

Angular03

提前生成今日项目包

- `ng new ngpro`
- `ng s -o`

作业

作业1: 待办事项

组件名: work01

```
ng g c work01
```

- 123123
- 123123

```
<p>work01 works!</p>

<div>
  <input type="text" [(ngModel)]="todo" />
  <!-- trim(): 取消字符串左右的空格 -->
  <button (click)="doAdd()" [disabled]="todo.trim().length == 0">
    确认录入
  </button>
</div>

<ul>
  <!-- ng-for-li -->
  <li *ngFor="let item of items; let i = index">
    <span>{{ item }}</span>
    <!-- splice(序号,个数) 代表从 某序号开始 删除指定个数元素 -->
    <button (click)="items.splice(i, 1)">删除</button>
  </li>
</ul>

<h3 *ngIf="items.length == 0">暂无待办事项</h3>
```

```

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

@Component({
  selector: 'app-work01',
  templateUrl: './work01.component.html',
  styleUrls: ['./work01.component.css'],
})
export class Work01Component implements OnInit {
  items = ['吃饭', '睡觉', '打亮亮'];

  todo = '';

  doAdd() {
    this.items.push(this.todo);

    // 双向数据绑定：
    // 方向1：数据变 UI变 -- 响应式编程！
    // 响应式：UI 会响应 数据的变化，联动！
    this.todo = '';
  }

  constructor() {}

  ngOnInit(): void {}
}

```

```

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

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

import { AppComponent } from './app.component';
import { Work01Component } from './work01/work01.component';

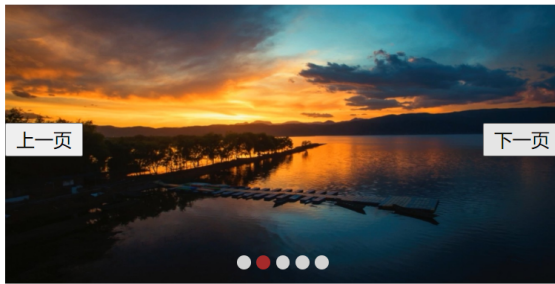
@NgModule({
  declarations: [AppComponent, work01Component],
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}

```

作业2

组件: work02

```
ng g c work02
```



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

@Component({
  selector: 'app-work02',
  templateUrl: './work02.component.html',
  styleUrls: ['./work02.component.css'],
})
export class Work02Component implements OnInit {
  images = ['01.jpg', '02.jpg', '03.jpg', '04.jpg', '05.jpg'];

  // 变量：变化的量
  // current page -> curP
  curP = 0; //当前序号

  constructor() {}

  // 生命周期，类似于 mounted 挂载时
  ngOnInit(): void {
    setInterval(() => {
      this.next();
    }, 2000);
  }

  next() {
    this.curP++;
    // if写法的 语法糖：{}中只有一行代码，则{}可以省略！
    // 例如：数组有5个元素，则序号是 0 1 2 3 4，即最大序号是4
    // 如果 序号+1之后的值 是5 就说明超出限制，则变为0
    if (this.curP == this.images.length) this.curP = 0;
  }

  prev() {
    this.curP--;
    // 最大序号 = 长度-1
    if (this.curP == -1) this.curP = this.images.length - 1;
  }
}
```

```
<p>work02 works!</p>

<div class="banner">
  <img [src]='"/assets/images/' + images[curP]" alt="" />

  <div class="banner-btns">
```

```

    <button (click)="prev()">上一页</button>
    <button (click)="next()">下一页</button>
  </div>

  <div class="banner-pages">
    <span
      *ngFor="let item of images; let i = index"
      [ngClass]="{ 'page-cur': i == curP }"
      (click)="curP = i"
    ></span>
  </div>
</div>

```

```

.banner {
  width: 400px;
  height: 200px;
  position: relative;
}

.banner > img {
  width: 100%;
  height: 100%;
}

.banner-btns {
  position: absolute;
  left: 0;
  right: 0;
  top: 50%;
  margin-top: -15px;
  display: flex;
  justify-content: space-between;
}

.banner-pages {
  position: absolute;
  left: 0;
  right: 0;
  bottom: 6px;
  text-align: center;
}

.banner-pages > span {
  display: inline-block;
  width: 10px;
  height: 10px;
  border-radius: 50%;
  background-color: lightgray;
  margin: 0 2px;
}

.page-cur {
  /* 最高优先级 */
  background-color: brown !important;
}

```

```
}
```

作业3

组件: work03

```
ng g c work03
```

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

@Pipe({
  name: 'gender',
})
export class GenderPipe implements PipeTransform {
  // <p>{{ 0 | gender }}</p>
  // <p>{{ 1 | gender: "en" }}</p>

  // 语言 language -> lang
  // 参数2: 可以传 也可以不传, 则必须拥有默认值来应对不传递的情况
  transform(value: number, lang = 'zh') {
    if (lang == 'zh') {
      if (value == 0) return '女';
      if (value == 1) return '男';
    }
    // let arr = ['women', 'Man'];
    // arr[value] value是0或1 即 arr[0] 或 arr[1]
    // 数组[下标]
    if (lang == 'en') return ['women', 'Man'][value];
  }
}
```

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

@Pipe({
  name: 'phoneHide',
})
export class PhoneHidePipe implements PipeTransform {
  // <p>{{ "18877889898" | phoneHide }}</p>

  transform(value: string) {
    // 小测试: 希望把手机号 中间4位 和 后4位 换位置
    // 18877779999 -> 18899997777
    // return value.replace(/(\d{3})(\d{4})(\d{4})/, '$1$3$2');

    // 18877889898 -> 188****9898
    // 思路: 截取 前3位 和 后4位, 重新拼接, 中间写 ****
    // 东哥: 正则表达式!
    // \d 代表数字 {n}代表n个 ()捕获组
```

```

    return value.replace(/(\d{3})\d{4}(\d{4})/, '$1****$2');
    //          第1个小括号    第2个小括号
    // $1 代表第1个小括号捕捉的内容
  }
}

```

```

<p>work03 works!</p>

<p>{{ "18877889898" | phoneHide }}</p>
<!-- 188****9898 -->
<!--
  管道生成命令: ng generate pipe 管道名
  简写: ng g p 管道名
  此处: ng g p phoneHide
-->

<p>{{ 1 | gender }}</p>
<p>{{ 0 | gender }}</p>
<p>{{ 1 | gender: "en" }}</p>
<p>{{ 0 | gender: "en" }}</p>
<!-- ng g p gender -->

<p appHide>123456</p>
<!--
  指令: ng g d 指令名
  此处: ng g d hide

  appHide 中, app是自定义指令的固定前缀
-->

```

作业4

继续使用 组件 work03

```

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

@Directive({
  selector: '[appHide]',
})
export class HideDirective {
  constructor(e: ElementRef) {
    console.log(e);
    // 通过看后台打印, 可以看到能够操作的属性
    e.nativeElement.style.backgroundColor = 'red';
    e.nativeElement.style.display = 'none';
  }
}

```

TypeScript

官方: <https://www.tslang.cn/>

TypeScript是JavaScript 的超集: 在js 基础上新增了更多的语法和特性.

由 微软公司 进行维护与开发!

浏览器默认只能运行 JavaScript语言.

ts语言对程序员在书写代码时非常友好:

- 提供丰富的代码提示
 - 代码书写更加快速 简单
- 提供丰富的报错提示
 - 可以第一时间发现错误并进行解决

类似于ES6语法, 程序员使用时简单快捷, 但是低版本浏览器不兼容: 在运行时要使用 babel工具编译为普通js

TS语言, 程序员使用时很强大, 但是运行到浏览器上 需要先编译为普通的js

编译工具是 TS 专门的工具, 需要安装:

- **-g** 代表是全局安装
- 确保你是中国镜像 和 环境变量的正确: 参考 day01 中的文档!

```
npm i -g typescript
```

```
+ typescript@3.9.7
added 1 package from 1 contributor in 1.704s
```

编译命令:

```
tsc xxxx.ts
```

```
// ts 文件的最基本特征: 静态类型
// 特点: 代码不需要运行, vscode 就可以通过程序员 给的提示 去分析代码 并提供 报错 or 代码提示

// 变量:类型名    这就是类型声明
// 有了声明以后, vscode 就可以帮你分析代码
function show(name: string) {
    // name. 会出现代码提示:  vscode自动分析变量类型并给出提示!
    return name.toUpperCase();
}

show("dongdong");

// 报错提示: vscode会自动分析 类型是否匹配. 提前报错, 扼杀在萌芽里!
show(123);
```

```
// 编译命令: tsc 文件名.ts  
// 例如 tsc 01.ts
```

```
// 静态类型的声明
```

```
class Demo {  
  name: string;  
  age: number; //浮点+整型  
  
  married: boolean;  
  
  abc: any; //任意类型  
  
  // 数组: []代表数组 string代表数组中的元素都是string类型  
  names: string[];  
  // 等价上方写法, 但是略长, 不常用  
  emps: Array<string>;  
  /**  
   * 变量名:类型名 = 值;  
   */  
  
  boss: object;  
}  
  
let d = new Demo();  
  
d.boss = { name: "华华" };  
d.boss = 123;  
  
d.names = ["mike", "lucy", true, 123];  
  
d.age = 123;  
d.age = 123.44;  
// d.age = "xx";  
  
d.married = true;  
d.married = false;  
  
d.abc = 123;  
d.abc = true;
```

```
// 自定义的对象类型
```

```
// 此处有一个数据类型
```

```
let emp = {  
  name: "亮亮",  
  age: 18,  
  phone: "18877894455",  
}
```



```

};
// 此对象类型含有3个属性名

/**
 * function 声明函数
 * class 声明类
 * interface 声明自定义对象类型
 */

interface Emp {
  name: string;
  age: number;
  phone: string;
}

// 如果有一个数组，要求数组中的元素 必须是 含有指定属性结构的对象，例如
let emps: Emp[]; //数组中的元素都是 Emp 类型的

emps = [
  { name: "东东", age: 12, phone: "17878889999" },
  { name: "亮亮", age: 12, phone: "17878889999" },
  { name: "然然", age: true },
];

```

```

/**
 * 访问权限词
 *
 * public : 公开的 -- 类外 类内 子类
 * protected : 保护的 -- 类内 子类
 * private : 私有的 -- 类内
 */

class Demo {
  // 公开的
  public paper = "渣渣坤的手纸";
  // 保护的
  protected money = "渣渣坤的钱";
  // 私有的
  private secret = "年轻时犯过的错误!";

  show() {
    console.log(this.money);
  }
}

// 面向对象三大特征：封装,继承,多态
class Son extends Demo {
  show() {
    this.paper;
    this.money;
    this.secret;
  }
}

```

```
}

let d = new Demo();
d.paper;

// 类外无法读取保护属性
// d.money;
```

生命周期

一个组件从出生到销毁整套过程中 经历的各种钩子函数

<https://angular.cn/guide/cheatsheet>



组件: myc01

```
ng g c myc01
```

constructor: 组件对象生成时 第一时间触发	myc01.component.ts:15
ngOnInit: 组件中的数据开始初始化	myc01.component.ts:20
ngAfterContentInit: 组件内容中的数据开始初始化时	myc01.component.ts:29
ngAfterContentChecked: 组件内容中的数据变化时	myc01.component.ts:33
ngAfterViewInit: UI视图初始化	myc01.component.ts:37
ngAfterViewChecked: UI视图变化时	myc01.component.ts:41
ngAfterContentChecked: 组件内容中的数据变化时	myc01.component.ts:33
ngAfterViewChecked: UI视图变化时	myc01.component.ts:41
ngOnDestroy: 组件销毁时	myc01.component.ts:47

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

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

  // 明天看一看东哥的面向对象
  // 构造方法: new 类(); 时自动触发
```

```

constructor() {
  // 1. 刚怀孕
  console.log('constructor: 组件对象生成时 第一时间触发');
}

ngOnInit(): void {
  // 2. 开始生长
  console.log('ngOnInit: 组件中的数据开始初始化');

  setInterval(() => {
    this.now = new Date().toLocaleTimeString();
  }, 1000);
}

// 数据变化 UI变化
ngAfterContentInit(): void {
  console.log('ngAfterContentInit: 组件内容中的数据开始初始化时');
}

ngAfterContentChecked(): void {
  console.log('ngAfterContentChecked: 组件内容中的数据变化时');
}

ngAfterViewInit(): void {
  console.log('ngAfterViewInit: UI视图初始化');
}

ngAfterViewChecked(): void {
  console.log('ngAfterViewChecked: UI视图变化时');
}

// 销毁
ngOnDestroy(): void {
  //组件销毁时
  console.log('ngOnDestroy: 组件销毁时');
}
}

```

服务

相当于vue的 vuex: 全局状态管理

用于在组件之间分享数据;

新建组件: myc02 和 myc03

```

ng g c myc02
ng g c myc03

```

服务生成命令: 数据服务 服务于组件, 为组件提供数据

生成服务并不会更新配置, 所以不需要重启

```
ng generate service 服务名
```

简写: `ng g s 服务名`

```
ng g s movie
```

依赖注入机制:

```
/**
 * 依赖注入机制
 *
 * 在日常生活中非常常见:
 *
 * 例子1: 超市门口的摇摇车: 有说明, 1元1次. 你玩的时候就必须投币1元
 * * 声明依赖: 要玩 必须投1元
 * * 注入: 玩的时候, 就投1元
 */

// 代码中的声明依赖写法
class Demo {
  name: string;

  //构造方法: 声明此类 初始化时必须传递一个参数, 必须是字符串类型
  // 声明依赖
  constructor(name: string) {
    this.name = name;
  }
}

//使用时: 要想初始化 必须满足依赖的条件: 注入一个 字符串类型的值
new Demo('dongdong');
```

myc02.component.ts

```
// 1. 引入
import { MovieService } from '../movie.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-myc02',
  templateUrl: './myc02.component.html',
  styleUrls: ['./myc02.component.css'],
})
export class Myc02Component implements OnInit {
  ms: MovieService;

  // 2. 使用
  // 声明依赖: 当前组件初始化的必备条件: 传入一个 MovieService 类型的值
  // 注入: 当系统初始化当前组件时, 就会受到依赖注入机制的影响, 必须传入一个 MovieService 类型的值
  // 使用时: 我们负责声明依赖, 系统负责自动注入操作!
```

```
// 系统隐式完成：
// let ms = new MovieService()
// new Myc02Component(ms)

// 变量名是你自己起的，MovieService 所以老师简写成 ms
constructor(ms: MovieService) {
  this.ms = ms;
}

ngOnInit(): void {}
}
```

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

<ul>
  <li *ngFor="let item of ms.movies; let i = index">
    <span>{{ item }}</span>
    <button (click)="ms.movies.splice(i, 1)">删除</button>
  </li>
</ul>
```

movie.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class MovieService {
  movies = [
    '阿凡达',
    '阿甘正传',
    '复仇者联盟',
    '林中小屋',
    '我是传奇',
    '僵尸世界大战',
  ];

  constructor() {}
}
```

服务练习

- 新的组件: myc04 然后显示到 页面上
- 创建服务: names

```
ng g s names
```

其中存放一个数组

```
names = ['亮亮', '东东', '然然']
```

- 在 myc04 组件中 通过依赖注入方式 引入 names 服务对象
- 把姓名显示到页面中, 列表方式即可

语法糖

```
import { NamesService } from '../names.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-myc05',
  templateUrl: './myc05.component.html',
  styleUrls: ['./myc05.component.css'],
})
export class Myc05Component implements OnInit {
  // 服务依赖注入的语法糖
  // 语法糖: 某些功能实现的代码是固定的, 所以为了偷懒, 使用一些快捷写法 来代替固定的代码.
  // 糖: 外国人喜欢吃甜食, 认为甜是幸福的味道.. 利用语法偷懒也很幸福
  // 例如:
  // if (xxx) { return xxx; }    ->  if (xxx) return xxx;
  // () => { return xxx; }    ->  () => xxx;
  //
  // let obj = {name:'东东', age: 18};
  // 写法1: let name=obj.name; let age=obj.age;
  // 语法糖: let {name, age} = obj;
  constructor(public ns: NamesService) {}
  // 固定的简化写法:
  // 权限词 变量名: 注入的类型名
  // 权限词在此处随意写, public protected private 都不影响. 因为此变量在类内使用, 无差别

  ngOnInit(): void {
    // this.
  }
}
```

网络请求服务

系统默认提供了很多完善的服务, 其中就包含网络服务.

- 新建组件: myc06
- 网络服务必须手动开启, 默认不加载此模块! -- **重启服务器**
 - 修改 **app.module.ts**

```

20 | // 网络服务模块
21 | import {HttpClientModule} from '@angular/common/http'; (1)
22 |
23 | @NgModule({
24 |   declarations: [AppComponent, Work01Component, Work02Component, Work03Component,
25 |     PhoneHidePipe, GenderPipe, HideDirective, Myc01Component, Myc02Component,
26 |     Myc03Component, Myc04Component, Myc05Component, Myc06Component],
27 |   imports: [BrowserModule, FormsModule, HttpClientModule],
28 |   providers: [],
29 |   bootstrap: [AppComponent],
30 | })
31 | export class AppModule {}

```

```

import { Component, OnInit } from '@angular/core';
// 1.引入网络服务
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-myc06',
  templateUrl: './myc06.component.html',
  styleUrls: ['./myc06.component.css'],
})
export class Myc06Component implements OnInit {
  res: any; //只有成员变量才能 在html中使用

  // 2.声明依赖
  constructor(public http: HttpClient) {}

  ngOnInit(): void {
    //免费接口地址: https://api.apiopen.top/api.html

    //发送get请求
    let url = 'https://api.apiopen.top/getImages';
    // axios.get().then().catch()
    // 此处用 subscribe(订阅) 代替 then(然后)
    this.http.get(url).subscribe((res) => {
      console.log(res);
      this.res = res; //保存返回值到 成员变量中, 才能在html中使用
    });
  }
}

```

<p>myc06 works!</p>

<!-- 免费接口, 有些图不存在, 后台爆红正常! 404是正常报错 -->

<!-- 网络请求是异步的, 请求完毕之前 res 是 undefined, 所以此处会报错 -->

<!-- 我们可以判断只有 res 不是 undefined 的时候 才使用 -->


```

<div *ngIf="res">
  <img
    *ngFor="let item of res.result"
    [src]="item.img"
    alt=""
    width="100"
    height="150"
  >

```

```
/>  
</div>
```

js快捷代码块插件



JavaScript Snippets

nathanchapman.javascriptsnippets

Nathan Chapman | 42,718 | ★★★★★ | 存储库 | v0.2.0

JavaScript (ES6) code snippets for VS Code

[禁用](#) [卸载](#) 此扩展已全局启用。


细节 功能贡献

```
}}, ${1:delay});
```

si→ setInterval

```
setInterval(() => {  
    ${0}  
}, ${1:delay});
```

自动引入: 当使用了一个没有引入的类时, 会自动引入




Auto Import

1.5.3

Automatically finds, parses and provides code actions and code completion for...

steoates

1M ★ 4.5



作业

制作网易新闻

接口地址: <https://api.apioopen.top/getWangYiNews>

图	标题
	时间

每一条新闻的样式