

04.11 【讲义】Typescript

#2021#

这节课的重点是实战, 对于ts的编译原理部分, 同学们了解一下就可以, 面试也不会问的特别细, 知道大概的流程就行。

之前没接触过Ts等强类型语言的同学: 这节课会有很多概念, 可能很难直接记住, 那么这些同学听这节课的时候, 目的就不是去死记硬背这些概念, 而且去体会ts的写法, 能不能优化自己在平时js编写时碰到的一些问题。

接触过ts的同学: 注意一下这节课里的一些泛型 以及各种实战代码与ts的结合。

基础知识

基础类型: number string boolean array object

1. enum: 枚举
2. type, interface
3. 联合类型 | (联合类型一次只能一种类型; 而交叉类型每次都是多个类型的合并类型。)
4. 交叉类型 & (联合类型一次只能一种类型; 而交叉类型每次都是多个类型的合并类型。)
5. typeof

typeof 操作符可以用来获取一个变量声明或对象的类型。

```
function toArray(x: number): Array<number> {  
  return [x];  
}  
  
type Func = typeof toArray; // -> (x: number) => number[]
```

6. keyof

Keyof 操作符可以用来一个对象中的所有 key 值：

```
interface Person {  
    name: string;  
    age: number;  
}  
  
type K1 = keyof Person; // "name" | "age"
```

7. in

In 用来遍历枚举类型：

```
type Keys = "a" | "b" | "c"  
  
type Obj = {  
    [p in Keys]: any  
} // -> { a: any, b: any, c: any }
```

8. extends

有时候我们定义的泛型不想过于灵活或者说想继承某些类等，可以通过 extends 关键字添加泛型约束。

```
interface ILengthwise {  
    length: number;  
}  
  
function loggingIdentity<T extends ILengthwise>(arg: T): T {  
    console.log(arg.length);  
}
```

```
    return arg;
}

loggingIdentity(3);
loggingIdentity({length: 10, value: 3});
```

9. Partial

Partial<T> 的作用就是将某个类型里的属性全部变为可选项？。

10. Required

Required<T> 的作用就是将某个类型里的属性全部变为必选项。

11. Readonly

Readonly<T> 的作用是将某个类型所有属性变为只读属性，也就意味着这些属性不能被重新赋值。

12. Record

Record<K extends keyof any, T> 的作用是将 K 中所有的属性的值转化为 T 类型。

```
interface PageInfo {
    title: string;
}

type Page = "home" | "about" | "contact";

const x: Record<Page, PageInfo> = {
    about: { title: "about" },
    contact: { title: "contact" },
    home: { title: "home" }
};
```

13. Exclude

Exclude<T, U> 的作用是将某个类型中属于另一个的类型移除掉。

```
type T0 = Exclude<"a" | "b" | "c", "a">; // "b" | "c"  
type T1 = Exclude<"a" | "b" | "c", "a" | "b">; // "c"
```

14. Extract

Extract<T, U> 的作用是从 T 中提取出 U。

```
type T0 = Extract<"a" | "b" | "c", "a" | "f">; // "a"  
type T1 = Extract<string | number | (() => void), Function>; // () => void
```

面试题及实战

1. 你觉得使用ts的好处是什么？

1.1 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。所以ts的功能比js只多不少。

1.2 Typescript 是纯面向对象的编程语言，包含类和接口的概念。

1.3 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。

1.4 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。

1.5 ts中有很多很方便的特性，比如可选链。

2. type 和 interface的异同

重点：用interface描述数据结构，用type描述类型

2.1 都可以描述一个对象或者函数

```
interface User {  
  name: string
```

```
    age: number
}

interface SetUser {
    (name: string, age: number): void;
}

type User = {
    name: string
    age: number
};

type SetUser = (name: string, age: number) => void;
```

2.2 都允许拓展 (extends)

interface 和 type 都可以拓展，并且两者并不是相互独立的，也就是说 interface 可以 extends type, type 也可以 extends interface。虽然效果差不多，但是两者语法不同。

```
// interface extends interface
interface Name {
    name: string;
}

interface User extends Name {
    age: number;
}

// type extends type
type Name = {
    name: string;
}

type User = Name & { age: number };

// interface extends type
type Name = {
    name: string;
}

interface User extends Name {
```

```
    age: number;
}

// type extends interface
interface Name {
    name: string;
}
type User = Name & {
    age: number;
}
```

2.3 只有type可以做的

Type 可以声明基本类型别名，联合类型，元组等类型

```
// 基本类型别名
type Name = string

// 联合类型
interface Dog {
    wong();
}
interface Cat {
    miao();
}

type Pet = Dog | Cat

// 具体定义数组每个位置的类型
type PetList = [Dog, Pet]

// 当你想获取一个变量的类型时，使用 typeof
let div = document.createElement('div');
type B = typeof div
```

3. 如何基于一个已有类型, 扩展出一个大部分内容相似, 但是有部分区别的类型?

首先可以通过Pick和Omit

```
interface Test {  
    name: string;  
    sex: number;  
    height: string;  
}  
  
type Sex = Pick<Test, 'sex'>;  
  
const a: Sex = { sex: 1 };  
  
type WithoutSex = Omit<Test, 'sex'>;  
  
const b: WithoutSex = { name: '1111', height: 'sss' };
```

比如Partial, Required.

再者可以通过泛型.

4. 什么是泛型, 泛型的具体使用?

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

```
interface Test<T = any> {  
    userId: T;  
}  
  
type TestA = Test<string>;  
type TestB = Test<number>;
```

```
const a: TestA = {  
  userId: '111',  
};  
  
const b: TestB = {  
  userId: 2222,  
};
```

4. 写一个计算时间的装饰器

代码

5. 写一个缓存的装饰器

代码

6. 实现一个路由跳转 通过ts约束参数的routeHelper

大量代码, 上课写

7. 实现一个基于ts和事件模式的countdown基础类

大量代码, 上课写

原理

看流程图.

1. Scanner 扫描器 (scanner.ts)

扫描器的作用就是将源代码生成token流
看图 扫描器.png

2. Parser 解析器 (parser.ts)

看图 解析器.png

3. Binder 绑定器 (binder.ts)

符号将 AST 中的声明节点与其它声明连接到相同的实体上。符号是语义系统的基本构造块。

```
function Symbol(flags: SymbolFlags, name: string) {  
    this.flags = flags;  
    this.name = name;  
    this.declarations = undefined;  
}
```

SymbolFlags 符号标志是个标志枚举，用于识别额外的符号类别（例如：变量作用域标志 FunctionScopedVariable 或 BlockScopedVariable 等）。

4. Checker 检查器 (checker.ts)

根据我们生成AST节点的声明起始节点位置，对传进来的字符串做位置类型语法等的校验与异常的抛出。

5. Emitter 发射器 (emitter.ts)

TypeScript 编译器提供了两个发射器：

emitter.ts: 它是 TS → JavaScript 的发射器

declarationEmitter.ts: 用于为 TypeScript 源文件 (.ts) 创建声明文件