EBook Gratis

APRENDIZAJE AngularJS

Free unaffiliated eBook created from **Stack Overflow contributors.**

#angularjs

Tabla de contenido

Acerca de 1

Capítulo 1: Empezando con AngularJS 2

Observaciones 2

Versiones 2

Examples 9

Empezando 9

Mostrar todas las construcciones angulares comunes 11

La importancia del alcance. 12

El mundo angular más simple posible. 14

ng-app 14

Directivas 14

Minificación en Angular 15

Tutoriales de vídeo de inicio de AngularJS 16

Capítulo 2: Alcances $ angulares 19

Observaciones 19

Examples 19

Ejemplo básico de la herencia $ scope 19

Evitar heredar valores primitivos. 19

Una función disponible en toda la aplicación. 20

Creando eventos personalizados de $ scope 21

Usando las funciones de $ scope 22

¿Cómo puede limitar el alcance de una directiva y por qué haría esto? 23

Capítulo 3: Almacenamiento de sesión 25

Examples 25

Manejo de la sesión de almacenamiento a través del servicio usando angularjs. 25

Servicio de almacenamiento de sesión: 25

En el controlador: 25

Capítulo 4: angularjs con filtro de datos, paginación etc 27

Introducción 27

Examples 27

Angularjs muestra datos con filtro, paginación 27
Capítulo 5: AngularJS gotchas y trampas 28
Examples 28
El enlace de datos bidireccional deja de funcionar 28
Ejemplo 28
Cosas que hacer cuando se utiliza html5Mode 29
7 pecados mortales de AngularJS 30
Capítulo 6: Carga lenta 35
Observaciones 35
Examples 35
Preparando tu proyecto para carga lenta 35
Uso 35
Uso con enrutador 36
UI-Router: 36
ngRuta: 36
Uso de inyección de dependencia 36
Utilizando la directiva 37
Capítulo 7: Cómo funciona el enlace de datos 38
Observaciones 38
Examples 38
Ejemplo de enlace de datos 38
Capítulo 8: Compartir datos 41
Observaciones 41
Examples 41
Usando ngStorage para compartir datos 41
Compartir datos de un controlador a otro usando el servicio 42
Capítulo 9: Componentes 43
Parámetros 43
Observaciones 44
Examples 44
Componentes básicos y ganchos de ciclo de vida 44

¿Qué es un componente? 44
Usando datos externos en el componente: 44
Uso de controladores en componentes 45
Usando "require" como un objeto 46
Componentes en angular JS 46
Capítulo 10: Constantes 48
Observaciones 48
Examples 48
Crea tu primera constante 48
Casos de uso 48
Capítulo 11: Controladores 51
Sintaxis 51
Examples 51
Su primer controlador 51
Creando Controladores 53
Creando controladores, minificación segura 53
El orden de las dependencias inyectadas es importante. 53
Usando ControllerAs en Angular JS 54
Creación de controladores angulares seguros para minificación 55
Controladores anidados 56
Capítulo 12: Controladores con ES6 57
Examples 57
Controlador 57
Capítulo 13: Decoradores 58
Sintaxis 58
Observaciones 58
Examples 58
Servicio de decoración, fábrica. 58
Decorar directiva 59
Decorar filtro 60
Capítulo 14: Depuración 61

Examples....61
Depuración básica en el marcado....61
Usando ng-inspect chrome extension....62
Consiguiendo el alcance del elemento....65
Capítulo 15: directiva de clase ng....67
Examples....67
Tres tipos de expresiones de clase ng....67
1. cuerda....67
2. Objeto....67
3. Array....68
Capítulo 16: Directivas incorporadas....69
Examples....69
Expresiones angulares - Texto vs. Número....69
ngRepetir....69
ngShow y ngOcultar....73
ngOpciones....74
ngModel....76
clase ng....77
ngIf....77
JavaScript....78
Ver....78
DOM si currentUser no está indefinido....78
DOM Si currentUser está currentUser....78
Promesa de la función....79
ngMouseenter y ngMouseleave....79
deshabilitado....80
ngDblclick....80
Hoja de trucos de directivas incorporadas....80
Haga clic en....82
ngRequisito....83
ng-model-options....83

capa ng 84
ngIncluir 84
ngSrc 85
ngPatrón 85
ngValue 86
ngCopy 86
Evitar que un usuario copie datos 86
ngPaste 86
ngHref 87
ngList 87
Capítulo 17: Directivas personalizadas 89
Introducción 89
Parámetros 89
Examples 91
Crear y consumir directivas personalizadas. 91
Directiva de plantilla de objeto de definición 92
Ejemplo de directiva básica 93
Cómo crear un componente resuable usando una directiva. 94
Directiva básica con plantilla y un alcance aislado. 96
Construyendo un componente reutilizable. 97
Directivo decorador 98
Directiva de herencia e interoperabilidad. 99
Capítulo 18: Directivas utilizando ngModelController 101
Examples 101
Un control simple: calificación 101
Un par de controles complejos: editar un objeto completo 103
Capítulo 19: El yo o esta variable en un controlador 107
Introducción 107
Examples 107
Entender el propósito de la variable del uno mismo 107
Capítulo 20: enrutador ui 109

Observaciones 109

Examples 109

Ejemplo básico 109

Vistas múltiples 110

Usando funciones de resolver para cargar datos 112

Vistas anidadas / Estados 113

Capítulo 21: Enrutamiento usando ngRoute 115

Observaciones 115

Examples 115

Ejemplo basico 115

Ejemplo de parámetros de ruta 116

Definición de comportamiento personalizado para rutas individuales. 118

Capítulo 22: estilo ng 120

Introducción 120

Sintaxis 120

Examples 120

Uso de estilo ng 120

Capítulo 23: Eventos 121

Parámetros 121

Examples 121

Utilizando sistema de eventos angulares. 121

$ scope. $ emit 121

$ scope. $ broadcast 121

Sintaxis: 122

Evento limpio registrado en AngularJS 122

Usos y significado 123

Siempre cancele el registro de $ rootScope. $ En los escuchas en el evento scope $ destory 125

Capítulo 24: Filtros 126

Examples 126

Su primer filtro 126

Javascript 126

HTML 127

Filtro personalizado para eliminar valores. 127

Filtro personalizado para dar formato a los valores. 127

Realizando filtro en una matriz hijo 128

Usando filtros en un controlador o servicio 129

Accediendo a una lista filtrada desde fuera de una repetición ng 130

Capítulo 25: Filtros personalizados 131

Examples 131

Ejemplo de filtro simple 131

example.js 131

example.html 131

Rendimiento esperado 131

Utilice un filtro en un controlador, un servicio o un filtro 131

Crear un filtro con parámetros. 132

Capítulo 26: Filtros personalizados con ES6 133

Examples 133

Filtro de tamaño de archivo usando ES6 133

Capítulo 27: Funciones auxiliares incorporadas 135

Examples 135

angular.equals 135

angular.isString 135

angular.isArray 136

angular.merge 136

angular.isDefinido y angular.isUndefined 136

angular.isfecha 137

angular.isNumber 137

angular.isFunción 138

angular.toJson 138

angular.fromJson 139

angular.noop 139

angular.isObjeto 140

angular.iselemento 140

angular.copy 140
identidad angular 141
angular.para cada 141
Capítulo 28: Impresión 143
Observaciones 143
Examples 143
Servicio de impresión 143
Capítulo 29: Interceptor HTTP 145
Introducción 145
Examples 145
Empezando 145
Generador interactivo de HTTP paso a paso 145
Mensaje flash en respuesta utilizando el interceptor http 146
En el archivo de vista 146
Archivo de comandos 147
Errores comunes 147
Capítulo 30: Inyección de dependencia 149
Sintaxis 149
Observaciones 149
Examples 149
Inyecciones 149
Inyecciones dinamicas 150
$ inyectar propiedad anotación 150
Cargar dinámicamente el servicio AngularJS en vainilla JavaScript 150
Capítulo 31: Migración a Angular 2+ 152
Introducción 152
Examples 152
Convertir su aplicación AngularJS en una estructura orientada a componentes 152
Comienza a dividir tu aplicación en componentes 152
¿Qué pasa con los controladores y las rutas? 154
¿Que sigue? 154

Conclusión 154
Introducción a los módulos Webpack y ES6. 155
Capítulo 32: Módulos 156
Examples 156
Módulos 156
Módulos 156
Capítulo 33: MVC angular 158
Introducción 158
Examples 158
La vista estática con controlador 158
mvc demo 158
Definición de la función del controlador 158
Añadiendo información al modelo. 158
Capítulo 34: ng-repetir 159
Introducción 159
Sintaxis 159
Parámetros 159
Observaciones 159
Examples 159
Iterando sobre las propiedades del objeto. 159
Seguimiento y duplicados 160
ng-repeat-start + ng-repeat-end 160
Capítulo 35: ng-view 162
Introducción 162
Examples 162
ng-view 162
Registro de navegación 162
Capítulo 36: Opciones de enlaces AngularJS (`=`, `@`, `&` etc.) 164
Observaciones 164
Examples 164
@ enlace unidireccional, atributo de enlace. 164

= enlace bidireccional....164

y función de enlace, expresión de enlace....165

Encuadernación disponible a través de una simple muestra....165

Vincular atributo opcional....166

Capítulo 37: Perfil de rendimiento....167

Examples....167

Todo sobre perfilado....167

Capítulo 38: Perfilado y Rendimiento....169

Examples....169

7 mejoras simples de rendimiento....169

1) Utilice ng-repetir con moderación....169

2) Atar una vez....169

3) Las funciones de alcance y los filtros toman tiempo....170

4 observadores....171

5) ng-if / ng-show....172

6) Deshabilitar la depuración....172

7) Usa la inyección de dependencia para exponer tus recursos....172

Atar una vez....173

Funciones de alcance y filtros....174

Vigilantes....174

Entonces, ¿qué es el observador?....175

ng-if vs ng-show....176

ng-si....176

ng-show....177

Ejemplo....177

Conclusión....177

Rebota tu modelo....177

Siempre anular el registro de los oyentes registrados en otros ámbitos distintos del alcan....178

Capítulo 39: Prepararse para la producción - Grunt....179

Examples....179

Ver precarga....179

Optimización de scripts....180
Capítulo 40: Promesas angulares con servicio $ q....183
Examples....183
Usando $ q.all para manejar múltiples promesas....183
Usando el constructor $ q para crear promesas....184
Operaciones diferidas usando $ q.defer....185
Usando promesas angulares con servicio $ q....185
Usando promesas de guardia....186
Propiedades....186
Envuelva el valor simple en una promesa usando $ q.when ()....188
$ q.when y su alias $ q.resolve....188
Evita el $ q diferido anti-patrón....188
Evita este Anti-Patrón....189
Capítulo 41: Proveedores....190
Sintaxis....190
Observaciones....190
Examples....190
Constante....190
Valor....191
Fábrica....191
Servicio....192
Proveedor....192
Capítulo 42: Proyecto Angular - Estructura de Directorio....194
Examples....194
Estructura de directorios....194
Ordenar por tipo (izquierda)....194
Ordenar por característica (derecha)....195
Capítulo 43: Pruebas unitarias....197
Observaciones....197
Examples....197
Unidad de prueba de un filtro....197
Prueba unitaria de un componente (1.5+)....198

Unidad de prueba de un controlador. 199

Unidad de prueba de un servicio. 199

Unidad de prueba una directiva 200

Capítulo 44: recorrido del bucle de digestión 202

Sintaxis 202

Examples 202

enlace de datos de dos vías 202

$ digerir y $ ver 202

el arbol $ scope 203

Capítulo 45: Servicio Distinguido vs Fábrica 206

Examples 206

Factory VS Service una vez por todas 206

Capítulo 46: Servicios 208

Examples 208

Cómo crear un servicio 208

Cómo utilizar un servicio 208

Creando un servicio usando angular.factory 209

$ sce - desinfecta y representa contenido y recursos en plantillas 209

Cómo crear un servicio con dependencias usando 'sintaxis de matriz' 210

Registro de un servicio 210

Diferencia entre Servicio y Fábrica. 211

Capítulo 47: SignalR con AngularJs 215

Introducción 215

Examples 215

SignalR y AngularJs [ChatProject] 215

Capítulo 48: solicitud de $ http 219

Examples 219

Usando $ http dentro de un controlador 219

Usando la solicitud $ http en un servicio 220

Tiempo de una solicitud $ http 221

Capítulo 49: Tareas roncas 223

Examples 223

Ejecutar la aplicación localmente 223
Capítulo 50: Usando AngularJS con TypeScript 226
Sintaxis 226
Examples 226
Controladores angulares en mecanografiado 226
Uso del controlador con la sintaxis de ControllerAs 227
Usando Bundling / Minification 228
¿Por qué ControllerAs Syntax? 229
Función del controlador 229
¿Por qué ControllerAs? 229
¿Por qué $ alcance? 230
Capítulo 51: Uso de directivas incorporadas. 231
Examples 231
Ocultar / Mostrar elementos HTML 231
Capítulo 52: Validación de formularios 233
Examples 233
Validación básica de formularios 233
Estados de forma y entrada 234
Clases de CSS 234
ngMessages 235
Enfoque tradicional 235
Ejemplo 235
Validación de formularios personalizados 236
Formularios anidados 236
Validadores asíncronos 237
Creditos 238

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angularjs

It is an unofficial and free AngularJS ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official AngularJS.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con AngularJS

Observaciones

AngularJS es un marco de aplicación web diseñado para simplificar el desarrollo de aplicaciones del lado del cliente enriquecido. Esta documentación es para Angular 1.x , el antecesor de la versión más moderna de Angular 2 o consulte la documentación de Stack Overflow para Angular 2 .

Versiones

Versión	Fecha de lanzamiento
1.6.5	2017-07-03
1.6.4	2017-03-31
1.6.3	2017-03-08
1.6.2	2017-02-07
1.5.11	2017-01-13
1.6.1	2016-12-23
1.5.10	2016-12-15
1.6.0	2016-12-08
1.6.0-rc.2	2016-11-24
1.5.9	2016-11-24
1.6.0-rc.1	2016-11-21
1.6.0-rc.0	2016-10-26
1.2.32	2016-10-11
1.4.13	2016-10-10
1.2.31	2016-10-10
1.5.8	2016-07-22
1.2.30	2016-07-21
1.5.7	2016-06-15

Versión	Fecha de lanzamiento
1.4.12	2016-06-15
1.5.6	2016-05-27
1.4.11	2016-05-27
1.5.5	2016-04-18
1.5.4	2016-04-14
1.5.3	2016-03-25
1.5.2	2016-03-19
1.4.10	2016-03-16
1.5.1	2016-03-16
1.5.0	2016-02-05
1.5.0-rc.2	2016-01-28
1.4.9	2016-01-21
1.5.0-rc.1	2016-01-16
1.5.0-rc.0	2015-12-09
1.4.8	2015-11-20
1.5.0-beta.2	2015-11-18
1.4.7	2015-09-30
1.3.20	2015-09-30
1.2.29	2015-09-30
1.5.0-beta.1	2015-09-30
1.5.0-beta.0	2015-09-17
1.4.6	2015-09-17
1.3.19	2015-09-17
1.4.5	2015-08-28
1.3.18	2015-08-19

Versión	Fecha de lanzamiento
1.4.4	2015-08-13
1.4.3	2015-07-15
1.3.17	2015-07-07
1.4.2	2015-07-07
1.4.1	2015-06-16
1.3.16	2015-06-06
1.4.0	2015-05-27
1.4.0-rc.2	2015-05-12
1.4.0-rc.1	2015-04-24
1.4.0-rc.0	2015-04-10
1.3.15	2015-03-17
1.4.0-beta.6	2015-03-17
1.4.0-beta.5	2015-02-24
1.3.14	2015-02-24
1.4.0-beta.4	2015-02-09
1.3.13	2015-02-09
1.3.12	2015-02-03
1.4.0-beta.3	2015-02-03
1.3.11	2015-01-27
1.4.0-beta.2	2015-01-27
1.4.0-beta.1	2015-01-20
1.3.10	2015-01-20
1.3.9	2015-01-15
1.4.0-beta.0	2015-01-14
1.3.8	2014-12-19

Versión	Fecha de lanzamiento
1.2.28	2014-12-16
1.3.7	2014-12-15
1.3.6	2014-12-09
1.3.5	2014-12-02
1.3.4	2014-11-25
1.2.27	2014-11-21
1.3.3	2014-11-18
1.3.2	2014-11-07
1.3.1	2014-10-31
1.3.0	2014-10-14
1.3.0-rc.5	2014-10-09
1.2.26	2014-10-03
1.3.0-rc.4	2014-10-02
1.3.0-rc.3	2014-09-24
1.2.25	2014-09-17
1.3.0-rc.2	2014-09-17
1.2.24	2014-09-10
1.3.0-rc.1	2014-09-10
1.3.0-rc.0	2014-08-30
1.2.23	2014-08-23
1.3.0-beta.19	2014-08-23
1.2.22	2014-08-12
1.3.0-beta.18	2014-08-12
1.2.21	2014-07-25
1.3.0-beta.17	2014-07-25

Versión	Fecha de lanzamiento
1.3.0-beta.16	2014-07-18
1.2.20	2014-07-11
1.3.0-beta.15	2014-07-11
1.2.19	2014-07-01
1.3.0-beta.14	2014-07-01
1.3.0-beta.13	2014-06-16
1.3.0-beta.12	2014-06-14
1.2.18	2014-06-14
1.3.0-beta.11	2014-06-06
1.2.17	2014-06-06
1.3.0-beta.10	2014-05-24
1.3.0-beta.9	2014-05-17
1.3.0-beta.8	2014-05-09
1.3.0-beta.7	2014-04-26
1.3.0-beta.6	2014-04-22
1.2.16	2014-04-04
1.3.0-beta.5	2014-04-04
1.3.0-beta.4	2014-03-28
1.2.15	2014-03-22
1.3.0-beta.3	2014-03-21
1.3.0-beta.2	2014-03-15
1.3.0-beta.1	2014-03-08
1.2.14	2014-03-01
1.2.13	2014-02-15
1.2.12	2014-02-08

Versión	Fecha de lanzamiento
1.2.11	2014-02-03
1.2.10	2014-01-25
1.2.9	2014-01-15
1.2.8	2014-01-10
1.2.7	2014-01-03
1.2.6	2013-12-20
1.2.5	2013-12-13
1.2.4	2013-12-06
1.2.3	2013-11-27
1.2.2	2013-11-22
1.2.1	2013-11-15
1.2.0	2013-11-08
1.2.0-rc.3	2013-10-14
1.2.0-rc.2	2013-09-04
1.0.8	2013-08-22
1.2.0rc1	2013-08-13
1.0.7	2013-05-22
1.1.5	2013-05-22
1.0.6	2013-04-04
1.1.4	2013-04-04
1.0.5	2013-02-20
1.1.3	2013-02-20
1.0.4	2013-01-23
1.1.2	2013-01-23
1.1.1	2012-11-27

Versión	Fecha de lanzamiento
1.0.3	2012-11-27
1.1.0	2012-09-04
1.0.2	2012-09-04
1.0.1	2012-06-25
1.0.0	2012-06-14
v1.0.0rc12	2012-06-12
v1.0.0rc11	2012-06-11
v1.0.0rc10	2012-05-24
v1.0.0rc9	2012-05-15
v1.0.0rc8	2012-05-07
v1.0.0rc7	2012-05-01
v1.0.0rc6	2012-04-21
v1.0.0rc5	2012-04-12
v1.0.0rc4	2012-04-05
v1.0.0rc3	2012-03-30
v1.0.0rc2	2012-03-21
g3-v1.0.0rc1	2012-03-14
g3-v1.0.0-rc2	2012-03-16
1.0.0rc1	2012-03-14
0.10.6	2012-01-17
0.10.5	2011-11-08
0.10.4	2011-10-23
0.10.3	2011-10-14
0.10.2	2011-10-08
0.10.1	2011-09-09

Versión	Fecha de lanzamiento
0.10.0	2011-09-02
0.9.19	2011-08-21
0.9.18	2011-07-30
0.9.17	2011-06-30
0.9.16	2011-06-08
0.9.15	2011-04-12
0.9.14	2011-04-01
0.9.13	2011-03-14
0.9.12	2011-03-04
0.9.11	2011-02-09
0.9.10	2011-01-27
0.9.9	2011-01-14
0.9.7	2010-12-11
0.9.6	2010-12-07
0.9.5	2010-11-25
0.9.4	2010-11-19
0.9.3	2010-11-11
0.9.2	2010-11-03
0.9.1	2010-10-27
0.9.0	2010-10-21

Examples

Empezando

Crea un nuevo archivo HTML y pega el siguiente contenido:

```
<!DOCTYPE html>
<html ng-app>
```

```
<head>
  <title>Hello, Angular</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
</head>
<body ng-init="name='World'">
  <label>Name</label>
  <input ng-model="name" />
  <span>Hello, {{ name }}!</span>
  <p ng-bind="name"></p>
</body>
</html>
```

Demo en vivo

Cuando abra el archivo con un navegador, verá un campo de entrada seguido del texto `Hello, World!` . La edición del valor en la entrada actualizará el texto en tiempo real, sin la necesidad de actualizar toda la página.

Explicación:

1. Cargue el marco angular desde una red de entrega de contenido.

   ```
   <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
   ```

2. Defina el documento HTML como una aplicación Angular con la directiva `ng-app`

   ```
   <html ng-app>
   ```

3. Inicialice la variable de `name` usando `ng-init`

   ```
   <body ng-init=" name = 'World' ">
   ```

   Tenga en cuenta que ng-init debe utilizarse únicamente con fines demostrativos y de prueba. Al crear una aplicación real, los controladores deben inicializar los datos.

4. Enlazar datos del modelo a la vista en controles HTML. Enlazar una `<input>` a la propiedad de `name` con `ng-model`

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

5. Mostrar contenido del modelo usando llaves dobles `{{ }}`

   ```
   <span>Hello, {{ name }}</span>
   ```

6. Otra forma de enlazar la propiedad de `name` es usar `ng-bind` lugar de manillares `"{{ }}"`

   ```
    <span ng-bind="name"></span>
   ```

Los últimos tres pasos establecen el *enlace de datos de dos vías* . Los cambios realizados en la

entrada actualizan el *modelo* , que se refleja en la *vista* .

Hay una diferencia entre usar manillares y `ng-bind` . Si usa manillares, puede ver el `Hello, {{name}}` real `Hello, {{name}}` medida que se carga la página antes de que se resuelva la expresión (antes de que se carguen los datos), mientras que si usa `ng-bind` , solo mostrará los datos cuando el nombre esta resuelto. Como alternativa, se puede utilizar la directiva `ng-cloak` para evitar que se muestren los manillares antes de compilarlos.

Mostrar todas las construcciones angulares comunes

El siguiente ejemplo muestra construcciones AngularJS comunes en un archivo:

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <style>.started { background: gold; }</style>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>
      function MyDataService() {
        return {
          getWorlds: function getWorlds() {
            return ["this world", "another world"];
          }
        };
      }

      function DemoController(worldsService) {
        var vm = this;
        vm.messages = worldsService.getWorlds().map(function(w) {
          return "Hello, " + w + "!";
        });
      }

      function startup($rootScope, $window) {
        $window.alert("Hello, user! Loading worlds...");
        $rootScope.hasStarted = true;
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .service("worldsService", [MyDataService])
        .controller("demoController", ["worldsService", DemoController])
        .config(function() {
          console.log('configuring application');
        })
        .run(["$rootScope", "$window", startup]);
    </script>
  </head>
  <body ng-class="{ 'started': hasStarted }" ng-cloak>
    <div ng-controller="demoController as vm">
      <ul>
        <li ng-repeat="msg in vm.messages">{{ msg }}</li>
      </ul>
    </div>
  </body>
</html>
```

A continuación se explica cada línea del archivo:

Demo en vivo

1. `ng-app="myDemoApp"` , la directiva ngApp que inicia la aplicación y le dice a Angular que un elemento DOM está controlado por un `angular.module` específico llamado `"myDemoApp"` ;
2. `<script src="angular.min.js">` es el primer paso para arrancar la biblioteca AngularJS ;

Se `DemoController` tres funciones (`MyDataService` , `DemoController` y `startup`), que se utilizan (y explican) a continuación.

3. `angular.module(...)` utilizado con una matriz como segundo argumento crea un nuevo módulo. Esta matriz se utiliza para proporcionar una lista de dependencias de módulo. En este ejemplo, encadenamos llamadas sobre el resultado de la función del `module(...)` ;

4. `.service(...)` crea un servicio angular y devuelve el módulo para el encadenamiento;

5. `.controller(...)` crea un controlador angular y devuelve el módulo para el encadenamiento;

6. `.config(...)` Utilice este método para registrar el trabajo que debe realizarse en la carga del módulo.

7. `.run(...)` se asegura de que el código se ejecute en el momento del inicio y tome una matriz de elementos como parámetro. Utilice este método para registrar el trabajo que se debe realizar cuando el inyector haya terminado de cargar todos los módulos.

   - el primer elemento es hacerle saber a Angular que la función de `startup` requiere que el servicio `$rootScope` se inyecte como un argumento;
   - el segundo elemento es permitir que Angular sepa que la función de `startup` requiere que el servicio integrado de `$window` se inyecte como un argumento;
   - el *último* elemento de la matriz, `startup` , es la función real para ejecutar en el inicio;

8. `ng-class` es la directiva ngClass para establecer una `class` dinámica, y en este ejemplo utiliza `hasStarted` en `$rootScope` dinámicamente

9. `ng-cloak` es una directiva para evitar que la plantilla html de Angular no procesada (por ejemplo, " `{{ msg }}` ") se muestre brevemente antes de que Angular haya cargado completamente la aplicación.

10. `ng-controller` es la directiva que le pide a Angular que cree una instancia de un nuevo controlador con un nombre específico para orquestar esa parte del DOM;

11. `ng-repeat` es la directiva para hacer una iteración angular sobre una colección y clonar una plantilla DOM para cada elemento;

12. `{{ msg }}` muestra la interpolación : la representación en el lugar de una parte del alcance o controlador;

La importancia del alcance.

Como Angular usa HTML para extender una página web y Javascript simple para agregar lógica, facilita la creación de una página web usando **ng-app** , **ng-controller** y algunas directivas

integradas como **ng-if** , **ng-repeat** , etc. Con la nueva sintaxis del **controlador** , los usuarios nuevos de Angular pueden adjuntar funciones y datos a su controlador en lugar de usar `$scope` .

Sin embargo, tarde o temprano, es importante entender qué es exactamente este asunto del `$scope` . Seguirá apareciendo en ejemplos, por lo que es importante tener un poco de comprensión.

La buena noticia es que es un concepto simple pero poderoso.

Cuando creas lo siguiente:

```
<div ng-app="myApp">
 <h1>Hello {{ name }}</h1>
</div>
```

¿Dónde vive el **nombre** ?

La respuesta es que Angular crea un objeto `$rootScope` . Esto es simplemente un objeto Javascript normal y, por lo tanto, **name** es una propiedad en el objeto `$rootScope` :

```
angular.module("myApp", [])
  .run(function($rootScope) {
    $rootScope.name = "World!";
  });
```

Y al igual que con el alcance global en Javascript, generalmente no es una buena idea agregar elementos al alcance global o `$rootScope` .

Por supuesto, la mayoría de las veces, creamos un controlador y ponemos nuestra funcionalidad requerida en ese controlador. Pero cuando creamos un controlador, Angular hace magia y crea un objeto `$scope` para ese controlador. Esto se refiere a veces como el **alcance local** .

Entonces, creando el siguiente controlador:

```
<div ng-app="myApp">
  <div ng-controller="MyController">
    <h1>Hello {{ name }}</h1>
  </div>
</div>
```

permitiría que se pueda acceder al ámbito local a través del parámetro `$scope` .

```
angular.module("myApp", [])
  .controller("MyController", function($scope) {
    $scope.name = "Mr Local!";
  });
```

Un controlador sin un parámetro de `$scope` puede simplemente no necesitarlo por alguna razón. Pero es importante darse cuenta de que, **incluso con la sintaxis de controllerAs** , el alcance local existe.

Dado que `$scope` es un objeto de JavaScript, Angular lo configura mágicamente para heredarlo prototípicamente de `$rootScope` . Y como puedes imaginar, puede haber una cadena de ámbitos. Por ejemplo, podría crear un modelo en un controlador principal y adjuntarlo al alcance del controlador principal como `$scope.model` .

Luego, a través de la cadena de prototipos, un controlador secundario podría acceder al mismo modelo localmente con `$scope.model` .

Nada de esto es inicialmente evidente, ya que solo es Angular haciendo su magia de fondo. Pero comprender el `$scope` es un paso importante para conocer cómo funciona Angular.

El mundo angular más simple posible.

Angular 1 es en el fondo un compilador DOM. Podemos pasarlo en HTML, ya sea como una plantilla o simplemente como una página web normal, y luego hacer que compile una aplicación.

Podemos decirle a Angular que trate una región de la página como una *expresión* usando la sintaxis de estilo de manillares `{{ }}` . Cualquier cosa entre las llaves se compilará, así:

```
{{ 'Hello' + 'World' }}
```

Esto dará como resultado:

```
HelloWorld
```

ng-app

Le decimos a Angular qué parte de nuestro DOM debe tratar como la plantilla maestra usando la *directiva* `ng-app` . Una directiva es un atributo o elemento personalizado con el que el compilador de plantillas angulares sabe cómo tratar. Agreguemos una directiva ng-app ahora:

```
<html>
  <head>
    <script src="/angular.js"></script>
  </head>
  <body ng-app>
    {{ 'Hello' + 'World' }}
  </body>
</html>
```

Ahora le he dicho al elemento del cuerpo que sea la plantilla raíz. Cualquier cosa en él será compilada.

Directivas

Las directivas son directivas de compilación. Extienden las capacidades del compilador Angular DOM. Por eso **Misko** , el creador de Angular, describe a Angular como:

"Lo que habría sido un navegador web si se hubiera construido para aplicaciones web.

Literalmente creamos nuevos atributos y elementos HTML, y hacemos que Angular los compile en una aplicación. `ng-app` es una directiva que simplemente enciende el compilador. Otras directivas incluyen:

- `ng-click` , que agrega un controlador de clic,
- `ng-hide` , que oculta condicionalmente un elemento, y
- `<form>` , que agrega comportamiento adicional a un elemento de formulario HTML estándar.

Angular viene con alrededor de 100 directivas integradas que le permiten realizar las tareas más comunes. También podemos escribir el nuestro, y estos serán tratados de la misma manera que las directivas integradas.

Construimos una aplicación Angular a partir de una serie de directivas, conectadas con HTML.

Minificación en Angular

¿Qué es la minificación?

Es el proceso de eliminar todos los caracteres innecesarios del código fuente sin cambiar su funcionalidad.

Sintaxis normal

Si usamos la sintaxis angular normal para escribir un controlador, luego de minimizar nuestros archivos, se romperá nuestra funcionalidad.

Controlador (antes de la minificación):

```
var app = angular.module('mainApp', []);
app.controller('FirstController', function($scope) {
    $scope.name= 'Hello World !';
});
```

Después de usar la herramienta de minificación, se minimizará como se muestra a continuación.

```
var app=angular.module("mainApp",[]);app.controller("FirstController",function(e){e.name=
'Hello World !'})
```

Aquí, la minificación eliminó los espacios innecesarios y la variable $ scope del código. Entonces, cuando usamos este código minificado, no se imprimirá nada a la vista. Porque $ scope es una parte crucial entre el controlador y la vista, que ahora es reemplazada por la pequeña variable 'e'. Por lo tanto, cuando ejecute la aplicación, se producirá un error de dependencia de Unknown Provider 'e'.

Hay dos formas de anotar su código con información de nombre de servicio que son seguras de minificación:

Sintaxis de anotación en línea

```
var app = angular.module('mainApp', []);
app.controller('FirstController', ['$scope', function($scope) {
    $scope.message = 'Hello World !';
}]);
```

$ inyectar Sintaxis de anotación de propiedad

```
FirstController.$inject = ['$scope'];
var FirstController = function($scope) {
    $scope.message = 'Hello World !';
}

var app = angular.module('mainApp', []);
app.controller('FirstController', FirstController);
```

Después de minificación, este código será

```
var
app=angular.module("mainApp",[]);app.controller("FirstController",["$scope",function(a){a.message="Hel
World !"}]);
```

Aquí, angular considerará la variable 'a' para ser tratada como $ scope, y mostrará la salida como 'Hello World!'.

Tutoriales de vídeo de inicio de AngularJS

Hay muchos buenos tutoriales en video para el marco de AngularJS en egghead.io

Browse Courses

all

WATCH LUKAS RUEBBELKE'S COURSE

Using Angular 2 Patterns in Angular 1.x Apps

Implementing modern component-based architectu in your new or existing Angular 1.x web application i breath of fresh air.

In this course, y...

0 of 13 lessons

WATCH AARON FROST'S COURSE

Introduction to Angular Material

Angular Material is an Angula native, UI component framew from Google. It is a reference implementation of Google's Material Design and provide...

0 of 7 lessons

WATCH KENT C. DODDS'S COURSE

AngularJS Authentication with JWT

JSON Web Tokens (JWT) are more modern approach to authentication. As the web m to a greater separation betwe the client and server, JWT pro

0 of 7 lessons

WATCH JOEL HOOKS'S COURSE

Learn Protractor Testing for AngularJS

Protractor is an end-to-end te framework for AngularJS applications. It allows you do the browser and test the expe state of your ap...

0 of 10 lessons

- https://egghead.io/courses/angularjs-app-from-scratch-getting-started

- https://egghead.io/courses/angularjs-application-architecture
- https://egghead.io/courses/angular-material-introduction
- https://egghead.io/courses/building-an-angular-1-x-ionic-application
- https://egghead.io/courses/angular-and-webpack-for-modular-applications
- https://egghead.io/courses/angularjs-authentication-with-jwt
- https://egghead.io/courses/angularjs-data-modeling
- https://egghead.io/courses/angular-automation-with-gulp
- https://egghead.io/courses/learn-protractor-testing-for-angularjs
- https://egghead.io/courses/ionic-quickstart-for-windows
- https://egghead.io/courses/build-angular-1-x-apps-with-redux
- https://egghead.io/courses/using-angular-2-patterns-in-angular-1-x-apps

Lea Empezando con AngularJS en línea: https://riptutorial.com/es/angularjs/topic/295/empezando-con-angularjs

Capítulo 2: Alcances $ angulares

Observaciones

Angular usa un **árbol** de ámbitos para vincular la lógica (desde controladores, directivas, etc.) a la vista y es el mecanismo principal detrás de la detección de cambios en AngularJS. Se puede encontrar una referencia más detallada para los ámbitos en docs.angularjs.org

La raíz del árbol es accesible a través del servicio **inyectable $ rootScope** . Todos los ámbitos de $ child heredan los métodos y las propiedades de su alcance de $ padre, permitiendo a los niños acceder a métodos sin el uso de los Servicios Angulares.

Examples

Ejemplo básico de la herencia $ scope

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' };
}]);

<div ng-app="app" ng-conroller="myController">
    <input ng-model="person.name" />
    <div ng-repeat="number in [0,1,2,3]">
        {{person.name}} {{number}}
    </div>
</div>
```

En este ejemplo, la directiva ng-repeat crea un nuevo ámbito para cada uno de sus hijos recién creados.

Estos ámbitos creados son secundarios de su ámbito principal (en este caso, el ámbito creado por myController) y, por lo tanto, heredan todas sus propiedades, como persona.

Evitar heredar valores primitivos.

En javascript, la asignación de un valor no primitivo (como Objeto, Array, Función y muchos más) mantiene una referencia (una dirección en la memoria) al valor asignado.

Asignar un valor primitivo (Cadena, Número, Booleano o Símbolo) a dos variables diferentes, y cambiar una, no cambiará ambas:

```
var x = 5;
var y = x;
y = 6;
console.log(y === x, x, y); //false, 5, 6
```

Pero con un valor que no es primitiva, ya que ambas variables son simplemente mantener las

referencias al mismo objeto, el cambio de una variable **va a** cambiar a la otra:

```
var x = { name : 'John Doe' };
var y = x;
y.name = 'Jhon';
console.log(x.name === y.name, x.name, y.name); //true, John, John
```

En angular, cuando se crea un ámbito, se le asignan todas sus propiedades principales. Sin embargo, cambiar las propiedades después solo afectará al ámbito principal si no es un valor primitivo:

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' }; //non-primitive
    $scope.name = 'Jhon Doe'; //primitive
}])
.controller('myController1', ['$scope', function($scope){}]);

<div ng-app="app" ng-controller="myController">
    binding to input works: {{person.name}}<br/>
    binding to input does not work: {{name}}<br/>
    <div ng-controller="myController1">
        <input ng-model="person.name" />
        <input ng-model="name" />
    </div>
</div>
```

Recuerde: en Angular, los ámbitos se pueden crear de muchas maneras (como directivas incorporadas o personalizadas, o la función `$scope.$new()`), y es probable que sea imposible mantener un registro del árbol de ámbitos.

Utilizar solo valores no primitivos como propiedades de alcance lo mantendrá en el lado seguro (a menos que necesite una propiedad para no heredar, u otros casos en los que tenga conocimiento de la herencia de alcance).

Una función disponible en toda la aplicación.

Tenga cuidado, este enfoque podría considerarse como un mal diseño para las aplicaciones angulares, ya que requiere que los programadores recuerden dónde se ubican las funciones en el árbol de alcance y que sean conscientes de la herencia del alcance. En muchos casos, se preferiría inyectar un servicio (práctica angular: uso de la herencia del alcance frente a la inyección .

Este ejemplo solo muestra cómo se puede usar la herencia de alcance para nuestras necesidades y cómo puede aprovecharla, y no las mejores prácticas para diseñar una aplicación completa.

En algunos casos, podríamos aprovechar la herencia del alcance y establecer una función como una propiedad del rootScope. De esta manera, todos los ámbitos de la aplicación (excepto los ámbitos aislados) heredarán esta función, y se puede llamar desde cualquier parte de la aplicación.

```
angular.module('app', [])
.run(['$rootScope', function($rootScope){
    var messages = []
    $rootScope.addMessage = function(msg){
        messages.push(msg);
    }
}]);

<div ng-app="app">
    <a ng-click="addMessage('hello world!')">it could be accsessed from here</a>
    <div ng-include="inner.html"></div>
</div>
```

inner.html:

```
<div>
    <button ng-click="addMessage('page')">and from here to!</button>
</div>
```

Creando eventos personalizados de $ scope

Al igual que los elementos HTML normales, es posible que $ scopes tengan sus propios eventos. Los eventos de $ scope se pueden suscribir de la siguiente manera:

```
 $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
});
```

Si necesita anular el registro de un detector de eventos, la función **$ on** devolverá una función de desvinculación. Para continuar con el ejemplo anterior:

```
var unregisterMyEvent = $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
    unregisterMyEvent();
});
```

Hay dos formas de activar su propio evento **$** scope **$ broadcast** y **$ emit** . Para notificar a los padres sobre el alcance de un evento específico, use **$ emit**

```
$scope.$emit('my-event', { custom: 'data' });
```

El ejemplo anterior activará los escuchas de eventos para `my-event` en el ámbito principal y continuará subiendo el árbol de **ámbitos** a **$ rootScope a** menos que un oyente llame a `stopPropagation` en el evento. Solo los eventos activados con **$** `stopPropagation` pueden llamar `stopPropagation`

El reverso de **$ emit** es **$ broadcast** , que activará a los oyentes de eventos en todos los ámbitos secundarios del árbol de ámbitos que sean secundarios del alcance que se denominó **$ broadcast** .

```
$scope.$broadcast('my-event', { custom: 'data' });
```

Los eventos activados con **$ broadcast** no pueden ser cancelados.

Usando las funciones de $ scope

Si bien la declaración de una función en $ rootscope tiene sus ventajas, también podemos declarar una función $ scope en cualquier parte del código inyectado por el servicio $ scope. Controlador, por ejemplo.

Controlador

```
myApp.controller('myController', ['$scope', function($scope){
    $scope.myFunction = function () {
        alert("You are in myFunction!");
    };
}]);
```

Ahora puedes llamar a tu función desde el controlador usando:

```
$scope.myfunction();
```

O a través de HTML que está bajo ese controlador específico:

```
<div ng-controller="myController">
    <button ng-click="myFunction()"> Click me! </button>
</div>
```

Directiva

Una directiva angular es otro lugar donde puedes usar tu alcance:

```
myApp.directive('triggerFunction', function() {
    return {
        scope: {
            triggerFunction: '&'
        },
        link: function(scope, element) {
            element.bind('mouseover', function() {
                scope.triggerFunction();
            });
        }
    };
});
```

Y en tu código HTML bajo el mismo controlador:

```
<div ng-controller="myController">
    <button trigger-function="myFunction()"> Hover over me! </button>
</div>
```

Por supuesto, puede usar ngMouseover para la misma cosa, pero lo especial de las directivas es que puede personalizarlas de la manera que desee. Y ahora sabes cómo usar tus funciones de $ scope dentro de ellas, ¡sé creativo!

¿Cómo puede limitar el alcance de una directiva y por qué haría esto?

El alcance se utiliza como el "pegamento" que usamos para comunicarnos entre el controlador principal, la directiva y la plantilla de la directiva. Cada vez que se arranca la aplicación AngularJS, se crea un objeto rootScope. Cada ámbito creado por los controladores, directivas y servicios se hereda de forma prototípica de rootScope.

Sí, podemos limitar el alcance de una directiva. Podemos hacerlo creando un ámbito aislado para la directiva.

Hay 3 tipos de ámbitos directivos:

1. Ámbito: Falso (Directiva utiliza su ámbito primario)
2. Ámbito: Verdadero (la directiva adquiere un nuevo ámbito)
3. Ámbito: {} (Directiva obtiene un nuevo ámbito aislado)

Directivas con el nuevo alcance aislado: cuando creamos un nuevo alcance aislado, no se heredará del alcance principal. Este nuevo ámbito se denomina Ámbito aislado porque está completamente separado de su ámbito principal. ¿Por qué? deberíamos usar un alcance aislado: debemos usar un alcance aislado cuando queremos crear una directiva personalizada porque nos aseguraremos de que nuestra directiva sea genérica y esté ubicada en cualquier lugar dentro de la aplicación. El ámbito principal no va a interferir con el ámbito de la directiva.

Ejemplo de alcance aislado:

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

app.controller("Ctrl1",function($scope){
    $scope.name = "Prateek";
    $scope.reverseName = function(){
        $scope.name = $scope.name.split('').reverse().join('');
    };
});
app.directive("myDirective", function(){
    return {
        restrict: "EA",
        scope: {},
        template: "<div>Your name is : {{name}}</div>"+
        "Change your name : <input type='text' ng-model='name'/>"
    };
});
```

Hay 3 tipos de prefijos que AngularJS proporciona para el alcance aislado estos son:

1. "@" (Enlace de texto / enlace unidireccional)
2. "=" (Enlace directo del modelo / Enlace bidireccional)
3. "&" (Enlace de comportamiento / Enlace de método)

Todos estos prefijos reciben datos de los atributos del elemento directivo como:

```
<div my-directive
  class="directive"
```

```
  name="{{name}}"
  reverse="reverseName()"
  color="color" >
</div>
```

Lea Alcances $ angulares en línea: https://riptutorial.com/es/angularjs/topic/3157/alcances---angulares

Capítulo 3: Almacenamiento de sesión

Examples

Manejo de la sesión de almacenamiento a través del servicio usando angularjs.

Servicio de almacenamiento de sesión:

Servicio de fábrica común que guardará y devolverá los datos de sesión guardados según la clave.

```
 'use strict';

/**
 * @ngdoc factory
 * @name app.factory:storageService
 * @description This function will communicate with HTML5 sessionStorage via Factory Service.
 */

app.factory('storageService', ['$rootScope', function($rootScope) {

    return {
        get: function(key) {
            return sessionStorage.getItem(key);
        },
        save: function(key, data) {
            sessionStorage.setItem(key, data);
        }
    };
}]);
```

En el controlador:

Inyecte la dependencia de storageService en el controlador para establecer y obtener los datos del almacenamiento de la sesión.

```
app.controller('myCtrl',['storageService',function(storageService) {

  // Save session data to storageService
  storageService.save('key', 'value');

  // Get saved session data from storageService
  var sessionData = storageService.get('key');

});
```

Lea Almacenamiento de sesión en línea:

https://riptutorial.com/es/angularjs/topic/8201/almacenamiento-de-sesion

Capítulo 4: angularjs con filtro de datos, paginación etc

Introducción

Ejemplo de proveedor y consulta sobre datos de visualización con filtro, paginación, etc. en Angularjs.

Examples

Angularjs muestra datos con filtro, paginación

```
<div ng-app="MainApp" ng-controller="SampleController">
    <input ng-model="dishName" id="search" class="form-control" placeholder="Filter text">
    <ul>
        <li dir-paginate="dish in dishes |  filter : dishName | itemsPerPage: pageSize"
pagination-id="flights">{{dish}}</li>
    </ul>
    <dir-pagination-controls boundary-links="true" on-page-
change="changeHandler(newPageNumber)" pagination-id="flights"></dir-pagination-controls>
</div>
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="pagination.js"></script>
<script type="text/javascript">

var MainApp = angular.module('MainApp', ['angularUtils.directives.dirPagination'])
MainApp.controller('SampleController', ['$scope', '$filter', function ($scope, $filter) {

    $scope.pageSize = 5;

    $scope.dishes = [
    'noodles',
    'sausage',
    'beans on toast',
    'cheeseburger',
    'battered mars bar',
    'crisp butty',
    'yorkshire pudding',
    'wiener schnitzel',
    'sauerkraut mit ei',
    'salad',
    'onion soup',
    'bak choi',
    'avacado maki'
    ];

    $scope.changeHandler = function (newPage) { };
}]);
</script>
```

Lea angularjs con filtro de datos, paginación etc en línea:
https://riptutorial.com/es/angularjs/topic/10821/angularjs-con-filtro-de-datos--paginacion-etc

Capítulo 5: AngularJS gotchas y trampas

Examples

El enlace de datos bidireccional deja de funcionar

Uno debe tener en cuenta que:

1. El enlace de datos de Angular se basa en la herencia prototípica de JavaScript, por lo que está sujeto a la sombra variable .
2. Un ámbito secundario normalmente se hereda prototípicamente de su ámbito primario. Una excepción a esta regla es una directiva que tiene un alcance aislado, ya que no se hereda prototípicamente.
3. Hay algunas directivas que crean un nuevo ámbito secundario: `ng-repeat` , `ng-switch` , `ng-view` , `ng-if` , `ng-controller` , `ng-include` , etc.

Esto significa que cuando intenta vincular de forma bidireccional algunos datos a una primitiva que se encuentra dentro de un ámbito secundario (o viceversa), es posible que las cosas no funcionen como se espera. Aquí hay un ejemplo de lo fácil que es *"romper"* AngularJS.

Este problema se puede evitar fácilmente siguiendo estos pasos:

1. Tener un "." Dentro de su plantilla HTML cada vez que enlace algunos datos
2. Utilice la sintaxis de `controllerAs` ya que promueve el uso de la vinculación a un objeto "punteado"
3. $ parent se puede usar para acceder a las variables de `scope` principal en lugar de al alcance secundario. como dentro de `ng-if` podemos usar `ng-model="$parent.foo"` ..

Una alternativa para lo anterior es vincular `ngModel` a una función getter / setter que actualizará la versión en caché del modelo cuando se le llame con argumentos, o lo devolverá cuando se le llame sin argumentos. Para utilizar una función getter / setter, debe agregar `ng-model-options="{ getterSetter: true }"` al elemento con el atributo `ngModal` , y llamar a la función getter si desea mostrar su valor en la expresión (Ejemplo de trabajo).

Ejemplo

Ver:

```
<div ng-app="myApp" ng-controller="MainCtrl">
    <input type="text" ng-model="foo" ng-model-options="{ getterSetter: true }">
    <div ng-if="truthyValue">
        <!-- I'm a child scope (inside ng-if), but i'm synced with changes from the outside
scope -->
        <input type="text" ng-model="foo">
    </div>
    <div>$scope.foo: {{ foo() }}</div>
</div>
```

Controlador:

```
angular.module('myApp', []).controller('MainCtrl', ['$scope', function($scope) {
    $scope.truthyValue = true;

    var _foo = 'hello'; // this will be used to cache/represent the value of the 'foo' model

    $scope.foo = function(val) {
        // the function return the the internal '_foo' varibale when called with zero
arguments,
        // and update the internal `_foo` when called with an argument
        return arguments.length ? (_foo = val) : _foo;
    };
}]);
```

Mejor práctica : es mejor mantener a los captadores rápidamente porque es probable que Angular los llame con más frecuencia que otras partes de su código (referencia).

Cosas que hacer cuando se utiliza html5Mode

Cuando se utiliza `html5Mode([mode])` es necesario que:

1. Especifica la URL base para la aplicación con un `<base href="">` en el encabezado de su `index.html` .

2. Es importante que la etiqueta `base` esté antes que cualquier etiqueta con solicitudes de URL. De lo contrario, esto podría dar lugar a este error: `"Resource interpreted as stylesheet but transferred with MIME type text/html"` . Por ejemplo:

```
<head>
    <meta charset="utf-8">
    <title>Job Seeker</title>

    <base href="/">

    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="/styles/main.css">
</head>
```

3. Si no desea especificar una etiqueta `base` , configure `$locationProvider` para que no requiera una etiqueta `base` pasando un objeto de definición con `requireBase:false` a `$locationProvider.html5Mode()` como esto:

```
$locationProvider.html5Mode({
    enabled: true,
    requireBase: false
});
```

4. Para poder admitir la carga directa de URL de HTML5, debe habilitar la reescritura de URL del lado del servidor. De AngularJS / Guía del desarrollador / Uso de $ location

> El uso de este modo requiere la reescritura de URL en el lado del servidor, básicamente, debe volver a escribir todos sus enlaces al punto de entrada de su

aplicación (por ejemplo, `index.html`). Requerir una etiqueta `<base>` también es importante para este caso, ya que permite a Angular diferenciar entre la parte de la URL que es la base de la aplicación y la ruta que debe ser manejada por la aplicación.

Puede encontrar un excelente recurso para ejemplos de reescritura de solicitudes para varias implementaciones de servidores HTTP en las preguntas frecuentes de ui-router: Cómo: Configurar su servidor para que funcione con html5Mode . Por ejemplo, apache

```
RewriteEngine on

# Don't rewrite files or directories
RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]

# Rewrite everything else to index.html to allow html5 state links
RewriteRule ^ index.html [L]
```

nginx

```
server {
    server_name my-app;

    root /path/to/app;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

Exprimir

```
var express = require('express');
var app = express();

app.use('/js', express.static(__dirname + '/js'));
app.use('/dist', express.static(__dirname + '/../dist'));
app.use('/css', express.static(__dirname + '/css'));
app.use('/partials', express.static(__dirname + '/partials'));

app.all('/*', function(req, res, next) {
    // Just send the index.html for other files to support HTML5Mode
    res.sendFile('index.html', { root: __dirname });
});

app.listen(3006); //the port you want to use
```

7 pecados mortales de AngularJS

A continuación se muestra la lista de algunos errores que los desarrolladores suelen cometer durante el uso de las funcionalidades de AngularJS, algunas lecciones aprendidas y soluciones para ellos.

1. Manipular el DOM a través del controlador.

Es legal, pero hay que evitarlo. Los controladores son los lugares donde define sus dependencias, vincula sus datos a la vista y crea más lógica de negocios. Técnicamente puede manipular el DOM en un controlador, pero siempre que necesite una manipulación similar o similar en otra parte de su aplicación, se necesitará otro controlador. Por lo tanto, la mejor práctica de este enfoque es crear una directiva que incluya todas las manipulaciones y usar la directiva en toda su aplicación. Por lo tanto, el controlador deja la vista intacta y hace su trabajo. En una directiva, la función de enlace es el mejor lugar para manipular el DOM. Tiene acceso completo al alcance y al elemento, por lo que al usar una directiva, también puede aprovechar la reutilización.

```
link: function($scope, element, attrs) {
    //The best place to manipulate DOM
}
```

Puede acceder a los elementos DOM en la función de vinculación de varias maneras, como el parámetro del `element` , el método `angular.element()` o el Javascript puro.

2. Enlace de datos en la transclusión.

AngularJS es famoso por su enlace de datos de dos vías. Sin embargo, es posible que a veces descubra que sus datos solo están enlazados en un solo sentido dentro de las directivas. Detente ahí, AngularJS no está mal, pero probablemente tú. Las directivas son lugares poco peligrosos ya que están involucrados los ámbitos de niños y los aislados. Supongamos que tiene la siguiente directiva con una transclusión

```
<my-dir>
    <my-transclusion>
    </my-transclusion>
</my-dir>
```

Y dentro de mi-transclusión, tiene algunos elementos que están vinculados a los datos en el ámbito externo.

```
<my-dir>
    <my-transclusion>
        <input ng-model="name">
    </my-transclusion>
</my-dir>
```

El código anterior no funcionará correctamente. Aquí, la transclusión crea un ámbito secundario y puede obtener la variable de nombre, correcto, pero cualquier cambio que realice en esta variable permanecerá allí. Entonces, realmente puedes acceder a esta variable como **$ parent.name** . Sin embargo, este uso podría no ser la mejor práctica. Un mejor enfoque sería envolver las variables dentro de un objeto. Por ejemplo, en el controlador puedes crear:

```
$scope.data = {
    name: 'someName'
}
```

Luego, en la transclusión, puede acceder a esta variable a través del objeto 'datos' y ver que el enlace bidireccional funciona perfectamente.

```
<input ng-model="data.name">
```

No solo en las transclusiones, sino en toda la aplicación, es una buena idea usar la notación punteada.

3. Directivas múltiples juntas

En realidad, es legal usar dos directivas juntas dentro del mismo elemento, siempre que cumpla con la regla: no pueden existir dos ámbitos aislados en el mismo elemento. En general, al crear una nueva directiva personalizada, asigna un ámbito aislado para facilitar el paso de parámetros. Suponiendo que las directivas myDirA y myDirB tengan ámbitos aislados y myDirC no, el siguiente elemento será válido:

```
<input my-dir-a my-dirc>
```

mientras que el siguiente elemento causará error de consola:

```
<input my-dir-a my-dir-b>
```

Por lo tanto, las directivas deben usarse con prudencia, teniendo en cuenta los ámbitos.

4. Uso indebido de $ emit

$ emit, $ broadcast y $ on, funcionan en un principio de remitente-receptor. En otras palabras, son un medio de comunicación entre los controladores. Por ejemplo, la siguiente línea emite el 'someEvent' del controlador A, que debe ser capturado por el controlador B en cuestión.

```
$scope.$emit('someEvent', args);
```

Y la siguiente línea captura el 'someEvent'

```
$scope.$on('someEvent', function(){});
```

Hasta ahora todo parece perfecto. Pero recuerde que, si el controlador B aún no se ha invocado, el evento no se detectará, lo que significa que se deben invocar los controladores de emisor y receptor para que esto funcione. Entonces, una vez más, si no está seguro de que definitivamente tiene que usar $ emit, construir un servicio parece una mejor manera.

5. Uso indebido de $ scope. $ Watch

$ scope. $ watch se usa para ver un cambio de variable. Cada vez que una variable ha cambiado, este método es invocado. Sin embargo, un error común cometido es cambiar la variable dentro de $ scope. $ Watch. Esto causará inconsistencias y un bucle infinito de digestión $ en algún punto.

```
$scope.$watch('myCtrl.myVariable', function(newVal) {
```

```
    this.myVariable++;
});
```

Así que en la función anterior, asegúrese de no tener operaciones en myVariable y newVal.

6. Encuadernación de métodos a vistas.

Este es uno de los pecados más mortíferos. AngularJS tiene enlaces bidireccionales, y siempre que algo cambia, las vistas se actualizan muchas veces. Entonces, si unes un método a un atributo de una vista, ese método podría llamarse cientos de veces, lo que también te vuelve loco durante la depuración. Sin embargo, solo hay algunos atributos que se crean para el enlace de métodos, como ng-click, ng-blur, ng-on-change, etc., que esperan métodos como paremeter. Por ejemplo, suponga que tiene la siguiente vista en su marca:

```
<input ng-disabled="myCtrl.isDisabled()" ng-model="myCtrl.name">
```

Aquí usted verifica el estado deshabilitado de la vista a través del método isDisabled. En el controlador myCtrl, tienes:

```
vm.isDisabled = function(){
    if(someCondition)
        return true;
    else
        return false;
}
```

En teoría, puede parecer correcto, pero técnicamente esto causará una sobrecarga, ya que el método se ejecutará innumerables veces. Para resolver esto, debes enlazar una variable. En su controlador, la siguiente variable debe existir:

```
vm.isDisabled
```

Puede volver a iniciar esta variable en la activación del controlador.

```
if(someCondition)
    vm.isDisabled = true
else
    vm.isDisabled = false
```

Si la condición no es estable, puede vincular esto a otro evento. Entonces deberías enlazar esta variable a la vista:

```
<input ng-disabled="myCtrl.isDisabled" ng-model="myCtrl.name">
```

Ahora, todos los atributos de la vista tienen lo que esperan y los métodos se ejecutarán solo cuando sea necesario.

7. No usar las funcionalidades de Angular.

AngularJS proporciona una gran comodidad con algunas de sus funcionalidades, no solo simplificando su código sino que también lo hace más eficiente. Algunas de estas características se enumeran a continuación:

1. **angular.forEach** para los bucles (Precaución, no puede "romperla"; solo puede evitar que entre al cuerpo, así que considere el rendimiento aquí).
2. **Elemento angular** para selectores DOM
3. **angular.copy** : use esto cuando no debe modificar el objeto principal
4. **Las validaciones de formularios** ya son impresionantes. Utilice sucio, prístino, tocado, válido, requerido y así sucesivamente.
5. Además del depurador Chrome, use **la depuración remota** para el desarrollo móvil también.
6. Y asegúrate de usar **Batarang** . Es una extensión gratuita de Chrome donde puedes inspeccionar fácilmente los ámbitos.
   .

Lea AngularJS gotchas y trampas en línea:
https://riptutorial.com/es/angularjs/topic/3208/angularjs-gotchas-y-trampas

Capítulo 6: Carga lenta

Observaciones

1. Si sus dependencias cargadas perezosas requieren otras dependencias cargadas perezosas, ¡asegúrese de cargarlas en el orden correcto!

```
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 {
  files: [
  'path/to/lazily/loaded/dependency1.js',
  'path/to/lazily/loaded/dependency2.js', //<--- requires lazily loaded dependency1
  'path/to/lazily/loaded/dependency.css'
  ],
  serie: true //Sequential load instead of parallel
 }
]);
```

Examples

Preparando tu proyecto para carga lenta

Después de incluir `oclazyload.js` en su archivo de índice, declare `ocLazyLoad` como una dependencia en `app.js`

```
//Make sure you put the correct dependency! it is spelled different than the service!
angular.module('app', [
 'oc.lazyLoad',
 'ui-router'
])
```

Uso

Para cargar archivos con `$ocLazyLoad` inyecte el servicio `$ocLazyLoad` en un controlador u otro servicio

```
.controller('someCtrl', function($ocLazyLoad) {
 $ocLazyLoad.load('path/to/file.js').then(...);
});
```

Los módulos angulares se cargarán automáticamente en angular.

Otra variación:

```
$ocLazyLoad.load([
  'bower_components/bootstrap/dist/js/bootstrap.js',
```

```
  'bower_components/bootstrap/dist/css/bootstrap.css',
  'partials/template1.html'
]);
```

Para una lista completa de variaciones visite la documentación oficial.

Uso con enrutador

UI-Router:

```
.state('profile', {
 url: '/profile',
 controller: 'profileCtrl as vm'
 resolve: {
  module: function($ocLazyLoad) {
   return $ocLazyLoad.load([
    'path/to/profile/module.js',
    'path/to/profile/style.css'
   ]);
  }
 }
});
```

ngRuta:

```
.when('/profile', {
     controller: 'profileCtrl as vm'
     resolve: {
      module: function($ocLazyLoad) {
       return $ocLazyLoad.load([
        'path/to/profile/module.js',
        'path/to/profile/style.css'
       ]);
      }
     }
    });
```

Uso de inyección de dependencia

La siguiente sintaxis le permite especificar dependencias en su `module.js` lugar de una especificación explícita cuando usa el servicio

```
//lazy_module.js
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 [
  'path/to/lazily/loaded/dependency.js',
  'path/to/lazily/loaded/dependency.css'
 ]
]);
```

Nota : ¡esta sintaxis solo funcionará para módulos cargados perezosamente!

Utilizando la directiva

```
<div oc-lazy-load="['path/to/lazy/loaded/directive.js',
'path/to/lazy/loaded/directive.html']">

<!-- myDirective available here -->
<my-directive></my-directive>

</div>
```

Lea Carga lenta en línea: https://riptutorial.com/es/angularjs/topic/6400/carga-lenta

Capítulo 7: Cómo funciona el enlace de datos

Observaciones

Entonces, si bien este concepto de enlace de datos en su conjunto es fácil para el desarrollador, es bastante pesado en el navegador, ya que Angular escucha cada cambio de evento y ejecuta el Ciclo de resumen. Debido a esto, siempre que adjuntemos algún modelo a la vista, asegúrese de que Scope esté lo más optimizado posible.

Examples

Ejemplo de enlace de datos

```
<p ng-bind="message"></p>
```

Este 'mensaje' debe adjuntarse al alcance del controlador de elementos actual.

```
$scope.message = "Hello World";
```

En un momento posterior, incluso si el modelo de mensaje se actualiza, ese valor actualizado se refleja en el elemento HTML. Cuando se compile angularmente, la plantilla "Hola mundo" se adjuntará al HTML interno del mundo actual. Angular mantiene un mecanismo de observación de todas las directivas relacionadas con la vista. Tiene un mecanismo de Ciclo de resumen en el que itera a través de la matriz Watchers, actualizará el elemento DOM si hay un cambio en el valor anterior del modelo.

No hay una verificación periódica de Ámbito si hay algún cambio en los Objetos adjuntos. No todos los objetos adjuntos al alcance son observados. El alcance prototípicamente mantiene un **$$ WatchersArray** . El alcance solo se repite a través de WatchersArray cuando se llama $ digest.

Angular agrega un observador a WatchersArray para cada uno de estos

1. {{expresión}} - En sus plantillas (y en cualquier otro lugar donde haya una expresión) o cuando definamos ng-model.
2. $ scope. $ watch ('expresión / función') - En su JavaScript solo podemos adjuntar un objeto de alcance para que lo vea angular.

La función **$ watch** tiene tres parámetros:

1. La primera es una función de observador que simplemente devuelve el objeto o simplemente podemos agregar una expresión.

2. La segunda es una función de escucha que se llamará cuando haya un cambio en el objeto. Todas las cosas como los cambios de DOM se implementarán en

esta función.

3. El tercero es un parámetro opcional que toma un booleano. Si es verdadero, el profundo angular observa el objeto y si su falso angular simplemente hace una referencia al objeto. La implementación aproximada de $ watch se ve así

```
Scope.prototype.$watch = function(watchFn, listenerFn) {
   var watcher = {
       watchFn: watchFn,
       listenerFn: listenerFn || function() { },
       last: initWatchVal  // initWatchVal is typically undefined
   };
   this.$$watchers.push(watcher); // pushing the Watcher Object to Watchers
};
```

Hay algo interesante en Angular llamado Digest Cycle. El ciclo $ digest comienza como resultado de una llamada a $ scope. $ Digest (). Supongamos que cambia un modelo de $ scope en una función de controlador a través de la directiva ng-click. En ese caso, AngularJS activa automáticamente un ciclo de $ digest llamando a $ digest (). Además de ng-click, hay otras directivas / servicios incorporados que le permiten cambiar los modelos (por ejemplo, ng-model, $ timeout, etc.) y desencadenar automáticamente un ciclo $ digest. La implementación aproximada de $ digest se ve así.

```
Scope.prototype.$digest = function() {
      var dirty;
      do {
          dirty = this.$$digestOnce();
      } while (dirty);
}
Scope.prototype.$$digestOnce = function() {
   var self = this;
   var newValue, oldValue, dirty;
   _.forEach(this.$$watchers, function(watcher) {
          newValue = watcher.watchFn(self);
          oldValue = watcher.last;   // It just remembers the last value for dirty checking
          if (newValue !== oldValue) { //Dirty checking of References
   // For Deep checking the object , code of Value
   // based checking of Object should be implemented here
             watcher.last = newValue;
             watcher.listenerFn(newValue,
                  (oldValue === initWatchVal ? newValue : oldValue),
                   self);
          dirty = true;
          }
     });
   return dirty;
 };
```

Si usamos la función **setTimeout () de** JavaScript para actualizar un modelo de alcance, Angular no tiene forma de saber qué puede cambiar. En este caso, es nuestra responsabilidad llamar a $ apply () manualmente, lo que desencadena un ciclo de $ digest. De manera similar, si tiene una directiva que configura un detector de eventos DOM y cambia algunos modelos dentro de la función del controlador, debe llamar a $ apply () para asegurarse de que los cambios surtan efecto. La gran idea de $ apply es que podemos ejecutar algunos códigos que no son conscientes

de Angular, que aún pueden cambiar las cosas en el alcance. Si envolvemos ese código en $ apply, se ocupará de llamar a $ digest (). Implementación aproximada de $ apply ().

```
Scope.prototype.$apply = function(expr) {
       try {
         return this.$eval(expr); //Evaluating code in the context of Scope
       } finally {
         this.$digest();
       }
};
```

Lea Cómo funciona el enlace de datos en línea:
https://riptutorial.com/es/angularjs/topic/2342/como-funciona-el-enlace-de-datos

Capítulo 8: Compartir datos

Observaciones

Una pregunta muy común cuando se trabaja con Angular es cómo compartir datos entre controladores. El uso de un servicio es la respuesta más frecuente y este es un ejemplo simple que muestra un patrón de fábrica para compartir cualquier tipo de objeto de datos entre dos o más controladores. Debido a que es una referencia de objeto compartido, una actualización en un controlador estará disponible inmediatamente en todos los demás controladores que usen el servicio. Tenga en cuenta que tanto el servicio y la fábrica y los dos proveedores .

Examples

Usando ngStorage para compartir datos

En primer lugar, incluya la fuente **ngStorage** en su index.html.

Un ejemplo de inyección de `ngStorage` src sería:

```
<head>
    <title>Angular JS ngStorage</title>
    <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src="https://rawgithub.com/gsklee/ngStorage/master/ngStorage.js"></script>
</head>
```

`ngStorage` le ofrece 2 `ngStorage` de almacenamiento: `$localStorage` y `$sessionStorage` . Necesita requerir ngStorage e inyectar los servicios.

Supongamos que si `ng-app="myApp"` , estaría inyectando `ngStorage` siguiente manera:

```
var app = angular.module('myApp', ['ngStorage']);
        app.controller('controllerOne', function($localStorage,$sessionStorage) {
        // an object to share
        var sampleObject = {
            name: 'angularjs',
            value: 1
        };
       $localStorage.valueToShare = sampleObject;
       $sessionStorage.valueToShare = sampleObject;
    })
.controller('controllerTwo', function($localStorage,$sessionStorage) {
    console.log('localStorage: '+ $localStorage +'sessionStorage: '+$sessionStorage);
})
```

`$localStorage` y `$sessionStorage` son accesibles globalmente a través de cualquier controlador siempre y cuando inyecte esos servicios en los controladores.

También puede utilizar el `localStorage` y `sessionStorage` de `HTML5` . Sin embargo, usar `HTML5`

localStorage requeriría que localStorage y deserialice sus objetos antes de usarlos o guardarlos.

Por ejemplo:

```
var myObj = {
    firstname: "Nic",
    lastname: "Raboy",
    website: "https://www.google.com"
}
//if you wanted to save into localStorage, serialize it
window.localStorage.set("saved", JSON.stringify(myObj));

//unserialize to get object
var myObj = JSON.parse(window.localStorage.get("saved"));
```

Compartir datos de un controlador a otro usando el servicio

Podemos crear un `service` para `set` y `get` los datos entre los `controllers` y luego inyectar ese servicio en la función del controlador donde queremos usarlo.

Servicio :

```
app.service('setGetData', function() {
  var data = '';
    getData: function() { return data; },
    setData: function(requestData) { data = requestData; }
});
```

Controladores:

```
app.controller('myCtrl1', ['setGetData',function(setGetData) {

  // To set the data from the one controller
  var data = 'Hello World !!';
  setGetData.setData(data);

}]);

app.controller('myCtrl2', ['setGetData',function(setGetData) {

  // To get the data from the another controller
  var res = setGetData.getData();
  console.log(res); // Hello World !!

}]);
```

Aquí, podemos ver que `myCtrl1` se usa para `setting` los datos y `myCtrl2` se usa para `getting` los datos. Entonces, podemos compartir los datos de un controlador a otro como este.

Lea Compartir datos en línea: https://riptutorial.com/es/angularjs/topic/1923/compartir-datos

Capítulo 9: Componentes

Parámetros

Parámetro	Detalles
=	Para el uso de enlace de datos de dos vías. Esto significa que si actualiza esa variable en el alcance de su componente, el cambio se reflejará en el alcance principal.
<	Enlaces unidireccionales cuando solo queremos leer un valor de un ámbito primario y no actualizarlo.
@	Parámetros de la cadena.
Y	Para devoluciones de llamadas en caso de que su componente necesite enviar algo a su ámbito principal.
-	-
LifeCycle Hooks	**Detalles** (requiere angular.version> = 1.5.3)
$ onInit ()	Se llama a cada controlador después de que todos los controladores de un elemento se hayan construido y se hayan inicializado sus enlaces. Este es un buen lugar para colocar el código de inicialización de su controlador.
$ onChanges (changesObj)	Se llama cada vez que se actualizan los enlaces unidireccionales. El `changesObj` es un hash cuyas claves son los nombres de las propiedades enlazadas que han cambiado, y los valores son un objeto de la forma `{ currentValue, previousValue, isFirstChange() }` .
$ onDestroy ()	Llamado a un controlador cuando se destruye su ámbito de contención. Utilice este gancho para liberar recursos externos, relojes y controladores de eventos.
$ postLink ()	Llamado después de que el elemento de este controlador y sus hijos hayan sido enlazados. Este gancho puede considerarse análogo a los ganchos ngAfterViewInit y ngAfterContentInit en Angular 2.
$ doCheck ()	Llamado en cada vuelta del ciclo de digestión. Brinda una oportunidad para detectar y actuar sobre los cambios. Cualquier acción que desee realizar en respuesta a los cambios que detecte debe invocarse desde este enlace; implementar esto no tiene efecto cuando se llama a $ onChanges.

Observaciones

Componente es un tipo especial de directiva que usa una configuración más simple que es adecuada para una estructura de aplicación basada en componentes. Los componentes se introdujeron en Angular 1.5, los ejemplos en esta sección **no funcionarán** con versiones anteriores de AngularJS.

Una guía de desarrollador completa sobre Componentes está disponible en https://docs.angularjs.org/guide/component

Examples

Componentes básicos y ganchos de ciclo de vida

¿Qué es un componente?

- Un componente es básicamente una directiva que usa una configuración más simple y que es adecuada para una arquitectura basada en componentes, que es de lo que trata Angular 2. Piense en un componente como un widget: un fragmento de código HTML que puede reutilizar en varios lugares diferentes de su aplicación web.

Componente

```
angular.module('myApp', [])
    .component('helloWorld', {
        template: '<span>Hello World!</span>'
    });
```

Margen

```
<div ng-app="myApp">
  <hello-world> </hello-world>
</div>
```

Demo en vivo

Usando datos externos en el componente:

Podríamos agregar un parámetro para pasar un nombre a nuestro componente, que se usaría de la siguiente manera:

```
angular.module("myApp", [])
  .component("helloWorld",{
    template: '<span>Hello {{$ctrl.name}}!</span>',
    bindings: { name: '@' }
```

```
});
```

Margen

```
<div ng-app="myApp">
  <hello-world name="'John'" > </hello-world>
</div>
```

Demo en vivo

Uso de controladores en componentes

Echemos un vistazo a cómo agregarle un controlador.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.myName = 'Alain';
      }
  });
```

Margen

```
<div ng-app="myApp">
  <hello-world name="John"> </hello-world>
</div>
```

CodePen Demo

Los parámetros pasados al componente están disponibles en el alcance del controlador justo antes `$onInit` función `$onInit` . Considera este ejemplo:

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.$onInit = function() {
          this.myName = "Mac" + this.name;
        }
      }
  });
```

En la plantilla de arriba, esto representaría "¡Hola John, soy MacJohn!".

Tenga en cuenta que `$ctrl` es el valor predeterminado Angular para `controllerAs` si no se especifica uno.

Demo en vivo

Usando "require" como un objeto

En algunos casos, es posible que deba acceder a los datos de un componente principal dentro de su componente.

Esto se puede lograr especificando que nuestro componente requiere ese componente principal, el requisito nos dará una referencia al controlador de componente requerido, que luego se puede usar en nuestro controlador como se muestra en el siguiente ejemplo:

> Observe que se garantiza que los controladores necesarios estarán listos solo después del gancho $ onInit.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      require: {
        parent: '^parentComponent'
      },
      controller: function () {
        // here this.parent might not be initiated yet

        this.$onInit = function() {
           // after $onInit, use this.parent to access required controller
           this.parent.foo();
        }

      }
  });
```

Sin embargo, tenga en cuenta que esto crea un acoplamiento estrecho entre el niño y el padre.

Componentes en angular JS

Los componentes en angularJS se pueden visualizar como una directiva personalizada (<html> esto en una directiva HTML, y algo como esto será una directiva personalizada <CUALQUIER COSA>). Un componente contiene una vista y un controlador. El controlador contiene la lógica de negocio que está enlazada con una vista, que el usuario ve. El componente difiere de una directiva angular porque contiene menos configuración. Un componente angular puede definirse así.

```
angular.module("myApp",[]).component("customer", {})
```

Los componentes se definen en los módulos angulares. Contienen dos argumentos, uno es el nombre del componente y el segundo es un objeto que contiene un par de valores clave, que define qué vista y qué controlador va a usar de esta manera.

```
angular.module("myApp",[]).component("customer", {
    templateUrl : "customer.html", // your view here
    controller: customerController, //your controller here
    controllerAs: "cust"        //alternate name for your controller
})
```

"myApp" es el nombre de la aplicación que estamos creando y cliente es el nombre de nuestro componente. Ahora, para llamarlo en el archivo html principal, lo pondremos así.

```
<customer></customer>
```

Ahora esta directiva será reemplazada por la vista que ha especificado y la lógica de negocios que ha escrito en su controlador.

NOTA: Recuerde que el componente toma un objeto como segundo argumento mientras que la directiva toma una función de fábrica como argumento.

Lea Componentes en línea: https://riptutorial.com/es/angularjs/topic/892/componentes

Capítulo 10: Constantes

Observaciones

MAYÚSCULAS su constante : escribir constante en mayúsculas es una buena práctica común que se usa en muchos idiomas. También es útil identificar claramente la naturaleza de los elementos inyectados:

Cuando vea `.controller('MyController', function($scope, Profile, EVENT))` , instantáneamente sabe que:

- `$scope` es un elemento angular
- `Profile` es un servicio personalizado o fábrica.
- `EVENT` es una constante angular

Examples

Crea tu primera constante

```
angular
  .module('MyApp', [])
  .constant('VERSION', 1.0);
```

Su constante ahora está declarada y se puede inyectar en un controlador, un servicio, una fábrica, un proveedor e incluso en un método de configuración:

```
angular
  .module('MyApp')
  .controller('FooterController', function(VERSION) {
    this.version = VERSION;
  });
```

```
<footer ng-controller="FooterController as Footer">{{ Footer.version }}</footer>
```

Casos de uso

No hay revolución aquí, pero la constante angular puede ser útil especialmente cuando tu aplicación y / o equipo comienza a crecer ... ¡o si simplemente te encanta escribir código hermoso!

- **Código refactor.** Ejemplo con los nombres de los eventos. Si utiliza muchos eventos en su aplicación, tiene los nombres de los eventos un poco en todas partes. A cuando un nuevo desarrollador se une a su equipo, nombra sus eventos con una sintaxis diferente, ... Puede evitar esto fácilmente agrupando los nombres de sus eventos en una constante:

```
angular
  .module('MyApp')
  .constant('EVENTS', {
    LOGIN_VALIDATE_FORM: 'login::click-validate',
    LOGIN_FORGOT_PASSWORD: 'login::click-forgot',
    LOGIN_ERROR: 'login::notify-error',
    ...
  });
```

```
angular
  .module('MyApp')
  .controller('LoginController', function($scope, EVENT) {
    $scope.$on(EVENT.LOGIN_VALIDATE_FORM, function() {
      ...
    });
  })
```

... y ahora, los nombres de su evento pueden beneficiarse del autocompletado!

- **Definir la configuración.** Localiza toda tu configuración en un mismo lugar:

```
angular
  .module('MyApp')
  .constant('CONFIG', {
    BASE_URL: {
      APP: 'http://localhost:3000',
      API: 'http://localhost:3001'
    },
    STORAGE: 'S3',
    ...
  });
```

- **Aislar las piezas.** A veces, hay algunas cosas de las que no estás muy orgulloso ... como el valor codificado, por ejemplo. En lugar de dejarlos en su código principal, puede crear una constante angular

```
angular
  .module('MyApp')
  .constant('HARDCODED', {
    KEY: 'KEY',
    RELATION: 'has_many',
    VAT: 19.6
  });
```

... y refactorizar algo como

```
$scope.settings = {
  username: Profile.username,
  relation: 'has_many',
  vat: 19.6
}
```

a

```
$scope.settings = {
  username: Profile.username,
  relation: HARDCODED.RELATION,
  vat: HARDCODED.VAT
}
```

Lea Constantes en línea: https://riptutorial.com/es/angularjs/topic/3967/constantes

Capítulo 11: Controladores

Sintaxis

- <htmlElement ng-controller = "controllerName"> ... </htmlElement>
- <script> app.controller ('controllerName', controllerFunction); </script>

Examples

Su primer controlador

Un controlador es una estructura básica que se usa en Angular para preservar el alcance y manejar ciertas acciones dentro de una página. Cada controlador está acoplado con una vista HTML.

A continuación se muestra una placa de referencia básica para una aplicación Angular:

```
<!DOCTYPE html>

<html lang="en" ng-app='MyFirstApp'>
    <head>
        <title>My First App</title>

        <!-- angular source -->
        <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>

        <!-- Your custom controller code -->
        <script src="js/controllers.js"></script>
    </head>
    <body>
        <div ng-controller="MyController as mc">
            <h1>{{ mc.title }}</h1>
            <p>{{ mc.description }}</p>
            <button ng-click="mc.clicked()">
                Click Me!
            </button>
        </div>
    </body>
</html>
```

Hay algunas cosas a tener en cuenta aquí:

```
<html ng-app='MyFirstApp'>
```

Establecer el nombre de la aplicación con `ng-app` permite acceder a la aplicación en un archivo Javascript externo, que se tratará a continuación.

```
<script src="js/controllers.js"></script>
```

Necesitaremos un archivo Javascript donde defina sus controladores y sus acciones / datos.

```
<div ng-controller="MyController as mc">
```

El atributo `ng-controller` establece el controlador para ese elemento DOM y todos los elementos que son hijos (recursivamente) debajo de él.

Puede tener múltiples del mismo controlador (en este caso, `MyController`) diciendo `... as mc` , le estamos dando a esta instancia del controlador un alias.

```
<h1>{{ mc.title }}</h1>
```

La notación `{{ ... }}` es una expresión angular. En este caso, esto establecerá el texto interno de ese elemento `<h1>` en cualquiera que sea el valor de `mc.title` .

Nota: Angular emplea enlace de datos de doble vía, lo que significa que, independientemente de cómo actualice el valor de `mc.title` , se reflejará tanto en el controlador como en la página.

También tenga en cuenta que las expresiones angulares *no* tienen que hacer referencia a un controlador. Una expresión Angular puede ser tan simple como `{{ 1 + 2 }}` o `{{ "Hello " + "World" }}` .

```
<button ng-click="mc.clicked()">
```

`ng-click` es una directiva angular, en este caso, vincula el evento click para que el botón active la función `MyController clicked()` de la instancia de `MyController` .

Con esas cosas en mente, escribamos una implementación del controlador `MyController` . Con el ejemplo anterior, escribiría este código en `js/controller.js` .

Primero, deberá crear una instancia de la aplicación Angular en su Javascript.

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

Tenga en cuenta que el nombre que aprobamos aquí es el mismo que el que estableció en su HTML con la directiva `ng-app` .

Ahora que tenemos el objeto de la aplicación, podemos usar eso para crear controladores.

```
app.controller('MyController', function(){
    var ctrl = this;

    ctrl.title = "My First Angular App";
    ctrl.description = "This is my first Angular app!";

    ctrl.clicked = function(){
        alert("MyController.clicked()");
    };
});
```

Nota: Para cualquier cosa que queramos ser parte de la instancia del controlador, usamos `this` palabra clave.

Esto es todo lo que se requiere para construir un controlador simple.

Creando Controladores

```
angular
    .module('app')
    .controller('SampleController', SampleController)

SampleController.$inject = ['$log', '$scope'];
function SampleController($log, $scope){
    $log.debug('*****SampleController******');

    /* Your code below */
}
```

Nota: El `.$inject` asegurará que sus dependencias no sean codificadas después de la minificación. Además, asegúrese de que esté en orden con la función nombrada.

Creando controladores, minificación segura

Hay un par de maneras diferentes para proteger la creación de su controlador de la minificación.

La primera se llama anotación de matriz en línea. Se parece a lo siguiente:

```
var app = angular.module('app');
app.controller('sampleController', ['$scope', '$http', function(a, b){
    //logic here
}]);
```

El segundo parámetro del método del controlador puede aceptar una matriz de dependencias. Como puede ver, he definido `$scope` y `$http` que deberían corresponder a los parámetros de la función del controlador en la que `a` será `$scope` , y `b` sería `$http` . Tenga en cuenta que el último elemento de la matriz debe ser su función de controlador.

La segunda opción es usar la propiedad `$inject` . Se parece a lo siguiente:

```
var app = angular.module('app');
app.controller('sampleController', sampleController);
sampleController.$inject = ['$scope', '$http'];
function sampleController(a, b) {
    //logic here
}
```

Esto hace lo mismo que la anotación de matriz en línea, pero proporciona un estilo diferente para aquellos que prefieren una opción sobre la otra.

El orden de las dependencias inyectadas es importante.

Al inyectar dependencias utilizando la forma de matriz, asegúrese de que la lista de dependencias coincida con la lista correspondiente de los argumentos pasados a la función del controlador.

Tenga en cuenta que en el siguiente ejemplo, `$scope` y `$http` se invierten. Esto causará un problema en el código.

```
// Intentional Bug: injected dependencies are reversed which will cause a problem
app.controller('sampleController', ['$scope', '$http',function($http, $scope) {
    $http.get('sample.json');
}]);
```

Usando ControllerAs en Angular JS

En Angular `$scope` es el pegamento entre el controlador y la vista que ayuda con todas nuestras necesidades de enlace de datos. Controlador Como es otra forma de enlazar el controlador y la vista, y se recomienda su uso en su mayoría. Básicamente, estas son las dos construcciones de controlador en Angular (es decir, $ scope y Controller As).

Diferentes maneras de usar Controller As son -

Sintaxis de ver controladores

```
<div ng-controller="CustomerController as customer">
    {{ customer.name }}
</div>
```

sintaxis del controlador

```
function CustomerController() {
    this.name = {};
    this.sendMessage = function() { };
}
```

controladorAs con vm

```
function CustomerController() {
    /*jshint validthis: true */
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}
```

`controllerAs` es azúcar sintáctica sobre `$scope` . Todavía puede enlazar a los métodos Ver y todavía tener acceso a `$scope` . El uso de `controllerAs` es una de las mejores prácticas sugeridas por el equipo de núcleo angular. Hay muchas razones para esto, algunas de ellas son:

- `$scope` está exponiendo a los miembros del controlador a la vista a través de un objeto intermediario. Al configurar `this.*` , Podemos exponer solo lo que queremos exponer desde el controlador a la vista. También sigue la forma estándar de JavaScript de usar esto.
- utilizando la sintaxis de `controllerAs` , tenemos un código más legible y se puede acceder a

la propiedad principal utilizando el nombre de alias del controlador principal en lugar de usar la sintaxis de `$parent` .

- Promueve el uso del enlace a un objeto "punteado" en la Vista (por ejemplo, customer.name en lugar del nombre), que es más contextual, más fácil de leer y evita cualquier problema de referencia que pueda ocurrir sin "puntos".
- Ayuda a evitar el uso de `$parent` calls en vistas con controladores anidados.
- Use una variable de captura para esto cuando use la sintaxis de `controllerAs` . Elija un nombre de variable consistente como `vm` , que significa ViewModel. Porque, `this` palabra clave es contextual y cuando se usa dentro de una función dentro de un controlador puede cambiar su contexto. Capturar el contexto de esto evita encontrar este problema.

NOTA: el uso de la sintaxis de `controllerAs` agrega a la referencia de alcance actual del controlador actual, por lo que está disponible como campo

```
<div ng-controller="Controller as vm>...</div>
```

`vm` está disponible como `$scope.vm` .

Creación de controladores angulares seguros para minificación

Para crear controladores angulares seguros para la minificación, cambiará los parámetros de la función del `controller` .

El segundo argumento en la función `module.controller` debe pasar una **matriz** , donde el **último parámetro** es la **función** del **controlador** , y cada parámetro anterior es el **nombre** de cada valor inyectado.

Esto es diferente del paradigma normal; que toma la **función de controlador** con los argumentos inyectados.

Dado:

```
var app = angular.module('myApp');
```

El controlador debe tener este aspecto:

```
app.controller('ctrlInject',
    [
        /* Injected Parameters */
        '$Injectable1',
        '$Injectable2',
        /* Controller Function */
        function($injectable1Instance, $injectable2Instance) {
            /* Controller Content */
        }
    ]
);
```

Nota: *No es necesario que los nombres de los parámetros inyectados coincidan, pero se unirán en orden.*

Esto se reducirá a algo similar a esto:

```
var
a=angular.module('myApp');a.controller('ctrlInject',['$Injectable1','$Injectable2',function(b,c){/*
Controller Content */}]);
```

El proceso de minificación reemplazará cada instancia de la `app` con `a` , cada instancia de `$Injectable1Instance` con `b` , y cada instancia de `$Injectable2Instance` con `c` .

Controladores anidados

Los controladores de anidamiento también encadenan el `$scope` . Cambiar una variable de `$scope` en el controlador anidado cambia la misma variable de `$scope` en el controlador principal.

```
.controller('parentController', function ($scope) {
    $scope.parentVariable = "I'm the parent";
});

.controller('childController', function ($scope) {
    $scope.childVariable = "I'm the child";

    $scope.childFunction = function () {
        $scope.parentVariable = "I'm overriding you";
    };
});
```

Ahora tratemos de manejar a los dos, anidados.

```
<body ng-controller="parentController">
    What controller am I? {{parentVariable}}
    <div ng-controller="childController">
        What controller am I? {{childVariable}}
        <button ng-click="childFunction()"> Click me to override! </button>
    </div>
</body>
```

Los controladores de anidamiento pueden tener sus beneficios, pero una cosa debe tenerse en cuenta al hacerlo. Llamar a la directiva `ngController` crea una nueva instancia del controlador, que a menudo puede crear confusión y resultados inesperados.

Lea Controladores en línea: https://riptutorial.com/es/angularjs/topic/601/controladores

Capítulo 12: Controladores con ES6

Examples

Controlador

es muy fácil escribir un controlador angularJS con ES6 si está familiarizado con la **programación orientada a objetos** :

```
class exampleContoller{

       constructor(service1,service2,...serviceN){
          let ctrl=this;
              ctrl.service1=service1;
              ctrl.service2=service2;
               .
               .
               .
              ctrl.service1=service1;
              ctrl.controllerName = 'Example Controller';
              ctrl.method1(controllerName)

       }

       method1(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
              ctrl.scopeName=param;
       }
       .
       .
       .
       methodN(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
       }


   }
   exampleContoller.$inject = ['service1','service2',...,'serviceN'];
   export default exampleContoller;
```

Lea Controladores con ES6 en línea: https://riptutorial.com/es/angularjs/topic/9419/controladores-con-es6

Capítulo 13: Decoradores

Sintaxis

- decorador (nombre, decorador);

Observaciones

> **Decorador** es una función que permite modificar un servicio , una fábrica , una directiva o un filtro antes de su uso. Decorator se utiliza para anular o modificar el comportamiento del servicio. El valor de retorno de la función de decoración puede ser el servicio original o un nuevo servicio que reemplaza, o envuelve y delega, el servicio original.

Cualquier decoración ***debe*** realizarse en la fase de `config` la aplicación angular mediante la inyección de `$provide` y utilizando su función `$provide.decorator` .

> La función de decorador tiene un objeto `$delegate` inyectado para proporcionar acceso al servicio que coincide con el selector en el decorador. Este `$delegate` será el servicio que estás decorando. El valor de retorno de la función proporcionada al decorador tendrá lugar en el servicio, directiva o filtro que se esté decorando.

Uno debería considerar usar el decorador solo si cualquier otro enfoque no es apropiado o resulta ser demasiado tedioso. Si la aplicación grande usa el mismo servicio y una parte cambia el comportamiento del servicio, es fácil crear confusión y / o errores en el proceso.

El caso de uso típico sería cuando tiene una dependencia de terceros que no puede actualizar, pero necesita que funcione de manera diferente o que la extienda.

Examples

Servicio de decoración, fábrica.

A continuación se muestra un ejemplo de decorador de servicios, anulando la fecha `null` devuelta por el servicio.

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myService', function($delegate) {
      $delegate.getDate = function() { // override with actual date object
        return new Date();
      };
      return $delegate;
    });
  })
```

```
  .service('myService', function() {
    this.getDate = function() {
      return null; // w/o decoration we'll be returning null
    };
  })
  .controller('myController', function(myService) {
    var vm = this;
    vm.date = myService.getDate();
  });
```

```
<body ng-controller="myController as vm">
  <div ng-bind="vm.date | date:'fullDate'"></div>
</body>
```

> Saturday, August 6, 2016

Decorar directiva

Las directivas se pueden decorar como servicios y podemos modificar o reemplazar cualquiera de sus funciones. Tenga en cuenta que se accede a la directiva en la posición 0 en $ matriz de delegado y el parámetro de nombre en el decorador debe incluir el sufijo de la `Directive` (distingue entre mayúsculas y minúsculas).

Por lo tanto, si la directiva se llama `myDate` , se puede acceder a ella usando `myDateDirective` usando `$delegate[0]` .

A continuación se muestra un ejemplo simple donde la directiva muestra la hora actual. Lo decoraremos para actualizar la hora actual en intervalos de un segundo. Sin decoración siempre se mostrará el mismo tiempo.

```
<body>
  <my-date></my-date>
</body>
```

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myDateDirective', function($delegate, $interval) {
      var directive = $delegate[0]; // access directive

      directive.compile = function() { // modify compile fn
        return function(scope) {
          directive.link.apply(this, arguments);
          $interval(function() {
            scope.date = new Date(); // update date every second
          }, 1000);
        };
      };

      return $delegate;
    });
  })
```

```
.directive('myDate', function() {
  return {
    restrict: 'E',
    template: '<span>Current time is {{ date | date:\'MM:ss\' }}</span>',
    link: function(scope) {
      scope.date = new Date(); // get current date
    }
  };
});
```

> Current time is 08:33

Decorar filtro

Al decorar filtros, el parámetro de nombre debe incluir el sufijo de `Filter` (distingue entre mayúsculas y minúsculas). Si el filtro se llama `repeat` , el parámetro decorador es `repeatFilter` . A continuación, decoraremos el filtro personalizado que repite cualquier cadena dada *n* veces para que el resultado se invierta. También puede decorar los filtros incorporados de angular de la misma forma, aunque no se recomienda ya que puede afectar la funcionalidad del marco.

```
<body>
  <div ng-bind="'i can haz cheeseburger ' | repeat:2"></div>
</body>

angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('repeatFilter', function($delegate) {
      return function reverse(input, count) {
        // reverse repeated string
        return ($delegate(input, count)).split('').reverse().join('');
      };
    });
  })
  .filter('repeat', function() {
    return function(input, count) {
      // repeat string n times
      return (input || '').repeat(count || 1);
    };
  });
```

> i can haz cheeseburger i can haz cheeseburger

> regrubeseehc zah nac i regrubeseehc zah nac i

Lea Decoradores en línea: https://riptutorial.com/es/angularjs/topic/5255/decoradores

Capítulo 14: Depuración

Examples

Depuración básica en el marcado.

Prueba de alcance y salida del modelo

```
<div ng-app="demoApp" ng-controller="mainController as ctrl">
    {{$id}}
    <ul>
        <li ng-repeat="item in ctrl.items">
            {{$id}}<br/>
            {{item.text}}
        </li>
    </ul>
    {{$id}}
    <pre>
       {{ctrl.items | json : 2}}
    </pre>
</div>


angular.module('demoApp', [])
.controller('mainController', MainController);

function MainController() {
    var vm = this;
    vm.items = [{
        id: 0,
        text: 'first'
    },
    {
        id: 1,
        text: 'second'
    },
    {
        id: 2,
        text: 'third'
    }];
}
```

A veces puede ayudar a ver si hay un nuevo alcance para solucionar los problemas de alcance. `$scope.$id` puede usarse en una expresión en cualquier parte de su marca para ver si hay un nuevo $ scope.

En el ejemplo, puede ver que fuera de la etiqueta ul es el mismo ámbito ($ id = 2) y dentro de la `ng-repeat` hay nuevos ámbitos secundarios para cada iteración.

Una salida del modelo en una etiqueta previa es útil para ver los datos actuales de su modelo. El filtro `json` crea una salida con formato de aspecto agradable. La etiqueta previa se usa porque dentro de esa etiqueta se mostrará correctamente cualquier carácter de línea nueva `\n` .

manifestación

Usando ng-inspect chrome extension

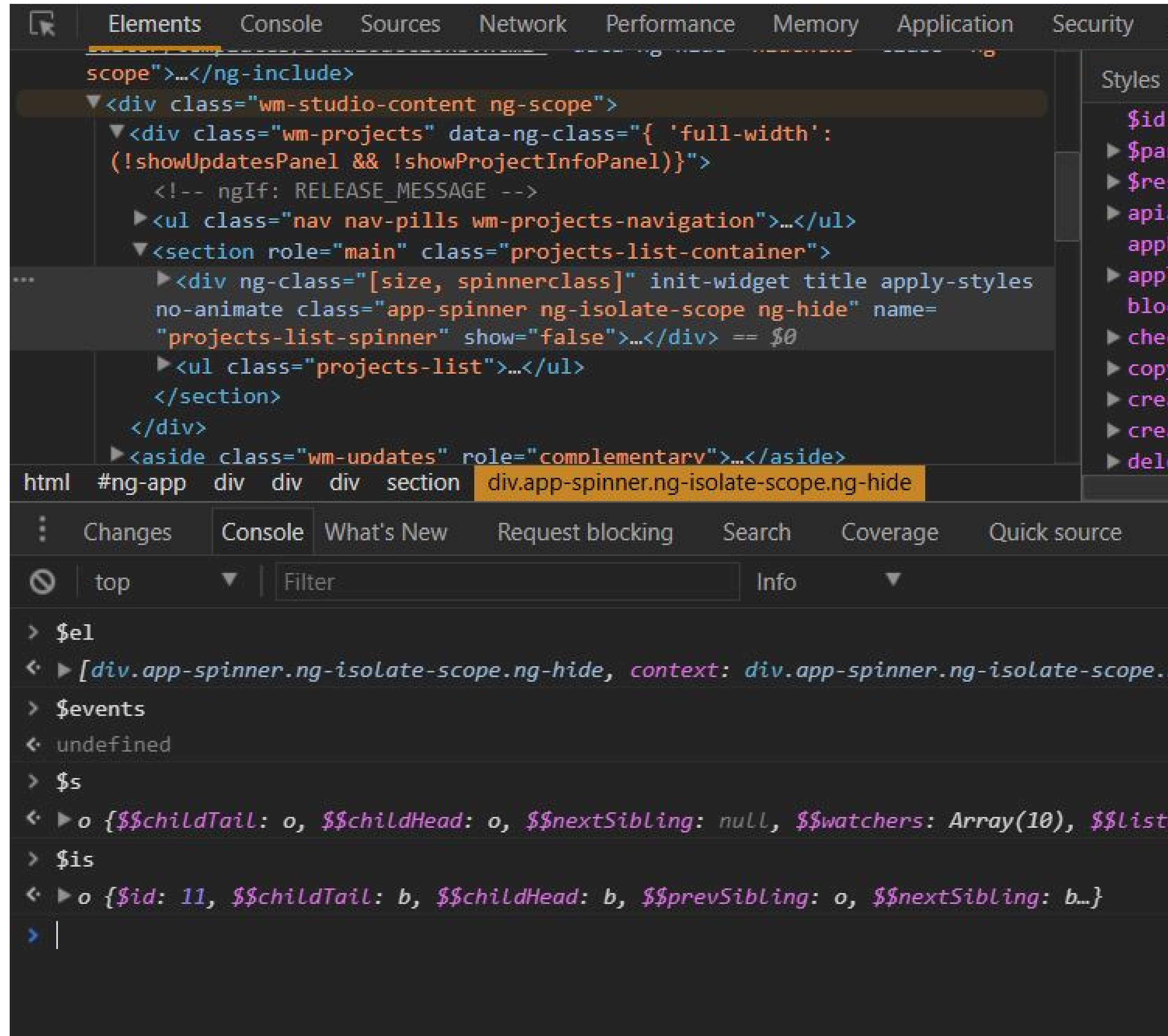
ng-inspect es una extensión ligera de Chrome para depurar aplicaciones AngularJS.

Cuando se selecciona un nodo en el panel de elementos, la información relacionada con el alcance se muestra en el panel de inspección ng.

Expone pocas variables globales para un acceso rápido de `scope/isolateScope` .

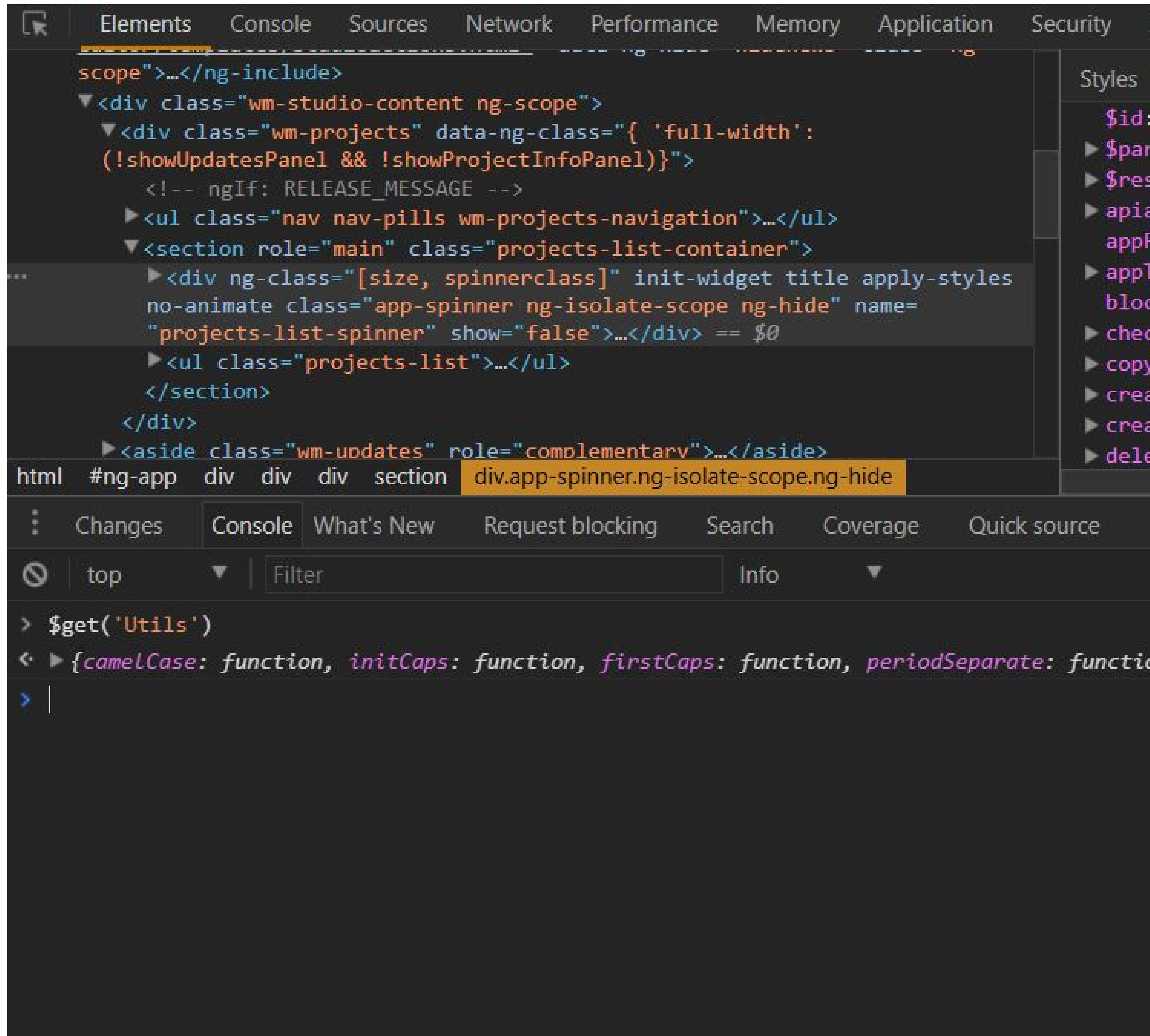
```
$s      -- scope of the selected node
$is     -- isolateScope of the selected node
$el     -- jQuery element reference of the selected node (requiers jQuery)
```

```
$events -- events present on the selected node (requires jQuery)
```



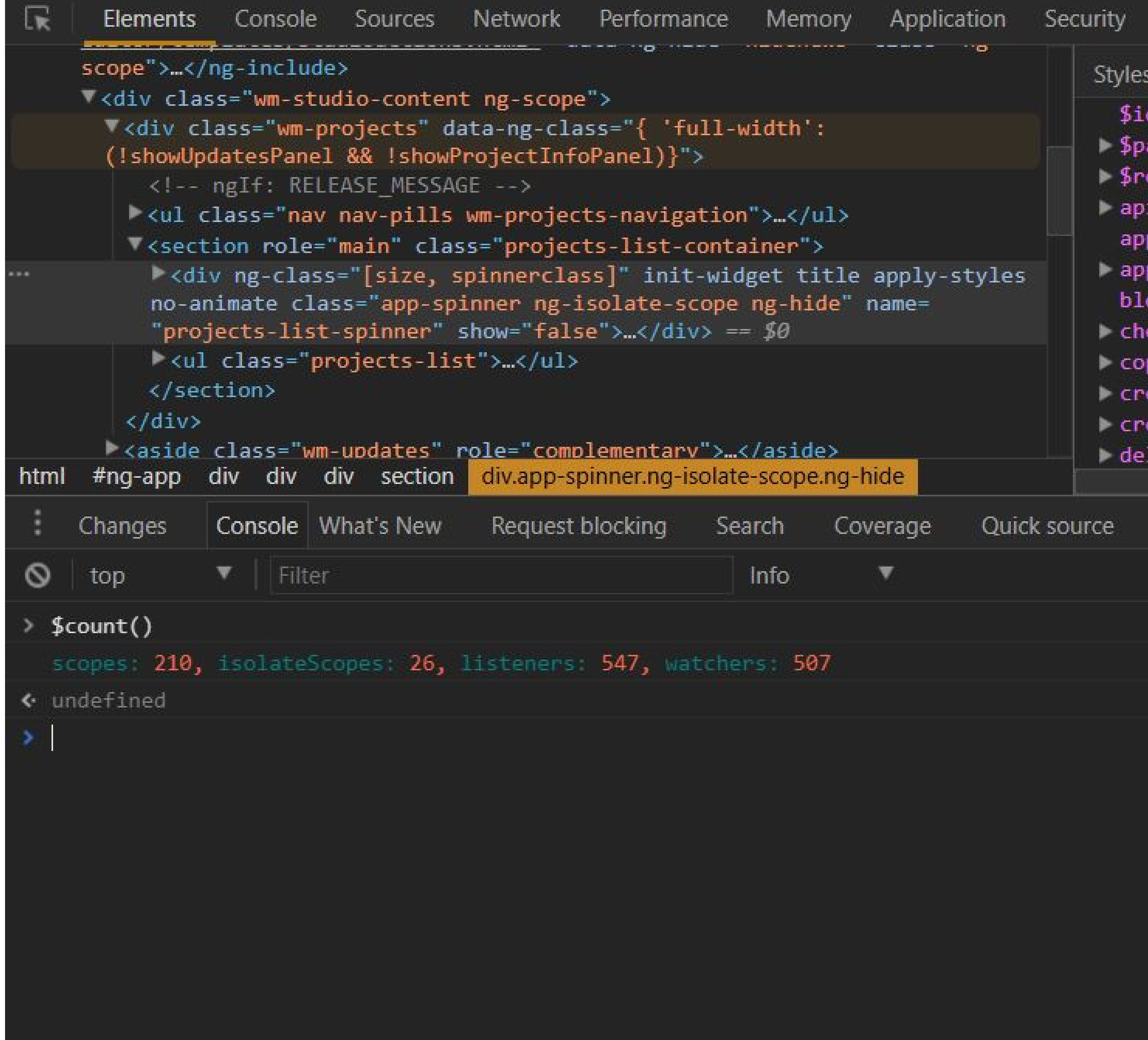
Proporciona fácil acceso a Servicios / Fábricas.

Use `$get()` para recuperar la instancia de un servicio / fábrica por nombre.



El rendimiento de la aplicación se puede monitorear contando el no.de ámbitos, aislar los microscopios, observadores y oyentes de la aplicación.

Use `$count()` para obtener el recuento de ámbitos, aislar microscopios, observadores y oyentes.



Nota: esta extensión solo funcionará cuando debugInfo esté habilitado.

Descargar ng-inspect aquí

Consiguiendo el alcance del elemento

En una aplicación angular, todo va alrededor del alcance, si pudiéramos obtener un alcance de elementos, entonces es fácil depurar la aplicación angular. Cómo acceder al alcance del elemento:

```
angular.element(myDomElement).scope();
e.g.
angular.element(document.getElementById('yourElementId')).scope()  //accessing by ID
```

Conseguir el alcance del controlador: -

```
angular.element('[ng-controller=ctrl]').scope()
```

Otra forma fácil de acceder a un elemento DOM desde la consola (como se mencionó jm) es haciendo clic en él en la pestaña 'elementos', y se almacena automáticamente como $ 0.

```
angular.element($0).scope();
```

Lea Depuración en línea: https://riptutorial.com/es/angularjs/topic/4761/depuracion

Capítulo 15: directiva de clase ng

Examples

Tres tipos de expresiones de clase ng

Angular admite tres tipos de expresiones en la directiva `ng-class` .

1. cuerda

```
<span ng-class="MyClass">Sample Text</span>
```

Especificar una expresión que se evalúa como una cadena le dice a Angular que la trate como una variable de $ scope. Angular verificará $ scope y buscará una variable llamada "MyClass". Cualquiera que sea el texto contenido en "MyClass" se convertirá en el nombre real de la clase que se aplica a este `<span>` . Puede especificar varias clases separando cada clase con un espacio.

En su controlador, puede tener una definición que se parece a esto:

```
$scope.MyClass = "bold-red deleted error";
```

Angular evaluará la expresión `MyClass` y encontrará la definición de $ scope. Aplicará las tres clases "rojo negrita", "borrado" y "error" al elemento `<span>` .

Especificar clases de esta manera le permite cambiar fácilmente las definiciones de clase en su controlador. Por ejemplo, es posible que deba cambiar la clase según las interacciones de otros usuarios o los nuevos datos que se cargan desde el servidor. Además, si tiene muchas expresiones para evaluar, puede hacerlo en una función que define la lista final de clases en una variable de `$scope` . Esto puede ser más fácil que intentar incluir muchas evaluaciones en el atributo `ng-class` en su plantilla HTML.

---

2. Objeto

Esta es la forma más comúnmente utilizada de definir clases usando `ng-class` porque le permite especificar evaluaciones que determinan qué clase usar.

Especifique un objeto que contiene pares clave-valor. La clave es el nombre de la clase que se aplicará si el valor (un condicional) se evalúa como verdadero.

```
<style>
    .red { color: red; font-weight: bold; }
```

```
    .blue { color: blue; }
    .green { color: green; }
    .highlighted { background-color: yellow; color: black; }
</style>

<span ng-class="{ red: ShowRed, blue: ShowBlue, green: ShowGreen, highlighted: IsHighlighted
}">Sample Text</span>

<div>Red: <input type="checkbox" ng-model="ShowRed"></div>
<div>Green: <input type="checkbox" ng-model="ShowGreen"></div>
<div>Blue: <input type="checkbox" ng-model="ShowBlue"></div>
<div>Highlight: <input type="checkbox" ng-model="IsHighlighted"></div>
```

---

3. Array

Una expresión que se evalúa como una matriz le permite usar una combinación de **cadenas** (vea el número 1 arriba) y los **objetos condicionales** (número 2 arriba).

```
<style>
    .bold { font-weight: bold; }
    .strike { text-decoration: line-through; }
    .orange { color: orange; }
</style>

<p ng-class="[ UserStyle, {orange: warning} ]">Array of Both Expression Types</p>
<input ng-model="UserStyle" placeholder="Type 'bold' and/or 'strike'"><br>
<label><input type="checkbox" ng-model="warning"> warning (apply "orange" class)</label>
```

Esto crea un campo de entrada de texto vinculado a la variable de alcance `UserStyle` que permite al usuario escribir cualquier nombre de clase. Estos se aplicarán dinámicamente al elemento `<p>` según los tipos de usuario. Además, el usuario puede hacer clic en la casilla de verificación que está vinculada a la variable de alcance de `warning` . Esto también se aplicará dinámicamente al elemento `<p>` .

Lea directiva de clase ng en línea: https://riptutorial.com/es/angularjs/topic/2395/directiva-de-clase-ng

Capítulo 16: Directivas incorporadas

Examples

Expresiones angulares - Texto vs. Número

Este ejemplo demuestra cómo se evalúan las expresiones angulares cuando se usa `type="text"` y `type="number"` para el elemento de entrada. Considere el siguiente controlador y vea:

Controlador

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

app.controller('ctrl', function($scope) {
    $scope.textInput = {
        value: '5'
    };
    $scope.numberInput = {
        value: 5
    };
});
```

Ver

```
<div ng-app="app" ng-controller="ctrl">
    <input type="text" ng-model="textInput.value">
    {{ textInput.value + 5 }}
    <input type="number" ng-model="numberInput.value">
    {{ numberInput.value + 5 }}
</div>
```

- Cuando se usa `+` en una expresión vinculada al ingreso de *texto* , el operador **concatenará** las cadenas (primer ejemplo), mostrando 55 en la pantalla `*` .
- Cuando se usa `+` en una expresión vinculada a la entrada de *números* , el operador devuelve la **suma** de los números (segundo ejemplo), mostrando 10 en la pantalla `*` .

`*` - Eso es hasta que el usuario cambia el valor en el campo de entrada, luego la pantalla cambiará en consecuencia.

Ejemplo de trabajo

ngRepetir

`ng-repeat` es una directiva integrada en Angular que le permite iterar una matriz o un objeto y le permite repetir un elemento una vez para cada elemento de la colección.

ng-repetir una matriz

```
<ul>
```

```
    <li ng-repeat="item in itemCollection">
       {{item.Name}}
    </li>
</ul>
```

Dónde:
item = **item** individual en la colección
itemCollection = La matriz que está iterando

ng-repetir un objeto

```
<ul>
    <li ng-repeat="(key, value) in myObject">
       {{key}} : {{value}}
    </li>
</ul>
```

Dónde:
clave = el nombre de la propiedad
valor = el valor de la propiedad
myObject = el objeto que estás iterando

filtrar su ng-repetir por la entrada del usuario

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText">
       {{string}}
    </li>
</ul>
```

Dónde:
searchText = el texto por el cual el usuario quiere filtrar la lista
stringArray = una matriz de cadenas, por ejemplo, `['string', 'array']`

También puede mostrar o hacer referencia a los elementos filtrados en otro lugar asignando un alias de salida de filtro `as aliasName` , de este modo:

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText as filteredStrings">
       {{string}}
    </li>
</ul>
<p>There are {{filteredStrings.length}} matching results</p>
```

ng-repeat-start y ng-repeat-end

Para repetir varios elementos de DOM definiendo un inicio y un punto final, puede usar las directivas `ng-repeat-start` y `ng-repeat-end` .

```
<ul>
    <li ng-repeat-start="item in [{a: 1, b: 2}, {a: 3, b:4}]">
        {{item.a}}
    </li>
    <li ng-repeat-end>
        {{item.b}}
    </li>
</ul>
```

Salida:

- 1
- 2
- 3
- 4

Es importante cerrar siempre `ng-repeat-start` con `ng-repeat-end` .

Variables

`ng-repeat` también expone estas variables dentro de la expresión

Variable	Tipo	Detalles
`$index`	Número	Es igual al índice de la iteración actual ($ index === 0 se evaluará como verdadero en el primer elemento iterado; vea `$first`)
`$first`	Booleano	Evalúa a verdadero en el primer elemento iterado
`$last`	Booleano	Evalúa a verdadero en el último elemento iterado
`$middle`	Booleano	Se evalúa como verdadero si el elemento se encuentra entre `$first` y `$last`
`$even`	Booleano	Se evalúa como verdadero en una iteración de número par (equivalente a `$index%2===0`)
`$odd`	Booleano	Se evalúa como verdadero en una iteración de número impar (equivalente a `$index%2===1`)

Consideraciones de rendimiento

La representación de `ngRepeat` puede ser lenta, especialmente cuando se usan colecciones grandes.

Si los objetos de la colección tienen una propiedad de identificador, siempre debe `track by` el

identificador en lugar de todo el objeto, que es la funcionalidad predeterminada. Si no hay ningún identificador presente, siempre puede usar el `$index` incorporado.

```
<div ng-repeat="item in itemCollection track by item.id">
<div ng-repeat="item in itemCollection track by $index">
```

Alcance de ngRepeat

`ngRepeat` siempre creará un ámbito secundario aislado, por lo que se debe tener cuidado si se debe acceder al ámbito principal dentro de la repetición.

Este es un ejemplo simple que muestra cómo puede establecer un valor en su ámbito principal desde un evento de clic dentro de `ngRepeat` .

```
scope val:  {{val}}<br/>
ctrlAs val: {{ctrl.val}}
<ul>
    <li ng-repeat="item in itemCollection">
        <a href="#" ng-click="$parent.val=item.value; ctrl.val=item.value;">
            {{item.label}} {{item.value}}
        </a>
    </li>
</ul>

$scope.val = 0;
this.val = 0;

$scope.itemCollection = [{
    id: 0,
    value: 4.99,
    label: 'Football'
},
{
    id: 1,
    value: 6.99,
    label: 'Baseball'
},
{
    id: 2,
    value: 9.99,
    label: 'Basketball'
}];
```

Si solo había `val = item.value` en `ng-click` , no se actualizará el `val` en el ámbito principal debido al ámbito aislado. Es por eso que se accede al alcance principal con `$parent` referencia de `$parent` o con la sintaxis de `controllerAs` (por ejemplo, `ng-controller="mainController as ctrl"`).

Anidado ng-repetir

También puede utilizar ng-nest anidado.

```
<div ng-repeat="values in test">
    <div ng-repeat="i in values">
      [{{$parent.$index}},{{$index}}] {{i}}
    </div>
```

```
</div>

var app = angular.module("myApp", []);
app.controller("ctrl", function($scope) {
  $scope.test = [
    ['a', 'b', 'c'],
    ['d', 'e', 'f']
  ];
});
```

Aquí para acceder al índice de padre ng-repeat dentro de hijo ng-repeat, puede usar `$parent.$index` .

ngShow y ngOcultar

La directiva `ng-show` muestra u oculta el elemento HTML en función de si la expresión que se le pasa es verdadera o falsa. Si el valor de la expresión es falsy, entonces se ocultará. Si es veraz entonces se mostrará.

La directiva `ng-hide` es similar. Sin embargo, si el valor es falsy, mostrará el elemento HTML. Cuando la expresión sea veraz la esconderá.

Ejemplo de JSBin de trabajo

Controlador :

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

angular.module('app')
  .controller('ExampleController', ExampleController);

function ExampleController() {

  var vm = this;

  //Binding the username to HTML element
  vm.username = '';

  //A taken username
  vm.taken_username = 'StackOverflow';

}
```

Ver

```
<section ng-controller="ExampleController as main">

    <p>Enter Password</p>
    <input ng-model="main.username" type="text">

    <hr>

    <!-- Will always show as long as StackOverflow is not typed in -->
    <!-- The expression is always true when it is not StackOverflow -->
    <div style="color:green;" ng-show="main.username != main.taken_username">
```

```
      Your username is free to use!
    </div>

    <!-- Will only show when StackOverflow is typed in -->
    <!-- The expression value becomes falsy -->
    <div style="color:red;" ng-hide="main.username != main.taken_username">
      Your username is taken!
    </div>

    <p>Enter 'StackOverflow' in username field to show ngHide directive.</p>

 </section>
```

ngOpciones

`ngOptions` es una directiva que simplifica la creación de un cuadro desplegable html para la selección de un elemento de una matriz que se almacenará en un modelo. El atributo ngOptions se usa para generar dinámicamente una lista de elementos `<option>` para el elemento `<select>` usando la matriz u objeto obtenido al evaluar la expresión de comprensión ngOptions.

Con `ng-options` el marcado se puede reducir a solo una etiqueta de selección y la directiva creará la misma selección:

```
<select ng-model="selectedFruitNgOptions"
        ng-options="curFruit as curFruit.label for curFruit in fruit">
</select>
```

Hay otra forma de crear opciones de `select` usando `ng-repeat` , pero no se recomienda usar `ng-repeat` ya que se usa principalmente para fines generales como, por ejemplo, `forEach` solo para hacer un bucle. Mientras que `ng-options` es específicamente para crear opciones de etiqueta `select` .

El ejemplo anterior usando `ng-repeat` sería

```
<select ng-model="selectedFruit">
    <option ng-repeat="curFruit in fruit" value="{{curFruit}}">
        {{curFruit.label}}
    </option>
</select>
```

EJEMPLO COMPLETO

Veamos el ejemplo anterior en detalle también con algunas variaciones en él.

Modelo de datos para el ejemplo:

```
$scope.fruit = [
    { label: "Apples", value: 4, id: 2 },
    { label: "Oranges", value: 2, id: 1 },
    { label: "Limes", value: 4, id: 4 },
    { label: "Lemons", value: 5, id: 3 }
];
```

```
<!-- label for value in array -->
<select ng-options="f.label for f in fruit" ng-model="selectedFruit"></select>
```

Etiqueta de opción generada en la selección:

```
 <option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Efectos:

`f.label` será la etiqueta de la `<option>` y el valor contendrá todo el objeto.

EJEMPLO COMPLETO

```
<!-- select as label for value in array -->
<select ng-options="f.value as f.label for f in fruit" ng-model="selectedFruit"></select>
```

Etiqueta de opción generada en la selección:

```
 <option value="4"> Apples </option>
```

Efectos:

`f.value` (4) será el valor en este caso mientras la etiqueta siga siendo la misma.

EJEMPLO COMPLETO

```
<!-- label group by group for value in array -->
<select ng-options="f.label group by f.value for f in fruit" ng-
model="selectedFruit"></select>
```

Etiqueta de opción generada en la selección:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Efectos:

Las opciones se agruparán en función de su `value` . Las opciones con el mismo `value` caerán en una categoría

EJEMPLO COMPLETO

```
<!-- label disable when disable for value in array -->
<select ng-options="f.label disable when f.value == 4 for f in fruit" ng-
model="selectedFruit"></select>
```

Etiqueta de opción generada en la selección:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Efectos:

"Manzanas" y "Limas" se deshabilitarán (no se puede seleccionar) debido a la condición `disable when f.value==4` . Todas las opciones con `value=4` serán deshabilitadas

EJEMPLO COMPLETO

```
<!-- label group by group for value in array track by trackexpr -->
<select ng-options="f.value as f.label group by f.value for f in fruit track by f.id" ng-model="selectedFruit"></select>
```

Etiqueta de opción generada en la selección:

```
<option value="4"> Apples </option>
```

Efectos:

No hay cambios visuales cuando se utiliza `trackBy` , pero Angular detectará los cambios por `id` lugar de por referencia, lo que siempre es una solución mejor.

EJEMPLO COMPLETO

```
<!-- label for value in array | orderBy:orderexpr track by trackexpr -->
<select ng-options="f.label for f in fruit | orderBy:'id' track by f.id" ng-model="selectedFruit"></select>
```

Etiqueta de opción generada en la selección:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Efectos:

`orderBy` es un filtro estándar de AngularJS que organiza las opciones en orden ascendente (por defecto), por lo que "Naranjas" aparecerá en primer lugar desde su `id` = 1.

EJEMPLO COMPLETO

Todos los `<select>` con `ng-options` deben tener adjunto `ng-model` .

ngModel

Con ng-model puedes vincular una variable a cualquier tipo de campo de entrada. Puede mostrar la variable utilizando llaves dobles, por ejemplo, `{{myAge}}` .

```
<input type="text" ng-model="myName">
```

```
<p>{{myName}}</p>
```

A medida que ingresa el campo de entrada o lo modifica de alguna manera, verá el valor en la actualización del párrafo al instante.

La variable ng-model, en este caso, estará disponible en su controlador como `$scope.myName` . Si está utilizando la sintaxis de `controllerAs` :

```
<div ng-controller="myCtrl as mc">
    <input type="text" ng-model="mc.myName">
    <p>{{mc.myName}}</p>
</div>
```

Necesitará consultar el alcance del controlador antes de que el alias del controlador definido en el atributo ng-controller a la variable ng-model. De esta manera, no necesitará inyectar `$scope` en su controlador para hacer referencia a su variable ng-model, la variable estará disponible como `this.myName` dentro de la función de su controlador.

clase ng

Supongamos que necesita mostrar el estado de un usuario y que tiene varias clases posibles de CSS que podrían usarse. Angular hace que sea muy fácil elegir de una lista de varias clases posibles que le permiten especificar una lista de objetos que incluye condicionales. Angular es capaz de usar la clase correcta basada en la veracidad de los condicionales.

Su objeto debe contener pares clave / valor. La clave es un nombre de clase que se aplicará cuando el valor (condicional) se evalúe como verdadero.

```
<style>
    .active { background-color: green; color: white; }
    .inactive { background-color: gray; color: white; }
    .adminUser { font-weight: bold; color: yellow; }
    .regularUser { color: white; }
</style>

<span ng-class="{
    active: user.active,
    inactive: !user.active,
    adminUser: user.level === 1,
    regularUser: user.level === 2
}">John Smith</span>
```

Angular verificará el objeto `$scope.user` para ver el estado `active` y el número de `level` . Dependiendo de los valores en esas variables, Angular aplicará el estilo correspondiente a `<span>` .

ngIf

[`ng-if`] es una directiva similar a [`ng-show`] pero inserta o elimina el elemento del DOM en lugar de simplemente ocultarlo. Angular 1.1.5 introdujo la directiva ng-if. Puede usar la directiva ng-if por encima de las versiones 1.1.5. Esto es útil porque Angular no procesará los resúmenes de los

elementos dentro de un `ng-if` eliminado `ng-if` reduce la carga de trabajo de Angular, especialmente para los enlaces de datos complejos.

A diferencia de `ng-show` , la directiva `ng-if` crea un ámbito secundario que usa herencia prototípica. Esto significa que establecer un valor primitivo en el ámbito secundario no se aplicará al padre. Para establecer una primitiva en el ámbito `$parent` propiedad `$parent` en el ámbito secundario deberá utilizarse.

JavaScript

```
angular.module('MyApp', []);

angular.module('MyApp').controller('myController', ['$scope', '$window', function
myController($scope, $window) {
    $scope.currentUser= $window.localStorage.getItem('userName');
}]);
```

Ver

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

DOM si `currentUser` no está indefinido

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <!-- ng-if: !currentUser -->
</div>
```

DOM Si `currentUser` está `currentUser`

```
<div ng-controller="myController">
    <!-- ng-if: currentUser -->
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
```

```
</div>
```

Ejemplo de trabajo

Promesa de la función

La directiva ngIf también acepta funciones, que lógicamente requieren devolver verdadero o falso.

```
<div ng-if="myFunction()">
    <span>Span text</span>
</div>
```

El texto de intervalo solo aparecerá si la función devuelve verdadero.

```
$scope.myFunction = function() {
    var result = false;
    // Code to determine the boolean value of result
    return result;
};
```

Como cualquier expresión angular, la función acepta cualquier tipo de variables.

ngMouseenter y ngMouseleave

Las `ng-mouseenter` y `ng-mouseleave` son útiles para ejecutar eventos y aplicar el estilo CSS cuando se desplaza dentro o fuera de sus elementos DOM.

La directiva `ng-mouseenter` ejecuta una expresión en un evento de ingreso de mouse (cuando el usuario ingresa el puntero del mouse sobre el elemento DOM en el que reside esta directiva)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-class="{'active': applyStyle}">
```

En el ejemplo anterior, cuando el usuario apunta su mouse sobre el `div` , `applyStyle` cambia a `true` , que a su vez aplica la clase `.active` CSS en la `ng-class` .

La directiva `ng-mouseleave` ejecuta una expresión y un evento de salida del mouse (cuando el usuario retira el cursor del mouse del elemento DOM en el que reside esta directiva)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-mouseleaver="applyStyle = false" ng-
class="{'active': applyStyle}">
```

Reutilizando el primer ejemplo, ahora cuando el usuario le quita el puntero del mouse de la `.active` , la clase `.active` se elimina.

deshabilitado

Esta directiva es útil para limitar los eventos de entrada basados en ciertas condiciones existentes.

La directiva `ng-disabled` acepta una expresión que debe evaluar valores verdaderos o falsos.

`ng-disabled` se utiliza para aplicar condicionalmente el atributo `disabled` en un elemento de `input` .

HTML

```
<input type="text" ng-model="vm.name">

<button ng-disabled="vm.name.length===0" ng-click="vm.submitMe">Submit</button>
```

`vm.name.length===0` se evalúa como verdadero si la longitud de la `input` es 0, que a su vez desactiva el botón, impidiendo que el usuario dispare el evento click de `ng-click`

ngDblclick

La directiva `ng-dblclick` es útil cuando desea vincular un evento de doble clic en sus elementos DOM.

Esta directiva acepta una expresión.

HTML

```
<input type="number" ng-model="num = num + 1" ng-init="num=0">

<button ng-dblclick="num++">Double click me</button>
```

En el ejemplo anterior, el valor retenido en la `input` se incrementará cuando se haga doble clic en el botón.

Hoja de trucos de directivas incorporadas

`ng-app` Establece la sección AngularJS.

`ng-init` Establece un valor variable predeterminado.

`ng-bind` Alternativa a la plantilla {{}}.

`ng-bind-template` Enlaza varias expresiones a la vista.

`ng-non-bindable` que los datos no son vinculables.

`ng-bind-html` Enlaza la propiedad HTML interna de un elemento HTML.

`ng-change` Evalúa la expresión especificada cuando el usuario cambia la entrada.

`ng-checked` Establece la casilla de verificación.

`ng-class` Establece la clase css dinámicamente.

`ng-cloak` Evita mostrar el contenido hasta que AngularJS tome el control.

`ng-click` Ejecuta un método o expresión cuando se hace clic en el elemento.

`ng-controller` Asocia una clase de controlador a la vista.

`ng-disabled` Controla la propiedad deshabilitada del elemento de formulario.

`ng-form` Establece un formulario

`ng-href` Vincule dinámicamente las variables AngularJS al atributo href.

`ng-include` Se utiliza para recuperar, compilar e incluir un fragmento de HTML externo a su página.

`ng-if` Quita o recrea un elemento en el DOM dependiendo de una expresión

`ng-switch` condicionalmente el control en función de la expresión correspondiente.

`ng-model` Enlaza elementos de entrada, selección, área de texto, etc. con propiedad de modelo.

`ng-readonly` Se utiliza para establecer el atributo readonly en un elemento.

`ng-repeat` Se utiliza para recorrer cada elemento de una colección para crear una nueva plantilla.

`ng-selected` Se utiliza para configurar la opción seleccionada en el elemento.

`ng-show/ng-hide` Muestra / `ng-show/ng-hide` elementos basados en una expresión.

`ng-src` Vincular dinámicamente las variables AngularJS al atributo src.

`ng-submit` Enlazar expresiones angulares a eventos de envío.

`ng-value` Enlazar expresiones angulares al valor de.

`ng-required` Enlazar expresiones angulares a eventos de envío.

`ng-style` Establece el estilo CSS en un elemento HTML.

`ng-pattern` Agrega el validador de patrones a ngModel.

`ng-maxlength` Agrega el validador de maxlength a ngModel.

`ng-minlength` Agrega el validador de minlength a ngModel.

`ng-classeven` Funciona en conjunto con ngRepeat y surte efecto solo en filas impares (pares).

`ng-classodd` Funciona junto con ngRepeat y surte efecto solo en filas impares (pares).

`ng-cut` Se utiliza para especificar el comportamiento personalizado en el evento de corte.

`ng-copy` Se utiliza para especificar el comportamiento personalizado en el evento de copia.

`ng-paste` Se utiliza para especificar el comportamiento personalizado en el evento de pegado.

`ng-options` Se utiliza para generar dinámicamente una lista de elementos para el elemento.

`ng-list` Se utiliza para convertir una cadena en una lista en función del delimitador especificado.

`ng-open` Se utiliza para establecer el atributo abierto en el elemento, si la expresión dentro de ngOpen es veraz.

Fuente (editado un poco)

Haga clic en

La directiva `ng-click` adjunta un evento de clic a un elemento DOM.

La directiva `ng-click` permite especificar un comportamiento personalizado cuando se hace clic en un elemento de DOM.

Es útil cuando desea adjuntar eventos de clic en los botones y manejarlos en su controlador.

Esta directiva acepta una expresión con el objeto de eventos disponible como `$event`

HTML

```
<input ng-click="onClick($event)">Click me</input>
```

Controlador

```
.controller("ctrl", function($scope) {
    $scope.onClick = function(evt) {
        console.debug("Hello click event: %o ",evt);
    }
})
```

HTML

```
<button ng-click="count = count + 1" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

HTML

```
<button ng-click="count()" ng-init="count=0">
  Increment
```

```
</button>
<span>
  count: {{count}}
</span>
```

Controlador

```
...

$scope.count = function(){
    $scope.count = $scope.count + 1;
}
...
```

Cuando se hace clic en el botón, una invocación de la función `onClick` imprimirá "Hola evento de clic" seguido del objeto de evento.

ngRequisito

`ng-required` agrega o elimina el atributo de validación `required` en un elemento, que a su vez habilitará y deshabilitará la clave de validación `require` para la `input` .

Se usa para definir opcionalmente si se requiere que un elemento de `input` tenga un valor no vacío. La directiva es útil cuando se diseña la validación en formularios HTML complejos.

HTML

```
<input type="checkbox" ng-model="someBooleanValue">
<input type="text" ng-model="username" ng-required="someBooleanValue">
```

ng-model-options

`ng-model-options` permite cambiar el comportamiento predeterminado de `ng-model` , esta directiva permite registrar eventos que se activarán cuando se actualice el modelo ng y adjuntar un efecto de rebote.

Esta directiva acepta una expresión que evaluará un objeto de definición o una referencia a un valor de alcance.

Ejemplo:

```
<input type="text" ng-model="myValue" ng-model-options="{'debounce': 500}">
```

El ejemplo anterior adjuntará un efecto de rebote de 500 milisegundos en `myValue` , lo que hará que el modelo se actualice 500 ms después de que el usuario haya terminado de escribir sobre la `input` (es decir, cuando `myValue` finalizado la actualización).

Propiedades de objeto disponibles

1. `updateOn` : especifica qué evento debe estar vinculado a la entrada

```
ng-model-options="{ updateOn: 'blur'}" // will update on blur
```

2. `debounce` : especifica un retraso de algunos milisegundos hacia la actualización del modelo

```
ng-model-options="{'debounce': 500}" // will update the model after 1/2 second
```

3. `allowInvalid` : un indicador booleano que permite un valor no válido para el modelo, evitando la validación de formularios por defecto, de forma predeterminada, estos valores se tratarían como `undefined` .

4. `getterSetter` : un indicador booleano que indica si se debe tratar el `ng-model` como una función getter / setter en lugar de un valor de modelo simple. La función se ejecutará y devolverá el valor del modelo.

   Ejemplo:

```
<input type="text" ng-model="myFunc" ng-model-options="{'getterSetter': true}">

$scope.myFunc = function() {return "value";}
```

5. `timezone` : define la zona horaria para el modelo si la entrada es de la `date` o la `time` . tipos

capa ng

> La directiva `ngCloak` se usa para evitar que el navegador muestre brevemente la plantilla html Angular en su forma original (sin compilar) mientras se carga la aplicación. - Ver fuente

HTML

```
<div ng-cloak>
  <h1>Hello {{ name }}</h1>
</div>
```

`ngCloak` se puede aplicar al elemento del cuerpo, pero el uso preferido es aplicar varias directivas ngCloak a pequeñas porciones de la página para permitir la representación progresiva de la vista del navegador.

La directiva `ngCloak` no tiene parámetros.

Ver también: prevenir el parpadeo

ngIncluir

ng-include le permite delegar el control de una parte de la página a un controlador específico. Es posible que desee hacer esto porque la complejidad de ese componente se está volviendo tal que desea encapsular toda la lógica en un controlador dedicado.

Un ejemplo es:

```
  <div ng-include
       src="'/gridview'"
       ng-controller='gridController as gc'>
  </div>
```

Tenga en cuenta que la `/gridview` deberá ser servida por el servidor web como una URL distinta y legítima.

Además, tenga en cuenta que el atributo `src` acepta una expresión angular. Esto podría ser una variable o una función llamada, por ejemplo, o, como en este ejemplo, una constante de cadena. En este caso, debe asegurarse de **envolver la URL de origen entre comillas simples** , para que se evalúe como una constante de cadena. Esta es una fuente común de confusión.

Dentro de `/gridview` html, puede referirse a `gridController` como si estuviera envuelto alrededor de la página, por ejemplo:

```
<div class="row">
  <button type="button" class="btn btn-default" ng-click="gc.doSomething()"></button>
</div>
```

ngSrc

El uso del marcado angular como `{{hash}}` en un atributo src no funciona bien. El navegador buscará desde la URL con el texto literal `{{hash}}` hasta que Angular reemplace la expresión dentro de `{{hash}}` . `ng-src` directiva `ng-src` anula el atributo `src` original para el elemento de etiqueta de imagen y resuelve el problema

```
<div ng-init="pic = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-src="{{pic}}">
</div>
```

ngPatrón

La directiva `ng-pattern` acepta una expresión que se evalúa como un patrón de expresión regular y usa ese patrón para validar una entrada de texto.

Ejemplo:

Digamos que queremos que un elemento `<input>` sea válido cuando su valor (ng-model) es una dirección IP válida.

Modelo:

```
<input type="text" ng-model="ipAddr" ng-pattern="ipRegex" name="ip" required>
```

Controlador:

```
$scope.ipRegex = /\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-
```

```
9]|[01]?[0-9][0-9]?)\b/;
```

ngValue

Utilizado principalmente en `ng-repeat` ngValue es útil cuando se generan dinámicamente listas de botones de radio usando ngRepeat

```
<script>
   angular.module('valueExample', [])
     .controller('ExampleController', ['$scope', function($scope) {
       $scope.names = ['pizza', 'unicorns', 'robots'];
       $scope.my = { favorite: 'unicorns' };
     }]);
</script>
 <form ng-controller="ExampleController">
   <h2>Which is your favorite?</h2>
     <label ng-repeat="name in names" for="{{name}}">
       {{name}}
       <input type="radio"
              ng-model="my.favorite"
              ng-value="name"
              id="{{name}}"
              name="favorite">
     </label>
   <div>You chose {{my.favorite}}</div>
 </form>
```

Plnkr de trabajo

ngCopy

La directiva `ngCopy` especifica el comportamiento que se ejecutará en un evento de copia.

Evitar que un usuario copie datos

```
<p ng-copy="blockCopy($event)">This paragraph cannot be copied</p>
```

En el controlador

```
$scope.blockCopy = function(event) {
    event.preventDefault();
    console.log("Copying won't work");
}
```

ngPaste

La directiva `ngPaste` especifica un comportamiento personalizado para ejecutarse cuando un usuario pega contenido

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste here'>
pasted: {{paste}}
```

ngHref

ngHref se usa en lugar del atributo href, si tenemos expresiones angulares dentro del valor href. La directiva ngHref anula el atributo href original de una etiqueta html usando el atributo href como etiqueta, etiqueta, etc.

La directiva ngHref asegura que el enlace no esté roto, incluso si el usuario hace clic en el enlace antes de que AngularJS haya evaluado el código.

Ejemplo 1

```
<div ng-init="linkValue = 'http://stackoverflow.com'">
    <p>Go to <a ng-href="{{linkValue}}">{{linkValue}}</a>!</p>
</div>
```

Ejemplo 2 Este ejemplo obtiene dinámicamente el valor href del cuadro de entrada y lo carga como valor href.

```
<input ng-model="value" />
<a id="link" ng-href="{{value}}">link</a>
```

Ejemplo 3

```
<script>
angular.module('angularDoc', [])
.controller('myController', function($scope) {
  // Set some scope value.
  // Here we set bootstrap version.
  $scope.bootstrap_version = '3.3.7';

  // Set the default layout value
  $scope.layout = 'normal';
});
</script>
<!-- Insert it into Angular Code -->
<link rel="stylesheet" ng-href="//maxcdn.bootstrapcdn.com/bootstrap/{{ bootstrap_version
}}/css/bootstrap.min.css">
<link rel="stylesheet" ng-href="layout-{{ layout }}.css">
```

ngList

La directiva `ng-list` se usa para convertir una cadena delimitada de una entrada de texto a una matriz de cadenas o viceversa.

La directiva `ng-list` usa un delimitador predeterminado de `", "` (espacio de coma).

Puede configurar el delimitador manualmente asignando `ng-list` un delimitador como este `ng-list="; "` .

En este caso, el delimitador se establece en un punto y coma seguido de un espacio.

Por defecto, `ng-list` tiene un atributo `ng-trim` que se establece en verdadero. `ng-trim` cuando es

falso, respetará el espacio en blanco en su delimitador. De forma predeterminada, `ng-list` no tiene en cuenta el espacio en blanco a menos que establezca `ng-trim="false"` .

Ejemplo:

```
angular.module('test', [])
  .controller('ngListExample', ['$scope', function($scope) {
    $scope.list = ['angular', 'is', 'cool!'];
}]);
```

Un delimitador del cliente se establece para ser `;` . Y el modelo del cuadro de entrada se establece en la matriz que se creó en el ámbito.

```
  <body ng-app="test" ng-controller="ngListExample">
    <input ng-model="list" ng-list="; " ng-trim="false">
  </body>
```

El cuadro de entrada se mostrará con el contenido: `angular; is; cool!`

Lea Directivas incorporadas en línea: https://riptutorial.com/es/angularjs/topic/706/directivas-incorporadas

Capítulo 17: Directivas personalizadas

Introducción

Aquí aprenderá sobre la característica de directivas de AngularJS. A continuación encontrará información sobre qué son las directivas, así como ejemplos básicos y avanzados de cómo usarlas.

Parámetros

Parámetro	Detalles
alcance	Propiedad para establecer el alcance de la directiva. Se puede establecer como falso, verdadero o como un alcance aislado: {@, =, <, &}.
alcance: falsy	La directiva utiliza el ámbito primario. No hay ámbito creado para directiva.
alcance: verdadero	La directiva hereda prototípicamente el ámbito principal como nuevo ámbito secundario. Si hay varias directivas en el mismo elemento que solicitan un nuevo alcance, entonces compartirán un nuevo alcance.
alcance: {@}	Enlace unidireccional de una propiedad de ámbito de directiva a un valor de atributo DOM. A medida que el valor del atributo se enlaza en el padre, cambiará en el ámbito de la directiva.
alcance: {=}	Enlace de atributo bidireccional que cambia el atributo en el padre primario si el atributo de la directiva cambia y viceversa.
alcance: {<}	Enlace unidireccional de una propiedad de ámbito de directiva y una expresión de atributo DOM. La expresión se evalúa en el padre. Esto vigila la identidad del valor principal, por lo que los cambios en una propiedad de objeto en el elemento primario no se reflejarán en la directiva. Los cambios en una propiedad de objeto en una directiva se reflejarán en el elemento primario, ya que ambos hacen referencia al mismo objeto
alcance: {&}	Permite que la directiva pase datos a una expresión para ser evaluada en el padre.
compilar: función	Esta función se utiliza para realizar la transformación DOM en la plantilla directiva antes de que se ejecute la función de enlace. Acepta `tElement` (la plantilla de directiva) y `tAttr` (lista de atributos declarados en la directiva). No tiene acceso al ámbito de aplicación. Puede devolver una función que se registrará como una función de `post-link` o puede devolver un objeto con propiedades `pre` y `post` con el que se registrará como funciones de

Parámetro	Detalles
	`post-link` `pre-link` y `post-link` .
enlace: función / objeto	La propiedad de enlace se puede configurar como una función u objeto. Puede recibir los siguientes argumentos: alcance (ámbito de la directiva), iElement (elemento DOM donde se aplica la directiva), iAttrs (colección de atributos del elemento DOM), controlador (matriz de controladores requeridos por la directiva), transcludeFn. Se utiliza principalmente para configurar escuchas de DOM, ver las propiedades del modelo en busca de cambios y actualizar el DOM. Se ejecuta después de clonar la plantilla. Se configura de forma independiente si no hay una función de compilación.
función de enlace previo	Función de enlace que se ejecuta antes que cualquier función de enlace hijo. De forma predeterminada, las funciones de enlace de directiva secundaria se ejecutan antes que las funciones de enlace de directiva principal y la función de enlace previo permite al enlace primario primero. Un caso de uso es si el niño requiere datos del padre.
función de enlace posterior	Función de enlace que los ejecutivos después de los elementos secundarios están vinculados al padre. Por lo general, se usa para adjuntar controladores de eventos y acceder a directivas secundarias, pero los datos requeridos por la directiva secundaria no deben establecerse aquí porque la directiva secundaria ya estará vinculada.
restringir: cadena	Define cómo llamar a la directiva desde el DOM. Valores posibles (Suponiendo que nuestro nombre de directiva sea `demoDirective`): `E` - Nombre del elemento (`<demo-directive></demo-directive>`), `A` - Atributo (`<div demo-directive></div>`), `C` - Clase correspondiente (`<div class="demo-directive"></div>`), `M` - Por comentario (`<!-- directive: demo-directive -->`). La propiedad de `restrict` también puede admitir múltiples opciones, por ejemplo, `restrict: "AC"` restringirá la directiva a *Atributo* O *Clase* . Si se omite, el valor **predeterminado** es `"EA"` (Elemento o Atributo).
requiere: 'demoDirective'	Localice el controlador de demoDirective en el elemento actual e inyecte su controlador como el cuarto argumento de la función de enlace. Lanzar un error si no se encuentra.
requiere: '? demoDirective'	Intente ubicar el controlador demoDirective o pase el nulo al enlace fn si no lo encuentra.
requiere: '^ demoDirective'	Localice el controlador de demoDirective buscando el elemento y sus padres. Lanzar un error si no se encuentra.
requiere: '^^ demoDirective'	Localice el controlador de demoDirective buscando los padres del elemento. Lanzar un error si no se encuentra.
requiere: '? ^	Intente ubicar el controlador de demoDirective buscando el elemento y

Parámetro	Detalles
demoDirective'	sus padres o pase el nulo al enlace fn si no lo encuentra.
require: '? ^^ demoDirective'	Intente localizar el controlador de demoDirective buscando los elementos principales del elemento, o pase el nulo al enlace fn si no lo encuentra.

Examples

Crear y consumir directivas personalizadas.

Las directivas son una de las características más poderosas de angularjs. Las directivas angulosas personalizadas se usan para ampliar la funcionalidad de html creando nuevos elementos html o atributos personalizados para proporcionar cierto comportamiento a una etiqueta html.

directiva.js

```
// Create the App module if you haven't created it yet
var demoApp= angular.module("demoApp", []);

// If you already have the app module created, comment the above line and create a reference
of the app module
var demoApp = angular.module("demoApp");


// Create a directive using the below syntax
// Directives are used to extend the capabilities of html element
// You can either create it as an Element/Attribute/class
// We are creating a directive named demoDirective. Notice it is in CamelCase when we are
defining the directive just like ngModel
// This directive will be activated as soon as any this element is encountered in html

demoApp.directive('demoDirective', function () {

  // This returns a directive definition object
  // A directive definition object is a simple JavaScript object used for configuring the
directive’s behaviour,template..etc
  return {
    // restrict: 'AE', signifies that directive is Element/Attribute directive,
    // "E" is for element, "A" is for attribute, "C" is for class, and "M" is for comment.
    // Attributes are going to be the main ones as far as adding behaviors that get used the
most.
    // If you don't specify the restrict property it will default to "A"
    restrict :'AE',

    // The values of scope property decides how the actual scope is created and used inside a
directive. These values can be either “false”, “true” or “{}”. This creates an isolate scope
for the directive.
    // '@' binding is for passing strings. These strings support {{}} expressions for
interpolated values.
    // '=' binding is for two-way model binding. The model in parent scope is linked to the
model in the directive's isolated scope.
    // '&' binding is for passing a method into your directive's scope so that it can be
called within your directive.
    // The method is pre-bound to the directive's parent scope, and supports arguments.
```

```
    scope: {
      name: "@",  // Always use small casing here even if it's a mix of 2-3 words
    },

    // template replaces the complete element with its text.
    template: "<div>Hello {{name}}!</div>",

    // compile is called during application initialization. AngularJS calls it once when html
page is loaded.
    compile: function(element, attributes) {
      element.css("border", "1px solid #cccccc");

      // linkFunction is linked with each element with scope to get the element specific data.
      var linkFunction = function($scope, element, attributes) {
        element.html("Name: <b>"+$scope.name +"</b>");
        element.css("background-color", "#ff00ff");
      };
      return linkFunction;
    }
  };
});
```

Esta directiva se puede utilizar en la aplicación como:

```
<html>

   <head>
      <title>Angular JS Directives</title>
   </head>
   <body>
   <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
   <script src="directive.js"></script>
    <div ng-app = "demoApp">
       <!-- Notice we are using Spinal Casing here -->
       <demo-directive name="World"></demo-directive>

      </div>
   </body>
</html>
```

Directiva de plantilla de objeto de definición

```
demoApp.directive('demoDirective', function () {
  var directiveDefinitionObject = {
    multiElement:
    priority:
    terminal:
    scope: {},
    bindToController: {},
    controller:
    controllerAs:
    require:
    restrict:
    templateNamespace:
    template:
    templateUrl:
    transclude:
```

```
    compile:
    link: function(){}
  };
  return directiveDefinitionObject;
});
```

1. `multiElement` - establecido en verdadero y todos los nodos DOM entre el inicio y el final del nombre de la directiva se recopilarán y agruparán como elementos de la directiva
2. `priority` : permite la especificación del orden para aplicar directivas cuando se definen múltiples directivas en un solo elemento DOM. Las directivas con números más altos se compilan primero.
3. `terminal` : se establece en verdadero y la prioridad actual será el último conjunto de directivas para ejecutar
4. `scope` : establece el ámbito de la directiva.
5. `bind to controller` : vincula las propiedades del ámbito directamente al controlador de directivas
6. `controller` - función constructor controlador
7. `require` - requiere otra directiva e inyectar su controlador como el cuarto argumento de la función de enlace.
8. `controllerAs` : referencia de nombre al controlador en el ámbito de la directiva para permitir que se haga referencia al controlador desde la plantilla de la directiva.
9. `restrict` : restringir la directiva a Elemento, Atributo, Clase o Comentario
10. `templateNameSpace` : establece el tipo de documento utilizado por la plantilla de directiva: html, svg o math. html es el predeterminado
11. `template` : marca html que por defecto reemplaza el contenido del elemento de la directiva, o envuelve el contenido del elemento de la directiva si la transclusión es verdadera
12. `templateUrl` - url proporcionada de forma asíncrona para la plantilla
13. `transclude` : extraiga el contenido del elemento donde aparece la directiva y haga que esté disponible para la directiva. Los contenidos se compilan y se proporcionan a la directiva como una función de transclusión.
14. `compile` - función para transformar la plantilla DOM
15. `link` : solo se utiliza si la propiedad de compilación no está definida. La función de enlace es responsable de registrar los escuchas de DOM y de actualizar el DOM. Se ejecuta después de que la plantilla ha sido clonada.

Ejemplo de directiva básica

superman-directiva.js

```
angular.module('myApp', [])
  .directive('superman', function() {
    return {
      // restricts how the directive can be used
      restrict: 'E',
      templateUrl: 'superman-template.html',
      controller: function() {
        this.message = "I'm superman!"
      },
      controllerAs: 'supermanCtrl',
      // Executed after Angular's initialization. Use commonly
```

```
      // for adding event handlers and DOM manipulation
      link: function(scope, element, attributes) {
        element.on('click', function() {
          alert('I am superman!')
        });
      }
    }
  });
```

superman-template.html

```
<h2>{{supermanCtrl.message}}</h2>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.js"></script>
  <script src="superman-directive.js"><script/>
</head>
<body>
<div ng-app="myApp">
  <superman></superman>
</div>
</body>
</html>
```

Puede consultar más sobre las funciones de `restrict` y `link` la directiva en la documentación oficial de AngularJS sobre directivas

Cómo crear un componente resuable usando una directiva.

Las directivas AngularJS son las que controlan la representación del HTML dentro de una aplicación AngularJS. Pueden ser un elemento HTML, atributo, clase o un comentario. Las directivas se utilizan para manipular el DOM, adjuntando nuevos comportamientos a elementos HTML, enlace de datos y muchos más. Algunos de los ejemplos de directivas que proporciona angular son ng-model, ng-hide, ng-if.

Del mismo modo, uno puede crear su propia directiva personalizada y hacerlos valiosos. Para crear referencias de directivas personalizadas. El sentido detrás de la creación de directivas reutilizables es hacer un conjunto de directivas / componentes escritos por usted, tal como angularjs nos proporciona el uso de angular.js. Estas directivas reutilizables pueden ser particularmente muy útiles cuando tiene un conjunto de aplicaciones / aplicación que requiere un comportamiento y apariencia consistentes. Un ejemplo de dicho componente reutilizable puede ser una barra de herramientas simple que puede usar en su aplicación o en diferentes aplicaciones, pero desea que se comporten igual o se vean igual.

En primer lugar, cree una carpeta llamada resuableComponents en la carpeta de su aplicación y haga reutilizableModuleApp.js

reutilizableModuleApp.js:

```
(function(){

 var reusableModuleApp = angular.module('resuableModuleApp', ['ngSanitize']);

 //Remember whatever dependencies you have in here should be injected in the app module where
it is intended to be used or it's scripts should be included in your main app
                               //We will be injecting ng-sanitize

 resubaleModuleApp.directive('toolbar', toolbar)

 toolbar.$inject=['$sce'];

 function toolbar($sce){

     return{
     restrict :'AE',
        //Defining below isolate scope actually provides window for the directive to take data
from app that will be using this.
        scope : {
                value1: '=',
                value2: '=',
                },

         }
     template : '<ul>  <li><a ng-click="Add()" href="">{{value1}}</a></li>  <li><a ng-
click="Edit()" href="#">{{value2}}</a></li> </ul> ',
         link : function(scope, element, attrs){

              //Handle's Add function
              scope.Add = function(){


              };

              //Handle's Edit function
              scope.Edit = function(){

              };
     }
  }

});
```

mainApp.js:

```
(function(){
   var mainApp = angular.module('mainApp', ['reusableModuleApp']); //Inject resuableModuleApp
in your application where you want to use toolbar component

   mainApp.controller('mainAppController', function($scope){
      $scope.value1 = "Add";
      $scope.value2 = "Edit";

   });

 });
```

index.html:

```
<!doctype html>
<html ng-app="mainApp">
<head>
 <title> Demo Making a reusable component
<head>
 <body ng-controller="mainAppController">

    <!-- We are providing data to toolbar directive using mainApp'controller -->
    <toolbar value1="value1" value2="value2"></toolbar>

   <!-- We need to add the dependent js files on both apps here -->
    <script src="js/angular.js"></script>
    <script src="js/angular-sanitize.js"></script>

    <!-- your mainApp.js should be added afterwards --->
     <script src="mainApp.js"></script>

    <!-- Add your reusable component js files here -->
     <script src="resuableComponents/reusableModuleApp.js"></script>

 </body>
</html>
```

Las directivas son componentes reutilizables por defecto. Cuando hace directivas en un módulo angular separado, en realidad lo hace exportable y reutilizable en diferentes aplicaciones angulares. Las nuevas directivas simplemente se pueden agregar dentro de reusableModuleApp.js y reusableModuleApp puede tener su propio controlador, servicios, objeto DDO dentro de la directiva para definir el comportamiento.

Directiva básica con plantilla y un alcance aislado.

La creación de una directiva personalizada con *alcance aislado* separará el alcance **dentro de** la directiva del alcance **externo** , a fin de evitar que nuestra directiva cambie accidentalmente los datos en el alcance principal y restrinja la lectura de datos privados del alcance principal.

Para crear un alcance aislado y aún permitir que nuestra directiva personalizada se comunique con el alcance externo, podemos usar la opción de `scope` que describe cómo **asignar** los enlaces del ámbito interno de la directiva con el alcance externo.

Los enlaces reales se realizan con **atributos** adicionales adjuntos a la directiva. Las configuraciones de enlace se definen con la opción de `scope` y un objeto con pares clave-valor:

- Una **clave** , que se corresponde con la propiedad de alcance aislado de nuestra directiva.
- Un **valor** , que le indica a Angular cómo vincular el ámbito interno de la directiva a un **atributo** coincidente

Ejemplo simple de una directiva con un alcance aislado:

```
var ProgressBar = function() {
  return {
```

```
    scope: { // This is how we define an isolated scope
      current: '=', // Create a REQUIRED bidirectional binding by using the 'current'
attribute
      full: '=?maxValue' // Create an OPTIONAL (Note the '?'): bidirectional binding using
'max-value' attribute to the `full` property in our directive isolated scope
    }
    template: '<div class="progress-back">' +
              '  <div class="progress-bar"' +
              '       ng-style="{width: getProgress()}">' +
              '  </div>' +
              '</div>',
    link: function(scope, el, attrs) {
      if (scope.full === undefined) {
        scope.full = 100;
      }
      scope.getProgress = function() {
        return (scope.current / scope.size * 100) + '%';
      }
    }
  }
}

ProgressBar.$inject = [];
angular.module('app').directive('progressBar', ProgressBar);
```

Ejemplo de cómo usar esta directiva y vincular los datos del alcance del controlador al alcance interno de la directiva:

Controlador:

```
angular.module('app').controller('myCtrl', function($scope) {
    $scope.currentProgressValue = 39;
    $scope.maxProgressBarValue = 50;
});
```

Ver:

```
<div ng-controller="myCtrl">
    <progress-bar current="currentProgressValue"></progress-bar>
    <progress-bar current="currentProgressValue" max-value="maxProgressBarValue"></progress-
bar>
</div>
```

Construyendo un componente reutilizable.

Las directivas se pueden utilizar para construir componentes reutilizables. Aquí hay un ejemplo de un componente de "carpeta de usuario":

userBox.js

```
angular.module('simpleDirective', []).directive('userBox', function() {
  return {
    scope: {
        username: '=username',
        reputation: '=reputation'
```

```
    },
    templateUrl: '/path/to/app/directives/user-box.html'
  };
});
```

Controller.js

```
var myApp = angular.module('myApp', ['simpleDirective']);

myApp.controller('Controller', function($scope) {

    $scope.user = "John Doe";
    $scope.rep = 1250;

    $scope.user2 = "Andrew";
    $scope.rep2 = 2850;

});
```

myPage.js

```
<html lang="en" ng-app="myApp">
  <head>
    <script src="/path/to/app/angular.min.js"></script>
    <script src="/path/to/app/js/controllers/Controller.js"></script>
    <script src="/path/to/app/js/directives/userBox.js"></script>
  </head>

  <body>

    <div ng-controller="Controller">
        <user-box username="user" reputation="rep"></user-box>
        <user-box username="user2" reputation="rep2"></user-box>
    </div>

  </body>
</html>
```

user-box.html

```
<div>{{username}}</div>
<div>{{reputation}} reputation</div>
```

El resultado será:

```
John Doe
1250 reputation
Andrew
2850 reputation
```

Directivo decorador

A veces es posible que necesite características adicionales de una directiva. En lugar de reescribir (copiar) la directiva, puede modificar el comportamiento de la directiva.

El decorador será ejecutado durante la fase de inyección.

Para ello, provea un .config a su módulo. La directiva se llama myDirective, por lo que tiene que configurar myDirectiveDirective. (esto en una convención angular [leer sobre proveedores]).

Este ejemplo cambiará el templateUrl de la directiva:

```
angular.module('myApp').config(function($provide){
        $provide.decorator('myDirectiveDirective', function($delegate){
             var directive = $delegate[0]; // this is the actual delegated, your directive
             directive.templateUrl = 'newTemplate.html'; // you change the directive template
        return $delegate;
    })
});
```

Este ejemplo agrega un evento onClick al elemento directivo cuando se hace clic, esto sucede durante la fase de compilación.

```
angular.module('myApp').config(function ($provide) {
        $provide.decorator('myDirectiveTwoDirective', function ($delegate) {
            var directive = $delegate[0];
            var link = directive.link; // this is directive link phase
            directive.compile = function () { // change the compile of that directive
                return function (scope, element, attrs) {
                    link.apply(this, arguments); // apply this at the link phase
                    element.on('click', function(){ // when add an onclick that log hello when
the directive is clicked.
                            console.log('hello!');
                    });
                };
            };
            return $delegate;
        });

    });
```

Se puede utilizar un enfoque similar tanto para los proveedores como para los servicios.

Directiva de herencia e interoperabilidad.

Las directivas angulares js se pueden anidar o hacer interoperables.

En este ejemplo, la directiva Adir expone a la directiva Bdir es su controlador $ scope, ya que Bdir requiere Adir.

```
angular.module('myApp',[]).directive('Adir', function () {
        return {
            restrict: 'AE',
            controller: ['$scope', function ($scope) {
                    $scope.logFn = function (val) {
                            console.log(val);
                        }
                  }]
            }
    })
```

Asegúrese de que los ajustes requieran: '^ Adir' (mire la documentación angular, algunas versiones no requieren ^ carácter).

```
.directive('Bdir', function () {
        return {
            restrict: 'AE',
            require: '^Adir', // Bdir require Adir
            link: function (scope, elem, attr, Parent) {
            // Parent is Adir but can be an array of required directives.
                elem.on('click', function ($event) {
                    Parent.logFn("Hello!"); // will log "Hello! at parent dir scope
                    scope.$apply(); // apply to parent scope.
                });
            }
        }
    }]);
```

Puedes anidar tu directiva de esta manera:

```
<div a-dir><span b-dir></span></div>
<a-dir><b-dir></b-dir> </a-dir>
```

No es necesario que las directivas estén anidadas en su HTML.

Lea Directivas personalizadas en línea: https://riptutorial.com/es/angularjs/topic/965/directivas-personalizadas

Capítulo 18: Directivas utilizando ngModelController

Examples

Un control simple: calificación

Permítanos construir un control simple, un widget de calificación, destinado a ser utilizado como:

```
<rating min="0" max="5" nullifier="true" ng-model="data.rating"></rating>
```

No hay CSS de lujo por ahora; esto se traduciría como:

```
0 1 2 3 4 5 x
```

Al hacer clic en un número selecciona esa calificación; y al hacer clic en la "x" se establece la calificación en nulo.

```
app.directive('rating', function() {

    function RatingController() {
        this._ngModel = null;
        this.rating = null;
        this.options = null;
        this.min = typeof this.min === 'number' ? this.min : 1;
        this.max = typeof this.max === 'number' ? this.max : 5;
    }

    RatingController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 1
            ngModel.$render = this._render.bind(this);
        }
    };

    RatingController.prototype._render = function() {
        this.rating = this._ngModel.$viewValue != null ? this._ngModel.$viewValue : -
Number.MAX_VALUE;
    };

    RatingController.prototype._calculateOptions = function() {
        if( this.min == null || this.max == null ) {
            this.options = [];
        }
        else {
            this.options = new Array(this.max - this.min + 1);
            for( var i=0; i < this.options.length; i++ ) {
                this.options[i] = this.min + i;
            }
        }
```

```
    };

    RatingController.prototype.setValue = function(val) {
        this.rating = val;
        // KEY POINT 2
        this._ngModel.$setViewValue(val);
    };

    // KEY POINT 3
    Object.defineProperty(RatingController.prototype, 'min', {
        get: function() {
            return this._min;
        },
        set: function(val) {
            this._min = val;
            this._calculateOptions();
        }
    });

    Object.defineProperty(RatingController.prototype, 'max', {
        get: function() {
            return this._max;
        },
        set: function(val) {
            this._max = val;
            this._calculateOptions();
        }
    });

    return {
        restrict: 'E',
        scope: {
            // KEY POINT 3
            min: '<?',
            max: '<?',
            nullifier: '<?'
        },
        bindToController: true,
        controllerAs: 'ctrl',
        controller: RatingController,
        require: ['rating', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<span ng-repeat="o in ctrl.options" href="#" class="rating-option" ng-
class="{\'rating-option-active\': o <= ctrl.rating}" ng-click="ctrl.setValue(o)">{{ o
}}</span>' +
            '<span ng-if="ctrl.nullifier" ng-click="ctrl.setValue(null)" class="rating-
nullifier">&#10006;</span>'
    };
});
```

Puntos clave:

1. Implemente `ngModel.$render` para transferir el *valor de vista* del modelo a su vista.
2. Llame a `ngModel.$setViewValue()` siempre que sienta que el valor de vista debe actualizarse.
3. El control puede, por supuesto, ser parametrizado; use `'<'` enlaces de alcance para los parámetros, si está en Angular> = 1.5 para indicar claramente la entrada - enlace de una

vía. Si tiene que tomar medidas cada vez que cambia un parámetro, puede usar una propiedad de JavaScript (consulte `Object.defineProperty()`) para guardar algunos relojes.

Nota 1: Para no complicar demasiado la implementación, los valores de calificación se insertan en una matriz: las `ctrl.options` . Esto no es necesario; una implementación más eficiente, pero también más compleja, podría utilizar la manipulación DOM para insertar / eliminar calificaciones cuando el cambio `min` / `max` .

Nota 2: Con la excepción de los enlaces de alcance `'<'` , este ejemplo se puede usar en Angular <1.5. Si está en Angular> = 1.5, sería una buena idea transformar esto en un componente y usar el gancho de ciclo de vida `$onInit()` para inicializar `min` y `max` , en lugar de hacerlo en el constructor del controlador.

Y un violín necesario: https://jsfiddle.net/h81mgxma/

Un par de controles complejos: editar un objeto completo

Un control personalizado no tiene que limitarse a cosas triviales como los primitivos; Puede editar cosas más interesantes. Aquí presentamos dos tipos de controles personalizados, uno para editar personas y otro para editar direcciones. El control de dirección se utiliza para editar la dirección de la persona. Un ejemplo de uso sería:

```
<input-person ng-model="data.thePerson"></input-person>
<input-address ng-model="data.thePerson.address"></input-address>
```

El modelo para este ejemplo es deliberadamente simplista:

```
function Person(data) {
  data = data || {};
  this.name = data.name;
  this.address = data.address ? new Address(data.address) : null;
}

function Address(data) {
  data = data || {};
  this.street = data.street;
  this.number = data.number;
}
```

El editor de direcciones:

```
app.directive('inputAddress', function() {

    InputAddressController.$inject = ['$scope'];
    function InputAddressController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputAddressController.prototype.setNgModel = function(ngModel) {
```

```
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 3
            ngModel.$render = this._render.bind(this);
        }
    };

    InputAddressController.prototype._makeWatch = function() {
        // KEY POINT 1
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Address(newval) : null);
                }
            }).bind(this)
        );
    };

    InputAddressController.prototype._render = function() {
        // KEY POINT 2
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Address(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputAddressController,
        require: ['inputAddress', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Street:</span><input type="text" ng-model="ctrl.value.street"
/></label>' +
                '<label><span>Number:</span><input type="text" ng-model="ctrl.value.number"
/></label>' +
            '</div>'
    };
});
```

Puntos clave:

1. Estamos editando un objeto; no queremos cambiar directamente el objeto que nos entregó nuestro padre (queremos que nuestro modelo sea compatible con el principio de inmutabilidad). Así que creamos una observación superficial sobre el objeto que se está editando y actualizamos el modelo con `$setViewValue()` cada vez que cambia una propiedad. Pasamos una *copia* a nuestros padres.
2. Cuando el modelo cambia desde el exterior, lo copiamos y guardamos la copia en nuestro alcance. De nuevo los principios de la inmutabilidad, aunque la copia interna no es

inmutable, la externa podría muy bien serlo. Además, reconstruimos el reloj (`this_unwatch();this._makeWatch();`), para evitar activar al observador por los cambios que nos presenta el modelo. (Solo queremos que el reloj se active por los cambios realizados en la interfaz de usuario).

3. Aparte de los puntos anteriores, implementamos `ngModel.$render()` y llamamos `ngModel.$setViewValue()` como lo haríamos con un control simple (consulte el ejemplo de calificación).

El código para el control personalizado de la persona es casi idéntico. La plantilla está utilizando la `<input-address>` . En una implementación más avanzada podríamos extraer los controladores en un módulo reutilizable.

```
app.directive('inputPerson', function() {

    InputPersonController.$inject = ['$scope'];
    function InputPersonController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputPersonController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            ngModel.$render = this._render.bind(this);
        }
    };

    InputPersonController.prototype._makeWatch = function() {
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Person(newval) : null);
                }
            }).bind(this)
        );
    };

    InputPersonController.prototype._render = function() {
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Person(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputPersonController,
        require: ['inputPerson', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
```

```
        },
        template:
            '<div>' +
                '<label><span>Name:</span><input type="text" ng-model="ctrl.value.name"
/></label>' +
                '<input-address ng-model="ctrl.value.address"></input-address>' +
            '</div>'
    };
});
```

Nota: aquí se escriben los objetos, es decir, tienen constructores adecuados. Esto no es obligatorio; El modelo puede ser simple objetos JSON. En este caso, simplemente use `angular.copy()` lugar de los constructores. Una ventaja adicional es que el controlador se vuelve idéntico para los dos controles y se puede extraer fácilmente en algún módulo común.

El violín: https://jsfiddle.net/3tzyqfko/2/

Dos versiones del violín que extrajeron el código común de los controladores: https://jsfiddle.net/agj4cp0e/ y https://jsfiddle.net/ugb6Lw8b/

Lea Directivas utilizando ngModelController en línea: https://riptutorial.com/es/angularjs/topic/2438/directivas-utilizando-ngmodelcontroller

Capítulo 19: El yo o esta variable en un controlador

Introducción

Esta es una explicación de un patrón común y generalmente se considera la mejor práctica que puede ver en el código de AngularJS.

Examples

Entender el propósito de la variable del uno mismo

Cuando use "controlador como sintaxis", le dará a su controlador un alias en el html cuando use la directiva ng-controller.

```
<div ng-controller="MainCtrl as main">
</div>
```

Luego puede acceder a las propiedades y métodos desde la variable ***principal*** que representa nuestra instancia de controlador. Por ejemplo, accedamos a la propiedad de ***saludo*** de nuestro controlador y la mostremos en la pantalla:

```
<div ng-controller="MainCtrl as main">
    {{ main.greeting }}
</div>
```

Ahora, en nuestro controlador, debemos establecer un valor en la propiedad de saludo de nuestra instancia del controlador (en lugar de $ scope o algo más):

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";
})
```

Con el fin de tener la pantalla HTML correctamente que necesitamos para establecer la propiedad de felicitación en ***esto*** dentro de nuestro cuerpo del controlador. Estoy creando una variable intermedia llamada ***self*** que contiene una referencia a esto. ¿Por qué? Considere este código:

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

```

```
  self.greeting = "Hello World";

  function itsLate () {
    this.greeting = "Goodnight";
  }

})
```

En este código anterior, puede esperar que el texto en la pantalla se actualice cuando se *llame* al método *ItsLate* , pero en realidad no lo hace. JavaScript usa las reglas de alcance del nivel de función, por lo que "esto" dentro de su estado actual se refiere a algo diferente que "esto" fuera del cuerpo del método. Sin embargo, podemos obtener el resultado deseado si usamos la ***propia*** variable:

```
 angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";

  function itsLate () {
    self.greeting = "Goodnight";
  }

})
```

Esta es la belleza de usar una variable "propia" en sus controladores: puede acceder a esta desde cualquier parte de su controlador y siempre puede estar seguro de que hace referencia a su instancia de controlador.

Lea El yo o esta variable en un controlador en línea:
https://riptutorial.com/es/angularjs/topic/8867/el-yo-o-esta-variable-en-un-controlador

Capítulo 20: enrutador ui

Observaciones

¿Qué es `ui-router` ?

> Angular UI-Router es un marco de enrutamiento de aplicaciones de una sola página del lado del cliente para AngularJS.
>
> Los marcos de enrutamiento para SPA actualizan la URL del navegador a medida que el usuario navega a través de la aplicación. A la inversa, esto permite cambios en la URL del navegador para conducir la navegación a través de la aplicación, lo que permite al usuario crear un marcador en una ubicación dentro del SPA.
>
> Las aplicaciones de UI-Router se modelan como un árbol jerárquico de estados. UI-Router proporciona una máquina de estados para administrar las transiciones entre los estados de las aplicaciones de manera similar a una transacción.

Enlaces útiles

Puedes encontrar la documentación oficial de la API aquí . Si tiene preguntas sobre `ui-router` VS `ng-router` , puede encontrar una respuesta razonablemente detallada aquí . Tenga en cuenta que `ng-router` ya ha lanzado una nueva actualización de ng-router llamada `ngNewRouter` (compatible con Angular 1.5 + / 2.0) que admite estados como ui-router. Puedes leer más sobre `ngNewRouter` aquí .

Examples

Ejemplo básico

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        templateUrl: "view-one.html",
        controller: 'controllerOne',
        controllerAs: 'ctrlOne'
```

```
      })
      .state('two', {
        url: "/two",
        templateUrl: "view-two.html",
        controller: 'controllerTwo',
        controllerAs: 'ctrlTwo'
      })
      .state('three', {
        url: "/three",
        templateUrl: "view-three.html",
        controller: 'controllerThree',
        controllerAs: 'ctrlThree'
      });

      $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">View One</a>
    <a ui-sref="two">View Two</a>
    <a ui-sref="three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Vistas múltiples

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .controller('controllerFour', function() {
    this.message = 'Hello world from Controller Four!';
```

```
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        views: {
          "viewA": {
            templateUrl: "view-one.html",
            controller: 'controllerOne',
            controllerAs: 'ctrlOne'
          },
          "viewB": {
            templateUrl: "view-two.html",
            controller: 'controllerTwo',
            controllerAs: 'ctrlTwo'
          }
        }
      })
      .state('two', {
        url: "/two",
        views: {
          "viewA": {
            templateUrl: "view-three.html",
            controller: 'controllerThree',
            controllerAs: 'ctrlThree'
          },
          "viewB": {
            templateUrl: "view-four.html",
            controller: 'controllerFour',
            controllerAs: 'ctrlFour'
          }
        }
      });

    $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">Route One</a>
    <a ui-sref="two">Route Two</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view="viewA"></div>
  <div ui-view="viewB"></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
```

```
  <script type="text/ng-template" id="view-four.html">
    <h1>{{ctrlFour.message}}</h1>
  </script>
</div>
```

Usando funciones de resolver para cargar datos

app.js

```
angular.module('myApp', ['ui.router'])
  .service('User', ['$http', function User ($http) {
    this.getProfile = function (id) {
      return $http.get(...) // method to load data from API
    };
  }])
  .controller('profileCtrl', ['profile', function profileCtrl (profile) {
    // inject resolved data under the name of the resolve function
    // data will already be returned and processed
    this.profile = profile;
  }])
  .config(['$stateProvider', '$urlRouterProvider', function ($stateProvider,
$urlRouterProvider) {
    $stateProvider
      .state('profile', {
        url: "/profile/:userId",
        templateUrl: "profile.html",
        controller: 'profileCtrl',
        controllerAs: 'vm',
        resolve: {
          profile: ['$stateParams', 'User', function ($stateParams, User) {
            // $stateParams will contain any parameter defined in your url
            return User.getProfile($stateParams.userId)
              // .then is only necessary if you need to process returned data
              .then(function (data) {
                return doSomeProcessing(data);
              });
          }]
        }
      }]);

      $urlRouterProvider.otherwise('/');
  });
```

profile.html

```
<ul>
  <li>Name: {{vm.profile.name}}</li>
  <li>Age: {{vm.profile.age}}</li>
  <li>Sex: {{vm.profile.sex}}</li>
</ul>
```

Ver la entrada de UI-Router Wiki en resuelve aquí .

Las funciones de resolución deben resolverse antes `$stateChangeSuccess` evento `$stateChangeSuccess` , lo que significa que la IU no se cargará hasta que *todas las* funciones de

resolución en el estado hayan finalizado. Esta es una excelente manera de asegurar que los datos estén disponibles para su controlador y su interfaz de usuario. Sin embargo, puede ver que una función de resolución debe ser rápida para evitar bloquear la interfaz de usuario.

Vistas anidadas / Estados

app.js

```
var app = angular.module('myApp',['ui.router']);

app.config(function($stateProvider,$urlRouterProvider) {

    $stateProvider

    .state('home', {
        url: '/home',
        templateUrl: 'home.html',
        controller: function($scope){
            $scope.text = 'This is the Home'
        }
    })

    .state('home.nested1',{
        url: '/nested1',
        templateUrl:'nested1.html',
        controller: function($scope){
            $scope.text1 = 'This is the nested view 1'
        }
    })

    .state('home.nested2',{
        url: '/nested2',
        templateUrl:'nested2.html',
        controller: function($scope){
            $scope.fruits = ['apple','mango','oranges'];
        }
    });

    $urlRouterProvider.otherwise('/home');

});
```

index.html

```
  <div ui-view></div>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
   <script src="angular-ui-router.min.js"></script>
   <script src="app.js"></script>
```

home.html

```
<div>
<h1> {{text}} </h1>
<br>
    <a ui-sref="home.nested1">Show nested1</a>
```

```
    <br>
    <a ui-sref="home.nested2">Show nested2</a>
    <br>

    <div ui-view></div>
</div>
```

nested1.html

```
<div>
<h1> {{text1}} </h1>
</div>
```

nested2.html

```
<div>
    <ul>
      <li ng-repeat="fruit in fruits">{{ fruit }}</li>
    </ul>
</div>
```

Lea enrutador ui en línea: https://riptutorial.com/es/angularjs/topic/2545/enrutador-ui

Capítulo 21: Enrutamiento usando ngRoute

Observaciones

El `ngRoute` es un módulo incorporado que proporciona directivas y servicios de enrutamiento y deeplinking para aplicaciones angulares.

La documentación completa sobre `ngRoute` está disponible en https://docs.angularjs.org/api/ngRoute

Examples

Ejemplo basico

Este ejemplo muestra la configuración de una pequeña aplicación con 3 rutas, cada una con su propia vista y controlador, utilizando la sintaxis del `controllerAs` .

Configuramos nuestro enrutador en la función angular `.config` .

1. `$routeProvider` en `.config`
2. Definimos nuestros nombres de ruta en el método `.when` con un objeto de definición de ruta.
3. Suministramos al método `.when` con un objeto que especifica nuestra `template` o `templateUrl` , `controller` y `controllerAs`

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
```

```
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Luego, en nuestro HTML definimos nuestra navegación usando `<a>` elementos con `href` , para un nombre de ruta de `helloRoute` lo haremos como `<a href="#/helloRoute">My route</a>`

También proporcionamos nuestra vista con un contenedor y la directiva `ng-view` para inyectar nuestras rutas.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Ejemplo de parámetros de ruta

Este ejemplo amplía el ejemplo básico pasando parámetros en la ruta para usarlos en el controlador

Para ello necesitamos:

1. Configure la posición y el nombre del parámetro en el nombre de la ruta
2. Inyecte el servicio `$routeParams` en nuestro controlador

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
```

```
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', ['$routeParams', function($routeParams) {
    var routeParam = $routeParams.paramName

    if ($routeParams.message) {
        // If a param called 'message' exists, we show it's value as the message
        this.message = $routeParams.message;
    } else {
        // If it doesn't exist, we show a default message
        this.message = 'Hello world from Controller Three!';
    }
  }])
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    .when('/three/:message', { // We will pass a param called 'message' with this route
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Luego, sin hacer ningún cambio en nuestras plantillas, solo agregando un nuevo enlace con un mensaje personalizado, podemos ver el nuevo mensaje personalizado en nuestra vista.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
    <!-- New link with custom message -->
    <a href="#/three/This-is-a-message">View Three with "This-is-a-message" custom message</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
```

```
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Definición de comportamiento personalizado para rutas individuales.

La forma más sencilla de definir el comportamiento personalizado para rutas individuales sería bastante fácil.

En este ejemplo lo usamos para autenticar a un usuario:

1) `routes.js` : crea una nueva propiedad (como `requireAuth`) para cualquier ruta deseada

```
angular.module('yourApp').config(['$routeProvider', function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'templates/home.html',
            requireAuth: true
        })
        .when('/login', {
            templateUrl: 'templates/login.html',
        })
        .otherwise({
            redirectTo: '/home'
        });
}])
```

2) En un controlador de nivel superior que no está vinculado a un elemento dentro de la `ng-view` (para evitar conflictos con `$routeProvider` angular), verifique si `newUrl` tiene la propiedad `requireAuth` y actúe en consecuencia

```
angular.module('YourApp').controller('YourController', ['$scope', 'session', '$location',
    function($scope, session, $location) {

        $scope.$on('$routeChangeStart', function(angularEvent, newUrl) {

            if (newUrl.requireAuth && !session.user) {
                // User isn’t authenticated
                $location.path("/login");
            }

        });
    }
]);
```

Lea Enrutamiento usando ngRoute en línea:

https://riptutorial.com/es/angularjs/topic/2391/enrutamiento-usando-ngroute

Capítulo 22: estilo ng

Introducción

La directiva 'ngStyle' le permite establecer el estilo CSS en un elemento HTML condicionalmente. Al igual que la forma en que podríamos usar el atributo de *estilo* en un elemento HTML en proyectos que no son AngularJS, podemos usar `ng-style` en angularjs para aplicar estilos basados en alguna condición booleana.

Sintaxis

- `<ANY ng-style="expression"></ANY >`
- `<ANY class="ng-style: expression;"> ... </ANY>`

Examples

Uso de estilo ng

El siguiente ejemplo cambia la opacidad de la imagen según el parámetro "estado".

```
<img class="img-responsive"  ng-src="{{imagesrc}}"
    ng-style="{'opacity' : (status == 2) ? 1 : 0.5}">
```

Lea estilo ng en línea: https://riptutorial.com/es/angularjs/topic/8773/estilo-ng

Capítulo 23: Eventos

Parámetros

Parámetros	Tipos de valores
evento	Object {name: "eventName", targetScope: Scope, defaultPrevented: false, currentScope: ChildScope}
args	Datos que se han pasado junto con la ejecución del evento.

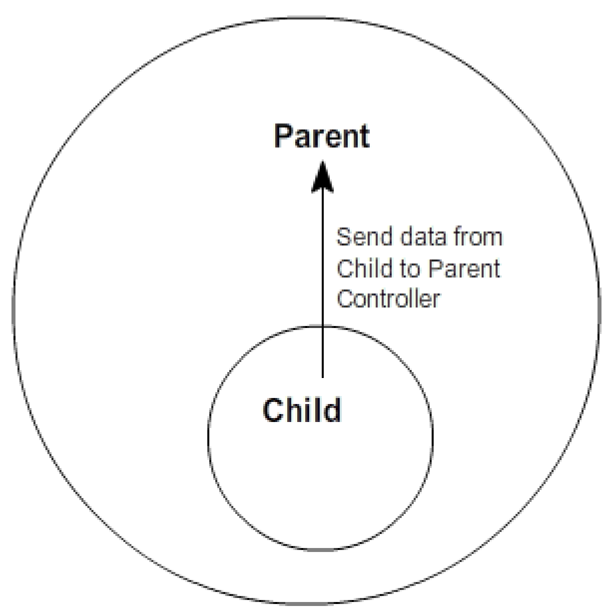
Examples

Utilizando sistema de eventos angulares.

$ scope. $ emit

El uso de `$scope.$emit` disparará un nombre de evento hacia arriba a través de la jerarquía de alcance y notificará a `$scope` . El ciclo de vida del evento comienza en el ámbito en el que se llamó a `$emit` .

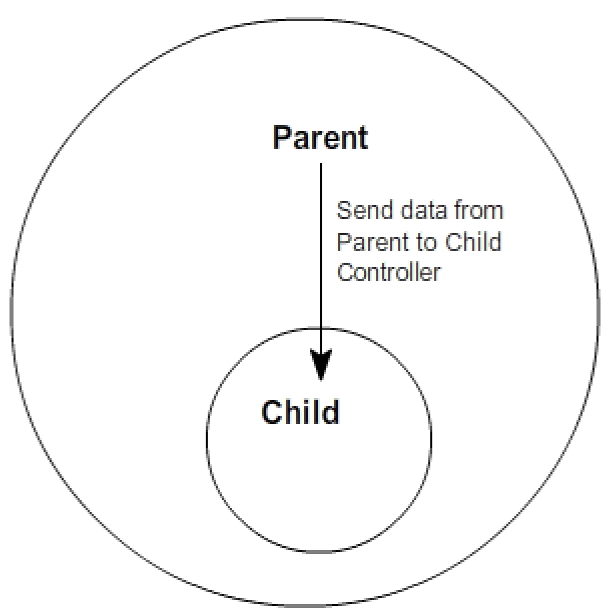
Wireframe de trabajo:



$ scope. $ broadcast

El uso de `$scope.$broadcast` desencadenará un evento en `$scope` . Podemos escuchar estos eventos usando `$scope.$on`

Wireframe de trabajo:



Sintaxis:

```
// firing an event upwards
$scope.$emit('myCustomEvent', 'Data to send');

// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'some value'
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data from the event'
});
```

En lugar de `$scope` , puede usar `$rootScope` , en ese caso, su evento estará disponible en todos los controladores, independientemente del alcance de los controladores.

Evento limpio registrado en AngularJS

El motivo para limpiar los eventos registrados, ya que incluso el controlador ha sido destruido, el manejo del evento registrado sigue vivo. Así que el código se ejecutará como inesperado seguro.

```
// firing an event upwards
$rootScope.$emit('myEvent', 'Data to send');

// listening an event
var listenerEventHandler = $rootScope.$on('myEvent', function(){
    //handle code
});

$scope.$on('$destroy', function() {
    listenerEventHandler();
});
```

Usos y significado

Estos eventos se pueden utilizar para comunicarse entre 2 o más controladores.

`$emit` envía un evento hacia arriba a través de la jerarquía de alcance, mientras que `$broadcast` envía un evento hacia abajo a todos los ámbitos secundarios. Esto se ha explicado muy bien aquí .

Básicamente, puede haber dos tipos de escenarios mientras se comunican entre los controladores:

---

1. Cuando los controladores tienen relación padre-hijo. (En su mayoría, podemos usar `$scope` en tales escenarios)

---

2. Cuando los controladores no son independientes entre sí y son necesarios para estar informados sobre la actividad de cada uno. (Podemos usar `$rootScope` en tales escenarios)

por ejemplo: para cualquier sitio web de comercio electrónico, supongamos que tenemos `ProductListController` (que controla la página de listado de productos cuando se hace clic en cualquier marca del producto) y `CartController` (para administrar los artículos del carrito). Ahora, cuando hacemos clic en el botón **Agregar al carrito** , también se debe informar a `CartController` , para que pueda reflejar el nuevo recuento de artículos del carrito / detalles en la barra de navegación del sitio web. Esto se puede lograr usando `$rootScope` .

Con `$scope.$emit`

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
```

```
                $scope.message = args.message;
            });
        });

        app.controller("SecondController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$emit('eventName', {message: msg});
            };
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px ;padding:5px;">
        <h1>Parent Controller</h1>
        <p>Emit Message : {{message}}</p>
        <br />
        <div ng-controller="SecondController" style="border:2px;padding:5px;">
            <h1>Child Controller</h1>
            <input ng-model="msg">
            <button ng-click="handleClick(msg);">Emit</button>
        </div>
    </div>
  </body>
</html>
```

Con `$scope.$broadcast` :

```
<html>
  <head>
    <title>Broadcasting</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$broadcast('eventName', {message: msg});
            };

        });

        app.controller("SecondController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px solid ; padding:5px;">
        <h1>Parent Controller</h1>
        <input ng-model="msg">
        <button ng-click="handleClick(msg);">Broadcast</button>
        <br /><br />
        <div ng-controller="SecondController" style="border:2px solid ;padding:5px;">
            <h1>Child Controller</h1>
            <p>Broadcast Message : {{message}}</p>
        </div>
```

```
    </div>
  </body>
</html>
```

Siempre cancele el registro de $ rootScope. $ En los escuchas en el evento scope $ destory

$ rootScope. $ en los escuchas permanecerán en la memoria si navega a otro controlador. Esto creará una pérdida de memoria si el controlador queda fuera del alcance.

No hacer

```
angular.module('app').controller('badExampleController', badExample);

badExample.$inject = ['$scope', '$rootScope'];
function badExample($scope, $rootScope) {

    $rootScope.$on('post:created', function postCreated(event, data) {});

}
```

Hacer

```
angular.module('app').controller('goodExampleController', goodExample);

goodExample.$inject = ['$scope', '$rootScope'];
function goodExample($scope, $rootScope) {

    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });

}
```

Lea Eventos en línea: https://riptutorial.com/es/angularjs/topic/1922/eventos

Capítulo 24: Filtros

Examples

Su primer filtro

Los filtros son un tipo especial de función que puede modificar la forma en que se imprime algo en la página, o se puede usar para filtrar una matriz o una acción de `ng-repeat` . Puede crear un filtro llamando al método `app.filter()` , pasándole un nombre y una función. Vea los ejemplos a continuación para detalles sobre la sintaxis.

Por ejemplo, vamos a crear un filtro que cambiará una cadena a mayúsculas (esencialmente un contenedor de la función javascript `.toUpperCase()`):

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

// just like making a controller, you must give the
// filter a unique name, in this case "toUppercase"
app.filter('toUppercase', function(){
    // all the filter does is return a function,
    // which acts as the "filtering" function
    return function(rawString){
        // The filter function takes in the value,
        // which we modify in some way, then return
        // back.
        return rawString.toUpperCase();
    };
});
```

Echemos un vistazo más de cerca a lo que está pasando arriba.

Primero, estamos creando un filtro llamado "toUppercase", que es como un controlador; `app.filter(...)` . Entonces, la función de ese filtro devuelve la función de filtro real. Esa función toma un solo objeto, que es el objeto que se filtra, y debe devolver la versión filtrada del objeto.

Nota: *En esta situación, asumimos que el objeto que se pasa al filtro es una cadena y, por lo tanto, sabemos que siempre se debe usar el filtro solo en las cadenas. Dicho esto, se podría realizar una mejora adicional en el filtro que recorre el objeto (si es una matriz) y luego hace que cada elemento sea una cadena en mayúscula.*

Ahora vamos a usar nuestro nuevo filtro en acción. Nuestro filtro se puede utilizar de dos maneras, ya sea en una plantilla angular o como una función javascript (como referencia angular inyectada).

Javascript

Simplemente inyecte el objeto `$filter` angular en su controlador, luego úselo para recuperar la

función de filtro usando su nombre.

```
app.controller("MyController", function($scope, $filter){
    this.rawString = "Foo";
    this.capsString = $filter("toUppercase")(this.rawString);
});
```

HTML

Para una directiva angular, use el símbolo de la tubería (`|`) seguido del nombre del filtro en la directiva después de la cadena real. Por ejemplo, digamos que tenemos un controlador llamado `MyController` que tiene una cadena llamada `rawString` como un elemento de ella.

```
<div ng-controller="MyController as ctrl">
    <span>Capital rawString: {{ ctrl.rawString | toUppercase }}</span>
</div>
```

Nota del editor: *Angular tiene una serie de filtros incorporados, que incluyen "mayúsculas", y el filtro "a mayúsculas" está pensado solo como una demostración para mostrar fácilmente cómo funcionan los filtros, pero* no *necesita crear su propia función de mayúsculas.*

Filtro personalizado para eliminar valores.

Un caso de uso típico de un filtro es eliminar valores de una matriz. En este ejemplo, pasamos una matriz y eliminamos cualquier nulo que se encuentre en ella, devolviendo la matriz.

```
function removeNulls() {
    return function(list) {
        for (var i = list.length - 1; i >= 0; i--) {
            if (typeof list[i] === 'undefined' ||
                    list[i] === null) {
                list.splice(i, 1);
            }
        }
        return list;
    };
}
```

Eso sería usado en el HTML como

```
{{listOfItems | removeNulls}}
```

o en un controlador como

```
listOfItems = removeNullsFilter(listOfItems);
```

Filtro personalizado para dar formato a los valores.

Otro caso de uso para los filtros es dar formato a un solo valor. En este ejemplo, pasamos un

valor y se nos devuelve un valor booleano verdadero apropiado.

```
function convertToBooleanValue() {
    return function(input) {
        if (typeof input !== 'undefined' &&
                input !== null &&
                (input === true || input === 1 || input === '1' || input
                        .toString().toLowerCase() === 'true')) {
            return true;
        }
        return false;
    };
}
```

Que en el HTML sería usado así:

```
{{isAvailable | convertToBooleanValue}}
```

O en un controlador como:

```
var available = convertToBooleanValueFilter(isAvailable);
```

Realizando filtro en una matriz hijo

Este ejemplo se realizó para demostrar cómo puede realizar un filtro profundo en una matriz *secundaria* sin la necesidad de un filtro personalizado.

Controlador:

```
(function() {
  "use strict";
  angular
    .module('app', [])
    .controller('mainCtrl', mainCtrl);

  function mainCtrl() {
    var vm = this;

    vm.classifications = ["Saloons", "Sedans", "Commercial vehicle", "Sport car"];
    vm.cars = [
       {
          "name":"car1",
          "classifications":[
             {
                "name":"Saloons"
             },
             {
                "name":"Sedans"
             }
          ]
       },
       {
          "name":"car2",
          "classifications":[
             {
```

```
                "name":"Saloons"
             },
             {
                "name":"Commercial vehicle"
             }
          ]
       },
       {
          "name":"car3",
          "classifications":[
             {
                "name":"Sport car"
             },
             {
                "name":"Sedans"
             }
          ]
       }
    ];
  }
})();
```

Ver:

```
<body ng-app="app" ng-controller="mainCtrl as main">
  Filter car by classification:
  <select ng-model="classificationName"
          ng-options="classification for classification in main.classifications"></select>
  <br>
  <ul>
    <li ng-repeat="car in main.cars |
                  filter: { classifications: { name: classificationName } } track by $index"
                  ng-bind-template="{{car.name}} - {{car.classifications | json}}">
    </li>
  </ul>
</body>
```

Compruebe la `DEMO` completa.

Usando filtros en un controlador o servicio

Al inyectar `$filter` , cualquier filtro definido en su módulo Angular puede usarse en controladores, servicios, directivas o incluso en otros filtros.

```
angular.module("app")
  .service("users", usersService)
  .controller("UsersController", UsersController);

function usersService () {
  this.getAll = function () {
    return [{
      id: 1,
      username: "john"
    }, {
      id: 2,
      username: "will"
    }, {
```

```
      id: 3,
      username: "jack"
    }];
  };
}

function UsersController ($filter, users) {
  var orderByFilter = $filter("orderBy");

  this.users = orderByFilter(users.getAll(), "username");
  // Now the users are ordered by their usernames: jack, john, will

  this.users = orderByFilter(users.getAll(), "username", true);
  // Now the users are ordered by their usernames, in reverse order: will, john, jack
}
```

Accediendo a una lista filtrada desde fuera de una repetición ng

Ocasionalmente, querrá acceder al resultado de sus filtros desde fuera de la `ng-repeat` , tal vez para indicar el número de elementos que se han filtrado. Puede hacer esto usando `as [variablename]` sintaxis `as [variablename]` en la `ng-repeat` .

```
<ul>
  <li ng-repeat="item in vm.listItems | filter:vm.myFilter as filtered">
    {{item.name}}
  </li>
</ul>
<span>Showing {{filtered.length}} of {{vm.listItems.length}}</span>
```

Lea Filtros en línea: https://riptutorial.com/es/angularjs/topic/1401/filtros

Capítulo 25: Filtros personalizados

Examples

Ejemplo de filtro simple

Los filtros formatean el valor de una expresión para mostrarla al usuario. Se pueden utilizar en plantillas de vista, controladores o servicios. Este ejemplo crea un filtro (`addZ`) y luego lo usa en una vista. Todo lo que hace este filtro es agregar una 'Z' mayúscula al final de la cadena.

example.js

```
angular.module('main', [])
    .filter('addZ', function() {
            return function(value) {
                return value + "Z";
            }
     })
    .controller('MyController', ['$scope', function($scope) {
        $scope.sample = "hello";
    }])
```

example.html

Dentro de la vista, el filtro se aplica con la siguiente sintaxis: `{ variable | filter}` . En este caso, la variable que definimos en el controlador, `sample` , se filtra por el filtro que creamos, `addZ` .

```
<div ng-controller="MyController">
   <span>{{sample | addZ}}</span>
</div>
```

Rendimiento esperado

```
helloZ
```

Utilice un filtro en un controlador, un servicio o un filtro

Tendrá que inyectar `$filter` :

```
angular
  .module('filters', [])
  .filter('percentage', function($filter) {
    return function (input) {
      return $filter('number')(input * 100) + ' %';
    };
  });
```

Crear un filtro con parámetros.

Por defecto, un filtro tiene un solo parámetro: la variable sobre la que se aplica. Pero puedes pasar más parámetros a la función:

```
angular
  .module('app', [])
  .controller('MyController', function($scope) {
    $scope.example = 0.098152;
  })
  .filter('percentage', function($filter) {
    return function (input, decimals) {
      return $filter('number')(input * 100, decimals) + ' %';
    };
  });
```

Ahora, puede dar una precisión al filtro de `percentage` :

```
<span ng-controller="MyController">{{ example | percentage: 2 }}</span>
=> "9.81 %"
```

... pero otros parámetros son opcionales, aún puede usar el filtro predeterminado:

```
<span ng-controller="MyController">{{ example | percentage }}</span>
=> "9.8152 %"
```

Lea Filtros personalizados en línea: https://riptutorial.com/es/angularjs/topic/2552/filtros-personalizados

Capítulo 26: Filtros personalizados con ES6

Examples

Filtro de tamaño de archivo usando ES6

Aquí tenemos un filtro de tamaño de archivo para describir cómo agregar un filtro de costo a un módulo existente:

```
let fileSize=function (size,unit,fixedDigit) {
return size.toFixed(fixedDigit) + ' '+unit;
};

  let fileSizeFilter=function () {
        return function (size) {
            if (isNaN(size))
                size = 0;

            if (size < 1024)
                return size + ' octets';

            size /= 1024;

            if (size < 1024)
               return fileSize(size,'Ko',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Mo',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Go',2);

            size /= 1024;
            return  fileSize(size,'To',2);
        };
    };
export default fileSizeFilter;
```

El filtro llama al módulo:

```
import fileSizeFilter from 'path...';
let myMainModule =
    angular.module('mainApp', [])
   .filter('fileSize', fileSizeFilter);
```

El código html donde llamamos al filtro:

```
<div ng-app="mainApp">
```

```
    <div>
        <input type="text" ng-model="size" />
    </div>
    <div>
        <h3>Output:</h3>
        <p>{{size| Filesize}}</p>
    </div>
</div>
```

Lea Filtros personalizados con ES6 en línea: https://riptutorial.com/es/angularjs/topic/9421/filtros-personalizados-con-es6

Capítulo 27: Funciones auxiliares incorporadas

Examples

angular.equals

La función `angular.equals` compara y determina si 2 objetos o valores son iguales, `angular.equals` realiza una comparación profunda y devuelve true si y solo si se cumple al menos una de las siguientes condiciones.

> angular.equals (valor1, valor2)

1. Si los objetos o valores pasan la comparación `===`
2. Si ambos objetos o valores son del mismo tipo, y todas sus propiedades también son iguales al usar `angular.equals`
3. Ambos valores son iguales a `NaN`
4. Ambos valores representan el mismo resultado de la expresión regular.

Esta función es útil cuando necesita comparar en profundidad objetos o matrices por sus valores o resultados en lugar de solo referencias.

Ejemplos

```
angular.equals(1, 1) // true
angular.equals(1, 2) // false
angular.equals({}, {}) // true, note that {}==={} is false
angular.equals({a: 1}, {a: 1}) // true
angular.equals({a: 1}, {a: 2}) // false
angular.equals(NaN, NaN) // true
```

angular.isString

La función `angular.isString` devuelve verdadero si el objeto o valor que se le asigna es del tipo `string`

> angular.isString (valor1)

Ejemplos

```
angular.isString("hello") // true
angular.isString([1, 2]) // false
angular.isString(42) // false
```

Este es el equivalente de realizar

```
typeof someValue === "string"
```

angular.isArray

La función `angular.isArray` devuelve verdadero si y solo si el objeto o valor que se le pasa es del tipo `Array` .

> angular.isArray (valor)

Ejemplos

```
angular.isArray([]) // true
angular.isArray([2, 3]) // true
angular.isArray({}) // false
angular.isArray(17) // false
```

Es el equivalente de

```
Array.isArray(someValue)
```

angular.merge

La función angular.merge toma todas las propiedades enumerables del objeto de origen para extender profundamente el objeto de destino.

La función devuelve una referencia al objeto de destino ahora extendido

> angular.merge (destino, fuente)

Ejemplos

```
angular.merge({}, {}) // {}
angular.merge({name: "king roland"}, {password: "12345"})
// {name: "king roland", password: "12345"}
angular.merge({a: 1}, [4, 5, 6]) // {0: 4, 1: 5, 2: 6, a: 1}
angular.merge({a: 1}, {b: {c: {d: 2}}}) // {"a":1,"b":{"c":{"d":2}}}
```

angular.isDefinido y angular.isUndefined

La función `angular.isDefined` prueba un valor si está definido

> angular.isDefined (someValue)

Este es el equivalente de realizar

```
value !== undefined; // will evaluate to true is value is defined
```

Ejemplos

```
angular.isDefined(42) // true
angular.isDefined([1, 2]) // true
angular.isDefined(undefined) // false
```

```
angular.isDefined(null) // true
```

La función `angular.isUndefined` prueba si un valor no está definido (en realidad es lo opuesto a `angular.isDefined`)

> angular.isUndefined (someValue)

Este es el equivalente de realizar

```
value === undefined; // will evaluate to true is value is undefined
```

O solo

```
!angular.isDefined(value)
```

Ejemplos

```
angular.isUndefined(42) // false
angular.isUndefined(undefined) // true
```

angular.isfecha

La función `angular.isDate` devuelve verdadero si y solo si el objeto que se le pasa es del tipo Fecha.

> angular.isDate (valor)

Ejemplos

```
angular.isDate("lone star") // false
angular.isDate(new Date()) // true
```

angular.isNumber

La función `angular.isNumber` devuelve verdadero si y solo si el objeto o valor que se le pasa es del tipo Número, esto incluye + Infinito, -Infinito y NaN

> angular.isNumber (valor)

Esta función no causará un tipo de coerción como

```
"23" == 23 // true
```

Ejemplos

```
angular.isNumber("23") // false
angular.isNumber(23) // true
angular.isNumber(NaN) // true
```

```
angular.isNumber(Infinity) // true
```

Esta función no causará un tipo de coerción como

```
"23" == 23 // true
```

angular.isFunción

La función `angular.isFunction` determina y devuelve verdadero si y solo si el valor pasado es una referencia a una función.

La función devuelve una referencia al objeto de destino ahora extendido

> angular.isFunción (fn)

Ejemplos

```
var onClick = function(e) {return e};
angular.isFunction(onClick); // true

var someArray = ["pizza", "the", "hut"];
angular.isFunction(someArray ); // false
```

angular.toJson

La función `angular.toJson` tomará un objeto y lo serializará en una cadena con formato JSON.

A diferencia de la función nativa `JSON.stringify` , esta función eliminará todas las propiedades que comienzan con `$$` (como angular generalmente prefijos propiedades internas con `$$`)

```
angular.toJson(object)
```

Como los datos deben ser serializados antes de pasar a través de una red, esta función es útil para convertir cualquier información que desee transmitir en JSON.

Esta función también es útil para la depuración, ya que funciona de manera similar a como lo `.toString` un método `.toString` .

Ejemplos:

```
angular.toJson({name: "barf", occupation: "mog", $$somebizzareproperty: 42})
// "{"name":"barf","occupation":"mog"}"
angular.toJson(42)
// "42"
angular.toJson([1, "2", 3, "4"])
// "[1,"2",3,"4"]"
var fn = function(value) {return value}
angular.toJson(fn)
// undefined, functions have no representation in JSON
```

angular.fromJson

La función `angular.fromJson` deserializará una cadena JSON válida y devolverá un objeto o una matriz.

> angular.fromJson (cadena | objeto)

Tenga en cuenta que esta función no se limita a solo cadenas, generará una representación de cualquier objeto que se le pase.

Ejemplos:

```
angular.fromJson("{\"yogurt\": \"strawberries\"}")
// Object {yogurt: "strawberries"}
angular.fromJson('{jam: "raspberries"}')
// will throw an exception as the string is not a valid JSON
angular.fromJson(this)
// Window {external: Object, chrome: Object, _gaq: Y, angular: Object, ng339: 3…}
angular.fromJson([1, 2])
// [1, 2]
typeof angular.fromJson(new Date())
// "object"
```

angular.noop

`angular.noop` es una función que no realiza ninguna operación, usted pasa `angular.noop` cuando necesita proporcionar un argumento de función que no hará nada.

> angular.noop ()

Un uso común de `angular.noop` puede ser proporcionar una devolución de llamada vacía a una función que, de lo contrario, generará un error cuando se le pase algo que no sea una función.

Ejemplo:

```
$scope.onSomeChange = function(model, callback) {
    updateTheModel(model);
    if (angular.isFunction(callback)) {
        callback();
    } else {
        throw new Error("error: callback is not a function!");
    }
};

$scope.onSomeChange(42, function() {console.log("hello callback")});
// will update the model and print 'hello callback'
$scope.onSomeChange(42, angular.noop);
// will update the model
```

Ejemplos adicionales:

```
angular.noop() // undefined
angular.isFunction(angular.noop) // true
```

angular.isObjeto

`angular.isObject` devuelve verdadero si y solo si el argumento que se le pasa es un objeto, esta función también devolverá verdadero para una matriz y devolverá falso para `null` , aunque tipo de `typeof null` es un `object` .

> angular.isObjeto (valor)

Esta función es útil para la verificación de tipos cuando necesita procesar un objeto definido.

Ejemplos:

```
angular.isObject({name: "skroob", job: "president"})
// true
angular.isObject(null)
// false
angular.isObject([null])
// true
angular.isObject(new Date())
// true
angular.isObject(undefined)
// false
```

angular.iselemento

`angular.isElement` devuelve true si el argumento que se le pasa es un elemento DOM o un elemento envuelto jQuery.

> angular.isElemento (elem)

Esta función es útil para escribir check si un argumento pasado es un elemento antes de ser procesado como tal.

Ejemplos:

```
angular.isElement(document.querySelector("body"))
// true
angular.isElement(document.querySelector("#some_id"))
// false if "some_id" is not using as an id inside the selected DOM
angular.isElement("<div></div>")
// false
```

angular.copy

La función `angular.copy` toma un objeto, una matriz o un valor y crea una copia profunda de él.

> angular.copy ()

Ejemplo:

Objetos:

```
let obj = {name: "vespa", occupation: "princess"};
let cpy = angular.copy(obj);
cpy.name = "yogurt"
// obj = {name: "vespa", occupation: "princess"}
// cpy = {name: "yogurt", occupation: "princess"}
```

Arrays:

```
var w = [a, [b, [c, [d]]]];
var q = angular.copy(w);
// q = [a, [b, [c, [d]]]]
```

En el ejemplo anterior, `angular.equals(w, q)` se evaluará como verdadero porque `.equals` comprueba la igualdad por valor. sin embargo, `w === q` se evaluará como falso porque la comparación estricta entre objetos y matrices se realiza por referencia.

identidad angular

La función `angular.identity` devuelve el primer argumento que se le pasa.

> identidad angular (argumento)

Esta función es útil para la programación funcional, puede proporcionar esta función como predeterminada en caso de que no se haya pasado una función esperada.

Ejemplos:

```
angular.identity(42) // 42
```

```
var mutate = function(fn, num) {
    return angular.isFunction(fn) ? fn(num) : angular.identity(num)
}

mutate(function(value) {return value-7}, 42) // 35
mutate(null, 42) // 42
mutate("mount. rushmore", 42) // 42
```

angular.para cada

El `angular.forEach` acepta un objeto y una función de iterador. A continuación, ejecuta la función de iterador sobre cada propiedad / valor enumerable del objeto. Esta función también funciona en matrices.

Al igual que la versión JS de `Array.prototype.forEach` La función no itera sobre las propiedades heredadas (propiedades de prototipo), sin embargo, la función no intentará procesar un `null` o un valor `undefined` y simplemente lo devolverá.

> angular.forEach (objeto, función (valor, clave) {// función});

Ejemplos:

```
angular.forEach({"a": 12, "b": 34}, (value, key) => console.log("key: " + key + ", value: " +
value))
// key: a, value: 12
// key: b, value: 34
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(key))
// will print the array indices: 1, 2, 3, 4, 5
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(value))
// will print the array values: 2, 4, 6, 7, 10
angular.forEach(undefined, (value, key) => console.log("key: " + key + ", value: " + value))
// undefined
```

Lea Funciones auxiliares incorporadas en línea:
https://riptutorial.com/es/angularjs/topic/3032/funciones-auxiliares-incorporadas

Capítulo 28: Impresión

Observaciones

Crear una clase ng-hide en el archivo css. ng-show / hide no funcionará sin la clase.

Más detalles

Examples

Servicio de impresión

Servicio:

```
angular.module('core').factory('print_service', ['$rootScope', '$compile', '$http',
'$timeout','$q',
    function($rootScope, $compile, $http, $timeout,$q) {

        var printHtml = function (html) {
            var deferred = $q.defer();
            var hiddenFrame = $('<iframe style="display:
none"></iframe>').appendTo('body')[0];

            hiddenFrame.contentWindow.printAndRemove = function() {
                hiddenFrame.contentWindow.print();
                $(hiddenFrame).remove();
                deferred.resolve();
            };

            var htmlContent =   "<!doctype html>"+
                                "<html>"+
                                    '<head><link rel="stylesheet" type="text/css"
href="/style/css/print.css"/></head>'+
                                    '<body onload="printAndRemove();">' +
                                        html +
                                    '</body>'+
                                "</html>";

            var doc = hiddenFrame.contentWindow.document.open("text/html", "replace");
            doc.write(htmlContent);
            doc.close();
            return deferred.promise;
        };

        var openNewWindow = function (html) {
            var newWindow = window.open("debugPrint.html");
            newWindow.addEventListener('load', function(){
                $(newWindow.document.body).html(html);
            }, false);
        };

        var print = function (templateUrl, data) {

            $rootScope.isBeingPrinted = true;
```

```
            $http.get(templateUrl).success(function(template){
                var printScope = $rootScope.$new()
                angular.extend(printScope, data);
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if(printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint, 1000);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html());
                        printScope.$destroy();
                    }
                };
                waitForRenderAndPrint();
            });
        };

        var printFromScope = function (templateUrl, scope, afterPrint) {
            $rootScope.isBeingPrinted = true;
            $http.get(templateUrl).then(function(response){
                var template = response.data;
                var printScope = scope;
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if (printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html()).then(function() {
                            $rootScope.isBeingPrinted = false;
                            if (afterPrint) {
                                afterPrint();
                            }
                        });
                    }
                };
                waitForRenderAndPrint();
            });
        };

        return {
            print : print,
            printFromScope : printFromScope
        }
    }
]);
```

Controlador :

```
var template_url = '/views/print.client.view.html';
print_service.printFromScope(template_url,$scope,function(){
      // Print Completed
});
```

Lea Impresión en línea: https://riptutorial.com/es/angularjs/topic/6750/impresion

Capítulo 29: Interceptor HTTP

Introducción

El servicio $ http de AngularJS nos permite comunicarnos con un backend y realizar solicitudes HTTP. Hay casos en los que queremos capturar cada solicitud y manipularla antes de enviarla al servidor. Otras veces nos gustaría capturar la respuesta y procesarla antes de completar la llamada. El manejo global de errores de http puede ser también un buen ejemplo de tal necesidad. Los interceptores son creados exactamente para tales casos.

Examples

Empezando

El servicio `$http` integrado de Angular nos permite enviar solicitudes HTTP. Con frecuencia, surge la necesidad de hacer cosas antes o después de una solicitud, por ejemplo, agregar a cada solicitud un token de autenticación o crear una lógica genérica de manejo de errores.

Generador interactivo de HTTP paso a paso

Crea un archivo HTML con el siguiente contenido:

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular Interceptor Sample</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
  <script src="app.js"></script>
  <script src="appController.js"></script>
  <script src="genericInterceptor.js"></script>
</head>
<body ng-app="interceptorApp">
    <div ng-controller="appController as vm">
        <button ng-click="vm.sendRequest()">Send a request</button>
    </div>
</body>
</html>
```

Agrega un archivo JavaScript llamado 'app.js':

```
var interceptorApp = angular.module('interceptorApp', []);

interceptorApp.config(function($httpProvider) {
        $httpProvider.interceptors.push('genericInterceptor');
});
```

Agregue otro llamado 'appController.js':

```
(function() {
```

```
    'use strict';

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

        vm.sendRequest = function(){
            $http.get('http://google.com').then(function(response){
                console.log(response);
            });
        };
    }

    angular.module('interceptorApp').controller('appController',['$http', appController]);
})();
```

Y finalmente el archivo que contiene el interceptor 'genericInterceptor.js':

```
(function() {
    "use strict";

    function genericInterceptor($q) {
        this.responseError = function (response) {
            return $q.reject(response);
        };

        this.requestError = function(request){
            if (canRecover(rejection)) {
                return responseOrNewPromise
            }
            return $q.reject(rejection);
        };

        this.response = function(response){
            return response;
        };

        this.request = function(config){
            return config;
        }
    }

    angular.module('interceptorApp').service('genericInterceptor', genericInterceptor);
})();
```

El 'genéricoInterceptor' cubre las posibles funciones que podemos anular agregando un comportamiento adicional a nuestra aplicación.

Mensaje flash en respuesta utilizando el interceptor http

En el archivo de vista

En el html base (index.html) donde usualmente incluimos los scripts angulares o el html que se comparte en la aplicación, dejamos un elemento div vacío, los mensajes flash aparecerán dentro de este elemento div

```
<div class="flashmessage" ng-if="isVisible">
    {{flashMessage}}
</div>
```

Archivo de comandos

En el método de configuración del módulo angular, inyecte el httpProvider, el httpProvider tiene una propiedad de matriz de interceptor, presione el interceptor personalizado. En el ejemplo actual, el interceptor personalizado solo intercepta la respuesta y llama a un método asociado a rootScope.

```
var interceptorTest = angular.module('interceptorTest', []);

    interceptorTest.config(['$httpProvider',function ($httpProvider) {

        $httpProvider.interceptors.push(["$rootScope",function ($rootScope) {
            return {     //intercept only the response
                        'response': function (response)
                                    {

$rootScope.showFeedBack(response.status,response.data.message);

                                      return response;
                                    }
                   };
        }]);

    }])
```

Dado que solo los proveedores pueden ser inyectados en el método de configuración de un módulo angular (es decir, httpProvider y no en el de rootscope), declare el método conectado al rootscope dentro del método de ejecución del módulo angular.

También muestre el mensaje dentro de $ timeout para que el mensaje tenga la propiedad flash, que está desapareciendo después de un tiempo límite. En nuestro ejemplo sus 3000 ms.

```
interceptorTest.run(["$rootScope","$timeout",function($rootScope,$timeout){
     $rootScope.showFeedBack = function(status,message){

        $rootScope.isVisible = true;
        $rootScope.flashMessage = message;
        $timeout(function(){$rootScope.isVisible = false },3000)
    }
}]);
```

Errores comunes

Al intentar inyectar **$ rootScope o cualquier otro servicio** dentro del método de **configuración** del módulo angular, el ciclo de vida de la aplicación angular no permite eso y se generará un error de proveedor desconocido. Solo los **proveedores** pueden ser inyectados en el método de **configuración** del módulo angular.

Lea Interceptor HTTP en línea: https://riptutorial.com/es/angularjs/topic/6484/interceptor-http

Capítulo 30: Inyección de dependencia

Sintaxis

- myApp.controller ('MyController', function ($ scope) {...}); // código no minificado
- myApp.controller ('MyController', ['$ scope', function ($ scope) {...}]); // apoyo a la minificación
- función MyController () {}

  MyController. $ Inject = ['$ scope'];

  myApp.controller ('MyController', MyController); // $ inyectar anotación
- $ injector.get ('inyectable'); // inyección dinámica / en tiempo de ejecución

Observaciones

Los proveedores no pueden ser inyectados en bloques de `run` .

Los servicios o valores no se pueden inyectar en bloques de `config` .

Asegúrese de anotar sus inyecciones para que su código no se rompa en la minificación.

Examples

Inyecciones

El ejemplo más simple de una inyección en una aplicación Angular: inyectar `$scope` a un `Controller` Angular:

```
angular.module('myModule', [])
.controller('myController', ['$scope', function($scope) {
    $scope.members = ['Alice', 'Bob'];
    ...
}])
```

Lo anterior ilustra una inyección de un `$scope` en un `controller` , pero es lo mismo si se inyecta cualquier módulo en otro. El proceso es el mismo.

El sistema de Angular se encarga de resolver las dependencias por usted. Si crea un servicio, por ejemplo, puede listarlo como en el ejemplo anterior y estará disponible para usted.

Puede usar DI - inyección de dependencia, donde quiera que esté definiendo un componente.

Tenga en cuenta que en el ejemplo anterior usamos lo que se denomina "Anotación de matriz en

línea". Es decir, escribimos explícitamente como cadenas los nombres de nuestras dependencias. Lo hacemos para evitar que la aplicación se rompa cuando el código se minimiza para Producción. La reducción de código cambia los nombres de las variables (pero no de las cadenas), lo que rompe la inyección. Mediante el uso de cadenas, Angular sabe qué dependencias queremos.

Muy importante: el orden de los nombres de las cadenas debe ser el mismo que el de los parámetros en la función.

Existen herramientas que automatizan este proceso y se encargan de esto por usted.

Inyecciones dinamicas

También hay una opción para solicitar componentes dinámicamente. Puedes hacerlo usando el servicio `$injector` :

```
myModule.controller('myController', ['$injector', function($injector) {
    var myService = $injector.get('myService');
}]);
```

Nota: si bien este método se puede usar para prevenir el problema de dependencia circular que podría interrumpir su aplicación, no se considera una buena práctica evitar el problema al usarlo. La dependencia circular generalmente indica que hay una falla en la arquitectura de su aplicación, y usted debe abordar eso en su lugar.

$ inyectar propiedad anotación

De manera equivalente, podemos usar la anotación de propiedad `$inject` para lograr lo mismo que arriba:

```
var MyController = function($scope) {
  // ...
}
MyController.$inject = ['$scope'];
myModule.controller('MyController', MyController);
```

Cargar dinámicamente el servicio AngularJS en vainilla JavaScript

Puede cargar los servicios de AngularJS en JavaScript de vainilla utilizando el método del `injector()` AngularJS. Cada elemento jqLite recuperado llamando `angular.element()` tiene un método `injector()` que se puede usar para recuperar el inyector.

```
var service;
var serviceName = 'myService';

var ngAppElement = angular.element(document.querySelector('[ng-app],[data-ng-app]') ||
document);
var injector = ngAppElement.injector();

if(injector && injector.has(serviceNameToInject)) {
```

```
    service = injector.get(serviceNameToInject);
}
```

En el ejemplo anterior, intentamos recuperar el elemento jqLite que contiene la raíz de la aplicación AngularJS (`ngAppElement`). Para hacer eso, usamos el método `angular.element()` , buscando un elemento DOM que contenga el atributo `ng-app` o `data-ng-app` o, si no existe, recurrimos al elemento del `document` . Usamos `ngAppElement` para recuperar la instancia del inyector (con `ngAppElement.injector()`). La instancia del inyector se usa para verificar si el servicio a inyectar existe (con `injector.has()`) y luego para cargar el servicio (con `injector.get()`) dentro de la variable de `service` .

Lea Inyección de dependencia en línea: https://riptutorial.com/es/angularjs/topic/1582/inyeccion-de-dependencia

Capítulo 31: Migración a Angular 2+

Introducción

AngularJS ha sido completamente reescrito usando el lenguaje TypeScript y renombrado a simplemente Angular.

Se puede hacer mucho en una aplicación AngularJS para facilitar el proceso de migración. Como dice la guía de actualización oficial , se pueden realizar varios "pasos de preparación" para refactorizar su aplicación, haciéndola mejor y más cercana al nuevo estilo Angular.

Examples

Convertir su aplicación AngularJS en una estructura orientada a componentes

En el nuevo marco angular, los **componentes** son los bloques de construcción principales que componen la interfaz de usuario. Por lo tanto, uno de los primeros pasos que ayuda a migrar una aplicación AngularJS al nuevo Angular es refactorizarlo en una estructura más orientada a los componentes.

Los componentes también se introdujeron en el antiguo AngularJS a partir de la versión **1.5+** . El uso de Componentes en una aplicación AngularJS no solo acercará su estructura al nuevo Angular 2+, sino que también lo hará más modular y más fácil de mantener.

Antes de continuar, recomiendo consultar la página de documentación oficial de AngularJS sobre Componentes , donde se explican bien sus ventajas y usos.

Preferiría mencionar algunos consejos sobre cómo convertir el antiguo código orientado a `ng-controller` al nuevo estilo orientado a `component` .

Comienza a dividir tu aplicación en componentes

Todas las aplicaciones orientadas a componentes tienen generalmente uno o pocos componentes que incluyen otros subcomponentes. Entonces, ¿por qué no crear el primer componente que simplemente contendrá su aplicación (o una gran parte de ella)?

Supongamos que tenemos un código asignado a un controlador, denominado `UserListController` , y queremos crear un componente del mismo, al que llamaremos `UserListComponent` .

HTML actual:

```
<div ng-controller="UserListController as listctrl">
    <ul>
        <li ng-repeat="user in myUserList">
            {{ user }}
        </li>
    </ul>
</div>
```

JavaScript actual:

```
app.controller("UserListController", function($scope, SomeService) {

    $scope.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

nuevo HTML:

```
<user-list></user-list>
```

nuevo JavaScript:

```
app.component("UserList", {
    templateUrl: 'user-list.html',
    controller: UserListController
});

function UserListController(SomeService) {

    this.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

Observe cómo ya no estamos inyectando `$scope` en la función del controlador y ahora estamos declarando `this.myUserList` lugar de `$scope.myUserList` ;

Nuevo archivo de plantilla `user-list.component.html` :

```
<ul>
    <li ng-repeat="user in $ctrl.myUserList">
        {{ user }}
    </li>
</ul>
```

Observe cómo nos estamos refiriendo ahora a la variable `myUserList` , que pertenece al

controlador, usando `$ctrl.myUserList` de html en lugar de `$scope.myUserList` .

Esto se debe a que, como probablemente haya descubierto después de leer la documentación, `$ctrl` en la plantilla ahora se refiere a la función del controlador.

¿Qué pasa con los controladores y las rutas?

En caso de que su controlador estuviera vinculado a la plantilla usando el sistema de enrutamiento en lugar de `ng-controller` , entonces si tiene algo como esto:

```
$stateProvider
  .state('users', {
    url: '/users',
    templateUrl: 'user-list.html',
    controller: 'UserListController'
  })
  // ..
```

simplemente puede cambiar su declaración de estado a:

```
$stateProvider
    .state('users', {
      url: '/',
      template: '<user-list></user-list>'
    })
    // ..
```

¿Que sigue?

Ahora que tiene un componente que contiene su aplicación (ya sea que contenga la aplicación completa o una parte de ella, como una vista), ahora debe comenzar a dividir su componente en múltiples componentes anidados, envolviendo partes de ella en nuevos subcomponentes , y así.

Deberías comenzar a usar las características de los componentes como

- **Entradas** y enlaces de **salidas**
- **ganchos de ciclo de vida** como `$onInit()` , `$onChanges()` , etc ...

Después de leer la documentación del Componente , ya debe saber cómo usar todas las funciones de esos componentes, pero si necesita un ejemplo concreto de una aplicación realmente simple, puede verificar esto .

Además, si dentro del controlador de su componente tiene algunas funciones que contienen una gran cantidad de código lógico, una buena idea puede ser considerar mover esa lógica a los servicios .

Conclusión

La adopción de un enfoque basado en componentes empuja su AngularJS un paso más hacia la migración al nuevo marco angular, pero también lo hace mejor y mucho más modular.

Por supuesto, hay muchos otros pasos que puede hacer para ir más allá en la nueva dirección Angular 2+, que enumeraré en los siguientes ejemplos.

Introducción a los módulos Webpack y ES6.

Al utilizar un ***cargador de módulos*** como Webpack , podemos beneficiarnos del sistema de módulos integrado disponible en **ES6** (así como en **TypeScript**). Luego podemos usar las funciones de importación y exportación que nos permiten especificar qué partes de código podemos compartir entre las diferentes partes de la aplicación.

Cuando luego llevamos nuestras aplicaciones a producción, los cargadores de módulos también hacen que sea más fácil empaquetarlos en paquetes de producción con baterías incluidas.

Lea Migración a Angular 2+ en línea: https://riptutorial.com/es/angularjs/topic/9942/migracion-a-angular-2plus

Capítulo 32: Módulos

Examples

Módulos

El módulo sirve como contenedor de diferentes partes de su aplicación, como controladores, servicios, filtros, directivas, etc. Los módulos pueden ser referenciados por otros módulos a través del mecanismo de inyección de dependencia de Angular.

Creando un módulo:

```
angular
    .module('app', []);
```

La matriz `[]` pasada en el ejemplo anterior es la *lista de módulos de la* `app` depende, si no hay dependencias, pasamos la matriz vacía, es decir, `[]` .

Inyectar un módulo como una dependencia de otro módulo:

```
angular.module('app', [
    'app.auth',
    'app.dashboard'
]);
```

Haciendo referencia a un módulo:

```
angular
    .module('app');
```

Módulos

El módulo es un contenedor para varias partes de sus aplicaciones: controlador, servicios, filtros, directivas, etc.

¿Por qué usar módulos?
La mayoría de las aplicaciones tienen un método principal que crea una instancia y conecta las diferentes partes de la aplicación.
Las aplicaciones angulares no tienen método principal.
Pero en AngularJs el proceso declarativo es fácil de entender y uno puede empaquetar código como módulos reutilizables.
Los módulos se pueden cargar en cualquier orden porque los módulos demoran la ejecución.

declarar un módulo

```
var app = angular.module('myApp', []);
// Empty array is list of modules myApp is depends on.
```

```
// if there are any required dependancies,
// then you can add in module, Like ['ngAnimate']

app.controller('myController', function() {

  // write your business logic here
});
```

Módulo de carga y dependencias

1. Bloques de configuración: se ejecutan durante la fase de configuración y proveedor.

```
angular.module('myModule', []).
config(function(injectables) {
  // here you can only inject providers in to config blocks.
});
```

2. Ejecutar bloques: se ejecuta después de que se crea el inyector y se utiliza para iniciar la aplicación.

```
angular.module('myModule', []).
run(function(injectables) {
  // here you can only inject instances in to config blocks.
});
```

Lea Módulos en línea: https://riptutorial.com/es/angularjs/topic/844/modulos

Capítulo 33: MVC angular

Introducción

En **AngularJS,** el patrón **MVC** se implementa en JavaScript y HTML. La vista se define en HTML, mientras que el modelo y el controlador se implementan en JavaScript. Hay varias maneras en que estos componentes pueden unirse en AngularJS, pero la forma más simple comienza con la vista.

Examples

La vista estática con controlador

mvc demo

Hola Mundo

Definición de la función del controlador

```
var indexController = myApp.controller("indexController", function ($scope) {
    // Application logic goes here
});
```

Añadiendo información al modelo.

```
var indexController = myApp.controller("indexController", function ($scope) {
    // controller logic goes here
    $scope.message = "Hello Hacking World"
});
```

Lea MVC angular en línea: https://riptutorial.com/es/angularjs/topic/8667/mvc-angular

Capítulo 34: ng-repetir

Introducción

La directiva `ngRepeat` una instancia de una plantilla una vez por elemento de una colección. La colección debe ser una matriz o un objeto. Cada instancia de plantilla tiene su propio ámbito, donde la variable de bucle dada se establece en el elemento de colección actual, y `$index` se establece en el índice o clave del elemento.

Sintaxis

- `<element ng-repeat="expression"></element>`
- `<div ng-repeat="(key, value) in myObj">...</div>`
- `<div ng-repeat="variable in expression">...</div>`

Parámetros

Variable	Detalles
`$index`	*número* iterador de desplazamiento del elemento repetido (0..length-1)
`$first`	*boolean* true si el elemento repetido es el primero en el iterador.
`$middle`	*boolean* true si el elemento repetido está entre el primero y el último en el iterador.
`$last`	*boolean* true si el elemento repetido es el último en el iterador.
`$even`	*boolean* true si la posición del iterador `$index` es par (en caso contrario, false).
`$odd`	*boolean* true si la posición del iterador `$index` es impar (en caso contrario, false).

Observaciones

AngularJS proporciona estos parámetros como variables especiales que están disponibles en la expresión ng-repeat y en cualquier lugar dentro de la etiqueta HTML en la que vive ng-repeat.

Examples

Iterando sobre las propiedades del objeto.

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

Por ejemplo

```
<div ng-repeat="n in [42, 42, 43, 43]">
  {{n}}
</div>
```

Seguimiento y duplicados

`ngRepeat` utiliza [$ watchCollection] para detectar cambios en la colección. Cuando ocurre un cambio, `ngRepeat` realiza los cambios correspondientes en el DOM:

- Cuando se agrega un elemento, se agrega una nueva instancia de la plantilla al DOM.
- Cuando se elimina un elemento, su instancia de plantilla se elimina del DOM.
- Cuando los artículos se reordenan, sus respectivas plantillas se reordenan en el DOM.

Duplicados

- `track by` cualquier lista que pueda incluir valores duplicados.
- `track by` también acelera los cambios de lista significativamente.
- Si no usa la función de `track by` en este caso, obtendrá el error: `[ngRepeat:dupes]`

```
$scope.numbers = ['1','1','2','3','4'];

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

ng-repeat-start + ng-repeat-end

AngularJS 1.2 `ng-repeat` maneja múltiples elementos con `ng-repeat-start` y `ng-repeat-end` :

```
// table items
$scope.tableItems = [
    {
        row1: 'Item 1: Row 1',
        row2: 'Item 1: Row 2'
    },
    {
        row1: 'Item 2: Row 1',
        row2: 'Item 2: Row 2'
    }
];

// template
<table>
    <th>
        <td>Items</td>
    </th>
    <tr ng-repeat-start="item in tableItems">
        <td ng-bind="item.row1"></td>
    </tr>
    <tr ng-repeat-end>
        <td ng-bind="item.row2"></td>
    </tr>
```

```
</table>
```

Salida:

Artículos
Artículo 1: Fila 1
Artículo 1: Fila 2
Artículo 2: Fila 1
Artículo 2: Fila 2

Lea ng-repetir en línea: https://riptutorial.com/es/angularjs/topic/8118/ng-repetir

Capítulo 35: ng-view

Introducción

ng-view es una de las directivas integradas que Angular utiliza como contenedor para cambiar entre vistas. {info} ngRoute ya no forma parte del archivo base angular.js, por lo que deberá incluir el archivo angular-route.js después de su archivo de base angular javascript. Podemos configurar una ruta utilizando la función "cuándo" del $ routeProvider. Primero debemos especificar la ruta, luego, en un segundo parámetro, proporcionar un objeto con una propiedad templateUrl y una propiedad de controlador.

Examples

ng-view

`ng-view` es una directiva que se usa con `$route` para representar una vista parcial en el diseño de la página principal. Aquí, en este ejemplo, Index.html es nuestro archivo principal y cuando el usuario llega a la ruta "/", templateURL home.html se representará en Index.html donde se menciona `ng-view` .

```
angular.module('ngApp', ['ngRoute'])

.config(function($routeProvider){
  $routeProvider.when("/",
    {
      templateUrl: "home.html",
      controller: "homeCtrl"
    }
  );
});

angular.module('ngApp').controller('homeCtrl',['$scope', function($scope) {
  $scope.welcome= "Welcome to stackoverflow!";
}]);

//Index.html
<body ng-app="ngApp">
    <div ng-view></div>
</body>

//Home Template URL or home.html
<div><h2>{{welcome}}</h2></div>
```

Registro de navegación

1. Inyectamos el módulo en la aplicación.

```
  var Registration=angular.module("myApp",["ngRoute"]);
```

2. ahora usamos $ routeProvider de "ngRoute"

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

});
```

3. finalmente, integrando la ruta, definimos el enrutamiento "/ add" a la aplicación en caso de que la aplicación obtenga "/ add", se desvía a regi.htm

```
 Registration.config(function($routeProvider) {
    $routeProvider
    .when("/add", {
        templateUrl : "regi.htm"
    })
});
```

Lea ng-view en línea: https://riptutorial.com/es/angularjs/topic/8833/ng-view

Capítulo 36: Opciones de enlaces AngularJS (`=`, `@`, `&` etc.)

Observaciones

Usa este plunker para jugar con ejemplos.

Examples

@ enlace unidireccional, atributo de enlace.

Pase un valor literal (no un objeto), como una cadena o un número.

El ámbito secundario obtiene su propio valor, si actualiza el valor, el ámbito principal tiene su propio valor anterior (el ámbito secundario no puede modificar el valor del alcance de parens). Cuando se cambia el valor del ámbito principal, también se cambiará el valor del ámbito primario. Todas las interpolaciones aparecen cada vez en una llamada de resumen, no solo en la creación de directivas.

```
<one-way text="Simple text." <!-- 'Simple text.' -->
         simple-value="123" <!-- '123' Note, is actually a string object. -->
         interpolated-value="{{parentScopeValue}}" <!-- Some value from parent scope. You
can't change parent scope value, only child scope value. Note, is actually a string object. --
>
         interpolated-function-value="{{parentScopeFunction()}}" <!-- Executes parent scope
function and takes a value. -->

         <!-- Unexpected usage. -->
         object-item="{{objectItem}}" <!-- Converts object|date to string. Result might be:
'{"a":5,"b":"text"}'. -->
         function-item="{{parentScopeFunction}}"> <!-- Will be an empty string. -->
</one-way>
```

= enlace bidireccional.

Al pasar un valor por referencia, desea compartir el valor entre ambos ámbitos y manipularlos desde ambos ámbitos. No debe usar {{...}} para la interpolación.

```
<two-way text="'Simple text.'" <!-- 'Simple text.' -->
         simple-value="123" <!-- 123 Note, is actually a number now. -->
         interpolated-value="parentScopeValue" <!-- Some value from parent scope. You may
change it in one scope and have updated value in another. -->
         object-item="objectItem" <!-- Some object from parent scope. You may change object
properties in one scope and have updated properties in another. -->

         <!-- Unexpected usage. -->
         interpolated-function-value="parentScopeFunction()" <!-- Will raise an error. -->
         function-item="incrementInterpolated"> <!-- Pass the function by reference and you
may use it in child scope. -->
```

```
</two-way>
```

Pasar la función por referencia es una mala idea: para permitir que el alcance cambie la definición de una función, y se crearán dos observadores innecesarios, debe minimizar el conteo de observadores.

y función de enlace, expresión de enlace.

Pasar un método en una directiva. Proporciona una forma de ejecutar una expresión en el contexto del ámbito principal. El método se ejecutará en el ámbito del elemento primario, puede pasar algunos parámetros desde el ámbito secundario allí. No debe usar {{...}} para la interpolación. Cuando utiliza & en una directiva, genera una función que devuelve el valor de la expresión evaluada en el ámbito principal (no es lo mismo que = donde acaba de pasar una referencia).

```
<expression-binding interpolated-function-value="incrementInterpolated(param)" <!--
interpolatedFunctionValue({param: 'Hey'}) will call passed function with an argument. -->
                    function-item="incrementInterpolated" <!-- functionItem({param: 'Hey'})()
will call passed function, but with no possibility set up a parameter. -->
                    text="'Simple text.'"  <!-- text() == 'Simple text.'-->
                    simple-value="123" <!-- simpleValue() == 123 -->
                    interpolated-value="parentScopeValue" <!-- interpolatedValue() == Some
value from parent scope. -->
                    object-item="objectItem"> <!-- objectItem() == Object item from parent
scope. -->
</expression-binding>
```

Todos los parámetros serán envueltos en funciones.

Encuadernación disponible a través de una simple muestra.

```
angular.component("SampleComponent", {
  bindings: {
     title: '@',
     movies: '<',
     reservation: "=",
     processReservation: "&"
  }
});
```

Aquí tenemos todos los elementos vinculantes.

@ indica que necesitamos un **enlace** muy **básico** , desde el ámbito principal al ámbito secundario, sin ningún observador, de ninguna manera. Cada actualización en el ámbito primario se mantendría en el ámbito primario, y cualquier actualización en el ámbito primario no se comunicaría al ámbito primario.

< indica un **enlace unidireccional** . Las actualizaciones en el ámbito principal se propagarían al ámbito secundario, pero cualquier actualización en el ámbito secundario no se aplicaría al ámbito primario.

= ya se conoce como un enlace de dos vías. Cada actualización en el ámbito primario se aplicaría en los secundarios, y cada actualización secundaria se aplicaría al ámbito primario.

& ahora se utiliza para un enlace de salida. De acuerdo con la documentación del componente, se debe utilizar para hacer referencia al método de alcance principal. En lugar de manipular el alcance de los niños, ¡simplemente llame al método principal con los datos actualizados!

Vincular atributo opcional

```
bindings: {
   mandatory: '='
   optional: '=?',
   foo: '=?bar'
}
```

Los atributos opcionales deben marcarse con un signo de interrogación: `=?` o `=?bar` . Es protección para `($compile:nonassign)` excepción.

Lea Opciones de enlaces AngularJS (`=`, `@`, `&` etc.) en línea: https://riptutorial.com/es/angularjs/topic/6149/opciones-de-enlaces-angularjs--------------amp---etc--

Capítulo 37: Perfil de rendimiento

Examples

Todo sobre perfilado

¿Qué es el perfil?

Por definición, la creación de perfiles es una forma de análisis dinámico de programas que mide, por ejemplo, el espacio (memoria) o la complejidad del tiempo de un programa, el uso de instrucciones particulares o la frecuencia y duración de las llamadas a funciones.

¿Por qué es necesario?

La creación de perfiles es importante porque no puede optimizar de manera efectiva hasta que sepa en qué está gastando la mayor parte de su programa. Sin medir el tiempo de ejecución de su programa (creación de perfiles), no sabrá si realmente lo ha mejorado.

Herramientas y Técnicas:

1. Las herramientas de desarrollo integradas de Chrome.

   Esto incluye un conjunto completo de herramientas que se utilizarán para la creación de perfiles. Puede profundizar para descubrir cuellos de botella en su archivo javascript, archivos css, animaciones, consumo de CPU, fugas de memoria, red, seguridad, etc.

   Haga una grabación de la línea de tiempo y busque eventos de Evaluate Script sospechosamente largos. Si encuentra alguno, puede habilitar el Perfilador de JS y volver a hacer su grabación para obtener información más detallada sobre exactamente a qué funciones de JS se llamó y cuánto tiempo tomó cada una. Lee mas...

2. FireBug (usar con Firefox)

3. Dynatrace (usar con IE)

4. Batarang (usar con Chrome)

   Es un complemento obsoleto para el navegador Chrome, aunque es estable y se puede usar para monitorear modelos, rendimiento y dependencias para una aplicación angular. Funciona bien para aplicaciones de pequeña escala y puede darle una idea de lo que la variable de alcance tiene en varios niveles. Le informa sobre observadores activos, expresiones de observación, colecciones de observación en la aplicación.

5. Observador (usar con Chrome)

   Interfaz de usuario agradable y simplista para contar el número de observadores en una aplicación Angular.

6. Use el siguiente código para averiguar manualmente el número de observadores en su aplicación angular (crédito para @Words Like Jared Number of watchers)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

7. Hay varias herramientas / sitios web disponibles en línea que facilitan una amplia gama de funcionalidades para crear un perfil de su aplicación.

   Uno de estos sitios es: https://www.webpagetest.org/

   Con esto, puede ejecutar una prueba gratuita de velocidad de sitios web desde múltiples ubicaciones en todo el mundo utilizando navegadores reales (IE y Chrome) y a velocidades de conexión reales de los consumidores. Puede ejecutar pruebas simples o realizar pruebas avanzadas que incluyen transacciones de varios pasos, captura de video, bloqueo de contenido y mucho más.

Próximos pasos:

Hecho con perfilado. Sólo te lleva a mitad del camino. La siguiente tarea es convertir sus descubrimientos en elementos de acción para optimizar su aplicación. Consulte esta documentación sobre cómo puede mejorar el rendimiento de su aplicación angular con simples trucos.

Feliz codificación :)

Lea Perfil de rendimiento en línea: https://riptutorial.com/es/angularjs/topic/7033/perfil-de-rendimiento

Capítulo 38: Perfilado y Rendimiento

Examples

7 mejoras simples de rendimiento

1) Utilice ng-repetir con moderación

El uso de `ng-repeat` en las vistas generalmente produce un rendimiento deficiente, especialmente cuando hay `ng-repeat` anidadas.

Esto es super lento!

```
<div ng-repeat="user in userCollection">
  <div ng-repeat="details in user">
    {{details}}
  </div>
</div>
```

Trate de evitar las repeticiones anidadas tanto como sea posible. Una forma de mejorar el rendimiento de `ng-repeat` es usar `track by $index` (o algún otro campo de identificación). Por defecto, `ng-repeat` rastrea todo el objeto. Con `track by` , Angular mira el objeto solo por `$index` o id de objeto.

```
<div ng-repeat="user in userCollection track by $index">
  {{user.data}}
</div>
```

Use otros enfoques como paginación , pergaminos virtuales , infinitos o limitTo: comience siempre que sea posible para evitar iterar en grandes colecciones.

---

2) Atar una vez

Angular tiene enlace de datos bidireccional. Viene con el costo de ser lento si se usa demasiado.

Rendimiento más lento

```
<!-- Default data binding has a performance cost -->
<div>{{ my.data }}</div>
```

Rendimiento más rápido (AngularJS> = 1.3)

```
<!-- Bind once is much faster -->
<div>{{ ::my.data }}</div>

<div ng-bind="::my.data"></div>
```

```
<!-- Use single binding notation in ng-repeat where only list display is needed  -->
<div ng-repeat="user in ::userCollection">
  {{::user.data}}
</div>
```

El uso de la notación "unir una vez" le dice a Angular que espere a que el valor se estabilice después de la primera serie de ciclos de digestión. Angular utilizará ese valor en el DOM, luego eliminará a todos los observadores para que se convierta en un valor estático y ya no esté vinculado al modelo.

El `{{}}` es mucho más lento.

Este `ng-bind` es una directiva y colocará un observador en la variable pasada. Por lo tanto, el `ng-bind` solo se aplicará cuando el valor pasado realmente cambie.

Por otro lado, los corchetes se verán sucios y se actualizarán en cada `$digest` , incluso si no es necesario.

---

3) Las funciones de alcance y los filtros toman tiempo

AngularJS tiene un bucle de digestión. Todas sus funciones están en una vista y los filtros se ejecutan cada vez que se ejecuta el ciclo de resumen. El bucle de resumen se ejecutará cada vez que se actualice el modelo y puede ralentizar la aplicación (el filtro se puede golpear varias veces antes de que se cargue la página).

Evita esto:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Mejor enfoque

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Donde el controlador puede ser:

```
app.controller('bigCalulations', function(valueService) {
    // bad, because this is called in every digest loop
    this.calculateMe = function() {
        var t = 0;
        for(i = 0; i < 1000; i++) {
            t += i;
        }
        return t;
```

```
    }
    // good, because this is executed just once and logic is separated in service to    keep
the controller light
    this.preCalulatedValue = valueService.valueCalculation(); // returns 499500
});
```

---

4 observadores

Los observadores bajan tremendamente el rendimiento. Con más observadores, el bucle de resumen tardará más tiempo y la interfaz de usuario se ralentizará. Si el observador detecta un cambio, iniciará el bucle de resumen y volverá a representar la vista.

Hay tres formas de hacer una observación manual de cambios variables en Angular.

`$watch()` - observa cambios de valor

`$watchCollection()` : vigila los cambios en la colección (mira más de `$watch` regular)

`$watch(..., true)` : **evite esto** tanto como sea posible, realizará una "observación profunda" y disminuirá el rendimiento (observa más que `watchCollection`)

Tenga en cuenta que si está vinculando variables en la vista, está creando nuevos relojes: use `{{::variable}}` para evitar la creación de un reloj, especialmente en bucles.

Como resultado, debe realizar un seguimiento de cuántos observadores está utilizando. Puede contar a los observadores con este script (crédito para @Words Like Jared Number of watchers)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
```

```
})();
```

---

5) ng-if / ng-show

Estas funciones son muy similares en comportamiento. `ng-if` elimina elementos del DOM, mientras que `ng-show` solo oculta los elementos pero conserva todos los controladores. Si tiene partes del código que no desea mostrar, use `ng-if` .

Depende del tipo de uso, pero a menudo uno es más adecuado que el otro.

- Si el elemento no es necesario, use `ng-if`
- Para activar / desactivar rápidamente, use `ng-show/ng-hide`

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special<!-- some complicated DOM --></p>
  <p ng-show="user.hasSubscribed">I am awesome<!-- switch this setting on and off --></p>
</div>
```

Si tiene dudas, use `ng-if` y pruebe!

---

6) Deshabilitar la depuración

De forma predeterminada, las directivas y los ámbitos de enlace dejan clases y marcas adicionales en el código para ayudar con varias herramientas de depuración. Deshabilitar esta opción significa que ya no procesa estos diversos elementos durante el ciclo de resumen.

```
angular.module('exampleApp', []).config(['$compileProvider', function ($compileProvider) {
    $compileProvider.debugInfoEnabled(false);
}]);
```

---

7) Usa la inyección de dependencia para exponer tus recursos.

La inyección de dependencia es un patrón de diseño de software en el que un objeto recibe sus dependencias, en lugar del objeto que las crea. Se trata de eliminar las dependencias codificadas y hacer posible cambiarlas cuando sea necesario.

Podría preguntarse sobre el costo de rendimiento asociado con el análisis de cadenas de todas las funciones inyectables. Angular se encarga de esto almacenando en caché la propiedad $ inject después de la primera vez. Así que esto no ocurre cada vez que una función necesita ser invocada.

CONSEJO PROFESIONAL: Si está buscando el enfoque con el mejor rendimiento, vaya con el

enfoque de anotación de propiedad de $ inyectar. Este enfoque evita por completo el análisis de la definición de la función porque esta lógica se ajusta dentro de la siguiente comprobación en la función de anotación: if (! ($ Inject = fn. $ Inject)). Si $ inyectar ya está disponible, ¡no se requiere análisis!

```
var app = angular.module('DemoApp', []);

var DemoController = function (s, h) {
    h.get('https://api.github.com/users/angular/repos').success(function (repos) {
        s.repos = repos;
    });
}
// $inject property annotation
DemoController['$inject'] = ['$scope', '$http'];

app.controller('DemoController', DemoController);
```

PRO TIP 2: puede agregar una directiva `ng-strict-di` en el mismo elemento que `ng-app` para optar al modo DI estricto, que generará un error cada vez que un servicio intente usar anotaciones implícitas. Ejemplo:

```
<html ng-app="DemoApp" ng-strict-di>
```

O si usas bootstrapping manual:

```
angular.bootstrap(document, ['DemoApp'], {
    strictDi: true
});
```

Atar una vez

Angular tiene reputación de tener un enlace de datos bidireccional impresionante. De forma predeterminada, Angular sincroniza continuamente los valores enlazados entre los componentes del modelo y de la vista en cualquier momento en que los datos cambian en el componente del modelo o en la vista.

Esto conlleva el costo de ser un poco lento si se usa demasiado. Esto tendrá un mayor impacto de rendimiento:

Mal rendimiento: `{{my.data}}`

Agregue dos dos puntos `::` antes del nombre de la variable para usar el enlace de una sola vez. En este caso, el valor solo se actualiza una vez que se define my.data. Usted está apuntando explícitamente a no observar los cambios en los datos. Angular no realizará ninguna verificación de valores, lo que dará como resultado que se evalúen menos expresiones en cada ciclo de resumen.

Buenos ejemplos de rendimiento utilizando un enlace de una sola vez

```
{{::my.data}}
```

```
<span ng-bind="::my.data"></span>
<span ng-if="::my.data"></span>
<span ng-repeat="item in ::my.data">{{item}}</span>
<span ng-class="::{ 'my-class': my.data }"></div>
```

Nota: Sin embargo, esto elimina el enlace de datos bidireccional para `my.data` , por lo que siempre que este campo cambie en su aplicación, el mismo no se reflejará en la vista automáticamente. Por **lo** tanto, **úselo solo para valores que no cambiarán a lo largo de la vida útil de su aplicación** .

Funciones de alcance y filtros.

AngularJS tiene un bucle de resumen y todas sus funciones en una vista y filtros se ejecutan cada vez que se ejecuta el ciclo de resumen. El bucle de resumen se ejecutará cada vez que se actualice el modelo y puede ralentizar la aplicación (el filtro se puede golpear varias veces, antes de que se cargue la página).

Debes evitar esto:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Mejor enfoque

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Donde la muestra del controlador es:

```
.controller("bigCalulations", function(valueService) {
  // bad, because this is called in every digest loop
  this.calculateMe = function() {
    var t = 0;
    for(i = 0; i < 1000; i++) {
      t = t + i;
    }
    return t;
  }
  //good, because it is executed just once and logic is separated in service to    keep the
controller light
  this.preCalulatedValue = valueService.caluclateSumm(); // returns 499500
});
```

Vigilantes

Los observadores necesitan ver algún valor y detectar que este valor ha cambiado.

Después de llamar a `$watch()` o `$watchCollection` nuevo observador se agrega a la colección de

observadores internos en el alcance actual.

Entonces, ¿qué es el observador?

Watcher es una función simple, que se llama en cada ciclo de resumen, y devuelve algo de valor. Angular verifica el valor devuelto, si no es el mismo que en la llamada anterior: se ejecutará una devolución de llamada que se pasó en el segundo parámetro para funcionar `$watch()` o `$watchCollection` .

```
(function() {
  angular.module("app", []).controller("ctrl", function($scope) {
    $scope.value = 10;
    $scope.$watch(
      function() { return $scope.value; },
      function() { console.log("value changed"); }
    );
  }
})();
```

Los observadores son asesinos de rendimiento. Cuantos más observadores tengas, más tiempo tardarán en hacer un bucle de resumen, más lenta será la IU. Si un observador detecta cambios, iniciará el ciclo de resumen (recálculo en todas las pantallas)

Hay tres formas de hacer una observación manual para cambios variables en Angular.

`$watch()` - solo observa los cambios de valor

`$watchCollection()` : vigila los cambios en la colección (mira más de $ regular)

`$watch(..., true)` : **evite esto** tanto como sea posible, realizará una "observación profunda" y eliminará el rendimiento (observa más que watchCollection)

Tenga en cuenta que si está vinculando variables en la vista, está creando nuevos observadores; use `{{::variable}}` no para crear observadores, especialmente en bucles

Como resultado, debe realizar un seguimiento de cuántos observadores está utilizando. Puede contar a los observadores con este script (crédito para @Words Like Jared - ¿Cómo contar el número total de relojes en una página?

```
(function() {
  var root = angular.element(document.getElementsByTagName("body")),
      watchers = [];

  var f = function(element) {

    angular.forEach(["$scope", "$isolateScope"], function(scopeProperty) {
      if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
        angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
          watchers.push(watcher);
        });
      }
    });
```

```
    angular.forEach(element.children(), function(childElement) {
      f(angular.element(childElement));
    });

  };

  f(root);

  // Remove duplicate watchers
  var watchersWithoutDuplicates = [];
  angular.forEach(watchers, function(item) {
    if(watchersWithoutDuplicates.indexOf(item) < 0) {
      watchersWithoutDuplicates.push(item);
    }
  });

  console.log(watchersWithoutDuplicates.length);

})();
```

Si no desea crear su propio script, hay una utilidad de código abierto llamada ng-stats que utiliza un gráfico en tiempo real integrado en la página para darle una idea del número de relojes que Angular está administrando, así como el Frecuencia y duración de los ciclos de digestión a lo largo del tiempo. La utilidad expone una función global llamada `showAngularStats` que puede llamar para configurar cómo desea que funcione el gráfico.

```
showAngularStats({
  "position": "topleft",
  "digestTimeThreshold": 16,
  "autoload": true,
  "logDigest": true,
  "logWatches": true
});
```

El código de ejemplo anterior muestra el siguiente cuadro en la página automáticamente (demostración interactiva).

ng-if vs ng-show

Estas funciones son muy similares en comportamiento. La diferencia es que `ng-if` elimina elementos del DOM. Si hay partes grandes del código que no se mostrarán, entonces `ng-if` es el camino a seguir. `ng-show` solo ocultará los elementos pero conservará todos los manejadores.

ng-si

La directiva ngIf elimina o vuelve a crear una parte del árbol DOM en función de una expresión. Si

la expresión asignada a ngIf se evalúa como un valor falso, entonces el elemento se elimina del DOM, de lo contrario, se reinserta en el DOM un clon del elemento.

ng-show

La directiva ngShow muestra u oculta el elemento HTML dado en función de la expresión proporcionada al atributo ngShow. El elemento se muestra u oculta al eliminar o agregar la clase de CSS ng-hide al elemento.

Ejemplo

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special
    <!-- some complicated DOM -->
  </p>
  <p ng-show="user.hasSubscribed">I am aweosme
    <!-- switch this setting on and off -->
  </p>
</div>
```

Conclusión

Depende del tipo de uso, pero a menudo uno es más adecuado que el otro (por ejemplo, si el 95% del tiempo no se necesita el elemento, use `ng-if` ; si necesita cambiar la visibilidad del elemento DOM, use `ng-show`).

En caso de duda, utilice `ng-if` y pruebe!

Nota : `ng-if` crea un nuevo alcance aislado, mientras que `ng-show` y `ng-hide` no lo hacen. Use `$parent.property` si no se puede acceder directamente a la propiedad del ámbito principal.

Rebota tu modelo

```
<div ng-controller="ExampleController">
    <form name="userForm">
        Name:
        <input type="text" name="userName"
           ng-model="user.name"
           ng-model-options="{ debounce: 1000 }" />
            <button ng-click="userForm.userName.$rollbackViewValue();
user.name=''">Clear</button><br />
    </form>
    <pre>user.name = </pre>
</div>
```

En el ejemplo anterior, estamos configurando un valor de rebote de 1000 milisegundos que es de 1 segundo. Esto es un retraso considerable, pero evitará que la entrada golpee repetidamente el

`ng-model` con muchos ciclos de `$digest` .

Al usar rebotar en sus campos de entrada y en cualquier otro lugar donde no se requiera una actualización instantánea, puede aumentar el rendimiento de sus aplicaciones Angular de manera sustancial. No solo puede retrasar el tiempo, sino que también puede retrasarse cuando se desencadena la acción. Si no desea actualizar su modelo ng con cada pulsación de tecla, también puede actualizar el desenfoque.

Siempre anular el registro de los oyentes registrados en otros ámbitos distintos del alcance actual

Siempre debe anular el registro de otros ámbitos que no sean su alcance actual como se muestra a continuación:

```
//always deregister these
$rootScope.$on(...);
$scope.$parent.$on(...);
```

No tiene que anular el registro de las listas en el alcance actual, ya que angular se haría cargo de ello:

```
//no need to deregister this
$scope.$on(...);
```

`$rootScope.$on` escuchas permanecerán en la memoria si navega a otro controlador. Esto creará una pérdida de memoria si el controlador queda fuera del alcance.

No hacer

```
angular.module('app').controller('badExampleController', badExample);
badExample.$inject = ['$scope', '$rootScope'];

function badExample($scope, $rootScope) {
    $rootScope.$on('post:created', function postCreated(event, data) {});
}
```

Hacer

```
angular.module('app').controller('goodExampleController', goodExample);
goodExample.$inject = ['$scope', '$rootScope'];

function goodExample($scope, $rootScope) {
    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });
}
```

Lea Perfilado y Rendimiento en línea: https://riptutorial.com/es/angularjs/topic/1921/perfilado-y-rendimiento

Capítulo 39: Prepararse para la producción - Grunt

Examples

Ver precarga

Cuando se solicita la primera vista, normalmente Angular realiza una solicitud `XHR` para obtener esa vista. Para proyectos de tamaño medio, el recuento de vistas puede ser significativo y puede ralentizar la capacidad de respuesta de la aplicación.

La **buena práctica es cargar previamente** todas las vistas a la vez para proyectos pequeños y medianos. Para proyectos más grandes, también es bueno agregarlos en algunos bultos significativos, pero algunos otros métodos pueden ser útiles para dividir la carga. Para automatizar esta tarea es útil utilizar tareas Grunt o Gulp.

Para precargar las vistas, podemos usar el objeto `$templateCache` . Ese es un objeto, donde angular almacena cada vista recibida del servidor.

Es posible usar el módulo `html2js` , que convertirá todas nuestras vistas en un módulo: archivo js. Entonces necesitaremos inyectar ese módulo en nuestra aplicación y eso es todo.

Para crear un archivo concatenado de todas las vistas podemos usar esta tarea.

```
module.exports = function (grunt) {
 //set up the location of your views here
 var viewLocation = ['app/views/**.html'];

 grunt.initConfig({
        pkg: require('./package.json'),
            //section that sets up the settings for concatenation of the html files into one
file
            html2js: {
                        options: {
                            base: '',
                            module: 'app.templates', //new module name
                            singleModule: true,
                            useStrict: true,
                            htmlmin: {
                                collapseBooleanAttributes: true,
                                collapseWhitespace: true
                            }
                        },
                        main: {
                            src: viewLocation,
                            dest: 'build/app.templates.js'
                        }
                    },
            //this section is watching for changes in view files, and if there was a change,
it will regenerate the production file. This task can be handy during development.
            watch: {
```

```
                views:{
                    files: viewLocation,
                    tasks: ['buildHTML']
                },
            }
        });

        //to automatically generate one view file
        grunt.loadNpmTasks('grunt-html2js');

        //to watch for changes and if the file has been changed, regenerate the file
        grunt.loadNpmTasks('grunt-contrib-watch');

        //just a task with friendly name to reference in watch
        grunt.registerTask('buildHTML', ['html2js']);
};
```

Para usar esta forma de concatinación, necesita hacer 2 cambios: En su archivo `index.html` necesita hacer referencia al archivo de vista concatenada

```
<script src="build/app.templates.js"></script>
```

En el archivo, donde está declarando su aplicación, necesita inyectar la dependencia

```
angular.module('app', ['app.templates'])
```

Si está utilizando enrutadores populares como el enrutador `ui-router` , no hay cambios en la forma en que hace referencia a las plantillas.

```
    .state('home', {
        url: '/home',
        views: {
            "@": {
                controller: 'homeController',
                //this will be picked up from $templateCache
                templateUrl: 'app/views/home.html'
            },
        }

    })
```

Optimización de scripts

Es una **buena práctica combinar los archivos JS** y minimizarlos. Para proyectos más grandes, podría haber cientos de archivos JS y agrega una latencia innecesaria para cargar cada archivo por separado del servidor.

Para la minificación angular se requiere tener todas las funciones anotadas. Eso es necesario para la inyección de dependencia angular adecuada minificación. (Durante la minificación, los nombres de las funciones y las variables serán renombrados y se interrumpirá la inyección de dependencia si no se toman acciones adicionales).

Durante minificaiton `$scope` y `myService` variables serán reemplazadas por algunos otros valores.

La inyección de dependencia angular funciona según el nombre, como resultado, estos nombres no deben cambiar

```
.controller('myController', function($scope, myService){
})
```

Angular entenderá la notación de matriz, porque la minificación no reemplazará los literales de cadena.

```
.controller('myController', ['$scope','myService', function($scope, myService){
}])
```

- En primer lugar vamos a concatinar todos los archivos de extremo a extremo.
- En segundo lugar usaremos el módulo `ng-annotate` , que preparará el código para la minificación.
- Finalmente `uglify` módulo `uglify` .

module.exports = function (grunt) {// configura la ubicación de tus scripts aquí para reutilizarlos en el código var scriptLocation = ['app / scripts / *. js'];

```
 grunt.initConfig({
        pkg: require('./package.json'),
            //add necessary annotations for safe minification
         ngAnnotate: {
            angular: {
                src: ['staging/concatenated.js'],
                dest: 'staging/anotated.js'
            }
        },
        //combines all the files into one file
        concat: {
                js: {
                    src: scriptLocation,
                    dest: 'staging/concatenated.js'
                }
            },
        //final uglifying
        uglify: {
            options: {
                report: 'min',
                mangle: false,
                sourceMap:true
            },
            my_target: {
                files: {
                    'build/app.min.js': ['staging/anotated.js']
                }
            }
        },

        //this section is watching for changes in JS files, and if there was a change, it will
regenerate the production file. You can choose not to do it, but I like to keep concatenated
version up to date
        watch: {
            scripts: {
                files: scriptLocation,
```

```
                tasks: ['buildJS']
            }
        }

});

    //module to make files less readable
    grunt.loadNpmTasks('grunt-contrib-uglify');

    //mdule to concatenate files together
    grunt.loadNpmTasks('grunt-contrib-concat');

    //module to make angularJS files ready for minification
    grunt.loadNpmTasks('grunt-ng-annotate');

    //to watch for changes and if the file has been changed, regenerate the file
    grunt.loadNpmTasks('grunt-contrib-watch');

    //task that sequentially executes all steps to prepare JS file for production
    //concatinate all JS files
    //annotate JS file (prepare for minification
    //uglify file
     grunt.registerTask('buildJS', ['concat:js', 'ngAnnotate', 'uglify']);
};
```

Lea Prepararse para la producción - Grunt en línea:
https://riptutorial.com/es/angularjs/topic/4434/prepararse-para-la-produccion---grunt

Capítulo 40: Promesas angulares con servicio $ q

Examples

Usando $ q.all para manejar múltiples promesas

Puede usar la función `$q.all` para llamar a un método `.then` después de que una serie de promesas se haya resuelto con éxito y recuperar los datos con los que se resolvieron.

Ejemplo:

JS:

```
$scope.data = []

$q.all([
    $http.get("data.json"),
    $http.get("more-data.json"),
    ]).then(function(responses) {
      $scope.data = responses.map((resp) => resp.data);
    });
```

El código anterior se ejecuta `$http.get` 2 veces para los datos en archivos JSON locales, cuando ambos `get` método completo que resuelven sus promesas asociadas, cuando se resuelvan todas las promesas de la matriz, la `.then` método comienza con las dos promesas de datos dentro de las `responses` argumento de la matriz.

Luego, los datos se asignan para que se puedan mostrar en la plantilla, luego podemos mostrar

HTML:

```
<ul>
   <li ng-repeat="d in data">
      <ul>
         <li ng-repeat="item in d">{{item.name}}: {{item.occupation}}</li>
      </ul>
   </li>
</ul>
```

JSON:

```
[{
  "name": "alice",
  "occupation": "manager"
}, {
  "name": "bob",
  "occupation": "developer"
}]
```

Usando el constructor $ q para crear promesas

La función constructora `$q` se utiliza para crear promesas a partir de API asíncronas que utilizan devoluciones de llamada para devolver resultados.

> $ q (función (resolver, rechazar) {...})

La función constructora recibe una función que se invoca con dos argumentos, `resolve` y `reject` que son funciones que se utilizan para resolver o rechazar la promesa.

Ejemplo 1:

```
function $timeout(fn, delay) {
    return = $q(function(resolve, reject) {
         setTimeout(function() {
              try {
                 let r = fn();
                 resolve(r);
              }
              catch (e) {
                 reject(e);
              }
         }, delay);
     };
}
```

El ejemplo anterior crea una promesa de la API WindowTimers.setTimeout . El marco AngularJS proporciona una versión más elaborada de esta función. Para uso, vea la referencia de la API del servicio de tiempo de espera de AngularJS $.

Ejemplo 2:

```
$scope.divide = function(a, b) {
    return $q(function(resolve, reject) {
      if (b===0) {
        return reject("Cannot devide by 0")
      } else {
        return resolve(a/b);
      }
    });
}
```

El código anterior que muestra una función de división promisificada, devolverá una promesa con el resultado o rechazará con una razón si el cálculo es imposible.

Luego puedes llamar y usar `.then`

```
$scope.divide(7, 2).then(function(result) {
    // will return 3.5
}, function(err) {
    // will not run
})
```

```
$scope.divide(2, 0).then(function(result) {
    // will not run as the calculation will fail on a divide by 0
}, function(err) {
    // will return the error string.
})
```

Operaciones diferidas usando $ q.defer

Podemos usar `$q` para aplazar las operaciones hacia el futuro mientras tenemos un objeto de promesa pendiente en el presente, mediante el uso de `$q.defer` Nosotros creamos una promesa que se resolverá o rechazará en el futuro.

Este método no es equivalente al uso del constructor `$q` , ya que usamos `$q.defer` a prometer una rutina existente que puede o no devolver (o alguna vez ha devuelto) una promesa.

Ejemplo:

```
var runAnimation = function(animation, duration) {
    var deferred = $q.defer();
    try {
        ...
        // run some animation for a given duration
        deferred.resolve("done");
    } catch (err) {
        // in case of error we would want to run the error hander of .then
        deferred.reject(err);
    }
    return deferred.promise;
}

// and then
runAnimation.then(function(status) {}, function(error) {})
```

1. Asegúrese de devolver siempre un objeto `deferred.promise` o `.then` o de arriesgarse a un error al invocar `.then`
2. Asegúrese de que siempre resuelve o rechaza su objeto diferido o, de lo `.then` posible que no se ejecute y corre el riesgo de una pérdida de memoria.

Usando promesas angulares con servicio $ q

`$q` es un servicio incorporado que ayuda a ejecutar funciones asíncronas y a usar sus valores de retorno (o excepción) cuando finalizan el procesamiento.

`$q` se integra con el mecanismo de observación del modelo `$rootScope.Scope` , lo que significa una propagación más rápida de la resolución o el rechazo en sus modelos y evita repinturas innecesarias en el navegador, lo que daría como resultado una interfaz de usuario parpadeante.

En nuestro ejemplo, llamamos a nuestra fábrica `getMyData` , que devuelve un objeto de promesa. Si el objeto se `resolved` , devuelve un número aleatorio. Si se `rejected` , devuelve un rechazo con un mensaje de error después de 2 segundos.

En fabrica angular

```
function getMyData($timeout, $q) {
  return function() {
    // simulated async function
    var promise = $timeout(function() {
      if(Math.round(Math.random())) {
        return 'data received!'
      } else {
        return $q.reject('oh no an error! try again')
      }
    }, 2000);
    return promise;
  }
}
```

Usando promesas de guardia

```
angular.module('app', [])
.factory('getMyData', getMyData)
.run(function(getData) {
  var promise = getData()
    .then(function(string) {
      console.log(string)
    }, function(error) {
      console.error(error)
    })
    .finally(function() {
      console.log('Finished at:', new Date())
    })
})
```

Para usar promesas, inyecte `$q` como dependencia. Aquí inyectamos `$q` en la fábrica `getMyData` .

```
var defer = $q.defer();
```

Una nueva instancia de aplazado se construye llamando a `$q.defer()`

Un objeto diferido es simplemente un objeto que expone una promesa, así como los métodos asociados para resolverla. Se construye utilizando la función `$q.deferred()` y expone tres métodos principales: `resolve()` , `reject()` y `notify()` .

- `resolve(value)` : resuelve la promesa derivada con el valor.
- `reject(reason)` - rechaza la promesa derivada con la razón.
- `notify(value)` : proporciona actualizaciones sobre el estado de la ejecución de la promesa. Esto puede ser llamado varias veces antes de que la promesa sea resuelta o rechazada.

Propiedades

Se accede al objeto de promesa asociado a través de la propiedad de promesa. `promise` - {Promesa} - promete el objeto asociado con este aplazado.

Se crea una nueva instancia de promesa cuando se crea una instancia diferida y se puede recuperar llamando a `deferred.promise` .

El propósito del objeto de `promise` es permitir que las partes interesadas tengan acceso al resultado de la tarea diferida cuando se complete.

Métodos de promesa

- `then(successCallback, [errorCallback], [notifyCallback])` - Independientemente de cuándo se cumplió o se rechazará la promesa, luego llama a una de las devoluciones de llamada de éxito o error de forma asincrónica tan pronto como el resultado esté disponible. Las devoluciones de llamada se llaman con un solo argumento: el resultado o la razón de rechazo. Además, la devolución de llamada de notificación se puede llamar cero o más veces para proporcionar una indicación de progreso, antes de que se resuelva o se rechace la promesa.
- `catch(errorCallback)` - abreviatura de promise.then (null, errorCallback)
- `finally(callback, notifyCallback)` : le permite observar el cumplimiento o rechazo de una promesa, pero hacerlo sin modificar el valor final.

Una de las características más poderosas de las promesas es la capacidad de encadenarlas. Esto permite que los datos fluyan a través de la cadena y sean manipulados y mutados en cada paso. Esto se demuestra con el siguiente ejemplo:

Ejemplo 1:

```
// Creates a promise that when resolved, returns 4.
function getNumbers() {

  var promise = $timeout(function() {
    return 4;
  }, 1000);

  return promise;
}

// Resolve getNumbers() and chain subsequent then() calls to decrement
// initial number from 4 to 0 and then output a string.
getNumbers()
  .then(function(num) {
      // 4
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 3
      console.log(num);
      return --num;
  })
   .then(function (num) {
      // 2
      console.log(num);
      return --num;
  })
```

```
  .then(function (num) {
      // 1
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 0
      console.log(num);
      return 'And we are done!';
  })
  .then(function (text) {
      // "And we are done!"
      console.log(text);
  });
```

Envuelva el valor simple en una promesa usando $ q.when ()

Si todo lo que necesita es envolver el valor en una promesa, no necesita usar la sintaxis larga como aquí:

```
//OVERLY VERBOSE
var defer;
defer = $q.defer();
defer.resolve(['one', 'two']);
return defer.promise;
```

En este caso puedes escribir:

```
//BETTER
return $q.when(['one', 'two']);
```

> ## $ q.when y su alias $ q.resolve
>
> Envuelve un objeto que podría ser un valor o una promesa (tercera parte) que luego se puede convertir en una promesa de $ q. Esto es útil cuando se trata de un objeto que puede o no ser una promesa, o si la promesa proviene de una fuente en la que no se puede confiar.
>
> - AngularJS $ q Referencia de la API del servicio - $ q.when

---

Con el lanzamiento de AngularJS v1.4.1

También puede utilizar una `resolve` alias consistente con ES6

```
//ABSOLUTELY THE SAME AS when
return $q.resolve(['one', 'two'])
```

Evita el $ q diferido anti-patrón

Evita este Anti-Patrón

```
var myDeferred = $q.defer();

$http(config).then(function(res) {
   myDeferred.resolve(res);
}, function(error) {
   myDeferred.reject(error);
});

return myDeferred.promise;
```

No hay necesidad de fabricar una promesa con `$q.defer` ya que el servicio $ http ya devuelve una promesa.

```
//INSTEAD
return $http(config);
```

Simplemente devuelva la promesa creada por el servicio $ http.

Lea Promesas angulares con servicio $ q en línea:
https://riptutorial.com/es/angularjs/topic/4379/promesas-angulares-con-servicio---q

Capítulo 41: Proveedores

Sintaxis

- constante (nombre, valor);
- valor (nombre, valor);
- fábrica (nombre, $ getFn);
- servicio (nombre, constructor);
- proveedor (nombre, proveedor);

Observaciones

Los proveedores son objetos singleton que se pueden inyectar, por ejemplo, en otros servicios, controladores y directivas. Todos los proveedores están registrados con diferentes "recetas", donde el `Provider` es el más flexible. Todas las recetas posibles son:

- Constante
- Valor
- Fábrica
- Servicio
- Proveedor

Los servicios, las fábricas y los proveedores están todos iniciados, el componente se inicializa solo si la aplicación depende de él.

Los decoradores están estrechamente relacionados con los proveedores. Los decoradores se utilizan para interceptar el servicio o la creación de fábrica para cambiar su comportamiento o anularlo (partes de).

Examples

Constante

`Constant` está disponible tanto en la configuración como en las fases de ejecución.

```
angular.module('app',[])
  .constant('endpoint', 'http://some.rest.endpoint') // define
  .config(function(endpoint) {
    // do something with endpoint
    // available in both config- and run phases
  })
  .controller('MainCtrl', function(endpoint) {       // inject
    var vm = this;
    vm.endpoint = endpoint;                          // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

> punto final = http: //some.rest.endpoint

Valor

`Value` está disponible tanto en la configuración como en las fases de ejecución.

```
angular.module('app',[])
  .value('endpoint', 'http://some.rest.endpoint') // define
  .run(function(endpoint) {
    // do something with endpoint
    // only available in run phase
  })
  .controller('MainCtrl', function(endpoint) {    // inject
    var vm = this;
    vm.endpoint = endpoint;                       // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

> punto final = http: //some.rest.endpoint

Fábrica

`Factory` está disponible en la fase de ejecución.

> La receta de Factory construye un nuevo servicio utilizando una función con cero o más argumentos (son dependencias de otros servicios). El valor de retorno de esta función es la instancia de servicio creada por esta receta.
>
> Factory puede crear un servicio de cualquier tipo, ya sea primitivo, objeto literal, función o incluso una instancia de un tipo personalizado.

```
angular.module('app',[])
  .factory('endpointFactory', function() {
    return {
      get: function() {
        return 'http://some.rest.endpoint';
      }
    };
  })
  .controller('MainCtrl', function(endpointFactory) {
    var vm = this;
    vm.endpoint = endpointFactory.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

> punto final = http: //some.rest.endpoint

Servicio

`Service` está disponible en la fase de ejecución.

> La receta de Servicio produce un servicio como las recetas de Valor o Fábrica, pero lo hace *invocando a un constructor con el nuevo operador* . El constructor puede tomar cero o más argumentos, que representan dependencias necesarias para la instancia de este tipo.

```
angular.module('app',[])
  .service('endpointService', function() {
    this.get = function() {
      return 'http://some.rest.endpoint';
    };
  })
  .controller('MainCtrl', function(endpointService) {
    var vm = this;
    vm.endpoint = endpointService.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

> punto final = http: //some.rest.endpoint

Proveedor

`Provider` está disponible tanto en la configuración como en las fases de ejecución.

> La receta del proveedor se define sintácticamente como un tipo personalizado que implementa un método `$get` .
>
> Debe usar la receta del proveedor solo cuando desee exponer una API para la configuración de toda la aplicación que debe realizarse antes de que se inicie la aplicación. Por lo general, esto es interesante solo para los servicios reutilizables cuyo comportamiento puede necesitar variar ligeramente entre las aplicaciones.

```
angular.module('app',[])
  .provider('endpointProvider', function() {
    var uri = 'n/a';

    this.set = function(value) {
```

```
      uri = value;
    };

    this.$get = function() {
      return {
        get: function() {
          return uri;
        }
      };
    };
  })
  .config(function(endpointProviderProvider) {
    endpointProviderProvider.set('http://some.rest.endpoint');
  })
  .controller('MainCtrl', function(endpointProvider) {
    var vm = this;
    vm.endpoint = endpointProvider.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

> punto final = http: //some.rest.endpoint

Sin la fase de `config` resultado sería

> punto final = n / a

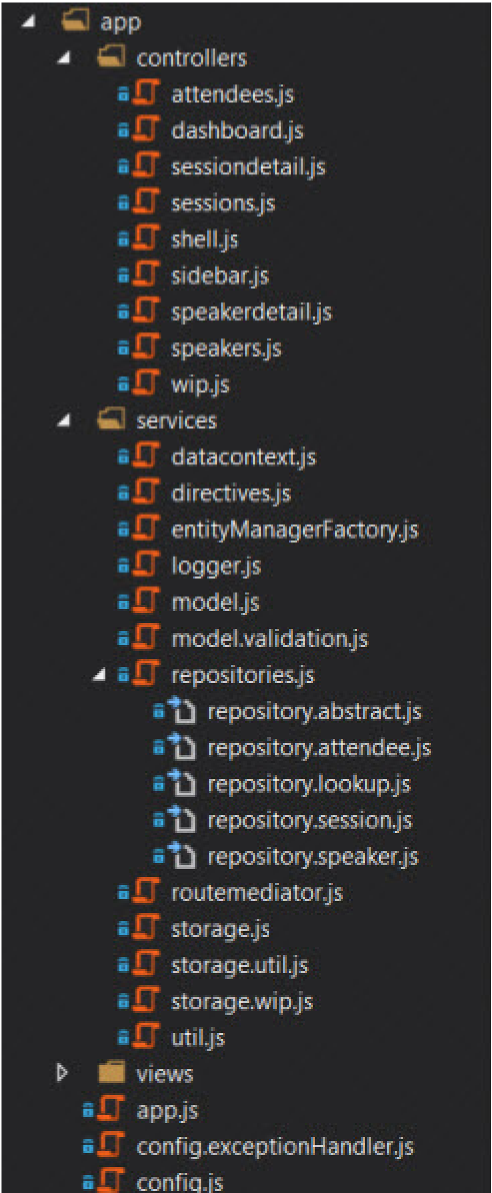
Lea Proveedores en línea: https://riptutorial.com/es/angularjs/topic/5169/proveedores

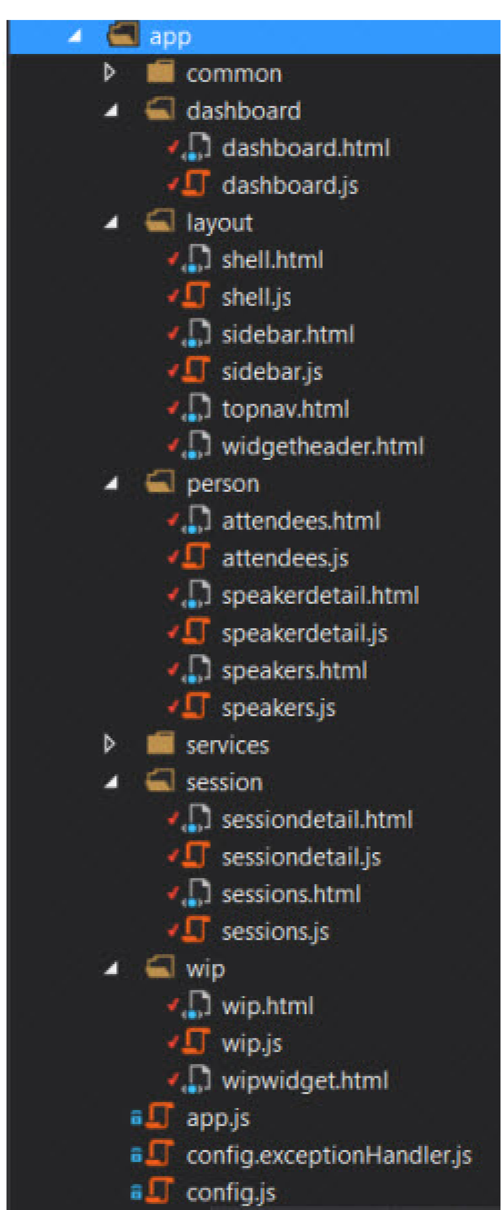
Capítulo 42: Proyecto Angular - Estructura de Directorio

Examples

Estructura de directorios

Una pregunta común entre los nuevos programadores angulares: "¿Cuál debería ser la estructura del proyecto?". Una buena estructura ayuda a un desarrollo de aplicaciones escalable. Cuando comenzamos un proyecto, tenemos dos opciones, **Ordenar por tipo** (izquierda) y **Ordenar por característica** (derecha). El segundo es mejor, especialmente en aplicaciones grandes, el proyecto se vuelve mucho más fácil de administrar.





Ordenar por tipo (izquierda)

La aplicación está organizada por el tipo de archivos.

- **Ventaja** : buena para aplicaciones pequeñas, para los programadores que solo comienzan a usar Angular y es fácil de convertir al segundo método.
- **Desventaja** : incluso para las aplicaciones pequeñas comienza a ser más difícil encontrar un archivo específico. Por ejemplo, una vista y su controlador están en dos carpetas separadas.

Ordenar por característica (derecha)

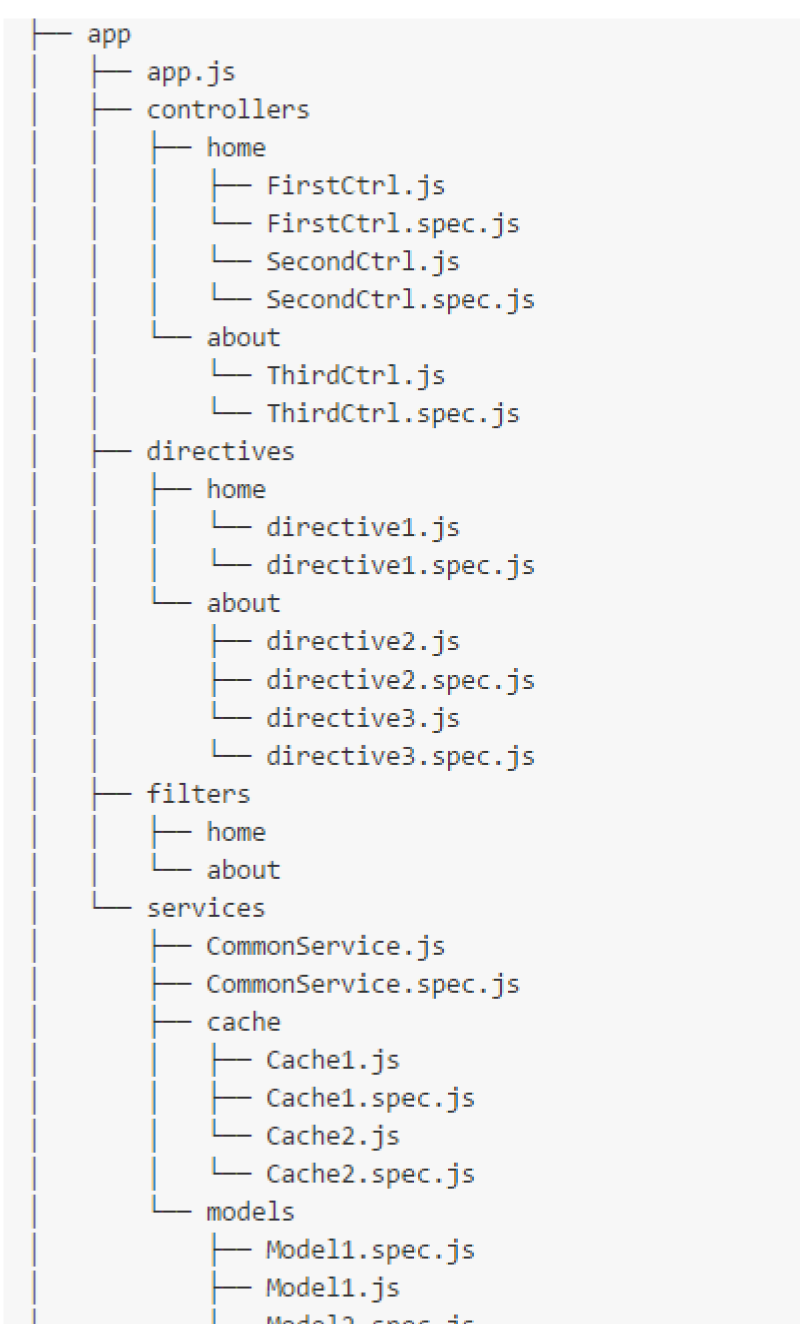
El método de organización sugerido donde los archivados se clasifican por tipo de características.

Todas las vistas de diseño y los controladores van en la carpeta de diseño, el contenido del administrador va en la carpeta de administración, etc.

- **Ventaja** : cuando se busca una sección del código que determina una determinada característica, todo está ubicado en una carpeta.
- **Desventaja** : los servicios son un poco diferentes ya que “atienden” muchas características.

Puedes leer más sobre esto en Estructura angular: Refactorización para el crecimiento

La estructura de archivos sugerida combina ambos métodos antes mencionados:



```
├── app
│   ├── app.js
│   ├── controllers
│   │   ├── home
│   │   │   ├── FirstCtrl.js
│   │   │   └── FirstCtrl.spec.js
│   │   │   └── SecondCtrl.js
│   │   │   └── SecondCtrl.spec.js
│   │   └── about
│   │       └── ThirdCtrl.js
│   │       └── ThirdCtrl.spec.js
│   ├── directives
│   │   ├── home
│   │   │   └── directive1.js
│   │   │   └── directive1.spec.js
│   │   └── about
│   │       ├── directive2.js
│   │       ├── directive2.spec.js
│   │       └── directive3.js
│   │       └── directive3.spec.js
│   ├── filters
│   │   ├── home
│   │   └── about
│   └── services
│       ├── CommonService.js
│       ├── CommonService.spec.js
│       ├── cache
│       │   ├── Cache1.js
│       │   ├── Cache1.spec.js
│       │   └── Cache2.js
│       │   └── Cache2.spec.js
│       └── models
│           ├── Model1.spec.js
│           ├── Model1.js
```

Crédito a: Guía de estilo angular

Lea Proyecto Angular - Estructura de Directorio en línea: https://riptutorial.com/es/angularjs/topic/6148/proyecto-angular---estructura-de-directorio

Capítulo 43: Pruebas unitarias

Observaciones

Este tema proporciona ejemplos para la prueba unitaria de las diversas construcciones en AngularJS. Las pruebas unitarias a menudo se escriben utilizando Jasmine , un popular marco de pruebas basado en el comportamiento. Cuando la unidad realice pruebas angulares, deberá incluir ngMock como una dependencia al ejecutar las pruebas de la unidad.

Examples

Unidad de prueba de un filtro.

Código del filtro:

```
angular.module('myModule', []).filter('multiplier', function() {
  return function(number, multiplier) {
    if (!angular.isNumber(number)) {
      throw new Error(number + " is not a number!");
    }
    if (!multiplier) {
      multiplier = 2;
    }
    return number * multiplier;
  }
});
```

La prueba:

```
describe('multiplierFilter', function() {
  var filter;

  beforeEach(function() {
    module('myModule');
    inject(function(multiplierFilter) {
      filter = multiplierFilter;
    });
  });

  it('multiply by 2 by default', function() {
    expect(filter(2)).toBe(4);
    expect(filter(3)).toBe(6);
  });

  it('allow to specify custom multiplier', function() {
    expect(filter(2, 4)).toBe(8);
  });

  it('throws error on invalid input', function() {
    expect(function() {
      filter(null);
    }).toThrow();
```

```
  });
});
```

¡Correr!

Observación: en la llamada de `inject` en la prueba, su filtro debe especificarse por su nombre + *Filtro* . La causa de esto es que cada vez que registra un filtro para su módulo, Angular lo registra con un `Filter` adjunto a su nombre.

Prueba unitaria de un componente (1.5+)

Código del componente:

```
angular.module('myModule', []).component('myComponent', {
  bindings: {
    myValue: '<'
  },
  controller: function(MyService) {
    this.service = MyService;
    this.componentMethod = function() {
      return 2;
    };
  }
});
```

La prueba:

```
describe('myComponent', function() {
  var component;

  var MyServiceFake = jasmine.createSpyObj(['serviceMethod']);

  beforeEach(function() {
    module('myModule');
    inject(function($componentController) {
      // 1st - component name, 2nd - controller injections, 3rd - bindings
      component = $componentController('myComponent', {
        MyService: MyServiceFake
      }, {
        myValue: 3
      });
    });
  });

  /** Here you test the injector. Useless. */

  it('injects the binding', function() {
    expect(component.myValue).toBe(3);
  });

  it('has some cool behavior', function() {
    expect(component.componentMethod()).toBe(2);
  });
});
```

¡Correr!

Unidad de prueba de un controlador.

Código del controlador:

```
angular.module('myModule', [])
  .controller('myController', function($scope) {
    $scope.num = 2;
    $scope.doSomething = function() {
      $scope.num += 2;
    }
  });
```

La prueba:

```
describe('myController', function() {
  var $scope;
  beforeEach(function() {
    module('myModule');
    inject(function($controller, $rootScope) {
      $scope = $rootScope.$new();
      $controller('myController', {
        '$scope': $scope
      })
    });
  });
  it('should increment `num` by 2', function() {
    expect($scope.num).toEqual(2);
    $scope.doSomething();
    expect($scope.num).toEqual(4);
  });
});
```

¡Correr!

Unidad de prueba de un servicio.

Código de servicio

```
angular.module('myModule', [])
  .service('myService', function() {
    this.doSomething = function(someNumber) {
      return someNumber + 2;
    }
  });
```

La prueba

```
describe('myService', function() {
  var myService;
  beforeEach(function() {
    module('myModule');
    inject(function(_myService_) {
      myService = _myService_;
    });
  });
```

```
  it('should increment `num` by 2', function() {
    var result = myService.doSomething(4);
    expect(result).toEqual(6);
  });
});
```

¡Correr!

Unidad de prueba una directiva

Código directivo

```
angular.module('myModule', [])
  .directive('myDirective', function() {
    return {
      template: '<div>{{greeting}} {{name}}!</div>',
      scope: {
        name: '=',
        greeting: '@'
      }
    };
  });
```

La prueba

```
describe('myDirective', function() {
  var element, scope;
  beforeEach(function() {
    module('myModule');
    inject(function($compile, $rootScope) {
      scope = $rootScope.$new();
      element = angular.element("<my-directive name='name' greeting='Hello'></my-directive>");
      $compile(element)(scope);
      /* PLEASE NEVER USE scope.$digest(). scope.$apply use a protection to avoid to run a
digest loop when there is already one, so, use scope.$apply() instead. */
      scope.$apply();
    })
  });

  it('has the text attribute injected', function() {
    expect(element.html()).toContain('Hello');
  });

  it('should have proper message after scope change', function() {
    scope.name = 'John';
    scope.$apply();
    expect(element.html()).toContain("John");
    scope.name = 'Alice';
    expect(element.html()).toContain("John");
    scope.$apply();
    expect(element.html()).toContain("Alice");
  });
});
```

¡Correr!

Lea Pruebas unitarias en línea: https://riptutorial.com/es/angularjs/topic/1689/pruebas-unitarias

Capítulo 44: recorrido del bucle de digestión

Sintaxis

- $ scope. $ watch (watchExpression, callback, [deep compare])
- $ scope. $ digest ()
- $ scope. $ apply ([exp])

Examples

enlace de datos de dos vías

Angular tiene algo de magia bajo su capucha. permite vincular DOM a variables js reales.

Angular utiliza un bucle, denominado " *bucle de resumen* ", que se llama después de cualquier cambio de una variable: llamadas devoluciones de llamada que actualizan el DOM.

Por ejemplo, la directiva `ng-model` adjunta un `keyup` eventListener a esta entrada:

```
<input ng-model="variable" />
```

Cada vez que se dispara el evento `keyup` , se inicia el *bucle de digestión* .

En algún momento, el *bucle de resumen* itera sobre una devolución de llamada que actualiza el contenido de este intervalo:

```
<span>{{variable}}</span>
```

El ciclo de vida básico de este ejemplo, resume (muy esquemáticamente) cómo funciona el ángulo:

1. Escaneo angular html
    - `ng-model` directiva `ng-model` crea una escucha `keyup` en la entrada
    - `expression` dentro de span agrega una devolución de llamada al *ciclo de digestión*
2. El usuario interactúa con la entrada
    - `keyup` oyente `keyup` comienza el *ciclo de digerir*
    - *ciclo de digestión de las* calles la devolución de llamada
    - La devolución de llamada actualiza los contenidos de span.

$ digerir y $ ver

La implementación del enlace de datos de dos vías, para lograr el resultado del ejemplo anterior, se podría realizar con dos funciones principales:

- **$ digest** se llama después de una interacción del usuario (vinculación DOM => variable)

- **$ watch** establece una devolución de llamada para que se llame después de los cambios de la variable (variable de enlace => DOM)

nota: este es un ejemplo es una demostración, no el código angular real

---

```
<input id="input"/>
<span id="span"></span>
```

Las dos funciones que necesitamos:

```
var $watches = [];
function $digest(){
    $watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
            $w.prevVal = val;
        }
    })
}
function $watch(val, callback){
    $watches.push({val:val, callback:callback, prevVal: val() })
}
```

Ahora podríamos usar estas funciones para conectar una variable al DOM (angular viene con directivas integradas que lo harán por usted):

```
var realVar;
//this is usually done by ng-model directive
input1.addEventListener('keyup',function(e){
    realVar=e.target.value;
    $digest()
}, true);

//this is usually done with {{expressions}} or ng-bind directive
$watch(function(){return realVar},function(val){
    span1.innerHTML = val;
});
```

Por supuesto, las implementaciones reales son más complejas y admiten parámetros como **el elemento** a enlazar y la **variable** a usar.

Un ejemplo de ejecución se puede encontrar aquí: https://jsfiddle.net/azofxd4j/

el arbol $ scope

El ejemplo anterior es lo suficientemente bueno cuando necesitamos vincular un solo elemento html a una sola variable.

En realidad, necesitamos unir muchos elementos a muchas variables:

```
<span ng-repeat="number in [1,2,3,4,5]">{{number}}</span>
```

Esta `ng-repeat` une 5 elementos a 5 variables llamadas `number` , ¡con un valor diferente para cada una de ellas!

La forma en que angular logra este comportamiento es mediante el uso de un contexto separado para cada elemento que necesita variables separadas. Este contexto se llama un alcance.

Cada ámbito contiene propiedades, que son las variables vinculadas al DOM, y las funciones `$digest` y `$watch` se implementan como métodos del alcance.

El DOM es un árbol, y las variables deben usarse en diferentes niveles del árbol:

```
<div>
    <input ng-model="person.name" />
    <span ng-repeat="number in [1,2,3,4,5]">{{number}} {{person.name}}</span>
</div>
```

Pero como vimos, el contexto (o alcance) de las variables dentro de `ng-repeat` es diferente al contexto que se encuentra arriba. Para resolver esto - angular implementa ámbitos como un árbol.

Cada alcance tiene una variedad de hijos, y llamar a su método `$digest` ejecutará todo su método `$digest` niños.

De esta manera, después de cambiar la entrada, se llama `$digest` para el alcance del div, que luego ejecuta `$digest` para sus 5 hijos, que actualizará su contenido.

Una implementación simple para un alcance, podría verse así:

```
function $scope(){
    this.$children = [];
    this.$watches = [];
}

$scope.prototype.$digest = function(){
    this.$watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
          $w.prevVal = val;
        }
    });
    this.$children.forEach(function(c){
        c.$digest();
    });
}

$scope.prototype.$watch = function(val, callback){
    this.$watches.push({val:val, callback:callback, prevVal: val() })
}
```

nota: este es un ejemplo es una demostración, no el código angular real

Lea recorrido del bucle de digestión en línea:
https://riptutorial.com/es/angularjs/topic/3156/recorrido-del-bucle-de-digestion

Capítulo 45: Servicio Distinguido vs Fábrica

Examples

Factory VS Service una vez por todas

Por definición:

Los servicios son básicamente funciones de constructor. Utilizan la palabra clave 'this'.

Las fábricas son funciones simples por lo tanto devuelven un objeto.

Bajo el capó:

Fábricas llamadas internamente a la función del proveedor.

Servicios de llamadas internas a función de fábrica.

Debate:

Las fábricas pueden ejecutar código antes de que devolvamos nuestro objeto literal.

Pero al mismo tiempo, los Servicios también se pueden escribir para devolver un objeto literal y para ejecutar código antes de regresar. Aunque eso es contraproducente ya que los servicios están diseñados para actuar como una función constructora.

De hecho, las funciones de constructor en JavaScript pueden devolver lo que quieran.

¿Entonces cual es mejor?

La sintaxis de servicios del constructor es más cercana a la sintaxis de clase de ES6. Así que la migración será fácil.

Resumen

Entonces, en resumen, el proveedor, la fábrica y el servicio son todos proveedores.

Una fábrica es un caso especial de un proveedor cuando todo lo que necesita en su proveedor es una función $ get (). Te permite escribirlo con menos código.

Un servicio es un caso especial de una fábrica cuando desea devolver una instancia de un nuevo objeto, con el mismo beneficio de escribir menos código.

```
mod.provider("myProvider", fur
    this.$get = function() {
        return new function()
            this.getValue = fu
                return "My Val
            };
        };
    };
});
```

Lea Servicio Distinguido vs Fábrica en línea:
https://riptutorial.com/es/angularjs/topic/7099/servicio-distinguido-vs-fabrica

Capítulo 46: Servicios

Examples

Cómo crear un servicio

```
angular.module("app")
  .service("counterService", function(){

    var service = {
        number: 0
    };

    return service;
   });
```

Cómo utilizar un servicio

```
    angular.module("app")

        // Custom services are injected just like Angular's built-in services
        .controller("step1Controller", ['counterService', '$scope', function(counterService, 
$scope) {
            counterService.number++;
            // bind to object (by reference), not to value, for automatic sync
            $scope.counter = counterService;
        })
```

En la plantilla que usa este controlador, escribirías:

```
// editable
<input ng-model="counter.number" />
```

o

```
// read-only
<span ng-bind="counter.number"></span>
```

Por supuesto, en código real, interactuarías con el servicio utilizando métodos en el controlador, que a su vez delegan en el servicio. El ejemplo anterior simplemente incrementa el valor del contador cada vez que se usa el controlador en una plantilla.

Servicios en Angularjs son singletons:

Los servicios son objetos singleton que se instancian solo una vez por aplicación (por el $ inyector) y se cargan perezosamente (creados solo cuando es necesario).

> Un singleton es una clase que solo permite crear una instancia de sí mismo, y brinda

un acceso simple y fácil a dicha instancia. Como se indica aquí

Creando un servicio usando angular.factory

Primero defina el servicio (en este caso usa el patrón de fábrica):

```
.factory('dataService', function() {
    var dataObject = {};
    var service = {
        // define the getter method
        get data() {
            return dataObject;
        },
        // define the setter method
        set data(value) {
            dataObject = value || {};
        }
    };
    // return the "service" object to expose the getter/setter
    return service;
})
```

Ahora puedes usar el servicio para compartir datos entre controladores:

```
.controller('controllerOne', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // create an object to store
    var someObject = {
        name: 'SomeObject',
        value: 1
    };
    // store the object
    this.dataService.data = someObject;
})

.controller('controllerTwo', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // this will automatically update with any changes to the shared data object
    this.objectFromControllerOne = this.dataService.data;
})
```

$ sce - desinfecta y representa contenido y recursos en plantillas

> $ sce ("Escape Contextual Estricto") es un servicio angular incorporado que automáticamente desinfecta el contenido y las fuentes internas en las plantillas.

inyectar fuentes **externas** y **HTML** en **bruto** en la plantilla requiere el ajuste manual de `$sce` .

En este ejemplo crearemos un filtro de saneamiento simple $ sce: `.

Manifestación

```
.filter('sanitizer', ['$sce', [function($sce) {
```

```
     return function(content) {
          return $sce.trustAsResourceUrl(content);
      };
}]);
```

Uso en plantilla

```
<div ng-repeat="item in items">

    // Sanitize external sources
    <ifrmae ng-src="{{item.youtube_url | sanitizer}}">

    // Sanitaize and render HTML
    <div ng-bind-html="{{item.raw_html_content| sanitizer}}"></div>

</div>
```

Cómo crear un servicio con dependencias usando 'sintaxis de matriz'

```
angular.module("app")
  .service("counterService", ["fooService", "barService", function(anotherService,
barService){

    var service = {
        number: 0,
        foo: function () {
            return fooService.bazMethod(); // Use of 'fooService'
        },
        bar: function () {
            return barService.bazMethod(); // Use of 'barService'
        }
    };

    return service;
   }]);
```

Registro de un servicio

La forma más común y flexible de crear un servicio utiliza la fábrica de API angular.module:

```
angular.module('myApp.services', []).factory('githubService', function() {
   var serviceInstance = {};
   // Our first service
   return serviceInstance;
});
```

La función de fábrica de servicios puede ser una función o una matriz, al igual que la forma en que creamos los controladores:

```
// Creating the factory through using the
// bracket notation
angular.module('myApp.services', [])
.factory('githubService', [function($http) {
}]);
```

Para exponer un método en nuestro servicio, podemos colocarlo como un atributo en el objeto de servicio.

```
angular.module('myApp.services', [])
    .factory('githubService', function($http) {
        var githubUrl = 'https://api.github.com';
        var runUserRequest = function(username, path) {
        // Return the promise from the $http service
        // that calls the Github API using JSONP
        return $http({
            method: 'JSONP',
            url: githubUrl + '/users/' +
            username + '/' +
            path + '?callback=JSON_CALLBACK'
        });
    }
// Return the service object with a single function
// events
return {
    events: function(username) {
    return runUserRequest(username, 'events');
    }
};
```

Diferencia entre Servicio y Fábrica.

1) Servicios

Un servicio es una función de `constructor` que se invoca una vez en tiempo de ejecución con el `new` , al igual que lo que haríamos con el Javascript simple con la única diferencia de que `AngularJs` está llamando al `new` detrás de escena.

Hay una regla del pulgar para recordar en caso de servicios

1. Los servicios son constructores que se llaman con `new`

Veamos un ejemplo sencillo en el que registraríamos un servicio que usa el servicio `$http` para obtener los detalles de los estudiantes y usarlos en el controlador

```
function StudentDetailsService($http) {
  this.getStudentDetails = function getStudentDetails() {
    return $http.get('/details');
  };
}

angular.module('myapp').service('StudentDetailsService', StudentDetailsService);
```

Acabamos de inyectar este servicio en el controlador.

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
```

```
angular.module('app').controller('StudentController', StudentController);
```

¿Cuándo usar?

Use `.service()` donde quiera que quiera usar un constructor. Normalmente se usa para crear API públicas como `getStudentDetails()` . Pero si no desea usar un constructor y desea usar un patrón de API simple, entonces no hay mucha flexibilidad en `.service()` .

2) Fábrica

A pesar de que podemos lograr todas las cosas usando `.factory()` que haríamos, usando `.services()` , no hace que `.factory()` "igual que" `.service()` . Es mucho más potente y flexible que `.service()`

Un `.factory()` es un patrón de diseño que se utiliza para devolver un valor.

Hay dos reglas para recordar en caso de fábricas.

1. Fábricas devuelven valores
2. Fábricas (pueden) crear objetos (cualquier objeto)

Veamos algunos ejemplos de lo que podemos hacer usando `.factory()`

Devolviendo objetos literales

Veamos un ejemplo donde se usa la fábrica para devolver un objeto usando un patrón de módulo Revelación básico

```
function StudentDetailsService($http) {
  function getStudentDetails() {
    return $http.get('/details');
  }
  return {
    getStudentDetails: getStudentDetails
  };
}

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Uso dentro de un controlador

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Cierres de retorno

¿Qué es un cierre?

Los cierres son funciones que se refieren a variables que se usan localmente, PERO definidas en

un ámbito de cierre.

A continuación se muestra un ejemplo de un cierre.

```
function closureFunction(name) {
  function innerClosureFunction(age) { // innerClosureFunction() is the inner function, a
closure
    // Here you can manipulate 'age' AND 'name' variables both
  };
};
```

La parte *"maravillosa"* es que puede acceder al `name` que está en el ámbito principal.

Vamos a usar el ejemplo de cierre anterior en `.factory()`

```
function StudentDetailsService($http) {
  function closureFunction(name) {
  function innerClosureFunction(age) {
    // Here you can manipulate 'age' AND 'name' variables
    };
  };
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Uso dentro de un controlador

```
function StudentController(StudentDetailsService) {
  var myClosure = StudentDetailsService('Student Name'); // This now HAS the
innerClosureFunction()
  var callMyClosure = myClosure(24); // This calls the innerClosureFunction()
};

angular.module('app').controller('StudentController', StudentController);
```

Creación de constructores / instancias

`.service()` crea constructores con una llamada a `new` como se ve arriba. `.factory()` también puede crear constructores con una llamada a `new`

Veamos un ejemplo de cómo lograr esto.

```
function StudentDetailsService($http) {
  function Student() {
    this.age = function () {
        return 'This is my age';
    };
  }
  Student.prototype.address = function () {
        return 'This is my address';
  };
  return Student;
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Uso dentro de un controlador

```
function StudentController(StudentDetailsService) {
  var newStudent = new StudentDetailsService();

  //Now the instance has been created. Its properties can be accessed.

 newStudent.age();
 newStudent.address();

};

angular.module('app').controller('StudentController', StudentController);
```

Lea Servicios en línea: https://riptutorial.com/es/angularjs/topic/1486/servicios

Capítulo 47: SignalR con AngularJs

Introducción

En este artículo, nos centramos en "Cómo crear un proyecto simple utilizando AngularJs y SignalR", en esta capacitación que necesita saber sobre "cómo crear una aplicación con angularjs", "cómo crear / utilizar el servicio en angular" y conocimientos básicos sobre SignalR "para esto recomendamos https://www.codeproject.com/Tips/590660/Introduction-to-SignalR) .

Examples

SignalR y AngularJs [ChatProject]

paso 1: crear proyecto

```
- Application
    - app.js
    - Controllers
        - appController.js
    - Factories
        - SignalR-factory.js
- index.html
- Scripts
    - angular.js
    - jquery.js
    - jquery.signalR.min.js
- Hubs
```

> Uso de la versión SignalR: signalR-2.2.1

Paso 2: Startup.cs y ChatHub.cs

Vaya a su directorio *"/ Hubs"* y agregue 2 archivos *[Startup.cs, ChatHub.cs]*

Startup.cs

```
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(SignalR.Hubs.Startup))]

namespace SignalR.Hubs
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

ChatHub.cs

```
using Microsoft.AspNet.SignalR;

namespace SignalR.Hubs
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message, string time)
        {
            Clients.All.broadcastMessage(name, message, time);
        }
    }
}
```

paso 3: crear una aplicación angular

Vaya a su directorio *"/ Aplicación"* y agregue el archivo *[app.js]*

app.js

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

paso 4: crear SignalR Factory

Vaya a su directorio *"/ Application / Factories"* y agregue el *archivo [SignalR-factory.js]*

SignalR-factory.js

```
app.factory("signalR", function () {
    var factory = {};

    factory.url = function (url) {
        $.connection.hub.url = url;
    }

    factory.setHubName = function (hubName) {
        factory.hub = hubName;
    }

    factory.connectToHub = function () {
        return $.connection[factory.hub];
    }

    factory.client = function () {
        var hub = factory.connectToHub();
        return hub.client;
    }

    factory.server = function () {
        var hub = factory.connectToHub();
        return hub.server;
    }

    factory.start = function (fn) {
        return $.connection.hub.start().done(fn);
    }
```

```
    return factory;
});
```

paso 5: actualizar app.js

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

app.run(function(signalR) {
    signalR.url("http://localhost:21991/signalr");
});
```

> localhost: 21991 / signalr | **Este es tu SignalR Hubs Urls**

paso 6: agregar controlador

Vaya al directorio *"/ Application / Controllers"* y *agregue el* archivo *[appController.js]*

```
app.controller("ctrl", function ($scope, signalR) {
    $scope.messages = [];
    $scope.user = {};

    signalR.setHubName("chatHub");

    signalR.client().broadcastMessage = function (name, message, time) {
        var newChat = { name: name, message: message, time: time };

        $scope.$apply(function() {
            $scope.messages.push(newChat);
        });
    };

    signalR.start(function () {
        $scope.send = function () {
            var dt = new Date();
            var time = dt.getHours() + ":" + dt.getMinutes() + ":" + dt.getSeconds();

            signalR.server().send($scope.user.name, $scope.user.message, time);
        }
    });
});
```

> signalR.setHubName ("chatHub") | **[ChatHub] (clase pública)>** *ChatHub.cs*
>
> **Nota:** no inserte *HubName* con mayúsculas, la **primera letra** es minúscula.
>
> **signalR.client ()** | este método intenta conectarse a sus hubs y obtener todas las funciones en los Hubs, en este ejemplo tenemos "chatHub", para obtener la función "broadcastMessage ()";

paso 7: agregar index.html en la ruta del directorio

index.html

```
<!DOCTYPE html>
```

```
<html ng-app="app" ng-controller="ctrl">
<head>
    <meta charset="utf-8" />
    <title>SignalR Simple Chat</title>
</head>
<body>
    <form>
        <input type="text" placeholder="name" ng-model="user.name" />
        <input type="text" placeholder="message" ng-model="user.message" />
        <button ng-click="send()">send</button>

        <ul>
            <li ng-repeat="item in messages">
                <b ng-bind="item.name"></b> <small ng-bind="item.time"></small> :
{{item.message}}
            </li>
        </ul>
    </form>

    <script src="Scripts/angular.min.js"></script>
    <script src="Scripts/jquery-1.6.4.min.js"></script>
    <script src="Scripts/jquery.signalR-2.2.1.min.js"></script>
    <script src="signalr/hubs"></script>
    <script src="app.js"></script>
    <script src="SignalR-factory.js"></script>
</body>
</html
```

Resultado con imagen

Usuario 1 (enviar y recibir)

Usuario 2 (enviar y recibir)

Lea SignalR con AngularJs en línea: https://riptutorial.com/es/angularjs/topic/9964/signalr-con-angularjs

Capítulo 48: solicitud de $ http

Examples

Usando $ http dentro de un controlador

El servicio `$http` es una función que genera una solicitud HTTP y devuelve una promesa.

Uso general

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

Uso dentro del controlador

```
appName.controller('controllerName',
    ['$http', function($http){

    // Simple GET request example:
    $http({
        method: 'GET',
        url: '/someUrl'
    }).then(function successCallback(response) {
        // this callback will be called asynchronously
        // when the response is available
    }, function errorCallback(response) {
        // called asynchronously if an error occurs
        // or server returns response with an error status.
    });
}])
```

Métodos de acceso directo

`$http` servicio `$http` también tiene métodos de acceso directo. Lea acerca de los métodos http aquí

Sintaxis

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

Métodos de acceso directo

- $ http.get
- $ http.head
- $ http.post
- $ http.put
- $ http.delete
- $ http.jsonp
- $ http.patch

Usando la solicitud $ http en un servicio

Las solicitudes HTTP se usan ampliamente en repetidas ocasiones en todas las aplicaciones web, por lo que es aconsejable escribir un método para cada solicitud común y luego usarlo en varios lugares a lo largo de la aplicación.

Crear un `httpRequestsService.js`

httpRequestsService.js

```
appName.service('httpRequestsService', function($q, $http){

    return {
        // function that performs a basic get request
        getName: function(){
            // make sure $http is injected
            return $http.get("/someAPI/names")
                .then(function(response) {
                    // return the result as a promise
                    return response;
                }, function(response) {
                    // defer the promise
                    return $q.reject(response.data);
                });
        },

        // add functions for other requests made by your app
        addName: function(){
            // some code...
        }
    }
})
```

El servicio anterior realizará una solicitud de obtención dentro del servicio. Esto estará disponible para cualquier controlador donde se haya inyectado el servicio.

Uso de la muestra

```
appName.controller('controllerName',
    ['httpRequestsService', function(httpRequestsService){

        // we injected httpRequestsService service on this controller
        // that made the getName() function available to use.
        httpRequestsService.getName()
            .then(function(response){
                // success
```

```
            }, function(error){
                // do something with the error
            })
    }])
```

Usando este enfoque, ahora podemos usar **httpRequestsService.js** en cualquier momento y en cualquier controlador.

Tiempo de una solicitud $ http

Las solicitudes de $ http requieren un tiempo que varía según el servidor, algunas pueden tardar unos milisegundos y otras pueden tardar unos segundos. A menudo, el tiempo requerido para recuperar los datos de una solicitud es crítico. Suponiendo que el valor de la respuesta es una matriz de nombres, considere el siguiente ejemplo:

Incorrecto

```
$scope.names = [];

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });

alert("The first name is: " + $scope.names[0]);
```

El acceso a `$scope.names[0]` justo debajo de la solicitud $ http a menudo generará un error: esta línea de código se ejecuta antes de que se reciba la respuesta del servidor.

Correcto

```
$scope.names = [];

$scope.$watch('names', function(newVal, oldVal) {
    if(!(newVal.length == 0)) {
        alert("The first name is: " + $scope.names[0]);
    }
});

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });
```

Usando el servicio $ watch , accedemos a la matriz `$scope.names` solo cuando se recibe la respuesta. Durante la inicialización, la función se llama aunque `$scope.names` se haya inicializado antes, por lo tanto, verificar si `newVal.length` es diferente de 0 es necesario. Tenga en cuenta que cualquier cambio realizado en `$scope.names` activará la función de observación.

Lea solicitud de $ http en línea: https://riptutorial.com/es/angularjs/topic/3620/solicitud-de---http

Capítulo 49: Tareas roncas

Examples

Ejecutar la aplicación localmente

> El siguiente ejemplo requiere que node.js esté instalado y npm esté disponible.
> El código completo de trabajo se puede obtener de GitHub @ https://github.com/mikkoviitala/angular-grunt-run-local

Por lo general, una de las primeras cosas que desea hacer al desarrollar una nueva aplicación web es hacer que se ejecute localmente.

A continuación encontrará un ejemplo completo de cómo lograrlo, usando grunt (javascript task runner), npm (administrador de paquetes de nodo) y bower (otro administrador de paquete).

Además de los archivos de su aplicación real , necesitará instalar algunas dependencias de terceros utilizando las herramientas mencionadas anteriormente. En el directorio de su proyecto, **preferiblemente root** , necesitará tres (3) archivos.

- package.json (dependencias gestionadas por npm)
- bower.json (dependencias gestionadas por bower)
- gruntfile.js (tareas roncas)

Así que su directorio de proyectos se ve así:

---

paquete.json

Vamos a estar instalando **ronco** sí, **matchdep** para hacer nuestra vida más fácil que nos permite filtrar dependencias por su nombre, **ronco-Express** utiliza para iniciar el servidor web expresa a través de gruñido y **gruñido abierta** para abrir direcciones URL / archivos de una tarea ronco.

Entonces, estos paquetes son todo acerca de la "infraestructura" y los ayudantes en los que construiremos nuestra aplicación.

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {},
  "devDependencies": {
    "grunt": "~0.4.1",
    "matchdep": "~0.1.2",
```

```
    "grunt-express": "~1.0.0-beta2",
    "grunt-open": "~0.2.1"
  },
  "scripts": {
    "postinstall": "bower install"
  }
}
```

Bower.json

Bower es (o al menos debería ser) todo acerca del front-end y lo usaremos para instalar **angular** .

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "angular": "~1.3.x"
  },
  "devDependencies": {}
}
```

gruntfile.js

Dentro de gruntfile.js tendremos la magia real de "aplicación local", que abre nuestra aplicación en una nueva ventana del navegador, que se ejecuta en http: // localhost: 9000 /

```
'use strict';

// see http://rhumaric.com/2013/07/renewing-the-grunt-livereload-magic/

module.exports = function(grunt) {
  require('matchdep').filterDev('grunt-*').forEach(grunt.loadNpmTasks);

  grunt.initConfig({
    express: {
      all: {
        options: {
          port: 9000,
          hostname: 'localhost',
          bases: [__dirname]
        }
      }
    },

    open: {
      all: {
        path: 'http://localhost:<%= express.all.options.port%>'
      }
    }
  });

  grunt.registerTask('app', [
    'express',
    'open',
    'express-keepalive'
  ]);
};
```

Uso

Para que su aplicación esté en funcionamiento desde cero, guarde los archivos anteriores en el directorio raíz de su proyecto (cualquier carpeta vacía servirá). Luego, inicie la consola / línea de comando y escriba lo siguiente para instalar todas las dependencias necesarias.

```
npm install -g grunt-cli bower
npm install
```

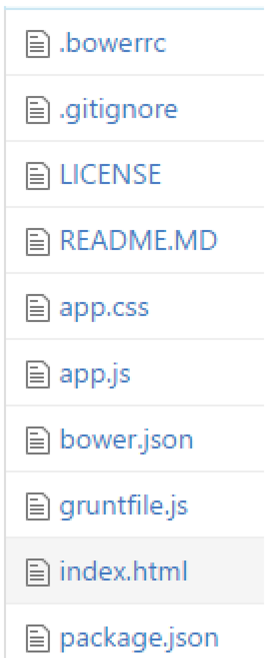
Y luego ejecuta tu aplicación usando

```
grunt app
```

Tenga en cuenta que sí, también necesitará sus archivos de aplicación reales.
Para un ejemplo casi mínimo, examine el repositorio de GitHub mencionado al principio de este ejemplo.

La estructura no es tan diferente. Solo hay una plantilla `index.html` , código angular en `app.js` y algunos estilos en `app.css` . Otros archivos son para configuración de Git y editor y algunas cosas genéricas. ¡Darle una oportunidad!

AngularJS application
Hello Stack Overflow Documentation (beta)



Lea Tareas roncas en línea: https://riptutorial.com/es/angularjs/topic/6077/tareas-roncas

Capítulo 50: Usando AngularJS con TypeScript

Sintaxis

- $ scope: ng.IScope - esta es una forma en mecanografiado para definir el tipo para una variable en particular.

Examples

Controladores angulares en mecanografiado

Como se define en la Documentación de AngularJS.

> Cuando se conecta un Controlador al DOM a través de la directiva ng-controller, Angular creará una instancia de un nuevo objeto Controlador, utilizando la función de constructor del Controlador especificado. Se creará un nuevo ámbito secundario y estará disponible como un parámetro inyectable para la función del constructor del Controlador como $ alcance.

Los controladores se pueden hacer muy fácilmente utilizando las clases de escritura.

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

El Javascript resultante es

```
var App;
(function (App) {
```

```
    var Controllers;
    (function (Controllers) {
        var Address = (function () {
            function Address() {
            }
            return Address;
        }());
        var SampleController = (function () {
            function SampleController($scope) {
                this.setUpWatches($scope);
            }
            SampleController.prototype.setUpWatches = function ($scope) {
                var _this = this;
                $scope.$watch(function () { return _this.firstName; }, function (n, o) {
                    //n is string and so is o
                });
            };
            ;
            return SampleController;
        }());
        Controllers.SampleController = SampleController;
    })(Controllers = App.Controllers || (App.Controllers = {}));
})(App || (App = {}));
//# sourceMappingURL=ExampleController.js.map
```

Después de hacer la clase de controlador, deje que el módulo angular js sobre el controlador se pueda hacer simple usando la clase

```
app
 .module('app')
 .controller('exampleController', App.Controller.SampleController)
```

Uso del controlador con la sintaxis de ControllerAs

El controlador que hemos creado puede ser instanciado y usado usando el `controller as` Sintaxis. Eso es porque hemos puesto la variable directamente en la clase del controlador y no en el `$scope` .

Usar el `controller as someName` es `controller as someName` el controlador de `$scope` sí mismo. Por lo tanto, no hay necesidad de inyectar $ scope como la dependencia en el controlador.

Forma tradicional:

```
// we are using $scope object.
app.controller('MyCtrl', function ($scope) {
  $scope.name = 'John';
});

<div ng-controller="MyCtrl">
  {{name}}
</div>
```

Ahora, con el `controller as` Sintaxis :

```
// we are using the "this" Object instead of "$scope"
app.controller('MyCtrl', function() {
  this.name = 'John';
});

<div ng-controller="MyCtrl as info">
  {{info.name}}
</div>
```

Si crea una instancia de una "clase" en JavaScript, puede hacer esto:

```
var jsClass = function () {
  this.name = 'John';
}
var jsObj = new jsClass();
```

Entonces, ahora podemos usar la instancia `jsObj` para acceder a cualquier método o propiedad de `jsClass` .

En angular, hacemos el mismo tipo de cosa. Usamos el controlador como sintaxis para la instanciación.

Usando Bundling / Minification

La forma en que se inyecta $ scope en las funciones del constructor del controlador es una forma de demostrar y usar la opción básica de inyección de dependencia angular, pero no está lista para la producción ya que no se puede minimizar. Eso es porque el sistema de minificación cambia los nombres de las variables y la inyección de dependencia de anguar utiliza los nombres de los parámetros para saber qué se debe inyectar. Entonces, para un ejemplo, la función constructora de ExampleController se reduce al siguiente código.

```
function n(n){this.setUpWatches(n)
```

y `$scope` se cambia a `n` !
para superar esto podemos agregar una matriz $ inject (`string[]`). De modo que la DI de angular sabe qué inyectar en qué posición está la función de constructor de los controladores.
Así que el texto escrito arriba cambia a

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
```

```
                //n is string and so is o
            });
        };
        static $inject : string[] = ['$scope'];
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

¿Por qué ControllerAs Syntax?

Función del controlador

La función de controlador no es más que una función de constructor de JavaScript. Por lo tanto, cuando una vista carga el `function context` la `function context` (**`this`**) se establece en el objeto controlador.

Caso 1 :

```
this.constFunction = function() { ... }
```

Se crea en el `controller object` , no en `$scope` . Las vistas no pueden acceder a las funciones definidas en el objeto controlador.

Ejemplo:

```
<a href="#123" ng-click="constFunction()"></a> // It will not work
```

Caso 2:

```
$scope.scopeFunction = function() { ... }
```

Se crea en el `$scope object` , no en el `controller object` . Las vistas solo pueden acceder a las funciones definidas en el objeto `$scope` .

Ejemplo:

```
<a href="#123" ng-click="scopeFunction()"></a> // It will work
```

¿Por qué ControllerAs?

- **`ControllerAs`** sintaxis de **`ControllerAs`** hace que sea mucho más claro dónde se manipulan los `oneCtrl.name` `anotherCtrl.name` `oneCtrl.name` y `anotherCtrl.name` hace que sea mucho más fácil identificar que tiene un `name` asignado por varios controladores diferentes para diferentes propósitos, pero si ambos usan el mismo `$scope.name` y dos elementos HTML

diferentes en una página que están vinculados a `{{name}}` lo que es difícil identificar cuál es de cuál controlador.

- Ocultando el `$scope` y exponiendo a los miembros desde el controlador a la vista a través de un `intermediary object` . Al configurar `this.*` , Podemos exponer solo lo que queremos exponer desde el controlador a la vista.

```
<div ng-controller="FirstCtrl">
    {{ name }}
    <div ng-controller="SecondCtrl">
        {{ name }}
        <div ng-controller="ThirdCtrl">
            {{ name }}
        </div>
    </div>
</div>
```

Aquí, en el caso anterior, `{{ name }}` será muy confuso de usar y tampoco sabemos cuál está relacionado con qué controlador.

```
<div ng-controller="FirstCtrl as first">
    {{ first.name }}
    <div ng-controller="SecondCtrl as second">
        {{ second.name }}
        <div ng-controller="ThirdCtrl as third">
            {{ third.name }}
        </div>
    </div>
</div>
```

¿Por qué $ alcance?

- Utilice `$scope` cuando necesite acceder a uno o más métodos de $ scope como `$watch` , `$digest` , `$emit` , `$http` etc.
- limite qué propiedades y / o métodos están expuestos a `$scope` , luego, explícitamente pasándolos a `$scope` según sea necesario.

Lea Usando AngularJS con TypeScript en línea: https://riptutorial.com/es/angularjs/topic/3477/usando-angularjs-con-typescript

Capítulo 51: Uso de directivas incorporadas.

Examples

Ocultar / Mostrar elementos HTML

Este ejemplo oculta mostrar elementos html.

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>

      function HideShowController() {
        var vm = this;
        vm.show=false;
        vm.toggle= function() {
          vm.show=!vm.show;
        }
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .controller("hideShowController", [HideShowController]);
    </script>
  </head>
  <body ng-cloak>
    <div ng-controller="hideShowController as vm">
      <a style="cursor: pointer;" ng-show="vm.show" ng-click="vm.toggle()">Show Me!</a>
      <a style="cursor: pointer;" ng-hide="vm.show" ng-click="vm.toggle()">Hide Me!</a>
    </div>
  </body>
</html>
```

Demo en vivo

Explicación paso a paso:

1. `ng-app="myDemoApp"` , la directiva ngApp le dice a angular que un elemento DOM está controlado por un módulo angular específico llamado "myDemoApp".
2. `<script src="[//angular include]">` include angular js.
3. `HideShowController` función `HideShowController` está definida y contiene otra función llamada `toggle` que ayuda a ocultar mostrar el elemento.
4. `angular.module(...)` crea un nuevo módulo.
5. `.controller(...)` Controlador angular y devuelve el módulo para el encadenamiento;
6. `ng-controller` directiva `ng-controller` es un aspecto clave de cómo angular admite los principios detrás del patrón de diseño Modelo-Vista-Controlador.
7. `ng-show` directiva `ng-show` muestra el elemento HTML dado si la expresión proporcionada es verdadera.
8. `ng-hide` directiva `ng-hide` oculta el elemento HTML dado si la expresión proporcionada es verdadera.

9. `ng-click` directiva `ng-click` dispara una función de conmutación dentro del controlador

Lea Uso de directivas incorporadas. en línea: https://riptutorial.com/es/angularjs/topic/7644/uso-de-directivas-incorporadas-

Capítulo 52: Validación de formularios

Examples

Validación básica de formularios

Una de las fortalezas de Angular es la validación de la forma del lado del cliente.

Tratar con las entradas de formularios tradicionales y tener que usar el procesamiento de estilo jQuery interrogativo puede llevar mucho tiempo y ser complicado. Angular le permite producir formas *interactivas* profesionales con relativa facilidad.

La directiva **ng-model** proporciona un enlace bidireccional con campos de entrada y, por lo general, el atributo **novalidate** también se coloca en el elemento de formulario para evitar que el navegador realice una validación nativa.

Por lo tanto, una forma simple se vería así:

```
<form name="form" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
</form>
```

Para que Angular valide las entradas, use exactamente la misma sintaxis que un elemento de *entrada* regular, a excepción de la adición del atributo **ng-model** para especificar a qué variable vincular en el alcance. El correo electrónico se muestra en el ejemplo anterior. Para validar un número, la sintaxis sería:

```
<input type="number" name="postalcode" ng-model="zipcode" />
```

Los pasos finales para la validación de formularios básicos se conectan a una función de envío de formularios en el controlador mediante **ng-submit** , en lugar de permitir que se realice el envío de formularios predeterminado. Esto no es obligatorio pero generalmente se usa, ya que las variables de entrada ya están disponibles en el alcance y, por lo tanto, están disponibles para su función de envío. También suele ser una buena práctica dar un nombre al formulario. Estos cambios darían lugar a la siguiente sintaxis:

```
<form name="signup_form" ng-submit="submitFunc()" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
  <button type="submit">Signup</button>
</form>
```

Este código anterior es funcional pero hay otra funcionalidad que proporciona Angular.

El siguiente paso es comprender que Angular adjunta atributos de clase utilizando **ng-pristine** , **ng-dirty** , **ng-valid** y **ng-invalid** para el procesamiento de formularios. El uso de estas clases en

su css le permitirá diseñar campos de entrada **válidos / no válidos** y **prístinos / sucios** , y alterar la presentación a medida que el usuario ingresa datos en el formulario.

Estados de forma y entrada

Las formas y entradas angulares tienen varios estados que son útiles al validar el contenido

Estados de entrada

Estado	Descripción
`$touched`	Campo ha sido tocado
`$untouched`	Campo no ha sido tocado
`$pristine`	El campo no ha sido modificado
`$dirty`	Campo ha sido modificado
`$valid`	El contenido del campo es válido
`$invalid`	El contenido del campo no es válido

Todos los estados anteriores son propiedades booleanas y pueden ser verdaderas o falsas.

Con estos, es muy fácil mostrar mensajes a un usuario.

```
<form name="myForm" novalidate>
    <input name="myName" ng-model="myName" required>
    <span ng-show="myForm.myName.$touched && myForm.myName.$invalid">This name is
invalid</span>
</form>
```

Aquí, estamos usando la directiva `ng-show` para mostrar un mensaje a un usuario si ha modificado un formulario pero no es válido.

Clases de CSS

Angular también proporciona algunas clases de CSS para formularios y entradas dependiendo de su estado

Clase	Descripción
`ng-touched`	Campo ha sido tocado
`ng-untouched`	Campo no ha sido tocado
`ng-pristine`	El campo no ha sido modificado
`ng-dirty`	Campo ha sido modificado

Clase	Descripción
`ng-valid`	Campo es valido
`ng-invalid`	El campo no es válido

Puedes usar estas clases para agregar estilos a tus formularios

```
input.ng-invalid {
  background-color: crimson;
}
input.ng-valid {
  background-color: green;
}
```

ngMessages

`ngMessages` se usa para mejorar el estilo para mostrar mensajes de validación en la vista.

Enfoque tradicional

Antes de `ngMessages` , normalmente mostramos los mensajes de validación utilizando las directivas angulares predefinidas `ng-class` . Este enfoque fue basura y una tarea `repetitive` .

Ahora, al usar `ngMessages` podemos crear nuestros propios mensajes personalizados.

Ejemplo

HTML:

```
<form name="ngMessagesDemo">
  <input name="firstname" type="text" ng-model="firstname" required>
  <div ng-messages="ngMessagesDemo.firstname.$error">
    <div ng-message="required">Firstname is required.</div>
  </div>
</form>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular-
messages.min.js"></script>
```

JS:

```
var app = angular.module('app', ['ngMessages']);

app.controller('mainCtrl', function ($scope) {
    $scope.firstname = "Rohit";
});
```

Validación de formularios personalizados

En algunos casos la validación básica no es suficiente. Validación personalizada de soporte angular que agrega funciones de `$validators` objeto `$validators` en el `ngModelController` :

```
angular.module('app', [])
  .directive('myValidator', function() {
    return {
      // element must have ng-model attribute
      // or $validators does not work
      require: 'ngModel',
      link: function(scope, elm, attrs, ctrl) {
        ctrl.$validators.myValidator = function(modelValue, viewValue) {
            // validate viewValue with your custom logic
            var valid = (viewValue && viewValue.length > 0) || false;
            return valid;
        };
      }
    };
```

El validador se define como una directiva que requiere `ngModel` , por lo que para aplicar el validador simplemente agregue la directiva personalizada al control de formulario de entrada.

```
<form name="form">
  <input type="text"
         ng-model="model"
         name="model"
         my-validator>
  <pre ng-bind="'my-validator returned: ' + form.model.$valid"></pre>
</form>
```

Y `my-validator` no tiene que aplicarse en el control de forma nativo. Puede ser cualquier elemento, siempre que sea `ng-model` en sus atributos. Esto es útil cuando tienes algún componente ui de compilación personalizado.

some text

my-validator returned: true

Formularios anidados

A veces es conveniente anidar formularios con el fin de agrupar los controles y las entradas de forma lógica en la página. Sin embargo, los formularios HTML5 no deben estar anidados. Suministros angulares `ng-form` lugar.

```
<form name="myForm" noValidate>
  <!-- nested form can be referenced via 'myForm.myNestedForm' -->
  <ng-form name="myNestedForm" noValidate>
    <input name="myInput1" ng-minlength="1" ng-model="input1" required />
    <input name="myInput2" ng-minlength="1" ng-model="input2" required />
  </ng-form>
```

```
  <!-- show errors for the nested subform here -->
  <div ng-messages="myForm.myNestedForm.$error">
    <!-- note that this will show if either input does not meet the minimum -->
    <div ng-message="minlength">Length is not at least 1</div>
  </div>
</form>

<!-- status of the form -->
<p>Has any field on my form been edited? {{myForm.$dirty}}</p>
<p>Is my nested form valid? {{myForm.myNestedForm.$valid}}</p>
<p>Is myInput1 valid? {{myForm.myNestedForm.myInput1.$valid}}</p>
```

Cada parte de la forma contribuye al estado general de la forma. Por lo tanto, si una de las entradas `myInput1` ha sido editada y está `$dirty` , su formulario que contiene también será `$dirty` . Esto va en cascada a cada formulario que contiene, por lo que tanto `myNestedForm` como `myForm` serán `$dirty` .

Validadores asíncronos

Los validadores asíncronos le permiten validar la información del formulario contra su backend (utilizando $ http).

Este tipo de validadores son necesarios cuando necesita acceder a la información almacenada en el servidor que no puede tener en su cliente por varios motivos, como la tabla de usuarios y otra información de la base de datos.

Para usar los validadores asíncronos, acceda al `ng-model` de su `input` y defina las funciones de devolución de llamada para la propiedad `$asyncValidators` .

Ejemplo:

El siguiente ejemplo verifica si un nombre proporcionado ya existe, el backend devolverá un estado que rechazará la promesa si el nombre ya existe o si no se proporcionó. Si el nombre no existe, devolverá una promesa resuelta.

```
ngModel.$asyncValidators.usernameValidate = function (name) {
     if (name) {
          return AuthenticationService.checkIfNameExists(name); // returns a promise
     } else {
          return $q.reject("This username is already taken!"); // rejected promise
     }
};
```

Ahora, cada vez que se cambia el `ng-model` de la entrada, esta función se ejecutará y devolverá una promesa con el resultado.

Lea Validación de formularios en línea: https://riptutorial.com/es/angularjs/topic/3979/validacion-de-formularios

Creditos

S. No	Capítulos	Contributors
1	Empezando con AngularJS	Abhishek Pandey, After Class, Andrés Encarnación, AnonymousNotReally, badzilla, Charlie H, Chirag Bhatia - chirag64, Community, daniellmb, David G., Devid Farinelli, Eugene, fracz, Franck Dernoncourt, Gabriel Pires, Gourav Garg , H. Pauwelyn, Igor Raush, jengeb, Jeroen, John F., Léo Martin, Lotus91, LucyMarieJ, M. Junaid Salaat, Maaz.Musa, Matt, Mikko Viitala, Mistalis, Nemanja Trifunovic, Nhan, Nico, pathe.kiran, Patrick, Pushpendra, Richard Hamilton, Stepan Suvorov, Stephen Leppik, Sunil Lama, superluminary, Syed Priom, timbo, Ven, vincentvanjoe, Yasin Patel, Ze Rubeus, Артем Комаров
2	Alcances $ angulares	Abhishek Maurya, elliot-j, Eugene, jaredsk, Liron Ilayev, MoLow, Prateek Gupta, RamenChef, ryansstack, Tony
3	Almacenamiento de sesión	Rohit Jindal
4	angularjs con filtro de datos, paginación etc	Paresh Maghodiya
5	AngularJS gotchas y trampas	Alon Eitan, Cosmin Ababei, doctorsherlock, Faruk Yazıcı, ngLover, Phil
6	Carga lenta	Muli Yulzary
7	Cómo funciona el enlace de datos	Lucas L, Sasank Sunkavalli, theblindprophet
8	Compartir datos	elliot-j, Grundy, Lex, Mikko Viitala, Mistalis, Nix, prit4fun, Rohit Jindal, sgarcia.dev, Sunil Lama
9	Componentes	Alon Eitan, Artem K., badzilla, BarakD, Hubert Grzeskowiak, John F., Juri, M. Junaid Salaat, Mansouri, Pankaj Parkar, Ravi Singh, sgarcia.dev, Syed Priom, Yogesh Mangaj
10	Constantes	Sylvain
11	Controladores	Adam Harrison, Aeolingamenfel, Alon Eitan, badzilla, Bon Macalindong, Braiam, chatuur, DerekMT12, Dr. Cool, Florian, George Kagan, Grundy, Jared Hooper, Liron Ilayev, M. Junaid

		Salaat, Mark Cidade, Matthew Green, Mike, Nad Flores, Praveen Poonia, RamenChef, Sébastien Deprez, sgarcia.dev, thegreenpizza, timbo, Und3rTow, WMios
12	Controladores con ES6	Bouraoui KACEM
13	Decoradores	Mikko Viitala
14	Depuración	Aman, AWolf, Vinay K
15	directiva de clase ng	Dr. Cool
16	Directivas incorporadas	Adam Harrison, Alon Eitan, Aron, AWolf, Ayan, Bon Macalindong, CENT1PEDE, Devid Farinelli, DillonChanis, Divya Jain, Dr. Cool, Eric Siebeneich, George Kagan, Grinn, gustavohenke, IncrediApp, kelvinelove, Krupesh Kotecha, Liron Ilayev, m.e.conroy, Maciej Gurban, Mansouri, Mikko Viitala, Mistalis, Mitul, MoLow, Naga2Raja, ngLover, Nishant123, Piet, redunderthebed, Richard Hamilton, svarog, tanmay, theblindprophet, timbo, Tomislav Stankovic, vincentvanjoe, Vishal Singh
17	Directivas personalizadas	Alon Eitan, br3w5, casraf, Cody Stott, Daniel, Everettss, Filipe Amaral, Gaara, Gavishiddappa Gadagi, Jinw, jkris, mnoronha, Pushpendra, Rahul Bhooteshwar, Sajal, sgarcia.dev, Stephan, theblindprophet, TheChetan, Yuri Blanc
18	Directivas utilizando ngModelController	Nikos Paraskevopoulos
19	El yo o esta variable en un controlador	It-Z, Jim
20	enrutador ui	George Kagan, H.T, Michael P. Bazos, Ryan Hamley, sgarcia.dev
21	Enrutamiento usando ngRoute	Alon Eitan, Alvaro Vazquez, camabeh, DotBot, sgarcia.dev, svarog
22	estilo ng	Divya Jain, Jim
23	Eventos	CodeWarrior, Nguyen Tran, Rohit Jindal, RyanDawkins, sgarcia.dev, shaN, Shashank Vivek
24	Filtros	Aeolingamenfel, developer033, Ed Hinchliffe, fracz, gustavohenke, Matthew Green, Nico
25	Filtros personalizados	doodhwala, Pat, Sylvain

26	Filtros personalizados con ES6	Bouraoui KACEM
27	Funciones auxiliares incorporadas	MoLow, Pranav C Balan, svarog
28	Impresión	ziaulain
29	Interceptor HTTP	G Akshay, Istvan Reiter, MeanMan, Mistalis, mnoronha
30	Inyección de dependencia	Andrea, badzilla, Gavishiddappa Gadagi, George Kagan, MoLow, Omri Aharon
31	Migración a Angular 2+	ShinDarth
32	Módulos	Alon Eitan, Ankit, badzilla, Bon Macalindong, Matthew Green, Nad Flores, ojus kulkarni, sgarcia.dev, thegreenpizza
33	MVC angular	Ashok choudhary, Gavishiddappa Gadagi, Jim
34	ng-repetir	Divya Jain, Jim, Sender, zucker
35	ng-view	Aayushi Jain, Manikandan Velayutham, Umesh Shende
36	Opciones de enlaces AngularJS (`=`, `@`, `&` etc.)	Alon Eitan, Lucas L, Makarov Sergey, Nico, zucker
37	Perfil de rendimiento	Deepak Bansal
38	Perfilado y Rendimiento	Ajeet Lakhani, Alon Eitan, Andrew Piliser, Anfelipe, Anirudha, Ashwin Ramaswami, atul mishra, Braiam, bwoebi, chris, Dania, Daniel Molin, daniellmb, Deepak Bansal, Divya Jain, DotBot, Dr. Cool, Durgpal Singh, fracz, Gabriel Pires, George Kagan, Grundy, JanisP, Jared Hooper, jhampton, John Slegers, jusopi, M22an, Matthew Green, Mistalis, Mudassir Ali, Nhan, Psaniko, Richard Hamilton, RyanDawkins, sgarcia.dev, theblindprophet, user3632710, vincentvanjoe, Yasin Patel, Ze Rubeus
39	Prepararse para la producción - Grunt	JanisP
40	Promesas angulares con servicio $ q	Alon Eitan, caiocpricci2, ganqqwerty, georgeawg, John F., Muli Yulzary, Praveen Poonia, Richard Hamilton, Rohit Jindal, svarog
41	Proveedores	Mikko Viitala

42	Proyecto Angular - Estructura de Directorio	jitender, Liron Ilayev
43	Pruebas unitarias	daniellmb, elliot-j, fracz, Gabriel Pires, Nico, ronapelbaum
44	recorrido del bucle de digestión	Alon Eitan, chris, MoLow, prit4fun
45	Servicio Distinguido vs Fábrica	Deepak Bansal
46	Servicios	Abdellah Alaoui, Alvaro Vazquez, AnonDCX, DotBot, elliot-j, Flash, Gavishiddappa Gadagi, Hubert Grzeskowiak, Lex, Nishant123
47	SignalR con AngularJs	Maher
48	solicitud de $ http	CENT1PEDE, jaredsk, Liron Ilayev
49	Tareas roncas	Mikko Viitala
50	Usando AngularJS con TypeScript	Parv Sharma, Rohit Jindal
51	Uso de directivas incorporadas.	Gourav Garg
52	Validación de formularios	Alon Eitan, fantarama, garyx, Mikko Viitala, Richard Hamilton, Rohit Jindal, shane, svarog, timbo