Angular02

生成今日使用的项目包:

```
$ ng new ngpro
```

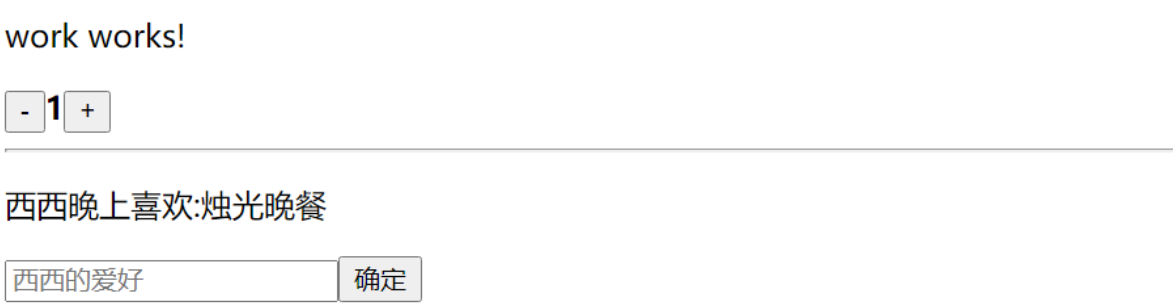
启动项目

```
$ ng s -o
```

编程最怕: **眼高手低** 一看就会 一写就废;

本阶段必备技能: **面向百度**编程

作业



work.ts

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

@Component({
  selector: 'app-work',
  templateUrl: './work.component.html',
  styleUrls: ['./work.component.css'],
})
export class WorkComponent implements OnInit {
  // 昨天说过的插件 prettier 配合保存时自动格式化
  // 按 ctrl+s保存时, 就会自动补全分号
  num = 1;

  hobby = '';
  like = '';

  saveLike() {
    this.like = this.hobby;
    this.hobby = '';
  }

  doMinuse() {
```

```
    if (this.num == 1) {
      alert('数量不能少于1');
      // return 可以终止代码执行
      return;
    }

    this.num = this.num - 1;
  }

  // 作业3:
  price = 7999;
  count = 1;

  // 作业4:
  images = [
    'https://tse3-mm.cn.bing.net/th/id/OIP.jwyAo-TrKtQj6K2AVVzxiAHaEK?
pid=Api&rs=1',

 'http://5b0988e595225.cdn.sohucs.com/images/20190126/e5ae4556b52042d3aa6355ef7d
0efbdd.jpeg',
    'https://tse3-mm.cn.bing.net/th/id/OIP.jyqDbZiaJiVNCONjcbxhXgHaEK?
pid=Api&rs=1',
    'https://tse3-mm.cn.bing.net/th/id/OIP.MASmSGUEiRwOc86zsRb1OwHaEK?
pid=Api&rs=1',
  ];

  // current page : 当前页  缩写 curp
  curp = 1; //记录当前页, 从1开始

  constructor() {}

  ngOnInit(): void {}
}

/**
 * 观察下方代码格式, 说出max的类型
 *
 * 1. max  全小写 变量
 * 2. Max  大驼峰 类名
 * 3. MAX  全大写 常量
 * 4. max() 小写带()  函数
 * 5. this.max()  带this开头, 带():  成员方法 类中的
 * 6. this.max   带this 不带();  成员属性, 类中的
 * 7. new Max()  带new 大驼峰   类
 * 8. 'max'      带引号 字符串
 *
 */
```

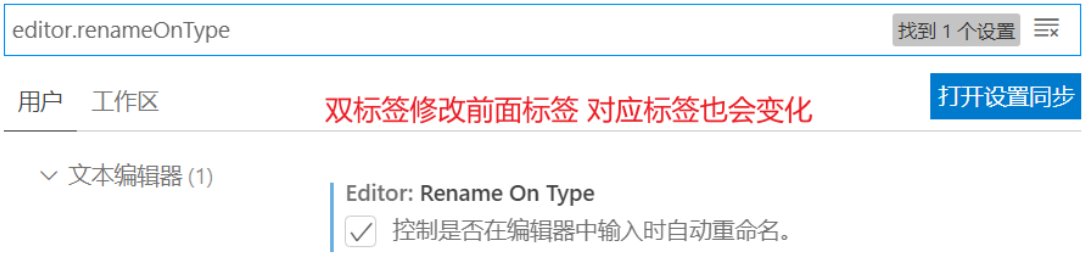
work.html

```
<p>work works!</p>

<!-- 作业1: 计数器 -->
<!-- ctrl+回车 -->
<button (click)="doMinuse()">-</button>
<b>{{ num }}</b>
```

```
<!--
  属性用 []   -- vue :
  事件用 ()   -- vue @
 -->
<button [disabled]="num == 20" (click)="num = num + 1">+</button>

<hr />

<!-- 作业2: -->
<p>西西晚上喜欢:{{ like }}</p>
<!--
  默认 angular 不加载form模块, 导致双向绑定不可用!
  手动加载即可:  app.module.ts
-->
<input type="text" placeholder="西西的爱好" [(ngModel)]="hobby" />
<button (click)="saveLike()">确定</button>

<hr />
<!-- 作业3:  -->
<p>商品名: iPhone</p>
<div>
  <span>商品价格:</span>
  <input type="text" [(ngModel)]="price" />
</div>
<div>
  <span>商品数量:</span>
  <button (click)="count = count - 1" [disabled]="count == 1">-</button>
  <b>{{ count }}</b>
  <button (click)="count = count + 1">+</button>
</div>
<p>总价格: {{ price * count }}</p>

<!-- 作业4 -->

<hr />
<!-- curp起始值为1;   下标取值从0开始 -->
<img [src]="images[curp - 1]" alt="" width="200" height="100" />
<div>
  <!-- 当前页时1, 则变为最后一页 -->
  <button (click)="curp = curp == 1 ? images.length : curp - 1">上一页</button>
  <!-- 当curp为最后一页, 则变回1 -->
  <button (click)="curp = curp == images.length ? 1 : curp + 1">下一页</button>
</div>
```

插件:

- 修改双标签时, 会自动联动

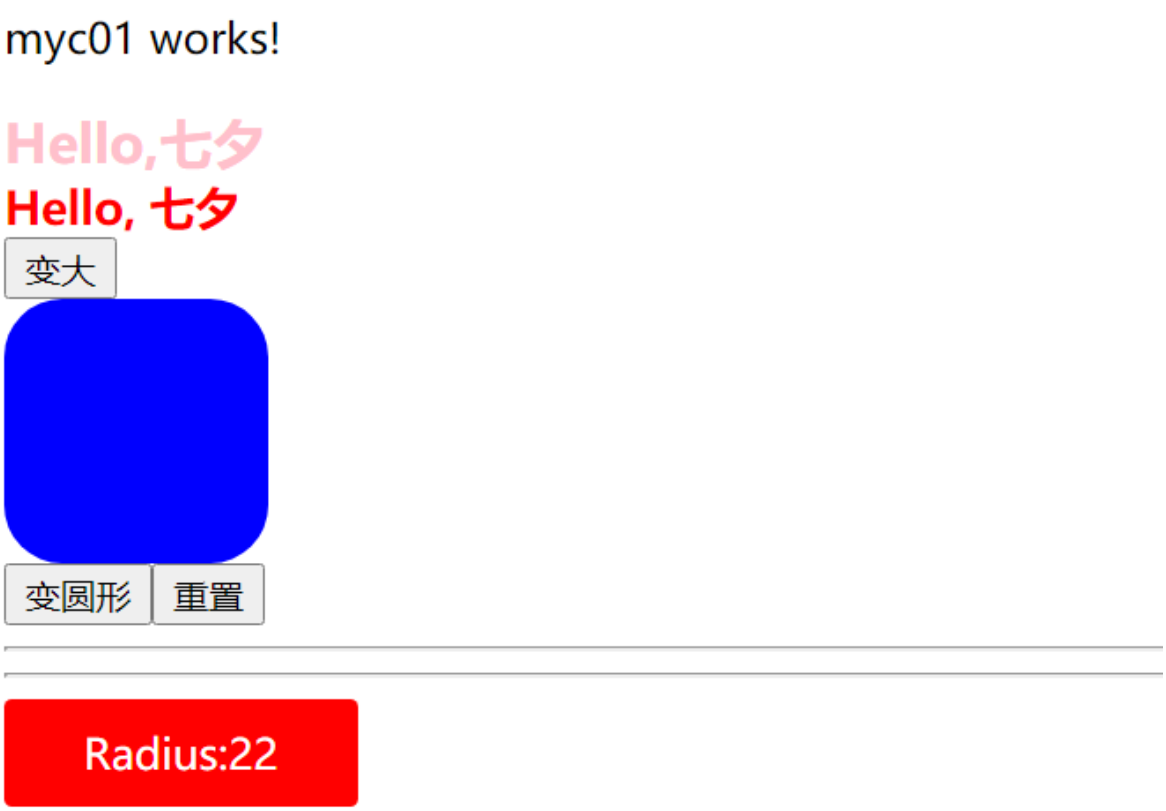


今日内容

风格样式属性

创建组件: `myc01`

```
$ ng g c myc01
```



ts

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

@Component({
  selector: 'app-myc01',
  templateUrl: './myc01.component.html',
  styleUrls: ['./myc01.component.css'],
})
export class Myc01Component implements OnInit {
  size = 17;
  // 半径
  radius = 0;
```

```
  constructor() {}

  ngOnInit(): void {}
}
```

html

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

<div>
  <b style="font-size: 20px; color: pink;">Hello,七夕</b>
</div>

<div>
  <!-- ng-style: 值为对象类型 -->
  <!-- js的属性名不允许中划线出现: 所以css中 font-size 要改成小驼峰 或 字符串 -->
  <b [ngStyle]="{ color: 'red', fontSize: size + 'px' }">Hello, 七夕</b>
</div>

<!-- 需求: 点击按钮 让文字变大 -->
<button (click)="size = size + 2">变大</button>

<div
  [ngStyle]="{
    width: '100px',
    height: '100px',
    'background-color': 'blue',
    borderRadius: radius + 'px'
  }"
></div>
<button (click)="radius = radius + 2">变圆形</button>
<button (click)="radius = 0">重置</button>

<hr />
<!-- radius<=10   显示normal样式 -->
<!-- radius>10 并 小于20   显示success样式 -->
<!-- radius>=20   显示danger样式 -->
<hr />

<!--
  与vue中的设计完全相同:  vue中  :class={样式名: true/false}
 -->
<span
  style="display: inline-block; border-radius: 3px;"
  [ngClass]="{
    normal: radius <= 10,
    success: radius > 10 && radius < 20,
    danger: radius >= 20
  }"
  >Radius:{{ radius }}</span
>
```

css

```
.danger {
  background-color: red;
  color: white;
  padding: 10px 30px;
}

.success {
  background-color: green;
  color: white;
  padding: 10px 30px;
}

.normal {
  background-color: lightblue;
  color: white;
  padding: 10px 30px;
}
```

条件渲染

if写法

生成组件: `myc02`

```
$ ng g c myc02
```

ts

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

@Component({
  selector: 'app-myc02',
  templateUrl: './myc02.component.html',
  styleUrls: ['./myc02.component.css'],
})
export class Myc02Component implements OnInit {
  // 分数
  score = 60;

  constructor() {}

  ngOnInit(): void {}
}
```

html

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

<!--
  条件渲染:
  vue中: v-if
  ng中: *ngIf
 -->
<button (click)="score = score - 10">减分</button>
<button (click)="score = score + 10">加分</button>
<h3>西西的面试分数: {{ score }}</h3>

<p *ngIf="score < 60">今天面试就到这里, 回去等消息吧</p>
<p *ngIf="score >= 60 && score <= 80">一会跟我们的人力资源同事聊一聊</p>
<p *ngIf="score > 80">请稍等, 我们老板来和你聊一聊</p>

<!-- if-else 写法 -->
<hr />
<ng-container *ngIf="score < 60; else xyz">
  <b>不及格</b>
</ng-container>
<!-- # 是 css中的id选择器写法;  而此处作者借鉴了这种id的快捷写法 -->
<!-- #elseTemplate 相当于 id='elseTemplate' -->
<ng-template #xyz>
  <b>及格</b>
</ng-template>
```

循环遍历

新建组件: `myc03`

```
$ ng g c myc03
```

ts

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

@Component({
  selector: 'app-myc03',
  templateUrl: './myc03.component.html',
  styleUrls: ['./myc03.component.css'],
})
export class Myc03Component implements OnInit {
  names = ['东东', '亮亮', '然然', '西西', '楠楠', '贝贝'];

  emps = [
    { name: '东东', age: 18, phone: '18844559988', sex: 1 },
    { name: '然然', age: 19, phone: '18889559988', sex: 1 },
    { name: '西西', age: 20, phone: '18899559988', sex: 0 },
    { name: '贝贝', age: 21, phone: '18800559988', sex: 1 },
    { name: '楠楠', age: 22, phone: '18822559988', sex: 0 },
  ];

  constructor() {}
```

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

html

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

<!--
  循环展示:
  vue: v-for="(item,index) in arr" :key="index"
 -->

<ul>
  <!-- ng-for-li -->
  <!-- 不同于 小程序;vue  不需要写key -->
  <li *ngFor="let item of names">
    <span>{{ item }}</span>
  </li>
</ul>

<hr />

<ul>
  <!-- ng-for-index -->
  <!-- 带有序号的写法 -->
  <li *ngFor="let item of names; let i = index">
    <span>序号: {{ i }}</span>
    <span>. {{ item }}</span>
  </li>
</ul>

<hr />

<table border="1">
  <tr>
    <td>序号</td>
    <td>姓名</td>
    <td>年龄</td>
    <td>手机号</td>
    <td>性别</td>
  </tr>
  <tr *ngFor="let item of emps; let i = index">
    <td>{{ i + 1 }}</td>
    <td>{{ item.name }}</td>
    <td>{{ item.age }}</td>
    <td>{{ item.phone }}</td>
    <td>{{ item.sex }}</td>
  </tr>
</table>
```

自定义指令

可以自定义一个属性: 添加到任意的元素上之后, 此元素就会自动变化DOM属性

> 例如: 我们可以制作一个神奇的戒指, 带在谁手上 谁就是你的女朋友!
>
> 你还可以制作一个神奇的帽子, 带在谁头上..

制作组件: `myc04`

```
$ ng g c myc04
```

自定义指令命令

> 指令生成之后, 最好重启服务器; 有的时候不会自动加载新指令

```
$ ng generate directive 指令名

简写:
$ ng g d 指令名
```

例如: blue指令, 让双标签的元素变为蓝色

```
$ ng g d blue
```

```
D:\WEBTN2004\17_Angular\Day02\ngpro>ng g d blue
CREATE src/app/blue.directive.spec.ts (216 bytes) spec.ts 都是无用的单元测试
CREATE src/app/blue.directive.ts (137 bytes) 指令文件
UPDATE src/app/app.module.ts (835 bytes) 指令会自动注册为全局指令
```

myc04.component.html

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

<ul>
  <li appBlue>亮亮</li>
  <!-- blue是一个自定义属性, 在ng中称为指令 -->
  <!-- 需要自定义: ng g d blue -->
  <li>然然</li>
  <li>东东</li>
  <!-- 自定义 hide 指令, 可以隐藏某个元素 -->
  <!-- * ng g d hide -->
  <!-- * style 中有 display 的属性,  none 代表隐藏 -->
  <li appHide>华哥</li>
</ul>

<hr />
<!-- 输入框自动焦点 -->
<div>
  <input type="text" />
</div>
<div>
  <!-- input元素的 focus() 方法可以获取焦点状态 -->
  <!-- ng g d focus -->
```

```
  <input type="text" appFocus />
</div>
<div>
  <input type="text" />
</div>
```

blue.directive.ts

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

@Directive({
  // 生成的指令名 会自带 App前缀 防止与系统自带的冲突
  selector: '[appBlue]',
})
export class BlueDirective {
  // 指令所在的元素, 会自动作为参数传递到构造方法中
  constructor(e: ElementRef) {
    console.log(e);
    e.nativeElement.style.color = 'blue';
    e.nativeElement.style.fontSize = '30px';
  }

  // TS语言: 具备的最重要的特性 -- 静态类型分析
  // 变量:类型名;   我们的vscode会根据用户给的类型 来给出智能的代码提示!
  show(name: string) {
    // name.pus
  }
}
```

focus

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

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

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

hide

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

@Directive({
  selector: '[appHide]',
})
export class HideDirective {
  constructor(e: ElementRef) {
    console.log(e);
    e.nativeElement.style.display = 'none';
  }
}
```

管道

管道就是vue中的过滤器: `filter`

管道: pipe

```
<tag>{{ 值 | 管道名 }}</tag>
```

不同于vue: vue本身不提供任何过滤器, 都需要自定义!

angular官方提供了一些比较常用的管道

新建组件

```
$ ng g c myc05
```

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

<!-- 提供提供的管道 -->
<p>变大写: {{ "hello xixi" | uppercase }}</p>
<p>变小写: {{ "WEB MEMEDA" | lowercase }}</p>
<p>首字母大写: {{ "nice to meet you" | titlecase }}</p>
<p>百分数: {{ 0.78555 | percent }}</p>
<!-- 2.2  最少整数位.保留的小数位 -->
<p>百分数: {{ 0.78555 | percent: "2.2" }}</p>
<p>百分数: {{ 0.085 | percent: "2.2" }}</p>
<p>千分位的钱: {{ 123456.789 | currency }}</p>
<p>千分位的钱: {{ 123456.789 | currency: "¥" }}</p>
<!-- 时间戳: 当前时间距离1970年1月1日的秒数 -->
<!-- 单位: 毫秒 -->
<p>日期: {{ 1598340423355 | date }}</p>
<!--
  year:  年 y
  month: 月 M
  day:   日 d
  hour:  时 h12小时  H24小时
  minute:分 m
  second:秒 s
 -->
<p>日期: {{ 1598340423355 | date: "yyyy-MM-dd HH:mm:ss" }}</p>
```

```
<p></p>
<p></p>
<p></p>
```

自定义管道 Pipe

生成组件:

```
$ ng g c myc06
```

生成管道的命令

```
$ ng generate pipe 管道名

简写:
$ ng g p 管道名
```

```
D:\WEBTN2004\17_Angular\Day02\ngpro>ng g p pow
CREATE src/app/pow.pipe.spec.ts (175 bytes)
CREATE src/app/pow.pipe.ts (211 bytes)自动生成
UPDATE src/app/app.module.ts (1163 bytes)自动注入全局
```

html

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

<!-- 管道: 修改双标签的值 -->
<!-- ng g p pow -->
<p>{{ 5 | pow }}</p>
<p>{{ 9 | pow }}</p>

<!-- 练习 -->
<!-- 1 出现男  0出现女 -->
<!-- ng g p gender -->
<p>{{ 1 | gender }}</p>
<p>{{ 0 | gender }}</p>

<!-- 带参数的管道 -->
<!-- 生成管道: ng g p love -->
<!-- 生成之后 重启 服务器 -->
<p>{{ "然然" | love: "小乔" }}</p>
<!-- 然然 喜欢 小乔 -->
<p>{{ "然然" | love: "亮亮" }}</p>

<!-- 练习 -->
<!-- ng g p fear -->
<p>{{ "然然" | fear: "宝宝":"绿色" }}</p>
<!--
  固定2个参数: 出现 然然害怕xx和xx
 -->
```

```
<!-- 带默认值的管道: 生成 ng g p like   重启 -->
<p>{{ "西西" | like }}</p>
<!-- 出现: 西西最喜欢 然然 老师 -->
<p>{{ "西西" | like: "东东" }}</p>
<!-- 出现: 西西最喜欢 东东 老师 -->
```

pow.pipe.ts

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

@Pipe({
  name: 'pow',
})
export class PowPipe implements PipeTransform {
  // <p>{{ 9 | pow }}</p>
  // 管道的写法: {{value | 管道名}}
  // value就会作为参数1 传递到下方的方法中!
  transform(value: number) {
    // 返回值就是 管道处理的结果
    return Math.pow(value, 2);
  }
}
// 新的管道:  要重启服务器才能生效!
```

love.pipe.ts

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

@Pipe({
  name: 'love',
})
export class LovePipe implements PipeTransform {
  // {{ "然然" | love: "小乔" }}
  // {{ 值 | 管道名: 参数2: 参数3: 参数4...}}
  transform(value: string, lover: string) {
    return value + '喜欢' + lover;
  }
}
```

fear.pipe.ts

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

@Pipe({
  name: 'fear',
})
export class FearPipe implements PipeTransform {
  // {{ "然然" | fear: "宝宝":"绿色" }}
  transform(value: string, name1: string, name2: string) {
    return value + '害怕' + name1 + '和' + name2;
  }
}
```

gender.pipe.ts

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

@Pipe({
  name: 'gender',
})
export class GenderPipe implements PipeTransform {
  // {{0|gender}}
  transform(value: number) {
    // if (value == 0) return '女';
    // if (value == 1) return '男';

    // arr[index]:  利用下标来取巧
    return ['女', '男'][value];
  }
}
```

带有默认值的 like

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

@Pipe({
  name: 'like',
})
export class LikePipe implements PipeTransform {
  // {{ "西西" | like: "东东" }}
  // {{ "西西" | like }}
  transform(value: string, name = '然然') {
    return value + '最喜欢' + name + '老师';
  }
}
```

TypeScript

官方: https://www.tslang.cn/

TypeScript是 JavaScript 的一个超集: 在JS的基础上进行了完善, 新增了很多新特性;

由微软公司进行开发并维护

```
浏览器默认只支持 HTML CSS JS
TS语言必须翻译成JS之后 才能在浏览器上执行:

类似于 ES6 语法在旧版本浏览器上无法识别, 运行时必须用 babel 来编译
```

安装 TS 的编译工具

```
npm i -g typescript
```

```
+ typescript@4.0.2
added 1 package from 1 contributor in 4.103s
```

查看版本号

```
$ tsc -v
```

编译ts为js文件

```
// 体验 静态类型 分析 和 编译操作

// 静态类型分析:  通过 变量:类型名 写法来告知 vscode 变量是什么类型的
// vscode就可以在 代码未运行的情况下 提前预判代码
function show(name: string) {
  // name.push();

  return name.toLowerCase();
}

console.log(show("HELLO"));

// 编译成JS之后运行
// 编译命令: tsc 文件名.ts

/**
 * 优点:
 * 写代码的时候 有充足的代码提示
 * 并且 可以预判出代码中的错误 给出提示
 * 对于程序员非常友好
 */
```

基础类型

```
// 可以设置的类型

class Emp {
  // 变量:类型 = 值
```

```
  name: string = "东东";

  // 数字类型: 整数 浮点
  age: number;

  // 布尔类型
  married: boolean;

  // 任意类型
  abc: any;

  // 多种类型: |代表或
  xyz: number | string;

  // 数组: 下方两种写法是等价的: 代表是 数组类型, 其中的值都是字符串
  names: Array<string>;
  words: string[];

  show() {
    // this.name.
    // this.age.
    // this.married = 1;
    this.abc = 12;
    this.abc = true;

    this.xyz = 12;
    this.xyz = "123";

    this.names = ["mike", "lucy", true, 123];
    this.words = ["mike", "lucy", true, 123];
  }
}
```

自定义类型

```
// 自定义数据类型

// 例如: 对象类型, 其中有name age sex 3个属性

// 新的关键词: interface 接口, 此处用于定义对象类型
// interface 和 function  class 都是关键词
interface User {
  name: string;
  age: number;
  sex: number;
}

// 变量 xixi 是 User 类型
let xixi: User = {
  name: "西西",
  age: 18,
  sex: 0,
};
```

```
// 男朋友类型: 类型名 BoyFriend
// 必备属性: house money car

// 类型声明
interface BoyFriend {
  house: string;
  money: number;
  car: string;
}

// 对象类型 Object
let san: BoyFriend = {
  house: "上海明珠塔前200平",
  money: 98888,
  car: "...",
};
```

访问控制词

```
// 面向对象中的 访问控制词
/**
 * 在绝大多数的面向对象语言中都存在 控制词
 * public 公开的
 * protected 保护的
 * private 私有的
 */

//  三个角色: 本人 外人 儿子
class Demo {
  // 公开的
  public name = "唐三";
  // 保护的
  protected money = "唐三的钱";
  // 私有的
  private avi = "唐三年轻时珍藏的电影";
}

// extends 继承
class Son extends Demo {
  show() {
    this.name; //公开的, 子类可以访问
    this.money; //保护的, 子类可以
    // this.avi;
  }
}

let san = new Demo();
san.name;
// 类外读
// san.money;
// san.avi;
```

作业

作业1: 轮播图

- 需要一个数组, 保存多张图片的地址
- 一个curp属性来代表当前页
- 循环数组, 来生成小圆点, 通过判断当前页 让小圆点具备不同的样式
    - 当前页的小圆点会变色
- 小圆点可以点击, 实现图片的变化

作业2: 待办事项

- 点击确定按钮, 把输入框的内容添加到下方列表中, 并清空输入框内容
- 删除按钮可以点击, 删除对应项目
- 当所有项目删除后, 则显示 暂无待办事项
- 输入框没有值的时候, 确定按钮无法点击: disabled

作业3: 自定义指令

```
<p appMore>西西</p>

可以让西西的 字变大 40px， 颜色变为橘黄色， 背景色变为紫色
```

作业3: 自定义管道

```
<p>{1 | sexy}</p>  出现 男
<p>{1 | sexy:'en'}</p> 出现 Male
<p>{1 | sexy:'full'}</p> 出现 男性

<p>{0 | sexy}</p>  出现 女
<p>{0 | sexy:'en'}</p> 出现 FeMale
<p>{0 | sexy:'full'}</p> 出现 女性

<p>{2 | sexy}</p>  出现 保密
<p>{2 | sexy:'en'}</p> 出现 Security
<p>{2 | sexy:'full'}</p> 出现 秘密
```