1. npm instart 或者npm start 出现connect ETIMEDOUT 104.24.112.177:443  
     
   说明被墙了 开VPN

2.npm start出问题   
第一步： npm update -g && npm install -g concurrently lite-server typescript  
不行的话 第二步：  
Change the start field in package.json from

"start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" "

to

"start": "concurrently \"npm run tsc:w\" \"npm run lite\" "

地址 <http://stackoverflow.com/questions/34335340/angular2-quickstart-npm-start-is-not-working-correctly>

2.1 如果出现 is not a module  
 第一步,检查相关模块export暴露的接口大小写是否规范  
 第二步，退出vs code 重新载入，会给你定位到更具体的错误位置

http://stackoverflow.com/questions/34629517/file-app-hero-ts-is-not-a-module-error-in-the-console-where-to-store-interfac/34674485

2.2 出现 can not find name promise  
 把tsconfig.json中的es5改为ES6

3.官网快速起步

app/app.module.ts 根模块

import { NgModule } from '@angular/core'; //ng代码模块

import { BrowserModule } from '@angular/platform-browser'; //运行在浏览器要导入这个

//导入的模块放在imports数组中

@NgModule({ //装饰器函数

imports: [ BrowserModule ]

})

export class AppModule { }  
  
  
组件模块app/app.component.ts  
组件即组件类，使用 @Component 来装饰一个组件类

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

@Component({ Component为ngmodule暴露的接口

给组件附件的元数据信息，装饰下面的类 AppComponent

selector: 'my-app', 为用来代表该组件的 HTML 元素指定 CSS 选择器

template: '<h1>My First Angular App</h1>'

})

export class AppComponent { } 导出组件供根模块调用

根模块导入刚创建的组件  
import { NgModule } from '@angular/core';

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

import { AppComponent } from './app.component'; 导入模块

@NgModule({

imports: [ BrowserModule ], 导入其他module，imports为应用模块列表

declarations: [ AppComponent ], 声明模块，模块要先导入再声明才能引用

bootstrap: [ AppComponent ] bootstrap引导函数，通知框架将组件渲染到DOM树上，通常是app启动的根组件，一般只有一个component。bootstrap中的组件会自动被放入到entryComponents中

})

export class AppModule { }  
  
创建文件让框架导入创建的根模块  
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

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

const platform = platformBrowserDynamic();

platform.bootstrapModule(AppModule);

4.装饰器 https://angular.cn/docs/ts/latest/guide/glossary.html#!#decorator  
装饰器是一个 **函数**，这个函数将元数据添加到类、类成员 ( 属性、方法 ) 和函数上。

装饰器是一个 JavaScript 的语言 [特性](https://github.com/wycats/javascript-decorators)，装饰器在 TypeScript 里面已经采纳并实现了，并被推荐到了 ES2016( 也就是 ES7) 。

要想应用装饰器，就把装饰器放到被装饰对象的上面或左边。

Angular 使用自己的一套装饰器来实现应用程序各部分之间的相互操作。 下面的例子中使用了 @Component 装饰器来将一个类标记为一个 Angular[组件 (Component)](https://angular.cn/docs/ts/latest/guide/glossary.html#component) ，并用 @Input 装饰器来装饰该组件的 name 属性。 @Component 装饰器中省略的参数对象还可以包含和组件有关的元数据。

1. @Component({...})
2. export class AppComponent {
3. constructor(@Inject('SpecialFoo') public foo:Foo) {}
4. @Input() name:string;
5. }

一个装饰器的作用域会被限制在它所装饰的东西上，这是一个语言级特性。在上面这个例子中，就算别的类在同一个文件中紧跟着上面的类也不会有任何装饰器“泄露”到其它类。

永远别忘了在装饰器后面加括号 () 。

4.教程-英雄编辑器

https://angular.cn/docs/ts/latest/tutorial/toh-pt1.html  
要现在根模块导入模块并添加到imports数组中，组件才能调用  
import { NgModule } from '@angular/core';

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

import { FormsModule } from '@angular/forms';

@NgModule({

imports: [

BrowserModule,

FormsModule

],

declarations: [

AppComponent

],

bootstrap: [ AppComponent ]

})

export class AppModule { }

<li \*ngFor="let hero of heroes">

把数组heroes的值循环赋给变量hero，let本地声明，  
\* 前缀表示 <li> 及其子元素组成了一个主控模板

<span class="badge">{{hero.id}}</span> {{hero.name}}  
class="badge" css选择器为class  
  
(click)="onSelect(hero)" 把组件提供的onSelect方法绑定在click事件上

5.主从结构

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

export class Hero{ /\*创建英雄类 \*/

id:number;

name:string;

}

const HEROES: Hero[] = [ /\*常量 创建由英雄类实例构成的数组 \*/

{ id: 1, name: 'Mr. Nice' },

{ id: 2, name: 'Narco' },

{ id: 3, name: 'Bombasto' },

{ id: 4, name: 'Celeritas' }

];

@Component({ /\*用component对类AppComponent进行装饰，给类添加元数据\*/

selector: 'my-app1', /\*模板名 指定了一个叫 my-app 的元素\*/

template: `

<h1>{{title}}</h1>

<ul class="heroes">

<li \*ngFor="let hero of heroes"

[class.selected]="hero === selectedHero" /\*当两位英雄一致时，它为 true ，否则为 false , 为true的时候，CSS样式选择器为selected， 属性绑定语法\*/

(click)="onSelect(hero)">

<span class="badge">{{hero.id}}</span> {{hero.name}}

</li>

</ul>

<div \*ngIf="selectedHero"> /\*当selectedHero 值为真时\*/

<h2>{{selectedHero.name}} 详情</h2>

<div><label>id: </label>{{selectedHero.id}}</div>

<div>

<label>name: </label>

<input [(ngModel)]="selectedHero.name" />

</div>

</div>

})

export class AppComponent { /\*组件（有指令和模板的类） \*/

title = 'list of heroes';

heroes = HEROES; /\*把顶部创建的数组HEROES赋值给heroes \*/

// hero : Hero ={ /\*实例化类Hero \*/

// id:1,

// name: 'DeMa'

// };

selectedHero:Hero; //要显示点击后的hero属性，所以不再需要固定的hero属性，替换注释部分

onSelect(hero:Hero):void {

this.selectedHero = hero; /\*点击一个英雄的时候，把它赋给 selectedHero 属性 \*/

}

}

6.多个组件

（1）把英雄详情从根组件appcomponent分离出来，放在hero-detail.component.ts文件中  
 （2） 因为hero-detail.component.ts和app.components.ts两个组件都有到Hero类，把这个类也从app.components.ts分离出来，放在hero.ts中。  
（3）在根模块app.Modules.ts中导入并声明hero-detail模块

（4）在app.components.ts中把属性selectedHero和hero-detail暴露的属性hero绑定在一起

具体代码：

hero-detail.component.ts  
import {Component, Input} from '@angular/core';

import {Hero} from './hero'; //导入Hero类

@Component({ //用 @Component 装饰器创建元数据

selector:'my-hero-detail', //模板名或组件名

template : `

<div \*ngIf="hero">

<h2>{{hero.name}} 详情</h2>

<div><label>id: </label>{{hero.id}}</div>

<div>

<label>name: </label>

<input [(ngModel)]="hero.name" />

</div>

</div>

`

})

export class HeroDetailComponent{

@Input() //@input() 会把紧跟其后的属性hero声明成输入属性,声明成输入属性才能被绑定

hero:Hero;

}

hero.ts

export class Hero{ /\*创建英雄类 \*/

id:number;

name:string;

}

app.module.ts

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

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

import {FormsModule} from '@angular/forms';

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

import { HeroDetailComponent } from './hero-detail.component'; //导入进来，以便声明后appcomponent组件能调用其接口

@NgModule({ //用ngmodule修饰类AppModule,@NgModule为顶部导入的ngmodule模块暴露的接口

imports: [ //添加内部模块（全局模块）//把这两个模块加入到imports数组中，作为所以模块内部成员使用

BrowserModule,

FormsModule

],

declarations: [ //声明外部模块（局部模块），声明的模块可以直接使用imports声明的内部模块

AppComponent,

HeroDetailComponent

],

bootstrap: [ AppComponent ] //定义AppComponent为根组件即起始组件，bootstrap为引导函数，通知框架将组件渲染到DOM树上，

})

export class AppModule { } //暴露接口,NgModule提供的export方法

app.component.ts

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

import {Hero} from './hero'; //导入Hero类

const HEROES: Hero[] = [ /\*创建由Hero类实例构成的数组 \*/

{ id: 1, name: 'Mr. Nice' },

{ id: 2, name: 'Narco' },

{ id: 3, name: 'Bombasto' },

{ id: 4, name: 'Celeritas' }

];

@Component({ /\*用component对类AppComponent进行装饰，给类添加元数据\*/

selector: 'my-app1', /\*模板名或组件名 指定了一个叫 my-app 的元素\*/

template: `

<h1>{{title}}</h1>

<ul class="heroes">

<li \*ngFor="let hero of heroes"

[class.selected]="hero === selectedHero"

(click)="onSelect(hero)">

<span class="badge">{{hero.id}}</span> {{hero.name}}

</li>

</ul>

<my-hero-detail [hero]="selectedHero"></my-hero-detail>

`,//导入外部组件my-hero-detail，用my-hero-detail代替主从结构的div，输入属性绑定的前提是目标hero被其所在的组件声明成输入属性

//在根模块appmodule已经导入了my-hero-detail所在的组件

//把 AppComponent 组件的 selectedHero 属性和 HeroDetailComponent 组件的 hero 属性绑定在一起

styles: [`

.selected {

background-color: #CFD8DC !important;

color: white;

}

.heroes {

margin: 0 0 2em 0;

list-style-type: none;

padding: 0;

width: 15em;

}

.heroes li {

cursor: pointer;

position: relative;

left: 0;

background-color: #EEE;

margin: .5em;

padding: .3em 0;

height: 1.6em;

border-radius: 4px;

}

.heroes li.selected:hover {

background-color: #BBD8DC !important;

color: white;

}

.heroes li:hover {

color: #607D8B;

background-color: #DDD;

left: .1em;

}

.heroes .text {

position: relative;

top: -3px;

}

.heroes .badge {

display: inline-block;

font-size: small;

color: white;

padding: 0.8em 0.7em 0 0.7em;

background-color: #607D8B;

line-height: 1em;

position: relative;

left: -1px;

top: -4px;

height: 1.8em;

margin-right: .8em;

border-radius: 4px 0 0 4px;

}

`]

})

export class AppComponent { /\*组件（有指令和模板的类） \*/

//类里的值可以被装饰器绑定在模板中

title = 'list of heroes';

heroes = HEROES; /\*把顶部创建的数组HEROES赋值给heroes \*/

// hero : Hero ={ /\*实例化类Hero \*/

// id:1,

// name: 'DeMa'

// };

selectedHero:Hero; //声明一个名为selectedHero的属性，类型为类Hero,要显示点击后的hero属性，所以不再需要固定的hero属性，替换注释部分

onSelect(hero:Hero):void {

this.selectedHero = hero; /\*把hero包含的值赋给 selectedHero \*/

}

}

7.教程-服务  
 目标:当一个组件中的数据需要被外部组件使用时，为了降低组件间耦合度，可以把这个数据单独分离出来放在一个数据文件中。  
 同时创建一个组件，这个组件专门为别的组件提供数据读写服务，提供数据服务的组件叫做服务。

服务方式是组件间通信的一种

(1把组件 AppComponent中的hero数组剪切出来，放在一个命名为mock-heroes.ts的文件中（即HeroService组件）。因为数据文件只存有数据，要想对数据进行操作，需要组件自身加载方法或者调用提供数据操作服务的组件。

（2）创造一个提供数据操作服务的组件HeroService：

定义HeroService组件的接口名为HeroService，并引入Injectable()双向数据绑定方法。  
 （3）把英雄详情模块分离出来成为一个组件