**type与interface区别**

**相同点. 都可以描述一个对象或者函数**

interface

interface IPerson {

name: string;

age: number;

}

interface SetPerson {

(name: string, age: number):void;

}

* type

type Person = {

name: string;

age: number;

}

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

注： 在ts编译成js后，所有的接口和type 都会被擦除掉

**不同点**

**1. 拓展**

接口可以扩展，但type不能extends和implement,但是type可以通过交叉类型实现interface的extends行为。interface 可以extends type，同时type也可以与interface类型交叉

1. interface extends interface, type 交叉类型可以实现

interface IAnimal {

name: string;

}

interface IPerson extends IAnimal{

age: number;

}

// type 交叉类型可以实现

type Animal = {

name: string;

}

type Person = Animal & { age: number};

1. interface extends type, type 与 interface 交叉

type Animal = {

name: string;

}

interface IPerson extends Animal {

age: number;

}

// type 与 interface 交叉

interface IAnimal {

name: string

}

type Person = IAnimal & {

age: number;

}

**2. Implements**

类可以以相同的方式实现接口或类型别名。但是请注意，类和接口被认为是静态的。因此，它们不能实现/扩展命名联合类型的类型别名

interface Point {

x: number;

y: number;

}

class SomePoint implements Point {

x: 1;

y: 2;

}

type Point2 = {

x: number;

y: number;

};

class SomePoint2 implements Point2 {

x: 1;

y: 2;

}

type PartialPoint = { x: number; } | { y: number; };

// A class may only implement another class or interface.

class SomePartialPoint implements PartialPoint {

x: 1;

y: 2;

}

**3. extends class**

类定义会创建两个东西：类的实例类型和一个构造函数。 因为类可以创建出类型，所以你能够在允许使用接口的地方使用类。

class Point {

x: number;

y: number;

}

interface Point3d extends Point {

z: number;

}

**4. type可以声明基本类型别名、联合类型、元祖等类型**

type Num = Number;

// 联合类型

type Shape = { kind: "circle"; radius: number } | { kind: "square"; x: number } | { kind: "triangle"; x: number; y: number };

// 元祖

type Size = [number, number];

type还可以使用typeof获取实例的类型进行赋值

let div = document.createElement('div');

type B = typeof div;

**5. 声明合并**

接口可以定义多次，并将被视为单个接口(即所有声明属性的合并)，而type不能定义多次

interface IPerson {

name: string;

age: number;

}

interface IPerson {

height: string;

}

let person:IPerson = {

name: 'xman',

age: 18,

height: '60kg'

}

**6. 映射类型**

type 能使用in关键字生成映射类型，但interface不行  
语法与索引签名的语法类型，内部使用了 for … in。 具有三个部分：

* 类型变量 K，它会依次绑定到每个属性。
* 字符串字面量联合的 Keys，它包含了要迭代的属性名的集合。
* 属性的结果类型。

type Keys = 'name' | 'height';

type Person = {

[key in Keys]: string; // 类型for... in

}

let person: Person = {

name: 'xman',

height: '60kg'

}

**其他**

export default interface IPerson {

name: string;

age: number;

}

type Person = {

name: string;

}

export default Person;

**7.type不能继承其他类型或者interface，但是他可以被interface继承，interface可以继承也可以被继承**

|  |
| --- |
| **type Animal = {  name:string,  haircolor:string }  interface Dog extends Animal{  }  let dog:Dog = {  name:'jiwawa',  haircolor:'black' }  console.log(dog);** |

### 如果你写type的时候用extends就会报错