Las 10 principales diferencias entre AngularJS y Angular

Antes de nada debes tener claro que, a pesar de compartir el nombre, **AngularJS y Angular no tienen nada que ver**. El mal llamado "Angular 2", que se llama solamente Angular, no es una nueva versión de AngularJS (también denominado Angular 1.x). **Es un nuevo framework**, escrito desde cero y con conceptos y formas de trabajar completamente distintos.

- Angular utiliza un **sistema de inyección de dependencias jerárquico impulsando su rendimiento de forma brutal**.
- También implementa la **detección de cambios basados en árboles unidireccionales**, que también incrementa el rendimiento.
- Según algunos datos oficiales, Angular puede llegar a ser **5 veces más rápido que AngularJS**.

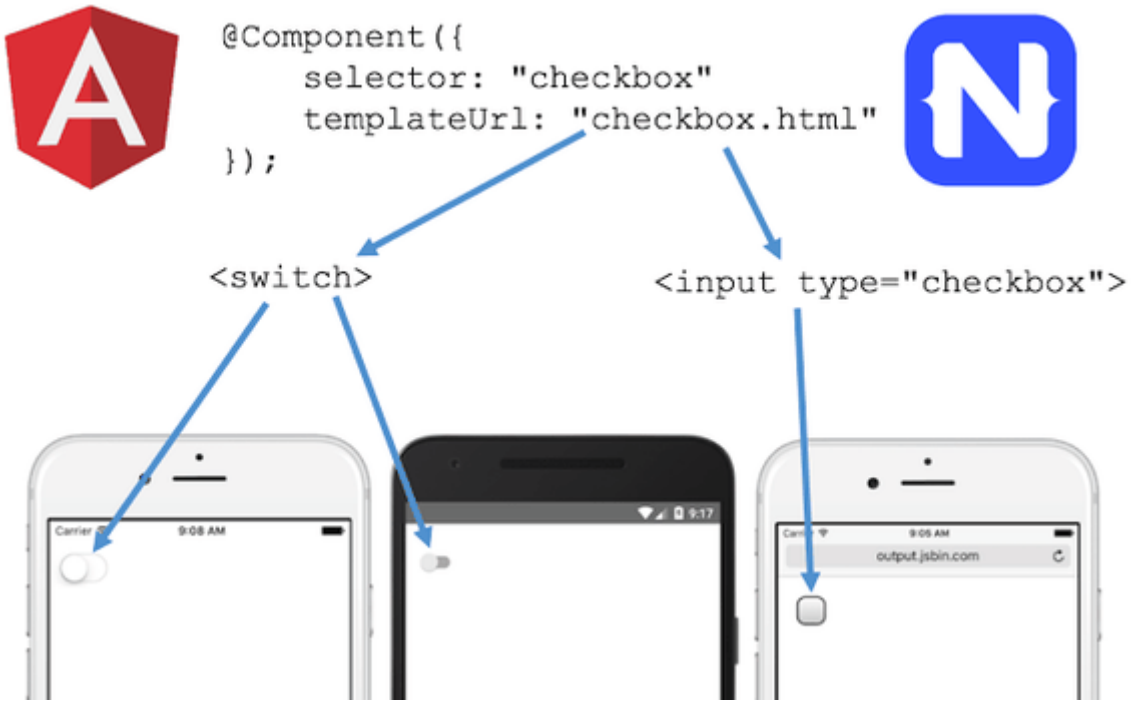
Angular tiene una curva de aprendizaje bastante empinada para los desarrolladores. Está todo escrito en **Typescript** y cumple con la especificación ES6. Como ya apuntábamos al principio, no se puede decir que sea una actualización de Angular 1.x. Se ha reescrito por completo e incluye muchas innovaciones que rompen con su predecesor.

1. Angular está orientado a móviles y tiene mejor rendimiento

Angular 1.x no se hizo para soportar móviles, mientras que Angular está orientado desde el principio a dar buen rendimiento y funcionar bien en dispositivos móviles.

AngularJS se diseñó para crear apps Web de tipo SPA, con enlazado a datos bidireccional. No había soporte para móviles, aunque sí existen otras bibliotecas que hacen que Angular 1.x se ejecute en móviles.

Angular se ha creado teniendo en cuenta una arquitectura orientada a móviles. Hay bibliotecas, por ejemplo **NativeScript**, que ayudan al desarrollo rápido para móviles con Angular. Además, renderiza el mismo código de forma distinta en navegadores web que en apps móviles.



(Gráfico obtenido de la documentación de NativeScript)

2. Angular ofrece más opciones a la hora de elegir lenguajes

Angular ofrece la posibilidad de elegir entre varios lenguajes a la hora de programar. Para escribir código Angular puedes usar cualquiera de los lenguajes: ECMAScript 5 de toda la vida, ES6, TypeScript o incluso Dart (de Google). Mientras que con Angular 1.x puedes usar ES5, ES6 y Dart.

Poder usar **TypeScript** es un gran avance ya que es una forma genial de escribir JavaScript. TypeScript es el lenguaje por defecto para desarrollar en Angular, y el que sin duda vas a querer utilizar. La mayor parte la documentación que encontrarás por ahí estará creada con TypeScript, así que debes aprenderlo.

3. Los controladores y el `$scope` de Angular 1.x han desaparecido

Podemos decir que en Angular los controladores se substituyen por **componentes**. Angular se basa en componentes web, con las ventajas que ello supone al adoptar un estándar de futuro, que cuando esté bien soportado por todos los navegadores ofrecerá más rendimiento todavía.

Por ejemplo:

Controlador Angular 1.x

En JavaScript:

```
var myApp = angular

.module("miModulo", [])

.controller("productoController", function($scope) {

          var prods = { name: "Prod1", quantity: 5 };

          $scope.products = prods;

});
```

Controlador Angular

En TypeScript:

```
import { Component } from '@angular/core';

@Component({

    selector: 'prodsdata',

    template: '<h3>{{prods?.name}}</h3>'

})`

export class ProductoComponent {

    prods = {name: 'Prod1', quantity: 5 };

}
```

Fíjate en cómo, además, hemos usado una interrogación después de `prods` para indicar que el objeto podría ser nulo (es un "anulable"), en cuyo caso no se produciría un error y solo se mostraría el nombre en caso de que el producto tenga algún dato. Ventajas de usar TypeScript.

4. La sintaxis de las directivas estructurales ha cambiado

En Angular, la sintaxis de las directivas estructurales ha cambiado, `ng-repeat` se sustituye por `*ngFor`, por ejemplo.

Directivas estructurales Angular 1.x:

```
<ul>

   <li ng-repeat="prod in productos">

     {{prod.name}}

   </li>

</ul>
```

Directivas estructurales Angular:

```
<ul>

  <li *ngFor="prod of productos">

    {{prod.name}}

  </li>

</ul>
```

El símbolo asterisco (`*`) se usa como prefijo para directivas estructurales, `in` se sustituye por `of` y se usa la sintaxis **camelCase**. Hay muchos más detalles de esta nueva sintaxis, pero de momento quédate con esto.

5. Angular usa directamente las propiedades de los elementos y los eventos estándar del DOM

Uno de los mayores cambios en Angular es, que usa directamente las propiedades de los elementos y los eventos estándar del DOM.

Por ello, muchas de las directivas integradas disponibles en Angular 1.x ya no se necesitan, como por ejemplo: `ng-href`, `ng-src`, `ng-show` o `ng-hide`.

Angular usa directamente `href`, `src` y propiedades `hidden`para obtener el mismo resultado.

Y lo mismo se puede decir de las directivas basadas en eventos como `ng-click` o `ng-blur`.

En AngularJS:

```
<button ng-click="doSomething()">
```

En Angular simplemente tomas el evento estándar y lo envuelves entre paréntesis:

```
<button (click)="doSomething()">
```

Nuevamente aquí hay muchos otros detalles a tener en cuenta, pero quédate con esta idea principal.

6. La directiva de datos enlazados en una sola dirección (one-way data binding) se sustituye por `[property]`

En Angular 1.x, `ng-bind` se usa para enlazar datos en una sola dirección (*one-way data binding*), lo que quiere decir que sólo se modifica el enlace desde el código hacia la vista, pero no al revés, lo que permite un mayor control de flujo dentro de la aplicación.

Con Angular esto se reemplaza por `[property]`, siendo "property" una propiedad válida del elemento del DOM sobre el que actuamos.

Por ejemplo, en Angular 1.x escribíamos:

```
<input ng-bind="prod.name"></input>
```

En Angular se utilizan simplemente corchetes sobre la propiedad estándar:

```
<input [value]="prod.name"></input>
```

Aunque existen otras variantes para lograr lo mismo.

7. Enlaces de datos de doble dirección (two-way data binding): `ng-model` se sustituye por `[(ngModel)]`

Este es el enlazado que todo el mundo conoce y usa en AngularJS. En Angular se retira esta sintaxis para lograr mayor seguridad, control y mejora del rendimiento.

En Angular 1.x hacíamos esto para tener un enlazado en dos direcciones:

```
<input ng-model="prod.name"></input>
```

En Angular la sintaxis equivalente sería:

```
<input [(ngModel)]="prod.name"></input>
```

Este *doble-binding* ofrece ventajas relevantes en la gestión avanzada de formularios.

8. Ha cambiado la sintaxis de la inyección de dependencias

Una de las grandes ventajas de Angular es la inyección de dependencias. Con Angular hay una manera distinta de llevar a cabo esto. Como en Angular todo son "clases", la inyección de dependencias se consigue mediante constructores.

En Angular 1.x:

```
var myApp = angular

   .module("miModulo", [])

   .controller("productoController", function($scope, $http) {

        var prods = { name: "Queso", quantity: 5 };

        $scope.productos = prods;

    });
```

En Angular:

```
import { Injectable } from '@angular/core';
```

```
import { Http } from '@angular/http';

import { Producto } from 'app/shared/Producto.ts';



@Injectable()

export class ProductosService {

    constructor(private _http: Http) { }

    getProductos() {

        return [new Producto(1, 'Queso'),

            new Producto(2, 'Pan',

            new Producto(3, 'Verdura')

        ];

    }

}
```

`@Injectable()` es la anotación de Angular que permite definir clases como servicios, importándolos a posteriori dentro de los módulos de Angular. Esto tiene enormes beneficios en lo que respecta al rendimiento, ya que es el propio Angular quien maneja la carga de entidades a través del *bootstrapping*.

9. Ha cambiado la sintaxis para hacer *routing*

En Angular 1.x, usamos `$routeProvider.when()` para configurar el enrutamiento. Sin embargo, en Angular se usa `@RouteConfig{(...})`. El `ng-view` que está presente en Angular 1.x se susitituye por `<router-outlet>`.

En Angular 1.x y JavaScript:

```
var app = angular

        .module("miModulo", ["ngRoute"])

        .config(function ($routeProvider) {
```

```
            $routeProvider

            .when("/inicio", { templateUrl: "inicio.html", controller:
"inicioController" })

            .when("/Productos",    {    templateUrl:    "productos.html",
controller: "productosController" })

        })

       .controller("inicioController", function ($scope) {

            $scope.message = "Página de inicio";

        })

       .controller("productosController", function ($scope) {

             $scope.prods = ["Queso", "Pan", "Verdura"];

       })
```

En Angular y TypeScript:

```
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { InicioComponent } from './inicio/inicio.component';

import { ProductoComponent } from './productos/producto.component';

const appRoutes: Routes = [

    {

        path: 'inicio',

        component: InicioComponent,

    },

    {

        path: 'productos',
```

```
        component: ProductoComponent,

    },

];

@NgModule({

    imports: [

        RouterModule.forRoot(appRoutes)

    ]

})

export class AppRoutingModule { }
```

Y en la página HTML:

```
<ul>

  <li><a routerLink="/inicio">Página de inicio</a></li>

  <li><a routerLink="/productos">Productos</a></li>

</ul>
```

10. La forma de arrancar una Aplicación Angular ha cambiado:

Angular 1.x tiene dos formas de arrancar. Una usando el atributo `ng-app` y la otra vía código:

```
<script>

   angular.element(document).ready(function() {

       angular.bootstrap(document, ['myApp']);

   });

</script>
```

En Angular la única forma de arrancar es vía código, en TypeScript:

```
import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';

 import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

En resumen

Como vemos, aunque siempre hay cosas en común, **AngularJS y Angular son muy diferentes** tanto en lo que se refiere a la forma de trabajar como a los conceptos que subyacen a ambas tecnologías.

Por eso **no se puede hablar de una nueva versión**, sino de un nuevo *framework*, que es necesario aprender prácticamente desde cero.