

# TypeScript 常见面试题

**TypeScript是Microsoft开发的JavaScript的开源超集，用于在不破坏现有程序的情况下添加附加功能。**

由于其独特的优势，例如,静态类型和许多速记符号，TypeScript现在被前端和全栈开发人员广泛用于大型项目。

## 1、TypeScript的主要特点是什么？

- A. 跨平台：TypeScript 编译器可以安装在任何操作系统上，包括 Windows、macOS 和 Linux。
- B. ES6 特性：TypeScript 包含计划中的 ECMAScript 2015 (ES6) 的大部分特性，例如箭头函数。
- C. 面向对象的语言：TypeScript提供所有标准的OOP功能，如类、接口和模块。
- D. 静态类型检查：TypeScript使用静态类型并帮助在编译时进行类型检查。因此，你可以在编写代码时发现编译时错误，而无需运行脚本。
- E. 可选的静态类型：如果你习惯了JavaScript的动态类型，TypeScript 还允许可选的静态类型。
- F. DOM操作：您可以使用TypeScript来操作DOM以添加或删除客户端网页元素。

## 2、使用 TypeScript 有什么好处？

- A. TypeScript 更具表现力，这意味着它的语法混乱更少。
- B. 由于高级调试器专注于在编译时之前捕获逻辑错误，因此调试很容易
- C. 静态类型使 TypeScript比JavaScript的动态类型更易于阅读和结构化。
- D. 由于通用的转译，它可以跨平台使用，在客户端和服务端项目中。

## 3、TypeScript 的内置数据类型有哪些？

- A. 数字类型：用于表示数字类型的值。TypeScript 中的所有数字都存储为浮点值。

ini 复制代码

```
let identifier: number = value;
```

B. 布尔类型：一个逻辑二进制开关，包含true或false

ini 复制代码

```
let identifier: string = " ";
```

C. Null 类型： Null 表示值未定义的变量。

ini 复制代码

```
let identifier: bool = Boolean value;
```

D. 未定义类型：一个未定义的字面量，它是所有变量的起点。

ini 复制代码

```
let num: number = null;
```

E. void 类型：分配给没有返回值的方法的类型。

ini 复制代码

```
let unusable: void = undefined;
```

## 4、TypeScript 目前的稳定版本是什么？

根据最新的去回答 当前的稳定版本是 4.2.3。

## 5、TypeScript 中的接口是什么？

1. 接口为使用该接口的对象定义契约或结构。
2. 接口是用关键字定义的interface，它可以包含使用函数或箭头函数的属性和方法声明。

typescript 复制代码

```
interface IEmployee {  
    empCode: number;  
    empName: string;  
    getSalary: (number) => number; // arrow function
```

```
getManagerName(number): string;  
}
```

## 6、TypeScript 中的模块是什么？

TypeScript 中的模块是相关变量、函数、类和接口的集合。

你可以将模块视为包含执行任务所需的一切的容器。可以导入模块以轻松地在项目之间共享代码。

[复制代码](#)

TypeScript 中的模块是相关变量、函数、类和接口的集合。

你可以将模块视为包含执行任务所需的一切的容器。可以导入模块以轻松地在项目之间共享代码。

## 7、后端如何使用TypeScript？

你可以将 Node.js 与 TypeScript 结合使用，将 TypeScript 的优势带入后端工作。

只需输入以下命令，即可将 TypeScript 编译器安装到你的 Node.js 中：

[css 复制代码](#)

```
npm i -g typescript
```

## 8、TypeScript 中的类型断言是什么？

TypeScript 中的类型断言的工作方式类似于其他语言中的类型转换，但没有 C# 和 Java 等语言中可能的类型检查或数据重组。类型断言对运行时没有影响，仅由编译器使用。

类型断言本质上是类型转换的软版本，它建议编译器将变量视为某种类型，但如果它处于不同的形式，则不会强制它进入该模型。

## 9、如何在 TypeScript 中创建变量？

你可以通过三种方式创建变量：var，let，和const。var是严格范围变量的旧风格。你应该尽可能避免使用，var因为它会在较大的项目中导致问题。

[ini 复制代码](#)

```
var num:number = 1
```

let是在 TypeScript中声明变量的默认方式。与var相比，let减少了编译时错误的数量并提高了代码的可读性。

ini 复制代码

```
let num:number = 1;
```

const创建一个其值不能改变的常量变量。它使用相同的范围规则，let并有助于降低整体程序的复杂性。

ini 复制代码

```
const num:number = 100;
```

## 10、在TypeScript中如何从子类调用基类构造函数？

你可以使用该super()函数来调用基类的构造函数。

typescript 复制代码

```
class Animal {
  name: string;
  constructor(theName: string) {
    this.name = theName;
  }
  move(distanceInMeters: number = 0) {
    console.log(`${this.name} moved ${distanceInMeters}m.`);
  }
}
class Snake extends Animal {
  constructor(name: string) {
    super(name);
  }
  move(distanceInMeters = 5) {
    console.log("Slithering...");
    super.move(distanceInMeters);
  }
}
```

## 11、解释如何使用 TypeScript mixin。

Mixin 本质上是在相反方向上工作的继承。Mixins 允许你通过组合以前类中更简单的部分类设置来构建新类。相反，类A继承类B来获得它的功能，类B从类A需要返回一个新类的附加功能。

## 12、TypeScript 中如何检查 null 和 undefined?

你可以使用 `juggle-check`，它检查 `null` 和 `undefined`，或者使用 `strict-check`，它返回 `true` 设置为 `null` 的值，并且不会评估 `true` 未定义的变量。

typescript 复制代码

```
//juggle
if (x == null) {
}

var a: number;
var b: number = null;
function check(x, name) {
  if (x == null) {
    console.log(name + ' == null');
  }
  if (x === null) {
    console.log(name + ' === null');
  }
  if (typeof x === 'undefined') {
    console.log(name + ' is undefined');
  }
}
check(a, 'a');
check(b, 'b');
```

## 13、TypeScript中的getter/setter是什么？你如何使用它们？

Getter 和 setter是特殊类型的方法，可帮助你根据程序的需要委派对私有变量的不同级别的访问。

Getters 允许你引用一个值但不能编辑它。Setter允许你更改变量的值，但不能查看其当前值。这些对于实现封装是必不可少的。

例如，新雇主可能能够了解get公司的员工人数，但无权set了解员工人数。

typescript 复制代码

```
const fullNameMaxLength = 10;
class Employee {
  private _fullName: string = "";
  get fullName(): string {
    return this._fullName;
  }
  set fullName(newName: string) {
```

```

    if (newName && newName.length > fullNameMaxLength) {
        throw new Error("fullName has a max length of " + fullNameMaxLength);
    }
    this._fullName = newName;
}
}
let employee = new Employee();
employee.fullName = "Bob Smith";
if (employee.fullName) {
    console.log(employee.fullName);
}

```

## 14、如何允许模块外定义的类可以访问？

你可以使用export关键字打开模块以供在模块外使用。

typescript 复制代码

```

module Admin {
    // use the export keyword in TypeScript to access the class outside
    export class Employee {
        constructor(name: string, email: string) { }
    }
    let alex = new Employee('alex', 'alex@gmail.com');
}
// The Admin variable will allow you to access the Employee class outside the module with the help of
let nick = new Admin.Employee('nick', 'nick@yahoo.com');

```

## 15、如何使用 Typescript 将字符串转换为数字？

与 JavaScript 类似，你可以使用parseInt或parseFloat函数分别将字符串转换为整数或浮点数。你还可以使用一元运算符+将字符串转换为最合适的数字类型，“3”成为整数，3而“3.14”成为浮点数3.14。

ini 复制代码

```

var x = "32";
var y: number = +x;

```

## 16、什么是 .map 文件，为什么/如何使用它？

甲.map文件是源地图，显示原始打字稿代码是如何解释成可用的JavaScript代码。它们有助于简化调试，因为你可以捕获任何奇怪的编译器行为。

调试工具还可以使用这些文件来允许你编辑底层的 TypeScript 而不是发出的 JavaScript 文件。

## 17、TypeScript 中的类是什么？你如何定义它们？

类表示一组相关对象的共享行为和属性。

例如，我们的类可能是Student，其所有对象都具有该attendClass方法。另一方面，John是一个单独的 type 实例，Student可能有额外的独特行为，比如attendExtracurricular。

你使用关键字声明类class：

typescript 复制代码

```
class Student {
  studCode: number;
  studName: string;
  constructor(code: number, name: string) {
    this.studName = name;
    this.studCode = code;
  }
}
```

## 18、TypeScript 与 JavaScript 有什么关系？

TypeScript 是 JavaScript 的开源语法超集，可编译为 JavaScript。所有原始 JavaScript 库和语法仍然有效，但 TypeScript 增加了 JavaScript 中没有的额外语法选项和编译器功能。

TypeScript 还可以与大多数与 JavaScript 相同的技术接口，例如 Angular 和 jQuery。

## 19、TypeScript 中的 JSX 是什么？

JSX 是一种可嵌入的类似于 XML 的语法，允许你创建 HTML。TypeScript 支持嵌入、类型检查和将 JSX 直接编译为 JavaScript。

## 20、TypeScript 支持哪些 JSX 模式？

TypeScript有内置的支持preserve，react和react-native。

- preserve 保持 JSX 完整以用于后续转换。
- react不经过 JSX转换，而是react.createElement作为.js文件扩展名发出和输出。

- react-native结合起来preserve, react因为它维护所有 JSX 和输出作为.js扩展。

## 21、如何编译 TypeScript 文件?

你需要调用 TypeScript 编译器tsc来编译文件。你需要安装 TypeScript 编译器, 你可以使用 npm.

```
npm install -g typescript
tsc <TypeScript File Name>
```

arduino 复制代码

## 22、TypeScript 中有哪些范围可用? 这与JS相比如何?

- 全局作用域: 在任何类之外定义, 可以在程序中的任何地方使用
- 函数/类范围: 在函数或类中定义的变量可以在该范围内的任何地方使用。
- 局部作用域/代码块: 在局部作用域中定义的变量可以在该块中的任何地方使用。

## 23、TypeScript 中的箭头/lambda 函数是什么?

胖箭头函数是用于定义匿名函数的函数表达式的速记语法。它类似于其他语言中的 lambda 函数。箭头函数可让你跳过function关键字并编写更简洁的代码。

## 24、解释rest参数和声明rest参数的规则。

其余参数允许你将不同数量的参数(零个或多个)传递给函数。当你不确定函数将接收多少参数时, 这很有用。其余符号之后的所有参数...都将存储在一个数组中。 例如:

```
function Greet(greeting: string, ...names: string[]) {
    return greeting + " " + names.join(", ") + "!";
}
Greet("Hello", "Steve", "Bill"); // returns "Hello Steve, Bill!"
Greet("Hello");// returns "Hello !"
```

csharp 复制代码

rest 参数必须是参数定义的最后一个, 并且每个函数只能有一个 rest 参数。

## 25、什么是三斜线指令? 有哪些三斜杠指令?



三斜线指令是单行注释，包含用作编译器指令的 XML 标记。每个指令都表示在编译过程中要加载的内容。三斜杠指令仅在其文件的顶部工作，并且将被视为文件中其他任何地方的普通注释。

- `///` 是最常见的指令，定义文件之间的依赖关系。
- `///` 类似于 `path` 但定义了包的依赖项。
- `///` 允许您显式包含内置 `lib` 文件。

## 26、Omit类型有什么作用？

`Omit`是实用程序类型的一种形式，它促进了常见的类型转换。`Omit`允许你通过传递电流Type并选择Keys在新类型中省略来构造类型。

```
Omit<Type, Keys>
```

python 复制代码

例如：

```
interface Todo {  
  title: string;  
  description: string;  
  completed: boolean;  
  createdAt: number;  
}  
  
type TodoPreview = Omit<Todo, "description">;
```

ini 复制代码

## 27、TypeScript中如何实现函数重载？

要在 TypeScript 中重载函数，只需创建两个名称相同但参数/返回类型不同的函数。两个函数必须接受相同数量的参数。这是 TypeScript 中多态性的重要组成部分。

例如，你可以创建一个 `add` 函数，如果它们是数字，则将两个参数相加，如果它们是字符串，则将它们连接起来。

```
function add(a:string, b:string):string;  
function add(a:number, b:number): number;  
function add(a: any, b:any): any {  
  return a + b;  
}
```

sql 复制代码

```
add("Hello ", "Steve"); // returns "Hello Steve"
add(10, 20); // returns 30
```

## 28、如何让接口的所有属性都可选？

你可以使用partial映射类型轻松地将所有属性设为可选。

## 29、什么时候应该使用关键字unknown？

unknown，如果你不知道预先期望哪种类型，但想稍后分配它，则应该使用该any关键字，并且该关键字将不起作用。

## 30、什么是装饰器，它们可以应用于什么？

装饰器是一种特殊的声明，它允许你通过使用@注释标记来一次性修改类或类成员。每个装饰器都必须引用一个将在运行时评估的函数。

例如，装饰器@sealed将对应于sealed函数。任何标有的@sealed都将用于评估sealed函数。

lua 复制代码

```
function sealed(target) {
  // do something with 'target' ...
}
```

它们可以附加到：

- 类声明
- 方法
- 配件
- 特性
- 参数

注意：默认情况下不启用装饰器。要启用它们，你必须experimentalDecorators从tsconfig.json文件或命令行编辑编译器选项中的字段。