Angular2 - root relative imports

32

7

I have a problem with imports in angular2/typescript. I'd like to use imports with some root like 'app/components/calendar', instead only way I am able to use is something like:

```
//app/views/order/order-view.ts
import {Calendar} from '../../components/calendar
```

where Calendar is defined like:

```
//app/components/calendar.ts
export class Calendar {
}
```

and this obviously gets much worse the lower in hierarchy you go, deepest is '../../..' but it is still very bad and brittle. Is there any way how to use paths relative to project root?

I am working in Visual Studio, and relative imports seem to be the only thing that makes VS able to recognize these imports.y

typescript angular

asked Jan 5 '16 at 15:04
Jarek
3,206 13 50 84

`"baseUrl"` maybe should set `"./src"` and `paths relate to it` will get it to work, it origin to be `"./"` , but my is not work, I don't know why! and then add `"paths": { "@app/*": ["app/*"] }` to `tsconfig.json` and use like `import { PageNotFoundComponent } from '@app/error-page/page-not-found.component';` – Cheney Apr 27 at 15:10

15

Set the compilerOption `"moduleResolution"` to `"classic"` .

Long answer

Are you using `tsconfig.json` ? I assume you are. I've been looking a way to make statements such as `import someModule = require("../../../../someModule"` into `import someModule=require("src/path/to/someModule")` . I found after hours wasted that tsc may use different algorithms for module resolution. I'm using atom and it creates the `tsconfig.json` with the compilerOption property `moduleResolution` set to `"node"` and it uses the shitty (excuse my french) module resolution algorithm of node. I just put "classic" and started working the obvious way.

** UPDATE **

Just don't use this solution. Embrace node module resolution algorithm. Community rests on it, so everything will break apart if you try to do otherwise. Use aliases or some of the other provided solutions.

edited Dec 20 '17 at 9:52

answered Jan 12 '16 at 0:40
caeus
851 11 23

2 "classic" option completely breaks angular2, do you know any way how could this be fixed? – Jarek Jan 12 '16 at 16:57

1 have your source files be in a folder named "node_modules". It is explained in this answer: stackoverflow.com/a/24461606/2142728 – caeus Jan 14 '16 at 2:53

put all your code inside a folder called "node_modules"... that's the other option – caeus Mar 31 '16 at 15:31

32 thats the most possible worst option i have ever heard of @caeus – nottinhill Jul 9 '16 at 11:35

4 Do any of the 19 people that have so far upvoted @nottinhill have a better solution? – Neutrino Sep 30 '17 at 12:26

Answer

22

As of TypeScript 2.0 you can set tsconfig.json properties `baseUrl` as following:

Then, you might use your desired manner of importing components, like:

```
import { Calendar } from 'components/calendar';
```

Appendix

Fallback paths resolution

An important consideration is that specifying `baseUrl` option causes TypeScript compilator to:

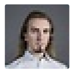
1. Look up a path with regards to baseUrl
2. On failed resolution (module not found), look up a path with regards to `moduleResolution` option

SystemJS

Since SystemJS is heavily used in Angular development stage, be sure to accordingly config also `systemjs.config.js` , so that it resolves paths correctly.

Source and further details: https://github.com/Microsoft/TypeScript/issues/5039

answered Feb 17 '17 at 20:30



Marek Dulowski
1,206 5 16

Wow, it works! +100 for @Marek Dulowski! I may have tried this last time I researched it, and didn't see it working, possibly because: [Note] you'll have to restart the server if you change config while it's running. – BBaysinger Dec 19 '17 at 19:24

Another reason I may have not had success with this months prior was that my tsconfig.app.json was overriding this property in my root tsconfig.json. Just figured this out for one of my older projects. Just delete the property. – BBaysinger Feb 25 '18 at 3:15

@Jarek This should be marked as the correct answer. – BBaysinger Jun 27 '18 at 18:29

1 I actually found this is NOT the ideal solution after I was having problems with it. I posted my new preferred solution here: stackoverflow.com/a/51599367/1253298 Please give it a try. – BBaysinger Jul 30 '18 at 17:32

7

One way would be to have files that re export and bundle the files with a shorter path.

You could have a components.ts folder in the root of your application with.

```
export {Calendar} from './components/calendar'
export {*} from './components/map'
```

And importing it from **components**

```
import {Calendar, Map} from '../../components';
```

This however will is better suited to exporting modules so others can use them, than the way to structure a project.

The alternative would be to forgo the use of import statements and use internal modules instead.

calendar.ts

```
module Components {
    export class Calendar {}
}
```

And you will be able to just use it in any of the files in your projects like this.

```
new Components.Calendar();
```

Or import it with an alias.

```
import Calendar = Components.Calendar;
new Calendar();
```

answered Jan 5 '16 at 16:37

toskv

4

With Angular 2.0, when you run `ng new my-proj`, it auto-creates a file `my-proj/src/tsconfig.app.json`. This file contains a `baseUrl` line:

```
"baseUrl": "./",
```

Thus, you shouldn't have to change anything about your config and you can scope all of your imports rooted on `src`. Assuming your project structure looks something like:

```
my-proj/
  README.md
  src/
    app/
      sidebar/
        sidebar.component.ts
      user.service.ts
```

you can import your files with:

```
import { Sidebar } from 'app/sidebar/sidebar.component';
import { UserService } from 'app/user.service';
```

Edited to add: Sublime does not read `src/tsconfig.json`, though, it only reads the top-level tsconfig.json (see this issue). If you're getting `Cannot find module 'app/...'` errors on your import lines in Sublime, you can add:

```
"baseUrl": "./src",
```

to your top-level tsconfig.json file.

edited May 4 '18 at 18:41

answered Apr 12 '18 at 16:35
kristina
17.5k 8 55 71

Basically, in `Angular 6` you can start the `import` from the **top** or **bottom**

1. From the top
   I prefer this way if it's closer from the root of the app

   ```
   import { DateService } from "src/app/shared-services/context/date.service";
   ```

2. From the bottom

   ```
   import { DateService } from "../../../../../../../shared-services/context/date.service";
   ```

Context:

TypeScript config: **tsconfig.json**

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2017",
      "dom"
    ]
  }
}
```

Angular's stack

```
ng -v
```

```
... http, language-service, platform-browser
... platform-browser-dynamic, router

Package                           Version
-----------------------------------------------------------
@angular-devkit/architect         0.6.8
@angular-devkit/build-angular     0.6.8
@angular-devkit/build-optimizer   0.6.8
@angular-devkit/core              0.6.8

@angular-devkit/schematics        0.6.8
@angular/cli                      6.0.8
@ngtools/webpack                  6.0.8
@schematics/angular               0.6.8
@schematics/update                0.6.8
rxjs                              6.2.1
typescript                        2.7.2
webpack                           4.8.3
```

answered Jul 18 '18 at 14:57
Marian07
858 1 14 31

0

You don't need to change anything from the default configuration of a new Angular project generated from Angular-CLI. Not sure about previous versions, but as of version 6, I know this is true. To do a root-relative import, you use `src/app/` at the beginning of your imports like:

```
import { VideoPlayerComponent } from 'src/app/pages/shared/video-player/video-
player.component';
```

This a better solution than @MarekDulowski's answer, because root-relative imports can be easily identifiable and easily found/searched for by querying `src/app/` .

I learned this from the solution here: https://stackoverflow.com/a/51582699/1253298