# Angular安装配置

安装node环境

安装npm环境

安装angular脚手架： npm install -g @angular/cli

查看安装是否成功：Ng version

创建项目：ng new 项目名称

打开项目： ng serve --open

创建组件时，指定module文件 ：ng g c componentName --module=app.module

创建有路由的项目 ： ng new 项目名称 --routing --defaults

# 组件

创建： ng generate component pages/welcome文件名称/子文件名称 （不需要可以不写子文件名称）



组件中定义的模板





Angular 组件需要一个用 template 或 templateUrl 定义的模板。但你不能在组件中同时拥有这两个语句。

## 组件的生命周期

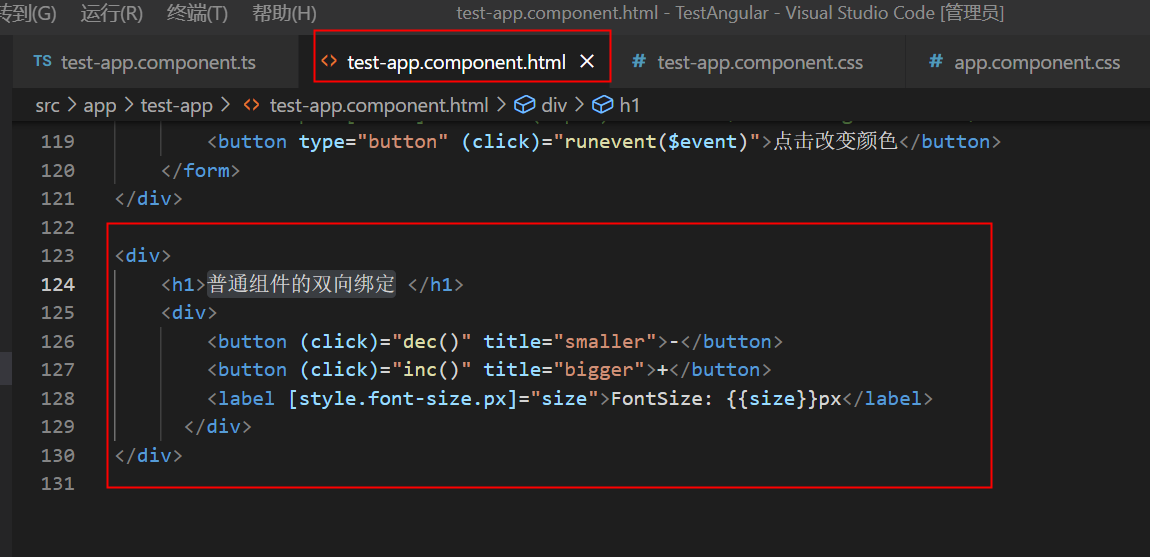
|  |  |  |
| --- | --- | --- |
| 方法 | 用途 | 时机 |
| ngOnChanges() | 当 Angular 设置或重新设置数据绑定的输入属性时响应。该方法接受当前和上一属性值的 SimpleChanges 对象  父子组件传参，数据改变是 |  |
| ngOnInit() | 在 Angular 第一次显示数据绑定和设置指令/组件的输入属性之后，初始化指令/组件  请求数据 | 在第一轮 ngOnChanges() 完成之后调用，只调用**一次**。而且即使没有调用过 ngOnChanges()，也仍然会调用 ngOnInit()（比如当模板中没有绑定任何输入属性时） |
| ngDoCheck() | 检测，并在发生 Angular 无法或不愿意自己检测的变化时作出反应  自定义操作 | 紧跟在每次执行变更检测时的 ngOnChanges() 和 首次执行变更检测时的 ngOnInit() 后调用 |
| ngAfterContentInit() | 当 Angular 把外部内容投影进组件视图或指令所在的视图之后调用。 | ngAfterContentInit() 和每次 ngDoCheck() 之后调用。 |
| ngAfterContentChecked() | 每当 Angular 检查完被投影到组件或指令中的内容之后调用 |  |
| ngAfterViewInit() | 当 Angular 初始化完组件视图及其子视图或包含该指令的视图之后调用 进行dom操作时调用 | 第一次 ngAfterContentChecked() 之后调用，只调用一次。 |
| ngAfterViewChecked() | 每当 Angular 做完组件视图和子视图或包含该指令的视图的变更检测之后调用。 | ngAfterViewInit() 和每次 ngAfterContentChecked() 之后调用 |
| ngOnDestroy() | 每当 Angular 每次销毁指令/组件之前调用并清扫。在这儿反订阅可观察对象和分离事件处理器，以防内存泄漏 | 在 Angular 销毁指令或组件之前立即调用。 |

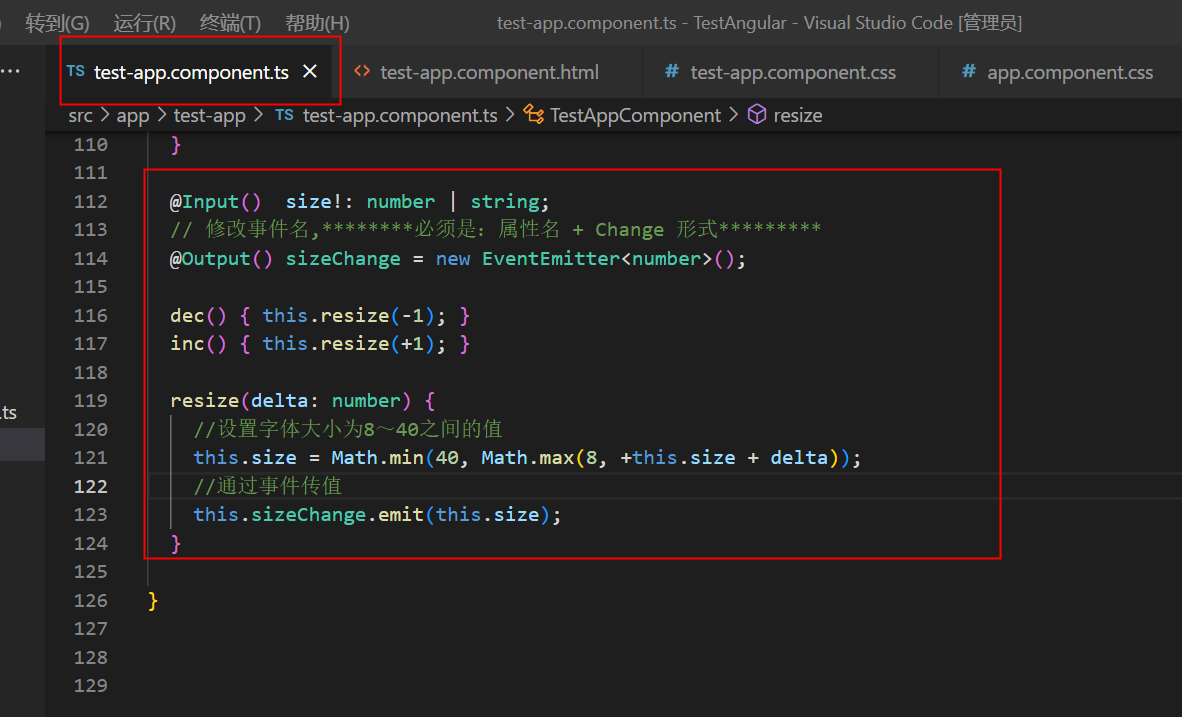
# 双向数据绑定

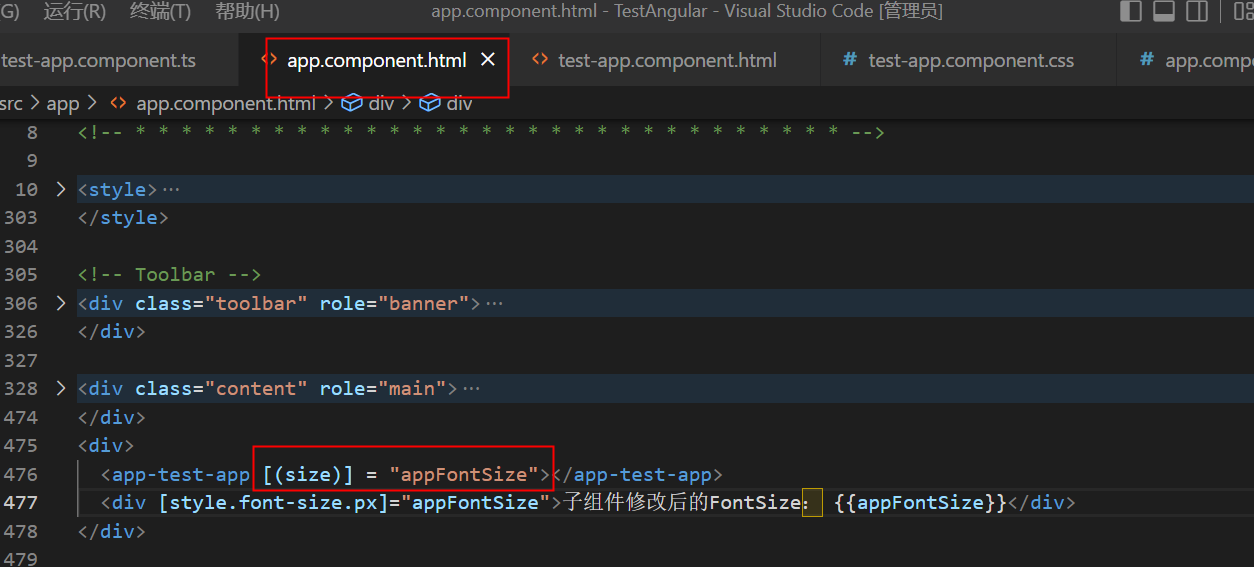
## 一．普通组件的双向数据绑定

在子组件的模块中创建两个按钮来绑定事件来改变文字的大小，然后在组件中size 属性是 @Input()，因此数据可以流入 TestAppComponent。sizeChange 事件是一个 @Output()，它允许数据从 TestAppComponent流出到父组件，接下来，有两个方法，dec() 用于减小字体大小，inc() 用于增大字体大小。这两种方法使用 resize() 在最小/最大值的约束内更改 size 属性的值，并发出带有新 size 值的事件。

在父组件中进行初始变量的声明，父模块中进行属性绑定，事件绑定

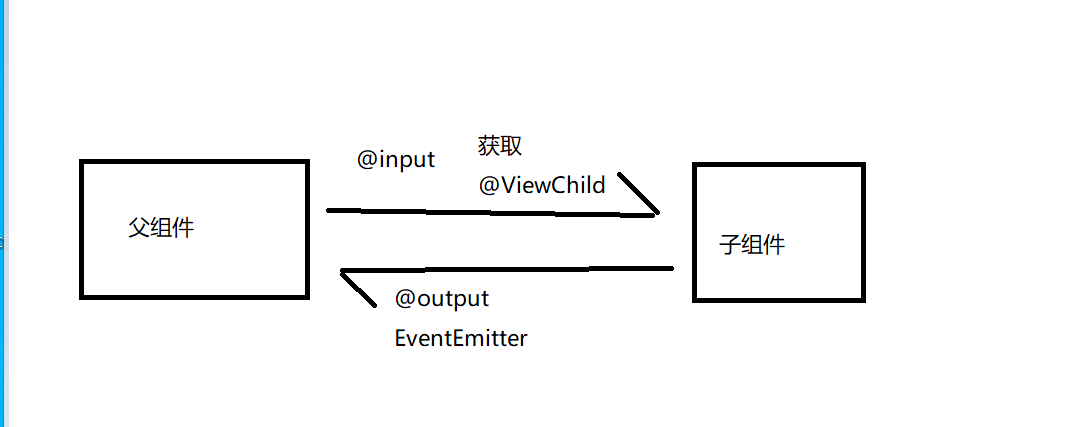








# 父子组件通讯



## 一．父组件向子组件传值 @input

1. 在父组件中引入子组件的中自定义一个属性，值从父组件中获取
2. 在子组件中ts中引入@input
3. 在子组件中接收传过来的值



## 二．父组件通过@viewChild主动获取子组件的数据和方法

1. 在调用子组件中添加标志名称

 <app-test-app #testappData></app-test-app>

1. 在父组件中引入ViewChild

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

3.使用@ViewChild获取子组件数据

 @ViewChild('testappData') data:any

 getChild(){ alert('获取子组件中的数据:'+this.data.username);}

## 三．子组件通过@output向父组件传参

1. 在子组件中引入output ，EventEmitter

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

2.声明

  // 修改事件名,\*\*\*\*\*\*\*\*必须是：属性名 + Change 形式\*\*\*\*\*\*\*\*\*

  @Output() newItemEvent = new EventEmitter<number>();

3.通过emit来传递数据

 this.newItemEvent.emit(this.size);

1. 父组件接收，父组件中appFontSize（$event）方法接收

<app-test-app (newItemEvent) = "appFontSize（$event）"></app-test-app>

# 路由

1. 创建一个有路由的项目 g new 项目名称 --routing --defaults
2. 创建几个子文件 ng generate component 名称
3. 在路由文件（app-routing.module.ts）中引入文件,

import { HomeComponent } from './pages/home/home.component';

const routes: Routes = [

  { path: '', pathMatch: 'full', redirectTo: '/welcome' },

  { path: 'welcome', loadChildren: () => import('./pages/welcome/welcome.module').then(m => m.WelcomeModule) },

  { path : 'Home' , component :HomeComponent },

];

1. 在需要显示所有路由文件的地方使用路由组件代替

<!-- 根据路由显示 -->

      <router-outlet></router-outlet>

1. 跳转 routerLink 路径 routerLinkActive 默认选择是添加这属性

<a routerLink = "/Home" routerLinkActive="active">222</a>

## 路由跳转传参

### Get传值

使用queryParams进行传参

<!-- 路由get传参 -->

<a [routerLink] = "['/Home']" [queryParams]="{aid:123}">222</a>

接收：在跳转的页面进行获取数据

首先需要引入ActivatedRoute，然后使用构造函数constructor进行声明，然后在ngOnInit进行获取this.route.queryParams

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

 name = "";

  constructor(private route:ActivatedRoute) { }

 ngOnInit(): void {

    //获取路由的值

    console.log(this.route);

    this.route.queryParams.subscribe( params =>{

      console.log(params);

      this.name = params['name']

    })

  }

### 动态路由

在配置理由时添加需要传递的参数

 { path : 'Home/:aid' , component :HomeComponent },

使用routerLink的第二个参数进行传参

<!-- 动态路由传参 -->

<a [routerLink]="['/Home/',4]">333</a>

接收：在跳转的页面进行获取数据

首先需要引入ActivatedRoute，然后使用构造函数constructor进行声明，然后在ngOnInit进行获取 this.route.params

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

 name = "";

  constructor(private route:ActivatedRoute) { }

 ngOnInit(): void {

    //获取路由的值

    console.log(this.route);

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

      console.log(data);

      this.name = data['aid']

    })

  }

### JS路由跳转

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

//js跳转页面 动态路由和get路由都可以

  ongetnewpage(){

  //动态路由 使用这种配置{ path : 'Home/:aid' , component :HomeComponent },

this.router.navigate(['/Home/','12ss3'])

    //get路由  使用这种配置{ path : 'Home' , component :HomeComponent },

    // let navigationExtras:NavigationExtras={

    //   queryParams:{aid:111}

    // }

// this.router.navigate(['/Home'],navigationExtras);

this.router.navigate(['/Home', { aid: '111'}]);

  }

### 嵌套路由

使用childen进行嵌套路由

{ path: 'first-component', component: FirstComponent,children: [ { path: 'child-a', component: ChildAComponent,}, { path: 'child-b', component: ChildBComponent }, ], },

# 服务

创建服务 ：ng g service 目录名称

配置服务： 在根目录中引入服务文件

修改ng-zrror库样式

# **方法总结：**

1. 类名等 前加 ****::ng-deep****
2. 类名等 前加 ****:root****
3. 类名等 前加 ****:host /deep/****

Angular 测试

分为单元测试和端到端测试（E2E)

工具：Karma+Jasmine

需要安装NodeJs

安装测试运行期Karma npm install -g karma

App.spec.ts

//细则套件describe('名称'，方法（）)

//可嵌套

// describe('', () =>{

//   describe('', () =>{

//   })

// })

/it编写测试用例

  //expect 内写期望的函数结果

  //expect(value1).toBe(value2) 创建测试预期 value1是真实值 value2是期望值 toBe 匹配的函数

  //expect(value1).not.toBe(value2)   not.toBe 否定式 真实值不什么期望值

  it('app测试', () => {

 expect(app).toBeTruthy();

})

匹配器函数

toBe(value) 相当于===

toEqual(value) 相当于==