# 学习进度

<https://www.bilibili.com/video/BV1i741157Fj?p=12&spm_id_from=pageDriver>

https://www.bilibili.com/video/BV1i741157Fj?p=13&spm\_id\_from=pageDriver

# 注意的地方

调试时，千万保证目录，如果开到上一级目录会无法进入断点

|  |
| --- |
|  |

|  |
| --- |
| {      // Use IntelliSense to learn about possible attributes.      // Hover to view descriptions of existing attributes.      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387      "version": "0.2.0",      "configurations": [     {         "type": "chrome",         "request": "launch",         "name": "Launch Chrome",         "url": "http://localhost:4200",         "webRoot": "${workspaceFolder}"     }      ]  } |

中文参考

<https://angular.cn/guide/setup-local>

安装

tsc : 无法加载文件 C:\Users\Administrator\AppData\Roaming\npm\tsc.ps1，因为在此系统上禁止运行脚本

以管理员身份 打开windows powershell 输入set-ExecutionPolicy RemoteSigned 选择 A 或者 Y

<https://angular.cn/docs>

**安装 Angular CLI**

npm install -g @angular/cli @latest

学习视频

<https://www.bilibili.com/video/BV1i741157Fj?p=2>

另一个视频

https://www.bilibili.com/video/BV1Wx411R7qt?from=search&seid=16765219986900933931&spm\_id\_from=333.337.0.0

官网

<https://angular.io/start>

中文

<https://www.tslang.cn/docs/handbook/typescript-in-5-minutes.html>

**常用插件**

Angular Follow Selector

Angular 7 Snippets

TSLint

Angular Language Service

**set-ExecutionPolicy RemoteSigned**

**A**

命令

**安装angular的客户端**

**npm install -g @angular/cli**

**创建工程**

**ng new my-app**

**npm install**

**ng serve --open**

**自动创建组件**

**ng generate component hello-everyone**

安装cnpm

|  |
| --- |
| #npm install -g cnpm --registry=https://registry.npm.taobao.org  设置代理  **npm config set registry  https://registry.npm.taobao.org** |

4安装tslib

|  |
| --- |
| npm install -g tslib |

# 1.安装typescript

|  |
| --- |
| npm install -g typescript  npm install -g typescript typings |

2.安装angular脚手架

|  |
| --- |
| npm install -g @angular/cli @latest |

# 2.创建项目

|  |
| --- |
| **如果没有npm设置代理，分两步走**  **ng new app-name  --skip-install**  **cnpm install**  **如果设置了代理**  ng new my-app |

|  |
| --- |
| npm start  **启动时打开浏览器**  npm start –open |

# 3.命令生成模块

|  |
| --- |
| 在项目根目录上敲入  ng g component component/news |

# 4.绑定自定义属性

|  |
| --- |
| <button class="{{className}}" [attr.data-index]="title">点我</button> |

# 5.ngif

|  |
| --- |
| <div \*ngIf="person==='广东人'"> |

# 6.ngFor循环

|  |
| --- |
| <li \*ngFor="let item of persons;let idx=index" (click)="choseperson(idx,item)">ID:{{idx}},姓名:{{item}}</li> |

switch

|  |
| --- |
| <div [ngSwitch]="homeState">      <div \*ngSwitchCase="'睡觉'">卧室</div>      <div \*ngSwitchCase="'发呆'">阳台</div>      <div \*ngSwitchDefault>厕所</div>  </div> |

# 7.数据双向绑定

需要导入

|  |
| --- |
| <input type="text" [(ngModel)]="username"/>  <label>{{username}}</label> |

ref方式

|  |
| --- |
| <input type="text" #refusername />  <button (click)="getRef(refusername)">获取ref</button>    getRef=(refname:any):void=>{      console.info(refname.value)    } |

# 8.表单数据绑定

|  |
| --- |
| <form action="" #formObj='ngForm'>      <input type="text" name="username1" [(ngModel)]="username1"/>      <input type="text" name="password1" [(ngModel)]="password1"/>      <button (click)="mysubmit(formObj)">提交</button>  </form>    mysubmit =(obj:any):void=>{      console.info(obj);    } |

# 9.自定义管道

|  |
| --- |
| ng g pipe filter/lcUpperCase |

# 10.向子组件传递数据

|  |
| --- |
| 子组件  import { Component, OnInit,***Input*** } from '@angular/core';  @Component({    selector: 'app-child',    templateUrl: './child.component.html',    styleUrls: ['./child.component.css']  })  export class ChildComponent implements OnInit {  ***@Input() item:any;***    constructor() { }    ngOnInit(): void {    }  }  子html  <h1>{{item}}</h1> |

|  |
| --- |
| 父组件，html  <app-child [item]="sendMessage"></app-child> |

# 11.子组件传父组件

|  |
| --- |
| 子元素  子元素中  ***@Output() childMsg = new EventEmitter()要和父元素中***  <app-child [item]="sendMessage" ***(childMsg)="getEevent($event)"***></app-child>  ***的名字一样***  import { Component, OnInit,Input,***Output,EventEmitter***} from '@angular/core';  @Component({    selector: 'app-child',    templateUrl: './child.component.html',    styleUrls: ['./child.component.css']  })  export class ChildComponent implements OnInit {    @Input() item:any;  ***@Output() childMsg = new EventEmitter()***    constructor() { }    ngOnInit(): void {    }  ***sendMsgToParent=():void=>{***  ***this.childMsg.emit({msg:'我是子元素，这是我发给父元素的消息'});***  ***}***  }  父亲元素  getChildMessage="";  getEevent=(event:any):void=>{      this.getChildMessage=event.msg;    }  父页面  <h1>来自子组件发来的消息{{getChildMessage}}</h1>  <app-child [item]="sendMessage" ***(childMsg)="getEevent($event)"***></app-child> |

# 12.生命周期

|  |
| --- |
|  |

|  |
| --- |
| 1. ngOnChanges - 当数据绑定输入属性的值发生变化时调用 2. ngOnInit - 在第一次 ngOnChanges 后调用 3. ngDoCheck - 自定义的方法，用于检测和处理值的改变 4. ngAfterContentInit - 在组件内容初始化之后调用 5. ngAfterContentChecked - 组件每次检查内容时调用 6. ngAfterViewInit - 组件相应的视图初始化之后调用 7. ngAfterViewChecked - 组件每次检查视图时调用 8. ngOnDestroy - 指令销毁前调用 |

|  |
| --- |
| ngOnChanges(){      console.log("数据发生变化时ngOnchanges")    }    ngOnInit(){      console.log("第一次显示数据绑定和指令输入属性之后，就会调用，只调用一次");    }    ngDoCheck(){      console.log("在ngOnChanges和ngOnInit发生之后，会进行一次调用")    }    ngAfterContentInit(){      console.log("数据内容渲染到视图之后")    }    ngAfterContentChecked(){      console.log("数据内容渲染到视图检测时候")    }    ngAfterViewInit(){      console.log("在组件和子组件完成初始化之后")    }    ngAfterViewChecked(){      console.log("在组件和子组件完成初始化检测之后")    }    ngOnDestory(){      console.log("在组件销毁之后")    } |

# 13.自定义指令

|  |
| --- |
| ng g directive directive/lcstyle  import { Directive ,**Input,ElementRef**} from '@angular/core';  @Directive({    selector: '[appLcstyle]'  })  export class LcstyleDirective {  **@Input() appLcstyle:string="";**    constructor(**public ref:ElementRef**) {     }    ngOnChanges(){  **this.ref.nativeElement.class=this.appLcstyle;**    }  }  页面使用  <h1 **[appLcstyle]="'abc'">**自定义指令指令</h1> |