# 1.Angular环境搭建

1.nodejs

2.cnpm

3.@angular/cli

ng v 查看安装成功



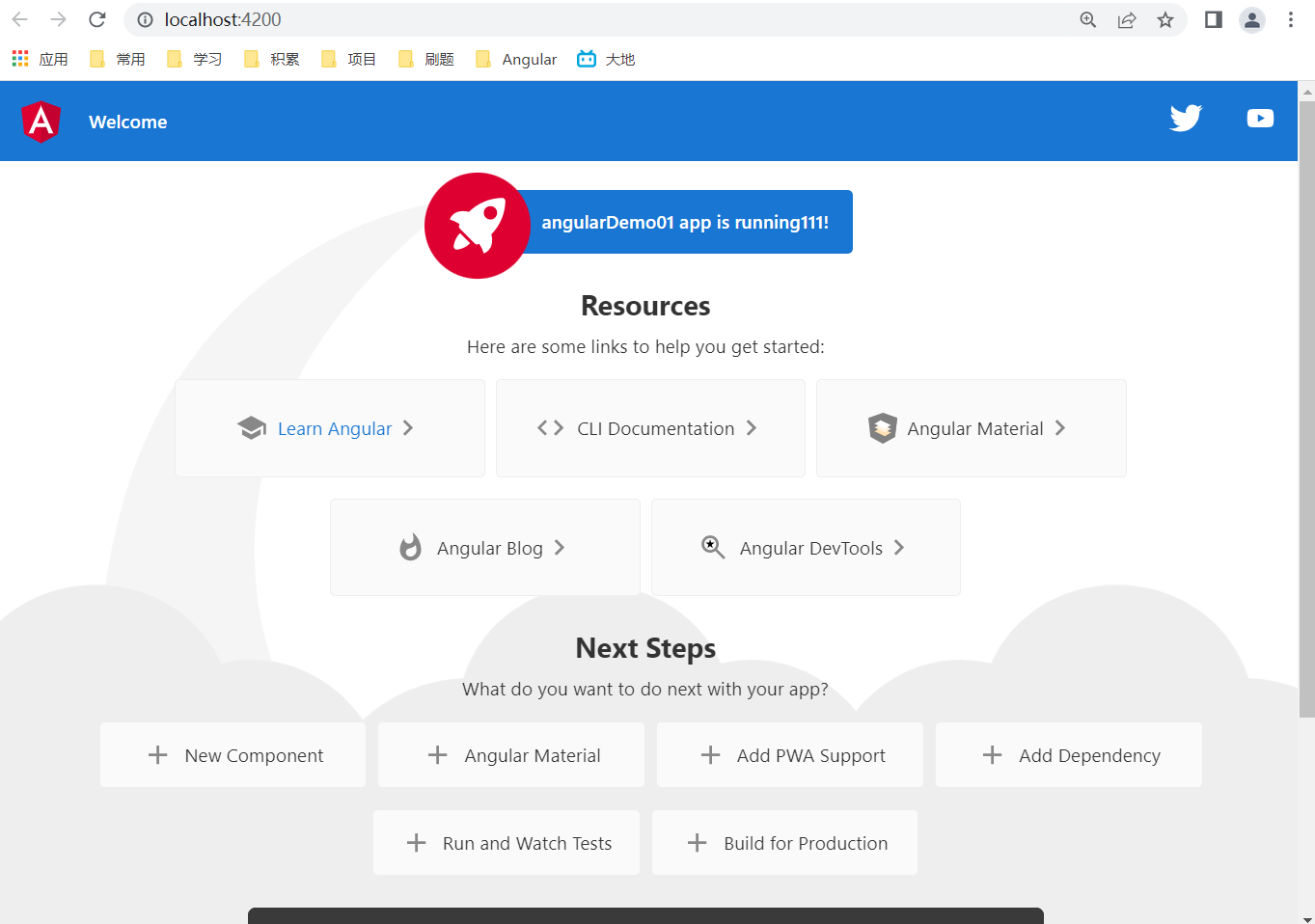
# 2.项目创建

1.创建项目

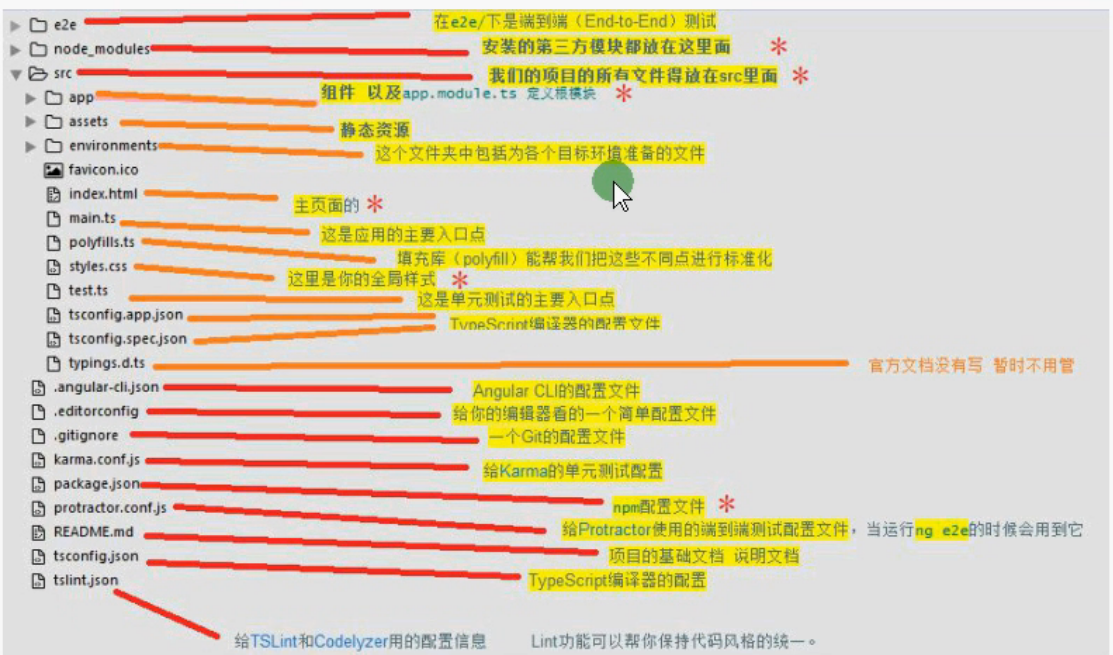
ng new 项目名称

2.运行项目

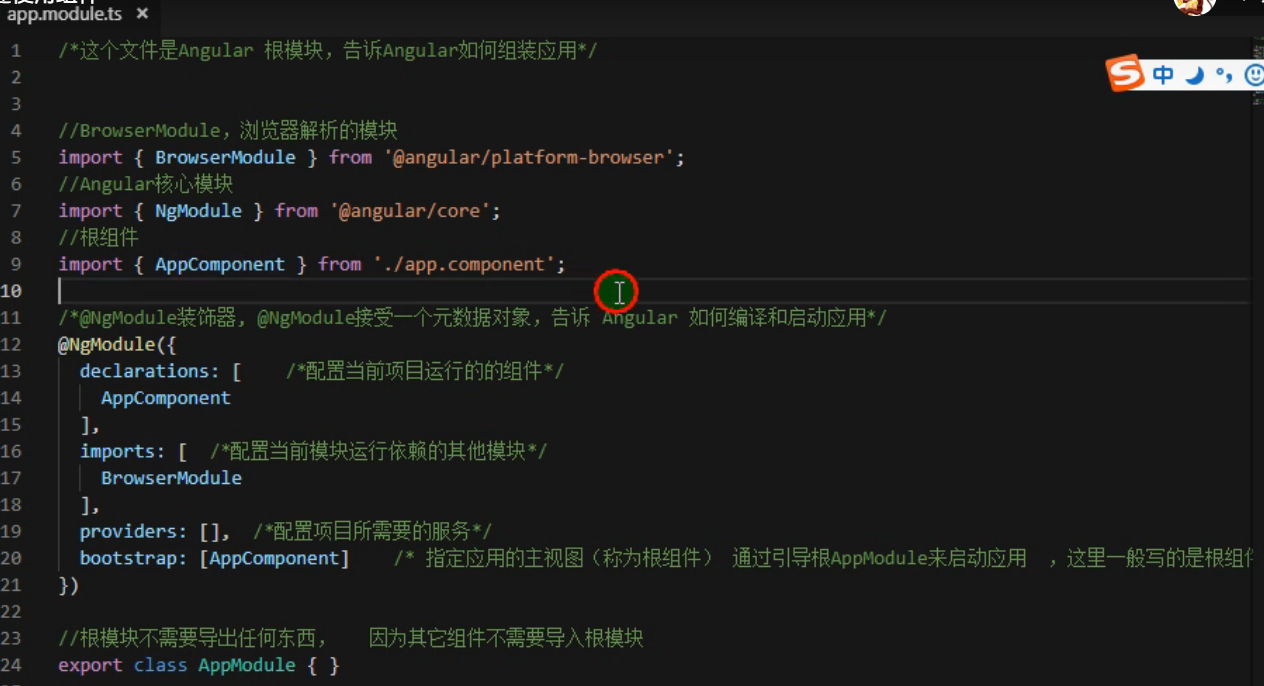
ng serve –open

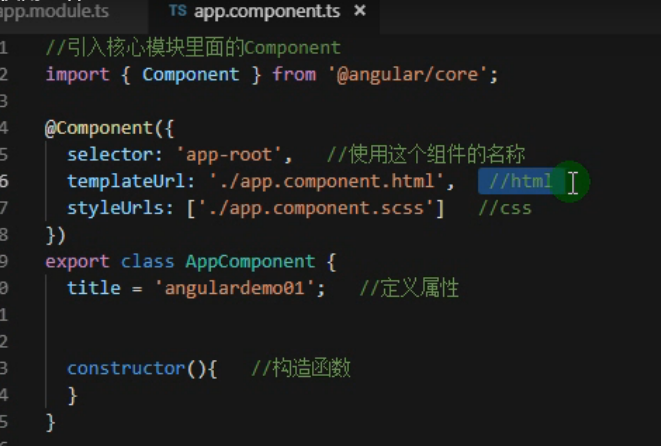


# 3.目录结构分析



app.module.ts详解

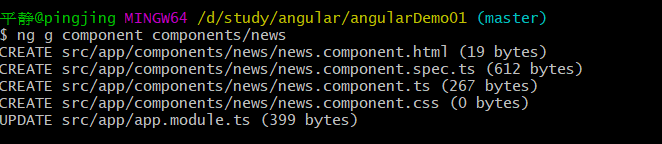


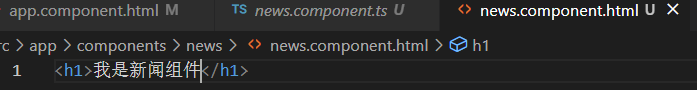


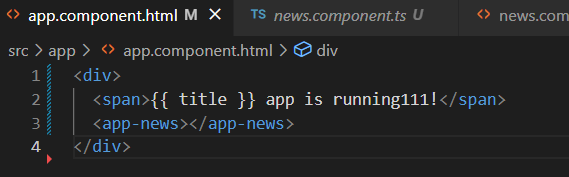
# 4.创建组件

ng g component 目录/组件名 简写ng g c xxx

ng g component components/news



使用



绑定数据

1)数据文字绑定：{{}}

public title:string = '我是新闻组件'

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

2)html绑定

  public htmlStr:string = '<h1>我是html</h1>'

<div [innerHtml]="htmlStr"></div>

3.简单计算

<div>{{1+1}}</div>

4.数据循环

# 5.模板讲解

模板就是html,

## 5.1插值语法 {{}}

<div>{{title}}</div>

<div>{{1+1}}</div>

<div>{{0.1+0.2}}</div>

<div>{{ title + '，拼接内容'}}</div>

<div>{{true? '当前为true': '当前为false'}}</div>

<div>{{date}}</div>

## 5.2 属性绑定 [属性名]=字符串/对象/数组

public boxid:string = 'box-id'

  public boxTitle:string = '我是鼠标悬浮提示'

  public redClass:string = 'red'

  public redStyle:string = 'red'

  public isGreen:boolean = true

  public moreClass:string = 'red font-weight'

  public moreStyle:string = 'color:red;border:1px solid green;width:300px'

  public isWeight100:boolean = true

Attr绑定

<div [id]="'box'">字符串</div>

<div [id]="boxid">变量</div>

<div [title]="'我是鼠标悬浮提示'">字符串</div>

<div [title]="boxTitle">变量</div>

<br/>

类绑定

class

<div [class]="'red'">单个样式绑定</div>

<div [class]="redClass">样式绑定</div>

<div [class.color-green]="'true'">样式是否显示由字符串控制</div>

<div [class.color-green]="isGreen">样式是否显示由变量控制</div>

<div [class]="'red font-weight'">多个样式字符串绑定</div>

<div [class]="moreClass">多个样式变量绑定</div>

<div [class]="{'color-green':isGreen,'font-weight': isWeight100}">多个样式类名条件渲染</div>

<div [class]="['red','font-weight']">类名数组渲染</div>

<div [ngClass]="{'color-green': isGreen,'font-weight':isWeight100}">ngClass对象方式绑定</div>

style

<div [style.width]="'300px'">单一样式绑定</div>

<div [style.width.px]="'300'">单一样式绑定单位拼接</div>

<div [style.color]="redStyle">单一样式绑定</div>

<div [style]="'color:red;border:1px solid green;width:300px'">多样式字符串绑定</div>

<div [style]="moreStyle">多样式变量绑定</div>

<div [style]="{color:'red',border:'1px solid green',width:'300px'}">多样式对象方式绑定</div>

<div [ngStyle]="{color:'red',border:'1px solid green',width:'300px'}">ngStyle对象方式绑定</div>

<div [ngStyle]="{color: isGreen ? 'green' : 'red',border:'1px solid green',width:'300px'}">ngStyle对象中使用变量方式绑定</div>

## 5.3 条件判断 \*ngIf="expression"

ngIf解析后是：ng-template [ngIf]="isGreen"

ngIf直接影响是否渲染

<div \*ngIf="isGreen">变量isGreen是true</div>

<div \*ngIf="!isGreen">变量isGreen是false</div>

<br/>

解析完如下：

<ng-template [ngIf]="isGreen">

    <div>变量isGreen是true</div>

</ng-template>

<br/>

有else的写法

<ng-container \*ngIf="isGreen; else elseTemplate">

    <div>变量isGreen是true</div>

</ng-container>

<ng-template #elseTemplate>

    <div>变量isGreen是false</div>

</ng-template>

## 5.4 循环语句 ngFor ngSwitch

5.4 循环语句 \*ngFor="let item of colors let i = index let o = odd"

<br />

for

<div \*ngFor="let item of colors let i = index let o = odd">

    {{item}}

    {{i}}

    {{o}}

</div>

<br />

解析后

<ng-template ngFor let-color [ngForOf]="colors" let-ii=index let-o=odd>

    <div>

        {{color}}

        {{ii}}

        {{o}}

    </div>

</ng-template>

<br />

switch

<div [ngSwitch]="type">

    <p \*ngSwitchCase="1">

        type是1

    </p>

    <p \*ngSwitchCase="2">

        type是2

    </p>

    <p \*ngSwitchDefault>

        type是other

    </p>

</div>

## 5.5 事件绑定 (事件类型)=”执行函数()”,传参$event

clickFunc(e: Event) {

    console.log(e)

    alert('你好')

  }

  changeFunc(e: Event) {

    let value = (e.target as HTMLInputElement).value

    console.log(value)

  }

<input (input)="changeFunc($event)" type="text" name="" id="" />

<button (click)="clickFunc($event)">点击</button>

## 5.6 模板引用变量 #

模板引用变量可以在组件上，则该变量会引用该组件实例

可以在html元素上，则该变量会引用该html元素

ng-template上使用，则该变量会引用一个TemplateRef实例来代表此模板

 getUsernameFunc(value:string){

    console.log(value)

  }

5.6 模板引用变量 #

<!-- 通过模板引用变量获取当前input的值 -->

<div>

    <input #username (input)="changeFunc($event)" type="text" name="" id="" />

    <button (click)="getUsernameFunc(username.value)">点击</button>

</div>

## 5.7 数据双向绑定 [(ngModel)]="name"

双向绑定是应用中组件共享数据的一种方式，使用双向绑定来监听事件并在父子组件间同步更新值

ngModel仅对表单元素有效，且需要导入FormsModule模块

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

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

import { FormsModule } from '@angular/forms'

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

import { NewsComponent } from './components/news/news.component';

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

import { HeaderComponent } from './components/header/header.component';

@NgModule({

  // 声明组件内用到的试图

  declarations: [

    AppComponent,

    NewsComponent,

    HomeComponent,

    HeaderComponent

  ],

  // 声明组件需要用到的类

  imports: [

    BrowserModule,

    FormsModule

  ],

  // 全局服务

  providers: [],

  // 根组件

  bootstrap: [AppComponent]

})

export class AppModule { }

5.7 数据双向绑定 [(ngModel)]="name"

<div>

    <input [(ngModel)]="name" />

    <p>{{name}}</p>

</div>

## 5.8 动态表单控件

1).在应用中注册响应式表单模块，该模块声明一些动态表单使用的指令

import { ReactiveFormsModule } from '@angular/forms'

imports: [

    ReactiveFormsModule

  ],

2）在组件中new一个FormControl的实例

import {FormControl} from '@angular/forms'

public age:FormControl = new FormControl('')

3）在模板中使用FormControl，注意age是个FormControl，所以要获取值需要age.value

5.8 动态表单控件 formControl ReactiveFormsModule

与数据双向绑定的区别:可以多组

<div>

    年龄:<input type="text" [formControl]="age"/>

    {{age.value}}

</div>

4）动态表单修改 this.age.setValue('18')

changeAge(){

    this.age.setValue('18')

  }

<button (click)="changeAge()">改变年龄</button>

## 5.9 动态表单控件组FormControl, FormGroup, formControlName

1) 在组件中new一个FormGroup的实例,每一个表单元素都是一个FormControl的实例

import {FormControl, FormGroup} from '@angular/forms'

public loginForm = new FormGroup({

    username: new FormControl(''),

    password: new FormControl('')

  })

2）在模板中使用

5.9 动态表单控件组

<form [formGroup]="loginForm">

    <label>

        用户名:

        <input type="text" formControlName="username" />

    </label>

    <label>

        密码:

        <input type="password" formControlName="password" />

    </label>

    <button (click)="submitForm()">提交</button>

</form>

3）控件组的值的获取

submitForm(){

    console.log(this.loginForm.value)

  }

## 5.10 表单验证：模板引用变量=”ngModel”+reqired+模板引用变量.valid



public forms2 = {

    username: '',

    age: ''

  }

submitForm2(){

    console.log(this.forms2)

  }

<form>

    <label>

        用户名:

        <input type="text" required #form2Username="ngModel" [(ngModel)]="forms2.username" name="form2Username" />

    </label>

    验证结果:{{form2Username.valid}}

    <label>

        年龄:

        <input type="text" required #form2Age="ngModel" [(ngModel)]="forms2.age"  name="form2Age" />

    </label>

    验证结果:{{form2Age.valid}}

    <button (click)="submitForm2()">提交</button>

</form>

还可以添加样式

form .ng-invalid {

    border: 1px solid red;

}

## 5.11 自定义表单验证

1）引入依赖

import { FormControl, FormGroup, FormBuilder, Validators } from '@angular/forms'

Validators是内置验证器

2）构造函数中注入FormBuilder

constructor(private B : FormBuilder) {}

// 2）构造函数中注入FormBuilder

  constructor(private B: FormBuilder) { }

3）在组件类的中定义对表单元素的校验

   // 3）在组件类的初始化函数中定义对表单元素的校验

   public loginForm3 = this.B.group({

     // 数组中第一项是默认值,第二项是验证器,支持内置验证器和自定义验证函数

    username: ['', [Validators.required, Validators.maxLength(6), Validators.minLength(3)]],

    password:['',[this.passwordVaild]],

    phone:['',[Validators.required,this.phoneVaild]]

  })

  passwordVaild(password:FormControl):any{

    let value = password.value || ''

    if(!value){

        return {msg:'请输入密码'}

    }

    let valid = value.match(/^[0-9a-zA-Z]{3,8}$/)

    return valid ? {} : {msg:'密码格式不正确'}

  }

  phoneVaild(phone:FormControl):any{

    let value = phone.value || ''

    if(!value){

      return {msg:'请输入手机号码'}

  }

  let valid = /^[0-9]{11}$/.test(value)

  return valid ? {} : {msg:'电话号码格式不正确'}

  }

  submitForm3(){

    console.log(this.loginForm3.get('username')?.errors)

    console.log(this.loginForm3.get('password')?.errors?.['msg'])

  }

4）模板中引用

5.11 自定义表单验证

<form [formGroup]="loginForm3">

    <label>

        用户名:

        <input type="text" formControlName="username" />

    </label>

    <span \*ngIf="this.loginForm3.get('username')?.errors?.['required']">请输入用户名</span>

    <span \*ngIf="this.loginForm3.get('username')?.errors?.['minlength']?.['requiredLength']

    || this.loginForm3.get('username')?.errors?.['maxlength']?.['requiredLength']">用户名必须是3到6位字符</span>

    <br />

    <label>

        密码:

        <input type="password" formControlName="password" />

    </label>

    <span>{{loginForm3.get('password')?.errors?.['msg']}}</span>

    <br />

    <label>

        手机号:

        <input type="text" formControlName="phone" />

    </label>

    <span>{{loginForm3.get('phone')?.errors?.['msg']}}</span>

    <br />

    <button (click)="submitForm3()">提交</button>

</form>

## 5.12 angular 管道

1）内置pipe



2)管道的用法



5.12 angular 管道

<div>

    日期格式化

    <p>{{dateStr | date:'yyyy-mm-dd hh:mm:ss'}}</p>

    大小写转换

    <p>{{ nameStr | uppercase | lowercase }}</p>

    json转换

    <p>{{ jsonObj | json }}</p>

    数值

    <!-- {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}

    即：整数位保留最小位数.小数位保留最小位数-小数位最大保留位置 -->

    <p>{{ numberTest | number: '1.2-2' }}</p>

    货币

    <p>{{numberTest | currency}}</p>

    百分比

    <p>{{floatTest | percent}}</p>

    自定义

    <p>{{nameStr | test1}}</p>

    自定义传参

    <p>{{nameStr | test2:'--'}}</p>

</div>

3）管道的创建

ng g p 管道名称

不传参

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

  name: 'test1'

})

export class Test1Pipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {

    return `<<<${value}<<<`;

  }

}

传参

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

  name: 'test2'

})

export class Test2Pipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {

    const concatStr = args[0];

    return concatStr ? `${concatStr}${value}${concatStr}` : value

  }

}

# 6.组件生命周期

 ngOnInit(): void { console.log('1.ngOnInit')} // 初始化,只调用一次

  ngOnChanges(): void { console.log('2.ngOnChanges');console.log(this.title)} // 属性监听的生命周期,该钩子中的操作会影响性能

  ngDoCheck(): void { console.log('3.ngDoCheck') }

  ngAfterContentInit(): void { console.log('4.ngAfterContentInit') }

  ngAfterContentChecked(): void { console.log('5.ngAfterContentChecked') }

  ngAfterViewInit(): void { console.log('6.ngAfterViewInit') }

  ngAfterViewChecked(): void { console.log('7.ngAfterViewChecked') }

  ngOnDestroy(): void { console.log('8.ngOnDestroy') } // 组件的销毁

# 7.组件交互

## 1）父传子@Input

父组件调用子组件时，使用[属性名]=属性值传递给子组件

<div style="border: 2px solid red; padding: 1rem;">

    <p>我是首页文件</p>

    <hr/>

    <button (click)="changeTitle()">改变显示</button>

    <app-header [title]="title"></app-header>

</div>

子组件使用@Input装饰器进行接收后，子组件模板中就可以正常使用了

@Input()

  title?:string

## 2) 子传父 @Output

父组件向子组件传递一个事件，该事件用来改变父组件中的值

 父传子title:

    <button (click)="changeTitle()">改变显示</button>

    <app-header [title]="title" (addTask)="addTaskFunc($event)"></app-header>

    <hr/>

    子传父addTask:

    <div \*ngFor="let item of tasks let i = index ">

        {{i+1}}.{{item}}

    </div>

子组件通过@Output 接受该事件，并new一个弹射事件EventEmitter

@Output() addTask = new EventEmitter()

子组件通过调用弹射事件. emit调用父组件中的方法，并传参，达到子传父的效果

addKeEr(){

this.addTask.emit('练习科二')

}

## 3）通过@ViewChild来获取子组件的实例和数据

父组件调用子组件的时候，添加模板引用变量，#名称

    <app-header #headerDom [title]="title" (addTask)="addTaskFunc($event)"></app-header>

父组件使用@ ViewChild(模板引用变量名称) 来获取子组件的数据和方法

@ViewChild('headerDom') child:any

  testFunc(){

    console.log(this.child.title)

    console.log(this.child.addKeEr())

  }

# 8.服务

## 1）定义：从组件抽离出来的代码叫服务，服务的本质是函数。

组件用于展示数据，服务用于获取和处理数据；

组件和服务区分开来提高模块性和复用性。

## 2）创建：ng g s 名称

ng g s servers/list

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

// 声明服务

@Injectable({

  // 设定作用域

  // 默认root 表示注入到app.module.ts, 可以设置为null 表示不设定区域, 也可以是某个模块的名字(一般是懒加载模式)

  providedIn: 'root'

})

export class ListService {

  constructor() { }

  public list:string[] = ['学习ts','学习angular']

  getList(){

    return this.list

  }

}

# 9.依赖注入

## 1）注册服务

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

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

import { FormsModule, ReactiveFormsModule } from '@angular/forms'

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

import { NewsComponent } from './components/news/news.component';

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

import { HeaderComponent } from './components/header/header.component';

import { Test1Pipe } from './pipes/test1.pipe';

import { Test2Pipe } from './pipes/test2.pipe';

import { ListService } from './servers/list.service'

@NgModule({

  // 声明组件内用到的试图

  declarations: [

    AppComponent,

    NewsComponent,

    HomeComponent,

    HeaderComponent,

    Test1Pipe,

    Test2Pipe

  ],

  // 声明组件需要用到的类

  imports: [

    BrowserModule,

    FormsModule,

    ReactiveFormsModule

  ],

  // 全局服务

  providers: [ListService],

  // 根组件

  bootstrap: [AppComponent]

})

export class AppModule { }

## 2）引入服务，注入构造函数，使用服务

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

import { ListService } from '../../servers/list.service'

@Component({

  selector: 'app-home',

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

  styleUrls: ['./home.component.css']

})

export class HomeComponent implements OnInit {

  public title:string = '李文静'

  // public tasks:string[] = ['学习ts','学习angular']

  public tasks:string[] = []

  addTaskFunc(task:string){

     this.tasks.push(task)

  }

  @ViewChild('headerDom') child:any

  testFunc(){

    console.log(this.child.title)

    console.log(this.child.addKeEr())

  }

  constructor(private listService:ListService) {}

  ngOnInit(): void {

    this.tasks = this.listService.getList()

  }

  changeTitle(){

    this.title = 'Wenda'

  }

}

# 10.路由

## 1）路由基础

路由在创建项目时即可添加，ng new routing-app --routing –defaults

如果一开始没加，也可以手动添加路由配置文件并注册，进行使用

### 1.1）路由配置app-routing.module.ts

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

import { RouterModule, Routes } from '@angular/router';

import { HomeComponent} from './components/home/home.component'

import { TaskComponent} from './components/task/task.component'

import { NewsComponent} from './components/news/news.component'

const routes: Routes = [

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

  { path: 'task', component: TaskComponent},

  { path: 'news', component: NewsComponent}

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

### 1.2）路由注册 app.module.ts

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

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

import { FormsModule, ReactiveFormsModule } from '@angular/forms'

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

import { NewsComponent } from './components/news/news.component';

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

import { HeaderComponent } from './components/header/header.component';

import { Test1Pipe } from './pipes/test1.pipe';

import { Test2Pipe } from './pipes/test2.pipe';

import { ListService } from './servers/list.service';

import { TaskComponent } from './components/task/task.component'

import { AppRoutingModule } from './app-routing.module';

@NgModule({

  // 声明组件内用到的试图

  declarations: [

    AppComponent,

    NewsComponent,

    HomeComponent,

    HeaderComponent,

    Test1Pipe,

    Test2Pipe,

    TaskComponent

  ],

  // 声明组件需要用到的类

  imports: [

    BrowserModule,

    FormsModule,

    ReactiveFormsModule,

    AppRoutingModule

  ],

  // 全局服务

  providers: [ListService],

  // 根组件

  bootstrap: [AppComponent]

})

export class AppModule { }

### 1.3）路由渲染app.component.html

<div style="border: 2px solid blue; padding: 1rem;">

  <span>{{ title }} app is running111!</span>

  <!-- <hr /> -->

  <!-- <app-news></app-news> -->

  <!-- <hr /> -->

  <!-- <app-home></app-home> -->

  <hr />

  <nav>

    <ul>

      <li><a routerLink="/home" routerLinkActive="active">首页</a></li>

      <li><a routerLink="/news" routerLinkActive="active">模板示例</a></li>

      <li><a routerLink="/task" routerLinkActive="active">服务示例</a></li>

    </ul>

  </nav>

  <!-- 路由组件渲染的位置 -->

  <router-outlet></router-outlet>

</div>

### 1.4）设置通配路由和默认路径

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

import { RouterModule, Routes } from '@angular/router';

import { HomeComponent} from './components/home/home.component'

import { TaskComponent} from './components/task/task.component'

import { NewsComponent} from './components/news/news.component'

import { ErrorComponent} from './components/error/error.component'

const routes: Routes = [

  // 默认路由配置

  { path: '', component: HomeComponent},

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

  { path: 'task', component: TaskComponent},

  { path: 'news', component: NewsComponent},

  // 通配路由

  { path: '\*\*', component: ErrorComponent}

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

## 2）路由嵌套

设置路由：

const routes: Routes = [

  // 默认路由配置

  { path: '', component: HomeComponent},

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

  { path: 'task', component: TaskComponent},

  { path: 'news', component: NewsComponent},

  { path: 'test',

    component: TestComponent,

    children:[

      {path: 'test1', component: Test1Component },

      {path: 'test2', component: Test2Component },

    ]

  },

  // 通配路由

  { path: '\*\*', component: ErrorComponent}

];

设置子路由组件渲染的位置并添加跳转

<p>test works!</p>

<nav>

    <ul>

      <li><a routerLink="/test/test1" routerLinkActive="active">Test1</a></li>

      <li><a routerLink="/test/test2" routerLinkActive="active">Test2</a></li>

    </ul>

  </nav>

 <!-- test子路由组件渲染的位置 -->

 <router-outlet></router-outlet>

## 3）路由传参

### 3.1）query

#### 3.1.1传参

<p>test works!</p>

<nav>

    <ul>

        <!-- 不传参 -->

        <!-- <li><a routerLink="/test/test2" routerLinkActive="active">Test2</a></li> -->

        <!-- query方式传参 在路由上看是?id=2&name=wenda -->

        <li><a routerLink="/test/test1" routerLinkActive="active" [queryParams]="{id:2,name:'wenda'}">Test1 query方式传参</a></li>

        <!-- param方式传参 在路由上看到的是/a/b 顺序严格要求 -->

        <li><a [routerLink]="['/test/test2', 2, 'wenda']" routerLinkActive="router-link-active">Test2 param方式传参</a></li>

         <!-- Liangzhong方式方式传参 在路由上看是/test/test2/2/wenda?id=2&name=wenda -->

        <li><a [routerLink]="['/test/test2', 2, 'wenda']" [queryParams]="{id:2,name:'wenda'}" routerLinkActive="router-link-active">Test2 query & param方式传参</a></li>

    </ul>

</nav>

<!-- test子路由组件渲染的位置 -->

<router-outlet></router-outlet>

#### 3.1.2获取参数

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

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

@Component({

  selector: 'app-test2',

  templateUrl: './test2.component.html',

  styleUrls: ['./test2.component.css']

})

export class Test2Component implements OnInit {

  constructor(private router:ActivatedRoute) { }

  ngOnInit(): void {

    console.log(this.router.snapshot.queryParams)

    this.router.params.subscribe((params:Params)=>{

       console.log(params)

    })

  }

}

### 3.2) param

#### 3.2.1)设置参数的名字 id

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

import { RouterModule, Routes } from '@angular/router';

import { HomeComponent} from './components/home/home.component'

import { TaskComponent} from './components/task/task.component'

import { NewsComponent} from './components/news/news.component'

import { ErrorComponent} from './components/error/error.component'

import { TestComponent} from './components/test/test.component'

import { Test1Component} from './components/test1/test1.component'

import { Test2Component} from './components/test2/test2.component'

const routes: Routes = [

  // 默认路由配置

  { path: '', component: HomeComponent},

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

  { path: 'task', component: TaskComponent},

  { path: 'news', component: NewsComponent},

  { path: 'test',

    component: TestComponent,

    children:[

      {path: 'test1', component: Test1Component },

      // 设置参数的名字 id

      {path: 'test2/:id', component: Test2Component },

    ]

  },

  // 通配路由

  { path: '\*\*', component: ErrorComponent}

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

#### 3.2.2) 传参

 <!-- param方式传参 在路由上看到的是/a/b -->

        <li><a [routerLink]="['/test/test2', 2]" routerLinkActive="router-link-active">Test2带参</a></li>

#### 3.2.4）接收参数

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

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

@Component({

  selector: 'app-test2',

  templateUrl: './test2.component.html',

  styleUrls: ['./test2.component.css']

})

export class Test2Component implements OnInit {

  constructor(private router:ActivatedRoute) { }

  ngOnInit(): void {

    this.router.params.subscribe((params:Params)=>{

       console.log(params)

    })

  }

}