TypeScript，主讲：汤小洋

一、TypeScript简介

1. 介绍

TypeScript是由微软开发的开源、跨平台的编程语言，简称TS

TypeScript是JavaScript的超集，是在JavaScript基础上进行的功能性扩展，语法更严格、更简洁

TypeScript无法直接运行，需要转换为JavaScript才能运行

TypeScript代码是定义在以 `.ts` 结尾的文件中，最终需要编译为 `.js` 结尾的文件

参考：https://www.tslang.cn

2. 优点

提供类型系统：增强了代码的可读性和可维护性，在编译阶段就能发现大部分错误

支持ES6：ES6规范是客户端脚本语言的发展方向

强大的IDE支持：类型检测、语法提示

二、环境配置

1. 安装环境

将.ts文件转换为.js文件，需要使用TypeScript编译器

使用npm全局安装TypeScript编译器工具

- 执行：`npm install typescript -g`，安装TypeScript编译器
- 安装后提供一个全局命令工具：`tsc`

```
tsc -v  // 查看版本
tsc -h  // 查看帮助
tsc 选项 ts文件路径 // 编译ts文件，默认会在相同位置生成一个同名的js文件
tsc 01.first.js
tsc --outDir ./js 01.first.js
```

注：VSCode内置了TypeScript插件，提供了TypeScript语法校验功能，会对所有.ts文件进行语法校验

2. TypeScript项目

如果一个目录下存在一个 `tsconfig.json` 文件，那么它意味着这个目录是TypeScript项目的根目录

- 包含tsconfig.json文件的目录为一个独立的TypeScript项目
- 可以在tsconfig.json文件中设置编译器选项，如转换规则、输出目录等
- 在项目根目录下执行： `tsc --init` ，可以进行TypeScript项目的初始化，会自动生成tsconfig.json文件

自动监视编译项目：

- 在TypeScript项目根目录下执行： `tsc -w -p tsconfig.json` ，对项目开启自动监视编译
- 在VSCode中执行：终端——>运行任务——>tsc:监视-tsconfig.json

tsconfig.json配置项可参考 https://www.tslang.cn/docs/handbook/compiler-options.html

三、TypeScript语法

TS对JS扩展的相关语法功能

1. 变量

JS中定义变量： `var/let/const 变量名 = 变量值;`

TS中定义变量： `var/let/const 变量名:数据类型 = 变量值;`

- 增加了数据类型的限制
- 该功能主要为开发者在开发阶段，提供了限定变量类型限制的功能

语言类型：

- 弱类型语言

  在定义变量时无需指定数据类型，且变量的类型可以修改，如 js、php

- 强类型语言

  在定义变量时需要指定数据类型，且变量的类型不能改变，如 java、ts

2. 数据类型

TS除了支持JS中的数据类型外，还扩展了一些数据类型：

- JS中的类型：string、number、boolean、Array、null和undefined
- TS新增类型：元组、枚举enum、any、void、never

3. 函数

语法：

```
function 函数名(参数名:数据类型,参数名?:数据类型,参数名:数据类型=默认值):返回值类型{
    // 函数体
}
```

4. 类

4.1 类的使用

一个类就是一个数据类型，定义一个类其实就是创建了一种数据类型

语法：

```
class 类名 {
    成员属性
    成员方法
    构造函数
}
```

4.2 继承

可以让一个类继承自另一个类，此时该类会继承另一个类中的属性和方法

继承而得到的类称为子类（派生类），被继承的类称为父类（超类/基类）

继承是一种 `is a` 的关系，比如： `Cat is a Animal—>猫是动物` 或 `Student is a Person—>学生是人`

语法：

```
class 子类 extends 父类 {

}
```

作用：

- 代码复用：将多个子类中相同的属性和方法放到父类中
- 功能扩展：子类可以有自己独特的属性和方法

方法的重写：

- 在子类中可以重写父类中的方法，称为方法重写override
- 用来重新定义子类的行为，解决父类和子类的差异性

4.3 修饰符

用来控制属性或方法的访问范围

- public（公开）：可以在任何地方访问
- protected（受保持）：可以在当前类和子类中访问，在类的外部无法访问
- private（私有）：只能在当前类中访问

4.4 封装

将类的属性封装在类中，不允许在类的外部直接访问，保护数据的安全，使内容可控

只能通过被授权的方法才能对数据进行访问，称为存取器 getters/setters

步骤：

1. 将属性私有化

   使用 `private` 修饰属性，命名上一般以下划线 `_` 开头
2. 提供对外访问的方法，用于对属性进行取值和赋值

   取值： `get 新属性名(){ 控制私有属性的取值 }`

   赋值： `set 新属性名(新值){ 控制私有属性的赋值 }`
3. 访问新属性，实际上就是在对私有属性进行操作

   取值： `对象名.新属性名`

   赋值： `对象名.新属性名=新值`

注：本质上是通过JS中的 `Object.defineProperty` 进行数据劫持

4.5 抽象类

如何防止父类被实例化？

如何保证子类必须重写父类的方法？

使用abstract关键字：

- 被abstract修饰的类，称为抽象类

  定义方式： `abstract class 类名{ }`

  抽象类不能被实例化，即不能使用new创建一个对象，只能被继承
- 被abstract修饰的方法，称为抽象方法

  定义方式： `abstract 方法名():返回值类型;`

  抽象方法只有声明，没有具体实现，即没有方法体，以分号结尾

特性：

- 抽象类中可以有抽象方法，也可以没有抽象方法
- 含有抽象方法的类，必须为抽象类
- 子类继承抽象类后，必须实现/重写抽象类中所有的抽象方法，否则子类仍然为抽象类

4.6 多态

多态是具有表现多种形态的能力的特征，即一种事物，具有多个形态

- 将父类的引用指向子类的对象
- 将父类作为方法形参，将子类的对象作为方法实参，从而实现多态

面向对象的三大特性：封装、继承、多态

5. 接口

5.1 接口的定义

接口是一种规范约束，起到限制和规范的作用，可以强制一个类必须符合某个规范，即实现接口

实现某个接口的类，必须实现这个接口中所有属性和方法

定义接口：

```
interface 接口名{
    声明属性 // 只能声明属性，不能为属性赋值
    声明方法; // 只能声明方法，不能定义方法体
}
```

实现接口：

```
class 类名 implements 接口名{
    // 必须实现接口中所有的属性和方法
}
```

5.2 约束其他类型

TS中的接口还可以对函数、对象、数组等进行约束

- 使用接口表示函数类型，即通过接口对函数进行约束
- 使用接口表示数组类型，即通过接口对数组进行约束
- 使用接口表示对象类型，即通过接口对对象进行约束

6. 泛型

6.1 简介

Generic Type泛型本质是参数化类型，所操作的数据类型被指定为一个参数，在使用时确定此类型。

通俗点来说，在定义时不知道具体的类型，在使用时要指定具体的类型，类似于参数，所以称为参数化类型

分类：

- 泛型类
- 泛型接口
- 泛型函数

6.2 用法

泛型类：

- 表示类中有一个未知的类型
- 定义方式：`class 类名<T>{}`，T表示的是一种类型，是泛型的类型参数，可以使用任意标识，一般常用T、E、K、V等
- 在使用类时需要在类名后通过`<类型>`指定具体的类型

泛型接口：

- 表示接口中有一个未知的类型
- 定义方式：`interface 接口名<T>{}`
- 在使用接口时需要在接口名后通过`<类型>`指定具体的类型

泛型函数：

- 表示函数中有一个未知的类型
- 定义方式：`function 函数名<T>(){}`
- 在调用方法时指定具体的类型

7. 模块

可以把一些公共的功能单独抽离成一个文件，作为一个模块，实现复用

- 每个模块都有一个独立的作用域，一个模块中的成员在模块外部是无法访问的
- 可以通过 导出/导入模块来实现模块间的访问

  导出模块：`export`

  导入模块：`import`

8. 命名空间

在代码量较大的情况下，为了避免变量命名冲突，可以将代码放到不同的命名空间中，更好的组织代码结构

- 每个命名空间都有一个独立的作用域，一个命名空间中的成员在外部是无法访问的
- 可以使用`export`关键字将命名空间内的成员进行导出，以便在外部访问

命名空间和模块的区别：

- 命名空间主要用于组织代码，避免命名冲突
- 模块主要用于代码的封装复用，一个模块中可以包含多个命名空间

9. 装饰器

装饰器是一种特殊类型的声明，它能够被附加到类、属性、方法或参数上，进行额外功能的扩展

简单来说，装饰器就是一个函数，该函数在运行时会被调用，并传入一个特殊的参数，称为装饰器函数

装饰器的种类：

- 类装饰器
- 属性装饰器
- 方法装饰器
- 参数装饰器

装饰器的写法：

- 普通装饰器，无法自定义传参
- 装饰器工厂，可以自定义传参

注：需要在 `tsconfig.json` 中启用装饰器功能： `"esModuleInterop": true`