Microsoft

Web Development Fundamentals

TypeScript 接口实作

Implement interfaces in TypeScript

Jun 2021
Microsoft Reactor | Ryan Chung

Microsoft

Ryan Chung

Instructor / DevelopIntelligence
Founder / MobileDev.TW

@ryanchung403 on WeChat
Ryan@MobileDev.TW

https://github.com/ryan403/reactor-sh-2021-typescript

Microsoft

Reactor

developer.microsoft.com/reactor/

@MSFTReactor on Twitter

900 XP

在 TypeScript 中实现接口

4 分钟 剩余 • 模块 • 已完成 6 个单元，共 8 个

★★★★★ 4.8 (33)

中级 | 开发人员 | 学生 | Azure | Visual Studio Code

JavaScript 不支持接口，因此，作为 JavaScript 开发人员，你可能有使用接口的经验，也可能没有经验。 在 TypeScript 中，可以像在传统的面向对象的编程中那样使用接口。 你还可以使用接口来定义对象类型，这是本模块的主要内容。

学习目标

通过学习本模块，你将了解如何：

- 解释在 TypeScript 中使用接口的原因。
- 声明和实例化接口。
- 扩展接口。
- 使用自定义数组类型声明接口。

先决条件

- TypeScript 知识
- 熟悉 JavaScript
- 熟悉 JavaScript 中的函数和数组
- 安装的软件：
  - Git
  - Visual Studio Code
  - Node.js
  - TypeScript

Microsoft Reactor

学习目标

- 了解如何在TypeScript中使用interface
- Interface的宣告与实体化
- Interface延伸(Extend)
- 客制化数组宣告于interface中

Interface 接口/介面

- 代码世界中的合同

练习：新进员工

- 打开VS Code，档案 -> 开启资料夹...
- 建立 interface_practice 资料夹
- 选择资料夹

练习：设置TypeScript专案

https://github.com/ryan403/reactor-sh-2021-typescript

- 在interface_practice资料夹中新增档案
  - main.ts
- 检视 -> 终端
  - tsc --init
- 检视tsconfig.json档案
  - 找到target，将es5改为ES2015
  - 找到outDir，取消注解，设定为build
- 在终端机中执行 tsc 读取最新jsconfig.json设置

```
"target": "ES2015",
"outDir": "build",
```

Microsoft Reactor

https://github.com/ryan403/reactor-sh-2021-typescript

main.ts

```
interface Employee{
    firstName:string;
    lastName:string;
    fullName():string;
}

let thisEmployee:Employee = {
    firstName:"Ryan",
    lastName:"Chung",
    fullName():string{
        return this.firstName + " " + this.lastName
    }
};

console.log(`Hello! ${thisEmployee.fullName()}`);
```

透过HTML网页执行JavaScript

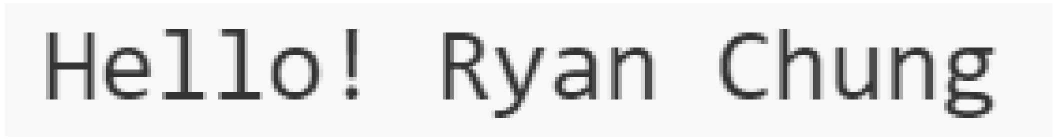
- 在interface_practice资料夹中
  - 建立index.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <link rel="stylesheet" href="">
    </head>
    <body>
        <h1>执行 main.js </h1>
        <script src="build/main.js"></script>
    </body>
</html>
```

测试

- 检视 -> 终端
- 产生js档案

  tsc
- 在侧边栏index.html点击滑鼠右键
  - Open with Live Server 或 使用预设浏览器开启
  - 网页在浏览器中开启后，同时打开开发人员工具->Console



```
Hello! Ryan Chung
```

为什么要用 interface?

- 将常用型态建立成interface
  - 方便重复使用
- 会做型别检查(type checking)
  - 错了会说
- 能确认每个这种型别的物件，都具备指定的属性与方法
  - 少了会说
- 明确了解要回传的内容是什么
  - 错了、少了都会说

为什么不直接写成自定义的Type

- Interface宣告后，还可以继续加新的属性

```
type EmployeeV2 = {
    firstName:string;
    lastName:string;
    fullName():string;
}
let thisEmployee2:EmployeeV2 = {
    firstName:"Ryan",
    lastName:"Chung",
    fullName():string{
        return this.firstName + " " + this.lastName
    }
};

console.log(`Hi! ${thisEmployee2.fullName()}`);
```

Interface宣告的注意事項

- PascalCase
  - 駝峰式大小寫，每個單字的首字大寫，也包含第一個單字
- 避開預先定義的type名稱
  - string, number, array, boolean, ...
- 首字避免是I(Interface)

Interface中的属性类型

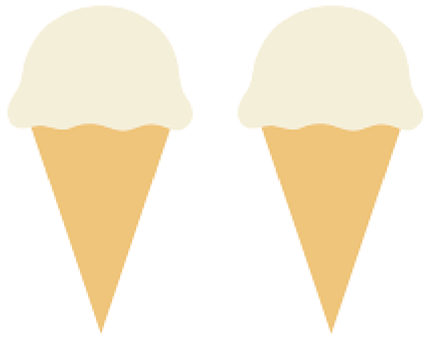
属性类型	说明	示例
Required	一定要有	firstName:string;
Optional	非必要 (没有不会报错)	firstName?:string;
Read only	必须在Object建立时指定值 (之后无法修改)	readonly firstName:string;

练习：来买冰淇淋

```
interface IceCream{
    flavor:string;
    scoops:number;
}

let myIceCream:IceCream = {
    flavor:'vanilla',
    scoops:2
}

console.log(`I have ${myIceCream.scoops} scoops of ${myIceCream.flavor} ice cream.`);
```



```
I have 2 scoops of vanilla ice cream.
```

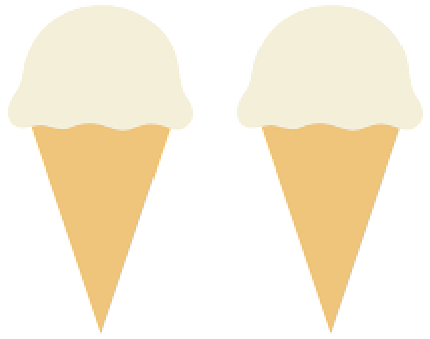
检视有没有买太多

```
interface IceCream{
    flavor:string;
    scoops:number;
}

let myIceCream:IceCream = {
    flavor:'vanilla',
    scoops:2
}

function countCheck(dessert:IceCream){
    if(dessert.scoops>=4){
        return dessert.scoops + '!? TOO many!!';
    }else{
        return 'Enjoy it!';
    }
}

console.log(`I have ${myIceCream.scoops} scoops of ${myIceCream.flavor} ice cream.`);
console.log(`${countCheck(myIceCream)}`);
```



```
I have 2 scoops of vanilla ice cream.
Enjoy it!
```

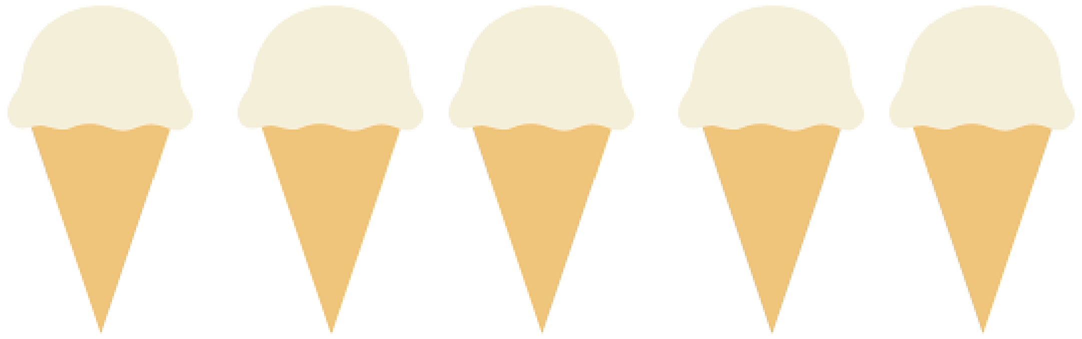
检视有没有买太多 – 测试

```
interface IceCream{
    flavor:string;
    scoops:number;
}

let myIceCream:IceCream = {
    flavor:'vanilla',
    scoops:5
}

function countCheck(dessert:IceCream){
    if(dessert.scoops>=4){
        return dessert.scoops + '!? TOO many!!';
    }else{
        return 'Enjoy it!';
    }
}

console.log(`I have ${myIceCream.scoops} scoops of ${myIceCream.flavor} ice cream.`);
console.log(`${countCheck(myIceCream)}`);
```



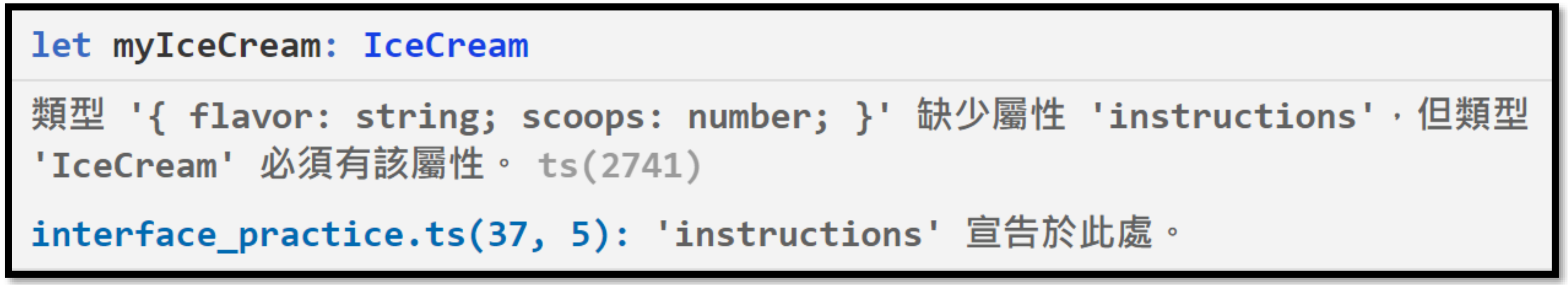
```
I have 5 scoops of vanilla ice cream.
5!? TOO many!!
```

增加屬性

- myIceCream会报错

```
interface IceCream{
    flavor:string;
    scoops:number;
    instructions:string;
}

let myIceCream:IceCream = {
    flavor:'vanilla',
    scoops:5
}
```



```
let myIceCream: IceCream
類型 '{ flavor: string; scoops: number; }' 缺少屬性 'instructions'，但類型
'IceCream' 必須有該屬性。 ts(2741)
interface_practice.ts(37, 5): 'instructions' 宣告於此處。
```

增加属性

- 若设定为选择性的就不会报错了

```
interface IceCream{
    flavor:string;
    scoops:number;
    instructions?:string;
}

let myIceCream:IceCream = {
    flavor:'vanilla',
    scoops:5
}
```

Extend 延伸

- 复制一个Interface的成员到另一个Interface
  - 所有Interface的必要的(Required)属性都要实作
  - 两个Interface可以有相同的属性名称(但型别 type 必须一致)

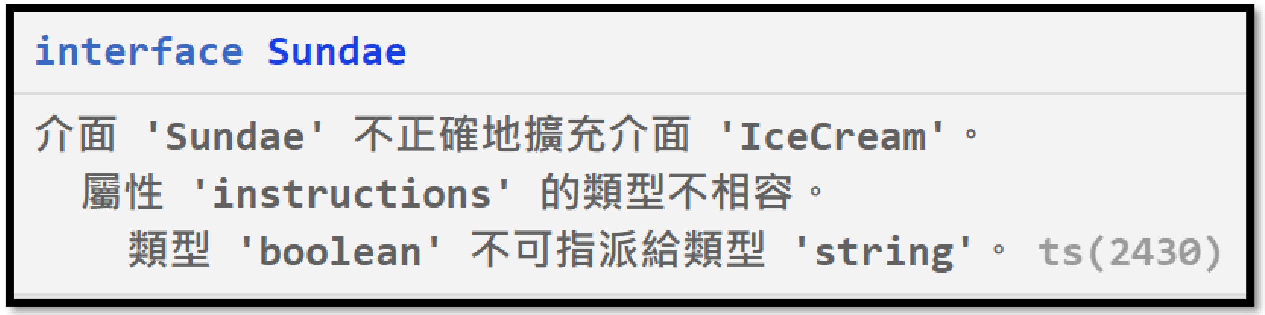
https://github.com/ryan403/reactor-sh-2021-typescript

练习：圣代与冰淇淋

- instructions属性与IceCream同名但不同Type

```
interface IceCream{
    flavor:string;
    scoops:number;
    instructions?:string;
}

interface Sundae extends IceCream{
    sauce: 'chocolate' | 'caramel' | 'strawberry';
    nuts?:boolean;
    whippedCream?:boolean;
    instructions?:boolean;
}
```



```
interface Sundae
介面 'Sundae' 不正確地擴充介面 'IceCream'。
  屬性 'instructions' 的類型不相容。
    類型 'boolean' 不可指派給類型 'string'。 ts(2430)
```

Microsoft Reactor

练习：圣代与冰淇淋

- instructions属性改为string type就不会报错了

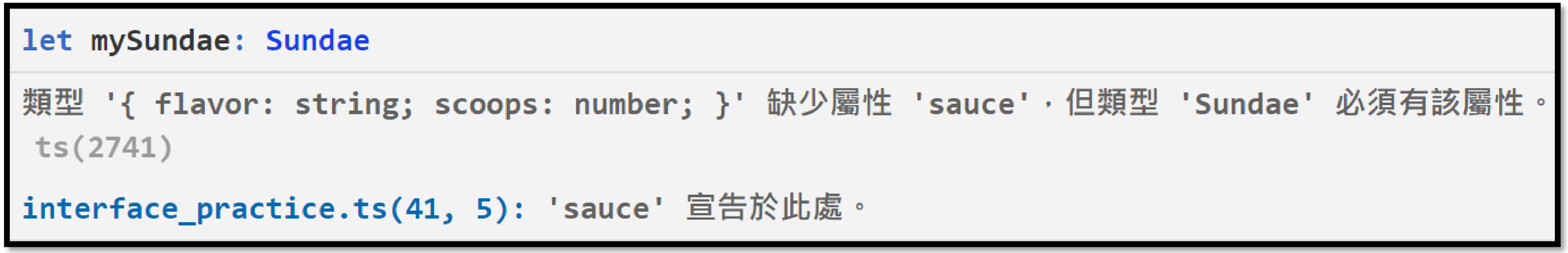
```
interface IceCream{
    flavor:string;
    scoops:number;
    instructions?:string;
}

interface Sundae extends IceCream{
    sauce: 'chocolate' | 'caramel' | 'strawberry';
    nuts?:boolean;
    whippedCream?:boolean;
    instructions?:string;
}
```

练习：圣代与冰淇淋

- 圣代必须要满足所有必要属性，选择性属性则视需求加入

```
let mySundae:Sundae = {
    flavor:'vanilla',
    scoops:2
}
```



```
let mySundae: Sundae
類型 '{ flavor: string; scoops: number; }' 缺少屬性 'sauce'，但類型 'Sundae' 必須有該屬性。
 ts(2741)
interface_practice.ts(41, 5): 'sauce' 宣告於此處。
```

练习：圣代与冰淇淋

- 圣代必须要满足所有必要属性，选择性属性则视需求加入

```
let mySundae:Sundae = {
    flavor:'vanilla',
    scoops:2,
    sauce:'caramel',
    nuts:true
}
```

数量检查

- 让传入值可以是冰淇淋也可以是圣代

```
function countCheck(dessert:IceCream | Sundae){
    if(dessert.scoops>=4){
        return dessert.scoops + '!? TOO many!!';
    }else{
        return 'Enjoy it!';
    }
}

console.log(`I have ${myIceCream.scoops} scoops of ${myIceCream.flavor} ice cream.`);
console.log(`${countCheck(myIceCream)}`);

console.log(`I have ${mySundae.scoops} scoops of ${mySundae.flavor} sundae with ${mySundae.sauce}.`);
console.log(`${countCheck(mySundae)}`);
```

```
I have 5 scoops of vanilla ice cream.
5!? TOO many!!
I have 2 scoops of vanilla sundae with caramel.
Enjoy it!
```

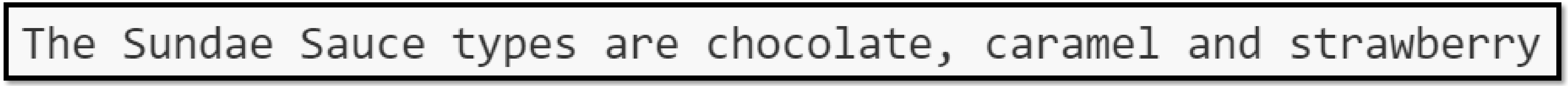
Indexable Type

- 有顺序性，可用[数字]取得

```
interface SauceType{
    [index:number]:string;
}

let sundaeSauceType:SauceType = ['chocolate','caramel','strawberry'];

console.log(`The Sundae Sauce types are ${sundaeSauceType[0]}, ${sundaeSauceType[1]} and ${sundaeSauceType[2]}`);
```



```
The Sundae Sauce types are chocolate, caramel and strawberry
```

练习：使用Interface描述 API的回传结果

- 预先定义好API回传的内容格式

- 打开VS Code，档案 -> 开启资料夹...
- 建立 api_get 资料夹
- 选择资料夹

https://github.com/ryan403/reactor-sh-2021-typescript

练习：设置TypeScript专案

- 在资料夹中新增档案
  - api_get.ts
- 检视 -> 终端
  - tsc --init

api_get.ts

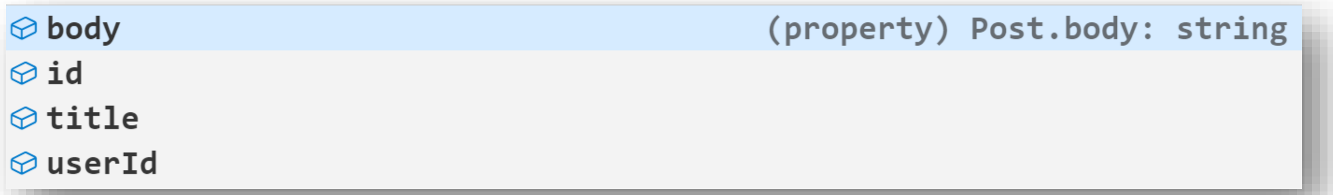
https://github.com/ryan403/reactor-sh-2021-typescript

```
declare function require(name:string):any;
var axios = require('axios');

interface Post{
    userId:number;
    id:number;
    title:string;
    body:string
}

var axios_config = {
    method:'get',
    url:'https://jsonplaceholder.typicode.com/posts'
};

axios(axios_config)
.then(function(response:any){
    let result = response.data as Post[];
    console.log(result[0].title);
})
.catch(function(error:any){
    console.log(error);
})
```



```
body        (property) Post.body: string
id
title
userId
```

会出现属性提示

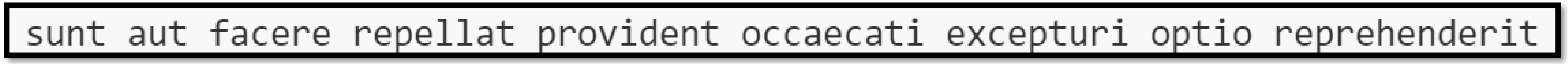
测试

- 检视 -> 终端
- 产生js档案

  tsc api_get.ts
- 安装所需套件

  npm install axios
- 执行js档案

  node api_get.js



```
sunt aut facere repellat provident occaecati excepturi optio reprehenderit
```

练习

calculateInterestOnlyLoanPayment	计算贷款利息
属性1 – principle	借贷本金
属性2 – interestRate	年利率(例如 5%，标示为5)

calculateConventionalLoanPayment	计算传统贷款金额
属性1 – principle	借贷本金
属性2 – interestRate	年利率(例如 5%，标示为5)
属性3 - months	借贷时间(以月表示)

练习

- 先宣告一个interface，命名为Loan
  - 里面有两个属性
    - principle
    - interestRate
- 再宣告一个interface，命名为ConventionalLoan
  - 延伸自 Loan
  - 加上额外的属性
    - months
- 使用上面两个interface来实现上页的函数

练习

- 下载练习用档案
  - git clone https://github.com/MicrosoftDocs/mslearn-typescript
- 打开指定练习档案
  - 使用VS Code
  - 档案 -> 开启资料夹
  - mslearn-typescript/code/module-03/m03-start

练习一：宣告Loan Interface

- 在module03.ts，移动至EXERCISE 1

```
/* Module 3: Implement interfaces in TypeScript
   Lab Start  */

/*  EXERCISE 1
    TODO: Declare the Loan interface. */

interface Loan{
    principle:number,
    interestRate:number
}
```

宣告ConventionalLoan Interface

- 移动至EXERCISE 1下方的TODO

```
/*  TODO: Declare the ConventionalLoan interface. */

interface ConventionalLoan extends Loan{
    months:number
}
```

练习二：修改函数

- 修改 calculateInterestOnlyLoanPayment

```
function calculateInterestOnlyLoanPayment(loanTerms:Loan): string {
    // Calculates the monthly payment of an interest only loan
    let interest:number = loanTerms.interestRate / 1200;
    // Calculates the Monthly Interest Rate of the loan
    let payment:number;
    payment = loanTerms.principle * interest;
    return 'The interest only loan payment is ' + payment.toFixed(2);
}
```

练习二：修改函数

- 修改 calculateConventionalLoanPayment

```
/*  TODO: Update the calculateConventionalLoanPayment function. */

function calculateConventionalLoanPayment(loanTerms:ConventionalLoan):string {
    // Calculates the monthly payment of a conventional loan
    let interest:number = loanTerms.interestRate / 1200;
    // Calculates the Monthly Interest Rate of the loan
    let payment:number;
    payment = loanTerms.principle * interest / (1 - (Math.pow(1 / (1 + interest), loanTerms.months)));
    return 'The conventional loan payment is ' + payment.toFixed(2);
}
```

https://github.com/ryan403/reactor-sh-2021-typescript

练习三：建立传入资料

- 符合两种interface的物件

```
let iOnly:Loan = {
    principle:30000,
    interestRate:5
}

let conTest:ConventionalLoan = {
    principle:30000,
    interestRate:5,
    months:180
}
```

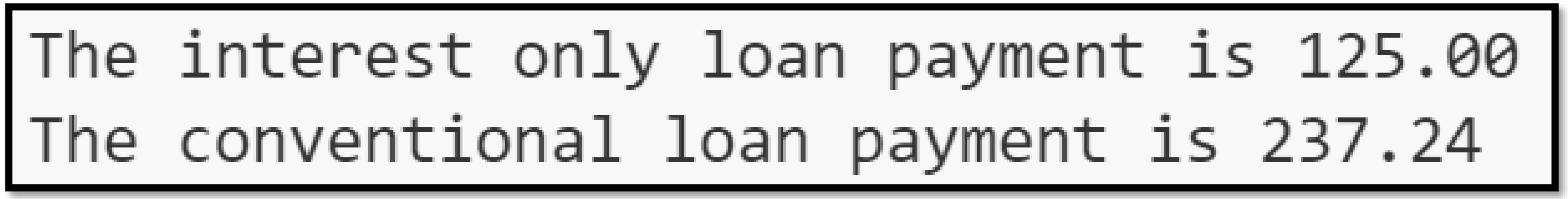
Microsoft Reactor

练习四：执行函数、观察结果

- 传入对应的函数

```
let interestOnlyPayment = calculateInterestOnlyLoanPayment(iOnly);
let conventionalPayment = calculateConventionalLoanPayment(conTest);

console.log(interestOnlyPayment);
console.log(conventionalPayment);
```



```
The interest only loan payment is 125.00
The conventional loan payment is 237.24
```

知识检查

1. 接口的主要工作是什么?

   ○ 定义对象的实现详细信息。

   ○ 描述对象的属性和返回类型。

   ○ 履行对象的代码协定。

2. 当省略接口中的属性时，如何防止类型系统引发错误?

   ○ 将属性设置为可选属性。

   ○ 将属性设置为必需属性。

   ○ 将属性设置为只读属性。

3. 用另一个接口扩展一个接口会发生什么情况?

   ○ 如果属性具有完全相同的名称，则多个接口可以具有相同的属性。

   ○ 如果两个接口具有名称相同但类型不同的属性，TypeScript 会完全忽略该属性。

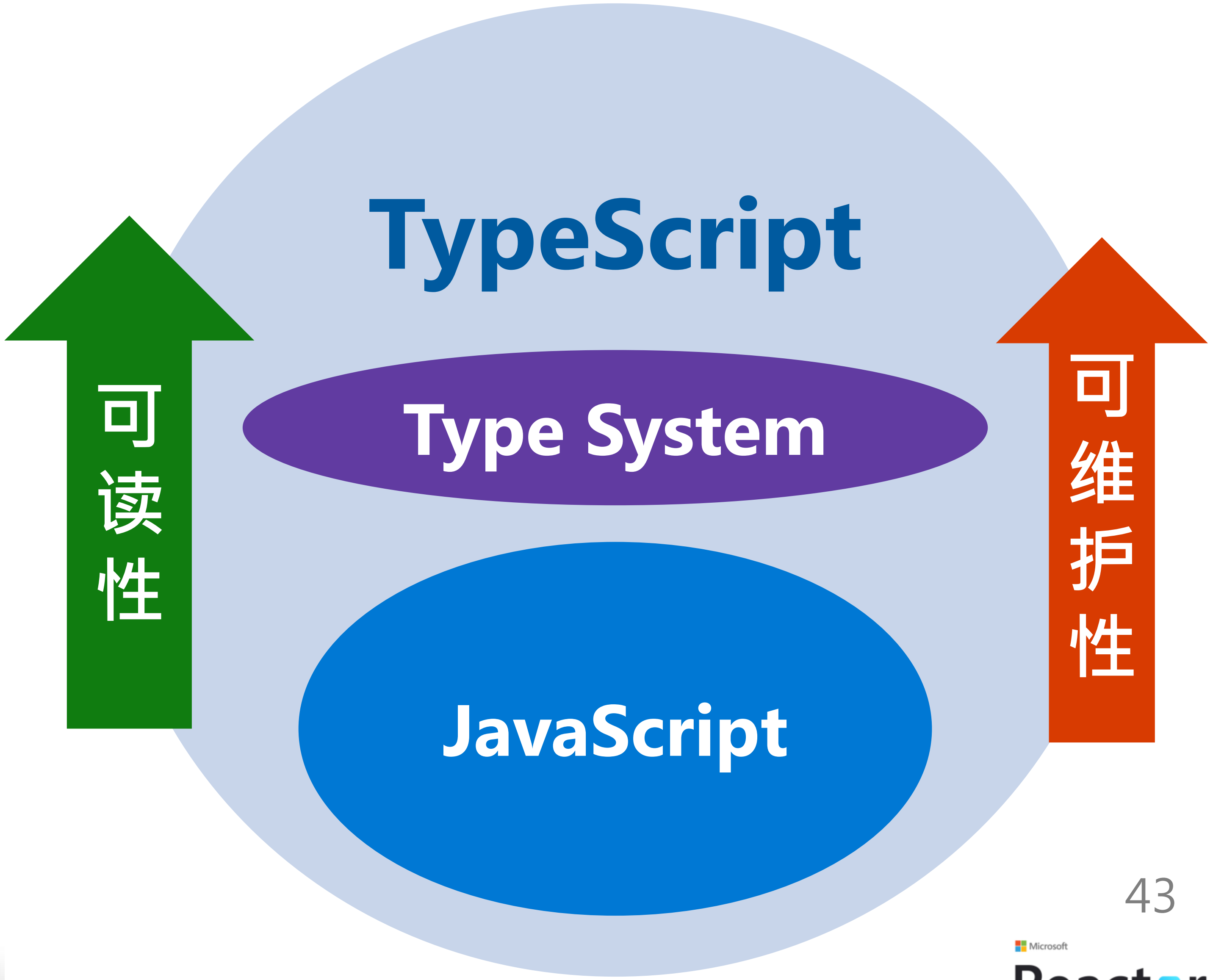
   ○ 必须从所有接口实现所有必需的属性。

Summary 摘要

- Interface(接口/介面)
  - 使用时机与好处
  - 宣告与实现
  - 属性类型
    - Required、Optional、Read Only
  - 延伸(Extend)
    - 使用方式
    - 注意事项

立志做一个不马虎的程序员！

- 更多的型态支持
  - 指定数据型态、多种输入型态
- 及早发现潜在的错误
  - 开发中提示、智慧校正
- 严谨、不含糊
  - 明确指定、选择性指定
- 提前应用新语法
  - Optional Chaining、ES7…



Microsoft

Reactor

developer.microsoft.com/reactor/
@MSFTReactor on Twitter

Microsoft

议程结束
感谢聆听

请记得填写课程回馈问卷 (Event ID : XXXXX)
https://aka.ms/Reactor/Survey