código de mi aplicación.

- Generalmente se coloca en la etiqueta HTML o BODY, pero puede colocarse más localizadamente.

```
<body ng-app="miApp">
```

angular.module().controller()

controller() es un método perteneciente a angular.module. Sirve para crear un controlador, que tiene como objetivo inicializar una aplicación.

```
var app = angular.module("miApp", []);
app.controller("MiAppController", function(){
    console.log("inicializo")
});
```

ng-controller="MiAppController"

Declaramos en qué parte de la página se usa el controlador.

- Puede ser en la misma etiqueta del ngApp o en un elemento anidado más específico.
- Dentro de la etiqueta, y sus hijas, tengo acceso a los datos inicializados en el controlador.

```
<body ng-app="miApp" ng-controller="MiAppController">
```

1.5

Inyección de dependencias

Inyección de dependencias

Es un patrón de diseño de software orientado a objetos usado en AngularJS.

- Si un objeto necesita de otro para funcionar, no lo instancia, sino que se le envía por parámetro al método que lo necesita.
- **Inyector de dependencias**: construye y envía automáticamente las dependencias a los métodos que las requieren.

Inyectamos el "scope"

Los controladores generalmente necesitarán que se les inyecte el scope.

* Vamos a entender el scope de momento como un contenedor de propiedades y métodos.

```
angular
    .module("miApp", [])
    .controller("MiAppController", function( $scope){
        //Esta manera de inyectar el scope no es la mejor
        $scope.peliculas = [];
        $scope.agregarPelicula = function(){
       $scope.peliculas.push($scope.nuevaPelicula);
        }
    });
```

Inyección alternativa

Como alternativa, fiable a la minimización de código, podemos realizar la inyección de este modo:

```
.controller("MiAppController",  ['$scope', function($scope){
    //Esta manera de inyectar el scope es mejor
    $scope.peliculas = [];
    $scope.agregarPelicula = function(){
        $scope.peliculas.push($scope.nuevaPelicula);
    }
}]);
```

2.1

Entendiendo scope

Capas de arquitectura AngularJS

2.2

Directivas diversas

ngClass

Permite alterar la/s clase/s (class de CSS) de un elemento de la página en base a los datos del scope.

- Puedes asignar clases en función de una variable (cadena) o una expresión boleana.

```
<tr ng-class="pelicula.clasificacion" ng-repeat="pelicula in peliculas
track by $index">
```

```
class="{{pelicula.clasificacion | lowercase}}"
```

ngRepeat

Permite filtrar y ordenar, entre otras cosas.

<tr ng-repeat="pelicula in peliculas | orderBy: 'nombre':true track by $index">

3.1

Vistas

Rutas en SPA

Permiten enlaces profundos a pantallas o estados de una página. Historial, botón de atrás, favoritos, etc.

http://example.com/index.php
http://example.com/index.php**#/seccion**
http://example.com/index.php**#/pagina_interna**

```
<nav>
    <a href="#/">Home</a> |
    <a href="#/seccion">Sección</a>
    <a href="#/pagina_interna">Otra página interna</a>
</nav>
```

Instalar sistema de routing

1) Descargar el archivo angular-route.js
2) Enlazarlo en la aplicación con <script>

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

3) Inyectar la dependencia en module()

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

Configurar rutas con $routeProvider

```
.config(function($routeProvider){
    //configuración de las rutas
    $routeProvider
        .when("/", {
            controller: "miAppVistasController",
            templateUrl: "vistas/home.html"
        })
        .when("/seccion", {
            controller: "miAppVistasController",
            templateUrl: "vistas/seccion.html"
        });
})
```

Vistas

Vistas se cargan en el elemento con la directiva ngView

```
<div ng-view></div>
```

Los archivos de las vistas independientes contienen solo el HTML necesario

```
<h2>Home</h2>
<p>Esta es la home</p>
```

4.1

Factorías

Qué son las factorías

Son contenedores de código que puedes usar a lo largo de varios puntos de una aplicación.

- Compartir datos entre varios controladores
- Compartir datos entre varias vistas
- Empaquetar funcionalidades comunes

Crear factorías

Igual que los controladores, las factorías se crean a partir del objeto "module".

```
angular.module().factory("nombreFactoria, function(){
    //codigo de la factoria
})
```

Factorías devuelven su interfaz

Los datos o funcionalidades que ofrecen las factorías, deben ser devueltos en alguna estructura de datos, típicamente un objeto.

```
factory("factoriaPrueba", function(){
    var datoPublico = "Público";
    var datoPrivado = "Privado";
    var metodoPublico = function(){
        console.log("ejecuto...");
    }
    return {
        dato: datoPublico,
        funcion: metodoPublico
    }
})
```

Inyectar factorías

Si deseamos usar una factoría debemos inyectarla al controlador, mediante su nombre.

```
controller("MiAppController", ["factoriaPrueba", function(factoriaPrueba){
    console.log(factoriaPrueba.dato);
    factoriaPrueba.funcion();
}]);
```

5.1

Ajax

Por qué Ajax

Nuestros ejemplos hasta ahora no guardan información. Se pierde al cerrar el navegador…

- Generalmente desearemos que los datos cargados se almacenen en una base de datos para posteriores accesos o para compartirlos con otros usuarios.
- Deseamos que las comunicaciones no recarguen la página (SPA).

API REST

En AngularJS es común que las comunicaciones con el servidor se realicen mediante una interfaz REST.

- El servidor envía la información a Angular en JSON
- Angular se encarga de mostrar esa información en sus vistas y aplica la funcionalidad necesaria para la interacción con el cliente.
- Para memorizar la información la envía al servidor mediante algún verbo del HTTP.

Service $http

Es un servicio que forma parte del "core" de Angular que nos ofrece la funcionalidad de Ajax.

$http
$http.get()
$http.post()
…

Al ser asíncrono, usa el "patrón promise" para ejecutar el código deseado cuando la solicitud recibe respuesta.

Inyectar $http

En los controladores donde deseemos usar Ajax debemos inyectar el service $http.

```
.controller("AppController", ["$http", function($http){

    //estoy listo para usar ajax con $http service

}]);
```

$http.get()

Realiza una solicitud mediante Ajax, usando el verbo "GET" del HTTP. Recibe los parámetros:

- URL (cadena con la URL a la que acceder)
- Config (objeto configuración, opcional)

Devuelve un objeto "promise"

```
$http.get("https://restcountries.eu/rest/v1/all")
```

Promise

El objeto promise acepta el método then() para indicar qué hacer cuando servidor nos devuelva una respuesta.

```
$http.get("https://restcountries.eu/rest/v1/all")
    .then(function(respuesta){
        //codigo a ejecutar con respuesta positiva
    }, function(respuesta){
        //codigo a ejecutar con respuesta negativa
    });
```

La **respuesta** de las funciones del then() es un objeto con datos sobre el response del servidor.

5.2

Crear un API

Crear un API para pruebas

JSON-Server, basado en NodeJS, te permite crear un "Fake API": https://github.com/typicode/json-server

- Instalas NodeJS
- Instalas el paquete json-server con npm

  ```
  $ npm install -g json-server
  ```
- Creas un archivo JSON con un conjunto de datos de prueba

  (En el proyecto con los ejemplos del taller tenemos el JSON para definir películas y clasificaciones)
- Arrancas el servidor del API

  ```
  $ json-server --watch db.json
  ```
- Listo! :)

5.3

Factorías de recursos API

Por qué Ajax

Finalmente, son las factorías en un ejemplo real los que te dan acceso a los datos de API

- Inyecto $http a la factoría
- La factoría encapsula el origen de los datos, ahora un servicio web REST.

Cambios en la factoría

```
.factory("peliculasFactory", ['$http', function($http){
  var getPeliculas = function(){
    return $http.get("http://localhost:3000/peliculas");
  }
  var crearPelicula = function(nuevaPelicula){
     return $http.post("http://localhost:3000/peliculas", nuevaPelicula);
  }
  return {
     getPeliculas: getPeliculas,
     crearPelicula: crearPelicula
  }
}])
```

Cambios en el controlador

Traigo el array de películas que te da la factoría

```
peliculasFactory.getPeliculas().then(function(res){
    $scope.peliculas = res.data;
});
```

Agrego una película

```
peliculasFactory.crearPelicula($scope.nuevaPelicula).then(function(res){
    peliculasFactory.getPeliculas().then(function(res){
        $scope.peliculas = res.data;
    });
});
```