微软开发的自由和开源编程语言，js的超集，本质上向js添加了可选的静态类型和基于类的面向对象编程。

基础类型

1. boolean
2. number
3. string
4. 数组类型　number[]　Array<number>
5. 元组类型　已知元素数量和类型的数组，各元素的类型不必相同
6. 枚举类型　枚举 enum 默认从0开始，也可手动赋值，取值时也可指定索引
7. any　　未确定变量类型
8. void　　与any类型相反，表示没有任何类型（常用于无返回值函数）只能赋予 undefined 或 null
9. null和undefined
10. never　　永不存在的值的类型，抛出异常或无返回值的函数/箭头函数
11. object　　非原始类型，也就是除number，string，boolean，symbol，null或undefined之外的类型
12. 联合类型　　`|`分割数据类型，变量类型不确定时只能所有联合类型的共有属性和方法，赋值时推断数据类型
13. 类型断言　　类型转换，不进行特殊的数据检查和解构，可用尖括号或as语法（JSX只能使用as）

类型推论：TypeScript 会在没有明确的指定类型的时候推测出一个类型。
如果定义的时候没有赋值，不管之后有没有赋值，都会被推断成 any 类型而完全不被类型检查：

变量声明

1. var
2. function
3. let　　块级作用域、重定义和屏蔽
4. const　　与let相同的特性，不可重新赋值
5. 解构　　对象、数组 属性重命名（对象专用） 默认值 函数声明
6. 展开　　将一个数组展开为另一个数组，或将一个对象展开为另一个对象
7. class

接口

TypeScript的核心原则之一是对值所具有的结构进行类型检查。它有时被称做“鸭式辨型法”或“结构性子类型化”。在TypeScript里，接口的作用就是为这些类型命名和为你的代码或第三方代码定义契约。

ts类型检查器不会去检查属性的顺序，只要相应的属性存在并且类型正确即可

```
interface LabelledValue {
   label: string;
}

function $printLabel(labelledObj: LabelledValue) {
   console.log(labelledObj.label);
}

let $myObj = { size: 10, label: "Size 10 Object" };
$printLabel($myObj);
```

readonly：只读
？：可选属性

额外的属性检查
使用接口未声明的属性会报错，若想绕开检查，可以使用**类型断言**

最佳方式：添加一个**字符串索引签名**
赋值给另一个变量也不会经过额外属性检查

注：通常情况下不应该绕过额外的属性检查，bug机率很高

函数类型

接口能够描述JavaScript中对象拥有的各种各样的外形。 除了描述带有属性的普通 对象外，接口也可以描述函数类
型。

接口调用是一个只要有参数列表和返回值类型的函数定义。参数列表里的每个参数都需要名字和类型。

类型检查：要求对应位置上的参数类型兼容；无指定类型，ts类型系统会推断参数类型

可索引的类型
描述对象索引的类型，还有相应的索引返回值类型

函数

普通函数
函数参数
type语句定义类型
interface语句定义类型

类：与js定义方法一致，但是增加了public， private， protected， readonly等访问控制修饰符

泛型：TypeScript 的泛型和接口使得具备较强的类型检查能力的同时，很好地兼顾了 JavaScript 语言的动态特性。

tsconfig.json 配置文件：每个 TypeScript 项目都需要一个 tsconfig.json 文件来指定相关的配置，比如告诉 TypeScript 编译器要将代码转换成 ES5 还是 ES6 代码等。

使用第三方模块

一般情况下在 TypeScript 中是不能"直接"使用 npm 上的模块的

由于 TypeScript 项目最终会编译成 JavaScript 代码执行，当我们在 TypeScript 源码中引入这些被编译成 JavaScript 的模块时，它需要相应的声明文件（.d.ts文件）来知道该模块类型信息，这些声明文件可以通过设置tsconfig.json中的declaration: true来自动生成。而那些不是使用 TypeScript 编写的模块，也可以通过手动编写声明文件来兼容 TypeScript（下文会讲解）。

为了让广大开发者更方便地使用 npm 上众多非 TypeScript 开发的模块，TypeScript 官方建立了一个名叫 DefinitelyTyped 的仓库，任何人都可以通过 GitHub 在上面修改或者新增 npm 模块的声明文件，经多几年多的发展，这个仓库已经包含了大部分常用模块的声明文件，而且仍然在继续不断完善。当遇到缺少模块声明文件的情况，开发者可以尝试通过 npm install @types/xxx 来安装模块声明文件即可。

如果我们使用的第三方模块在 DefinitelyTyped 找不到对应声明文件，也可以尝试使用 require()这个终极的解决方法，它会将模块解析成 any 类型，不好的地方就是没有静态类型检查了。

编写 typings 声明文件

编写 .d.ts 文件还是比较繁琐的，比如要完整地给 express 编写声明文件，首先得了解这个模块都有哪些接口，而且 JavaScript 模块普遍接口比较 灵活，同一个方法名可能接受各种各样的参数组合。所以，大多数情况下我们只会定义我们需要用到的接口，下文以 express 模块为例。

TSLint 代码规范检查

在编写 JavaScript 代码时，我们可以通过 ESLint 来进行代码规范检查，编写 TypeScript 代码时也可以使用 TSLint，两者在配置上也有些相似。对于初学者来说，使用 TSLint 可以知道哪些程序的写法是不被推荐的，从而养成更好的 TypeScript 代码风格。

发布模块

相比直接使用 JavaScript 编写的 npm 模块，使用 TypeScript 编写的模块需要增加以下几个额外的工作：

- 发布前将 TypeScript 源码编译成 JavaScript.
- 需要修改 tsconfig.json 的配置，使得编译时生成模块对应的 .d.ts 文件。
- 在 package.json 文件增加 types 属性。

单元测试

要执行使用 TypeScript 编写的单元测试程序，可以有两种方法：
* 先通过 tsc 编译成 JavaScript 代码后，再执行。
* 直接执行 .ts 源文件。

从 JavaScript 项目渐进式迁移

- 项目代码太多，无法短时间内将源码从 JavaScript 改写为 TypeScript，两者需要共存一段时间。
- 依赖的第三方库没有 .d.ts 文件。
- 原 JavaScript 代码使用了很多的动态特性，无法简单转换代码。

在 Webpack 中使用

Webpack 是当前最流行的前端构建工具之一，要在 Webpack 中使用 TypeScript 可以通过 ts-loader模块来实现。

通过开源项目进一步学习

通过学习相关大型开源项目的代码，可以学习到别人是怎么使用 TypeScript 的。以下是当前比较火热的 TypeScript 开源项目：

- TypeScript 编译器，其本身也是通过 TypeScript 编写的。
- Visual Studio Code，微软开源的跨平台代码编辑器。
- AngularJS 2，谷歌开源的用于构建跨平台 Web 应用的开发框架。
- TSLint，一款 TypeScript 代码规范检查工具。
- Ant Design，一套企业级的 UI 设计语言和 React 实现。
- RxJS 5，JavaScript 响应式编程库。