前言

基于项目进行实战归纳的angular学习文档，github地址：https://github.com/NK-DMZ/AngularProject。参照b站学习视频，部分地方对照官网参照进行理解性研究。

项目部分截图

angularStudyProject

Angular学习项目

点击到达页尾

是否启用生命周期函数组件

这是生命周期组件里面的信息！

改变msg的值

数据绑定

title属性的数据绑定可以通过花括号和中括号

鼠标移动过来看看

小若同学

字符串模板解析

<h5>这是h5标签</h5>

这是h5标签

这是h5标签

这是h5标签

<h5>这是h5标签</h5>

---

angular模板运行简单运算

1+3=4

数据循环

- 我不是枪神
- 长津湖
- 这个杀手不太冷
- 熊出没
- 不要忘记我爱你

带索引

- 0---我不是枪神
- 1---长津湖
- 2---这个杀手不太冷
- 3---熊出没
- 4---不要忘记我爱你

- 王语嫣---25
- 陆家嘴---35
- 马云---59
- 富途牛---18

- **宝马**

  宝马X3 --- 40

  宝马X5 --- 70

  宝马X6 --- 90

- **比亚迪**

  汉EV --- 22

  海豚 --- 9.6

- **路虎**

  揽胜极光 --- 38.8

  发现 --- 68.9

  神行者2 --- 39.8

  路虎卫士 --- 79.8

图片引入

ng表达式不能出现的

NG表达式中禁止出现new关键字,NG表达式中JSON是undefined

双向数据绑定——MVVM

首先需要进行引入:

1.app.module.ts文件中引入:import { FormsModule } from '@angular/forms';

2.app.module.ts文件中在imports进行声明:添加 FormsModule

这是默认值 这是默认值

双向绑定表单

输入框：

姓名：

男 女

城市： 深圳

爱好： 吃饭 睡觉 敲代码

备注：

获取表单的内容

{ "modelName": "", "sex": "男", "cityList": ["北京", "上海", "深圳"], "city": "深圳", "hobby": [{ "hobbayName": "吃饭", "checked": false }, { "hobbayName": "睡觉", "checked": false }, { "hobbayName": "敲代码", "checked": true }], "mark": "" }

双向数据绑定(姓名)

双向数据绑定(性别) 男

双向数据绑定(城市) 深圳

双向数据绑定(爱好)
[{ "hobbayName": "吃饭", "checked": false }, { "hobbayName": "睡觉", "checked": false }, { "hobbayName": "敲代码", "checked": true }]

双向数据绑定(备注)

条件判断语句

*ngIf图片使用

改变图片显示

---

*ngIf和*ngFor结合使用

1---我不是枪神 2---长津湖 3---这个杀手不太冷 4---熊出没 5---不要忘记我爱你

*ngIf和*ngFor结合使用

已经支付

---

hidden使用

改变hidden显示

hidden的值显示:hidden

ngClass使用

ngClass演示

1---我不是枪神 2---长津湖 3---这个杀手不太冷 4---熊出没 5---不要忘记我爱你

ngStyle使用

ngStyle使用

ngStyle使用

这是原生js操作dom

this is box

这是利用angular进行操作dom

这是box1，在ngAfterViewInit生命周期进行渲染，后面隐藏后没有进行渲染字体颜色

这是利用 ViewChild 实现操作dom

这是一个dom节点

修改box1组件的显示和隐藏

这里是dom操作中的子组件

点击获取子组件的方法

这里是dom操作中的子组件

事件

这是事件数据

点击触发run方法

获取数据方法

修改数据方法

表单事件、事件对象

keydownValue

keydownValue

直接修改属性值 | 获取事件dom

修改颜色

引入其他组件

这是introduce页面!

我是header

管道

知乎别人的解释

date

new Date: Wed Sep 07 2022 15:58:33 GMT+0800 (中国标准时间)

管道写法:

2022-09-07 15:58:33

22-9-7 15:58:33 PM | Wednesday,2022-September-7 | Wed,2022-Sep-7

日期格式

大小写

HI,JACK!

hi,jack!

把数字转换成货币字符串

$55.00

把数字转换成带小数点的字符串

1505---1,505.000

把数字转换成百分比字符串

0.14---14%

通过串联管道应用两种格式

Wed Sep 07 2022 15:58:33 GMT+0800 (中国标准时间) --- SEP 7, 2022

---

add

TodoList

待办事项

已完成事项

父子传值(子组件通过Input修饰器获取父组件的属性与方法)， parent-one 是父组件， child-one 是子组件

子组件ChildOne

首页组件的标题---我是父组件的msg

点击执行父组件的方法

子组件通过Input修饰器获取父组件的属性与方法

这里的title与msg都是父组件中的属性,这么直接传到子组件,run是父组件的方法，home是将父组件作为对象传递过去；

子父传值(父组件利用 ViewChild 实现调用子组件的属性和方法)，news是父组件，footer是子组件

这是父组件news页面

子组件footer

这是子组件footer的内容

获取子组件footer的msg | 获取子组件footer的childTake方法

child-three works!

通过@outer给父组件广播事件

1.首先在子组件引入,Output,EventEmitter
2.然后声明一个outer变量等于EventEmitter
3.通过EventEmitter广播数据
4.在父组件监听广播事件

目前常见的异步编程的几种方法:

1、回调函数

2、事件监听/发布订阅

3、Promise

4、Rxjs

返回数据进行打印，均延迟3秒

同步方法获取数据 | 回调方法获取数据 | Promise方法获取数据 | RxJs 方法获取数据
RxJs 方法获取数据（取消） | RxJs 方法多次调用

Rxjs的工具应用示例

Rxjs的工具过滤器 | Rxjs的工具map | Rxjs的工具组合使用

原生方法获取数据

原生方法获取数据

get使用

1.先在app.module.ts中引入 import { HttpClientModule } from '@angular/common/http'
2.然后在 imports: [
BrowserModule,
AppRoutingModule,
FormsModule,
HttpClientModule
],中加入 HttpClientModule
3.再在primordial.component.ts文件中引入import { HttpClient } from '@angular/common/http';
4.接着在这个文件中constructor中引入： constructor(public http:HttpClient) { }
5.然后在函数getFun中进行使用

post使用

1.先在app.module.ts中引入 import { HttpClientModule } from '@angular/common/http'
2.在imports中引入
3.在primordial.component.ts文件中引入import { HttpClient, HttpHeaders } from '@angular/common/http';
4.函数postFun进行使用

jsonp使用

1.在app.module.ts中引入import { HttpClientJsonpModule } from '@angular/common/http';
2.在imports中引入
3.函数jsonpFun中使用

axios 使用

方案1:
1.安装axios
npm install axios --save
2.用到的地方引入axios
方案2:
1.创建http服务
ng g service services/httpservice
2.在httpservice.service.ts引入
import axios from 'axios'
3.axiosGet函数中使用
4.在app.module.ts引入
import { HttpserviceService } from './services/httpservice.service';
5.进行引入
providers: [HttpserviceService],
6.在primordial.component.ts中引入
import { HttpserviceService } from '../../services/httpservice.service';
7.在constructor中加入
constructor(public http: HttpClient,public httpService:HttpserviceService) { }

点击按钮get获取数据　点击按钮post获取数据　通过jsonp获取数据　通过axios获取数据

get原生使用(微博热榜)

- 湖南涉外 车祸
- 芜湖一中致歉
- 渡难关庐定安
- 教育局回应家长拒绝送礼被逼退群
- 上海一女子去优衣库给全家偷衣服
- 张翰东八区的先生们开分2.4
- 三亚00后老板中秋扣员工50%工资代尽孝
- 国庆10月1日至7日放假调休
- 李小冉说庆余年2剧本还没出来
- 苹果将捐款支持四川救援和重建

路由使用步骤

1.创建组件
2.在app.module.ts中挂载
3.在app-routing.module.ts中进行挂载
Routes中配置,有两个属性path和component
4.匹配不到路由的时候加载组件
path: '**', redirectTo: 'routeOne'
5.选中时激活css样式
routerLinkActive="active"

路由1　路由2　路由3　路由传值父组件　JS路由传值父组件

router-value-child works!

aid是1

这里的代码是:"aidobj['aid']"
错误写法是： "objtest.aid"
报错提示:
Property 'aid' does not exist on type '{ }'.ngtsc(2339)

父组件get传值子组件

1.父组件部分传值***(看router-value-child.component.ts文件,这里被转成HTML了)***

- {{key}}-{{item}}

2.子组件接收
2.1 在子组件先进行引入
import { ActivatedRoute } from '@angular/router';
2.2 constructor中引入
constructor(public route: ActivatedRoute) { }
2.3 ngOnInit中使用
this.route.queryParams.subscribe((data) => {
console.log(data);
});

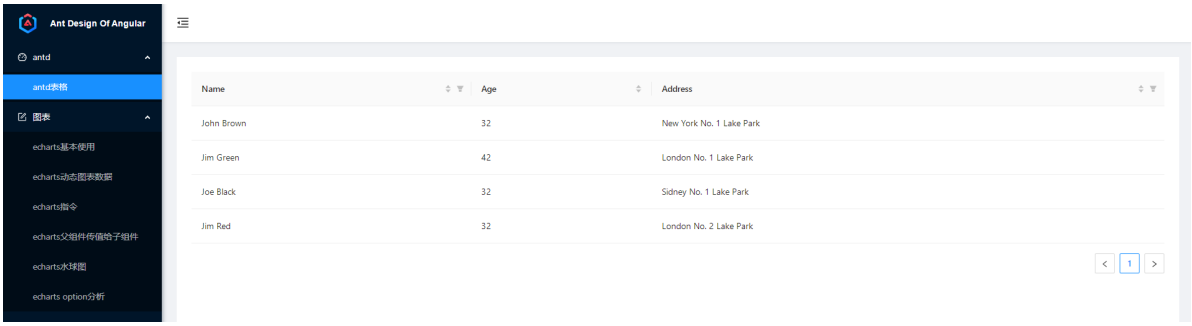
此文件夹下的组件用于演示路由嵌套

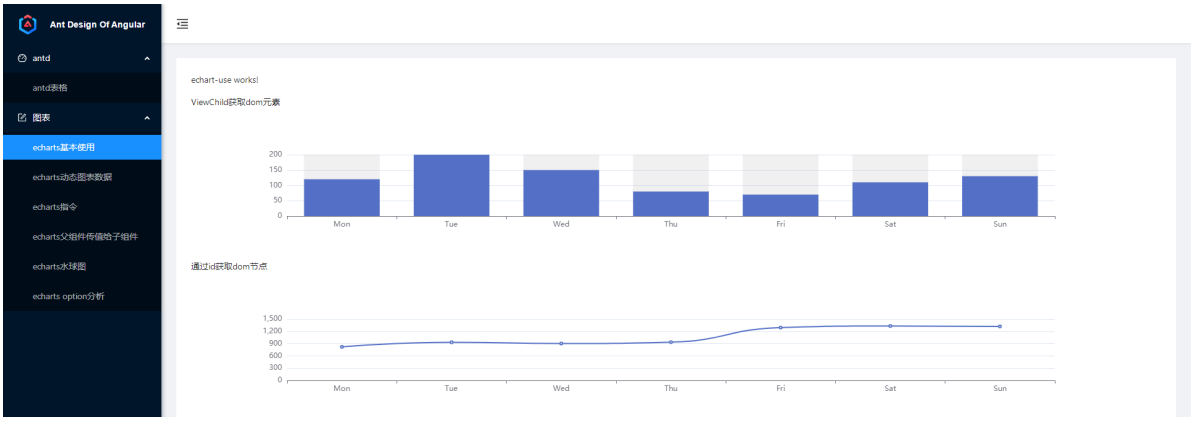
nested-route works!

新闻　商品

页尾部分

angularAntdEcharts





一.快速上手

官方文档

https://angular.cn/ angular官网
https://ng.ant.design/docs/introduce/zh ng-zorro官网
https://www.tslang.cn/index.html ts官网
https://yarnpkg.com/getting-started/migration yarn官网

项目基础架构

项目根目录下创建三个主要文件夹：

- `public`：用于存放项目的静态资源
- `scripts`：用于存放 webpack 的配置文件
- `src`：用于存放项目的代码文件
  为了区分开 webpack 的开发和生产环境，因此需要两套配置文件，这两套配置有很多地方是共通的，为了代码优雅，可以使用第三方包 `webpack-merge` 来将公共配置分别导入两套文件，因此需要在 `scripts` 目录下创建三个文件：
- `webpack.common.js`：用于编写公共配置
- `webpack.dev.js`：用于编写开发环境配置
- `webpack.prod.js`：用于编写生产环境配置

区分环境

虽然都分开了配置，但是在公共配置中，还是可能会出现某个配置的某个选项在开发环境和生产环境中采用不同的配置，这个时候有两种选择：

- 分别在 dev 和 prod 配置文件中写一遍，common 中就不写了

- 设置某个环境变量，根据这个环境变量来判别不同环境
  为了代码优雅性选择第二种方案，下载所需第三方包：

> `cross-env`：统一配置Node环境变量

```
yarn add cross-env@7.03
```

> 不同操作系统设置环境变量的方式不一定相同，cross-env可以将其统一，比如Mac 电脑上使用 export NODE_ENV=development，而Windows 电脑上使用的是 set NODE_ENV=development
> 在 `scripts/config` 目录下新建 `env.js` 文件用于管理启动环境
> 在node中，全局变量 process 表示的是当前的node进程，process.env 包含着关于系统环境的信息，NODE_ENV 是用户一个自定义的变量，该变量会在下面配置启动命令时配上，这里先写上

```
const isDevelopment = process.env.NODE_ENV === 'development'
const isProduction = process.env.NODE_ENV === 'production'
module.exports = {  isDevelopment,  isProduction}
```

管理公共常量

在 `scripts` 目录下新建 `constant.js` 文件，用于统一管理公共常量
首先先写公共项目根路径和启动端口及IP

```
// scripts/constant.js
const path = require('path')
const PROJECT_PATH = path.resolve(__dirname, '../')   // 项目根路径
const SERVER_HOST = '127.0.0.1'
const SERVER_PORT = 3000
export {
  PROJECT_PATH,
  SERVER_HOST,
  SERVER_PORT,
}
```

webpack 配置的路径一般要求绝对路径写法，所以项目中往往会如下这样配置，可读性不佳

```
module.exports = {
  entry: {
    app: path.resolve(__dirname, '../../src/index.js')
  },
  ...
}
```

使用配置的常量则可以如下配置

```
module.exports = {
  entry: {
     app: path.resolve(PROJECT_PATH, './src/index.tsx'),
  },
  ...
}
```

创建新文件

Component(组件):

npm run ng g component ./core/my-new-component

Directive(指令):

npm run ng g directive ./core/my-new-directive
Pipe(管道):

npm run ng g pipe ./core/my-new-pipe
Service(服务):

npm run ng g service ./core/my-new-service
Class(类):

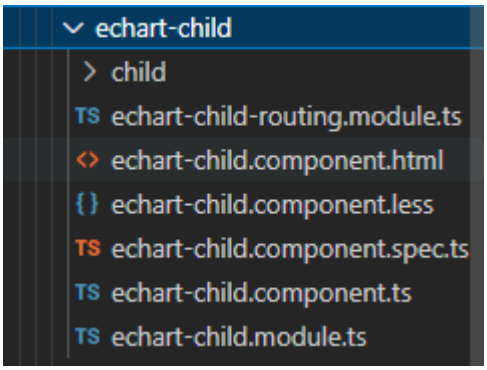
npm run ng g class ./core/my-new-class
Interface(接口):

npm run ng g interface ./core/my-new-interface
Enum(枚举):

npm run ng g enum ./core/my-new-enum
Module(模块):

npm run ng g module ./core/my-module

这里说明一下，路由module没有创建指令，目前在study项目中创建的一个完整的组件包含组件（4个文件）、模块（1个文件）、路由模块（1个文件）。如图示：



其中.html是写页面的，.less是写样式的，.ts是写页面逻辑的，-routing.module.ts是写路由的，module是写模块引入的，.spec.ts是测试文件。

二.生命周期函数

官网详细说明可以查阅 https://angular.cn/guide/lifecycle-hooks
Angular 会按以下顺序调用钩子方法：

序号	钩子方法	详细信息
1	ngOnChanges	当输入或输出绑定值更改时。
2	ngOnInit	在第一个 ngOnChanges 之后。
3	ngDoCheck	开发人员的自定义变更检测。
4	ngAfterContentInit	组件内容初始化后。
5	ngAfterContentChecked	在每次检查组件内容之后。
6	ngAfterViewInit	在组件的视图被初始化之后。
7	ngAfterViewChecked	在每次检查组件视图之后。
8	ngOnDestroy	就在指令被销毁之

上表如果不够直观可以看看下面项目中的输出展示，当我们启动页面时，生命周期依次执行。

如果我们修改页面数据再来看看执行了哪些生命周期函数：

由程序执行可知ngDoCheck 方法、ngAfterContentChecked 方法、ngAfterViewChecked 方法在数值改变时会被执行。

使用比较频繁的函数有：

ngOnInit函数，用于数据请求；ngAfterViewInit函数，常用于echarts实例初始化；ngOnDestroy函数，用于指令中echarts实例的销毁。

三.数据使用

标题 [title]

```
public student: string = "student!";

<div [title]="student">
  <p>小茗同学</p>
</div>
```

展示的效果：当鼠标悬停在div上时会有提示，提示内容为"student!"。

字符串模板解析 [innerHTML]

当我们想把字符串解析的时候一般会想到下面这种使用方法:

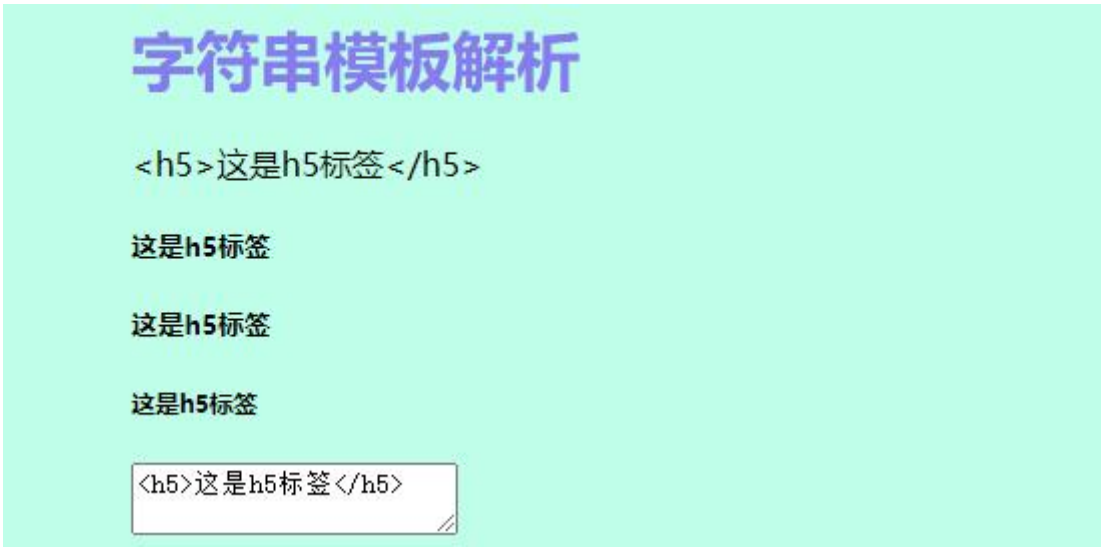
```
public htmlContent: string = "<h5>这是h5标签</h5>";

<div>
  {{htmlContent}}
</div>
```

但是这种使用方法达不到我们所需要的效果。这时我们可以使用[innerHTML]对html元素进行解析。

```
<span [innerHTML]="htmlContent"></span>
<p [innerHTML]="htmlContent"></p>
<h5 [innerHTML]="htmlContent"></h5>
<textarea [innerHTML]="htmlContent"></textarea>
```

由图可见如果仅仅只是通过{{}}语法是无法让浏览器解析出html模板的，我们无论是用p标签、span标签还是h5标签都会解析成变量中的h5标签，而textarea解析的会带有自己的格式。



图片引入 [src]

```
public imgUrl: string =
'http://bpic.588ku.com/back_pic/03/52/10/03579335225368f.jpg!/fh/300/quality/90/
unsharp/true/compress/true'

<img [src]="imgUrl" alt="">
```

双向数据绑定 [(ngModel)]

1.app.module.ts文件中引入:

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

2.app.module.ts文件中在imports进行声明:添加 FormsModule
3.html使用

```
<input type="text" [(ngModel)] = 'keywords'> {{keywords}}
```

class类 [ngClass]

[ngClass]="{'red':(key===0),'grey':(key===1),'orange':(key===2),'lawngreen':(key===3),'sienna':(key===4)}"
'red'是类名，后面的(key===0)其实就是TRUE和FALSE。

```
<ul *ngFor="let item of arrlist;let key = index;" [ngClass]="{'red':
(key===0),'grey':(key===1),'orange':(key===2),'lawngreen':(key===3),'sienna':
(key===4)}">
    <li>{{key+1}}---{{item}}</li>
</ul>
```

style样式 [ngStyle]

效果和class类似

```
<p [ngStyle]="{'color':attrColor}">ngStyle使用</p>

下面代码不会生效
style = "'color':red"
<p [ngStyle]="{style}">ngStyle使用</p>
```

*ngFor数据循环

*ngFor用于数组的循环

```
// 定义数组
public arr: string[] = [
  '我不是枪神',
  '长津湖',
  '这个杀手不太冷',
  '熊出没',
  '不要忘记我爱你',
];

<ul>
  <p>带索引</p>
  <li *ngFor="let item of arr;let key = index" class="ulli">{{key}}---{{item}}
</li>
</ul>
```

带索引

- 0---我不是枪神
- 1---长津湖
- 2---这个杀手不太冷
- 3---熊出没
- 4---不要忘记我爱你

*ngIf和[hidden]选择性展示

ngif与hidden的区别和v-if与v-show的区别一样，一个是直接让dom节点消失，一个是让dom隐藏。

ngif使用

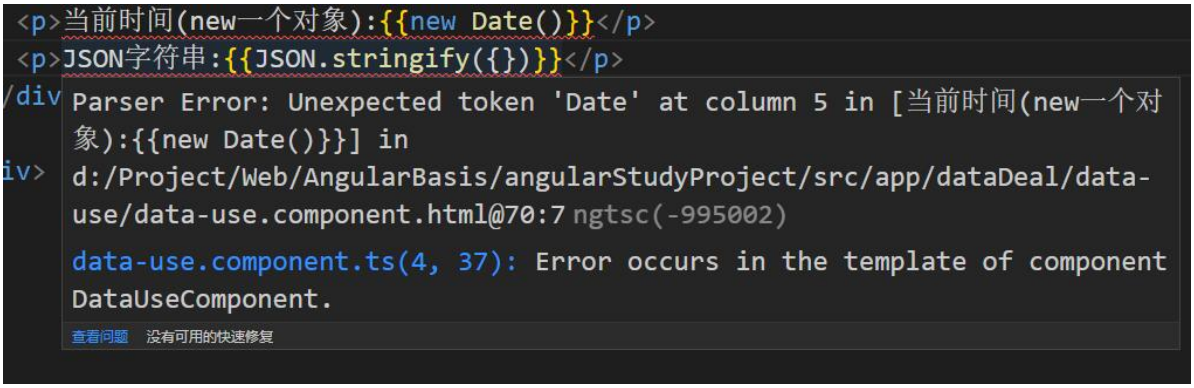
```
<button (click)="changeFlag()">改变图片显示</button> <br><br>
<img *ngIf="flag" src="../../../assets/images/boolean (1).png" alt="">
<img *ngIf="!flag" src="../../../assets/images/boolean (2).png" alt="">
```

[hidden]使用

```
<button (click)="changeHidden()">改变hidden显示</button> <br><br>
<p [hidden]="hidden">hidden</p>
```

ng表达式不能出现的

NG表达式中禁止出现new关键字,NG表达式中JSON是undefined



四.DOM操作

原生js操作dom

```
<div id="domHtml">
  这是原生JavaScript对dom进行操作
</div>

let oBox = document.getElementById('domHtml');
console.log(oBox.innerHTML);
oBox.style.color = "red"
```

这是原生JavaScript对dom进行操作

被angular操作过的DOM元素

无法在ngOnInit生命周期获取

```
let oBox1: any = document.getElementById('box1');
console.log(oBox1.innerHTML);
oBox.style.color = "blue"
```

可以在ngAfterViewInit生命周期正常使用

```
let oBox1: any = document.getElementById('box1');
console.log(`这是domAction中的oBox1:${oBox1.innerHTML}`);
oBox1.style.color = "blue"
```

利用 ViewChild 实现操作dom

1.引入 ViewChild

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

2.在html页面使用

```
<div #myBox>
  <h3>这是利用 ViewChild 实现操作dom</h3>
  这是一个dom节点
</div>
```

3.获取dom节点

```
@ViewChild('myBox') myBox: any;
```

4.利用ViewChild进行操作

```
this.myBox.nativeElement.style.width = '300px'
this.myBox.nativeElement.style.height = '80px'
this.myBox.nativeElement.style.background = 'yellow'
```

通过ViewChild操作子组件

1.html使用

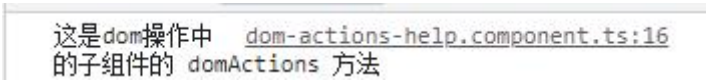
```
<app-dom-actions-help #domActionChild>
  <h3>这是操作子组件</h3>
</app-dom-actions-help>
<button (click)="getClick()">点击获取子组件的方法</button>
```

2.获取dom节点

```
@ViewChild('domActionChild') domActionChild: any;
```

3.调用子组件的方法

```
getClick() {
  this.domActionChild.domActions()
}
```



五.事件

普通事件

html使用

```
<p><button (click)="run()">点击触发run方法</button></p>
```

ts中调用

```
run(){
  alert('事件run方法被触发')
}
```

表单事件与事件对象

html使用

```
<p>{{keydownValueOne}}</p>
<p>{{keydownValueTwo}}</p>
<input type="text" (keydown)="keydown()" placeholder="直接修改属性值">
<input type="text" (keyup)="keyupEvent($event)" placeholder="获取事件dom">
```

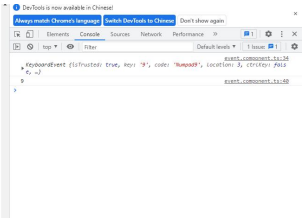
ts中调用

```
keydown(){
  this.keydownValueOne='keydownValue进行修改'
}
keyupEvent(e:any){
  console.log(e);
  if(e.keyCode == 13){
    console.log('按了一下回车');
  }else{
    // e.target 获取dom对象
    // 获取对象的值
    console.log(e.target.value);
  }
}
```

函数调用前

在第一个输入框修改值会触发keydown函数

在第二个对话框修改内容会触发keyupEvent函数，能清晰打印出e.target



通过事件对象获取dom修改样式

html使用

```
<p><button (click)="changeColor($event)">修改颜色</button></p>
```

ts中调用

```
changeColor(event:any){
  let dom:any = event.target;
  dom.style.color = 'blue'
}
```

六.组件引用

angularAntdEcharts项目对这一知识点有详细说明

七.管道

管道的作用

用于在模板表单式中对给定的数据进行转换和格式化，并输出处理后的数据，如：转换日期格式、货币格式、数字精度、过滤列表数据等。

在模板中使用“|”符号来使用管道，转换参数是以“:”符号接在管道名称之后，如果由多个参数，可以顺序添加。如果需要同时使用多个管道处理数据可以按照转换顺序用“|”符号在尾部继续添加，
如：

```
data | myCustomPipe: params | myCustomPipe: params | myCustomPipe
```

管道中的数据变更检测方式

管道会在每次dom事件之后（鼠标移动、计时器、服务器响应、按键）对绑定的需要转换的数据的变更进行检测，而且针对基础数据类型和应用类型的检测规则有所不同。

- a、针对基础数据类型的检测：对String\Number\Boolean\null\undefined\Symbol的变更，也称之为纯变更，采用的是纯管道（给定输入会返回固定的输出），会直接重新执行管道并返回转换结果
- b、针对引用数据类型的检测：对Object\Function\Date\Array\Map\Set等的变更，也称之为非纯变更，采用的是非纯管道（即需要对数据进行深层次的检索来确定是否变更）
- 注意：在使用纯管道来处理引用类型的数据时，可以采取返回一个全新的数据对象来激活纯管道的变更检测机制，而非只更改数据内部的给别属性或者添加内容

管道的使用

详细使用

日期格式

日期

代码:

```
public pipesTime:any = new Date();

<span style="color: burlywood;">new Date:</span> {{pipesTime}}
{{pipesTime | date:'yyyy-MM-dd  HH:mm:ss'}} <br>
{{pipesTime | date:'yy-M-d  H:m:s a'}}
<span style="margin:0  50px;">|</span>
{{pipesTime | date:'EEEE,y-MMMM-d'}}
<span style="margin:0  50px;">|</span>
{{pipesTime | date:'EEE,y-MMM-d'}}  {{pipesTime | date:'yyyy-MM-dd  HH:mm:ss'}}
<br>
{{pipesTime | date:'yy-M-d  H:m:s a'}}
<span style="margin:0  50px;">|</span>
{{pipesTime | date:'EEEE,y-MMMM-d'}}
<span style="margin:0  50px;">|</span>
{{pipesTime | date:'EEE,y-MMM-d'}}
```

效果:

大小写

```
public pipesString:string = 'hi,Jack!'

<p>{{pipesString | uppercase }}</p>
<p>{{pipesString | lowercase }}</p>
```

把数字转换成货币字符串

```
public pipesMoney:number = 55
<p>{{pipesMoney | currency }}</p>
```

把数字转换成带小数点的字符串

```
public pipesNum:number = 1505
<p>{{pipesNum}}---{{pipesNum | number:'1.3-5' }}</p>
```

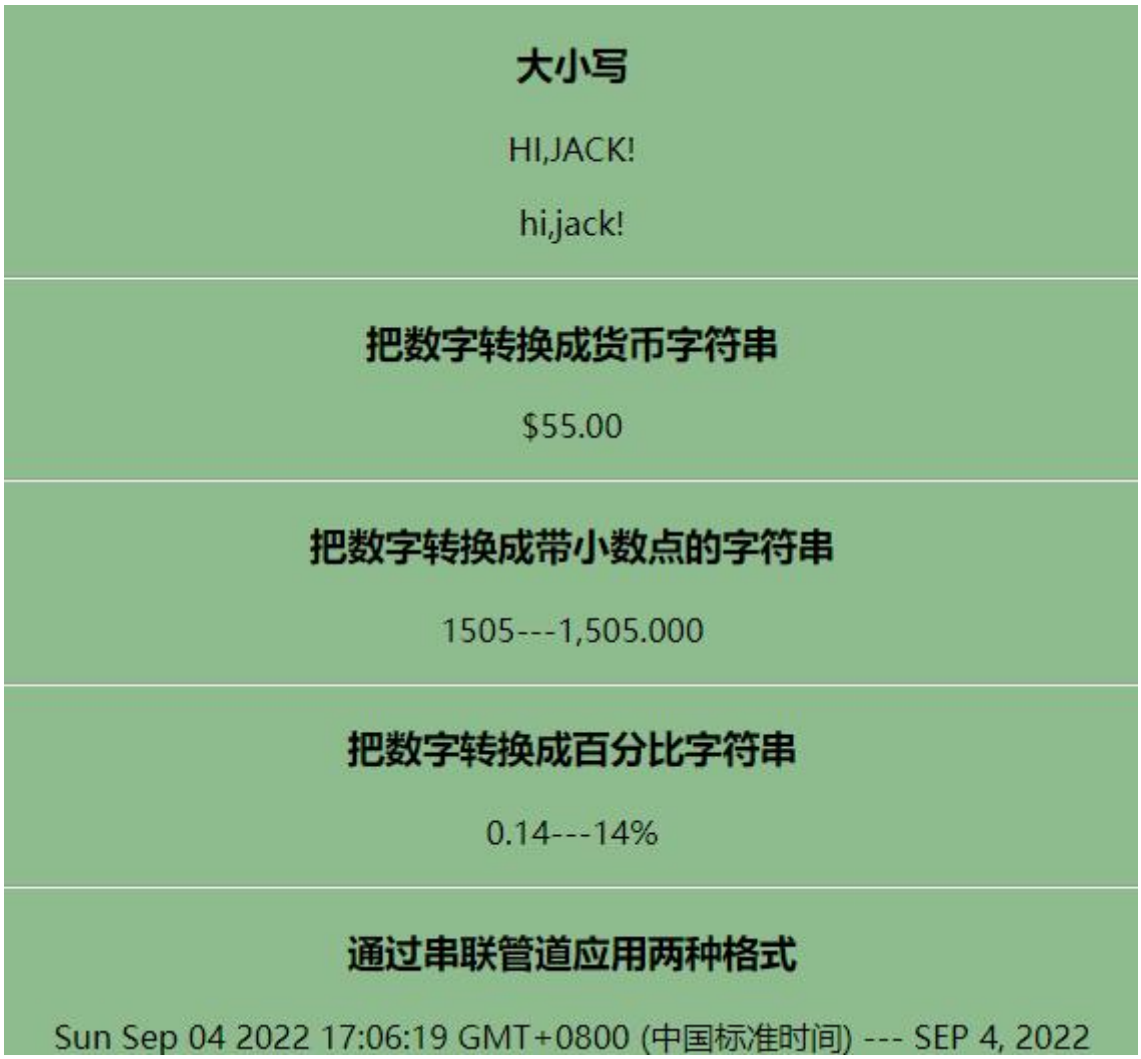
把数字转换成百分比字符串

```
public pipesTrans:number = 0.14
<p>{{pipesTrans}}---{{pipesTrans | percent }}</p>
```

通过串联管道应用两种格式

```
public pipesTrans:number = 0.14
<p>{{pipesTrans}}---{{pipesTrans | percent }}</p>
```

运行结果



八.组件传值

子组件通过Input修饰器获取父组件的属性与方法

1.在父组件html使用子组件，并用[]=""进行传值

```
父组件html

<h2>父子传值(子组件通过Input修饰器获取父组件的属性与方法)， parent-one 是父组件， child-one 是子组件</h2>
<app-child-one [title]="title" [msg]="msg" [run]="run" [parent_One]="this"></app-child-one>
<p>
  这里的title与msg都是父组件中的属性,这么直接传到子组件,run是父组件的方法，home是将父组件作为对象传递过去;
</p>
```

2.子组件通过使用input接受父组件的传值，需要先import引入再进行使用，可以在ts文件进行加工，也可以直接在子组件html中直接使用。对其进行操作可以在ngAfterViewInit()生命周期函数。

```
子组件ts文件
import { Component, OnInit, Input  } from '@angular/core';

// 接收父组件传过来的值
@Input() title:any;
@Input() msg:any;
@Input() run:any
@Input() parent_One:any
childRun():void{
  this.run();
  console.log('传递home对象进行操作');
  this.parent_One.work()
}
```

- 父组件核心语句在于 []=""
- 子组件核心语句在于 @Input()

子组件向父组件传值ViewChild

父组件利用 ViewChild 实现调用子组件的属性和方法。
1.父组件html中使用子组件，在子组件中通过 # 获取元素属性

```
父组件html页面
<h2>子父传值(父组件利用 ViewChild 实现调用子组件的属性和方法)，news是父组件，footer是子组件
</h2>
<app-child-two #footer></app-child-two>
<button (click)="getChildMsg()">获取子组件footer的msg</button>
<button (click)="getChildTake()">获取子组件footer的childTake方法</button>
```

2.父组件ts文件中需要引入ViewChild获取子组件。

```
父组件ts文件
@ViewChild('footer') footer: any;
getChildMsg() {
  // 获取footer子组件的数据
  console.log(this.footer.childMsg);
}

getChildTake() {
  this.footer.childTake()
}
```

```
子组件html页面
<div class="childTwo">
    <h2>子组件footer</h2>
    <p>{{childMsg}}</p>
</div>
```

```
子组件ts文件
public childMsg:string = "这是子组件footer的内容"

childTake(){
  console.log('这是子组件footer的childTake方法');
}
```

子组件传值给父组件Output()+EventEmitter

1.首先在子组件引入,Output,EventEmitter

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

2.然后声明一个outer变量等于EventEmitter

```
@Output() private outer = new EventEmitter();
```

定义子组件其他使用的内容

```
public msChildThree = "这是第三种事件驱动传值的子组件"

fun1(){
  console.log('这是fun1方法');
}

sendParent(){
  // this.outer.emit('这是子组件的数据')
  this.outer.emit(this.msChildThree)
}
```

3.通过EventEmitter广播数据

```
子组件html页面
<button (click)="sendParent()">通过@outer给父组件广播事件</button>
```

4.在父组件监听广播事件

```
父组件html
<app-child-three (outer)="funChild($event)"></app-child-three>

父组件ts文件
funChild(e:any){
  // 子组件给父组件广播的数据
  console.log(e);
  console.log('这是父组件中的方法，e是子组件广播给父组件的数据');
}
```

九.异步编程

同步方法获取数据

函数直接调用就是同步方法获取数据

```
<button (click)="synchronization()">同步方法获取数据</button>

synchronization() {
  // 1. 同步方法
  let data = this.request.getData();
  console.log(data);
}

getData(){
  return "Request中的数据进行获取！"
}
```

1、回调函数

Callback函数调用请求；
getCallbackData函数通过延时函数模拟异步请求，获取数据后回传给函数。

```
<button (click)="Callback()">回调方法获取数据</button>

Callback() {
  // 2.callback通过回调方法获取异步数据
  this.request.getCallbackData((data: any) => {
    console.log(data);
  })
}

getCallbackData(cb:any){
  setTimeout(() => {
    let userName:string = "回调方法";
    cb(userName);
  }, 3000);
```

```
}
```

2、事件监听/发布订阅

这一部分在前面讲父子传值的时候有说明过，这里不详细说明。

```
```

3、Promise

通过promise这种方法很常见，不详细说明。

```
<button (click)="promiseFun()">Promise方法获取数据</button>

promiseFun() {
  // 3.Promise获取异步数据
  let promiseData = this.request.getPromiseData();
  promiseData.then((data) => {
    console.log(data);
  })
}

getPromiseData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      let msg = "Promise函数"
      resolve(msg);
    }, 3000);
  })
}
```

4、Rxjs

rxjs可以像promise一样使用，使用点注意：

- 1.rxjdata.subscribe(()=>{})获取数据方式
- 2.需要new一个Observable对象
- 3.observer.next(rxjsname)继续执行
- 4.还需要进行引入


```
<button (click)="rxjsFun()">RxJs 方法获取数据</button>

import { Observable } from 'rxjs';
```

```
// 4.rxjs处理异步
rxjsFun() {
  let rxjdata = this.request.getRxjsData();
  rxjdata.subscribe(data => {
    console.log(data);
  })
}

getRxjsData(){
  return new Observable(observer =>{
    setTimeout(() => {
      let rxjsname = "Rxjs"
      observer.next(rxjsname)
    }, 3000);
  })
}
```

Promise的创建之后，动作是无法撤回的。Observable不一样，动作可以通过unsbscribe()方法中途撤回，而且 Observable在内部做了智能的处理。这里的关键语句是 recallVar.unsubscribe();这行语句取消了获取数据。


```
<button (click)="recall()">RxJs 方法获取数据</button>

recall() {
  console.log('rxjs撤回函数执行，不会打印获取的数据');
  let rxjdata = this.request.getRxjsData();
  let recallVar = rxjdata.subscribe(data => {
    console.log(data);
  })
  setTimeout(() => {
    recallVar.unsubscribe();
  }, 1000);
}
```

这里的连续调用执行其实和执行单词没有太大区别，只是调用的函数不一样，setTimeout只执行一次，setInterval可以一直执行。


```
<button (click)="rxjsCount()">RxJs 方法多次调用</button>

rxjsCount() {
  let rxjscountSum = this.request.getRxjsCount();
  rxjscountSum.subscribe(data =>{
    console.log(data);
  })
}
```

```
getRxjsCount() {
  let count = 0;
  return new Observable<any>(observable => {
    setInterval(() => {
      count++;
      let msg = 'count连续执行' + count;
      observable.next(msg);
    }, 1500)
  })
}
```

十.数据获取

get使用

- 1.先在app.module.ts中引入

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

- 2.然后在

```
 imports: [
        BrowserModule,
        AppRoutingModule,
        FormsModule,
        HttpClientModule
],
```

中加入 HttpClientModule

- 3.再在primordial.component.ts文件中引入

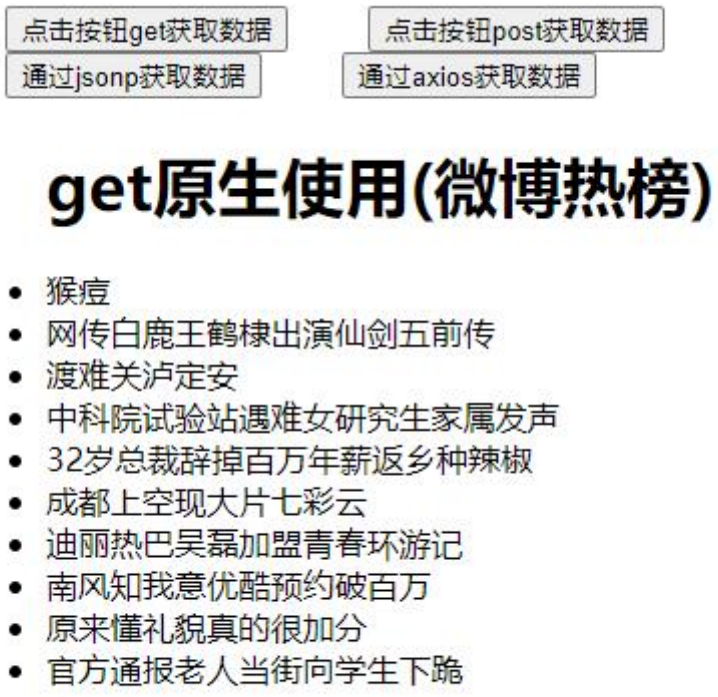
```
import { HttpClient } from '@angular/common/http';
```

- 4.接着在这个文件中constructor中引入：

```
constructor(public http:HttpClient) { }
```

- 5.然后在函数getFun中进行使用

```
getFun() {
  let api = "https://tenapi.cn/resou/"
  this.http.get(api).subscribe((response: any) => {
    this.get_list_subscribe = response.list;
    this.title = "get原生使用(微博热榜)"
    this.post_resText = "";
    this.jsonp_resText = "";
    this.axios_list = [];
  })
}
```



post使用

- 1.先在app.module.ts中引入

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

- 2.在imports中引入,方法同上，如果用get已经引入这里就不需要操作
- 3.在primordial.component.ts文件中引入

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

- 4.函数postFun进行使用

```
postFun() {
  const httpOptions = { headers: new HttpHeaders({ 'Content-Type':
'application/json' }) }
  let api = 'http://127.0.0.1/postuse'
  this.http.post(api, { "url": "baidu.com" }, httpOptions).subscribe((res: any)
=> {
    this.post_resText = res.msg;
    this.title = "post原生使用"
    this.get_list_subscribe = [];
    this.jsonp_resText = "";
    this.axios_list = [];
  })
}
```

jsonp使用

- 1.在app.module.ts中引入

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

- 2.在imports中引入
- 3.函数jsonpFun中使用

```
jsonpFun() {
  // jsonp请求，服务器必须支持jsonp
  let api = "http://127.0.0.1:3000/getscript"
  this.http.jsonp(api, 'callback').subscribe((res) => {
    this.jsonp_resText = res
    this.title = "jsonp原生使用"
    this.get_list_subscribe = [];
    this.post_resText = "";
    this.axios_list = [];
  })
}
```

axios使用

安装axios

```
npm install axios --save
```

方案1:

- 1.创建http服务

```
ng g service services/httpservice
```

- 2.在httpservice.service.ts引入

```
import axios from 'axios'
```

- 3.axiosGet函数中使用

这里的使用和promise在其他语言中的使用差不多

```
axiosGet(api: any) {
  return new Promise((resolve, reject) => {
    axios.get(api).then(function (response) {
      resolve(response)
    }).catch(function (error) {
      console.log(error);
    })
  })
}
```

- 4.在app.module.ts引入

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

- 5.在app.module.ts中的providers进行导入
  `providers: [HttpserviceService],`
- 6.在primordial.component.ts（使用axios的组件）中引入

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

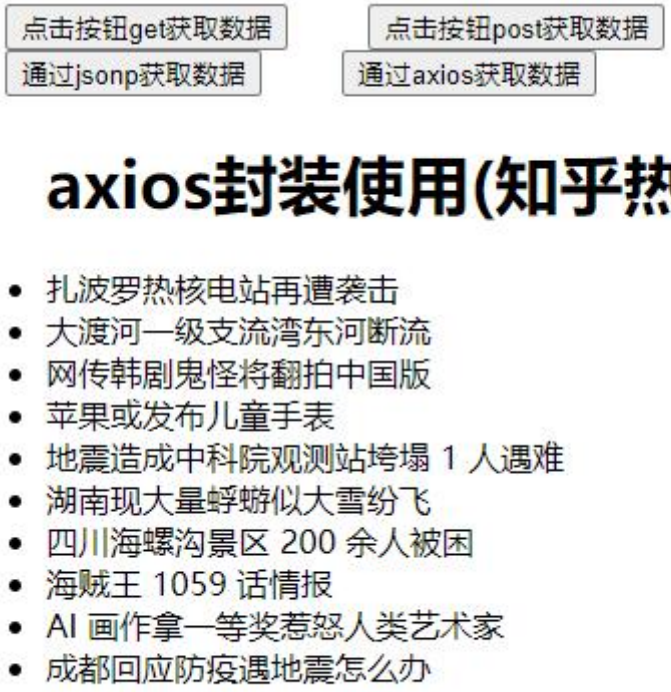
- 7.在constructor中加入

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

- 8.primordial.component.ts在文件中进行使用

先定义api接口，然后使用，获取数据后进行修改页面内容

```
axiosFunPackaging() {
  let api = "https://tenapi.cn/zhihuresou/"
  this.httpService.axiosGet(api).then((res: any) => {
    this.axios_list = res.data.list;
    this.title = "axios封装使用(知乎热榜)"
    this.get_list_subscribe = [];
    this.post_resText = "";
    this.jsonp_resText = "";
  })
}
```



方案2:

- 在用到的地方直接引入axios，大概步骤可以参照方案1。

十一.路由使用

使用步骤

- 1.创建组件
- 2.在app.module.ts中挂载
- 3.在app-routing.module.ts中进行挂载
  Routes中配置,有两个属性path和component
- 4.匹配不到路由的时候加载组件

```
path: '**', redirectTo: 'routeOne'
```

- 5.选中时激活css样式
  routerLinkActive="active"

一般情况使用

这里拿app-routing.module.ts文件举例，前面是引入import，重点在于routes，`{ path: '', pathMatch: 'full', redirectTo: '/app' }`,这行代码的含义是没有匹配到路由的话跳转到/app路由上。后一行代码的含义是在跳转到/app路由上加载AppComponent组件。

`{ path: '', pathMatch: 'full', redirectTo: '/app' }`等同于`{ path: '**', redirectTo: 'app'}`

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AppComponent } from './app.component'
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: '/app' },
  { path: 'app', component: AppComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

嵌套路由

这里的核心是children路由数组。
当选择到news路由后，不进行路由选择默认跳转到newone路由，加载NewoneComponent组件，选择newtwo路由会跳转到NewtwoComponent组件。

```
{
  path: 'news', component: NewsComponent,
  children: [
    { path: 'newone', component: NewoneComponent },
    { path: 'newtwo', component: NewtwoComponent },
    // 未匹配路由时默认跳转
    { path: '**', redirectTo: 'newone' }
  ]
},
```

路由传值父组件

父组件get传值子组件

1.父组件部分传值

```
<ul>
    <li *ngFor="let item of list;let key = index">
        <a [routerLink]="['/routeValueChildDynamic/']" [queryParams]="{aid:key}">
{{key}}-{{item}}</a>
        <!-- 也可以跳转路由，但无法传值 -->
        <!-- <a [routerLink]="['/routeValueChildDynamic/',key]">{{key}}-{{item}}
</a> -->
    </li>
</ul>
```

2.子组件接收

- 2.1 在子组件先进行引入

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

- 2.2 constructor中引入

```
constructor(public route: ActivatedRoute) { }
```

- 2.3 ngOnInit中使用

```
this.route.queryParams.subscribe((data) => {
  console.log(data);
});
```

动态路由传值

需要在路由文件【app-routing.module.ts】进行配置

1.配置动态路由(在router-value-parent.component.html)文件

2.子组件接收
2.1 在子组件先进行引入

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

2.2 constructor中引入

```
  constructor(public route: ActivatedRoute) { }
```

2.3 ngOnInit中使用

```
  this.route.queryParams.subscribe((data) => {
    console.log(data);
  });
```

十二.angular使用antd和echarts

使用 antd 框架

ng add ng-zorro-antd

-yes

-yes

zh_cn

sidemenu

antd 使用样例 ———— table

创建 antd 模块

```
ng g module antd/table
```

创建 antd 模块下的组件

```
ng g component antd/table --module=app
```

在 table.module.ts 引入并 import

NzTableModule是antd的表格组件，TableComponent是自身html组件。

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { TableRoutingModule } from './table-routing.module';
import { TableComponent } from './table.component';
import { NzTableModule } from 'ng-zorro-antd/table';
```

在 table.component.ts 中定义数据

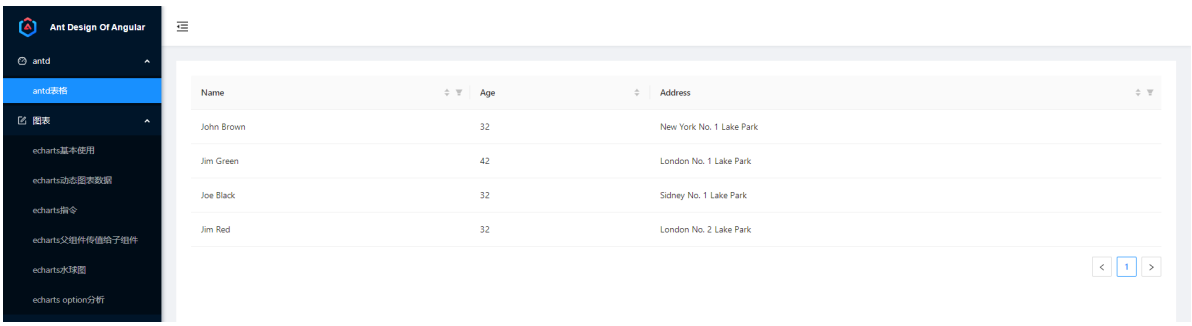
```
interface Person {
  key: string;
  name: string;
  age: number;
  address: string;
}

listOfData: Person[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sidney No. 1 Lake Park',
  },
];
```

在 table.component.html 使用

```
<nz-table #basicTable [nzData]="listOfData">
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>Address</th>
      <th>Action</th>
    </tr>
```

```
  </thead>
  <tbody>
    <tr *ngFor="let data of basicTable.data">
      <td>{{ data.name }}</td>
      <td>{{ data.age }}</td>
      <td>{{ data.address }}</td>
      <td>
        <a>Action 一 {{ data.name }}</a>
        <a>Delete</a>
      </td>
    </tr>
  </tbody>
</nz-table>
```



使用 echarts

```
yarn add echarts
```

- 1.html中用ViewChild获取dom元素
  这里一定要给元素定义宽高。
  `<div #echrts id="echrtsView"></div>`
- 2.在所使用的组件中引用
  ectarts-test.component.ts 文件中

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { ECharts, init } from 'echarts';
```

- 3.ts文件中接收获取的dom
  `@ViewChild('echrts') echrts: any;`
- 4.定义图表对象
  `echartsEcharts!: ECharts;`
- 5.定义 EChartsOption 对象

```
echartsEChartsOption = {
  xAxis: {
    type: 'category',
    data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
  },
  yAxis: {
    type: 'value'
  },
  series: [
    {
      data: [120, 200, 150, 80, 70, 110, 130],
      type: 'bar',
```

```
      showBackground: true,
      backgroundStyle: {
        color: 'rgba(180, 180, 180, 0.2)'
      }
    }
  ]
};
```

- 6.在生命周期中进行使用
  freshEcharts函数是对它进行数据更新，看实际情况决定是否使用。

```
ngAfterViewInit(): void {
  this.echartsEcharts = init(this.echrts.nativeElement);
  this.echartsEcharts.setOption(this.echartsEChartsOption);
  // this.freshEcharts()
}
```

- 7.更新数据

```
freshEcharts() {
  // 修改数据
  let arr = [];
  for (let index = 0; index < 3; index++) {
    arr.push(index);
  }
  // 修改数据

  this.echartsEChartsOption.series[0].data = arr;
  this.echartsEcharts.setOption(this.echartsEChartsOption);
}
```