TypeScript 类

目标

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

大纲

# 类的定义

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。

## 定义类

与C#、Java、AS3等面向对象语言类似，在TypeScript中定义类用Class关键字，如下所示：

|  |
| --- |
| **class** Employee {  **fullName**:**string**;   **constructor**( fullName:**string** )  {  **this**.**fullName** = fullName;  }   greet()  {  **return** "Hello, " + **this**.**fullName**;  } } |

­以上代码，我们定义了一个 Employee类。其包含3个成员：1个公有属性和1个构造函数方法，以及一个公有的greet方法。

**注意：**在类中，访问任意一个成员，都必须在前面加上“this”关键词。

## 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。

## 类访问修饰符

### public（默认）

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.` );  } } |

### 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; |

### 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 |

### static

类的静态成员，这些属性和方法存在于类本身上而不是类的实例上。

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

**思考：**想想如何让**class** Grid无法被实例化？

## 类存取器

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。必须设置TypeScript的编译参数“--target ES5”。

# 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。

|  |
| --- |
| // 基类 Animal **class** Animal {  **name**:**string**;   **constructor**( theName:**string** ){ **this**.**name** = theName; }   move( distanceInMeters:**number** = 0 )  {  ***console***.log( `${**this**.**name**} moved ${distanceInMeters}m.` );  } }  // Snake 类继承于 Animal **class** Snake **extends** Animal {  **constructor**( name:**string** ){ **super**( name ); }   move( distanceInMeters = 5 )  {  ***console***.log( "Slithering..." );  **super**.move( distanceInMeters );  } }  // Horse 类继承于 Animal **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()”。

### 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。

——————下午内容——————

# 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。

# 练习题

## 设计可以动物互动

动物类：“杂食、草食、肉食”三种动物

食物类：肉块、草堆

场地类：提供彼此竞争生存环境，最终只剩下一只动物

* 动物类要求具备“动物食性”属性，如“杂食、草食、肉食”
* 动物类需要不同的名字，即使是同食性同种类动物，用于输出吞食结果和最终存活结果
* 创建20个以上的动物实例
* 让20个动物“进入场地”后“互相残杀”，并输出不同的吞食结果  
  **规则为：**1、按顺序让当前动物去随机对其它目标（动物或食物）进行吞食2、肉食吃肉，草食吃草，杂食吃肉、草、及动物  
  3、每隔一段时间投放肉块x2、草堆x3，让动物吞食
* 吃一次草Exp+0.5、吃一次肉Exp+1、吃一次动物Exp+2，Exp满1则Level+1
* 等级影响肉食和杂食吃动物的结果，即所吃动物不能高于自身等级
* 输出最终存活的动物编号