OPEN ORIENTED

凹 凸 实 验 室

TypeScript 介绍

余澈

Why TypeScript?


```
> var sum = (str) =>
    str
      .match(/\d+/g)
      .reduce((n, acc) => n + Number(acc), 0)

  sum(`
  ----
  1|2|3|4|5
  ----
  `)
< 15
> sum(null)
✖ ▶Uncaught TypeError: Cannot read property 'match' of null                VM3264:3
      at sum (<anonymous>:3:6)
      at <anonymous>:1:1
> sum('我长得像吴彦祖')
✖ ▶Uncaught TypeError: Cannot read property 'reduce' of null               VM3264:4
      at sum (<anonymous>:4:5)
      at <anonymous>:1:1
```

Why TypeScript?


```
[ts] Object is possibly 'null'.
```

```ts
const sum = (str: string) =>
  str
    .match(/\d+/g)
    .reduce((n, acc) => n + Number(acc), 0)

sum(null)
```

```
[ts] Argument of type 'null' is not assignable to parameter of type 'string'.
```

That’s Why

- 增加了代码的可读性和可维护性
  - 在编译阶段就能发现错误
  - 类型是最好的文档，同时可以生成 Markdown
  - 通过 Language Server Protocol 提供 IDE 和编辑器增强
  - 类型推论
- 与 JavaScript 兼容良好
  - 本身就是 JavaScript 的超集，把 .js 改为 .ts 就能使用
- 社区活跃且强大
  - 作者 Anders HejIsberg 是世界顶级的语言和编译器专家
  - 大部分热门框架/库都有类型定义文件

基础类型


```ts
// 布尔值
const isDone: boolean = false
const createdByNewBoolean: boolean = new Boolean(1)
// Type 'Boolean' is not assignable to type 'boolean'.

// 数值
const decLiteral: number = 6
const hexLiteral: number = 0xf00d

// 字符串
const myName: string = 'yuche'
const say: string = `I'm a bitch`

// 空值
function logName(): void {
  console.log('My name is yuche')
}

// Null 和 Undefined
const u: undefined = undefined
const n: null = null
```

数组


```ts
const fibonacci: number[] = [1, 1, 2, 3, 5]

const fibonacci2: number[] = [1, '1', 2, 3, 5]
// Type 'number | string' is not assignable to type 'number'.

const fibonacci3: (number | string)[] = [1, '1', 2, 3, 5]

const fibonacci4: any[] = [1, '1', 2, 3, 5, () => 8]

const fibonacci5: Array<number> = [1, 1, 2, 3, 5]
// we will talk about this later

// aruguments is a Array-like Object
function f () {
  const args: IArguments = arguments
  // IArguments is a interface
}

// Tuple
const people: [string, number] =  ['Steve Jobs', 56]
```

函数


```ts
function sum(x: number, y: number): number {
  return x + y
}
sum(1, 2, 3);
// Error: Supplied parameters do not match any signature of call target.

// 可选型
function sum2(x: number, y?: number): number {
  return y ? x + y : x
}
sum2(1) // ok
sum2(1, 2) // also ok

function sum3(x?: number, yyyy: number): number {
  return yyyy ? x + yyyy : x
}
// Error: A required parameter cannot follow an optional parameter.
// 可选型必须在必须参数之后
```

函数


```ts
// 参数默认值
function sum4(x?: number, y = 0): number {
  return x + y
}
sum4(5) // 5
sum4(5, 5) // 10
sum4(5, '5')
// y 会被自动推导为 number

// 剩余参数
function concat(array: any[], ...items: any[]) {
  return array.concat(items)
}

let a = [];
concat(a, 1, 2, 3); // [1, 2, 3]

// 重载
// 等会儿有时间在实战讲
```

Interface 接口

02

在面向对象语言中，接口（Interfaces）是一个很重要的概念，它是对行为的抽象，而具体如何行动需要由类（classes）去实现（implements）。

TypeScript 中的接口是一个非常灵活的概念，除了可用于对类的一部分行为进行抽象以外，也常用于对「对象的形状（Shape）」进行描述。

Interface 接口


```ts
interface Person {
  name: string
  age: number
  die: () => void
}

const Steve: Person = {
  name: 'Steve Jobs',
  age: 56,
  die() {
    console.log(`I'm dead.`)
  }
}

const Bill: Person = {
  name: 'Bill Gates',
  age: 61
}
// Type '{ name: string; age: number; }' is not assignable to type 'Person'.
```

Interface 接口


```ts
interface Person {
  name: string
  age?: number,
  die?: () => void
}

const Jobs: Person = {
  name: 'Steve Jobs',
  age: 56,
  die() {
    console.log(`I'm dead`)
  }
}

const Bill: Person = {
  name: 'Steve Gates'
}
// ok
```

Interface 接口

02

```ts
// 只读属性
interface Person {
  readonly id: number
  name: string
  age?: number
  die?: () => void
}

const Jobs: Person = {
  id: 10086,
  name: 'Steve Jobs',
  age: 56,
  die() {
    console.log(`I'm dead`)
  }
}

Jobs.id = 10010
// Cannot assign to 'id' because it is a constant or a read-only property.
```

类 Class

TypeScript 拥有所有目前 ECMAScript class 进入标准或还在提案的属性。包括继承、存取器、静态方法、装饰器等。

在这个基础之上又添加 public、private 和 protected 三种修饰符。

类 Class


```
class Animal {
  public name;
  public constructor(name) {
    this.name = name;
  }
}
let a = new Animal('Jack');
console.log(a.name); // Jack
a.name = 'Tom';
console.log(a.name); // Tom

class Animal2 {
  private name;
  public constructor(name) {
    this.name = name;
  }
}
let b = new Animal2('Rose')
b.name = 'Nancy'
// Property 'name' is private and only accessible within class 'Animal2'.
```

类 Class


```
class Animal {
  protected name;
  public constructor(name) {
    this.name = name;
  }
}

class Cat extends Animal {
  constructor(name) {
    super(name);
    console.log(this.name); // ok
  }
}

// public 修饰的属性或方法是公有的，可以在任何地方被访问到
// 默认所有的属性和方法都是 public 的
// private 修饰的属性或方法是私有的，不能在声明它的类的外部访问
// protected 修饰的属性或方法是受保护的，它和 private 类似，区别是它在子类中也是允许被访问的
```

泛型 Generics

02

泛型（Generics）是指在定义函数、接口或类的时候，不预先指定具体的类型，而在使用的时候再指定类型的一种特性。

泛型 Generics


```ts
function createArray<T>(value: T, len: number): Array<T> {
  return Array(len).fill(value)
}

createArray('100', 3) // type is string[]
createArray(100, 3) // type is number[]

// 多个类型
function swap<T1, T2>(tuple: [T1, T2]): [T2, T1] {
  return [tuple[1], tuple[0]]
}
swap([7, 'seven']) // ['seven', 7]
swap([() => 7, 'seven']) // ['seven', Function]
// error free!
```

泛型 Generics

02

```ts
// 泛型约束

function getLength<T>(arg: T): number {
  return arg.length
}
// [ts] Property 'length' does not exist on type 'T'.

interface Lengthwise {
  length: number;
}
function getLength2<T extends Lengthwise>(arg: T): number {
  return arg.length
}
getLength2([]) // ok
getLength2('') // ok
getLength2(123)
// Argument of type '123' is not assignable to parameter of type 'Lengthwise'.
```

泛型 Generics


```tsx
// 泛型类
interface Props { size: 'big' | 'small' }
interface State { visbiliy: boolean }

class App extends React.Component<Props, State> {
  state = { visbiliy: true }
  constructor(props, context) {
    super(props, context)
  }
  handleClick = () => this.setState({ show: false })
  // 'show' is not in interface State
  render() {
    return this.state.visbiliy
      && <button type={this.props.size} onClick={this.handleClick} />
  }
}

ReactDOM.render(<App size='middle' />, document.body)
// 'middle' is not assignable for Props.size
```

实例演示

One More Thing

THANKS

FOR YOUR WATCHING

02

OPEN ORIENTED

凹 凸 实 验 室