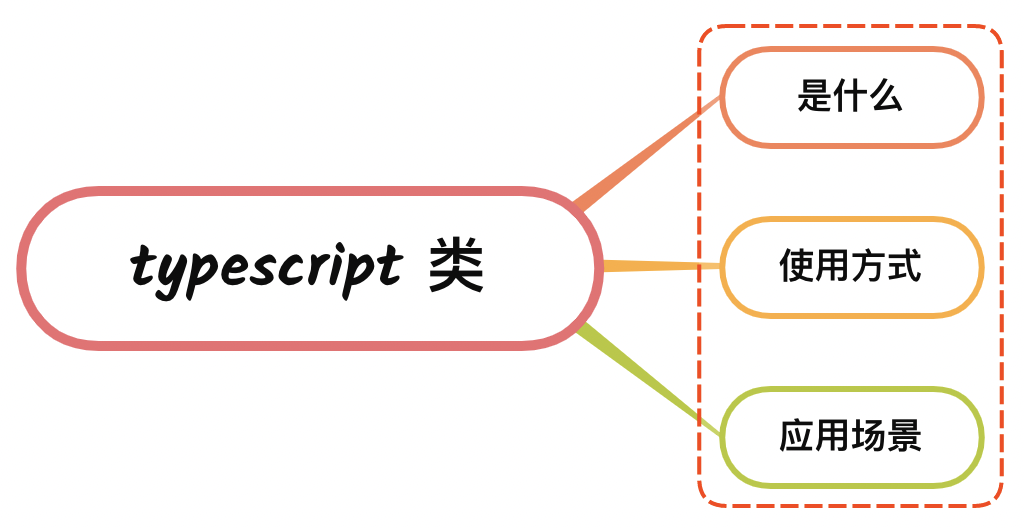
# 面试官：说说你对 TypeScript 中类的理解？应用场景？



## 一、是什么

类（Class）是面向对象程序设计（OOP，Object-Oriented Programming）实现信息封装的基础

类是一种用户定义的引用数据类型，也称类类型

传统的面向对象语言基本都是基于类的，JavaScript 基于原型的方式让开发者多了很多理解成本

在 ES6 之后，JavaScript 拥有了 class 关键字，虽然本质依然是构造函数，但是使用起来已经方便了许多

但是JavaScript 的class依然有一些特性还没有加入，比如修饰符和抽象类

TypeScript 的 class 支持面向对象的所有特性，比如 类、接口等

## 二、使用方式

定义类的关键字为 class，后面紧跟类名，类可以包含以下几个模块（类的数据成员）：

* **字段** ： 字段是类里面声明的变量。字段表示对象的有关数据。
* **构造函数**： 类实例化时调用，可以为类的对象分配内存。
* **方法**： 方法为对象要执行的操作

如下例子：

class Car {  
 // 字段  
 engine:string;  
  
 // 构造函数  
 constructor(engine:string) {  
 this.engine = engine  
 }  
  
 // 方法  
 disp():void {  
 console.log("发动机为 : "+this.engine)  
 }  
}

### 继承

类的继承使用过extends的关键字

class Animal {  
 move(distanceInMeters: number = 0) {  
 console.log(`Animal moved ${distanceInMeters}m.`);  
 }  
}  
  
class Dog extends Animal {  
 bark() {  
 console.log('Woof! Woof!');  
 }  
}  
  
const dog = new Dog();  
dog.bark();  
dog.move(10);  
dog.bark();

Dog是一个 派生类，它派生自 Animal 基类，派生类通常被称作子类，基类通常被称作 超类

Dog类继承了Animal类，因此实例dog也能够使用Animal类move方法

同样，类继承后，子类可以对父类的方法重新定义，这个过程称之为方法的重写，通过super关键字是对父类的直接引用，该关键字可以引用父类的属性和方法，如下：

class PrinterClass {  
 doPrint():void {  
 console.log("父类的 doPrint() 方法。")  
 }  
}  
  
class StringPrinter extends PrinterClass {  
 doPrint():void {  
 super.doPrint() // 调用父类的函数  
 console.log("子类的 doPrint()方法。")  
 }  
}

### 修饰符

可以看到，上述的形式跟ES6十分的相似，typescript在此基础上添加了三种修饰符：

* 公共 public：可以自由的访问类程序里定义的成员
* 私有 private：只能够在该类的内部进行访问
* 受保护 protect：除了在该类的内部可以访问，还可以在子类中仍然可以访问

### 私有修饰符

只能够在该类的内部进行访问，实例对象并不能够访问



并且继承该类的子类并不能访问，如下图所示：

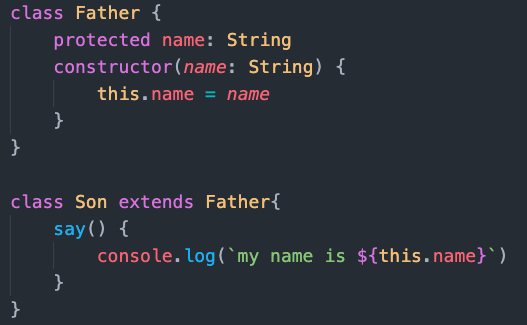


### 受保护修饰符

跟私有修饰符很相似，实例对象同样不能访问受保护的属性，如下：



有一点不同的是 protected 成员在子类中仍然可以访问



除了上述修饰符之外，还有只读**修饰符**

#### 只读修饰符

通过readonly关键字进行声明，只读属性必须在声明时或构造函数里被初始化，如下：



除了实例属性之外，同样存在静态属性

### 静态属性

这些属性存在于类本身上面而不是类的实例上，通过static进行定义，访问这些属性需要通过 类型.静态属性 的这种形式访问，如下所示：

class Square {  
 static width = '100px'  
}  
  
console.log(Square.width) // 100px

上述的类都能发现一个特点就是，都能够被实例化，在 typescript中，还存在一种抽象类

### 抽象类

抽象类做为其它派生类的基类使用，它们一般不会直接被实例化，不同于接口，抽象类可以包含成员的实现细节

abstract关键字是用于定义抽象类和在抽象类内部定义抽象方法，如下所示：

abstract class Animal {  
 abstract makeSound(): void;  
 move(): void {  
 console.log('roaming the earch...');  
 }  
}

这种类并不能被实例化，通常需要我们创建子类去继承，如下：

class Cat extends Animal {  
  
 makeSound() {  
 console.log('miao miao')  
 }  
}  
  
const cat = new Cat()  
  
cat.makeSound() // miao miao  
cat.move() // roaming the earch...

## 三、应用场景

除了日常借助类的特性完成日常业务代码，还可以将类（class）也可以作为接口，尤其在 React 工程中是很常用的，如下：

export default class Carousel extends React.Component<Props, State> {}

由于组件需要传入 props 的类型 Props ，同时有需要设置默认 props 即 defaultProps，这时候更加适合使用class作为接口

先声明一个类，这个类包含组件 props 所需的类型和初始值：

// props的类型  
export default class Props {  
 public children: Array<React.ReactElement<any>> | React.ReactElement<any> | never[] = []  
 public speed: number = 500  
 public height: number = 160  
 public animation: string = 'easeInOutQuad'  
 public isAuto: boolean = true  
 public autoPlayInterval: number = 4500  
 public afterChange: () => {}  
 public beforeChange: () => {}  
 public selesctedColor: string  
 public showDots: boolean = true  
}

当我们需要传入 props 类型的时候直接将 Props 作为接口传入，此时 Props 的作用就是接口，而当需要我们设置defaultProps初始值的时候，我们只需要:

public static defaultProps = new Props()

Props 的实例就是 defaultProps 的初始值，这就是 class作为接口的实际应用，我们用一个 class 起到了接口和设置初始值两个作用，方便统一管理，减少了代码量

## 参考文献

* https://www.tslang.cn/docs/handbook/classes.html
* https://www.runoob.com/typescript/ts-class.html