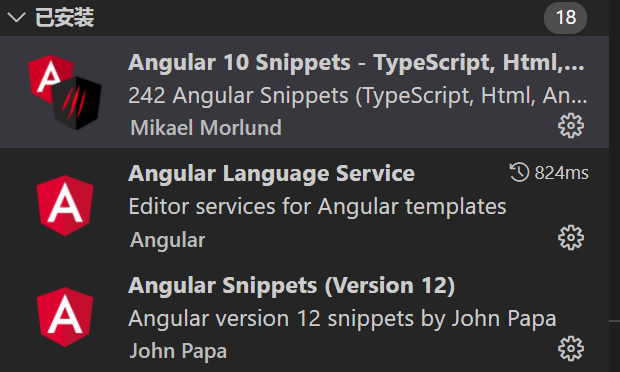
# 一、环境搭建

1、先安装nodejs

2、安装angular cli

IDE: vscode

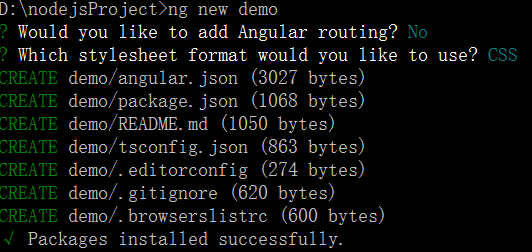
安装以下扩展：



# 二、创建项目

命令行使用 ng new 项目名称

如下，执行ng new demo



npm install编译项目

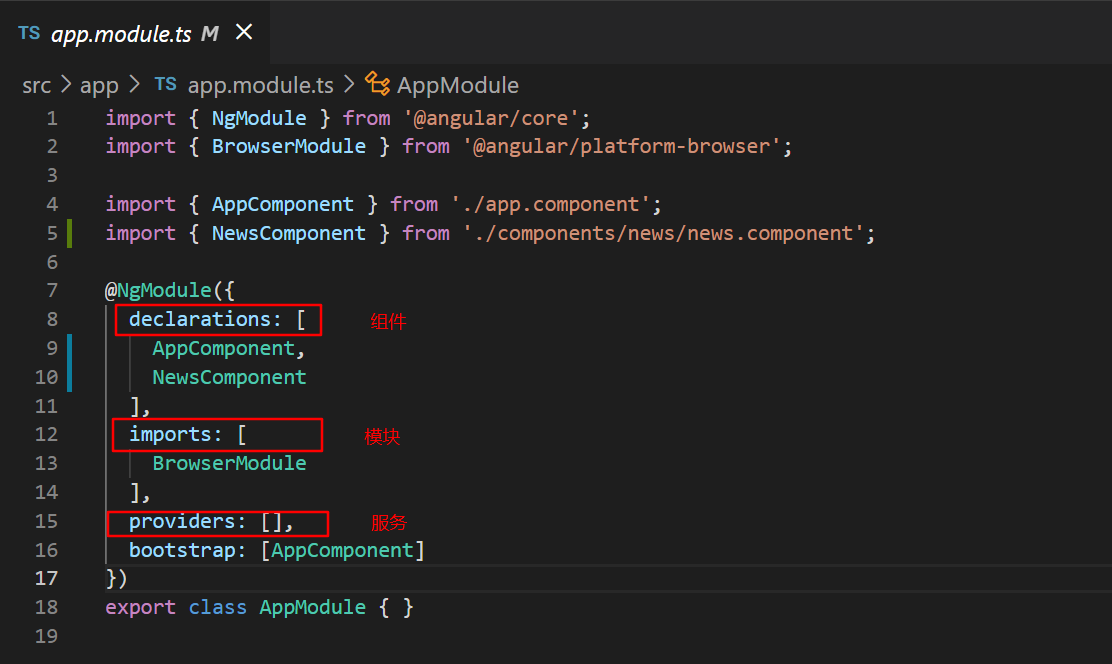
ng serve --open在浏览器打开项目

# 三、angular目录结构

package.json

app：定义模块以及服务

---app.module.ts

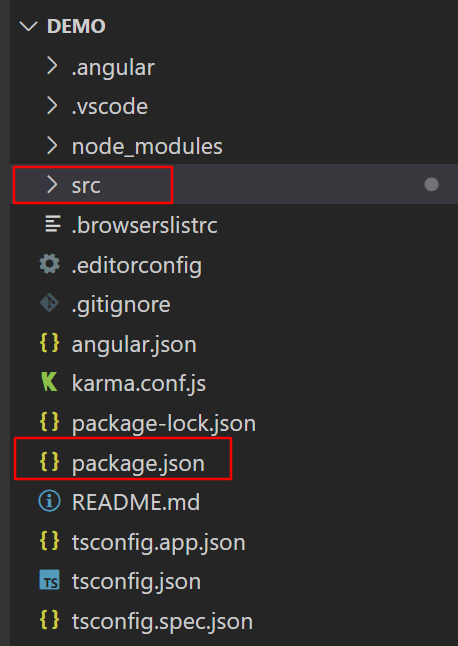


assets:静态资源

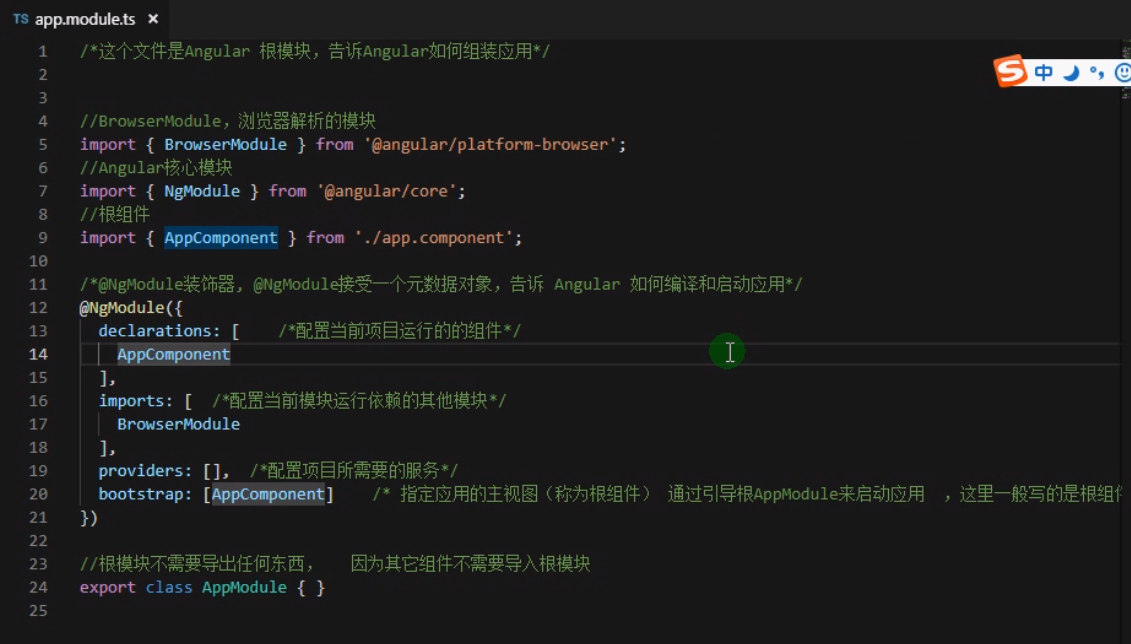
index.html：入口文件

style.css：公共的css文件

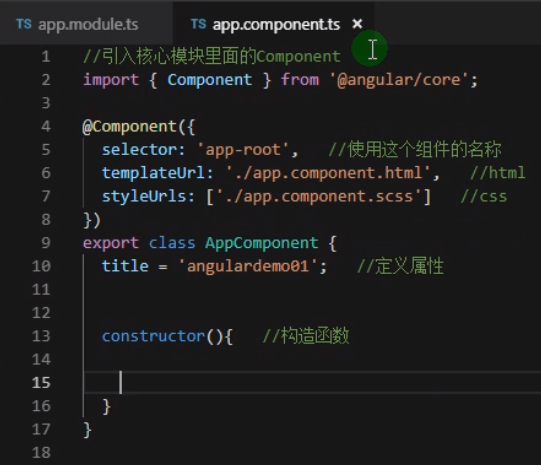
其他不用管



## app.module.ts



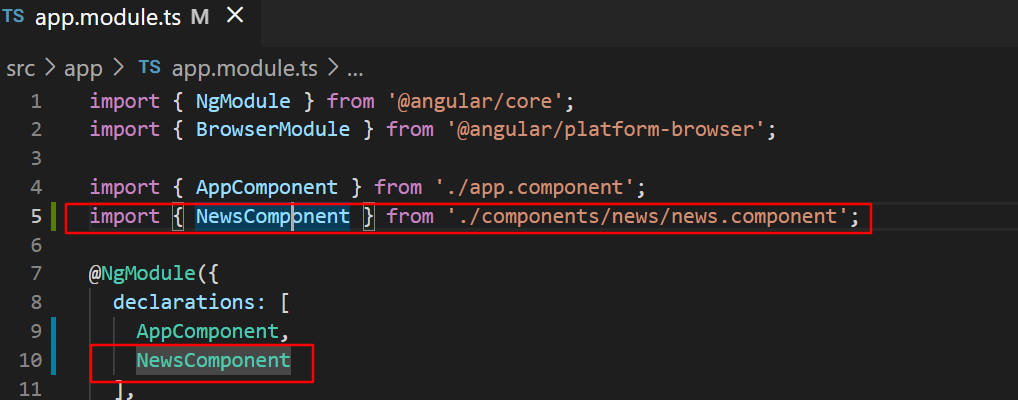
## app.component.ts



## 自定义组件

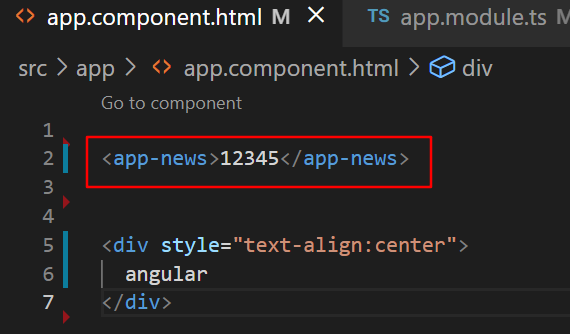
ng g component components/news



使用该命令创建的组件会自动在app.module.ts中引入，可以调用其他组件。

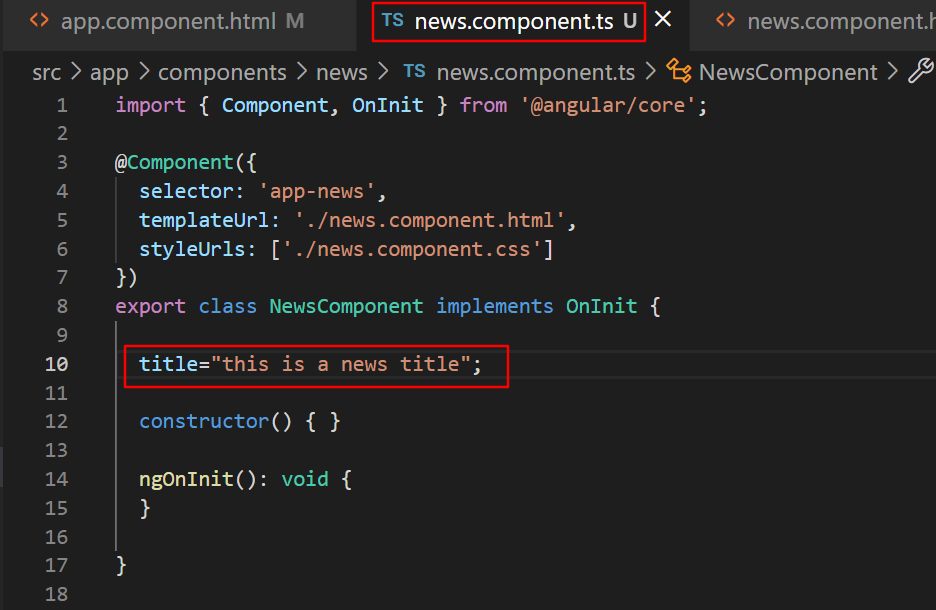
## 在根组件中使用自定义组件

在app.component.html中直接添加即可。



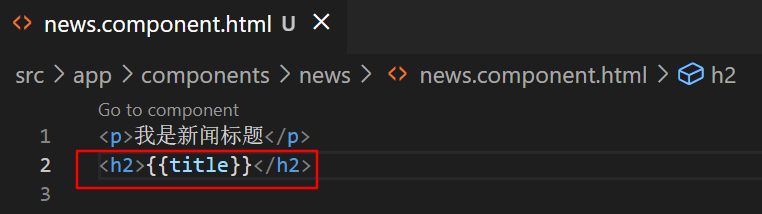
# 四、数据绑定

使用ng g component命令创建新的模块，在对应模块component.ts中添加变量。

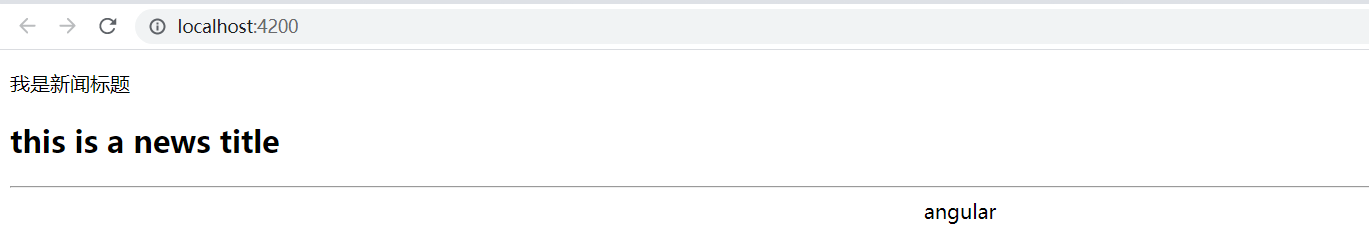


在对应的html文件中使用双花括号访问数据{{}}

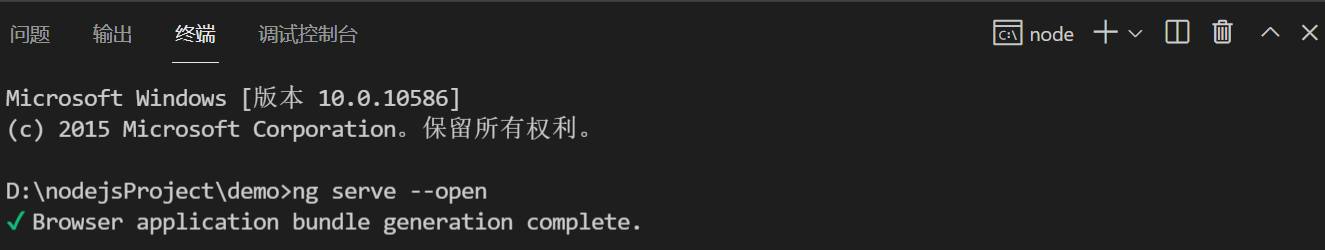
支持热加载



使用ng serve --open运行项目，使用浏览器访问<http://localhost:4200/>



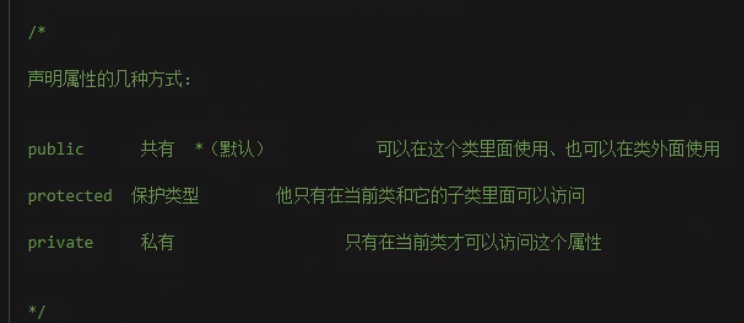
或者使用vscode自带的终端，输入ng serve --open



# 五、angular组件、组件模板

xxx.component.ts

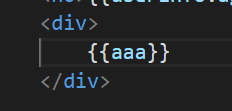
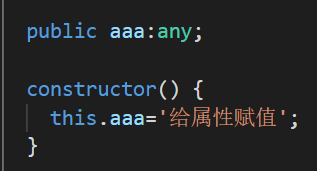
## 声明属性的方式：

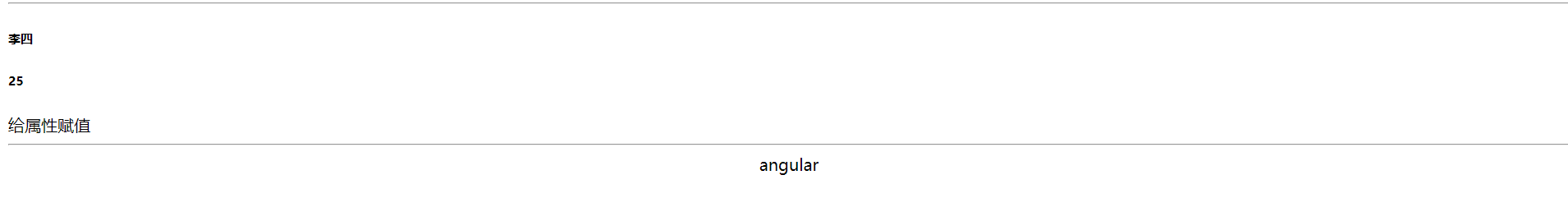


如果没有指定修饰符，默认为public。

不能使用var声明属性，var表示是一个变量，而不是一个属性。

**或者先声明一个属性，在构造方法中赋值。**

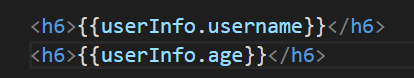




## 声明对象

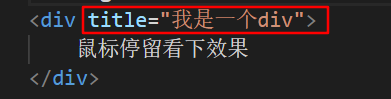


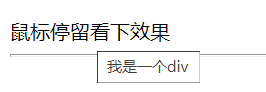
调用对象的属性：



## 绑定属性

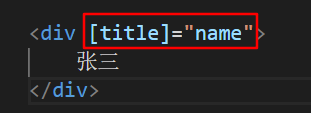
①绑定静态属性

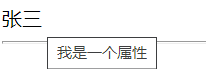




②绑定动态属性

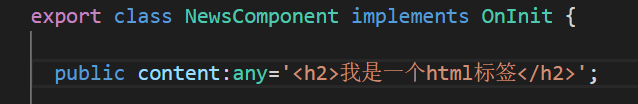




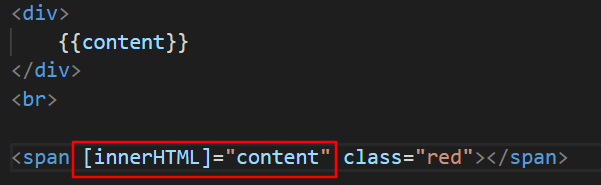


③绑定HTML

ts



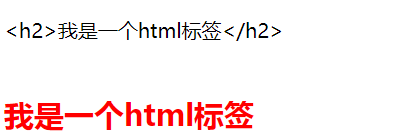
html



css



效果：



## angular模板允许简单运算

1+2={{1+2}}

## 定义数组

//定义数组

  arr=['1111','2222','3333'];

  //推荐

  public list:any[]=["橘子",50,"苹果"];

  public items:Array<any>=['1','2','3'];

  public userlist:any[]=[{

    username:'小王',

    age:20

  },{

    username:'小李',

    age:25

  },{

    username:'小兰',

    age:24

  }];

### \*ngFor普通循环

<ul>

    <li \*ngFor="let item of arr">

        {{item}}

    </li>

</ul>



### \*ngFor遍历复杂数组

public userlist:any[]=[{

    username:'小王',

    age:20

  },{

    username:'小李',

    age:25

  },{

    username:'小兰',

    age:24

  }];

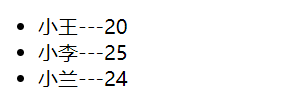
<ul>

    <li \*ngFor="let item of userlist">

        {{item.username}}---{{item.age}}

    </li>

</ul>



### 嵌套循环

ts

public cars:any[]=[{

    category:'奥迪',

    list:[{

      series:'奥迪A4',

      price:'30w'

    },{

      series:'奥迪A6',

      price:'50w'

    },{

      series:'奥迪A8',

      price:'70w'

    }]

  },{

    category:'本田',

    list:[{

      series:'雅阁',

      price:'20w'

    },{

      series:'思域',

      price:'15w'

    },{

      series:'CRV',

      price:'20w'

    }]

  },{

    category:'奔驰',

    list:[{

      series:'奔驰A级',

      price:'25w'

    },{

      series:'奔驰C级',

      price:'35w'

    },{

      series:'奔驰E级',

      price:'45w'

    }]

  }];

html

<ul>

    <li \*ngFor="let item of cars">

        {{item.category}}

        <ol>

            <li \*ngFor="let car of item.list">

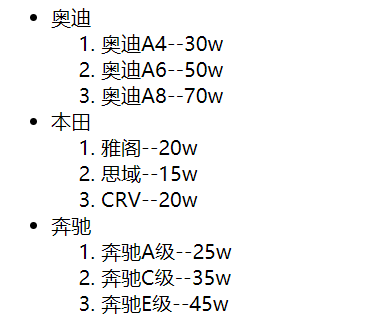
                {{car.series}}--{{car.price}}

            </li>

        </ol>

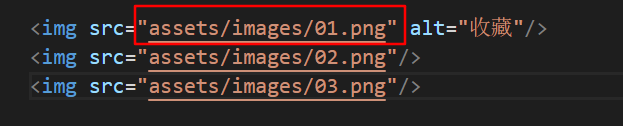
    </li>

</ul>



## 组件合成

①绑定图片



静态资源存放在assets/images目录下



②绑定动态图片

ts

public picurl="https://gimg2.baidu.com/image\_search/src=http%3A%2F%2Finews.gtimg.com%2Fnewsapp\_match%2F0%2F14252120071%2F0&refer=http%3A%2F%2Finews.gtimg.com";

html

<img [src]="picurl"/>

## 循环数据并显示索引

ts

public list:any[] = [

    {

      title:'标题1'

    },

    {

      title:'标题2'

    },

    {

      title:'标题3'

    }

  ];

html

<ul>

<li \*ngFor="let item of list;let i=index">

{{i}}--{{item.title}}

</li>

</ul>

## 条件判断语句

### ngIf

ts

public flag:boolean=true;

html

<div \*ngIf="flag">

    <img src="assets/images/01.jpg"/>

</div>

<div \*ngIf="!flag">

    <img src="assets/images/02.jpg"/>

</div>

ngIf可以在所有的标签中使用

<ul>

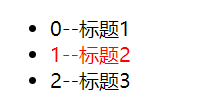
    <li \*ngFor="let item of list;let i=index">

        <span \*ngIf="i==1" class="red">{{i}}--{{item.title}}</span>

        <span \*ngIf="i!=1">{{i}}--{{item.title}}</span>

    </li>

</ul>



### ngSwitch

ts

public orderStatus:number=5; /\* 1:已支付 2：订单已确认 3：已发货 4：已收货 其他：无效订单 \*/

html

<div [ngSwitch]="orderStatus">

<span \*ngSwitchCase="1">已支付</span>

<span \*ngSwitchCase="2">订单已确认</span>

<span \*ngSwitchCase="3">已发货</span>

<span \*ngSwitchCase="4">已收货</span>

<span \*ngSwitchDefault>无效订单</span>

</div>

## 动态样式

### [ngClass]

html

<div [ngClass]="{'red':true,'blue':false}">

    ngClass演示

</div>

<hr>

<div [ngClass]="{'purple':flag,'green':!flag}">

    ngClass演示

</div>

css

.red{

    color: red;

}

.blue{

    color: blue;

}

.green{

    color: green;

}

.purple{

    color:purple;

    font-weight: 80%;

    font-family: 'Times New Roman', Times, serif;

}

遍历数组添加样式

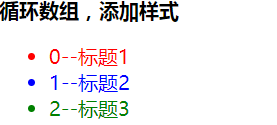
<ul>

    <li \*ngFor="let item of list;let i=index" [ngClass]="{'red':i==0,'blue':i==1,'green':i==2}">

        {{i}}--{{item.title}}

    </li>

</ul>



### [ngStyle]

<p style="color:red">我是一个段落</p>

<p [ngStyle]="{'color':'blue'}">ngStyle演示</p>

<p [ngStyle]="{'color':attr}">ngStyle动态获取属性</p>

注意：[ngStyle]的{}里面，属性和值都要加引号。

## 管道

ts

public today:any=new Date();

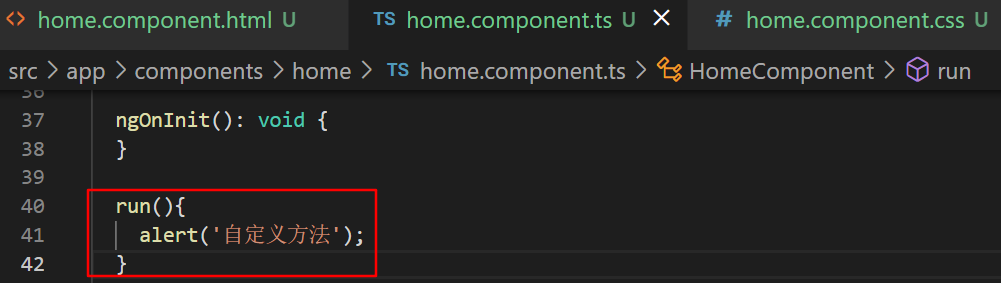
html

{{today | date:"yyyy-MM-dd HH:mm:ss"}}



自定义管道参考：<http://bbs.itying.com/topic/5bf519657e9f5911d41f2a34>

## 执行事件

ts在类中添加自定义方法：  


html

<button (click)="run()">执行事件</button>

* 获取属性的值

ts

 getData(){

    alert(this.title);

  }

html

<button (click)="getData()">执行事件获取数据</button>

* 设置属性的值

ts

public title:string="我是一个标题";

  setData(){

    this.title="我是改变后的标题";

  }

html

<button (click)="setData()">改变属性的数据</button>

* 表单事件&事件对象

keydown事件

ts

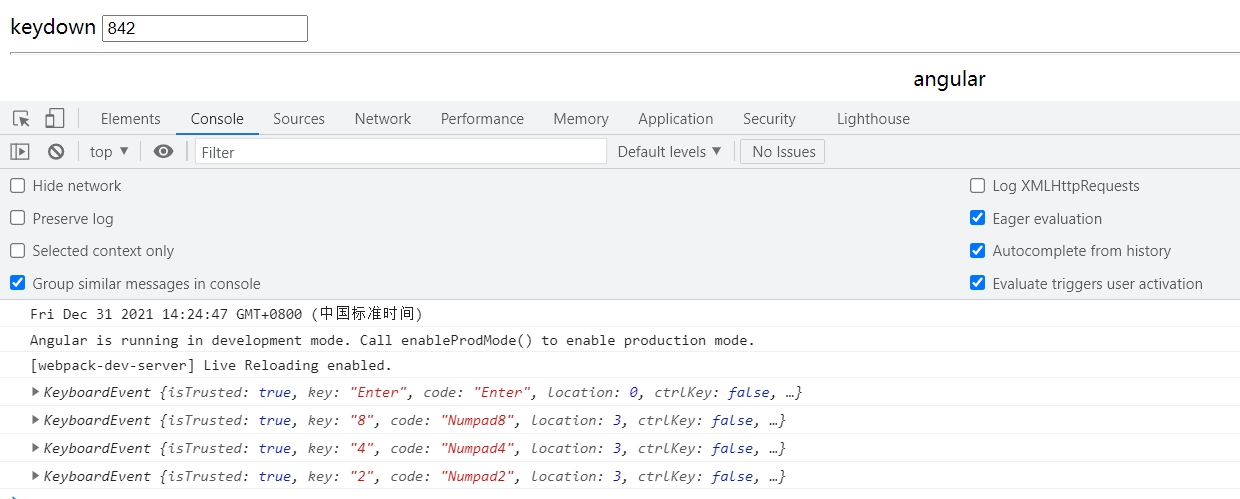
keyDown(e:any){

    console.log(e);

  }

html

<input type="text" (keydown)="keyDown($event)" />



keyup事件

ts

  keyUp(e:any){

    console.log(e)

  }

html

<input type="text" (keyup)="keyUp($event)" />



**获取事件对象**

**ts**

  runEvent(event:any){

    var dom:any=event.target; //建议指定类型

    dom.style.color='red';

  }

**html**

<button (click)="runEvent($event)">执行方法获取事件对象</button>

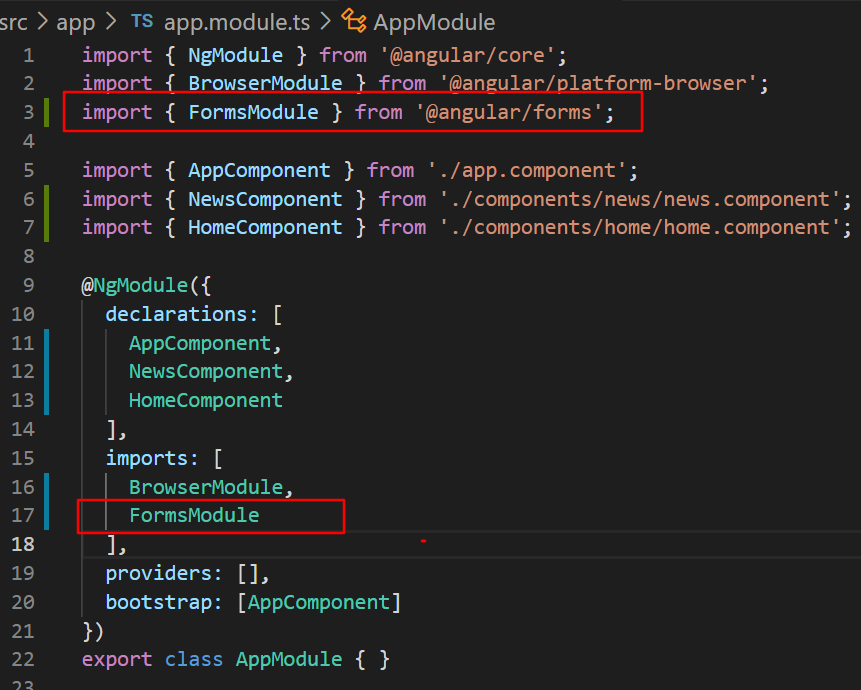
## 双向数据绑定（重点）

**MVVM:视图改变影响数据，数据改变也会影响视图**

需要引入FormsModule

在app.module.ts引入

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



home.component.ts

  changekeywords(){

    this.keywords="改变后的keywords";

  }

  getkeywords(){

    console.log(this.keywords);

  }

html

<input type="text" [(ngModel)]="keywords" />

{{keywords}}

<br>

<br>

<button (click)="changekeywords()">改变keywords的值</button>

<br>

<br>

<button (click)="getkeywords()">获取keywords的值</button>

# 六、angular表单

获取input、checkbox、radio、select、textarea的内容

## 6.1获取input内容

需要先导入FormsModule模块。

在app.module.ts添加import { FormsModule } from '@angular/forms';

在imports中添加

imports: [

    FormsModule

  ],

使用双向数据绑定获取输入内容

ts

  public personInfo:any={

    username:''

  }

doSubmit(){

    // dom操作

    /\*

    let username:any = document.getElementById("username");

    console.log(username.value);

    \*/

    //双向数据绑定

    console.log(this.personInfo);

  }

html

<div class="person">

    <ul>

        <li>姓名：<input type="text" id="username"  [(ngModel)]="personInfo.username"/></li>

    </ul>

    <button (click)="doSubmit()">获取表单的内容</button>

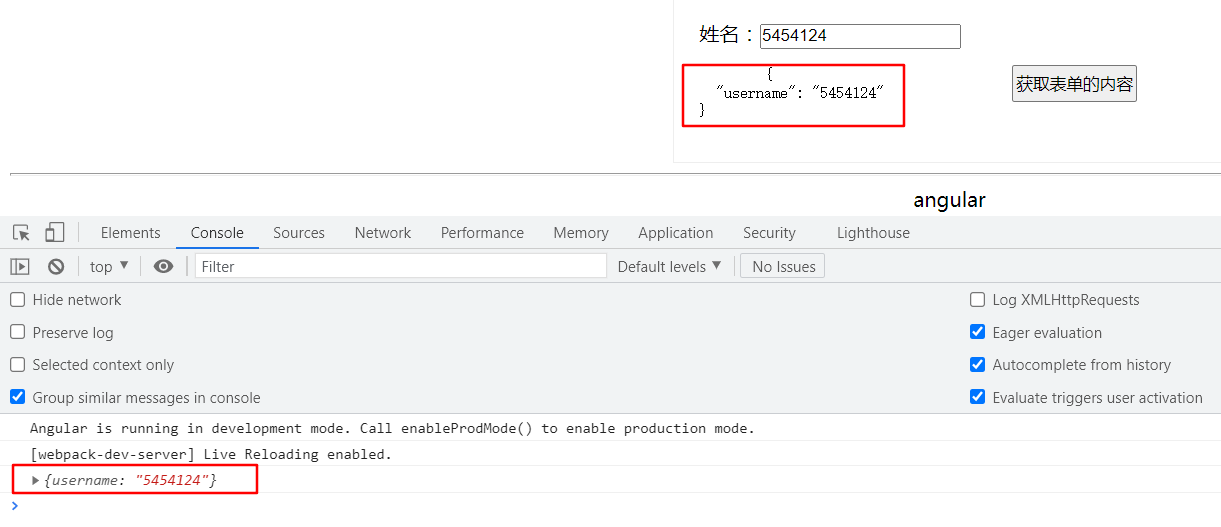
    <pre>

        {{personInfo | json}}

    </pre>

</div>

效果如下：



## 6.2获取单选框radio的值

ts

  public personInfo:any={

    username:'',

    sex:'1'

  }

html

<div class="person">

    <ul>

        <li>姓名：<input type="text" id="username"  [(ngModel)]="personInfo.username"/></li>

        <li>性别：

            <input type="radio" value="1" name="sex" id="sex1" [(ngModel)]="personInfo.sex"/><label for="sex1">男</label>

            <input type="radio" value="2" name="sex" id="sex2" [(ngModel)]="personInfo.sex"/><label for="sex2">女</label>

        </li>

    </ul>

    <button (click)="doSubmit()">获取表单的内容</button>

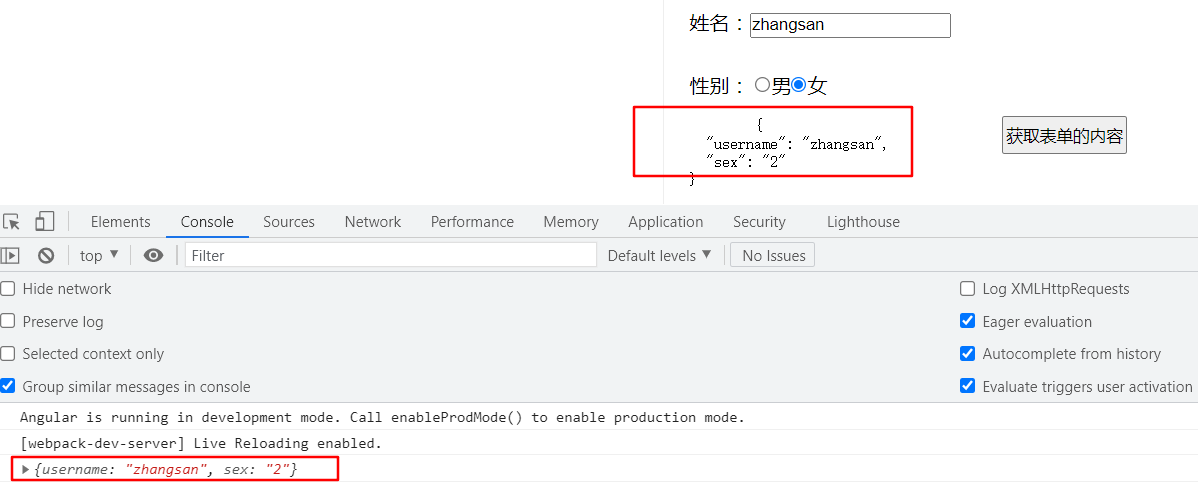
    <pre>

        {{personInfo | json}}

    </pre>

</div>

效果如下：



## 6.3获取select下拉列表内容

ts

 public personInfo:any={

    username:'',

    sex:'1',

    cityList:['北京','上海','广州','深圳','杭州'],

    city:'北京'

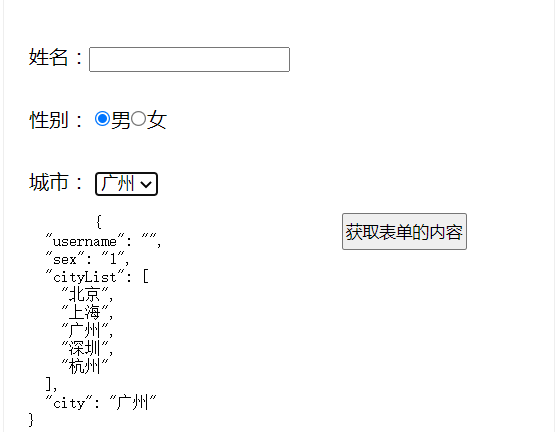
  }

html

<select name="city" [(ngModel)]="personInfo.city">

                <option [value]="item" \*ngFor="let item of personInfo.cityList">{{item}}</option>

            </select>



## 6.4获取CheckBox复选框的内容

ts

public personInfo:any=              {

    username:'',

    sex:'1',

    cityList:['北京','上海','广州','深圳','杭州'],

    city:'北京',

    hobby:[{

      title:'旅游',

      checked:true

    },{

      title:'唱歌',

      checked:false

    },{

      title:'跑步',

      checked:false

    }]

  }

html

<li>爱好：

            <span \*ngFor="let item of personInfo.hobby;let key=index">

                <input type="checkbox" [(ngModel)]="item.checked" [id]="'check'+key" /><label [for]="'check'+key">{{item.title}} &nbsp;&nbsp;</label>

            </span>

        </li>



## 6.5获取textarea内容

ts

public personInfo:any={

    username:'',

    sex:'1',

    cityList:['北京','上海','广州','深圳','杭州'],

    city:'北京',

    hobby:[{

      title:'旅游',

      checked:true

    },{

      title:'唱歌',

      checked:false

    },{

      title:'跑步',

      checked:false

    }],

    mark:''

  }

html

<li>备注：

            <textarea name="mark" [(ngModel)]="personInfo.mark" rows="5"></textarea>

        </li>



# 七、angular实现搜索缓存功能

需求描述：搜索框输入的内容，显示在搜索框下面，重复的内容不会再次出现。

## 7.1search.component.ts

import { Component, OnInit } from ‘@angular/core’;

@Component({

  selector: ‘app-search’,

  templateUrl: ‘./search.component.html’,

  styleUrls: [‘./search.component.css’]

})

export class SearchComponent implements OnInit {

  public keyword:string=’’;

  public historyList:any[]=[];

  constructor() { }

  ngOnInit(): void {

  }

  doSearch(){

if (this.historyList.indexOf(this.keyword) == -1){

      this.historyList.push(this.keyword);

}

this.keyword=’’;

//console.log(this.keyword);

  }

}

## 7.2search.component.html

<div class=”search”>

<input type=”text” [(ngModel)]=”keyword”/> <button (click)=”doSearch()”>搜索</button>

<hr />

<span>搜索历史</span>

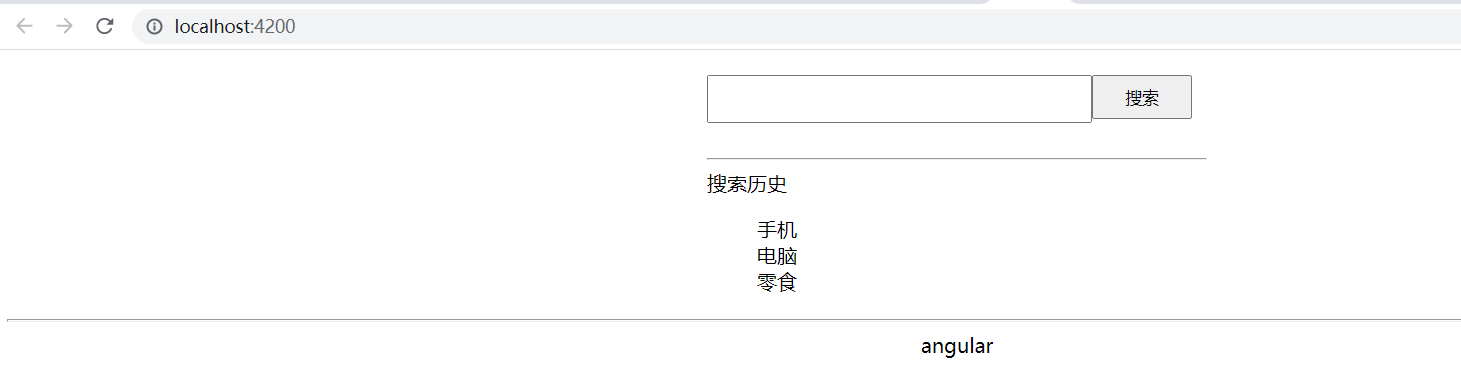
<ul>

        <li \*ngFor=”let item of historyList”>{{item}}</li>

</ul>

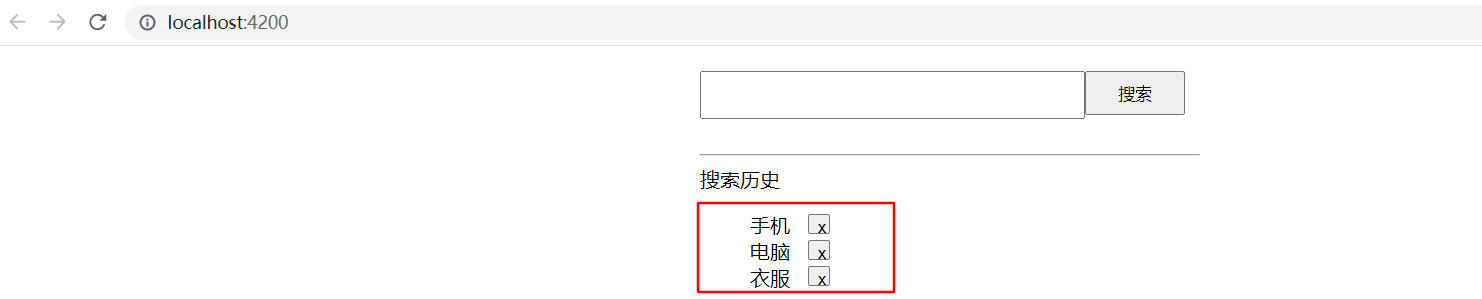
</div>

效果图



## 7.3删除历史记录

需求：每条搜索记录可以删除。如下图，点击记录后面的x，可以删除一条记录。



ts

  deleteHistory(key:any){

    // alert(key);

    this.historyList.splice(key,1);

  }

html

<ul>

        <li \*ngFor="let item of historyList;let key=index">

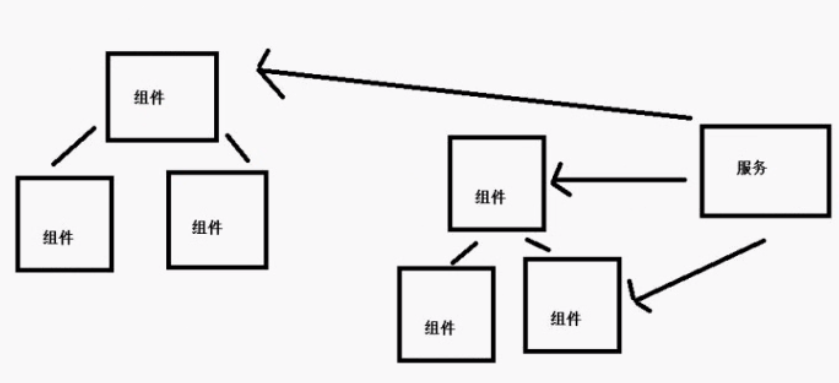
            {{item}} &nbsp;&nbsp;<button class="del-btn" (click)="deleteHistory(key)">x</button>

        </li>

    </ul>

# 八、angular服务&数据持久化

## 8.1 angular中的服务



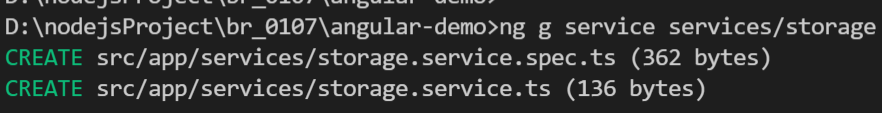
angular是基于模块化、组件化的开发方式，默认情况下一个组件无法调用另一个组件内的方法，组件间是相互隔离的。将公共的功能封装起来放到服务中去，在需要调用的组件中，只要引入该服务，就可以调用公共方法。

## 8.2创建服务

ng g service service-name

创建到指定目录下：

ng g service services/storage



## 8.3引入并配置服务

在app.module.ts中引入服务，并配置

import { StorageService } from './services/storage.service';

@NgModule({

…

providers: [StorageService],

…

})

storage.service.ts中有如下方法：

export class StorageService {

  constructor() { }

  get(){

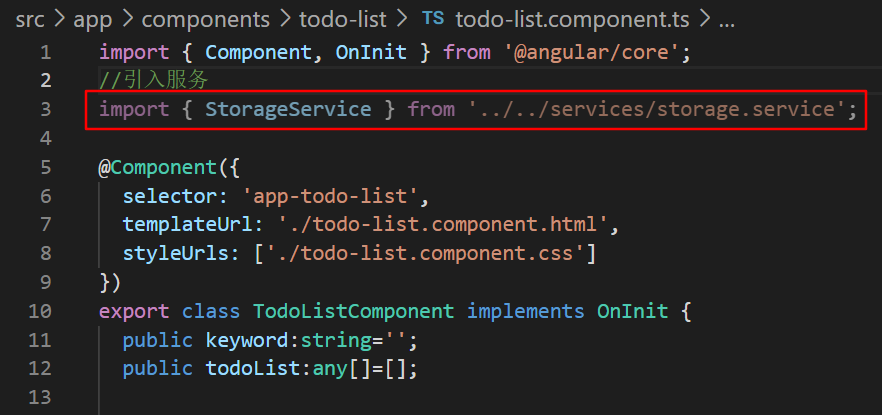
    return 'this is a service';

  }

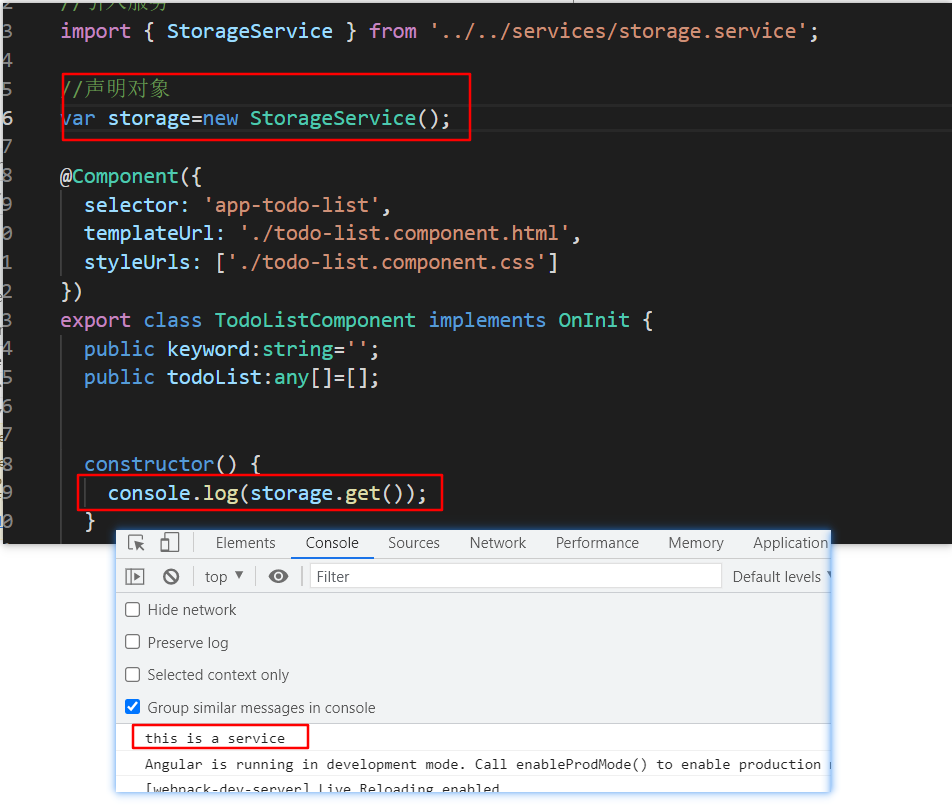
}

## 8.4在组件中使用服务

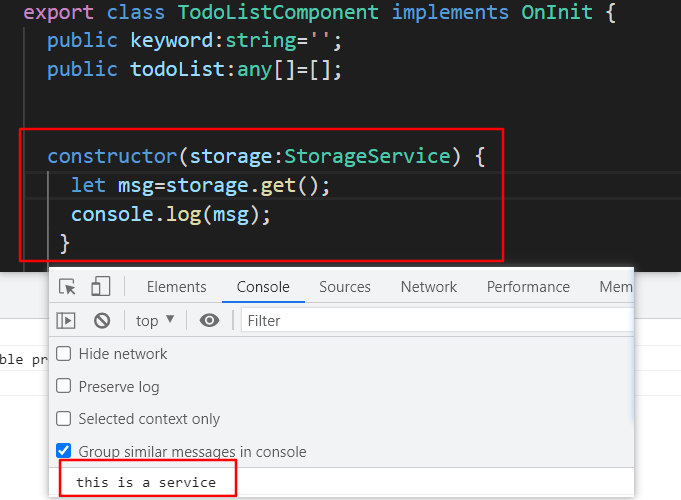
需要在要调用服务的组件中引入服务，



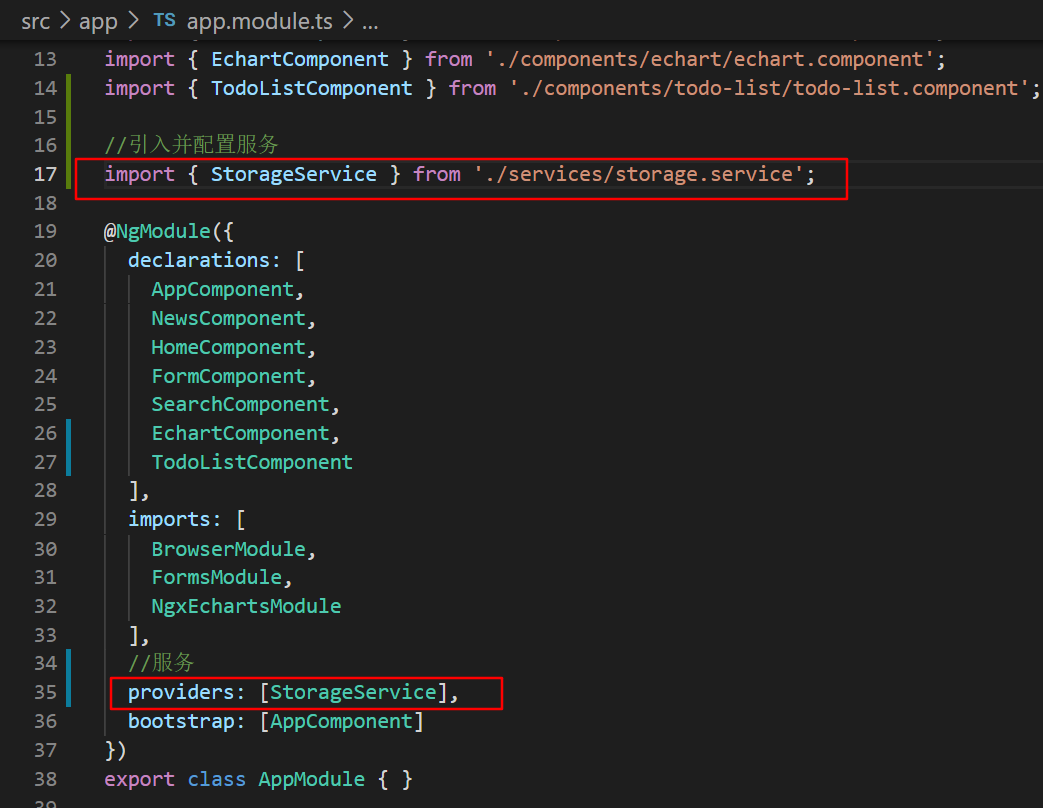
### 不推荐的调用方式

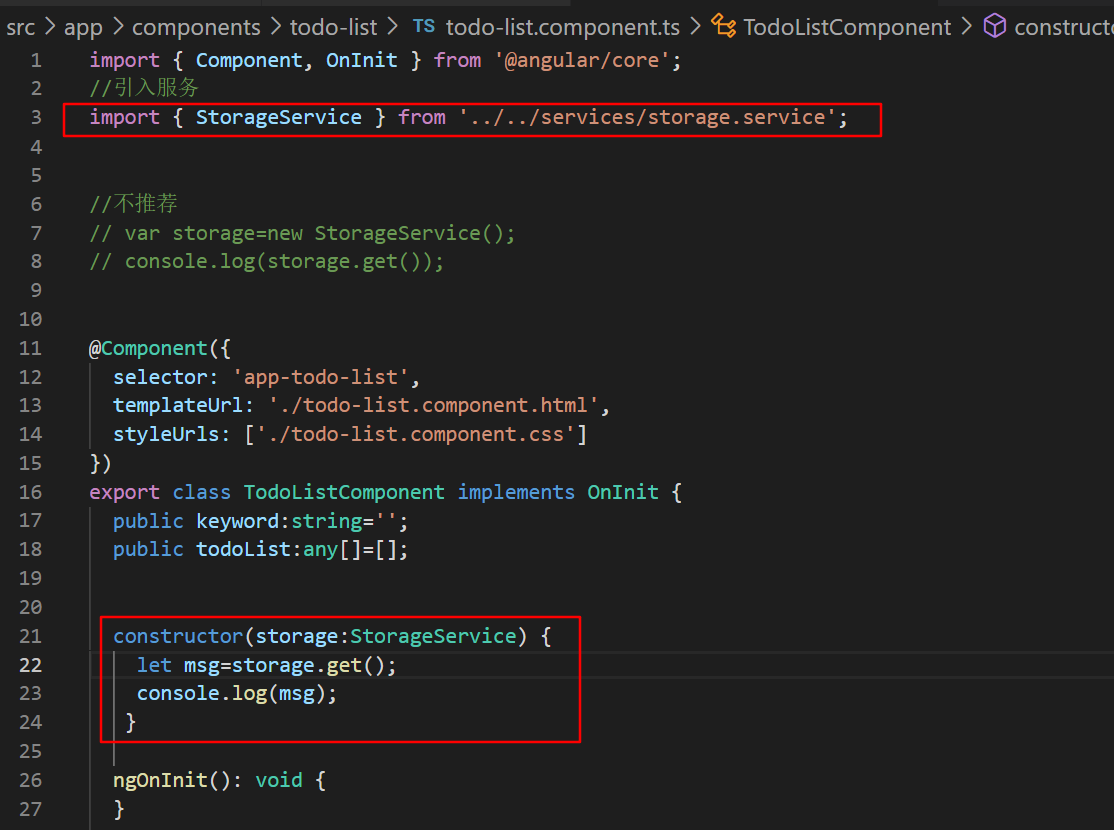


### 官方推荐用法



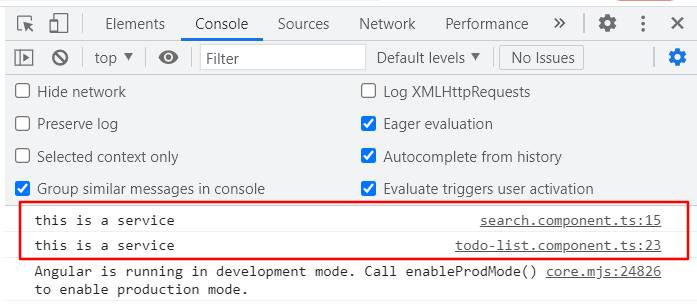
【注意】在组件中使用服务时，需要引入两次服务。一次是在app.module.ts引入并配置，一次是在对应组件的xxx.component.ts中引入。





### 在多个组件中调用服务

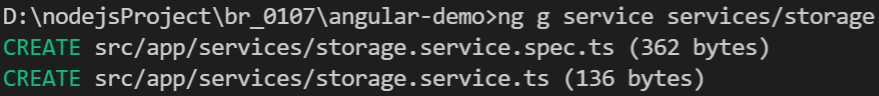
search.component.ts和todo-list.component.ts中分别引入服务，调用get()：



## 8.5 angular服务小结

1、创建服务

ng g service services/服务名



2、app.module.ts引入创建的服务，并声明服务

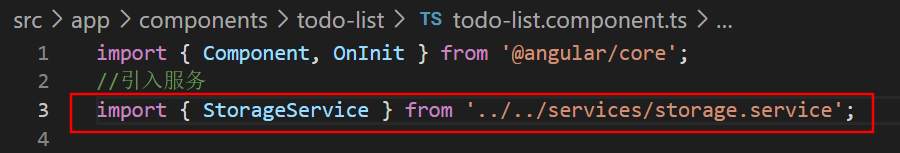
//引入服务

import { StorageService } from './services/storage.service';

//声明服务

  providers: [StorageService],

3、在用到的组件里面，引入服务



4、在构造函数中初始化

  constructor(public storage:StorageService) {

    let msg=this.storage.get();

    console.log(msg);

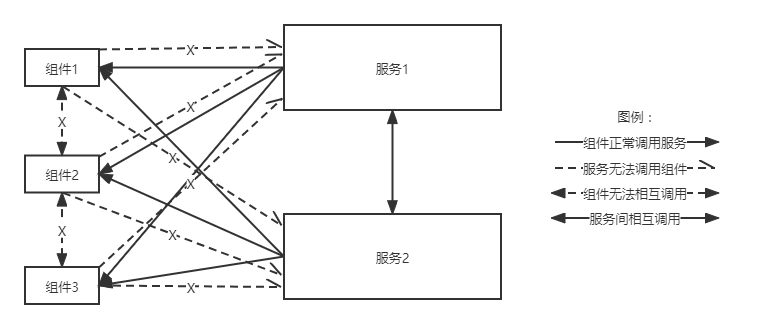
   }

angular中，所有的组件都可以调用服务，只需要引入服务即可。

服务和服务之间可以相互调用。

反过来，服务无法调用组件中的方法；组件之间的方法也无法相互调用。

调用关系如下图所示：



## 8.6 angular数据持久化

需求：实现页面刷新后，历史记录仍然存在。

未实现数据持久化前，输入的内容刷新后就没有了，如下图：



数据持久化后，在页面刷新后，数据依旧存在



## 8.7在服务中缓存数据

  set(key:string,value:any){

    localStorage.setItem(key,JSON.stringify(value));

  }

  get(key:string){

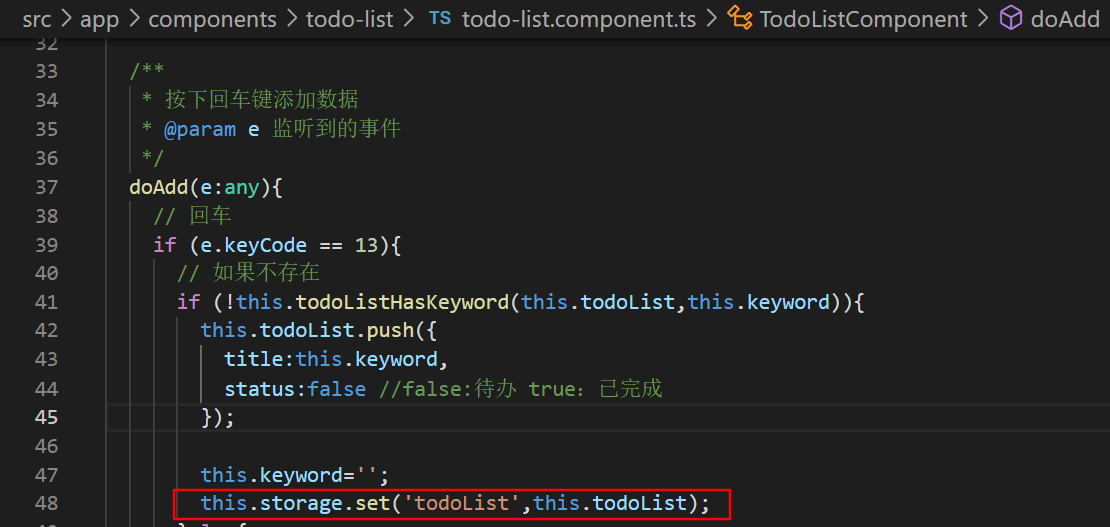
    let str = localStorage.getItem(key);

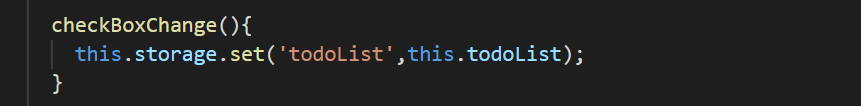
    return JSON.parse((str != null ? str : ""));

  }

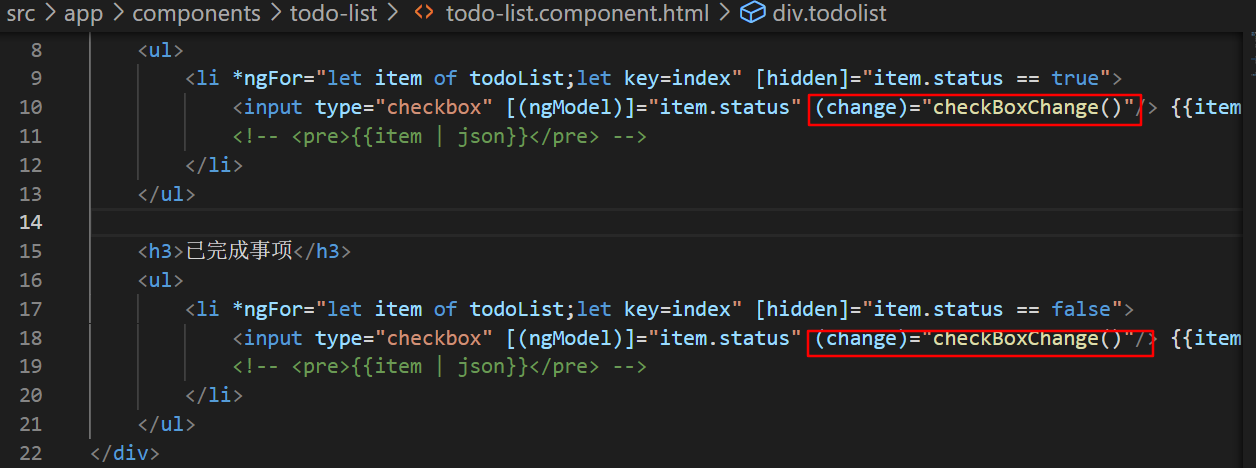
## 8.8组件调用服务

todolist.component.ts

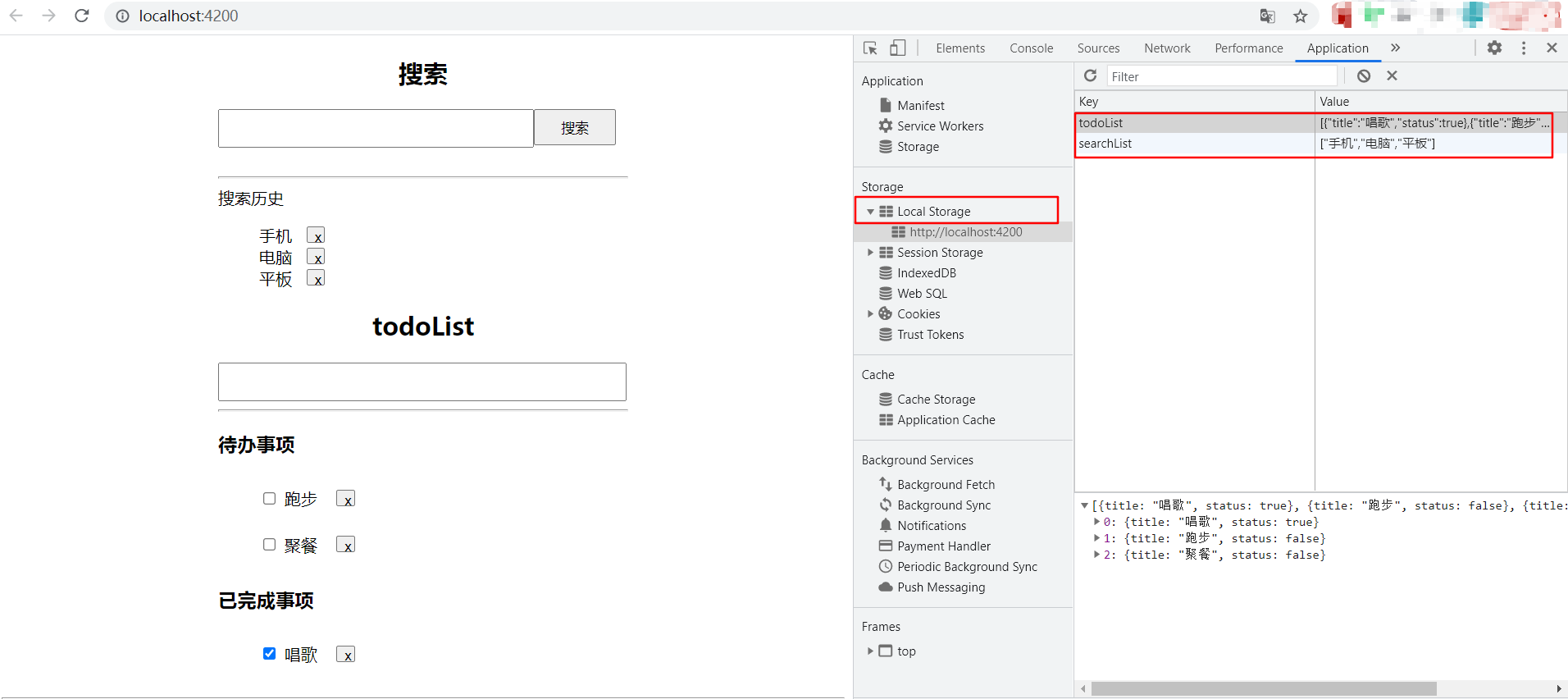




todolist.component.html



运行结果：

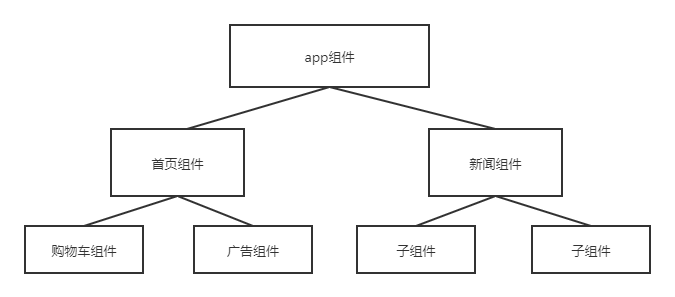


# angular中的dom操作&css动画

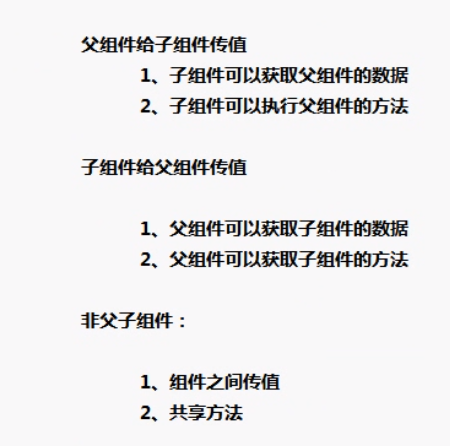
# angular父子组件&组件间通讯

## angular父子组件

父子组件是一种相对关系，如下图所示，app组件相对于首页组件是父组件，首页组件相对于app组件是子组件；在首页组件引入购物车组件，首页组件相对于购物车组件就是父组件。



父子组件之间的传值分为下面几种情况：



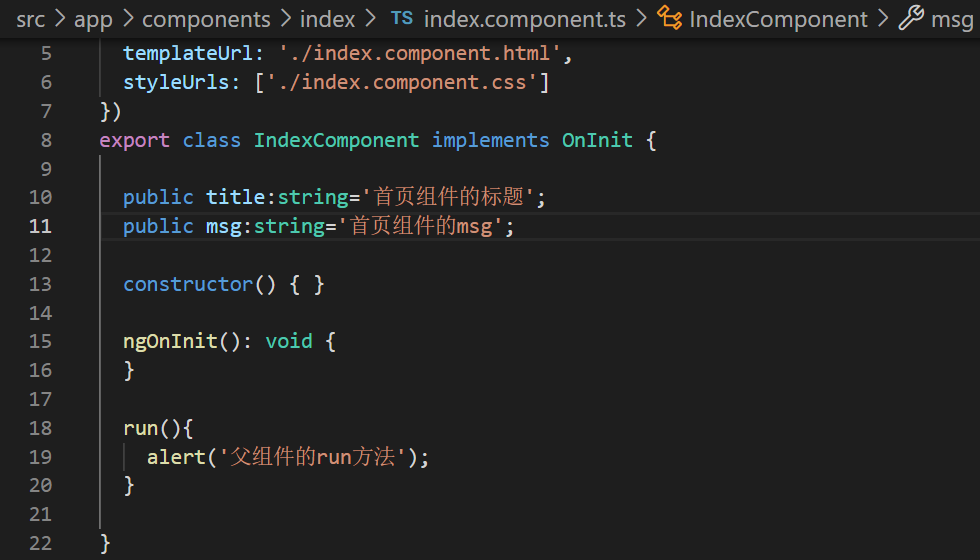
### 父组件给子组件传值@input

父组件不仅可以给子组件传递简单的数据，还可以把自己的方法以及整个父组件传给子组件。

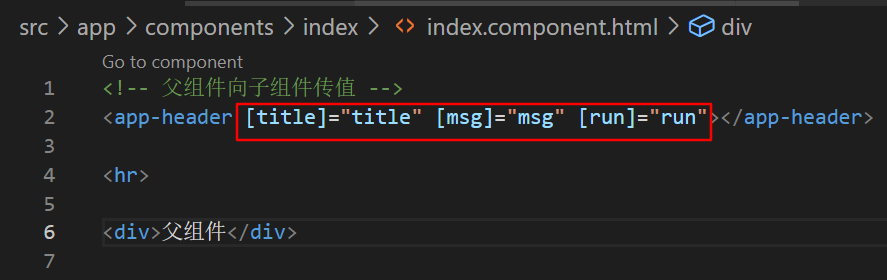
父组件：index.component

子组件：header.component

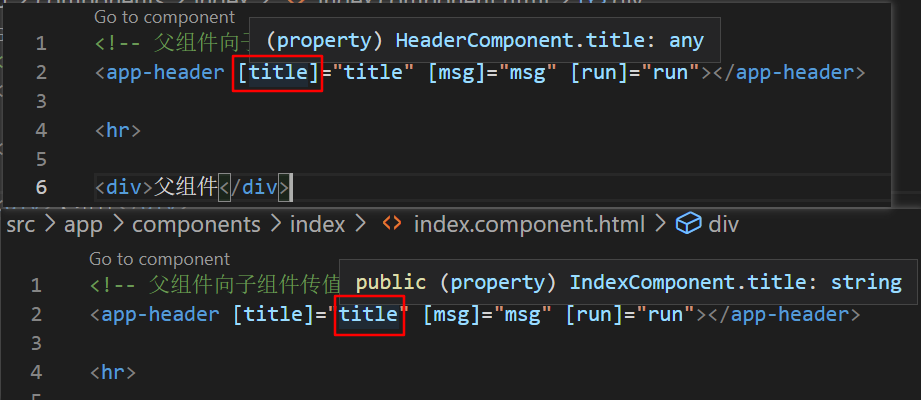
1、在父组件中定义属性和方法



2、父组件引入子组件



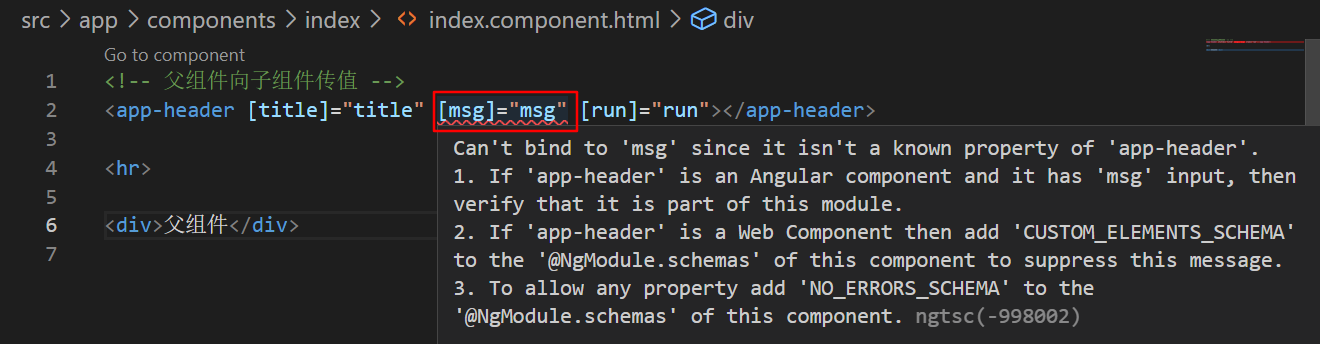
[]内是子组件的属性名称，""内是父组件的属性名称。如下：



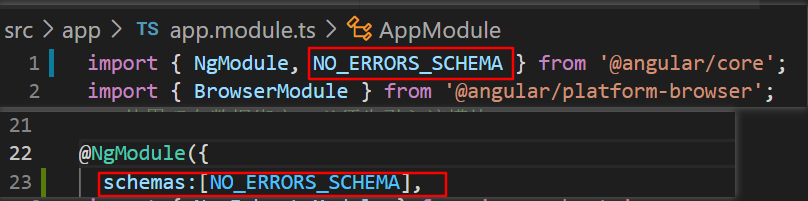
3、子组件接收父组件传入的数据



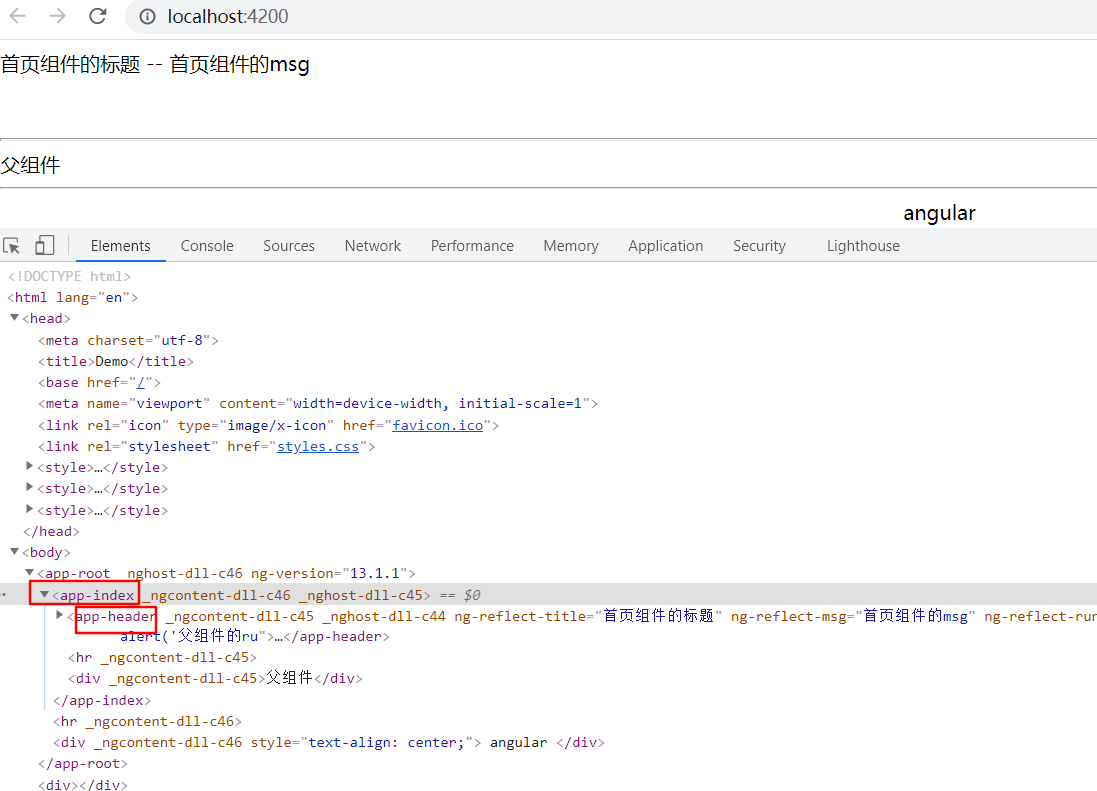
【注意】如果父组件向子组件传值时，如果子组件还未定义相应的属性名，编译器会报错。此时可以在app.module.ts中添加schemas:[NO\_ERRORS\_SCHEMA]关闭该提示。



关闭错误提示，不建议这样做。

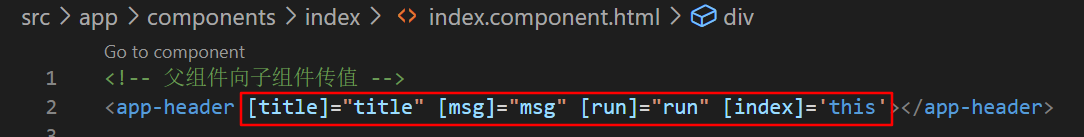


ng serve --open运行：

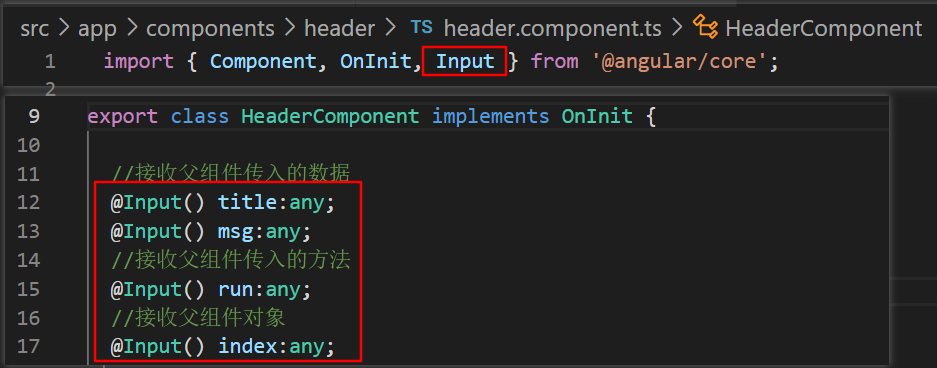


### 父组件向子组件传值小结

1、在父组件引入子组件的地方使用[]传入值(数据、方法、对象)



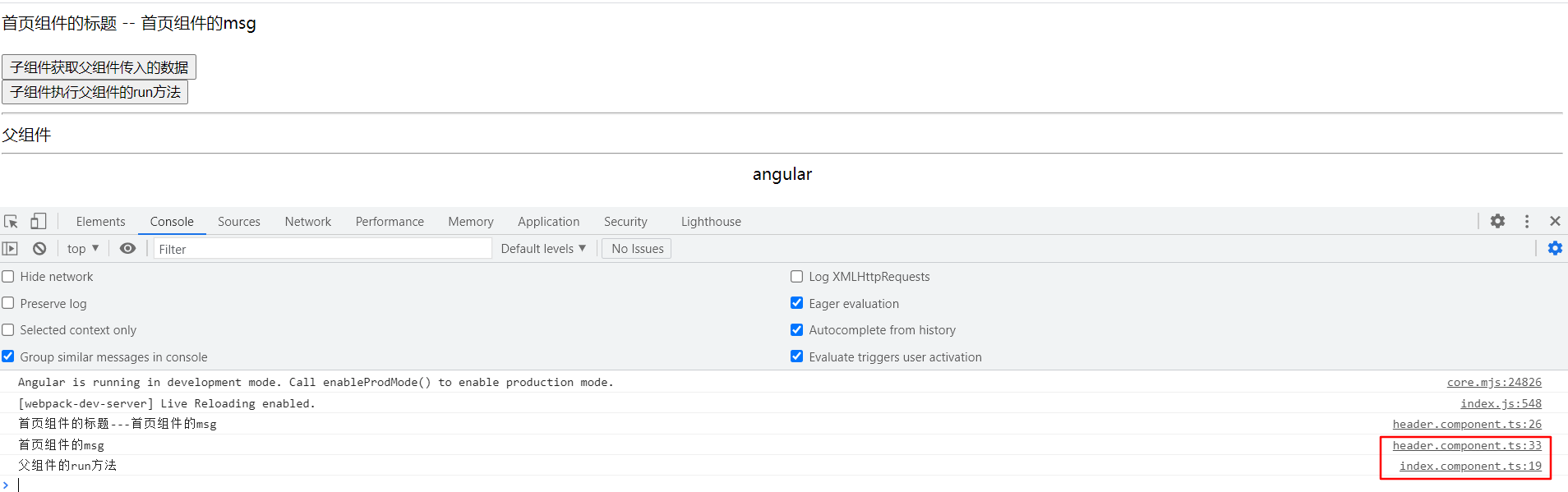
2、子组件导入Input，接收父组件传来的值



3、在子组件中进行调用



运行截图



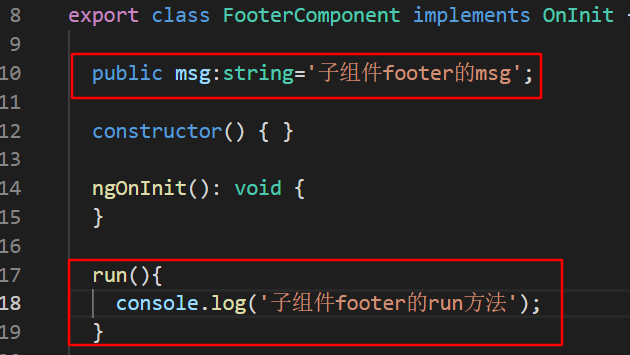
### 父组件从子组件获取数据和方法@ViewChild

1、父组件引入子组件时，给子组件设置标识

index.component.html



子组件footer.component.ts中定义属性和方法



2、父组件导入ViewChild，获取子组件对象



3、父组件使用获取到的子组件对象调用属性和方法

  getChildMsg(){

    console.log(this.footer.msg);

  }

  getChildRun(){

    this.footer.run();

  }

### 子组件使用@Output触发父组件的方法

#### 1.子组件引入output&EventEmitter

//footer.component.ts

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

#### 2.子组件实例化EventEmitter

//footer.component.ts

@Output() public outer = new EventEmitter();

#### 3.在父组件中监听

//index.component.html

<app-footer #footer (outer)="parentRun($event)"></app-footer>

父组件中监听的对象必须和子组件中定义的一样，如此处为outer。

//index.component.ts

  parentRun(e:any){

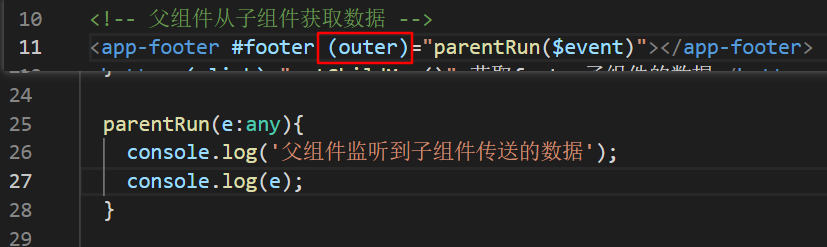
    console.log('父组件监听到子组件传送的数据');

    console.log(e);

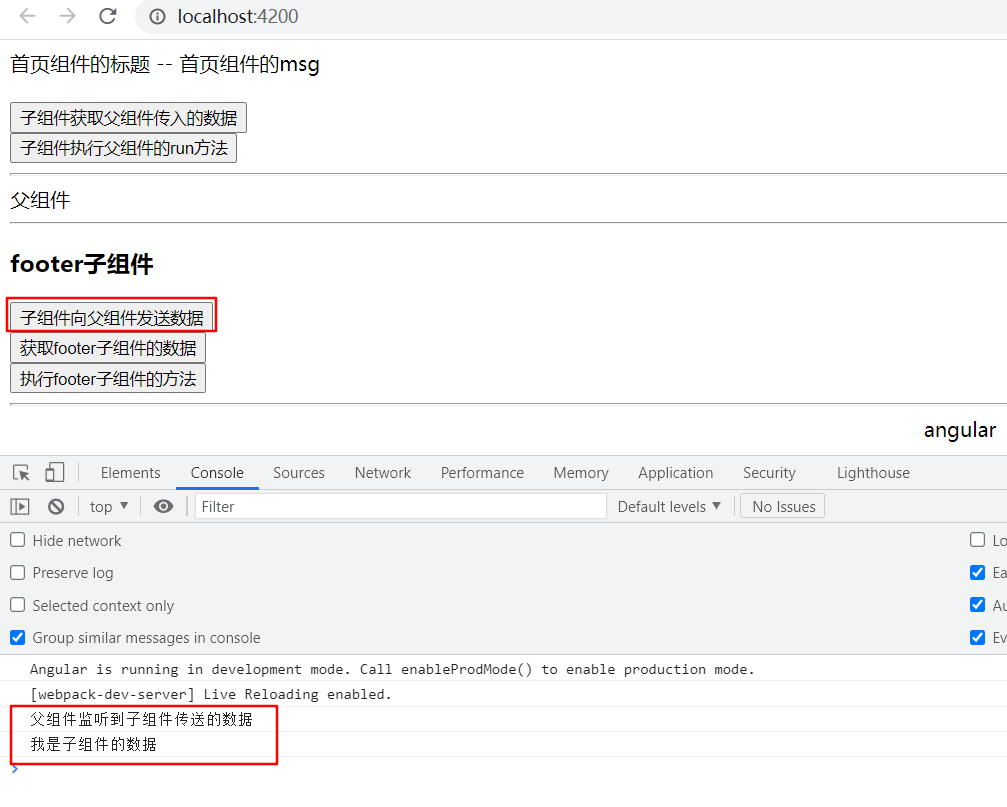
  }

子组件中广播数据时，会触发父组件的方法。





运行截图



## 组件间通讯小结

1、父组件给子组件传值

使用@Input

2、子组件给父组件传值

使用@ViewChild（推荐）或者@Output(较复杂，一般不用)。

3、非父子组件间通讯：

1）使用LocalStorage

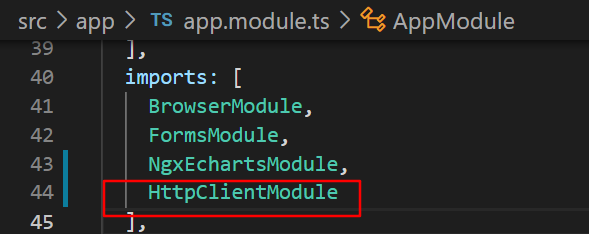
2）通过Angular服务

# angular数据交互

## angular get请求数据

1、在app.module.ts中引入HttpClientModule并注入

import { HttpClientModule } from '@angular/common/http';



2、在用到get方法的组件中引入HttpClient

import { HttpClient } from '@angular/common/http';

在构造函数中声明对象：

  constructor(public http:HttpClient) { }

3、get方法请求数据

getData(){

    let api = "http://a.itying.com/api/productlist";

    this.http.get(api).subscribe((response:any)=>{

      console.log(response);

      this.list = response.result;

    });

  }

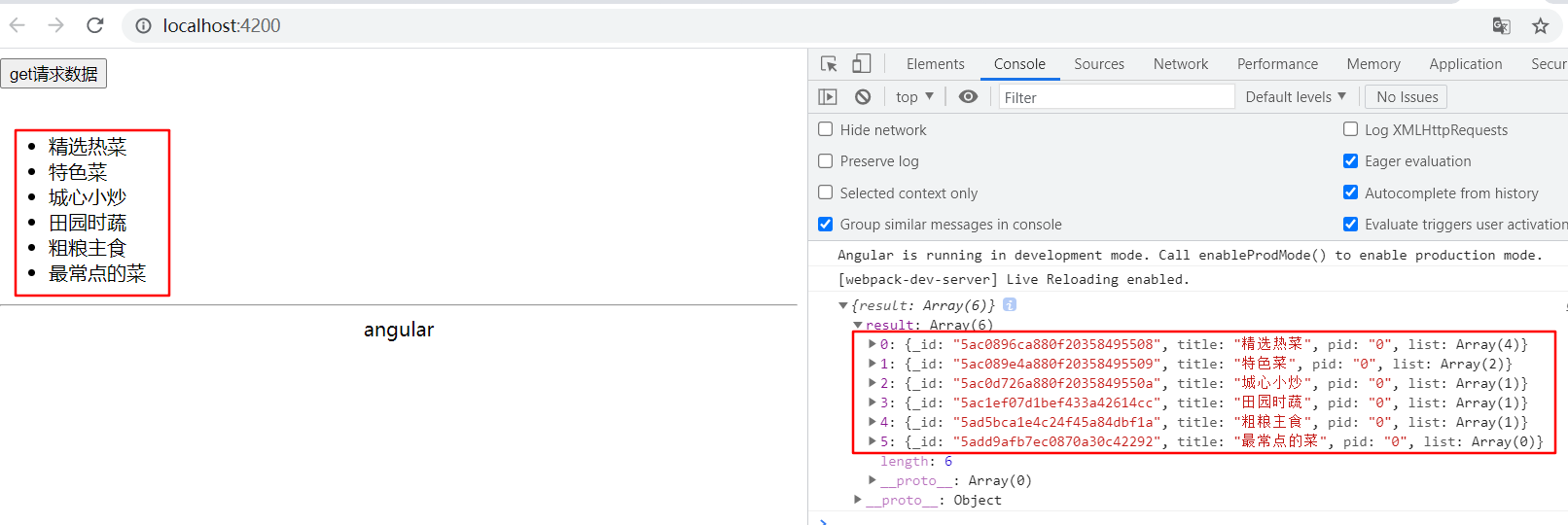
4、页面渲染数据

<ul>

    <li \*ngFor="let item of list">{{item.title}}</li>

</ul>

效果：



## angular post提交数据

1、在app.module.ts引入HttpClientModule

import { HttpClientModule } from '@angular/common/http';

imports: [

    HttpClientModule

  ],

2、在要使用的组件中引入HttpHeaders

data-interaction.component.ts

import { HttpClient, HttpHeaders } from '@angular/common/http';

post提交数据的方法：

  doLogin(){

    //手动设置请求的类型

    const httpOptions = {headers:new HttpHeaders({'Content-Type':'application/json'})};

    let api = "http://127.0.0.1:3000/doLogin";

    this.http.post(api,{'username':'张三','age':20},httpOptions).subscribe((response)=>{

      console.log(response);

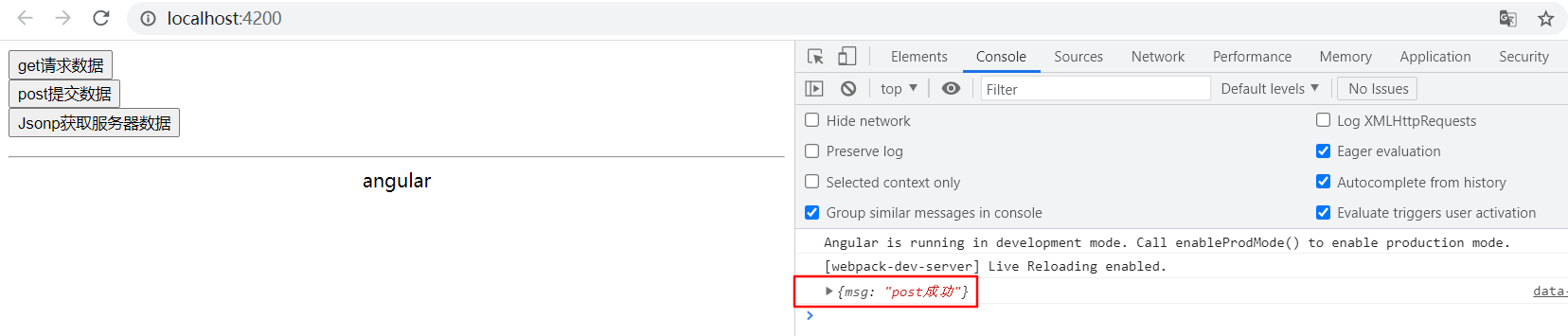
    });

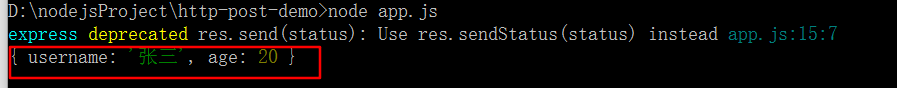
  }

3、页面调用

<button (click)="doLogin()">post提交数据</button>

在本地使用nodejs启动http服务，监听3000端口





## angular jsonp请求数据

1、在app.module.ts引入HttpClientJsonpModule

import { HttpClientModule, HttpClientJsonpModule } from '@angular/common/http';

imports: [

    HttpClientModule,

    HttpClientJsonpModule

  ],

2、在要使用的组件中引入HttpClient

import { HttpClient } from '@angular/common/http';

定义jsonp调用方法

  getJsonpData(){

    //jsonp请求，服务器必须支持jsonp,如下：

    /\*

    http://a.itying.com/api/productlist?callback=xxx

    http://a.itying.com/api/productlist?cb=xxx

    \*/

    let api = "http://a.itying.com/api/productlist";

    this.http.jsonp(api, 'callback').subscribe((response)=>{

      console.log(response);

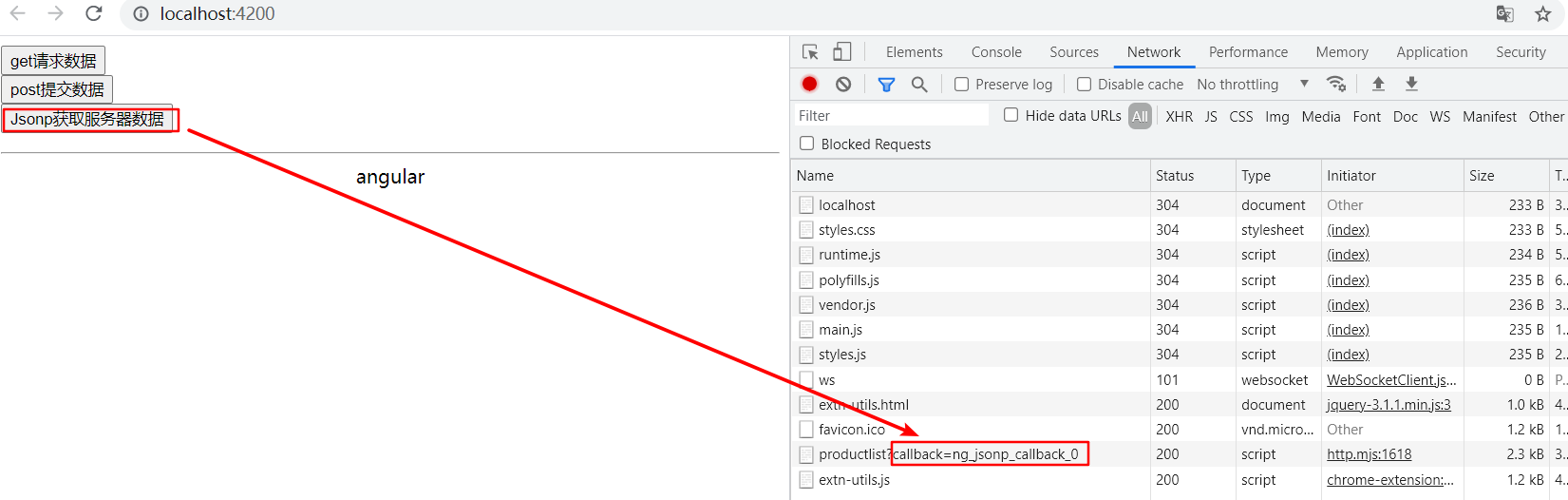
    });

  }

3、页面调用

data-interaction.component.html

<button (click)="getJsonpData()">Jsonp获取服务器数据</button>



## angular使用第三方模块axios请求数据

1、安装axios

npm install axios --save

2、创建服务

ng g service services/httpservice

3、在httpservice.service.ts引入axios

import axios from 'axios';

封装axios get请求

  axiosGet(api:any){

    return new Promise((resolve,reject)=>{

      axios.get(api).then(function(response){

        resolve(response);

      });

    });

  }

4、在app.module.ts引入服务并配置服务

import { HttpserviceService } from './services/httpservice.service';

providers: [

    HttpserviceService

  ],

5、在组件data-interaction.component.ts中引入服务

import { HttpserviceService } from '../../services/httpservice.service';

构造方法中声明服务对象：

constructor(public http:HttpClient, public httpService:HttpserviceService) { }

6、调用服务中的方法

  getAxiosData(){

    let api = "http://a.itying.com/api/productlist";

    this.httpService.axiosGet(api).then((data)=>{

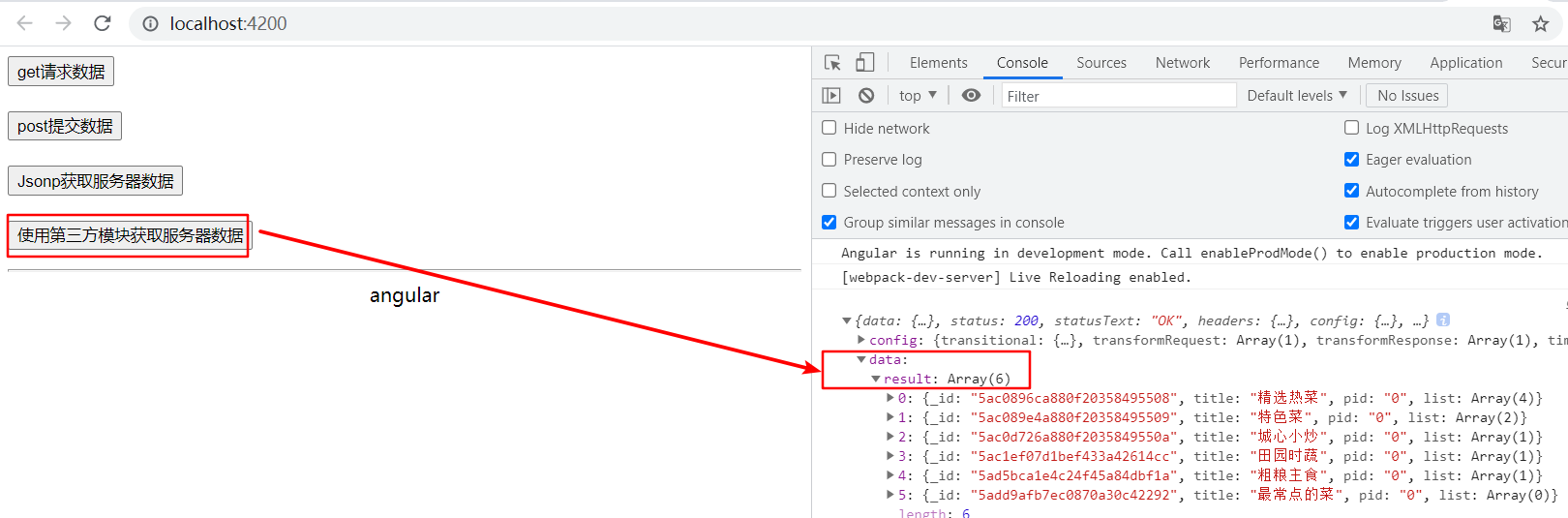
      console.log(data);

    });

  }

7、页面显示

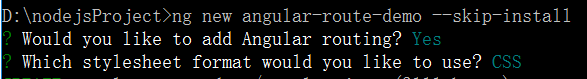
<button (click)="getAxiosData()">使用第三方模块获取服务器数据</button>



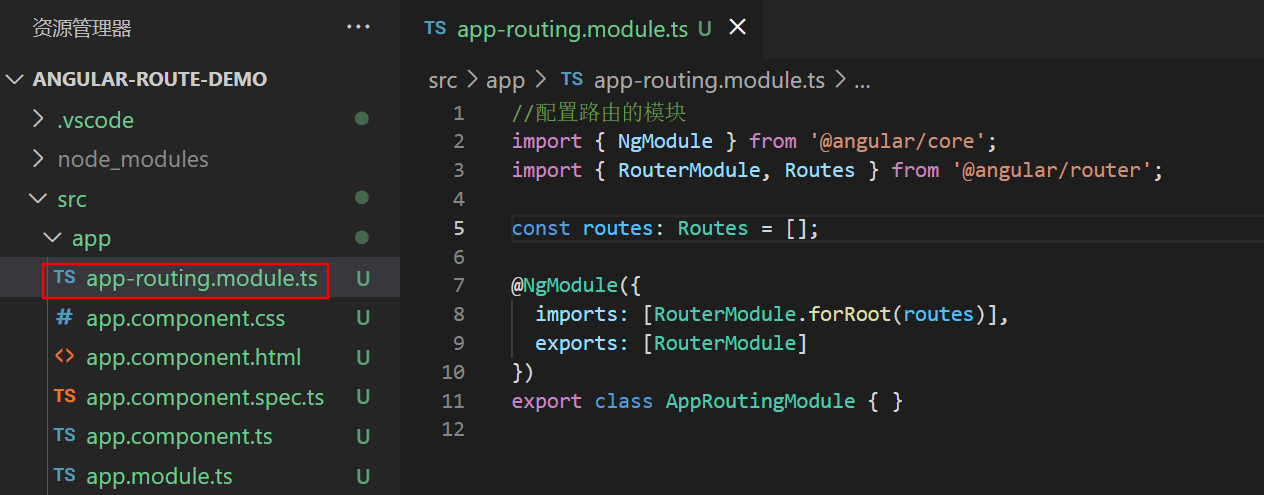
# Angular动态路由

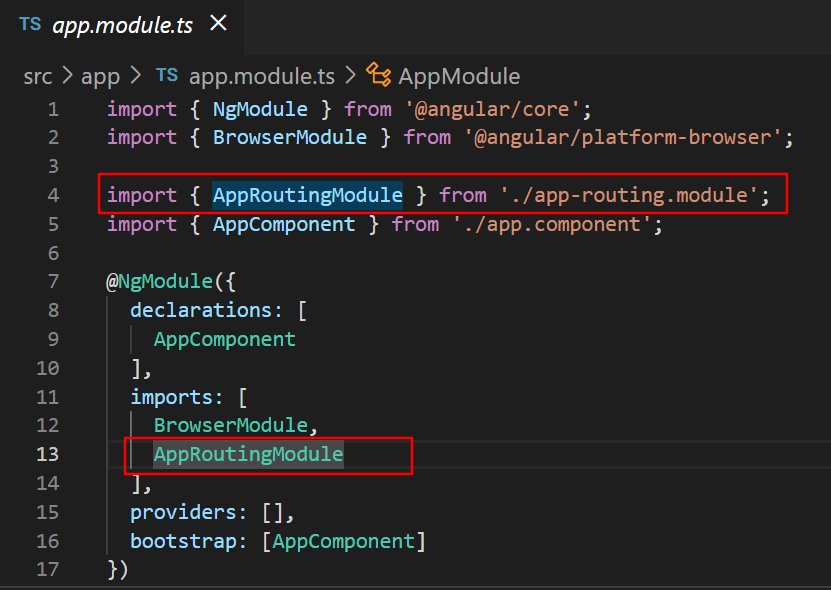
路由就是根据不同的url地址，让根组件动态挂载其他组件来实现一个单页面应用。

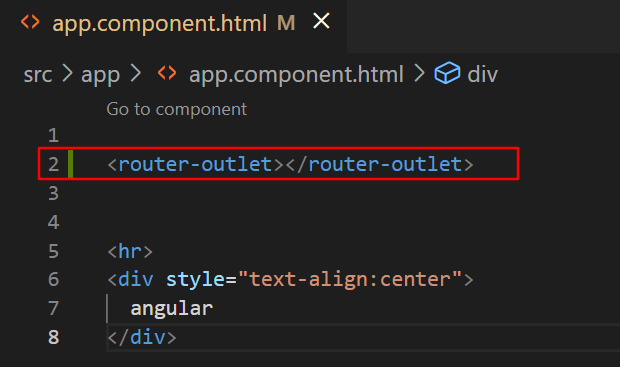
## 创建带路由的项目



与不带路由项目的区别：多了以下内容







## 配置路由

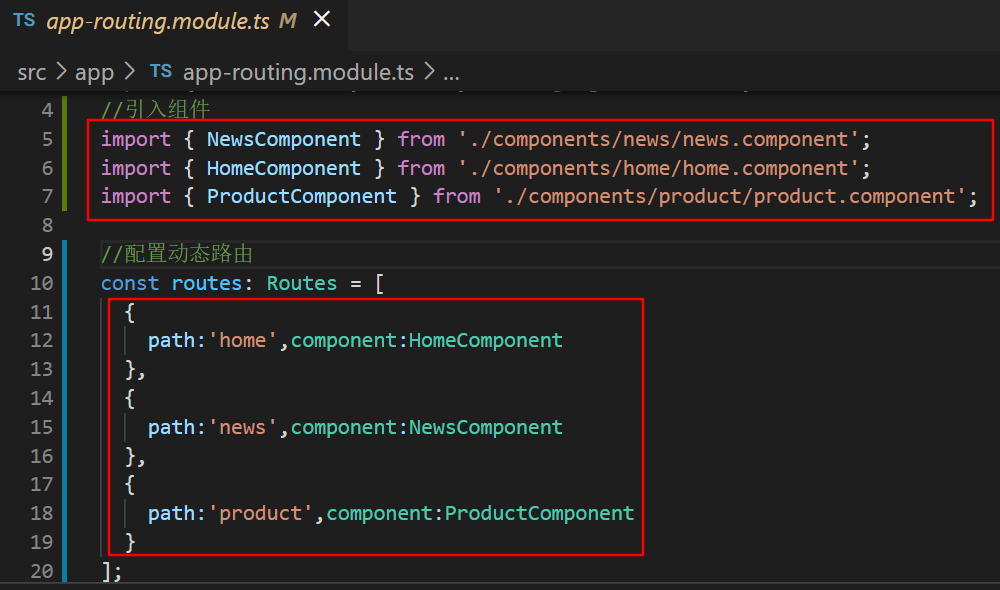
1、创建组件

ng g component components/news

ng g component components/product

ng g component components/home

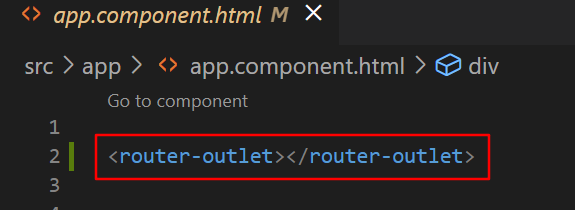
2、在app-routing.module.ts中引入创建的组件



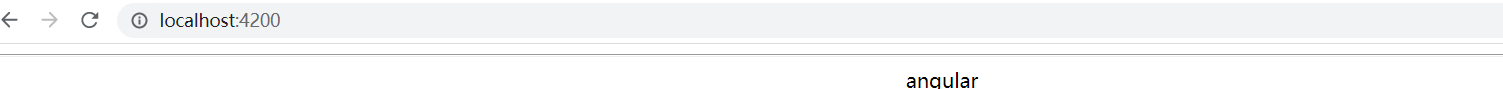
路由的特点：匹配到其中一个时，就不会继续向下匹配。

path指定url，component指定访问哪个组件。

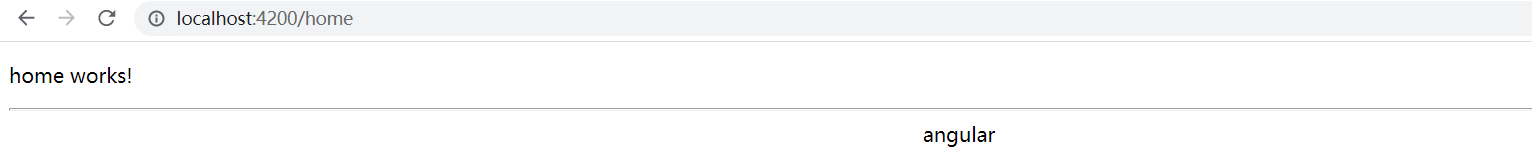
3、在app.component.html中，配置router-outlet显示动态加载的路由。

4、在浏览器url中访问：

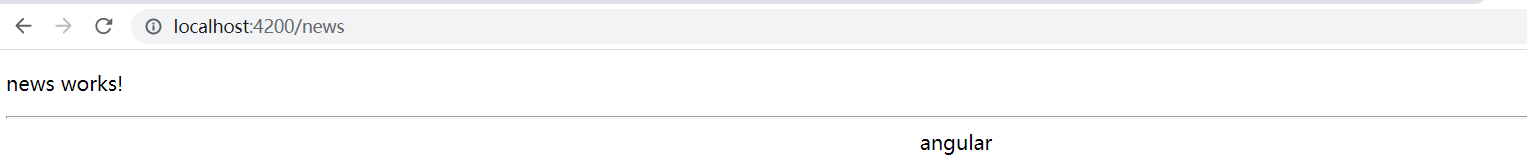
http://localhost:4200



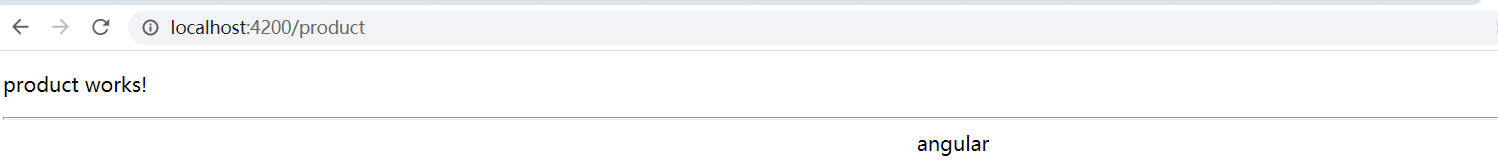
http://localhost:4200/home



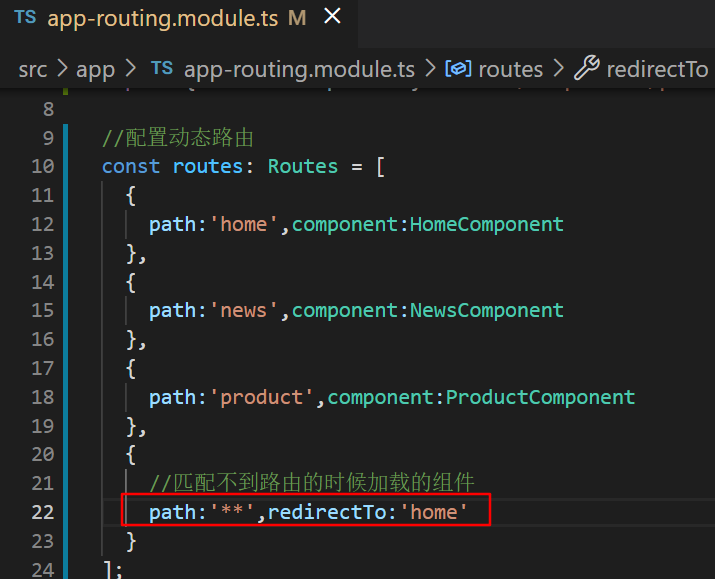
http://localhost:4200/news



<http://localhost:4200/product>



## 默认路由



## 设置默认选中路由

routerLinkActive="active"

<header class="header">

  <a [routerLink]="[ '/home' ]" routerLinkActive="active">首页</a>

  <a [routerLink]="[ '/news' ]" routerLinkActive="active">新闻组件</a>

  <a [routerLink]="[ '/product' ]" routerLinkActive="active">商品页面</a>

</header>

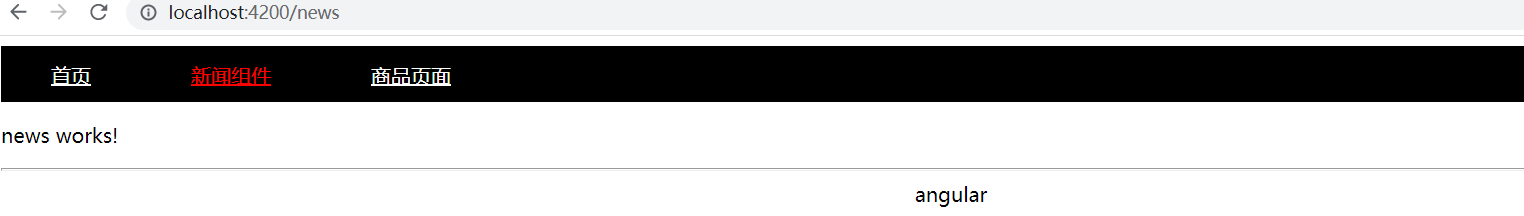
给选中的路由添加样式：

.header .active{

    color: red;

}

效果：

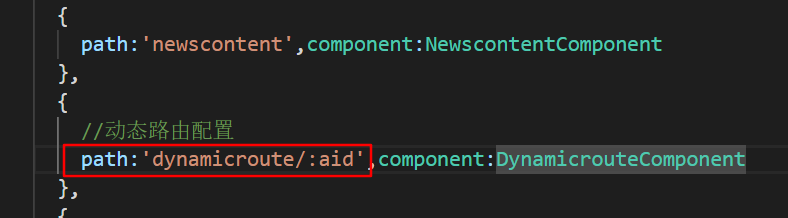


# Angular路由跳转传值

## 动态路由

1、动态路由传值时需要额外配置路由

app-routing.module.ts



2、跳转

news.component.html

<h2>动态路由</h2>

<ul>

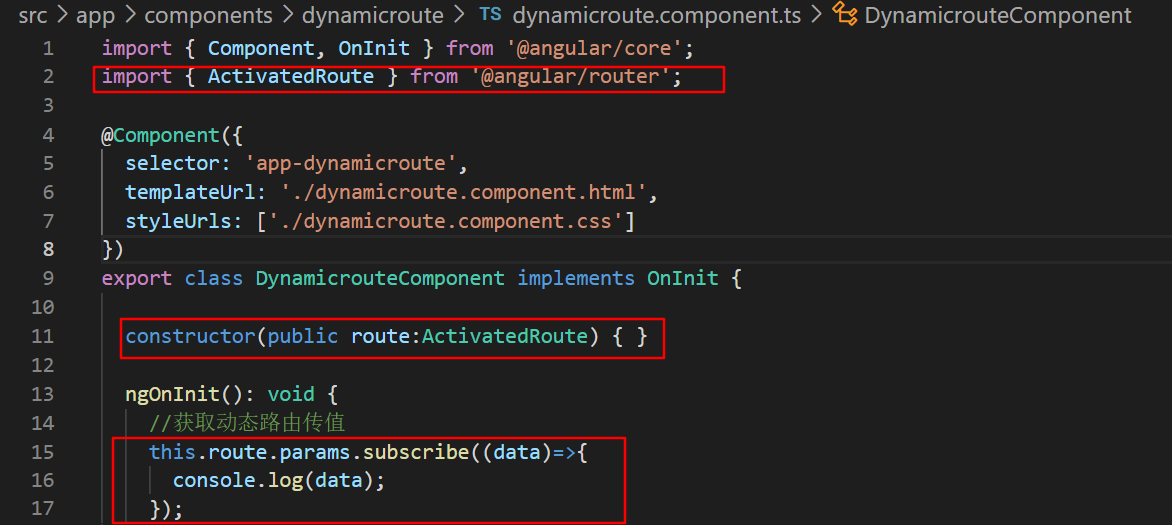
    <li \*ngFor="let item of list; let key=index">

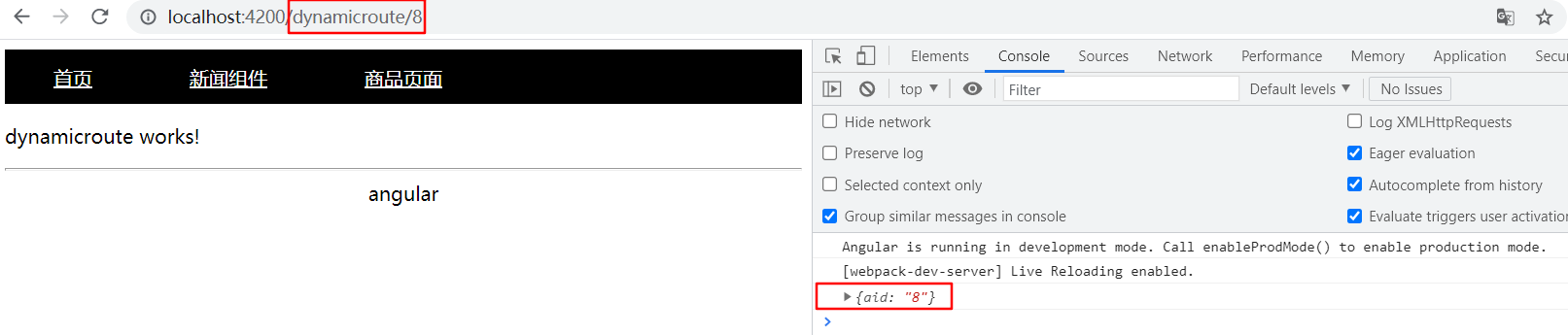
        <a [routerLink]="[ '/dynamicroute', key ]">{{key}} -- {{item}}</a>

    </li>

</ul>

3、dynamicroute.component.ts接收





## get传值

get传值不需要进行额外的路由配置。

news模块传值给newscontent模块

1、跳转

news.component.html

    <li \*ngFor="let item of list;let key=index">

        <a [routerLink]="['/newscontent']" [queryParams]="{aid:key}">{{key}} -- {{item}}</a>

    </li>

2、newscontent接收

newscontent.component.ts引入ActivatedRoute

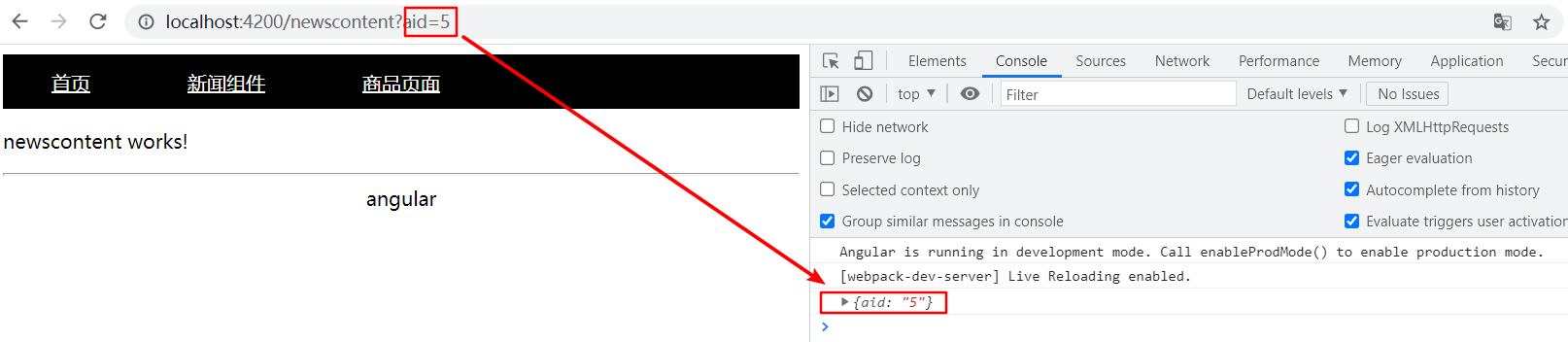
import { ActivatedRoute } from '@angular/router';

获取get传值

this.route.queryParams.subscribe((data)=>{

      console.log(data);

    });



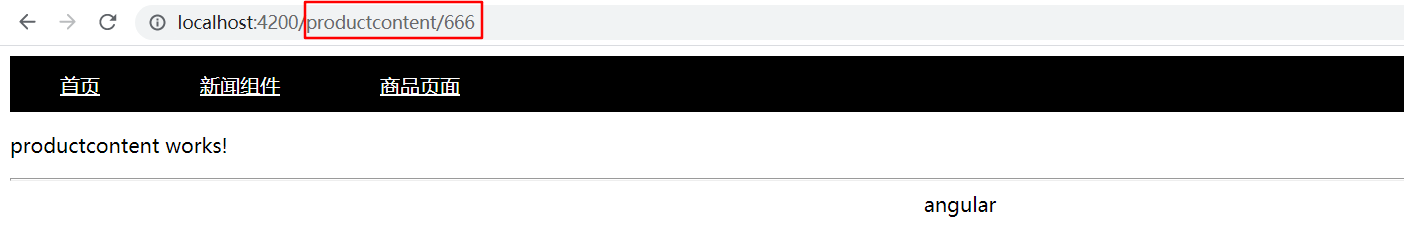
# 动态路由的js跳转

1、引入Router模块



2、页面调用

<button (click)="getProductContent()">js跳转路由</button>



# get传值js跳转

1、引入NavigationExtras模块

import { Router, NavigationExtras } from '@angular/router';

2、跳转

goNews(){

    let queryParam:NavigationExtras={

      queryParams:{'aid':888}

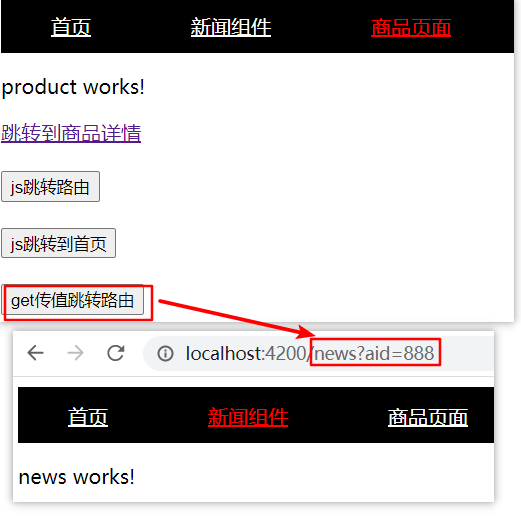
    };

    this.router.navigate(['/news'],queryParam);

  }

3、页面调用

<button (click)="goNews()">get传值跳转路由</button>



# angular grid展示表格

1、创建angular grid项目

ng new angular-ui-grid-demo

2、安装grid依赖

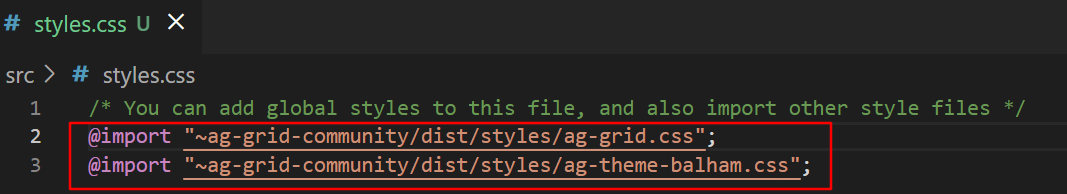
npm install ag-grid-angular --save

npm install ag-grid-community --save

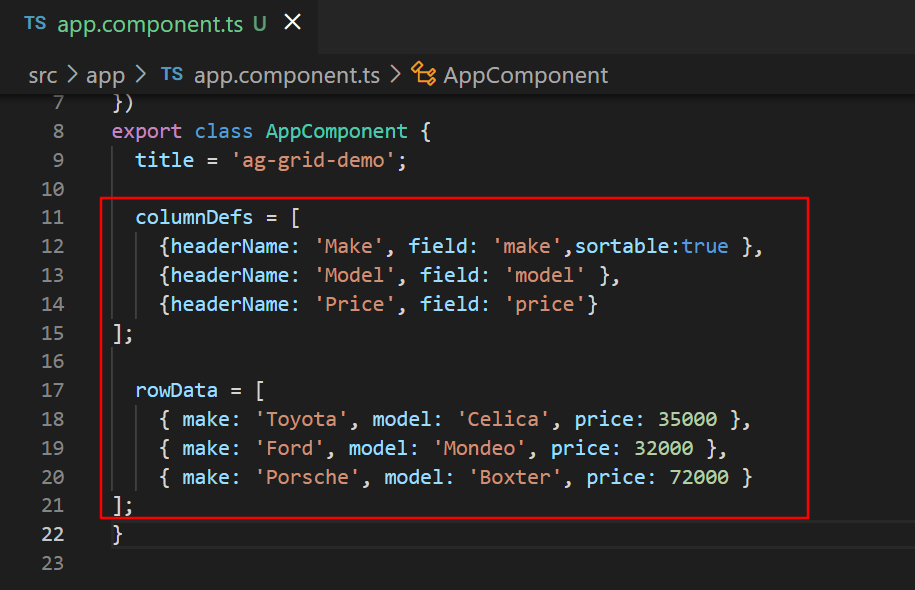
3、修改app.module.ts



4、修改styles.css



5、修改app.component.ts



6、修改app.component.html

<ag-grid-angular

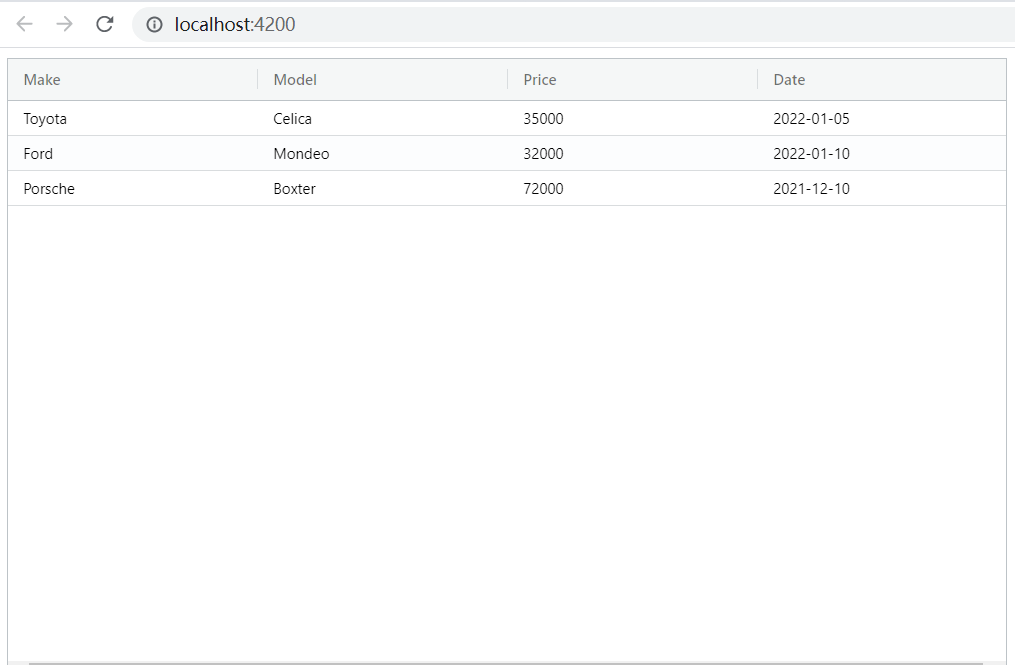
    style="width: 500px;height: 500px;"

    class="ag-theme-balham"

    [rowData]="rowData"

    [columnDefs]="columnDefs"></ag-grid-angular>

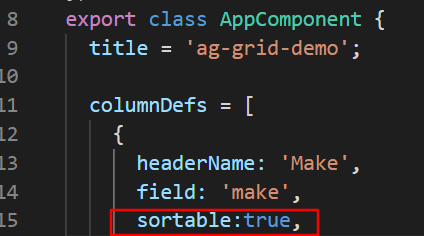
默认表格效果：



## 表格排序

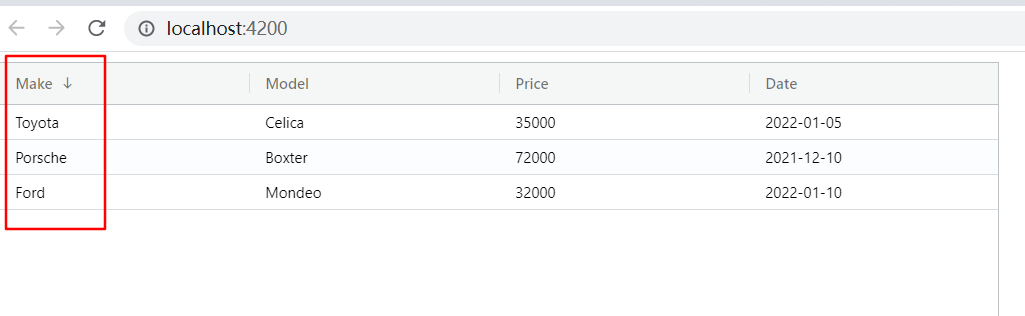
修改app.component.ts，添加

sortable:true,



效果：点击表头可以向上或向下排序





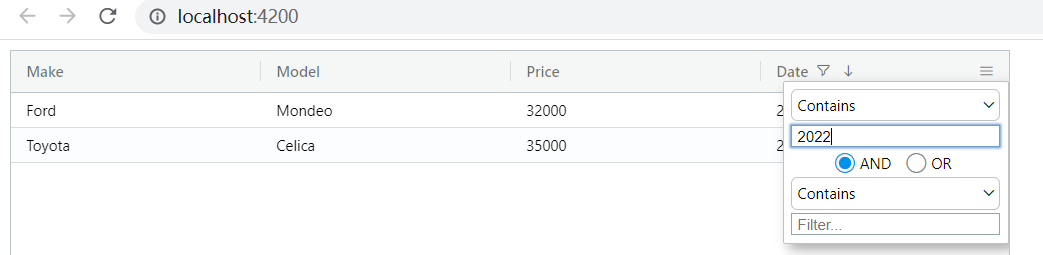
## 条件过滤

修改app.component.ts，添加

filter:true,



效果：



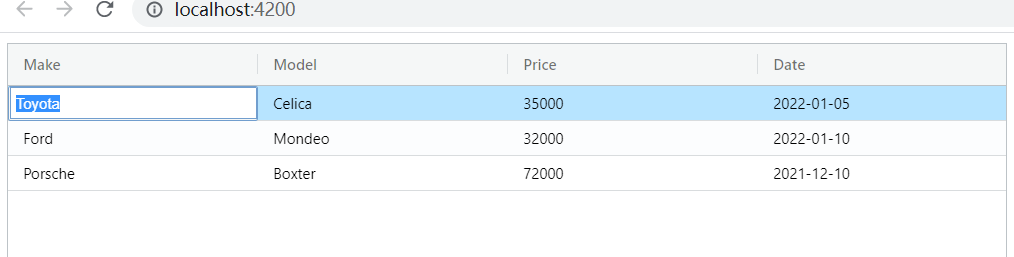
## 表格内容可编辑

修改app.component.ts，添加

editable: true



效果：



## demo链接

angular grid表格demo地址：<https://github.com/Angeliau/angular-ui-grid-demo.git>

# angular使用echarts

安装指定版本的依赖(使用不同版本的angular可能会有版本不兼容问题)

npm install echarts@4.4.0 ngx-echarts@4.2.1 --save

安装合适的依赖：推荐使用以下命令

npm install echarts --save

npm install ngx-echarts --save

## app.module.ts配置

// echarts

import { NgxEchartsModule } from 'ngx-echarts';

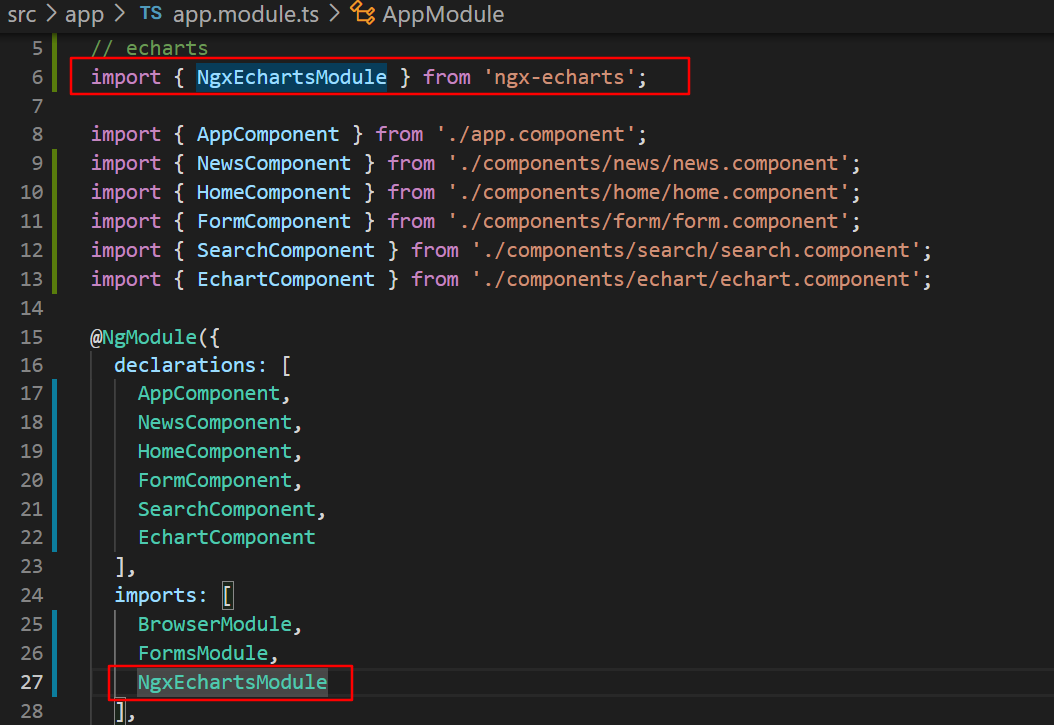
imports: [

    BrowserModule,

    FormsModule,

    NgxEchartsModule

  ],



## echart.component.ts

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

import \* as echarts from "echarts";

@Component({

  selector: 'app-echart',

  templateUrl: './echart.component.html',

  styleUrls: ['./echart.component.css']

})

export class EchartComponent implements OnInit {

  public obj:any={};

  constructor() { }

  ngOnInit(): void {

    this.data();

  }

  data(){

    this.obj={

      title:{

        text:'echarts入门案例',

      },

      tooltip:{},

      legend:{

        data:['销量']

      },

      xAxis:{

        data:['衬衫','羊毛衫','裤子','短袖','皮鞋']

      },

      yAxis:{},

      series:[{

        name:'销量',

        type:'bar',

        data:[5,20,36,28,16]

      }]

    }

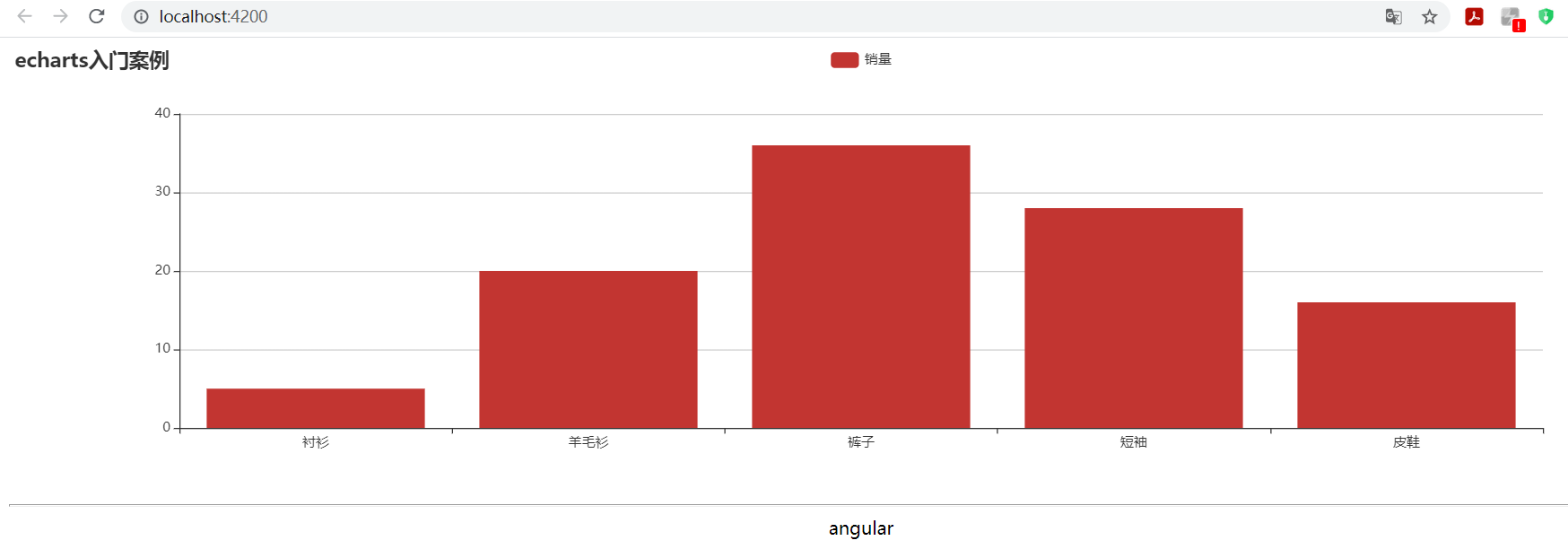
  }

}

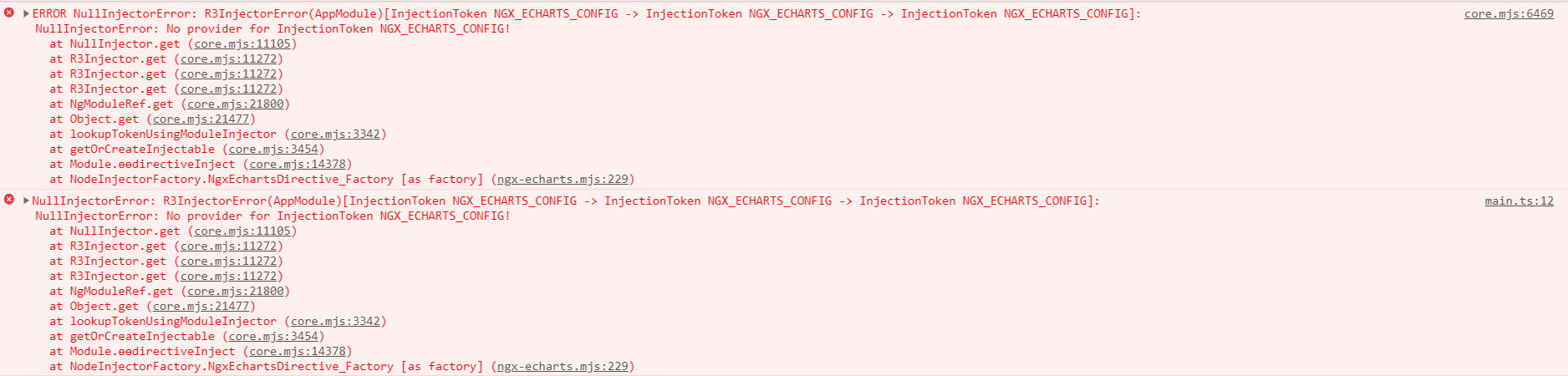
## echart.component.html

<div echarts [options]="obj"></div>

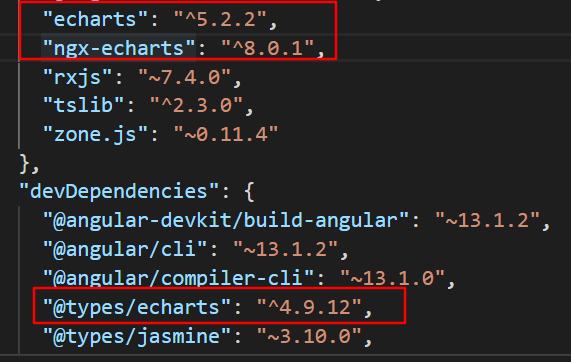
ng serve --open运行：



## echart版本不兼容问题

刚开始配置时，安装的echarts与ngx-echarts的版本不兼容，报错

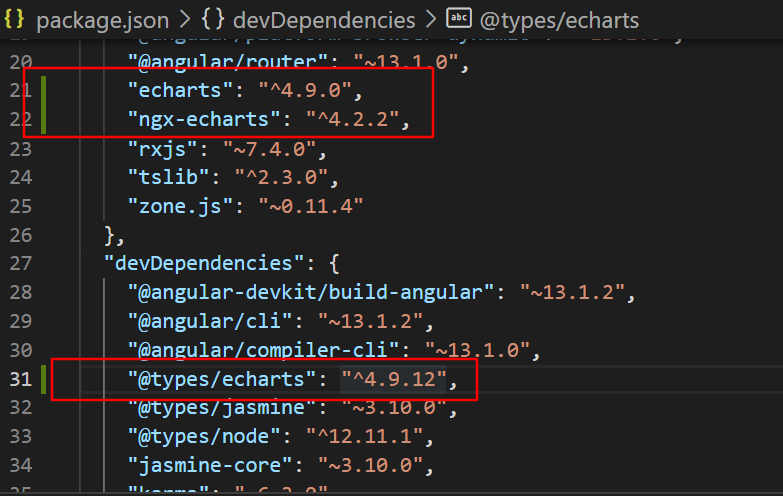
不兼容的版本如下所示：



重新安装合适的依赖

npm install echarts --save

npm install ngx-echarts --save



更换版本后，编译成功。

因此推荐使用

npm install echarts --save

npm install ngx-echarts --save

安装合适的依赖版本。

## demo链接

<https://github.com/Angeliau/angular-demo.git>