02、TypeScript 类

# 目标

1. 了解TypeScript中的类及与JavaScript中类的区别
2. 熟悉TypeScript中类的相关编程技术

# 大纲

1. 介绍

JavaScript程序使用函数和基于原型的继承来模拟创建类。

而在TypeScript里，允许开发者使用类的面向对象特性来创建类。

1. 类的定义

与C#、Java、AS3类似，在TypeScript中定义类用Class

**class** Employee  
{  
 **fullName**:**string**;  
  
 **constructor**( message:**string** )  
 {  
 **this**.**fullName** = message;  
 }  
}

以上代码，我们定义了一个 Employee类。其包含1个公有属性和1个构造函数方法。

在类中，访问任意一个成员，都必须在前面加上“this”关键词。

1. 类的继承

在TypeScript里，我们可以使用常用的面向对象模式。

基于类的程序设计中最基本的模式是允许使用继承来扩展一个类。即“子类”对“父类”进行加强和扩展。

**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);  
 }  
}  
  
**class** Horse **extends** Animal {  
 **constructor**(name: **string**) { **super**(name); }  
 move(distanceInMeters = 45) {  
 ***console***.log("Galloping...");  
 **super**.move(distanceInMeters);  
 }  
}  
  
**let *sam*** = **new** Snake("Sammy the Python");  
**let *tom***: Animal = **new** Horse("Tommy the Palomino");  
  
***sam***.move();  
***tom***.move(34);

这个例子展示了TypeScript中继承的一些特征，与其它语言类似。 我们使用extends来创建子类。你可以看到Horse和Snake类是基类Animal的子类，并且可以访问其属性和方法。

子类访问父类的构造函数时使用 “super()”代替，访问父类其它成员使用“super.prop或者super.method()”。

1. 知识点：

* 在子类中重写父类的方法。这种做法称之为“覆盖”。
* 子类的构造函数中，通过使用“super()”来执行父类的构造方法。
* 子类实例可以赋值给父类类型，而反之父类实例不能赋值给子类类型。以上tom被声明为Animal类型，但它继承于Animal仍然可以被看成是一个Animal类型。

1. 构造函数

在TypeScript中定义类时，使用constructor方法来视作类的构造函数。

constructor方法是类在被实例化时，首先执行的方法。

在子类的构造方法中，通过 “super()”来调用父类的构造方法。

1. 类访问修饰符
2. 默认为公有（public）

在上面的例子里，我们可以自由的访问程序里定义的成员。

其它语言如C#要求，必须明确地使用public指定成员是可见的。

在TypeScript里，每个成员默认为public的。

你也可以明确的将一个成员标记成public。

我们可以用下面的方式来重写上面的Animal类：

**class** Animal  
{  
 **public name**:**string**;  
  
 **public constructor**( theName:**string** ){ **this**.**name** = theName; }  
  
 move( distanceInMeters:**number** )  
 {  
 ***console***.log( `${**this**.**name**} moved ${distanceInMeters}m.` );  
 }  
}

1. private

当成员被标记成private时，它就不能在声明它的类的外部访问。即该成员被标记为私有。

例如：

**class** Animal  
{  
 **private name**:**string**;  
  
 **constructor**( theName:**string** ){ **this**.**name** = theName; }  
}  
  
**new** Animal( "Cat" ).**name**; // Error: 'name' is private;

1. protected

protected修饰符与private修饰符的行为很相似，同样在类的外部无法被访问。

但有一点不同，protected成员在子类中仍然可以访问。即通过“super”关键字来进行访问。

构造函数默认不能加这个关键词修饰。

**class** Person  
{  
 **protected name**:**string**;  
  
 **constructor**( theName:**string** ){ **this**.**name** = theName; }  
}  
  
// Employee can extend Person  
**class** Employee **extends** Person  
{  
 **private department**:**string**;  
  
 **constructor**( name:**string**, department:**string** )  
 {  
 **super**( name );  
 **this**.**department** = department;  
 }  
  
 **public** getElevatorPitch()  
 {  
 **return** `Hello, my name is ${**this**.**name**} and I work in ${**this**.**department**}.`;  
 }  
}  
  
**let *howard*** = **new** Employee( "Howard", "Sales" );  
**let *john*** = **new** Person( "John" ); // Error: The 'Person' constructor is protected

1. static

类的静态成员，这些属性和方法存在于类本身上面而不是类的实例上。

**class** Grid  
{  
 **static** *ORIGIN* = { **x**: 0, **y**: 0 };  
  
 **static** *CALC*( x:**number**, y:**number** )  
 {  
 **return** x + y;  
 }  
}  
  
**var *g***:**number** = Grid.*CALC*( 1, 2 );  
***g*** = Grid.*CALC*( Grid.*ORIGIN*.**x**, Grid.*ORIGIN*.**y** );

1. 类的存取器（get和set）

TypeScript支持getters/setters来截取对对象成员的访问。 它能帮助你有效的控制对对象成员的访问。

**class** Employee  
{  
 // 公有属性  
 **fullName**:**string**;  
}  
//-------------------------------------------------------  
**let *passcode*** = "secret passcode";  
  
**class** Employee  
{  
 // 私有属性  
 **private \_fullName**:**string**;  
  
 **get** fullName():**string** {  
 **return this**.**\_fullName**;  
 }  
  
 **set** fullName( newName:**string** )  
 {  
 **if**( ***passcode*** && ***passcode*** == "secret passcode" )  
 {  
 **this**.**\_fullName** = newName;  
 }  
 **else** {  
 ***console***.log( "Error: Unauthorized update of employee!" );  
 }  
 }  
}

直接使用公有属性和同时具备get和set方法时的效果是一样的。

当只有get方法时，这个属性就变成了“只读”属性。只能访问不能修改。

当只有set 方法时，这个属性就变成了“只写”属性。只能赋值修改，不能访问。

注意：若要使用存取器，要求设置编译器输出目标为ECMAScript 5或更高。

PhpStorm的Setting中必须设置TypeScript的编译参数“--target ES5”。

1. 抽象类

抽象类是供其它类继承的基类。

他们一般不会直接被实例化，而是实例化继承并实现抽象方法的子类。

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。

**abstract class** Animal  
{  
 **abstract** makeSound():**void**;  
  
 move():**void** {  
 ***console***.log( 'roaming the earch...' );  
 }  
}  
  
**class** Horse **extends** Animal  
{  
 **public** makeSound():**void** {  
 ***console***.log( "yeah....hoho...." );  
 }  
}

**注意**：只要有一个抽象方法存在于类中，那么该类的声明中 class 关键字前也同样必须加上 abstract 关键词。

1. 练习实例

设计一个类Student类，包含name属性和say()方法，而say()是抽象方法。

再设计2个类（Me类和Him类）都继承于Student，都要实现say()方法。

创建2个类的实例，分别调用 say() 方法，在控制台输出“说话内容”。