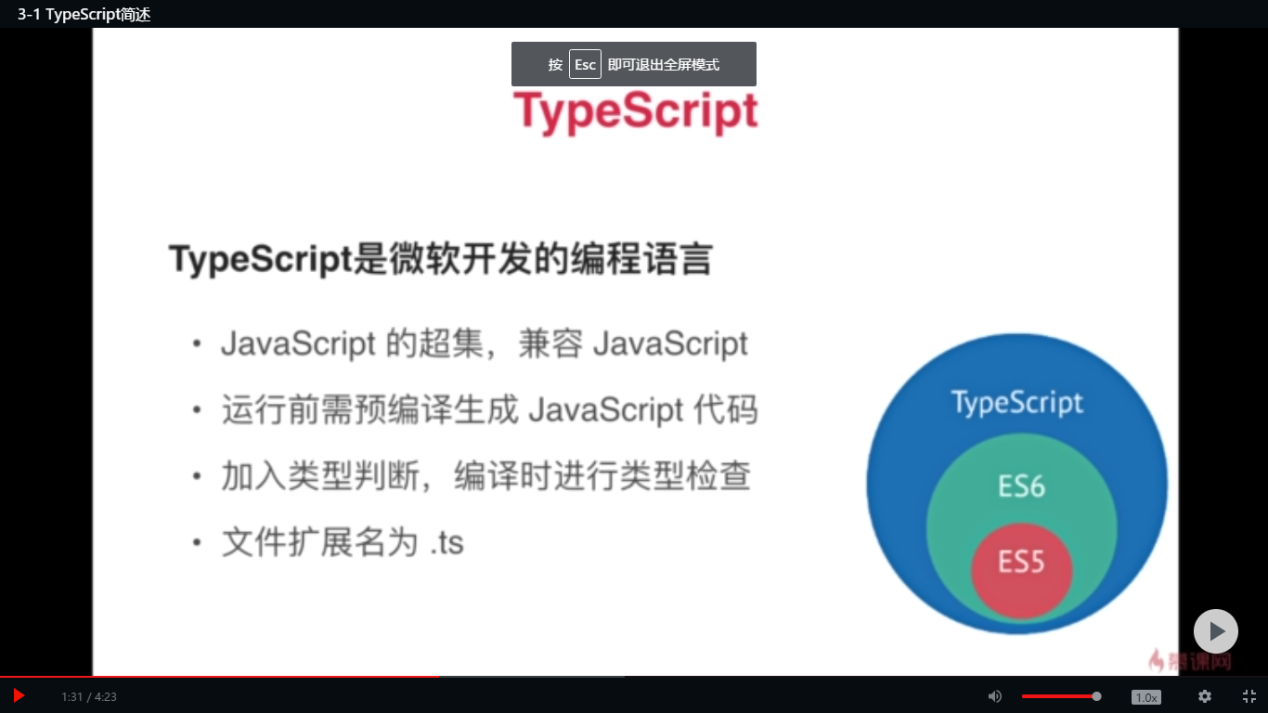
1. npm install -g typescript
2. tsc -v
3. 在.ts文件所在的文件夹下打开命令行，运行tsc hello-ionic.ts，即可把.ts文件编译成js文件。



## 类型

每个变量、参数、函数后面都用冒号标明类型，简单地说TypeScript = type + JaveScript

var book: string = “Angular 2”;

var num: number = 123;

var man: boolean = true;

function log(msg: string): void {

console.log(msg);

}

function test(): string { //返回值的类型为string

return “”;

}

**:any**赋值的话任何类型均可。

**:void**声明该函数不需要任何返回值。

还可以用class / 接口自定义类型：

class Person {

name: string;

age: number;

}

var abc: Person = new Person; //abc就是Person类型

可选参数：在参数声明后面用问号来标明此参数为可选参数，必须声明在必选参数后面。

默认参数：在参数声明后面用等号来指定参数的默认值，带默认值的参数要声明在最后一个。

function test(a: string, b?: string, c: string = “Joey”){

console.log(c);

}

test(“bp”, “bd”, “qp”);

test(“bp”, “bd”); //有默认参数了就可以不传

test(“bp”);

## 字符串 ` ` 的用法

1. 多行字符串：把字符串放在 **``** 里，换行也不会报错
2. 字符串模板：在多行字符串里用一个表达式去插入变量或者调用方法。

var username = “Flora”;

var getName = function(){

return “I’am Flora”;

}

console.log(**`**hello, **${**username**}`**);

console.log(`hello, ${getName}`);

1. 自动拆分字符串

function test(template, name, age){

console.log(template); //[“hello, my name is”, “, I’m”, “”]

console.log(name); //Lee

console.log(age); //18

}

var myname = “Lee”;

var getAge = function(){

return 18;

}

test`hello, my name is ${myname}, I’m ${getAge()}`

## 函数

1. Rest and Spread操作符：用来声明任意数量的参数。

function foo(...args) {

args.forEach(function(arg){

console.log(arg);

})

}

以下用法属于ES6，TypeScript还不支持。

function foo(a, b, c){

console.log(a);

console.log(b);

console.log(c);

}

var arr1 = [1,2];

var arr2 = [3,4,5,6,7];

foo(...arr1); //打印出1，2，undefined

foo(...arr2); //打印出3，4，5

1. generator 函数：控制函数的执行过程，手动暂停和恢复代码执行，属于ES6特性，TypeScript还不支持。

function\* do() { //用**\***关键字来声明generator 函数

console.log(“start”):

yield;

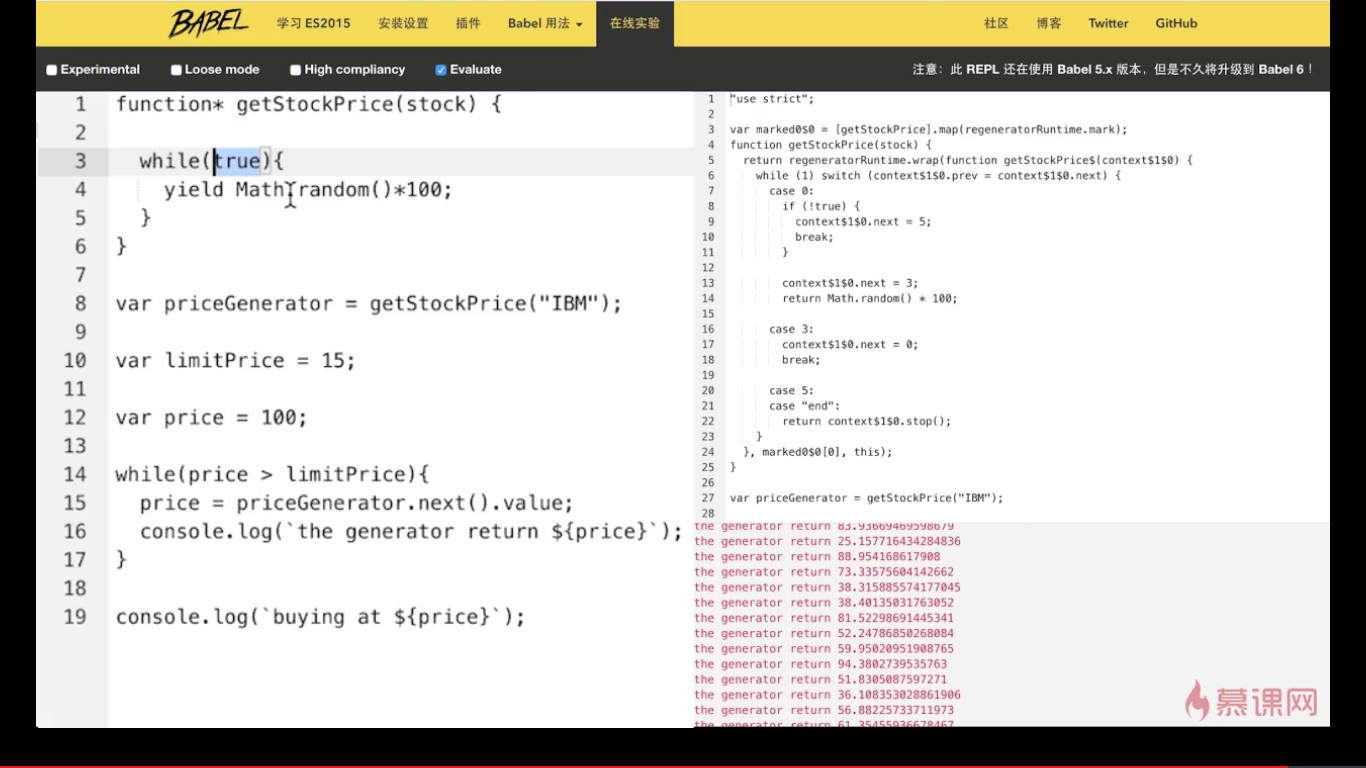
console.log(“finish”);

}

var fn = do();

fn.next();

fn.next(); //每调一次next，就会在yield停一次



1. 解构表达式：通过表达式将对象或数组拆解成任意数量的变量。

function getStock() {  
 return {  
 code: 'IBM',  
 price:100  
 }  
 }  
 //var stock = getStock();  
 //var code = stock.code;  
 //var price = stock.price; 以上三句等价于下面一句：

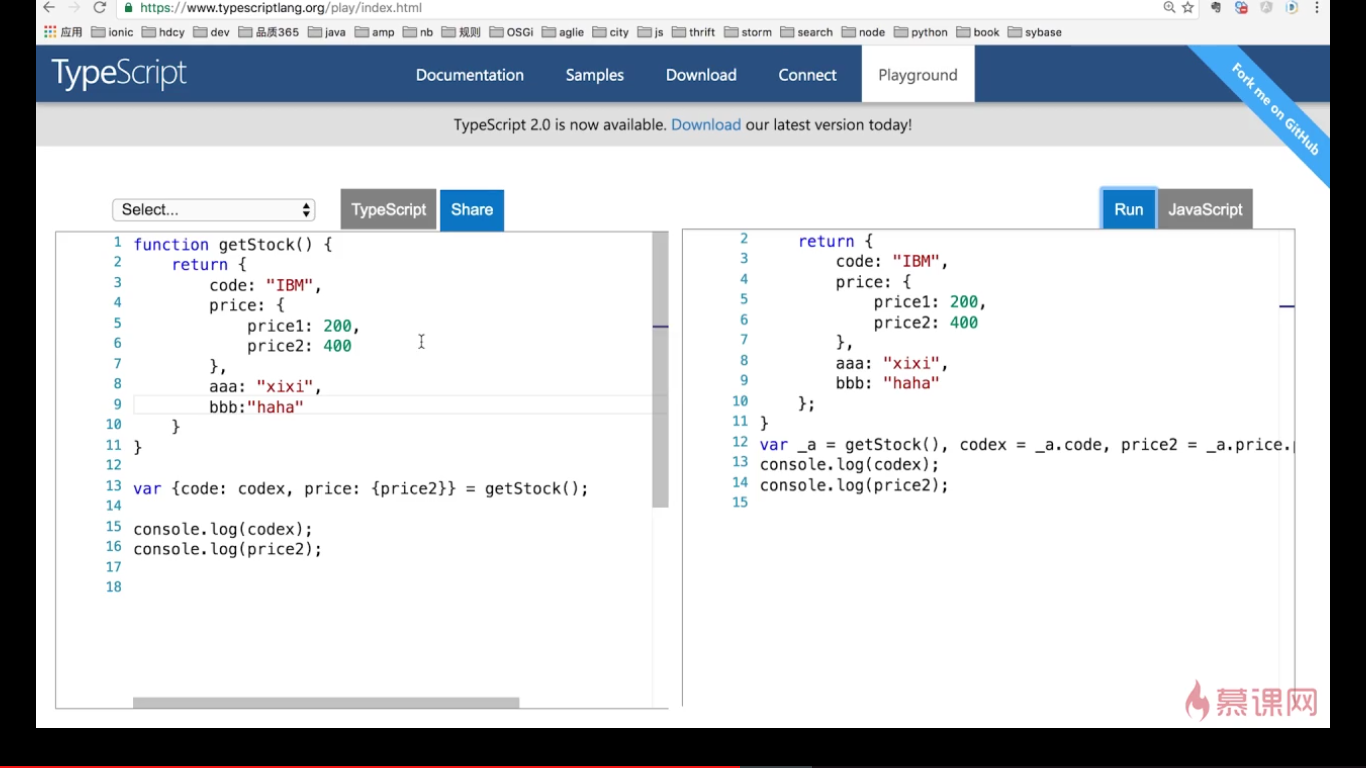
var {code, price} = getStock();

//或者var {code:company, price} = getStock();相当于把对象里的属性放到一个本地变量里面

console.log(code);

console.log(price);

解构表达式的嵌套



针对数组的解析表达式：

var arr = [1,2,3,4,5,6];

var [num1,num2] = arr;

var [, , num1,num2] = arr;

var [num1, , num2] = arr;

var [num1,num2, ...others] = arr;

var arr = [33,44,55,77,88,22];

function fn([num1, num2, ...others]){

console.log(num1); //33

console.log(others); //[55,77,88,22]

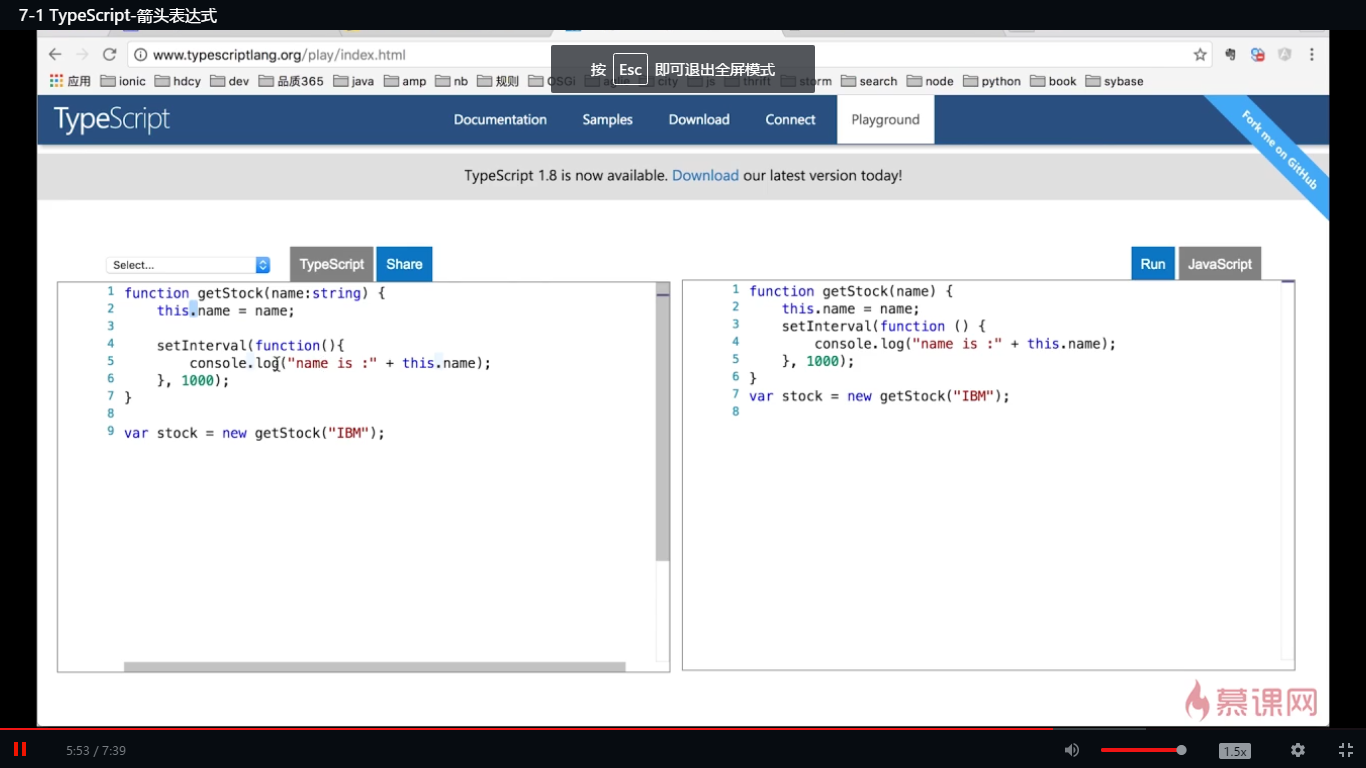
}

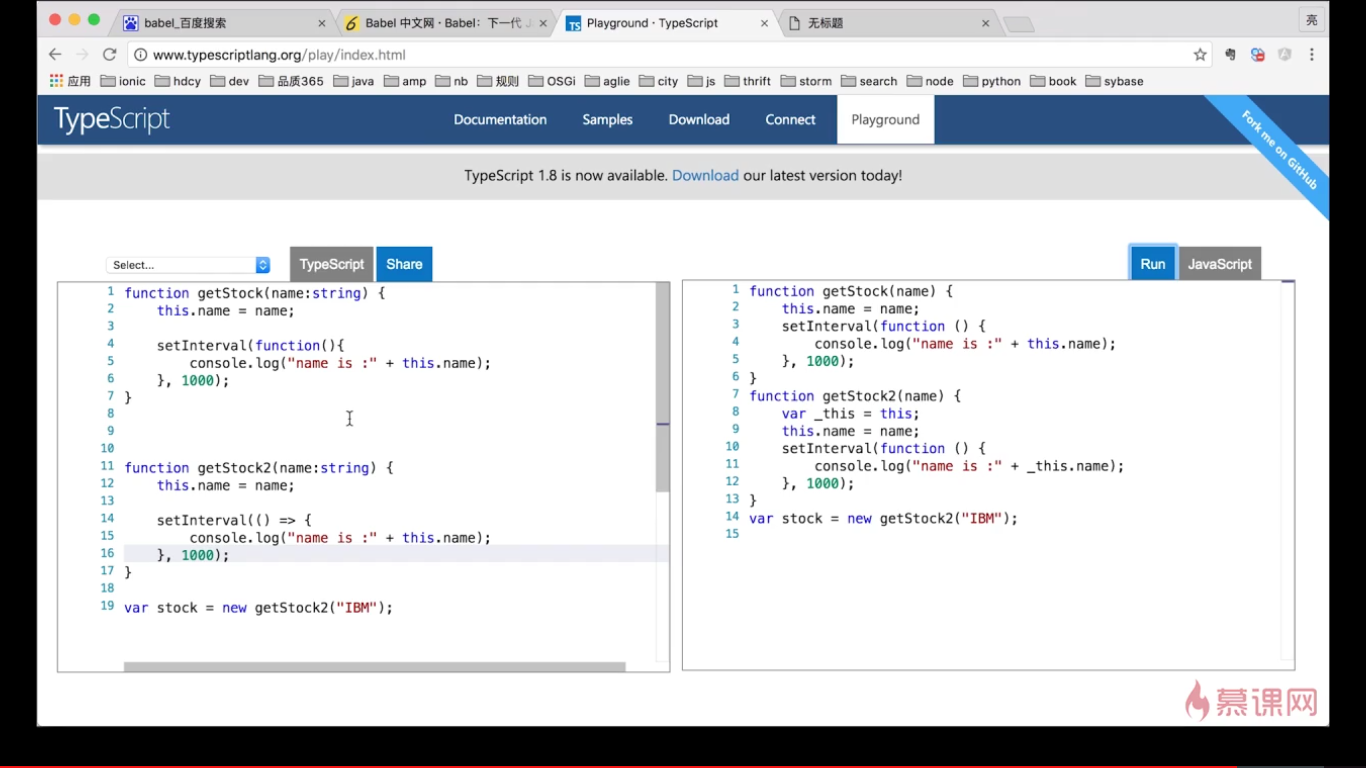
fn(arr);

1. 箭头表达式：用来声明匿名函数，消除传统匿名函数this指针的问题。

var myArr = [33,44,55,77,88];

console.log(myArr.filter(value => value%2 ==0)); //打印44，88





1. 循环：forEach for-in for-of

forEach不允许打断循环，不支持break关键字。every方法可以打断。

for-in用来循环键或属性名。

var myArr = [3,5,6,9];

myArr.desc = “four number”;

for(var n in myArr){

console.log(n); //打印0,1,2,3,desc

console.log(myArr[n]); //打印3,5,6,9

}

for-of用来循环值。

var myArr = [11,44,55,23];

myArr.desc = “four number”;

for(var n of myArr) {

if(n>2) break;

console.log(n);

}

for-of也可以用来循环字符串。

for(var n of “four number”){

console.log(n);

}

## 类和接口

1. 类的声明

class Person {

name; //属性

eat() { //方法

console.log(“I am eating”);

}

}

var p = new Person(); //实例化

访问控制符有三个：

public 公有，类内外都可以用，如果不声明，默认就是public

private 私有，只能在类的内部用；

protected 受保护的， 只能在类内部与子类中使用。

class里加上constructor方法就成了构造函数。constructor方法在类被实例化时会被调用，而且只调用一次。

class Person {

constructor(public name: string) { //等价于this.name = name

} //此处的public写与不写是不同的，不写的话就未声明name属性

eat() { //方法

console.log(“I am eating”);

}

}

1. 类的继承extends

子类拥有父类所有的属性和方法

class Employee **extends** Person{

code: string;

work(){

console.log(“I am working”);

}

}

var e = new Employee(“Lee”);

e.eat();

super有两个用法：

1. 调父类的构造函数，因为子类的构造函数必须要调用父类的构造函数。
2. 调父类的其他的方法

class Employee extends Person{

constructor(name: string, code: string){

**super**(name);

this.code = code;

}

code: string;

work(){

**super**.eat();

this.sell();

}

sell(){

console.log(“I am selling”);

}

}

var e = new Employee(“Lee”, “346”);

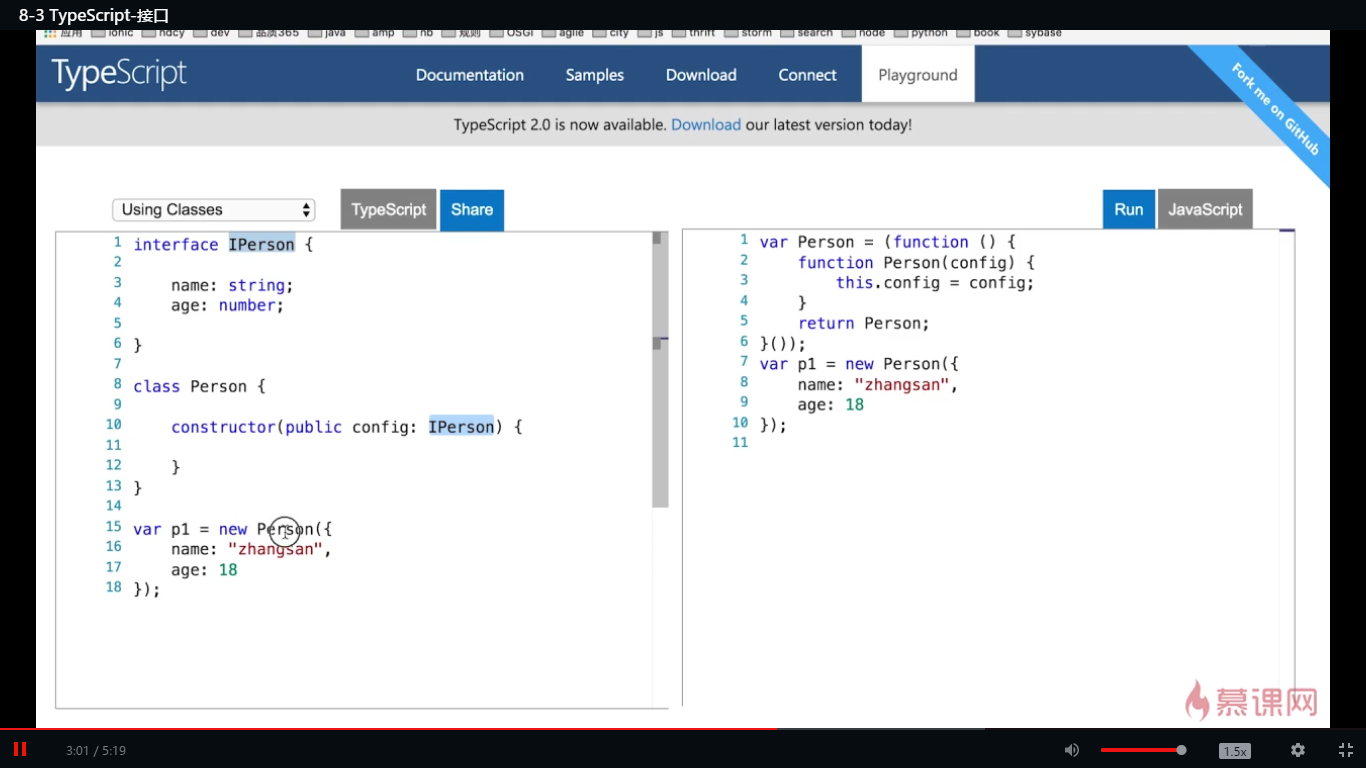
e.work();

e.sell()也可以，如果不希望员工只干活不吃饭，可以在sell前加private，把该方法变成私有方法，外部就不能再调用了。

## 接口 interface

用来建立某种代码约定，使得其开发者在调用某个方法或创建新的类时必须遵循接口所定义的代码约定。

1. 接口用作一个方法的参数的类型声明，当调用该方法时，TypeScript会去检查你传入的参数是否满足接口所声明的所有属性。



1. 接口声明方法

**Interface Shape** {

area(): number;

}

**class** Rectangle **implements Shape** { //实现接口

constructor(

private width: number,

private length: number

){}

area() { //实现接口的时候必须实现接口里的方法

return this.width\* this.length;

}

}

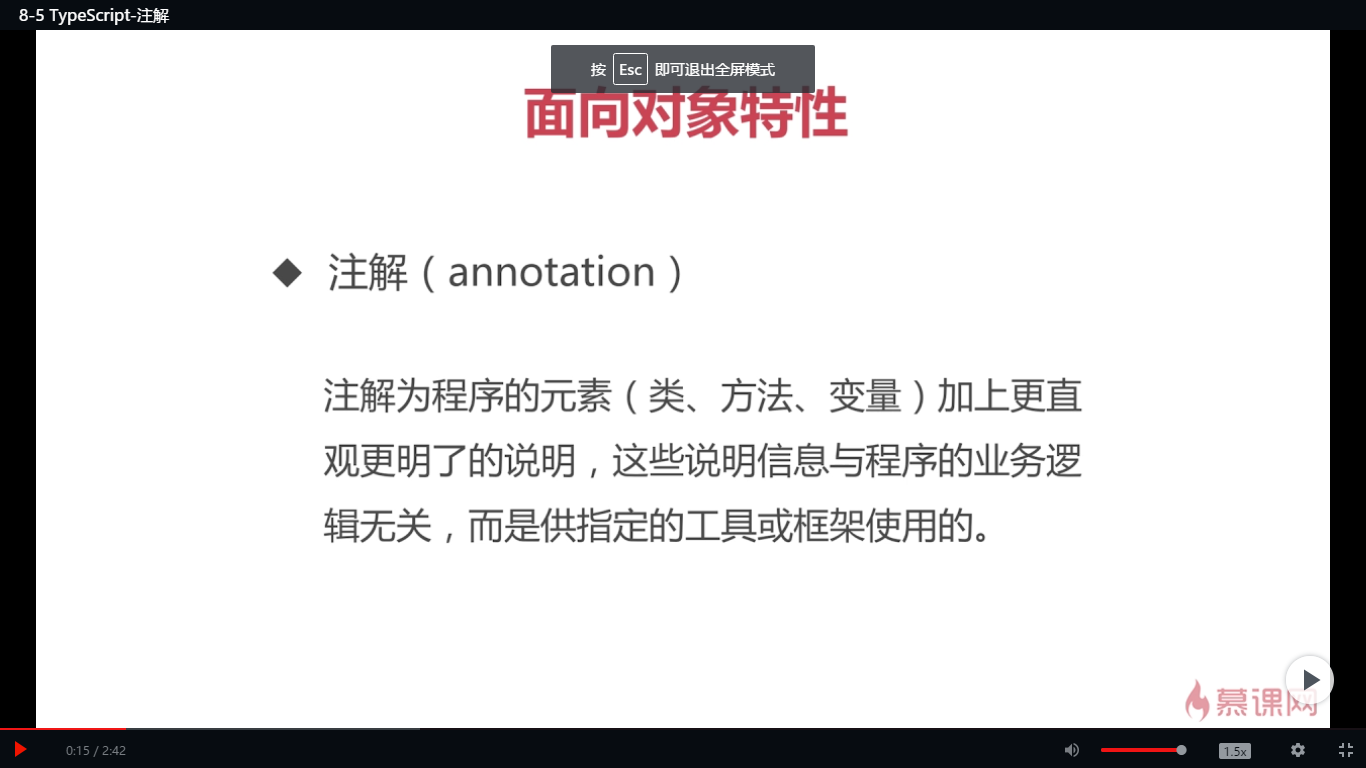
## 模块

模块可帮助开发者将代码分割为可重用的单元。开发者可自行决定将模块中的哪些资源（类、方法、变量）暴露出去供外部使用，哪些资源只在模块内使用。

在TypeScript里，一个文件就是一个模块。在模块内部有两个关键字来支撑模块的特性：import 和export。import 用来引入别的模块中的资源，export 用于暴露资源供别的模块使用。

**import {**func**}** **from** “./a”;

./a其实是a.ts文件，后缀名不用引用。



## 装饰器

一种特殊类型的声明，能够被附加到类、方法、访问符、属性或参数上

@SomeDecorator ({ //装饰器用来修饰一个类

//配置

})

class SomeClass {}

TypeScript并不会帮我们实现装饰器的逻辑，需要自己定义。SomeDecorator本质上是一个函数，它返回了另外一个匿名函数。该匿名函数接收的参数指向了被修饰类的引用。在匿名函数的函数体内可以对类添加一些信息。

function SomeDecorator(config: any){

return function(cls: any){ //cls为类的构造函数

cls.isSealed = true; //这个静态属性用来拘束类能否派生出子类

return cls;

}

}

Angular2所使用的@Component正是通过装饰器机制实现的，具体的代码实现放在了angular/core核心代码里。

@Component({

selector: "hello",

template: "<p>{{greeting}}</p>"

})

export class HelloComponent {

private greeting: string;

constructor() {

this.greeting = "Hello, Angular 2!";

}

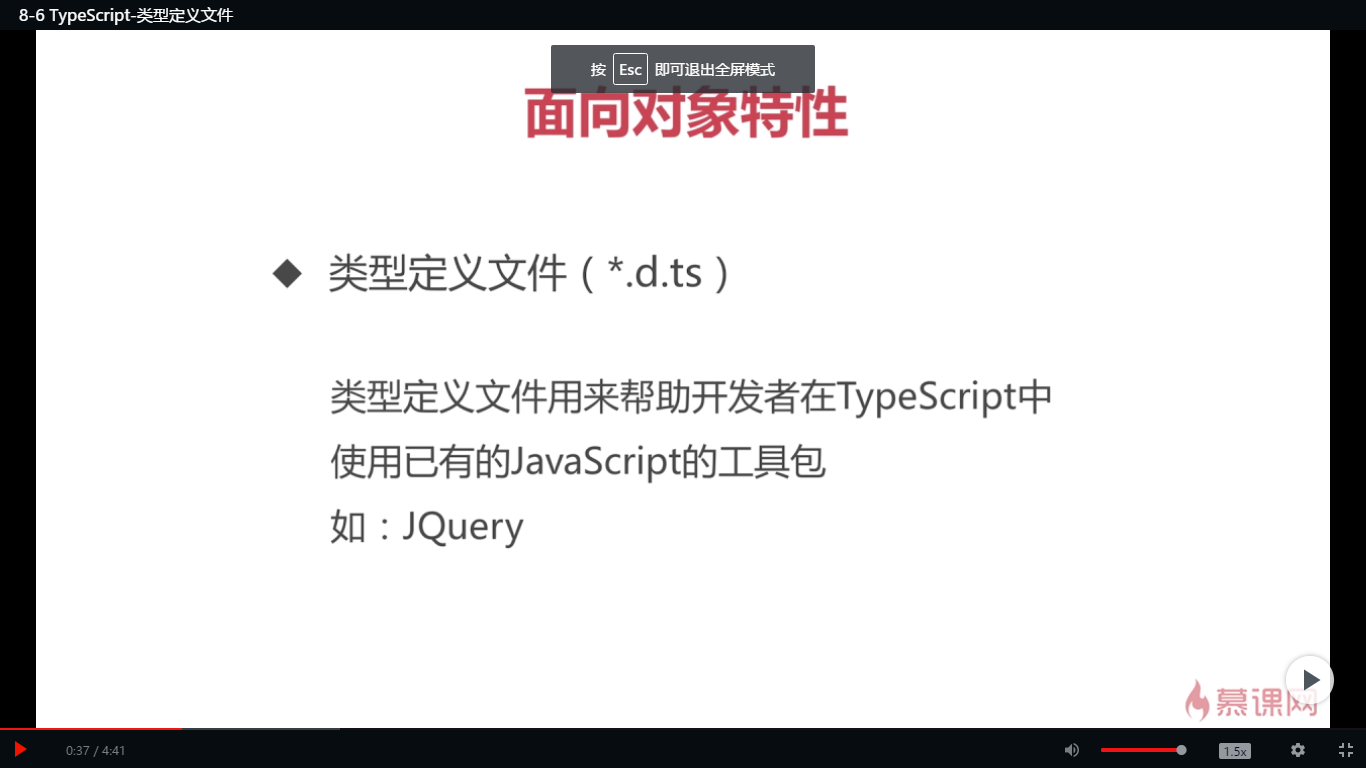
}

## 泛型 generic

参数化的类型，一般用来限制集合的内容。

var workers: Array**<**Person**>** = [];

**< >**规定了数组里面只能放Person。



Typings工具是专门用来安装类型定义文件的，可以在github上下载。