# Angular入门基础

## 第1节AngularJs概述

### 1.1什么是AngularJs

AngularJs是一个Javascript framework，也就是一个js框架，它能够帮助我们创建Web、Mobile、Desktop Application。

AngularJS是由谷歌公司创建，它遵循的是MIT协议，开源开放。

它的官网地址是：

<https://angularjs.org/> 1.X系列，2009年发布；

<https://angularjs.io/> 2.X、4.X系列，2016年9月发布2系列，2017年3月发布4系列。

AngularJs这套js框架针对于SPA（Single-Page-Application）应用以及商业业务流程性应用的开发具有特别的优势。

目前已经有许许多多的个人与企业利用AngularJs进行项目开发，也已经有许多成熟的案例，这里有一个网站进行了项目罗列。

<https://www.madewithangular.com/>

注：SPA--> <http://blog.csdn.net/u012974787/article/details/74856870/>

### 1.2AngularJs的优势

我们已经拥有了许多的JavaScript Framework来帮助我们进行Web Application应用的开发，那么我们为什么还要选择AngularJs呢？

1. Dependency Injection依赖注入；
2. Two Way Data-Binding双向数据绑定；
3. Test测试；
4. Model-View Controller控制；
5. Directive（指令），Pipe（管道操作），service（服务），animate（动画）等。

M：model（数据）

V：view（显示）

C：controller（控制器）

### 1.3angular版本之间的比较

Angular1=AngularJs：一个应用非常广泛的JS框架，发布时间久远（2009年google发布）。

Angular2=Angular：相对于Angular1，是一个完全重构的JS框架，虽然有一些概念得以沿用，但与Angular1基本没有什么关系了（2016年9月发布正式版）。

Angular4=Angular2+：是对Angular2的向下兼容性做了一些更新与改进（由于为了解决版本冲突问题，官方跳过了Angular3这个版本）（2017年3月正式版）。

### 1.4建立第一个angular4应用

从CLI开始Angular4项目（CLI，Command-Line-Interface是一个非常好的工具，特别是对于越来越复杂的Angular项目来说，它能够让我们更多的关注流程，而不需要去关注代码内容）。

<https://cli.angular.io/>

1.安装angular的CLI需要NodeJs的环境基础，<https://nodejs.org/>；

2.运行：npm install -g @angular/cli，进行angular的CLI安装；

3.运行：ng new angular-project-name，进行angular项目的创建（注意：因为需要安装node\_modules，所以会比较慢）；

4.运行：cd angular-project-name，进入到创建的项目目录中；

5.运行：ng serve （注意：不是server，是serve），运行刚刚创建的angular项目，运行地址为： <http://localhost:4200>。

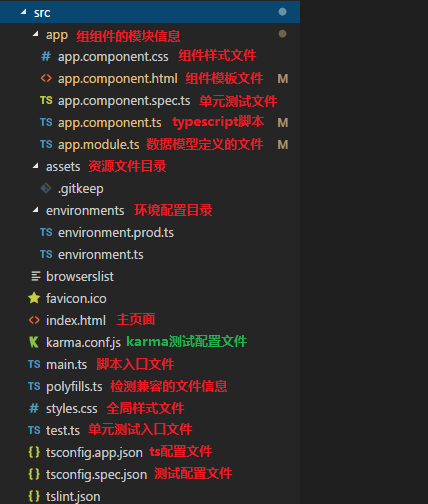
注意：如果CMD命令窗口新建项目编译失败，首先使用Git Bash打开进行新建，再编译运行。

### 1.5项目的文件目录及结构

1）主文件目录



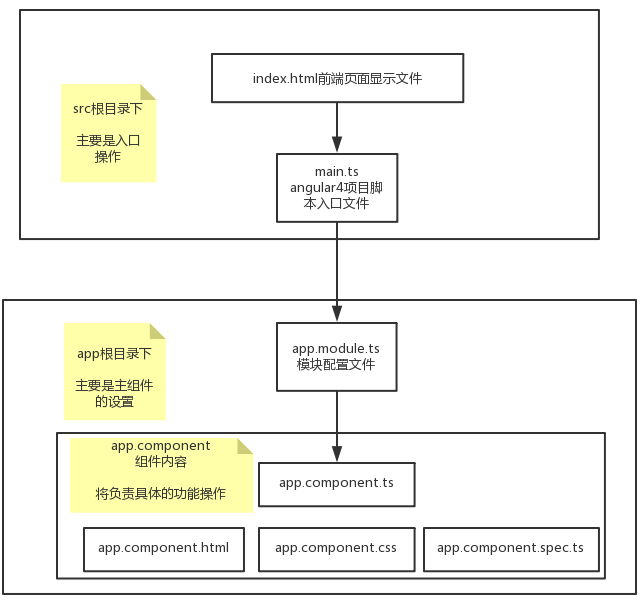
1. src文件目录



1. 其他

Typings.d.ts 系统模块定义文件

1. angular入口结构



### 1.6建立一个输入文本框angular4应用

1. app.module.ts引入表单依赖

|  |
| --- |
| import { FormsModule } from '@angular/forms';  import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { AppComponent } from './app.component';  @NgModule({  declarations: [  AppComponent  ],  imports: [  BrowserModule,  FormsModule  ],  providers: [],  bootstrap: [AppComponent]  })  export class AppModule { } |

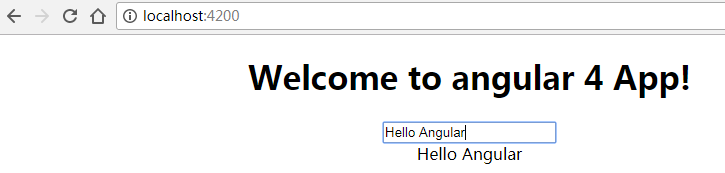
2）app.component.ts修改title

|  |
| --- |
| import { Component } from '@angular/core';  @Component({  selector: 'app-root',  templateUrl: './app.component.html',  styleUrls: ['./app.component.css']  })  export class AppComponent {  title = 'angular 4 App';  } |

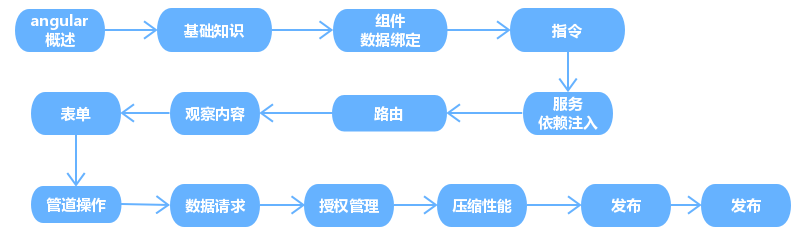
3）增加一个imput输入框并显示输入内容

|  |
| --- |
| <!--The content below is only a placeholder and can be replaced.-->  <div style="text-align:center">  <h1>  Welcome to {{ title }}!  </h1>  <input type="text" [(ngModel)]="name"/>  <br/>  {{ name }}  </div> |

4）效果展示



### 1.7angular总结结构



### 1.8TypeScript

TypeScript比普通js功能更为强大，比如数据类型、类的定义、接口操作等；

TypeScript最终编译为JavaScript。

### 1.9bootstrap框架安装

1）方式一

<http://www.bootcdn.cn/bootstrap/>

引入CDN链接依赖

<link href="https://cdn.bootcss.com/bootstrap/4.1.1/css/bootstrap-grid.css" rel="stylesheet">

添加按钮组：

<div class="btn-group" role="group" aria-label="...">

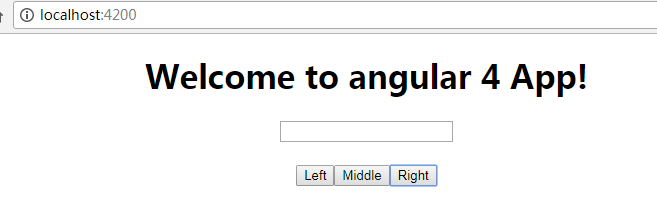
<button type="button" class="btn btn-default">Left</button>

<button type="button" class="btn btn-default">Middle</button>

<button type="button" class="btn btn-default">Right</button>

</div>

保存后自动编译



1. 方式二

安装bootstrap资源到本地

my-first-angular-app>npm i bootstrap -S

+ bootstrap@4.1.1

added 1 package and updated 1 package in 53.542s

angular.json引入bootstrap依赖

"styles": [

"src/styles.css",

"../node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

## 第2节基础知识

### 2.1如何搭建一个应用

现象描述：在运行ng serve以后，打开<http://localhost:4200/>运行程序，根据上一节的知识延续，将显示一个带有title及部分带bootstra框架页面的内容，在第一次打开项目的时候，会显示Loading...的内容，而之后刷新页面的时候，将不会再出现Loading...的内容。

问题思路：查看index.html源码，会看到一些引入的js脚本代码，例如：

<script type="text/javascript" src="runtime.js"></script>

<script type="text/javascript" src="polyfills.js"></script>

<script type="text/javascript" src="styles.js"></script>

<script type="text/javascript" src="vendor.js"></script>

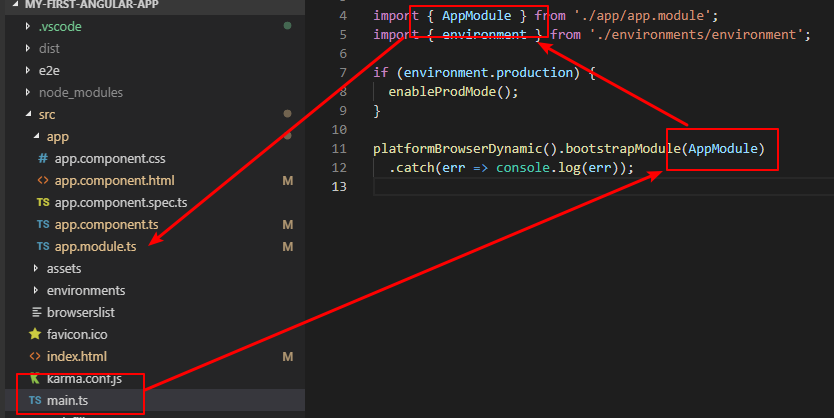
<script type="text/javascript" src="main.js"></script>

Angular已经将我们的项目通过webpack进行了脚本打包，而再次刷新操作以后，代码已经被缓存，执行速度将不再影响，所以我们就看不到Loading...进度操作了？思考一下，这种说明是否正确？（不正确）运作的原理并非如此，我们需要从Angular4的启动原理来理解这一现象的原因。

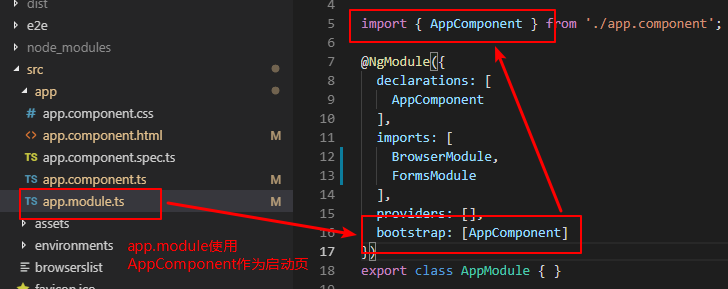
回想一下之前的内容，我们有提到Loading...到底去了哪里，我们通过入口文件及主文件等内容的渲染操作会发现以下结果：Angular4最终还是一个js的框架，它是对HTML的DOM节点进行操作的，我们的项目在加载完成以后，其实已经将<app-root>Loading...</app-root>的代码进行了DOM渲染的替换，所以Loading...再次刷新页面时看不到，所以，这与脚本缓存的问题无关，而是DOM对象的渲染替换问题。

启动入口：

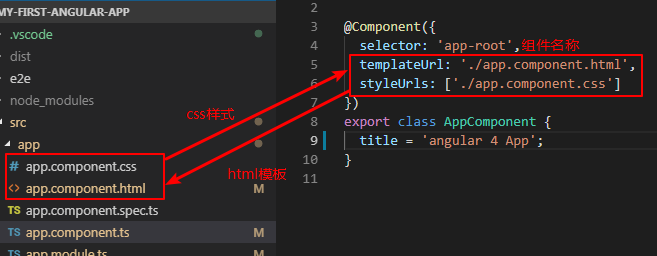
Index.html --> main.ts --> app.module.ts



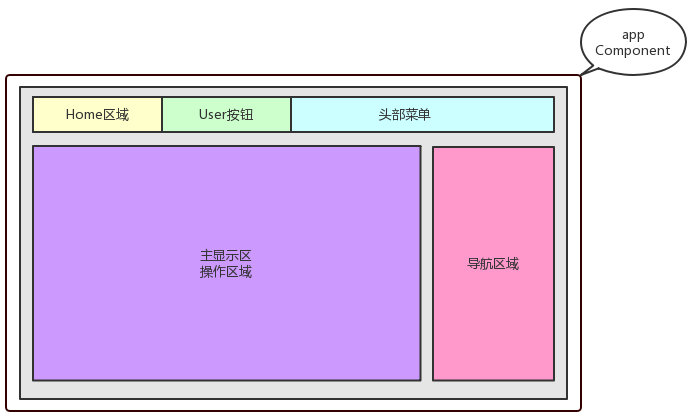
App.module.ts使用@NgModule的bootstrap



App.component.ts --> @Component定义样式和模板文件的引用



### 2.2组件的重要性



头部菜单、按钮、主显示区、导航区域等内容都可以是单独的或是嵌套的组件，组件操作在Angular4开发中项目的核心及基础。

### 2.3创建一个组件

1. 在app目录下新建一个server目录；
2. 创建一个server.component.ts组件文件；

|  |
| --- |
| import { Component } from "@angular/core";  @Component({  // 对象选择器  selector:'app-server',  // 对象模板页，对应的html文件  templateUrl:"./server.component.html",  // 样式  styles:[  `  .online{  color:white  }  `  ]  })  export class ServerComponent{  } |

创建Angular组件的方法有三步：

①从@angular/core中引入Component装饰器

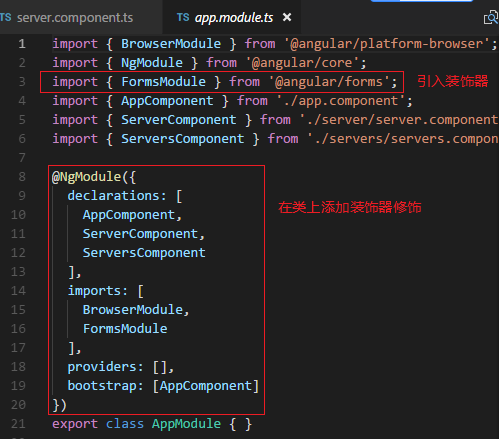
②建立一个普通的类，并用@Component装饰它

③在Component中，设置selector自定义标签，以及template模板

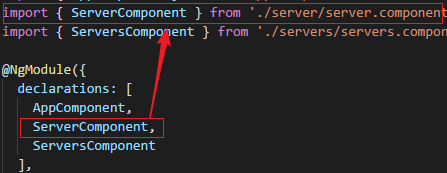
1. 创建一个server.component.html页面文件；

|  |
| --- |
| <p>这是一个server组件~！</p> |

1. 分析一下app.module.ts，其实也是装饰器引入、类的创建、以及装饰器修饰等几大部分；



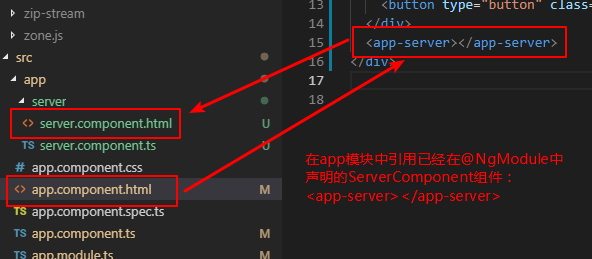
1. 在app.module.ts中引入ServerComponent组件，并将它设置到declarations声明中，声明一下我们的模块中应用到了哪些组件内容；



NgModule的主要属性如下：

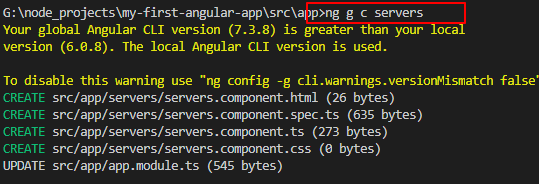
* declarations：模块内部Components/Directives/Pipes的列表，声明一下这个模块内部成员
* providers：指定应用程序的根级别需要使用的service。（Angular2中没有模块级别的service，所有在NgModule中声明的Provider都是注册在根级别的Dependency Injector中）
* imports：导入其他module，其它module暴露的出的Components、Directives、Pipes等可以在本module的组件中被使用。比如导入CommonModule后就可以使用NgIf、NgFor等指令。
* exports：用来控制将哪些内部成员暴露给外部使用。导入一个module并不意味着会自动导入这个module内部导入的module所暴露出的公共成员。除非导入的这个module把它内部导入的module写到exports中。
* bootstrap：通常是app启动的根组件，一般只有一个component。bootstrap中的组件会自动被放入到entryComponents中。
* entryCompoenents: 不会再模板中被引用到的组件。这个属性一般情况下只有ng自己使用，一般是bootstrap组件或者路由组件，ng会自动把bootstrap、路由组件放入其中。 除非不通过路由动态将component加入到dom中，否则不会用到这个属性。

### 2.4使用自定义组件



ServerComponent组件在@Component（{}）中定义的selector选择器名称，在被引用时，使用<app-server></app-server>形式进行引用。

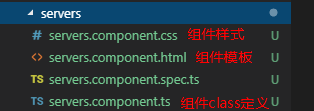
### 2.5CLI嵌套组件与重复组件



使用CLI工具创建组件：

> ng generate component servers

自动生成文件：



组件class定义：

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

@Component({

selector: 'app-servers',

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

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

})

export class ServersComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

组件模板：

<p id="servers" >

<label class="pa1">servers works!</label>

</p>

组件样式：

#servers .pa1 {

color: red;

}

效果展示：



重复组件servers.component.html

|  |
| --- |
| <app-server></app-server>  <app-server></app-server> |

在父组件app.component.html中引用

|  |
| --- |
| <h2>这是一个APP根组件</h2>  <app-servers></app-servers> |

### 2.6使用组件模板

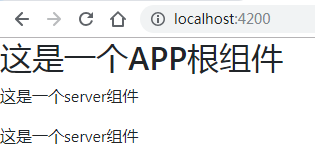
（1）网页显示的模板templateUrl外置文件设置

|  |
| --- |
| @Component({  selector: 'app-servers',  templateUrl: './servers.component.html',  styleUrls: ['./servers.component.css']  }) |

1. 内置模板设置

|  |
| --- |
| @Component({  selector: 'app-servers',  // templateUrl: './servers.component.html',  template:`<app-server></app-server>  <app-server></app-server>`,  styleUrls: ['./servers.component.css']  }) |

效果一样：



### 2.7工作风格（样式）

|  |
| --- |
| <div class="container">  <div class="row">  <div class="xs-col-12"></div>  <br />  <h2>这是一个APP根组件</h2>  </div>  <app-servers></app-servers>  </div> |

|  |
| --- |
| @Component({  selector: 'app-server',  templateUrl: './server.component.html',  styles:[`  p {  color: blue  }  `]  }) |

### 2.8对象选择器

（1）默认

|  |
| --- |
| @Component({  selector: 'app-servers',  template:`<app-server></app-server>  <app-server></app-server>`,  styleUrls: ['./servers.component.css']  }) |

|  |
| --- |
| <app-servers></app-servers> |

（2）属性选择器

|  |
| --- |
| @Component({  // selector: 'app-servers',  // 2 selector作为div属性  selector: '[app-servers]',  // templateUrl: './servers.component.html',  template:`<app-server></app-server>  <app-server></app-server>`,  styleUrls: ['./servers.component.css']  }) |

|  |
| --- |
| <div app-servers></div> |

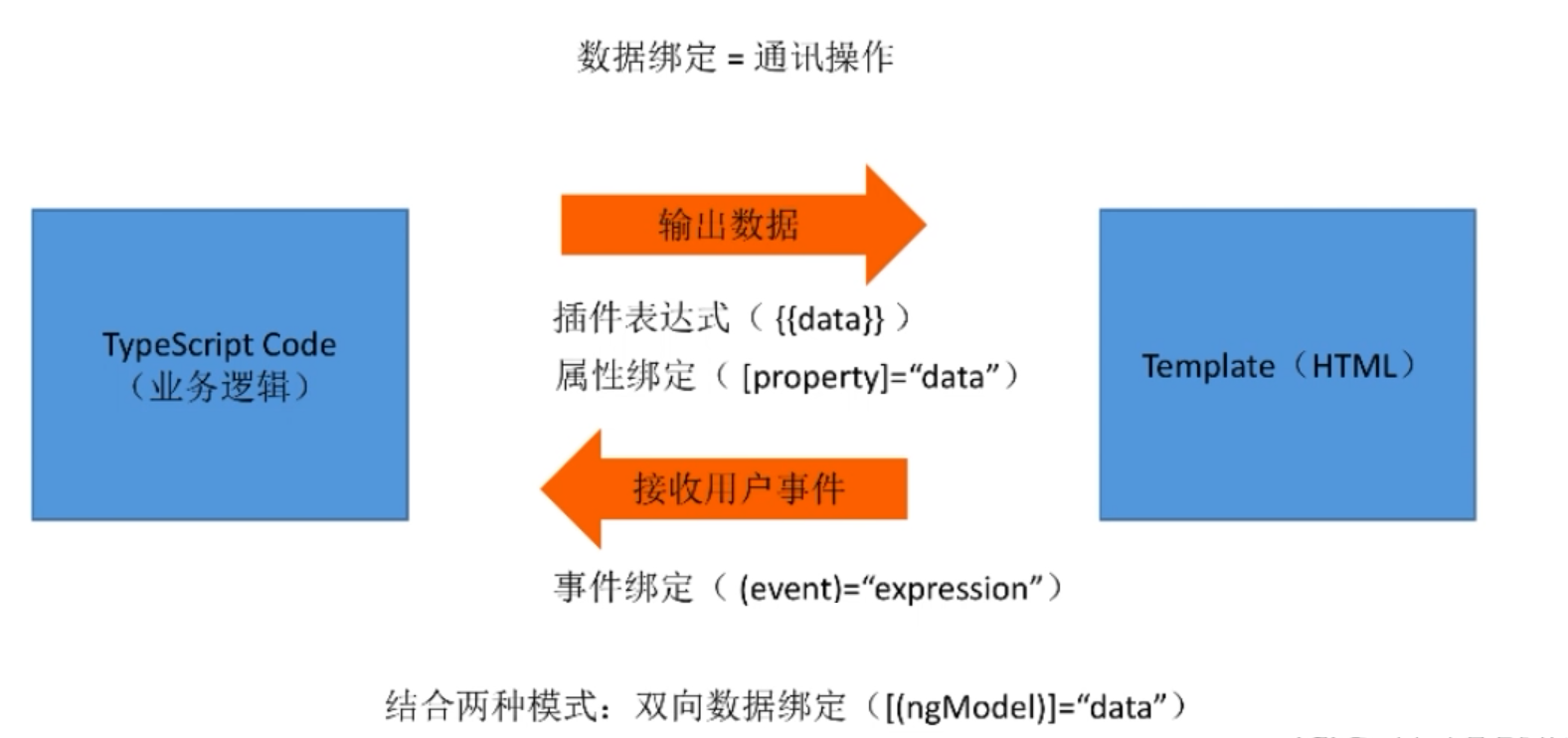
（3）类选择器

|  |
| --- |
| @Component({  // selector: 'app-servers',  // 2 selector作为div属性  // selector: '[app-servers]',  // 3 selector作为类选择器  selector: '.app-servers',  // templateUrl: './servers.component.html',  template:`<app-server></app-server>  <app-server></app-server>`,  styleUrls: ['./servers.component.css']  }) |

|  |
| --- |
| <div class='app-servers'></div> |

## 第3节 组件数据绑定

### 3.1数据绑定



### 3.2字符串插值

业务逻辑层-->前端模板页

|  |
| --- |
| export class ServerComponent {  serverId = 10;  serverStatus = '离线';  getServerStatus() {  return '在线';  }  } |

|  |
| --- |
| <p>服务编号{{serverId}}，状态为{{serverStatus}}</p>  <p>{{'服务描述：'}}服务编号{{serverId}}，状态为{{serverStatus}}</p>  <p>{{'服务描述：'}}服务编号{{serverId}}，状态为{{getServerStatus()}}</p> |



### 3.3属性绑定

业务逻辑层-->前端模板页

|  |
| --- |
| <button type="button" class="btn" [disabled]="allowServer">添加服务</button> |

|  |
| --- |
| @Component({  // selector: 'app-servers',  // 2 selector作为div属性  // selector: '[app-servers]',  // 3 selector作为类选择器  selector: '.app-servers',  templateUrl: './servers.component.html',  /\*template:`<app-server></app-server>  <app-server></app-server>`,\*/  styleUrls: ['./servers.component.css']  })  export class ServersComponent implements OnInit {  allowServer = true;  constructor() {  setTimeout(() => {  this.allowServer = false;  }, 2000);  }  ngOnInit() {  }  } |

### 3.4字符串插值运算

业务逻辑层-->前端模板页

|  |
| --- |
| <p>{{allowServer}}</p>  <p [innerText]="allowServer"></p> |

|  |
| --- |
| <img src="{{imgUrl}}" width="100" height="100"/>  <img [src]="imgUrl" width="100" height="100"/> |



插值如果是字符串类型，属性绑定和插值表达式等同；

如果是判断bool类型，html属性只能使用属性绑定

### 3.5事件绑定

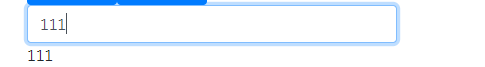


前端模板页-->业务逻辑层

|  |
| --- |
| <button type="button" class="btn btn-primary" [disabled]="allowServer">添加服务</button>  <button type="button" class="btn btn-primary" [disabled]="allowServer" (click)="onServerCreate($event)">添加服务</button> |

|  |
| --- |
| onServerCreate(event : any) {  console.log('[onServerCreate] event :', event );  this.serverCreationStatus = '服务已经创建';  } |

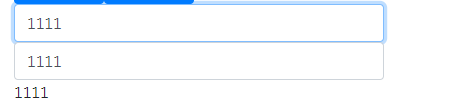
### 3.6数据绑定到事件



|  |
| --- |
| <input type="text" class="form-control col-sm-4" (input)="updateServerName($event)"/> |

|  |
| --- |
| updateServerName(event: any){  this.serverName = event.target.value;  } |

### 3.7双向数据绑定



Cpe.module.ts中需要import导入FormsModule

|  |
| --- |
| <input type="text" class="form-control col-sm-4" [(ngModel)]="serverName" />  {{serverName}} |

|  |
| --- |
| @NgModule({  declarations: [  AppComponent,  ServerComponent,  ServersComponent  ],  imports: [  BrowserModule,  FormsModule,  FormsModule  ],  providers: [],  bootstrap: [AppComponent]  })  export class AppModule { } |

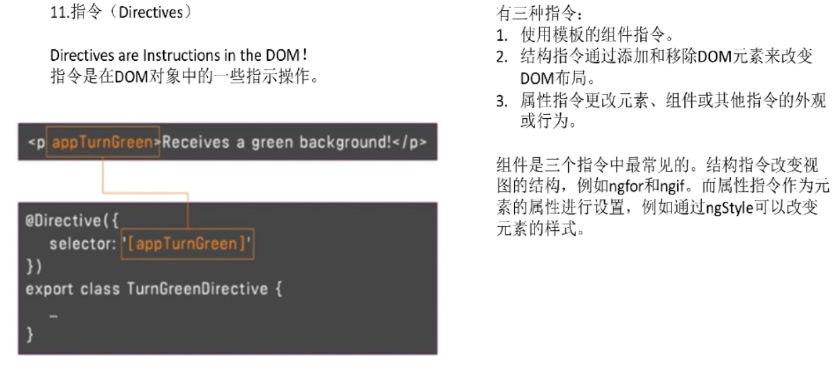
### 3.8各种形式的数据绑定



|  |
| --- |
| onServerCreate(event : any) {  console.log('[onServerCreate] event :', event );  this.serverCreationStatus = '服务已经创建: ' + this.serverName ;  } |

## 第4节指令

### 4.1指令操作



### 4.2 \*ngIf指令



|  |
| --- |
| <P \*ngIf="serverCreated">服务已创建，服务名：{{serverName}}</P> |

|  |
| --- |
| onServerCreate(event : any) {  console.log('[onServerCreate] event :', event );  this.serverCreated = true;  this.serverCreationStatus = '服务已经创建: ' + this.serverName ;  } |

### 4.3 \*ngIf的else情况

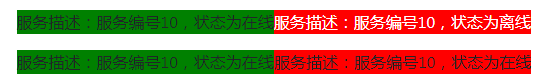
结构指令，改变DOM结构



|  |
| --- |
| <P \*ngIf="serverCreated; else noServer">服务已创建，服务名：{{serverName}}</P>  <ng-template #noServer>  <p>服务没有创建</p>  </ng-template> |

### 4.4ngStyle和ngClass样式指令

属性指令，不改变DOM结构，只改变样式



|  |
| --- |
| <p [ngStyle]="{backgroundColor : getColor()}" [ngClass]="{online: serverStatus==='离线'}">{{'服务描述：'}}服务编号{{serverId}}，状态为{{serverStatus}}</p>  <p [ngStyle]="{backgroundColor : getColor()}">{{'服务描述：'}}服务编号{{serverId}}，状态为{{getServerStatus()}}</p> |

|  |
| --- |
| constructor() {  this.serverStatus = Math.random() > 0.5 ? '在线' : '离线';  }  getColor() {  return this.serverStatus === '离线' ? 'red' : 'green';  } |

### 4.5\*ngFor循环指令与循环索引



|  |
| --- |
| <app-server \*ngFor="let server of servers; let i = index" [ngStyle] = "{color : i>=4?'blue':'yellow'}"></app-server> |

|  |
| --- |
| servers = ['ServerTest', 'ServerTest1', 'ServerTest2'];  onServerCreate(event : any) {  console.log('[onServerCreate] event :', event );  this.serverCreated = true;  this.servers.push(this.serverName);  this.serverCreationStatus = '服务已经创建: ' + this.serverName ;  } |