TS 实战及面试题

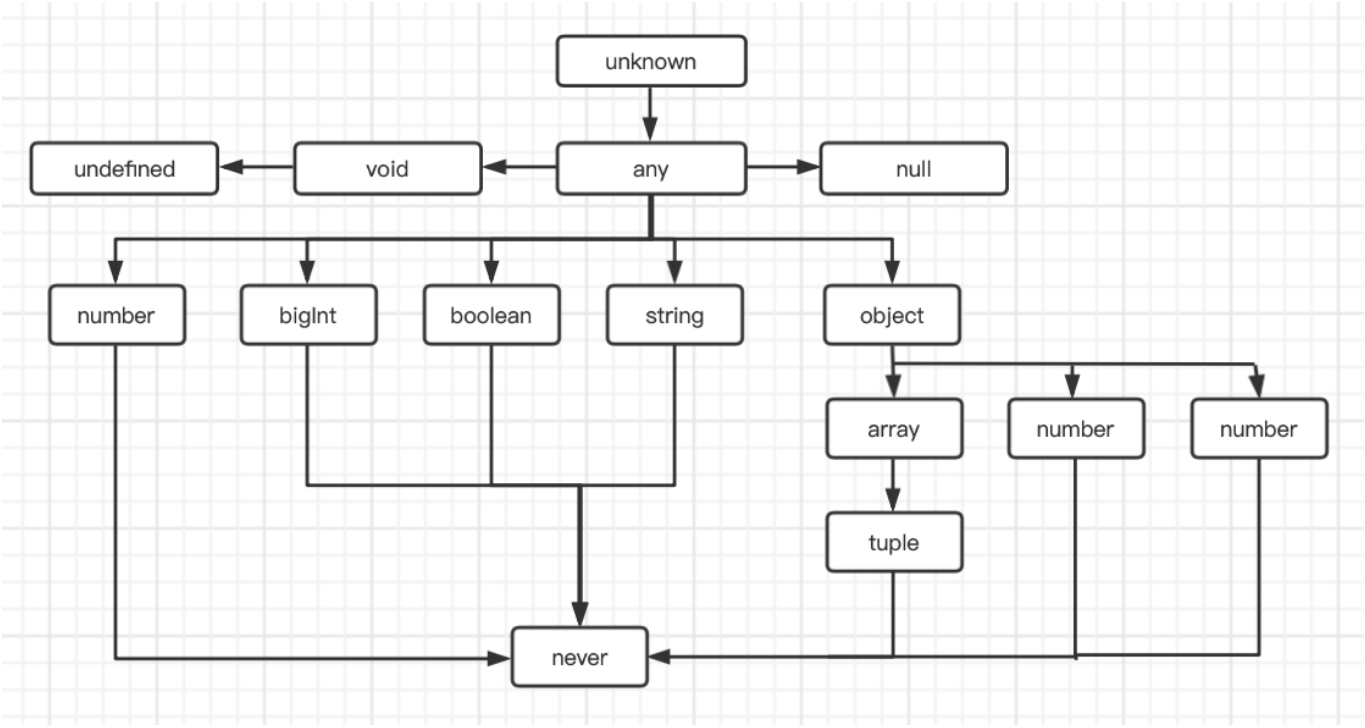
面试题

1、类型推论 & 可赋值性

a.什么是类型推论?

b.以下代码ts推论出的类型是什么?

```
let a = 1024;
let b = '1024';
const c = 'apple';
let d  = [true, false, true];
let e =  { name: 'apple'}
let f = null;
```



可赋值性：数组，布尔，数字，对象，函数，类、字符串，字面量类型，满足以下任一条件时，A类型可以赋值给B类型。

（1）A是B的子类型

（2）A是any类型

规则2是规则1的例外 2、类型断言

```
function formatInput(input: string):string {
    return input.slice(0, 10);
```

```
}
function getUserInput(): string | number {
    return 'test';
}
let input = getUserInput();
formatInput(input as string);
formatInput(<string>input);
```

3、type 和 interface的异同

interface侧重于描述数据结构，type（类型别名）侧重于描述类型

```
type age = number;
type dataType = number | string ;
type Method = 'GET' | 'POST' | 'PUT' | 'DELETE';

type User = {
    name: string
    age: number
}

interface User1 extends User {
    age: number;
}
const user1:User1 = {
    name: 'John',
    age: 12
}
```

异同点

两者的异同 1.相同点

a.都可以描述一个对象或者函数

```
// interface
interface User {
    name: string;
    age: number;
}

interface SetUser {
    (name: string, age: number): void;
}
// type
type User = {
    name: string;
    age: number;
```

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

b、interface和type都可以拓展，interface可以extends type, type也可以extends interface. 效果差不多，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.不同点

a.类型别名可以用于其它类型 （联合类型、元组类型、基本类型（原始值）），interface不支持

```
type PartialPointX = { x: number };
type PartialPointY = { y: number };

// union(联合)
type PartialPoint = PartialPointX | PartialPointY;
```

```
// tuple(元祖)
type Data = [PartialPointX, PartialPointY];

//primitive(原始值)
type Name = Number;

// typeof的返回值
let div = document.createElement('div');
type B = typeof div;
```

b.interface 可以多次定义 并被视为合并所有声明成员 type 不支持

```
interface Point {
  x: number;
}
interface Point {
  y: number;
}

const point: Point = { x: 1, y: 2 };
```c
interface User {
    name: string;
    age: number;
}

interface User {
    sex: string;
}
//User接口为:
{
    name: string;
    age: number;
    sex: string;
}
```

c.type 能使用 in 关键字生成映射类型，但 interface 不行。

```
type Keys = 'firstname' | 'surname';

type DudeType = {
    [key in Keys]: string;
};

const test: DudeType = {
    firstname: 'Pawel',
    surname: 'Grzybek',
};
```
```

提问：

```
// 1
type Options= {
    baseURL: string
    cacheSize?: number
    env?: 'prod' | 'dev'
}
// 2
class API {
    constructor(options: Options){}
}
// 3
new API({
    baseURL: 'http://myapi.site.com',
    env: 'prod'
})
// 4
new API({
    baseURL: 'http://myapi.site.com',
    badEnv: 'prod'
})
// 5
new API({
    baseURL: 'http://myapi.site.com',
    badEnv: 'prod'
} as Options)
// 6
let badOptions ={
    baseURL: 'http://myapi.site.com',
    badEnv: 'prod'
}
new API({badOptions})
// 7
let options: Options = {
    baseURL: 'http://myapi.site.com',
    badEnv: 'prod'
}
new API({badOptions})
```

4、装饰器问题

	类装饰器	方法装饰器	访问器装饰器	方法参数装饰器	属性装饰器
装饰器参数	类的构造函数	1、对于静态成员来说是类的构造函数，对于实例成员是类的原型对象。 2、成员的名字。 3、成员的属性描述符。	1、对于静态成员来说是类的构造函数，对于实例成员是类的原型对象。 2、成员的名字。 3、成员的属性描述符。	1、对于静态成员来说是类的构造函数，对于实例成员是类的原型对象。 2、参数的名字。 3、参数在函数参数列表中的索引。	1、对于静态成员来说是类的构造函数，对于实例成员是类的原型对象。 2、成员的名字。

执行顺序

a、有多个参数装饰器时：从最后一个参数依次向前执行

b、方法和方法参数中参数装饰器先执行。

c、类装饰器总是最后执行。

d、方法和属性装饰器，谁在前面谁先执行。因为参数属于方法一部分，所以参数会一直紧紧挨着方法执行。

5、接口类型

属性类接口 函数类接口 可索引接口 类类型接口 扩展接口

实战

1、axios 封装

2、TS 装饰器

感谢大家！