TypeScript 接口

目标

1. 熟悉TypeScript中接口的理论和使用

大纲

# 接口

## 接口概述

TypeScript中接口的意义在于为值（函数、对象、数组、类实例等）定义“类型”。

TypeScript中，会对所有值进行类型检查，接口的作用就在于对这些类型进行命名，和为你的代码或者第三方代码（尤其是JavaScript库）定义类型契约。

在TypeScript中使用关键字“interface”来描述一个接口对象。

接口在TypeScript与JavaScript协同工作时，有着举足轻重的地位，是实现TypeScript与JavaScript进行互通的重要途径。

## 定义接口

TypeScript中以“interface”关键词来定义接口，接口名以 I开头，后接首字母大写的接口名。

在面对第三方JS程序的接口规划需求时，根据实际需要规划接口名。

举个简单栗子：

**function** *animal*( conf:{ **name**:**string** } )  
{  
 ***console***.log( conf.**name** );  
}  
  
**let *config*** = { **name**: "Lion", **sex**: "male" };  
*animal*( ***config*** );

类型检查器会查看animal的调用：

animal有一个参数，并要求这个对象参数有一个名为name类型为string的属性。

需要注意的是，我们传入的对象参数实际上可能会包含很多属性，但是编译器只会检查那些必需的属性是否存在，并且其类型是否匹配。

下面我们重写上面的例子，这次使用接口来描述，必须包含一个name属性且类型为string：

**interface IAnimal**{  
 **name**:**string**;  
}  
  
**function** *animal*( conf:**IAnimal** )  
{  
 ***console***.log( conf.**name** );  
}  
  
**let *config*** = { **name**: "Lion", **sex**: "male" };  
*animal*( ***config*** );

IAnimal接口就好比一个名字，用来描述一个数据对象。

它代表有一个name属性且类型为string的对象。

需要注意的是，TypeScript中接口使用并没有像其它语言那样严格的只能用在类的描述上，而是可以用来描述对象、数组、函数、类等多种类型。

TypeScript类型检查器不关心属性顺序，只关注值的外形，只要相应的属性存在并且类型也是对的就可以。

## 接口可选参数

接口中的属性并不都是必要的。在遵循一定的条件时，有些属性甚至可以不存在。

尤其是在对象作为传给函数的参数时，往往只包含部分属性在里面。

接口可选参数的实现方式与函数的可选参数相同，使用类型前加“？”。

**interface SquareConfig**{  
 **color**?: **string**;  
 **width**?: **number**;  
}

## 只读属性

一些对象属性只能在对象刚刚创建的时候修改其值。

你可以在属性名前用 readonly来指定只读属性:

**interface Point**{  
 **readonly x**:**number**;  
 **readonly y**:**number**;  
}

你可以通过赋值一个对象字面量来构造一个Point。 赋值后， x和y再也不能被改变了。

**let *p1***:**Point** = { **x**: 10, **y**: 20 };  
***p1***.**x** = 5; // error!

# 函数接口

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。

为了使用接口表示函数类型，我们需要给接口定义一个调用签名。

它就像是一个只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。

**interface ICreateAnimal**{  
 ( type:**string**, name:**string** ): **boolean**;  
}

对于函数类型的类型检查来说，函数的参数名不需要与接口里定义的名字相匹配。

函数的参数会逐个进行检查，要求对应位置上的参数类型是兼容的。

**let** *ca*:**ICreateAnimal** = **function**( type:**string**, name:**string** ):**boolean**{  
 ***console***.log( `创建一只名为 ${name} 的 ${type}` );  
 **return true**;  
};

# 数组接口

与使用接口描述函数类型差不多，我们也可以描述数组类型。

数组类型具有一个index类型表示索引的类型，还有一个相应的返回值类型表示通过索引得到的元素的类型。

**interface StringArray**{  
 // 表示用数值索引的字符串元素集合数组  
 [**index**:**number**]:**string**;  
}  
  
**let *myArray***:**StringArray**;  
***myArray*** = [ "Bob", "Fred" ];

数组接口的定义中，支持两种索引类型，即number和string。

——————下午内容——————

# 类接口

## 定义类接口

在TypeScript中，可以使用接口来确保类拥有指定的结构，以确保在多个类进行相互工作时拥有必须的属性或方法等特征，从而保证程序的正确运转。

而JavaScript中无法做到这些，因而给程序带来太多可能造成问题的“不确定因素”。

## 实现类接口

和C#、Java、AS3中使用方式相同，在TypeScript中我们使用implements关键字来让类实现某个接口，以让这个类符合接口指定的约定。

**interface IAnimal**{  
 **name**:**string**;  
 makeSound():**void**;  
}  
  
**class** Animal **implements IAnimal**{  
 **name**:**string**;  
  
 makeSound():**void** {  
 ***console***.log( "yoho..." );  
 }  
}

注意：接口只能描述类的公共部分，而不关注私有部分。这种机制不允许我们通过接口来检查一个类的实例的私有部分。

## 类的继承和类的接口实现的比较

|  |  |
| --- | --- |
| 类继承（extends） | 类实现接口（implements） |
| 类只能继承单独一个类  class B extends A | 类可以同时实现多个接口  多个接口名用“,”隔开  **class C** **implements IX**, **IY** |
| 只能具备和共享父类的类型特征 | 使多个毫无关系的类具备相同的类型特征 |
| 基础骨架结构用继承 | 扩展增强用接口进行补丁 |

# 扩展合并接口

和类一样，接口也可以相互扩展。

这让我们能够从一个接口里复制成员到另一个接口里，可以更灵活地将接口分割到可重用的模块里。

## 通过接口继承extends扩展接口

**interface Shape**{  
 **color**:**string**;  
}  
  
**interface Square extends Shape**{  
 **sideLength**:**number**;  
}  
  
**let *square*** = <**Square**>{};  
***square***.**color** = "blue";  
***square***.**sideLength** = 10;

接口可以扩展多个接口，从而成为多个接口的组合。

**interface Shape**{  
 **color**:**string**;  
}  
  
**interface PenStroke**{  
 **penWidth**:**number**;  
}  
  
**interface Square extends Shape**, **PenStroke**{  
 **sideLength**:**number**;  
}  
  
**let *square*** = <**Square**>{};  
***square***.**color** = "blue";  
***square***.**sideLength** = 10;  
***square***.**penWidth** = 5.0;

警告：继承多个接口时，如果继承的接口中，存在同名属性且类型不同的话，不能编译通过。

## 接口合并扩展接口

TypeScript中允许存在相同名称的接口，同名的成员保留一份，不同名的部分则进行增加扩展，以使接口功能的支持得到加强。

**interface IPosition**{  
 **x**:**number**;  
 **y**:**number**;  
}  
  
**interface IPosition**{  
 **z**:**number**;  
}

# 定义环境

通常在使用第三方库的环境里，若需要在TypeScript中正确引用库中的成员，则需要对这些成员进行定义，描述其“使用协议”。

定义环境数据，使用“declare”关键字进行定义。

## 全局变量和常量

常用于定义JavaScript环境下的全局变量，从而在TypeScript中能够正确使用，但必须确保其真实存在。

**declare let *foo***:**number**;  
***console***.log( "Half the number of widgets is " + (***foo*** / 2) );

定义变量一般使用“declare let”或者“declare const”进行。

## 全局函数

TypeScript中使用“declare function”来定义函数声明。

**declare function** *greet*( greeting:**string** ):**void**;  
*greet*( "hello, world" );

## 声明接口

TypeScript中使用“declare interface”来定义接口声明，语法与定义接口无异，只是多加declare关键字。

**declare interface ILib**{  
 hello():**void  
 length**:**number**;  
}

## 类

TypeScript中使用“declare class”描述一个类或像类一样的对象， 类可以有属性和方法。

**declare class** Greeter  
{  
 **constructor**( greeting:**string** );  
  
 **greeting**:**string**;  
 showGreeting():**void**;  
}

## 带属性的对象

代码如下：

**let *count***:**number** = ***myLib***.**numberOfGreetings**;  
**let *result***:**string** = ***myLib***.*makeGreeting*( "hello, world" );  
***console***.log( "The computed greeting is:" + ***result*** );

将myLib作为Object对象进行声明：

**declare let *myLib***:{  
 **numberOfGreetings**:**number**;  
 makeGreeting( value:**string** ):**string**;  
};

将myLib作为Class实例进行声明：

**declare let *myLib***:**ILib**;  
**declare interface ILib**{  
 **numberOfGreetings**:**number**;  
 makeGreeting( value:**string** ):**string**;  
}

将myLib作为命名空间（库作用域）进行声明：

**declare namespace myLib**{  
 **function** *makeGreeting*( s:**string** ):**string**;  
 **let** *numberOfGreetings*:**number**;  
}

## 同名函数多种使用方法

在TypeScript中，存在多个同名函数的情况，我们称之为“函数重载”，即描述一个函数能以不同的形态或参数列表进行使用的情况。

**let *width*** = *css*( "width" );  
*css*( { **width**: 100 } );  
  
**declare function** *css*( name:**string** ):**any**;  
**declare function** *css*( properties:{} ):**void**;

## 声明文件

通常描述一个完整的第三方库的声明时，会将所有声明独立到一个文件中统一起来，其作用在于，将声明与逻辑代码进行分离，引用声明文件时即可正常使用相关的内容。

声明文件以“\*.d.ts”结尾。

引用声明文件时，只需要使用以下语法：

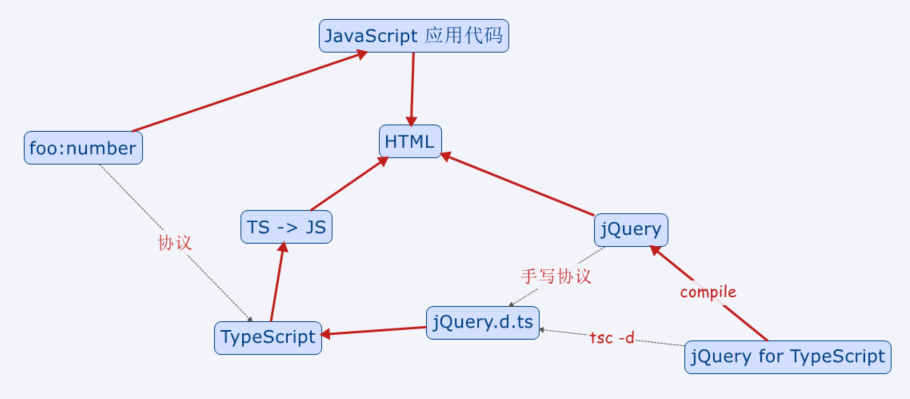
///<reference path="IAnimal.d.ts"/>

对于JavaScript所开发的第三方库，不可避免的必须手写声明文件。

而针对“\*.ts”文件，可以快速的自动生成声明文件，通过命令行进行使用：

tsc \*.ts -d

若编译“声明文件”时，指定的“\*.ts”依赖于其它文件，则所有相关的“\*.ts”文件都会被自动编译出“声明文件”。



# 练习题

## 设计一个动物接口IAnimal

有个name:string 属性

有个eat(target:IAnimal):boolean;方法

以及一个eatType:number属性代表自身的食性

另一个 eatTo:number 属性代表觅食倾向，在eat到不同类型的动物时要有不同的输出结果。

eat到不想吃的动物时输出趣味性的结果，并返回 false。

设计3种以上动物类，实现IAnimal接口。

要求 eatType 使用枚举来进行全局统一规划。

**enum** EatType{*Meat*, *Grass*}  
***lion***.eatType = EatType.*Meat*;