EBook Gratis

APRENDIZAJE Angular 2

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular2

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angular-2

It is an unofficial and free Angular 2 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 Angular 2.

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 Angular 2

Observaciones

Esta sección proporciona una descripción general de cómo instalar y configurar Angular2 + para su uso en diversos entornos e IDE mediante herramientas como la comunidad desarrollada de angular-cli .

La versión anterior de Angular es AngularJS o también llamada Angular 1. Consulte aquí la documentación .

Versiones

Versión	Fecha de lanzamiento
4.3.3	2017-08-02
4.3.2	2017-07-26
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.0	2017-06-08
4.1.0	2017-04-26
4.0.0	2017-03-23
2.3.0	2016-12-08
2.2.0	2016-11-14
2.1.0	2016-10-13
2.0.2	2016-10-05
2.0.1	2016-09-23
2.0.0	2016-09-14
2.0.0-rc.7	2016-09-13
2.0.0-rc.6	2016-08-31
2.0.0-rc.5	2016-08-09
2.0.0-rc.4	2016-06-30

Versión	Fecha de lanzamiento
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03
2.0.0-rc.0	2016-05-02

Examples

Instalar angular2 con angular-cli

Este ejemplo es una configuración rápida de Angular 2 y cómo generar un proyecto de ejemplo rápido.

Requisitos previos:

- Node.js v4 o superior.
- npm v3 o mayor o hilo .

Abra un terminal y ejecute los comandos uno por uno:

```
npm install -g @angular/cli
```

o

```
yarn global add @angular/cli
```

Dependiendo de su elección del gestor de paquetes.

El comando anterior instala @ **angular / cli** globalmente, agregando el ejecutable `ng` a PATH.

Para configurar un nuevo proyecto

Navegue con el terminal a una carpeta donde desee configurar el nuevo proyecto.

Ejecutar los comandos:

```
ng new PROJECT_NAME
cd PROJECT_NAME
ng serve
```

Así es, ahora tiene un proyecto de ejemplo simple hecho con Angular 2. Ahora puede navegar al enlace que se muestra en el terminal y ver qué está ejecutando.

Para agregar a un proyecto existente

Navegue a la raíz de su proyecto actual.

Ejecuta el comando:

```
ng init
```

Esto agregará los andamios necesarios para su proyecto. Los archivos se crearán en el directorio actual, así que asegúrese de ejecutar esto en un directorio vacío.

Ejecutando el proyecto localmente

Para ver e interactuar con su aplicación mientras se ejecuta en el navegador, debe iniciar un servidor de desarrollo local que aloje los archivos para su proyecto.

```
ng serve
```

Si el servidor se inició correctamente, debe mostrar una dirección en la que se ejecuta el servidor. Por lo general es esto:

```
http://localhost:4200
```

Fuera de la caja, este servidor de desarrollo local está conectado con la recarga de módulos calientes, por lo que cualquier cambio en el html, mecanografiado o css, hará que el navegador se vuelva a cargar automáticamente (pero se puede desactivar si se desea).

Generación de componentes, directivas, tuberías y servicios.

El comando `ng generate <scaffold-type> <name>` (o simplemente `ng g <scaffold-type> <name>`) le permite generar automáticamente componentes Angulares:

```
# The command below will generate a component in the folder you are currently at
ng generate component my-generated-component
# Using the alias (same outcome as above)
ng g component my-generated-component
```

Hay varios tipos posibles de andamios que angular-cli puede generar:

Tipo de andamio	Uso
Módulo	`ng g module my-new-module`

Tipo de andamio	Uso
Componente	`ng g component my-new-component`
Directiva	`ng g directive my-new-directive`
Tubo	`ng g pipe my-new-pipe`
Servicio	`ng g service my-new-service`
Clase	`ng g class my-new-class`
Interfaz	`ng g interface my-new-interface`
Enumerar	`ng g enum my-new-enum`

También puede reemplazar el nombre del tipo por su primera letra. Por ejemplo:

`ng gm my-new-module` para generar un nuevo módulo o `ng gc my-new-component` para crear un componente.

Construcción / Bundling

Cuando haya terminado de construir su aplicación web Angular 2 y desee instalarla en un servidor web como Apache Tomcat, todo lo que debe hacer es ejecutar el comando de compilación con o sin el conjunto de indicadores de producción. La producción minimiza el código y se optimiza para un entorno de producción.

```
ng build
```

o

```
ng build --prod
```

Luego busque en la carpeta raíz del proyecto una carpeta `/dist` , que contiene la compilación.

Si desea obtener los beneficios de un paquete de producción más pequeño, también puede usar la compilación de la plantilla Anticipada, que elimina el compilador de la plantilla de la compilación final:

```
ng build --prod --aot
```

Examen de la unidad

Angular 2 proporciona pruebas unitarias integradas, y cada elemento creado por angular-cli genera una prueba unitaria básica, que puede ampliarse. Las pruebas unitarias se escriben con jazmín y se ejecutan a través de Karma. Para iniciar la prueba ejecuta el siguiente comando:

```
ng test
```

Este comando ejecutará todas las pruebas en el proyecto y las volverá a ejecutar cada vez que cambie un archivo fuente, ya sea una prueba o un código de la aplicación.

Para más información también visite: página angular-cli github

Primeros pasos con Angular 2 sin angular-cli.

Angular 2.0.0-rc.4

En este ejemplo crearemos un "¡Hola mundo!" aplicación con un solo componente raíz (`AppComponent`) por simplicidad.

Requisitos previos:

- Node.js v5 o posterior
- npm v3 o posterior

> **Nota:** puede verificar las versiones ejecutando los `node -v` y `npm -v` en la consola / terminal.

Paso 1

Crea e ingresa una nueva carpeta para tu proyecto. Llamémoslo `angular2-example` .

```
mkdir angular2-example
cd angular2-example
```

Paso 2

Antes de comenzar a escribir nuestro código de aplicación, agregaremos los 4 archivos que se proporcionan a continuación: `package.json` , `tsconfig.json` , `typings.json` y `systemjs.config.js` .

> **Descargo de responsabilidad:** los mismos archivos se pueden encontrar en el inicio rápido oficial de 5 minutos .

`package.json` : nos permite descargar todas las dependencias con npm y proporciona una ejecución de script simple para hacer la vida más fácil para proyectos simples. (Debería considerar usar algo como Gulp en el futuro para automatizar tareas).

```
{
  "name": "angular2-example",
  "version": "1.0.0",
  "scripts": {
    "start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" ",
    "lite": "lite-server",
    "postinstall": "typings install",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "typings": "typings"
  },
```

```
  "license": "ISC",
  "dependencies": {
    "@angular/common": "2.0.0-rc.4",
    "@angular/compiler": "2.0.0-rc.4",
    "@angular/core": "2.0.0-rc.4",
    "@angular/forms": "0.2.0",
    "@angular/http": "2.0.0-rc.4",
    "@angular/platform-browser": "2.0.0-rc.4",
    "@angular/platform-browser-dynamic": "2.0.0-rc.4",
    "@angular/router": "3.0.0-beta.1",
    "@angular/router-deprecated": "2.0.0-rc.2",
    "@angular/upgrade": "2.0.0-rc.4",
    "systemjs": "0.19.27",
    "core-js": "^2.4.0",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.6",
    "zone.js": "^0.6.12",
    "angular2-in-memory-web-api": "0.0.14",
    "bootstrap": "^3.3.6"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.2.0",
    "typescript": "^1.8.10",
    "typings":"^1.0.4"
  }
}
```

`tsconfig.json` : configura el transpiler de TypeScript.

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  }
}
```

`typings.json` - Hace que TypeScript reconozca las bibliotecas que estamos usando.

```
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160602141332",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160621231320"
  }
}
```

`systemjs.config.js` : configura SystemJS (también puede usar webpack).

```
/**
 * System configuration for Angular 2 samples
```

```
 * Adjust as necessary for your application's needs.
 */
(function(global) {
  // map tells the System loader where to look for things
  var map = {
    'app':                        'app', // 'dist',
    '@angular':                   'node_modules/@angular',
    'angular2-in-memory-web-api': 'node_modules/angular2-in-memory-web-api',
    'rxjs':                       'node_modules/rxjs'
  };
  // packages tells the System loader how to load when no filename and/or no extension
  var packages = {
    'app':                        { main: 'main.js',  defaultExtension: 'js' },
    'rxjs':                       { defaultExtension: 'js' },
    'angular2-in-memory-web-api': { main: 'index.js', defaultExtension: 'js' },
  };
  var ngPackageNames = [
    'common',
    'compiler',
    'core',
    'forms',
    'http',
    'platform-browser',
    'platform-browser-dynamic',
    'router',
    'router-deprecated',
    'upgrade',
  ];
  // Individual files (~300 requests):
  function packIndex(pkgName) {
    packages['@angular/'+pkgName] = { main: 'index.js', defaultExtension: 'js' };
  }
  // Bundled (~40 requests):
  function packUmd(pkgName) {
    packages['@angular/'+pkgName] = { main: '/bundles/' + pkgName + '.umd.js',
defaultExtension: 'js' };
  }
  // Most environments should use UMD; some (Karma) need the individual index files
  var setPackageConfig = System.packageWithIndex ? packIndex : packUmd;
  // Add package entries for angular packages
  ngPackageNames.forEach(setPackageConfig);
  var config = {
    map: map,
    packages: packages
  };
  System.config(config);
})(this);
```

Paso 3

Instalemos las dependencias escribiendo

```
npm install
```

En la consola / terminal.

Etapa 4

Crea `index.html` dentro de la carpeta `angular2-example` .

```
<html>
  <head>
    <title>Angular2 example</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app></my-app>
  </body>
</html>
```

Su aplicación se procesará entre las etiquetas `my-app` .

Sin embargo, Angular todavía no sabe *qué* renderizar. Para decirle eso, definiremos `AppComponent` .

Paso 5

Cree una subcarpeta llamada `app` donde podamos definir los componentes y servicios que conforman nuestra aplicación. (En este caso, sólo se va a contener el `AppComponent` código y `main.ts`).

```
mkdir app
```

Paso 6

Crea el archivo `app/app.component.ts`

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

@Component({
  selector: 'my-app',
  template: `
    <h1>{{title}}</h1>
    <ul>
        <li *ngFor="let message of messages">
            {{message}}
        </li>
    </ul>
  `
})
```

```
export class AppComponent {
    title = "Angular2 example";
    messages = [
        "Hello World!",
        "Another string",
        "Another one"
    ];
}
```

¿Qué esta pasando? Primero, estamos importando el decorador `@Component` que usamos para darle a Angular la etiqueta y la plantilla HTML para este componente. Luego, estamos creando la clase `AppComponent` con variables de `title` y `messages` que podemos usar en la plantilla.

Ahora veamos esa plantilla:

```
<h1>{{title}}</h1>
<ul>
    <li *ngFor="let message of messages">
        {{message}}
    </li>
</ul>
```

Estamos mostrando la variable de `title` en una etiqueta `h1` y luego hacemos una lista que muestra cada elemento de la matriz de `messages` usando la directiva `*ngFor` . Para cada elemento de la matriz, `*ngFor` crea una variable de `message` que usamos dentro del elemento `li` . El resultado será:

```
<h1>Angular 2 example</h1>
<ul>
    <li>Hello World!</li>
    <li>Another string</li>
    <li>Another one</li>
</ul>
```

Paso 7

Ahora creamos un archivo `main.ts` , que será el primer archivo que Angular `main.ts` .

Crea el archivo `app/main.ts`

```
import { bootstrap }    from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Estamos importando la función `bootstrap` y la clase `AppComponent` , luego usamos `bootstrap` para decirle a Angular qué componente usar como raíz.

Paso 8

Es hora de encender tu primera aplicación. Tipo

```
npm start
```

en su consola / terminal. Esto ejecutará una secuencia de comandos preparada desde `package.json` que inicia lite-server, abre su aplicación en una ventana del navegador y ejecuta el transpiler TypeScript en modo de vigilancia (por lo `.ts` archivos `.ts` se transcribirán y el navegador se actualizará cuando guarde los cambios) .

¿Ahora que?

Consulte la guía oficial de Angular 2 y los otros temas en la documentación de StackOverflow .

También puede editar `AppComponent` para usar plantillas externas, estilos o agregar / editar variables de componentes. Debería ver sus cambios inmediatamente después de guardar los archivos.

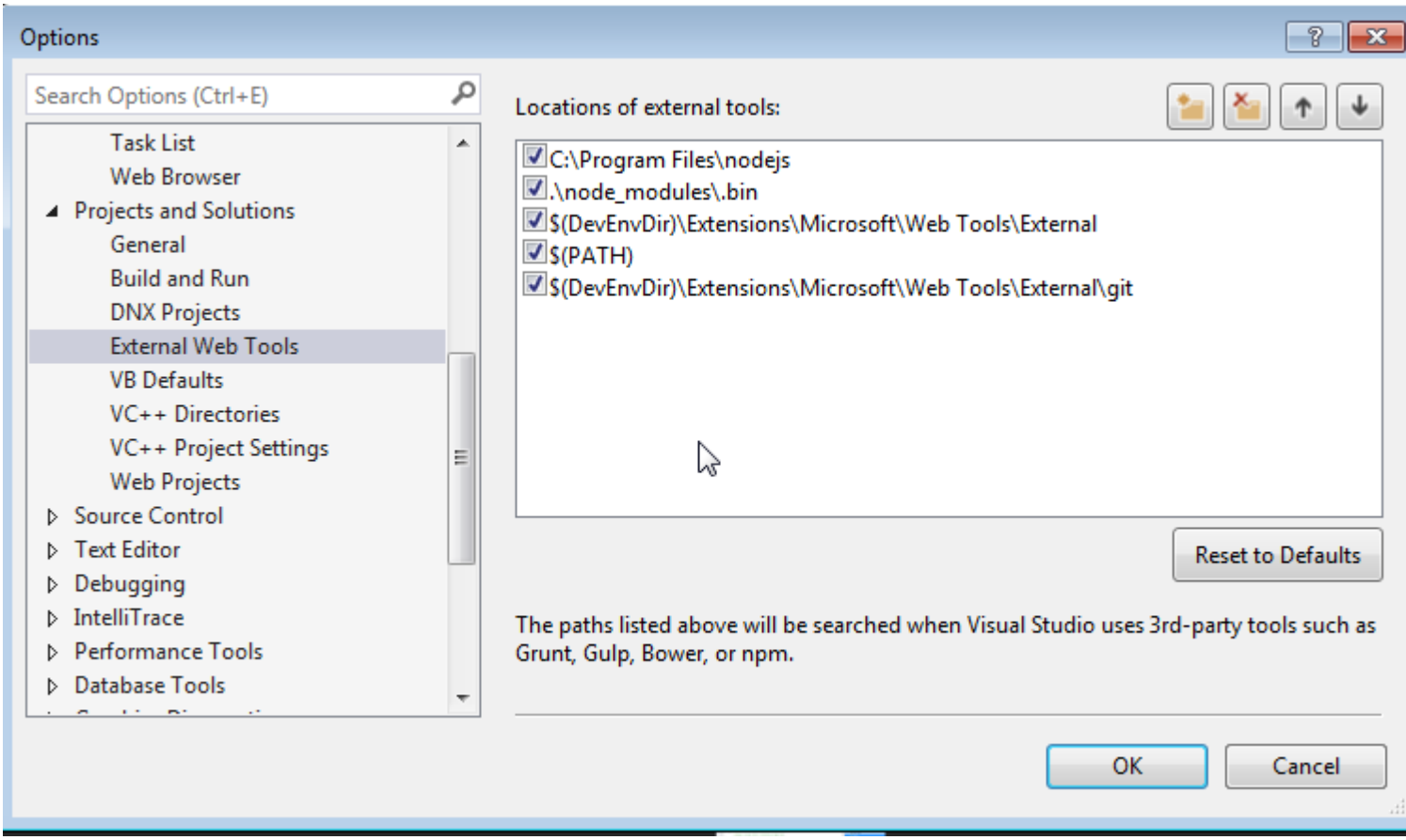
Mantener Visual Studios sincronizados con las actualizaciones de NPM y NODE

Paso 1: localice su descarga de Node.js, normalmente se instala en C: / archivos de programa / nodejs

Paso 2: Abre Visual Studios y navega a "Herramientas> Opciones"

Paso 3: en la ventana de opciones, vaya a "Proyectos y soluciones> Herramientas web externas"

Paso 4: agregue una nueva entrada con su ubicación de archivo Node.js (C: / program files / nodejs), IMPORTANTE use los botones de flecha en el menú para mover su referencia a la parte superior de la lista.



Paso 5: reinicie Visual Studios y ejecute una instalación npm, en su proyecto, desde la ventana de comandos de npm

Conseguir a través de esa empresa molesta proxy

Si está intentando ejecutar un sitio Angular2 en su computadora de trabajo con Windows en XYZ MegaCorp, es probable que tenga problemas para comunicarse con el proxy de la empresa.

Hay (al menos) dos administradores de paquetes que necesitan pasar por el proxy:

1. NPM
2. Mecanografía

Para NPM necesita agregar las siguientes líneas al archivo `.npmrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT]/
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT]/
```

Para Typings necesita agregar las siguientes líneas al archivo `.typingsrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT]/
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT]/
rejectUnauthorized=false
```

Es probable que estos archivos aún no existan, por lo que puede crearlos como archivos de texto en blanco. Se pueden agregar a la raíz del proyecto (en el mismo lugar que `package.json` o puede

colocarlos en `%HOMEPATH%` y estarán disponibles para todos sus proyectos).

La parte que no es evidente y es la razón principal por la que la gente piensa la configuración del proxy no están funcionando es el `%5C` , que es la codificación URL de la `\` para separar los nombres de dominio y de usuario. Gracias a Steve Roberts por eso: usar npm detrás del proxy corporativo .pac

Comenzando con Angular 2 con node.js / expressjs backend (ejemplo http incluido)

Vamos a crear un simple "¡Hola mundo!" aplicación con Angular2 2.4.1 (cambio `@NgModule`) con un nodo.js (expressjs) backend.

Prerrequisitos

- Node.js v4.xx o superior
- npm v3.xx o superior o hilo

A continuación, ejecute `npm install -g typescript` o `yarn global add typescript npm install -g typescript yarn global add typescript` para instalar typescript globalmente

Mapa vial

Paso 1

Cree una nueva carpeta (y el directorio raíz de nuestro back-end) para nuestra aplicación. Llamémoslo `Angular2-express` .

línea de comando :

```
mkdir Angular2-express
cd Angular2-express
```

Paso 2

Cree el `package.json` (para dependencias) y `app.js` (para bootstrapping) para nuestra aplicación `node.js`

paquete.json:

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "start": "node app.js"
```

```
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.13.3",
    "express": "^4.13.3"
  }
}
```

app.js:

```
var express = require('express');
var app = express();
var server = require('http').Server(app);
var bodyParser = require('body-parser');

server.listen(process.env.PORT || 9999, function(){
    console.log("Server connected. Listening on port: " + (process.env.PORT || 9999));
});

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}) );

app.use( express.static(__dirname + '/front' ) );

app.get('/test', function(req,res){ //example http request receiver
  return res.send(myTestVar);
});

//send the index.html on every page refresh and let angular handle the routing
app.get('/*',  function(req, res, next) {
    console.log("Reloading");
    res.sendFile('index.html', { root: __dirname });
});
```

A continuación, ejecute una `npm install` o `yarn` para instalar las dependencias.

Ahora nuestra estructura de back-end está completa. Vamos a pasar al frente.

Paso 3

Nuestro front-end debe estar en una carpeta llamada `front` dentro de nuestra carpeta `Angular2-express` .

línea de comando:

```
mkdir front
cd front
```

Al igual que hicimos con nuestro back-end, nuestro front-end también necesita los archivos de dependencia. Continuemos y `systemjs.config.js` los siguientes archivos: `package.json` , `systemjs.config.js` , `tsconfig.json`

paquete.json :

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "scripts": {
    "tsc": "tsc",
    "tsc:w": "tsc -w"
  },
  "licenses": [
    {
      "type": "MIT",
      "url": "https://github.com/angular/angular.io/blob/master/LICENSE"
    }
  ],
  "dependencies": {
    "@angular/common": "~2.4.1",
    "@angular/compiler": "~2.4.1",
    "@angular/compiler-cli": "^2.4.1",
    "@angular/core": "~2.4.1",
    "@angular/forms": "~2.4.1",
    "@angular/http": "~2.4.1",
    "@angular/platform-browser": "~2.4.1",
    "@angular/platform-browser-dynamic": "~2.4.1",
    "@angular/platform-server": "^2.4.1",
    "@angular/router": "~3.4.0",
    "core-js": "^2.4.1",
    "reflect-metadata": "^0.1.8",
    "rxjs": "^5.0.2",
    "systemjs": "0.19.40",
    "zone.js": "^0.7.4"
  },
  "devDependencies": {
    "@types/core-js": "^0.9.34",
    "@types/node": "^6.0.45",
    "typescript": "2.0.2"
  }
}
```

systemjs.config.js:

```
/**
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    defaultJSExtensions:true,
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {
      // our app is within the app folder
      app: 'app',
      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
      '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
      '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-
browser.umd.js',
```

```
      '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-
dynamic/bundles/platform-browser-dynamic.umd.js',
      '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
      '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
      '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
      // other libraries
      'rxjs':                      'npm:rxjs',
      'angular-in-memory-web-api': 'npm:angular-in-memory-web-api',
    },
    // packages tells the System loader how to load when no filename and/or no extension
    packages: {
      app: {
        main: './main.js',
        defaultExtension: 'js'
      },
      rxjs: {
        defaultExtension: 'js'
      }
    }
  });
})(this);
```

tsconfig.json:

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "compileOnSave": true,
  "exclude": [
    "node_modules/*"
  ]
}
```

A continuación, ejecute una `npm install` o `yarn` para instalar las dependencias.

Ahora que nuestros archivos de dependencia están completos. Vayamos a nuestro `index.html` :

index.html:

```
<html>
  <head>
    <base href="/">
    <title>Angular2-express</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
     <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
```

```
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>

  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

Ahora estamos listos para crear nuestro primer componente. Crea una carpeta llamada `app` dentro de nuestra carpeta `front` .

línea de comando:

```
mkdir app
cd app
```

Hagamos los siguientes archivos llamados `main.ts` , `app.module.ts` , `app.component.ts`

main.ts:

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

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

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

app.module.ts:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpModule } from "@angular/http";

import { AppComponent }   from './app.component';

@NgModule({
  imports:      [
    BrowserModule,
    HttpModule
  ],
  declarations: [
    AppComponent
  ],
  providers:[ ],
  bootstrap:    [ AppComponent ]
})
export class AppModule {}
```

app.component.ts:

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

@Component({
  selector: 'my-app',
  template: 'Hello World!',
  providers: []
})
export class AppComponent {
  constructor(private http: Http){
    //http get example
    this.http.get('/test')
     .subscribe((res)=>{
       console.log(res);
     });
  }
}
```

Después de esto, compile los archivos mecanografiados en archivos javascript. Vaya a 2 niveles desde el directorio actual (dentro de la carpeta Angular2-express) y ejecute el siguiente comando.

línea de comando:

```
cd ..
cd ..
tsc -p front
```

Nuestra estructura de carpetas debe verse como:

```
Angular2-express
├── app.js
├── node_modules
├── package.json
├── front
|   ├── package.json
|   ├── index.html
|   ├── node_modules
|   ├── systemjs.config.js
|   ├── tsconfig.json
|   ├── app
|   |   ├── app.component.ts
|   |   ├── app.component.js.map
|   |   ├── app.component.js
|   |   ├── app.module.ts
|   |   ├── app.module.js.map
|   |   ├── app.module.js
|   |   ├── main.ts
|   |   ├── main.js.map
|   |   ├── main.js
```

Finalmente, dentro de la carpeta Angular2-express, ejecute el comando `node app.js` en la línea de comandos. Abra su navegador favorito y verifique `localhost:9999` para ver su aplicación.

¡Vamos a sumergirnos en Angular 4!

Angular 4 ya está disponible! En realidad, Angular usa semver desde Angular 2, lo que requiere

que se aumente el número mayor cuando se introdujeron cambios de ruptura. El equipo de Angular pospuso las características que causan cambios de última hora, que se lanzarán con Angular 4. Angular 3 se omitió para poder alinear los números de versión de los módulos principales, porque el Router ya tenía la versión 3.

Según el equipo de Angular, las aplicaciones de Angular 4 consumirán menos espacio y serán más rápidas que antes. Han separado el paquete de animación del paquete @ angular / core. Si alguien no está utilizando el paquete de animación, el espacio adicional de código no terminará en la producción. La sintaxis de enlace de plantilla ahora admite sintaxis de estilo if / else. Angular 4 ahora es compatible con la versión más reciente de Typescript 2.1 y 2.2. Entonces, Angular 4 va a ser más emocionante.

Ahora te mostraré cómo hacer la configuración de Angular 4 en tu proyecto.

Comencemos la configuración angular de tres maneras diferentes:

Puede usar Angular-CLI (interfaz de línea de comandos), instalará todas las dependencias por usted.

- Puede migrar de Angular 2 a Angular 4.
- Puedes usar github y clonar el Angular4-boilerplate. (Es el más fácil.)
- Configuración angular mediante Angular-CLI (interfaz de línea de comandos).

Antes de comenzar a usar Angular-CLI, asegúrese de tener un nodo instalado en su máquina. Aquí, estoy usando el nodo v7.8.0. Ahora, abra su terminal y escriba el siguiente comando para Angular-CLI.

```
npm install –g @angular/cli
```

o

```
yarn global add @angular/cli
```

Dependiendo del gestor de paquetes que utilices.

Instalemos Angular 4 utilizando Angular-CLI.

```
ng new Angular4–boilerplate
```

cd Angular4-boilerplate Estamos listos para Angular 4. Es un método bastante fácil y directo.

Configuración angular al migrar de Angular 2 a Angular 4

Ahora vamos a ver el segundo enfoque. Le mostraré cómo migrar Angular 2 a Angular 4. Para eso necesita clonar cualquier proyecto de Angular 2 y actualizar las dependencias de Angular 2 con la dependencia de Angular 4 en su paquete.json de la siguiente manera:

```
"dependencies": {
    "@angular/animations": "^4.1.0",
    "@angular/common": "4.0.2",
    "@angular/compiler": "4.0.2",
    "@angular/core": "^4.0.1",
    "@angular/forms": "4.0.2",
    "@angular/http": "4.0.2",
    "@angular/material": "^2.0.0-beta.3",
    "@angular/platform-browser": "4.0.2",
    "@angular/platform-browser-dynamic": "4.0.2",
    "@angular/router": "4.0.2",
    "typescript": "2.2.2"
   }
```

Estas son las principales dependencias de Angular 4. Ahora puede instalar npm y luego npm comenzar a ejecutar la aplicación. Para referencia mi package.json.

Disposición angular del proyecto github

Antes de comenzar este paso, asegúrese de tener git instalado en su máquina. Abra su terminal y clone la placa de calderas angular4 usando el siguiente comando:

```
git@github.com:CypherTree/angular4-boilerplate.git
```

Luego instale todas las dependencias y ejecútelo.

```
npm install

npm start
```

Y ya ha terminado con la configuración de Angular 4. Todos los pasos son muy sencillos para que pueda optar a cualquiera de ellos.

Estructura del directorio de la placa de placa angular4

```
Angular4-boilerplate
-karma
-node_modules
-src
   -mocks
   -models
      -loginform.ts
      -index.ts
   -modules
      -app
        -app.component.ts
     -app.component.html
      -login
    -login.component.ts
    -login.component.html
    -login.component.css
      -widget
     -widget.component.ts
     -widget.component.html
     -widget.component.css
```

```
    ........
   -services
       -login.service.ts
    -rest.service.ts
   -app.routing.module.ts
   -app.module.ts
   -bootstrap.ts
   -index.html
   -vendor.ts
-typings
-webpack
-package.json
-tsconfig.json
-tslint.json
-typings.json
```

Comprensión básica de la estructura del directorio:

Todo el código reside en la carpeta src.

La carpeta de simulacros es para datos simulados que se utilizan para fines de prueba.

La carpeta modelo contiene la clase y la interfaz que se utiliza en el componente.

La carpeta de módulos contiene una lista de componentes como aplicación, inicio de sesión, widget, etc. Todos los componentes contienen archivos mecanografiados, html y css. index.ts es para exportar toda la clase.

carpeta de servicios contiene la lista de servicios utilizados en la aplicación. He separado el servicio de descanso y el servicio de diferentes componentes. En servicio de reposo contiene diferentes métodos http. El servicio de inicio de sesión funciona como mediador entre el componente de inicio de sesión y el servicio de descanso.

El archivo app.routing.ts describe todas las rutas posibles para la aplicación.

app.module.ts describe el módulo de la aplicación como componente raíz.

bootstrap.ts ejecutará toda la aplicación.

La carpeta webpack contiene el archivo de configuración webpack.

El archivo package.json es para todas las listas de dependencias.

El karma contiene la configuración del karma para la prueba de la unidad.

node_modules contiene la lista de paquetes de paquetes.

Permite comenzar con el componente de inicio de sesión. En login.component.html

```
<form>Dreamfactory - Addressbook 2.0
 <label>Email</label> <input id="email" form="" name="email" type="email" />
 <label>Password</label> <input id="password" form="" name="password"
 type="password" />
 <button form="">Login</button>
```

```
</form>
```

En login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { Form, FormGroup } from '@angular/forms';
import { LoginForm } from '../../models';
import { LoginService } from '../../services/login.service';

@Component({
    selector: 'login',
    template: require('./login.component.html'),
    styles: [require('./login.component.css')]
})
export class LoginComponent {

    constructor(private loginService: LoginService, private router: Router, form: LoginForm) {
}

    getLogin(form: LoginForm): void {
      let username = form.email;
      let password = form.password;
      this.loginService.getAuthenticate(form).subscribe(() => {
          this.router.navigate(['/calender']);
      });
    }
}
```

Necesitamos exportar este componente a index.ts.

```
export * from './login/login.component';
```

Necesitamos establecer rutas para iniciar sesión en app.routes.ts

```
const appRoutes: Routes = [
   {
       path: 'login',
       component: LoginComponent
   },
   ........
   {
       path: '',
       pathMatch: 'full',
       redirectTo: '/login'
   }
];
```

En el componente raíz, el archivo app.module.ts solo necesita importar ese componente.

```
.....
import { LoginComponent } from './modules';
......
@NgModule({
    bootstrap: [AppComponent],
    declarations: [
```

```
      LoginComponent
      .....
      .....
      ]
     .....
})
export class AppModule { }
```

y después de eso npm install y npm start. ¡Aqui tienes! Puede consultar la pantalla de inicio de sesión en su localhost. En caso de cualquier dificultad, puede referirse a la placa de caldera angular4.

Básicamente, puedo sentir que el paquete de construcción es menor y una respuesta más rápida con la aplicación Angular 4 y, aunque encontré Exactamente similar a Angular 2 en la codificación.

Lea Empezando con Angular 2 en línea: https://riptutorial.com/es/angular2/topic/789/empezando-con-angular-2

Capítulo 2: Actualizar mecanografías

Examples

Actualice las tipografías cuando: se escriban WARN en desuso

Mensaje de advertencia:

```
typings WARN deprecated 10/25/2016: "registry:dt/jasmine#2.5.0+20161003201800" is deprecated
(updated, replaced or removed)
```

Actualizar la referencia con:

```
npm run typings -- install dt~jasmine --save --global
```

Reemplace [jazmine] por cualquier biblioteca que esté lanzando advertencia

Lea Actualizar mecanografías en línea: https://riptutorial.com/es/angular2/topic/7814/actualizar-mecanografias

Capítulo 3: Agregue componentes dinámicamente usando ViewContainerRef.createComponent

Examples

Un componente contenedor que agrega componentes dinámicos declarativamente

Un componente personalizado que toma el tipo de un componente como entrada y crea una instancia de ese tipo de componente dentro de sí mismo. Cuando la entrada se actualiza, el componente dinámico agregado previamente se elimina y el nuevo se agrega en su lugar.

```
@Component({
  selector: 'dcl-wrapper',
  template: `<div #target></div>`
})
export class DclWrapper {
  @ViewChild('target', {
    read: ViewContainerRef
  }) target;
  @Input() type;
  cmpRef: ComponentRef;
  private isViewInitialized: boolean = false;

  constructor(private resolver: ComponentResolver) {}

  updateComponent() {
    if (!this.isViewInitialized) {
      return;
    }
    if (this.cmpRef) {
      this.cmpRef.destroy();
    }
    this.resolver.resolveComponent(this.type).then((factory: ComponentFactory < any > ) => {
      this.cmpRef = this.target.createComponent(factory)
        // to access the created instance use
        // this.cmpRef.instance.someProperty = 'someValue';
        // this.cmpRef.instance.someOutput.subscribe(val => doSomething());
    });
  }

  ngOnChanges() {
    this.updateComponent();
  }

  ngAfterViewInit() {
    this.isViewInitialized = true;
    this.updateComponent();
  }

  ngOnDestroy() {
```

```
    if (this.cmpRef) {
      this.cmpRef.destroy();
    }
  }
}
```

Esto le permite crear componentes dinámicos como

```
<dcl-wrapper [type]="someComponentType"></dcl-wrapper>
```

Ejemplo de plunker

Agregar componente dinámicamente en un evento específico (clic)

Archivo de componentes principales:

```
//our root app component
import {Component, NgModule, ViewChild, ViewContainerRef, ComponentFactoryResolver,
ComponentRef} from '@angular/core'
import {BrowserModule} from '@angular/platform-browser'
import {ChildComponent} from './childComp.ts'

@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>Hello {{name}}</h2>
      <input type="button" value="Click me to add element" (click) = addElement()> // call the
function on click of the button
      <div #parent> </div> // Dynamic component will be loaded here
    </div>
  `,
})
export class App {
  name:string;

  @ViewChild('parent', {read: ViewContainerRef}) target: ViewContainerRef;
  private componentRef: ComponentRef<any>;

  constructor(private componentFactoryResolver: ComponentFactoryResolver) {
    this.name = 'Angular2'
  }

  addElement(){
    let childComponent =
this.componentFactoryResolver.resolveComponentFactory(ChildComponent);
    this.componentRef = this.target.createComponent(childComponent);
  }
}
```

childComp.ts:

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

@Component({
  selector: 'child',
```

```
  template: `
    <p>This is Child</p>
  `,
})
export class ChildComponent {
  constructor(){

  }
}
```

app.module.ts:

```
@NgModule({
  imports: [ BrowserModule ],
  declarations: [ App, ChildComponent ],
  bootstrap: [ App ],
  entryComponents: [ChildComponent] // define the dynamic component here in module.ts
})
export class AppModule {}
```

Ejemplo de plunker

Matriz de componentes generada dinámicamente en la plantilla html en Angular2

Podemos crear componentes dinámicos y obtener las instancias de componentes en una matriz y, finalmente, representarlos en una plantilla.

Por ejemplo, podemos considerar dos componentes de widgets, ChartWidget y PatientWidget que extendieron la clase WidgetComponent que quería agregar en el contenedor.

ChartWidget.ts

```
@Component({
selector: 'chart-widget',
templateUrl: 'chart-widget.component.html',
providers: [{provide: WidgetComponent, useExisting: forwardRef(() => ChartWidget) }]
})

export class ChartWidget extends WidgetComponent implements OnInit {
       constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
    }
    ngOnInit() {}
     close(){
      console.log('close');
    }
    refresh(){
      console.log('refresh');
    }
    ...
}
```

chart-widget.compoment.html (usando el panel primeng)

```
<p-panel [style]="{'margin-bottom':'20px'}">
    <p-header>
        <div class="ui-helper-clearfix">
           <span class="ui-panel-title" style="font-size:14px;display:inline-block;margin-
top:2px">Chart Widget</span>
            <div class="ui-toolbar-group-right">
               <button pButton type="button" icon="fa-window-minimize"
(click)="minimize()"</button>
              <button pButton type="button" icon="fa-refresh" (click)="refresh()"></button>
              <button pButton type="button"  icon="fa-expand" (click)="expand()" ></button>
             <button pButton type="button" (click)="close()" icon="fa-window-close"></button>
                   </div>
                </div>
    </p-header>
      some data
</p-panel>
```

DataWidget.ts

```
@Component({
    selector: 'data-widget',
    templateUrl: 'data-widget.component.html',
    providers: [{provide: WidgetComponent, useExisting: forwardRef(() =>DataWidget) }]
    })

export class DataWidget extends WidgetComponent implements OnInit {
       constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
    }
    ngOnInit() {}
    close(){
      console.log('close');
    }
    refresh(){
      console.log('refresh');
    }
    ...
}
```

data-widget.compoment.html (igual que chart-widget usando primeng Panel)

WidgetComponent.ts

```
@Component({
  selector: 'widget',
  template: '<ng-content></ng-content>'
})
export  class WidgetComponent{
}
```

Podemos crear instancias de componentes dinámicos seleccionando los componentes preexistentes. Por ejemplo,

```
@Component({

    selector: 'dynamic-component',
    template: `<div #container><ng-content></ng-content></div>`
```

```
})
export class DynamicComponent {
@ViewChild('container', {read: ViewContainerRef}) container: ViewContainerRef;

    public addComponent(ngItem: Type<WidgetComponent>): WidgetComponent {
    let factory = this.compFactoryResolver.resolveComponentFactory(ngItem);
    const ref = this.container.createComponent(factory);
    const newItem: WidgetComponent = ref.instance;
    this._elements.push(newItem);
    return newItem;
  }
}
```

Finalmente lo usamos en el componente de la aplicación. app.component.ts

```
@Component({
  selector: 'app-root',
  templateUrl: './app/app.component.html',
  styleUrls: ['./app/app.component.css'],
  entryComponents: [ChartWidget,  DataWidget],
})

export class AppComponent {
   private elements: Array<WidgetComponent>=[];
   private WidgetClasses = {
       'ChartWidget': ChartWidget,
       'DataWidget': DataWidget
  }
  @ViewChild(DynamicComponent) dynamicComponent:DynamicComponent;

   addComponent(widget: string ): void{
     let ref= this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
     this.elements.push(ref);
     console.log(this.elements);

     this.dynamicComponent.resetContainer();
  }
}
```

app.component.html

```
<button (click)="addComponent('ChartWidget')">Add ChartWidget</button>
<button (click)="addComponent('DataWidget')">Add DataWidget</button>

<dynamic-component [hidden]="true" ></dynamic-component>

<hr>
Dynamic Components
<hr>
<widget *ngFor="let item of elements">
    <div>{{item}}</div>
   <div [innerHTML]="item._ngEl.nativeElement.innerHTML | sanitizeHtml">
   </div>
</widget>
```

https://plnkr.co/edit/lugU2pPsSBd3XhPHiUP1?p=preview

Alguna modificación por @yurzui para usar el evento del mouse en los widgets

view.directive.ts

importe {ViewRef, Directiva, Entrada, ViewContainerRef} desde '@ angular / core';

```
@Directive({
    selector: '[view]'
})
export class ViewDirective {
  constructor(private vcRef: ViewContainerRef) {}

  @Input()
  set view(view: ViewRef) {
    this.vcRef.clear();
    this.vcRef.insert(view);
  }

  ngOnDestroy() {
    this.vcRef.clear()
  }
}
```

app.component.ts

```
private elements: Array<{ view: ViewRef, component: WidgetComponent}> = [];

...
addComponent(widget: string ): void{
  let component = this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
  let view: ViewRef = this.dynamicComponent.container.detach(0);
  this.elements.push({view,component});

  this.dynamicComponent.resetContainer();
}
```

app.component.html

```
<widget *ngFor="let item of elements">
  <ng-container *view="item.view"></ng-container>
</widget>
```

https://plnkr.co/edit/JHpIHR43SvJd0OxJVMfV?p=preview

Lea Agregue componentes dinámicamente usando ViewContainerRef.createComponent en línea: https://riptutorial.com/es/angular2/topic/831/agregue-componentes-dinamicamente-usando-viewcontainerref-createcomponent

Capítulo 4: Angular - ForLoop

Sintaxis

1. <div * ngFor = "dejar elemento de elementos; dejar i = indexar"> {{i}} {{elemento}} </ div>

Observaciones

La directiva estructural `*ngFor` se ejecuta como un bucle en una colección y repite un fragmento de html para cada elemento de una colección.

`@View` decorator ahora está en desuso. Los desarrolladores deben usar las propiedades de `template` o 'templateUrl' para `@Component` decorator.

Examples

Angular 2 for-loop

Para hacer clic plnkr en vivo ...

```
<!doctype html>
<html>
<head>
    <title>ng for loop in angular 2 with ES5.</title>
    <script type="text/javascript" src="https://code.angularjs.org/2.0.0-
alpha.28/angular2.sfx.dev.js"></script>
    <script>
        var ngForLoop = function () {
            this.msg = "ng for loop in angular 2 with ES5.";
            this.users = ["Anil Singh", "Sunil Singh", "Sushil Singh", "Aradhya", 'Reena'];
        };

        ngForLoop.annotations = [
                new angular.Component({
                    selector: 'ngforloop'
                }),
                new angular.View({
                    template: '<H1>{{msg}}</H1>' +
                            '<p> User List : </p>' +
                            '<ul>' +
                            '<li *ng-for="let user of users">' +
                            '{{user}}' +
                            '</li>' +
                            '</ul>',
                    directives: [angular.NgFor]
                })
        ];

        document.addEventListener("DOMContentLoaded", function () {
            angular.bootstrap(ngForLoop);
        });
    </script>
```

```
</head>
<body>
    <ngforloop></ngforloop>
    <h2>
      <a href="http://www.code-sample.com/" target="_blank">For more detail...</a>
    </h2>
</body>
</html>
```

NgFor - Markup For Loop

La directiva **NgFor crea** una instancia de una plantilla por elemento desde un iterable. El contexto para cada plantilla instanciada se hereda del contexto externo con la variable de bucle dada establecida en el elemento actual del iterable.

Para personalizar el algoritmo de seguimiento predeterminado, NgFor admite la opción **trackBy** . **trackBy** toma una función que tiene dos argumentos: índice y elemento. Si se proporciona **trackBy** , las pistas angulares cambian según el valor de retorno de la función.

```
<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
    {{i}} - {{item.name}}
</li>
```

Opciones adicionales : NgFor proporciona varios valores exportados que pueden ser asignados a las variables locales:

- **El índice** se establecerá en la iteración del bucle actual para cada contexto de plantilla.
- **primero** se establecerá en un valor booleano que indica si el elemento es el primero en la iteración.
- **el último** se establecerá en un valor booleano que indica si el elemento es el último en la iteración.
- **incluso** se establecerá en un valor booleano que indica si este elemento tiene un índice par.
- **odd** se establecerá en un valor booleano que indica si este elemento tiene un índice impar.

* ngFor en las filas de la tabla

```
<table>
    <thead>
        <th>Name</th>
        <th>Index</th>
    </thead>
    <tbody>
        <tr *ngFor="let hero of heroes">
            <td>{{hero.name}}</td>
        </tr>
    </tbody>
</table>
```

* ngFor con componente

```
   @Component({
```

```
    selector: 'main-component',
    template: '<example-component
                  *ngFor="let hero of heroes"
                  [hero]="hero"></example-component>'
  })


  @Component({
     selector: 'example-component',
     template: '<div>{{hero?.name}}</div>'
  })

  export class ExampleComponent {
    @Input() hero : Hero = null;
  }
```

* ngPara X cantidad de artículos por fila

El ejemplo muestra 5 artículos por fila:

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 5 == 0" class="row">
    {{ item }}
    <div *ngIf="i + 1 < items.length">{{ items[i + 1] }}</div>
    <div *ngIf="i + 2 < items.length">{{ items[i + 2] }}</div>
    <div *ngIf="i + 3 < items.length">{{ items[i + 3] }}</div>
    <div *ngIf="i + 4 < items.length">{{ items[i + 4] }}</div>
  </div>
</div>
```

Lea Angular - ForLoop en línea: https://riptutorial.com/es/angular2/topic/6543/angular---forloop

Capítulo 5: Angular 2 - Transportador

Examples

Prueba de enrutamiento de Navbar con transportador

Primero creamos navbar.html básico con 3 opciones. (Inicio, Lista, Crear)

```
<nav class="navbar navbar-default" role="navigation">
<ul class="nav navbar-nav">
  <li>
    <a id="home-navbar" routerLink="/home">Home</a>
  </li>
  <li>
    <a id="list-navbar" routerLink="/create" >List</a>
  </li>
  <li>
    <a id="create-navbar" routerLink="/create">Create</a>
  </li>
</ul>
```

segundo vamos a crear navbar.e2e-spec.ts

```
describe('Navbar', () => {

  beforeEach(() => {
    browser.get('home'); // before each test navigate to home page.
  });

  it('testing Navbar', () => {
    browser.sleep(2000).then(function(){
      checkNavbarTexts();
      navigateToListPage();
    });
  });

  function checkNavbarTexts(){
    element(by.id('home-navbar')).getText().then(function(text){ // Promise
      expect(text).toEqual('Home');
    });

    element(by.id('list-navbar')).getText().then(function(text){ // Promise
      expect(text).toEqual('List');
    });

    element(by.id('create-navbar')).getText().then(function(text){ // Promise
      expect(text).toEqual('Create');
    });
  }

  function navigateToListPage(){
    element(by.id('list-home')).click().then(function(){ // first find list-home a tag and
than click
        browser.sleep(2000).then(function(){
          browser.getCurrentUrl().then(function(actualUrl){ // promise
```

```
            expect(actualUrl.indexOf('list') !== -1).toBeTruthy(); // check the current url is
list
          });
        });

    });
  }
});
```

Transportador Angular2 - Instalación

ejecuta los siguientes comandos en cmd

- `npm install -g protractor`
- `webdriver-manager update`
- `webdriver-manager start`

** Cree el archivo protractor.conf.js en la raíz principal de la aplicación.

muy importante decleare useAllAngular2AppRoots: true

```
  const config = {
  baseUrl: 'http://localhost:3000/',

  specs: [
      './dev/**/*.e2e-spec.js'
  ],

  exclude: [],
  framework: 'jasmine',

  jasmineNodeOpts: {
    showColors: true,
    isVerbose: false,
    includeStackTrace: false
  },

  directConnect: true,

  capabilities: {
    browserName: 'chrome',
    shardTestFiles: false,
    chromeOptions: {
      'args': ['--disable-web-security ','--no-sandbox', 'disable-extensions', 'start-
maximized', 'enable-crash-reporter-for-testing']
    }
  },

  onPrepare: function() {
    const SpecReporter = require('jasmine-spec-reporter');
    // add jasmine spec reporter
    jasmine.getEnv().addReporter(new SpecReporter({ displayStacktrace: true }));

    browser.ignoreSynchronization = false;
  },
  useAllAngular2AppRoots: true
};
```

```
if (process.env.TRAVIS) {
  config.capabilities = {
    browserName: 'firefox'
  };
}

exports.config = config;
```

Crear una prueba básica en el directorio dev.

```
describe('basic test', () => {

  beforeEach(() => {
    browser.get('http://google.com');
  });

  it('testing basic test', () => {
    browser.sleep(2000).then(function(){
      browser.getCurrentUrl().then(function(actualUrl){
        expect(actualUrl.indexOf('google') !== -1).toBeTruthy();
      });
    });
  });
});
```

corre en cmd

```
protractor conf.js
```

Lea Angular 2 - Transportador en línea: https://riptutorial.com/es/angular2/topic/8900/angular-2---transportador

Capítulo 6: Angular 2 Cambio de detección y activación manual.

Examples

Ejemplo basico

Componente principal:

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

@Component({
  selector: 'parent-component',
  templateUrl: './parent-component.html'
})
export class ParentComponent {
  users : Array<User> = [];
  changeUsersActivation(user : User){
    user.changeButtonState();
  }
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto',false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}


export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }

}
```

HTML principal:

```
<div>
  <child-component [usersDetails]="users"
                   (changeUsersActivation)="changeUsersActivation($event)">
   </child-component>
</div>
```

componente hijo:

```
import {Component, Input, EventEmitter, Output} from '@angular/core';
import {User} from "./parent.component";

@Component({
  selector: 'child-component',
  templateUrl: './child-component.html',
  styles: [`
     .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
     }

  `]
})
export class ChildComponent{
  @Input() usersDetails : Array<User> = null;
  @Output() changeUsersActivation = new EventEmitter();

  triggerEvent(user : User){
    this.changeUsersActivation.emit(user);
  }
}
```

HTML hijo:

```
<div>
  <div>
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th></th>
        </tr>
      </thead>
      <tbody *ngIf="user !== null">
        <tr *ngFor="let user of usersDetails">
          <td>{{user.firstName}}</td>
          <td><button class="btn" (click)="triggerEvent(user)">{{user.active}}</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Lea Angular 2 Cambio de detección y activación manual. en línea:
https://riptutorial.com/es/angular2/topic/8971/angular-2-cambio-de-deteccion-y-activacion-manual-

Capítulo 7: Angular 2 Data Driven Forms

Observaciones

`this.myForm = this.formBuilder.group`

crea un objeto de formulario con la configuración del usuario y lo asigna a la variable this.myForm.

`'loginCredentials': this.formBuilder.group`

método crea un grupo de controles que consisten en un **nombre** de control de **formulario,** por ejemplo. `login` y valor `['', Validators.required],` donde el primer parámetro es el valor inicial de la entrada del formulario y secons es un validador o una matriz de validadores como en `'email': ['', [Validators.required, customValidator]],`

```
'hobbies': this.formBuilder.array
```

Crea una matriz de grupos donde el índice del grupo es **formGroupName** en la matriz y se accede a él como:

```
<div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
   <div formGroupName="{{i}}">...</div>
</div>
```

```
onAddHobby() {
    (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
      'hobby': new FormControl('', Validators.required)
    }))
  }
```

este método de muestra agrega nuevo formGroup a la matriz. El acceso actual requiere especificar el tipo de control al que queremos acceder, en este ejemplo, este tipo es: `<FormArray>`

```
removeHobby(index: number){
    (<FormArray>this.myForm.find('hobbies')).removeAt(index);
  }
```

Se aplican las mismas reglas que las anteriores para eliminar un control de formulario específico de la matriz

Examples

Formulario de datos

Componente

```
    import {Component, OnInit} from '@angular/core';
import {
    FormGroup,
    FormControl,
    FORM_DIRECTIVES,
    REACTIVE_FORM_DIRECTIVES,
    Validators,
    FormBuilder,
    FormArray
} from "@angular/forms";
import {Control} from "@angular/common";

@Component({
  moduleId: module.id,
  selector: 'app-data-driven-form',
  templateUrl: 'data-driven-form.component.html',
  styleUrls: ['data-driven-form.component.css'],
  directives: [FORM_DIRECTIVES, REACTIVE_FORM_DIRECTIVES]
})
export class DataDrivenFormComponent  implements OnInit {
  myForm: FormGroup;

  constructor(private formBuilder: FormBuilder) {}

  ngOnInit() {
    this.myForm = this.formBuilder.group({
      'loginCredentials': this.formBuilder.group({
        'login': ['', Validators.required],
        'email': ['',  [Validators.required, customValidator]],
        'password': ['',  Validators.required]
      }),
      'hobbies': this.formBuilder.array([
        this.formBuilder.group({
          'hobby': ['', Validators.required]
        })
      ])
    });
  }

  removeHobby(index: number){
    (<FormArray>this.myForm.find('hobbies')).removeAt(index);
  }

  onAddHobby() {
    (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
      'hobby': new FormControl('', Validators.required)
    }))
  }

  onSubmit() {
    console.log(this.myForm.value);
  }
}

function customValidator(control: Control): {[s: string]: boolean} {
  if(!control.value.match("[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-
]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?")) {
    return {error: true}
  }
}
```

Marcado HTML

```
<h3>Register page</h3>
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <div formGroupName="loginCredentials">
    <div class="form-group">
      <div>
        <label for="login">Login</label>
        <input  id="login" type="text" class="form-control" formControlName="login">
      </div>
      <div>
        <label for="email">Email</label>
        <input  id="email" type="text" class="form-control"  formControlName="email">
      </div>
      <div>
        <label for="password">Password</label>
        <input  id="password" type="text" class="form-control"  formControlName="password">
      </div>
    </div>
  </div>
  <div class="row" >
    <div  formGroupName="hobbies">
      <div class="form-group">
        <label>Hobbies array:</label>
        <div  *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
          <div formGroupName="{{i}}">
            <input id="hobby_{{i}}" type="text" class="form-control"  formControlName="hobby">
            <button *ngIf="myForm.find('hobbies').length > 1"
(click)="removeHobby(i)">x</button>
          </div>
        </div>
        <button (click)="onAddHobby()">Add hobby</button>
      </div>
    </div>
  </div>
  <button type="submit" [disabled]="!myForm.valid">Submit</button>
</form>
```

Lea Angular 2 Data Driven Forms en línea: https://riptutorial.com/es/angular2/topic/6463/angular-2-data-driven-forms

Capítulo 8: Angular 2 Formas de actualización

Observaciones

Angular 2 permite acceder a la instancia de ngForm creando una variable de plantilla local. Angular 2 expone instancias de directivas como ngForm especificando la propiedad exportAs de los metadatos de la directiva. Ahora, la ventaja aquí es que, sin mucha codificación, puede acceder a la instancia de ngForm y usarla para acceder a los valores enviados o para verificar si todos los campos son válidos usando propiedades (válido, enviado, valor, etc.).

```
#f = ngForm (creates local template instance "f")
```

ngForm emite el evento "ngSubmit" cuando se envía (consulte la documentación de @Output para obtener más detalles sobre el emisor del evento)

```
(ngSubmit)= "login(f.value,f.submitted)"
```

"ngModel" crea un control de formulario en combinación con el atributo "nombre" de entrada.

```
<input type="text" [(ngModel)]="username" placeholder="enter username" required>
```

Cuando se envía el formulario, f.value tiene el objeto JSON que representa los valores enviados.

{nombre de usuario: 'Sachin', contraseña: 'Bienvenido1'}

Examples

Formulario de cambio de contraseña simple con validación de control múltiple

Los siguientes ejemplos utilizan el nuevo formulario API introducido en RC3.

pw-change.template.html

```
<form class="container" [formGroup]="pwChangeForm">
    <label for="current">Current Password</label>
    <input id="current" formControlName="current" type="password" required><br />

    <label for="newPW">New Password</label>
    <input id="newPW" formControlName="newPW" type="password" required><br/>
    <div *ngIf="newPW.touched && newPW.newIsNotOld">
        New password can't be the same as current password.
    </div>
```

```
    <label for="confirm">Confirm new password</label>
    <input id="confirm" formControlName="confirm" type="password" required><br />
    <div *ngIf="confirm.touched && confirm.errors.newMatchesConfirm">
        The confirmation does not match.
    </div>

    <button type="submit">Submit</button>
</form>
```

pw-change.component.ts

```
import {Component} from '@angular/core'
import {REACTIVE_FORM_DIRECTIVES, FormBuilder, AbstractControl, FormGroup,
    Validators} from '@angular/forms'
import {PWChangeValidators} from './pw-validators'

@Component({
    moduleId: module.id
    selector: 'pw-change-form',
    templateUrl: `./pw-change.template.html`,
    directives: [REACTIVE_FORM_DIRECTIVES]
})

export class PWChangeFormComponent {
    pwChangeForm: FormGroup;

    // Properties that store paths to FormControls makes our template less verbose
    current: AbstractControl;
    newPW: AbstractControl;
    confirm: AbstractControl;

    constructor(private fb: FormBuilder) { }
    ngOnInit() {
        this.pwChangeForm = this.fb.group({
            current: ['', Validators.required],
            newPW: ['', Validators.required],
            confirm: ['', Validators.required]
        }, {
            // Here we create validators to be used for the group as a whole
            validator: Validators.compose([
                PWChangeValidators.newIsNotOld,
                PWChangeValidators.newMatchesConfirm
            ])
        );
        this.current = this.pwChangeForm.controls['current'];
        this.newPW = this.pwChangeForm.controls['newPW'];
        this.confirm = this.pwChangeForm.controls['confirm'];
    }
}
```

pw-validators.ts

```
import {FormControl, FormGroup} from '@angular/forms'
export class PWChangeValidators {

    static OldPasswordMustBeCorrect(control: FormControl) {
```

```
        var invalid = false;
        if (control.value != PWChangeValidators.oldPW)
            return { oldPasswordMustBeCorrect: true }
        return null;
    }

    // Our cross control validators are below
    // NOTE: They take in type FormGroup rather than FormControl
    static newIsNotOld(group: FormGroup){
        var newPW = group.controls['newPW'];
        if(group.controls['current'].value == newPW.value)
            newPW.setErrors({ newIsNotOld: true });
        return null;
    }

    static newMatchesConfirm(group: FormGroup){
        var confirm = group.controls['confirm'];
        if(group.controls['newPW'].value !== confirm.value)
            confirm.setErrors({ newMatchesConfirm: true });
        return null;
    }
}
```

Una esencia que incluye algunas clases de bootstrap se puede encontrar aquí .

Angular 2: Formas impulsadas por plantillas

```
import { Component } from '@angular/core';
import { Router , ROUTER_DIRECTIVES} from '@angular/router';
import { NgForm }    from '@angular/forms';

@Component({
    selector: 'login',
    template: `
<h2>Login</h2>
<form #f="ngForm" (ngSubmit)="login(f.value,f.valid)" novalidate>
    <div>
        <label>Username</label>
        <input type="text" [(ngModel)]="username" placeholder="enter username" required>
    </div>
    <div>
        <label>Password</label>
        <input type="password" name="password" [(ngModel)]="password" placeholder="enter
password" required>
    </div>
    <input class="btn-primary" type="submit" value="Login">
</form>`
   //For long form we can use **templateUrl** instead of template
})

export class LoginComponent{

    constructor(private router : Router){ }

    login (formValue: any, valid: boolean){
        console.log(formValue);

        if(valid){
            console.log(valid);
```

```
        }
    }
}
```

Formulario Angular 2 - Correo electrónico personalizado / validación de contraseña

Para la demostración en vivo haga clic en ..

Índice de aplicaciones ts

```
import {bootstrap} from '@angular/platform-browser-dynamic';
import {MyForm} from './my-form.component.ts';

bootstrap(MyForm);
```

Validador personalizado

```
import {Control} from @'angular/common';

export class CustomValidators {
  static emailFormat(control: Control): [[key: string]: boolean] {
    let pattern:RegExp = /\S+@\S+\.\S+/;
    return pattern.test(control.value) ? null : {"emailFormat": true};
  }
}
```

Componentes de la forma ts

```
import {Component} from '@angular/core';
import {FORM_DIRECTIVES, NgForm, FormBuilder, Control, ControlGroup, Validators} from
'@angular/common';
import {CustomValidators} from './custom-validators';

@Component({
  selector: 'my-form',
  templateUrl: 'app/my-form.component.html',
  directives: [FORM_DIRECTIVES],
  styleUrls: ['styles.css']
})
export class MyForm {
  email: Control;
  password: Control;
  group: ControlGroup;

  constructor(builder: FormBuilder) {
    this.email = new Control('',
      Validators.compose([Validators.required, CustomValidators.emailFormat])
    );

    this.password = new Control('',
      Validators.compose([Validators.required, Validators.minLength(4)])
    );

    this.group = builder.group({
```

```
      email: this.email,
      password: this.password
    });
  }

  onSubmit() {
    console.log(this.group.value);
  }
}
```

HTML de componentes de formulario

```
<form [ngFormModel]="group" (ngSubmit)="onSubmit()" novalidate>

  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" [ngFormControl]="email">

    <ul *ngIf="email.dirty && !email.valid">
      <li *ngIf="email.hasError('required')">An email is required</li>
    </ul>
  </div>

  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" [ngFormControl]="password">

    <ul *ngIf="password.dirty && !password.valid">
      <li *ngIf="password.hasError('required')">A password is required</li>
      <li *ngIf="password.hasError('minlength')">A password needs to have at least 4
characterss</li>
    </ul>
  </div>

  <button type="submit">Register</button>

</form>
```

Angular 2: Formas reactivas (también conocidas como formas dirigidas por modelos)

Este ejemplo usa Angular 2.0.0 Final Release

registration-form.component.ts

```
import { FormGroup,
    FormControl,
    FormBuilder,
    Validators }              from '@angular/forms';

@Component({
    templateUrl: "./registration-form.html"
})
export class ExampleComponent {
    constructor(private _fb: FormBuilder) { }
```

```
exampleForm = this._fb.group({
    name: ['DefaultValue', [<any>Validators.required, <any>Validators.minLength(2)]],
    email: ['default@defa.ult', [<any>Validators.required, <any>Validators.minLength(2)]]
})
```

registration-form.html

```
<form [formGroup]="exampleForm" novalidate (ngSubmit)="submit(exampleForm)">
    <label>Name: </label>
    <input type="text" formControlName="name"/>
    <label>Email: </label>
    <input type="email" formControlName="email"/>
    <button type="submit">Submit</button>
</form>
```

Formularios Angular 2 (Formularios reactivos) con formulario de registro y confirmación de validación de contraseña

app.module.ts

Agregue estos en su archivo app.module.ts para usar formularios reactivos

```
import { NgModule } from '@angular/core';
    import { BrowserModule } from '@angular/platform-browser';
    import { FormsModule, ReactiveFormsModule } from '@angular/forms';
    import { AppComponent } from './app.component';
    @NgModule({
      imports: [
        BrowserModule,
        FormsModule,
        ReactiveFormsModule,
     ],
    declarations: [
    AppComponent
     ]
  providers: [],
  bootstrap: [
    AppComponent
   ]
  })
export class AppModule {}
```

app.component.ts

```
import { Component,OnInit } from '@angular/core';
import template from './add.component.html';
import { FormGroup,FormBuilder,Validators } from '@angular/forms';
import { matchingPasswords } from './validators';
@Component({
    selector: 'app',
    template
})
```

```
export class AppComponent implements OnInit {
    addForm: FormGroup;
    constructor(private formBuilder: FormBuilder) {
    }
    ngOnInit() {

    this.addForm = this.formBuilder.group({
            username: ['', Validators.required],
            email: ['', Validators.required],
            role: ['', Validators.required],
            password: ['', Validators.required],
            password2: ['', Validators.required] },
          { validator: matchingPasswords('password', 'password2')
        })
    };

addUser() {
        if (this.addForm.valid) {
            var adduser = {
                username: this.addForm.controls['username'].value,
                email: this.addForm.controls['email'].value,
                password: this.addForm.controls['password'].value,
                profile: {
                    role: this.addForm.controls['role'].value,
                    name: this.addForm.controls['username'].value,
                    email: this.addForm.controls['email'].value
                }
            };
          console.log(adduser);// adduser var contains all our form values. store it where you
want
            this.addForm.reset();// this will reset our form values to null
        }
    }
}
```

app.component.html

```
<div>
  <form [formGroup]="addForm">
   <input type="text" placeholder="Enter username" formControlName="username" />
   <input type="text" placeholder="Enter Email Address" formControlName="email"/>
   <input type="password" placeholder="Enter Password" formControlName="password" />
   <input type="password" placeholder="Confirm Password" name="password2"
formControlName="password2"/>
   <div class='error' *ngIf="addForm.controls.password2.touched">
    <div class="alert-danger errormessageadduser"
*ngIf="addForm.hasError('mismatchedPasswords')">                                    Passwords do
not match
  </div>
</div>
<select name="Role" formControlName="role">
    <option value="admin" >Admin</option>
    <option value="Accounts">Accounts</option>
    <option value="guest">Guest</option>
</select>
<br/>
<br/>
<button type="submit" (click)="addUser()"><span><i class="fa fa-user-plus" aria-
hidden="true"></i></span> Add User </button>
```

```
</form>
</div>
```

validadores.ts

```
export function matchingPasswords(passwordKey: string, confirmPasswordKey: string) {
    return (group: ControlGroup): {
        [key: string]: any
    } => {
        let password = group.controls[passwordKey];
        let confirmPassword = group.controls[confirmPasswordKey];

        if (password.value !== confirmPassword.value) {
            return {
                mismatchedPasswords: true
            };
        }
    }
}
```

Angular2 - Form Builder

FormComponent.ts

```
import {Component} from "@angular/core";
import {FormBuilder} from "@angular/forms";

@Component({
 selector: 'app-form',
 templateUrl: './form.component.html',
 styleUrls: ['./form.component.scss'],
 providers : [FormBuilder]
})

export class FormComponent{
  form : FormGroup;
  emailRegex = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;

  constructor(fb: FormBuilder) {

    this.form = fb.group({
      FirstName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      LastName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      Email : new FormControl({value: null}, Validators.compose([
        Validators.required,
        Validators.maxLength(15),
        Validators.pattern(this.emailRegex)]))
    });
  }
}
```

form.component.html

```
<form class="form-details" role="form" [formGroup]="form">
    <div class="row input-label">
      <label class="form-label" for="FirstName">First name</label>
      <input
        [formControl]="form.controls['FirstName']"
        type="text"
        class="form-control"
        id="FirstName"
        name="FirstName">
    </div>
    <div class="row input-label">
      <label class="form-label" for="LastName">Last name</label>
      <input
        [formControl]="form.controls['LastName']"
        type="text"
        class="form-control"
        id="LastName"
        name="LastName">
    </div>
    <div class="row">
      <label class="form-label" for="Email">Email</label>
      <input
        [formControl]="form.controls['Email']"
        type="email"
        class="form-control"
        id="Email"
        name="Email">
    </div>
    <div class="row">
      <button
        (click)="submit()"
        role="button"
        class="btn btn-primary submit-btn"
        type="button"
        [disabled]="!form.valid">Submit</button>
    </div>
  </div>
</form>
```

Lea Angular 2 Formas de actualización en línea:
https://riptutorial.com/es/angular2/topic/4607/angular-2-formas-de-actualizacion

Capítulo 9: Angular2 animaciones

Introducción

El sistema de animación de Angular le permite crear animaciones que se ejecutan con el mismo tipo de rendimiento nativo que las animaciones de CSS puro. También puede integrar estrechamente su lógica de animación con el resto del código de su aplicación, para facilitar el control.

Examples

Animación básica: hace la transición de un elemento entre dos estados controlados por un atributo de modelo.

app.component.html

```
<div>
  <div>
    <div *ngFor="let user of users">
      <button
        class="btn"
        [@buttonState]="user.active"
        (click)="user.changeButtonState()">{{user.firstName}}</button>
    </div>
  </div>
</div>
```

app.component.ts

```
import {Component, trigger, state, transition, animate, style} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styles: [`
     .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
     }

  `],
  animations: [
    trigger('buttonState', [
      state('true', style({
        background: '#04b104',
        transform: 'scale(1)'
      })),
```

```
      state('false',   style({
        background: '#e40202',
        transform: 'scale(1.1)'
      })),
      transition('true => false', animate('100ms ease-in')),
      transition('false => true', animate('100ms ease-out'))
    ])
  ]
})
export class AppComponent {
  users : Array<User> = [];
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto',false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}


export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }

}
```

Lea Angular2 animaciones en línea: https://riptutorial.com/es/angular2/topic/8970/angular2-animaciones

Capítulo 10: Angular2 CanActivate

Examples

Angular2 CanActivate

Implementado en un enrutador:

```
export const MainRoutes: Route[] = [{
   path: '',
   children: [ {
      path: 'main',
      component: MainComponent ,
      canActivate : [CanActivateRoute]
   }]
}];
```

El archivo `canActivateRoute` :

```
@Injectable()
  export class  CanActivateRoute implements CanActivate{
  constructor(){}
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
     return true;
  }
}
```

Lea Angular2 CanActivate en línea: https://riptutorial.com/es/angular2/topic/8899/angular2-canactivate

Capítulo 11: Angular2 en la memoria web de la API

Observaciones

Solicité principalmente este tema porque no pude encontrar ninguna información sobre la configuración de múltiples rutas API con Angular2-In-Memory-Web-Api. Terminé resolviéndolo yo mismo, y pensé que esto podría ser útil para otros.

Examples

Configuración básica

mock-data.ts

Crear los datos api simulados

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' },
      { id: '4', name: 'Object D' }
    ];

    return {mock};
  }
}
```

main.ts

Haga que el inyector de dependencia proporcione las solicitudes InMemoryBackendService para XHRBackend. Además, proporcionar una clase que incluye una

```
createDb()
```

Función (en este caso, MockData) que especifica las rutas de API simuladas para las solicitudes SEED_DATA.

```
import { XHRBackend, HTTP_PROVIDERS } from '@angular/http';
import { InMemoryBackendService, SEED_DATA } from 'angular2-in-memory-web-api';
import { MockData } from './mock-data';
import { bootstrap } from '@angular/platform-browser-dynamic';

import { AppComponent } from './app.component';

bootstrap(AppComponent, [
```

```
    HTTP_PROVIDERS,
    { provide: XHRBackend, useClass: InMemoryBackendService },
    { provide: SEED_DATA,  useClass: MockData }
]);
```

mock.service.ts

Ejemplo de llamada a una solicitud de obtención para la ruta API creada

```
import { Injectable }     from '@angular/core';
import { Http, Response } from '@angular/http';
import { Mock } from './mock';

@Injectable()
export class MockService {
  // URL to web api
  private mockUrl = 'app/mock';

  constructor (private http: Http) {}

  getData(): Promise<Mock[]> {
    return this.http.get(this.mockUrl)
                    .toPromise()
                    .then(this.extractData)
                    .catch(this.handleError);
  }

  private extractData(res: Response) {
    let body = res.json();
    return body.data || { };
  }

  private handleError (error: any) {
    let errMsg = (error.message) ? error.message :
      error.status ? `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Promise.reject(errMsg);
  }
}
```

Configuración de múltiples rutas API de prueba

mock-data.ts

```
export class MockData {
    createDb() {
        let mock = [
            { id: '1', name: 'Object A' },
            { id: '2', name: 'Object B' },
            { id: '3', name: 'Object C' }
        ];

        let data = [
            { id: '1', name: 'Data A' },
            { id: '2', name: 'Data B' },
            { id: '3', name: 'Data C' }
        ];
```

```
        return { mock, data };
    }
}
```

Ahora, puedes interactuar con ambos

```
app/mock
```

y

```
app/data
```

Para extraer sus datos correspondientes.

Lea Angular2 en la memoria web de la API en línea:
https://riptutorial.com/es/angular2/topic/6576/angular2-en-la-memoria-web-de-la-api

Capítulo 12: Angular2 Entrada () salida ()

Examples

Entrada()

Componente principal: Inicializar listas de usuarios.

```
@Component({
  selector: 'parent-component',
  template: '<div>
               <child-component [users]="users"></child-component>
             </div>'
})
export class ParentComponent implements OnInit{
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com');
    users.push(new User('B', 'B', 'B@gmail.com');
    users.push(new User('C', 'C', 'C@gmail.com');
  }
}
```

El componente hijo obtiene al usuario del componente principal con Input ()

```
@Component({
selector: 'child-component',
  template: '<div>
                  <table *ngIf="users !== null">
                    <thead>
                         <th>Name</th>
                         <th>FName</th>
                         <th>Email</th>
                    </thead>
                    <tbody>
                        <tr *ngFor="let user of users">
                            <td>{{user.name}}</td>
                            <td>{{user.fname}}</td>
                            <td>{{user.email}}</td>
                        </tr>
                    </tbody>
                  </table>

             </div>',
})
export class ChildComponent {
  @Input() users : List<User> = null;
}


export class User {
  name : string;
```

```
  fname : string;
  email : string;

  constructor(_name : string, _fname : string, _email : string){
     this.name = _name;
     this.fname = _fname;
     this.email = _email;
  }
}
```

Ejemplo simple de propiedades de entrada

Elemento principal html

```
<child-component [isSelected]="inputPropValue"></child-component>
```

Elemento padre ts

```
export class AppComponent {
     inputPropValue: true
}
```

Componente hijo ts:

```
export class ChildComponent {
    @Input() inputPropValue = false;
}
```

Componente hijo html:

```
<div [class.simpleCssClass]="inputPropValue"></div>
```

Este código enviará el inputPropValue desde el componente principal al hijo y tendrá el valor que hemos establecido en el componente principal cuando llegue allí, falso en nuestro caso. Luego podemos usar ese valor en el componente secundario para, por ejemplo, agregar una clase a un elemento.

Lea Angular2 Entrada () salida () en línea: https://riptutorial.com/es/angular2/topic/8943/angular2-entrada----salida---

Capítulo 13: Angular2 proporciona datos externos a la aplicación antes de bootstrap

Introducción

En esta publicación, demostraré cómo pasar datos externos a la aplicación Angular antes de que la aplicación se inicie. Estos datos externos pueden ser datos de configuración, datos heredados, servidores renderizados, etc.

Examples

Vía de inyección de dependencia

En lugar de invocar directamente el código de inicio del Angular, envuelva el código de inicio en una función y exporte la función. Esta función también puede aceptar parámetros.

```
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { AppModule } from "./src/app";
export function runAngular2App(legacyModel: any) {
       platformBrowserDynamic([
           { provide: "legacyModel", useValue: model }
       ]).bootstrapModule(AppModule)
       .then(success => console.log("Ng2 Bootstrap success"))
       .catch(err => console.error(err));
}
```

Luego, en cualquier servicio o componente podemos inyectar el "modelo heredado" y obtener acceso a él.

```
import { Injectable } from "@angular/core";
@Injectable()
export class MyService {
   constructor(@Inject("legacyModel") private legacyModel) {
      console.log("Legacy data – ", legacyModel);
    }
 }
```

Requerir la aplicación y luego ejecutarlo.

```
require(["myAngular2App"], function(app) {
   app.runAngular2App(legacyModel); // Input to your APP
});
```

Lea Angular2 proporciona datos externos a la aplicación antes de bootstrap en línea: https://riptutorial.com/es/angular2/topic/9203/angular2-proporciona-datos-externos-a-la-aplicacion-antes-de-bootstrap

Capítulo 14: Angular2 utilizando webpack

Examples

Configuración de webpack angular 2

webpack.config.js

```
const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")
const WebpackNotifierPlugin = require('webpack-notifier');

module.exports = {

    // set entry point for your app module
    "entry": {
        "app": helpers.root("app/main.module.ts"),
    },

    // output files to dist folder
    "output": {
        "filename": "[name].js",
        "path": helpers.root("dist"),
        "publicPath": "/",
    },

    "resolve": {
        "extensions": ['.ts', '.js'],
    },

    "module": {
        "rules": [
            {
                "test": /\.ts$/,
                "loaders": [
                    {
                        "loader": 'awesome-typescript-loader',
                        "options": {
                            "configFileName": helpers.root("./tsconfig.json")
                        }
                    },
                    "angular2-template-loader"
                ]
            },
        ],
    },

    "plugins": [

        // notify when build is complete
        new WebpackNotifierPlugin({title: "build complete"}),

        // get reference for shims
        new webpack.DllReferencePlugin({
            "context": helpers.root("src/app"),
```

```
            "manifest": helpers.root("config/polyfills-manifest.json")
        }),

        // get reference of vendor DLL
        new webpack.DllReferencePlugin({
            "context": helpers.root("src/app"),
            "manifest": helpers.root("config/vendor-manifest.json")
        }),

        // minify compiled js
        new webpack.optimize.UglifyJsPlugin(),
    ],
}
```

vendor.config.js

```
const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")

module.exports = {
    // specify vendor file where all vendors are imported
    "entry": {
        // optionally add your shims as well
        "polyfills": [helpers.root("src/app/shims.ts")],
        "vendor": [helpers.root("src/app/vendor.ts")],
    },

    // output vendor to dist
    "output": {
        "filename": "[name].js",
        "path": helpers.root("dist"),
        "publicPath": "/",
        "library": "[name]"
    },

    "resolve": {
        "extensions": ['.ts', '.js'],
    },

    "module": {
        "rules": [
            {
                "test": /\.ts$/,
                "loaders": [
                    {
                        "loader": 'awesome-typescript-loader',
                        "options": {
                            "configFileName": helpers.root("./tsconfig.json")
                        }
                    },
                ]
            },
        ],
    },

    "plugins": [

        // create DLL for entries
        new webpack.DllPlugin({
```

```
            "name": "[name]",
            "context": helpers.root("src/app"),
            "path": helpers.root("config/[name]-manifest.json")
        }),

        // minify generated js
        new webpack.optimize.UglifyJsPlugin(),
    ],
}
```

helpers.js

```
var path = require('path');

var _root = path.resolve(__dirname, '..');

function root(args) {
  args = Array.prototype.slice.call(arguments, 0);
  return path.join.apply(path, [_root].concat(args));
}

exports.root = root;
```

vendor.ts

```
import "@angular/platform-browser"
import "@angular/platform-browser-dynamic"
import "@angular/core"
import "@angular/common"
import "@angular/http"
import "@angular/router"
import "@angular/forms"
import "rxjs"
```

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Angular 2 webpack</title>

    <script src="/dist/vendor.js" type="text/javascript"></script>
    <script src="/dist/app.js" type="text/javascript"></script>
</head>
<body>
    <app>loading...</app>
</body>
</html>
```

paquete.json

```
{
  "name": "webpack example",
  "version": "0.0.0",
  "description": "webpack",
  "scripts": {
```

```
    "build:webpack": "webpack --config config/webpack.config.js",
    "build:vendor": "webpack --config config/vendor.config.js",
    "watch": "webpack --config config/webpack.config.js --watch"
  },
  "devDependencies": {
    "@angular/common": "2.4.7",
    "@angular/compiler": "2.4.7",
    "@angular/core": "2.4.7",
    "@angular/forms": "2.4.7",
    "@angular/http": "2.4.7",
    "@angular/platform-browser": "2.4.7",
    "@angular/platform-browser-dynamic": "2.4.7",
    "@angular/router": "3.4.7",
    "webpack": "^2.2.1",
    "awesome-typescript-loader": "^3.1.2",
  },
  "dependencies": {
  }
}
```

Lea Angular2 utilizando webpack en línea: https://riptutorial.com/es/angular2/topic/9554/angular2-utilizando-webpack

Capítulo 15: Angular2 Validaciones personalizadas

Parámetros

parámetro	descripción
controlar	Este es el control que se está validando. Por lo general, deseará ver si control.value cumple con algunos criterios.

Examples

Ejemplos de validadores personalizados:

Angular 2 tiene dos tipos de validadores personalizados. Validadores síncronos como en el primer ejemplo que se ejecutarán directamente en el cliente y validadores asíncronos (el segundo ejemplo) que puede usar para llamar a un servicio remoto y hacer la validación por usted. En este ejemplo, el validador debe llamar al servidor para ver si un valor es único.

```
export class CustomValidators {

static cannotContainSpace(control: Control) {
    if (control.value.indexOf(' ') >= 0)
        return { cannotContainSpace: true };

    return null;
}

static shouldBeUnique(control: Control) {
    return new Promise((resolve, reject) => {
        // Fake a remote validator.
        setTimeout(function () {
            if (control.value == "exisitingUser")
                resolve({ shouldBeUnique: true });
            else
                resolve(null);
        }, 1000);
    });
}}
```

Si su valor de control es válido, simplemente devuelve el valor nulo a la persona que llama. De lo contrario, puede devolver un objeto que describe el error.

Usando validadores en el Formbuilder

```
   constructor(fb: FormBuilder) {
    this.form = fb.group({
```

```
        firstInput: ['', Validators.compose([Validators.required,
CustomValidators.cannotContainSpace]), CustomValidators.shouldBeUnique],
        secondInput: ['', Validators.required]
    });
}
```

Aquí usamos el FormBuilder para crear un formulario muy básico con dos cuadros de entrada. El FromBuilder toma una matriz para tres argumentos para cada control de entrada.

1. El valor por defecto del control.
2. Los validadores que se ejecutarán en el cliente. Puede usar Validators.compose ([arrayOfValidators]) para aplicar varios validadores en su control.
3. Uno o más validadores asíncronos de manera similar al segundo argumento.

Obtener / establecer parámetros de controles de formBuilder

Hay 2 formas de configurar los parámetros de controles de formBuilder.

1. Al inicializar:

```
exampleForm : FormGroup;
constructor(fb: FormBuilder){
  this.exampleForm = fb.group({
      name : new FormControl({value: 'default name'}, Validators.compose([Validators.required,
Validators.maxLength(15)]))
   });
}
```

2.Después de inicializar:

```
this.exampleForm.controls['name'].setValue('default name');
```

Obtener valor de control de formBuilder:

```
let name = this.exampleForm.controls['name'].value();
```

Lea Angular2 Validaciones personalizadas en línea:
https://riptutorial.com/es/angular2/topic/6284/angular2-validaciones-personalizadas

Capítulo 16: Angular-cli

Introducción

Aquí encontrará cómo comenzar con angular-cli, generando nuevo componente / service / pipe / module con angular-cli, agregue 3 partes como bootstrap, construya un proyecto angular.

Examples

Crear una aplicación Angular2 vacía con angular-cli

Requisitos:

- NodeJS: Página de descarga
- npm o hilo

Ejecute los siguientes comandos con cmd desde la nueva carpeta del directorio:

1. `npm install -g @angular/cli` o `yarn global add @angular/cli`
2. `ng new PROJECT_NAME`
3. `cd PROJECT_NAME`
4. `ng serve`

Abra su navegador en localhost: 4200

Generación de componentes, directivas, tuberías y servicios.

simplemente use su cmd: puede usar el comando ng generate (o simplemente ng g) para generar componentes angulares:

- Componente: `ng g component my-new-component`
- Directiva: `ng g directive my-new-directive`
- Tubería: `ng g pipe my-new-pipe`
- Servicio: `ng g service my-new-service`
- Clase: `ng g class my-new-classt`
- Interfaz: `ng g interface my-new-interface`
- Enumeración: `ng g enum my-new-enum`
- Módulo: `ng g module my-module`

Añadiendo libs de terceros

En angular-cli.json puedes cambiar la configuración de la aplicación.

Si desea agregar ng2-bootstrap por ejemplo:

1. `npm install ng2-bootstrap --save` o `yarn add ng2-bootstrap`

2. En angular-cli.json solo agregue la ruta de la rutina de carga en los módulos de nodo.

```
"scripts": [
    "../node_modules/jquery/dist/jquery.js",
    "../node_modules/bootstrap/dist/js/bootstrap.js"
 ]
```

construir con angular-cli

En angular-cli.json at outDir key puede definir su directorio de compilación;

estos son equivalentes

```
ng build --target=production --environment=prod
ng build --prod --env=prod
ng build --prod
```

y asi son estos

```
ng build --target=development --environment=dev
ng build --dev --e=dev
ng build --dev
ng build
```

Al crear, puede modificar la etiqueta base () en su index.html con --base-href su opción-url.

Establece la etiqueta base href en / myUrl / en tu index.html

```
ng build --base-href /myUrl/
ng build --bh /myUrl/
```

Nuevo proyecto con scss / sass como hoja de estilo.

Los archivos de estilo predeterminados generados y compilados por `@angular/cli` son **css** .

Si quiere usar **scss** en su lugar, genere su proyecto con:

```
ng new project_name --style=scss
```

Si quieres usar **sass** , genera tu proyecto con:

```
ng new project_name --style=sass
```

Establezca el hilo como gestor de paquetes predeterminado para @ angular / cli

Yarn es una alternativa para npm, el gestor de paquetes predeterminado en @ angular / cli. Si

desea utilizar hilados como gestor de paquetes para @ angular / cli, siga estos pasos:

Requerimientos

- hilo (`npm install --global yarn` o vea la página de instalación)
- @ angular / cli (`npm install -g @angular/cli` o `yarn global add @angular/cli`)

Para configurar el hilo como @ angular / cli package manager:

```
ng set --global packageManager=yarn
```

Para restablecer npm como @ angular / cli package manager:

```
ng set --global packageManager=npm
```

Lea Angular-cli en línea: https://riptutorial.com/es/angular2/topic/8956/angular-cli

Capítulo 17: Animación

Examples

Transición entre estados nulos

```
    @Component({
        ...
        animations: [
            trigger('appear', [
                transition(':enter', [
                    style({
                      //style applied at the start of animation
                    }),
                animate('300ms ease-in', style({
                    //style applied at the end of animation
                }))
                ])
            ])
        ]
    })
    class AnimComponent {

    }
]
```

Animando entre múltiples estados

El `<div>` en esta plantilla crece a `50px` y luego a `100px` y luego se reduce a `20px` al hacer clic en el botón.

Cada `state` tiene un estilo asociado descrito en los metadatos de `@Component` .

La lógica para cualquier `state` esté activo puede administrarse en la lógica del componente. En este caso, el `size` variable del componente contiene el valor de cadena "pequeño", "mediano" o "grande".

El elemento `<div>` responde a ese valor a través del `trigger` especificado en los metadatos de `@Component` : `[@size]="size"` .

```
@Component({
  template: '<div [@size]="size">Some Text</div><button
(click)="toggleSize()">TOGGLE</button>',
  animations: [
    trigger('size', [
      state('small', style({
        height: '20px'
      })),
      state('medium', style({
        height: '50px'
      })),
      state('large', style({
```

```
        height: '100px'
      })),
      transition('small => medium', animate('100ms')),
      transition('medium => large', animate('200ms')),
      transition('large => small', animate('300ms'))
    ])
  ]
})
export class TestComponent {

    size: string;

    constructor(){
        this.size = 'small';
    }
    toggleSize(){
        switch(this.size) {
            case 'small':
                this.size = 'medium';
                break;
            case 'medium':
                this.size = 'large';
                break;
            case 'large':
                this.size = 'small';
        }
    }
}
```

Lea Animación en línea: https://riptutorial.com/es/angular2/topic/8127/animacion

Capítulo 18: Barril

Introducción

Un barril es una forma de acumular exportaciones de varios módulos ES2015 en un único módulo de conveniencia ES2015. El propio barril es un archivo de módulo ES2015 que reexporta exportaciones seleccionadas de otros módulos ES2015.

Examples

Usando barril

Por ejemplo, sin un barril, un consumidor necesitaría tres declaraciones de importación:

```
import { HeroComponent } from '../heroes/hero.component.ts';
import { Hero }          from '../heroes/hero.model.ts';
import { HeroService }   from '../heroes/hero.service.ts';
```

Podemos agregar un barril creando un archivo en la misma carpeta de componentes. En este caso, la carpeta se llama 'heroes' llamada index.ts (usando las convenciones) que exporta todos estos elementos:

```
export * from './hero.model.ts';   // re-export all of its exports
export * from './hero.service.ts'; // re-export all of its exports
export { HeroComponent } from './hero.component.ts'; // re-export the named thing
```

Ahora un consumidor puede importar lo que necesita del barril.

```
import { Hero, HeroService } from '../heroes/index';
```

Aún así, esto puede convertirse en una línea muy larga; que podría reducirse aún más.

```
import * as h from '../heroes/index';
```

¡Eso es bastante reducido! El `* as h` importa todos los módulos y alias como h

Lea Barril en línea: https://riptutorial.com/es/angular2/topic/10717/barril

Capítulo 19: Bootstrap módulo vacío en angular 2

Examples

Un modulo vacío

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

@NgModule({
  declarations: [], // components your module owns.
  imports: [], // other modules your module needs.
  providers: [], // providers available to your module.
  bootstrap: [] // bootstrap this root component.
})
export class MyModule {}
```

Este es un módulo vacío que no contiene declaraciones, importaciones, proveedores ni componentes para bootstrap. Utilice esta una referencia.

Un módulo con red en el navegador web.

```
// app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpModule } from '@angular/http';
import { MyRootComponent } from './app.component';

@NgModule({
  declarations: [MyRootComponent],
  imports: [BrowserModule, HttpModule],
  bootstrap: [MyRootComponent]
})
export class MyModule {}
```

`MyRootComponent` es el componente raíz empaquetado en `MyModule` . Es el punto de entrada a su aplicación Angular 2.

Bootstrapping su módulo

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { MyModule }               from './app.module';

platformBrowserDynamic().bootstrapModule( MyModule );
```

En este ejemplo, `MyModule` es el módulo que contiene su componente raíz. Al `MyModule` su aplicación Angular 2 está lista para funcionar.

Módulo raíz de aplicación

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Bootstrapping estático con clases de fábrica

Podemos arrancar una aplicación de forma estática tomando la salida de Javascript ES5 de las clases de fábrica generadas. Entonces podemos usar esa salida para arrancar la aplicación:

```
import { platformBrowser } from '@angular/platform-browser';
import { AppModuleNgFactory } from './main.ngfactory';

// Launch with the app module factory.
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

Esto hará que el paquete de aplicaciones sea mucho más pequeño, ya que toda la compilación de la plantilla ya se realizó en un paso de compilación, usando ngc o llamando directamente a sus componentes internos.

Lea Bootstrap módulo vacío en angular 2 en línea:
https://riptutorial.com/es/angular2/topic/5508/bootstrap-modulo-vacio-en-angular-2

Capítulo 20: cobertura de la prueba angular-cli

Introducción

la cobertura de prueba se define como una técnica que determina si nuestros casos de prueba cubren realmente el código de la aplicación y la cantidad de código que se ejerce cuando ejecutamos esos casos de prueba.

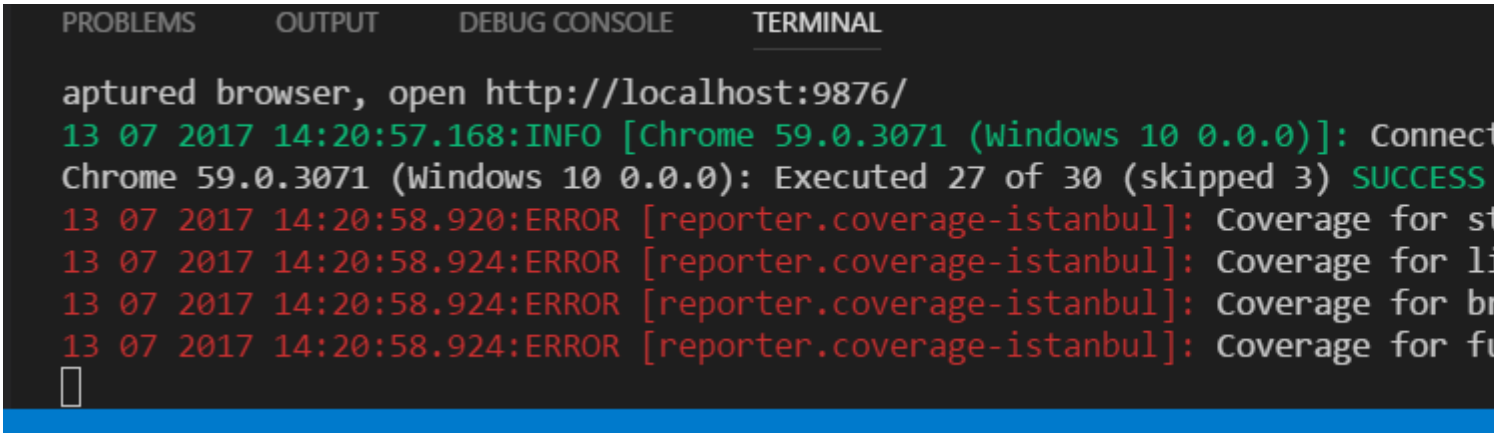
La CLI angular ha incorporado una función de cobertura de código con solo un simple comando `ng test --cc`

Examples

Una simple prueba de la base de comando angular-cli cobertura

Si desea ver las estadísticas generales de cobertura de prueba que, por supuesto, en CLI angular, puede simplemente escribir el siguiente comando y ver la parte inferior de la ventana del símbolo del sistema para obtener resultados.

```
ng test --cc // or --code-coverage
```



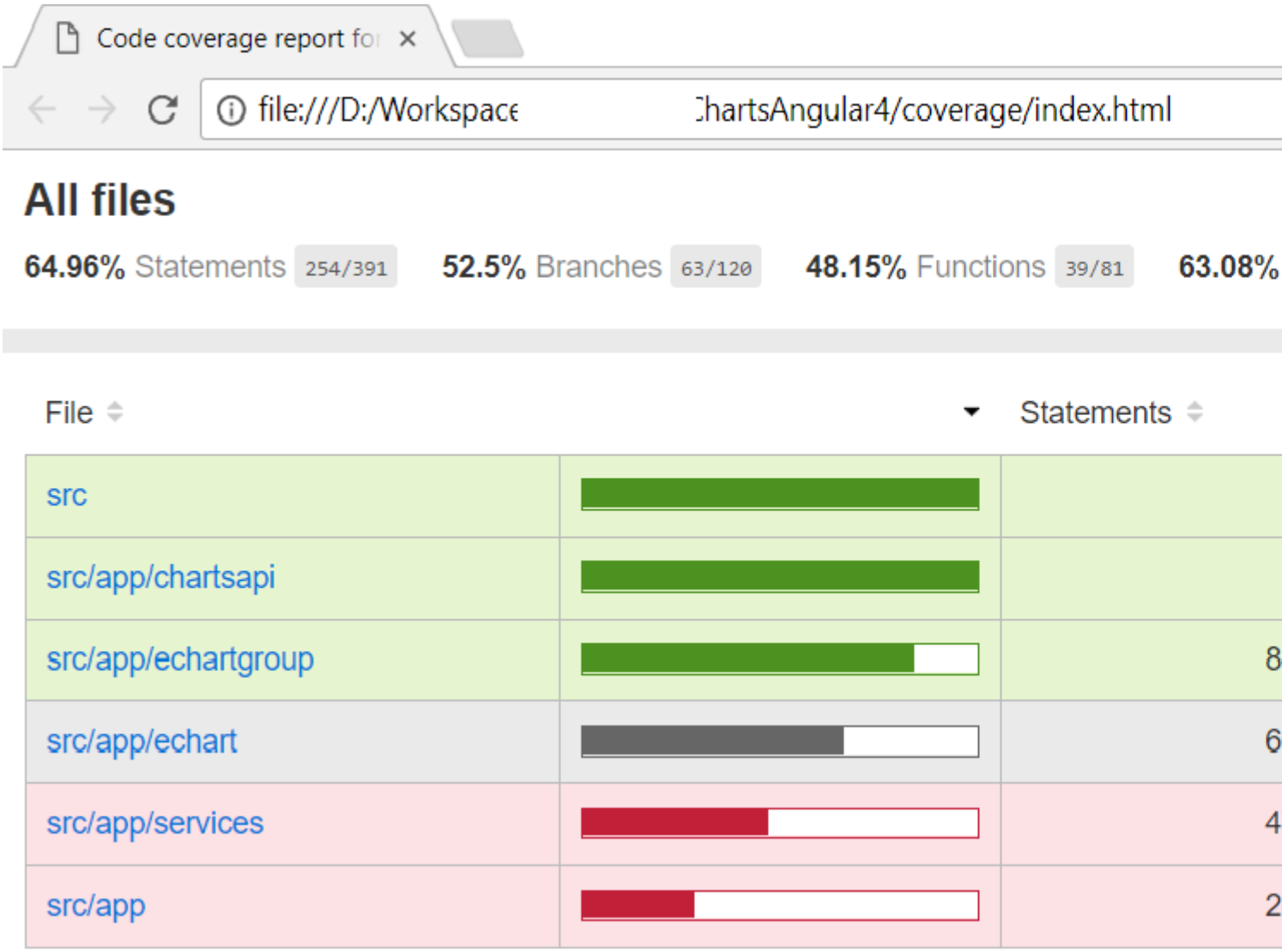
Informe detallado de la cobertura de pruebas gráficas de componentes individuales

Si desea ver la cobertura individual de las pruebas de los componentes, siga estos pasos.

1. `npm install --save-dev karma-teamcity-reporter`
2. `Add `require('karma-teamcity-reporter')` to list of plugins in karma.conf.js`
3. `ng test --code-coverage --reporters=teamcity,coverage-istanbul`

tenga en cuenta que la lista de reporteros está separada por comas, ya que hemos agregado un nuevo reportero, teamcity.

después de ejecutar este comando, puede ver la `coverage` la carpeta en su directorio y abrir `index.html` para obtener una vista gráfica de la cobertura de la prueba.



También puede establecer el umbral de cobertura que desea alcanzar, en `karma.conf.js` , así.

```
coverageIstanbulReporter: {
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true,
      thresholds: {
        statements: 90,
        lines: 90,
        branches: 90,
        functions: 90
      }
    },
```

Lea cobertura de la prueba angular-cli en línea:
https://riptutorial.com/es/angular2/topic/10764/cobertura-de-la-prueba-angular-cli

Capítulo 21: Cómo usar ngfor

Introducción

`ngFor` utiliza la directiva ngFor para crear una instancia de una plantilla para cada elemento de un objeto iterable. Esta directiva vincula lo iterable con el DOM, por lo que si el contenido de los iterables cambios, el contenido del DOM también se cambiará.

Examples

Ejemplo de lista desordenada

```
<ul>
  <li *ngFor="let item of items">{{item.name}}</li>
</ul>
```

Ejemplo de plantilla más completa

```
<div *ngFor="let item of items">
  <p>{{item.name}}</p>
  <p>{{item.price}}</p>
  <p>{{item.description}}</p>
</div>
```

Ejemplo de seguimiento de la interacción actual

```
<div *ngFor="let item of items; let i = index">
  <p>Item number: {{i}}</p>
</div>
```

En este caso, tomaré el valor de índice, que es la iteración actual del bucle.

Angial2 alias valores exportados

Angular2 proporciona varios valores exportados que pueden ser asignados a las variables locales. Estos son:

- índice
- primero
- último
- incluso
- impar

Excepto el `index` , los otros toman un valor `Boolean` . Como en el ejemplo anterior que usa el índice, se puede usar cualquiera de estos valores exportados:

```
<div *ngFor="let item of items; let firstItem = first; let lastItem = last">
  <p *ngIf="firstItem">I am the first item and I am gonna be showed</p>
  <p *ngIf="firstItem">I am not the first item and I will not show up :(</p>
  <p *ngIf="lastItem">But I'm gonna be showed as I am the last item :)</p>
</div>
```

* ngPara tubo

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'even'
})

export class EvenPipe implements PipeTransform {
    transform(value: string): string {
        if(value && value %2 === 0){
          return value;
        }
    }
}

@Component({
      selector: 'example-component',
      template: '<div>
                      <div *ngFor="let number of numbers | even">
                          {{number}}
                      </div>
                </div>'
})

export class exampleComponent {
    let numbers : List<number> = Array.apply(null, {length: 10}).map(Number.call, Number)
}
```

Lea Cómo usar ngfor en línea: https://riptutorial.com/es/angular2/topic/8051/como-usar-ngfor

Capítulo 22: Cómo usar ngif

Introducción

* **NgIf** : elimina o recrea una parte del árbol DOM en función de la evaluación de una expresión. Es una directiva estructural y las directivas estructurales modifican el diseño del DOM al agregar, reemplazar y eliminar sus elementos.

Sintaxis

- <div * ngIf = "false"> test <! - se evalúa como falso ->
- <div * ngIf = "undefined"> test <! - se evalúa como falso ->
- <div * ngIf = "null"> test <! - se evalúa como falso ->
- <div * ngIf = "0"> prueba <! - se evalúa como falso ->
- <div * ngIf = "NaN"> test <! - se evalúa como falso ->
- <div * ngIf = ""> test <! - se evalúa como falso ->
- Todos los demás valores se evalúan como verdaderos.

Examples

Mostrar un mensaje de carga

Si nuestro componente no está listo y está esperando datos del servidor, podemos agregar el cargador usando * ngIf. **Pasos:**

Primero declara un booleano:

```
loading: boolean = false;
```

A continuación, en su componente agregue un `ngOnInit` ciclo de vida llamado `ngOnInit`

```
ngOnInit() {
   this.loading = true;
}
```

y después de obtener datos completos del servidor, establezca que carga booleano en falso.

```
 this.loading=false;
```

En su plantilla html use * ngIf con la propiedad de `loading` :

```
<div *ngIf="loading" class="progress">
   <div class="progress-bar info" style="width: 125%;"></div>
</div>
```

Mostrar mensaje de alerta en una condición

```
<p class="alert alert-success" *ngIf="names.length > 2">Currently there are more than 2
names!</p>
```

Para ejecutar una función al principio o al final de * ngFor loop Usando * ngIf

NgFor proporciona algunos valores que pueden ser asignados a las variables locales.

- **índice** : (variable) posición del elemento actual en el iterable que comienza en 0
- **first** - (boolean) true si el elemento actual es el primer elemento en el iterable
- **last** - (boolean) true si el elemento actual es el último elemento del iterable
- **even** - (boolean) true si el índice actual es un número par
- **impar** - (booleano) verdadero si el índice actual es un número impar

```
<div *ngFor="let note of csvdata; let i=index; let lastcall=last">
      <h3>{{i}}</h3> <-- to show index position
      <h3>{{note}}</h3>
      <span *ngIf="lastcall">{{anyfunction()}} </span><-- this lastcall boolean value will be
true only if this is last in loop
      // anyfunction() will run at the end of loop same way we can do at start
    </div>
```

Use * ngIf con * ngFor

Si bien no se le permite usar `*ngIf` y `*ngFor` en el mismo div (dará un error en el tiempo de ejecución), puede anidar el `*ngIf` en el `*ngFor` para obtener el comportamiento deseado.

Ejemplo 1: sintaxis general

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="<your condition here>">

   <!-- Execute code here if statement true -->

  </div>
</div>
```

Ejemplo 2: Mostrar elementos con índice par

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 2 == 0">
    {{ item }}
  </div>
</div>
```

La desventaja es que es necesario agregar un elemento `div` externo adicional.

Pero considere este caso de uso donde un elemento `div` necesita ser iterado (usando * ngFor) y también incluye una verificación de si el elemento necesita ser eliminado o no (usando * ngIf),

pero no es preferible agregar un `div` adicional. En este caso, puede usar la etiqueta de `template` para el * ngFor:

```
<template ngFor let-item [ngForOf]="items">
    <div *ngIf="item.price > 100">
    </div>
</template>
```

De esta manera, no es necesario agregar un `div` externo adicional y, además, el elemento `<template>` no se agregará al DOM. Los únicos elementos agregados en el DOM del ejemplo anterior son los elementos `div` iterados.

Nota: En Angular v4 `<template>` ha sido desaprobado en favor de `<ng-template>` y se eliminará en v5. En Angular v2.x, las versiones `<template>` siguen siendo válidas.

Lea Cómo usar ngif en línea: https://riptutorial.com/es/angular2/topic/8346/como-usar-ngif

Capítulo 23: Compilación anticipada (AOT) con Angular 2

Examples

1. Instalar dependencias de Angular 2 con compilador.

NOTA: para obtener los mejores resultados, asegúrese de que su proyecto se haya creado utilizando Angular-CLI.

```
npm install angular/{core,common,compiler,platform-browser,platform-browser-
dynamic,http,router,forms,compiler-cli,tsc-wrapped,platform-server}
```

No tiene que realizar este paso si el proyecto ya tiene angular 2 y todas estas dependencias instaladas. Sólo asegúrese de que el `compiler` está allí.

2. Agregue `angularCompilerOptions` a su archivo` tsconfig.json`

```
...
"angularCompilerOptions": {
    "genDir": "./ngfactory"
}
...
```

Esta es la carpeta de salida del compilador.

3. Ejecuta ngc, el compilador angular

desde la raíz de su proyecto `./node_modules/.bin/ngc -p src` donde `src` es donde vive todo su código angular 2. Esto generará una carpeta llamada `ngfactory` donde `ngfactory` todo el código compilado.

`"node_modules/.bin/ngc" -p src` para windows

4. Modifique el archivo `main.ts` para usar NgFactory y el navegador de plataforma estático

```
// this is the static platform browser, the usual counterpart is @angular/platform-browser-
dynamic.
import { platformBrowser } from '@angular/platform-browser';

// this is generated by the angular compiler
import { AppModuleNgFactory } from './ngfactory/app/app.module.ngfactory';

// note the use of `bootstrapModuleFactory`, as opposed to `bootstrapModule`.
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

En este punto deberías poder ejecutar tu proyecto. En este caso, mi proyecto se creó utilizando Angular-CLI.

```
> ng serve
```

¿Por qué necesitamos compilación, compilación de flujo de eventos y ejemplo?

P. ¿Por qué necesitamos compilación? Respuesta Necesitamos una compilación para lograr un mayor nivel de eficiencia de nuestras aplicaciones Angulares.

Echa un vistazo al siguiente ejemplo,

```
// ...
compile: function (el, scope) {
  var dirs = this._getElDirectives(el);
  var dir;
  var scopeCreated;
  dirs.forEach(function (d) {
    dir = Provider.get(d.name + Provider.DIRECTIVES_SUFFIX);
    if (dir.scope && !scopeCreated) {
      scope = scope.$new();
      scopeCreated = true;
    }
    dir.link(el, scope, d.value);
  });
  Array.prototype.slice.call(el.children).forEach(function (c) {
    this.compile(c, scope);
  }, this);
},
// ...
```

Usando el código de arriba para renderizar la plantilla,

```
<ul>
  <li *ngFor="let name of names"></li>
</ul>
```

Es mucho más lento en comparación con:

```
// ...
this._text_9 = this.renderer.createText(this._el_3, '\n', null);
this._text_10 = this.renderer.createText(parentRenderNode, '\n\n', null);
this._el_11 = this.renderer.createElement(parentRenderNode, 'ul', null);
this._text_12 = this.renderer.createText(this._el_11, '\n  ', null);
this._anchor_13 = this.renderer.createTemplateAnchor(this._el_11, null);
this._appEl_13 = new import2.AppElement(13, 11, this, this._anchor_13);
this._TemplateRef_13_5 = new import17.TemplateRef_(this._appEl_13,
viewFactory_HomeComponent1);
this._NgFor_13_6 = new import15.NgFor(this._appEl_13.vcRef, this._TemplateRef_13_5,
this.parentInjector.get(import18.IterableDiffers), this.ref);
// ...
```

Flujo de eventos con compilación anticipada de tiempo

En contraste, con AoT pasamos por los siguientes pasos:

1. Desarrollo de la aplicación Angular 2 con TypeScript.
2. Recopilación de la aplicación con ngc.
3. Realiza la compilación de las plantillas con el compilador Angular para TypeScript.
4. Recopilación del código de TypeScript a JavaScript.
5. Liar
6. Minificación.
7. Despliegue.

Aunque el proceso anterior parece un poco más complicado, el usuario solo sigue los pasos:

1. Descarga todos los activos.
2. Bootstraps angulares.
3. La aplicación se renderiza.

Como puede ver, falta el tercer paso, lo que significa que UX más rápido / mejor, y además de herramientas como angular2-seed y angle-cli, automatizará dramáticamente el proceso de construcción.

Espero que te pueda ayudar! ¡Gracias!

Uso de la compilación de AoT con CLI angular

La interfaz de línea de comandos de Angular CLI tiene soporte de compilación de AoT desde la versión beta 17.

Para construir su aplicación con compilación AoT, simplemente ejecute:

```
ng build --prod --aot
```

Lea Compilación anticipada (AOT) con Angular 2 en línea:
https://riptutorial.com/es/angular2/topic/6634/compilacion-anticipada--aot--con-angular-2

Capítulo 24: Componentes

Introducción

Los componentes angulares son elementos compuestos por una plantilla que representará su aplicación.

Examples

Un componente simple

Para crear un componente, agregamos el decorador `@Component` en una clase que pasa algunos parámetros:

- `providers` : recursos que se inyectarán en el componente constructor
- `selector` : el selector de consultas que encontrará el elemento en el HTML y reemplazará por el componente
- `styles` : `styles` línea. NOTA: NO use este parámetro con require, funciona en el desarrollo, pero cuando construye la aplicación en producción, todos sus estilos se pierden
- `styleUrls` : Array de ruta a archivos de estilo
- `template` : cadena que contiene su HTML
- `templateUrl` : Ruta a un archivo HTML

Hay otros parámetros que puede configurar, pero los enumerados son los que más utilizará.

Un ejemplo simple:

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

@Component({
  selector: 'app-required',
  styleUrls: ['required.component.scss'],
  // template: `This field is required.`,
  templateUrl: 'required.component.html',
})
export class RequiredComponent { }
```

Plantillas y Estilos

Las plantillas son archivos HTML que pueden contener lógica.

Puede especificar una plantilla de dos maneras:

Pasando plantilla como una ruta de archivo

```
@Component({
```

```
  templateUrl: 'hero.component.html',
})
```

Pasando una plantilla como un código en línea

```
@Component({
  template: `<div>My template here</div>`,
})
```

Las plantillas pueden contener estilos. Los estilos declarados en `@Component` son diferentes de su archivo de estilo de aplicación, todo lo que se aplique en el componente estará restringido a este alcance. Por ejemplo, digamos que usted agrega:

```
div { background: red; }
```

Todos los `div` s dentro del componente serán rojos, pero si tiene otros componentes, otros divs en su HTML no se cambiarán en absoluto.

El código generado se verá así:

```
<style>div[_ngcontent-c1] { background: red; }</style>
```

Puede agregar estilos a un componente de dos maneras:

Pasando una matriz de rutas de archivos

```
@Component({
  styleUrls: ['hero.component.css'],
})
```

Pasando una matriz de códigos en línea

```
styles: [ `div { background: lime; }` ]
```

No debe usar los `styles` con `require` ya que no funcionará cuando compile su aplicación para producción.

Probando un componente

hero.component.html

```
<form (ngSubmit)="submit($event)" [formGroup]="form" novalidate>
  <input type="text" formControlName="name" />
  <button type="submit">Show hero name</button>
</form>
```

hero.component.ts

```
import { FormControl, FormGroup, Validators } from '@angular/forms';

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

@Component({
  selector: 'app-hero',
  templateUrl: 'hero.component.html',
})
export class HeroComponent {
  public form = new FormGroup({
    name: new FormControl('', Validators.required),
  });

  submit(event) {
    console.log(event);
    console.log(this.form.controls.name.value);
  }
}
```

hero.component.spec.ts

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';

import { HeroComponent } from './hero.component';
import { ReactiveFormsModule } from '@angular/forms';

describe('HeroComponent', () => {
  let component: HeroComponent;
  let fixture: ComponentFixture<HeroComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [HeroComponent],
      imports: [ReactiveFormsModule],
    }).compileComponents();

    fixture = TestBed.createComponent(HeroComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  }));

  it('should be created', () => {
    expect(component).toBeTruthy();
  });

  it('should log hero name in the console when user submit form', async(() => {
    const heroName = 'Saitama';
    const element = <HTMLFormElement>fixture.debugElement.nativeElement.querySelector('form');

    spyOn(console, 'log').and.callThrough();

    component.form.controls['name'].setValue(heroName);

    element.querySelector('button').click();

    fixture.whenStable().then(() => {
      fixture.detectChanges();
      expect(console.log).toHaveBeenCalledWith(heroName);
    });
  }));
```

```
  it('should validate name field as required', () => {
    component.form.controls['name'].setValue('');
    expect(component.form.invalid).toBeTruthy();
  });
});
```

Componentes de anidación

Los componentes se representarán en su `selector` respectivo, por lo que puede usar eso para anidar componentes.

Si tienes un componente que muestra un mensaje:

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

@Component({
  selector: 'app-required',
  template: `{{name}} is required.`
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}
```

Puede usarlo dentro de otro componente usando la `app-required` (el selector de este componente):

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

@Component({
  selector: 'app-sample',
  template: `
    <input type="text" name="heroName" />
    <app-required name="Hero Name"></app-required>
  `
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}
```

Lea Componentes en línea: https://riptutorial.com/es/angular2/topic/10838/componentes

Capítulo 25: Configuración de la aplicación ASP.net Core para trabajar con Angular 2 y TypeScript

Introducción

ESCENARIO: ASP.NET Core background Angular 2 Front-End Angular 2 Componentes usando Asp.net Core Controllers

De esta forma se puede implementar la aplicación Angular 2 sobre Asp.Net Core. También nos permite llamar a los Controladores MVC desde los componentes de Angular 2 con la vista de resultados de MVC que admite Angular 2.

Examples

Asp.Net Core + Angular2 + Gulp

Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using CoreAngular000.Data;
using CoreAngular000.Models;
using CoreAngular000.Services;
using Microsoft.Extensions.FileProviders;
using System.IO;

namespace CoreAngular000
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
        var builder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional:
true);

        if (env.IsDevelopment())
```

```
        {

            builder.AddUserSecrets<Startup>();
        }

        builder.AddEnvironmentVariables();
        Configuration = builder.Build();
    }

    public IConfigurationRoot Configuration { get; }


    public void ConfigureServices(IServiceCollection services)
    {
        // Add framework services.
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

        services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>()
            .AddDefaultTokenProviders();

        services.AddMvc();

        // Add application services.
        services.AddTransient<IEmailSender, AuthMessageSender>();
        services.AddTransient<ISmsSender, AuthMessageSender>();
    }

   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
    {
        loggerFactory.AddConsole(Configuration.GetSection("Logging"));
        loggerFactory.AddDebug();

        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseDatabaseErrorPage();
            app.UseBrowserLink();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
        }

        app.UseDefaultFiles();
        app.UseStaticFiles();
        app.UseStaticFiles(new StaticFileOptions
        {
            FileProvider = new
PhysicalFileProvider(Path.Combine(env.ContentRootPath, "node_modules")),
            RequestPath = "/node_modules"
        });

        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
```

```
    }
}
}
```

tsConfig.json

```
    {
  "compilerOptions": {
    "diagnostics": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "lib": [ "es2015", "dom" ],
    "listFiles": true,
    "module": "commonjs",
    "moduleResolution": "node",
    "noImplicitAny": true,
    "outDir": "wwwroot",
    "removeComments": false,
   "rootDir": "wwwroot",
    "sourceMap": true,
    "suppressImplicitAnyIndexErrors": true,
    "target": "es5"
  },
  "exclude": [
    "node_modules",
    "wwwroot/lib/"
  ]
}
```

Paquete.json

```
   {
 "name": "angular dependencies and web dev package",
 "version": "1.0.0",
 "description": "Angular 2 MVC. Samuel Maícas Template",
 "scripts": {},
 "dependencies": {
   "@angular/common": "~2.4.0",
   "@angular/compiler": "~2.4.0",
   "@angular/core": "~2.4.0",
   "@angular/forms": "~2.4.0",
   "@angular/http": "~2.4.0",
   "@angular/platform-browser": "~2.4.0",
   "@angular/platform-browser-dynamic": "~2.4.0",
   "@angular/router": "~3.4.0",
  "angular-in-memory-web-api": "~0.2.4",
   "systemjs": "0.19.40",
   "core-js": "^2.4.1",
   "rxjs": "5.0.1",
   "zone.js": "^0.7.4"
 },
 "devDependencies": {
   "del": "^2.2.2",
   "gulp": "^3.9.1",
   "gulp-concat": "^2.6.1",
   "gulp-cssmin": "^0.1.7",
   "gulp-htmlmin": "^3.0.0",
   "gulp-uglify": "^2.1.2",
   "merge-stream": "^1.0.1",
```

```
    "tslint": "^3.15.1",
    "typescript": "~2.0.10"
  },
  "repository": {}
}
```

Bundleconfig.json

```
    [
  {
    "outputFileName": "wwwroot/css/site.min.css",
     "inputFiles": [
      "wwwroot/css/site.css"
    ]
  },
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  }
]
```

Convierta bundleconfig.json en gulpfile (haga clic con el botón derecho del ratón en bundleconfig.json en Solution Explorer, Bundler & Minifier> Convert to Gulp

Vistas / Inicio / Index.cshtml

```
    @{
    ViewData["Title"] = "Home Page";
}
<div>{{ nombre }}</div>
```

Para la carpeta wwwroot use https://github.com/angular/quickstart seed. Necesita: **index.html main.ts, systemjs-angular-loader.js, systemjs.config.js, tsconfig.json** y la **carpeta de** la **aplicación**

wwwroot / Index.html

```
    <html>
  <head>
    <title>SMTemplate Angular2 & ASP.NET Core</title>
    <base href="/">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">


    <script src="node_modules/core-js/client/shim.min.js"></script>

    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
```

```
    <script src="systemjs.config.js"></script>
    <script>
      System.import('main.js').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <my-app>Loading AppComponent here ...</my-app>
  </body>
</html>
```

Puede llamarlo a Controladores desde templateUrl. wwwroot / app / app.component.ts

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

@Component({
  selector: 'my-app',
  templateUrl: '/home/index',
 })
export class AppComponent  { nombre = 'Samuel Maícas'; }
```

[Semilla] Asp.Net Core + Angular2 + Gulp en Visual Studio 2017

1. Descargar semilla
2. Ejecutar dotnet restore
3. Ejecutar npm instalar

Siempre. Disfrutar.

https://github.com/SamML/CoreAngular000

MVC <-> Angular 2

Cómo: LLAMAR ANGULAR 2 HTML / JS COMPONENT DESDE ASP.NET Core CONTROLLER:

Llamamos al HTML en lugar de devolver View ()

```
 return File("~/html/About.html", "text/html");
```

Y carga componente angular en el html. Aquí podemos decidir si queremos trabajar con el mismo o diferente módulo. Depende de la situación.

wwwroot / html / About.html

```
    <!DOCTYPE html>
<html>
  <head>
    <title>About Page</title>
    <base href="/">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
      <link href="../css/site.min.css" rel="stylesheet" type="text/css"/>
```

```
    <script src="../node_modules/core-js/client/shim.min.js"></script>

    <script src="../node_modules/zone.js/dist/zone.js"></script>
    <script src="../node_modules/systemjs/dist/system.src.js"></script>

    <script src="../systemjs.config.js"></script>
    <script>
      System.import('../main.js').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <aboutpage>Loading AppComponent here ...</aboutpage>
  </body>
</html>
```

(*) Ya esta semilla necesita cargar la lista completa de recursos

Cómo: LLAMAR ASP.NET Core Controller para mostrar una vista MVC con soporte Angular2:

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

@Component({
  selector: 'aboutpage',
  templateUrl: '/home/about',
})
export class AboutComponent  {

}
```

Lea Configuración de la aplicación ASP.net Core para trabajar con Angular 2 y TypeScript en línea: https://riptutorial.com/es/angular2/topic/9543/configuracion-de-la-aplicacion-asp-net-core-para-trabajar-con-angular-2-y-typescript

Capítulo 26: Creación de una biblioteca de npm Angular

Introducción

Cómo publicar su NgModule, escrito en TypeScript en el registro de npm. Configuración de proyecto npm, compilador de scripts, rollup y compilación de integración continua.

Examples

Módulo mínimo con clase de servicio.

Estructura de archivos

```
/
    -src/
        awesome.service.ts
        another-awesome.service.ts
        awesome.module.ts
    -index.ts
    -tsconfig.json
    -package.json
    -rollup.config.js
    -.npmignore
```

Servicio y modulo

Coloque su trabajo impresionante aquí.

src / awesome.service.ts:

```
export class AwesomeService {
    public doSomethingAwesome(): void {
        console.log("I am so awesome!");
    }
}
```

src / awesome.module.ts:

```
import { NgModule } from '@angular/core'
import { AwesomeService } from './awesome.service';
import { AnotherAwesomeService } from './another-awesome.service';

@NgModule({
    providers: [AwesomeService, AnotherAwesomeService]
```

```
})
export class AwesomeModule {}
```

Haga que su módulo y servicio sean accesibles afuera.

/index.ts:

```
export { AwesomeService } from './src/awesome.service';
export { AnotherAwesomeService } from './src/another-awesome.service';
export { AwesomeModule } from './src/awesome.module';
```

Compilacion

En compilerOptions.paths debe especificar todos los módulos externos que utilizó en su paquete.

/tsconfig.json

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "declaration": true,
    "stripInternal": true,
    "experimentalDecorators": true,
    "strictNullChecks": false,
    "noImplicitAny": true,
    "module": "es2015",
    "moduleResolution": "node",
    "paths": {
      "@angular/core": ["node_modules/@angular/core"],
      "rxjs/*": ["node_modules/rxjs/*"]
    },
    "rootDir": ".",
    "outDir": "dist",
    "sourceMap": true,
    "inlineSources": true,
    "target": "es5",
    "skipLibCheck": true,
    "lib": [
      "es2015",
      "dom"
    ]
  },
  "files": [
    "index.ts"
  ],
  "angularCompilerOptions": {
    "strictMetadataEmit": true
  }
}
```

Especifique sus externos de nuevo

/rollup.config.js

```
export default {
    entry: 'dist/index.js',
    dest: 'dist/bundles/awesome.module.umd.js',
    sourceMap: false,
    format: 'umd',
    moduleName: 'ng.awesome.module',
    globals: {
        '@angular/core': 'ng.core',
        'rxjs': 'Rx',
        'rxjs/Observable': 'Rx',
        'rxjs/ReplaySubject': 'Rx',
        'rxjs/add/operator/map': 'Rx.Observable.prototype',
        'rxjs/add/operator/mergeMap': 'Rx.Observable.prototype',
        'rxjs/add/observable/fromEvent': 'Rx.Observable',
        'rxjs/add/observable/of': 'Rx.Observable'
    },
    external: ['@angular/core', 'rxjs']
}
```

Ajustes de NPM

Ahora, vamos a colocar algunas instrucciones para npm

/package.json

```
{
  "name": "awesome-angular-module",
  "version": "1.0.4",
  "description": "Awesome angular module",
  "main": "dist/bundles/awesome.module.umd.min.js",
  "module": "dist/index.js",
  "typings": "dist/index.d.ts",
  "scripts": {
    "test": "",
    "transpile": "ngc",
    "package": "rollup -c",
    "minify": "uglifyjs dist/bundles/awesome.module.umd.js --screw-ie8 --compress --mangle --
comments --output dist/bundles/awesome.module.umd.min.js",
    "build": "rimraf dist && npm run transpile && npm run package && npm run minify",
    "prepublishOnly": "npm run build"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/maciejtreder/awesome-angular-module.git"
  },
  "keywords": [
    "awesome",
    "angular",
    "module",
    "minimal"
  ],
  "author": "Maciej Treder <contact@maciejtreder.com>",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/maciejtreder/awesome-angular-module/issues"
  },
  "homepage": "https://github.com/maciejtreder/awesome-angular-module#readme",
```

```
  "devDependencies": {
    "@angular/compiler": "^4.0.0",
    "@angular/compiler-cli": "^4.0.0",
    "rimraf": "^2.6.1",
    "rollup": "^0.43.0",
    "typescript": "^2.3.4",
    "uglify-js": "^3.0.21"
  },
  "dependencies": {
    "@angular/core": "^4.0.0",
    "rxjs": "^5.3.0"
  }
}
```

También podemos especificar qué archivos, npm debe ignorar.

/.npmignore

```
node_modules
npm-debug.log
Thumbs.db
.DS_Store
src
!dist/src
plugin
!dist/plugin
*.ngsummary.json
*.iml
rollup.config.js
tsconfig.json
*.ts
!*.d.ts
.idea
```

Integración continua

Finalmente puedes configurar la integración continua.

.travis.yml

```
language: node_js
node_js:
- node

deploy:
  provider: npm
  email: contact@maciejtreder.com
  api_key:
    secure: <your api key>
  on:
    tags: true
    repo: maciejtreder/awesome-angular-module
```

La demostración se puede encontrar aquí: https://github.com/maciejtreder/awesome-angular-module

Lea Creación de una biblioteca de npm Angular en línea:
https://riptutorial.com/es/angular2/topic/10704/creacion-de-una-biblioteca-de-npm-angular

Capítulo 27: Crear un paquete Angular 2+ NPM

Introducción

A veces necesitamos compartir algún componente entre algunas aplicaciones y publicarlo en npm es una de las mejores formas de hacerlo.

Hay algunos trucos que necesitamos saber para poder usar un componente normal como paquete npm sin cambiar la estructura como estilos externos integrados.

Puedes ver un ejemplo mínimo aquí

Examples

Paquete mas simple

Aquí compartimos un flujo de trabajo mínimo para crear y publicar un paquete Angular 2+ npm.

Archivos de configuración

Necesitamos algunos archivos de configuración para decirle a `git` , `npm` , `gulp` y `typescript` cómo actuar.

.gitignore

Primero creamos un archivo `.gitignore` para evitar la `.gitignore` de versiones de archivos y carpetas no deseados. El contenido es:

```
npm-debug.log
node_modules
jspm_packages
.idea
build
```

.npmignore

En segundo lugar, creamos un archivo `.npmignore` para evitar la publicación de archivos y carpetas no deseados. El contenido es:

```
examples
node_modules
src
```

gulpfile.js

Necesitamos crear un `gulpfile.js` para decirle a Gulp cómo compilar nuestra aplicación. Esta parte es necesaria porque debemos minimizar e incluir todas las plantillas y estilos externos antes de publicar nuestro paquete. El contenido es:

```
var gulp = require('gulp');
var embedTemplates = require('gulp-angular-embed-templates');
var inlineNg2Styles = require('gulp-inline-ng2-styles');

gulp.task('js:build', function () {
    gulp.src('src/*.ts') // also can use *.js files
        .pipe(embedTemplates({sourceType:'ts'}))
        .pipe(inlineNg2Styles({ base: '/src' }))
        .pipe(gulp.dest('./dist'));
});
```

index.d.ts

`index.d.ts` archivo `index.d.ts` al importar un módulo externo. Ayuda editor con auto-finalización y consejos de función.

```
export * from './lib';
```

index.js

Este es el punto de entrada del paquete. Cuando instale este paquete usando NPM e importe en su aplicación, solo necesita pasar el nombre del paquete y su aplicación aprenderá dónde encontrar cualquier componente EXPORTADO de su paquete.

```
exports.AngularXMinimalNpmPackageModule = require('./lib').AngularXMinimalNpmPackageModule;
```

Usamos la carpeta `lib` porque cuando compilamos nuestro código, la salida se coloca dentro de la carpeta `/lib` .

paquete.json

Este archivo se utiliza para configurar su publicación npm y define los paquetes necesarios para que funcione.

```
{
  "name": "angular-x-minimal-npm-package",
  "version": "0.0.18",
  "description": "An Angular 2+ Data Table that uses HTTP to create, read, update and delete
data from an external API such REST.",
  "main": "index.js",
  "scripts": {
    "watch": "tsc -p src -w",
    "build": "gulp js:build && rm -rf lib && tsc -p dist"
  },
```

```
  "repository": {
    "type": "git",
    "url": "git+https://github.com/vinagreti/angular-x-minimal-npm-package.git"
  },
  "keywords": [
    "Angular",
    "Angular2",
    "Datatable",
    "Rest"
  ],
  "author": "bruno@tzadi.com",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/vinagreti/angular-x-minimal-npm-package/issues"
  },
  "homepage": "https://github.com/vinagreti/angular-x-minimal-npm-package#readme",
  "devDependencies": {
    "gulp": "3.9.1",
    "gulp-angular-embed-templates": "2.3.0",
    "gulp-inline-ng2-styles": "0.0.1",
    "typescript": "2.0.0"
  },
  "dependencies": {
    "@angular/common": "2.4.1",
    "@angular/compiler": "2.4.1",
    "@angular/core": "2.4.1",
    "@angular/http": "2.4.1",
    "@angular/platform-browser": "2.4.1",
    "@angular/platform-browser-dynamic": "2.4.1",
    "rxjs": "5.0.2",
    "zone.js": "0.7.4"
  }
}
```

dist / tsconfig.json

Crea una carpeta dist y coloca este archivo dentro. Este archivo se utiliza para indicar a Typescript cómo compilar su aplicación. Dónde obtener la carpeta mecanografiada y dónde colocar los archivos compilados.

```
{
  "compilerOptions": {
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "mapRoot": "",
    "rootDir": ".",
    "target": "es5",
    "lib": ["es6", "es2015", "dom"],
    "inlineSources": true,
    "stripInternal": true,
    "module": "commonjs",
    "moduleResolution": "node",
    "removeComments": true,
    "sourceMap": true,
    "outDir": "../lib",
    "declaration": true
  }
}
```

Después de crear los archivos de configuración, debemos crear nuestro componente y módulo. Este componente recibe un clic y muestra un mensaje. Se utiliza como una etiqueta html `<angular-x-minimal-npm-package></angular-x-minimal-npm-package>` . Simplemente instale este paquete npm y cargue su módulo en el modelo que desea usar.

src / angular-x-minimal-npm-package.component.ts

```
import {Component} from '@angular/core';
@Component({
    selector: 'angular-x-minimal-npm-package',
    styleUrls: ['./angular-x-minimal-npm-package.component.scss'],
    templateUrl: './angular-x-minimal-npm-package.component.html'
})
export class AngularXMinimalNpmPackageComponent {
    message = "Click Me ...";
    onClick() {
        this.message = "Angular 2+ Minimal NPM Package. With external scss and html!";
    }
}
```

src / angular-x-minimal-npm-package.component.html

```
<div>
  <h1 (click)="onClick()">{{message}}</h1>
</div>
```

src / angular-x-data-table.component.css

```
h1{
    color: red;
}
```

src / angular-x-minimal-npm-package.module.ts

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

import { AngularXMinimalNpmPackageComponent } from './angular-x-minimal-npm-
package.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AngularXMinimalNpmPackageComponent ],
  exports:  [ AngularXMinimalNpmPackageComponent ],
  entryComponents: [ AngularXMinimalNpmPackageComponent ],
})
export class AngularXMinimalNpmPackageModule {}
```

Después de eso, debes compilar, construir y publicar tu paquete.

Construir y compilar

Para construir usamos `gulp` y para compilar usamos `tsc` . El comando se establece en el archivo package.json, en la opción `scripts.build` . Tenemos este conjunto `gulp js:build && rm -rf lib && tsc -p dist` . Esta es nuestra cadena de tareas que hará el trabajo por nosotros.

Para compilar y compilar, ejecute el siguiente comando en la raíz de su paquete:

```
npm run build
```

Esto activará la cadena y terminará con su compilación en la carpeta `/dist` y el paquete compilado en su carpeta `/lib` . Es por eso que en `index.js` exportamos el código de la carpeta `/lib` y no de `/src` .

Publicar

Ahora solo necesitamos publicar nuestro paquete para poder instalarlo a través de npm. Para eso, simplemente ejecuta el comando:

```
npm publish
```

¡¡¡Eso es todo!!!

Lea Crear un paquete Angular 2+ NPM en línea:
https://riptutorial.com/es/angular2/topic/8790/crear-un-paquete-angular-2plus-npm

Capítulo 28: CRUD en Angular2 con API Restful

Sintaxis

- @Injectable () // Indica al inyector de dependencia que inyecte dependencias al crear una instancia de este servicio.
- request.subscribe () // Aquí es donde *realmente* realiza la solicitud. Sin esto su solicitud no será enviada. También desea leer respuesta en la función de devolución de llamada.
- constructor (wikiService privado: WikipediaService) {} // Dado que tanto nuestro servicio como sus dependencias son inyectables por el inyector de dependencias, es una buena práctica inyectar el servicio al componente para probar la aplicación de la unidad.

Examples

Leer de una API Restful en Angular2

Para separar la lógica de la API del componente, estamos creando el cliente de la API como una clase separada. Esta clase de ejemplo realiza una solicitud a la API de Wikipedia para obtener artículos de wiki aleatorios.

```
    import { Http, Response } from '@angular/http';
    import { Injectable } from '@angular/core';
    import { Observable }     from 'rxjs/Observable';
    import 'rxjs/Rx';

    @Injectable()
    export class WikipediaService{
        constructor(private http: Http) {}

        getRandomArticles(numberOfArticles: number)
        {
            var request =
this.http.get("https://en.wikipedia.org/w/api.php?action=query&list=random&format=json&rnlimit="
+ numberOfArticles);
            return request.map((response: Response) => {
                return response.json();
            },(error) => {
                console.log(error);
                //your want to implement your own error handling here.
            });
        }
    }
```

Y tener un componente para consumir nuestro nuevo cliente API.

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

```
import { WikipediaService } from './wikipedia.Service';

@Component({
    selector: 'wikipedia',
    templateUrl: 'wikipedia.component.html'
})
export class WikipediaComponent implements OnInit {
    constructor(private wikiService: WikipediaService) { }

    private articles: any[] = null;
    ngOnInit() {
        var request = this.wikiService.getRandomArticles(5);
        request.subscribe((res) => {
            this.articles = res.query.random;
        });
    }
}
```

Lea CRUD en Angular2 con API Restful en línea:
https://riptutorial.com/es/angular2/topic/7343/crud-en-angular2-con-api-restful

Capítulo 29: custom ngx-bootstrap datepicker + entrada

Examples

fecha ngx-bootstrap datepicker

datepicker.component.html

```
<div (clickOutside)="onClickedOutside($event)" (blur)="onClickedOutside($event)">
      <div class="input-group date" [ngClass]="{'disabled-icon': disabledDatePicker == false
}">
        <input (change)="changedDate()" type="text" [ngModel]="value" class="form-control"
id="{{id}}" (focus)="openCloseDatepicker()" disabled="{{disabledInput}}" />
        <span id="openCloseDatePicker" class="input-group-addon"
(click)="openCloseDatepicker()">
                <span class="glyphicon-calendar glyphicon"></span>
            </span>
      </div>

      <div class="dp-popup" *ngIf="showDatepicker">
        <datepicker [startingDay]="1" [startingDay]="dt" [minDate]="min"  [(ngModel)]="dt"
(selectionDone)="onSelectionDone($event)"></datepicker>
      </div>
    </div>
```

datepicker.component.ts

```
import {Component, Input, EventEmitter, Output, OnChanges, SimpleChanges, ElementRef, OnInit}
from "@angular/core";
import {DatePipe} from "@angular/common";
import {NgModel} from "@angular/forms";
import * as moment from 'moment';

@Component({
  selector: 'custom-datepicker',
  templateUrl: 'datepicker.component.html',
  providers: [DatePipe, NgModel],
  host: {
    '(document:mousedown)': 'onClick($event)',
  }
})

export class DatepickerComponent implements OnChanges , OnInit{
  ngOnInit(): void {
    this.dt = null;
  }

  inputElement : ElementRef;
  dt: Date = null;
  showDatepicker: boolean = false;

  @Input() disabledInput : boolean = false;
```

```
@Input() disabledDatePicker: boolean = false;
@Input() value: string = null;
@Input() id: string;
@Input() min: Date = null;
@Input() max: Date = null;

@Output() dateModelChange = new EventEmitter();
constructor(el: ElementRef) {
  this.inputElement = el;
}

changedDate(){
  if(this.value === ''){
    this.dateModelChange.emit(null);
  }else if(this.value.split('/').length === 3){
    this.dateModelChange.emit(DatepickerComponent.convertToDate(this.value));
  }
}

clickOutSide(event : Event){
  if(this.inputElement.nativeElement !== event.target) {
    console.log('click outside', event);
  }
}

onClick(event) {
  if (!this.inputElement.nativeElement.contains(event.target)) {
    this.close();
  }
}
ngOnChanges(changes: SimpleChanges): void {
  if (this.value !== null && this.value !== undefined && this.value.length > 0) {
    this.value = null;
    this.dt  = null;
  }else {
    if(this.value !== null){
      this.dt = new Date(this.value);
      this.value = moment(this.value).format('MM/DD/YYYY');
    }
  }
}

private static transformDate(date: Date): string {
  return new DatePipe('pt-PT').transform(date, 'MM/dd/yyyy');
}

openCloseDatepicker(): void {
  if (!this.disabledDatePicker) {
    this.showDatepicker = !this.showDatepicker;
  }
}

open(): void {
  this.showDatepicker = true;
}

close(): void {
  this.showDatepicker = false;
}

private apply(): void {
```

```
    this.value = DatepickerComponent.transformDate(this.dt);
    this.dateModelChange.emit(this.dt);
  }

  onSelectionDone(event: Date): void {
    this.dt = event;
    this.apply();
    this.close();
  }

  onClickedOutside(event: Date): void {
    if (this.showDatepicker) {
      this.close();
    }
  }

  static convertToDate(val : string): Date {
    return new Date(val.replace('/','-'));
  }

}
```

Lea custom ngx-bootstrap datepicker + entrada en línea:
https://riptutorial.com/es/angular2/topic/10549/custom-ngx-bootstrap-datepicker-plus-entrada

Capítulo 30: Depuración de la aplicación mecanografiada Angular2 utilizando código de Visual Studio

Examples

Configuración de Launch.json para tu espacio de trabajo

1. Activar depuración desde el menú - ver> depurar
2. devuelve algún error durante el inicio de la depuración, muestra la notificación emergente y abre launch.json desde esta notificación emergente. Es solo porque launch.json no está configurado para su área de trabajo. Copie y pegue el siguiente código en launch.json // new launch.json
   tu viejo launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch Extension",
            "type": "extensionHost",
            "request": "launch",
            "runtimeExecutable": "${execPath}",
            "args": [
                "--extensionDevelopmentPath=${workspaceRoot}"
            ],
            "stopOnEntry": false,
            "sourceMaps": true,
            "outDir": "${workspaceRoot}/out",
            "preLaunchTask": "npm"
        }
    ]
}
```

Ahora actualiza tu launch.json como a continuación
nuevo launch.json
** // recuerda, por favor menciona tu ruta main.js **

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch",
            "type": "node",
            "request": "launch",
            "program": "${workspaceRoot}/app/main.js", // put your main.js path
            "stopOnEntry": false,
```

```
            "args": [],
            "cwd": "${workspaceRoot}",
            "preLaunchTask": null,
            "runtimeExecutable": null,
            "runtimeArgs": [
                "--nolazy"
            ],
            "env": {
                "NODE_ENV": "development"
            },
            "console": "internalConsole",
            "sourceMaps": false,
            "outDir": null
        },
        {
            "name": "Attach",
            "type": "node",
            "request": "attach",
            "port": 5858,
            "address": "localhost",
            "restart": false,
            "sourceMaps": false,
            "outDir": null,
            "localRoot": "${workspaceRoot}",
            "remoteRoot": null
        },
        {
            "name": "Attach to Process",
            "type": "node",
            "request": "attach",
            "processId": "${command.PickProcess}",
            "port": 5858,
            "sourceMaps": false,
            "outDir": null
        }
    ]
}
```

3. Ahora que la depuración está funcionando, muestra una ventana emergente de notificación para la depuración paso a paso

Lea Depuración de la aplicación mecanografiada Angular2 utilizando código de Visual Studio en línea: https://riptutorial.com/es/angular2/topic/7139/depuracion-de-la-aplicacion-mecanografiada-angular2-utilizando-codigo-de-visual-studio

Capítulo 31: Detectar eventos de redimensionamiento

Examples

Un componente que escucha en el evento de cambio de tamaño de la ventana.

Supongamos que tenemos un componente que se ocultará en un cierto ancho de ventana.

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

@Component({
  ...
  template: `
    <div>
      <p [hidden]="!visible" (window:resize)="onResize($event)" >Now you see me...</p>
      <p>now you dont!</p>
    </div>
  `
  ...
})
export class MyComponent {
  visible: boolean = false;
  breakpoint: number = 768;

  constructor() {
  }

  onResize(event) {
    const w = event.target.innerWidth;
    if (w >= this.breakpoint) {
      this.visible = true;
    } else {
      // whenever the window is less than 768, hide this component.
      this.visible = false;
    }
  }
}
```

Una etiqueta `p` en nuestra plantilla se ocultará cuando sea `visible` falso. `visible` cambiará el valor cada vez que se `onResize` controlador de eventos `onResize` . Su llamada se produce cada `window:resize` tiempo `window:resize` dispara un evento.

Lea Detectar eventos de redimensionamiento en línea:
https://riptutorial.com/es/angular2/topic/5276/detectar-eventos-de-redimensionamiento

Capítulo 32: Directivas

Sintaxis

- `<input [value]="value">` - Enlaza el `name` miembro de la clase de valor del atributo.
- `<div [attr.data-note]="note">` - Enlaza el atributo `data-note` a la `note` variable.
- `<p green></p>` - Directiva personalizada

Observaciones

La principal fuente de información sobre las directivas de Angular 2 es la documentación oficial https://angular.io/docs/ts/latest/guide/attribute-directives.html

Examples

Directiva de atributos

```
<div [class.active]="isActive"></div>

<span [style.color]="'red'"></span>

<p [attr.data-note]="'This is value for data-note attribute'">A lot of text here</p>
```

Componente es una directiva con plantilla.

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>Angular 2 App</h1>
    <p>Component is directive with template</p>
  `
})
export class AppComponent {
}
```

Directivas estructurales

```
<div *ngFor="let item of items">{{ item.description }}</div>

<span *ngIf="isVisible"></span>
```

Directiva personalizada

```
import {Directive, ElementRef, Renderer} from '@angular/core';

@Directive({
  selector: '[green]',
})

class GreenDirective {
  constructor(private _elementRef: ElementRef,
              private _renderer: Renderer) {
    _renderer.setElementStyle(_elementRef.nativeElement, 'color', 'green');
  }
}
```

Uso:

```
<p green>A lot of green text here</p>
```

* ngPara

form1.component.ts:

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

// Defines example component and associated template
@Component({
    selector: 'example',
    template: `
      <div *ngFor="let f of fruit"> {{f}} </div>
      <select required>
        <option *ngFor="let f of fruit" [value]="f"> {{f}} </option>
      </select>
    `
})

// Create a class for all functions, objects, and variables
export class ExampleComponent {
    // Array of fruit to be iterated by *ngFor
    fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];
}
```

Salida:

```
<div>Apples</div>
<div>Oranges</div>
<div>Bananas</div>
<div>Limes</div>
<div>Lemons</div>
<select required>
  <option value="Apples">Apples</option>
  <option value="Oranges">Oranges</option>
  <option value="Bananas">Bananas</option>
  <option value="Limes">Limes</option>
  <option value="Lemons">Lemons</option>
</select>
```

En su forma más simple, `*ngFor` tiene dos partes: `let` **`variableName`** `of` **`object/array`**

En el caso de la `fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons']; ,`

Las manzanas, las naranjas, etc. son los valores dentro de la variedad de `fruit` .

`[value]="f"` será igual a cada `fruit` actual (`f`) que `*ngFor` ha repetido.

A diferencia de AngularJS, Angular2 no ha continuado con el uso de `ng-options` para `<select>` y `ng-repeat` para todas las demás repeticiones generales.

`*ngFor` es muy similar a `ng-repeat` con una sintaxis ligeramente variada.

Referencias:

Angular2 | Visualización de datos

Angular2 | ngPara

Angular2 | Formas

Copia a la directiva del portapapeles

En este ejemplo, vamos a crear una directiva para copiar un texto en el portapapeles haciendo clic en un elemento

copy-text.directive.ts

```
import {
    Directive,
    Input,
    HostListener
} from "@angular/core";

@Directive({
    selector: '[text-copy]'
})
export class TextCopyDirective {

    // Parse attribute value into a 'text' variable
    @Input('text-copy') text:string;

    constructor() {
    }


    // The HostListener will listen to click events and run the below function, the
HostListener supports other standard events such as mouseenter, mouseleave etc.
    @HostListener('click') copyText() {

        // We need to create a dummy textarea with the text to be copied in the DOM
        var textArea = document.createElement("textarea");

        // Hide the textarea from actually showing
        textArea.style.position = 'fixed';
        textArea.style.top = '-999px';
        textArea.style.left = '-999px';
```

```
        textArea.style.width = '2em';
        textArea.style.height = '2em';
        textArea.style.padding = '0';
        textArea.style.border = 'none';
        textArea.style.outline = 'none';
        textArea.style.boxShadow = 'none';
        textArea.style.background = 'transparent';

        // Set the texarea's content to our value defined in our [text-copy] attribute
        textArea.value = this.text;
        document.body.appendChild(textArea);

        // This will select the textarea
        textArea.select();

        try {
            // Most modern browsers support execCommand('copy'|'cut'|'paste'), if it doesn't
it should throw an error
            var successful = document.execCommand('copy');
            var msg = successful ? 'successful' : 'unsuccessful';
            // Let the user know the text has been copied, e.g toast, alert etc.
            console.log(msg);
        } catch (err) {
            // Tell the user copying is not supported and give alternative, e.g alert window
with the text to copy
            console.log('unable to copy');
        }

        // Finally we remove the textarea from the DOM
        document.body.removeChild(textArea);
    }
}

export const TEXT_COPY_DIRECTIVES = [TextCopyDirective];
```

some-page.component.html

Recuerde inyectar TEXT_COPY_DIRECTIVES en la matriz de directivas de su componente

```
...
    <!-- Insert variable as the attribute's value, let textToBeCopied = 'http://facebook.com/'
-->
    <button [text-copy]="textToBeCopied">Copy URL</button>
    <button [text-copy]="'https://www.google.com/'">Copy URL</button>
...
```

Probando una directiva personalizada

Dada una directiva que resalta texto en eventos del mouse.

```
import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({ selector: '[appHighlight]' })
export class HighlightDirective {
  @Input('appHighlight') // tslint:disable-line no-input-rename
  highlightColor: string;
```

```
  constructor(private el: ElementRef) { }

  @HostListener('mouseenter')
  onMouseEnter() {
    this.highlight(this.highlightColor || 'red');
  }

  @HostListener('mouseleave')
  onMouseLeave() {
    this.highlight(null);
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```

Se puede probar así

```
import { ComponentFixture, ComponentFixtureAutoDetect, TestBed } from '@angular/core/testing';

import { Component } from '@angular/core';
import { HighlightDirective } from './highlight.directive';

@Component({
  selector: 'app-test-container',
  template: `
    <div>
      <span id="red" appHighlight>red text</span>
      <span id="green" [appHighlight]="'green'">green text</span>
      <span id="no">no color</span>
    </div>
  `
})
class ContainerComponent { }

const mouseEvents = {
  get enter() {
    const mouseenter = document.createEvent('MouseEvent');
    mouseenter.initEvent('mouseenter', true, true);
    return mouseenter;
  },
  get leave() {
    const mouseleave = document.createEvent('MouseEvent');
    mouseleave.initEvent('mouseleave', true, true);
    return mouseleave;
  },
};

describe('HighlightDirective', () => {
  let fixture: ComponentFixture<ContainerComponent>;
  let container: ContainerComponent;
  let element: HTMLElement;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [ContainerComponent, HighlightDirective],
      providers: [
        { provide: ComponentFixtureAutoDetect, useValue: true },
      ],
```

```
    });

    fixture = TestBed.createComponent(ContainerComponent);
    // fixture.detectChanges(); // without the provider
    container = fixture.componentInstance;
    element = fixture.nativeElement;
  });

  it('should set background-color to empty when mouse leaves with directive without
arguments', () => {
    const targetElement = <HTMLSpanElement>element.querySelector('#red');

    targetElement.dispatchEvent(mouseEvents.leave);
    expect(targetElement.style.backgroundColor).toEqual('');
  });

  it('should set background-color to empty when mouse leaves with directive with arguments',
() => {
    const targetElement = <HTMLSpanElement>element.querySelector('#green');

    targetElement.dispatchEvent(mouseEvents.leave);
    expect(targetElement.style.backgroundColor).toEqual('');
  });

  it('should set background-color red with no args passed', () => {
    const targetElement = <HTMLSpanElement>element.querySelector('#red');

    targetElement.dispatchEvent(mouseEvents.enter);
    expect(targetElement.style.backgroundColor).toEqual('red');
  });

  it('should set background-color green when passing green parameter', () => {
    const targetElement = <HTMLSpanElement>element.querySelector('#green');

    targetElement.dispatchEvent(mouseEvents.enter);
    expect(targetElement.style.backgroundColor).toEqual('green');
  });
});
```

Lea Directivas en línea: https://riptutorial.com/es/angular2/topic/2202/directivas

Capítulo 33: Directivas y componentes: @Input @Output

Sintaxis

1. Enlace unidireccional del componente principal al componente anidado: [propertyName]
2. Enlace unidireccional de componente anidado a componente principal: (propertyName)
3. Enlace bidireccional (también conocido como notación de caja de banana): [(propertyName)]

Examples

Ejemplo de entrada

@input es útil para enlazar datos entre componentes

Primero, importalo en tu componente

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

Luego, agregue la entrada como una propiedad de su clase de componente

```
@Input() car: any;
```

Digamos que el selector de su componente es 'auto-componente', cuando llama al componente, agregue el atributo 'automóvil'

```
<car-component [car]="car"></car-component>
```

Ahora se puede acceder a su automóvil como un atributo en su objeto (this.car)

Ejemplo completo:

1. car.entity.ts

```
export class CarEntity {
   constructor(public brand : string, public color : string) {
   }
}
```

2. car.component.ts

```
import { Component, Input } from '@angular/core';
import {CarEntity} from "./car.entity";

@Component({
```

```
    selector: 'car-component',
    template: require('./templates/car.html'),
})

export class CarComponent {
    @Input() car: CarEntity;

    constructor() {
        console.log('gros');
    }
}
```

3. garage.component.ts

```
import { Component } from '@angular/core';
import {CarEntity} from "./car.entity";
import {CarComponent} from "./car.component";

@Component({
    selector: 'garage',
    template: require('./templates/garage.html'),
    directives: [CarComponent]
})

export class GarageComponent {
    public cars : Array<CarEntity>;

    constructor() {
        var carOne : CarEntity = new CarEntity('renault', 'blue');
        var carTwo : CarEntity = new CarEntity('fiat', 'green');
        var carThree : CarEntity = new CarEntity('citroen', 'yellow');
        this.cars = [carOne, carTwo, carThree];
    }
}
```

4. garage.html

```
<div *ngFor="let car of cars">
<car-component [car]="car"></car-component>
</div>
```

5. car.html

```
<div>
    <span>{{ car.brand }}</span> |
    <span>{{ car.color }}</span>
</div>
```

Angular2 @Input y @Output en un componente anidado

Una directiva de botón que acepta un `@Input()` para especificar un límite de clic hasta que el botón se deshabilite. El componente principal puede escuchar un evento que se emitirá cuando se alcance el límite de clics a través de `@Output` :

```
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
    selector: 'limited-button',
    template: `<button (click)="onClick()"
                       [disabled]="disabled">
                   <ng-content></ng-content>
               </button>`,
    directives: []
})

export class LimitedButton {
    @Input() clickLimit: number;
    @Output() limitReached: EventEmitter<number> = new EventEmitter();

    disabled: boolean = false;

    private clickCount: number = 0;

    onClick() {
        this.clickCount++;
        if (this.clickCount === this.clickLimit) {
            this.disabled = true;
            this.limitReached.emit(this.clickCount);
        }
    }
}
```

Componente principal que utiliza la directiva Button y alerta un mensaje cuando se alcanza el límite de clics:

```
import { Component } from '@angular/core';
import { LimitedButton } from './limited-button.component';

@Component({
    selector: 'my-parent-component',
    template: `<limited-button [clickLimit]="2"
                               (limitReached)="onLimitReached($event)">
                   You can only click me twice
               </limited-button>`,
    directives: [LimitedButton]
})

export class MyParentComponent {
    onLimitReached(clickCount: number) {
        alert('Button disabled after ' + clickCount + ' clicks.');
    }
}
```

Angular2 @Input con datos asíncronos

A veces es necesario obtener datos de forma asíncrona antes de pasarlos a un componente secundario para usarlos. Si el componente hijo intenta usar los datos antes de que se hayan recibido, generará un error. Puede usar `ngOnChanges` para detectar cambios en `@Input` s de un componente y esperar hasta que se definan antes de actuar sobre ellos.

Componente padre con llamada asíncrona a un punto final

```
import { Component, OnChanges, OnInit } from '@angular/core';
import { Http, Response } from '@angular/http';
import { ChildComponent } from './child.component';

@Component ({
    selector : 'parent-component',
    template : `
        <child-component [data]="asyncData"></child-component>
    `
})
export class ParentComponent {

    asyncData : any;

    constructor(
        private _http : Http
    ){}

    ngOnInit () {
        this._http.get('some.url')
            .map(this.extractData)
            .subscribe(this.handleData)
            .catch(this.handleError);
    }

    extractData (res:Response) {
        let body = res.json();
        return body.data || { };
    }

    handleData (data:any) {
        this.asyncData = data;
    }

    handleError (error:any) {
        console.error(error);
    }
}
```

Componente hijo que tiene datos asíncronos como entrada.

Este componente hijo toma los datos asíncronos como entrada. Por lo tanto, debe esperar a que los datos existan antes de usarlos. Usamos ngOnChanges, que se activa cada vez que cambia la entrada de un componente, verificamos si existen los datos y los utilizamos si es así. Observe que la plantilla para el niño no se mostrará si una propiedad que se basa en los datos que se están pasando no es verdadera.

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

@Component ({
    selector : 'child-component',
    template : `
        <p *ngIf="doesDataExist">Hello child</p>
    `
})
export class ChildComponent {

    doesDataExist: boolean = false;

    @Input('data') data : any;

    // Runs whenever component @Inputs change
    ngOnChanges () {
        // Check if the data exists before using it
        if (this.data) {
            this.useData(data);
        {
    }

    // contrived example to assign data to reliesOnData
    useData (data) {
        this.doesDataExist = true;
    }
}
```

Lea Directivas y componentes: @Input @Output en línea:
https://riptutorial.com/es/angular2/topic/3046/directivas-y-componentes---input--output

Capítulo 34: Directivas y servicios comúnmente incorporados.

Introducción

@ angular / común: directivas y servicios que se necesitan comúnmente @ angular / core - la estructura del núcleo angular

Examples

Clase de ubicación

La ubicación es un servicio que las aplicaciones pueden usar para interactuar con la URL de un navegador. Dependiendo de la LocationStrategy que se use, la ubicación permanecerá en la ruta de la URL o en el segmento de hash de la URL.

La ubicación es responsable de normalizar la URL contra la base href de la aplicación.

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

@Component({
    selector: 'app-component'
})
class AppCmp {

  constructor(_location: Location) {

    //Changes the browsers URL to the normalized version of the given URL,
    //and pushes a new item onto the platform's history.
    _location.go('/foo');

  }

  backClicked() {
    //Navigates back in the platform's history.
    this._location.back();
  }

  forwardClicked() {
    //Navigates forward in the platform's history.
    this._location.back();
  }
}
```

AsyncPipe

La tubería asíncrona se suscribe a un Observable o Promesa y devuelve el último valor que ha emitido. Cuando se emite un nuevo valor, el conducto asíncrono marca el componente que se

debe verificar para detectar cambios. Cuando el componente se destruye, la tubería asíncrona se cancela automáticamente para evitar posibles pérdidas de memoria.

```
@Component({
  selector: 'async-observable-pipe',
  template: '<div><code>observable|async</code>: Time: {{ time | async }}</div>'
})
export class AsyncObservablePipeComponent {
  time = new Observable<string>((observer: Subscriber<string>) => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

Visualizando la versión angular2 utilizada en tu proyecto.

Para mostrar la versión actual, podemos usar **VERSION** del paquete @ angular / core.

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

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>
  <h2>Current Version: {{ver}}</h2>
  `,
})
export class AppComponent  {
   name = 'Angular2';
   ver = VERSION.full;
}
```

Pipa de la moneda

La canalización de moneda le permite trabajar con sus datos como números regulares pero mostrarlos con el formato de moneda estándar (símbolo de moneda, decimales, etc.) en la vista.

```
@Component({
  selector: 'currency-pipe',
  template: `<div>
    <p>A: {{myMoney | currency:'USD':false}}</p>
    <p>B: {{yourMoney | currency:'USD':true:'4.2-2'}}</p>
  </div>`
})
export class CurrencyPipeComponent {
  myMoney: number = 100000.653;
  yourMoney: number = 5.3495;
}
```

El tubo lleva tres parámetros opcionales:

- **currencyCode** : le permite especificar el código de moneda ISO 4217.
- **symbolDisplay** : booleano que indica si se debe utilizar el símbolo de moneda
- **digitInfo** : le permite especificar cómo deben mostrarse los lugares decimales.

Más documentación sobre la tubería de divisas:

https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html

Lea Directivas y servicios comúnmente incorporados. en línea: https://riptutorial.com/es/angular2/topic/8252/directivas-y-servicios-comunmente-incorporados-

Capítulo 35: Directrices de atributos para afectar el valor de las propiedades en el nodo host mediante el decorador @HostBinding.

Examples

@HostBinding

El decorador @HostBinding nos permite programar un valor de propiedad en el elemento host de la directiva. Funciona de manera similar a un enlace de propiedad definido en una plantilla, excepto que se dirige específicamente al elemento host. El enlace se verifica para cada ciclo de detección de cambios, por lo que puede cambiar dinámicamente si se desea. Por ejemplo, digamos que queremos crear una directiva para los botones que agrega dinámicamente una clase cuando presionamos sobre ella. Eso podría parecer algo como:

```
import { Directive, HostBinding, HostListener } from '@angular/core';

@Directive({
  selector: '[appButtonPress]'
})
export class ButtonPressDirective {
  @HostBinding('attr.role') role = 'button';
  @HostBinding('class.pressed') isPressed: boolean;

  @HostListener('mousedown') hasPressed() {
    this.isPressed = true;
  }
  @HostListener('mouseup') hasReleased() {
    this.isPressed = false;
  }
}
```

Tenga en cuenta que para ambos casos de uso de @HostBinding estamos pasando un valor de cadena por el cual queremos afectar la propiedad. Si no suministramos una cadena al decorador, entonces se usará el nombre del miembro de la clase. En el primer @HostBinding, estamos configurando de forma estática el atributo de función para el botón. Para el segundo ejemplo, la clase presionada se aplicará cuando isPressed sea verdadero

Lea Directrices de atributos para afectar el valor de las propiedades en el nodo host mediante el decorador @HostBinding. en línea: https://riptutorial.com/es/angular2/topic/9455/directrices-de-atributos-para-afectar-el-valor-de-las-propiedades-en-el-nodo-host-mediante-el-decorador--hostbinding-

Capítulo 36: Diseño de material angular

Examples

Md2Seleccionar

Componente :

```
<md2-select [(ngModel)]="item" (change)="change($event)" [disabled]="disabled">
<md2-option *ngFor="let i of items" [value]="i.value" [disabled]="i.disabled">
{{i.name}}</md2-option>
</md2-select>
```

Seleccione permitir al usuario seleccionar la opción de las opciones.

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

Md2Tooltip

La información sobre herramientas es una directiva, permite al usuario mostrar texto de sugerencias mientras el mouse del usuario se desplaza sobre un elemento.

Una información sobre herramientas tendría el siguiente marcado.

```
<span tooltip-direction="left" tooltip="On the Left!">Left</span>
<button tooltip="some message"
    tooltip-position="below"
    tooltip-delay="1000">Hover Me
</button>
```

Md2Toast

Toast es un servicio que muestra notificaciones en la vista.

Crea y muestra una simple noticia de brindis.

```
import {Md2Toast} from 'md2/toast/toast';

@Component({
 selector: "..."
})

export class ... {

...
constructor(private toast: Md2Toast) { }
toastMe() {
```

```
this.toast.show('Toast message...');

---  or  ---

this.toast.show('Toast message...', 1000);
}

...

}
```

Md2Datepicker

Selector de fecha permite al usuario seleccionar fecha y hora.

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

ver para más detalles aquí

Md2Accordion y Md2Collapse

Md2Collapse : Colapso es una directiva, permite al usuario cambiar la visibilidad de la sección.

Ejemplos

Un colapso tendría el siguiente marcado.

```
<div [collapse]="isCollapsed">
  Lorum Ipsum Content
</div>
```

Md2Accordion : El acordeón permite al usuario cambiar la visibilidad de las múltiples secciones.

Ejemplos

Un acordeón tendría el siguiente marcado.

```
<md2-accordion [multiple]="multiple">
  <md2-accordion-tab *ngFor="let tab of accordions"
                     [header]="tab.title"
                     [active]="tab.active"
                     [disabled]="tab.disabled">
    {{tab.content}}
  </md2-accordion-tab>
  <md2-accordion-tab>
    <md2-accordion-header>Custom Header</md2-accordion-header>
    test content
  </md2-accordion-tab>
</md2-accordion>
```

Lea Diseño de material angular en línea: https://riptutorial.com/es/angular2/topic/10005/diseno-de-material-angular

Capítulo 37: Dropzone en Angular2

Examples

Zona de descenso

Biblioteca de envoltorios angular 2 para Dropzone.

npm install angular2-dropzone-wrapper --save-dev

Cargue el módulo para su módulo de aplicación

```
import { DropzoneModule } from 'angular2-dropzone-wrapper';
import { DropzoneConfigInterface } from 'angular2-dropzone-wrapper';

const DROPZONE_CONFIG: DropzoneConfigInterface = {
  // Change this to your upload POST address:
  server: 'https://example.com/post',
  maxFilesize: 10,
  acceptedFiles: 'image/*'
};

@NgModule({
  ...
  imports: [
    ...
    DropzoneModule.forRoot(DROPZONE_CONFIG)
  ]
})
```

USO DE COMPONENTES

Simplemente reemplace el elemento que normalmente se pasaría a Dropzone con el componente Dropzone.

```
<dropzone [config]="config" [message]="'Click or drag images here to upload'"
(error)="onUploadError($event)" (success)="onUploadSuccess($event)"></dropzone>
```

Crear componente dropzone

```
import {Component} from '@angular/core';
@Component({
    selector: 'app-new-media',
    templateUrl: './dropzone.component.html',
    styleUrls: ['./dropzone.component.scss']
})
export class DropZoneComponent {


    onUploadError(args: any) {
        console.log('onUploadError:', args);
    }
```

```
    onUploadSuccess(args: any) {
        console.log('onUploadSuccess:', args);
    }
}
```

Lea Dropzone en Angular2 en línea: https://riptutorial.com/es/angular2/topic/10010/dropzone-en-angular2

Capítulo 38: Ejemplo para rutas como / route / subruta para urls estáticas

Examples

Ejemplo de ruta básica con árbol de rutas secundarias

app.module.ts

```
import {routes} from "./app.routes";

@NgModule({
   declarations: [AppComponent],
   imports: [BrowserModule, mainModule.forRoot(), RouterModule.forRoot(routes)],
   providers: [],
   bootstrap: [AppComponent]
})

export class AppModule { }
```

app.routes.ts

```
import { Routes } from '@angular/router';
import {SubTreeRoutes} from "./subTree/subTreeRoutes.routes";

export const routes: Routes = [
  ...SubTreeRoutes,
  { path: '',  redirectTo: 'home', pathMatch: 'full'}
];
```

subTreeRoutes.ts

```
import {Route} from '@angular/router';
import {exampleComponent} from "./example.component";

export const SubTreeRoutes: Route[] = [
  {
    path: 'subTree',
    children: [
      {path: '',component: exampleComponent}
    ]
  }
];
```

Lea Ejemplo para rutas como / route / subruta para urls estáticas en línea: https://riptutorial.com/es/angular2/topic/8910/ejemplo-para-rutas-como---route---subruta-para-urls-estaticas

Capítulo 39: Ejemplos de componentes avanzados

Observaciones

Recuerda que Angular 2 tiene que ver con la responsabilidad singular. No importa lo pequeño que sea su componente, dedique una lógica separada para cada componente. Ya sea un botón, un enlace de anclaje elegante, un encabezado de diálogo o incluso un subelemento de sidenav.

Examples

Selector de imágenes con vista previa

En este ejemplo, vamos a crear un selector de imágenes que muestra una vista previa de la imagen antes de subirla. La vista previa también admite archivos de arrastrar y soltar en la entrada. En este ejemplo, solo cubriré la carga de archivos individuales, pero puede hacer un pequeño esfuerzo para que la carga de múltiples archivos funcione.

image-preview.html

Este es el diseño html de nuestra vista previa de imagen

```
<!-- Icon as placeholder when no file picked -->
<i class="material-icons">cloud_upload</i>

<!-- file input, accepts images only. Detect when file has been picked/changed with Angular's
native (change) event listener -->
<input type="file" accept="image/*" (change)="updateSource($event)">

<!-- img placeholder when a file has been picked. shows only when 'source' is not empty -->
<img *ngIf="source" [src]="source" src="">
```

image-preview.ts

Este es el archivo principal para nuestro componente `<image-preview>`

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

@Component({
    selector: 'image-preview',
    styleUrls: [ './image-preview.css' ],
    templateUrl: './image-preview.html'
})
export class MtImagePreviewComponent {
```

```
    // Emit an event when a file has been picked. Here we return the file itself
    @Output() onChange: EventEmitter<File> = new EventEmitter<File>();

    constructor() {}

    // If the input has changed(file picked) we project the file into the img previewer
    updateSource($event: Event) {
        // We access he file with $event.target['files'][0]
        this.projectImage($event.target['files'][0]);
    }

    // Uses FileReader to read the file from the input
    source:string = '';
    projectImage(file: File) {
        let reader = new FileReader;
        // TODO: Define type of 'e'
        reader.onload = (e: any) => {
            // Simply set e.target.result as our <img> src in the layout
            this.source = e.target.result;
            this.onChange.emit(file);
        };
        // This will process our file and get it's attributes/data
        reader.readAsDataURL(file);
    }
}
```

another.component.html

```
<form (ngSubmit)="submitPhoto()">
    <image-preview (onChange)="getFile($event)"></image-preview>
    <button type="submit">Upload</button>
</form>
```

Y eso es. Mucho más fácil de lo que era en AngularJS 1.x. Realmente hice este componente basado en una versión anterior que hice en AngularJS 1.5.5.

Filtrar los valores de la tabla por la entrada

Importe `ReactiveFormsModule` , y luego

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormControl } from '@angular/forms';
import { Subscription } from 'rxjs';

@Component({
  selector: 'component',
  template: `
    <input [formControl]="control" />
    <div *ngFor="let item of content">
      {{item.id}} - {{item.name}}
    </div>
  `
})
export class MyComponent implements OnInit, OnDestroy {

  public control = new FormControl('');
```

```
  public content: { id: number; name: string; }[];

  private originalContent = [
    { id: 1, name: 'abc' },
    { id: 2, name: 'abce' },
    { id: 3, name: 'ced' }
  ];

  private subscription: Subscription;

  public ngOnInit() {
    this.subscription = this.control.valueChanges.subscribe(value => {
      this.content = this.originalContent.filter(item => item.name.startsWith(value));
    });
  }

  public ngOnDestroy() {
    this.subscription.unsubscribe();
  }

}
```

Lea Ejemplos de componentes avanzados en línea:
https://riptutorial.com/es/angular2/topic/5597/ejemplos-de-componentes-avanzados

Capítulo 40: Encuadernación de datos Angular2

Examples

@Entrada()

Componente principal: Inicializar listas de usuarios.

```
@Component({
  selector: 'parent-component',
  template: '<div>
               <child-component [users]="users"></child-component>
             </div>'
})
export class ParentComponent implements OnInit{
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com');
    users.push(new User('B', 'B', 'B@gmail.com');
    users.push(new User('C', 'C', 'C@gmail.com');
  }
}
```

El componente hijo obtiene al usuario del componente principal con Input ()

```
@Component({
selector: 'child-component',
  template: '<div>
                  <table *ngIf="users !== null">
                    <thead>
                         <th>Name</th>
                         <th>FName</th>
                         <th>Email</th>
                    </thead>
                    <tbody>
                        <tr *ngFor="let user of users">
                            <td>{{user.name}}</td>
                            <td>{{user.fname}}</td>
                            <td>{{user.email}}</td>
                        </tr>
                    </tbody>
                  </table>

             </div>',
})
export class ChildComponent {
  @Input() users : List<User> = null;
}
```

```
export class User {
  name : string;
  fname : string;
  email : string;

  constructor(_name : string, _fname : string, _email : string){
     this.name = _name;
     this.fname = _fname;
     this.email = _email;
  }
}
```

Lea Encuadernación de datos Angular2 en línea:
https://riptutorial.com/es/angular2/topic/9036/encuadernacion-de-datos-angular2

Capítulo 41: Enrutamiento

Examples

Enrutamiento básico

El enrutador permite la navegación de una vista a otra según las interacciones del usuario con la aplicación.

Los siguientes son los pasos para implementar el enrutamiento básico en Angular 2:

Precaución básica : Asegúrate de tener la etiqueta.

```
    <base href='/'>
```

como el primer hijo debajo de su etiqueta de cabecera en su archivo index.html. Esta etiqueta dice que su carpeta de aplicaciones es la raíz de la aplicación. Angular 2 entonces sabría organizar sus enlaces.

El primer paso es verificar si está señalando las dependencias de enrutamiento correctas / más recientes en package.json -

```
"dependencies": {
  ......
  "@angular/router": "3.0.0-beta.1",
  ......
}
```

El segundo paso es definir la ruta según su definición de clase:

```
class Route {
  path : string
  pathMatch : 'full'|'prefix'
  component : Type|string
  .........
}
```

En un archivo de rutas (`route/routes.ts`), importe todos los componentes que necesita configurar para diferentes rutas de enrutamiento. La ruta vacía significa que la vista se carga de forma predeterminada. ":" en la ruta indica un parámetro dinámico pasado al componente cargado.

Las rutas se ponen a disposición de la aplicación a través de la inyección de dependencia. El método ProviderRouter se llama con RouterConfig como parámetro para que pueda ser inyectado a los componentes para llamar a tareas específicas de enrutamiento.

```
import { provideRouter, RouterConfig } from '@angular/router';
import { BarDetailComponent } from '../components/bar-detail.component';
import { DashboardComponent } from '../components/dashboard.component';
```

```
import { LoginComponent } from '../components/login.component';
import { SignupComponent } from '../components/signup.component';

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup',   component: SignupComponent }
];

export const APP_ROUTER_PROVIDER = [provideRouter(appRoutes)];
```

El tercer paso es arrancar el proveedor de ruta.

En sus `main.ts` (Puede ser cualquier nombre. Básicamente, debe estar definido en el archivo principal en systemjs.config)

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './components/app.component';
import { APP_ROUTER_PROVIDER } from "./routes/routes";

bootstrap(AppComponent, [ APP_ROUTER_PROVIDER ]).catch(err => console.error(err));
```

El cuarto paso es cargar / mostrar los componentes del enrutador según la ruta a la que se accede. La directiva se utiliza para indicar a angular dónde cargar el componente. Para utilizar importar los ROUTER_DIRECTIVES.

```
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'demo-app',
  template: `
    .....................................
    <div>
      <router-outlet></router-outlet>
    </div>
    .....................................
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
```

Quinto paso es enlazar las otras rutas. De forma predeterminada, RouterOutlet cargará el componente para el cual se especifica la ruta vacía en el RouterConfig. La directiva RouterLink se usa con la etiqueta de anclaje html para cargar los componentes adjuntos a las rutas. RouterLink genera el atributo href que se utiliza para generar enlaces. Por ejemplo:

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

@Component({
  selector: 'demo-app',
  template: `
    <a [routerLink]="['/login']">Login</a>
```

```
    <a [routerLink]="['/signup']">Signup</a>
    <a [routerLink]="['/dashboard']">Dashboard</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }
```

Ahora, estamos bien con el enrutamiento a la ruta estática. RouterLink también admite la ruta dinámica al pasar parámetros adicionales junto con la ruta.

importar {Componente} desde '@ angular / core'; importe {ROUTER_DIRECTIVES} desde '@ angular / router';

```
@Component({
  selector: 'demo-app',
  template: `
        <ul>
          <li *ngFor="let bar of bars | async">
            <a [routerLink]="['/bars', bar.id]">
              {{bar.name}}
            </a>
          </li>
        </ul>
    <div>
      <router-outlet></router-outlet>
    </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }
```

RouterLink toma una matriz donde el primer elemento es la ruta de enrutamiento y los elementos subsiguientes son para los parámetros de enrutamiento dinámico.

Rutas infantiles

A veces tiene sentido anidar vistas o rutas entre sí. Por ejemplo, en el panel de control desea varias vistas secundarias, similares a las pestañas pero implementadas a través del sistema de enrutamiento, para mostrar los proyectos, contactos y mensajes de los usuarios. Para soportar tales escenarios, el enrutador nos permite definir rutas secundarias.

Primero ajustamos nuestro `RouterConfig` desde arriba y agregamos las rutas secundarias:

```
import { ProjectsComponent } from '../components/projects.component';
import { MessagesComponent} from '../components/messages.component';

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent,
    children: [
```

```
      { path: '', redirectTo: 'projects', pathMatch: 'full' },
      { path: 'projects', component: 'ProjectsComponent' },
      { path: 'messages', component: 'MessagesComponent' }
    ] },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup',   component: SignupComponent }
];
```

Ahora que tenemos nuestras rutas secundarias definidas, tenemos que asegurarnos de que esas rutas secundarias se puedan mostrar dentro de nuestro `DashboardComponent` , ya que ahí es donde hemos agregado las secundarias. Anteriormente aprendimos que los componentes se muestran en una etiqueta `<router-outlet></router-outlet>` . Similar declaramos otro `RouterOutlet` en el `DashboardComponent` :

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

@Component({
  selector: 'dashboard',
  template: `
    <a [routerLink]="['projects']">Projects</a>
    <a [routerLink]="['messages']">Messages</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `
})
export class DashboardComponent { }
```

Como puede ver, hemos agregado otro `RouterOutlet` en el que se mostrarán las rutas secundarias. Por lo general, la ruta con una ruta vacía se mostrará, sin embargo, configuramos una redirección a la ruta de los `projects` , porque queremos que se muestre inmediatamente cuando se carga la ruta del `dashboard` . Dicho esto, necesitamos una ruta vacía, de lo contrario obtendrá un error como este:

```
Cannot match any routes: 'dashboard'
```

Entonces, al agregar la ruta *vacía* , es decir, una ruta con una ruta vacía, hemos definido un punto de entrada para el enrutador.

ResolveData

Este ejemplo le mostrará cómo puede resolver los datos obtenidos de un servicio antes de representar la vista de su aplicación.

Utiliza angular / router 3.0.0-beta.2 al momento de escribir

users.service.ts

```
...
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';
```

```
import { User } from './user.ts';



@Injectable()
export class UsersService {

    constructor(public http:Http) {}

    /**
     * Returns all users
     * @returns {Observable<User[]>}
     */
    index():Observable<User[]> {

        return this.http.get('http://mywebsite.com/api/v1/users')
            .map((res:Response) => res.json());
    }

    /**
     * Returns a user by ID
     * @param id
     * @returns {Observable<User>}
     */
    get(id:number|string):Observable<User> {

        return this.http.get('http://mywebsite.com/api/v1/users/' + id)
            .map((res:Response) => res.json());
    }
}
```

usuarios.resolver.ts

```
...
import { UsersService } from './users.service.ts';
import { Observable } from 'rxjs/Rx';
import {
    Resolve,
    ActivatedRouteSnapshot,
    RouterStateSnapshot
} from "@angular/router";



@Injectable()
export class UsersResolver implements Resolve<User[] | User> {

    // Inject UsersService into the resolver
    constructor(private service:UsersService) {}

    resolve(route:ActivatedRouteSnapshot, state:RouterStateSnapshot):Observable<User[] | User>
{
        // If userId param exists in current URL, return a single user, else return all users
        // Uses brackets notation to access `id` to suppress editor warning, may use dot
notation if you create an interface extending ActivatedRoute with an optional id? attribute
        if (route.params['id']) return this.service.get(route.params['id']);
        return this.service.index();
    }
}
```

users.component.ts

Este es un componente de la página con una lista de todos los usuarios. Funcionará de manera similar para el componente de la página de detalles del usuario, reemplazará `data.users` con `data.user` o cualquier tecla definida en *app.routes.ts* (ver más abajo)

```
...
import { ActivatedRoute} from "@angular/router";



@Component(...)
export class UsersComponent {

    users:User[];


    constructor(route: ActivatedRoute) {
        route.data.subscribe(data => {
            // data['Match key defined in RouterConfig, see below']
            this.users = data.users;
        });
    }

    /**
     * It is not required to unsubscribe from the resolver as Angular's HTTP
     * automatically completes the subscription when data is received from the server
     */
}
```

app.routes.ts

```
...
import { UsersResolver } from './resolvers/users.resolver';



export const routes:RouterConfig = <RouterConfig>[
    ...
    {
        path: 'user/:id',
        component: UserComponent,
        resolve: {
            // hence data.user in UserComponent
            user: UsersResolver
        }
    },
    {
        path: 'users',
        component: UsersComponent,
        resolve: {
            // hence data.users in UsersComponent, note the pluralisation
            users: UsersResolver
        }
    },
    ...
]
...
```

app.resolver.ts

Opcionalmente, agrupar varios resolutores juntos.

IMPORTANTE: los *servicios que se utilizan en la resolución deben importarse primero o obtendrá un 'No hay proveedor para ... Error de resolución'. Recuerde que estos servicios estarán disponibles en todo el mundo y que ya no tendrá que declararlos en ningún* `providers` *de componentes. Asegúrese de darse de baja de cualquier suscripción para evitar la pérdida de memoria*

```
...
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

export const ROUTE_RESOLVERS = [
    ...,
    UsersService,
    UsersResolver
]
```

main.browser.ts

Los resolvers tienen que ser inyectados durante el arranque.

```
...
import {bootstrap} from '@angular/platform-browser-dynamic';
import { ROUTE_RESOLVERS } from './app.resolver';

bootstrap(<Type>App, [
    ...
    ...ROUTE_RESOLVERS
])
.catch(err => console.error(err));
```

Enrutamiento con niños

Al contrario de la documentación original, encontré que esta es la forma de anidar correctamente las rutas de los niños dentro del archivo app.routing.ts o app.module.ts (según sus preferencias). Este enfoque funciona cuando se usa WebPack o SystemJS.

El siguiente ejemplo muestra las rutas de inicio, inicio / contador y inicio / contador / obtener datos. Las primeras y últimas rutas son ejemplos de redirecciones. Finalmente, al final del ejemplo es una forma adecuada de exportar la Ruta a importar en un archivo separado. Por ej. app.module.ts

Para explicar con más detalle, Angular requiere que tenga una ruta sin ruta en la matriz secundaria que incluye el componente principal, para representar la ruta principal. Es un poco confuso, pero si piensa en una URL en blanco para una ruta secundaria, esencialmente sería igual a la misma URL que la ruta principal.

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

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

import { HomeComponent } from "./components/home/home.component";
import { FetchDataComponent } from "./components/fetchdata/fetchdata.component";
import { CounterComponent } from "./components/counter/counter.component";

const appRoutes: Routes = [
    {
        path: "",
        redirectTo: "home",
        pathMatch: "full"
    },
    {
        path: "home",
        children: [
            {
                path: "",
                component: HomeComponent
            },
            {
                path: "counter",
                children: [
                    {
                        path: "",
                        component: CounterComponent
                    },
                    {
                        path: "fetch-data",
                        component: FetchDataComponent
                    }
                ]
            }
        ]
    },
    {
        path: "**",
        redirectTo: "home"
    }
];

@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

Gran ejemplo y descripción a través de Siraj

Lea Enrutamiento en línea: https://riptutorial.com/es/angular2/topic/2334/enrutamiento

Capítulo 42: Enrutamiento (3.0.0+)

Observaciones

Hay algunos trucos más que podemos hacer con el enrutador (como restringir el acceso), pero se pueden cubrir en un tutorial separado.

Si necesita una nueva ruta, simplemente modifique `app.routes.ts` y siga los siguientes pasos:

1. Importar el componente
2. Añadir a la matriz de `routes` . Asegúrese de incluir una nueva `path` y `component` .

Examples

Bootstrapping

Ahora que las rutas están definidas, debemos informar a nuestra aplicación sobre las rutas. Para hacer esto, reinicie al proveedor que exportamos en el ejemplo anterior.

Encuentre su configuración de arranque (debería estar en `main.ts` , pero **su millaje puede variar**).

```
//main.ts

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

//Import the App component (root component)
import { App } from './app/app';

//Also import the app routes
import { APP_ROUTES_PROVIDER } from './app/app.routes';

bootstrap(App, [
  APP_ROUTES_PROVIDER,
])
.catch(err => console.error(err));
```

Configurando enrutador de salida

Ahora que el enrutador está configurado y nuestra aplicación sabe cómo manejar las rutas, debemos mostrar los componentes reales que configuramos.

Para hacerlo, configure su plantilla / archivo HTML para su componente de **nivel superior (aplicación) de la siguiente** manera:

```
//app.ts

import {Component} from '@angular/core';
import {Router, ROUTER_DIRECTIVES} from '@angular/router';
```

```
@Component({
    selector: 'app',
    templateUrl: 'app.html',
    styleUrls: ['app.css'],
    directives: [
        ROUTER_DIRECTIVES,
    ]
})
export class App {
    constructor() {
    }
}

<!-- app.html -->

<!-- All of your 'views' will go here -->
<router-outlet></router-outlet>
```

El elemento `<router-outlet></router-outlet>` cambiará el contenido dado la ruta. Otro buen aspecto de este elemento es que *no* tiene que ser el único elemento en su HTML.

Por ejemplo: Digamos que quería una barra de herramientas en cada página que se mantiene constante entre las rutas, de forma similar al aspecto del Desbordamiento de pila. Puede anidar el `<router-outlet>` bajo los elementos para que solo cambien ciertas partes de la página.

Cambio de rutas (utilizando plantillas y directivas)

Ahora que las rutas están configuradas, necesitamos alguna forma de cambiar realmente las rutas.

Este ejemplo mostrará cómo cambiar las rutas usando la plantilla, pero es posible cambiar las rutas en TypeScript.

Aquí hay un ejemplo (sin enlace):

```
<a routerLink="/home">Home</a>
```

Si el usuario hace clic en ese enlace, se dirigirá a `/home` . El enrutador sabe cómo manejar `/home` , por lo que mostrará el componente de `Home` .

Aquí hay un ejemplo con enlace de datos:

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

Lo que requeriría que existiera una matriz llamada `links` , así que agregue esto a `app.ts` :

```
public links[] = [
    'home',
    'login'
]
```

Esto recorrerá la matriz y agregará un elemento `<a>` con la directiva `routerLink` = el valor del elemento actual en la matriz, creando esto:

```
 <a routerLink="home">home</a>
 <a routerLink="login">login</a>
```

Esto es particularmente útil si tiene muchos enlaces, o tal vez los enlaces necesitan ser cambiados constantemente. Dejamos que Angular maneje el ajetreado trabajo de agregar enlaces con solo darle la información que requiere.

En este momento, los `links[]` son estáticos, pero es posible alimentarlos con datos de otra fuente.

Configuración de las rutas

NOTA: Este ejemplo se basa en la versión 3.0.0.-beta.2 de @ angular / router. Al momento de escribir esto, esta es la última versión del enrutador.

Para usar el enrutador, defina rutas en un nuevo archivo de TypeScript como el

```
//app.routes.ts

import {provideRouter} from '@angular/router';

import {Home} from './routes/home/home';
import {Profile} from './routes/profile/profile';

export const routes = [
    {path: '', redirectTo: 'home'},
    {path: 'home', component: Home},
    {path: 'login', component: Login},
];

export const APP_ROUTES_PROVIDER = provideRouter(routes);
```

En la primera línea, importamos `provideRouter` para que podamos informarle a nuestra aplicación cuáles son las rutas durante la fase de arranque.

`Home` y `Profile` son solo dos componentes como ejemplo. Deberá importar cada `Component` que necesite como ruta.

Luego, exporta la matriz de rutas.

`path` : la ruta al componente. **NO NECESITA USAR '/ .....'** Angular lo hará automáticamente

`component` : el componente a cargar cuando se accede a la ruta

`redirectTo` : *opcional* . Si necesita redirigir automáticamente a un usuario cuando accede a una ruta en particular, proporcione esto.

Finalmente, exportamos el enrutador configurado. `provideRouter` devolverá un proveedor al que podemos impulsar para que nuestra aplicación sepa cómo manejar cada ruta.

Control de acceso desde o hacia una ruta

El enrutador angular predeterminado permite navegar incondicionalmente hacia y desde cualquier ruta. Este no es siempre el comportamiento deseado.

En un escenario en el que se le puede permitir condicionalmente a un usuario navegar hacia o desde una ruta, se puede usar un **Guardia de Ruta** para restringir este comportamiento.

Si su escenario se ajusta a uno de los siguientes, considere usar un protector de ruta,

- El usuario debe estar autenticado para navegar al componente de destino.
- El usuario debe estar autorizado para navegar al componente de destino.
- El componente requiere una solicitud asíncrona antes de la inicialización.
- Componente requiere la entrada del usuario antes de navegar fuera de.

Cómo funcionan los guardias de ruta

Los guardias de ruta funcionan devolviendo un valor booleano para controlar el comportamiento de la navegación del enrutador. Si se devuelve *true* , el enrutador continuará con la navegación al componente de destino. Si se devuelve *falso* , el enrutador denegará la navegación al componente de destino.

Interfaces de guardia de ruta

El enrutador soporta múltiples interfaces de guarda:

- *CanActivate* : se produce entre la navegación de ruta.
- *CanActivateChild* : ocurre entre la navegación de la ruta a una ruta secundaria.
- *CanDeactivate* : ocurre cuando se navega lejos de la ruta actual.
- *CanLoad* : se produce entre la navegación de ruta a un módulo de función cargado de forma asíncrona.
- *Resolver* : se utiliza para realizar la recuperación de datos antes de la activación de la ruta.

Estas interfaces se pueden implementar en su guardia para otorgar o eliminar el acceso a ciertos procesos de navegación.

Guardias de ruta sincrónica frente a asíncrona

Las protecciones de ruta permiten que las operaciones síncronas y asíncronas controlen condicionalmente la navegación.

Guardia de ruta sincrónica

Un protector de ruta síncrono devuelve un valor booleano, como al calcular un resultado inmediato, para controlar de forma condicional la navegación.

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

@Injectable()
export class SynchronousGuard implements CanActivate {
  canActivate() {
    console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

Guardia de ruta asíncrona

Para un comportamiento más complejo, un protector de ruta puede bloquear de forma asíncrona la navegación. Un protector de ruta asíncrono puede devolver una Observable o Promesa.

Esto es útil para situaciones como esperar la entrada del usuario para responder una pregunta, esperar para guardar con éxito los cambios en el servidor o esperar a recibir datos de un servidor remoto.

```
import { Injectable }     from '@angular/core';
import { CanActivate, Router, ActivatedRouteSnapshot, RouterStateSnapshot } from
'@angular/router';
import { Observable }     from 'rxjs/Rx';
import { MockAuthenticationService } from './authentication/authentication.service';

@Injectable()
export class AsynchronousGuard implements CanActivate {
    constructor(private router: Router, private auth: MockAuthenticationService) {}

    canActivate(route:ActivatedRouteSnapshot,
state:RouterStateSnapshot):Observable<boolean>|boolean {
        this.auth.subscribe((authenticated) => {
            if (authenticated) {
                return true;
            }
            this.router.navigateByUrl('/login');
            return false;
        });
    }
}
```

Añadir guardia a la configuración de la ruta

Archivo *app.routes*

Las rutas protegidas se han podido `canActivate` a `Guard`

```
import { provideRouter, Router, RouterConfig, CanActivate } from '@angular/router';

//components
import { LoginComponent } from './login/login.component';
import { DashboardComponent } from './dashboard/dashboard.component';

export const routes: RouterConfig = [
    { path: 'login', component: LoginComponent },
    { path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }
]
```

Exporte el **APP_ROUTER_PROVIDERS** para ser usado en la aplicación bootstrap

```
export const APP_ROUTER_PROVIDERS = [
    AuthGuard,
    provideRouter(routes)
];
```

Usa Guard en la aplicación bootstrap

Archivo *main.ts* (o *boot.ts*)

Considere los ejemplos anteriores:

1. **Crear la guardia** (donde se crea la Guardia) y
2. **Agregar protección a la configuración de la ruta** , (donde la Guardia está configurada para la ruta, luego se exporta **APP_ROUTER_PROVIDERS**),
   Podemos acoplar el bootstrap a Guard de la siguiente manera

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { provide } from '@angular/core';

import { APP_ROUTER_PROVIDERS } from './app.routes';
import { AppComponent } from './app.component';

bootstrap(AppComponent, [
    APP_ROUTER_PROVIDERS
])
.then(success => console.log(`Bootstrap success`))
.catch(error => console.log(error));
```

Uso de solucionadores y guardias

Estamos utilizando una protección de nivel superior en nuestra configuración de ruta para capturar al usuario actual en la carga de la primera página, y una resolución para almacenar el valor del `currentUser` , que es nuestro usuario autenticado desde el backend.

Una versión simplificada de nuestra implementación tiene el siguiente aspecto:

Aquí está nuestra ruta de nivel superior:

```
export const routes = [
{
  path: 'Dash',
  pathMatch : 'prefix',
  component: DashCmp,
  canActivate: [AuthGuard],
  resolve: {
      currentUser: CurrentUserResolver
  },
  children: [...[
    path: '',
    component: ProfileCmp,
    resolve: {
        currentUser: currentUser
    }
  ]]
  }
];
```

Aquí está nuestro servicio de `AuthService`

```
import { Injectable } from '@angular/core';
import { Http, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/do';

@Injectable()
export class AuthService {
  constructor(http: Http) {
    this.http = http;

    let headers = new Headers({ 'Content-Type': 'application/json' });
    this.options = new RequestOptions({ headers: headers });
  }
  fetchCurrentUser() {
    return this.http.get('/api/users/me')
      .map(res => res.json())
      .do(val => this.currentUser = val);
 }
}
```

Aquí está nuestro `AuthGuard` :

```
import { Injectable } from '@angular/core';
import { CanActivate } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  canActivate(route, state) {
    return Observable
      .merge(this.auth.fetchCurrentUser(), Observable.of(true))
      .filter(x => x == true);
  }
```

```
}
```

Aquí está nuestro `CurrentUserResolver` :

```
import { Injectable } from '@angular/core';
import { Resolve } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
export class CurrentUserResolver implements Resolve {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  resolve(route, state) {
    return this.auth.currentUser;
  }
}
```

Lea Enrutamiento (3.0.0+) en línea: https://riptutorial.com/es/angular2/topic/1208/enrutamiento--3-0-0plus-

Capítulo 43: examen de la unidad

Examples

Prueba unitaria basica

archivo componente

```
@Component({
  selector: 'example-test-compnent',
  template: '<div>
                  <div>{{user.name}}</div>
                  <div>{{user.fname}}</div>
                  <div>{{user.email}}</div>
             </div>'
})

export class ExampleTestComponent implements OnInit{

    let user :User = null;
    ngOnInit(): void {
       this.user.name = 'name';
       this.user.fname= 'fname';
       this.user.email= 'email';
    }

}
```

Archivo de prueba

```
describe('Example unit test component', () => {
  let component: ExampleTestComponent ;
  let fixture: ComponentFixture<ExampleTestComponent >;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ExampleTestComponent]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(ExampleTestComponent );
    component = fixture.componentInstance;
    fixture.detectChanges();
  });


  it('ngOnInit should change user object values', () => {
    expect(component.user).toBeNull(); // check that user is null on initialize
    component.ngOnInit(); // run ngOnInit

    expect(component.user.name).toEqual('name');
    expect(component.user.fname).toEqual('fname');
```

```
    expect(component.user.email).toEqual('email');
  });
});
```

Lea examen de la unidad en línea: https://riptutorial.com/es/angular2/topic/8955/examen-de-la-unidad

Capítulo 44: Ganchos de ciclo de vida

Observaciones

Disponibilidad de eventos

`AfterViewInit` y `AfterViewChecked` solo están disponibles **en Componentes** , y **no en Directivas** .

Orden de eventos

- `OnChanges` (varias veces)
- `OnInit` (una vez)
- `DoCheck` (varias veces)
- `AfterContentInit` (una vez)
- `AfterContentChecked` (varias veces)
- `AfterViewInit` (una vez) (solo componente)
- `AfterViewChecked` (varias veces) (solo componente)
- `OnDestroy` (una vez)

Otras lecturas

- Documentación angular - Ganchos de ciclo de vida

Examples

OnInit

Se activa cuando se han inicializado las propiedades de componente o directiva.

(Antes de las de las directivas infantiles).

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

@Component({
    selector: 'so-oninit-component',
    templateUrl: 'oninit-component.html',
    styleUrls: ['oninit-component.']
})
class OnInitComponent implements OnInit {

    ngOnInit(): void {
        console.log('Component is ready !');
    }
}
```

EnDestroy

Se activa cuando se destruye la instancia del componente o directiva.

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

@Component({
    selector: 'so-ondestroy-component',
    templateUrl: 'ondestroy-component.html',
    styleUrls: ['ondestroy-component.']
})
class OnDestroyComponent implements OnDestroy {

    ngOnDestroy(): void {
        console.log('Component was destroyed !');
    }
}
```

OnChanges

Se activa cuando una o más de las propiedades del componente o directiva han sido cambiadas.

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

@Component({
    selector: 'so-onchanges-component',
    templateUrl: 'onchanges-component.html',
    styleUrls: ['onchanges-component.']
})
class OnChangesComponent implements OnChanges {
    @Input() name: string;
    message: string;

    ngOnChanges(changes: SimpleChanges): void {
        console.log(changes);
    }
}
```

En el evento de cambio se registrará

```
name: {
    currentValue: 'new name value',
    previousValue: 'old name value'
},
message: {
    currentValue: 'new message value',
    previousValue: 'old message value'
}
```

AfterContentInit

Incendio una vez finalizada la inicialización del contenido del componente o directiva.

(Justo después de OnInit)

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

@Component({
    selector: 'so-aftercontentinit-component',
    templateUrl: 'aftercontentinit-component.html',
    styleUrls: ['aftercontentinit-component.']
})
class AfterContentInitComponent implements AfterContentInit {

    ngAfterContentInit(): void {
        console.log('Component content have been loaded!');
    }
}
```

AfterContentChecked

Disparar después de que la vista haya sido completamente inicializada.

(Solo disponible para componentes)

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

@Component({
    selector: 'so-aftercontentchecked-component',
    templateUrl: 'aftercontentchecked-component.html',
    styleUrls: ['aftercontentchecked-component.']
})
class AfterContentCheckedComponent implements AfterContentChecked {

    ngAfterContentChecked(): void {
        console.log('Component content have been checked!');
    }
}
```

AfterViewInit

Se activa después de inicializar la vista de componente y cualquiera de sus vistas secundarias. Este es un útil gancho de ciclo de vida para complementos fuera del ecosistema de Angular 2. Por ejemplo, podría usar este método para inicializar un selector de fecha jQuery en función del marcado que Angular 2 ha renderizado.

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

@Component({
    selector: 'so-afterviewinit-component',
    templateUrl: 'afterviewinit-component.html',
    styleUrls: ['afterviewinit-component.']
})
class AfterViewInitComponent implements AfterViewInit {

    ngAfterViewInit(): void {
        console.log('This event fire after the content init have been loaded!');
    }
}
```

AfterViewChecked

Disparo después de que se haya verificado la vista, del componente.

(Solo disponible para componentes)

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

@Component({
    selector: 'so-afterviewchecked-component',
    templateUrl: 'afterviewchecked-component.html',
    styleUrls: ['afterviewchecked-component.']
})
class AfterViewCheckedComponent implements AfterViewChecked {

    ngAfterViewChecked(): void {
        console.log('This event fire after the content have been checked!');
    }
}
```

DoCheck

Permite escuchar cambios solo en propiedades especificadas

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

@Component({
    selector: 'so-docheck-component',
    templateUrl: 'docheck-component.html',
    styleUrls: ['docheck-component.']
})
class DoCheckComponent implements DoCheck {
    @Input() elements: string[];
    differ: any;
    ngDoCheck(): void {
        // get value for elements property
        const changes = this.differ.diff(this.elements);

        if (changes) {
            changes.forEachAddedItem(res => console.log('Added', r.item));
            changes.forEachRemovedItem(r => console.log('Removed', r.item));
        }
    }
}
```

Lea Ganchos de ciclo de vida en línea: https://riptutorial.com/es/angular2/topic/2935/ganchos-de-ciclo-de-vida

Capítulo 45: Instalar complementos de terceros con angular-cli@1.0.0-beta.10

Observaciones

Es posible instalar otras bibliotecas siguiendo este enfoque, sin embargo, puede ser necesario especificar el tipo de módulo, el archivo principal y la extensión predeterminada.

```
'lodash': {
  format: 'cjs',
  defaultExtension: 'js',
  main: 'index.js'
}
```

```
'moment': {
  main: 'moment.js'
}
```

Examples

Añadiendo librería jquery en proyecto angular-cli.

1. Instalar jquery a través de npm:

```
npm install jquery --save
```

   Instalar mecanografías para la biblioteca:

Para agregar mecanografías para una biblioteca, haga lo siguiente:

```
typings install jquery --global --save
```

---

2. Agregue jquery al archivo angular-cli-build.js a la matriz vendorNpmFiles:

   Esto es necesario para que el sistema de compilación recoja el archivo. Después de configurar el angular-cli-build.js debería verse así:

   Examine los `node_modules` y busque los archivos y carpetas que desea agregar a la carpeta del proveedor.

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');

module.exports = function(defaults) {
  return new Angular2App(defaults, {
```

```
    vendorNpmFiles: [
      // ...
      'jquery/dist/*.js'


    ]
  });
};
```

---

3. Configure las asignaciones de SystemJS para saber dónde buscar jquery:

   La configuración de SystemJS se encuentra en system-config.ts y, una vez que se haya realizado la configuración personalizada, la sección relacionada debería verse así:

```
/** Map relative paths to URLs. */
const map: any = {
  'jquery': 'vendor/jquery'
};

/** User packages configuration. */
const packages: any = {

// no need to add anything here for jquery

};
```

---

4. En su src / index.html agregue esta línea

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

   Sus otras opciones son:

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

o

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

y

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

---

5. Importación y uso de la biblioteca jquery en los archivos de origen de su proyecto:

   Importe la biblioteca jquery en sus archivos fuente .ts como este:

```
declare var $:any;

@Component({
```

```
})
export class YourComponent {
  ngOnInit() {
    $.("button").click(function(){
       // now you can DO, what ever you want
     });
     console.log();
  }
}
```

Si siguió los pasos correctamente, ahora debería tener una biblioteca jQuery trabajando en su proyecto. ¡Disfrutar!

Agregar una biblioteca de terceros que no tenga mecanografía

> ¡Note, esto es solo para angular-cli hasta la versión 1.0.0-beta.10!

Algunas bibliotecas o complementos pueden no tener tipografías. Sin estos, TypeScript no puede escribirlos y, por lo tanto, provoca errores de compilación. Estas bibliotecas se pueden seguir utilizando de forma diferente a los módulos importados.

1. Incluya una referencia de script para la biblioteca en su página (`index.html`)

   ```
   <script src="//cdn.somewhe.re/lib.min.js" type="text/javascript"></script>
   <script src="/local/path/to/lib.min.js" type="text/javascript"></script>
   ```

   - Estos scripts deben agregar un global (por ejemplo, `THREE` , `mapbox` , `$` , etc.) o adjuntarlos a un global

2. En el componente que los requiere, use `declare` para inicializar una variable que coincida con el nombre global utilizado por la biblioteca. Esto permite que TypeScript sepa que ya se ha inicializado. [1]

   ```
   declare var <globalname>: any;
   ```

   Algunas bibliotecas se adjuntan a la `window` , que deberían extenderse para poder acceder a la aplicación.

   ```
   interface WindowIntercom extends Window { Intercom: any; }
   declare var window: WindowIntercom;
   ```

3. Utilice la lib en sus componentes según sea necesario.

   ```
   @Component { ... }
   export class AppComponent implements AfterViewInit {
       ...
       ngAfterViewInit() {
           var geometry = new THREE.BoxGeometry( 1, 1, 1 );
           window.Intercom('boot', { ... }
       }
   }
   ```

- NOTA: Algunas bibliotecas pueden interactuar con el DOM y deben usarse en el método de ciclo de vida del componente apropiado.

Lea Instalar complementos de terceros con angular-cli@1.0.0-beta.10 en línea: https://riptutorial.com/es/angular2/topic/2328/instalar-complementos-de-terceros-con-angular-cli-1-0-0-beta-10

Capítulo 46: Interacciones de los componentes

Sintaxis

- `<element [variableName]="value"></element> //Declaring input to child when using @Input() method.`
- `<element (childOutput)="parentFunction($event)"></element> //Declaring output from child when using @Output() method.`
- `@Output() pageNumberClicked = new EventEmitter(); //Used for sending output data from child component when using @Output() method.`
- `this.pageNumberClicked.emit(pageNum); //Used to trigger data output from child component. when using @Output() method.`
- `@ViewChild(ComponentClass) //Property decorator is required when using ViewChild.`

Parámetros

Nombre	Valor
pageCount	Se utiliza para indicar el número de páginas que se crearán al componente secundario.
pageNumberClicked	Nombre de la variable de salida en el componente hijo.
páginaCambiado	Función en el componente principal que escucha la salida de componentes secundarios.

Examples

Interacción padre - hijo usando las propiedades @Input y @Output

Tenemos un componente DataListComponent que muestra los datos que obtenemos de un servicio. DataListComponent también tiene un PagerComponent como hijo.

PagerComponent crea una lista de números de página en función del número total de páginas que recibe de DataListComponent. PagerComponent también le permite a DataListComponent saber cuándo el usuario hace clic en cualquier número de página a través de la propiedad Salida.

```
import { Component, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from './dataList.service';
import { PagerComponent } from './pager.component';

@Component({
    selector: 'datalist',
    template: `
        <table>
        <tr *ngFor="let person of personsData">
```

```
            <td>{{person.name}}</td>
            <td>{{person.surname}}</td>
        </tr>
        </table>

        <pager [pageCount]="pageCount" (pageNumberClicked)="pageChanged($event)"></pager>
    `
})
export class DataListComponent {
    private personsData = null;
    private pageCount: number;

    constructor(private dataListService: DataListService) {
        var response = this.dataListService.getData(1); //Request first page from the service
        this.personsData = response.persons;
        this.pageCount = response.totalCount / 10;//We will show 10 records per page.
    }

    pageChanged(pageNumber: number){
        var response = this.dataListService.getData(pageNumber); //Request data from the
service with new page number
        this.personsData = response.persons;
    }
}

@NgModule({
    imports: [CommonModule],
    exports: [],
    declarations: [DataListComponent, PagerComponent],
    providers: [DataListService],
})
export class DataListModule { }
```

PagerComponent enumera todos los números de página. Establecemos un evento de clic en cada uno de ellos para que podamos informar al padre sobre el número de página seleccionada.

```
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
    selector: 'pager',
    template: `
    <div id="pager-wrapper">
        <span *ngFor="#page of pageCount" (click)="pageClicked(page)">{{page}}</span>
    </div>
    `
})
export class PagerComponent {
    @Input() pageCount: number;
    @Output() pageNumberClicked = new EventEmitter();
    constructor() { }

    pageClicked(pageNum){
        this.pageNumberClicked.emit(pageNum); //Send clicked page number as output
    }
}
```

Interacción padre-hijo usando ViewChild

Viewchild ofrece una forma de interacción de padres a hijos. No hay comentarios o resultados del niño cuando se usa ViewChild.

Tenemos un DataListComponent que muestra cierta información. DataListComponent tiene PagerComponent como hijo. Cuando el usuario realiza una búsqueda en DataListComponent, obtiene datos de un servicio y le pide a PagerComponent que actualice el diseño de la paginación según el nuevo número de páginas.

```
import { Component, NgModule, ViewChild } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from './dataList.service';
import { PagerComponent } from './pager.component';

@Component({
    selector: 'datalist',
    template: `<input type='text' [(ngModel)]="searchText" />
                <button (click)="getData()">Search</button>
        <table>
        <tr *ngFor="let person of personsData">
            <td>{{person.name}}</td>
            <td>{{person.surname}}</td>
        </tr>
        </table>

        <pager></pager>
    `
})
export class DataListComponent {
    private personsData = null;
    private searchText: string;

    @ViewChild(PagerComponent)
    private pagerComponent: PagerComponent;

    constructor(private dataListService: DataListService) {}

    getData(){
        var response = this.dataListService.getData(this.searchText);
        this.personsData = response.data;
        this.pagerComponent.setPaging(this.personsData / 10); //Show 10 records per page
    }
}

@NgModule({
    imports: [CommonModule],
    exports: [],
    declarations: [DataListComponent, PagerComponent],
    providers: [DataListService],
})
export class DataListModule { }
```

De esta manera puede llamar a funciones definidas en componentes secundarios.

El componente hijo no está disponible hasta que el componente padre se representa. Intentar acceder al niño antes que a los padres `AfterViewInit` life cyle hook causará una excepción.

Interacción bidireccional padre-hijo a través de un servicio.

Servicio que se utiliza para la comunicación:

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

@Injectable()
export class ComponentCommunicationService {

    private componentChangeSource = new Subject();
    private newDateCreationSource = new Subject<Date>();

    componentChanged$ = this.componentChangeSource.asObservable();
    dateCreated$ = this.newDateCreationSource.asObservable();

    refresh() {
        this.componentChangeSource.next();
    }

    broadcastDate(date: Date) {
        this.newDateCreationSource.next(date);
    }
}
```

Componente principal:

```
import { Component, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
    selector: 'parent',
    template: `
    <button (click)="refreshSubsribed()">Refresh</button>
    <h1>Last date from child received: {{lastDate}}</h1>
    <child-component></child-component>
    `
})
export class ParentComponent implements OnInit {

    lastDate: Date;
    constructor(private communicationService: ComponentCommunicationService) { }

    ngOnInit() {
        this.communicationService.dateCreated$.subscribe(newDate => {
            this.lastDate = newDate;
        });
    }

    refreshSubsribed() {
        this.communicationService.refresh();
    }
}
```

Componente hijo:

```
import { Component, OnInit, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
```

```
    selector: 'child-component',
    template: `
    <h1>Last refresh from parent: {{lastRefreshed}}</h1>
    <button (click)="sendNewDate()">Send new date</button>
    `
})
export class ChildComponent implements OnInit {

    lastRefreshed: Date;
    constructor(private communicationService: ComponentCommunicationService) { }

    ngOnInit() {
        this.communicationService.componentChanged$.subscribe(event => {
            this.onRefresh();
        });
    }

    sendNewDate() {
        this.communicationService.broadcastDate(new Date());
    }

    onRefresh() {
        this.lastRefreshed = new Date();
    }
}
```

AppModule:

```
@NgModule({
    declarations: [
        ParentComponent,
        ChildComponent
    ],
    providers: [ComponentCommunicationService],
    bootstrap: [AppComponent] // not included in the example
})
export class AppModule {}
```

Lea Interacciones de los componentes en línea:
https://riptutorial.com/es/angular2/topic/7400/interacciones-de-los-componentes

Capítulo 47: Interacciones de los componentes

Introducción

Compartir información entre diferentes directivas y componentes.

Examples

Pase datos de padre a hijo con enlace de entrada

HeroChildComponent tiene dos propiedades de entrada, típicamente adornadas con decoraciones @Input.

```
import { Component, Input } from '@angular/core';
import { Hero } from './hero';
@Component({
  selector: 'hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @Input('master') masterName: string;
}
```

La propiedad de entrada de intercepción cambia con un setter

Use un establecedor de propiedades de entrada para interceptar y actuar sobre un valor del padre.

El definidor de la propiedad de entrada de nombre en el NameChildComponent secundario recorta el espacio en blanco de un nombre y reemplaza un valor vacío con texto predeterminado.

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'name-child',
  template: '<h3>"{{name}}"</h3>'
})
export class NameChildComponent {
  private _name = '';
  @Input()
  set name(name: string) {
    this._name = (name && name.trim()) || '<no name set>';
  }
  get name(): string { return this._name; }
}
```

Aquí está el NameParentComponent que muestra las variaciones de nombre, incluido un nombre con todos los espacios:

```
import { Component } from '@angular/core';
@Component({
  selector: 'name-parent',
  template: `
  <h2>Master controls {{names.length}} names</h2>
  <name-child *ngFor="let name of names" [name]="name"></name-child>
  `
})
export class NameParentComponent {
  // Displays 'Mr. IQ', '<no name set>', 'Bombasto'
  names = ['Mr. IQ', '   ', '  Bombasto  '];
}
```

Padre escucha para evento infantil

El componente hijo expone una propiedad EventEmitter con la que emite eventos cuando ocurre algo. El padre se une a esa propiedad de evento y reacciona a esos eventos.

La propiedad EventEmitter del niño es una propiedad de salida, típicamente adornada con una decoración @Output como se ve en este VoterComponent:

```
import { Component, EventEmitter, Input, Output } from '@angular/core';
@Component({
  selector: 'my-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)"  [disabled]="voted">Agree</button>
    <button (click)="vote(false)" [disabled]="voted">Disagree</button>
  `
})
export class VoterComponent {
  @Input()  name: string;
  @Output() onVoted = new EventEmitter<boolean>();
  voted = false;
  vote(agreed: boolean) {
    this.onVoted.emit(agreed);
    this.voted = true;
  }
}
```

Al hacer clic en un botón se activa la emisión de un verdadero o falso (la carga útil booleana).

El principal VoteTakerComponent vincula un controlador de eventos (onVoted) que responde a la carga útil del evento secundario ($ evento) y actualiza un contador.

```
import { Component }      from '@angular/core';
@Component({
  selector: 'vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
    <my-voter *ngFor="let voter of voters"
      [name]="voter"
```

```
      (onVoted)="onVoted($event)">
    </my-voter>
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];
  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}
```

El padre interactúa con el niño a través de la variable local

Un componente principal no puede utilizar el enlace de datos para leer propiedades secundarias o invocar métodos secundarios. Podemos hacer ambas cosas creando una variable de referencia de plantilla para el elemento secundario y luego hacer referencia a esa variable dentro de la plantilla principal como se ve en el siguiente ejemplo.

Tenemos un hijo CountdownTimerComponent que repetidamente cuenta atrás hasta cero y lanza un cohete. Tiene métodos de inicio y detención que controlan el reloj y muestra un mensaje de estado de cuenta regresiva en su propia plantilla.

```
import { Component, OnDestroy, OnInit } from '@angular/core';
@Component({
  selector: 'countdown-timer',
  template: '<p>{{message}}</p>'
})
export class CountdownTimerComponent implements OnInit, OnDestroy {
  intervalId = 0;
  message = '';
  seconds = 11;
  clearTimer() { clearInterval(this.intervalId); }
  ngOnInit()    { this.start(); }
  ngOnDestroy() { this.clearTimer(); }
  start() { this.countDown(); }
  stop()  {
    this.clearTimer();
    this.message = `Holding at T-${this.seconds} seconds`;
  }
  private countDown() {
    this.clearTimer();
    this.intervalId = window.setInterval(() => {
      this.seconds -= 1;
      if (this.seconds === 0) {
        this.message = 'Blast off!';
      } else {
        if (this.seconds < 0) { this.seconds = 10; } // reset
        this.message = `T-${this.seconds} seconds and counting`;
      }
    }, 1000);
  }
}
```

Veamos el CountdownLocalVarParentComponent que aloja el componente del temporizador.

```
import { Component }                from '@angular/core';
import { CountdownTimerComponent }  from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-lv',
  template: `
  <h3>Countdown to Liftoff (via local variable)</h3>
  <button (click)="timer.start()">Start</button>
  <button (click)="timer.stop()">Stop</button>
  <div class="seconds">{{timer.seconds}}</div>
  <countdown-timer #timer></countdown-timer>
  `,
  styleUrls: ['demo.css']
})
export class CountdownLocalVarParentComponent { }
```

El componente principal no puede enlazar datos a los métodos de inicio y parada del niño ni a su propiedad de segundos.

Podemos colocar una variable local (#timer) en la etiqueta () que representa el componente secundario. Eso nos da una referencia al componente secundario en sí mismo y la capacidad de acceder a cualquiera de sus propiedades o métodos desde la plantilla principal.

En este ejemplo, conectamos los botones principales al inicio y al final del niño y utilizamos la interpolación para mostrar la propiedad de los segundos del niño.

Aquí vemos al padre y al niño trabajando juntos.

Padre llama a un ViewChild

El enfoque de la variable local es simple y fácil. Pero está limitado porque el cableado padre-hijo debe hacerse completamente dentro de la plantilla padre. El componente principal en sí no tiene acceso al elemento secundario.

No podemos usar la técnica de variables locales si una instancia de la clase de componente principal debe leer o escribir valores de componente secundario o debe llamar a métodos de componente secundario.

Cuando la clase de componente principal requiere ese tipo de acceso, inyectamos el componente secundario en el principal como un ViewChild.

Ilustraremos esta técnica con el mismo ejemplo de temporizador de cuenta regresiva. No cambiaremos su apariencia o comportamiento. El niño CountdownTimerComponent también es el mismo.

Estamos cambiando de la variable local a la técnica ViewChild únicamente con fines de demostración. Aquí está el padre, CountdownViewChildParentComponent:

```
import { AfterViewInit, ViewChild } from '@angular/core';
import { Component }                from '@angular/core';
import { CountdownTimerComponent }  from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-vc',
  template: `
```

```
  <h3>Countdown to Liftoff (via ViewChild)</h3>
  <button (click)="start()">Start</button>
  <button (click)="stop()">Stop</button>
  <div class="seconds">{{ seconds() }}</div>
  <countdown-timer></countdown-timer>
  `,
  styleUrls: ['demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {
  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
  seconds() { return 0; }
  ngAfterViewInit() {
    // Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
    // but wait a tick first to avoid one-time devMode
    // unidirectional-data-flow-violation error
    setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }
  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}
```

Se necesita un poco más de trabajo para obtener la vista secundaria en la clase de componente principal.

Importamos referencias al decorador ViewChild y al gancho del ciclo de vida de AfterViewInit.

Inyectamos el hijo CountdownTimerComponent en la propiedad privada timerComponent a través de la decoración de la propiedad @ViewChild.

La variable local #timer desaparece de los metadatos del componente. En su lugar, enlazamos los botones con los propios métodos de inicio y detención del componente principal y presentamos los segundos de tic-tac en una interpolación alrededor del método de segundos del componente principal.

Estos métodos acceden directamente al componente del temporizador inyectado.

El gancho del ciclo de vida de ngAfterViewInit es una arruga importante. El componente del temporizador no está disponible hasta que Angular muestra la vista principal. Así que mostramos 0 segundos inicialmente.

Luego Angular llama al gancho del ciclo de vida de ngAfterViewInit, momento en el cual es demasiado tarde para actualizar la visualización de la cuenta principal de los segundos de la cuenta regresiva. La regla de flujo de datos unidireccional de Angular nos impide actualizar las vistas principales en el mismo ciclo. Tenemos que esperar un turno antes de poder mostrar los segundos.

Usamos setTimeout para esperar una marca y luego revisar el método de los segundos para que tome valores futuros del componente del temporizador.

Padres e hijos se comunican a través de un servicio.

Un componente padre y sus hijos comparten un servicio cuya interfaz permite la comunicación

bidireccional dentro de la familia.

El ámbito de la instancia de servicio es el componente principal y sus elementos secundarios. Los componentes fuera de este subárbol de componentes no tienen acceso al servicio ni a sus comunicaciones.

Este MissionService conecta el MissionControlComponent con varios hijos de AstronautComponent.

```
import { Injectable } from '@angular/core';
import { Subject }    from 'rxjs/Subject';
@Injectable()
export class MissionService {
  // Observable string sources
  private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // Observable string streams
  missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
  confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

MissionControlComponent proporciona la instancia del servicio que comparte con sus hijos (a través de la matriz de metadatos del proveedor) e inyecta esa instancia en sí misma a través de su constructor:

```
import { Component }          from '@angular/core';
import { MissionService }     from './mission.service';
@Component({
  selector: 'mission-control',
  template: `
    <h2>Mission Control</h2>
    <button (click)="announce()">Announce mission</button>
    <my-astronaut *ngFor="let astronaut of astronauts"
      [astronaut]="astronaut">
    </my-astronaut>
    <h3>History</h3>
    <ul>
      <li *ngFor="let event of history">{{event}}</li>
    </ul>
  `,
  providers: [MissionService]
})
export class MissionControlComponent {
  astronauts = ['Lovell', 'Swigert', 'Haise'];
  history: string[] = [];
  missions = ['Fly to the moon!',
              'Fly to mars!',
              'Fly to Vegas!'];
  nextMission = 0;
  constructor(private missionService: MissionService) {
```

```
    missionService.missionConfirmed$.subscribe(
      astronaut => {
        this.history.push(`${astronaut} confirmed the mission`);
      });
  }
  announce() {
    let mission = this.missions[this.nextMission++];
    this.missionService.announceMission(mission);
    this.history.push(`Mission "${mission}" announced`);
    if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
  }
}
```

El AstronautComponent también inyecta el servicio en su constructor. Cada AstronautComponent es un elemento secundario de MissionControlComponent y, por lo tanto, recibe la instancia de servicio de su padre:

```
import { Component, Input, OnDestroy } from '@angular/core';
import { MissionService } from './mission.service';
import { Subscription }   from 'rxjs/Subscription';
@Component({
  selector: 'my-astronaut',
  template: `
    <p>
      {{astronaut}}: <strong>{{mission}}</strong>
      <button
        (click)="confirm()"
        [disabled]="!announced || confirmed">
        Confirm
      </button>
    </p>
  `
})
export class AstronautComponent implements OnDestroy {
  @Input() astronaut: string;
  mission = '<no mission announced>';
  confirmed = false;
  announced = false;
  subscription: Subscription;
  constructor(private missionService: MissionService) {
    this.subscription = missionService.missionAnnounced$.subscribe(
      mission => {
        this.mission = mission;
        this.announced = true;
        this.confirmed = false;
    });
  }
  confirm() {
    this.confirmed = true;
    this.missionService.confirmMission(this.astronaut);
  }
  ngOnDestroy() {
    // prevent memory leak when component destroyed
    this.subscription.unsubscribe();
  }
}
```

Tenga en cuenta que capturamos la suscripción y anula la suscripción cuando se destruye el

AstronautComponent. Este es un paso de guardia de fuga de memoria. No hay riesgo real en esta aplicación porque la vida útil de un AstronautComponent es la misma que la vida misma de la aplicación. Eso no siempre sería cierto en una aplicación más compleja.

No agregamos esta protección a MissionControlComponent porque, como padre, controla la vida útil de MissionService. El registro de Historial demuestra que los mensajes viajan en ambas direcciones entre el MissionControlComponent padre y el hijo AstronautComponent, facilitado por el servicio:

Lea Interacciones de los componentes en línea:
https://riptutorial.com/es/angular2/topic/9454/interacciones-de-los-componentes

Capítulo 48: Interceptor Http

Observaciones

Lo que hacemos con la clase HttpServiceLayer es extender la clase Http desde angular y agregarle nuestra propia lógica.

Luego inyectamos esa clase en la clase bootstrap de la aplicación y le decimos a Angular que importamos la clase Http, en la parte posterior para insertar HttpServiceLayer.

En cualquier parte del código podemos simplemente importar

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

Pero nuestra clase será utilizada para cada llamada.

Examples

Clase simple Extendiendo la clase Http de Angular

```
import { Http, Request, RequestOptionsArgs, Response, RequestOptions, ConnectionBackend,
Headers } from '@angular/http';
import { Router } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/empty';
import 'rxjs/add/observable/throw';
import 'rxjs/add/operator/catch';
import { ApplicationConfiguration } from '../application-configuration/application-
configuration';

/**
* This class extends the Http class from angular and adds automaticaly the server URL(if in
development mode) and 2 headers by default:
* Headers added: 'Content-Type' and 'X-AUTH-TOKEN'.
* 'Content-Type' can be set in any othe service, and if set, it will NOT be overwritten in
this class any more.
*/
export class HttpServiceLayer extends Http {

  constructor(backend: ConnectionBackend, defaultOptions: RequestOptions, private _router:
Router, private appConfig: ApplicationConfiguration) {
    super(backend, defaultOptions);
  }

  request(url: string | Request, options?: RequestOptionsArgs): Observable<Response> {
    this.getRequestOptionArgs(options);
    return this.intercept(super.request(this.appConfig.getServerAdress() + url, options));
  }

  /**
  * This method checks if there are any headers added and if not created the headers map and
ads 'Content-Type' and 'X-AUTH-TOKEN'
```

```
  * 'Content-Type' is not overwritten if it is allready available in the headers map
  */
  getRequestOptionArgs(options?: RequestOptionsArgs): RequestOptionsArgs {
    if (options == null) {
      options = new RequestOptions();
    }
    if (options.headers == null) {
      options.headers = new Headers();
    }

    if (!options.headers.get('Content-Type')) {
      options.headers.append('Content-Type', 'application/json');
    }

    if (this.appConfig.getAuthToken() != null) {
      options.headers.append('X-AUTH-TOKEN', this.appConfig.getAuthToken());
    }

    return options;
  }

  /**
  * This method as the name sugests intercepts the request and checks if there are any errors.
  * If an error is present it will be checked what error there is and if it is a general one
then it will be handled here, otherwise, will be
  * thrown up in the service layers
  */
  intercept(observable: Observable<Response>): Observable<Response> {

    //  return observable;
    return observable.catch((err, source) => {
      if (err.status == 401) {
        this._router.navigate(['/login']);
        //return observable;
        return Observable.empty();
      } else {
        //return observable;
        return Observable.throw(err);
      }
    });
  }
}
```

Usando nuestra clase en lugar de Http de Angular

Después de extender la clase Http, necesitamos decirle a angular que use esta clase en lugar de la clase Http.

Para hacer esto, en nuestro módulo principal (o según las necesidades, solo un módulo en particular), debemos escribir en la sección de proveedores:

```
export function httpServiceFactory(xhrBackend: XHRBackend, requestOptions: RequestOptions,
router: Router, appConfig: ApplicationConfiguration) {
  return  new HttpServiceLayer(xhrBackend, requestOptions, router, appConfig);
}

import { HttpModule, Http, Request, RequestOptionsArgs, Response, XHRBackend, RequestOptions,
ConnectionBackend, Headers } from '@angular/http';
```

```
import { Router } from '@angular/router';

@NgModule({
  declarations: [ ... ],
  imports: [ ... ],
  exports: [ ... ],
  providers: [
    ApplicationConfiguration,
    {
      provide: Http,
      useFactory: httpServiceFactory,
      deps: [XHRBackend, RequestOptions, Router, ApplicationConfiguration]
}
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Nota: ApplicationConfiguration es solo un servicio que utilizo para guardar algunos valores durante la duración de la aplicación.

Simple HttpClient AuthToken Interceptor (Angular 4.3+)

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AuthHeaderInterceptor implements HttpInterceptor {

  constructor(private userService: UserService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    if (this.userService.isAuthenticated()) {
      req = req.clone({
        setHeaders: {
          Authorization:  `Bearer ${this.userService.token}`
        }
      });
    }
    return next.handle(req);
  }
}
```

Proporcionar Interceptor (some-module.module.ts)

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

Lea Interceptor Http en línea: https://riptutorial.com/es/angular2/topic/1413/interceptor-http

Capítulo 49: Inyección de Dependencia y Servicios.

Examples

Servicio de ejemplo

servicios / my.service.ts

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

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

El registro del proveedor de servicios en el método bootstrap hará que el servicio esté disponible globalmente.

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

En la versión RC5, el registro del proveedor de servicios global se puede realizar dentro del archivo del módulo. Para obtener una única instancia de su servicio para toda su aplicación, el servicio debe ser declarado en la lista de proveedores en el módulo de su aplicación.
app_module.ts

```
import { NgModule }       from '@angular/core';
import { BrowserModule  } from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routes/app.routes';
import { HttpModule} from '@angular/http';

import { AppComponent }   from './app.component';
import { MyService } from 'services/my.service';

import { routing } from './app-resources/app-routes/app.routes';

@NgModule({
    declarations: [ AppComponent ],
    imports:      [ BrowserModule,
                    routing,
```

```
                    RouterModule,
                    HttpModule ],
    providers: [    appRoutingProviders,
                    MyService
    ],
    bootstrap:    [AppComponent],
})
export class AppModule {}
```

Uso en `MyComponent`

componentes / my.component.ts

Enfoque alternativo para registrar proveedores de aplicaciones en componentes de aplicaciones. Si agregamos proveedores a nivel de componente siempre que el componente se represente, se creará una nueva instancia del servicio.

```
import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({
    ...
    ...
    providers:[MyService] //
 })
export class MyComponent implements OnInit {
    data: any[];
    // Creates private variable myService to use, of type MyService
    constructor(private myService: MyService) { }

    ngOnInit() {
        this.data = this.myService.getData();
    }
}
```

Ejemplo con Promise.resolve

servicios / my.service.ts

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

@Injectable()
export class MyService {
    data: any = [1, 2, 3];

    getData() {
        return Promise.resolve(this.data);
    }
}
```

`getData()` ahora actúa como una llamada REST que crea una Promesa, que se resuelve de inmediato. Los resultados pueden ser portátiles dentro de `.then()` y también se pueden detectar errores. Esta es una buena práctica y convención para métodos asíncronos.

componentes / my.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({...})
export class MyComponent implements OnInit {
    data: any[];
    // Creates private variable myService to use, of type MyService
    constructor(private myService: MyService) { }

    ngOnInit() {
        // Uses an "arrow" function to set data
        this.myService.getData().then(data => this.data = data);
    }
}
```

Probando un servicio

Dado un servicio que puede iniciar sesión un usuario:

```
import 'rxjs/add/operator/toPromise';

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';

interface LoginCredentials {
  password: string;
  user: string;
}

@Injectable()
export class AuthService {
  constructor(private http: Http) { }

  async signIn({ user, password }: LoginCredentials) {
    const response = await this.http.post('/login', {
      password,
      user,
    }).toPromise();

    return response.json();
  }
}
```

Se puede probar así:

```
import { ConnectionBackend, Http, HttpModule, Response, ResponseOptions } from
'@angular/http';
import { TestBed, async, inject } from '@angular/core/testing';

import { AuthService } from './auth.service';
import { MockBackend } from '@angular/http/testing';
import { MockConnection } from '@angular/http/testing';

describe('AuthService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [
```

```
      AuthService,
      Http,
      { provide: ConnectionBackend, useClass: MockBackend },
    ]
  });
});

it('should be created', inject([AuthService], (service: AuthService) => {
  expect(service).toBeTruthy();
}));

// Alternative 1
it('should login user if right credentials are passed', async(
  inject([AuthService], async (authService) => {
    const backend: MockBackend = TestBed.get(ConnectionBackend);
    const http: Http = TestBed.get(Http);

    backend.connections.subscribe((c: MockConnection) => {
      c.mockRespond(
        new Response(
          new ResponseOptions({
            body: {
              accessToken: 'abcdef',
            },
          }),
        ),
      );
    });

    const result = await authService.signIn({ password: 'ok', user: 'bruno' });

    expect(result).toEqual({
      accessToken: 'abcdef',
    });
  }))
);

// Alternative 2
it('should login user if right credentials are passed', async () => {
  const backend: MockBackend = TestBed.get(ConnectionBackend);
  const http: Http = TestBed.get(Http);

  backend.connections.subscribe((c: MockConnection) => {
    c.mockRespond(
      new Response(
        new ResponseOptions({
          body: {
            accessToken: 'abcdef',
          },
        }),
      ),
    );
  });

  const authService: AuthService = TestBed.get(AuthService);

  const result = await authService.signIn({ password: 'ok', user: 'bruno' });

  expect(result).toEqual({
    accessToken: 'abcdef',
  });
```

```
  });

  // Alternative 3
  it('should login user if right credentials are passed', async (done) => {
    const authService: AuthService = TestBed.get(AuthService);

    const backend: MockBackend = TestBed.get(ConnectionBackend);
    const http: Http = TestBed.get(Http);

    backend.connections.subscribe((c: MockConnection) => {
      c.mockRespond(
        new Response(
          new ResponseOptions({
            body: {
              accessToken: 'abcdef',
            },
          }),
        ),
      );
    });

    try {
      const result = await authService.signIn({ password: 'ok', user: 'bruno' });

      expect(result).toEqual({
        accessToken: 'abcdef',
      });

      done();
    } catch (err) {
      fail(err);
      done();
    }
  });
});
```

Lea Inyección de Dependencia y Servicios. en línea: https://riptutorial.com/es/angular2/topic/4187/inyeccion-de-dependencia-y-servicios-

Capítulo 50: Mejora de fuerza bruta

Introducción

Si desea actualizar la versión de Angular CLI de su proyecto, puede encontrar errores y errores difíciles de solucionar simplemente al cambiar el número de versión de Angular CLI en su proyecto. Además, debido a que la CLI angular oculta gran parte de lo que ocurre en el proceso de compilación y agrupación, realmente no se puede hacer mucho cuando las cosas van mal allí.

A veces, la forma más sencilla de actualizar la versión de Angular CLI del proyecto es simplemente armar un nuevo proyecto con la versión de Angular CLI que desea utilizar.

Observaciones

Debido a que Angular 2 es *tan* modular y encapsulado, puede simplemente copiar todos sus componentes, servicios, tuberías, directivas y luego completar el NgModule como estaba en el proyecto anterior.

Examples

Andamios un nuevo proyecto de CLI angular

```
ng new NewProject
```

o

```
ng init NewProject
```

Lea Mejora de fuerza bruta en línea: https://riptutorial.com/es/angular2/topic/9152/mejora-de-fuerza-bruta

Capítulo 51: Mocking @ ngrx / Tienda

Introducción

@ ngrx / Store se está utilizando cada vez más en los proyectos de Angular 2. Como tal, se requiere que la Tienda se inyecte en el constructor de cualquier Componente o Servicio que desee usarla. Pruebas unitarias La tienda no es tan fácil como probar un servicio simple. Al igual que con muchos problemas, hay una gran variedad de formas de implementar soluciones. Sin embargo, la receta básica es escribir una clase simulada para la interfaz Observer y escribir una clase simulada para Tienda. Luego puedes inyectar Store como proveedor en tu TestBed.

Parámetros

nombre	descripción
valor	siguiente valor a observar
error	descripción
errar	error de ser lanzado
súper	descripción
acción $	observador simulado que no hace nada a menos que esté definido para hacerlo en la clase simulada
acciónReductor $	observador simulado que no hace nada a menos que esté definido para hacerlo en la clase simulada
obs $	simulacro observable

Observaciones

El observador es un genérico, pero debe ser de `any` tipo para evitar la complejidad de las pruebas unitarias. La razón de esta complejidad, es que el constructor de la Tienda espera argumentos de observador con diferentes tipos genéricos. El uso de `any` evita esta complicación.

Es posible pasar valores nulos al súper constructor de StoreMock, pero esto restringe el número de aserciones que se pueden usar para probar a la clase en el futuro.

El Componente que se usa en este ejemplo solo se usa como contexto para la forma en que uno podría inyectar la Tienda como un suministro en la configuración de la prueba.

Examples

Mock observador

```
class ObserverMock implements Observer<any> {
  closed?: boolean = false; // inherited from Observer
  nextVal: any = ''; // variable I made up

  constructor() {}

  next = (value: any): void => { this.nextVal = value; };
  error = (err: any): void => { console.error(err); };
  complete = (): void => { this.closed = true; }
}

let actionReducer$: ObserverMock = new ObserverMock();
let action$: ObserverMock = new ObserverMock();
let obs$: Observable<any> = new Observable<any>();

class StoreMock extends Store<any> {
  constructor() {
    super(action$, actionReducer$, obs$);
  }
}

describe('Component:Typeahead', () => {
    beforeEach(() => {
        TestBed.configureTestingModule({
            imports: [...],
            declarations: [Typeahead],
            providers: [
                {provide: Store, useClass: StoreMock} // NOTICE useClass instead of useValue
            ]
        }).compileComponents();
    });
});
```

Prueba de unidad para componente con Mock Store

Esta es una prueba unitaria de un componente que tiene *Almacén* como una dependencia. Aquí, estamos creando una nueva clase llamada *MockStore* que se inyecta en nuestro componente en lugar de la tienda habitual.

```
import { Injectable } from '@angular/core';
import { TestBed, async} from '@angular/core/testing';
import { AppComponent } from './app.component';
import {DumbComponentComponent} from "./dumb-component/dumb-component.component";
import {SmartComponentComponent} from "./smart-component/smart-component.component";
import {mainReducer} from "./state-management/reducers/main-reducer";
import { StoreModule } from "@ngrx/store";
import { Store } from "@ngrx/store";
import {Observable} from "rxjs";


class MockStore {
  public dispatch(obj) {
    console.log('dispatching from the mock store!')
  }
```

```
  public select(obj) {
    console.log('selecting from the mock store!');

    return Observable.of({})
  }
}


describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],
      imports: [
        StoreModule.provideStore({mainReducer})
      ],
      providers: [
        {provide: Store, useClass: MockStore}
      ]
    });
  });

  it('should create the app', async(() => {

    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
```

Prueba unitaria para espionaje de componentes en la tienda

Esta es una prueba unitaria de un componente que tiene *Almacén* como una dependencia. Aquí, podemos usar una tienda con el "estado inicial" predeterminado y, al mismo tiempo, evitar que se *realicen* acciones de envío cuando se llama a *store.dispatch ()* .

```
import {TestBed, async} from '@angular/core/testing';
import {AppComponent} from './app.component';
import {DumbComponentComponent} from "./dumb-component/dumb-component.component";
import {SmartComponentComponent} from "./smart-component/smart-component.component";
import {mainReducer} from "./state-management/reducers/main-reducer";
import {StoreModule} from "@ngrx/store";
import {Store} from "@ngrx/store";
import {Observable} from "rxjs";

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],
      imports: [
        StoreModule.provideStore({mainReducer})
      ]
```

```
    });

  });


  it('should create the app', async(() => {
    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;

  var mockStore = fixture.debugElement.injector.get(Store);
  var storeSpy = spyOn(mockStore, 'dispatch').and.callFake(function () {
  console.log('dispatching from the spy!');
});


}));


});
```

Angular 2 - Simulacro observable (servicio + componente)

Servicio

- He creado el servicio de correos con el método postRequest.

```
import {Injectable} from '@angular/core';
import {Http, Headers, Response} from "@angular/http";
import {PostModel} from "./PostModel";
import 'rxjs/add/operator/map';
import {Observable} from "rxjs";

@Injectable()
export class PostService {

  constructor(private _http: Http) {
  }

  postRequest(postModel: PostModel) : Observable<Response> {
      let headers = new Headers();
      headers.append('Content-Type', 'application/json');
    return this._http.post("/postUrl", postModel, {headers})
      .map(res => res.json());
  }
}
```

Componente

- Creé un componente con parámetro de resultado y función postExample que llama a postService.
- cuando la solicitud posterior a la solicitud sea exitosa, el parámetro del resultado debe ser 'Success' o 'Fail'

```
import {Component} from '@angular/core';
import {PostService} from "./PostService";
import {PostModel} from "./PostModel";
```

```
@Component({
  selector: 'app-post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.scss'],
  providers : [PostService]
})
export class PostComponent{


  constructor(private _postService : PostService) {

  let postModel = new PostModel();
  result : string = null;
  postExample(){
    this._postService.postRequest(this.postModel)
      .subscribe(
        () =>  {
          this.result = 'Success';
        },
        err =>  this.result = 'Fail'
      )
  }
}
```

servicio de prueba

- cuando quiera probar el servicio que usa http, debe usar mockBackend. y se lo inyectan.
- Necesitas también inyectar postService.

```
describe('Test PostService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [
        PostService,
        MockBackend,
        BaseRequestOptions,
        {
          provide: Http,
          deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        }
      ]
    });
  });



  it('sendPostRequest function return Observable', inject([PostService, MockBackend],
(service: PostService, mockBackend: MockBackend) => {
    let mockPostModel = PostModel();

    mockBackend.connections.subscribe((connection: MockConnection) => {
      expect(connection.request.method).toEqual(RequestMethod.Post);
      expect(connection.request.url.indexOf('postUrl')).not.toEqual(-1);
      expect(connection.request.headers.get('Content-Type')).toEqual('application/json');
```

```
    });

    service
      .postRequest(PostModel)
      .subscribe((response) => {
        expect(response).toBeDefined();
      });
  }));
});
```

componente de prueba

```
describe('testing post component', () => {
  let component: PostComponent;
  let fixture: ComponentFixture<postComponent>;


  let mockRouter = {
    navigate: jasmine.createSpy('navigate')
  };

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [PostComponent],
      imports: [RouterTestingModule.withRoutes([]),ModalModule.forRoot() ],
      providers: [PostService ,MockBackend,BaseRequestOptions,
        {provide: Http, deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        },
        {provide: Router, useValue: mockRouter}
      ],
      schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(PostComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });




  it('test postRequest success', inject([PostService, MockBackend], (service: PostService,
mockBackend: MockBackend) => {
    fixturePostComponent = TestBed.createComponent(PostComponent);
    componentPostComponent = fixturePostComponent.componentInstance;
    fixturePostComponent.detectChanges();


    component.postExample();
    let postModel = new PostModel();
    let response = {
      'message' : 'message',
      'ok'      : true
    };
    mockBackend.connections.subscribe((connection: MockConnection) => {
```

```
      postComponent.result = 'Success'
      connection.mockRespond(new Response(
        new ResponseOptions({
          body: response
        })
      ))
    });
    service.postRequest(postModel)
      .subscribe((data) => {
        expect(component.result).toBeDefined();
        expect(PostComponent.result).toEqual('Success');
        expect(data).toEqual(response);
      });
  }));
});
```

Tienda simple

simple.action.ts

```
import { Action } from '@ngrx/store';

export enum simpleActionTpye {
    add = "simpleAction_Add",
    add_Success = "simpleAction_Add_Success"
}

export class simpleAction {
    type: simpleActionTpye
    constructor(public payload: number) { }
}
```

simple.efficts.ts

```
import { Effect, Actions } from '@ngrx/effects';
import { Injectable } from '@angular/core';
import { Action } from '@ngrx/store';
import { Observable } from 'rxjs';

import { simpleAction, simpleActionTpye } from './simple.action';


@Injectable()
export class simpleEffects {

    @Effect()
    addAction$: Observable<simpleAction> = this.actions$
        .ofType(simpleActionTpye.add)
        .switchMap((action: simpleAction) => {
            console.log(action);

            return Observable.of({ type: simpleActionTpye.add_Success, payload: action.payload
})
            //   if you have an api use this code
            // return this.http.post(url).catch().map(res=>{ type: simpleAction.add_Success,
payload:res})
        });
    constructor(private actions$: Actions) { }
```

```
}
```

simple.reducer.ts

```
import { Action, ActionReducer } from '@ngrx/store';

import { simpleAction, simpleActionTpye } from './simple.action';

export const simpleReducer: ActionReducer<number> = (state: number = 0, action: simpleAction):
number => {
    switch (action.type) {
        case simpleActionTpye.add_Success:
            console.log(action);
            return state + action.payload;
        default:
            return state;
    }
}
```

tienda / index.ts

```
import { combineReducers, ActionReducer, Action, StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { ModuleWithProviders } from '@angular/core';
import { compose } from '@ngrx/core';

import { simpleReducer } from "./simple/simple.reducer";
import { simpleEffects } from "./simple/simple.efficts";


export interface IAppState {
    sum: number;
}

// all new reducers should be define here
const reducers = {
    sum: simpleReducer
};

export const store: ModuleWithProviders = StoreModule.forRoot(reducers);
export const effects: ModuleWithProviders[] = [
    EffectsModule.forRoot([simpleEffects])
];
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core';

import { effects, store } from "./Store/index";
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
```

```
    BrowserModule,
    // store
    store,
    effects
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.ts

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

import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

import { IAppState } from './Store/index';
import { simpleActionTpye } from './Store/simple/simple.action';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';

  constructor(private store: Store<IAppState>) {
    store.select(s => s.sum).subscribe((res) => {
      console.log(res);
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 1
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 2
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 3
    })
  }
}
```

resultado 0 1 3 6

Lea Mocking @ ngrx / Tienda en línea: https://riptutorial.com/es/angular2/topic/8038/mocking---ngrx---tienda

Capítulo 52: Módulos

Introducción

Los módulos angulares son contenedores para diferentes partes de su aplicación.

Puede tener módulos anidados, su `app.module` ya está anidando otros módulos como `BrowserModule` y puede agregar `RouterModule` y así sucesivamente.

Examples

Un modulo simple

Un módulo es una clase con el decorador `@NgModule` . Para crear un módulo agregamos `@NgModule` pasando algunos parámetros:

- `bootstrap` : El componente que será la raíz de su aplicación. Esta configuración solo está presente en su módulo raíz
- `declarations` : Recursos que el módulo declara. Cuando agrega un nuevo componente, debe actualizar las declaraciones (`ng generate component` hace automáticamente)
- `exports` : recursos que el módulo exporta que se pueden usar en otros módulos.
- `imports` : recursos que el módulo utiliza de otros módulos (solo se aceptan clases de módulos)
- `providers` : Recursos que pueden inyectarse (di) en un componente.

Un ejemplo simple:

```
import { AppComponent } from './app.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

@NgModule({
  bootstrap: [AppComponent]
  declarations: [AppComponent],
  exports: [],
  imports: [BrowserModule],
  providers: [],
})
export class AppModule { }
```

Módulos de anidamiento

Los módulos pueden ser anidadas mediante el uso de la `imports` parámetro de `@NgModule` decorador.

Podemos crear un `core.module` en nuestra aplicación que contendrá cosas genéricas, como un `ReservePipe` (una tubería que invierte una cadena) y agruparlas en este módulo:

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { ReversePipe } from '../reverse.pipe';

@NgModule({
  imports: [
    CommonModule
  ],
  exports: [ReversePipe], // export things to be imported in another module
  declarations: [ReversePipe],
})
export class CoreModule { }
```

Luego en el `app.module` :

```
import { CoreModule } from 'app/core/core.module';

@NgModule({
  declarations: [...], // ReversePipe is available without declaring here
                       // because CoreModule exports it
  imports: [
    CoreModule,        // import things from CoreModule
    ...
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Lea Módulos en línea: https://riptutorial.com/es/angular2/topic/10840/modulos

Capítulo 53: Módulos de funciones

Examples

Un módulo de funciones

```
// my-feature.module.ts
import { CommonModule } from '@angular/common';
import { NgModule }     from '@angular/core';

import { MyComponent } from './my.component';
import { MyDirective } from './my.directive';
import { MyPipe }      from './my.pipe';
import { MyService }   from './my.service';

@NgModule({
  imports:      [ CommonModule ],
  declarations: [ MyComponent, MyDirective, MyPipe ],
  exports:      [ MyComponent ],
  providers:    [ MyService ]
})
export class MyFeatureModule { }
```

Ahora, en tu raíz (normalmente `app.module.ts`):

```
// app.module.ts
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }    from './app.component';
import { MyFeatureModule } from './my-feature.module';

@NgModule({
  // import MyFeatureModule in root module
  imports:      [ BrowserModule, MyFeatureModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Lea Módulos de funciones en línea: https://riptutorial.com/es/angular2/topic/6551/modulos-de-funciones

Capítulo 54: ngrx

Introducción

Ngrx es una biblioteca poderosa que puedes usar con **Angular2** . La idea subyacente es combinar dos conceptos que funcionen bien juntos para tener una **aplicación reactiva** con un **contenedor de estado** predecible: - [Redux] [1] - [RxJs] [2] Las principales ventajas: - Compartir datos en su aplicación entre sus componentes va a ser más fácil: probar la lógica central de su aplicación consiste en probar funciones puras, sin ninguna dependencia de Angular2 (¡muy fácil!) [1]: http://redux.js.org [2]: http: // reactivex. io / rxjs

Examples

Ejemplo completo: iniciar sesión / cerrar sesión de un usuario

Prerrequisitos

Este tema **no** es sobre Redux y / o Ngrx:

- Necesitas estar cómodo con Redux
- Al menos entender los conceptos básicos de RxJs y el patrón observable

---

Primero, definamos un ejemplo desde el principio y juguemos con algún código:

Como desarrollador, quiero:

1. Tener una `IUser` interfaz que define las propiedades de un `User`
2. Declare las acciones que usaremos más adelante para manipular al `User` en la `Store`
3. Definir el estado inicial del `UserReducer`
4. Crear el reductor `UserReducer`
5. Importe nuestro `UserReducer` en nuestro módulo principal para construir la `Store`
6. Utilice los datos de la `Store` para mostrar información en nuestra vista

Alerta de spoiler : si desea probar la demostración de inmediato o leer el código antes de comenzar, aquí hay un Plunkr (vista de incrustación o vista de ejecución).

1) Definir la interfaz `IUser`

Me gusta dividir mis interfaces en dos partes:

- Las propiedades que obtendremos de un servidor.
- Las propiedades que definimos solo para la interfaz de usuario (por ejemplo, un botón debe estar girando)

Y aquí está la interfaz que `IUser` :

`user.interface.ts`

```
export interface IUser {
  // from server
  username: string;
  email: string;

  // for UI
  isConnecting: boolean;
  isConnected: boolean;
};
```

---

2) Declarar las acciones para manipular al `User`

Ahora tenemos que pensar qué tipo de acciones se supone que deben manejar nuestros ***reductores*** .
Vamos a decir aquí:

`user.actions.ts`

```
export const UserActions = {
  // when the user clicks on login button, before we launch the HTTP request
  // this will allow us to disable the login button during the request
  USR_IS_CONNECTING: 'USR_IS_CONNECTING',
  // this allows us to save the username and email of the user
  // we assume those data were fetched in the previous request
  USR_IS_CONNECTED: 'USR_IS_CONNECTED',

  // same pattern for disconnecting the user
  USR_IS_DISCONNECTING: 'USR_IS_DISCONNECTING',
  USR_IS_DISCONNECTED: 'USR_IS_DISCONNECTED'
};
```

Pero antes de usar esas acciones, permítame explicar por qué vamos a necesitar un servicio para enviar **algunas** de esas acciones para nosotros:

Digamos que queremos conectar a un usuario. Entonces, haremos clic en el botón de inicio de sesión y esto es lo que sucederá:

- Haga clic en el botón
- El componente captura el evento y llama a `userService.login`
- `userService.login` método `userService.login dispatch` un evento para actualizar nuestra propiedad de la tienda: `user.isConnecting`
- Se `setTimeout` una llamada HTTP (usaremos un `setTimeout` en la demostración para simular el **comportamiento asíncrono**)
- Una vez que finalice la llamada `HTTP` , enviaremos otra acción para advertir a nuestra tienda que un usuario está registrado

`user.service.ts`

```
@Injectable()
export class UserService {
  constructor(public store$: Store<AppState>) { }

  login(username: string) {
    // first, dispatch an action saying that the user's tyring to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_CONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      // some email (or data) that you'd have get as HTTP response
      let email = `${username}@email.com`;

      this.store$.dispatch({ type: UserActions.USR_IS_CONNECTED, payload: { username, email }
});
    }, 2000);
  }

  logout() {
    // first, dispatch an action saying that the user's tyring to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTED });
    }, 2000);
  }
}
```

---

3) Definir el estado inicial del `UserReducer`

`user.state.ts`

```
export const UserFactory: IUser = () => {
  return {
    // from server
    username: null,
    email: null,

    // for UI
    isConnecting: false,
    isConnected: false,
    isDisconnecting: false
  };
};
```

---

4) Crear el reductor `UserReducer`

Un reductor toma 2 argumentos:

- El estado actual
- Una `Action` de tipo `Action<{type: string, payload: any}>`

Recordatorio: un reductor debe inicializarse en algún momento

Como definimos el estado predeterminado de nuestro reductor en la parte 3), podremos usarlo así:

`user.reducer.ts`

```
export const UserReducer: ActionReducer<IUser> = (user: IUser, action: Action) => {
  if (user === null) {
    return userFactory();
  }

  // ...
}
```

Con suerte, hay una forma más fácil de escribir eso al usar nuestra función de `factory` para devolver un objeto y dentro del reductor usar un valor de parámetros predeterminado (ES6):

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  // ...
}
```

Luego, debemos manejar cada acción en nuestro reductor: ***SUGERENCIA*** : use la función ES6 `Object.assign` para mantener nuestro estado inmutable

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  switch (action.type) {
    case UserActions.USR_IS_CONNECTING:
      return Object.assign({}, user, { isConnecting: true });

    case UserActions.USR_IS_CONNECTED:
      return Object.assign({}, user, { isConnecting: false, isConnected: true, username:
action.payload.username });

    case UserActions.USR_IS_DISCONNECTING:
      return Object.assign({}, user, { isDisconnecting: true });

    case UserActions.USR_IS_DISCONNECTED:
      return Object.assign({}, user, { isDisconnecting: false, isConnected: false });

    default:
      return user;
  }
};
```

5) Importe nuestro `UserReducer` en nuestro módulo principal para construir la `Store`

`app.module.ts`

```
@NgModule({
    declarations: [
    AppComponent
    ],
    imports: [
    // angular modules
    // ...

    // declare your store by providing your reducers
    // (every reducer should return a default state)
    StoreModule.provideStore({
        user: UserReducer,
        // of course, you can put as many reducers here as you want
        // ...
    }),

    // other modules to import
    // ...
    ]
});
```

6) Utilice los datos de la `Store` para mostrar información en nuestra vista

Ahora todo está listo en el lado lógico y solo tenemos que mostrar lo que queremos en dos componentes:

- `UserComponent` : **[Dumb component]** Solo pasaremos el objeto de usuario de la tienda usando la propiedad `@Input` y el conducto `async` . De esta manera, el componente recibirá al usuario solo una vez que esté disponible (¡y el `user` será de tipo `IUser` y no de tipo `Observable<IUser>` !)
- `LoginComponent` **[componente inteligente]** Inyectaremos directamente la `Store` en este componente y trabajaremos solo en el `user` como `Observable` .

`user.component.ts`

```
@Component({
  selector: 'user',
  styles: [
    '.table { max-width: 250px; }',
    '.truthy { color: green; font-weight: bold; }',
    '.falsy { color: red; }'
  ],
  template: `
    <h2>User information :</h2>
```

```
    <table class="table">
      <tr>
        <th>Property</th>
        <th>Value</th>
      </tr>

      <tr>
        <td>username</td>
        <td [class.truthy]="user.username" [class.falsy]="!user.username">
          {{ user.username ? user.username : 'null' }}
        </td>
      </tr>

      <tr>
        <td>email</td>
        <td [class.truthy]="user.email" [class.falsy]="!user.email">
          {{ user.email ? user.email : 'null' }}
        </td>
      </tr>

      <tr>
        <td>isConnecting</td>
        <td [class.truthy]="user.isConnecting" [class.falsy]="!user.isConnecting">
          {{ user.isConnecting }}
        </td>
      </tr>

      <tr>
        <td>isConnected</td>
        <td [class.truthy]="user.isConnected" [class.falsy]="!user.isConnected">
          {{ user.isConnected }}
        </td>
      </tr>

      <tr>
        <td>isDisconnecting</td>
        <td [class.truthy]="user.isDisconnecting" [class.falsy]="!user.isDisconnecting">
          {{ user.isDisconnecting }}
        </td>
      </tr>
    </table>
  `
})
export class UserComponent {
  @Input() user;

  constructor() { }
}
```

login.component.ts

```
@Component({
  selector: 'login',
  template: `
    <form
      *ngIf="!(user | async).isConnected"
      #loginForm="ngForm"
      (ngSubmit)="login(loginForm.value.username)"
    >
```

```
      <input
        type="text"
        name="username"
        placeholder="Username"
        [disabled]="(user | async).isConnecting"
        ngModel
      >

      <button
        type="submit"
        [disabled]="(user | async).isConnecting || (user | async).isConnected"
      >Log me in</button>
    </form>

    <button
      *ngIf="(user | async).isConnected"
      (click)="logout()"
      [disabled]="(user | async).isDisconnecting"
    >Log me out</button>
  `
})
export class LoginComponent {
  public user: Observable<IUser>;

  constructor(public store$: Store<AppState>, private userService: UserService) {
      this.user = store$.select('user');
  }

  login(username: string) {
    this.userService.login(username);
  }

  logout() {
    this.userService.logout();
  }
}
```

Como `Ngrx` es una combinación de los conceptos de `Redux` y `RxJs` , puede ser bastante difícil de entender al principio. Pero este es un patrón poderoso que le permite, como hemos visto en este ejemplo, tener una *aplicación reactiva en la* que puede compartir fácilmente sus datos. ¡No olvide que hay un Plunkr disponible y puede hacerlo para hacer sus propias pruebas!

Espero que haya sido útil, aunque el tema es bastante largo, ¡salud!

Lea ngrx en línea: https://riptutorial.com/es/angular2/topic/8086/ngrx

Capítulo 55: Omitir la desinfección para valores de confianza

Parámetros

Parámetros	Detalles
selector	nombre de la etiqueta que hace referencia a su componente en el html
plantilla (templateUrl)	una cadena que representa html que se insertará donde sea que esté la etiqueta `<selector>` . templateUrl es una ruta a un archivo html con el mismo comportamiento
tubería	una matriz de tuberías que son utilizadas por este componente.

Observaciones

SUPER IMPORTANTE!

EL DESHABILITAR EL DESINFECTANTE LE PERMITE AL RIESGO DE XSS (secuencias de comandos entre sitios) Y OTROS VECTORES DE ATAQUE. POR FAVOR ASEGÚRESE DE QUE CONFIA EN LO QUE ESTÁ OBTENIENDO EL 100%

El uso de Pipes le relega a cambiar solo los valores de los atributos de esta manera:

```
<tag [attribute]="expression or variable reference | pipeName">
```

No puedes usar tuberías de esta manera:

```
<tag attribute="expression or variable reference | pipeName">
```

o de esta manera

```
<tag attribute={{expression or variable reference | pipeName}}
```

Examples

Derivación de desinfección con tuberías (para la reutilización del código)

El proyecto sigue la estructura de la guía de inicio rápido de Angular2 aquí .

```
RootOfProject
|
+-- app
|   |-- app.component.ts
|   |-- main.ts
|   |-- pipeUser.component.ts
|   \-- sanitize.pipe.ts
|
|-- index.html
|-- main.html
|-- pipe.html
```

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Esto encuentra el archivo index.html en la raíz del proyecto, y se basa en eso.

app.component.ts

```
import { Component } from '@angular/core';
import { PipeUserComponent } from './pipeUser.component';

@Component({
    selector: 'main-app',
    templateUrl: 'main.html',
    directives: [PipeUserComponent]
})

export class AppComponent { }
```

Este es el componente de nivel superior que agrupa otros componentes que se utilizan.

pipeUser.component.ts

```
import { Component } from '@angular/core';
import { IgnoreSanitize } from "./sanitize.pipe";

@Component({
    selector: 'pipe-example',
    templateUrl: "pipe.html",
    pipes: [IgnoreSanitize]
})

export class PipeUserComponent{
    constructor () { }
    unsafeValue: string = "unsafe/picUrl?id=";
    docNum: string;

    getUrl(input: string): any {
        if(input !== undefined) {
```

```
            return this.unsafeValue.concat(input);
            // returns : "unsafe/picUrl?id=input"
        } else {
            return "fallback/to/something";
        }
    }
}
```

Este componente proporciona la vista para que la tubería funcione.

sanitize.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizationService } from '@angular/platform-browser';

@Pipe({
    name: 'sanitaryPipe'
})
export class IgnoreSanitize implements PipeTransform {

   constructor(private sanitizer: DomSanitizationService){}

   transform(input: string) : any {
       return this.sanitizer.bypassSecurityTrustUrl(input);
   }

}
```

Esta es la lógica que describe lo que formatea la tubería.

index.html

```
<head>
    Stuff goes here...
</head>
<body>
    <main-app>
        main.html will load inside here.
    </main-app>
</body>
```

main.html

```
<othertags>
</othertags>

<pipe-example>
    pipe.html will load inside here.
</pipe-example>

<moretags>
</moretags>
```

pipe.html

```
<img [src]="getUrl('1234') | sanitaryPipe">
<embed [src]="getUrl() | sanitaryPipe">
```

Si fueras a inspeccionar el html mientras la aplicación se está ejecutando, verías que se ve así:

```
<head>
    Stuff goes here...
</head>

<body>

    <othertags>
    </othertags>

    <img [src]="getUrl('1234') | sanitaryPipe">
    <embed [src]="getUrl() | sanitaryPipe">

    <moretags>
    </moretags>

</body>
```

Lea Omitir la desinfección para valores de confianza en línea:
https://riptutorial.com/es/angular2/topic/5942/omitir-la-desinfeccion-para-valores-de-confianza

Capítulo 56: Optimización de la representación mediante ChangeDetectionStrategy

Examples

Predeterminado vs OnPush

Considere el siguiente componente con una entrada `myInput` y un valor interno llamado `someInternalValue` . Ambos se utilizan en la plantilla de un componente.

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

@Component({
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input() myInput: any;

  someInternalValue: any;

  // ...
}
```

De forma predeterminada, la propiedad `changeDetection:` en el decorador de componentes se establecerá en `ChangeDetectionStrategy.Default` ; implícito en el ejemplo. En esta situación, cualquier cambio en cualquiera de los valores de la plantilla activará una nueva representación de `MyComponent` . En otras palabras, si cambio `myInput` o `someInternalValue` angular 2 ejercerá energía y volverá a generar el componente.

Sin embargo, supongamos que solo queremos volver a renderizar cuando cambian las entradas. Considere el siguiente componente con `changeDetection:` configurado en `ChangeDetectionStrategy.OnPush`

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

@Component({
  changeDetection: ChangeDetectionStrategy.OnPush
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
```

```
})
class MyComponent {
  @Input() myInput: any;

  someInternalValue: any;

  // ...
}
```

Al establecer `changeDetection:` to `ChangeDetectionStrategy.OnPush` , `MyComponent` solo se volverá a representar cuando cambien sus entradas. En este caso, `myInput` deberá recibir un nuevo valor de su padre para desencadenar una nueva representación.

Lea Optimización de la representación mediante ChangeDetectionStrategy en línea: https://riptutorial.com/es/angular2/topic/2644/optimizacion-de-la-representacion-mediante-changedetectionstrategy

Capítulo 57: Orden por pipa

Introducción

Cómo escribir el tubo de pedido y usarlo.

Examples

El tubo

La implementación de Pipe

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
  name: 'orderBy',
  pure: false
})
export class OrderBy implements PipeTransform {

  value:string[] =[];

  static _orderByComparator(a:any, b:any):number{

    if(a === null || typeof a === 'undefined') a = 0;
    if(b === null || typeof b === 'undefined') b = 0;

    if((isNaN(parseFloat(a)) || !isFinite(a)) || (isNaN(parseFloat(b)) || !isFinite(b))){
      //Isn't a number so lowercase the string to properly compare
      if(a.toLowerCase() < b.toLowerCase()) return -1;
      if(a.toLowerCase() > b.toLowerCase()) return 1;
    }else{
      //Parse strings as numbers to compare properly
      if(parseFloat(a) < parseFloat(b)) return -1;
      if(parseFloat(a) > parseFloat(b)) return 1;
    }

    return 0; //equal each other
  }

  transform(input:any, config:string = '+'): any{

    //make a copy of the input's reference
    this.value = [...input];
    let value = this.value;

    if(!Array.isArray(value)) return value;

    if(!Array.isArray(config) || (Array.isArray(config) && config.length === 1)){
      let propertyToCheck:string = !Array.isArray(config) ? config : config[0];
      let desc = propertyToCheck.substr(0, 1) === '-';

      //Basic array
      if(!propertyToCheck || propertyToCheck === '-' || propertyToCheck === '+'){
```

```
        return !desc ? value.sort() : value.sort().reverse();
      }else {
        let property:string = propertyToCheck.substr(0, 1) === '+' ||
propertyToCheck.substr(0, 1) === '-'
          ? propertyToCheck.substr(1)
          : propertyToCheck;

        return value.sort(function(a:any,b:any){
          return !desc
            ? OrderBy._orderByComparator(a[property], b[property])
            : -OrderBy._orderByComparator(a[property], b[property]);
        });
      }
    } else {
      //Loop over property of the array in order and sort
      return value.sort(function(a:any,b:any){
        for(let i:number = 0; i < config.length; i++){
          let desc = config[i].substr(0, 1) === '-';
          let property = config[i].substr(0, 1) === '+' || config[i].substr(0, 1) === '-'
            ? config[i].substr(1)
            : config[i];

          let comparison = !desc
            ? OrderBy._orderByComparator(a[property], b[property])
            : -OrderBy._orderByComparator(a[property], b[property]);

          //Don't return 0 yet in case of needing to sort by next property
          if(comparison !== 0) return comparison;
        }

        return 0; //equal each other
      });
    }
  }
}
```

Cómo usar la tubería en el HTML - orden ascendente por nombre

---

```
<table>
   <thead>
    <tr>
     <th>First Name</th>
     <th>Last Name</th>
    <th>Age</th>
    </tr>
</thead>
<tbody>
   <tr *ngFor="let user of users | orderBy : ['firstName']>
       <td>{{user.firstName}}</td>
       <td>{{user.lastName}}</td>
       <td>{{user.age}}</td>
   </tr>
</tbody>
</table>
```

Cómo usar la tubería en el HTML - orden descendente por nombre

---

```
<table>
   <thead>
    <tr>
     <th>First Name</th>
     <th>Last Name</th>
    <th>Age</th>
    </tr>
</thead>
<tbody>
   <tr *ngFor="let user of users | orderBy : ['-firstName']>
       <td>{{user.firstName}}</td>
       <td>{{user.lastName}}</td>
       <td>{{user.age}}</td>
   </tr>
</tbody>
</table>
```

Lea Orden por pipa en línea: https://riptutorial.com/es/angular2/topic/8969/orden-por-pipa

Capítulo 58: Perezoso cargando un modulo

Examples

Ejemplo de carga perezosa

Los módulos de **carga** diferida nos ayudan a disminuir el tiempo de inicio. Con la carga lenta nuestra aplicación no necesita cargar todo a la vez, solo necesita cargar lo que el usuario espera ver cuando se carga la aplicación por primera vez. Los módulos que se cargan perezosamente solo se cargarán cuando el usuario navegue a sus rutas.

app / app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule  } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { EagerComponent } from './eager.component';
import { routing } from './app.routing';
@NgModule({
  imports: [
    BrowserModule,
    routing
  ],
  declarations: [
    AppComponent,
    EagerComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

app / app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `<h1>My App</h1>    <nav>
      <a routerLink="eager">Eager</a>
      <a routerLink="lazy">Lazy</a>
    </nav>
    <router-outlet></router-outlet>
  `
})
export class AppComponent {}
```

app / app.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { EagerComponent } from './eager.component';
const routes: Routes = [
  { path: '', redirectTo: 'eager', pathMatch: 'full' },
```

```
  { path: 'eager', component: EagerComponent },
  { path: 'lazy', loadChildren: './lazy.module' }
];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

app / eager.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: '`<p>Eager Component</p>`'
})
export class EagerComponent {}
```

No hay nada especial en LazyModule que no tenga su propio enrutamiento y un componente llamado LazyComponent (pero no es necesario nombrar su módulo o simliar).

app / lazy.module.ts

```
import { NgModule } from '@angular/core';
import { LazyComponent }   from './lazy.component';
import { routing } from './lazy.routing';
@NgModule({
  imports: [routing],
  declarations: [LazyComponent]
})
export class LazyModule {}
```

app / lazy.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LazyComponent } from './lazy.component';
const routes: Routes = [
  { path: '', component: LazyComponent }
];
export const routing: ModuleWithProviders = RouterModule.forChild(routes);
```

app / lazy.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: `<p>Lazy Component</p>`
})
export class LazyComponent {}
```

Lea Perezoso cargando un modulo en línea:
https://riptutorial.com/es/angular2/topic/7751/perezoso-cargando-un-modulo

Capítulo 59: Plantillas

Introducción

Las plantillas son muy similares a las plantillas en Angular 1, aunque hay muchos pequeños cambios sintácticos que hacen más claro lo que está sucediendo.

Examples

Plantillas angulares 2

UNA PLANTILLA SIMPLE

Comencemos con una plantilla muy simple que muestra nuestro nombre y nuestra cosa favorita:

```
<div>
  Hello my name is {{name}} and I like {{thing}} quite a lot.
</div>
```

`{}`: RENDER

Para representar un valor, podemos usar la sintaxis de doble rizado estándar:

```
My name is {{name}}
```

Las canalizaciones, antes conocidas como "Filtros", transforman un valor en un nuevo valor, como localizar una cadena o convertir un valor de punto flotante en una representación de moneda:

`[]`: PROPIEDADES DE UNIÓN

Para resolver y enlazar una variable a un componente, use la sintaxis []. Si tenemos este.currentVolume en nuestro componente, lo pasaremos a nuestro componente y los valores permanecerán sincronizados:

```
<video-control [volume]="currentVolume"></video-control>
(): HANDLING EVENTS
```

`()`: MANEJAR EVENTOS Para escuchar un evento en un componente, usamos la sintaxis ()

```
<my-component (click)="onClick($event)"></my-component>
```

`[()]`: VINCULACIÓN DE DATOS DE DOS VÍAS

Para mantener un enlace actualizado con la entrada del usuario y otros eventos, use la sintaxis [()]. Piense en ello como una combinación de manejar un evento y vincular una propiedad:

<input [(ngModel)] = "myName"> El valor this.myName de su componente se mantendrá sincronizado con el valor de entrada.

`*` : EL ASTERISCO

Indica que esta directiva trata este componente como una plantilla y no lo dibujará como está. Por ejemplo, ngFor toma nuestro y lo estampa para cada artículo en artículos, pero nunca muestra nuestra inicial ya que es una plantilla:

```
<my-component *ngFor="#item of items">
</my-component>
```

Otras directivas similares que funcionan con plantillas en lugar de componentes renderizados son * ngIf y * ngSwitch.

Lea Plantillas en línea: https://riptutorial.com/es/angular2/topic/9471/plantillas

Capítulo 60: Probando ngModel

Introducción

Es un ejemplo de cómo puede probar un componente en Angular2 que tiene un ngModel.

Examples

Prueba basica

```
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { Component, DebugElement } from '@angular/core';
import { dispatchEvent } from "@angular/platform-browser/testing/browser_util";
import { TestBed, ComponentFixture} from '@angular/core/testing';
import {By} from "@angular/platform-browser";

import { MyComponentModule } from 'ng2-my-component';
import { MyComponent } from './my-component';

describe('MyComponent:',()=> {

  const template = `
    <div>
       <my-component type="text" [(ngModel)]="value" name="TestName" size="9" min="3" max="8"
placeholder="testPlaceholder" disabled=false required=false></my-component>
    </div>
  `;

  let fixture:any;
  let element:any;
  let context:any;

  beforeEach(() => {

      TestBed.configureTestingModule({
          declarations: [InlineEditorComponent],
          imports: [
            FormsModule,
            InlineEditorModule]
      });
      fixture = TestBed.overrideComponent(InlineEditorComponent, {
      set: {
        selector:"inline-editor-test",
        template: template
      }})
      .createComponent(InlineEditorComponent);
      context = fixture.componentInstance;
      fixture.detectChanges();
  });

  it('should change value of the component', () => {
```

```
        let input = fixture.nativeElement.querySelector("input");
        input.value = "Username";
        dispatchEvent(input, 'input');
        fixture.detectChanges();

        fixture.whenStable().then(() => {
            //this button dispatch event for save the text in component.value
            fixture.nativeElement.querySelectorAll('button')[0].click();
            expect(context.value).toBe("Username");
        });
    });
});
```

Lea Probando ngModel en línea: https://riptutorial.com/es/angular2/topic/8693/probando-ngmodel

Capítulo 61: Probando una aplicación Angular 2

Examples

Instalando el framework de pruebas Jasmine

La forma más común de probar aplicaciones de Angular 2 es con el marco de prueba Jasmine. Jasmine te permite probar tu código en el navegador.

Instalar

Para comenzar, todo lo que necesita es el paquete `jasmine-core` (no `jasmine`).

```
npm install jasmine-core --save-dev --save-exact
```

Verificar

Para verificar que Jasmine está configurado correctamente, cree el archivo `./src/unit-tests.html` con el siguiente contenido y ábralo en el navegador.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html;charset=utf-8">
  <title>Ng App Unit Tests</title>
  <link rel="stylesheet" href="../node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>
</head>
<body>
   <!-- Unit Testing Chapter #1: Proof of life.  -->
   <script>
     it('true is true', function () {
       expect(true).toEqual(true);
     });
   </script>
</body>
</html>
```

Configurando pruebas con Gulp, Webpack, Karma y Jasmine

Lo primero que necesitamos es decirle a karma que use Webpack para leer nuestras pruebas, bajo una configuración que configuramos para el motor del paquete web. Aquí, estoy usando

babel porque escribo mi código en ES6, puedes cambiarlo por otros sabores, como Typescript. O uso plantillas Pug (anteriormente Jade), no tienes que hacerlo.

Aún así, la estrategia sigue siendo la misma.

Por lo tanto, esta es una configuración webpack:

```
const webpack = require("webpack");
let packConfig = {
    entry: {},
    output: {},
    plugins:[
        new webpack.DefinePlugin({
            ENVIRONMENT: JSON.stringify('test')
        })
    ],
    module: {
       loaders: [
        {
            test: /\.js$/,
            exclude:/(node_modules|bower_components)/,
            loader: "babel",
            query:{
                presets:["es2015", "angular2"]
            }
        },
        {
            test: /\.woff2?$|\.ttf$|\.eot$|\.svg$/,
            loader: "file"
        },
        {
            test: /\.scss$/,
            loaders: ["style", "css", "sass"]
        },
        {
            test: /\.pug$/,
            loader: 'pug-html-loader'
        },
        ]
    },
    devtool : 'inline-cheap-source-map'
};
module.exports = packConfig;
```

Y luego, necesitamos un archivo karma.config.js para usar esa configuración de webpack:

```
const packConfig = require("./webpack.config.js");
module.exports = function (config) {
    config.set({
    basePath: '',
    frameworks: ['jasmine'],
    exclude:[],
    files: [
        {pattern: './karma.shim.js', watched: false}
    ],

    preprocessors: {
        "./karma.shim.js":["webpack"]
```

```
    },
    webpack: packConfig,

    webpackServer: {noInfo: true},

    port: 9876,

    colors: true,

    logLevel: config.LOG_INFO,

    browsers: ['PhantomJS'],

    concurrency: Infinity,

    autoWatch: false,
    singleRun: true
});
};
```

Hasta ahora, le hemos dicho a Karma que use el paquete web, y le hemos dicho que comience en un archivo llamado **karma.shim.js** . Este archivo tendrá el trabajo de actuar como punto de partida para el paquete web. webpack leerá este archivo y utilizará las declaraciones de **importación** y **requerimiento** para reunir todas nuestras dependencias y ejecutar nuestras pruebas.

Así que ahora, veamos el archivo karma.shim.js:

```
// Start of ES6 Specific stuff
import "es6-shim";
import "es6-promise";
import "reflect-metadata";
// End of ES6 Specific stuff

import "zone.js/dist/zone";
import "zone.js/dist/long-stack-trace-zone";
import "zone.js/dist/jasmine-patch";
import "zone.js/dist/async-test";
import "zone.js/dist/fake-async-test";
import "zone.js/dist/sync-test";
import "zone.js/dist/proxy-zone";

import 'rxjs/add/operator/map';
import 'rxjs/add/observable/of';

Error.stackTraceLimit = Infinity;

import {TestBed} from "@angular/core/testing";
import { BrowserDynamicTestingModule, platformBrowserDynamicTesting} from "@angular/platform-
browser-dynamic/testing";

TestBed.initTestEnvironment(
     BrowserDynamicTestingModule,
     platformBrowserDynamicTesting());

let testContext = require.context('../src/app', true, /\.spec\.js/);
testContext.keys().forEach(testContext);
```

En esencia, estamos importando **TestBed** desde las pruebas de núcleo angular e iniciando el entorno, ya que debe iniciarse solo una vez para todas nuestras pruebas. Luego, vamos a través del directorio **src / app** recursivamente y leemos cada archivo que termina con **.spec.js** y los alimentamos a testContext, por lo que se ejecutarán.

Usualmente trato de poner mis exámenes en el mismo lugar que la clase. A gusto personal, me facilita importar dependencias y refactorizar pruebas con clases. Pero si desea poner sus pruebas en otro lugar, como en el directorio **src / test** , por ejemplo, aquí tiene la oportunidad. cambia la línea antes del último en el archivo karma.shim.js.

Perfecto. ¿lo que queda? ah, la tarea truculenta que usa el archivo karma.config.js que hicimos anteriormente:

```
gulp.task("karmaTests",function(done){
    var Server = require("karma").Server;
    new Server({
        configFile : "./karma.config.js",
        singleRun: true,
        autoWatch: false
    }, function(result){
        return result ? done(new Error(`Karma failed with error code ${result}`)):done();
    }).start();
});
```

Ahora estoy iniciando el servidor con el archivo de configuración que creamos, diciéndole que se ejecute una vez y que no observe los cambios. Considero que esto se adapta mejor a mí, ya que las pruebas solo se ejecutarán si estoy listo para que se ejecuten, pero, por supuesto, si quieres algo diferente, sabes dónde cambiar.

Y como ejemplo de código final, aquí hay un conjunto de pruebas para el tutorial de Angular 2, "Tour of Heroes".

```
import {
    TestBed,
    ComponentFixture,
    async
} from "@angular/core/testing";

import {AppComponent} from "./app.component";
import {AppModule} from "./app.module";
import Hero from "./hero/hero";

describe("App Component", function () {

    beforeEach(()=> {
        TestBed.configureTestingModule({
            imports: [AppModule]
        });

        this.fixture = TestBed.createComponent(AppComponent);
        this.fixture.detectChanges();
    });

    it("Should have a title", async(()=> {
        this.fixture.whenStable().then(()=> {
```

```
            expect(this.fixture.componentInstance.title).toEqual("Tour of Heros");
        });
    }));

    it("Should have a hero", async(()=> {
        this.fixture.whenStable().then(()=> {
            expect(this.fixture.componentInstance.selectedHero).toBeNull();
        });
    }));

    it("Should have an array of heros", async(()=>
        this.fixture.whenStable().then(()=> {
            const cmp = this.fixture.componentInstance;
            expect(cmp.heroes).toBeDefined("component should have a list of heroes");
            expect(cmp.heroes.length).toEqual(10, "heroes list should have 10 members");
            cmp.heroes.map((h, i)=> {
                expect(h instanceof Hero).toBeTruthy(`member ${i} is not a Hero instance.
${h}`)
            });
        })));

        it("Should have one list item per hero", async(()=>
        this.fixture.whenStable().then(()=> {
            const ul = this.fixture.nativeElement.querySelector("ul.heroes");
            const li = Array.prototype.slice.call(
                this.fixture.nativeElement.querySelectorAll("ul.heroes>li"));
            const cmp = this.fixture.componentInstance;
            expect(ul).toBeTruthy("There should be an unnumbered list for heroes");
            expect(li.length).toEqual(cmp.heroes.length, "there should be one li for each
hero");
            li.forEach((li, i)=> {
                expect(li.querySelector("span.badge"))
                    .toBeTruthy(`hero ${i} has to have a span for id`);
                expect(li.querySelector("span.badge").textContent.trim())
                    .toEqual(cmp.heroes[i].id.toString(), `hero ${i} had wrong id displayed`);
                expect(li.textContent)
                    .toMatch(cmp.heroes[i].name, `hero ${i} has wrong name displayed`);
            });
        })));

    it("should have correct styling of hero items", async(()=>
        this.fixture.whenStable().then(()=> {
            const hero = this.fixture.nativeElement.querySelector("ul.heroes>li");
            const win = hero.ownerDocument.defaultView ||hero.ownerDocument.parentWindow;
            const styles = win.getComputedStyle(hero);
            expect(styles["cursor"]).toEqual("pointer", "cursor should be pointer on hero");
            expect(styles["borderRadius"]).toEqual("4px", "borderRadius should be 4px");
        })));

    it("should have a click handler for hero items",async(()=>
        this.fixture.whenStable().then(()=>{
            const cmp = this.fixture.componentInstance;
            expect(cmp.onSelect)
                .toBeDefined("should have a click handler for heros");
            expect(this.fixture.nativeElement.querySelector("input.heroName"))
                .toBeNull("should not show the hero details when no hero has been selected");
            expect(this.fixture.nativeElement.querySelector("ul.heroes li.selected"))
                .toBeNull("Should not have any selected heroes at start");

            spyOn(cmp,"onSelect").and.callThrough();
            this.fixture.nativeElement.querySelectorAll("ul.heroes li")[5].click();
```

```
            expect(cmp.onSelect)
                .toHaveBeenCalledWith(cmp.heroes[5]);
            expect(cmp.selectedHero)
                .toEqual(cmp.heroes[5], "click on hero should change hero");
        })
    ));
});
```

Cabe destacar cómo tenemos antes que **cada ()** configurar un módulo de prueba y crear el componente en prueba, y cómo llamamos **detectChanges ()** para que el ángulo pase por el enlace doble y todo.

Tenga en cuenta que cada prueba es una llamada a **async ()** y siempre espera a que la promesa de **WhenStable se** resuelva antes de examinar el dispositivo. Luego tiene acceso al componente a través de **componentInstance** y al elemento a través de **nativeElement** .

Hay una prueba que está comprobando el estilo correcto. como parte del Tutorial, el equipo de Angular demuestra el uso de estilos dentro de los componentes. En nuestra prueba, usamos **getComputedStyle ()** para verificar que los estilos provienen de donde especificamos, sin embargo, necesitamos el objeto Window para eso, y lo obtenemos del elemento como se puede ver en la prueba.

Servicio de prueba HTTP

Por lo general, los servicios llaman a Api remoto para recuperar / enviar datos. Pero las pruebas unitarias no deberían hacer llamadas de red. Angular utiliza `XHRBackend` clase `XHRBackend` para hacer solicitudes http. El usuario puede anular esto para cambiar el comportamiento. El módulo de prueba angular proporciona las clases `MockBackend` y `MockConnection` que se pueden usar para probar y afirmar las solicitudes http.

`posts.service.ts` Este servicio llega a un punto final de API para recuperar la lista de publicaciones.

```
import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable }     from 'rxjs/rx';

import 'rxjs/add/operator/map';

export interface IPost {
    userId: number;
    id: number;
    title: string;
    body: string;
}

@Injectable()
export class PostsService {
    posts: IPost[];

    private postsUri = 'http://jsonplaceholder.typicode.com/posts';
```

```
    constructor(private http: Http) {
    }

    get(): Observable<IPost[]> {
        return this.http.get(this.postsUri)
                .map((response) => response.json());
    }
}
```

`posts.service.spec.ts` Aquí, `posts.service.spec.ts` servicio anterior `posts.service.spec.ts` llamadas a la API de HTTP.

```
import { TestBed, inject, fakeAsync } from '@angular/core/testing';
import {
    HttpModule,
    XHRBackend,
    ResponseOptions,
    Response,
    RequestMethod
} from '@angular/http';
import {
    MockBackend,
    MockConnection
} from '@angular/http/testing';

import { PostsService } from './posts.service';

describe('PostsService', () => {
    // Mock http response
    const mockResponse = [
        {
            'userId': 1,
            'id': 1,
            'title': 'sunt aut facere repellat provident occaecati excepturi optio
reprehenderit',
            'body': 'quia et suscipit\nsuscipit recusandae consequuntur expedita et
cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet
architecto'
        },
        {
            'userId': 1,
            'id': 2,
            'title': 'qui est esse',
            'body': 'est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea
dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui
aperiam non debitis possimus qui neque nisi nulla'
        },
        {
            'userId': 1,
            'id': 3,
            'title': 'ea molestias quasi exercitationem repellat qui ipsa sit aut',
            'body': 'et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut
ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et
velit aut'
        },
        {
            'userId': 1,
            'id': 4,
            'title': 'eum et est occaecati',
            'body': 'ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda
```

```
provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt
voluptatem rerum illo velit'
        }
    ];

    beforeEach(() => {
        TestBed.configureTestingModule({
            imports: [HttpModule],
            providers: [
                {
                    provide: XHRBackend,
                    // This provides mocked XHR backend
                    useClass: MockBackend
                },
                PostsService
            ]
        });
    });

    it('should return posts retrieved from Api', fakeAsync(
        inject([XHRBackend, PostsService],
            (mockBackend, postsService) => {
                mockBackend.connections.subscribe(
                    (connection: MockConnection) => {
                        // Assert that service has requested correct url with expected method
                        expect(connection.request.method).toBe(RequestMethod.Get);

expect(connection.request.url).toBe('http://jsonplaceholder.typicode.com/posts');
                        // Send mock response
                        connection.mockRespond(new Response(new ResponseOptions({
                            body: mockResponse
                        })));
                    });

                postsService.get()
                    .subscribe((posts) => {
                        expect(posts).toBe(mockResponse);
                    });

            })));
});
```

Pruebas de componentes angulares - Básico

El código del componente se da a continuación.

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

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent{
  title = 'welcome';
}
```

Para la prueba angular, angular proporciona sus utilidades de prueba junto con el marco de prueba que ayuda a escribir el buen caso de prueba en angular. Las utilidades angulares se

pueden importar desde `@angular/core/testing`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { MyAppComponent } from './banner-inline.component';

describe('Tests for MyAppComponent', () => {

  let fixture: ComponentFixture<MyAppComponent>;
  let comp: MyAppComponent;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        MyAppComponent
      ]
    });
  });

  beforeEach(() => {

    fixture = TestBed.createComponent(MyAppComponent);
    comp = fixture.componentInstance;

  });

  it('should create the MyAppComponent', () => {

      expect(comp).toBeTruthy();

  });

});
```

En el ejemplo anterior, solo hay un caso de prueba que explica el caso de prueba para la existencia de componentes. En el ejemplo anterior, se utilizan utilidades de prueba angular como `TestBed` y `ComponentFixture` .

`TestBed` se utiliza para crear el módulo de prueba angular y configuramos este módulo con el método `configureTestingModule` para producir el entorno de módulo para la clase que queremos probar. El módulo de prueba debe configurarse antes de la ejecución de cada caso de prueba, por eso configuramos el módulo de prueba en la función `beforeEach` .

`createComponent` método `createComponent` de `TestBed` se utiliza para crear la instancia del componente bajo prueba. `createComponent` devuelve el `ComponentFixture` . El accesorio proporciona acceso a la instancia del componente en sí.

Lea Probando una aplicación Angular 2 en línea:
https://riptutorial.com/es/angular2/topic/2329/probando-una-aplicacion-angular-2

Capítulo 62: redux angular

Examples

BASIC

app.module.ts

```
import {appStoreProviders} from "./app.store";
providers : [
  ...
  appStoreProviders,
  ...
]
```

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";



export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
    window['devToolsExtension'] ?
    window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

app.reducer.ts

```
export interface AppState {
  example : string
}

const rootReducer: Reducer<AppState> = combineReducers<AppState>({
 example : string
});

export default rootReducer;
```

tienda.ts

```
export interface IAppState {
  example?: string;
}

export const INITIAL_STATE: IAppState = {
  example: null,
};

export function rootReducer(state: IAppState = INITIAL_STATE, action: Action): IAppState {
  switch (action.type) {
    case EXAMPLE_CHANGED:
      return Object.assign(state, state, (<UpdateAction>action));
    default:
      return state;
  }
}
```

acciones.ts

```
import {Action} from "redux";
export const EXAMPLE_CHANGED = 'EXAMPLE CHANGED';

export interface UpdateAction extends Action {
  example: string;
}
```

Obtener estado actual

```
import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
    constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
    getExampleState(){
      console.log(this.store.getState().example);
   }
}
```

cambian de estado

```
import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
    constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
    setExampleState(){
       this.store.dispatch(updateExample("new value"));
   }
}
```

acciones.ts

```
export interface UpdateExapleAction extends Action {
```

```
  example?: string;
}

export const updateExample: ActionCreator<UpdateExapleAction> =
   (newVal) => ({
     type: EXAMPLE_CHANGED,
     example: newVal
});
```

Añadir herramienta de cromo redux

app.store.ts

```
 import {InjectionToken} from '@angular/core';
 import {createStore, Store, compose, StoreEnhancer} from 'redux';
 import {AppState, default as reducer} from "../app.reducer";


 export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
     window['devToolsExtension'] ?
     window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
   return createStore<AppState>(
     reducer,
     compose(devtools)
   );
}

    export const appStoreProviders = [
      {provide: AppStore, useFactory: createAppStore}
    ];
```

instalar redux DevTools chrome extention

Lea redux angular en línea: https://riptutorial.com/es/angular2/topic/10652/redux-angular

Capítulo 63: Servicio EventEmitter

Examples

Resumen de la clase

```
class EventEmitter extends Subject {
    constructor(isAsync?: boolean)
    emit(value?: T)
    subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

Componente de clase

```
@Component({
  selector: 'zippy',
  template: `
  <div class="zippy">
    <div (click)="toggle()">Toggle</div>
    <div [hidden]="!visible">
      <ng-content></ng-content>
    </div>
 </div>`})
export class Zippy {
  visible: boolean = true;
  @Output() open: EventEmitter<any> = new EventEmitter();
  @Output() close: EventEmitter<any> = new EventEmitter();
  toggle() {
    this.visible = !this.visible;
    if (this.visible) {
      this.open.emit(null);
    } else {
      this.close.emit(null);
    }
  }
}
```

Eventos emisores

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Atrapando el evento

Crear un servicio-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
    navchange: EventEmitter<number> = new EventEmitter();
    constructor() {}
    emitNavChangeEvent(number) {
        this.navchange.emit(number);
```

```
    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}
```

Crear un componente para usar el servicio.

```
import {Component} from 'angular2/core';
import {NavService} from '../services/NavService';

@Component({
    selector: 'obs-comp',
    template: `obs component, item: {{item}}`
    })
    export class ObservingComponent {
    item: number = 0;
    subscription: any;
    constructor(private navService:NavService) {}
    ngOnInit() {
        this.subscription = this.navService.getNavChangeEmitter()
        .subscribe(item => this.selectedNavItem(item));
    }
    selectedNavItem(item: number) {
        this.item = item;
    }
    ngOnDestroy() {
        this.subscription.unsubscribe();
    }
}

@Component({
    selector: 'my-nav',
    template:`
        <div class="nav-item" (click)="selectedNavItem(1)">nav 1 (click me)</div>
        <div class="nav-item" (click)="selectedNavItem(2)">nav 2 (click me)</div>
    `,
})
export class Navigation {
    item = 1;
    constructor(private navService:NavService) {}
    selectedNavItem(item: number) {
        console.log('selected nav item ' + item);
        this.navService.emitNavChangeEvent(item);
    }
}
```

Ejemplo vivo

Un ejemplo vivo de esto se puede encontrar aquí .

Lea Servicio EventEmitter en línea: https://riptutorial.com/es/angular2/topic/9159/servicio-eventemitter

Capítulo 64: Sujetos y observables angulares RXJS con solicitudes API

Observaciones

Hacer solicitudes de API con el servicio Angular 2 Http y RxJS es muy similar a trabajar con promesas en Angular 1.x.

Usa la clase Http para hacer peticiones. La clase Http expone los métodos para emitir solicitudes HTTP `GET` , `POST` , `PUT` , `DELETE` , `PATCH` , `HEAD` a través de los métodos correspondientes. También expone un método de `request` genérico para emitir cualquier tipo de solicitud HTTP.

Todos los métodos de la clase `Http` devuelven un `Observable<Response>` , al que puede aplicar operaciones RxJS . Se llama al método `.subscribe()` y se pasa una función a la que se llamará cuando se devuelvan datos en el flujo observable.

El flujo observable para una solicitud contiene solo un valor: la `Response` , y se completa / resuelve cuando la solicitud HTTP se completa con éxito, o errores / fallas si se produce un error.

Tenga en cuenta que los observables devueltos por el módulo `Http` son *fríos* , lo que significa que si se suscribe al observable varias veces, la solicitud de origen se *ejecutará* una vez por cada suscripción. Esto puede suceder si desea consumir el resultado en varios componentes de su aplicación. Para solicitudes GET, esto podría causar algunas solicitudes adicionales, pero esto puede crear resultados inesperados si se suscribe más de una vez a las solicitudes PUT o POST.

Examples

Solicitud basica

El siguiente ejemplo muestra una solicitud HTTP GET simple. `http.get()` devuelve un `Observable` que tiene el método `subscribe` . Este anexa los datos devueltos a la matriz de `posts` .

```
var posts = []

getPosts(http: Http):void {
    this.http.get(`https://jsonplaceholder.typicode.com/posts`)
        .map(response => response.json())
        .subscribe(post => posts.push(post));
}
```

Encapsular las solicitudes de API

Puede ser una buena idea encapsular la lógica de manejo de HTTP en su propia clase. La siguiente clase expone un método para obtener publicaciones. Llama al método `http.get()` y llama a `.map` en el `Observable` devuelto para convertir el objeto `Response` en un objeto `Post` .

```
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";

@Injectable()
export class BlogApi {

  constructor(private http: Http) {
  }

  getPost(id: number): Observable<Post> {
    return this.http.get(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .map((response: Response) => {
        const srcData = response.json();
        return new Post(srcData)
      });
  }
}
```

El ejemplo anterior utiliza una clase de `Post` para contener los datos devueltos, que podrían tener el siguiente aspecto:

```
export class Post {
  userId: number;
  id: number;
  title: string;
  body: string;

  constructor(src: any) {
      this.userId = src && src.userId;
      this.id = src && src.id;
      this.title = src && src.title;
      this.body = src && src.body;
  }
}
```

Un componente puede ahora utilizar el `BlogApi` clase para recuperar fácilmente `Post` de datos sin preocuparse de los trabajos de la `Http` clase.

Espera múltiples solicitudes

Un escenario común es esperar a que finalicen varias solicitudes antes de continuar. Esto se puede lograr utilizando el método `forkJoin` .

En el siguiente ejemplo, `forkJoin` se usa para llamar a dos métodos que devuelven `Observables` . La devolución de llamada especificada en el método `.subscribe` se llamará cuando ambos Observables se hayan completado. Los parámetros proporcionados por `.subscribe` coinciden con el orden dado en la llamada a `.forkJoin` . En este caso, primero `posts` luego `tags` .

```
loadData() : void {
    Observable.forkJoin(
        this.blogApi.getPosts(),
        this.blogApi.getTags()
    ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
        this.posts = posts;
        this.tags = tags;
```

```
    }));
}
```

Lea Sujetos y observables angulares RXJS con solicitudes API en línea: https://riptutorial.com/es/angular2/topic/3577/sujetos-y-observables-angulares-rxjs-con-solicitudes-api

Capítulo 65: Título de la página

Introducción

¿Cómo se puede cambiar el título de la página?

Sintaxis

- `setTitle(newTitle: string): void;`
- `getTitle(): string;`

Examples

cambiando el título de la página

1. Primero necesitamos proporcionar servicio de título.
2. Usando setTitle

```
import {Title} from "@angular/platform-browser";
@Component({
  selector: 'app',
  templateUrl: './app.component.html',
  providers : [Title]
})

export class AppComponent implements {
   constructor( private title: Title) {
     this.title.setTitle('page title changed');
   }
}
```

Lea Título de la página en línea: https://riptutorial.com/es/angular2/topic/8954/titulo-de-la-pagina

Capítulo 66: Trabajador del servicio

Introducción

Veremos cómo configurar un servicio que funciona en angular, para permitir que nuestra aplicación web tenga capacidades fuera de línea.

Un trabajador del servicio es un script especial que se ejecuta en segundo plano en el navegador y administra las solicitudes de red a un origen determinado. Se instala originalmente por una aplicación y permanece residente en la máquina / dispositivo del usuario. El navegador lo activa cuando se carga una página desde su origen y tiene la opción de responder a las solicitudes HTTP durante la carga de la página.

Examples

Añadir Service Worker a nuestra aplicación

Primero, en caso de que estés consultando mobile.angular.io, la bandera --mobile ya no funciona.

Así que para empezar, podemos crear un proyecto normal con cli angular.

```
ng new serviceWorking-example
cd serviceWorking-example
```

Ahora lo importante, para decir a Cli angular que queremos usar el trabajador de servicio que tenemos que hacer:

ng set apps.0.serviceWorker = true

Si por alguna razón no tiene instalado @ angular / service-worker, verá un mensaje:

> Su proyecto se configura con serviceWorker = true, pero @ angular / service-worker no está instalado. Ejecute `npm install --save-dev @angular/service-worker` e intente nuevamente, o ejecute `ng set apps.0.serviceWorker=false` en su .angular-cli.json.

Verifique .angular-cli.json y ahora debería ver esto: "serviceWorker": verdadero

Cuando este indicador es verdadero, las compilaciones de producción se configurarán con un trabajador de servicio.

Se generará un archivo ngsw-manifest.json (o se aumentará en caso de que tengamos que crear un ngsw-manifest.json en la raíz del proyecto, generalmente esto se hace para especificar el enrutamiento, en un futuro probablemente se hará de forma automática) en el directorio dist / root, y el script del trabajador del servicio se copiará allí. Se agregará una secuencia de comandos corta a index.html para registrar al trabajador del servicio.

Ahora si construimos la aplicación en modo de producción ng build --prod

Y compruebe dist / carpeta.

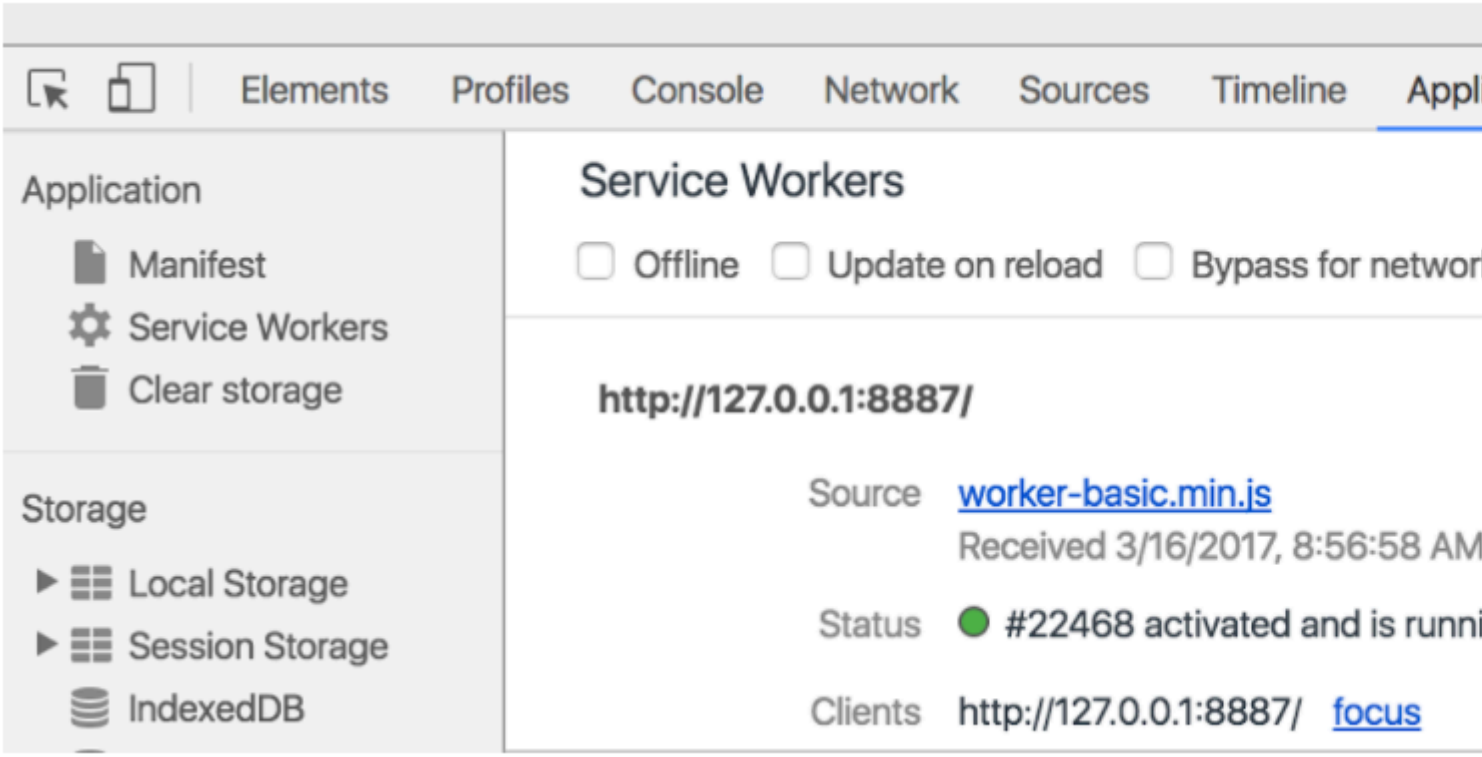
Verás tres nuevos archivos allí:

- worker-basic.min.js
- sw-register.HASH.bundle.js
- ngsw-manifest.json

Además, index.html ahora incluye esta secuencia de comandos sw-register, que registra un Trabajador de Servicio Angular (ASW) para nosotros.

Actualice la página en su navegador (servido por el servidor web para Chrome)

Abrir herramientas de desarrollo. Ir a la aplicación -> Trabajadores de servicio



¡Bien ahora el trabajador de servicio está en marcha!

Ahora nuestra aplicación debería cargarse más rápido y deberíamos poder usar la aplicación sin conexión.

Ahora, si habilita el modo sin conexión en la consola Chrome, debería ver que nuestra aplicación en http: // localhost: 4200 / index.html funciona sin conexión a Internet.

Pero en http: // localhost: 4200 / tenemos un problema y no se carga, esto se debe a que el caché de contenido estático solo sirve para los archivos listados en el manifiesto.

Por ejemplo, si el manifiesto declara una URL de /index.html, las solicitudes a /index.html serán respondidas por el caché, pero una solicitud a / o / some / route irá a la red.

Ahí es donde entra el complemento de redirección de rutas. Lee una configuración de

enrutamiento del manifiesto y redirige las rutas configuradas a una ruta de índice específica.

Actualmente, esta sección de configuración debe estar escrita a mano (19-7-2017). Eventualmente, se generará a partir de la configuración de ruta presente en la fuente de la aplicación.

Así que si ahora creamos o ngsw-manifest.json en la raíz del proyecto

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

Y construimos de nuevo nuestra aplicación, ahora cuando vamos a http: // localhost: 4200 / , deberíamos ser redirigidos a http: // localhost: 4200 / index.html .

Para más información sobre enrutamiento, lea la documentación oficial aquí.

Aquí puede encontrar más documentación sobre los trabajadores de servicios:

https://developers.google.com/web/fundamentals/getting-started/primers/service-workers

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

Y aquí puede ver una forma alternativa de implementar el servicio trabajando con la biblioteca de precache SW:

https://coryrylan.com/blog/fast-offline-angular-apps-with-service-workers

Lea Trabajador del servicio en línea: https://riptutorial.com/es/angular2/topic/10809/trabajador-del-servicio

Capítulo 67: Tubería

Introducción

El tubo | el carácter se utiliza para aplicar tuberías en Angular 2. Las tuberías son muy similares a los filtros en AngularJS, ya que ambas ayudan a transformar los datos en un formato específico.

Parámetros

Función / Parámetro	Explicación
@Pipe ({nombre, puro})	Metadatos para tubería, debe preceder inmediatamente a la clase de tubería.
nombre: *cadena*	lo que usarás dentro de la plantilla
puro: *booleano*	el valor predeterminado es verdadero, marque esto como falso para que su tubería sea reevaluada con más frecuencia
transformar (valor, args []?)	La función que se llama a transformar los valores en la plantilla.
valor: *cualquiera*	El valor que quieres transformar.
Args: *cualquiera []*	Los argumentos que puede necesitar incluir en su transformación. Marque args opcionales con el? operador como tal transformar (valor, arg1, arg2?)

Observaciones

Este tema trata sobre Angular2 Pipes , un mecanismo para transformar y formatear datos dentro de plantillas HTML en una aplicación Angular2.

Examples

Tuberías de encadenamiento

Las tuberías pueden estar encadenadas.

```
<p>Today is {{ today | date:'fullDate' | uppercase}}.</p>
```

Tubos personalizados

my.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'myPipe'})
export class MyPipe implements PipeTransform {

  transform(value:any, args?: any):string {
    let transformedValue = value; // implement your transformation logic here
    return transformedValue;
  }

}
```

my.component.ts

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

@Component({
  selector: 'my-component',
  template: `{{ value | myPipe }}`
})
export class MyComponent {

    public value:any;

}
```

my.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { MyComponent } from './my.component';
import { MyPipe } from './my.pipe';

@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [
    MyComponent,
    MyPipe
  ],
})
export class MyModule { }
```

Tubos incorporados

Angular2 viene con algunos tubos integrados:

Tubo	Uso	Ejemplo
DatePipe	date	{{ dateObj | date }} // output is 'Jun 15, 2015'

Tubo	Uso	Ejemplo
UpperCasePipe	uppercase	`{{ value | uppercase }} // output is 'SOMETEXT'`
LowerCasePipe	lowercase	`{{ value | lowercase }} // output is 'sometext'`
CurrencyPipe	currency	`{{ 31.00 | currency:'USD':true }} // output is '$31'`
PercentPipe	percent	`{{ 0.03 | percent }} //output is %3`

Hay otros. Busque aquí su documentación.

Ejemplo

hotel-booking.component.ts

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

@Component({
    moduleId: module.id,
    selector: 'hotel-reservation',
    templateUrl: './hotel-reservation.template.html'
})
export class HotelReservationComponent {
    public fName: string =  'Joe';
    public lName: string = 'SCHMO';
    public reservationMade: string = '2016-06-22T07:18-08:00'
    public reservationFor: string = '2025-11-14';
    public cost: number = 99.99;
}
```

hotel-reservation.template.html

```
<div>
    <h1>Welcome back {{fName | uppercase}} {{lName | lowercase}}</h1>
    <p>
        On {reservationMade | date} at {reservationMade | date:'shortTime'} you
        reserved room 205 for {reservationDate | date} for a total cost of
        {cost | currency}.
    </p>
</div>
```

Salida

```
Welcome back JOE schmo
On Jun 26, 2016 at 7:18 you reserved room 205 for Nov 14, 2025 for a total cost of
$99.99.
```

Depuración Con JsonPipe

El JsonPipe se puede usar para depurar el estado de cualquier interno dado.

Código

```
@Component({
  selector: 'json-example',
  template: `<div>
    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>
    <p>With JSON pipe:</p>
    <pre>{{object | json}}</pre>
  </div>`
})
export class JsonPipeExample {
  object: Object = {foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}};
}
```

Salida

```
Without JSON Pipe:
object
With JSON pipe:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

Tubería personalizada disponible a nivel mundial

Para hacer que una tubería personalizada esté disponible en toda la aplicación, Durante la aplicación bootstrap, extienda PLATFORM_PIPES.

```
import { bootstrap }    from '@angular/platform-browser-dynamic';
import { provide, PLATFORM_PIPES } from '@angular/core';

import { AppComponent } from './app.component';
import { MyPipe } from './my.pipe'; // your custom pipe

bootstrap(AppComponent, [
  provide(PLATFORM_PIPES, {
            useValue: [
                MyPipe
            ],
            multi: true
        })
]);
```

Tutorial aquí: https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2

Creación de tubería personalizada

app / pipes.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'truthy'})
export class Truthy implements PipeTransform {
```

```
  transform(value: any, truthy: string, falsey: string): any {
    if (typeof value === 'boolean'){return value ? truthy : falsey;}
    else return value
  }
}
```

app / my-component.component.ts

```
import { Truthy} from './pipes.pipe';

@Component({
  selector: 'my-component',
  template: `
    <p>{{value | truthy:'enabled':'disabled' }}</p>
  `,
  pipes: [Truthy]
})
export class MyComponent{ }
```

Desenvolver valores asíncronos con una tubería asíncrona

```
import { Component } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

@Component({
  selector: 'async-stuff',
  template: `
    <h1>Hello, {{ name | async }}</h1>
    Your Friends are:
    <ul>
      <li *ngFor="let friend of friends | async">
        {{friend}}
      </li>
    </ul>
  `
})
class AsyncStuffComponent {
  name = Promise.resolve('Misko');
  friends = Observable.of(['Igor']);
}
```

Se convierte en

```
<h1>Hello, Misko</h1>
Your Friends are:
<ul>
  <li>
    Igor
  </li>
</ul>
```

Extendiendo una tubería existente

```
import { Pipe, PipeTransform } from '@angular/core';
```

```
import { DatePipe } from '@angular/common'


@Pipe({name: 'ifDate'})
export class IfDate implements PipeTransform {
  private datePipe: DatePipe = new DatePipe();

  transform(value: any, pattern?:string) : any {
    if (typeof value === 'number') {return value}
    try {
      return this.datePipe.transform(value, pattern)
    } catch(err) {
      return value
    }
  }
}
```

Tubos de estado

Angular 2 ofrece dos tipos diferentes de tuberías: sin estado y con estado. Las tuberías son sin estado por defecto. Sin embargo, podemos implementar tuberías con estado estableciendo la propiedad `pure` en `false` . Como puede ver en la sección de parámetros, puede especificar un `name` y declarar si la tubería debe ser pura o no, es decir, sin estado o sin estado. Mientras que los datos fluyen a través de una tubería sin estado (que es una función pura) que **no** recuerda nada, los datos pueden ser gestionados y recordados por tuberías con estado. Un buen ejemplo de una tubería con estado es el `AsyncPipe` proporcionado por Angular 2.

Importante

Observe que la mayoría de las tuberías deben caer en la categoría de tuberías sin estado. Esto es importante por razones de rendimiento, ya que Angular puede optimizar tuberías sin estado para el detector de cambios. Así que use las tuberías con estado con cautela. En general, la optimización de tuberías en Angular 2 tiene una mayor mejora de rendimiento sobre los filtros en Angular 1.x. En Angular 1, el ciclo de resumen siempre tuvo que volver a ejecutar todos los filtros aunque los datos no hayan cambiado en absoluto. En Angular 2, una vez que se ha calculado el valor de una tubería, el detector de cambios sabe que no debe volver a ejecutar esta tubería a menos que cambie la entrada.

Implementación de una tubería con estado.

```
import {Pipe, PipeTransform, OnDestroy} from '@angular/core';

@Pipe({
  name: 'countdown',
  pure: false
})
export class CountdownPipe implements PipeTransform, OnDestroy  {
  private interval: any;
  private remainingTime: number;

  transform(value: number, interval: number = 1000): number {
    if (!parseInt(value, 10)) {
      return null;
    }
```

```
    if (typeof this.remainingTime !== 'number') {
      this.remainingTime = parseInt(value, 10);
    }

    if (!this.interval) {
      this.interval = setInterval(() => {
        this.remainingTime--;

        if (this.remainingTime <= 0) {
          this.remainingTime = 0;
          clearInterval(this.interval);
          delete this.interval;
        }
      }, interval);
    }

    return this.remainingTime;
  }

  ngOnDestroy(): void {
    if (this.interval) {
      clearInterval(this.interval);
    }
  }
}
```

A continuación, puede utilizar la tubería como de costumbre:

```
{{ 1000 | countdown:50 }}
{{ 300 | countdown }}
```

Es importante que su tubería también implemente la interfaz `OnDestroy` para que pueda limpiar una vez que se destruya su tubería. En el ejemplo anterior, es necesario borrar el intervalo para evitar pérdidas de memoria.

Tubo dinámico

Escenario de caso de uso: una vista de tabla consta de diferentes columnas con diferentes formatos de datos que deben transformarse con diferentes canalizaciones.

table.component.ts

```
...
import { DYNAMIC_PIPES } from '../pipes/dynamic.pipe.ts';

@Component({
    ...
    pipes: [DYNAMIC_PIPES]
})
export class TableComponent {
    ...

    // pipes to be used for each column
    table.pipes = [ null, null, null, 'humanizeDate', 'statusFromBoolean' ],
    table.header = [ 'id', 'title', 'url', 'created', 'status' ],
```

```
    table.rows = [
        [ 1, 'Home', 'home', '2016-08-27T17:48:32', true ],
        [ 2, 'About Us', 'about', '2016-08-28T08:42:09', true ],
        [ 4, 'Contact Us', 'contact', '2016-08-28T13:28:18', false ],
        ...
    ]
    ...

}
```

dynamic.pipe.ts

```
import {
    Pipe,
    PipeTransform
} from '@angular/core';
// Library used to humanize a date in this example
import * as moment from 'moment';

@Pipe({name: 'dynamic'})
export class DynamicPipe implements PipeTransform {

    transform(value:string, modifier:string) {
        if (!modifier) return value;
        // Evaluate pipe string
        return eval('this.' + modifier + '(\'' + value + '\')')
    }

    // Returns 'enabled' or 'disabled' based on input value
    statusFromBoolean(value:string):string {
        switch (value) {
            case 'true':
            case '1':
                return 'enabled';
            default:
                return 'disabled';
        }
    }

    // Returns a human friendly time format e.g: '14 minutes ago', 'yesterday'
    humanizeDate(value:string):string {
        // Humanize if date difference is within a week from now else returns 'December 20,
2016' format
        if (moment().diff(moment(value), 'days') < 8) return moment(value).fromNow();
        return moment(value).format('MMMM Do YYYY');
    }
}

export const DYNAMIC_PIPES = [DynamicPipe];
```

table.component.html

```
<table>
    <thead>
        <td *ngFor="let head of data.header">{{ head }}</td>
    </thead>
    <tr *ngFor="let row of table.rows; let i = index">
        <td *ngFor="let column of row">{{ column | dynamic:table.pipes[i] }}</td>
    </tr>
```

```
</table>
```

Resultado

```
| ID | Page Title     | Page URL    | Created          | Status     |
-------------------------------------------------------------------
|  1 | Home           | home        | 4 minutes ago    | Enabled    |
|  2 | About Us       | about       | Yesterday        | Enabled    |
|  4 | Contact Us     | contact     | Yesterday        | Disabled   |
-------------------------------------------------------------------
```

Probando un tubo

Dada una tubería que invierte una cuerda.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}
```

Se puede probar configurando el archivo de especificaciones como este.

```
import { TestBed, inject } from '@angular/core/testing';

import { ReversePipe } from './reverse.pipe';

describe('ReversePipe', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [ReversePipe],
    });
  });

  it('should be created', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe).toBeTruthy();
  }));

  it('should reverse a string', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe.transform('abc')).toEqual('cba');
  }));
});
```

Lea Tubería en línea: https://riptutorial.com/es/angular2/topic/1165/tuberia

Capítulo 68: Usa componentes web nativos en Angular 2

Observaciones

Cuando utiliza un componente web en su plantilla de Angular 2, angular intentará encontrar un componente con un selector que coincida con la etiqueta personalizada del componente web, que por supuesto no puede y producirá un error.

La solución es importar un "esquema de elementos personalizados" en el módulo que contiene el componente. Esto hará que Angular acepte cualquier etiqueta personalizada, que no coincida con ningún selector de componentes.

Examples

Incluye esquema de elementos personalizados en tu módulo

```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AboutComponent } from './about.component';

@NgModule({
    imports: [ CommonModule ],
    declarations: [ AboutComponent ],
    exports: [ AboutComponent ],
    schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
})

export class AboutModule { }
```

Usa tu componente web en una plantilla

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

@Component({
  selector: 'myapp-about',
  template: `<my-webcomponent></my-webcomponent>`
})
export class AboutComponent { }
```

Lea Usa componentes web nativos en Angular 2 en línea:
https://riptutorial.com/es/angular2/topic/7414/usa-componentes-web-nativos-en-angular-2

Capítulo 69: Usando bibliotecas de terceros como jQuery en Angular 2

Introducción

Al crear aplicaciones utilizando Angular 2.x, hay ocasiones en las que es necesario utilizar bibliotecas de terceros como jQuery, Google Analytics, API de JavaScript de integración de chat y etc.

Examples

Configuración mediante angular-cli

NPM

Si se instala una biblioteca externa como `jQuery` usando NPM

```
npm install --save jquery
```

Agregue ruta de script en su `angular-cli.json`

```
"scripts": [
    "../node_modules/jquery/dist/jquery.js"
]
```

Carpeta de activos

También puede guardar el archivo de biblioteca en su directorio de `assets/js` e incluirlo en `angular-cli.json`

```
"scripts": [
    "assets/js/jquery.js"
]
```

Nota

Guarde su biblioteca principal `jquery` y sus dependencias, como `jquery-cycle-plugin` en el directorio activos y añadir a ambos en `angular-cli.json` , asegúrese de que se mantiene el orden de las dependencias.

Usando jQuery en componentes angulares 2.x

Para usar `jquery` en sus componentes de Angular 2.x, declare una variable global en la parte superior

Si usa `$` para jQuery

```
declare var $: any;
```

Si usas `jQuery` para jQuery

```
declare var jQuery: any
```

Esto permitirá usar `$` o `jQuery` en su componente Angular 2.x.

Lea Usando bibliotecas de terceros como jQuery en Angular 2 en línea: https://riptutorial.com/es/angular2/topic/9285/usando-bibliotecas-de-terceros-como-jquery-en-angular-2

Capítulo 70: Zone.js

Examples

Obtener referencia a NgZone

`NgZone` referencia de `NgZone` se puede inyectar a través de la inyección de dependencia (DI).

my.component.ts

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

@Component({...})
export class Mycomponent implements NgOnInit {
  constructor(private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      // Do something outside Angular so it won't get noticed
    });
  }
}
```

Uso de NgZone para realizar múltiples solicitudes HTTP antes de mostrar los datos

`runOutsideAngular` se puede usar para ejecutar código fuera de Angular 2 para que no active la detección de cambios innecesariamente. Esto se puede usar para, por ejemplo, ejecutar varias solicitudes HTTP para obtener todos los datos antes de representarlos. Para ejecutar código de nuevo en el interior angular 2, `run` el método de `NgZone` se puede utilizar.

my.component.ts

```
import { Component, OnInit, NgZone } from '@angular/core';
import { Http } from '@angular/http';

@Component({...})
export class Mycomponent implements OnInit {
  private data: any[];
  constructor(private http: Http, private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      this.http.get('resource1').subscribe((data1:any) => {
        // First response came back, so its data can be used in consecutive request
        this.http.get(`resource2?id=${data1['id']}`).subscribe((data2:any) => {
          this.http.get(`resource3?id1=${data1['id']}&id2=${data2}`).subscribe((data3:any) =>
{
            this._ngZone.run(() => {
              this.data = [data1, data2, data3];
            });
          });
        });
      });
```

```
    });
  }
}
```

Lea Zone.js en línea: https://riptutorial.com/es/angular2/topic/4184/zone-js

Creditos

S. No	Capítulos	Contributors
1	Empezando con Angular 2	acdcjunior, Alexander Ciesielski, beagleknight, Bean0341, Bhoomi Bhalani, BogdanC, briantyler , cDecker32, Christopher Moore, Community, daniellmb, drbishop, echonax, Ekin Yücel, elliot-j , etayluz, ettanany, Everettss, H. Pauwelyn, Harry, He11ion, Janco Boscan, Jim, Kaspars Bergs, Logan H, Madhu Ranjan, michaelbahr, Michal Pietraszko, Mihai, nick, Nicolas Irisarri, Peter, QoP, rickysullivan, Shahzad, spike, theblindprophet, user6939352
2	Actualizar mecanografías	kEpEx
3	Agregue componentes dinámicamente usando ViewContainerRef.createComponent	amansoni211, daniellmb, Günter Zöchbauer, jupiter24, Khaled
4	Angular - ForLoop	aholtry, Anil Singh, Berseker59, gerl, Johan Van de Merwe, ob1, Pujan Srivastava, Stephen Leppik, Yoav Schniederman
5	Angular 2 - Transportador	Yoav Schniederman
6	Angular 2 Cambio de detección y activación manual.	Yoav Schniederman
7	Angular 2 Data Driven Forms	MatWaligora, ThunderRoid
8	Angular 2 Formas de actualización	Amit kumar, Anil Singh, Christopher Taylor, Highmastdon, Johan Van de Merwe, K3v1n, Manmeet Gill, mayur, Norsk, Sachin S, victoroniibukun, vijaykumar, Yoav Schniederman
9	Angular2 animaciones	Yoav Schniederman
10	Angular2 CanActivate	Companjo, Yoav Schniederman
11	Angular2 en la memoria web de la API	Jaime Still
12	Angular2 Entrada () salida ()	Kaloyan, Yoav Schniederman

13	Angular2 proporciona datos externos a la aplicación antes de bootstrap	Ajey
14	Angular2 utilizando webpack	luukgruijs
15	Angular2 Validaciones personalizadas	Arnold Wiersma, Norsk, Yoav Schniederman
16	Angular-cli	BogdanC, Yoav Schniederman
17	Animación	Gaurav Mukherjee, Nate May
18	Barril	TechJhola
19	Bootstrap módulo vacío en angular 2	AryanJ-NYC, autoboxer, Berseker59, Eric Jimenez, Krishan, Sanket, snorkpete
20	cobertura de la prueba angular-cli	ahmadalibaloch
21	Cómo usar ngfor	Jorge, Yoav Schniederman
22	Cómo usar ngif	Amit kumar, ob1, ppovoski, samAlvin
23	Compilación anticipada (AOT) con Angular 2	Anil Singh, Eric Jimenez, Harry, Robin Dijkhof
24	Componentes	BrunoLM
25	Configuración de la aplicación ASP.net Core para trabajar con Angular 2 y TypeScript	Oleksii Aza, Sam
26	Creación de una biblioteca de npm Angular	Maciej Treder
27	Crear un paquete Angular 2+ NPM	BogdanC, Janco Boscan, vinagreti
28	CRUD en Angular2 con API Restful	bleakgadfly, Sefa
29	custom ngx-bootstrap datepicker + entrada	Yoav Schniederman
30	Depuración de la aplicación mecanografiada Angular2 utilizando código de Visual Studio	PSabuwala
31	Detectar eventos de redimensionamiento	Eric Jimenez
32	Directivas	acdcjunior, Andrei Zhytkevich, borislemke,

		BrunoLM, daniellmb, Everettss, lexith, Stian Standahl, theblindprophet
33	Directivas y componentes: @Input @Output	acdcjunior, dafyddPrys, Everettss, Joel Almeida, lexith, muetzerich, theblindprophet, ThomasP1988
34	Directivas y servicios comúnmente incorporados.	Jim, Sanket
35	Directrices de atributos para afectar el valor de las propiedades en el nodo host mediante el decorador @HostBinding.	Max Karpovets
36	Diseño de material angular	Ketan Akbari, Shailesh Ladumor
37	Dropzone en Angular2	Ketan Akbari
38	Ejemplo para rutas como / route / subruta para urls estáticas	Yoav Schniederman
39	Ejemplos de componentes avanzados	borislemke, smnbbrv
40	Encuadernación de datos Angular2	Yoav Schniederman
41	Enrutamiento	aholtry, Apmis, AryanJ-NYC, borislemke, camwhite, Kaspars Bergs, LordTribual, Sachin S , theblindprophet
42	Enrutamiento (3.0.0+)	Ai_boy, Alexis Le Gal, Everettss, Gerard Simpson, Kaspars Bergs, mast3rd3mon, meorfi, rivanov, SlashTag, smnbbrv, theblindprophet, ThomasP1988, Trent
43	examen de la unidad	Yoav Schniederman
44	Ganchos de ciclo de vida	Alexandre Junges, daniellmb, Deen John, muetzerich, Sbats, theblindprophet
45	Instalar complementos de terceros con angular-cli@1.0.0-beta.10	Alex Morales, Daredzik, filoxo, Kaspars Bergs, pd farhad
46	Interacciones de los componentes	H. Pauwelyn, Janco Boscan, LLL, Sefa
47	Interceptor Http	Everettss, Mihai, Mike Kovetsky, Nilz11, Paul Marshall, peeskillet, theblindprophet
48	Inyección de Dependencia y	BrunoLM, Eduardo Carísio, Kaspars Bergs,

	Servicios.	Matrim, Roope Hakulinen, Syam Pradeep, theblindprophet
49	Mejora de fuerza bruta	Jim, Treveshan Naidoo
50	Mocking @ ngrx / Tienda	BrianRT, Hatem, Jim, Lucas, Yoav Schniederman
51	Módulos	BrunoLM
52	Módulos de funciones	AryanJ-NYC, gsc
53	ngrx	Maxime
54	Omitir la desinfección para valores de confianza	Scrambo
55	Optimización de la representación mediante ChangeDetectionStrategy	daniellmb, Eric Jimenez, Everettss
56	Orden por pipa	Yoav Schniederman
57	Perezoso cargando un modulo	Batajus, M4R1KU, Shannon Young, Syam Pradeep
58	Plantillas	Max Karpovets
59	Probando ngModel	jesussegado
60	Probando una aplicación Angular 2	Arun Redhu, michaelbahr, nick, Reza, Rumit Parakhiya
61	redux angular	Yoav Schniederman
62	Servicio EventEmitter	Abrar Jahin
63	Sujetos y observables angulares RXJS con solicitudes API	daniellmb, Maciej Treder, Ronald Zarīts, Sam Storie, Sébastien Temprado, willydee
64	Título de la página	Yoav Schniederman
65	Trabajador del servicio	Roberto Fernandez
66	Tubería	acdcjunior, Boris, borislemke, BrunoLM, Christopher Taylor, Chybie, daniellmb, Daredzik, elliot-j, Everettss, Fredrik Lundin, Jarod Moser, Jeff Cross, Jim, Kaspars Bergs, Leon Adler, Lexi , LordTribual, michaelbahr, Philipp Kief, theblindprophet
67	Usa componentes web nativos en	ugreen

	Angular 2	
68	Usando bibliotecas de terceros como jQuery en Angular 2	Ashok Vishwakarma
69	Zone.js	Roope Hakulinen