TypeScript笔记

一、TypeScript安装和运行

VSCode终端中安装： npm i -g typescript

```
问题1：TS 代码能直接在 Node.js 里面运行吗？
问题2：该如何处理呢？
不能
1 TS代码 -> JS代码 2 执行
JS
① TS代码 -> JS代码：在当前目录打开终端，输入命令 tsc hello.ts 敲回车。
② 执行JS：输入命令 node hello.js（注意：后缀为 .js）。
3. 执行代码，分两步：
解释：
l tsc hello.ts 会生成一个 hello.js 文件。
l node hello.js 表示执行这个 JS 文件中的代码。
```

简化方法：

```
简化方式：使用 ts-node 包，“直接”在 Node.js 中执行 TS 代码。
l 安装命令：npm i –g ts-node。
l 使用方式：ts-node hello.ts。
解释：
l ts-node 包内部偷偷的将 TS -> JS，然后，执行 JS 代码。
l ts-node 包提供了命令 ts-node，用来执行 TS 代码。
```

二、TypeScript变量和数据类型

1.变量基本使用

变量的使用分为两步：1 声明变量并指定类型 2 给变量赋值。

```
let age: number = 18;
```

注意：声明变量的时候要**指定变量的类型**

2.类型注解

```
let age: number;
```

: number 就是类型注解。

类型注解：是一种**为变量添加类型约束**的方式。

3.变量的命名规则

变量名称只能出现：数字、字母、下划线（_）、美元符号（$），并且不能以 数字 开头。

注意：变量名称**区分大小写。**

```
// age 和 Age 是两个不同的变量
let age: number = 18
let Age: number = 20
```

可以使用驼峰命名法

类型别名

类型别名(自定义类型)：为任意类型起别名

当同一类型被多次使用时，可以通过类型别名，简化该类型的使用

1、使用type关键字来创建类型别名

2、创建类型别名后，可以**直接使用该类型别名作为变量的类型注解**

```
// 定义类型别名：用 type 关键字声明类型别名
type CustomArray = (number | string)[]

let arr2: CustomArray = [1, 2, 3, 'r']
let arr3: CustomArray = [1, 'g', 7, 'j']
```

4.数据类型概述

TypeScript 中的数据类型分为两大类：**1 原始类型（基本数据类型） 2 对象类型（复杂数据类型）**

1.常用基本数据类型

常用的基本数据类型有 5 个：**number / string / boolean / undefined / null。**

示例代码：

```
// 变量 age 的类型是 number（数字类型）
let age: number = 18

// 此处的 'Hello TS' 是 string（字符串类型）
console.log('Hello TS')
```

1.数字类型

数字类型：包含整数值和浮点型（小数）值。

```
// 数字类型：整数
let age: number = 18
// 数字类型：小数
let score: number = 99.9
```

也可以包含：正数和负数。

2.字符串类型

字符串：由零个或多个字符串联而成的，用来表示文本信息。

字符串可以使用单引号（'）或双引号（"），**推荐：使用单引号。**

字符串类型的类型注解为：string，声明变量时要添加类型注解

```
let food: string = '糖葫芦'
```

3.Boolean类型

布尔类型，用来表示真或假

只有两个值，分别是： true 和 false。true 表示真，false 表示假

布尔类型的类型注解为：boolean

```
// 真
let isStudying: boolean = true
// 假
let isPlayingGame: boolean = false
```

4.undefined、null

共同特点：只有一个值，值为类型本身

undefined 类型的值为： undefined。

null 类型的值为：null。

```
// 类型注解为：undefined
let u: undefined = undefined
// 类型注解为：null
let n: null = null
```

undefined：表示声明但未赋值的变量值（找不到值）。

```
let u: undefined
console.log(u) // 变量u的值为 undefined
```

null：表示声明了变量并已赋值，值为 null（能找到，值就是 null）

这些类型的值，也叫做字面量

```
18 // 数字字面量
'保温杯里泡枸杞' // 字符串字面量
true / false // 布尔字面量
undefined
null
```

三、TypeScript运算符

1.运算符概述

运算符也称为操作符，用来实现赋值（=）、算术运算、比较等功能的符号。

常用的运算符：算术运算符、 赋值运算符 、递增/递减运算符 、比较运算符、逻辑运算符

2、算术运算符

算术运算符包含：加（+）、减（-）、乘（*）、除（/）

算术运算符：进行算术运算时使用的符号，用于两个数值之间的计算

```
// 加
console.log(1 + 2) // 3
// 减
console.log(2 - 1) // 1
// 乘
console.log(2 * 3) // 6
// 除
console.log(4 / 2) // 2
```

加号的其他作用

注意：+ 号，不仅可以用于加法计算，还能实现字符串拼接

```
// 字符串拼接（拼串）
console.log('周杰'+'伦')  //输出：周杰伦
```

```
console.log(1 + 2) // 结果为：3
console.log(1 + '2')// 结果为：'12'
console.log('1' + 2)// 结果为：'12'
```

规律：加号两边只要有一边是字符串，就执行字符串拼接。

注意：除加号以外，其他算术运算符只应该跟数字类型一起使用。

其他方式：将字符串类型转换为数字类型。

```
console.log(2 - +'1')
// +'1' 表示将 '1'（string） => 1（number）
// 所以，2 - +'1' ==> 2 - 1 ==> 结果为：1
```

记住：在字符串前面添加 + 号，可以将 string 转化为 number（字符串内容为数字时才有意义）。

3、赋值运算符

赋值运算符：将等号右边的值赋值给它左边的变量，比如：等号（=）

```
// 等号：将 18 赋值给左侧的变量 age
let age: number = 18
```

4、自增和自减运算符

自增（++）运算符是 += 1 的简化形式；

自减（--）运算符是 -= 1 的简化形式

解释：age++ 会让变量 age 的值加 1。

作用：自增（++）运算符用来实现变量的值加 1；自减（--）运算符实现变量的值减 1。

注意：++ 或 --，只能让变量的值增加或减少 1。

5、比较运算符

比较运算符：用于比较两个数据的值，并返回其比较的结果，**结果为布尔类型**

比较运算符包含 6 个：

大于（>）

大于等于（>=）

小于（<）

小于等于（<=）

等于（===）

不等于（!==）

```
// 大于
console.log(1 > 2)  // 结果为：false
// 大于等于
console.log(3 >= 2) // 结果为：true
// 小于
console.log(1 < 2)  // 结果为：true
// 小于等于
console.log(3 <= 2) // 结果为：false
// 相等
console.log(3 === 4)// 结果为：false
// 不相等
console.log(3 !== 4)// 结果为：true

```

注意：比较运算符常用于数字类型的比较。

6、逻辑运算符

与逻辑运算符一一对应：**与（并且）、或（或者）、非（不是）。**

逻辑运算符通常用于布尔类型的计算，并且结果也是布尔类型

1、与（逻辑与），用 && 符号来表示。当 && 两边的值同时为true，结果才为true；否则，为false

```
true && false // 结果为：false
```

2、或（逻辑或），用 || 符号来表示。当 || 两边的值只要有一个为true，结果就为true；否则，为false。

```
true || false // 结果为：true
```

3、 非（逻辑非），符号为 !（叹号），表示取反，即：true□false 而 false□true。

```
!true // 结果为：false
```

四、TypeScript语句

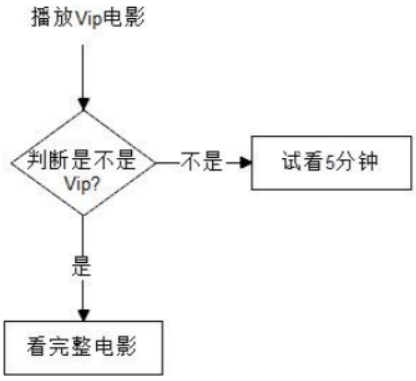
1、条件语句

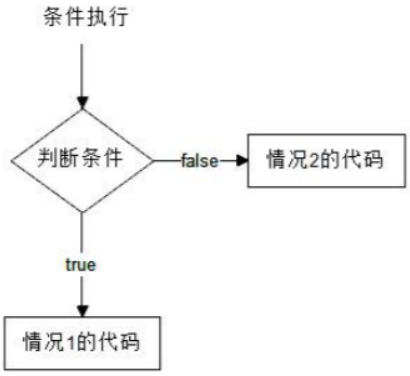
条件语句：根据判断条件的结果（真或假），来执行不同的代码，从而实现不同功能。

条件执行时，首先判断条件是否满足。

I 、如果 条件满足，就做某件事情（情况1）

2、如果 条件不满足，就做另外一件事情（情况2）





条件语句，也叫分支语句，不同的情况就是不同的分支。

1.1 if 语句

在 TypeScript 中 if 语句就是实现条件判断的

if 语句的语法：

```
if (判断条件) {
    条件满足时，要做的事情
}
```

解释：Ⅰ、判断条件：布尔类型（true 或 false）。

如果判断条件为真，就执行要做的事情；

否则，如果判断条件为假，则不执行花括号中的代码。

补充概念说明：**语句，是一个完整的句子，用来使某件事情发生（或实现某个功能）**

1.2 else 语句

在 TypeScript 中 **else 语句必须配合 if 语句**来使用。

else 语句表示：条件不满足，要做的事情（if 语句的对立面）

else 语句的语法：

```
if (判断条件) {
    条件满足时，要做的事情
} else {
    条件不满足，要做的事情
}
```

否则，如果判断条件为假，就执行 条件不满足时要做的事情。

2、三元运算符

三元运算符的作用与 if...else 语句类似。

作用：根据判断条件的真假，得到不同的结果

语法：

```
结果 = 判断条件 ? 值1 : 值2
```

如果判断条件为真，结果为 值1；

否则，如果判断条件为假，结果为 值2

注意：得到结果的类型由值1和值2的类型决定（值1和值2的类型相同）

3、循环语句

在 TypeScript 中，要实现重复做某件事情，就需要用到循环语句，来减少重复劳动提升效率

3.1 for 循环

在 TypeScript 中，for 循环就是实现重复做某件事情的循环语句。

注意：for 循环是 TS 基础知识的重难点，语法比较复杂。

for 循环的组成：

1. 初始化语句：声明计数器变量用来记录循环次数（执行一次）。
2. 判断条件：判断循环次数是否达到目标次数。
3. 计数器更新：完成一次循环让计数器数量加1。
4. 循环体：循环的代码，也就是要重复做的事情

语法：

```
for (初始化语句; 判断条件; 计数器更新) {
    循环体
}
```

初始化语句：声明计数器变量，记录循环次数

判断条件：判断循环次数是否达到目标次数

计数器更新：计数器数量加1。

循环体：重复执行的代码，也就是要重复做的事情。

```
// 作业写 3 遍：
for (let i: number = 1; i <= 3; i++) {
    console.log('北冥有鱼，其名为鲲。鲲之大，一锅装不下')
}
```

1. 初始化语句：只会执行一次。
2. 重复执行的部分：判断条件、循环的代码、计数器更新

break和continue

break 和 continue 常用在循环语句中，用来改变循环的执行过程

for 循环执行的特点是：连续且不间断

1、 break 能够让循环提前结束（终止循环）。

2、 continue 能够让循环间断执行（跳过本次循环，继续下一次循环）

TypeScript数组

1、数组概述

数组，是用于**存放多个数据的集合。**

```
let names: string[] = ['迪丽热巴', '古力娜扎', '马尔扎哈']
```

注意：**数组中，通常是相同类型的数据**

联合类型：数组中也可以既有number又有string：

```
let arr: (number | string)[] = [1,'a',2,'b']
```

2、创建数组

一共有两种语法格式创建数组

1、 语法一

```
let names: string[] = []
```

[]（中括号）表示数组。**如果数组中没有内容，就是一个空数组。**

数组的类型注解由两部分组成：类型+[]

此处表示字符串类型的数组（只能出现字符串类型）

数组，多个元素之间使用逗号（,）分隔。 **数组中的每一项内容称为：元素**

2、语法二

```
let names: string[] = new Array()
```

```
let names: string[] = new Array('迪丽热巴', '古力娜扎', '马尔扎哈')
// 相当于：
let names: string[] = ['迪丽热巴', '古力娜扎', '马尔扎哈']
```

3、数组长度和索引

3.1数组长度

表示数组中**元素的个数**，通过**数组的 length 属性**来获取

```
let foods: string[] = ['煎饼', '馒头', '米饭']
```

获取数组长度：

```
console.log(foods.length)  // 3
```

3.2数组索引

数组中的每个元素都有自己的序号

我们把数组中元素的序号，称为：**索引（下标），数组中的元素与索引一一对应**

注意：数组索引是从 0 开始的。

```
let foods: string[] = ['煎饼', '馒头', '米饭']
// 数组的索引分别为： 0 1 2
```

数组的长度（length）和最大索引的关系： 最大索引为：length - 1

4、取值与存值

4.1取值

从数组中，获取到某一个元素的值，就是从数组中**取值**

语法：

```
数组名称[索引]
```

数组中的元素与索引是一一对应的，**通过索引获取到某一个元素的值**

获取最爱的食物：煎饼

```
console.log(foods[0])
```

4.2存值

如果要**修改数组中某个元素的值**，就要使用数组存值

先获取到要修改的元素，然后，再存值

语法：

```
数组名称[索引] = 新值
```

将馒头替换为包子：

```
foods[1] = '包子'
```

4.3添加元素

存值的语法是：**数组名称[索引] = 新值**，**根据索引是否存在**，有两种功能：**1 修改元素 2 添加元素。**

```
let foods: string[] = ['煎饼', '馒头', '米饭']
```

1、 如果索引存在，就表示：修改元素。

```
foods[1] = '包子'
```

2、如果索引不存在，就表示：添加元素

```
foods[3] = '面'
```

添加元素的通用写法：数组名称[数组长度] = 新值

5、遍历数组

遍历数组，也就是把数组中的所有元素挨个获取一次

使用 for 循环

```
let nums: number[] = [100, 200, 300]
```

遍历以上数组：

```
for(let i: number = 0; i <= nums.length; i++){
    console.log(nums[i]);
}
```

注意1：因为数组索引是从0开始的，所以计数器i的默认值为0。

注意2：应该根据数组长度来计算，公式为数组长度减一，也就是：**nums.length – 1（最大索引）。**

TypeScript函数

1、函数概述

需求：计算数组nums中所有元素的和。

```
let nums: number[] = [1, 3, 5];

let sum: number = 0;

for(let i: number = 0; i<= nums.length; i++){
    sum += nums[i];
}

console.log(sum);
```

利用函数可以实现以上代码的复用

使用**函数来包装（封装）相似的代码，在需要的时候调用函数，相似的代码不再重复写。**

```
function getSum(nums: number[]) {
    let sum: number = 0
    for (let i: number = 0; i < nums.length; i++) {
        sum += nums[i]
    }
    console.log(sum)
}
getSum(nums1) // 计算nums1中所有元素的和
getSum(nums2) // 计算nums2中所有元素的和
```

所谓函数，就是声明一次但却可以调用任意多次的一段代码

意义：实现代码复用，提升开发效率

封装：将一段代码包装起来，隐藏细节

2、函数的使用

函数的使用分为两步：**1 声明函数 2 调用函数 （类比变量）**

1、声明函数

```
function 函数名称() {
    函数体
}
```

函数体：表示要实现功能的代码，复用的代码

```
function sing() {
    console.log('五环之歌')
}
```

2、调用函数

```
函数名称()
```

```
sing()
```

注意：只有调用函数后，函数中的代码才会执行

3、函数的参数

```
// 调用函数时，告诉函数要唱的歌曲名称
sing('五环之歌')
sing('探清水河')

// 声明函数时，接收传入的歌曲名称
function sing(songName: string) {
    console.log(songName)
}
```

函数（sing）中歌曲名称：固定值 -----> 动态传入的值。

函数参数的作用：增加了函数的灵活性、通用性，针对相同的功能，能够适应更多的数据（值）

3.1 形参和实参

函数参数分为两部分：1 形参 2 实参

1、形参

声明函数时指定的参数，放在声明函数的小括号中（挖坑）

语法：形参名称: 类型注解

类似于变量声明，但是没有赋值

```
function sing(songName: string) { }
```

2、实参

调用函数时传入的参数，放在调用函数的小括号中（填坑）

```
sing('五环之歌')
```

实参是一个具体的值（比如：'字符串'、18、[]等），**用来赋值给形参。**

1、声明函数时的参数：**形参**

作用： 指定函数能够接收什么数据

2、调用函数时的参数：**实参**

作用：是一个具体的值，用来赋值给形参

通过形参和实参的配合，函数可以接收动态数据，从而让函数变得更加灵活、强大。

3、注意点

1、根据具体的功能，函数参数可以有多个，参数之间使用逗号（,）来分隔。

```
function fn(name: string, age: number) { }
fn('刘老师', 18)
```

2、**实参和形参按照顺序，一一对应。**

```
function fn(name: string, age: number) { }
fn('刘老师', 18) // name -> '刘老师', age -> 18
```

3、**实参必须符合形参的类型要求，否则会报错**

```
function sing(songName: string) {}
sing(18) // 报错！ 形参要求是 string 类型，但是，实参是 number 类型
```

技巧：调用函数时，鼠标放在函数名称上，会显示该函数的参数以及类型

4、函数的返回值

函数返回值的作用：将函数内部计算的结果返回，以便于使用该结果继续参与其他的计算

注意：如果没有指定函数的返回值，那么，函**数返回值的默认类型为 void**

1、返回值的使用

步骤：1 指定返回值类型 2 指定返回值

1、指定返回值类型

```
function fn(): 类型注解 {

}
```

```
function fn(): number{

}
```

2、指定返回值

```
function fn(): 类型注解 {
    return 返回值
}
```

在函数体中，使用 return 关键字来返回函数执行的结果

```
function fn(): number{
    return 18
}
```

注意：**返回值必须符合返回值类型的类型要求**，否则会报错

2、基本使用

1、使用变量接收函数返回值

```
let result: 类型注解 = fn()
```

使用变量接收函数返回值的时候，相当于：**直接将返回值赋值给变量**

```
let result: number = fn(){
    return 18
}
```

注意：**变量（result）的类型与函数（fn）的返回值类型要一致。**

2、 直接使用函数调用的结果（返回值），进行其他计算

```
console.log( fn() * 10 )
```

3、return的说明

1、将函数内部的计算结果返回。

2、终止函数代码执行，即：return 后面的代码不会执行。

```
 function fn(): number {
    return 18
    console.log('我不会执行，放在这，没有意义')
}
```

3、return 只能在函数中使用，否则会报错。

4、return 可以单独使用（后面可以不跟内容），用来刻意终止函数的执行。

5、函数总结

函数，即：声明一次但却可以调用任意多次的一段代码

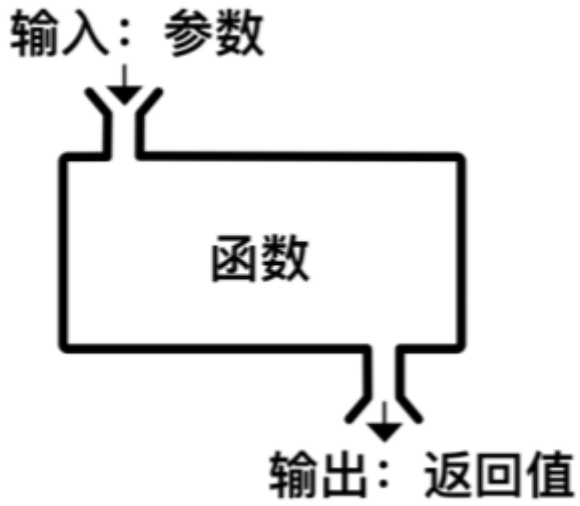
通过将要实现的功能，使用函数封装起来，实现代码复用，提升开发效率。

函数的三种主要内容： **1 参数 2 函数体 3 返回值**

简化过程： 1. 输入（参数） -- 可选

2. 处理（函数体）

3. 输出（返回值）-- 可选



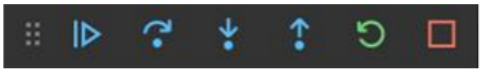
6、函数进阶-函数调试

借助断点调试，观察代码的执行过程。

关键点：1 在哪个位置打断点？ 2 如何调试？

一．断点位置：函数调用所在位置。

二．调试函数常用按钮：



单步调试：表示执行下一行代码，但是，遇到函数调用时，进入函数内部。

单步跳出：表示跳出当前函数（函数中的代码执行完成），继续执行函数调用后的下一行代码。

```
function work() {
    console.log('早上9点开始工作')
    play()
    console.log('晚上6点结束工作')
}
function play() {
    console.log('早上9:30开始吃鸡')
    console.log('晚上5:30结束吃鸡')
}
work()
```

结论1：函数里面，还可以继续调用其他函数。

结论2：函数，按照顺序一行行的执行代码，当遇到调用其他函数时，先完成该函数调用，再继续执行代码。

7、函数进阶-变量作用域

一个变量的作用域指的是：代码中定义变量的区域，它决定了变量的使用范围。

在 TS（或JS）中，函数可以形成作用域，叫做：函数作用域。

根据范围的不同，变量可以分为两种：1 局部变量 2 全局变量

1、局部变量

表示在**函数内部声明的变量**，该变量只**能在函数内部使用**

```
function fn() {
    // 变量 num 是局部变量
    let num: number = 1
    console.log(num) // 此处能访问到变量 num
}
fn()
console.log(num) // 问题：此处能访问到变量 num 吗？ 不能
```

2、全局变量

表示在**函数外部声明的变量**，该**变量在当前 ts 文件的任何地方都可以使用**

```
// 变量 num 是全局变量
let num: number = 1
function fn() {
    console.log(num) // 问题：此处能访问到变量 num 吗？  能
}
fn()
console.log(num) // 问题：此处能访问到变量 num 吗？  能
```

TypeScript对象

1、对象概述

TS 中的对象：**一组相关属性和方法的集合，并且是无序的**

```
{
    name: '周杰伦',
    gender: '男',
    height: 175,
    sing: funtion () {
        console.log('故事的小黄花 从出生那年就飘着')
    }
}
```

变量缺点：**一个变量只能存储一个数据，多个变量之间没有任何关联（相关性）**

使用对象，对象在描述事物（一组相关数据）时，结构更加清晰、明了。

在 TS 中，如果要描述一个事物或一组相关数据，就可以使用对象来实现。

2、创建对象

语法：

```
let person = {}
```

此处的 {}（花括号、大括号）表示对象。而对象中没有属性或方法时，称为：空对象。

对象中的属性或方法，采用键值对的形式，键、值之间使用冒号（:）来配对。

键（key）----> 名称

值（value）----> 具体的数据。

多个键值对之间，通过逗号（,）来分隔（类比数组）

属性和方法的区别：值是不是函数，如果是，就称为方法；否则，就是普通属性

```
let person = {
    sayHi: function () {
        console.log('大家好，我是一个方法')
    }
}
```

注意：函数用作方法时可以省略function后面的函数名称，也叫做匿名函数。

函数没有名称，如何调用？ 此处的sayHi相当于函数名称，将来通过对象的sayHi就可以调用了。

如果一个函数是单独出现的，没有与对象关联，我们称为函数；否则，称为方法

3、接口

3.1 对象的类型注解

TS 中的对象是结构化的，结构简单来说就是对象有什么属性或方法。

在使用对象前，就可以根据需求，提前设计好对象的结构。

比如，创建一个对象，包含姓名、年龄两个属性。

思考过程：1 对象的结构包含姓名、年龄两个属性 2 姓名 → 字符串类型，年龄 → 数值类型 3 创建对象。

这就是对象的结构化类型（左侧），即：对该对象值（右侧）的结构进行类型约束。

或者说：建立一种契约，约束对象的结构。

语法说明：

对象类型注解的语法类似于对象自身的语法。

注意：键值对中的值是类型！（因为这是对象的类型注解）。

注意：多个键值对之间使用分号（;）分隔，并且分号可省略。

对象类型注解：

注意：键值对中的值是类型！（因为这是对象的类型注解）。

注意：多个键值对之间使用分号（;）分隔，并且分号可省略。

对象方法的类型注解

```
let person: {
    sayHi: () => void
    sing: (name: string) => void
    sum: (num1: number, num2: number) => number
}
```

技巧：鼠标放在变量名称上，VSCode就会给出该变量的类型注解。

箭头（=>）**左边小括号中的内容：表示方法的参数类型**

箭头（=>）**右边的内容：表示方法的返回值类型**

方法类型注解的关键点：**1 参数 2 返回值。**

3.2 接口的使用

直接在对象名称后面写类型注解的坏处：1 代码结构不简洁 2 无法复用类型注解。

接口：为对象的类型注解命名，并为你的代码建立契约来约束对象的结构

语法：

```
interface IUser {
  name: string
  age: number
}
```



```
let p1: IUser = {
  name: 'jack',
  age: 18
}
```

interface 表示接口，接口名称约定以I开头。

推荐：**使用接口来作为对象的类型注解。**

4、取值和存值

4.1 取值

取值，即：拿到对象中的属性或方法并使用。

获取对象中的属性，称为：访问属性

获取对象中的方法并调用，称为：调用方法

1、访问属性

```
let jay = { name: '周杰伦', height: 175 }
```

需求：获取到对象（jay）的name属性。

```
jay.name
```

说明：通过点语法（.）就可以访问对象中的属性。

技巧：在输入点语法时，利用VSCode给出来的提示，利用上下键快速选择要访问的属性名称

2、调用方法

```
let jay = {
    sing: function () {
        console.log('故事的小黄花 从出生那年就飘着')
    }
}
```

需求：调用对象（jay）的sing方法，让他唱歌

```
jay.sing()
```

说明：通过点语法（.）就先拿到方法名称，然后，通过小括号调用方法

4.1 存值

存值，即修改（设置）对象中属性的值

```
let jay = { name: '周杰伦', height: 175 }
```

需求：将对象（jay）的name属性的值修改为'周董'

```
jay.name = '周董'
```

先通过点语法获取到name属性，然后，将新值'周董'赋值给该属性

注意：设置的新值，也必须符合该属性的类型要求

注意：几乎不会修改对象中的方法。

复杂数据类型：object（对象、数组）、function（函数）

5、内置对象

内置对象，是 TS/JS 自带的一些基础对象，提供了TS开发时所需的基础或必要的能力

已经用过的内置对象：数组

学习方式——查文档

文档地址：MDN（更标准） / W3school（国内）

5.1 数组对象

数组是 TS 中最常用、最重要的内置对象之一，掌握数组的常用操作能够显著提升开发效率。

数组的常用操作：添加、删除、遍历、过滤等。

重点学习：1 属性（length） 2 方法（push、forEach、some）。

length属性

length 属性：获取数组长度。

```
let songs: string[] = ['五环之歌', '探清水河', '晴天']
```

获取数组长度：

```
songs.length
```

push()方法

push 方法：**添加元素**（在数组**最后一项元素的后面添加**）。

```
let songs: string[] = ['五环之歌', '探清水河', '晴天']
```

添加元素

```
songs.push('花海')
```

forEach()方法

forEach 方法：**遍历数组**

```
let songs: string[] = ['五环之歌', '探清水河', '晴天']
```

原始方法：

```
for (let i: number = 0; i < songs.length; i++) {
    console.log('索引为', i, '元素为', songs[i])
}
```

forEach()方法

```
songs.forEach(function (item, index) {
    console.log('索引为', index, '元素为', item)
})
```

注意：forEach 方法的参数是一个函数，这种函数也称为回调函数

forEach 方法的执行过程：**遍历整个数组，为数组的每一项元素，调用一次回调函数**

回调函数的两个参数：

```
1. item 表示数组中的每个元素，相当于 songs[i]。
```

2. index 表示索引，相当于 i。

注意：**此处的回调函数，是作为 forEach 方法的实参传入，不应该指定类型注解**

some()方法

需求：判断数组中是否包含大于10的数字

```
let nums: number[] = [1, 12, 9, 8, 6]
```

使用forEach()方法

```
nums.forEach(function(item,index){
    if(item > 10)
        console.log('数组中有大于10的数字！')
})
```

效率低下

some 方法：遍历数组，查找是否有一个满足条件的元素（如果有，就可以停止循环）

循环特点：根据回调函数的返回值，决定是否停止循环。如果返回 true，就停止；返回 false，就继续循环

```
nums.some(function (num){
    if(num > 10){
        return true
    }
    return false
})
```

some 方法的返回值：布尔值。

如果找到满足条件的元素，结果为 true; 否则，为 false。

查找是否包含满足条件的元素时，使用 some；

对数组中每个元素都进行相同的处理时，就用 forEach

6、类型推论

某些没有明确指出类型的地方，类型推论会帮助提供类型

发生类型推论的2种常见场景：1 声明变量并初始化时 2 决定函数返回值时。

> let age~~: number~~ = 18 // => let age = 18 function sum(num1: number, num2: number)~~: number~~ { return num1 + num2 } // => function sum(num1: number, num2: number) { return num1 + num2 }

TS中的两种文件类型

1、.ts文件（代码实现文件）

1、既包含类型信息又可执行代码

2、可以被编译为.js文件，然后执行代码

3、用途：编写程序代码的地方

2、.d.ts文件（类型声明文件）

1、只包含类型信息的类型声明文件

2、不会生成.js文件，仅用于提供类型信息

3、用途：为JS提供类型信息

总结：.ts是implementation（代码实现文件）

.d.ts是declaration（类型声明文件）

若要为JS库提供类型信息，要使用.d.ts文件

3、类型声明文件的使用

1、使用已有的类型声明文件

2、创建自己的类型声明文件

TypeScript之Web开发