**Angular4**

目录

[一、安装最新版本的nodejs 1](#_Toc6271)

[二、全局安装 Angular CLI 脚手架工具 1](#_Toc79)

[三、创建项目 2](#_Toc3993)

[四、目录结构分析 2](#_Toc15416)

[五、创建angualr组件 使用组件 4](#_Toc13756)

[六、创建angualr服务使用服务 8](#_Toc21565)

[七、Angular4.x get post以及 jsonp请求数据 10](#_Toc12491)

[八、路由 动态路由 14](#_Toc25060)

[九、父子组件传值 19](#_Toc21611)

# 一、安装最新版本的nodejs

**注意：**请先在终端/控制台窗口中运行命令 node -v 和 npm -v， **来验证一下你正在运行 node 6.9.x 和 npm 3.x.x 以上的版本。** 更老的版本可能会出现错误，更新的版本则没问题。

# 二、全局安装 Angular CLI 脚手架工具

1. **使用 npm命令安装**

npm install -g @angular/cli

**2.使用 cnpm命令安装**

cnpm install -g @angular/cli

# 三、创建项目

1. **打开cmd找到你要创建项目的目录**
2. **创建项目**

ng new 项目名称 **创建一个项目**

ng new my-app

1. **进入刚才创建的项目里面启动服务**

cd my-app

cnpm install //安装依赖

ng serve --open

# 四、目录结构分析

**看官网**



## 1. app.module.ts、组件分析

定义AppModule，这个根模块会告诉Angular如何组装该应用。 目前，它只声明了AppComponent。 稍后它还会声明更多组件。

//Angular 模块类描述应用的部件是如何组合在一起的。 每个应用都至少有一个 Angular 模块，也就是根模块，

// 用来引导并运行应用。 你可以为它取任何名字。常规名字是AppModule。 也就是 app.module.ts文件

/\*引入组件\*/

import { BrowserModule } from '@angular/platform-browser'; /\*BrowserModule，浏览器解析的模块\*/

import { NgModule } from '@angular/core'; /\*angualrjs核心模块\*/

import { FormsModule } from '@angular/forms'; /\*表单数据绑定 表单验证需要的模块\*/

i

import { AppComponent } from './app.component'; /\*根组件\*/

@NgModule接受一个元数据对象，告诉 Angular 如何编译和启动应用。\*/

@NgModule({

declarations: [ /\*引入当前项目运行的的组件\*/

AppComponent

],

imports: [ /\*引入当前模块运行依赖的其他模块\*/

BrowserModule,

FormsModule

],

providers: [], /\*定义的服务 回头放在这个里面\*/

bootstrap: [AppComponent] /\* 指定应用的主视图（称为根组件） 通过引导根AppModule来启动应用 ，这里一般写的是根组件\*/

})

export class AppModule { }

## 2 创建angualr组件

<https://github.com/angular/angular-cli>

**创建组件：**

ng g component components/header

**组件内容详解：**

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

@Component({

selector: 'app-header', /\*使用组件的名称\*/

templateUrl: './header.component.html', /\*html模板\*/

styleUrls: ['./header.component.css'] /\*css样式\*/

})

export class HeaderComponent implements OnInit {

constructor() { /\*构造函数\*/

}

ngOnInit() { /\*初始化加载的生命周期函数\*/

}

}

# 五、创建angualr组件 使用组件

| **Scaffold** | **Usage** |
| --- | --- |
| [Component](https://github.com/angular/angular-cli/wiki/generate-component) | ng g component my-new-component  指定目录创建 ：ng g component components/Footer |
| [Directive](https://github.com/angular/angular-cli/wiki/generate-directive) | ng g directive my-new-directive |
| [Pipe](https://github.com/angular/angular-cli/wiki/generate-pipe) | ng g pipe my-new-pipe |
| [Service](https://github.com/angular/angular-cli/wiki/generate-service) | ng g service my-new-service |
| [Class](https://github.com/angular/angular-cli/wiki/generate-class) | ng g class my-new-class |
| [Guard](https://github.com/angular/angular-cli/wiki/generate-guard) | ng g guard my-new-guard |
| [Interface](https://github.com/angular/angular-cli/wiki/generate-interface) | ng g interface my-new-interface |
| [Enum](https://github.com/angular/angular-cli/wiki/generate-enum) | ng g enum my-new-enum |
| [Module](https://github.com/angular/angular-cli/wiki/generate-module) | ng g module my-module |

## 数据文本绑定

{{}}

<h1>

{{title}}

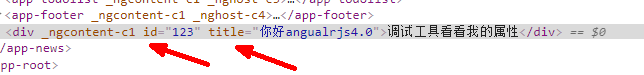
</h1>

## 2.绑定属性绑定 html

this.h="<h2>这是一个h2用[innerHTML]来解析</h2>"

<div [innerHTML]="h"></div>

<div [id]="id" [title]="msg">调试工具看看我的属性</div>



## 3、\*ngFor 普通循环

<ul>

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

{{item}}

</li>

</ul>

<ul>

<li \*ngFor="let item of list2; let i = index">

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

</li>

</ul>

## 4、条件判断

<p \*ngIf="list.length > 3">这是ngIF判断是否显示</p>

## 5、执行事件

<button class="button" (click)="getData()">

点击按钮触发事件

</button>

getData(){ /\*自定义方法获取数据\*/

//获取

alert(this.msg);

}

## 6、双向数据绑定

**<input [(ngModel)]="inputValue">**

**注意引入：FormsModule**

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

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

FooterComponent,

NewsComponent

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**使用：**

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

{{inputValue}}

# 六、创建angualr服务使用服务

|  |
| --- |
| ng g service my-new-service  创建到指定目录下面  ng g service services/storage |

**1.app.module.ts 里面引入创建的服务**

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

**2.** **NgModule 里面的providers 里面依赖注入服务**

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

FooterComponent,

NewsComponent,

TodolistComponent

],

imports: [

BrowserModule,

FormsModule

],

providers: [StorageService],

bootstrap: [AppComponent]

})

export class AppModule { }

**3、使用的页面引入服务，注册服务**

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

constructor(private storage: StorageService) {

}

**使用**

addData(){

// alert(this.username);

this.list.push(this.username);

this.storage.set('todolist',this.list);

}

removerData(key){

console.log(key);

this.list.splice(key,1);

this.storage.set('todolist',this.list);

}

# 七、Angular4.x get post以及 jsonp请求数据

**不使用rxjs请求数据**

**1.引入HttpModule 、JsonpModule**

普通的 HTTP 调用并不需要用到 JsonpModule，不过稍后我们就会演示对 JSONP 的支持， 所以现在就加载它，免得再回来改浪费时间。

import { HttpModule, JsonpModule } from '@angular/http';

**2.HttpModule 、JsonpModule依赖注入**

@NgModule({

declarations: [

AppComponent,

HomeComponent,

NewsComponent,

NewscontentComponent

],

imports: [

BrowserModule,

FormsModule,

HttpModule,

JsonpModule,

AppRoutingModule

],

providers: [StorageService,NewsService],

bootstrap: [AppComponent]

})

export class AppModule { }

**使用Http、Jsonp：**

**1、在需要请求数据的地方引入 Http**

import {Http,Jsonp} from "@angular/http";

**2、构造函数内申明：**

constructor(private http:Http,private jsonp:Jsonp) { }

**3、对应的方法内使用http请求数据**

this.http.get("http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20&page=1")

.subscribe(

function(data){

console.log(data);

},function(err){

console.log('失败');

}

);

this.jsonp.get("http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20&page=1&callback=JSONP\_CALLBACK")

.subscribe(

function(data){

console.log(data);

},function(err){

console.log('失败');

}

);

**使用Post**

1. **引入Headers 、Http模块 用的地方**

import {Http,Jsonp,Headers} from "@angular/http";

1. **实例化Headers**

private headers = new Headers({'Content-Type': 'application/json'});

**3.post提交数据**

this.http

.post('http://localhost:8008/api/test',

JSON.stringify({username: 'admin'}), {headers:this.headers})

// .toPromise()

.subscribe(function(res){

console.log(res.json());

});

**使用rxjs请求数据**

**RxJS是一种针对异步数据流编程工具，或者叫响应式扩展编程；**可不管如何解释RxJS其目标就是异步编程，Angular引入RxJS为了就是让异步可控、更简单。

大部分RxJS操作符都不包括在Angular的Observable基本实现中，基本实现只包括Angular本身所需的功能。

如果想要更多的RxJS功能，我们必须**导入**其所定义的库来扩展Observable对象， 以下是**这个**模块所需导入的所有RxJS操作符：

1. **引入Http 、Jsonp、RxJs 模块**

import {Http,Jsonp} from "@angular/http";

import {Observable} from "rxjs";

import "rxjs/Rx";

你可能并不熟悉这种import 'rxjs/Rx'语法，它缺少了花括号中的导入列表：{...}。

这是因为我们并不需要操作符本身，这种情况下，我们所做的其实是导入这个库，加载并运行其中的脚本， 它会把操作符添加到Observable类中。

1. **构造函数内申明：**

constructor(private http:Http,private jsonp:Jsonp) { }

**3、get请求**

this.http.get("http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20&page=1")

.map(res => res.json()) .subscribe(

function(data){

console.log(data);

}

);

**4、Jsonp请求**

this.jsonp.get("http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20&page=1&callback=JSONP\_CALLBACK")

.map(res => res.json()) .subscribe(

function(data){

console.log(data);

}

);

http.get 方法中返回一个Observable对象，我们之后调用RxJS的map操作符对返回的数据做处理。

# 八、路由

## 8.1创建路由

ng new 项目名称 --routing

E:\1704angular4.x\angular\_demo>ng new angulardemo02 –routing

**app-routing.module.ts**

const routes: Routes = [

{

path: 'home',

component:HomeComponent

// children: []

},

{

path: 'news',

component:NewsComponent

// children: []

}

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

];

## 8.2动态路由

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

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

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

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

import { NewcontentComponent } from './components/newcontent/newcontent.component';

const routes: Routes = [

{

path: 'home',

component:HomeComponent

// children: []

},

{

path: 'news',

component:NewsComponent

// children: []

},{

path: 'newscontent/:aid',

component:NewcontentComponent

// children: []

},

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

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

获取传值

1.引入

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

2.实例化

constructor(private route:ActivatedRoute) { }

3.

ngOnInit() {

// console.log();

this.route.params.subscribe(function(data){

console.log(data);

})

}

## 8.3默认跳转路由

<a routerLink="/home">首页</a>

<a routerLink="/news">新闻</a>

//刚进来路由为空跳转的路由

{

path:'',

redirectTo:'home',

pathMatch:"full"

}

//匹配不到路由的时候加载的组件 或者跳转的路由

{

path: '\*\*', /\*任意的路由\*/

// component:HomeComponent

redirectTo:'home'

}

## 8.4 routerLinkActive设置routerLink默认选中路由

<h1>

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

<a routerLink="/news" routerLinkActive="active">新闻</a>

</h1>

.active{

color:red;

}

## 8.5路由的js跳转

1. **引入**

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

**2.初始化**

export class HomeComponent implements OnInit {

constructor(private router: Router) {

}

ngOnInit() {

}

goNews(){

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

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

}

}

**3.路由跳转**

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

## 8.6父子路由

1. **创建组件引入组件**

import { NewsaddComponent } from './components/newsadd/newsadd.component';

import { NewslistComponent } from './components/newslist/newslist.component';

1. **配置路由**

{

path: 'news',

component:NewsComponent,

children: [

{

path:'newslist',

component:NewslistComponent

},

{

path:'newsadd',

component:NewsaddComponent

}

]

}

1. **父组件中定义router-outlet**

<router-outlet></router-outlet>

# 九、父子组件传值

1. **父组件给子组件传值**

1.子组件

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

2.父组件调用子组件

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

3.子组件中接收数据

export class HeaderComponent implements OnInit {

@Input() msg:string

constructor() { }

ngOnInit() {

}

}

1. **子组件给父组件传值**
2. **子组件引入Output 和 EventEmitter**

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

**2.子组件中实例化 EventEmitter**

@Output() private outer=new EventEmitter<string>();

/\*用EventEmitter 和output装饰器配合使用 <string>指定类型变量\*/

**3. 子组件通过 EventEmitter 对象outer实例广播数据**

sendParent(){

// alert('zhixing');

this.outer.emit('msg from child')

}

**4.父组件调用子组件的时候，定义接收事件 , outer就是子组件的EventEmitter 对象outer**

<app-header (outer)="runParent($event)"></app-header>

**5.父组件接收到数据会调用自己的runParent方法，这个时候就能拿到子组件的数据**

//接收子组件传递过来的数据

runParent(msg:string){

alert(msg);

}