1、什么是TypeScript？ TypeScript是JavaScript的加强版，它给JavaScript添加了可选的静态类型和基于类的面向对象编程，它拓展了JavaScript的语法。

而且TypeScript不存在跟浏览器不兼容的问题，因为在编译时，它产生的都是JavaScript代码。

2、TypeScript 和 JavaScript 的区别是什么？ Typescript 是 JavaScript 的超集，可以被编译成 JavaScript 代码。用 JavaScript 编写的合法代码，在 TypeScript 中依然有效。Typescript 是纯面向对象的编程语言，包含类和接口的概念。 程序员可以用它来编写面向对象的服务端或客户端程序，并将它们编译成 JavaScript 代码。

TypeScript和 JavaScript的关系 TypeScript 引入了很多面向对象程序设计的特征，包括：

interfaces 接口 classes 类 enumerated types 枚举类型 generics 泛型 modules 模块 主要不同点如下： TS 是一种面向对象编程语言，而 JS 是一种脚本语言（尽管 JS 是基于对象的）。 TS 支持可选参数， JS 则不支持该特性。 TS 支持静态类型，JS 不支持。 TS 支持接口，JS 不支持接口。

3为什么要用 TypeScript ？ TS 在开发时就能给出编译错误， 而 JS 错误则需要在运行时才能暴露。 作为强类型语言，你可以明确知道数据的类型。代码可读性极强，几乎每个人都能理解。 TS 非常流行，被很多业界大佬使用。像 Asana、Circle CI 和 Slack 这些公司都在用 TS。

4、TypeScript 和 JavaScript 哪个更好？ 由于 TS 的先天优势，TS 越来越受欢迎。但是TS 最终不可能取代 JS，因为 JS 是 TS 的核心。

选择 TypeScript 还是 JavaScript 要由开发者自己去做决定。如果你喜欢类型安全的语言，那么推荐你选择 TS。如果你已经用 JS 好久了，你可以选择走出舒适区学习 TS，也可以选择坚持自己的强项，继续使用 JS。

5、什么是泛型？ 泛型是指在定义函数、接口或类的时候，不预先指定具体的类型，使用时再去指定类型的一种特性。 可以把泛型理解为代表类型的参数 // 我们希望传入的值是什么类型，返回的值就是什么类型 // 传入的值可以是任意的类型，这时候就可以用到 泛型

// 如果使用 any 的话，就失去了类型检查的意义

function createArray1(length: any, value: any): Array { let result: any = []; for (let i = 0; i < length; i++) { result[i] = value; } return result; }

let result = createArray1(3, 'x'); console.log(result);

// 最傻的写法：每种类型都得定义一种函数 function createArray2(length: number, value: string): Array { let result: Array = []; for (let i = 0; i < length; i++) { result[i] = value; } return result; }

function createArray3(length: number, value: number): Array { let result: Array = []; for (let i = 0; i < length; i++) { result[i] = value; } return result; } // 或者使用函数重载，写法有点麻烦 function createArray4(length: number, value: number): Array function createArray4(length: number, value: string): Array function createArray4(length: number, value: any): Array { let result: Array = []; for (let i = 0; i < length; i++) { result[i] = value; } return result; } createArray4(6, '666'); //使用泛型 // 有关联的地方都改成 function createArray(length: number, value: T): Array { let result: T[] = []; for (let i = 0; i < length; i++) { result[i] = value; } return result; } // 使用的时候再指定类型 let result = createArray(3, 'x'); // 也可以不指定类型，TS 会自动类型推导 let result2 = createArray(3, 'x'); console.log(result); 6、TS中的类 TypeScript 是面向对象的 JavaScript。而其中的类描述了所创建的对象共同的属性和方法。

传统的JavaScript程序使用函数和基于原型的继承来创建可重用的组件，但这对于熟悉使用面向对象方式的程序员来说有些棘手，因为他们用的是基于类的继承并且对象是从类构建出来的。

从ECMAScript 2015，也就是ECMAScript 6，JavaScript程序将可以使用这种基于类的面向对象方法。在TypeScript里允许开发者现在就使用这些特性，并且编译后的JavaScript可以在所有主流浏览器和平台上运行，

7、什么是构造函数，构造函数作用是什么？ 构造函数 ，是一种特殊的方法。主要用来在创建对象时初始化对象， 即为对象成员变量赋初始值，总与new运算符一起使用在创建对象的语句中。而TypeScript的构造函数用关键字constructor来实现。可以通过this（和java/C#一样代表对象实例的成员访问）关键字来访问当前类体中的属性和方法。

8、实例化是什么？ 一般情况下，创建一个类后并不能直接的对属性和方法进行引用，必须对类进行实例化，即创建一个对象。TypeScript中用new 关键字创建对象。实例化后通过“.”来访问属性和方法

9、方法重写是什么？ 子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写

重写的作用在于子类可以根据需要，定义特定于自己的行为。也就是说子类能够根据需要实现父类的方法。

10、什么是可索引类型接口？ 一般用来约束数组和对象

/ 数字索引——约束数组 // index 是随便取的名字，可以任意取名 // 只要 index 的类型是 number，那么值的类型必须是 string interface StringArray { // key 的类型为 number ， 一般都代表是数组 // 限制 value 的类型为 string [index:number]:string } let arr:StringArray = ['aaa','bbb']; console.log(arr);

// 字符串索引——约束对象 // 只要 index 的类型是 string，那么值的类型必须是 string interface StringObject { // key 的类型为 string ， 一般都代表是对象 // 限制 value 的类型为 string [index:string]:string } let obj:StringObject = {name:'ccc'}; 11、什么是函数类型接口？ 对方法传入的参数和返回值进行约束 // 注意区别

// 普通的接口 interface discount1{ getNum : (price:number) => number }

// 函数类型接口 interface discount2{ // 注意: // ":" 前面的是函数的签名，用来约束函数的参数 // ":" 后面的用来约束函数的返回值 (price:number):number } let cost:discount2 = function(price:number):number{ return price * .8; }

// 也可以使用类型别名 type Add = (x: number, y: number) => number let add: Add = (a: number, b: number) => a + b 12、什么是类类型接口？ 如果接口用于一个类的话，那么接口会表示“行为的抽象” 对类的约束，让类去实现接口，类可以实现多个接口 接口只能约束类的公有成员（实例属性/方法），无法约束私有成员、构造函数、静态属性/方法

// 接口可以在面向对象编程中表示为行为的抽象 interface Speakable { name: string;

```
 // ":" 前面的是函数签名，用来约束函数的参数
// ":" 后面的用来约束函数的返回值
speak(words: string): void
```

}

interface Speakable2 { age: number; }

class Dog implements Speakable, Speakable2 { name!: string; age = 18;

```
  speak(words: string) {
      console.log(words);
  }
```

}

let dog = new Dog(); dog.speak('汪汪汪'); 13、什么是混合类型接口？ 一个对象可以同时做为函数和对象使用

interface FnType { (getName:string):string; }

interface MixedType extends FnType{ name:string; age:number; } interface Counter { (start: number): string; interval: number; reset(): void; }

function getCounter(): Counter { let counter = function (start: number) { }; counter.interval = 123; counter.reset = function () { }; return counter; }

let c = getCounter(); c(10); c.reset(); c.interval = 5.0; 14、never 和 void 的区别？ void 表示没有任何类型（可以被赋值为 null 和 undefined）。 never 表示一个不包含值的类型，即表示永远不存在的值。 拥有 void 返回值类型的函数能正常运行。拥有 never 返回值类型的函数无法正常返回，无法终止，或会抛出异常。

15、TS的学前基础？ 因为 TypeScript 是对 JavaScript 的扩展，更准确的说是 ECMAScript。所以，我们学习我们这套 TypeScript 的课程，需要具备 ECMAScript 语言的基础：

熟悉语法基础（变量、语句、函数等基础概念） 掌握内置对象（Array、Date 等）的使用 面向对象基本概念（构造函数、原型、继承）