Cubre los elementos del framework hasta la versión **1.3.5**

AngularJS

paso a paso

Crea aplicaciones complejas de forma fácil

MAIKEL RIVERO DORTA

Leanpub

AngularJs Paso a Paso

La primera guía completa en español para adentrarse paso a paso en el mundo de AngularJS

Maikel José Rivero Dorta

Este libro está a la venta en http://leanpub.com/angularjs-paso-a-paso

Esta versión se publicó en 2016-02-02

¡Twitea sobre el libro!

Por favor ayuda a Maikel José Rivero Dorta hablando sobre el libro en Twitter!

El tweet sugerido para este libro es:

"AngularJS Paso a Paso" un libro de @mriverodorta para empezar desde cero. Adquiere tu copia en http://bit.ly/AngularJSPasoAPaso

El hashtag sugerido para este libro es #AngularJS.

Descubre lo que otra gente está diciendo sobre el libro haciendo click en este enlace para buscar el hashtag en Twitter:

https://twitter.com/search?q=#AngularJS

Dedicado a

En primer lugar este libro esta dedicado a todos los que de alguna forma u otra me han apoyado en llevar a cabo la realización de este libro donde plasmo mis mejores deseos de compartir mi conocimiento.

En segundo lugar a toda la comunidad de desarrolladores de habla hispana que en múltiples ocasiones no encuentra documentación en su idioma, ya sea como referencia o para aprender nuevas tecnologías.

Índice general

Agradecimientos

Quisiera agradecer a varias personas que me han ayudado en lograr este proyecto. Primero que todo a Jasel Morera por haber revisado el libro y corregido mucho de los errores de redacción ya que no soy escritor y en ocasiones no sé cómo expresarme y llegar a las personas de una manera correcta. También agradecer a Anxo Carracedo por la foto de los pasos que aparece en la portada. A Wilber Zada Rosendi @wil63r[1] por el diseño de la portada. También a todos los demás que de una forma u otra me han ayudado a hacer realidad esta idea de escribir para la comunidad.

[1] http://twitter.com/wil63r

Traducciones

Si te gustaría traducir este libro a otro lenguaje, por favor escríbeme a **@mriverodorta** con tus intenciones. Ofreceré el 35% de las ganancias por cada libro vendido en tu traducción, la cual será vendida al mismo precio que el original. Además de una página en el libro para la presentación del traductor.

Nótese que el libro ha sido escrito en formato markdown con las especificaciones de Leanpub, las traducciones deberán seguir los mismos pasos.

Prólogo

AngularJs paso a paso cubre el desarrollo de aplicaciones con el framework AngularJs. En este libro se tratarán temas esenciales para el desarrollo de aplicaciones web del lado del cliente. Además, trabajaremos con peticiones al servidor, consumiendo servicios REST y haciendo que nuestro sistema funcione en tiempo real sin tener que recargar la página de nuestro navegador.

Para quien es este libro

Está escrito para desarrolladores de aplicaciones que posean un modesto conocimiento de **Javascript**, así como de **HTML5** y que necesiten automatizar las tareas básicas en el desarrollo de una aplicación web, específicamente en sistemas de una sola página, manejo de rutas, modelos, peticiones a servidores mediante Ajax, manejo de datos en tiempo real y otros.

Que necesitas para este libro

Para un correcto aprendizaje de este libro es necesario una serie de complementos que te permitirán ejecutar los ejemplos y construir tu propia aplicación. Si estaremos hablando sobre el framework **AngularJS** es esencial que lo tengas a tu alcance, lo mismo usando el CDN de Google o mediante una copia en tu disco duro. También necesitarás un navegador para ver el resultado de tu aplicación, recomiendo Google Chrome por su gran soporte de HTML5 y sus herramientas para el desarrollo. Además de lo anteriormente mencionado necesitarás un editor de código. Más adelante estaremos hablando sobre algunas utilidades que harían el desarrollo más fácil pero que no son estrictamente necesarias.

Entiéndase

Se emplearán diferentes estilos de texto, para distinguir entre los diferentes tipos de información. Aquí hay algunos ejemplos de los estilos y explicación de su significado.

Lo ejemplos de los códigos serán mostrado de la siguiente forma:

```
<!DOCTYPE html>
<html lang="es" ng-app="MiApp">
<head>
  <meta charset="UTF-8">
  <title>Titulo</title>
</head>
<body>
  <div ng-controller="MiCtrl">Hola Mundo!</div>
</body>
</html>
```

Feedback

El feedback de los lectores siempre es bienvenido. Me gustaría saber qué piensas acerca de este libro que te ha gustado más y que no te ha gustado. Lo tendré presente para próximas actualizaciones. Para enviar un feedback envía un tweet a **@mriverodorta**.

Errata

Este es el primer libro que escribo así que asumo que encontraran varios errores. Tú puedes ayudarme a corregirlos enviándome un tweet con el error que has encontrado a **@mriverodorta** junto con los detalles del error.

Los errores serán solucionados a medida que sean encontrados. De esta forma estarán arreglados en próximas versiones del libro.

Preguntas

Si tienes alguna pregunta relacionada con algún aspecto del libro puedes hacerla a **@mriverodorta** con tus dudas.

Recursos

AngularJS posee una gran comunidad a su alrededor además del equipo de Google que trabaja dedicado a este framework. A continuación, mencionaré algunos de los sitios donde puedes encontrar recursos y documentación relacionada al desarrollo con AngularJS.

Sitios de referencia

- Sitio web oficial http://www.angularjs.org[2]
- Google+ https://plus.google.com/u/0/communities/115368820700870330756[3]
- Proyecto en Github https://github.com/angular/angular.js[4]
- Grupo de Google angular@googlegroups.com
- Canal en Youtube http://www.youtube.com/user/angularjs[5]
- Twitter @angularjs

Extensiones

La comunidad alrededor de AngularJS ha desarrollado gran cantidad de librerías y extensiones adicionales que agregan diferentes funcionalidades al framework y tienen sitio en: http://ngmodules.org[6].

IDE y Herramientas

Si eres un desarrollador web, para trabajar con AngularJS no es necesario que utilices algo diferente de lo que ya estés acostumbrado, puedes seguir usando HTML y Javascript como lenguajes y si estás dando tus primeros pasos en este campo podrás utilizar un editor de texto común. Aunque te recomendaría usar un [7]IDE que al comienzo te será de mucha ayuda con alguna de sus funciones como el auto-completamiento de código, hasta que tengas un mayor entendimiento de las propiedades y funciones. A continuación, recomendare algunos:

- WebStorm: Es un potente IDE multiplataforma que podrás usar lo mismo en Mac, Linux o Windows. Además, se le puede instalar un Plugin para el trabajo con AngularJS que fue desarrollado por la comunidad.
- SublimeText: También multiplataforma y al igual posee un plugin para AngularJS pero no es un IDE es sólo un editor de texto.
- Espreso: Sólo disponible en Mac enfocado para su uso en el frontend.

Navegador

Nuestra aplicación de AngularJS funciona a través de los navegadores más populares en la actualidad (Google Chrome, Safari, Mozilla Firefox). Aunque recomiendo Google Chrome ya que posee una extensión llamada Batarang para inspeccionar aplicaciones AngularJS y la misma puede ser instalada desde Chrome Web Store.

[2] http://www.angularjs.org
[3] https://plus.google.com/u/0/communities/115368820700870330756
[4] https://github.com/angular/angular.js
[5] http://www.youtube.com/user/angularjs
[6] http://ngmodules.org
[7] Integrated Development Environment

Alcance

Este libro abarcará la mayoría de los temas relacionados con el framework **AngularJS**. Está dirigido a aquellos desarrolladores que ya poseen conocimientos sobre el uso de **AngularJS** y quisieran indagar sobre algún tema en específico. A continuación, describiré por capítulos los temas tratados en este libro.

Capítulo 1: Primeros pasos

En este capítulo se abordarán los temas iniciales para el uso del framework, sus principales vías para obtenerlo y su inclusión en la aplicación. Además de la definición de la aplicación, usos de las primeras directivas y sus ámbitos. La creación del primer controlador y su vinculación con la vista y el modelo. Se explicarán los primeros pasos para el uso del servicio **$scope**.

Capítulo 2: Estructura

Este capítulo se describirá la importancia de tener una aplicación organizada. La estructura de los directorios y archivos. Comentarios sobre el proyecto **angular-seed** para pequeñas aplicaciones y las recomendaciones para aquellas de estructura medianas o grandes. Además de analizar algunos de los archivos esenciales para hacer que el mantenimiento de la aplicación sea sencillo e intuitivo.

Capítulo 3: Módulos

En este capítulo comenzaremos por aislar la aplicación del entorno global con la creación del módulo. Veremos cómo definir los controladores dentro del módulo. También veremos cómo Angular resuelve el problema de la minificación en la inyección de dependencias y por último los métodos de configuración de la aplicación y el espacio para tratar eventos de forma global con el método **config()** y **run()** del módulo.

Capítulo 4: Servicios

AngularJS dispone de una gran cantidad de servicios que hará que el desarrollo de la aplicación sea más fácil mediante la inyección de dependencias. También comenzaremos a definir servicios específicos para la aplicación y se detallarán cada una de las vías para crearlos junto con sus ventajas.

Capítulo 5: Peticiones al servidor

Otra de las habilidades de **AngularJS** es la interacción con el servidor. En este capítulo trataremos lo relacionado con las peticiones a los servidores mediante el servicio **$http**. Como hacer peticiones a recursos en un servidor remoto, tipos de peticiones y más.

Capítulo 6: Directivas

Las directivas son una parte importante de **AngularJS** y así lo reflejará la aplicación que creemos con el framework. En este capítulo haremos un recorrido por las principales directivas, con ejemplos de su uso para que sean más fáciles de asociar. Además, se crearán directivas específicas para la aplicación.

Capítulo 7: Filtros

En este capítulo trataremos todo lo relacionado con los filtros, describiendo los que proporciona angular en su núcleo. También crearemos filtros propios para realizar acciones específicas de la aplicación. Además de su uso en las vistas y los controladores y servicios.

Capítulo 8: Rutas

Una de las principales características de AngularJS es la habilidad que tiene para crear aplicaciones de una sola página. En este capítulo estaremos tratando sobre el módulo **ngRoute**, el tema del manejo de rutas sin recargar la página, los eventos que se procesan en los cambios de rutas. Además, trataremos sobre el servicio **$location**.

Capítulo 9: Eventos

Realizar operaciones dependiendo de las interacciones del usuario es esencial para las aplicaciones hoy en día. **Angular** permite crear eventos y dispararlos a lo largo de la aplicación notificando todos los elementos interesados para tomar acciones. En este capítulo veremos el proceso de la propagación de eventos hacia los **$scopes** padres e hijos, así como escuchar los eventos tomando acciones cuando sea necesario.

Capítulo 10: Recursos

En la actualidad existen cada vez más servicios RESTful en internet, en este capítulo comenzaremos a utilizar el servicio **ngResource** de **Angular**. Realizaremos peticiones a un API REST y ejecutaremos operaciones CRUD en el servidor a través de este servicio.

Capítulo 11: Formularios y Validación

Hoy en día la utilización de los formularios en la web es masiva, por lo general todas las aplicaciones web necesitan al menos uno de estos. En este capítulo vamos a ver como emplear las directivas para validar formularios, así como para mostrar errores dependiendo de la información introducida por el usuario en tiempo real.

Extra: Servidor API RESTful

En el Capítulo 10 se hace uso de una API RESTful para demostrar el uso del servicio **$resource**. En este extra detallaré el proceso de instalación y uso de este servidor que a la vez viene incluido con el libro y estará disponible con cada compra. El servidor esta creado utilizando **NodeJs**, **Express.js** y **MongoDB**.

Introducción

A lo largo de los años hemos sido testigo de los avances y logros obtenidos en el desarrollo web desde la creación de **World Wide Web**. Si comparamos una aplicación de aquellos entonces con una actual notaríamos una diferencia asombrosa, eso nos da una idea de cuan increíble somos los desarrolladores, cuantas ideas maravillosas se han hecho realidad y en la actualidad son las que nos ayudan a obtener mejores resultados en la creación de nuevos productos.

A medida que el tiempo avanza, las aplicaciones se hacen más complejas y se necesitan soluciones más inteligentes para lograr un producto final de calidad. Simultáneamente se han desarrollado nuevas herramientas que ayudan a los desarrolladores a lograr fines en menor tiempo y con mayor eficiencia. Hoy en día las aplicaciones web tienen una gran importancia, por la cantidad de personas que utilizan Internet para buscar información relacionada a algún tema de interés, hacer compras, socializar, presentar su empresa o negocio, en fin, un sin número de posibilidades que nos brinda la red de redes.

Una de las herramientas que nos ayudará mucho en el desarrollo de una aplicación web es **AngularJS**, un framework desarrollado por **Google**, lo que nos da una idea de las bases y el soporte del framework por la reputación de su creador. En adición goza de una comunidad a su alrededor que da soporte a cada desarrollador con soluciones a todo tipo de problemas.

Por estos tiempos existen una gran cantidad de frameworks que hacen un increíble trabajo a la hora de facilitar las tareas de desarrollo. Pero AngularJS viene siendo como el más popular diría yo, por sus componentes únicos, los cuales estaremos viendo más adelante.

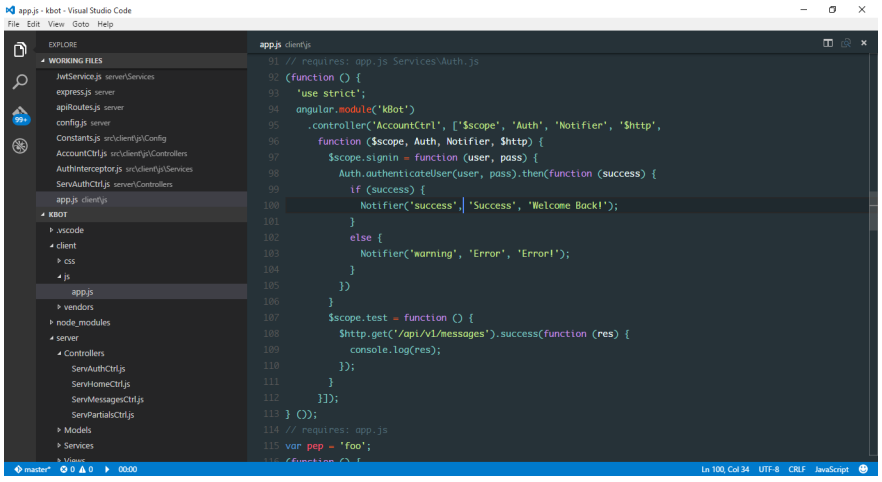
En este libro estaremos tratando el desarrollo de aplicaciones web con la ayuda de AngularJS y veremos cómo esta obra maestra de framework nos hará la vida más fácil a la hora de desarrollar aplicaciones web.

Entorno de desarrollo

Es esencial que para sentirnos cómodos con el desarrollo tengamos a la mano cierta variedad de utilidades para ayudarnos a realizar las tareas de una forma más fácil y en menor tiempo. Esto lo podemos lograr con un buen editor de texto o un IDE. No se necesita alguno específicamente, podrás continuar utilizando el que estás acostumbrado si ya has trabajado Javascript anteriormente.

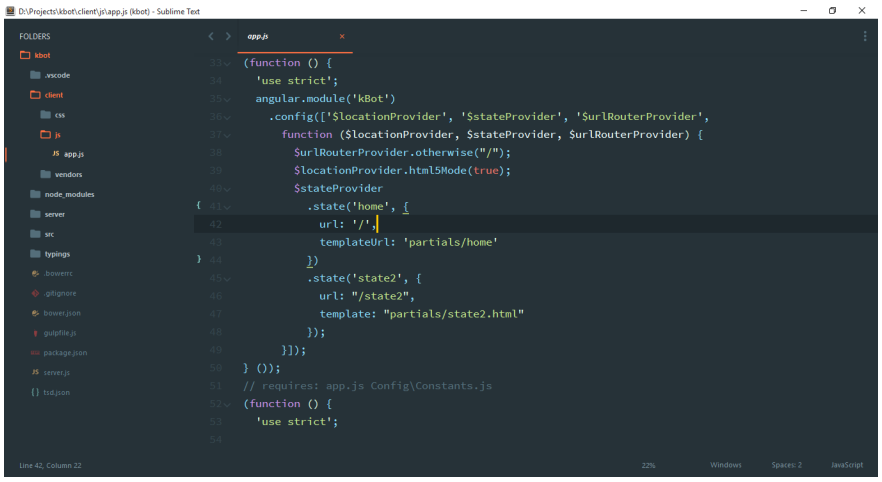
Seleccionando el editor

Existen una gran variedad de editores e IDE en el mercado hoy en día, pero hay algunos que debemos prestar especial atención. Me refiero a editores como **Visual Studio Code** o **Sublime Text 2/3** y al IDE **JetBrains WebStorm**, los tres son multi plataforma. Personalmente uso **Visual Studio Code** para mi desarrollo de día a día, con este editor podremos escribir código de una forma muy rápida gracias a las posibilidades que brinda el uso de las referencias a los archivos de definición.



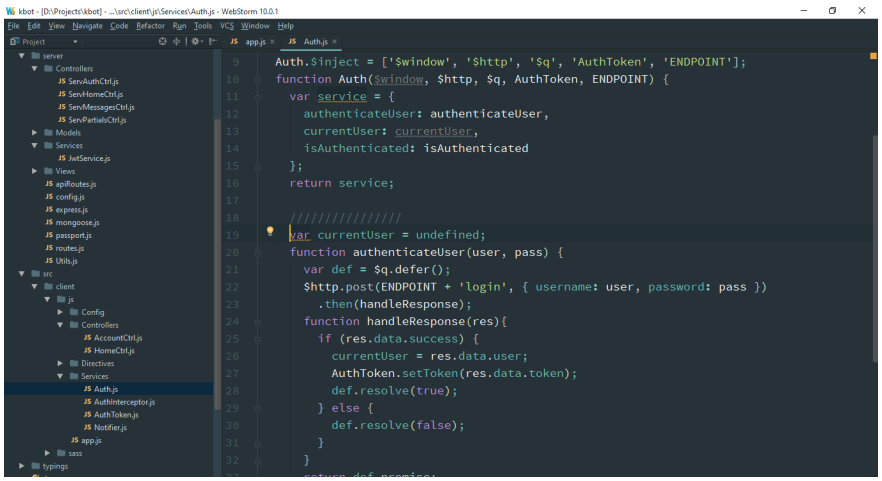
Visual Studio Code

Para **Sublime Text** existen plugins que te ayudarán a aumentar la productividad. El primer plugin es **AngularJs** desarrollado por el grupo de **Angular-UI**, solo lo uso para el auto completamiento de las directivas en las vistas así que en sus opciones deshabilito el auto completamiento en el Javascript. El segundo plugin es **AngularJS Snippets** el cual uso para la creación de controladores, directivas, servicios y más en el Javascript. Estos dos plugins aumentan en gran cantidad la velocidad en que escribes código.



Sublime Text

Por otra parte **WebStorm** es un IDE con todo tipo de funcionalidades, auto completamiento de código, inspección, debug, control de versiones, refactorización y además también tiene un plugin para el desarrollo con **AngularJS** que provee algunas funcionalidades similares a los de **Sublime Text**.



WebStorm

Preparando el servidor

Habiendo seleccionado ya el editor o IDE que usarás para escribir código el siguiente paso es tener listo un servidor donde poder desarrollar la aplicación. En esta ocasión también tenemos varias opciones, si deseas trabajar online Plunker[8] es una buena opción y Cloud9[9] es una opción aún más completa donde podrás sincronizar tu proyecto

[8] http://plnkr.co
[9] http://cloud9.io

mediante git y trabajar en el pc local o en el editor online.

En caso de que quieras tener tu propio servidor local para desarrollo puedes usar **NodeJs** con **ExpressJS** para crear una aplicación. Veamos un ejemplo.

Archivo: App/server.js

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

app.use(express.static(__dirname+'/public'))
   .get('*', function(req, res){
     res.sendFile('/public/index.html', {root:__dirname});
    }).listen(3000);
```

Después de tener este archivo listo ejecutamos el comando **node server.js** y podremos acceder a la aplicación en la maquina local por el puerto 3000 (localhost:3000). Todas las peticiones a la aplicación serán redirigidas a index.html que se encuentra en la carpeta public. De esta forma podremos usar el sistema de rutas de **AngularJS** con facilidad.

Otra opción es usar el servidor **Apache** ya sea instalado en local en el pc como servidor http o por las herramientas **AMP**. Para Mac **MAMP**, windows **WAMP** y linux **LAMP**. Con este podremos crear un host virtual para la aplicación. En la configuración de los sitios disponibles de apache crearemos un virtualhost como el ejemplo siguiente.

```
<VirtualHost *:80>
    # DNS que servirá a este proyecto
    ServerName miapp.dev
    # La direccion de donde se encuentra la aplicacion
    DocumentRoot /var/www/miapp
    # Reglas para la reescritura de las direcciones
    <Directory /var/www/miapp>
        RewriteEngine on
        # No reescribir archivos o directorios.
        RewriteCond %{REQUEST_FILENAME} -f [OR]
        RewriteCond %{REQUEST_FILENAME} -d
        RewriteRule ^ - [L]

        # Reescribir todo lo demás a index.html para usar el modo de rutas HTML5
        RewriteRule ^ index.html [L]
    </Directory>
</VirtualHost>
```

Después de haber configurado el host virtual para la aplicación necesitamos crear el dns local para que responda a nuestra aplicación. En Mac y Linux esto se puede lograr en el archivo **/etc/hosts** y en Windows está en la carpeta dentro de la carpeta del sistema **C:Windows\system32Drivers\etc\hosts**. Escribiendo la siguiente línea al final del archivo.

```
1  127.0.0.1 miapp.dev
```

Después de haber realizado los pasos anteriores reiniciamos el servicio de apache para que cargue las nuevas configuraciones y podremos acceder a la aplicación desde el navegador visitando http://miapp.dev.

Gestionando dependencias

En la actualidad la comunidad desarrolla soluciones para problemas específicos cada vez más rápido. Estas soluciones son compartidas para que otros desarrolladores puedan hacer uso de ellas sin tener que volver a reescribir el código. Un ejemplo es **jQuery**, **LoDash**, **Twitter Bootstrap**, **Backbone** e incluso el mismo **AngularJS**. Sería un poco engorroso si para la aplicación que fuéramos a desarrollar necesitáramos un número considerado de estas librerías y tuviéramos que buscarlas y actualizarlas de forma manual.

Con el objetivo de resolver este problema **Twitter** desarrolló una herramienta llamada **bower** que funciona como un gestor de dependencias y a la vez nos da la posibilidad de compartir nuestras creaciones con la comunidad. Esta herramienta se encargará de obtener todas las dependencias de la aplicación y mantenerlas actualizada por nosotros.

Para instalar bower necesitamos tener instalado previamente **npm** y **NodeJs** en el pc. Ejecutando el comando `npm install -g bower` en la consola podremos instalar bower de forma global en el sistema. Luego de tenerlo instalado podremos comenzar a gestionar las dependencias de la aplicación. Lo primero que necesitamos es crear un archivo **bower.json** donde definiremos el nombre de la aplicación y las dependencias. El archivo tiene la siguiente estructura.

Archivo: App/bower.json

```
{
  "name": "miApp",
  "dependencies": {
    "angular": "~1.2.*"
  }
}
```

De esta forma estamos diciendo a **bower** que nuestra aplicación se llama **miApp** y que necesita **angular** para funcionar. Una vez más en la consola ejecutamos `bower install` en la carpeta que tiene el archivo **bower.json**. Este creará una carpeta **bower_components** donde incluirá el framework para que lo podamos usar en la aplicación.

La creación del archivo **bower.json** lo podemos lograr de forma interactiva. En la consola vamos hasta el directorio de la aplicación y ejecutamos `bower init`. Bower nos hará una serie de preguntas relacionadas con la aplicación y luego creará el archivo **bower.json** con los datos que hemos indicado. Teniendo el archivo listo podemos proceder a instalar dependencias de la aplicación ejecutando `bower install --save angular` lo que instalará **AngularJS** como la vez anterior. El parámetro **–save** es muy importante porque es el que escribirá la dependencia en el archivo **bower.json** de lo contrario **AngularJS** sería instalado pero no registrado como dependencia.

Una de las principales ventajas que nos proporciona **Bower** es que podremos distribuir la aplicación sin ninguna de sus dependencias. Podremos excluir la carpeta de las dependencias sin problemas ya que en cada lugar donde se necesiten las dependencias podremos ejecutar `bower install` y bower las gestionará por nosotros. Esto es muy útil a la hora de trabajar en grupo con sistemas de control de versiones como **Github** ya que en el repositorio solo estaría el archivo **bower.json** y las dependencias en las maquinas locales de los desarrolladores.

Para saber más sobre el uso de **Bower** puedes visitar su página oficial y ver la documentación para conocer acerca de cada una de sus características.

AngularJS y sus características

Con este framework tendremos la posibilidad de escribir una aplicación de manera fácil, que con solo leerla podríamos entender qué es lo que se quiere lograr sin esforzarnos demasiado. Además de ser un framework que sigue el patrón MVC[10] nos brinda otras posibilidades como la vinculación de datos en dos vías y la inyección de dependencia. Sobre estos términos estaremos tratando más adelante.

Plantillas

AngularJS nos permite crear aplicaciones de una sola página, o sea podemos cargar diferentes partes de la aplicación sin tener que recargar todo el contenido en el navegador. Este comportamiento es acompañado por un motor de plantillas que genera contenido dinámico con un sistema de expresiones evaluadas en tiempo real.

El mismo tiene una serie de funciones que nos ayuda a escribir plantillas de una forma organizada y fácil de leer, además de automatizar algunas tareas como son: las iteraciones y condiciones para mostrar contenido. Este sistema es realmente innovador y usa HTML como lenguaje para las plantillas. Es suficientemente inteligente como para detectar las interacciones del usuario, los eventos del navegador y los cambios en los modelos actualizando solo lo necesario en el DOM[11] y mostrar el contenido al usuario.

Estructura MVC

La idea de la estructura MVC no es otra que presentar una organización en el código, donde el manejo de los datos (**M**odelo) estará separado de la lógica (**C**ontrolador) de la aplicación, y a su vez la información presentada al usuario (**V**istas) se encontrará totalmente independiente. Es un proceso bastante sencillo donde el usuario interactúa con las vistas de la aplicación, éstas se comunican con los controladores notificando las acciones del usuario, los controladores realizan peticiones a los modelos y estos gestionan la solicitud según la información brindada. Esta estructura provee una organización esencial a la hora de desarrollar aplicaciones de gran escala, de lo contrario sería muy difícil mantenerlas o extenderlas. Es importante aclarar mencionar que en esta estructura el modelo se refiere a los diferentes tipos de servicios que creamos con Angular.

[10](Model View Controller) Estructura de Modelo, Vista y Controlador introducido en los 70 y obtuvo su popularidad en el desarrollo de aplicaciones de escritorio.

[11]Doccument Object Model

Vinculación de datos

Desde que el DOM pudo ser modificado después de haberse cargado por completo, librerías como jQuery hicieron que la web fuera más amigable. Permitiendo de esta manera que en respuesta a las acciones del usuario el contenido de la página puede ser modificado sin necesidad de recargar el navegador. Esta posibilidad de modificar el DOM en cualquier momento es una de las grandes ventajas que utiliza AngularJS para vincular datos con la vista.

Pero eso no es nuevo, jQuery ya lo hacía antes, lo innovador es, ¿Que tan bueno sería si pudiéramos lograr vincular los datos que tenemos en nuestros modelos y controladores sin escribir nada de código? Seria increíble verdad, pues AngularJS lo hace de una manera espectacular. En otras palabras, nos permite definir que partes de la vista serán sincronizadas con propiedades de Javascript de forma automática. Esto ahorra enormemente la cantidad de código que tendríamos que escribir para mostrar los datos del modelo a la vista, que en conjunto con la estructura **MVC** funciona de maravillas.

Directivas

Si vienes del dominio de jQuery esta será la parte donde te darás cuenta que el desarrollo avanza de forma muy rápida y que seleccionar elementos para modificarlos posteriormente, como ha venido siendo su filosofía, se va quedando un poco atrás comparándolo con el alcance de AngularJS. jQuery en si es una librería que a lo largo de los años ha logrado que la web en general se vea muy bien con respecto a tiempos pasados. A su vez tiene una popularidad que ha ganado con resultados demostrados y posee una comunidad muy amplia alrededor de todo el mundo.

Uno de los complementos más fuertes de AngularJS son las directivas, éstas vienen a remplazar lo que en nuestra web haría jQuery. Más allá de seleccionar elementos del DOM, AngularJS nos permite *extender* la sintaxis de HTML. Con el uso del framework nos daremos cuenta de una gran cantidad de atributos que no son parte de las especificaciones de HTML.

AngularJS tiene una gran cantidad de directivas que permiten que las plantillas sean fáciles de leer y a su vez nos permite llegar a grandes resultados en unas pocas líneas. Pero todo no termina ahí, AngularJS nos brinda la posibilidad de crear nuestras propias directivas para extender el HTML y hacer que nuestra aplicación funcione mucho mejor.

Inyección de dependencia

AngularJS está basado en un sistema de inyección de dependencias donde nuestros controladores piden los objetos que necesitan para trabajar a través del constructor.

Luego AngularJS los inyecta de forma tal que el controlador puede usarlo como sea necesario. De esta forma el controlador no necesita saber cómo funciona la dependencia ni cuáles son las acciones que realiza para entregar los resultados.

Así estamos logrando cada vez más una organización en nuestro código y logrando lo que es una muy buena práctica: "Los controladores deben responder a un principio de responsabilidad única". En otras palabras, el controlador es para controlar, o sea recibe peticiones y entregar respuestas basadas en estas peticiones, no genera el mismo las respuestas. Si todos nuestros controladores siguen este patrón nuestra aplicación será muy fácil de mantener incluso si su proceso de desarrollo es retomado luego de una pausa de largo tiempo.

Si no estás familiarizado con alguno de los conceptos mencionados anteriormente o no te han quedado claros, no te preocupes, todos serán explicados en detalle más adelante. Te invito a que continúes ya que a mi modo de pensar la programación es más de código y no de tantos de conceptos. Muchas dudas serán aclaradas cuando lo veas en la práctica.

Capítulo 1: Primeros pasos

En este capítulo daremos los primeros pasos para el uso de AngularJS. Debemos entender que no es una librería que usa funciones para lograr un fin, AngularJS está pensado para trabajar por módulos, esto le brida una excelente organización a nuestra aplicación. Comenzaremos por lo más básico como es la inclusión de AngularJS y sus plantillas en HTML.

Vías para obtener AngularJS

Existen varias vías para obtener el framework, mencionaré tres de ellas:

La primera forma es descargando el framework de forma manual desde su web oficial http://www.angularjs.org[12] donde tenemos varias opciones, la versión normal y la versión comprimida. Para desarrollar te recomiendo que uses la versión normal ya que la comprimida está pensada para aplicaciones en estado de producción además de no mostrar la información de los errores.

La segunda vía es usar el framework directamente desde el CDN de Google. También encontrará la versión normal y la comprimida. La diferencia de usar una copia local o la del CDN se pone en práctica cuando la aplicación está en producción y un usuario visita cualquier otra aplicación que use la misma versión de AngularJS de tu aplicación, el CDN no necesitará volver a descargar el framework ya que ya el navegador lo tendrá en cache. De esta forma tu aplicación iniciará más rápido.

En tercer lugar, es necesario tener instalado en el pc **npm** y **Bower**. Npm es el gestor de paquetes de NodeJS que se obtiene instalando Nodejs desde su sitio oficial http://nodejs.org[13]. Bower es un gestor de paquetes para el frontend. No explicaré esta vía ya que está fuera del alcance de este libro, pero esta opción esta explicada en varios lugares en Internet, así que una pequeña búsqueda te llevara a obtenerlo.

Nosotros hemos descargado la versión normal desde el sitio oficial y la pondremos en un directorio */lib/angular.js* para ser usado.

Incluyendo AngularJS en la aplicación

Ya una vez descargado el framework lo incluiremos simplemente como incluimos un archivo Javascript externo:

[12] http://www.angularjs.org
[13] http://nodejs.org

```
<script src="lib/angular.js"></script>
```

Si vamos a usar el CDN de Google seria de la siguiente forma:

```
<script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.21/angular.js">
</script>
```

De esta forma ya tenemos el framework listo en nuestra aplicación para comenzar a usarlo.

Atributos HTML5

Como AngularJS tiene un gran entendimiento del HTML, nos permite usar las directivas sin el prefijo **data** por ejemplo, obtendríamos el mismo resultado si escribiéramos el código *data-ng-app* que si escribiéramos *ng-app*. La diferencia está a la hora de que el código pase por los certificadores que al ver atributos que no existen en las especificaciones de HTML5 pues nos darían problemas.

La aplicación

Después de tener AngularJS en nuestra aplicación necesitamos decirle donde comenzar y es donde aparecen las *Directivas*. La directiva **ng-app** define nuestra aplicación. Es un atributo de **clave**=”**valor**” pero en casos de que no hayamos definido un módulo no será necesario darle un valor al atributo. Más adelante hablaremos de los módulos ya que sería el valor de este atributo, por ahora solo veremos lo más elemental.

AngularJS se ejecutará en el ámbito que le indiquemos, es decir abarcará todo el entorno donde usemos el atributo **ng-app**. Si lo usamos en la declaración de HTML entonces se extenderá por todo el documento, en caso de ser usado en alguna etiqueta como por ejemplo en el *body* su alcance se verá reducido al cierre de la misma. Veamos el ejemplo.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>{{Respuesta}}</title>
6  </head>
7  <body  ng-app>
8    <div class="container">
9      Entiendes el contenido de este libro?
10     <input type="checkbox" ng-model="respuesta">
11     <div ng-hide="respuesta">
12       <h2>Me esforzare más!</h2>
13     </div>
14     <div ng-show="respuesta">
15       <h2>Felicidades!</h2>
16     </div>
17   </div>
18   <script src="lib/angular.js"></script>
19 </body>
20 </html>
```

En este ejemplo encontramos varias directivas nuevas, pero no hay que preocuparse, explicaremos todo a lo largo del libro. Podemos observar lo que analizábamos del ámbito de la aplicación en el ejemplo anterior, en la línea 5 donde definimos el título de la página hay unos {{ }}, en angular se usa para mostrar la información del modelo que declaramos en la línea 10 con la directiva **ng-model**. Vamos a llamarlo variables para entenderlo mejor, cuando definimos un modelo con **ng-model** creamos una variable y en el título estamos tratando de mostrar su contenido con la notación {{ }}.

Podemos percatarnos que no tendremos el resultado esperado ya que el título está fuera del ámbito de la aplicación, porque ha sido definida en la línea 7 que es el *body*. Lo que quiere decir que todo lo que esté fuera del *body* no podrá hacer uso de nuestra aplicación. Prueba mover la declaración de **ng-app** a la etiqueta de declaración de HTML en la línea 2 y observa que el resultado es el correcto ya que ahora el título está dentro del ámbito de la aplicación.

Cuidado.

Sólo se puede tener una declaración de **ng-app** por página, sin importar que los ámbitos estén bien definidos.

Ya has comenzado a escribir tu primera aplicación con AngularJS, a diferencia de los clásicos **Hola Mundo!** esta vez hemos hecho algo diferente. Se habrán dado cuenta lo

sencillo que fue interactuar con el usuario y responder a los eventos del navegador, y ni siquiera hemos escrito una línea de Javascript, interesante verdad, pues lo que acabamos de hacer es demasiado simple para la potencia de AngularJS, veremos cosas más interesantes a lo largo del Libro.

A continuación, se analizará las demás directivas que hemos visto en el ejemplo anterior. Para entender el comportamiento de la directiva **ng-model** necesitamos saber qué son los scopes en AngularJS. Pero lo dejaremos para último ya que en ocasiones es un poco complicado explicarlo por ser una característica única de AngularJS y si vienes de usar otros frameworks como Backbone o EmberJS esto resultará un poco confuso.

En el ejemplo anterior hemos hecho uso de otras dos directivas, **ng-show** y **ng-hide** las cuales son empleadas como lo dice su nombre para mostrar y ocultar contenidos en la vista. El funcionamiento de estas directivas es muy sencillo muestra u oculta un elemento HTML basado en la evaluación de la expresión asignada al atributo de la directiva. En otras palabras, evalúa a verdadero o falso la expresión para mostrar u ocultar el contenido del elemento HTML. Hay que tener en cuenta que un valor falso se considerara cualquiera de los siguientes resultados que sean devueltos por la expresión.

- f
- 0
- false
- no
- n
- []

Preste especial atención a este último porque nos será de gran utilidad a la hora de mostrar u ocultar elementos cuando un arreglo esté vacío.

Esta directiva logra su función, pero no por arte de magia, es muy sencillo, AngularJS tiene un amplio manejo de clases **CSS** las cuales vienen incluidas con el framework. Un ejemplo es *.ng-hide*, que tiene la propiedad **display** definida como **none** lo que indica a **CSS** ocultar el elemento que ostente esta clase, además tiene una marca **!important** para que tome un valor superior a otras clases que traten de mostrar el elemento. Las directivas que muestran y ocultan contenido aplican esta clase en caso que quieran ocultar y la remueven en caso que quieran mostrar elementos ya ocultos.

Aquí viene una difícil, **Scopes** y su uso en AngularJS. Creo que sería una buena idea ir viendo su comportamiento y su uso a lo largo del libro y no tratar de definir su concepto ahora, ya que solo confundiría las cosas. Se explicará de forma sencilla según se vaya utilizando. En esencia el scope es el componente que une las plantillas (Vistas) con los controladores, creo que por ahora será suficiente con esto. En el ejemplo anterior en la línea 10 donde utilizamos la directiva **ng-model** hemos hecho uso del scope para definir una variable, la cual podemos usar como cualquier otra variable en Javascript.

Realmente la directiva **ng-model** une un elemento HTML a una propiedad del **$scope** en el controlador. Si esta vez **$scope** tiene un *$* al comienzo, no es un error de escritura, es debido a que **$scope** es un servicio de AngularJS, otro de los temas que estaremos tratando más adelante. En resumen el modelo *respuesta* definido en la línea 10 del ejemplo anterior estaría disponible en el controlador como **$scope.respuesta** y totalmente sincronizado en tiempo real gracias a el motor de plantillas de AngularJS.

Tomando el Control

Veamos ahora un ejemplo un poco más avanzado en el cual ya estaremos usando Javascript y definiremos el primer controlador.

Esta es la parte de la estructura MVC que maneja la lógica de nuestra aplicación. Recibe las interacciones del usuario con nuestra aplicación, eventos del navegador, y las transforma en resultados para mostrar a los usuarios.

Veamos el ejemplo:

```
1  <body  ng-app>
2    <div class="container" ng-controller="miCtrl">
3      <h1>{{ mensaje }}</h1>
4    </div>
5    <script>
6    function miCtrl ($scope) {
7        $scope.mensaje = 'Mensaje desde el controlador';
8    }
9    </script>
10   <script src="lib/angular.js"></script>
11 </body>
```

En este ejemplo hemos usado una nueva directiva llamada **ng-controller** en la línea 2. Esta directiva es la encargada de definir que controlador estaremos usando para el ámbito del elemento HTML donde es utilizada. El uso de esta etiqueta sigue el mismo patrón de ámbitos que el de la directiva **ng-app**. Como has podido notar el controlador es una simple función de Javascript que recibe un parámetro, y en su código sólo define una propiedad mensaje dentro del parámetro.

Esta vez no es un parámetro lo que estamos recibiendo, AngularJS interpretará el código con la inyección de dependencias, como **$scope** es un servicio del framework, creará una nueva instancia del servicio y lo inyectará dentro del controlador haciéndolo así disponible para vincular los datos con la vista. De esta forma todas las propiedades que asignemos al objeto **$scope** estarán disponibles en la vista en tiempo real y completamente sincronizado. El controlador anterior hace que cuando usemos *{{ mensaje }}* en la

vista tenga el valor que habíamos definido en la propiedad con el mismo nombre del **$scope**.

Habrán notado que al recargar la página primero muestra la sintaxis de *{{ mensaje }}* y después muestra el contenido de la variable del controlador. Este comportamiento es debido a que el controlador aún no ha sido cargado en el momento que se muestra esa parte de la plantilla. Lo mismo que pasa cuando tratas de modificar el **DOM** y este aún no está listo. Los que vienen de usar **jQuery** saben a qué me refiero, es que en el momento en que se está tratando de mostrar la variable, aún no ha sido definida. Ahora, si movemos los scripts hacia el principio de la aplicación no tendremos ese tipo de problemas ya que cuando se trate de mostrar el contenido de la variable, esta vez si ya ha sido definido.

Veamos el siguiente ejemplo:

```
<body  ng-app>
  <script src="lib/angular.js"></script>
  <script>
  function miCtrl ($scope) {
      $scope.mensaje = 'Mensaje desde el controlador';
  }
  </script>
  <div class="container" ng-controller="miCtrl">
    <h1>{{ mensaje }}</h1>
  </div>
</body>
```

De esta forma el problema ya se ha resuelto, pero nos lleva a otro problema, que pasa si tenemos grandes cantidades de código y todos están en el comienzo de la página. Les diré que pasa, simplemente el usuario tendrá que esperar a que termine de cargar todos los scripts para que comience a aparecer el contenido, en muchas ocasiones el usuario se va de la página y no espera a que termine de cargar. Claro, no es lo que queremos para nuestra aplicación, además de que es una mala práctica poner los scripts al inicio de la página. Como jQuery resuelve este problema es usando el evento *ready* del **Document**, en otras palabras, el estará esperando a que el **DOM** esté listo y después ejecutará las acciones pertinentes.

Con AngularJS podríamos hacer lo mismo, pero esta vez usaremos algo más al estilo de AngularJS, es una directiva: **ng-bind**=”**expresion**”. Esencialmente **ng-bind** hace que AngularJS remplace el contenido del elemento HTML por el valor devuelto por la expresión. Hace lo mismo que ** {{ }} ** pero con la diferencia de que es una directiva y no se mostrara nada hasta que el contenido no esté listo.

Veamos el siguiente ejemplo:

```
<body  ng-app>
  <div class="container" ng-controller="miCtrl">
    <h1 ng-bind="mensaje"></h1>
  </div>
  <script>
  function miCtrl ($scope) {
      $scope.mensaje = 'Mensaje desde el controlador';
  }
  </script>
  <script src="lib/angular.js"></script>
</body>
```

Como podemos observar en el ejemplo anterior ya tenemos los scripts al final y no tenemos el problema de mostrar contenido no deseado. Al comenzar a cargarse la página se crea el elemento H1 pero sin contenido, y no es hasta que Angular tenga listo el contenido en el controlador y vinculado al **$scope** que se muestra en la aplicación.

Debo destacar que con el uso de la etiqueta **ng-controller** estamos creando un nuevo *scope* para su ámbito cada vez que es usada. Lo anterior, significa que cuando existan tres controladores diferentes cada uno tendrá su propio *scope* y no será accesible a las propiedades de uno al otro. Por otra parte, los controladores pueden estar anidados unos dentro de otros, de esta forma también obtendrán un *scope* nuevo para cada uno, con la diferencia de que el *scope* del controlador hijo tendrá acceso a las propiedades del padre en caso de que no las tenga definidas en sí mismo.

Veamos el siguiente ejemplo:

```
<body  ng-app>
  <div class="container">
    <div ng-controller="padreCtrl">
      <button ng-click="logPadre()">Padre</button>
      <div ng-controller="hijoCtrl">
        <button ng-click="logHijo()">Hijo</button>
        <div ng-controller="nietoCtrl">
          <button ng-click="logNieto()">Nieto</button>
        </div>
      </div>
    </div>
  </div>
  <script>
    function padreCtrl ($scope) {
      $scope.padre = 'Soy el padre';
      $scope.logPadre = function(){
```

```
17          console.log($scope.padre);
18        }
19      }
20      function hijoCtrl ($scope) {
21        $scope.hijo = 'Soy el primer Hijo';
22        $scope.edad = 36;
23        $scope.logHijo = function(){
24          console.log($scope.hijo, $scope.edad);
25        }
26      }
27      function nietoCtrl ($scope) {
28        $scope.nieto = 'Soy el nieto';
29        $scope.edad = 4;
30        $scope.logNieto = function(){
31          console.log($scope.nieto, $scope.edad, $scope.hijo);
32        }
33      }
34    </script>
35    <script src="lib/angular.js"></script>
36  </body>
```

Ops, quizás se haya complicado un poco el código, pero lo describiremos a continuación. Para comenzar veremos que hay una nueva directiva **ng-click**=”“. Esta directiva no tiene nada de misterio, por si misma se explica sola, es la encargada de especificar el comportamiento del evento **Click** del elemento y su valor es evaluado. En cada uno de los botones se le ha asignado un evento **Click** para ejecutar una función en el controlador.

Como han podido observar también cada uno de los controladores están anidados uno dentro de otros, el controlador *nietoCtrl* dentro de *hijoCtrl* y este a su vez dentro de *padreCtrl*. Veamos el contenido de los controladores. En cada uno se definen propiedades y una función que posteriormente es llamada por el evento **Click** de cada botón de la vista. En el *padreCtrl* se ha definido la propiedad *padre* en el **$scope** y ésta es impresa a la consola al ejecutarse la función *logPadre*.

En el *hijoCtrl* se ha definido la propiedad *hijo* y *edad* que igualmente serán impresas a la consola. En el *nietoCtrl* se han definido las propiedades *nieto* y *edad*, de igual forma se imprimen en la consola. Pero en esta ocasión trataremos de imprimir también la propiedad *hijo* la cual no está definida en el **$scope**, así que AngularJS saldrá del controlador a buscarla en el **$scope** del padre.

El resultado de este ejemplo se puede ver en el navegador con el uso de las herramientas de desarrollo en su apartado consola.

Quizás te habrás preguntado si el **$scope** del *padreCtrl* tiene un **scope** padre. Pues la respuesta es si el **$rootScope**. El cual es también un servicio que puede ser inyectado

en el controlador mediante la inyección de dependencias. Este **rootScope** es creado con la aplicación y es único para toda ella, o sea todos los controladores tienen acceso a este **rootScope** lo que quiere decir que todas las propiedades y funciones asignadas a este scope son visibles por todos los controladores y este no se vuelve a crear hasta la página no es recargada.

Estarás pensando que el **rootScope** es la vía de comunicación entre controladores. Puede ser usado con este fin, aunque no es una buena práctica, para cosas sencillas no estaría nada mal. Pero no es la mejor forma de comunicarse entre controladores, ya veremos de qué forma se comunican los controladores en próximos capítulos.

Bindings

El uso del $scope para unir la vista con el controlador y tener disponibilidad de los datos en ambos lugares es una de las principales ventajas que tiene Angular sobre otros frameworks. Aunque no es un elemento único de Angular si es destacable que en otros es mucho más complicado hacer este tipo de vínculo. Para ver lo sencillo que sería recoger información introducida por el usuario, y a la vez mostrarla en algún lugar de la aplicación completamente actualizada en tiempo real, veamos el siguiente ejemplo.

```
1  <body  ng-app>
2    <div ng-controller="ctrl">
3      <p ng-bind="mensaje"></p>
4      <input type="text" ng-model="mensaje">
5    </div>
6    <script src="lib/angular.js"></script>
7    <script>
8      function ctrl($scope) {
9        $scope.mensaje = '';
10     }
11   </script>
12 </body>
```

En el ejemplo anterior podemos observar que a medida que escribimos en la caja de texto, automáticamente se va actualizando en tiempo real en el controlador como en la vista. Como todas las cosas esta funcionalidad viene con un costo adicional, y es que ahora Angular estará pendiente de los cambios realizados por el usuario. Esto significa que en cada interacción del usuario angular ejecutara un **$digest** para actualizar cada elemento necesario.

En cada ocasión que necesitemos observar cambios en algún modelo, Angular colocara un observador (**$watch**) para estar al tanto de algún cambio y poder actualizar la vista

correctamente. Esta funcionalidad es especialmente útil cuando estamos pidiendo datos a los usuarios o esperando algún tipo de información desde un servidor remoto.

También podremos colocar nuestros propios observadores ya que **$watch** es uno de los métodos del servicio **$scope**. Más adelante explicare como establecer observadores y tomar acciones cuando estos se ejecuten.

El método **$digest** procesa todos los observadores (**$watch**) declarados en el $scope y sus hijos. Debido a que algún $watch puede hacer cambios en el modelo, $digest continuará ejecutando los observadores hasta que se deje de hacer cambios. Esto quiere decir que es posible entrar en un bucle infinito, lo que llevaría a un error. Cuando el número de iteraciones sobrepasa 10 este método lanzara un error 'Maximum iteration limit exceeded'.

En la aplicación mientras más modelos tenemos más $watch serán declarados y a la vez más largo será el proceso de $digest. En grandes aplicaciones es importante mantener el control de los ciclos ya que este proceso podría afectar de manera sustancial el rendimiento de la aplicación.

Bind Once Bindings

Una de las nuevas funcionalidades de la versión 1.3 del framework es la posibilidad de crear bind de los modelos sin necesidad de volver a actualizarlos. Es importante mencionar que el uso de esta nueva funcionalidad debe utilizarse cuidadosamente ya que podría traer problemas para la aplicación. Como explique anteriormente en cada ocasión que esperamos cambios en el modelo, es registrado un nuevo $watch para ser ejecutado en el $digest.

Con el nuevo método de hacer binding al modelo Angular simplemente imprimirá el modelo en la vista y se olvidará que tiene que actualizarlo. Esto quiere decir que no estará pendiente de cambios en el modelo para ejecutar el $digest. De esta forma la aplicación podría mejorar en rendimiento drásticamente. Esto es de gran utilidad ya que muchas de las ocasiones donde utilizamos el modelo no tienen cambios después de que se carga la vista, y aun así Angular está observando los cambios en cada uno de ellos.

Es importante que esta funcionalidad se utilice de manera sabia en los lugres que estás seguro que no es necesario actualizar. Por lo general esta funcionalidad tendrá mejor utilidad en grandes aplicaciones donde el $digest ralentiza la ejecución dado la gran cantidad de modelos y ciclos que necesita en las actualizaciones.

Para hacer "one time binding" es muy sencillo solo necesitas poner '::' delante del modelo. Vamos a verlo en una nueva versión del ejemplo anterior.

```
<body  ng-app='app'>
  <div ng-controller="ctrl">
    <p ng-bind="::mensaje"></p>
    <input type="text" ng-model="mensaje">
  </div>
  <script src="bower_components/angular/angular.js"></script>
  <script>
    angular.module('app', [])
      .controller('ctrl', function($scope){
        $scope.mensaje = 'Primer mensaje';
      });
  </script>
</body>
```

Al hacer cambios en la caja de texto podrás notar que en la parte superior no se actualiza el valor. Como podrás darte cuenta esta nueva funcionalidad es muy útil. Existen otros lugares donde podemos hacer uso de esta funcionalidad, como son dentro de la directiva **ng-repeat** para transformar una colección en 'one time binding'. Algo que destacar en el uso con la directiva **ng-repeat** es que los elementos de la colección no se convertirán en 'one time binding'. En otro de los lugares donde podemos hacer uso es dentro de las directivas propias que crees para tu aplicación.

Observadores

Es muy sencillo implementar nuestros propios observadores para actuar cuando se cambia el modelo de alguno de los elementos que observamos. Primero, el servicio *$scope* tiene un método **$watch** que es el que utilizaremos para observar cambios. Este método recibe varios parámetros, primero es una cadena de texto especificando el modelo al que se quiere observar. El segundo parámetro es una función que se ejecutara cada vez que el modelo cambie, esta recibe el nuevo valor y el valor anterior. Y existe un tercer parámetro que es utilizado para comprobar referencias de objetos, pero este no lo utilizaremos muy a menudo.

Vamos a crear un ejemplo con una especie de validación muy sencilla a través del uso de **$watch**. Crearemos un elemento *input* de tipo *password* y comprobaremos si la contraseña tiene un mínimo de 6 caracteres. De no cumplir con esa condición se mostrará un mensaje de error al usuario. Para empezar, crearemos el HTML necesario.

```
<div ng-controller="Controlador">
  <form action="#">
    Contraseña: <input type="password" ng-model="password">
    <p ng-show="errorMinimo">Error: No cumple con el mínimo de caracteres (6)</p>
  </form>
</div>
```

Con la directiva *ng-model* estamos vinculando el *password* con el *$scope* para poder observarlo. No te preocupes por la directiva que se muestra a continuación *ng-show* ya que esta se explicará más adelante en el libro, solo necesitas saber que será la encargada de mostrar y ocultar el mensaje de error. Ahora necesitamos crear el controlador para observar los cambios.

```
angular.module('app', [])
  .controller('Controlador', function ($scope) {
    $scope.errorMinimo = false;
    $scope.$watch('password', function (nuevo, anterior) {
      if (!nuevo) return;
      if (nuevo.length < 6) {
        $scope.errorMinimo = true;
      } else {
        $scope.errorMinimo = false;
      }
    })
  });
```

En el controlador inyectamos el servicio *$scope* y le asignamos una variable *errorMinimo* que será la encargada de definir si se muestra o no el error de validación. Acto seguido implementamos el observador mediante el método **$watch** del *$scope*. Como primer parámetro le pasaremos la cadena que definimos como modelo con la directiva *ng-model* en el HTML. Como segundo parámetro será una funciona anónima que recibirá como parámetros el valor nuevo y el valor anterior. Dentro comprobamos si existe un valor nuevo, de lo contrario salimos de la función. En caso de que exista un valor nuevo comprobamos que este tenga 6 o más caracteres, y definimos el valor de la variable *errorMinimo*.

Ahora podremos ver el ejemplo en funcionamiento. Cuando comencemos a escribir en el veremos que el error aparece mientras no tenemos un mínimo de 6 caracteres en él.

Observadores para grupos

En la versión 1.3 de Angular se añadió una nueva opción para observar grupo de modelos. En esencia el funcionamiento es el mismo al método *$watch* pero en esta ocasión

observará un grupo de modelos y ejecutará la misma acción para cualquier cambio en estos. El nuevo método **watchGroup** recibe como primer parámetro un arreglo de cadenas de texto con el nombre de cada uno de los elementos que se quieren observar.

Como segundo parámetro una función que se ejecutara cuando cualquiera de los elementos observados tenga un cambio. Como con el método *watch* esta función también recibe los valores nuevos y los anteriores, pero en esta ocasión es un arreglo con los nuevos y otro con los antiguos. Es importante mencionar que el orden en que aparecen los valores en el arreglo es el mismo en el que se especificaron en el primer parámetro de *watchGroup*.

Para ver un ejemplo de su uso, vamos a crear algo similar al ejemplo realizado para *watch* pero en esta ocasión validaremos dos elementos *password* y comprobaremos que el valor de uno coincida con el otro. De no coincidir los valores, mostraremos un error anunciando al usuario que los valores no coinciden.

Primero comenzaremos creando el HTML necesario para mostrar dos elementos *password* y el mensaje de error. A cada uno de los elementos le daremos un modelo con la directiva *ng-model,* los cuales serán los mismos que observaremos más adelante en el controlador.

```
1 <div ng-controller="Controlador">
2   <form action="#">
3     Contraseña: <input type="password" ng-model="password"><br><br>
4     Rectificar: <input type="password" ng-model="password2">
5     <p ng-hide="coincidencia">Error: Las contraseñas no coinciden</p>
6   </form>
7 </div>
```

Ahora crearemos el controlador para observar los cambios en el modelo. Primero inyectamos el servicio *$scope* y le asignamos una variable *coincidencia* que será la encargada de mostrar o no el error de validación. Después observaremos el grupo de elementos pasándole como primer parámetro al método *$watchGroup,* un arreglo con los nombres de los modelos que queremos observar.

Como segundo parámetro pasaremos una función anónima que recibirá los valores nuevos y anteriores. Dentro comprobamos que existan valores nuevos, de lo contrario salimos de la función. En caso de que haya valores nuevos, comprobaremos el primer valor del arreglo nuevos contra el segundo valor. Si los valores coinciden marcaremos la coincidencia como verdadero de lo contrario pasaremos un valor falso.

```
angular.module('app', [])
  .controller('Controlador', function ($scope) {
    $scope.coincidencia = false;
    $scope.$watchGroup(['password', 'password2'], function (nuevos, anteriores) {
      if (!nuevos) return;
      if (nuevos[0] === nuevos[1]) {
        $scope.coincidencia = true;
      } else {
        $scope.coincidencia = false;
      }
    })
  });
```

Ahora que el ejemplo está completo puedes ponerlo en práctica y probar escribiendo en los dos elementos *password* para comprobar su funcionalidad. En versiones anteriores, para lograr un comportamiento similar a este, era necesario observar cada uno de los elementos de forma individual.

Controladores como objetos

Debido a la herencia del **$scope** cuando tratamos con controladores anidados, en ocasiones terminamos remplazando elementos por error. Esto podría traer comportamientos no deseados e inesperados en la aplicación, en muchas ocasiones costaría un poco de trabajo encontrar el motivo de los errores. Para solucionar este tipo de problemas y colisiones innecesarias podemos utilizar la sintaxis **controller as**. De esta forma no estaremos utilizando el objeto $scope para exponer elementos de la vista, si no que se utilizara el controlador como un objeto.

Esta sintaxis está disponible desde la versión 1.1.5 de Angular como beta y se hizo estable en versiones posteriores. Para utilizar los controladores por esta vía debemos exponer los elementos como propiedades del mismo controlador utilizando la palabra **this**. De esta forma cuando necesitamos utilizar algún elemento del controlador lo haremos como mismo accedemos a una propiedad de un objeto JavaScript. Veamos un ejemplo.

```
1  <body  ng-app>
2    <div ng-controller="ctrl as lista">
3      {{ lista.elementos }}
4    </div>
5    <script src="lib/angular.js"></script>
6    <script>
7      function ctrl() {
8        this.elementos = 'uno, dos, tres, cuatro.';
9      }
10   </script>
11 </body>
```

Como podrás observar en la línea dos del ejemplo se utiliza la directiva ng-controller y la sintaxis **controller as** de la que hablamos anteriormente. A este controlador le asignamos un nombre *lista* para poder utilizarlo como objeto. En la línea tres del ejemplo interpolamos la propiedad elementos del controlador. Después en la línea ocho exponemos la propiedad elementos del controlador con una cadena de texto. De esta forma la vista está conectada al controlador al igual que si utilizáramos el $scope.

Utilizando este tipo de sintaxis ganamos algunas posibilidades, pero a la vez también perdemos. Si usamos el controlador como un objeto ganamos en cuanto a la organización del código, ya que siempre sabremos de donde proviene el elemento que estamos accediendo. Pero a la vez perdemos la herencia ya que no estaremos accediendo a propiedades del $scope sino del objeto que exponemos en el controlador. Orta punto a tener en cuenta es que al no usar el $scope para unir el controlador con la vista, perderás la posibilidad de utilizar las demás bondades que brinda el objeto $scope en sí.

Controladores Globales

Si estas utilizando una versión de Angular 1.3.X los ejemplos anteriores no te funcionarán, ya que desde esa versión en adelante esta deshabilitado el uso de controladores como funciones globales. Aunque no es recomendado utilizar este tipo de sintaxis para definir los controladores, esta puede ser activada nuevamente mediante la configuración de la aplicación. Para lograrlo debemos primero definir un módulo, en el código a continuación se definirá un módulo para poder configurar la aplicación, este contenido estará detallado en el capítulo tres, si no lo entiendes, no te preocupes, continua y más adelante entenderás a la perfección.

```
<body ng-app="app">
  <div class="container" ng-controller="miCtrl">
    <h1>{{ mensaje }}</h1>
  </div>
  <script src="lib/angular-1.3.js"></script>
  <script>
    var app = angular.module('app',[]);
    app.config(function($controllerProvider){
      $controllerProvider.allowGlobals();
    });
    function miCtrl ($scope) {
        $scope.mensaje = 'Mensaje desde el controlador';
    }
  </script>
</body>
```

En el ejemplo anterior se ha utilizado el mismo controlador del primer ejemplo, pero en esta ocasión se ha incluido el archivo de Angular 1.3 el cual no permite utilizar controladores como funciones globales por defecto. Pero a través de la configuración de la aplicación podemos re activar este comportamiento gracias al método `allowGlobals` del $controllerProvider. Si no entiendes el código anterior no te preocupes, te parecerá mucho más fácil en el futuro cuando expliquemos los módulos y configuración de la aplicación.

Habiendo definido esta configuración ya podemos continuar utilizando funciones globales como controladores. Esta forma de definir controladores es considerada una mala práctica y puede traer problemas graves a tu aplicación debido a la colisión de nombres entre otros.

Capítulo 2: Estructura

AngularJs no define una estructura para la aplicación, tanto en la organización de los ficheros como en los módulos, el framework permite al desarrollador establecer una organización donde mejor considere y más cómodo se sienta trabajando.

Estructura de ficheros.

Antes de continuar con el aprendizaje del framework, creo que es importante desde un principio, tener la aplicación organizada ya que cuando se trata de ordenar una aplicación después de estar algo avanzado, se tiene que parar el desarrollo y en muchas ocasiones hay que reescribir partes del código para que encajen con la nueva estructura que se quiere usar.

Con respecto a este tema es recomendado organizar la aplicación por carpetas temáticas. Los mismos desarrolladores de Angular nos proveen de un proyecto base para iniciar pequeñas aplicaciones. Este proyecto llamado **angular-seed** está disponible para todos en su página de Github: https://github.com/angular/angular-seed y a continuación veremos una breve descripción de su organización.

A lo largo del tiempo que se ha venido desarrollando este proyecto, **angular-seed** ha cambiado mucho en su estructura. En este momento en que estoy escribiendo este capítulo la estructura es la siguiente.

```
App
├── components
|   ├── version
|   |   ├── interpolate-filter.js
|   |   ├── version-directive.js
|   |   ├── version.js
├── view1
|   ├── view1.html
|   ├── view1.js
├── view2
|   ├── view2.html
|   ├── view2.js
├── app.css
├── app.js
├── index.html
```

Al observar la organización de **angular-seed** veremos que en su **app.js** declaran la aplicación y además requieren como dependencias cada una de las vistas y la directiva. Si la aplicación tomara un tamaño considerable, la lista de vistas en los requerimientos del módulo principal sería un poco incómoda de manejar. Dentro de cada carpeta de vista existe un archivo para manejar todo lo relacionado con la misma. En éste crean un nuevo módulo para la vista con toda su configuración y controladores.

Realmente es una semilla para comenzar a crear una aplicación. Un punto de partida para tener una idea de la organización que esta puede tomar. A medida que la aplicación vaya tomando tamaño se puede ir cambiando la estructura. Si es una pequeña aplicación podrás usar **angular-seed** sin problemas. Si tu punto de partida es una aplicación mediana o grande más adelante se explican otras opciones para organizar tu aplicación.

A continuación, hablaremos sobre una de las posibles la organización que se pueden seguir para las medianas aplicaciones. De esta forma los grupos de trabajo podrán localizar las porciones de código de forma fácil.

```
App
├── css
├── img
├── index.html
├── js
|   ├── app.js
|   ├── Config
|   ├── Controllers
|   ├── Directives
|   ├── Filters
|   ├── Models
|   ├── Services
├── partials
```

En esencia ésta es la estructura de directorios para una aplicación mediana. En caso de que se fuera a construir una aplicación grande es recomendable dividirla por módulos, para ello se usaría esta estructura por cada módulo:

```
App
├── css
├── img
├── index.html
├── js
│   ├── app.js
│   ├── Registro
│   │   ├── Registro.js
│   │   ├── Config
│   │   ├── Controllers
│   │   ├── Directives
│   │   ├── Filters
│   │   ├── Models
│   │   ├── Services
│   ├── Blog
│   │   ├── Blog.js
│   │   ├── Config
│   │   ├── Controllers
│   │   ├── Directives
│   │   ├── Filters
│   │   ├── Models
│   │   ├── Services
│   ├── Tienda
│   │   ├── Tienda.js
│   │   ├── Config
│   │   ├── Controllers
│   │   ├── Directives
│   │   ├── Filters
│   │   ├── Models
│   │   ├── Services
│   ├── Ayuda
│   │   ├── Ayuda.js
│   │   ├── Config
│   │   ├── Controllers
│   │   ├── Directives
│   │   ├── Filters
│   │   ├── Models
│   │   ├── Services
├── partials
│   ├── Registro
│   ├── Blog
│   ├── Tienda
```

```
|   ├── Ayuda
```

Como podemos observar al establecer esta estructura en nuestro proyecto, no importa cuánto este crezca, siempre se mantendrá organizado y fácil de mantener. En este libro utilizaremos la estructura para una aplicación mediana, está disponible en el repositorio de Github https://github.com/mriverodorta/ang-starter y viene con ejemplos.

Al observar el contenido de la estructura nos podemos percatar de lo que significa cada uno de sus archivos. Ahora analizaremos algunos de ellos.

En el directorio app/js es donde se guarda todo el código de nuestra aplicación, con excepción de la página principal de entrada a la aplicación y las plantillas (partials), que estarán a un nivel superior junto a los archivos de estilos y las imágenes. En el archivo app.js es donde declararemos la aplicación (módulo) y definiremos sus dependencias. Si has obtenido ya la copia de **ang-starter** desde https://github.com/mriverodorta/ang-starter, veras varios archivos con una configuración inicial para la aplicación. A continuación, describiré el objetivo de cada uno de ellos.

Estructura de la aplicación

Por lo general en términos de programación al crear una aplicación y ésta ser iniciada en muchas ocasiones, necesitamos definir una serie de configuraciones para garantizar un correcto funcionamiento en la aplicación en general. En muchos casos se necesita que estén disponibles desde el mismo inicio de la aplicación, para que los componentes internos que se cargan después puedan funcionar correctamente.

AngularJS nos permite lograr este tipo de comportamiento mediante el método **run()** de los módulos. Este método es esencial para la inicialización de la aplicación. Solo recibe una función como parámetro o un arreglo si utilizamos inyección de dependencia. Este método se ejecutará cuando la aplicación haya cargado todos los módulos. Para el uso de esta funcionalidad se ha dispuesto el archivo **Bootstrap.js**, donde podremos definir comportamientos al inicio de la aplicación. En caso de que se necesite aislar algún comportamiento del archivo **Bootstrap.js** se puede hacer perfectamente ya que **AngularJS** permite la utilización del método **run()** del módulo tantas veces como sea necesario.

Un ejemplo del aislamiento lo veremos en el archivo **Security.js**, donde haremos uso del método **run()** para configurar la seguridad de la aplicación desde el inicio de la misma.

En la mayoría de las aplicaciones se necesitan el uso de las constantes. Los módulos de **AngularJS** proveen un método **constant()** para la declaración de constantes y son un *servicio* con una forma muy fácil de declarar. Este método recibe dos parámetros, **nombre** y **valor**, donde el nombre es el que utilizaremos para inyectar la constante en cualquier lugar que sea necesario dentro de la aplicación, el valor puede ser una cadena

de texto, número, arreglo, objeto e incluso una función. Las constantes las definiremos en el archivo **Constants.js**.

Como he comentado en ocasiones, **AngularJS** tiene una gran cantidad de servicios que los hace disponibles mediante la inyección de dependencias. Muchos de estos servicios pueden ser configurados antes de ser cargando el módulo, para cuando este esté listo ya los servicios estén configurados. Para esto existe el método **config()** de los módulos. Este método recibe como parámetro un arreglo o función para configurar los servicios. Como estamos tratando con aplicaciones de una sola página, el manejo de rutas es esencial para lograrlo. La configuración del servicio **$routeProvider** donde se definen las rutas debe ser configurado con el método **config()** del módulo, ya que necesita estar listo para cuando el módulo este cargado por completo. Estas rutas las podremos definir en el archivo **Routes.js** del cual hablaremos más adelante.

Las aplicaciones intercambian información con el servidor mediante **AJAX**, por lo que es importante saber qué **AngularJS** lo hace a través del servicio **$http**. El mismo puede ser configurado mediante su proveedor **$httpProvider** para editar los **headers** enviados al servidor en cada petición o transformar la respuesta del mismo antes de ser entregada por el servicio. Este comportamiento puede ser configurado en el archivo **HTTP.js**.

En esencia, éste es el contenido de la carpeta **App/Config** de igual forma se puede continuar creando archivos de configuración según las necesidades de cada aplicación y a medida que se vayan usando los servicios. Las configuraciones de los módulos de terceros deben estar situados en **App/Config/Packages** para lograr una adecuada estructura.

Los directorios restantes dentro de la carpeta **App** tienen un significado muy simple: **Controllers**, **Directives** y **Filters** serán utilizados para guardar los controladores, directivas y filtros respectivamente. La carpeta de **Services** será utilizada para organizar toda la lógica de nuestra aplicación que pueda ser extraída de los controladores, logrando de esta forma tener controladores con responsabilidades únicas. Y por último en la carpeta **Models** se maneja todo lo relacionado con datos en la aplicación.

Si logramos hacer uso de esta estructura obtendremos como resultado una aplicación organizada y fácil de mantener.

Capítulo 3: Módulos

Hasta ahora hemos estado declarando el controlador como una función de Javascript en el entorno global, para los ejemplos estaría bien, pero no para una aplicación real. Ya sabemos que el uso del entorno global puede traer efectos no deseados para la aplicación. **AngularJS** nos brinda una forma muy inteligente de resolver este problema y se llama **Módulos**.

Creando módulos

Los módulos son una forma de definir un espacio para nuestra aplicación o parte de la aplicación ya que una aplicación puede constar de varios módulos que se comunican entre sí. La directiva **ng-app** que hemos estado usando en los ejemplos anteriores es el atributo que define cual es el módulo que usaremos para ese ámbito de la aplicación. Aunque si no se define ningún módulo se puede usar **AngularJS** para aplicaciones pequeñas, no es recomendable.

En el siguiente ejemplo definiremos el primer módulo y lo llamaremos **miApp**, a continuación, haremos uso de él.

```
<body  ng-app="miApp">
  <div class="container" ng-controller="miCtrl">
    {{ mensaje }}
  </div>
  <script src="lib/angular.js"></script>
  <script>
    angular.module('miApp', [])
    .controller('miCtrl', function ($scope) {
      $scope.mensaje = 'AngularJS Paso a Paso';
    });
  </script>
</body>
```

En el ejemplo anterior tenemos varios conceptos nuevos. Comencemos por mencionar que al incluir el archivo **angular.js** en la aplicación, éste hace que esté disponible el objeto **angular** en el entorno global o sea como propiedad del objeto **window**, lo podemos comprobar abriendo la consola del navegador en el ejemplo anterior y ejecutando `console.dir(angular)` o `console.dir(window.angular)`

A través de este objeto crearemos todo lo relacionado con la aplicación.

Para definir un nuevo módulo para la aplicación haremos uso del método **module** del objeto **angular** como se puede observar en la *línea 7*. Este método tiene dos funcionalidades: crear nuevos módulos o devolver un módulo existente. Para crear un nuevo módulo es necesario pasar dos parámetros al método. El primer parámetro es el nombre del módulo que queremos crear y el segundo una lista de módulos necesarios para el funcionamiento del módulo que estamos creando. La segunda funcionalidad es obtener un módulo existente, en este caso sólo pasaremos un primer parámetro al método, que será el nombre del módulo que queremos obtener y este será devuelto por el método.

Minificación y Compresión

En el ejemplo anterior donde creábamos el módulo comenzamos a crear los controladores fuera del espacio global, de esta forma no causará problemas con otras librerías o funciones que hayamos definido en la aplicación. En esta ocasión el controlador es creado por un método del módulo que recibe dos parámetros. El primero es una cadena de texto definiendo el nombre del controlador, o un **objeto** de llaves y valores donde la **llave** sería el nombre del controlador y el **valor** el constructor del controlador. El segundo parámetro será una función que servirá como constructor del controlador, este segundo parámetro lo usaremos si hemos pasado una cadena de texto como primer parámetro.

Hasta este punto todo marcha bien, pero en caso de que la aplicación fuera a ser minificada[14] tendríamos un problema ya que la dependencia **$scope** seria reducida y quedaría algo así como:

```
1  .controller('miCtrl',function(a){
```

AngularJS no podría inyectar la dependencia del controlador ya que **a** no es un servicio de **AngularJS**. Este problema tiene una solución muy fácil porque AngularJS nos permite pasar un arreglo como segundo parámetro del método **controller**. Este arreglo contendrá una lista de dependencias que son necesarias para el controlador y como último elemento del arreglo la función de constructor. De esta forma al ser minificado nuestro script no se afectarán los elementos del arreglo por ser solo cadenas de texto y quedaría de la siguiente forma:

```
1  .controller('miCtrl',['$scope',function(a){
```

[14]Minificar es el proceso por el que se someten los scripts para reducir tamaño y así aumentar la velocidad de carga del mismo.

AngularJS al ver este comportamiento inyectará cada uno de los elementos del arreglo a cada uno de las dependencias del controlador. En este caso el servicio **$scope** será inyectado como **a** en el constructor y la aplicación funcionará correctamente.

Es importante mencionar que el orden de los elementos del arreglo será el mismo utilizado por AngularJS para inyectarlos en los parámetros del constructor. En caso de equivocarnos a la hora de ordenar las dependencias podría resultar en comportamientos no deseados. En lo adelante para evitar problemas de minificación el código será escrito como en el siguiente ejemplo.

```
<script>
  angular.module('miApp', [])
  .controller('miCtrl', ['$scope', function ($scope) {
    $scope.mensaje = 'AngularJS Paso a Paso';
  }]);
</script>
```

Inyectar dependencias mediante $inject

Hasta el momento hemos visto como inyectar dependencias mediante la notación del arreglo como se explicó en el apartado de la minificación. Existe otra vía la cual nos permitirá escribir código más fácil de leer e interpretar. Haciendo uso de la propiedad **$inject** de las funciones que utilizaremos, podremos especificar que necesitamos inyectar en estas. Para ver su funcionamiento vamos a ver el siguiente ejemplo.

```
angular.module('app', [])
  .controller('AppCtrl', AppCtrl);

AppCtrl.$inject = ['$scope', '$interval', '$http', '$log'];
function AppCtrl($scope, $interval, $http, $log){
  // Contenido del controlador
}
```

Como habrás podido comprobar es mucho más fácil de entender el código si lo creamos especificando funciones separadas. Hay varias ventajas que nos permite separar el controlador a su propia función nombrada y no en una función anónima. La primera es que es mucho más descriptivo el código a la hora de interpretarlo. La más importante ventaja es la de poder especificar una propiedad **$inject** con todas las dependencias que necesita el controlador.

Esta versión de la inyección de dependencia es la más utilizada por los desarrolladores. Por este motivo en lo adelante esteremos intercambiando entre esta vía para inyectar las dependencias y la que ya sabias anteriormente. De esta forma te será más fácil recordarlas.

Inyección de dependencia en modo estricto

En ocasiones puede ocurrir que olvidemos poner la anotación de alguna de las dependencias que necesita la aplicación. En este caso cuando vallamos a producción y el código seaminificado podríamos tener graves problemas. Para solucionar este problema en Angular 1.3 incluye una nueva directiva **ng-strict-di** que impedirá que la aplicación funcione hasta que todas las dependencias sean anotadas correctamente. Esta directiva debe ser utilizada en el mismo elemento HTML donde definimos la aplicación con **ng-app**.

```
1  <html lang="en" ng-app="miApp" ng-strict-di>
```

Este no es uno de los cambios más importantes de esta versión, pero para los desarrolladores que utilizan las dependencias anotadas les es de gran utilidad.

Configurando la aplicación

En el ciclo de vida de la aplicación AngularJS nos permite configurar ciertos elementos antes de que los módulos y servicios sean cargados. Esta configuración la podemos hacer mediante el módulo que vamos a utilizar para la aplicación. El módulo posee un método **config()** que aceptará como parámetro una función donde inyectaremos las dependencias y configuraremos. Este método es ejecutado antes de que el propio módulo sea cargado.

A lo largo del libro estaremos haciendo uso de este método para configurar varios servicios. Es importante mencionar que un módulo puede tener varias configuraciones, estas serán ejecutadas por orden de declaración. En lo adelante también mencionamos varios servicios que pueden ser configurados en el proceso de configuración del módulo y será refiriendo a ser configurado mediante este método.

La inyección de dependencia en esta función de configuración solo inyectará dos tipos de elementos. El primero serán los servicios que sean definidos con el método **provider**. El segundo son las constantes definidas en la aplicación. Si tratáramos de inyectar algún otro tipo de servicio o **value** obtendríamos un error. La sintaxis de la configuración es la siguiente.

```
1  angular.module('miApp')
2    .config(['$httpProvider', function ($httpProvider) {
3      // Configuraciones al servicio $http.
4    }]);
```

Método run

En algunas ocasiones necesitaremos configurar otros servicios que no hayan sido declarados con el método **provider** del módulo. Para esto el método **config** del módulo no nos funcionará ya que los servicios aún no han sido cargados, incluso ni siquiera el módulo. AngularJS nos permite configurar los demás elementos necesarios de la aplicación justo después de que todos los módulos, servicios han sido cargados completamente y están listos para usarse.

El método **run()** del módulo se ejecutará justo después de terminar con la carga de todos los elementos necesarios de la aplicación. Este método también acepta una función como parámetro y en esta puedes hacer inyección de dependencia. Como todos los elementos han sido cargados puedes inyectar lo que sea necesario. Este método es un lugar ideal para configurar los eventos ya que tendremos acceso al **$rootScope** donde podremos configurar eventos para la aplicación de forma global.

Otro de los usos más comunes es hacer un chequeo de autenticación con el servidor, escuchar para si el servidor cierra la sesión del usuario por tiempo de inactividad cerrarla también en la aplicación cliente. Escuchar los eventos de cambios de la ruta y del servicio **$location**. La Sintaxis es esencialmente igual a la del método **config**.

```
1  angular.module('miApp')
2    .run(['$rootScope', function ($rootScope) {
3      $rootScope.$on('$routeChangeStart', function(e, next,current){
4        console.log('Se comenzará a cambiar la ruta hacia' + next.originalPath);
5      })
6    }]);
```

Después de haber visto como obtener AngularJS, la manera de insertarlo dentro de la aplicación, la forma en que este framework extiende los elementos HTML con nuevos atributos, la definición, la aplicación con módulos y controladores considero que has dado tus primeros pasos. Pero no termina aquí, queda mucho por recorrer. Esto tan solo es el comienzo.