



# Web Development Fundamentals

## TypeScript 基础

### Intro to TypeScript

Jun 2021

Microsoft Reactor | Ryan Chung

```
led by player to  
s.load_image("kg.png")  
  
[self]:  
    initialize Dog object and create Text of  
g, self).__init__(image = Dog.image  
    x = games.mouse.x  
    bottom = games.sc  
  
re = games.Text(value = 0, size = 24  
    top = 5, right = gam  
  
reen.add(self.score)  
1 = games.Text(value = 0, size = 24  
    top = 5, left = gam
```



# Ryan Chung

Instructor / DevelopIntelligence  
Founder / MobileDev.TW

@ryanchung403 on WeChat  
Ryan@MobileDev.TW







# Reactor



[developer.microsoft.com/reactor/](https://developer.microsoft.com/reactor/)  
[@MSFTReactor](#) on Twitter



# TypeScript 入门

36 分钟 • 模块 • 8 单元

★★★★★ 4.8 (87)

初级

开发人员

学生

Visual Studio Code

本模块将介绍 TypeScript 语言、创建它的原因，以及如何将其用于 JavaScript 开发。你还将设置一个 TypeScript 开发环境，供日后练习。

## 学习目标

在本模块中，你将：

- 了解在 Web 开发中 TypeScript 相比 JavaScript 的改进。
- 选择一个 TypeScript 编辑器。
- 安装 TypeScript。
- 在 Visual Studio Code 中设置 TypeScript 项目。

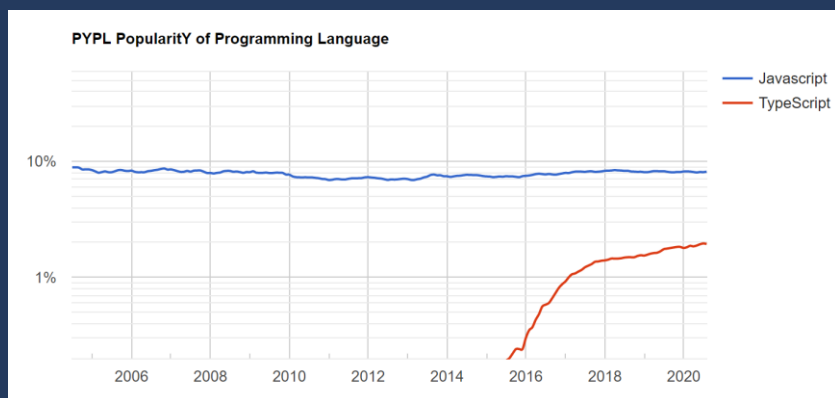
## 先决条件

- JavaScript 知识

# 学习目标

- 了解TypeScript对于JavaScript的改进部分
- 选择TypeScript编辑器
- 安装TypeScript
- 在VS Code中设定TypeScript专案

# TypeScript 简介



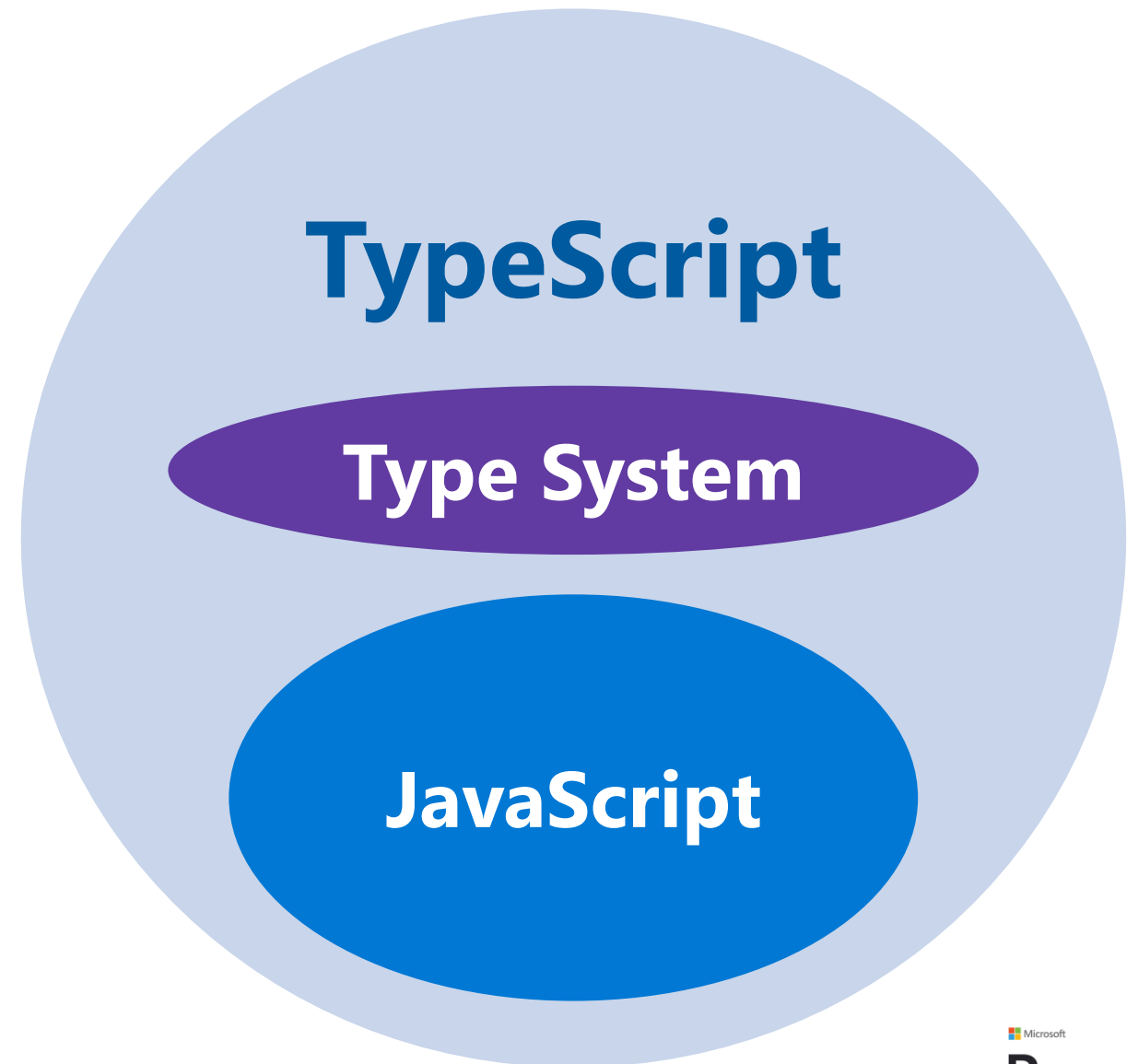
Worldwide, Apr 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.5 %	-1.0 %
2		Java	17.51 %	-0.6 %
3		JavaScript	8.19 %	+0.2 %
4		C#	7.05 %	-0.2 %
5	↑	C/C++	6.73 %	+1.0 %
6	↓	PHP	6.23 %	+0.0 %
7		R	3.86 %	+0.0 %
8		Objective-C	2.77 %	+0.3 %
9	↑	TypeScript	1.87 %	-0.0 %
10	↓	Swift	1.85 %	-0.3 %

Pierre Carbonnelle, 2020

# 概要

- 你已经在撰写TypeScript!
- 可以写得更严谨
- 协助找出潜在异常程序片段
- JS的用途日益扩大
  - 小特效 -> 服务



# 线上执行语法测试 – TypeScript Playground

- 左边编辑后按下Run，右边观察结果

The screenshot displays the TypeScript Playground web application. The interface is divided into two main sections: a left editor and a right results panel.

**Left Panel (Editor):**

- Header: "TypeScript" logo, navigation links (Download, Docs, Handbook, Community, Playground, Tools), and a "Search Docs" button.
- Sub-header: "Playground" tab, "TS Config" dropdown, "Examples" dropdown, "What's New" dropdown, and a "Settings" button.
- Toolbar: "v4.0.2" dropdown, "Run" button, "Export" dropdown, and "Share" button.
- Code Editor: Contains the following TypeScript code:

```
1 // Welcome to the TypeScript Playground, this is a website
2 // which gives you a chance to write, share and learn TypeScript.
3
4 // You could think of it in three ways:
5 //
6 // - A place to learn TypeScript in a place where nothing can break
7 // - A place to experiment with TypeScript syntax, and share the URLs w
8 // - A sandbox to experiment with different compiler features of TypeSc
9
10 const anExampleVariable = "Hello World"
11 console.log(anExampleVariable)
12
13 // To learn more about the language, click above in "Examples" or "What
14 // Otherwise, get started by removing these comments and the world is yc
15
```

**Right Panel (Results):**

- Header: Tabs for ".JS", ".D.TS", "Errors", "Logs", and "Plugins". The ".JS" tab is selected.
- Code Output: The same code as the left panel is shown, but with the addition of a "use strict"; directive at the top:

```
"use strict";
// Welcome to the TypeScript Playground, this is a website
// which gives you a chance to write, share and learn TypeScript.
// You could think of it in three ways:
//
// - A place to learn TypeScript in a place where nothing can break
// - A place to experiment with TypeScript syntax, and share the URLs with others
// - A sandbox to experiment with different compiler features of TypeScript
const anExampleVariable = "Hello World";
console.log(anExampleVariable);
// To learn more about the language, click above in "Examples" or "What's New".
// Otherwise, get started by removing these comments and the world is your playground.
```

<https://www.typescriptlang.org/play/>



# 线上执行语法测试 – TypeScript Playground

- 在左边区块打上，然后按下Run

```
function addNumbers(x,y){  
  return x+y;  
}
```

```
console.log(addNumbers(3,6))
```

.JS .D.TS Errors **2** Logs Plugins

```
"use strict";  
function addNumbers(x, y) {  
  return x + y;  
}  
console.log(addNumbers(3, 6));
```

# 线上执行语法测试 – TypeScript Playground

- 右边区块切换至Errors瞧瞧

```
function addNumbers(x,y){  
  return x+y;  
}
```

```
console.log(addNumbers(3,6))
```

.JS .D.TS Errors<sup>2</sup> Logs Plugins

## Errors in code

Parameter 'x' implicitly has an 'any' type.

Parameter 'y' implicitly has an 'any' type.

# 线上执行语法测试 – TypeScript Playground

- 但还是可以计算出结果

```
function addNumbers(x,y){  
  return x+y;  
}
```

```
console.log(addNumbers(3,6))
```

.JS .D.TS Errors **2** Logs Plugins

[LOG]: 9

# 线上执行语法测试 – TypeScript Playground

- 修改程式码进行测试

```
function addNumbers(x,y){  
  return x+y;  
}
```

```
console.log(addNumbers("three",6))
```

.JS .D.TS Errors **2** Logs Plugins

[LOG]: 9

---

[LOG]: "three6"

# 线上执行语法测试 – TypeScript Playground

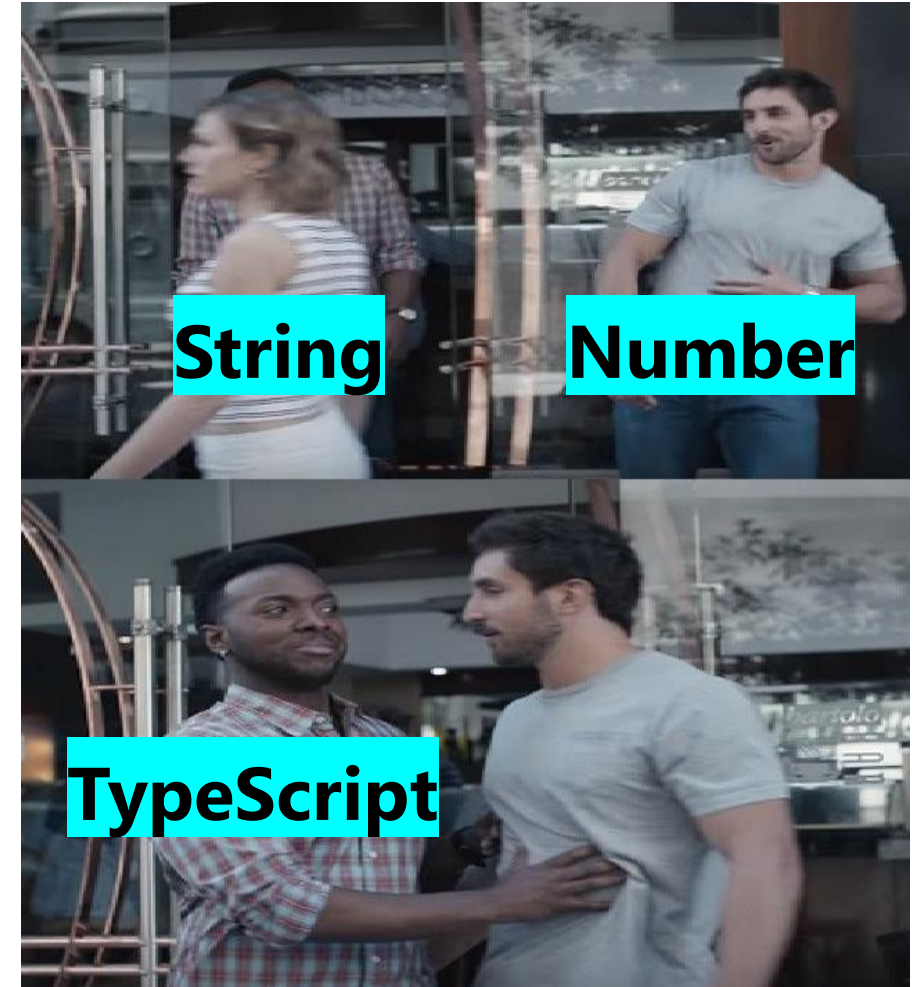
- 修改程式码进行测试

```
function addNumbers(x:number,y:number){  
  return x+y;  
}
```

```
console.log(addNumbers("three",6))
```

Argument of type 'string' is not assignable to parameter of type 'number'. (2345)

[Peek Problem \(Alt+F8\)](#). No quick fixes available





# 线上执行语法测试 – TypeScript Playground

- 再次修改程式码进行测试

```
function addNumbers(x:number,y:number){  
    return x+y;  
}
```

[.JS](#) [.D.TS](#) [Errors](#) [Logs](#) [Plugins](#)

```
console.log(addNumbers(3,6))
```

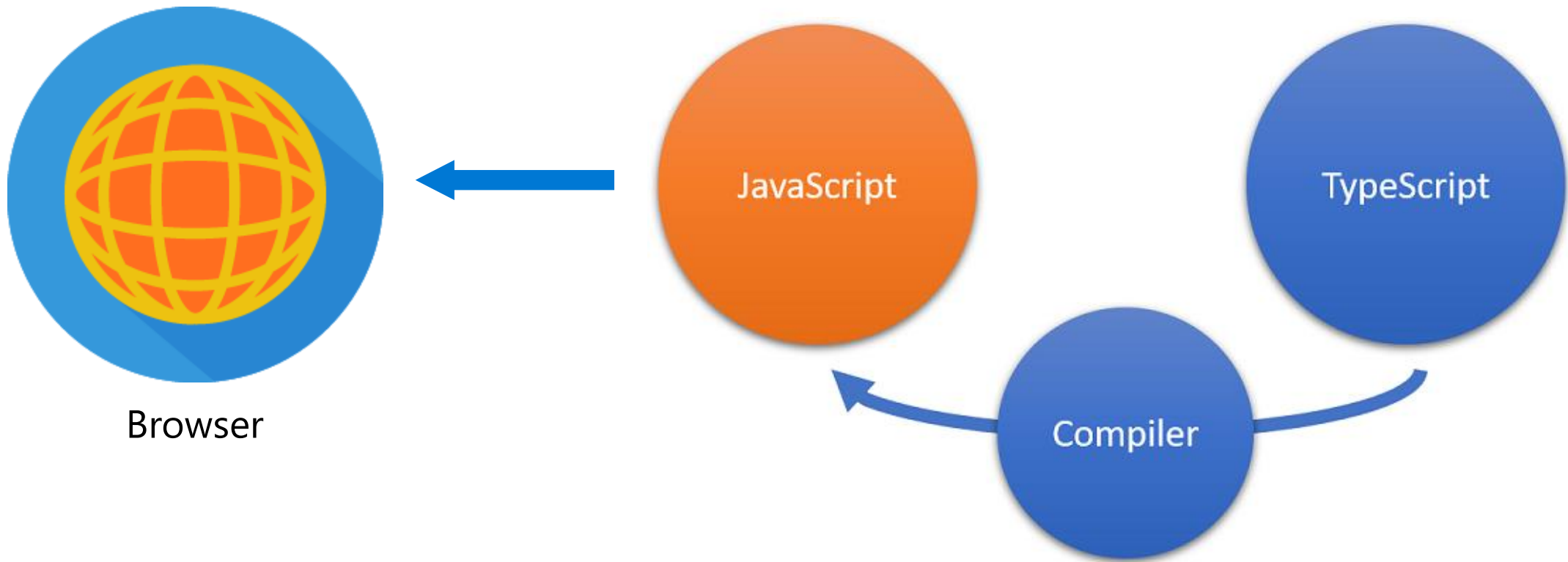
[LOG]: 9

[LOG]: "three6"

[LOG]: 9

# JavaScript, TypeScript & 浏览器

- TypeScript -> 编译 -> JavaScript
- JavaScript -> 给浏览器

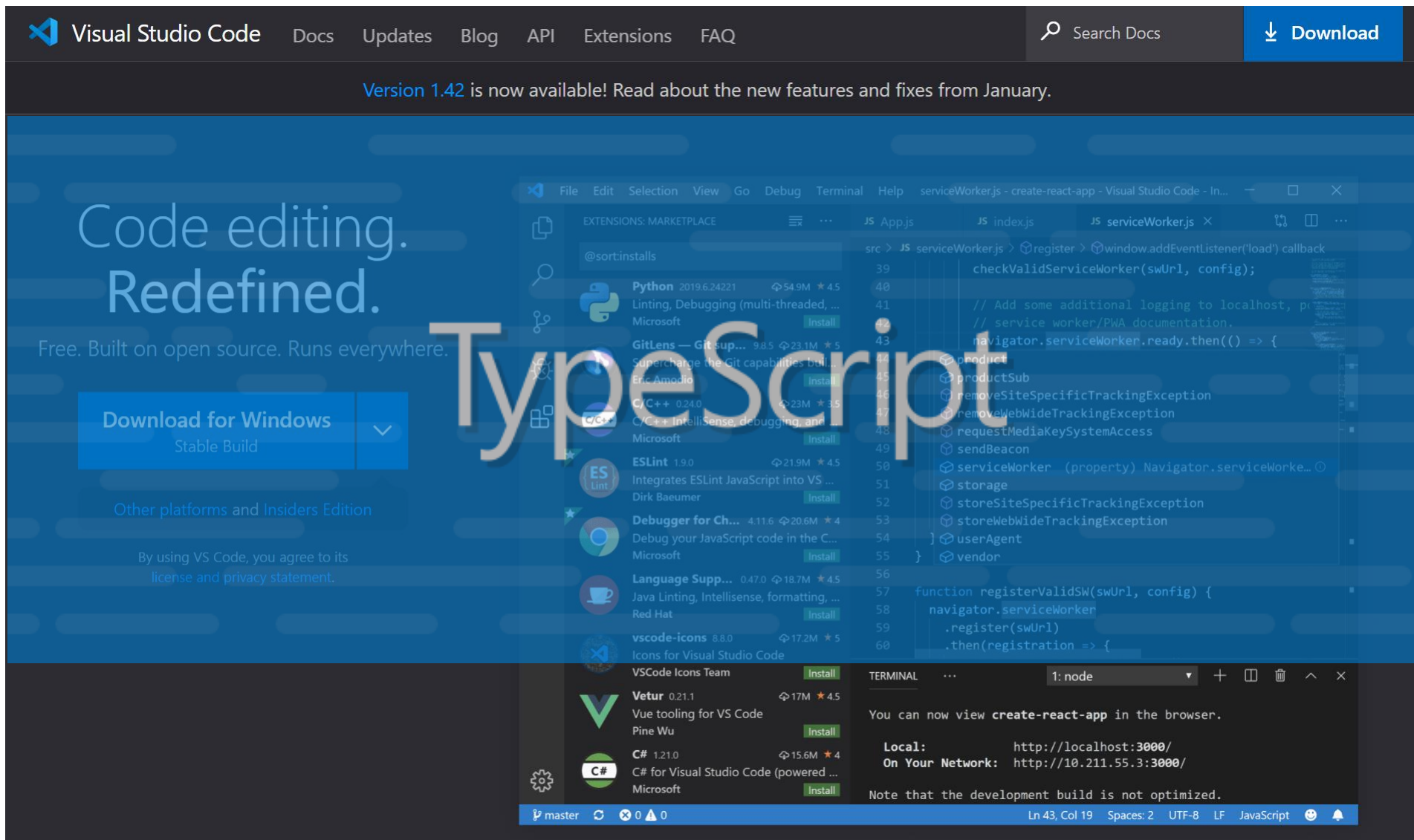


# 本地端安装 TypeScript 编译程序

- 打开命令提示字符
- 输入
  - `npm install -g typescript`
- 检查版本
  - `tsc --version`

# 开发环境

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



<https://code.visualstudio.com/>

17

Microsoft

Reactor

# 第一个TypeScript专案

- 建立文件夹HelloTS
- 新增档案helloworld.ts
- 输入内容

```
let message:string = "Hello World";  
console.log(message);
```

- 执行
  - Ctrl + ~ 带出终端机
  - 输入 tsc helloworld.ts
  - 产生helloworld.js
  - 再输入 node helloworld.js

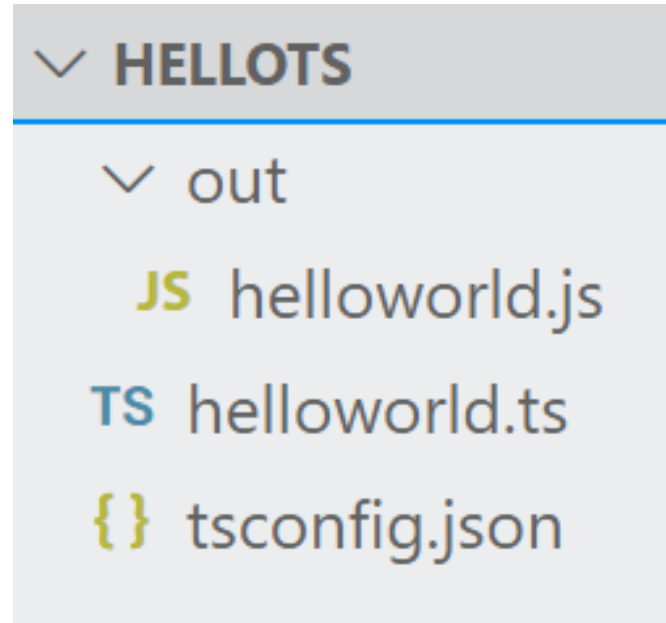


# 增加TypeScript配置文件：tsconfig.json

```
{  
  "compileOnSave": true,  
  "compilerOptions": {  
    "target": "es5",  
    "module": "commonjs",  
    "outDir": "out"  
  }  
}
```

# 将js输出到别的文件夹

- 建立out文件夹
- 终端机执行
  - tsc



## 练习：设置TypeScript专案

- 开启VS Code
- 档案 -> 开启资料夹...
- 新增资料夹 Module 01 Exercise
- 选择资料夹

## 练习：设置TypeScript专案

- 在Module 01 Exercise资料夹中新增档案
  - `module01.ts`
- 检视 -> 终端
  - `tsc --init`
- 检视`tsconfig.json`档案
  - 找到`target`，将`es5`改为`ES2015`  
`"target": "ES2015",`
  - 找到`outDir`，取消注解，设定为`build`  
`"outDir": "build",`
- 在Module 01 Exercise资料夹中新增资料夹
  - `build`
- 在终端机中执行 `tsc` 读取最新`tsconfig.json`设置

# 编译TypeScript至JavaScript

- 编辑module01.ts

```
function addNumbers(x, y){  
    return x + y;  
}  
console.log(addNumbers(3,6));
```

- 此时VS Code已提示错误

```
1 function addNumbers(x, y){  
2     return x + y;  
3 }  
4 console.log(addNumbe
```

(parameter) x: any

參數 'x' 隱含了 'any' 類型。 ts(7006)

檢視問題 (Alt+F8) 快速修復... (Ctrl+.)

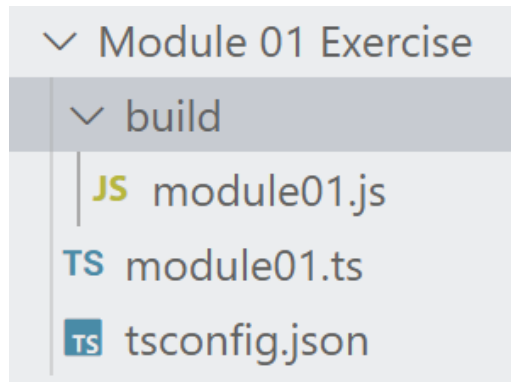


# 编译TypeScript至JavaScript

- 修改module01.ts

```
function addNumbers(x:number, y:number){  
    return x + y;  
}  
console.log(addNumbers(3,6));
```

- 终端机执行 tsc，顺利产生module01.js



# 编译TypeScript至JavaScript

- 观察module01.ts与module01.js
- 点击右上方  向右分割编辑器，将module01.js显示在右方

```
TS module01.ts × tsconfig.json
Module 01 Exercise > TS module01.ts > ...
1 function addNumbers(x:number, y:number){
2     return x + y;
3 }
4 console.log(addNumbers(3,6));
```

```
TS module01.ts × tsconfig.json
Module 01 Exercise > TS module01.ts > ...
1 function addNumbers(x:number, y:number){
2     return x + y;
3 }
4 console.log(addNumbers(3,6));

JS module01.js ×
Module 01 Exercise > build > JS module01.js > ...
1 "use strict";
2 function addNumbers(x, y) {
3     return x + y;
4 }
5 console.log(addNumbers(3, 6));
6
```

# 编译TypeScript至JavaScript

- 执行module01.js
  - 检视 -> 终端
- 确认是否有看到输出结果

9

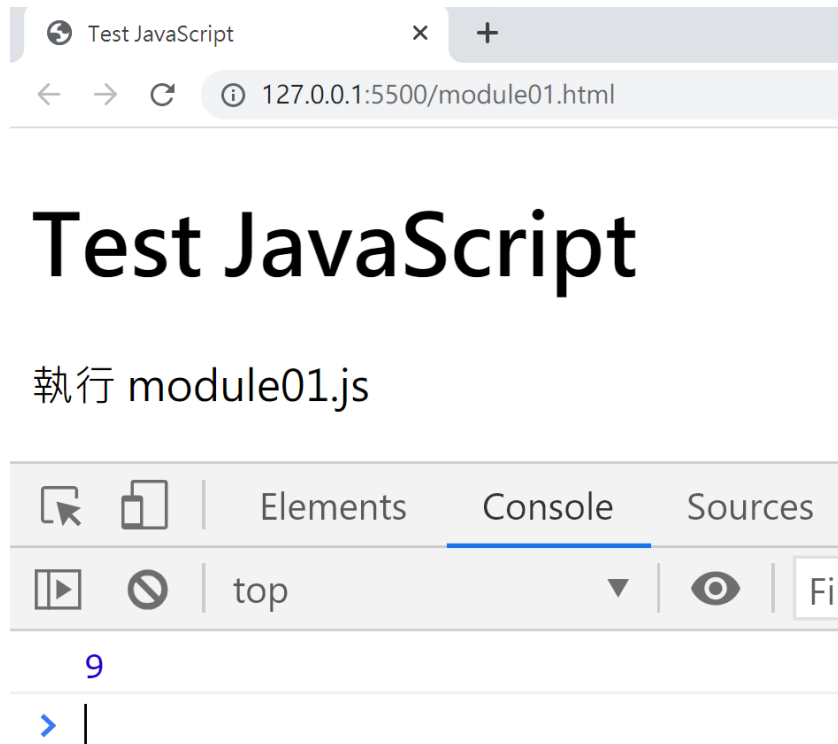
# 透过HTML网页执行JavaScript

- 在Module 01 Exercise资料夹中
  - 建立module01.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Test JavaScript</title>
    <link rel="stylesheet" href="">
  </head>
  <body>
    <h1>Test JavaScript</h1>
    <p id="date"></p>
    <p>执行 module01.js</p>
    <script src="./build/module01.js"></script>
  </body>
</html>
```

# 透过HTML网页执行JavaScript

- 在侧边栏module01.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





# 知识检查

1. TypeScript 和 JavaScript 之间有什么关系？

- ☐ TypeScript 是 JavaScript 的一个超集。
- ☐ TypeScript 与 JavaScript 完全相同。
- ☐ TypeScript 是 JavaScript 的一个子集。

2. 为什么需要先将 TypeScript 代码编译（或转译）为 JavaScript 才能在应用程序中使用？

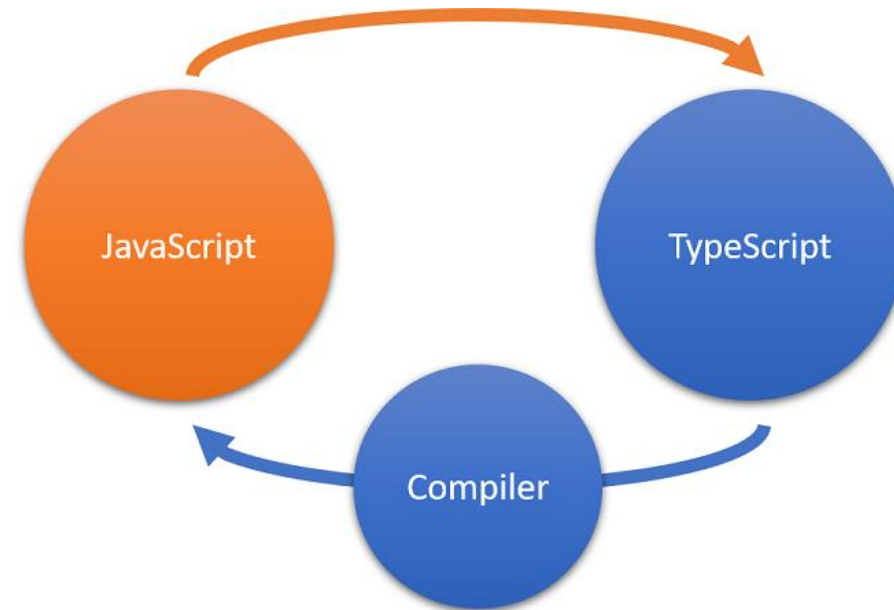
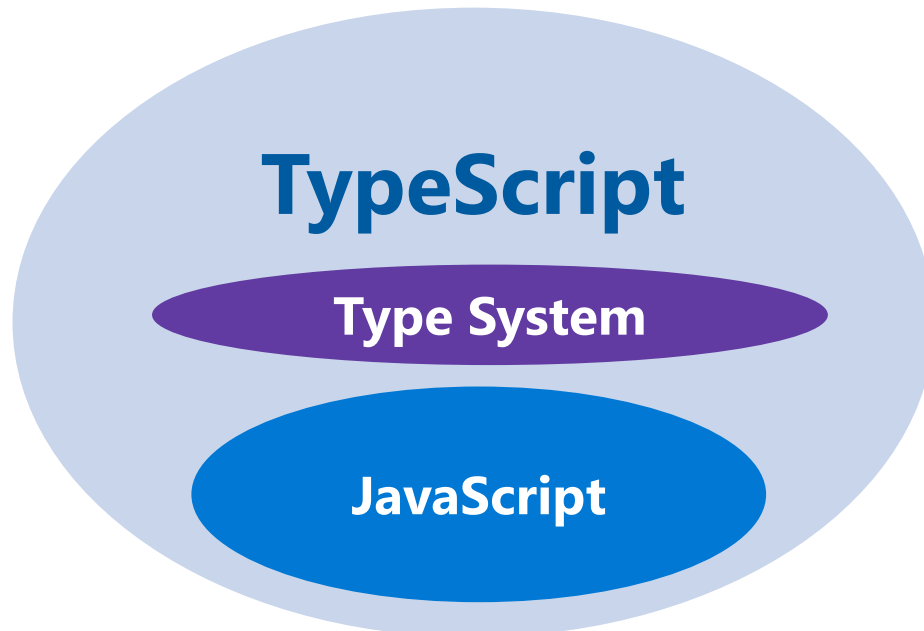
- ☐ 无需执行此操作，因为只需使用 .js 扩展名重命名 TypeScript 文件它就可以正常工作。
- ☐ TypeScript 代码是有效的 JavaScript，而 JavaScript 代码不是有效的 TypeScript。
- ☐ TypeScript 包含与浏览器不兼容的代码功能。

3. `npm install -g typescript` 命令有什么作用？

- ☐ 安装 npm，以便你可以安装 TypeScript。
- ☐ 在计算机上全局安装 TypeScript 编译器。
- ☐ 安装 Node.js 和 npm。

# 摘要

- TypeScript VS. JavaScript
- 安装TypeScript环境
- 使用VS Code开发与编译TypeScript





# 在 TypeScript 中声明变量类型

38 分钟 • 模块 • 10 单元

★★★★★ 4.8 (44)

中级

开发人员

学生

Azure

Visual Studio Code

JavaScript 是一种动态类型的语言。虽然这种语言可以简单地声明变量，但在某些情况下会导致意想不到的结果。通过 TypeScript 中的静态类型系统，你可以描述对象的形状，从而提供更好的文档并允许 TypeScript 验证代码是否正常工作。在 TypeScript 中，命名空间、类、属性、函数、变量和其他语言实体的声明将类型与这些实体相关联。类型形成和与语言实体关联的方式取决于实体的种类。该模块介绍了一些可用的类型，并展示了如何将它们与变量相关联。后面的模块将检查接口、函数和类如何使用静态类型。

## 学习目标

通过学习本模块，你将了解如何：

- 说明在 TypeScript 中声明类型变量的优点。
- 使用基元类型声明变量。
- 使用对象类型声明变量。
- 使用联合和交叉类型声明变量。

## 先决条件

- 熟悉 JavaScript。
- 熟悉在 JavaScript 中使用 `let` 和 `const` 声明变量。
- TypeScript 的基本知识。
- 安装的软件：
  - [Git](#)
  - [Visual Studio Code](#)
  - [Node.js](#)
  - TypeScript

# 学习目标

- 了解在TypeScript中宣告型别的好处
- 原生型别(primitive type)宣告
- 物件型别(object type)宣告
- Union与Intersection型别宣告

## 型别宣告练习

- 打开VS Code，档案 -> 开启资料夹...
- 建立 Module 02 Exercise 资料夹
- 选择资料夹

## 练习：设置TypeScript专案

- 在Module 02 Exercise资料夹中新增档案
  - helloType.ts
- 检视 -> 终端
  - tsc --init
- 检视tsconfig.json档案
  - 找到target，将es5改为ES2015 "target": "ES2015",
  - 找到outDir，取消注解，设定为build "outDir": "build",
- 在Module 02 Exercise资料夹中新增资料夹
  - build
- 在终端机中执行 tsc 读取最新jsconfig.json设置

# 透过HTML网页执行JavaScript

- 在Module 02 Exercise资料夹中
  - 建立module02.html

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

# 透过HTML网页执行JavaScript

- 在侧边栏module02.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



# helloType.ts

- 宣告x为number型态
  - 试着给予x不是数字的值
- y因为给予1而也获得了number型态
  - 试着设定y不是数字的值
- z因为没有初始值，而获得了any型态
  - 试着给予z数字、非数字的值

```
let x:number;  
let y = 1;  
let z;
```

# Type 型態

any

## Primitive types

boolean  
number  
string  
enum  
void

## Object types

class  
interface  
array  
literals

Type  
parameters

null  
undefined

# Primitive type 原生型态

- boolean, number, string, enum
- void
  - 用在标示函数没有回传值
- null, undefined
  - 其他型态的子型态(subtype)

# Primitive type 原生型态

```
let isMale:boolean = true;  
let user_weight:number = 77;  
let user_name = "Ryan";
```

```
enum CurrentStatus {  
    Working,  
    Playing,  
    Sleeping,  
    Eating  
}
```

```
let user_status:CurrentStatus = CurrentStatus.Playing;
```

```
console.log(user_status);
```

1



# Primitive type 原生型态

```
let isMale:boolean = true;  
let user_weight:number = 77;  
let user_name = "Ryan";
```

```
enum CurrentStatus {  
    Working,  
    Playing,  
    Sleeping,  
    Eating  
}
```

```
let user_status:CurrentStatus = CurrentStatus.Playing;
```

```
console.log(CurrentStatus[user_status]);
```

**Playing**



# 列举 enum

- 具有默认顺序性的限定选项

```
enum Days {星期天, 星期一, 星期二, 星期三, 星期四, 星期五, 星期六};
```

```
console.log("您预约的是"+Days[6]);
```

您预约的是星期六

# 枚举 enum

- 有需要也可自行指定

```
enum Days {星期天=7, 星期一=1, 星期二=2, 星期三=3, 星期四=4, 星期五=5, 星期六=6};
```

```
console.log("您预约的是"+Days[7]);
```

您预约的是星期天

# TypeScript世界中的弹性 - any

- 给予不同资料型态也可以

```
let anyTypeValue:any = 10;  
anyTypeValue = "Hello";  
anyTypeValue = true;
```



# TypeScript世界中的弹性 - any

- 甚至这样也不会报错

```
let anyTypeValue:any = 10;  
anyTypeValue = "Hello";  
anyTypeValue = true;
```

```
console.log(anyTypeValue.unknownProperty);  
anyTypeValue();  
anyTypeValue.toUpperCase();
```

# TypeScript世界中的弹性 - any

- 但执行js就出事了...

```
let anyTypeValue:any = 10;  
anyTypeValue = "Hello";  
anyTypeValue = true;
```

```
console.log(anyTypeValue.unknownProperty);  
anyTypeValue();  
anyTypeValue.toUpperCase();
```

```
out\helloType.js:31  
anyTypeValue();  
^
```

```
TypeError: anyTypeValue is not a function  
at Object.<anonymous>
```

# TypeScript 新兵报到 – 弹性中带有原则 Unknown

- 跟any一样，这样写是可以的

```
let unknownValue:unknown = 10;  
unknownValue = "Hello";  
unknownValue = true;
```

# TypeScript 新兵报到 – 弹性中带有原则 Unknown

- 这样写就不允许了

```
let unknownValue:unknown = 10;  
unknownValue = "Hello";  
unknownValue = true;
```

```
console.log(unknownValue.unknownProperty);  
unknownValue();  
unknownValue.toUpperCase();
```

```
let unknownValue: unknown
```

```
物件的類型為 '未知'。 ts(2571)
```

# 如果你真的非常确定，你可以这么干... (Type assertion)

```
let unknownValue: unknown = 10;  
unknownValue = "Hello";  
//unknownValue = true;  
  
//console.log(unknownValue.unknownProperty);  
//unknownValue();  
console.log((unknownValue as string).toUpperCase());
```

HELLO



## 也可以这样子

```
let unknownValue:unknown = 10;  
unknownValue = "Hello";  
//unknownValue = true;  
  
//console.log(unknownValue.unknownProperty);  
//unknownValue();  
console.log((unknownValue as string).toUpperCase());  
console.log(<string>unknownValue.toLowerCase());
```

HELLO  
hello



## 如果还是不放心，可以这么写

```
let unknownValue:unknown = 10;  
unknownValue = "Hello";  
//unknownValue = true;  
  
//console.log(unknownValue.unknownProperty);  
//unknownValue();  
if(typeof unknownValue === 'string'){  
    console.log((unknownValue as string).toUpperCase());  
    console.log(<string>unknownValue.toLowerCase());  
}else{  
    console.log("It's not a string");  
}
```

HELLO  
hello



# 万一不是字符串，依然可以执行

```
let unknownValue: unknown = 10;  
//unknownValue = "Hello";  
unknownValue = true;  
  
//console.log(unknownValue.unknownProperty);  
//unknownValue();  
if(typeof unknownValue === 'string'){  
    console.log((unknownValue as string).toUpperCase());  
    console.log(<string>unknownValue.toLowerCase());  
}else{  
    console.log("It's not a string");  
}
```

It's not a string





# 指定多种输入数据类型(联合 Union)

- 允许字符串或字符串数组作为输入值

```
function getLength(obj: string | string[]){  
    return obj.length;  
}
```

```
console.log(getLength("Hello"));  
console.log(getLength(["David", "John", "Ryan"]));
```

5

3

# 指定多种输入数据型别(联合 Union)

- 允许字符串或字符串数组作为输入值
- 对应产生不同回应

```
function getLength(obj: string | string[]){  
  if(typeof obj === "string"){  
    return "来了一个勇者，叫做"+obj;  
  }else{  
    return "对方来了"+obj.length+"个人";  
  }  
}
```

```
console.log(getLength("王小明"));  
console.log(getLength(["张三", "李四", "王五"]));
```

来了一个勇者，叫做王小明  
对方来了3个人

# 指定多种输入数据型别(联合 Union)

- 允许字符串或数字作为输入值
- 对应产生不同回应

```
function getNumber(obj: number | string){  
  if(typeof obj === "string"){  
    return "国字的"+obj;  
  }else{  
    return `${obj}+3=${obj+3}`  
  }  
}
```

```
console.log(getNumber("七"));  
console.log(getNumber(7));
```

国字的七  
 $7 + 3 = 10$

# 指定多种输入数据类型别(交叉 Intersection)

- 该型态则拥有所有的属性

```
type BasicProfile = {  
  name:string;  
  age:number;  
}  
  
type ExtraProfile = {  
  education:string;  
  work_experience:number;  
}  
  
type FullProfile = BasicProfile & ExtraProfile;  
  
let newbie1_full_profile:FullProfile = {  
  name:"Ryan",  
  age:99,  
  education:"Master",  
  work_experience:15  
}
```

# 自定义数据类型(literal type)

- 输入时会有选择效果

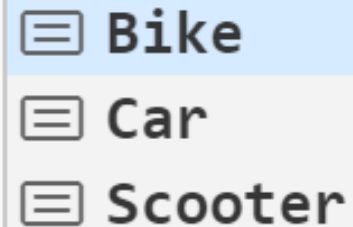
```
type trafficTools = "Bike" | "Car" | "Scooter";
```

```
let myTrafficTool:trafficTools = "Bike";
```

```
console.log(myTrafficTool);
```

```
let myTrafficTool:trafficTools = ""
```

```
console.log(myTrafficTool);
```



Bike  
Car  
Scooter

# 自定义数据型态(literal type)

- 乱打会被发现...

```
type trafficTools = "Bike" | "Car" | "Scooter";
```

```
let myTrafficTool:trafficTools = "Boat";
```

```
console.log(myTrafficTool);
```

```
let myTrafficTool: trafficTools
```

```
類型 '"Boat"' 不可指派給類型 'trafficTools'。 ts(2322)
```

# 自定义数据型态(literal type)

- 数字也可以

```
type dice = 1|2|3|4|5|6;
```

```
let diceRoll:dice;
```

```
diceRoll = 6;
```

# 指定数据型别的数组

## • Array<elemType>

```
type StringArray = Array<String>;  
type NumberArray = Array<number>;  
type ObjectWithNameArray = Array<{name:string}>;
```

```
let className:StringArray = ["HTML", "CSS", "JavaScript", "TypeScript"];  
let audienceNumber:NumberArray = [666,777,888,999];  
let instructors:ObjectWithNameArray = [  
    {name:"Ryan"}, {name:"David"}, {name:"John"}, {name:"Marry"}  
];
```

```
console.log(`在${className[0]}课中，有${audienceNumber[0]}人参与，讲师是  
${instructors[0].name}`);
```

在HTML课中，有666人参与，讲师是Ryan



# 指定数据型别的数组

- 也可写成

```
type StringArray = string[];  
type NumberArray = number[];  
type ObjectWithNameArray = {name:string}[];
```

```
let className:StringArray = ["HTML", "CSS", "JavaScript", "TypeScript"];  
let audienceNumber:NumberArray = [666,777,888,999];  
let instructors:ObjectWithNameArray =  
    [{name:"Ryan"},{name:"David"},{name:"John"},{name:"Marry"}];  
  
console.log("在"+className[0]+"课中，有"+audienceNumber[0]+"人参与，讲师是"+  
    instructors[0].name);
```

在HTML课中，有666人参与，讲师是Ryan

# 多重数据型别的数组

- 指定需要的数据型别

```
type multiTypeArray = Array<String|number>;  
let myMultiTypeArray:multiTypeArray = [1,2,3,"one","two","three"];
```

# 多重数据型别的数组

- 指定需要的数据型别

```
type multiTypeArray = Array<String|number>;  
let myMultiTypeArray:multiTypeArray = [1,2,3,"one","two","three"];
```

- 不过乱写还是会被揪出来...

```
type multiTypeArray = Array<String|number>;  
let myMultiTypeArray:multiTypeArray = [1,2,3,"one","two","three"];  
let myMultiTypeArray2:multiTypeArray = [1,2,3,"one","two","three",true];
```

類型 'boolean' 不可指派給類型 'number | String'。 ts(2322)

# 多重数据型别的数组

- 直接赋予值可以这样写

```
let thisNameAndAge2:Array<String|number> = ['Jessica',18];
```

# 具有顺序的多重数据类型数组(Tuple)

- 使用时记得按照顺序赋予值

```
type orderedMultiTypeArray = [string, number];  
let thisNameAndAgeWithOrder:orderedMultiTypeArray = ["Luis",80];
```

- 直接赋予值可以这样写

```
let thisNameAndAgeWithOrder2:[string,number] = ['Ryan',99];
```

## 综合练习

- 下载档案

`git clone https://github.com/MicrosoftDocs/mslearn-typescript`

- 开启练习资料夹

- `mslearn-typescript/code/module-02/m02-start`

- 开启练习档案

- `module02.ts`

# Exercise 1

- 在宣告处增加资料型别

# Exercise 1

- 在宣告处增加资料型别

```
let firstName:string;  
let lastName:string;  
let fullName:string;  
let age:number;  
let ukCitizen:boolean;
```

```
firstName = 'Rebecca';  
lastName = 'Smith';  
age = 42;  
ukCitizen = false;  
fullName = firstName + " " + lastName;
```

```
if (ukCitizen) {  
    console.log("My name is " + fullName + ", I'm " + age + ", and I'm a citizen of the United Kingdom.");  
} else {  
    console.log("My name is " + fullName + ", I'm " + age + ", and I'm not a citizen of the United Kingdom.");  
}
```



## Exercise 2

- 修正程式码与数值，让a顺利产生12

## Exercise 2

- 修正程式码与数值，让a顺利产生12

```
let x:number;  
let y:number;  
let a:number;
```

```
x = 5;  
y = 7;  
a = x + y;
```

```
console.log(a);
```

## Exercise 3

- 建立enum，并修改使用enum作为函数输入

# Exercise 3

- 建立enum，并修改使用enum作为函数输入

```
enum Season{  
  "Fall",  
  "Winter",  
  "Spring",  
  "Summer"  
};
```

```
function whichMonths(season: Season){  
  let monthsInSeason: string;  
  switch (season) {  
    case Season.Fall:  
      monthsInSeason = "September to November";  
      break;  
    case Season.Winter:  
      monthsInSeason = "December to February";  
      break;  
    case Season.Spring:  
      monthsInSeason = "March to May";  
      break;  
    case Season.Summer:  
      monthsInSeason = "June to August";  
  }  
  return monthsInSeason;  
}
```

```
console.log(whichMonths(Season.Fall));
```

## Exercise 4

- 将randomNumbers宣告为数字数组
- 将nextNumber宣告为数字

## Exercise 4

- 将randomNumbers宣告为数字数组
- 将nextNumber宣告为数字

```
let randomNumbers: Array<number> = [];  
let nextNumber: number;  
  
for (let i = 0; i < 10; i++) {  
    nextNumber = Math.floor(Math.random() * (100 - 1)) + 1;  
    randomNumbers.push(nextNumber);  
}  
  
console.log(randomNumbers);
```

# 知识检查

1. `boolean`、`number`、`string` 和 `enum` 类型是 `any` 的哪一类子类型的示例？

- ☐ 类型参数。
- ☐ 对象类型。
- ☐ 基元类型。

2. 以下哪种类型是对象类型的示例？

- ☐ `Array`。
- ☐ `void`。
- ☐ 类型参数。

3. `any` 和 `unknown` 类型之间的主要区别是什么？

- ☐ 可以将任何值赋给 `unknown`，但是 `any` 类型有一些约束。
- ☐ 可以访问 `unknown` 类型的属性，但不能访问 `any` 类型的属性。
- ☐ 可以访问 `any` 类型的属性，但不能访问 `unknown` 类型的属性。

## 知识检查

4. TypeScript 中告诉编译器“我知道我在做什么吗”的功能叫什么？

- ☐ 字面量收缩。
- ☐ 类型断言。
- ☐ 类型保护。

5. 什么是元组？

- ☐ 具有无限数量的相同类型元素的数组。
- ☐ 具有特定数量的相同类型元素的数组。
- ☐ 具有特定数量的一种或多种类型元素的数组。



# 摘要

- 变数宣告静态型别的好处
- Primitive Type
  - boolean, number, string, enum
- Object Type
  - class, interface, array, literal
- 多重资料型别
  - Union Type(|)
  - Intersection Type(&)
  - literal
  - Tuple





# Reactor



[developer.microsoft.com/reactor/](https://developer.microsoft.com/reactor/)  
[@MSFTReactor](#) on Twitter

# 议程结束 感谢聆听



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

© 2019 Microsoft Corporation. All rights reserved. The text in this document is available under the Creative Commons Attribution 3.0 License, additional terms may apply. All other content contained in this document (including, without limitation, trademarks, logos, images, etc.) are not included within the Creative Commons license grant. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. Some examples are for illustration only and are fictitious. No real association is intended or inferred. Microsoft makes no warranties, express or implied, with respect to the information provided here.