Typescript

只读属性 readonly ; interface point{ readonly x:number }; //x只读

ReadonlyArray<number> = a;

**function** **f**(this: void) { // 确保`this`在这个独立的函数中无法使用 }

一、基本类型：

布尔：let isDone:Boolean=true;

数字：let n1:number=3;//十进制，十六进制，八进制，二进制

字符串：let name:string=’lilei’; let age:string=’my age is ${age + 1}’;

数组：let list:number[]=[1,2,3]; let list:Array<number>=[1,2,3];

元组：let x:[string,number];

x=[‘hello’,10];//OK与x=[10,’hello’];//Error

x[0]substr(1);//OK与x[1].substr(1);//Error

访问一个越界元素，会使用联合型替代：

X[3]=’world’;//OK与x[6]=true;//Error

枚举：1 -- enum Color {Red, Green, Blue};let c:Color=Color.Green;

2 -- enum Color {Red = 1, Green = 3, Blue=8}; let c:Color=Color.Green;默认是从0开始

3 – enum Color {Red=1,Green,Blue}; let colorName:string=Color[2];

任意值：let list:any[]=[1,true,’free’];list[1]=100;

空值：与any类型相反，它表示没有任何类型。

1 – function test():void{ console.log(‘test’) };

2 – let unusable:void=undefined; 当对变量使用时，只能赋值undefined和null

Null和Undefined：与void相同，意义不大

1 – 默认null和undefined是所有类型的子类型,所以它们可以赋值给number类型等

2 – 在使用 –strictNullChecks标记时，let a:null=null;只能是null与let b:undefined=undefined;只能是undeinfed

Never:表示那些用不存在的值的类型。它是所有类型的子类型，但any不能赋值给never

类型断言：1 -- <>;let val:any=’test’;let strlength:number=(<string>val).length;

2 – as; let val:any=’test’;let strlength:number=(val as string).length;

二、变量声明

Var,let,const

解构：1 – 基本 let input=[,2];let [one,two]=input;

2 –作用函数参数,function f([one,two]:[number,number]){}

3—let [one,…rest]=[1,2,3,4]; rest就是[2,3,4];

4 – 忽略 [one]=[1,2,3,4]; one是1;let [,two,,four]=[1,2,3,4];

5 – 对象解构 let o={a:’foo’,b:12,c:’bar’};let {a,b}=o;

6 – 属性重名命let {a:newName1}=o;let {a}:{a:string}=o;

7 – 默认值 function do(obj:{a:tring,b?:number}){let {a:b=1001}=obj;}

函数声明：type C = {a:string,b?:number}; function f({a,b}:C):void{};

三、接口

初探：interface LabelledValue{label:string}

Function printLabel(labelledValue:LabelledValue){ console.log(labelledValue.label); }  
 let myObj={size:10,label:’Size 10 object’};//只要有label属性就行 printLabel(myObj);

可选属性：interfaceSquareConfig { color?:string,width?:number };  
 function createSquare(config:SquareConfig):{color:string;area:number}{

Let newSquare={color:’white’,area:100};

If(config.color) newSquare.color=config.color;

If(config.width) newSquare.area=config.width\*config.width;

return newSquare;

}

Let mySquare= createSquare({color:’black’});

只读属性：interface Point{ readonly x:number,readonly y:number };

let p1:Point={x:10,y:20}; p1.x=5; //

1. directives,pipes,components,是把方法集成到module中，再引入到其他module中，就可以直接使用。
2. 管道：

2.1纯管道与非纯管道   
import {Pipe,PipeTransform} from ‘@angular/core’

@Pipe({name:’format’,pure:false})

Export class TestPipe implements PipeTransform{ transform(value:number,exponent:string){} }

**纯管道**就是Angular只有检查到输入值发生纯变更时，才会执行纯管道。纯变更指的是，原始类型值(String,Number,Boolean,Symbol)的改变，或者对象引用的改变(对象值改变不是纯变更，不会执行).

**非纯管道**就是Angular会在每个组件的变更检测周期执行非纯管道。所以，如果使用非纯管道，我们就得注意性能问题了。

**内置管道：**DatePipe，JsonPipe，UpperCasePipe，LowerCasePipe，DecimalPipe，CurrentcyPipe，SlicePipe，PercentPipe

1. 常见指令：

3.1\*ngIf;

3.2[ngSwitch]--\*ngSwitchCase;

3.3<div [style.background-color=”’yellow’”],[ngStyle]=”{color:’white’,’background-color’:’blue’}”>

Test  
<div>  
3.4 [ngClass]

* 1. [ngFor]
  2. ngNonBindabble <div ngNonBindabble></div>

3.7 <ng-template #hidden>

<p>You are not allowed to see our secret</p>

</ng-template>

<p \*ngIf="shown; else hidden">

Our secret is being happy

</p>

* 1. [hidden]

1. angular的8个钩子函数

4.1 ngOnChanges  
@input属性(输入属性)发生变化时，会调用。非此属性，不会调用。

当输入属性为对象时，当对象的属性值发生变化时，不会调用，当对象的引用变化时会触发。先于ngOnInit调用。

4.2 ngOnInit

只执行一次，dom操作可放在其中。

4.3 ngDoCheck

每次发生变更检测时会被调用

ngDoCheck() 是Angular中的变更检测机制.它由 zone.js 来实现的.其行为是只要你的Angular中的某个组件发生异步事件.就会检查整个组件树,以保证组件属性的变化或页面的变化是同步的.所以 ngDoCheck() 的触发相当频繁的.并且是我们无法预料到的.也许我们在页面上的一个无意识操作,就会触发几个甚至几十个的 ngDoCheck() 生命周期钩子.

4.4 ngAfterContentInit

在组建内容初始化之后调用

* 1. ngAfterContentChecked

内容投影：父组件写在子标签之间的内容会被渲染到子模板的ng-content中去，类似vue的slot组件及子组件每次检查内容时调用; 当父子组件都有该钩子时，父组件先执行.

4.6 **ngAfterViewInit**

组件相应的视图初始化之后调用

**4.7 ngAfterViewChecked**

组件及子组件每次检查视图时调用; 当父子组件都有该钩子时，子组件先执行。

ngAfterViewChecked与ngAfterViewInt中不允许修改绑定的属性（@input属性），否则抛出异常

4.8 ngOnDestory

销毁，事件解绑。

编码风格指南

1. 单一职责
2. 命名约定
3. 编程约定
4. 应用程序解构（平面化）
5. Components/Pages
6. Directives
7. Providers/Services

# 一、单一职责

## 1.1、坚持每个文件只定义一样东西（服务或组件） 58e97a61-58a6-4e2a-a38a-e1b272ba90fe.png

## 1.2、考虑把文件的大小限制在400行内

## 1.3、单组件意义

1) 非常容易阅读、维护，并能防止在版本控制系统里与团队冲突

2) 单独的组件通常都是该文件默认的输出，这样可以利用路由实现按需加载

3) 增强代码可重用性和阅读性，减少出错的可能性

4) 防止一些隐蔽的程序缺陷

A、共享变量

B、创建意外的闭包

C、与依赖之间产生意外耦合

# 二、命名约定

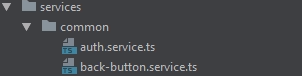
## 2.1、总体命名原则

1) 数组变量名用单词复数  
a96d6d07-fcd5-424e-b869-113b25154b8d.png

2) 非数组变量名用单词单数   
889d90a5-864e-4959-a565-5abc6b895f06.png

## 2.2、使用点和横分割文件名

1) 在描述性名字里面，使用横杠来分割单词  
2fc7fe9a-a66f-4cb2-89dc-4ec093dd6cf8.png

2) 使用点来分割描述性名字和类型名  


3) 使用下列所在文件夹名英文单数为文件类型名

A、\*.service如（home.service.ts）

B、\*.component（home-title.component.ts）

C、\*.pipe（dateFormat.pipe.ts）

D、\*.directive（validate.directive.ts）

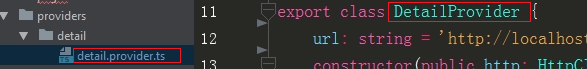
E、\*.model（hero.model.ts）

F、\*.provider如（[home.provider.ts](http://home.provider.ts)）

G、\*.page如（home.page.ts）

## 2.3、组件与指令命名约定

1) 保持符号（类）的名字与他所在的文件名字相同



2) 符号的类型（比如组件，服务，指令等）附加到符号名的后面

服务：  
90449fcc-8ae4-4826-9eb2-6fc13c663dd2.png

组件：

53790b80-0e33-48d8-8d34-e3fcfa4e87f8.png

指令：

02dd13df-a931-4726-8b3b-8cefd4ed091d.png

3) 大写驼峰命名法

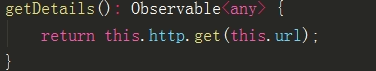
53790b80-0e33-48d8-8d34-e3fcfa4e87f8.png

## 2.4、服务名[provider/service]命名约定

1) 大写驼峰命名法   
59bb2d2c-d73f-4316-8f5b-5920209ba9bd.png

2) logger不需要后缀

3) 保持符号（类）的名字与他所在的文件名字相同

4) http请求方法名为http Method+请求内容，如：getCustomers、postCusteomer  


5) 数据处理转换等在这一层，数据转换方法名为转换前的类型To转换后的类型

## 2.5、引导

1) 把应用的引导程序和平台相关的逻辑放到main.ts的文件里

2) 避免把业务逻辑放入main.ts，考虑把他们放入组件和服务里面

## 2.6、指令选择器（比如ngIf）

1) 使用小驼峰命名法

A、指令里定义的属性名字与它们绑定的视图HTML属性名一直

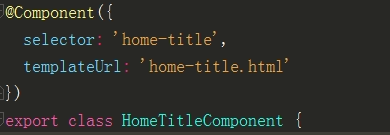
B、Angular HTML解析器是大小写敏感的，它识别小写驼峰命名

## 2.7、为组件自定义前缀

1) 为组件的选择器使用自定义前缀

A、防止名字冲突

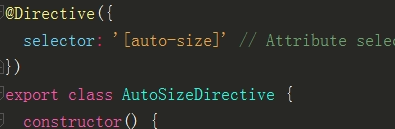
B、组件和元素更容易被识别

2) 使用前缀来识别特性区域或者应用程序本身  


## 2.8、为指令添加自定义前缀

1) 为指令的选择器使用自定义的前缀

A、防止命名冲突

B、指令更容易被识别  


## 2.9、管道名

## 2.10、单元测试文件

1) 测试规范文件的名字应该和被测试的组件名字一样

2) 测试配置文件命名应该跟随后缀.spec

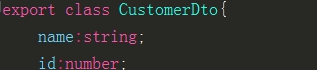
提供一个与karma或者其他测试运行器相配的命名模式

## 2.11、端到端测试文件名

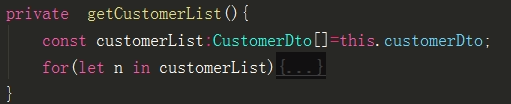
1) 端到端测试配置文件应该和它们所测试的特性同名，并加上后缀.e2e-spec

# 三、编程约定

## 3.1、类

1) 使用大驼峰命名法来命名类  


## 3.2、常量

1) 用const声明变量，除非它们的值在应用的生命周期内会发生变化  


2) 把常量名拼写为小驼峰

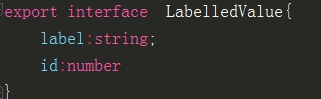
82481547-5c56-43e3-b25d-955eda1be5ee.png

小驼峰变量名（heroRoutes）比传统的“大写蛇形命名法”HERO\_ROUTES更容易阅读和理解

3) 容许现存的const变量沿用大写蛇形命名法

## 3.3、接口

1) 大写驼峰命名法

2) 不要在接口名字前面加I前缀  


## 3.4、属性和方法

1) 小写驼峰命名法

2) 避免使用下划线为前缀来命名私有属性和方法

A、JavaScript不支持真正的私有属性和方法

B、TypeScript工具让识别私有或公有属性和方法变得很简单

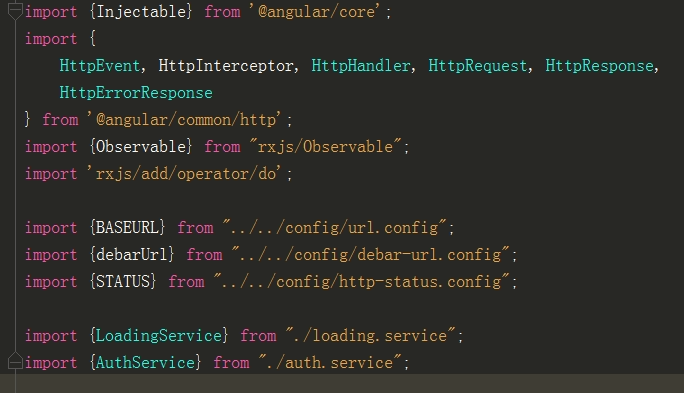
3) 属性在构造器之前，方法在构造器之后  


## 3.5、导入语句解构表达式中的空格

1) 在import声明的大括号里留一个空格字符

空格让import声明更易于阅读

2) import从上到下为框架基础模块、项目基础模块（包括config）、Provider、service按类型以空行隔开



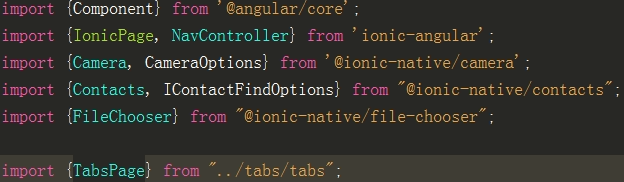
## 3.6、导入语句中的空行

1) 在第三方导入和自己代码的导入之间留一个空行

空行可以让阅读和定位本地导入变得更加容易

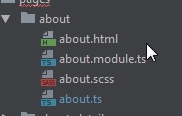
2) 按模块名字的字母顺序排列导入行

按字母顺序排列可以让阅读和定位本地导入更加容易

3) 在结构表达式中按字母顺序排列导入的东西  


# 四、应用程序解构（平面化）

## 4.1、总体结构指导原则

1) 每个组件一个目录，包含ts，html，scss等文件  


## 4.2、共享目录

1) 在单个特性范围内，把所有共享的文件放到common目录

A、在一个特性范围内，分离出组件共享的文件

B、使得在一个特性范围内，更加已与定位共享文件

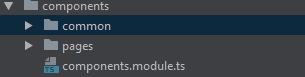
998db444-b713-40cf-bfb7-5c3852c1a143.png

## 4.3、根据特性划分的目录结构

1) 把目录的名字命名为它包含的特性名字

## 4.4、布局组件

1) 把定义总体布局的组件放到common目录

A、需要一个地方来存放应用程序的布局，比如导航条，页脚和其他整个应用中都需要的东西   


## 4.5、封装

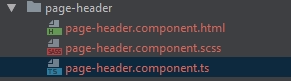
1) 用来导入、归集和重新导出项目

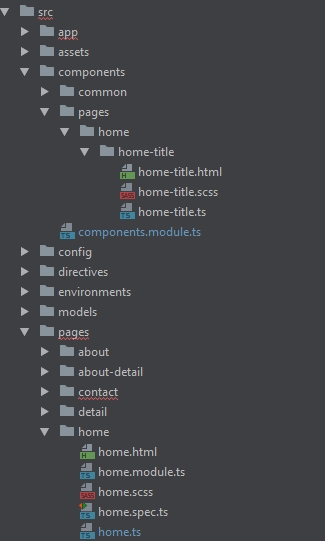
2) 封装文件命名为index.ts

# 五、Component/Pages

## 5.1、命名和存放目录

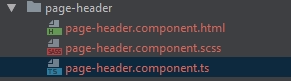
1) 烤串命名法(中线命名法)



2) src/component/pages文件夹内结构保持和src/pages一致, Component文件放入common和pages对应的目录  


## 5.2、模板和样式

1) 提取到文件中，不建议在组件中写模板和样式

2) 以组件名加html、css结尾  


## 5.3、Input和Output装饰器

1)坚持使用@Input和@Output

2)避免使用@Dirctive和@Component装饰器里的inputs和outputs

3)把@Input或者@Output放到他们装饰的属性的同一行

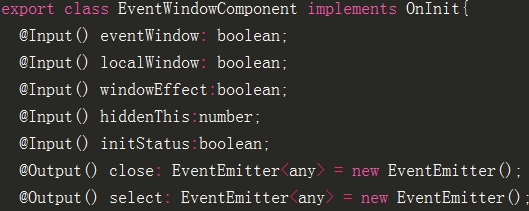
4)避免重命名@Input和@Output

5) 不要给输出属性加前缀

A、命名事件时，不要带on

a、内置事件命名一致，比如按钮点击

b、Angular允许另一种备选语法on-\*如果事件的名字本身带有前缀

on，那么绑定的表达式可能是on-onEven

## 5.4、成员顺序

1) 把属性成员放到顶部，方法成员紧随其后

2) 先放公共成员，再放私有成员，并按照字母顺序排列，成员变量在构造器前，成员方法在构造器后  


## 5.5、展示逻辑放到组件类

1) 组件的展示逻辑放到组件类而非模板类，可以增强测试性，维护性和重复使用性

# 六、Directive

## 6.1、使用指令来加强已有元素

1) 需要有无模板的展示逻辑时，使用属性型指令



## 6.2、使用HostListener和HostBinding类装饰器

1) 与属性或方法名字相关性的@HostListener和@HostBinding应该只在一个地方被修改:在指令的类里。

2) 反过来，如果我们使用宿主属性，我们需要在控制器内修改属性声明，然后在指令相关的元数据里修改。

# 七、Providers/Services

## 7.1、单例

1) 在同一个注入器内，把服务当做单例使用。使用它们来共享数据和功能  


## 7.2、单一

1) 服务成长到超出单一用法时，新建一个服务

2) 服务有多个职责时，它很难被测试

3) 服务有多个职责时，每个注入它的组件或者服务都会被迫承担它们的(多个职责)所有的重量

## 7.3、提供服务

1) 在被共享范围内的顶级组件里，将服务提供到Angular的注入器里

2) 顶层组件提供服务，其子级组件中可见并共享

3) 不同的两个组件需要一个服务的不同的实例时，单独提供

## 7.4、@Injectable()类装饰器

1) 使用@Injectable()类装饰器，而非@Inject参数装饰器

2) Angular的依赖注入机制，是根据在服务的构造函数里面的类型的声明，来解析所有服务的依赖的

3) 当服务只接受类型令牌相关的依赖时，比起在每个构造函数参数上使用@Inject(),@Injectable()的语法简洁多了  


## 7.5、数据服务

1) 分离数据调用

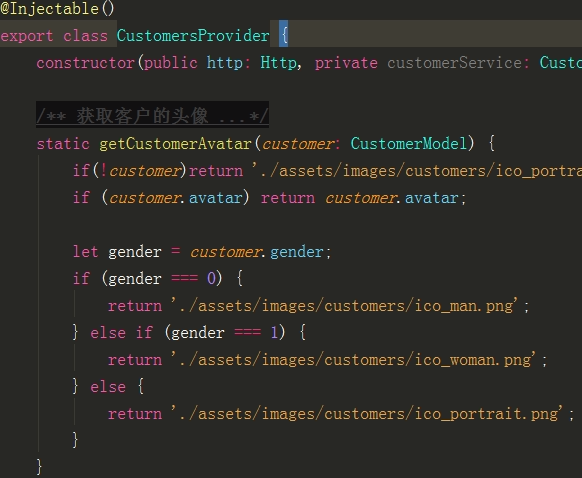
A、把数据操作和数据互动重构到服务里

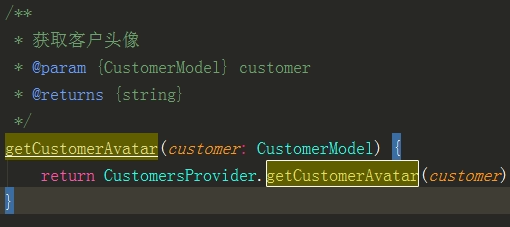
B、让数据服务来负责XHR调用，本地存储，内存存储或者其他数据操作

C、简化组件，聚焦视图

D、便于测试

E、对外部使用者隐藏具体实施细节

（1）客户服务获取头像实例中，根据XHR响应结果判断并返回头像图片地址；  


（2）客户组件中直接调取客户服务，并得到客户头像结果  


2) 生命周期钩子