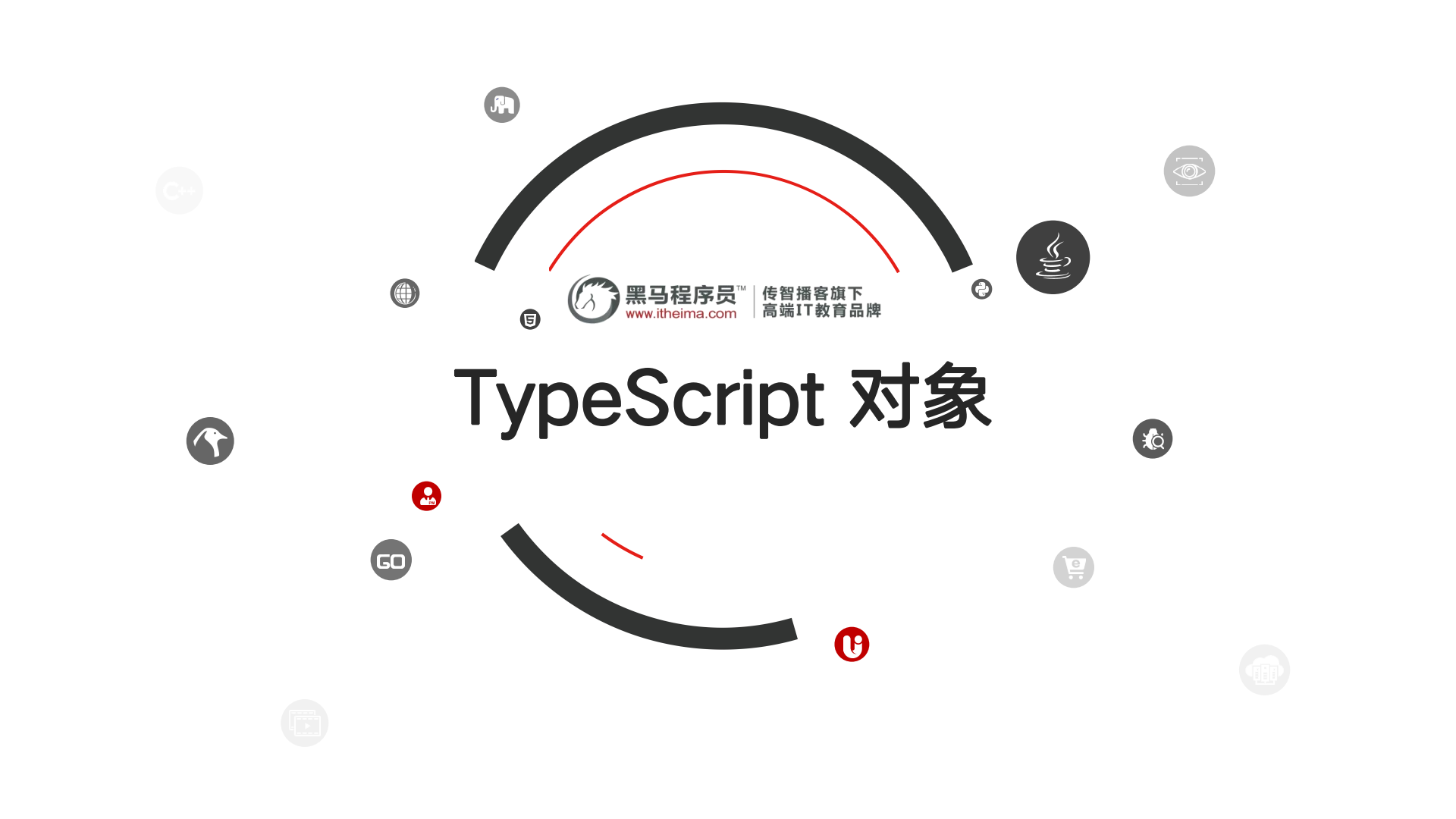
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
TypeScript 对象

目录 Contents

1. 对象概述

生活中，对象是一个具体的事物，比如：你的电脑、你的手机、古力娜扎、周杰伦（周董）等都是对象。

程序员都知道：万物皆对象。

这些具体的事物，都有自己的特征和行为：

特征：

你的电脑：尺寸、重量、价格等
你的手机：品牌、屏幕大小、颜色等
古力娜扎：年龄、身高、三围等

行为：

你的电脑：浏览网页、写代码等
你的手机：播放视频、吃鸡等
古力娜扎：演电影、配音等

1. 对象概述

TypeScript 中的对象，是对生活中具体事物的抽象，使得我们可以通过代码来描述具体的事物。

TS 中的对象，也是由特征和行为组成的，它们有各自专业的名称：属性（特征）和方法（行为）。

- 理解 TS 中的对象：一组相关属性和方法的**集合**，并且是**无序**的。

```
// 演示对象：
{
  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')
  }
}
```

1. 对象概述

疑问：为什么要有对象？

需求：使用 TS 代码描述周杰伦。

方案一：使用多个变量

```
let name: string = '周杰伦'
let gender: string = '男'
let height: number = 175
function sing() { ... }
```

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）。

1. 对象概述

疑问：为什么要有对象？

需求：使用 TS 代码描述周杰伦。

方案二：使用数组，一次可以存储多个数据：

```
['周杰伦', '男', 175, function sing() { ... }]
```

缺点：不知道数组中的每个元素表示什么。

正确姿势：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。

```
{ name: '周杰伦', gender: '男', height: 175, sing: function () { ... }}
```

1. 对象概述

总结：

对象：一组相关属性和方法的**集合**，并且是**无序**的。

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。

目录 Contents

2. 创建对象

注意：先学习对象的基本使用，再学习对象的类型注解（对象的类型注解类似于对象自身的语法）。

对象的语法：

```
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。

```
let person = {
  键1: 值1,
  键2: 值2
}
```

```
let person = {
  name: '刘老师',
  age: 18
}
```

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）。

2. 创建对象

现在，对象person具有两个属性：

```
let person = { name: '刘老师', age: 18 }
```

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。

2. 创建对象

总结：

对象中的属性或方法，采用键值对的形式，因此，对象是无序键值对的集合。

- 使用什么符号创建对象？ 花括号（{}）
- 键（key）、值（value）之间通过什么符号配对？ 冒号（:）
- 多个属性或方法之间使用什么符号分隔？ 逗号（,）
- 属性和方法的区别？ 值是不是函数

目录 Contents

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。

```
let person: {
  name: string;
  age: number;
}
```

```
person = {
  name: '刘老师',
  age: 18
}
```

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。

3. 接口

3.1 对象的类型注解

语法说明：

```
let person: {
  name: string;
  age: number;
}
```

```
person = {
  name: '刘老师',
  age: 18
}
```

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。

3. 接口

3.1 对象的类型注解

总结：

TS 中的对象是结构化的，在使用对象前，就可以根据需求，提前设计好对象的结构。

对象的结构化类型（类型注解）：建立一种契约，约束对象的结构。

```
let person: {
  name: string
  age: number
}
```

```
person = {
  name: '刘老师',
  age: 18
}
```

注意点：类型注解中键值对的值为类型！

3.2 对象方法的类型注解

问题：如何给对象中的方法，添加类型注解？

技巧：鼠标放在变量名称上，VSCode就会给出该变量的类型注解。

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

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。

注意：技巧是辅助，更重要的是理解。

3.3 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。

目录 Contents

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。

- 访问属性

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

需求：获取到对象（jay）的name属性。

```
console.log(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。

4. 取值和存值

4.1 取值

- 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. 取值和存值

4.1 取值

补充说明：

```
console.log(参数1, 参数2, ...)
```

实际上，console是一个对象，而log是该对象提供的一个方法。

并且，log方法可以有多个参数。

```
console.log('我叫', jay.name)
```

4. 取值和存值

4.1 取值

总结：

通过什么符号，来访问对象中的属性或方法？　　点语法（.）

注意：方法需要调用，所以，通过点语法拿到方法名称后，不要忘记使用小括号调用！

技巧：通过点语法，访问对象属性时，利用VSCode出来的提示，快速选择要访问的属性或方法。

该技巧很实用，特别是访问别人创建的对象时（比如：console对象）。

4. 取值和存值

4.2 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'
```

解释：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。

```
console.log(jay.name) // 周董
```

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。

总结

对象是对现实生活中具体事物（特征和行为）的抽象，可以使用对象来描述这些具体的事物。

对象包含：1 属性 2 方法。

简单来说：对象就是无序键值对的集合。

对象是结构化的，它的类型注解就是从对象的结构（属性、方法）出发，进行类型约束和检查。

推荐：使用接口来作为对象的类型注解，建立一种契约，约束对象的结构。

TS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）。

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

复杂数据类型：`object`（对象、数组）、`function`（函数）。

目录 Contents

5. 内置对象

5.1 概述

对象的两种来源：1 自己创建 2 其他人创建（编程语言自带或第三方）。

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。

1. 学习内置对象，需要学什么？ 常用属性和方法
2. 怎么学？ 查文档

5.2 学习方式 - 查文档

注意：内置对象中提供了非常多的方法或属性，以满足开发中各种各样的需求。

编程不是死记硬背，而是掌握一定的技巧，查文档就是最重要的一个。

文档地址：MDN（更标准）/ W3school（国内）

5. 内置对象

总结：

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。

学什么？学内置对象中的属性或方法。

怎么学？查文档，文档地址：MDN（更标准）/ W3school（国内）

5.3 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）。

5. 内置对象

5.3 数组对象 - length

- `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
```

5. 内置对象

5.3 数组对象 - push

- `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']
```

使用 `push` 方法：

```
songs.push('痒')
```

原来的方式：使用数组长度作为索引

```
songs[songs.length] = '痒' // => songs[3] = '痒'
```

5.3 数组对象 - forEach

- forEach 方法：遍历数组。

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

原来的方式：使用 for 循环遍历数组

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

使用 forEach:

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

5.3 数组对象 - forEach

forEach 的使用说明：

```
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。

5. 内置对象

5.3 数组对象 - forEach

forEach 方法的说明：

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

疑问：不需要为回调函数的参数或返回值指定类型注解吗？

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！

forEach 方法，可以根据当前数组的类型，自动推导出回调函数中参数的类型。

注意：回调函数中的参数可以用任意名称，并且，如果没有用到，可以省略。

```
songs.forEach(function (a, b) {}) // OK! a → 数组元素  b → 索引
songs.forEach(function (item) {}) // OK! 索引没用到，直接省略
```

5.3 数组对象 - some

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：

```
let has: boolean = false
nums.forEach(function (num) {
  if (num > 10) {
    has = true
  }
})
```

问题：遍历整个数组（循环执行了5次），无法中间停止，这种情况下，效率低。

5.3 数组对象 - some

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
})
```

5.3 数组对象 - some

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。

```
let has: boolean = 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。

补充：TS 的类型推论

在 TS 中，某些没有明确指出类型的地方，类型推论会帮助提供类型。

换句话说：由于类型推论的存在，这些地方，类型注解可以省略不写！

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 }
```

注意：这两种情况下，类型注解可以省略不写！

推荐：能省略类型注解的地方，就省略（偷懒、充分利用TS类型推论的能力，提升开发效率）。

学习的时候，培养大家去建立类型思维；出师了，可以去繁就简。

传智播客旗下高端IT教育品牌