Angular03

知识点回顾

- 新的命令

```
安装angular
npm install -g @angular/cli

生成项目包
ng new 包名
ng n 包名

启动
ng serve --open
ng s -o

ng s


生成组件
ng generate component 组件名
ng g c 组件名
```

- 指令

指令作用	vue	angular
双标签内容	{{}}	{{}}
属性	v-bind:属性名='' :属性名=""	[属性名]=""
事件	v-on:事件名="" @事件名=""	(事件名)=""
html	v-html=""	[innerHTML]=""
双向绑定	v-model=""	[(ngModel)]=" " 必须手动加载Form模块

指令

条件渲染

vue中

- `v-if` 和 `v-else`

angular中

- `*ngIf`

生成组件:

```
ng g c myc01
```

```
<p>myc01 works!</p>

<h3>您的面试分数:{{ score }}</h3>
<!--
  if判断
  在vue中: v-if
  在ng 中: *ngIf

  if判断的本质是在操作: style的 display属性.  display:none; 就代表隐藏
-->
<button (click)="score = score - 10" [disabled]="score == 0" *ngIf="score > 0">
  减分
</button>
<button
  (click)="score = score + 10"
  [disabled]="score == 100"
  *ngIf="score < 100"
>
  加分
</button>

<!-- if-else -->
<!--
  基础格式: *ngIf="条件"; else id
  否则 则使用 #id 的标签
 -->
<ng-container *ngIf="score >= 60; else abc">
  <p>恭喜! 通过面试.</p>
</ng-container>
<!-- # 在css中是id选择器. ng的作者用#代替了id= -->
<ng-template #abc>
  <p>很遗憾! 面试未通过.</p>
</ng-template>
```

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

@Component({
  selector: 'app-myc01',
  templateUrl: './myc01.component.html',
  styleUrls: ['./myc01.component.css'],
})
export class Myc01Component implements OnInit {
  score = 60;

  constructor() {}
```

```
  ngOnInit(): void {}
}
```

数组遍历

vue中

```
v-for="(item, index) in items"  :key="index"
```

ng中??

生成组件: `ng g c myc02`

```
<p>myc02 works!</p>

<!--
  vue中:
  v-for="(item,index) in items" :key="index"
 -->

<ul>
  <!-- 提示: ng-for -->
  <li *ngFor="let item of names">{{ item }}</li>
</ul>

<!-- 带有index -->
<ul>
  <!-- 提示: ng-for-index -->
  <li *ngFor="let item of names; let i = index">
    <span>index: {{ i }}</span>
    <br />
    <span>name: {{ item }}</span>
  </li>
</ul>

<!--
  vue中, wx小程序中: 列表中的元素必须带有 key 唯一标识, 否则后台报错!

  angular中, key不存在, 不需要写
 -->

<table border="1">
  <tr>
    <td>序号</td>
    <td>名称</td>
    <td>单价</td>
    <td>数量</td>
    <td>总价</td>
  </tr>
```

```
  <tr *ngFor="let item of products; let i = index">
    <td>{{ i + 1 }}</td>
    <td>{{ item.title }}</td>
    <td>{{ item.price }}</td>
    <td>{{ item.count }}</td>
    <td>{{ item.price * item.count }}</td>
  </tr>
</table>
<hr />
<div>
  <div>
    <span>商品名: </span>
    <input type="text" [(ngModel)]="title" />
  </div>
  <div>
    <span>单价: </span>
    <input type="number" [(ngModel)]="price" />
  </div>
  <div>
    <span>数量: </span>
    <input type="number" [(ngModel)]="count" />
  </div>
  <button (click)="doAdd()">添加</button>
</div>
```

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

@Component({
  selector: 'app-myc02',
  templateUrl: './myc02.component.html',
  styleUrls: ['./myc02.component.css'],
})
export class Myc02Component implements OnInit {
  names = ['东东', '亮亮', '然然', '小新', '华哥'];

  products = [
    { title: 'iPhone', price: 7999, count: 5 },
    { title: 'Oppo', price: 5999, count: 2 },
    { title: 'vivo', price: 4999, count: 1 },
    { title: 'HuaWei', price: 6999, count: 4 },
  ];

  title = '';
  price = 0;
  count = 0;

  doAdd() {
    // ts: 不同于js, 是类型严格语言

    let p = {
      title: this.title,
      price: this.price,
      count: this.count,
    };
```

```
    this.products.push(p);

    this.title = '';
    this.price = 0;
    this.count = 0;
  }

  constructor() {}

  ngOnInit(): void {}
}
```

yc02.component.ts TS app.module.ts ×

app > TS app.module.ts > AppModule

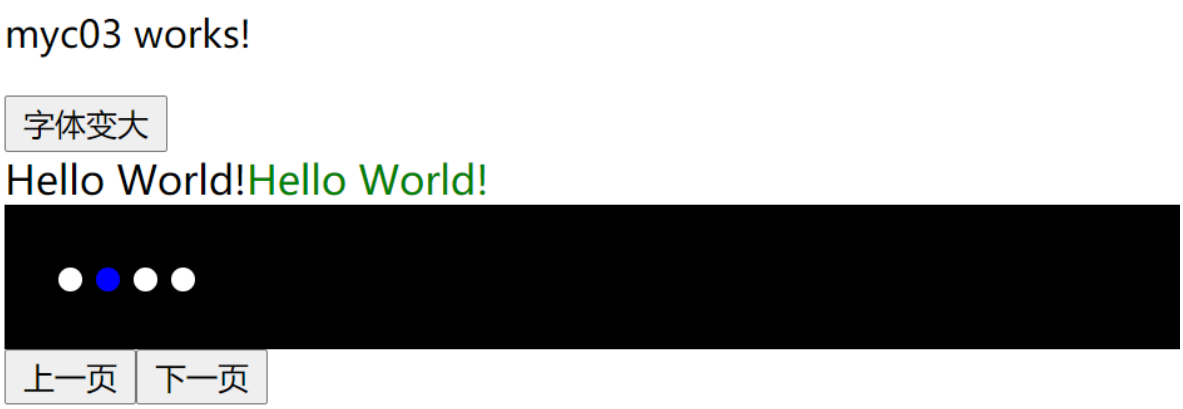
```
import { Myc02Component } from './myc02/myc02.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [AppComponent, Myc01Component, Myc02Component],
  // 配置文件有变更，最好重启服务器
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

样式

新建组件: `ng g c myc03`

动态样式: 依赖style属性，其中的值是变量.

使用场景: 有时候页面上的文字颜色等配置 是通过服务器传递的数据，这样就可以在不更新html代码的情况下, 利用服务器接口传递CSS样式的值，让页面发生变化

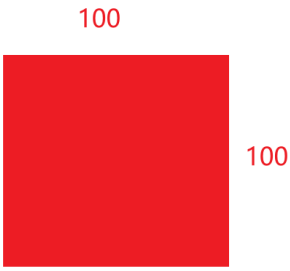



```
import { Component, OnInit } from '@angular/core';

@Component({
```

```
  selector: 'app-myc03',
  templateUrl: './myc03.component.html',
  styleUrls: ['./myc03.component.css'],
})
export class Myc03Component implements OnInit {
  size = 17;

  curP = 0; //当前页

  nextPage() {
    this.curP++;

    if (this.curP == 4) this.curP = 0;
  }

  prevPage() {
    this.curP--;
    if (this.curP < 0) this.curP = 3;
  }

  constructor() {}

  ngOnInit(): void {}
}
```

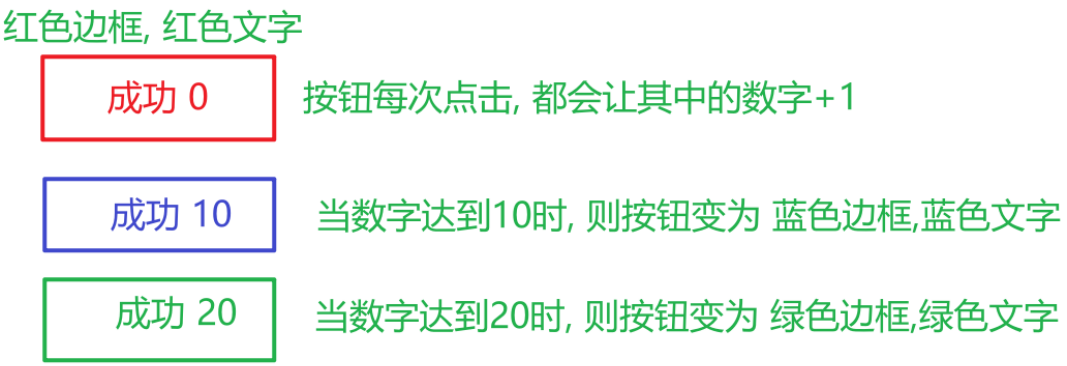
```
<p>myc03 works!</p>

<button (click)="size = size + 5">字体变大</button>
<br />
<!-- 静态样式写法 -->
<span style="font-size: 17px">Hello World!</span>
<!-- 动态样式写法 ng-style -->
<!-- 值是对象类型: 属性名不能带 中划线 -->
<!--
  解决办法:
  font-size -> fontSize 转小驼峰
  font-size -> 'font-size'
 -->
<span [ngStyle]="{ color: 'green', fontSize: size + 'px' }">Hello World!</span>

<!-- 动态css 样式类 -->
<div class="pages">
  <!-- ng-class -->
  <!-- 与vue相同, 动态class的值是对象类型, 依赖true/false 来决定样式是否生效 -->
  <span [ngClass]="{ cur: curP == 0 }"></span>
  <span [ngClass]="{ cur: curP == 1 }"></span>
  <span [ngClass]="{ cur: curP == 2 }"></span>
  <span [ngClass]="{ cur: curP == 3 }"></span>
</div>
<button (click)="prevPage()">上一页</button>
<button (click)="nextPage()">下一页</button>
```

练习1:



每次点击, 都可以变化方块的 圆角, 每次 border-radius +10

练习2:




```
<p>myc04 works!</p>

<div class="cube" [ngStyle]="{ 'border-radius': radius + 'px' }"></div>

<button (click)="radius = radius + 10">变圆</button>
<br /><br />

<button
  [ngClass]="{ red: num < 10, blue: num >= 10 && num < 20, green: num >= 20 }"
  (click)="num = num + 1"
>
  成功 {{ num }}
</button>
```

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

@Component({
  selector: 'app-myc04',
  templateUrl: './myc04.component.html',
  styleUrls: ['./myc04.component.css'],
})
export class Myc04Component implements OnInit {
  radius= 0;
```

```
  num = 0;


  constructor() {}

  ngOnInit(): void {}
}
```

```
.cube {
  width: 100px;
  height: 100px;
  background-color: red;
}

.red {
  border-color: red;
  color: red !important;
}

.blue {
  border-color: blue;
  color: blue;
}

.green {
  border-color: green;
  color: green;
}
```

自定义指令

生成组件

```
ng g c myc05
```

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHidden]',
})
export class HiddenDirective {
  // <tag appHidden></tag>
  // tag就是appHidden的宿主
  constructor(e: ElementRef) {
    // 指令就是标签的属性, 他出生时一定会携带所在的标签作为参数
    console.log(e);
    //如果没有打印, 则重启服务器

    // 根据打印结果, 操作原生DOM
    e.nativeElement.style.display = 'none';
```

```
  }
}
```

```
<p>myc05 works!</p>

<!-- 自定义指令 -->
<!--
  指令: 就是标签的属性
  angular本身提供了一些系统的属性 -- 指令
  *ngIf  *ngFor  ngStyle  ngClass...
 -->

<!-- appHidden 能够隐藏这个标签 -->
<b appHidden>Hello World!</b>

<p appHidden>我是p标签</p>

<!--
  新的命令行:
  ng generate directive 指令名
  ng g d 指令名

  例如: ng g d hidden
 -->

<!-- ng g d danger -->
<p appDanger>危险</p>

<hr />
<input type="text" />
<!-- 指令: appFocus  让输入框自动获得焦点状态 -->
<input type="text" appFocus />
<!-- ng g d focus -->
<input type="text" />
```

练习

```
生成一个指令， appDanger， 其所在的标签的 文本颜色会变为红色文字，带有红色边框

<p appDanger>危险</p>

生成指令: ng g d danger
```

焦点指令

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appFocus]',
})
export class FocusDirective {
  constructor(e: ElementRef) {
    console.log(e);

    e.nativeElement.focus();
  }
}
```

隐藏指令

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHidden]',
})
export class HiddenDirective {
  // <tag appHidden></tag>
  // tag就是appHidden的宿主
  constructor(e: ElementRef) {
    // 指令就是标签的属性, 他出生时一定会携带所在的标签作为参数
    console.log(e);
    //如果没有打印, 则重启服务器

    // 根据打印结果, 操作原生DOM
    e.nativeElement.style.display = 'none';
  }
}
```

危险指令

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appDanger]',
})
export class DangerDirective {
  constructor(e: ElementRef) {
    e.nativeElement.style.color = 'red';
    e.nativeElement.style.border = '1px solid red';
  }
}
```

管道 pipe

> 指令是用来操作 标签的 DOM属性

管道是操作双标签的内容. 在vue中称为 filter 过滤器

```
{{ 值 | 过滤器: 参数:参数 }}
```

vue中的过滤器都需要人为制作, 而 angular 管道提供了一些常用的过滤器可以直接使用, 当然也能定制.

生成组件

```
ng g c myc06
```

大写: DONGDONG

小写: dongdong

单词首字母大写: Nice To Meet You

百分数: 56%

保留小数位: 5.56%

保留小数位: 05.56%

千进位钱:$123,456.78

千进位钱:¥123,456.78

2020-09-24 15:45:22

2020-09-24 03:45:22

2020-9-24 15:45:22

```
<p>myc06 works!</p>

<!-- 系统管道 -->
<p>大写: {{ "dongdong" | uppercase }}</p>
<p>小写: {{ "DONGDONG" | lowercase }}</p>
<p>单词首字母大写: {{ "nice to meet you" | titlecase }}</p>
<p>百分数: {{ 0.55555 | percent }}</p>
<p>保留小数位: {{ 0.05555 | percent: "0.2" }}</p>
<!-- percent: 'n.m'  整数最少n位, 小数最好m位  -->
<p>保留小数位: {{ 0.05555 | percent: "2.2" }}</p>
<p>千进位钱:{{ 123456.78 | currency }}</p>
<p>千进位钱:{{ 123456.78 | currency: "¥" }}</p>
<!-- 日期格式:  -->
<!-- 数据库中的时间通常为时间戳格式:  当前时间距离1970年1月1日凌晨的秒数 -->
<!--
  year  年  y
  month 月  M
  day   日  d
  hour  时  h12小时  H24小时
  minute分  m
  second秒  s
 -->
<!-- 时间戳单位要求是 毫秒 -->
<p>{{ 1600933522520 | date: "yyyy-MM-dd HH:mm:ss" }}</p>
<p>{{ 1600933522520 | date: "yyyy-MM-dd hh:mm:ss" }}</p>
<!-- 重复两位 代表自动补0 -->
```

```
<p>{{ 1600933522520 | date: "y-M-d H:m:s" }}</p>
```

自定义管道

```
生成新的组件:
ng g c myc07
```

```
<p>myc07 works!</p>

<!-- 自制开平方管道 -->
<p>{{ 81 | sqrt }}</p>
<p>{{ 25 | sqrt }}</p>
<!--
  生成管道命令:
  ng generate pipe 管道名
  ng g p 管道名

  例如: ng g p sqrt
 -->

<!-- 制作获取 平方的 管道 -->
<p>{{ 81 | pingfang }}</p>
<p>{{ 15 | pingfang }}</p>
<!-- ng g p pingfang -->

<!-- 带有参数的管道 -->

<p>{{ 5 | pow: 2 }}</p>
<p>{{ 5 | pow: 5 }}</p>
<!-- 生成: ng g p pow -->

<!-- 带有默认参数的管道 -->
<!-- gender: 性别 -->
<!-- 0女性 1男性 2未知 -->
<p>{{ 1 | gender }}</p>
<p>{{ 0 | gender }}</p>
<p>{{ 2 | gender }}</p>
<!-- ng g p gender -->
<p>{{ 1 | gender: "en" }}</p>
<p>{{ 0 | gender: "en" }}</p>
<p>{{ 2 | gender: "en" }}</p>
<!-- 带有参数 'en'  则出现 Male  Female Unknown -->
```

性别管道

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

@Pipe({
```

```
  name: 'gender',
})
export class GenderPipe implements PipeTransform {
  // language: 语言, 简称 lang
  // 中文通常用 zh 表示
  transform(value: number, lang = 'zh') {
    // if (value == 0) return '女性';
    // if (value == 1) return '男性';
    // if (value == 2) return '未知';
    if (lang == 'zh') {
      return ['女性', '男性', '未知'][value];
    }
    // [][]   数组[下标]
    // 如果value=0,  则 是['女性', '男性', '未知'][0]

    if (lang == 'en') {
      return ['Female', 'Male', 'Unknown'][value];
    }
  }
}
```

平方管道

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

@Pipe({
  name: 'pingfang',
})
export class PingfangPipe implements PipeTransform {
  transform(value: number) {
    return value * value;
  }
}
```

次幂管道

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

@Pipe({
  name: 'pow',
})
export class PowPipe implements PipeTransform {
  // 语法糖 v | name : n : m
  // 系统会自动转化上方语法糖为:
  // new NamePipe().transform(v, n, m)

  // 参数名是自定义的, 随便起. 但是最好有含义   exp是指数的缩写
  transform(value: number, exp: number) {
    return Math.pow(value, exp);
  }
}
```

开平方

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

@Pipe({
  name: 'sqrt',
})
export class SqrtPipe implements PipeTransform {
  // {{ 81 | sqrt }}
  // 81 | sqrt  属于语法糖: 本质 new SqrtPipe().transform(81)
  transform(value: number) {
    return Math.sqrt(value);
  }
}
```

生命周期

什么是生命周期: 一个组件 从生成 到 最终销毁 之间经历的过程. 这个过程中的每个环节都会有对应的函数会被触发. 我们就可以在这些函数中进行对应的操作.

创建组件:

```
ng g c myc08
```

```
  ngOnDestroy: 组件即将销毁时                    myc08.component.ts:37
  constructor: 构造方法, 组件出生时第一时间调用    myc08.component.ts:12
  ngOnInit: 组件中的内容开始初始化                 myc08.component.ts:17
  ngAfterContentInit: 数据初始化完毕              myc08.component.ts:21
  ngAfterContentChecked: 数据发生变更             myc08.component.ts:29
  ngAfterViewInit: 用户界面初始化完毕             myc08.component.ts:25
  ngAfterViewChecked: 用户界面发生变更            myc08.component.ts:33
  ngAfterContentChecked: 数据发生变更             myc08.component.ts:29
  ngAfterViewChecked: 用户界面发生变更            myc08.component.ts:33
```

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

@Component({
  selector: 'app-myc08',
  templateUrl: './myc08.component.html',
  styleUrls: ['./myc08.component.css'],
})
export class Myc08Component implements OnInit {
  names = ['东东', '然然'];

  constructor() {
    console.log('constructor: 构造方法, 组件出生时第一时间调用');
  }
```

```
  // 类似于 vue 的mounted 方法, 挂载时, 发送网络请求
  ngOnInit(): void {
    console.log('ngOnInit: 组件中的内容开始初始化');
  }

  ngAfterContentInit(): void {
    console.log('ngAfterContentInit: 数据初始化完毕');
  }

  ngAfterViewInit(): void {
    console.log('ngAfterViewInit: 用户界面初始化完毕');
  }

  ngAfterContentChecked(): void {
    console.log('ngAfterContentChecked: 数据发生变更');
  }

  ngAfterViewChecked(): void {
    console.log('ngAfterViewChecked: 用户界面发生变更');
  }

  ngOnDestroy(): void {
    console.log('ngOnDestroy: 组件即将销毁时');
  }
}
```

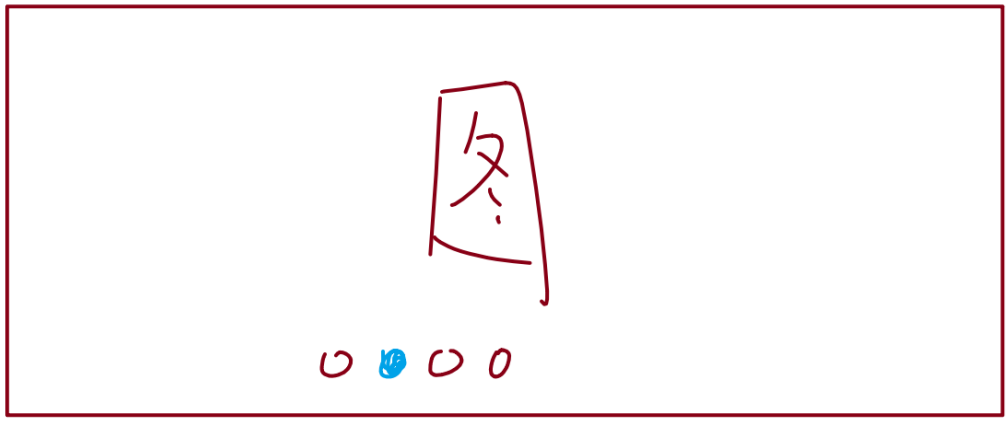
```
<p>myc08 works!</p>

<button (click)="names.push('亮亮')">新增</button>
<ul>
  <li *ngFor="let item of names">
    {{ item }}
  </li>
</ul>
```

作业

轮播图练习

- 图片存放在数组中, 使用网络图片或本地图片都可以
- 小圆点会随着页数发生变化
- 上一页 和 下一页按钮, 当达到 第一页 或最后一页时, 不可点击
- 中间的 1 2 3 4 页是通过循环生成的. 要求当前页的按钮不可点击. 即 如果是图3 则 页数3 不能点
- 可以自动切换, 每隔2.5秒, 自动切换到下一页
- 小圆点可以点击, 点击后切换到对应图片

> disabled 属性



上一页 1 2 3 4 下一页

代办事项练习

- 输入框为空 或者 都是空格时，则确认按钮不可点击
- 点击确定按钮, 可以把输入框的值添加到列表中 并 清空输入框
- 列表显示所有代办事项
- 列表的删除按钮 可以删除对应代办事项
- 当所有代办事项删除后, 则显示 `暂无代办事项` 字样

请输入代办事项 确定

* 吃饭 删除
* 睡觉 删除
* 打亮亮 删除

暂无代办事项