EBook Gratuito

APPRENDIMENTO
Angular 2

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular2

Sommario

Di

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

Capitolo 1: Iniziare con Angular 2

Osservazioni

Questa sezione fornisce una panoramica su come installare e configurare Angular2 + per l'uso in vari ambienti e gli strumenti IDE che utilizzano come la comunità sviluppata angular-cli .

La versione precedente di Angular è AngularJS o anche denominata Angular 1. Vedi qui la documentazione .

Versioni

Versione	Data di rilascio
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

Versione	Data di rilascio
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

Installare angular2 con angular-cli

Questo esempio è una rapida configurazione di Angular 2 e come generare un progetto di esempio rapido.

Prerequisiti:

- Node.js v4 o versioni successive .
- npm v3 o maggiore o filato .

Apri un terminale ed esegui i comandi uno a uno:

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

o

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

a seconda della scelta del gestore pacchetti.

Il comando precedente installa @ **angular / cli a** livello globale, aggiungendo l'eseguibile `ng` a PATH.

Per impostare un nuovo progetto

Navigare con il terminale in una cartella in cui si desidera impostare il nuovo progetto.

Esegui i comandi:

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

Cioè, ora hai un semplice progetto di esempio realizzato con Angular 2. Ora puoi navigare al link

visualizzato nel terminale e vedere cosa sta facendo.

Per aggiungere a un progetto esistente

Passa alla radice del tuo progetto attuale.

Esegui il comando:

```
ng init
```

Questo aggiungerà l'impalcatura necessaria al tuo progetto. I file verranno creati nella directory corrente, quindi assicurati di eseguirlo in una directory vuota.

Esecuzione del progetto a livello locale

Per vedere e interagire con la tua applicazione mentre è in esecuzione nel browser, devi avviare un server di sviluppo locale che ospita i file per il tuo progetto.

```
ng serve
```

Se il server è stato avviato correttamente, dovrebbe visualizzare un indirizzo al quale è in esecuzione il server. Di solito è questo:

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

Subito dopo questo server di sviluppo locale è collegato a Hot Module Reloading, quindi qualsiasi modifica a html, typescript o css attiverà il browser per essere ricaricato automaticamente (ma può essere disabilitato se lo si desidera).

Generazione di componenti, direttive, condotte e servizi

Il comando `ng generate <scaffold-type> <name>` (o semplicemente `ng g <scaffold-type> <name>`) consente di generare automaticamente componenti angolari:

```
# 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
```

Esistono diversi tipi di scaffold che angular-cli può generare:

Tipo di impalcatura	uso
Modulo	`ng g module my-new-module`
Componente	`ng g component my-new-component`
Direttiva	`ng g directive my-new-directive`
Tubo	`ng g pipe my-new-pipe`
Servizio	`ng g service my-new-service`
Classe	`ng g class my-new-class`
Interfaccia	`ng g interface my-new-interface`
enum	`ng g enum my-new-enum`

Puoi anche sostituire il nome del tipo con la sua prima lettera. Per esempio:

`ng gm my-new-module` per generare un nuovo modulo o `ng gc my-new-component` per creare un componente.

Costruire / Bundling

Quando hai finito di costruire la tua app Web Angular 2 e desideri installarla su un server Web come Apache Tomcat, tutto ciò che devi fare è eseguire il comando build con o senza il set di flag di produzione. La produzione minimizza il codice e ottimizza per un ambiente di produzione.

```
ng build
```

o

```
ng build --prod
```

Quindi cerca nella directory root dei progetti una cartella `/dist` , che contiene la build.

Se desideri i vantaggi di un bundle di produzione più piccolo, puoi anche utilizzare la compilazione di modelli Ahead-of-Time, che rimuove il compilatore di modelli dalla build finale:

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

Test unitario

Angular 2 fornisce test unità integrati e ogni elemento creato da angular-cli genera un test unitario di base, che può essere ampliato. I test unitari sono scritti usando il gelsomino ed eseguiti attraverso Karma. Per iniziare il test, eseguire il seguente comando:

```
ng test
```

Questo comando eseguirà tutti i test nel progetto e li eseguirà di nuovo ogni volta che un file di origine cambia, sia che si tratti di un test o di un codice dell'applicazione.

Per maggiori informazioni visita anche: pagina github angular-cli

Iniziare con Angular 2 senza angular-cli.

Angolare 2.0.0-rc.4

In questo esempio creeremo un "Hello World!" app con un solo componente root (`AppComponent`) per semplicità.

Prerequisiti:

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

> **Nota:** è possibile verificare le versioni eseguendo il `node -v` e `npm -v` nella console / terminale.

Passo 1

Crea e inserisci una nuova cartella per il tuo progetto. Chiamiamolo `angular2-example` .

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

Passo 2

Prima di iniziare a scrivere il nostro codice app, aggiungeremo i 4 file forniti di seguito: `package.json` , `tsconfig.json` , `typings.json` e `systemjs.config.js` .

> **Dichiarazione di non responsabilità:** gli stessi file possono essere trovati nel Quickstart ufficiale di 5 minuti .

`package.json` - Ci consente di scaricare tutte le dipendenze con npm e offre una semplice esecuzione di script per semplificare la vita di progetti semplici. (Dovresti considerare di usare qualcosa come Gulp in futuro per automatizzare le attività).

```
{
  "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 il traspolatore TypeScript.

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

`typings.json` - Rende a TypeScript le librerie che stiamo 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 (puoi anche usare il 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);
```

Passaggio 3

Installiamo le dipendenze digitando

```
npm install
```

nella console / terminale.

Passaggio 4

Crea `index.html` all'interno della cartella di `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>
```

La tua applicazione verrà visualizzata tra i tag `my-app` .

Tuttavia, Angular continua a non sapere *cosa* renderizzare. Per dirlo, definiremo `AppComponent` .

Passaggio 5

Crea una sottocartella chiamata `app` cui possiamo definire i componenti e i servizi che compongono la nostra app. (In questo caso, sarà solo contiene `AppComponent` codice e `main.ts` .)

```
mkdir app
```

Passaggio 6

Creare l' `app/app.component.ts` file `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"
    ];
}
```

Cosa sta succedendo? Per prima cosa, stiamo importando il decoratore `@Component` che usiamo per dare a Angular il tag e il modello HTML per questo componente. Quindi, stiamo creando la classe `AppComponent` con variabili di `title` e `messages` che possiamo utilizzare nel modello.

Ora diamo un'occhiata a quel modello:

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

Stiamo visualizzando la variabile `title` in un tag `h1` e quindi facendo una lista che mostra ogni elemento dell'array dei `messages` usando la direttiva `*ngFor` . Per ogni elemento dell'array, `*ngFor` crea una variabile di `message` che usiamo all'interno dell'elemento `li` . Il risultato sarà:

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

Passaggio 7

Ora creiamo un file `main.ts` , che sarà il primo file `main.ts` da Angular.

Crea l' `app/main.ts` file `app/main.ts`

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

bootstrap(AppComponent);
```

Stiamo importando la funzione `bootstrap` e la classe `AppComponent` , quindi utilizzando il `bootstrap` per indicare a Angular quale componente utilizzare come root.

Passaggio 8

È ora di avviare la tua prima app. genere

```
npm start
```

nella tua console / terminale. Questo eseguirà uno script preparato da `package.json` che avvia lite-server, apre l'app in una finestra del browser ed esegue il traspolatore TypeScript in modalità orologio (quindi i file `.ts` verranno transpiled e il browser si aggiornerà quando salverete le modifiche) .

E adesso?

Controlla la guida ufficiale di Angular 2 e gli altri argomenti sulla documentazione di StackOverflow .

È inoltre possibile modificare `AppComponent` per utilizzare modelli esterni, stili o aggiungere / modificare variabili componenti. Dovresti vedere le tue modifiche subito dopo aver salvato i file.

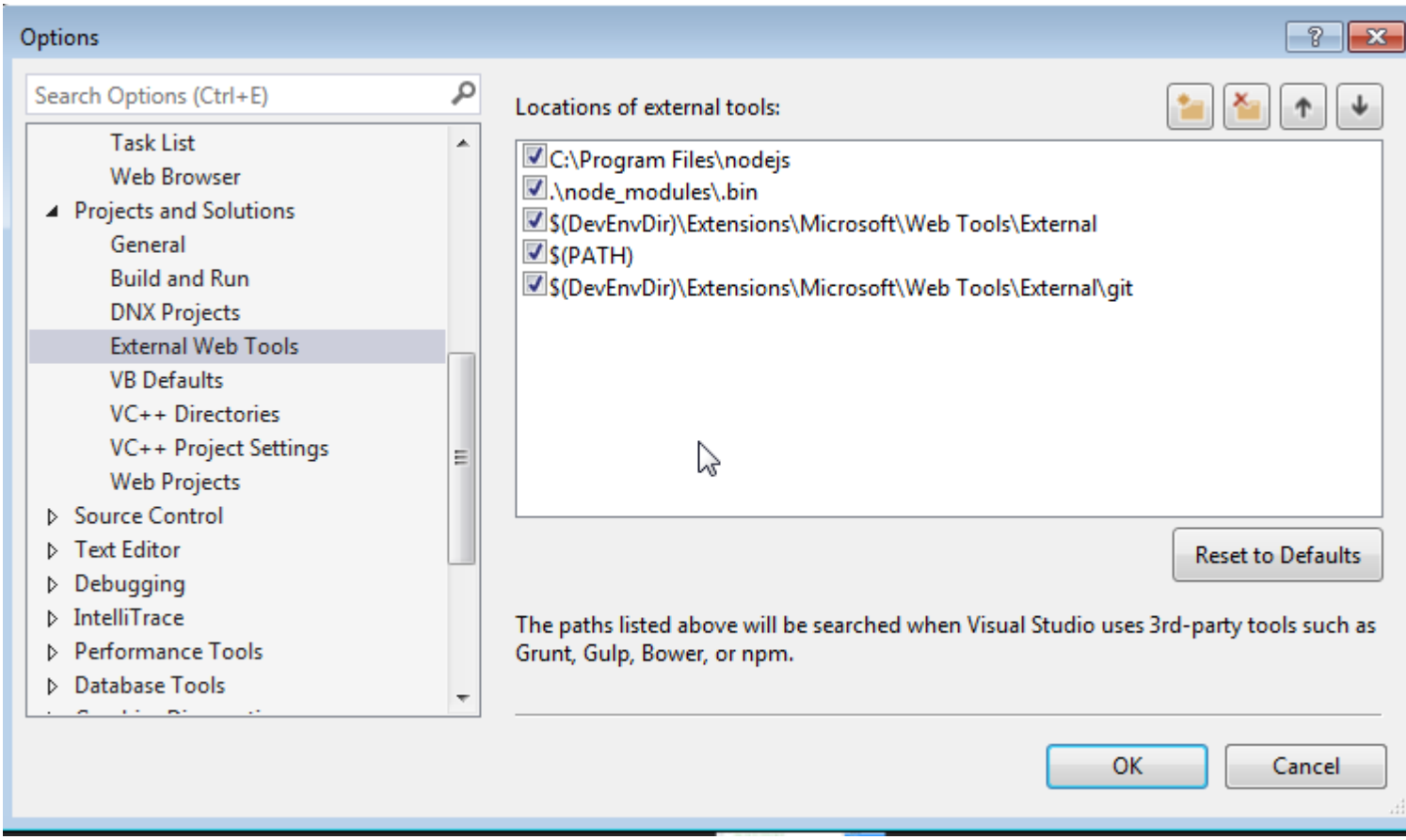
Mantenere i Visual Studio sincronizzati con gli aggiornamenti NPM e NODE

Passaggio 1: individuare il download di Node.js, in genere viene installato in C: / program files / nodejs

Passaggio 2: apri Visual Studio e vai a "Strumenti> Opzioni"

Passaggio 3: nella finestra delle opzioni, passare a "Progetti e soluzioni> Strumenti Web esterni"

Passaggio 4: aggiungere una nuova voce con il percorso del file Node.js (C: / program files / nodejs), IMPORTANTE utilizzare i pulsanti freccia nel menu per spostare il riferimento in cima all'elenco.



Passaggio 5: riavviare Visual Studios ed eseguire un'installazione npm, contro il progetto, dalla finestra di comando npm

Superare quel fastidioso proxy aziendale

Se stai tentando di ottenere un sito Angular2 in esecuzione sul tuo computer Windows su XYZ MegaCorp, è probabile che tu stia riscontrando problemi nel superare il proxy aziendale.

Ci sono (almeno) due gestori di pacchetti che devono passare attraverso il proxy:

1. NPM
2. tipizzazioni

Per NPM è necessario aggiungere le seguenti righe al file `.npmrc` :

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

Per Digitazioni è necessario aggiungere le seguenti righe al file `.typingsrc` :

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

Probabilmente questi file non esistono ancora, quindi puoi crearli come file di testo vuoti. Possono essere aggiunti alla radice del progetto (stesso posto di `package.json` oppure puoi metterli in `%HOMEPATH%`

e saranno disponibili per tutti i tuoi progetti.

Il bit che non è ovvio ed è il motivo principale per cui le persone pensano che le impostazioni del proxy non funzionino è il `%5C` che è la codifica dell'URL di `\` per separare il dominio e i nomi utente. Grazie a Steve Roberts per quello: Utilizzando npm dietro proxy aziendale .pac

Iniziare con Angular 2 con node.js / expressjs backend (esempio http incluso)

Creeremo un semplice "Hello World!" app con Angular2 2.4.1 (`@NgModule` change) con un backend node.js (expressjs).

Prerequisiti

- Node.js v4.xx o successivo
- npm v3.xx o superiore o filato

Quindi eseguire `npm install -g typescript` o `yarn global add typescript` per installare typescript a livello globale

Roadmap

Passo 1

Crea una nuova cartella (e la directory principale del nostro back-end) per la nostra app. Chiamiamolo `Angular2-express` .

riga di comando :

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

Passo 2

Crea il `package.json` (per le dipendenze) e `app.js` (per il bootstrap) per la nostra app `node.js`

package.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 });
});
```

Quindi eseguire `npm install` o un `yarn npm install` per installare le dipendenze.

Ora la nostra struttura di back-end è completa. Passiamo al front-end.

Fase 3

Il nostro front-end dovrebbe trovarsi in una cartella chiamata `front` nella nostra cartella `Angular2-express` .

riga di comando:

```
mkdir front
cd front
```

Proprio come abbiamo fatto con il nostro back-end, il nostro front-end ha bisogno anche dei file di dipendenza. Andiamo avanti e creare i seguenti file: `package.json` , `systemjs.config.js` , `tsconfig.json`

package.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/*"
  ]
}
```

Quindi eseguire `npm install` o un `yarn npm install` per installare le dipendenze.

Ora che i nostri file di dipendenza sono completi. Passiamo al nostro `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>
```

Ora siamo pronti per creare il nostro primo componente. Crea una cartella denominata `app` nella nostra cartella `front` .

riga di comando:

```
mkdir app
cd app
```

Facciamo i seguenti file denominati `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);
     });
  }
}
```

Dopo questo, compila i file dattiloscritti in file javascript. Vai su 2 livelli dalla cartella corrente (all'interno della cartella Angular2-express) ed esegui il comando seguente.

riga di comando:

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

La nostra struttura di cartelle dovrebbe essere simile a:

```
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
```

Infine, all'interno della cartella Angular2-express, eseguire il comando `node app.js` nella riga di comando. Apri il tuo browser preferito e controlla `localhost:9999` per vedere la tua app.

Immergiti in Angular 4!

Angular 4 è ora disponibile! In realtà Angular usa semver dal Angular 2, che richiede che il numero maggiore venga aumentato quando vengono introdotte le modifiche di rottura. Il team di Angular

ha posticipato le funzionalità che causano modifiche alle interruzioni, che verranno rilasciate con Angular 4. Angular 3 è stato saltato per poter allineare i numeri di versione dei moduli core, poiché il router aveva già la versione 3.

Come per il team Angular, le applicazioni Angular 4 saranno meno dispendiose e più veloci di prima. Hanno un pacchetto di animazione separato dal pacchetto @ angular / core. Se qualcuno non utilizza il pacchetto di animazione, lo spazio extra di codice non finirà nella produzione. La sintassi dell'associazione modello ora supporta la sintassi di stile / else. Angular 4 è ora compatibile con la versione più recente di Typescript 2.1 e 2.2. Quindi, Angular 4 sarà più eccitante.

Ora ti mostrerò come eseguire l'installazione di Angular 4 nel tuo progetto.

Iniziamo la configurazione angolare con tre diversi modi:

Puoi usare Angular-CLI (Command Line Interface), installerà tutte le dipendenze per te.

- È possibile migrare da Angular 2 a Angular 4.
- Puoi usare github e clonare la piastra di cottura Angular4. (È il più semplice.)
- Impostazione angolare utilizzando Angular-CLI (Command Line Interface).

Prima di iniziare a utilizzare Angular-CLI, assicurarsi di aver installato il nodo nel computer. Qui, sto usando il nodo v7.8.0. Ora, apri il terminale e digita il seguente comando per Angular-CLI.

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

o

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

a seconda del gestore di pacchetti che usi.

Installiamo Angular 4 usando Angular-CLI.

```
ng new Angular4-boilerplate
```

cd Angular4-boilerplate Siamo tutti impostati per Angular 4. Il suo metodo abbastanza semplice e diretto

Impostazione angolare migrando da Angular 2 a Angular 4

Ora vediamo il secondo approccio. Ti mostrerò come migrare Angular 2 in Angular 4. Per questo hai bisogno di clonare qualsiasi progetto Angular 2 e aggiornare le dipendenze Angular 2 con la dipendenza di Angular 4 nel tuo pacchetto.json come segue:

```
"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"
   }
```

Queste sono le principali dipendenze per Angular 4. Ora è possibile installare npm e quindi npm avviare l'applicazione. Per riferimento my package.json.

Configurazione angolare dal progetto github

Prima di iniziare questo passaggio assicurati di aver installato git nel tuo computer. Apri il tuo terminale e clona l'angolare4-boilerplate usando il comando seguente:

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

Quindi installare tutte le dipendenze ed eseguirlo.

```
npm install

npm start
```

E hai finito con l'installazione di Angular 4. Tutti i passaggi sono molto semplici in modo da poterli scegliere.

Struttura della directory della piastra angolare4

```
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
```

Comprensione di base per la struttura delle directory:

Tutto il codice risiede nella cartella src.

la cartella dei mock è per i dati falsi che vengono utilizzati a scopo di test.

la cartella del modello contiene la classe e l'interfaccia utilizzata nel componente.

la cartella modules contiene l'elenco di componenti come app, login, widget ecc. Tutto il componente contiene dattiloscritto, html e css file. index.ts è per esportare tutta la classe.

la cartella servizi contiene l'elenco dei servizi utilizzati nell'applicazione. Ho separato il servizio di riposo e il servizio di componenti diversi. Nel servizio di riposo contiene diversi metodi http. Il servizio di accesso funge da mediatore tra il componente di accesso e il servizio di assistenza.

il file app.routing.ts descrive tutti i possibili percorsi per l'applicazione.

app.module.ts descrive il modulo app come componente root.

bootstrap.ts eseguirà l'intera applicazione.

la cartella webpack contiene il file di configurazione del webpack.

Il file package.json è per tutti gli elenchi di dipendenze.

karma contiene la configurazione del karma per il test dell'unità.

node_modules contiene un elenco di pacchetti di pacchetti.

Iniziamo con il componente di accesso. In 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>
```

In 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']);
      });
    }
}
```

Dobbiamo esportare questo componente in index.ts.

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

abbiamo bisogno di impostare percorsi per il login in app.routes.ts

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

Nel componente root, il file app.module.ts ti serve solo per importare quel componente.

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

```
})
export class AppModule { }
```

e dopo l'installazione di npm e l'avvio di npm. Ecco qui! Puoi controllare la schermata di login nel tuo localhost. In caso di difficoltà, è possibile fare riferimento all'angolare4-boilerplate.

Fondamentalmente riesco a percepire meno pacchetti costruttivi e una risposta più rapida con l'applicazione Angular 4 e sebbene abbia trovato Esattamente simile a Angular 2 nella codifica.

Leggi Iniziare con Angular 2 online: https://riptutorial.com/it/angular2/topic/789/iniziare-con-angular-2

Capitolo 2: Aggiorna i tipi

Examples

Aggiornare digitazioni quando: typings WARN deprecato

Messaggio di avviso:

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

Aggiorna il riferimento con:

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

Sostituisci [jazmine] per qualsiasi libreria che lancia avvisi

Leggi Aggiorna i tipi online: https://riptutorial.com/it/angular2/topic/7814/aggiorna-i-tipi

Capitolo 3: Aggiornamento di moduli angolari 2

Osservazioni

Angular 2 consente di accedere all'istanza ngForm creando una variabile di modello locale. Angular 2 espone istanze direttive come ngForm specificando la proprietà exportAs dei metadati della direttiva. Ora, il vantaggio qui è che senza molto codice è possibile accedere all'istanza ngForm e usarlo per accedere ai valori inviati o per verificare se tutti i campi sono validi usando le proprietà (valido, inviato, valore ecc.).

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

ngForm emette l'evento "ngSubmit" quando viene inviato (controlla la documentazione @Output per maggiori dettagli sull'emettitore di eventi)

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

"ngModel" crea un controllo modulo in combinazione con l'attributo "nome" di input.

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

Quando viene inviato il modulo, f.value ha l'oggetto JSON che rappresenta i valori inviati.

{username: 'Sachin', password: 'Welcome1'}

Examples

Modulo di modifica password semplice con convalida multi controllo

Gli esempi seguenti utilizzano la nuova API di form introdotta in 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;
    }
}
```

Un esempio che include alcune classi di bootstrap può essere trovato qui .

Angolare 2: forme guidate da modelli

```
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);
        }
```

```
    }
}
```

Angular 2 Form - Convalida email / password personalizzata

Per la demo dal vivo clicca ..

Indice delle app ts

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

bootstrap(MyForm);
```

Validatore personalizzato

```
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};
  }
}
```

Componenti modulo 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);
  }
}
```

Modulo HTML componenti

```
<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>
```

Angolare 2: forme reattive (ovvero forme guidate dal modello)

Questo esempio utilizza 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>
```

Angular 2 Forms (Reactive Forms) con modulo di registrazione e conferma della convalida della password

app.module.ts

Aggiungi questi nel tuo file app.module.ts per utilizzare moduli reattivi

```
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>
```

validators.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>
```

Leggi Aggiornamento di moduli angolari 2 online:
https://riptutorial.com/it/angular2/topic/4607/aggiornamento-di-moduli-angolari-2

Capitolo 4: Aggiungi componenti dinamicamente usando ViewContainerRef.createComponent

Examples

Un componente wrapper che aggiunge i componenti dinamici in modo dichiarativo

Un componente personalizzato che accetta il tipo di un componente come input e crea un'istanza di quel tipo di componente all'interno di se stesso. Quando l'input viene aggiornato, il componente dinamico aggiunto in precedenza viene rimosso e viene aggiunto il nuovo.

```
@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();
    }
  }
}
```

Questo ti permette di creare componenti dinamici come

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

Esempio di Plunker

Aggiungi dinamicamente un componente a un evento specifico (fai clic)

File di componente principale:

```
//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 {}
```

Esempio di Plunker

Rendering di array di componenti creati dinamicamente su template html in Angular2

Possiamo creare componenti dinamici e ottenere le istanze del componente in un array e infine renderlo sul modello.

Ad esempio, possiamo considerare due componenti widget, ChartWidget e PatientWidget che estendono la classe WidgetComponent che volevo aggiungere nel contenitore.

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 il pannello di 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 (uguale al widget grafico usando il pannello di primeng)

WidgetComponent.ts

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

possiamo creare istanze di componenti dinamici selezionando i componenti preesistenti. Per esempio,

```
@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 usiamo nel componente app. 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

Alcune modifiche apportate da @yurzui per utilizzare l'evento del mouse sui widget

view.directive.ts

import {ViewRef, Directive, Input, ViewContainerRef} da '@ 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

Leggi Aggiungi componenti dinamicamente usando ViewContainerRef.createComponent online: https://riptutorial.com/it/angular2/topic/831/aggiungi-componenti-dinamicamente-usando-viewcontainerref-createcomponent

Capitolo 5: Angolare - ForLoop

Sintassi

1. <div * ngFor = "let item of items; let i = index"> {{i}} {{item}} </ div>

Osservazioni

La direttiva strutturale `*ngFor` viene eseguita come un ciclo in una raccolta e ripete una parte di html per ciascun elemento di una raccolta.

`@View` decorator ora è deprecato. Gli sviluppatori dovrebbero utilizzare le proprietà `template` o 'templateUrl' per `@Component` decorator.

Examples

Angular 2 for-loop

Per live plnkr clicca ...

```
<!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 direttiva **NgFor** istanzia un modello una volta per elemento da un iterabile. Il contesto per ogni modello istanziato eredita dal contesto esterno con la variabile di ciclo specificata impostata sull'elemento corrente dal iterabile.

Per personalizzare l'algoritmo di tracciamento predefinito, NgFor supporta **l'** opzione **trackBy** . **trackBy** accetta una funzione che ha due argomenti: index e item. Se viene dato **trackBy** , le tracce angolari cambiano in base al valore di ritorno della funzione.

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

Opzioni aggiuntive : NgFor fornisce diversi valori esportati che possono essere alterati in variabili locali:

- **l'indice** verrà impostato sull'iterazione del ciclo corrente per ogni contesto del modello.
- **il primo** sarà impostato su un valore booleano che indica se l'elemento è il primo nell'iterazione.
- **l'ultimo** sarà impostato su un valore booleano che indica se l'elemento è l'ultimo nell'iterazione.
- **anche** verrà impostato su un valore booleano che indica se questo elemento ha un indice pari.
- **dispari** sarà impostato su un valore booleano che indica se questo elemento ha un indice dispari.

* ngPer le righe del tavolo

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

* ngPer il 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;
  }
```

* ngPer la quantità X di articoli per riga

L'esempio mostra 5 articoli per riga:

```
<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>
```

Leggi Angolare - ForLoop online: https://riptutorial.com/it/angular2/topic/6543/angolare---forloop

Capitolo 6: Angolare 2 - Goniometro

Examples

Test del routing Navbar con Goniometro

Per prima cosa, crea il navbar.html di base con 3 opzioni. (Home, Elenco, Crea)

```
<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>
```

secondo consente di creare 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
          });
        });

    });
  }
});
```

Angular2 Goniometro - Installazione

eseguire i comandi follows in cmd

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

** crea il file protractor.conf.js nella root principale dell'app.

molto importante per 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;
```

crea un test di base nella directory 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();
      });
    });
  });
});
```

corri in cmd

```
protractor conf.js
```

Leggi Angolare 2 - Goniometro online: https://riptutorial.com/it/angular2/topic/8900/angolare-2---goniometro

Capitolo 7: Angolare 2 Rilevamento delle modifiche e attivazione manuale

Examples

Esempio di base

Componente genitore:

```
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 padre:

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

componente figlio:

```
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 figlio:

```
<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>
```

Leggi Angolare 2 Rilevamento delle modifiche e attivazione manuale online: https://riptutorial.com/it/angular2/topic/8971/angolare-2-rilevamento-delle-modifiche-e-attivazione-manuale

Capitolo 8: Angular2 CanActivate

Examples

Angular2 CanActivate

Implementato in un router:

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

Il file `canActivateRoute` :

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

Leggi Angular2 CanActivate online: https://riptutorial.com/it/angular2/topic/8899/angular2-canactivate

Capitolo 9: Angular2 fornisce dati esterni all'App prima del bootstrap

introduzione

In questo post illustrerò come passare dati esterni all'app Angular prima dell'avvio dell'app. Questi dati esterni potrebbero essere dati di configurazione, dati legacy, server resi, ecc.

Examples

Via Iniezione di dipendenza

Invece di richiamare direttamente il codice bootstrap di Angular, avvolgere il codice bootstrap in una funzione ed esportare la funzione. Questa funzione può anche accettare parametri.

```
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));
}
```

Quindi, in qualsiasi servizio o componente, possiamo iniettare il "modello legacy" e accedervi.

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

Richiedi l'app e poi eseguila.

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

Leggi Angular2 fornisce dati esterni all'App prima del bootstrap online: https://riptutorial.com/it/angular2/topic/9203/angular2-fornisce-dati-esterni-all-app-prima-del-bootstrap

Capitolo 10: Angular2 Input () output ()

Examples

Ingresso()

Componente principale: inizializza gli elenchi degli utenti.

```
@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');
  }
}
```

Componente figlio ottiene l'utente dal componente principale 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;
  }
}
```

Semplice esempio di proprietà di input

Elemento genitore html

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

Elemento genitore ts

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

Componente figlio ts:

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

Html componente figlio:

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

Questo codice invierà l'inputPropValue dal componente padre al figlio e avrà il valore che abbiamo impostato nel componente genitore quando arriva lì - falso nel nostro caso. Possiamo quindi utilizzare quel valore nel componente figlio per, ad esempio, aggiungere una classe a un elemento.

Leggi Angular2 Input () output () online: https://riptutorial.com/it/angular2/topic/8943/angular2-input----output---

Capitolo 11: Angular2 utilizza il webpack

Examples

Configurazione del pacchetto Web angolare 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>
```

package.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": {
  }
}
```

Leggi Angular2 utilizza il webpack online: https://riptutorial.com/it/angular2/topic/9554/angular2-utilizza-il-webpack

Capitolo 12: Angular-cli

introduzione

Qui troverai come iniziare con angular-cli, generare nuovi componenti / servizi / pipe / module con angular-cli, aggiungere 3 parti come bootstrap, costruire un progetto angolare.

Examples

Creare un'applicazione Angular2 vuota con angular-cli

Requisiti:

- NodeJS: pagina di download
- npm o filato

Esegui i seguenti comandi con cmd dalla nuova cartella di directory:

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

Apri il tuo browser su localhost: 4200

Generazione di componenti, direttive, condotte e servizi

usa semplicemente il tuo cmd: puoi usare il comando ng generate (o solo ng g) per generare componenti Angolari:

- Componente: `ng g component my-new-component`
- Direttiva: `ng g directive my-new-directive`
- Tubo: `ng g pipe my-new-pipe`
- Servizio: `ng g service my-new-service`
- Classe: `ng g class my-new-classt`
- Interfaccia: `ng g interface my-new-interface`
- Enum: `ng g enum my-new-enum`
- Modulo: `ng g module my-module`

Aggiunta di librerie di terze parti

In angular-cli.json puoi modificare la configurazione dell'app.

Se si desidera aggiungere ng2-bootstrap ad esempio:

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

2. In angular-cli.json basta aggiungere il percorso del bootstrap su node-modules.

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

costruire con angular-cli

In angular-cli.json alla chiave outDir puoi definire la tua cartella di compilazione;

questi sono equivalenti

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

e così sono questi

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

Quando costruisci puoi modificare il tag di base () nel tuo index.html con l'opzione --base-href your-url.

Imposta il tag base href su / myUrl / nel tuo index.html

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

Nuovo progetto con scss / sass come foglio di stile

I file di stile di default generati e compilati da `@angular/cli` sono **css** .

Se invece vuoi usare **scss** , genera il tuo progetto con:

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

Se vuoi usare **sass** , genera il tuo progetto con:

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

Imposta il filo come gestore pacchetti predefinito per @ angular / cli

Yarn è un'alternativa per npm, il gestore di pacchetti predefinito su @ angular / cli. Se si desidera

utilizzare il filo come gestore pacchetti per @ angular / cli, attenersi alla seguente procedura:

Requisiti

- filato (`npm install --global yarn` o vedi la pagina di installazione)
- @ `npm install -g @angular/cli` / cli (`npm install -g @angular/cli` o `yarn global add @angular/cli`)

Per impostare il filato come @ angular / cli package manager:

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

Per ripristinare npm come @ angular / cli package manager:

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

Leggi Angular-cli online: https://riptutorial.com/it/angular2/topic/8956/angular-cli

Capitolo 13: Animazione

Examples

Transizione tra stati nulli

```
    @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 {

    }
]
```

Animazione tra più stati

Il `<div>` in questo modello cresce a `50px` e quindi a `100px` e poi si riduce a `20px` quando si fa clic sul pulsante.

Ogni `state` ha uno stile associato descritto nei metadati di `@Component` .

La logica per qualsiasi `state` attivo può essere gestita nella logica del componente. In questo caso, la `size` variabile del componente mantiene il valore di stringa "piccolo", "medio" o "grande".

L'elemento `<div>` risponde a quel valore attraverso il `trigger` specificato nei metadati `@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';
        }
    }
}
```

Leggi Animazione online: https://riptutorial.com/it/angular2/topic/8127/animazione

Capitolo 14: Animazioni Angular2

introduzione

Il sistema di animazione di Angular ti consente di creare animazioni che funzionano con lo stesso tipo di prestazioni native che si trovano nelle pure animazioni CSS. È anche possibile integrare strettamente la logica di animazione con il resto del codice dell'applicazione, per facilità di controllo.

Examples

Animazione di base - Transita un elemento tra due stati guidato da un attributo modello.

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;
  }

}
```

Leggi Animazioni Angular2 online: https://riptutorial.com/it/angular2/topic/8970/animazioni-angular2

Capitolo 15: API Web Angular2 in memoria

Osservazioni

Ho principalmente richiesto questo argomento perché non sono riuscito a trovare alcuna informazione sull'impostazione di più percorsi API con Angular2-In-Memory-Web-Api. Finito per capirlo da solo, e ho pensato che questo potesse essere d'aiuto agli altri.

Examples

Impostazione di base

mock-data.ts

Creare i dati di API mock

```
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

Chiedere all'iniettore di dipendenza di fornire InMemoryBackendService per le richieste XHRBackend. Inoltre, fornire una classe che include a

```
createDb()
```

funzione (in questo caso, MockData) che specifica i percorsi API simulati per le richieste 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

Esempio di chiamata di una richiesta di acquisizione per la rotta dell'API creata

```
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);
  }
}
```

Impostazione di più percorsi API di test

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 };
    }
```

```
}
```

Ora puoi interagire con entrambi

```
app/mock
```

e

```
app/data
```

per estrarre i loro dati corrispondenti.

Leggi API Web Angular2 in memoria online: https://riptutorial.com/it/angular2/topic/6576/api-web-angular2-in-memoria

Capitolo 16: barile

introduzione

Un barile è un modo per trasferire le esportazioni da diversi moduli ES2015 in un singolo modulo ES2015. La canna stessa è un file di modulo ES2015 che riesporta le esportazioni selezionate di altri moduli ES2015.

Examples

Usando il barilotto

Ad esempio senza un barile, un consumatore avrebbe bisogno di tre istruzioni di importazione:

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

Possiamo aggiungere un barile creando un file nella stessa cartella del componente. In questo caso la cartella viene chiamata 'heroes' denominata index.ts (utilizzando le convenzioni) che esporta tutti questi elementi:

```
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
```

Ora un consumatore può importare ciò di cui ha bisogno dal barile.

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

Tuttavia, questa può diventare una linea molto lunga; che potrebbe essere ulteriormente ridotto.

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

È piuttosto ridotto! Il `* as h` importa tutti i moduli e gli alias come h

Leggi barile online: https://riptutorial.com/it/angular2/topic/10717/barile

Capitolo 17: Bootstrap Modulo vuoto in angolare 2

Examples

Un modulo vuoto

```
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 {}
```

Questo è un modulo vuoto che non contiene dichiarazioni, importazioni, provider o componenti per il bootstrap. Usa questo riferimento.

Un modulo con collegamento in rete sul browser 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` è il componente root `MyModule` in `MyModule` . È il punto di accesso all'applicazione Angular 2.

Avvio automatico del modulo

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

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

In questo esempio, `MyModule` è il modulo contenente il componente root. Con l'avvio automatico di `MyModule` l'app Angular 2 è pronta per l'uso.

Modulo radice dell'applicazione

```
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 { }
```

Bootstrap statico con classi factory

Possiamo eseguire il bootstrap di un'applicazione in modo statico prendendo il semplice output JavaScript ES5 delle classi factory generate. Quindi possiamo usare quell'output per avviare l'applicazione:

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

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

Questo farà sì che il pacchetto di applicazioni sia molto più piccolo, poiché tutta la compilazione del modello è già stata eseguita in una fase di creazione, utilizzando ngc o chiamando direttamente i suoi interni.

Leggi Bootstrap Modulo vuoto in angolare 2 online:
https://riptutorial.com/it/angular2/topic/5508/bootstrap-modulo-vuoto-in-angolare-2

Capitolo 18: Brute Force Upgrading

introduzione

Se si desidera aggiornare la versione del CLI di Angular CLI, è possibile che si verifichino errori e bachi difficili da correggere semplicemente modificando il numero di versione della CLI angolare nel progetto. Inoltre, poiché la CLI Angolare nasconde molto di ciò che accade nel processo di creazione e bundle, non si può fare molto quando le cose vanno male lì.

A volte il modo più semplice per aggiornare la versione del CLI di Angular del progetto consiste semplicemente nel creare un nuovo proejct con la versione della CLI Angolare che si desidera utilizzare.

Osservazioni

Poiché Angular 2 è *così* modulare e incapsulato, puoi praticamente copiare tutti i componenti, i servizi, i tubi, le direttive e quindi compilare NgModule come nel vecchio progetto.

Examples

Scaffolding a New Angular CLI Project

```
ng new NewProject
```

o

```
ng init NewProject
```

Leggi Brute Force Upgrading online: https://riptutorial.com/it/angular2/topic/9152/brute-force-upgrading

Capitolo 19: Bypassare la disinfezione per valori attendibili

Parametri

Parametri	Dettagli
selettore	nome del tag fai riferimento al tuo componente in html
modello (TemplateURL)	una stringa che rappresenta html che verrà inserita ovunque sia il tag `<selector>` . templateUrl è un percorso di un file html con lo stesso comportamento
tubi	una serie di pipe utilizzate da questo componente.

Osservazioni

SUPER IMPORTANTE!

DISABLING SANITIZING TIENE A RISCHIO DI XSS (Cross-Site Scripting) E ALTRI VETTORI D'ATTACCO. PER FAVORE ASSICURATI DI FIDARTI COSA STAI OTTENENDO 100%

L'utilizzo di Pipes ti relega a modificare solo i valori degli attributi in questo modo:

```
<tag [attribute]="expression or variable reference | pipeName">
```

non sei in grado di usare i tubi in questo modo:

```
<tag attribute="expression or variable reference | pipeName">
```

o in questo modo

```
<tag attribute={{expression or variable reference | pipeName}}
```

Examples

Bypassing Sanitizing con pipe (per riutilizzo del codice)

Il progetto segue la struttura dalla guida rapida di Angular2 qui .

```
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);
```

Questo trova il file index.html nella radice del progetto e ne costruisce.

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 { }
```

Questo è il componente di livello superiore che raggruppa altri componenti utilizzati.

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";
        }
    }
}
```

Questo componente fornisce la vista per il tubo con cui lavorare.

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);
   }

}
```

Questa è la logica che descrive il formato del tubo.

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">
```

Se dovessi ispezionare l'html mentre l'app è in esecuzione, vedresti che assomiglia a questo:

```
<head>
    Stuff goes here...
</head>

<body>

    <othertags>
    </othertags>

    <img [src]="getUrl('1234') | sanitaryPipe">
    <embed [src]="getUrl() | sanitaryPipe">

    <moretags>
    </moretags>

</body>
```

Leggi Bypassare la disinfezione per valori attendibili online:
https://riptutorial.com/it/angular2/topic/5942/bypassare-la-disinfezione-per-valori-attendibili

Capitolo 20: Come usare ngfor

introduzione

La direttiva `ngFor` viene utilizzata da Angular2 per creare un'istanza di un modello una volta per ogni elemento in un oggetto iterabile. Questa direttiva lega l'iterabile al DOM, quindi se il contenuto dell'iter iterabile cambia, anche il contenuto del DOM verrà modificato.

Examples

Esempio di elenco non ordinato

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

Esempio di modello più complesso

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

Tracciamento dell'esempio di interazione corrente

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

In questo caso, prenderò il valore dell'indice, che è l'iterazione del loop corrente.

Valori esportati con alias Angular2

Angular2 fornisce diversi valori esportati che possono essere alterati in variabili locali. Questi sono:

- indice
- primo
- scorso
- anche
- dispari

Eccetto `index` , gli altri prendono un valore `Boolean` . Come l'esempio precedente che utilizza l'indice, può essere utilizzato uno qualsiasi di questi valori esportati:

```
<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>
```

* ngPer tubi

```
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)
}
```

Leggi Come usare ngfor online: https://riptutorial.com/it/angular2/topic/8051/come-usare-ngfor

Capitolo 21: Come usare ngif

introduzione

* **NgIf** : rimuove o ricrea una parte dell'albero DOM in base alla valutazione di un'espressione. È una direttiva strutturale e le direttive strutturali alterano il layout del DOM aggiungendo, sostituendo e rimuovendo i suoi elementi.

Sintassi

- <div * ngIf = "false"> test </ div> <! - restituisce false ->
- <div * ngIf = "undefined"> test </ div> <! - restituisce false ->
- <div * ngIf = "null"> test </ div> <! - restituisce false ->
- <div * ngIf = "0"> test </ div> <! - restituisce false ->
- <div * ngIf = "NaN"> test </ div> <! - restituisce false ->
- <div * ngIf = ""> test </ div> <! - restituisce false ->
- Tutti gli altri valori sono valutati come veri.

Examples

Mostra un messaggio di caricamento

Se il nostro componente non è pronto e in attesa di dati dal server, possiamo aggiungere il caricatore usando * ngIf. **passi:**

Prima dichiara un booleano:

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

Successivamente, nel componente aggiungi un hook per il ciclo di vita chiamato `ngOnInit`

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

e dopo aver ottenuto i dati completi dal set di server, si carica booleano su falso.

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

Nel tuo template html usa * ngIf con la proprietà di `loading` :

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

Mostra messaggio di avviso a una condizione

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

Per eseguire una funzione all'inizio o alla fine del ciclo * ngFor utilizzando * ngIf

NgFor fornisce alcuni valori che possono essere alterati a variabili locali

- **index** - (variabile) posizione dell'elemento corrente nel iterabile a partire da 0
- **first** - (boolean) true se l'elemento corrente è il primo elemento nel iterable
- **last** - (boolean) true se l'elemento corrente è l'ultimo elemento nel iterable
- **even** - (booleano) true se l'indice corrente è un numero pari
- **dispari** - (booleano) vero se l'indice corrente è un numero dispari

```
<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>
```

Usa * ngIf con * ngFor

Mentre non ti è permesso usare `*ngIf` e `*ngFor` nello stesso div (darà un errore nel runtime) puoi nidificare `*ngIf` in `*ngFor` per ottenere il comportamento desiderato.

Esempio 1: sintassi generale

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

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

  </div>
</div>
```

Esempio 2: Visualizza elementi con indice pari

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

Lo svantaggio è che è necessario aggiungere un ulteriore elemento `div` esterno.

Considerate questo caso d'uso in cui un elemento `div` deve essere iterato (usando * ngFor) e

include anche un controllo se l'elemento deve essere rimosso o meno (usando * ngIf), ma non è preferibile aggiungere un `div` aggiuntivo. In questo caso puoi usare il tag `template` per * ngFor:

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

In questo modo non è necessario aggiungere un `div` esterno aggiuntivo e inoltre l'elemento `<template>` non verrà aggiunto al DOM. Gli unici elementi aggiunti nel DOM dell'esempio precedente sono gli elementi `div` iterati.

Nota: in Angular v4 `<template>` è stato deprecato a favore di `<ng-template>` e verrà rimosso nella v5. Nelle versioni di Angular v2.x `<template>` è ancora valido.

Leggi Come usare ngif online: https://riptutorial.com/it/angular2/topic/8346/come-usare-ngif

Capitolo 22: Compilazione A prima del tempo (AOT) con Angular 2

Examples

1. Installare le dipendenze Angular 2 con il compilatore

NOTA: per ottenere risultati ottimali, assicurarsi che il progetto sia stato creato utilizzando Angular-CLI.

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

Non è necessario eseguire questo passaggio se si progetta già con angular 2 e tutte queste dipendenze installate. Assicurati che il `compiler` sia lì.

2. Aggiungi `angularCompilerOptions` al tuo file` tsconfig.json`

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

Questa è la cartella di output del compilatore.

3. Eseguire ngc, il compilatore angolare

dalla radice del tuo progetto `./node_modules/.bin/ngc -p src` dove `src` è dove risiede tutto il tuo codice angolare 2. Questo genererà una cartella chiamata `ngfactory` dove tutto il tuo codice compilato vivrà.

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

4. Modificare il file `main.ts` per utilizzare NgFactory e il browser della piattaforma statica

```
// 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);
```

A questo punto dovresti essere in grado di eseguire il tuo progetto. In questo caso, il mio progetto è stato creato utilizzando la Angular-CLI.

```
> ng serve
```

Perché abbiamo bisogno di compilazione, compilazione degli eventi Flow ed esempi?

D. Perché abbiamo bisogno della compilazione? Ans. Abbiamo bisogno di una compilazione per raggiungere un più alto livello di efficienza delle nostre applicazioni Angular.

Dai un'occhiata al seguente esempio,

```
// ...
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 il codice sopra per rendere il modello,

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

È molto più lento rispetto a:

```
// ...
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);
// ...
```

Flusso di eventi con la compilazione anticipata

Al contrario, con AoT otteniamo i seguenti passaggi:

1. Sviluppo dell'applicazione Angular 2 con TypeScript.
2. Compilazione dell'applicazione con ngc.
3. Esegue la compilazione dei modelli con il compilatore angolare su TypeScript.
4. Compilazione del codice TypeScript in JavaScript.
5. Bundling.
6. Minification.
7. Distribuzione.

Sebbene il processo di cui sopra sembra leggermente più complicato, l'utente passa solo attraverso i passaggi:

1. Scarica tutte le risorse.
2. Bootstrap angolari.
3. L'applicazione viene renderizzata.

Come puoi vedere, manca il terzo passo, che significa UX più veloce / migliore e oltre a strumenti come angular2-seed e angular-cli automatizzerà il processo di costruzione in modo drammatico.

Spero possa aiutarti! Grazie!

Utilizzo della compilazione AoT con CLI angolare

L'interfaccia della riga di comando CLI di Angular ha il supporto alla compilazione AoT dalla versione beta 17.

Per costruire la tua app con la compilazione AoT, esegui semplicemente:

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

Leggi Compilazione A prima del tempo (AOT) con Angular 2 online:
https://riptutorial.com/it/angular2/topic/6634/compilazione-a-prima-del-tempo--aot--con-angular-2

Capitolo 23: componenti

introduzione

I componenti angolari sono elementi composti da un modello che renderà la tua applicazione.

Examples

Un componente semplice

Per creare un componente, aggiungiamo `@Component` decorator in una classe che passa alcuni parametri:

- `providers` : risorse che verranno iniettate nel costruttore del componente
- `selector` : il selettore di query che troverà l'elemento nell'HTML e lo sostituirà con il componente
- `styles` : `styles` incorporati. NOTA: NON utilizzare questo parametro con require, funziona sullo sviluppo ma quando si crea l'applicazione in produzione tutti i tuoi stili vengono persi
- `styleUrls` : matrice del percorso per i file di stile
- `template` : stringa che contiene il tuo HTML
- `templateUrl` : percorso di un file HTML

Ci sono altri parametri che puoi configurare, ma quelli elencati sono quelli che userai di più.

Un semplice esempio:

```
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 { }
```

Modelli e stili

I modelli sono file HTML che possono contenere logica.

Puoi specificare un modello in due modi:

Passando il modello come percorso del file

```
@Component({
  templateUrl: 'hero.component.html',
})
```

Passare un modello come codice in linea

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

I modelli possono contenere stili. Gli stili dichiarati in `@Component` sono diversi dal file di stile dell'applicazione, qualsiasi cosa applicata nel componente sarà limitata a questo ambito. Ad esempio, supponi di aggiungere:

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

Tutte le `div` all'interno del componente saranno rosse, ma se hai altri componenti, altre div nel tuo codice HTML non saranno affatto cambiate.

Il codice generato sarà simile a questo:

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

Puoi aggiungere stili a un componente in due modi:

Passando una serie di percorsi di file

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

Passando una serie di codici in linea

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

Non si dovrebbero usare gli `styles` con `require` in quanto non funzionerà quando si costruisce l'applicazione in produzione.

Test di 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();
  });
});
```

Componenti di nidificazione

I componenti verranno visualizzati nel rispettivo `selector` , quindi è possibile utilizzarli per nidificare i componenti.

Se hai un componente che mostra un messaggio:

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

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

Puoi usarlo all'interno di un altro componente usando `app-required` (il selettore di questo 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 = '';
}
```

Leggi componenti online: https://riptutorial.com/it/angular2/topic/10838/componenti

Capitolo 24: Configurazione dell'applicazione ASP.net Core per lavorare con Angular 2 e TypeScript

introduzione

SCENARIO: ASP.NET Core background Angular 2 Front-End Angular 2 Componenti che utilizzano i controller Asp.net Core

In questo modo è possibile implementare Angular 2 su Asp.Net Core. Chiamiamo anche MVC Controller dai componenti Angular 2 con il risultato MVC View support 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/"
  ]
}
```

Package.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
  }
]
```

Converti bundleconfig.json in gulpfile (RightClick bundleconfig.json su solution explorer, Bundler & Minifier> Converti in Gulp

Vista / Home / Index.cshtml

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

Per la cartella wwwroot utilizzare https://github.com/angular/quickstart seed. Hai bisogno di: **index.html main.ts, systemjs-angular-loader.js, systemjs.config.js, tsconfig.json** E la **cartella dell'app**

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>
```

Puoi chiamarlo come controller da templateUrl. wwwroot / app / app.component.ts

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

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

[Seme] Asp.Net Core + Angular2 + Gulp su Visual Studio 2017

1. Scarica seme
2. Eseguire il ripristino dotnet
3. Esegui l'installazione di npm

Sempre. Godere.

https://github.com/SamML/CoreAngular000

MVC <-> Angolare 2

Procedura: CALL ANGULAR 2 HTML / JS COMPONENT FROM ASP.NET Core CONTROLLER:

Chiamiamo l'HTML invece restituiamo View ()

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

E carica il componente angolare nell'html. Qui possiamo decidere se vogliamo lavorare con un modulo uguale o diverso. Dipende dalla situazione

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>
```

(*) Già questo seme ha bisogno di caricare l'intero elenco di risorse

Come: CALL ASP.NET Core Controller per mostrare una vista MVC con supporto Angular2:

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

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

}
```

Leggi Configurazione dell'applicazione ASP.net Core per lavorare con Angular 2 e TypeScript online: https://riptutorial.com/it/angular2/topic/9543/configurazione-dell-applicazione-asp-net-core-per-lavorare-con-angular-2-e-typescript

Capitolo 25: copertura del test angular-cli

introduzione

la copertura del test è definita come una tecnica che determina se i nostri casi di test coprono effettivamente il codice dell'applicazione e la quantità di codice che viene esercitata quando eseguiamo questi casi di test.

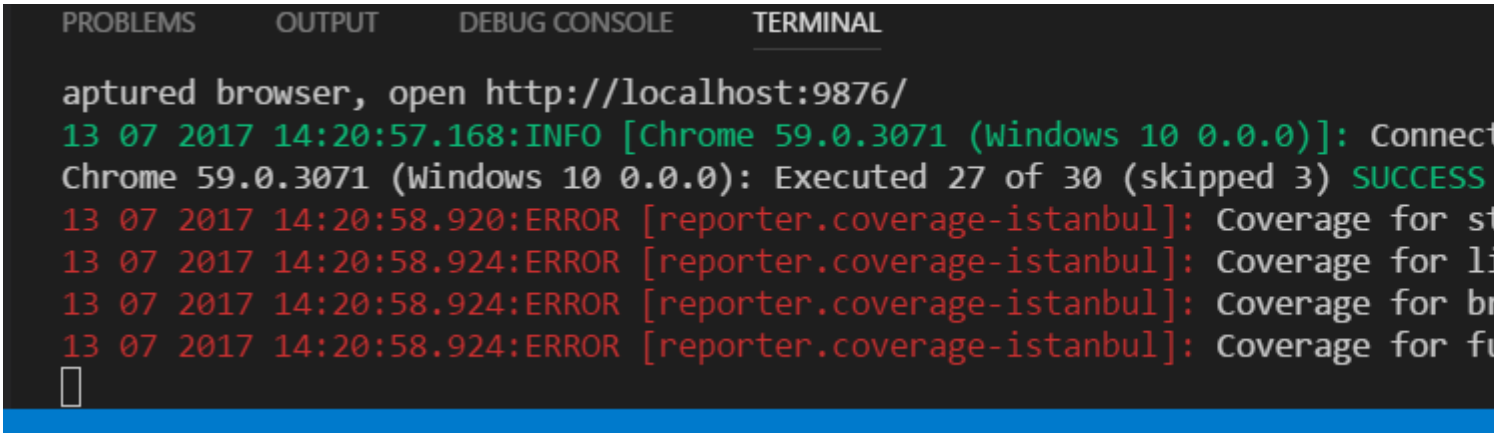
La CLI angolare ha incorporato la funzionalità di copertura del codice con un semplice comando `ng test --cc`

Examples

Una semplice copertura per test di comando base angular-cli

Se si desidera visualizzare le statistiche generali sulla copertura del test rispetto a quanto indicato nella CLI Angolare, è sufficiente digitare sotto il comando e visualizzare i risultati nella parte inferiore della finestra del prompt dei comandi.

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



Report dettagliati sulla copertura del test grafico di base dei singoli componenti

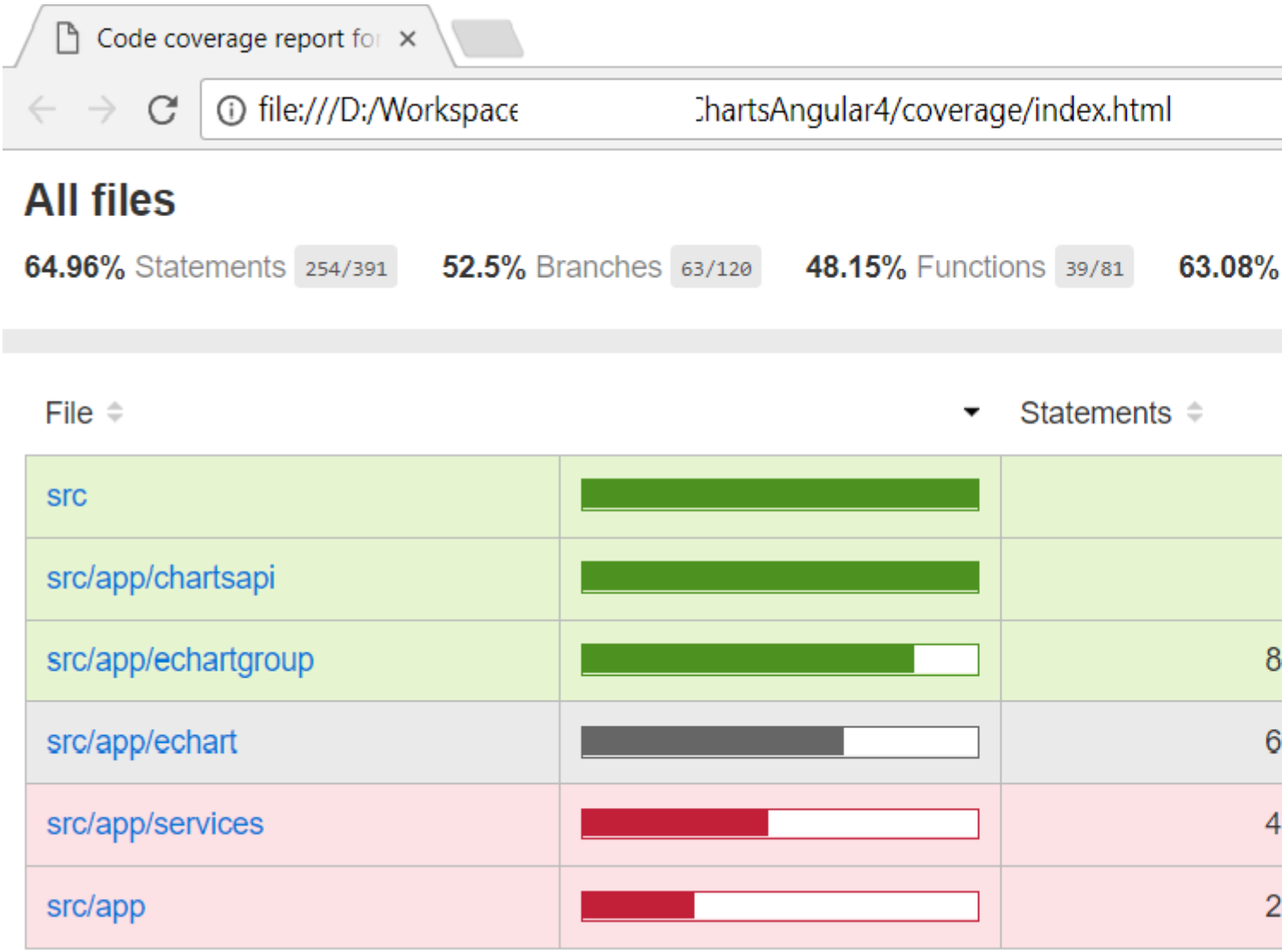
se si desidera vedere la copertura individuale dei componenti dei test, attenersi alla seguente procedura.

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`

si noti che l'elenco dei reporter è separato da virgole, poiché abbiamo aggiunto un nuovo reporter,

teamcity.

dopo aver eseguito questo comando puoi vedere la `coverage` della cartella nella tua directory e aprire `index.html` per una visualizzazione grafica della copertura del test.



Puoi anche impostare la soglia di copertura che vuoi raggiungere, in `karma.conf.js` , in questo modo.

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

Leggi copertura del test angular-cli online: https://riptutorial.com/it/angular2/topic/10764/copertura-del-test-angular-cli

Capitolo 26: Crea un pacchetto Angular 2+ NPM

introduzione

A volte abbiamo bisogno di condividere alcuni componenti tra alcune app e pubblicarlo in npm è uno dei modi migliori per farlo.

Ci sono alcuni trucchi che dobbiamo sapere per essere in grado di utilizzare un componente normale come pacchetto npm senza modificare la struttura come inlining di stili esterni.

Puoi vedere un esempio minimale qui

Examples

Pacchetto più semplice

Qui stiamo condividendo un flusso di lavoro minimo per costruire e pubblicare un pacchetto Angular 2+ npm.

File di configurazione

Abbiamo bisogno di alcuni file di configurazione per dire a `git` , `npm` , `gulp` e `typescript` come comportarsi.

.gitignore

Per prima cosa creiamo un file `.gitignore` per evitare il versionamento di file e cartelle indesiderati. Il contenuto è:

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

.npmignore

In secondo luogo creiamo un file `.npmignore` per evitare di pubblicare file e cartelle indesiderati. Il contenuto è:

```
examples
node_modules
src
```

gulpfile.js

Dobbiamo creare un `gulpfile.js` per dire a Gulp come compilare la nostra applicazione. Questa parte è necessaria perché è necessario ridurre a icona e integrare tutti i modelli e gli stili esterni prima di pubblicare il pacchetto. Il contenuto è:

```
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

Il file `index.d.ts` viene utilizzato da typescript durante l'importazione di un modulo esterno. Aiuta l'editor con i suggerimenti per il completamento automatico e la funzione.

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

index.js

Questo è il punto di ingresso del pacchetto. Quando si installa questo pacchetto utilizzando NPM e si importa nell'applicazione, è sufficiente passare il nome del pacchetto e l'applicazione imparerà dove trovare qualsiasi componente ESPORTATO del pacchetto.

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

Abbiamo usato la cartella `lib` perché quando compiliamo il nostro codice, l'output è collocato all'interno della cartella `/lib` .

package.json

Questo file è usato per configurare la pubblicazione di npm e definisce i pacchetti necessari per farlo funzionare.

```
{
  "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

Creare una cartella dist e posizionare questo file all'interno. Questo file è usato per dire a Typescript come compilare la tua applicazione. Dove trovare la cartella dattiloscritto e dove mettere i file compilati.

```
{
  "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
  }
}
```

Dopo aver creato i file di configurazione, dobbiamo creare il nostro componente e modulo. Questo componente riceve un clic e visualizza un messaggio. È usato come un tag html `<angular-x-minimal-npm-package></angular-x-minimal-npm-package>` . Installa questo pacchetto npm e carica il suo modulo nel modello che desideri utilizzarlo.

src / angolare-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 / angolare-x-minimal-NPM-package.component.html

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

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

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

src / angolare-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 {}
```

Successivamente, è necessario compilare, compilare e pubblicare il pacchetto.

Costruisci e compila

Per build usiamo `gulp` e per la compilazione usiamo `tsc` . Il comando è impostato nel file package.json, all'opzione `scripts.build` . Abbiamo questo set `gulp js:build && rm -rf lib && tsc -p dist` . Questa è la nostra catena di compiti che farà il lavoro per noi.

Per compilare e compilare, eseguire il seguente comando nella radice del pacchetto:

```
npm run build
```

Questo attiverà la catena e `/dist` cartella build in `/dist` e il pacchetto compilato nella cartella `/lib` . Questo è il motivo per cui in `index.js` abbiamo esportato il codice dalla cartella `/lib` e non da `/src` .

Pubblicare

Ora abbiamo solo bisogno di pubblicare il nostro pacchetto in modo che possiamo installarlo tramite npm. Per questo, basta eseguire il comando:

```
npm publish
```

Questo è tutto!!!

Leggi Crea un pacchetto Angular 2+ NPM online:
https://riptutorial.com/it/angular2/topic/8790/crea-un-pacchetto-angular-2plus-npm

Capitolo 27: Creazione di una libreria Angular npm

introduzione

Come pubblicare il tuo NgModule, scritto in TypeScript nel registro di NPM. Configurazione del progetto npm, del compilatore dattiloscritto, del rollup e dell'integrazione continua.

Examples

Modulo minimale con classe di servizio

Struttura del file

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

Servizio e modulo

Metti il tuo fantastico lavoro qui.

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 {}
```

Rendi accessibile il modulo e il servizio.

/index.ts:

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

Compilazione

In compilerOptions.paths è necessario specificare tutti i moduli esterni utilizzati nel pacchetto.

/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
  }
}
```

Specifica di nuovo i tuoi esterni

/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']
}
```

Impostazioni NPM

Ora, consente di inserire alcune istruzioni per 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"
  }
}
```

Possiamo anche specificare quali file, npm dovrebbe ignorare

/.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
```

Integrazione continua

Finalmente puoi configurare la build di integrazione 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 demo può essere trovata qui: https://github.com/maciejtreder/awesome-angular-module

Leggi Creazione di una libreria Angular npm online:
https://riptutorial.com/it/angular2/topic/10704/creazione-di-una-libreria-angular-npm

Capitolo 28: CRUD in Angular2 con Restful API

Sintassi

- @Injectable () // Indica all'iniettore di dipendenza di iniettare dipendenze durante la creazione dell'istanza di questo servizio.
- request.subscribe () // Questo è dove *effettivamente* fare la richiesta. Senza questo la tua richiesta non verrà inviata. Inoltre, si desidera leggere la risposta nella funzione di callback.
- costruttore (wikiService privato: WikipediaService) {} // Dato che sia il nostro servizio che le sue dipendenze sono iniettabili dall'iniettore di dipendenze, è buona pratica iniettare il servizio sul componente per il test dell'unità dell'app.

Examples

Leggi da un'API riposante in Angular2

Per separare la logica dell'API dal componente, stiamo creando il client API come classe separata. Questa classe di esempio effettua una richiesta all'API di Wikipedia per ottenere articoli di wiki casuali.

```
    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.
            });
        }
    }
```

E avere un componente per consumare il nostro nuovo client 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;
        });
    }
}
```

Leggi CRUD in Angular2 con Restful API online: https://riptutorial.com/it/angular2/topic/7343/crud-in-angular2-con-restful-api

Capitolo 29: Databinding angolare2

Examples

@Ingresso()

Componente principale: inizializza gli elenchi degli utenti.

```
@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');
  }
}
```

Componente figlio ottiene l'utente dal componente principale 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;
  }
}
```

Leggi Databinding angolare2 online: https://riptutorial.com/it/angular2/topic/9036/databinding-angolare2

Capitolo 30: Debug di un'applicazione dattiloscritto Angular2 utilizzando il codice di Visual Studio

Examples

Launch.json setup per il tuo spazio di lavoro

1. Attiva Debug dal menu: visualizza> debug
2. restituisce qualche errore durante l'avvio del debug, mostra la notifica di pop out e apre launch.json da questa notifica popup È solo perché launch.json non è impostato per il tuo spazio di lavoro. copia e incolla sotto il codice in launch.json // new launch.json
   il tuo vecchio 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"
        }
    ]
}
```

Ora aggiorna il tuo launch.json come sotto
nuovo launch.json
** // ricorda per favore menziona il tuo percorso main.js in esso **

```
{
    "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. Ora il debug funziona, mostra un popup di notifica per il debug passo dopo passo

Leggi Debug di un'applicazione dattiloscritto Angular2 utilizzando il codice di Visual Studio online: https://riptutorial.com/it/angular2/topic/7139/debug-di-un-applicazione-dattiloscritto-angular2-utilizzando-il-codice-di-visual-studio

Capitolo 31: Design materiale angolare

Examples

Md2Select

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>
```

Selezionare consentire all'utente di selezionare l'opzione dalle opzioni.

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

Md2Tooltip

Tooltip è una direttiva, consente all'utente di mostrare il testo suggerimento mentre il mouse dell'utente passa con il mouse su un elemento.

Un tooltip avrebbe il seguente markup.

```
<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 è un servizio che mostra le notifiche nella vista.

Crea e mostra una semplice notazione su pane tostato.

```
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

Datepicker consente all'utente di selezionare data e ora.

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

vedere per maggiori dettagli qui

Md2Accordion e Md2Collapse

Md2Collapse : Collapse è una direttiva, consente all'utente di alternare visiblity della sezione.

Esempi

Un collasso avrebbe il seguente markup.

```
<div [collapse]="isCollapsed">
  Lorum Ipsum Content
</div>
```

Md2Accordion : Fisarmonica consente all'utente di attivare la visiblity delle sezioni multiple.

Esempi

Una fisarmonica avrebbe il seguente markup.

```
<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>
```

Leggi Design materiale angolare online: https://riptutorial.com/it/angular2/topic/10005/design-materiale-angolare

Capitolo 32: direttive

Sintassi

- `<input [value]="value">` - Associa il `name` membro della classe del valore dell'attributo.
- `<div [attr.data-note]="note">` - Associa l'attributo `data-note` alla `note` variabile.
- `<p green></p>` - Direttiva personalizzata

Osservazioni

La principale fonte di informazioni sulle direttive Angular 2 è la documentazione ufficiale https://angular.io/docs/ts/latest/guide/attribute-directives.html

Examples

Direttiva attributi

```
<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>
```

Il componente è una direttiva con modello

```
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 {
}
```

Direttive strutturali

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

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

Direttiva personalizzata

```
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>
```

* ngFor

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'];
}
```

Produzione:

```
<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>
```

Nella sua forma più semplice, `*ngFor` ha due parti: `let` **`variableName`** `of` **`object/array`**

Nel caso di `fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];` ,

Mele, arance e così via sono i valori all'interno del `fruit` dell'array.

`[value]="f"` sarà uguale a ogni `fruit` corrente (`f`) che `*ngFor` ha iterato sopra.

A differenza di AngularJS, Angular2 non ha continuato con l'uso di `ng-options` per `<select>` e `ng-repeat` per tutte le altre ripetizioni generali.

`*ngFor` è molto simile a `ng-repeat` con sintassi leggermente variata.

Riferimenti:

Angolare2 | Visualizzazione dei dati

Angolare2 | ngFor

Angolare2 | Le forme

Copia nella direttiva Appunti

In questo esempio creeremo una direttiva per copiare un testo negli appunti facendo clic su 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

Ricorda di inserire TEXT_COPY_DIRECTIVES nella matrice delle direttive del tuo 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>
...
```

Test di una direttiva personalizzata

Data una direttiva che evidenzia il testo sugli eventi 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;
  }
}
```

Può essere testato in questo modo

```
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');
  });
});
```

Leggi direttive online: https://riptutorial.com/it/angular2/topic/2202/direttive

Capitolo 33: Direttive e componenti: @Input @Output

Sintassi

1. Associazione unidirezionale dal componente principale al componente nidificato: [propertyName]
2. Associazione unidirezionale dal componente nidificato al componente principale: (propertyName)
3. Associazione a due vie (nota anche come notazione della scatola di banane): [(propertyName)]

Examples

Esempio di input

@input è utile per legare i dati tra i componenti

Innanzitutto, importalo nel tuo componente

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

Quindi, aggiungi l'input come una proprietà della classe del componente

```
@Input() car: any;
```

Diciamo che il selettore del tuo componente è 'car-component', quando chiami il componente, aggiungi l'attributo 'car'

```
<car-component [car]="car"></car-component>
```

Ora la tua auto è accessibile come un attributo nel tuo oggetto (this.car)

Esempio 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 e @Output in un componente nidificato

Una direttiva Button che accetta un `@Input()` per specificare un limite di clic fino a quando il

pulsante non viene disabilitato. Il componente principale può ascoltare un evento che verrà emesso quando viene raggiunto il limite di clic tramite `@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 padre che utilizza la direttiva Button e avvisa un messaggio quando viene raggiunto il limite di clic:

```
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 dati asincroni

A volte è necessario recuperare i dati in modo asincrono prima di passarli a un componente figlio da utilizzare. Se il componente figlio tenta di utilizzare i dati prima che sia stato ricevuto, genera un errore. È possibile utilizzare `ngOnChanges` per rilevare le modifiche nei componenti ' `@Input` s e

attendere fino a quando non vengono definiti prima di agire su di essi.

Componente padre con chiamata asincrona a un endpoint

```
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 figlio che ha dati asincroni come input

Questo componente figlio accetta i dati asincroni come input. Pertanto, prima di utilizzarlo, è necessario attendere che i dati esistano. Usiamo ngOnChanges che si attiva ogni volta che l'input di un componente cambia, controlla se i dati esistono e li usa se lo fa. Si noti che il modello per il bambino non mostrerà se una proprietà che si basa sui dati passati non è vera.

```
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;
    }
}
```

Leggi Direttive e componenti: @Input @Output online:
https://riptutorial.com/it/angular2/topic/3046/direttive-e-componenti---input--output

Capitolo 34: Direttive e servizi comunemente incorporati

introduzione

@ angolare / comune - direttive e servizi comunemente necessari @ angolare / core: il framework del nucleo angolare

Examples

Location Class

La posizione è un servizio che le applicazioni possono utilizzare per interagire con l'URL di un browser. A seconda di quale LocationStrategy viene utilizzata, la posizione rimarrà sul percorso dell'URL o sul segmento hash dell'URL.

La posizione è responsabile della normalizzazione dell'URL rispetto all'href di base dell'applicazione.

```
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 pipe async si iscrive a Observable o Promise e restituisce l'ultimo valore che ha emesso.

Quando viene emesso un nuovo valore, la pipe async contrassegna il componente da verificare per le modifiche. Quando il componente viene distrutto, la pipe asincrona si annulla automaticamente per evitare potenziali perdite di 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);
  });
}
```

Visualizzazione della versione angular2 corrente utilizzata nel progetto

Per visualizzare la versione corrente, possiamo usare **VERSION** dal pacchetto @ 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;
}
```

Tubo di valuta

Il tubo valuta consente di lavorare con i dati dell'utente come numeri normali, ma di visualizzarli con la formattazione valuta standard (simbolo di valuta, cifre decimali, ecc.) Nella 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;
}
```

La pipe accetta tre parametri opzionali:

- **currencyCode** : consente di specificare il codice valuta ISO 4217.
- **symbolDisplay** : booleano che indica se utilizzare il simbolo di valuta
- **digitInfo** : consente di specificare la modalità di visualizzazione delle posizioni decimali.

Più documentazione sul tubo valutario:
https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html

Leggi Direttive e servizi comunemente incorporati online:
https://riptutorial.com/it/angular2/topic/8252/direttive-e-servizi-comunemente-incorporati

Capitolo 35: Direttive sugli attributi per influenzare il valore delle proprietà sul nodo host utilizzando il decoratore @HostBinding.

Examples

@HostBinding

Il decoratore @HostBinding ci consente di impostare in modo programmatico un valore di proprietà sull'elemento host della direttiva. Funziona in modo simile a un vincolo di proprietà definito in un modello, tranne per il target specifico dell'elemento host. La rilegatura viene controllata per ogni ciclo di rilevamento delle modifiche, quindi può essere modificata dinamicamente, se lo si desidera. Ad esempio, diciamo che vogliamo creare una direttiva per i pulsanti che aggiunge dinamicamente una classe quando la premiamo. Potrebbe sembrare qualcosa di simile:

```
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;
  }
}
```

Si noti che per entrambi i casi di utilizzo di @HostBinding stiamo passando un valore stringa per quale proprietà vogliamo influenzare. Se non forniamo una stringa al decoratore, verrà utilizzato il nome del membro della classe. Nel primo @HostBinding, impostiamo staticamente l'attributo role al pulsante. Per il secondo esempio, la classe premuta verrà applicata quando isPressed è true

Leggi Direttive sugli attributi per influenzare il valore delle proprietà sul nodo host utilizzando il decoratore @HostBinding. online: https://riptutorial.com/it/angular2/topic/9455/direttive-sugli-attributi-per-influenzare-il-valore-delle-proprieta-sul-nodo-host-utilizzando-il-decoratore--hostbinding-

Capitolo 36: Dropzone in Angular2

Examples

Zona di rilascio

Libreria wrapper angolare 2 per Dropzone.

npm install angular2-dropzone-wrapper --save-dev

Carica il modulo per il modulo dell'app

```
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)
  ]
})
```

UTILIZZO DEI COMPONENTI

Sostituisci semplicemente l'elemento che verrebbe automaticamente passato a Dropzone con il componente dropzone.

```
<dropzone [config]="config" [message]="'Click or drag images here to upload'"
(error)="onUploadError($event)" (success)="onUploadSuccess($event)"></dropzone>
```

Crea un 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);
    }
}
```

Leggi Dropzone in Angular2 online: https://riptutorial.com/it/angular2/topic/10010/dropzone-in-angular2

Capitolo 37: Esempi di componenti avanzati

Osservazioni

Ricorda che Angular 2 riguarda esclusivamente la responsabilità singolare. Non importa quanto piccolo sia il tuo componente, dedica una logica separata per ogni singolo componente. Che si tratti di un pulsante, di un collegamento di ancoraggio di fantasia, di un'intestazione di dialogo o anche di un elemento secondario di sidenav.

Examples

Selettore immagini con anteprima

In questo esempio, creeremo un selettore di immagini che prevede l'anteprima della tua immagine prima del caricamento. L'anteprima supporta anche il trascinamento dei file nell'input. In questo esempio, coprirò solo il caricamento di singoli file, ma puoi armeggiare un po 'per far funzionare il caricamento di più file.

image-preview.html

Questo è il layout html della nostra anteprima dell'immagine

```
<!-- 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="">
```

immagine-preview.ts

Questo è il file principale per il nostro 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>
```

E questo è tutto. Molto più facile di quanto non fosse in AngularJS 1.x. Realmente ho realizzato questo componente basandomi su una versione precedente che ho realizzato in AngularJS 1.5.5.

Filtra i valori della tabella in base all'input

Importa `ReactiveFormsModule` e quindi

```
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();
  }

}
```

Leggi Esempi di componenti avanzati online: https://riptutorial.com/it/angular2/topic/5597/esempi-di-componenti-avanzati

Capitolo 38: Esempio di percorsi come / route / subroute per gli URL statici

Examples

Esempio di percorso di base con albero dei percorsi secondari

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}
    ]
  }
];
```

Leggi Esempio di percorsi come / route / subroute per gli URL statici online: https://riptutorial.com/it/angular2/topic/8910/esempio-di-percorsi-come---route---subroute-per-gli-url-statici

Capitolo 39: EventEmitter Service

Examples

Panoramica della classe

```
class EventEmitter extends Subject {
    constructor(isAsync?: boolean)
    emit(value?: T)
    subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

Componente di classe

```
@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);
    }
  }
}
```

Emmiting Events

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Cattura l'evento

Crea un servizio-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
    navchange: EventEmitter<number> = new EventEmitter();
    constructor() {}
    emitNavChangeEvent(number) {
        this.navchange.emit(number);
```

```
    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}
```

Creare un componente per utilizzare il servizio-

```
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);
    }
}
```

Esempio dal vivo

Un esempio dal vivo per questo può essere trovato qui .

Leggi EventEmitter Service online: https://riptutorial.com/it/angular2/topic/9159/eventemitter-service

Capitolo 40: Forme basate su dati angolari 2

Osservazioni

`this.myForm = this.formBuilder.group`

crea un oggetto modulo con la configurazione dell'utente e lo assegna alla variabile this.myForm.

`'loginCredentials': this.formBuilder.group`

metodo crea un gruppo di controlli che consistono in un **formControlName** es. `login` e valore `['', Validators.required]`, dove il primo parametro è il valore iniziale del form input e il secons è un validatore o un array di validatori come in `'email': ['', [Validators.required, customValidator]],`

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

Crea una serie di gruppi in cui l'indice del gruppo è **formGroupName** nell'array e vi si accede come:

```
<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)
    }))
  }
```

questo metodo di esempio aggiunge nuovo formGroup all'array. Attualmente l'accesso richiede di specificare il tipo di controllo a cui vogliamo accedere, in questo esempio questo tipo è: `<FormArray>`

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

le stesse regole di cui sopra si applicano alla rimozione di un controllo di modulo specifico dall'array

Examples

Modulo guidato dai dati

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}
  }
}
```

Markup 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>
```

Leggi Forme basate su dati angolari 2 online: https://riptutorial.com/it/angular2/topic/6463/forme-basate-su-dati-angolari-2

Capitolo 41: Http Interceptor

Osservazioni

Quello che facciamo con la classe HttpServiceLayer è estendere la classe Http da angolare e aggiungere la nostra logica ad essa.

Iniettiamo quindi quella classe nella classe bootstrap dell'applicazione e diciamo angolare che importiamo la classe Http, nella parte posteriore per inserire HttpServiceLayer.

Ovunque nel codice possiamo semplicemente importare

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

Ma la nostra classe sarà utilizzata per ogni chiamata.

Examples

Classe semplice Estensione della classe Http di 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 la nostra classe invece di Http di Angular

Dopo aver esteso la classe Http, dobbiamo dire a angular di usare questa classe invece della classe Http.

Per fare questo, nel nostro modulo principale (o in base alle esigenze, solo un particolare modulo), dobbiamo scrivere nella sezione dei provider:

```
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 è solo un servizio che uso per contenere alcuni valori per la durata dell'applicazione

Simple HttpClient AuthToken Interceptor (Angolare 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);
  }
}
```

Fornitura di Interceptor (some-module.module.ts)

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

Leggi Http Interceptor online: https://riptutorial.com/it/angular2/topic/1413/http-interceptor

Capitolo 42: Installazione di plugin di terze parti con angular-cli@1.0.0-beta.10

Osservazioni

È possibile installare altre librerie seguendo questo approccio, tuttavia, potrebbe essere necessario specificare il tipo di modulo, il file principale e l'estensione predefinita.

```
'lodash': {
  format: 'cjs',
  defaultExtension: 'js',
  main: 'index.js'
}
```

```
'moment': {
  main: 'moment.js'
}
```

Examples

Aggiunta di librerie jquery nel progetto angular-cli

1. Installa jquery via npm:

```
npm install jquery --save
```

Installa i tipi di codice per la libreria:

Per aggiungere tipizzazioni per una libreria, effettuare le seguenti operazioni:

```
typings install jquery --global --save
```

---

2. Aggiungi jquery al file angular-cli-build.js nell'array vendorNpmFiles:

   Questo è richiesto in modo che il sistema di build prelevi il file. Dopo l'installazione, angular-cli-build.js dovrebbe apparire così:

   Sfoglia i `node_modules` e cerca i file e le cartelle che desideri aggiungere alla cartella del venditore.

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');

module.exports = function(defaults) {
  return new Angular2App(defaults, {
```

```
    vendorNpmFiles: [
      // ...
      'jquery/dist/*.js'


    ]
  });
};
```

---

3. Configura i mapping SystemJS per sapere dove cercare jquery:

   La configurazione di SystemJS si trova in system-config.ts e dopo aver completato la configurazione personalizzata la sezione relativa dovrebbe essere simile a:

```
/** 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. Nel tuo src / index.html aggiungi questa riga

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

   Le tue altre opzioni sono:

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

o

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

e

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

---

5. Importazione e utilizzo della libreria jquery nei file di origine del progetto:

   Importa libreria jQuery nei tuoi file .ts di origine in questo modo:

```
declare var $:any;

@Component({
```

```
})
export class YourComponent {
  ngOnInit() {
    $.("button").click(function(){
       // now you can DO, what ever you want
     });
     console.log();
  }
}
```

Se hai seguito correttamente i passaggi dovresti avere una libreria jQuery che lavora nel tuo progetto. Godere!

Aggiungi la libreria di terze parti che non ha digitazioni

> Si noti, questo è solo per angular-cli fino alla versione 1.0.0-beta.10!

Alcune librerie o plugin potrebbero non avere digitazioni. Senza questi, TypeScript non può scriverli e quindi causa errori di compilazione. Queste librerie possono ancora essere utilizzate, ma in modo diverso rispetto ai moduli importati.

1. Includi un riferimento di script alla libreria sulla tua pagina (`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>
   ```

   - Questi script dovrebbero aggiungere un globale (ad esempio `THREE` , `mapbox` , `$` , ecc.) O allegare a un globale

2. Nel componente che richiede questi, utilizzare `declare` per inizializzare una variabile corrispondente al nome globale utilizzato dalla lib. Ciò consente a TypeScript di sapere che è già stato inizializzato. [1]

   ```
   declare var <globalname>: any;
   ```

   Alcune librerie si collegano alla `window` , che dovrebbe essere estesa per essere accessibile nell'app.

   ```
   interface WindowIntercom extends Window { Intercom: any; }
   declare var window: WindowIntercom;
   ```

3. Usa la lib nei tuoi componenti secondo necessità.

   ```
   @Component { ... }
   export class AppComponent implements AfterViewInit {
       ...
       ngAfterViewInit() {
           var geometry = new THREE.BoxGeometry( 1, 1, 1 );
           window.Intercom('boot', { ... }
       }
   }
   ```

- NOTA: alcune librerie potrebbero interagire con il DOM e dovrebbero essere utilizzate nel metodo appropriato del ciclo di vita dei componenti .

Leggi Installazione di plugin di terze parti con angular-cli@1.0.0-beta.10 online: https://riptutorial.com/it/angular2/topic/2328/installazione-di-plugin-di-terze-parti-con-angular-cli-1-0-0-beta-10

Capitolo 43: Interazioni componenti

Sintassi

- `<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.`

Parametri

Nome	Valore
pageCount	Utilizzato per indicare il numero di pagine da creare nel componente figlio.
pageNumberClicked	Nome della variabile di output nel componente figlio.
pageChanged	Funzione al componente padre che ascolta i componenti figlio in uscita.

Examples

Interazione padre-figlio con proprietà @Input e @Output

Abbiamo un componente DataList che mostra i dati estratti da un servizio. DataListComponent ha anche un PagerComponent dato che è figlio.

PagerComponent crea l'elenco dei numeri di pagina in base al numero totale di pagine ricevute da DataListComponent. PagerComponent consente inoltre a DataListComponent di sapere quando l'utente fa clic su qualsiasi numero di pagina tramite la proprietà Output.

```
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 elenca tutti i numeri di pagina. Impostiamo l'evento click su ognuno di essi in modo che possiamo far sapere al genitore il numero di pagina cliccato.

```
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
    }
}
```

Parent - Interazione bambino usando ViewChild

Viewchild offre un'interazione unidirezionale da genitore a figlio. Non vi è alcun feedback o output da parte del bambino quando viene utilizzato ViewChild.

Abbiamo un componente DataList che mostra alcune informazioni. DataListComponent ha PagerComponent come figlio. Quando l'utente effettua una ricerca su DataListComponent, ottiene un dato da un servizio e chiede a PagerComponent di aggiornare il layout di paging in base al nuovo numero di pagine.

```
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 { }
```

In questo modo è possibile chiamare le funzioni definite sui componenti figlio.

Il componente figlio non è disponibile fino al rendering del componente principale. Si tenta di accedere al bambino prima che i genitori `AfterViewInit` gancio cyle vita causerà un'eccezione.

Interazione genitore-figlio bidirezionale attraverso un servizio

Servizio utilizzato per la comunicazione:

```
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 genitore:

```
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 figlio:

```
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 {}
```

Leggi Interazioni componenti online: https://riptutorial.com/it/angular2/topic/7400/interazioni-componenti

Capitolo 44: Interazioni componenti

introduzione

Condividi le informazioni tra diverse direttive e componenti.

Examples

Passa i dati da genitore a figlio con il bind di input

HeroChildComponent ha due proprietà di input, tipicamente adornate con decorazioni @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;
}
```

Intercetta le proprietà di input con un setter

Utilizzare un setter di proprietà di input per intercettare e agire su un valore dal genitore.

Il setter della proprietà di input del nome nel figlio NameChildComponent taglia lo spazio bianco da un nome e sostituisce un valore vuoto con il testo predefinito.

```
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; }
}
```

Ecco il NameParentComponent che mostra le varianti del nome incluso un nome con tutti gli spazi:

```
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  '];
}
```

Il genitore ascolta l'evento figlio

Il componente figlio espone una proprietà EventEmitter con la quale emette eventi quando succede qualcosa. Il genitore si collega a quella proprietà dell'evento e reagisce a quegli eventi.

La proprietà EventEmitter del child è una proprietà di output, tipicamente decorata con una decorazione @Output come vista in questo 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;
  }
}
```

Facendo clic su un pulsante si attiva l'emissione di un vero o falso (il payload booleano).

Il genitore VoteTakerComponent associa un gestore di eventi (onVoted) che risponde al payload dell'evento figlio ($ event) e aggiorna un contatore.

```
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++;
  }
}
```

Il genitore interagisce con il figlio tramite la variabile locale

Un componente padre non può utilizzare l'associazione dati per leggere le proprietà figlio o richiamare i metodi figlio. Possiamo fare entrambe le cose creando una variabile di riferimento modello per l'elemento figlio e quindi facendo riferimento a tale variabile all'interno del modello principale come mostrato nell'esempio seguente.

Abbiamo un figlio CountdownTimerComponent che conta più volte fino a zero e lancia un razzo. Ha i metodi di avvio e arresto che controllano l'orologio e visualizza un messaggio di stato del conto alla rovescia nel proprio modello.

```
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);
  }
}
```

Vediamo CountdownLocalVarParentComponent che ospita il componente timer.

```
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 { }
```

Il componente principale non può eseguire il bind dei dati sui metodi di avvio e arresto del figlio né sulla sua proprietà secondi.

Possiamo posizionare una variabile locale (#timer) sul tag () che rappresenta il componente figlio. Questo ci dà un riferimento al componente figlio stesso e alla possibilità di accedere a qualsiasi sua proprietà o metodo dal modello principale.

In questo esempio, colleghiamo i pulsanti padre all'avvio e all'arresto del figlio e utilizziamo l'interpolazione per visualizzare la proprietà secondi del figlio.

Qui vediamo il genitore e il bambino che lavorano insieme.

Il genitore chiama un ViewChild

L'approccio variabile locale è semplice e facile. Ma è limitato perché il cablaggio genitore-figlio deve essere fatto interamente all'interno del modello principale. Il componente principale non ha accesso al bambino.

Non è possibile utilizzare la tecnica della variabile locale se un'istanza della classe del componente padre deve leggere o scrivere valori del componente figlio o deve chiamare metodi del componente figlio.

Quando la classe del componente genitore richiede quel tipo di accesso, iniettiamo il componente figlio nel genitore come ViewChild.

Illustreremo questa tecnica con lo stesso esempio di Countdown Timer. Non cambieremo il suo aspetto o comportamento. Il figlio CountdownTimerComponent è lo stesso.

Stiamo passando dalla variabile locale alla tecnica ViewChild esclusivamente a scopo dimostrativo. Ecco il genitore, 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(); }
}
```

Ci vuole un po 'più di lavoro per ottenere la vista figlio nella classe del componente genitore.

Importiamo i riferimenti al decoratore ViewChild e al gancio del ciclo di vita AfterViewInit.

Iniettiamo il figlio CountdownTimerComponent nella proprietà timerComponent privata tramite la decorazione della proprietà @ViewChild.

La variabile locale #timer è scomparsa dai metadati del componente. Invece leghiamo i pulsanti ai metodi di avvio e arresto del componente principale e presentiamo i secondi di spunta in un'interpolazione attorno al metodo dei secondi del componente genitore.

Questi metodi accedono direttamente al componente timer iniettato.

L'hook del ciclo di vita ngAfterViewInit è una ruga importante. Il componente del timer non è disponibile fino a quando Angular non visualizza la vista genitore. Quindi visualizziamo inizialmente 0 secondi.

Quindi Angular chiama l'hook del ciclo di vita ngAfterViewInit in cui è troppo tardi per aggiornare la visualizzazione della visualizzazione genitore dei secondi del conto alla rovescia. La regola del flusso di dati unidirezionale di Angular ci impedisce di aggiornare la vista genitore nello stesso ciclo. Dobbiamo aspettare un turno prima di poter visualizzare i secondi.

Usiamo setTimeout per aspettare un segno di spunta e quindi rivedere il metodo dei secondi in modo che prenda i valori futuri dal componente timer.

Genitori e figli comunicano tramite un servizio

Un componente principale e i suoi figli condividono un servizio la cui interfaccia consente la comunicazione bidirezionale all'interno della famiglia.

L'ambito dell'istanza del servizio è il componente principale e i relativi elementi secondari. I componenti esterni alla sottostruttura di questo componente non hanno accesso al servizio o alle loro comunicazioni.

Questo MissionService collega MissionControlComponent a più bambini 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);
  }
}
```

Il MissionControlComponent fornisce sia l'istanza del servizio che condivide con i suoi figli (attraverso l'array di metadati del provider) e inietta tale istanza in se stessa attraverso il suo costruttore:

```
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; }
  }
```

```
}
```

Anche l'AstronautComponent inietta il servizio nel suo costruttore. Ogni AstronautComponent è figlio di MissionControlComponent e pertanto riceve l'istanza di servizio del genitore:

```
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();
  }
}
```

Si noti che acquisiamo la sottoscrizione e annulliamo l'iscrizione quando DistronautComponent viene distrutto. Questo è un passo per la perdita di memoria. Non ci sono rischi reali in questa app perché la durata di un AstronautComponent è uguale alla durata della stessa app. Ciò non sarebbe sempre vero in un'applicazione più complessa.

Non aggiungiamo questa guardia a MissionControlComponent perché, in quanto genitore, controlla la durata del MissionService. Il registro cronologia dimostra che i messaggi viaggiano in entrambe le direzioni tra i genitori MissionControlComponent e i bambini AstronautComponent, facilitati dal servizio:

Leggi Interazioni componenti online: https://riptutorial.com/it/angular2/topic/9454/interazioni-

componenti

Capitolo 45: Lifecycle Hooks

Osservazioni

Disponibilità degli eventi

`AfterViewInit` e `AfterViewChecked` sono disponibili solo **in Componenti** e **non in Direttive** .

Ordine degli eventi

- `OnChanges` (più volte)
- `OnInit` (una volta)
- `DoCheck` (più volte)
- `AfterContentInit` (una volta)
- `AfterContentChecked` (più volte)
- `AfterViewInit` (una volta) (solo Component)
- `AfterViewChecked` (più volte) (solo Component)
- `OnDestroy` (una volta)

Ulteriori letture

- Documentazione angolare - Ganci del ciclo di vita

Examples

OnInit

Attivato quando le proprietà di componente o direttiva sono state inizializzate.

(Prima di quelli delle direttive minori)

```
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 !');
    }
}
```

OnDestroy

Fired quando l'istanza di componente o direttiva viene distrutta.

```
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

Attivato quando una o più proprietà del componente o della direttiva sono state modificate.

```
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);
    }
}
```

L'evento di cambiamento verrà registrato

```
name: {
    currentValue: 'new name value',
    previousValue: 'old name value'
},
message: {
    currentValue: 'new message value',
    previousValue: 'old message value'
}
```

AfterContentInit

Il fuoco dopo l'inizializzazione del contenuto del componente o della direttiva è terminato.

(Subito dopo 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

Fuoco dopo che la vista è stata completamente inizializzata.

(Disponibile solo per componenti)

```
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

Viene eseguito dopo l'inizializzazione della vista componente e di qualsiasi sua vista secondaria. Questo è un utile hook per il ciclo di vita dei plugin al di fuori dell'ecosistema di Angular 2. Ad esempio, è possibile utilizzare questo metodo per inizializzare un selettore di date jQuery in base alla revisione che Angular 2 ha eseguito il rendering.

```
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

L'incendio dopo il controllo della vista, del componente, è terminato.

(Disponibile solo per componenti)

```
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

Permette di ascoltare le modifiche solo su proprietà specificate

```
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));
        }
    }
}
```

Leggi Lifecycle Hooks online: https://riptutorial.com/it/angular2/topic/2935/lifecycle-hooks

Capitolo 46: Mocking @ ngrx / Store

introduzione

@ ngrx / Store sta diventando sempre più utilizzato nei progetti Angular 2. Come tale, lo Store deve essere iniettato nel costruttore di qualsiasi componente o servizio che desidera utilizzarlo. Test delle unità Il negozio non è così facile come testare un semplice servizio. Come con molti problemi, ci sono una miriade di modi per implementare soluzioni. Tuttavia, la ricetta di base è scrivere una classe di simulazione per l'interfaccia di Observer e scrivere una classe di simulazione per Store. Quindi puoi iniettare Store come fornitore nel tuo TestBed.

Parametri

nome	descrizione
valore	prossimo valore da osservare
errore	descrizione
sbagliare	errore da lanciare
super	descrizione
azione $	simulare Observer che non fa nulla se non viene definito di farlo nella classe simulata
actionReducer $	simulare Observer che non fa nulla se non viene definito di farlo nella classe simulata
obs $	finto osservabile

Osservazioni

Observer è un generico, ma deve essere di tipo `any` per evitare la complessità del testing unitario. La ragione di questa complessità è che il costruttore di Store si aspetta argomenti Observer con diversi tipi generici. Usando `any` evita questa complicazione.

È possibile trasferire valori null nel super costruttore di StoreMock, ma ciò limita il numero di asserzioni che possono essere utilizzate per testare la classe più avanti.

Il componente utilizzato in questo esempio viene utilizzato come contesto per il modo in cui si farebbe l'iniezione di Store come fornitura nell'impostazione di test.

Examples

Observer Mock

```
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();
    });
});
```

Test unitario per componente con Mock Store

Questo è un test unitario di un componente che ha *Store* come dipendenza. Qui, stiamo creando una nuova classe chiamata *MockStore* che viene iniettata nel nostro componente invece del solito Store.

```
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();
  }));
```

Test unitario per componenti Spying On Store

Questo è un test unitario di un componente che ha *Store* come dipendenza. Qui, siamo in grado di utilizzare un negozio con lo "stato iniziale" predefinito, impedendogli di effettuare azioni di invio quando viene chiamato *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 - Mock Observable (servizio + componente)

servizio

- Ho creato il servizio postale con il metodo 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

- Ho creato un componente con il parametro result e la funzione postExample che chiama su postService.
- quando il risultato del post richiesto supera il parametro result dovrebbe essere 'Success' else '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'
      )
  }
}
```

servizio di test

- quando vuoi testare il servizio che usa http devi usare mockBackend. e inietti ad esso.
- è necessario anche iniettare 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 di prova

```
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);
      });
  }));
});
```

Negozio semplice

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;
    }
}
```

negozio / 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
    })
  }
}
```

risultato 0 1 3 6

Leggi Mocking @ ngrx / Store online: https://riptutorial.com/it/angular2/topic/8038/mocking---ngrx---store

Capitolo 47: Modelli

introduzione

I modelli sono molto simili ai modelli in Angular 1, anche se ci sono molte piccole modifiche sintattiche che rendono più chiaro cosa sta succedendo.

Examples

Angolare 2 modelli

UN MODELLO SEMPLICE

Iniziamo con un modello molto semplice che mostra il nostro nome e la nostra cosa preferita:

```
<div>
  Hello my name is {{name}} and I like {{thing}} quite a lot.
</div>
```

`{}`: RENDERING

Per rendere un valore, possiamo usare la sintassi double-curly standard:

```
My name is {{name}}
```

I pipe, precedentemente noti come "Filtri", trasformano un valore in un nuovo valore, come la localizzazione di una stringa o la conversione di un valore a virgola mobile in una rappresentazione di valuta:

`[]`: PROPRIETÀ VINCOLANTI

Per risolvere e associare una variabile a un componente, utilizzare la sintassi []. Se nel nostro componente abbiamo this.currentVolume, lo passeremo al nostro componente e i valori rimarranno sincronizzati:

```
<video-control [volume]="currentVolume"></video-control>
(): HANDLING EVENTS
```

`()`: MANEGGIARE EVENTI Per ascoltare un evento su un componente, usiamo la sintassi ()

```
<my-component (click)="onClick($event)"></my-component>
```

`[()]`: BINDING DI DUE-WAY DATA

Per mantenere aggiornato l'associazione in base all'input dell'utente e ad altri eventi, utilizzare la sintassi [()]. Pensala come una combinazione di gestione di un evento e associazione di una

proprietà:

<input [(ngModel)] = "myName"> Il valore this.myName del componente rimarrà sincronizzato con il valore di input.

`*` : L'ASTERISCO

Indica che questa direttiva considera questo componente come un modello e non lo disegna così com'è. Ad esempio, ngFor prende il nostro e lo timbra per ogni oggetto negli articoli, ma non rende mai il nostro iniziale poiché è un modello:

```
<my-component *ngFor="#item of items">
</my-component>
```

Altre direttive simili che funzionano su modelli piuttosto che su componenti renderizzati sono * ngIf e * ngSwitch.

Leggi Modelli online: https://riptutorial.com/it/angular2/topic/9471/modelli

Capitolo 48: moduli

introduzione

I moduli angolari sono contenitori per diverse parti della tua app.

Puoi avere moduli nidificati, il tuo `app.module` sta già annidando altri moduli come `BrowserModule` e puoi aggiungere `RouterModule` e così via.

Examples

Un modulo semplice

Un modulo è una classe con il decoratore `@NgModule` . Per creare un modulo aggiungiamo `@NgModule` passando alcuni parametri:

- `bootstrap` : il componente che sarà la radice della tua applicazione. Questa configurazione è presente solo sul modulo radice
- `declarations` : risorse dichiarate dal modulo. Quando aggiungi un nuovo componente devi aggiornare le dichiarazioni (`ng generate component` fa automaticamente)
- `exports` : Risorse che il modulo esporta che può essere utilizzato in altri moduli
- `imports` : risorse che il modulo utilizza da altri moduli (sono accettate solo le classi di moduli)
- `providers` : risorse che possono essere iniettate (di) in un componente

Un semplice esempio:

```
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 { }
```

Moduli di annidamento

I moduli possono essere nidificati utilizzando il parametro `imports` del decoratore `@NgModule` .

Possiamo creare un `core.module` nella nostra applicazione che conterrà cose generiche, come un `ReservePipe` (una pipe che inverte una stringa) e raggruppa quelli in questo modulo:

```
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 { }
```

Quindi, `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 { }
```

Leggi moduli online: https://riptutorial.com/it/angular2/topic/10840/moduli

Capitolo 49: Moduli funzione

Examples

Un modulo funzione

```
// 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 { }
```

Ora, nella tua root (solitamente `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 { }
```

Leggi Moduli funzione online: https://riptutorial.com/it/angular2/topic/6551/moduli-funzione

Capitolo 50: NGRX

introduzione

Ngrx è una potente libreria che puoi usare con **Angular2** . L'idea alla base è quella di unire due concetti che giocano bene insieme per avere **un'app reattiva** con un **contenitore di stato** prevedibile: - [Redux] [1] - [RxJs] [2] I principali vantaggi: - Condivisione dei dati nella tua app tra i tuoi componenti sta andando più facile - Testare la tua logica di base dell'app consiste nel testare pure funzioni, senza alcuna dipendenza da Angular2 (molto semplice!) [1]: http://redux.js.org [2]: http: // reactivex. io / rxjs

Examples

Esempio completo: Login / logout di un utente

Prerequisiti

Questo argomento **non** riguarda Redux e / o Ngrx:

- Devi stare tranquillo con Redux
- Almeno capire le basi di RxJs e modello osservabile

Per prima cosa, definiamo un esempio fin dall'inizio e giochiamo con un codice:

Come sviluppatore, voglio:

1. Avere un'interfaccia `IUser` che definisce le proprietà di un `User`
2. Dichiarare le azioni che useremo in seguito per manipolare l' `User` nello `Store`
3. Definire lo stato iniziale di `UserReducer`
4. Crea il riduttore `UserReducer`
5. Importa il nostro `UserReducer` nel nostro modulo principale per costruire lo `Store`
6. Utilizza i dati dallo `Store` per visualizzare le informazioni nella nostra vista

Avviso spoiler : se vuoi provare subito la demo o leggere il codice prima ancora di iniziare, ecco un Plunkr (visualizzazione incorporata o visualizzazione di esecuzione).

1) Definire `IUser` interfaccia `IUser`

Mi piace dividere le mie interfacce in due parti:

- Le proprietà che otterremo da un server
- Le proprietà che definiamo solo per l'interfaccia utente (se un pulsante deve ruotare ad esempio)

Ed ecco l'interfaccia `IUser` che `IUser` :

`user.interface.ts`

```
export interface IUser {
  // from server
  username: string;
  email: string;

  // for UI
  isConnecting: boolean;
  isConnected: boolean;
};
```

---

2) Dichiarare le azioni per manipolare l' `User`

Ora dobbiamo pensare a quale tipo di azioni devono essere gestite dai nostri ***riduttori*** .
Diciamo qui:

`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'
};
```

Ma prima di utilizzare tali azioni, lasciami spiegare perché avremo bisogno di un servizio per inviare **alcune** di queste azioni per noi:

Diciamo che vogliamo connettere un utente. Quindi faremo clic su un pulsante di accesso ed ecco cosa succederà:

- Clicca sul pulsante
- Il componente intercetta l'evento e chiama `userService.login`
- `userService.login` metodo `userService.login dispatch` un evento per aggiornare la nostra proprietà del negozio: `user.isConnecting`
- Viene lanciata una chiamata HTTP (useremo un `setTimeout` nella demo per simulare il **comportamento asincrono**)
- Una volta terminata la chiamata `HTTP` , invieremo un'altra azione per avvisare il nostro archivio che un utente è registrato

`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) Definire lo stato iniziale di `UserReducer`

`user.state.ts`

```
export const UserFactory: IUser = () => {
  return {
    // from server
    username: null,
    email: null,

    // for UI
    isConnecting: false,
    isConnected: false,
    isDisconnecting: false
  };
};
```

---

4) Crea il riduttore `UserReducer`

Un riduttore accetta 2 argomenti:

- Lo stato attuale
- `Action` di tipo `Action<{type: string, payload: any}>`

Promemoria: un riduttore deve essere inizializzato ad un certo punto

Come abbiamo definito lo stato predefinito del nostro riduttore nella parte 3, saremo in grado di usarlo in questo modo:

`user.reducer.ts`

```
export const UserReducer: ActionReducer<IUser> = (user: IUser, action: Action) => {
  if (user === null) {
    return userFactory();
  }

  // ...
}
```

Speriamo che ci sia un modo più semplice per scrivere che usando la nostra funzione di `factory` per restituire un oggetto e all'interno del riduttore usare un valore di parametri di default (ES6):

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  // ...
}
```

Quindi, dobbiamo gestire tutte le azioni del nostro riduttore: ***SUGGERIMENTO*** : utilizzare la funzione `Object.assign` ES6 per mantenere il nostro stato immutabile

```
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) Importa il nostro `UserReducer` nel nostro modulo principale per costruire lo `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) Utilizzare i dati dallo `Store` per visualizzare le informazioni nella nostra vista

Tutto è ora pronto per il lato logico e dobbiamo solo mostrare ciò che vogliamo in due componenti:

- `UserComponent` : **[Componente stupido]** `UserComponent` semplicemente l'oggetto utente dallo store usando la proprietà `@Input` e la pipe `async` . In questo modo, il componente riceverà l'utente solo una volta disponibile (e l' `user` sarà di tipo `IUser` e non di tipo `Observable<IUser>` !)
- `LoginComponent` **[Componente intelligente]** `LoginComponent` direttamente lo `Store` in questo componente e lavoreremo solo `user` come `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();
  }
}
```

Dato che `Ngrx` è un'unione dei concetti di `Redux` e `RxJs` , può essere piuttosto difficile capire all'inizio un outs. Ma questo è uno schema potente che ti consente, come abbiamo visto in questo esempio, di avere *un'app reattiva* ed è possibile condividere facilmente i tuoi dati. Non dimenticare che c'è un Plunkr disponibile e puoi forarlo per fare i tuoi test!

Spero sia stato utile anche se l'argomento è abbastanza lungo, evviva!

Leggi NGRX online: https://riptutorial.com/it/angular2/topic/8086/ngrx

Capitolo 51: Operaio di servizio

introduzione

Vedremo come configurare un servizio che funzioni su angolare, per consentire alla nostra app Web di avere funzionalità offline.

Un lavoratore del servizio è uno script speciale che viene eseguito in background nel browser e gestisce le richieste di rete verso una determinata origine. Inizialmente è installato da un'app e rimane residente sul dispositivo / dispositivo dell'utente. Viene attivato dal browser quando viene caricata una pagina dalla sua origine e ha la possibilità di rispondere alle richieste HTTP durante il caricamento della pagina

Examples

Aggiungi il Service Worker alla nostra app

Per prima cosa nel caso in cui tu stia consultando mobile.angular.io la flag --mobile non funziona più.

Quindi, per iniziare, possiamo creare un progetto normale con cli angolare.

```
ng new serviceWorking-example
cd serviceWorking-example
```

Ora la cosa importante, per dire a angular cli che vogliamo usare il service worker, dobbiamo fare:

ng set apps.0.serviceWorker = true

Se per qualche motivo non hai installato @ angular / service-worker, verrà visualizzato un messaggio:

> Il tuo progetto è configurato con serviceWorker = true, ma @ angular / service-worker non è installato. Eseguire `npm install --save-dev @angular/service-worker` e riprovare, oppure eseguire `ng set apps.0.serviceWorker=false` nel proprio .angular-cli.json.

Controlla il .angular-cli.json e ora dovresti vedere questo: "serviceWorker": true

Quando questo flag è true, le build di produzione verranno impostate con un worker di servizio.

Un file ngsw-manifest.json verrà generato (o incrementato nel caso in cui abbiamo creato un ngsw-manifest.json nella radice del progetto, di solito questo è fatto per specificare il routing, in un futuro questo sarà probabilmente fatto automaticamente) in dist / root, e lo script di worker del servizio verrà copiato lì. Un breve script verrà aggiunto a index.html per registrare il lavoratore del servizio.

Ora se costruiamo l'app in modalità di produzione ng build --prod

E controlla dist / cartella.

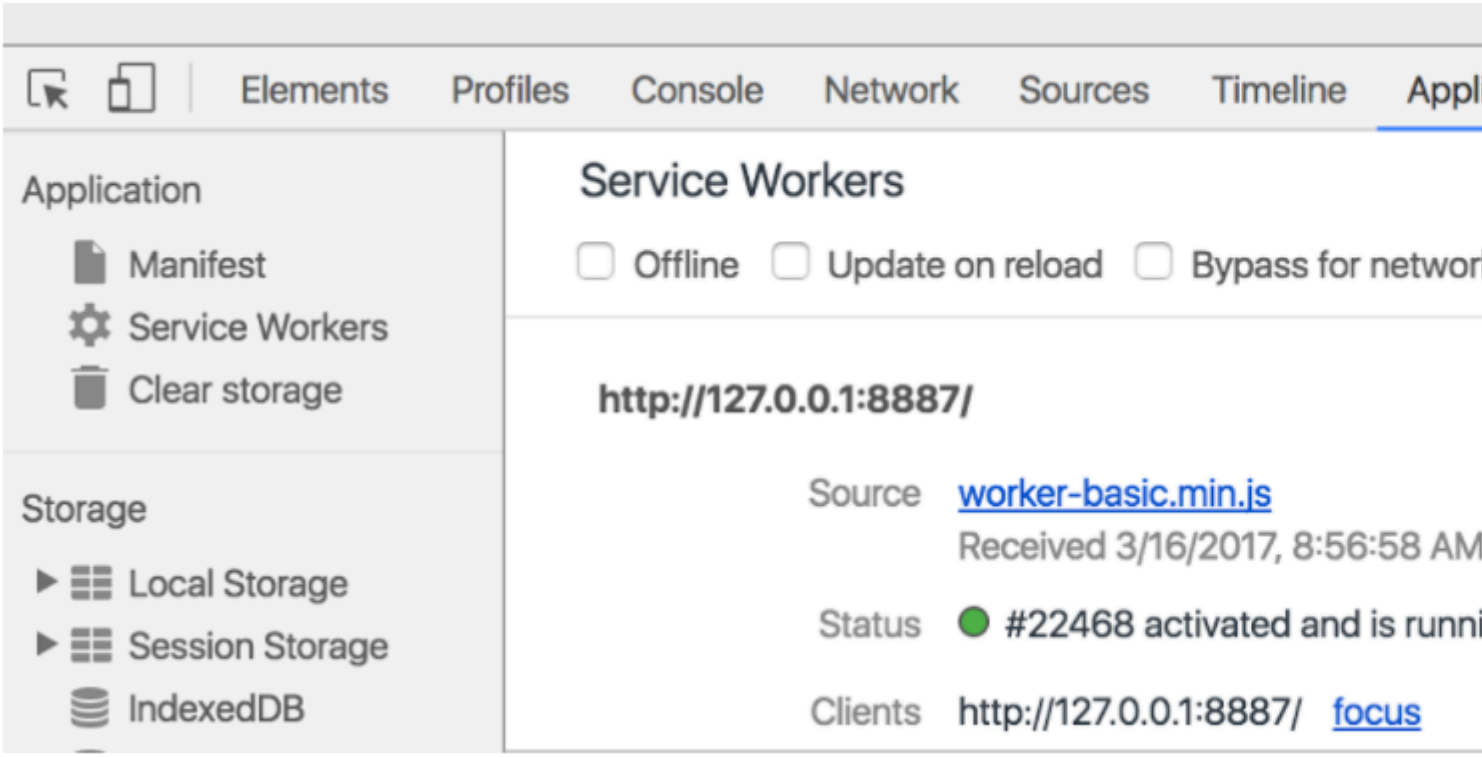
Vedrai tre nuovi file lì:

- operai basic.min.js
- sw-register.HASH.bundle.js
- ngsw-manifest.json

Inoltre, index.html ora include questo script sw register, che registra per noi un Angular Service Worker (ASW).

Aggiorna la pagina nel browser (servita da Web Server per Chrome)

Apri Strumenti per sviluppatori. Vai all'Applicazione -> Operatori di servizio



Bene, ora il Service Worker è attivo e funzionante!

Ora la nostra applicazione dovrebbe caricarsi più velocemente e dovremmo essere in grado di utilizzare l'app offline.

Ora se abiliti la modalità offline nella console di Chrome, dovresti vedere che la nostra app in http: // localhost: 4200 / index.html funziona senza connessione a Internet.

Ma in http: // localhost: 4200 / abbiamo un problema e non si carica, questo è dovuto alla cache del contenuto statico che serve solo i file elencati nel manifest.

Ad esempio, se il manifest dichiara un URL di /index.html, le richieste a /index.html riceveranno risposta dalla cache, ma una richiesta a / o / some / route andrà alla rete.

È qui che entra in gioco il plugin di reindirizzamento delle rotte. Legge una configurazione di

routing dal manifest e reindirizza i percorsi configurati a una determinata route dell'indice.

Attualmente, questa sezione di configurazione deve essere scritta a mano (19-7-2017). Alla fine, verrà generato dalla configurazione del percorso presente nella sorgente dell'applicazione.

Quindi se ora creiamo o ngsw-manifest.json nella radice del progetto

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

E costruiamo di nuovo la nostra app, ora quando andiamo a http: // localhost: 4200 / , dovremmo essere reindirizzati a http: // localhost: 4200 / index.html .

Per ulteriori informazioni sul routing leggere la documentazione ufficiale qui

Qui puoi trovare più documentazione sui lavoratori del servizio:

https://developers.google.com/web/fundamentals/getting-started/primers/service-workers

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

E qui puoi vedere un modo alternativo per implementare il servizio che funziona usando la libreria precache SW:

https://coryrylan.com/blog/fast-offline-angular-apps-with-service-workers

Leggi Operaio di servizio online: https://riptutorial.com/it/angular2/topic/10809/operaio-di-servizio

Capitolo 52: Ordinare per tubo

introduzione

Come scrivere la pipe per l'ordine e usarla.

Examples

La pipa

L'implementazione del tubo

```
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
      });
    }
  }
}
```

Come usare la pipe nell'HTML: ordine crescente per nome

```
<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>
```

Come usare la pipe nell'HTML: ordine decrescente per nome

```
<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>
```

Leggi Ordinare per tubo online: https://riptutorial.com/it/angular2/topic/8969/ordinare-per-tubo

Capitolo 53: Ottimizzazione del rendering con ChangeDetectionStrategy

Examples

Default vs OnPush

Considerare il seguente componente con un ingresso `myInput` e un valore interno chiamato `someInternalValue` . Entrambi sono usati nel modello di 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;

  // ...
}
```

Per impostazione predefinita, `changeDetection:` proprietà nel decoratore del componente sarà impostata su `ChangeDetectionStrategy.Default` ; implicito nell'esempio. In questa situazione, qualsiasi modifica a uno qualsiasi dei valori nel modello attiverà un re-rendering di `MyComponent` . In altre parole, se cambio `myInput` o `someInternalValue` angolo 2 eserciterà energia e re-render il componente.

Supponiamo, tuttavia, che vogliamo solo ri-renderizzare quando cambiano gli input. Considera il seguente componente con `changeDetection:` impostato su `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;

  // ...
}
```

Impostando `changeDetection:` su `ChangeDetectionStrategy.OnPush` , `MyComponent` verrà ri-renderizzato solo quando i suoi input cambiano. In questo caso, `myInput` dovrà ricevere un nuovo valore dal suo genitore per attivare un re-rendering.

Leggi Ottimizzazione del rendering con ChangeDetectionStrategy online: https://riptutorial.com/it/angular2/topic/2644/ottimizzazione-del-rendering-con-changedetectionstrategy

Capitolo 54: personalizzato ngx-bootstrap datepicker + input

Examples

personalizzato 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('/','-'));
  }

}
```

Leggi personalizzato ngx-bootstrap datepicker + input online:
https://riptutorial.com/it/angular2/topic/10549/personalizzato-ngx-bootstrap-datepicker-plus-input

Capitolo 55: Pigro che carica un modulo

Examples

Esempio di caricamento pigro

I moduli di **caricamento pigri** ci aiutano a ridurre il tempo di avvio. Con il caricamento lazy la nostra applicazione non ha bisogno di caricare tutto in una volta, ha solo bisogno di caricare ciò che l'utente si aspetta di vedere quando l'app carica prima. I moduli caricati pigramente verranno caricati solo quando l'utente naviga verso i propri percorsi.

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 {}
```

Non c'è niente di speciale in LazyModule se non ha il proprio routing e un componente chiamato LazyComponent (ma non è necessario nominare il modulo o simliar così).

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 {}
```

Leggi Pigro che carica un modulo online: https://riptutorial.com/it/angular2/topic/7751/pigro-che-carica-un-modulo

Capitolo 56: Pipes

introduzione

Il tubo `|` il carattere viene utilizzato per applicare i tubi in Angular 2. I tubi sono molto simili ai filtri in AngularJS in quanto entrambi aiutano a trasformare i dati in un formato specificato.

Parametri

Funzione / Parametro	Spiegazione
@Pipe ({nome, puro})	i metadati per pipe, devono precedere immediatamente la classe di pipe
nome: *stringa*	cosa userai all'interno del modello
puro: *booleano*	il valore predefinito è true, contrassegnalo come falso per fare rivedere il pipe più spesso
transform (value, args []?)	la funzione che viene chiamata per trasformare i valori nel modello
valore: *qualsiasi*	il valore che desideri trasformare
args: *any []*	gli argomenti che potresti aver bisogno di includere nella tua trasformazione. Contrassegnare argomenti opzionali con il? operatore come tale trasforma (valore, arg1, arg2?)

Osservazioni

Questo argomento copre Angular2 Pipes , un meccanismo per trasformare e formattare i dati all'interno di modelli HTML in un'applicazione Angular2.

Examples

Tubi concatenati

I tubi possono essere incatenati.

```
<p>Today is {{ today | date:'fullDate' | uppercase}}.</p>
```

Tubi personalizzati

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 { }
```

Tubi incorporati

Angular2 viene fornito con alcuni tubi incorporati:

Tubo	uso	Esempio
DatePipe	date	{{ dateObj | date }} // output is 'Jun 15, 2015'

Tubo	uso	Esempio
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

Ce ne sono altri Guarda qui per la loro documentazione.

Esempio

hotel-reservation.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>
```

Produzione

```
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.
```

Debugging con JsonPipe

JsonPipe può essere utilizzato per il debug dello stato di qualsiasi dato interno.

Codice

```
@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]}};
}
```

Produzione

```
Without JSON Pipe:
object
With JSON pipe:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

Tubo personalizzato disponibile a livello globale

Per rendere disponibile un'ampia gamma di applicazioni personalizzate, durante il bootstrap dell'applicazione, estendere 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 qui: https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2

Creazione di pipe personalizzate

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{ }
```

Scomporre i valori asincroni con il tubo asincrono

```
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']);
}
```

diventa:

```
<h1>Hello, Misko</h1>
Your Friends are:
<ul>
  <li>
    Igor
  </li>
</ul>
```

Estensione di un tubo esistente

```
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
    }
  }
}
```

Stateful Pipes

Angular 2 offre due diversi tipi di tubi: stateless e stateless. I pipe sono stateless per impostazione predefinita. Tuttavia, possiamo implementare pipe stateful impostando la proprietà `pure` su `false` . Come puoi vedere nella sezione dei parametri, puoi specificare un `name` e dichiarare se la pipe deve essere pura o no, ovvero statica o stateless. Mentre i dati fluiscono attraverso una pipe senza stato (che è una pura funzione) che **non** ricorda nulla, i dati possono essere gestiti e ricordati da pipe stateful. Un buon esempio di pipe stateful è `AsyncPipe` fornito da Angular 2.

Importante

Si noti che la maggior parte dei tubi dovrebbe rientrare nella categoria dei tubi senza stato. Questo è importante per le prestazioni, dal momento che Angular può ottimizzare i tubi senza stato per il rilevatore di modifiche. Quindi usate prudentemente i tubi di stato. In generale, l'ottimizzazione dei tubi in Angular 2 ha un importante miglioramento delle prestazioni rispetto ai filtri in 1.x angolare. In Angular 1 il ciclo di digest ha sempre dovuto rieseguire tutti i filtri anche se i dati non sono stati modificati. In Angular 2, una volta calcolato il valore di una pipe, il rilevatore di modifiche sa di non eseguire nuovamente questa pipe a meno che l'input non cambi.

Implementazione di una pipe stateful

```
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);
    }
  }
}
```

È quindi possibile utilizzare la pipa come al solito:

```
{{ 1000 | countdown:50 }}
{{ 300 | countdown }}
```

È importante che anche la tua pipe implementa l'interfaccia di `OnDestroy` modo che tu possa ripulire una volta distrutta la pipa. Nell'esempio sopra, è necessario cancellare l'intervallo per evitare perdite di memoria.

Dynamic Pipe

Scenario di utilizzo: una vista tabella è composta da diverse colonne con un diverso formato di dati che deve essere trasformato con pipe differenti.

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>
```

Risultato

```
| 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   |
-------------------------------------------------------------------
```

Test di una pipa

Dato un tubo che inverte una corda

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}
```

Può essere testato configurando il file spec come questo

```
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');
  }));
});
```

Leggi Pipes online: https://riptutorial.com/it/angular2/topic/1165/pipes

Capitolo 57: redox angolare

Examples

Di base

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;
```

store.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;
  }
}
```

actions.ts

```
import {Action} from "redux";
export const EXAMPLE_CHANGED = 'EXAMPLE CHANGED';

export interface UpdateAction extends Action {
  example: string;
}
```

Ottieni lo stato attuale

```
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);
   }
}
```

cambia stato

```
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"));
   }
}
```

actions.ts

```
export interface UpdateExapleAction extends Action {
```

```
  example?: string;
}

export const updateExample: ActionCreator<UpdateExapleAction> =
   (newVal) => ({
     type: EXAMPLE_CHANGED,
     example: newVal
});
```

Aggiungi lo strumento di chrome 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}
    ];
```

installare Redux DevTools chrome extention

Leggi redox angolare online: https://riptutorial.com/it/angular2/topic/10652/redox-angolare

Capitolo 58: Rilevazione degli eventi di ridimensionamento

Examples

Un componente in ascolto sull'evento di ridimensionamento della finestra.

Supponiamo di avere un componente che si nasconderà ad una certa larghezza della finestra.

```
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;
    }
  }
}
```

Un tag `p` nel nostro modello si nasconde ogni volta che `visible` è falso. `visible` cambierà valore ogni volta che viene richiamato il gestore di eventi `onResize` . La sua chiamata si verifica ogni volta che si visualizza una `window:resize` scattare un evento.

Leggi Rilevazione degli eventi di ridimensionamento online:
https://riptutorial.com/it/angular2/topic/5276/rilevazione-degli-eventi-di-ridimensionamento

Capitolo 59: Routing

Examples

Routing di base

Il router abilita la navigazione da una vista all'altra in base alle interazioni dell'utente con l'applicazione.

Di seguito sono riportate le fasi di implementazione del routing di base in Angular 2:

Precauzione di base : assicurati di avere il tag

```
    <base href='/'>
```

come il primo figlio sotto il tag head nel file index.html. Questo tag indica che la cartella dell'app è la root dell'applicazione. Angular 2 saprebbe quindi organizzare i tuoi collegamenti.

Il primo passo è controllare se stai puntando a corrette / ultime dipendenze di routing in package.json -

```
"dependencies": {
  ......
  "@angular/router": "3.0.0-beta.1",
  ......
}
```

Il secondo passo è definire il percorso come da sua definizione di classe -

```
class Route {
  path : string
  pathMatch : 'full'|'prefix'
  component : Type|string
  .........
}
```

In un file di rotte (`route/routes.ts`), importa tutti i componenti che devi configurare per percorsi di routing differenti. Percorso vuoto significa che la vista è caricata di default. ":" nel percorso indica il parametro dinamico passato al componente caricato.

Le rotte sono rese disponibili per l'applicazione tramite l'iniezione dipendente. Il metodo ProviderRouter viene chiamato con RouterConfig come parametro, in modo che possa essere iniettato ai componenti per chiamare le attività specifiche del routing.

```
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)];
```

Il terzo passaggio consiste nel riavviare il provider di route.

Nel tuo `main.ts` (può essere un nome qualsiasi, in pratica, dovrebbe il tuo file principale definito in 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));
```

Il quarto passaggio consiste nel caricare / visualizzare i componenti del router in base al percorso cui si accede. la direttiva è usata per dire angolare dove caricare il componente. Per utilizzare importare 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]
})
```

Il quinto passo è quello di collegare gli altri percorsi. Per impostazione predefinita, RouterOutlet caricherà il componente per il quale il percorso vuoto è specificato in RouterConfig. La direttiva RouterLink viene utilizzata con il tag di ancoraggio html per caricare i componenti collegati alle rotte. RouterLink genera l'attributo href che viene utilizzato per generare collegamenti. Per esempio:

```
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 { }
```

Ora, siamo bravi con il routing al percorso statico. RouterLink supporta il percorso dinamico anche passando parametri aggiuntivi insieme al percorso.

importare {Component} da '@ angular / core'; importare {ROUTER_DIRECTIVES} da '@ 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 accetta un array in cui il primo elemento è il percorso per il routing e gli elementi successivi sono per i parametri di routing dinamico.

Percorsi per bambini

A volte ha senso annidare viste o rotte l'una dentro l'altra. Ad esempio sul cruscotto si desiderano diverse viste secondarie, simili alle schede ma implementate tramite il sistema di routing, per mostrare i progetti, i contatti, i messaggi degli utenti. Per supportare tali scenari, il router ci consente di definire percorsi figlio.

Per prima cosa `RouterConfig` nostro `RouterConfig` dall'alto e aggiungiamo le rotte `RouterConfig` :

```
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 }
];
```

Ora che abbiamo definito le rotte del nostro bambino, dobbiamo assicurarci che le rotte dei bambini possano essere visualizzate all'interno del nostro `DashboardComponent` , poiché è lì che abbiamo aggiunto i bambini. In precedenza abbiamo appreso che i componenti sono visualizzati in un `<router-outlet></router-outlet>` . Analogamente dichiariamo un altro `RouterOutlet` in `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 { }
```

Come puoi vedere, abbiamo aggiunto un altro `RouterOutlet` in cui verranno visualizzate le rotte `RouterOutlet` . Di solito viene mostrata la rotta con un percorso vuoto, tuttavia, impostiamo un reindirizzamento sulla rotta dei `projects` , perché vogliamo che vengano mostrati immediatamente quando viene caricato il percorso del `dashboard` . Detto questo, abbiamo bisogno di un percorso vuoto, altrimenti riceverai un errore come questo:

```
Cannot match any routes: 'dashboard'
```

Quindi, aggiungendo il percorso *vuoto* , ovvero un percorso con un percorso vuoto, abbiamo definito un punto di ingresso per il router.

ResolveData

Questo esempio ti mostrerà come puoi risolvere i dati recuperati da un servizio prima di mostrare la vista dell'applicazione.

Utilizza angular / router 3.0.0-beta.2 al momento della scrittura

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());
    }
}
```

users.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

Questo è un componente di pagina con un elenco di tutti gli utenti. `data.users` allo stesso modo per il componente della pagina dei dettagli Utente, sostituirà `data.users` con `data.user` o qualsiasi altra chiave definita in *app.routes.ts* (vedi sotto)

```
...
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

Opzionalmente raggruppa più risolutori insieme.

IMPORTANTE: i *servizi utilizzati in resolver devono essere importati per primi o si otterrà un 'Nessun provider per ... Errore Risolutore'. Ricorda che questi servizi saranno disponibili a livello globale e non dovrai più dichiararli in alcun* `providers` *di componenti. Assicurati di annullare l'iscrizione a qualsiasi abbonamento per evitare perdite di memoria*

```
...
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

export const ROUTE_RESOLVERS = [
    ...,
    UsersService,
    UsersResolver
]
```

main.browser.ts

I resolver devono essere iniettati durante il bootstrap.

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

bootstrap(<Type>App, [
    ...
    ...ROUTE_RESOLVERS
])
.catch(err => console.error(err));
```

Instradamento con i bambini

Contrariamente alla documentazione originale, ho trovato questo modo di annidare correttamente le rotte dei bambini all'interno del file app.routing.ts o app.module.ts (a seconda delle preferenze). Questo approccio funziona quando si utilizza WebPack o SystemJS.

L'esempio seguente mostra i percorsi per casa, casa / contatore e home / contatore / fetch-data. Il primo e l'ultimo instradamento sono esempi di reindirizzamenti. Alla fine dell'esempio, infine, c'è un modo corretto per esportare il percorso da importare in un file separato. Per es. app.module.ts

Per spiegare ulteriormente, Angular richiede di avere un percorso senza percorsi nell'array per bambini che include il componente principale, per rappresentare la rotta principale. È un po 'confuso, ma se si pensa a un URL vuoto per un percorso figlio, equivarrebbe sostanzialmente allo stesso URL del percorso principale.

```
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 { }
```

Ottimo esempio e descrizione tramite Siraj

Leggi Routing online: https://riptutorial.com/it/angular2/topic/2334/routing

Capitolo 60: Routing (3.0.0+)

Osservazioni

Ci sono alcuni altri trucchi che possiamo fare con il router (come limitare l'accesso), ma questi possono essere trattati in un tutorial separato.

Se hai bisogno di un nuovo percorso, modifica semplicemente `app.routes.ts` e segui i seguenti passi:

1. Importa il componente
2. Aggiungi all'array delle `routes` . Assicurati di includere un nuovo `path` e `component` .

Examples

bootstrapping

Ora che i percorsi sono definiti, dobbiamo far conoscere la nostra applicazione sui percorsi. Per fare ciò, avvia il provider che abbiamo esportato nell'esempio precedente.

Trova la tua configurazione di bootstrap (dovrebbe essere in `main.ts` , ma il **tuo chilometraggio può variare**).

```
//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));
```

Configurazione del router-outlet

Ora che il router è configurato e la nostra app sa come gestire i percorsi, dobbiamo mostrare i componenti reali che abbiamo configurato.

Per fare ciò, configura il tuo modello / file HTML per il tuo componente di **livello superiore (app) in questo** modo:

```
//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>
```

L'elemento `<router-outlet></router-outlet>` cambierà il contenuto del percorso. Un altro aspetto positivo di questo elemento è che *non* deve essere l'unico elemento nel codice HTML.

Ad esempio: diciamo che volevi una barra degli strumenti in ogni pagina che rimane costante tra i percorsi, in modo simile a come appare Overflow dello stack. È possibile nidificare `<router-outlet>` sotto gli elementi in modo che cambino solo alcune parti della pagina.

Modifica dei percorsi (utilizzando modelli e direttive)

Ora che i percorsi sono impostati, abbiamo bisogno di un modo per cambiare effettivamente le rotte.

Questo esempio mostra come modificare le rotte usando il modello, ma è possibile modificare le rotte in TypeScript.

Ecco un esempio (senza binding):

```
<a routerLink="/home">Home</a>
```

Se l'utente fa clic su quel collegamento, si dirigerà verso `/home` . Il router sa come gestire `/home` , quindi mostrerà il componente `Home` .

Ecco un esempio con associazione dati:

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

Il che richiederebbe un array chiamato `links` esistere, quindi aggiungilo a `app.ts` :

```
public links[] = [
    'home',
    'login'
]
```

Questo eseguirà un ciclo attraverso la matrice e aggiungerà un elemento `<a>` con la direttiva `routerLink` = il valore dell'elemento corrente nell'array, creando questo:

```
 <a routerLink="home">home</a>
 <a routerLink="login">login</a>
```

Questo è particolarmente utile se hai molti link, o forse i link devono essere costantemente modificati. Lasciamo che Angular gestisca il lavoro intenso di aggiungere collegamenti semplicemente fornendogli le informazioni necessarie.

Al momento, i `links[]` sono statici, ma è possibile alimentarli da un'altra fonte.

Impostazione delle rotte

NOTA: questo esempio si basa sulla versione 3.0.0.-beta.2 di @ angular / router. Al momento della stesura, questa è l'ultima versione del router.

Per utilizzare il router, definire le rotte in un nuovo file TypeScript come tale

```
//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);
```

Nella prima riga, importiamo `provideRouter` modo che possiamo consentire alla nostra applicazione di sapere quali sono i percorsi durante la fase di bootstrap.

`Home` e il `Profile` sono solo due componenti come esempio. Dovrai importare ogni `Component` hai bisogno come percorso.

Quindi, esportare la serie di percorsi.

`path` : il percorso del componente. **NON HA BISOGNO DI UTILIZZARE '/ .....'** Angolare lo farà automaticamente

`component` : il componente da caricare quando si accede al percorso

`redirectTo` : *facoltativo* . Se è necessario reindirizzare automaticamente un utente quando accede a un percorso particolare, fornire questo.

Infine, esportiamo il router configurato. `provideRouter` restituirà un provider che possiamo boostrap in modo che la nostra applicazione sappia come gestire ogni percorso.

Controllo dell'accesso ao da una rotta

Il router angolare predefinito consente la navigazione da e verso qualsiasi percorso in modo incondizionato. Questo non è sempre il comportamento desiderato.

In uno scenario in cui un utente può essere autorizzato a navigare da o verso una rotta, è possibile utilizzare una **protezione di percorso** per limitare questo comportamento.

Se il tuo scenario si adatta a uno dei seguenti, prendi in considerazione l'utilizzo di un Route Guard,

- L'utente deve essere autenticato per navigare verso il componente di destinazione.
- L'utente deve essere autorizzato a navigare verso il componente di destinazione.
- Il componente richiede una richiesta asincrona prima dell'inizializzazione.
- Il componente richiede l'input dell'utente prima di essere allontanato.

Come funzionano le Guardie della Route

Le guardie del percorso funzionano restituendo un valore booleano per controllare il comportamento della navigazione del router. Se viene restituito *true* , il router continuerà con la navigazione verso il componente di destinazione. Se viene restituito *false* , il router negherà la navigazione al componente di destinazione.

Route Guard Interfaces

Il router supporta più interfacce di protezione:

- *CanActivate* : si verifica tra la navigazione del percorso.
- *CanActivateChild* : si verifica tra la navigazione del percorso e un percorso *secondario* .
- *CanDeactivate* : si verifica quando si naviga lontano dal percorso corrente.
- *CanLoad* : si verifica tra la navigazione del percorso e un modulo funzione caricato in modo asincrono.
- *Risolvi* : utilizzato per eseguire il recupero dei dati prima dell'attivazione del percorso.

Queste interfacce possono essere implementate in guardia per concedere o rimuovere l'accesso a determinati processi di navigazione.

Protezioni di percorso sincrone contro asincrone

Route Guards consente operazioni sincrone e asincrone per controllare in modo condizionale la

navigazione.

Protezione di percorso sincrona

Una protezione di percorso sincrona restituisce un valore booleano, ad esempio calcolando un risultato immediato, al fine di controllare in modo condizionale la navigazione.

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

@Injectable()
export class SynchronousGuard implements CanActivate {
  canActivate() {
    console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

Protezione di percorso asincrona

Per un comportamento più complesso, una protezione di percorso può bloccare in modo asincrono la navigazione. Una protezione percorso asincrona può restituire un Osservabile o una Promessa.

Ciò è utile per situazioni come aspettare che l'input dell'utente risponda a una domanda, in attesa di salvare con successo le modifiche sul server o in attesa di ricevere i dati recuperati da un server 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;
        });
    }
}
```

Aggiungi guardia per instradare la configurazione

File *app.routes*

Le rotte protette possono `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] }
}
```

Esportare **APP_ROUTER_PROVIDERS** da utilizzare nel bootstrap dell'app

```
export const APP_ROUTER_PROVIDERS = [
    AuthGuard,
    provideRouter(routes)
];
```

Usa Guard nel bootstrap dell'app

File *main.ts* (o *boot.ts*)

Considera gli esempi sopra:

1. **Crea la guardia** (dove viene creata la Guardia) e
2. **Aggiungi guardia per instradare la configurazione** , (dove Guard è configurato per il percorso, quindi viene esportato **APP_ROUTER_PROVIDERS**),
   possiamo accoppiare il bootstrap a Guard come segue

```
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));
```

Usando Resolver e Guardie

Stiamo utilizzando una protezione di primo livello nella nostra configurazione di instradamento per catturare l'utente corrente sul caricamento della prima pagina e un resolver per memorizzare il valore di `currentUser` , che è il nostro utente autenticato dal back-end.

Una versione semplificata della nostra implementazione è la seguente:

Ecco la nostra rotta di primo livello:

```
export const routes = [
{
  path: 'Dash',
  pathMatch : 'prefix',
  component: DashCmp,
  canActivate: [AuthGuard],
  resolve: {
      currentUser: CurrentUserResolver
  },
  children: [...[
    path: '',
    component: ProfileCmp,
    resolve: {
        currentUser: currentUser
    }
  ]]
  }
];
```

Ecco il nostro `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);
 }
}
```

Ecco il nostro `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);
  }
}
```

Ecco il nostro `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;
  }
}
```

Leggi Routing (3.0.0+) online: https://riptutorial.com/it/angular2/topic/1208/routing--3-0-0plus-

Capitolo 61: Servizi e iniezione delle dipendenze

Examples

Servizio di esempio

servizi / my.service.ts

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

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

La registrazione del fornitore di servizi nel metodo bootstrap renderà il servizio disponibile a livello globale.

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

Nella versione RC5 la registrazione del fornitore di servizi globali può essere effettuata all'interno del file del modulo. Per ottenere una singola istanza del servizio per l'intera applicazione, il servizio deve essere dichiarato nell'elenco dei provider nel modulo ng dell'applicazione.
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 {}
```

Utilizzo in `MyComponent`

componenti / my.component.ts

Approccio alternativo per registrare i fornitori di applicazioni nei componenti dell'applicazione. Se aggiungiamo i provider a livello di componente ogni volta che il componente viene reso, creerà una nuova istanza del servizio.

```
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();
    }
}
```

Esempio con Promise.resolve

servizi / my.service.ts

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

@Injectable()
export class MyService {
    data: any = [1, 2, 3];

    getData() {
        return Promise.resolve(this.data);
    }
}
```

`getData()` ora agisce come una chiamata REST che crea una Promessa, che viene immediatamente risolta. I risultati possono essere `.then()` all'interno di `.then()` e possono essere rilevati anche errori. Questa è una buona pratica e una convenzione per i metodi asincroni.

componenti / 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);
    }
}
```

Test di un servizio

Dato un servizio che può accedere ad un utente:

```
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();
  }
}
```

Può essere testato in questo modo:

```
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();
    }
  });
});
```

Leggi Servizi e iniezione delle dipendenze online:
https://riptutorial.com/it/angular2/topic/4187/servizi-e-iniezione-delle-dipendenze

Capitolo 62: Soggetti e osservabili angolari RXJS con richieste API

Osservazioni

Fare richieste API con il servizio Http 2 di Angular e RxJS è molto simile a lavorare con le promesse in 1.x angolare.

Utilizzare la classe Http per effettuare richieste. La classe Http espone i metodi per l'emissione delle richieste HTTP `GET` , `POST` , `PUT` , `DELETE` , `PATCH` , `HEAD` richieste tramite i metodi corrispondenti. Espone anche un metodo di `request` generico per l'emissione di qualsiasi tipo di richiesta HTTP.

Tutti i metodi della classe `Http` restituiscono una `Observable<Response>` , a cui è possibile applicare le operazioni RxJS . Si chiama il metodo `.subscribe()` e si passa una funzione da chiamare quando i dati vengono restituiti nel flusso Observable.

Il flusso Observable per una richiesta contiene solo un valore - la `Response` , e completa / stabilisce quando la richiesta HTTP è completata con successo, o errori / errori se viene generato un errore.

Nota: gli osservabili restituiti dal modulo `Http` sono *freddi* , il che significa che se ti iscrivi più volte all'osservabile, la richiesta di origine verrà *eseguita* una volta per ogni sottoscrizione. Questo può accadere se si desidera consumare il risultato in più componenti della propria applicazione. Per le richieste GET questo potrebbe solo causare alcune richieste extra, ma questo può creare risultati imprevisti se si iscrive più di una volta a richieste PUT o POST.

Examples

Richiesta di base

L'esempio seguente mostra una semplice richiesta HTTP GET. `http.get()` restituisce un `Observable` che ha il metodo `subscribe` . Questo accoda i dati restituiti alla matrice dei `posts` .

```
var posts = []

getPosts(http: Http):void {
    this.http.get(`https://jsonplaceholder.typicode.com/posts`)
        .map(response => response.json())
        .subscribe(post => posts.push(post));
}
```

Incapsulamento delle richieste API

Potrebbe essere una buona idea incapsulare la logica di gestione HTTP nella sua stessa classe. La seguente classe espone un metodo per ottenere i post. Chiama il metodo `http.get()` e chiama `.map` sul `Observable` restituito per convertire l'oggetto `Response` in un oggetto `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)
      });
  }
}
```

L'esempio precedente utilizza una classe `Post` per contenere i dati restituiti, che potrebbero avere il seguente aspetto:

```
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 ora può utilizzare la classe `BlogApi` per recuperare facilmente i dati del `Post` senza preoccuparsi dei meccanismi della classe `Http` .

Attendi più richieste

Uno scenario comune è quello di attendere il completamento di un certo numero di richieste prima di continuare. Questo può essere realizzato usando il metodo `forkJoin` .

Nell'esempio seguente, `forkJoin` viene utilizzato per chiamare due metodi che restituiscono `Observables` . Il callback specificato nel metodo `.subscribe` verrà chiamato quando entrambi gli osservabili sono completi. I parametri forniti da `.subscribe` corrispondono all'ordine dato nella chiamata a `.forkJoin` . In questo caso, i primi `posts` poi `tags` .

```
loadData() : void {
    Observable.forkJoin(
        this.blogApi.getPosts(),
        this.blogApi.getTags()
    ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
        this.posts = posts;
        this.tags = tags;
```

```
    }));
}
```

Leggi Soggetti e osservabili angolari RXJS con richieste API online:
https://riptutorial.com/it/angular2/topic/3577/soggetti-e-osservabili-angolari-rxjs-con-richieste-api

Capitolo 63: Test di ngModel

introduzione

È un esempio di come è possibile testare un componente in Angular2 con un ngModel.

Examples

Test di base

```
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");
        });
    });
});
```

Leggi Test di ngModel online: https://riptutorial.com/it/angular2/topic/8693/test-di-ngmodel

Capitolo 64: Test di un'app Angular 2

Examples

Installazione del framework di test Jasmine

Il modo più comune per testare le app Angular 2 è con il framework di test Jasmine. Jasmine ti permette di testare il tuo codice nel browser.

Installare

Per iniziare, tutto ciò che serve è il pacchetto `jasmine-core` (non `jasmine`).

```
npm install jasmine-core --save-dev --save-exact
```

Verificare

Per verificare che Jasmine sia configurato correttamente, crea il file `./src/unit-tests.html` con il seguente contenuto e aprilo nel browser.

```
<!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>
```

Impostazione dei test con Gulp, Webpack, Karma e Jasmine

La prima cosa di cui abbiamo bisogno è di dire al karma di usare Webpack per leggere i nostri test, in una configurazione che abbiamo impostato per il motore del webpack. Qui, sto usando Babele perché scrivo il mio codice in ES6, puoi cambiarlo per altri gusti, come Typescript. O io uso i modelli Pug (precedentemente Jade), non è necessario.

Tuttavia, la strategia rimane la stessa.

Quindi, questa è una configurazione del 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;
```

E poi, abbiamo bisogno di un file karma.config.js per usare quella configurazione di 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
});
};
```

Finora, abbiamo detto a Karma di usare il webpack e gli abbiamo detto di iniziare da un file chiamato **karma.shim.js** . questo file avrà il compito di fungere da punto di partenza per il webpack. webpack leggerà questo file e userà le istruzioni di **importazione** e **richiesta** per raccogliere tutte le nostre dipendenze ed eseguire i nostri test.

Quindi ora, diamo un'occhiata al file 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);
```

In sostanza, stiamo importando **TestBed** dal test del core angolare e avviando l'ambiente, poiché deve essere avviato solo una volta per tutti i nostri test. Quindi, stiamo **esaminando la** directory **src / app in** modo ricorsivo e leggendo tutti i file che terminano con **.spec.js** e li inoltriamo a testContext, quindi verranno eseguiti.

Di solito cerco di mettere i miei test nello stesso posto della classe. Sapore personale, mi rende più facile importare dipendenze e refactoring con le classi. Ma se vuoi mettere i tuoi test da qualche altra parte, ad esempio sotto la directory **src / test,** ad esempio, ecco la possibilità. cambia la riga prima dell'ultima nel file karma.shim.js.

Perfezionare. cos'è rimasto? ah, il compito di gulp che usa il file karma.config.js che abbiamo fatto sopra:

```
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();
});
```

Ora sto avviando il server con il file di configurazione che abbiamo creato, dicendogli di eseguirlo una volta e di non controllare le modifiche. Trovo questo per me meglio come i test verranno eseguiti solo se sono pronto per loro di correre, ma ovviamente se vuoi che tu sappia dove cambiare.

E come esempio finale del codice, ecco una serie di test per il tutorial di 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");
        })
```

```
    ));
});
```

Degno di nota in questo è il modo in cui abbiamo **primaOach ()** configurare un modulo di test e creare il componente in test, e come chiamiamo **detectChanges () in** modo che l'angolare attraversi effettivamente il doppio legame e tutto.

Si noti che ogni test è una chiamata a **async ()** e attende sempre che **whenStable** prometta di risolvere prima di esaminare l'apparecchiatura. Quindi ha accesso al componente attraverso **componentInstance** e all'elemento tramite **nativeElement** .

C'è un test che sta controllando lo stile corretto. come parte del Tutorial, il team di Angular dimostra l'uso degli stili all'interno dei componenti. Nel nostro test, usiamo **getComputedStyle ()** per verificare che gli stili vengano da dove abbiamo specificato, tuttavia abbiamo bisogno dell'oggetto Window per quello, e lo stiamo ottenendo dall'elemento come potete vedere nel test.

Test del servizio Http

Solitamente, i servizi chiamano Api remoti per recuperare / inviare dati. Ma i test unitari non dovrebbero effettuare chiamate di rete. Angular utilizza `XHRBackend` classe `XHRBackend` per fare richieste http. L'utente può sovrascriverlo per modificare il comportamento. Il modulo di test angolare fornisce classi `MockBackend` e `MockConnection` che possono essere utilizzate per testare e far valere richieste http.

`posts.service.ts` Questo servizio raggiunge un endpoint api per recuperare l'elenco dei post.

```
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` Qui, `posts.service.spec.ts` servizio di cui sopra `posts.service.spec.ts` giro le chiamate http api.

```
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);
                    });

            })));
});
```

Test di componenti angolari - Di base

Il codice componente è dato come di seguito.

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

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent{
  title = 'welcome';
}
```

Per il test angolare, l'angolare fornisce le sue utilità di test insieme al framework di test che aiuta a scrivere il caso di test positivo in angolare. Le utility angolari possono essere importate da `@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();

  });

});
```

Nell'esempio sopra, c'è solo un caso di test che spiega il caso di test per l'esistenza del componente. Nell'esempio sopra `TestBed` vengono utilizzate utility di test angolari come `TestBed` e `ComponentFixture` .

`TestBed` viene utilizzato per creare il modulo di test angolare e configuriamo questo modulo con il metodo `configureTestingModule` per produrre l'ambiente del modulo per la classe che vogliamo testare. Modulo di test da configurare prima dell'esecuzione di ogni caso di test è per questo che configuriamo il modulo di test nella funzione `beforeEach` .

`createComponent` metodo `createComponent` di `TestBed` viene utilizzato per creare l'istanza del componente in prova. `createComponent` restituisce `ComponentFixture` . L'apparecchiatura fornisce l'accesso all'istanza del componente stessa.

Leggi Test di un'app Angular 2 online: https://riptutorial.com/it/angular2/topic/2329/test-di-un-app-angular-2

Capitolo 65: test unitario

Examples

Test unitario di base

file 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';
    }

}
```

File di prova

```
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');
  });
});
```

Leggi test unitario online: https://riptutorial.com/it/angular2/topic/8955/test-unitario

Capitolo 66: Titolo della pagina

introduzione

Come puoi cambiare il titolo della pagina

Sintassi

- `setTitle(newTitle: string): void;`
- `getTitle(): string;`

Examples

cambiare il titolo della pagina

1. Per prima cosa dobbiamo fornire il servizio Title.
2. Utilizzo di 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');
   }
}
```

Leggi Titolo della pagina online: https://riptutorial.com/it/angular2/topic/8954/titolo-della-pagina

Capitolo 67: Utilizzando librerie di terze parti come jQuery in Angular 2

introduzione

Quando si creano applicazioni utilizzando Angular 2.x, è possibile che vengano utilizzate librerie di terze parti come jQuery, Google Analytics, API JavaScript di integrazione della chat e così via.

Examples

Configurazione usando angular-cli

NPM

Se la libreria esterna come `jQuery` viene installata usando NPM

```
npm install --save jquery
```

Aggiungi il percorso di script nel tuo `angular-cli.json`

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

Cartella delle risorse

È anche possibile salvare il file della libreria nella directory `assets/js` e includere lo stesso in `angular-cli.json`

```
"scripts": [
    "assets/js/jquery.js"
]
```

Nota

Salvate la `jquery` libreria principale e le relative dipendenze come `jquery-cycle-plugin` nella directory assets e aggiungetele entrambe in `angular-cli.json` , assicuratevi che l'ordine sia mantenuto per le dipendenze.

Utilizzo di jQuery nei componenti Angular 2.x.

Per utilizzare `jquery` nei componenti Angular 2.x, dichiarare una variabile globale in alto

Se si usa `$` per jQuery

```
declare var $: any;
```

Se si utilizza `jQuery` per jQuery

```
declare var jQuery: any
```

Ciò consentirà l'utilizzo di `$` o `jQuery` nel componente Angular 2.x.

Leggi Utilizzando librerie di terze parti come jQuery in Angular 2 online: https://riptutorial.com/it/angular2/topic/9285/utilizzando-librerie-di-terze-parti-come-jquery-in-angular-2

Capitolo 68: Utilizzare i webcomponents nativi in Angular 2

Osservazioni

Quando si utilizza un componente Web nel modello Angular 2, angular proverà a trovare un componente con un selettore corrispondente al tag personalizzato del componente Web, che ovviamente non può e genererà un errore.

La soluzione è di importare uno "schema di elementi personalizzato" nel modulo che contiene il componente. Questo renderà angolare accettare qualsiasi tag personalizzato, che non corrisponde ad alcun selettore di componenti ng.

Examples

Includi schema di elementi personalizzati nel modulo

```
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 il tuo webcomponent in un template

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

@Component({
  selector: 'myapp-about',
  template: `<my-webcomponent></my-webcomponent>`
})
export class AboutComponent { }
```

Leggi Utilizzare i webcomponents nativi in Angular 2 online:
https://riptutorial.com/it/angular2/topic/7414/utilizzare-i-webcomponents-nativi-in---angular-2

Capitolo 69: Validazioni personalizzate Angular2

Parametri

parametro	descrizione
controllo	Questo è il controllo che viene validato. In genere vorrete vedere se control.value soddisfa alcuni criteri.

Examples

Esempi di validatori personalizzati:

Angular 2 ha due tipi di validatori personalizzati. Validatori sincroni come nel primo esempio che verrà eseguito direttamente sul client e sui validatori asincroni (il secondo esempio) che è possibile utilizzare per chiamare un servizio remoto per eseguire la convalida. In questo esempio il validatore dovrebbe chiamare il server per vedere se un valore è univoco.

```
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);
    });
}}
```

Se il tuo valore di controllo è valido, devi semplicemente restituire null al chiamante. Altrimenti puoi restituire un oggetto che descrive l'errore.

Utilizzo dei validatori nel Formbuilder

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

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

Qui usiamo FormBuilder per creare un modulo molto semplice con due caselle di input. FromBuilder acquisisce un array per tre argomenti per ciascun controllo di input.

1. Il valore predefinito del controllo.
2. I validatori che verranno eseguiti sul client. È possibile utilizzare Validators.compose ([arrayOfValidators]) per applicare più validatori sul proprio controllo.
3. Uno o più validatori asincroni in modo simile al secondo argomento.

get / set formBuilder controlla i parametri

Ci sono 2 modi per impostare i parametri dei controlli formBuilder.

1. All'inizializzazione:

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

2. Dopo l'inizializzazione:

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

Ottieni il valore di controllo del formBuilder:

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

Leggi Validazioni personalizzate Angular2 online:
https://riptutorial.com/it/angular2/topic/6284/validazioni-personalizzate-angular2

Capitolo 70: Zone.js

Examples

Ottenere riferimenti a NgZone

`NgZone` riferimento `NgZone` può essere iniettato tramite l'Iniezione di dipendenza (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
    });
  }
}
```

Utilizzo di NgZone per eseguire più richieste HTTP prima di mostrare i dati

`runOutsideAngular` può essere utilizzato per eseguire il codice all'esterno di Angular 2 in modo che non `runOutsideAngular` rilevamento di modifiche inutilmente. Questo può essere usato per eseguire più richieste HTTP per ottenere tutti i dati prima di renderli. Per eseguire nuovamente il codice all'interno di Angular 2, è possibile `run` metodo di `NgZone` di `NgZone` .

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];
            });
          });
        });
      });
    });
```

```
    }
}
```

Leggi Zone.js online: https://riptutorial.com/it/angular2/topic/4184/zone-js

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare 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	Aggiorna i tipi	kEpEx
3	Aggiornamento di moduli angolari 2	Amit kumar, Anil Singh, Christopher Taylor, Highmastdon, Johan Van de Merwe, K3v1n, Manmeet Gill, mayur, Norsk, Sachin S, victoroniibukun, vijaykumar, Yoav Schniederman
4	Aggiungi componenti dinamicamente usando ViewContainerRef.createComponent	amansoni211, daniellmb, Günter Zöchbauer, jupiter24, Khaled
5	Angolare - ForLoop	aholtry, Anil Singh, Berseker59, gerl, Johan Van de Merwe, ob1, Pujan Srivastava, Stephen Leppik, Yoav Schniederman
6	Angolare 2 - Goniometro	Yoav Schniederman
7	Angolare 2 Rilevamento delle modifiche e attivazione manuale	Yoav Schniederman
8	Angular2 CanActivate	Companjo, Yoav Schniederman
9	Angular2 fornisce dati esterni all'App prima del bootstrap	Ajey
10	Angular2 Input () output ()	Kaloyan, Yoav Schniederman
11	Angular2 utilizza il webpack	luukgruijs
12	Angular-cli	BogdanC, Yoav Schniederman

13	Animazione	Gaurav Mukherjee, Nate May
14	Animazioni Angular2	Yoav Schniederman
15	API Web Angular2 in memoria	Jaime Still
16	barile	TechJhola
17	Bootstrap Modulo vuoto in angolare 2	AryanJ-NYC, autoboxer, Berseker59, Eric Jimenez, Krishan, Sanket, snorkpete
18	Brute Force Upgrading	Jim, Treveshan Naidoo
19	Bypassare la disinfezione per valori attendibili	Scrambo
20	Come usare ngfor	Jorge, Yoav Schniederman
21	Come usare ngif	Amit kumar, ob1, ppovoski, samAlvin
22	Compilazione A prima del tempo (AOT) con Angular 2	Anil Singh, Eric Jimenez, Harry, Robin Dijkhof
23	componenti	BrunoLM
24	Configurazione dell'applicazione ASP.net Core per lavorare con Angular 2 e TypeScript	Oleksii Aza, Sam
25	copertura del test angular-cli	ahmadalibaloch
26	Crea un pacchetto Angular 2+ NPM	BogdanC, Janco Boscan, vinagreti
27	Creazione di una libreria Angular npm	Maciej Treder
28	CRUD in Angular2 con Restful API	bleakgadfly, Sefa
29	Databinding angolare2	Yoav Schniederman
30	Debug di un'applicazione dattiloscritto Angular2 utilizzando il codice di Visual Studio	PSabuwala
31	Design materiale angolare	Ketan Akbari, Shailesh Ladumor
32	direttive	acdcjunior, Andrei Zhytkevich, borislemke, BrunoLM, daniellmb, Everettss, lexith, Stian Standahl, theblindprophet
33	Direttive e componenti: @Input	acdcjunior, dafyddPrys, Everettss, Joel Almeida,

	@Output	lexith, muetzerich, theblindprophet, ThomasP1988
34	Direttive e servizi comunemente incorporati	Jim, Sanket
35	Direttive sugli attributi per influenzare il valore delle proprietà sul nodo host utilizzando il decoratore @HostBinding.	Max Karpovets
36	Dropzone in Angular2	Ketan Akbari
37	Esempi di componenti avanzati	borislemke, smnbbrv
38	Esempio di percorsi come / route / subroute per gli URL statici	Yoav Schniederman
39	EventEmitter Service	Abrar Jahin
40	Forme basate su dati angolari 2	MatWaligora, ThunderRoid
41	Http Interceptor	Everettss, Mihai, Mike Kovetsky, Nilz11, Paul Marshall, peeskillet, theblindprophet
42	Installazione di plugin di terze parti con angular-cli@1.0.0-beta.10	Alex Morales, Daredzik, filoxo, Kaspars Bergs, pd farhad
43	Interazioni componenti	H. Pauwelyn, Janco Boscan, LLL, Sefa
44	Lifecycle Hooks	Alexandre Junges, daniellmb, Deen John, muetzerich, Sbats, theblindprophet
45	Mocking @ ngrx / Store	BrianRT, Hatem, Jim, Lucas, Yoav Schniederman
46	Modelli	Max Karpovets
47	moduli	BrunoLM
48	Moduli funzione	AryanJ-NYC, gsc
49	NGRX	Maxime
50	Operaio di servizio	Roberto Fernandez
51	Ordinare per tubo	Yoav Schniederman
52	Ottimizzazione del rendering con ChangeDetectionStrategy	daniellmb, Eric Jimenez, Everettss

53	personalizzato ngx-bootstrap datepicker + input	Yoav Schniederman
54	Pigro che carica un modulo	Batajus, M4R1KU, Shannon Young, Syam Pradeep
55	Pipes	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
56	redox angolare	Yoav Schniederman
57	Rilevazione degli eventi di ridimensionamento	Eric Jimenez
58	Routing	aholtry, Apmis, AryanJ-NYC, borislemke, camwhite, Kaspars Bergs, LordTribual, Sachin S , theblindprophet
59	Routing (3.0.0+)	Ai_boy, Alexis Le Gal, Everettss, Gerard Simpson, Kaspars Bergs, mast3rd3mon, meorfi, rivanov, SlashTag, smnbbrv, theblindprophet, ThomasP1988, Trent
60	Servizi e iniezione delle dipendenze	BrunoLM, Eduardo Carísio, Kaspars Bergs, Matrim, Roope Hakulinen, Syam Pradeep, theblindprophet
61	Soggetti e osservabili angolari RXJS con richieste API	daniellmb, Maciej Treder, Ronald Zarīts, Sam Storie, Sébastien Temprado, willydee
62	Test di ngModel	jesussegado
63	Test di un'app Angular 2	Arun Redhu, michaelbahr, nick, Reza, Rumit Parakhiya
64	test unitario	Yoav Schniederman
65	Titolo della pagina	Yoav Schniederman
66	Utilizzando librerie di terze parti come jQuery in Angular 2	Ashok Vishwakarma
67	Utilizzare i webcomponents nativi in Angular 2	ugreen
68	Validazioni personalizzate Angular2	Arnold Wiersma, Norsk, Yoav Schniederman

69	Zone.js	Roope Hakulinen