Microsoft

Web Development Fundamentals

网站开发系列 - TypeScript 基础

Sep 2020
Microsoft Reactor | Ryan Chung

Microsoft

Ryan Chung

Instructor / DevelopIntelligence
Founder / MobileDev.TW

@ryanchung403 on WeChat
Ryan@MobileDev.TW

Microsoft

Reactor

developer.microsoft.com/reactor/

@MSFTReactor on Twitter

Web On-line Workshop agenda 网页开发在线研讨会议程

Intro to TypeScript 网页开发入门 – TypeScript

19:30	Welcome 开场
19:35	Introduction to TypeScript TypeScript 基本介绍
19:55	Installing TypeScript 开发环境安装与设定
20:15	Interfaces & Specified Data Types 界面与指定型别
20:30	5-minute break 中场休息
20:35	Running your TypeScript web app 网站综合练习
21:00	Event end 研讨会结束

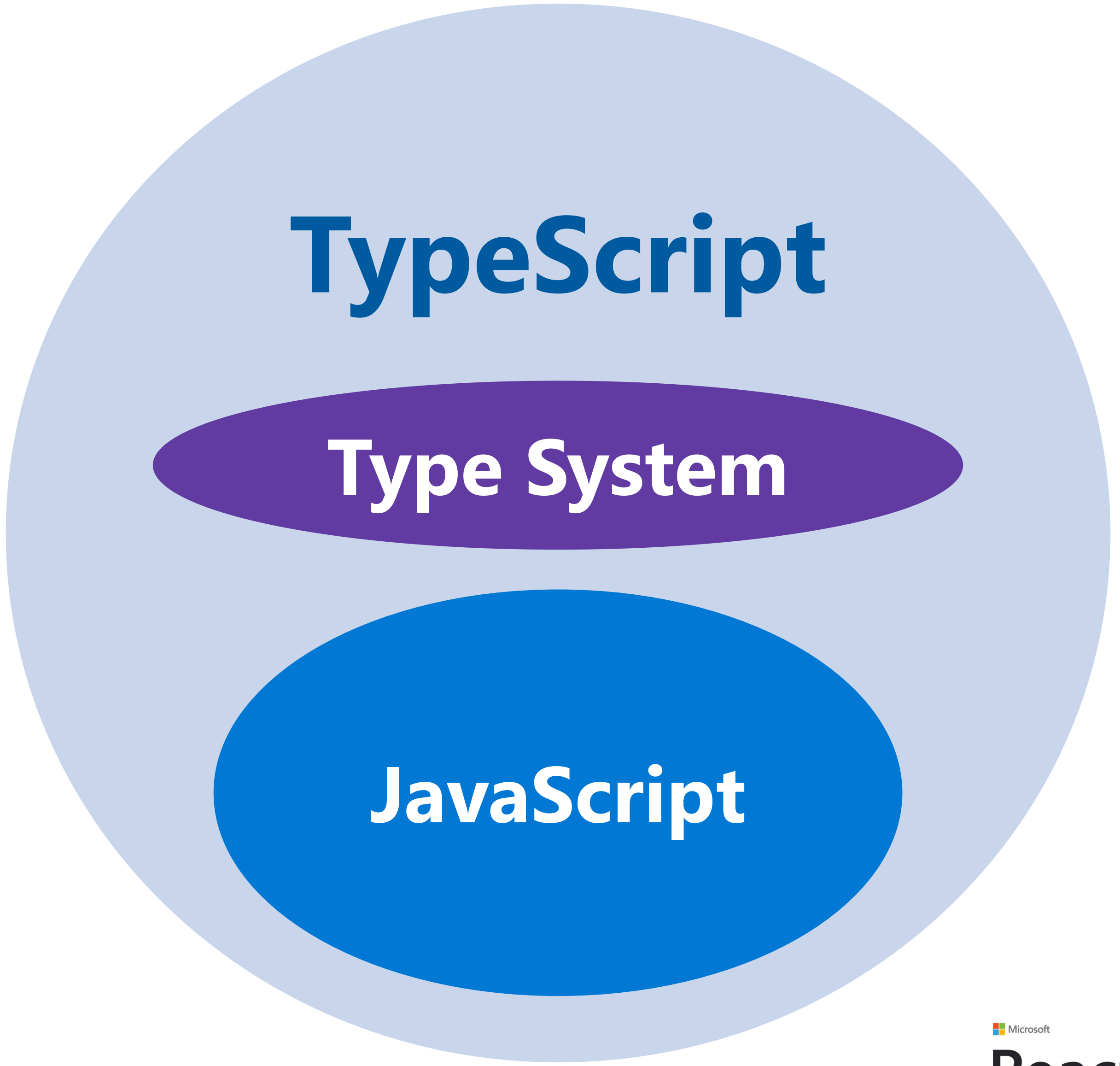
Reactor

Microsoft

TypeScript 简介

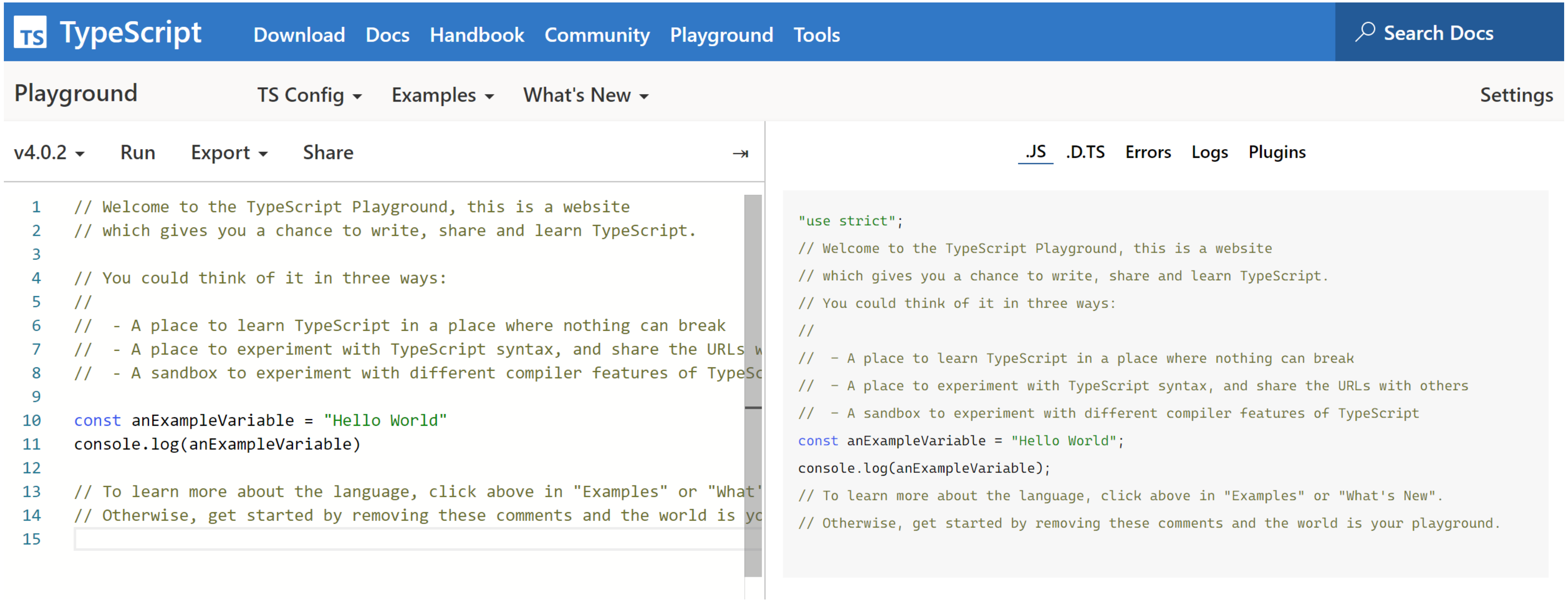
概要

- 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



在线执行语法测试 – TypeScript Playground

- 左边编辑后按下Run，右边观察结果



```
// 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 w
//  - A sandbox to experiment with different compiler features of TypeSc

const anExampleVariable = "Hello World"
console.log(anExampleVariable)

// To learn more about the language, click above in "Examples" or "What'
// Otherwise, get started by removing these comments and the world is yo

```

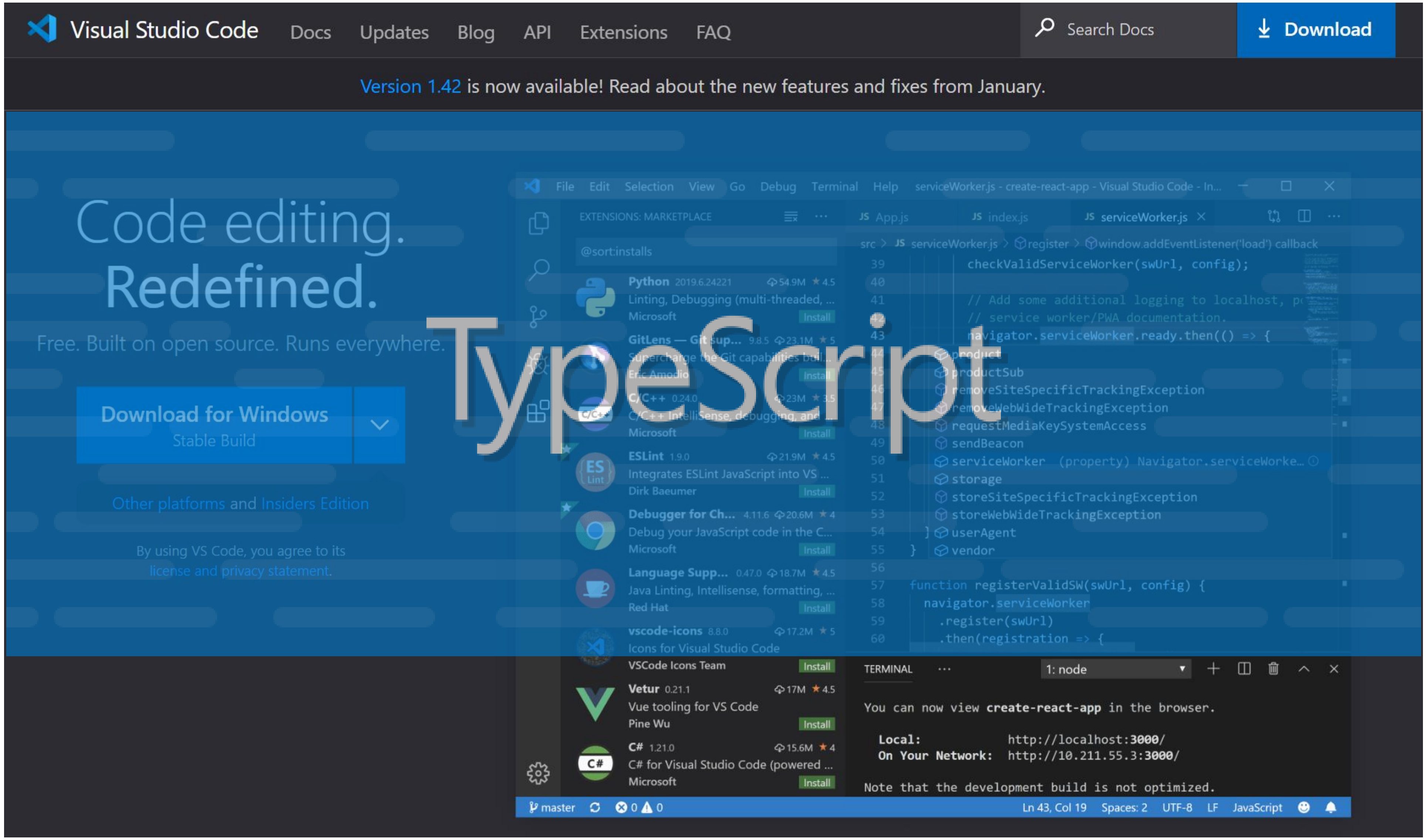
```
"use strict";
// Welcome to the TypeScript Playground, this is a website
// which gives you a chance to write, share and learn TypeScript.
// You could think of it in three ways:
//
//  - A place to learn TypeScript in a place where nothing can break
//  - A place to experiment with TypeScript syntax, 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/

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



Version 1.42 is now available! Read about the new features and fixes from January.

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Other platforms and Insiders Edition

By using VS Code, you agree to its
license and privacy statement.

Python 2019.6.24221 54.9M ★4.5
Linting, Debugging (multi-threaded, ...
Microsoft Install

GitLens — Git sup... 9.8.5 23.1M ★5
Supercharge the Git capabilities buil...
Eric Amodio Install

C/C++ 0.24.0 23M ★3.5
C/C++ IntelliSense, debugging, and ...
Microsoft Install

ESLint 1.9.0 21.9M ★4.5
Integrates ESLint JavaScript into VS ...
Dirk Baeumer Install

Debugger for Ch... 4.11.6 20.6M ★4
Debug your JavaScript code in the C...
Microsoft Install

Language Supp... 0.47.0 18.7M ★4.5
Java Linting, Intellisense, formatting, ...
Red Hat Install

vscode-icons 8.8.0 17.2M ★5
Icons for Visual Studio Code
VSCode Icons Team Install

Vetur 0.21.1 17M ★4.5
Vue tooling for VS Code
Pine Wu Install

C# 1.21.0 15.6M ★4
C# for Visual Studio Code (powered ...
Microsoft Install

```
    checkValidServiceWorker(swUrl, config);

    // Add some additional logging to localhost, p
    // service worker/PWA documentation.
    navigator.serviceWorker.ready.then(() => {
      product
      productSub
      removeSiteSpecificTrackingException
      removeWebWideTrackingException
      requestMediaKeySystemAccess
      sendBeacon
      serviceWorker  (property) Navigator.serviceWorke...
      storage
      storeSiteSpecificTrackingException
      storeWebWideTrackingException
    } userAgent
  }   vendor

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
```

```
You can now view create-react-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.211.55.3:3000/

Note that the development build is not optimized.
```

https://code.visualstudio.com/

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);
```

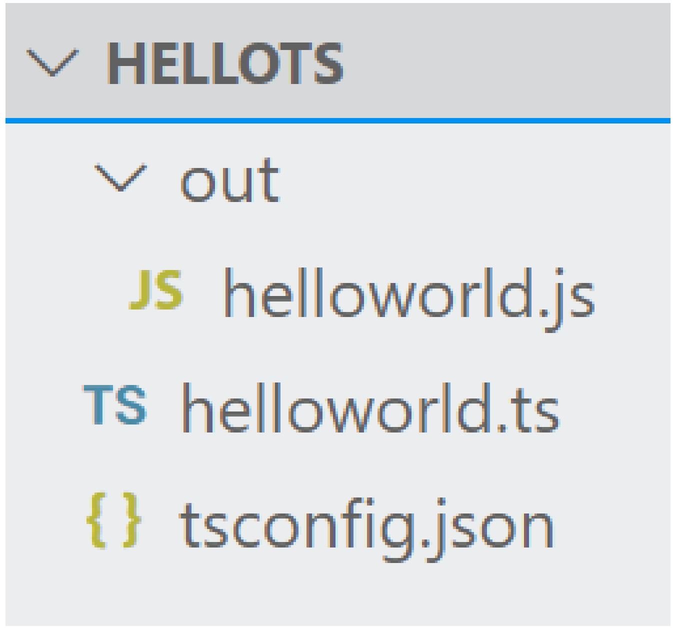
- 执行
  - Ctrl + ~ 带出终端机
  - 输入 tsc helloworld.ts
  - 产生helloworld.js
  - 再输入 node helloworld.js

增加TypeScript配置文件：tsconfig.json

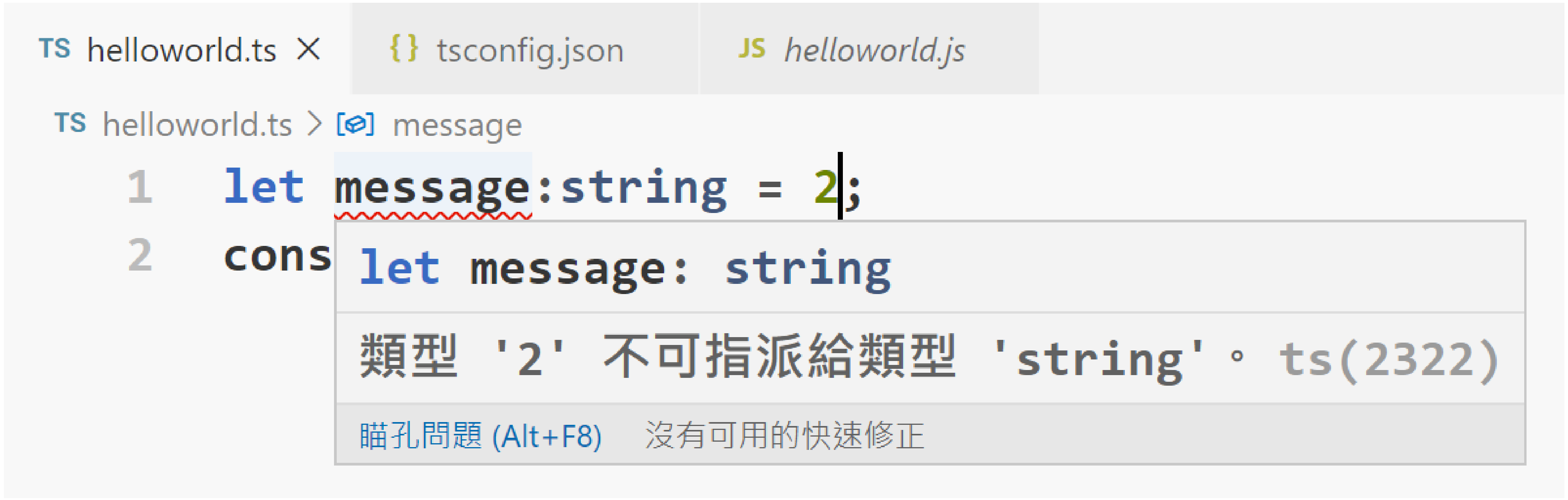
```json
{
    "compileOnSave": true,
    "compilerOptions": {
        "target": "es5",
        "module": "commonjs",
        "outDir": "out"
    }
}
```

将js输出到别的文件夹

- 建立out文件夹
- 终端机执行
  - tsc

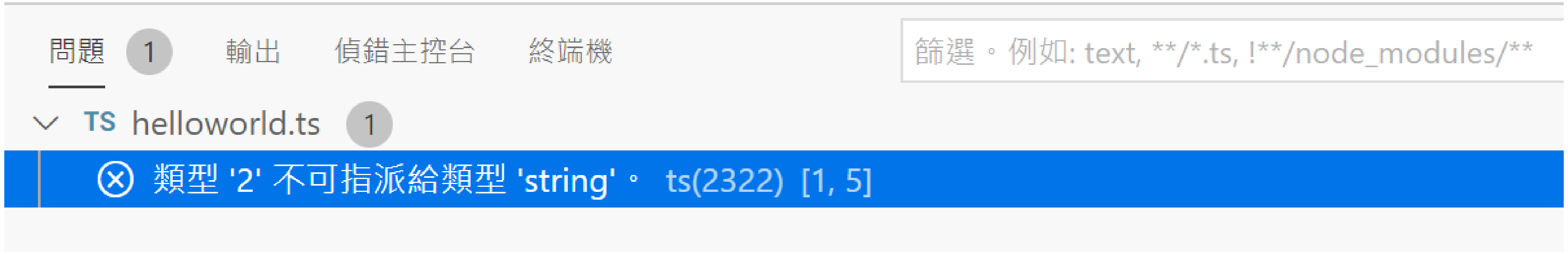


温馨提示：数据型态错误



```
let message:string = 2;
cons
```

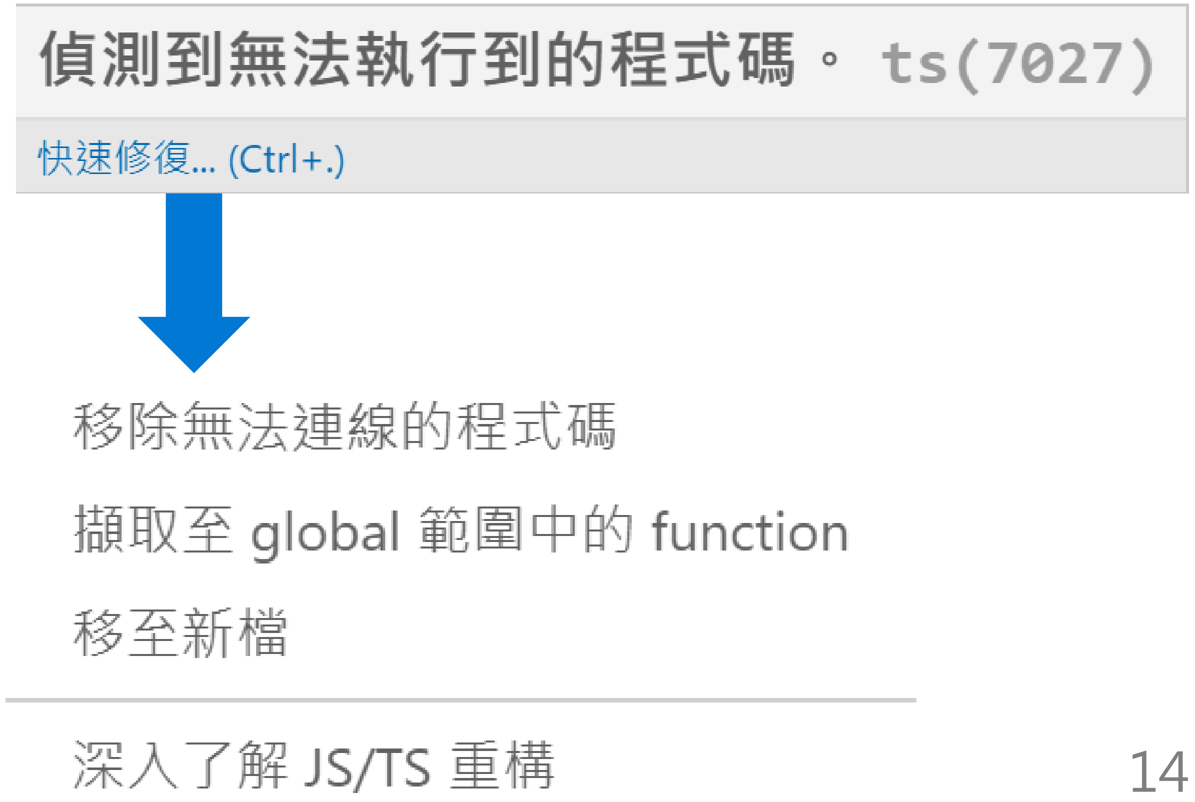
```
let message: string
類型 '2' 不可指派給類型 'string'。 ts(2322)
```



```
類型 '2' 不可指派給類型 'string'。 ts(2322) [1, 5]
```

温馨提示：你创造了一个无人可达的境地...

```
let message:string = "Hello World";
if(false){
    message="never happen";
}
console.log(message);
```


偵測到無法執行到的程式碼。 ts(7027)

快速修復... (Ctrl+.)

移除無法連線的程式碼

擷取至 global 範圍中的 function

移至新檔

深入了解 JS/TS 重構

TypeScript Debugging

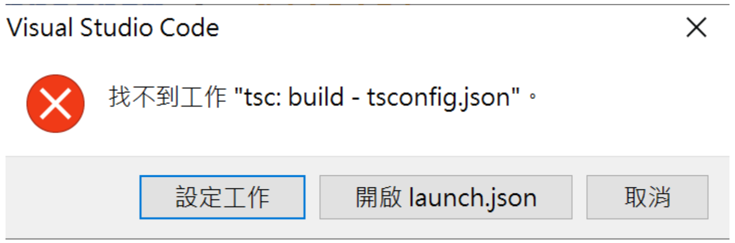
- 修改 tsconfig.json

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

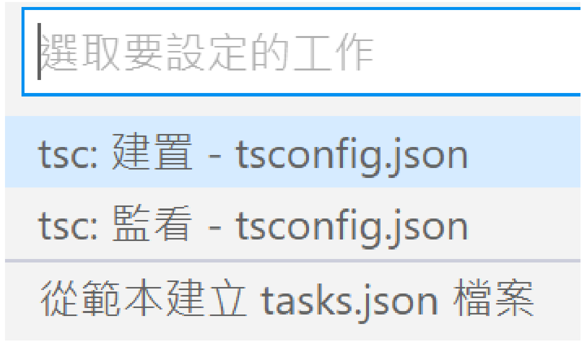
- 再次于终端机执行 tsc，输出helloworld.js.map

侦错测试

- 上方选单 -> 执行 -> 启动侦错 -> 选择 Node.js
- 设定工作 tsc: 建置 – tsconfig.json
- 左边选单第四个(执行)->建立launch.json 档案->Node.js
- 执行 -> 启动侦错 -> 设定工作 ->tsc:建置-tsconfig.json
- 修改launch.json的preLaunchTask跟tasks.json的label相符
  - 两边都改成 tsc-build
- 执行 -> 启动侦错


Visual Studio Code

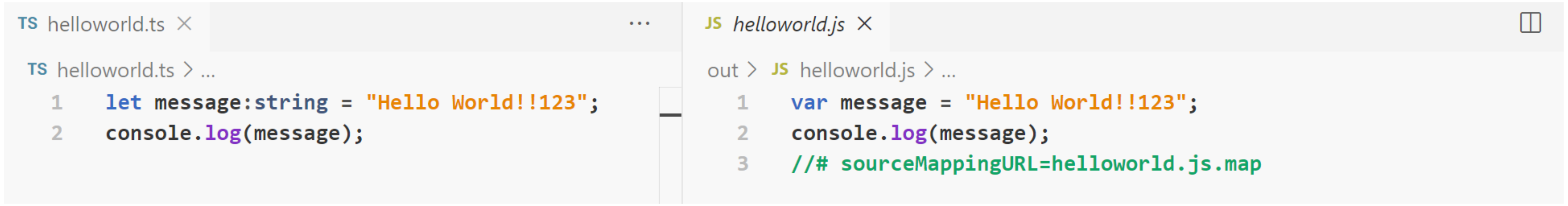
找不到工作 "tsc: build - tsconfig.json"。


選取要設定的工作
tsc: 建置 - tsconfig.json
tsc: 監看 - tsconfig.json
從範本建立 tasks.json 檔案

```
"preLaunchTask": "tsc-build"
```

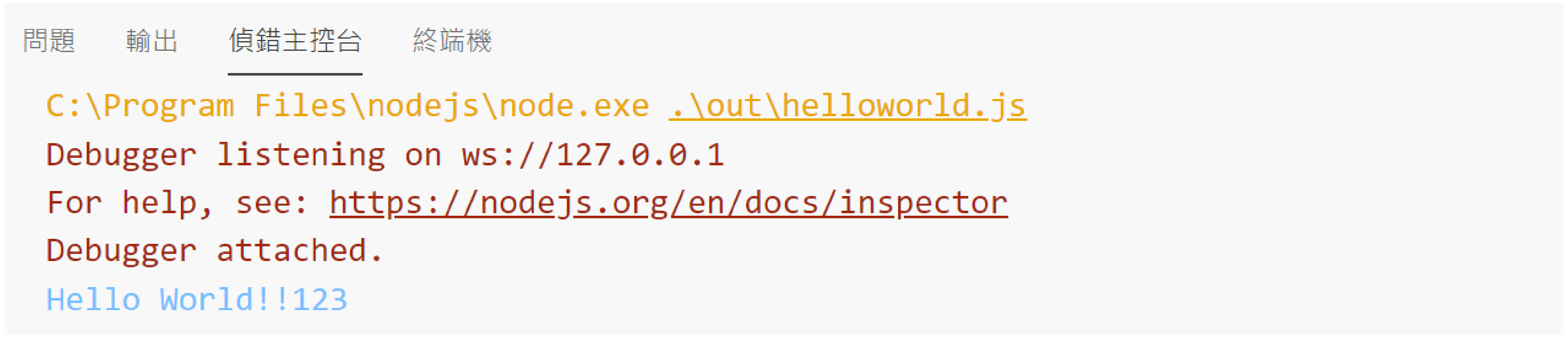
同时开启ts与js进行测试

- 观察Console输出



```
let message:string = "Hello World!!123";
console.log(message);
```

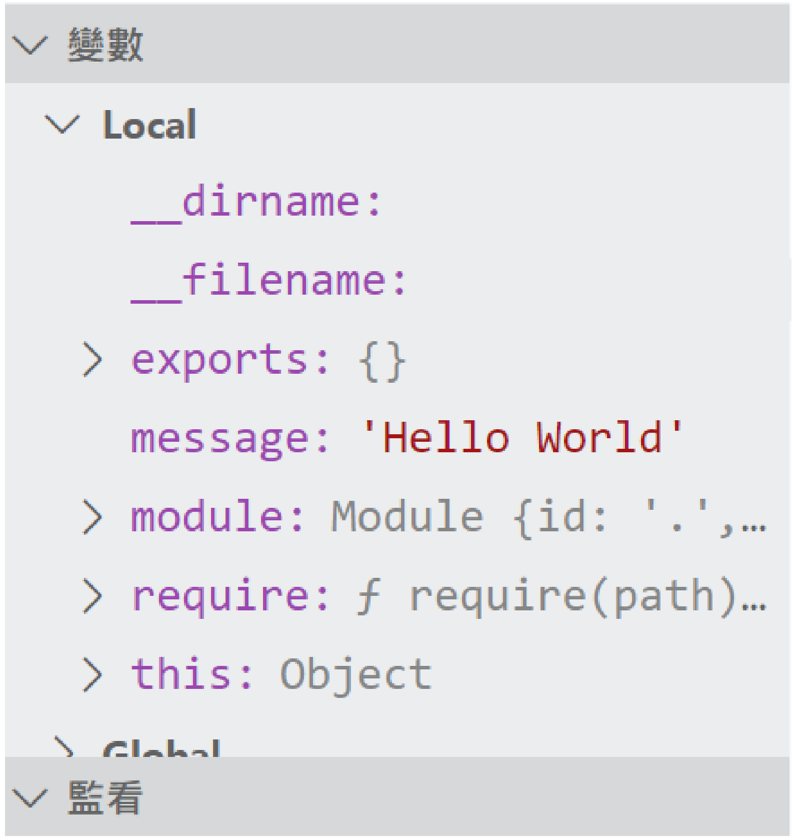
```
var message = "Hello World!!123";
console.log(message);
//# sourceMappingURL=helloworld.js.map
```



```
C:\Program Files\nodejs\node.exe .\out\helloworld.js
Debugger listening on ws://127.0.0.1
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
Hello World!!123
```

设置断点

- 打开helloworld.ts
- 在第2行行号前点一下出现红点
- 执行 -> 启动侦错
- 观察左上角变量内容

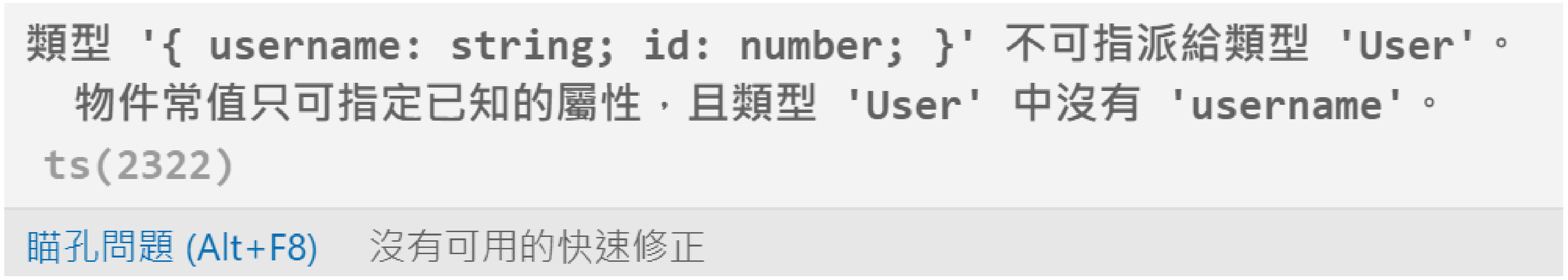


```
∨ 變數
  ∨ Local
      __dirname:
      __filename:
    > exports: {}
      message: 'Hello World'
    > module: Module {id: '.',…
    > require: ƒ require(path)…
    > this: Object
  > Global
∨ 監看
```

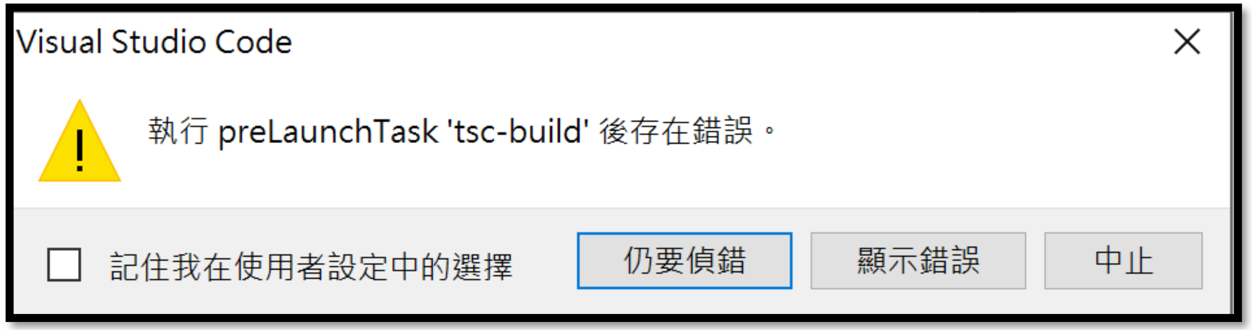
自定义数据型态与检查机制

- 自动检查是否输入了错误的数据结构

```
interface User{
      name:String;
      id:number
}

const user:User = {
      username:"Ryan",
      id:666
};
```



類型 '{ username: string; id: number; }' 不可指派給類型 'User'。
物件常值只可指定已知的屬性，且類型 'User' 中沒有 'username'。
ts(2322)



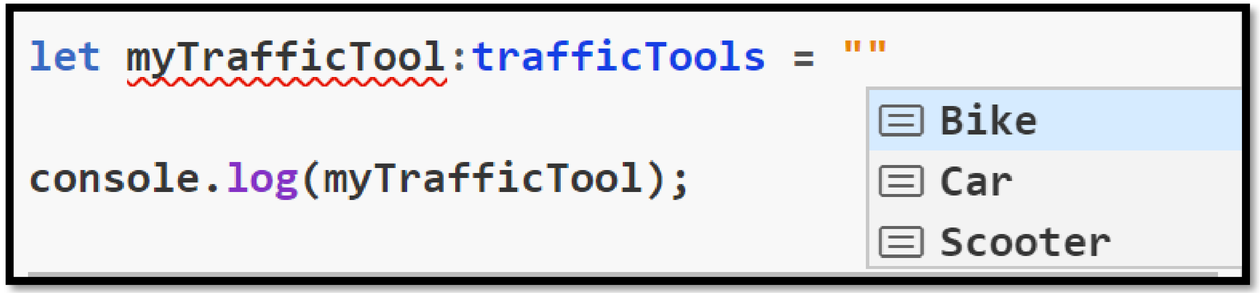
Visual Studio Code

執行 preLaunchTask 'tsc-build' 後存在錯誤。

自定义数据型态

- 输入时会有选择效果

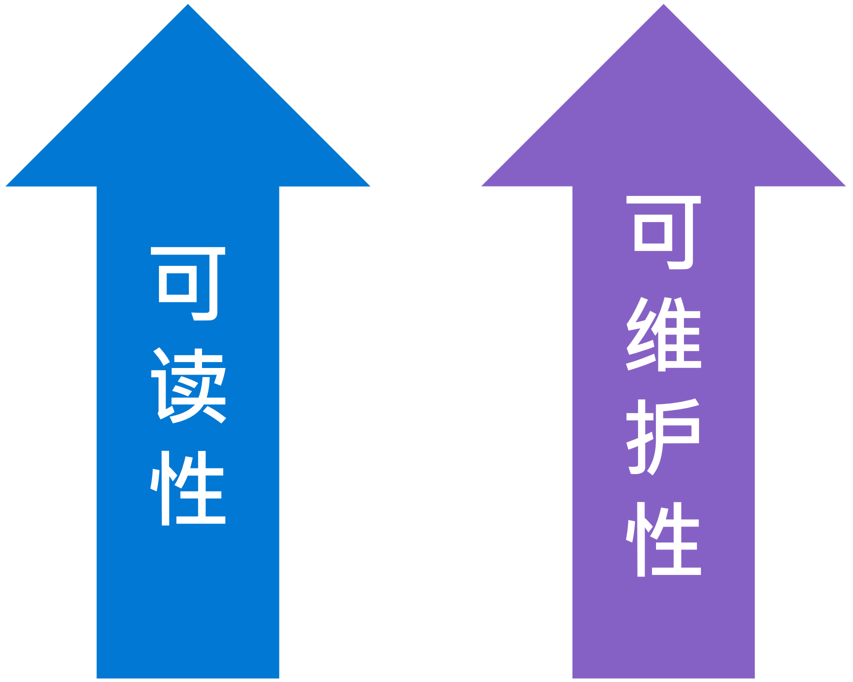
```
type trafficTools = "Bike" | "Car" | "Scooter";

let myTrafficTool:trafficTools = "Bike";

console.log(myTrafficTool);
```



```
let myTrafficTool:trafficTools = ""

console.log(myTrafficTool);
```

TypeScript的功用

- 更多的型态支持
- 及早发现潜在的错误
- 严谨、不含糊
- 提前应用新语法



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

```
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
```

Microsoft Reactor

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
```

Microsoft Reactor

自动数据型别比对 (型别推论)

- thisPoint没有指定数据型别，但比对后与printPoint要的相符

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

function printPoint(p:Point){
    console.log(p.x+","+p.y);
}

const thisPoint = {x:12, y:26};
printPoint(thisPoint);
```

```
12, 26
```

自动数据型别比对(型别推论)

- 有额外元素没关系，但必要项目都存在即可

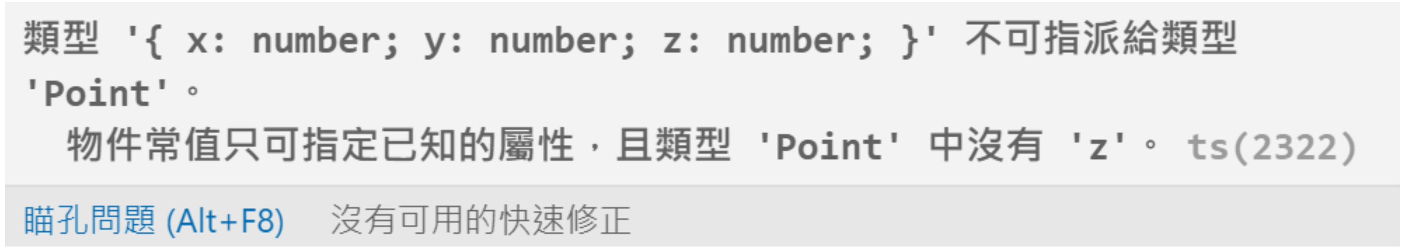
```
interface Point{
    x:number;
    y:number;
}

function printPoint(p:Point){
    console.log(p.x+","+p.y);
}

const threePoint = {x:12, y:26, z:89};
printPoint(threePoint);
```

```
12, 26
```

自动数据型别比对(型别推论)

- 但threePoint如果一开始宣告就说自己是遵循Point界面则会报错

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

function printPoint(p:Point){
     console.log(p.x+","+p.y);
}

const threePoint:Point = {x:12, y:26, z:89};

printPoint(threePoint);
```



類型 '{ x: number; y: number; z: number; }' 不可指派給類型 'Point'。
  物件常值只可指定已知的屬性，且類型 'Point' 中沒有 'z'。 ts(2322)

瞄孔問題 (Alt+F8)　沒有可用的快速修正

```
12, 26
```

自动数据型别比对(型别推论)

- 如果有需要也可以修改Point接口，增加一个Optional的属性

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

function printPoint(p:Point){
    console.log(p.x+","+p.y);
}

const threePoint:Point = {x:12, y:26, z:89};

printPoint(threePoint);
```

不会报错!

```
12, 26
```

Microsoft Reactor

自动数据型别比对(型别推论)

- 原本只有x, y的thisPoint也可正常宣告

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

function printPoint(p:Point){
    console.log(p.x+","+p.y);
}

const thisPoint:Point = {x:12, y:26};
printPoint(thisPoint);

const threePoint:Point = {x:12, y:26, z:89};
printPoint(threePoint);
```

```
12, 26
12, 26
```

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
```

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
```

列举 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
```

类别 class

- 可以建立属性，并且给予默认值

```
class Person{
    name:String = "王小明";
    constructor(name?: String){
        name ? this.name = name : null;
    }
    sayHi(){
        return `你好，我是${this.name}`;
    }
}

let ming = new Person();
console.log(ming.sayHi());
```

```
你好，我是王小明
```

类别 class

- 有传入值时，使用传入值

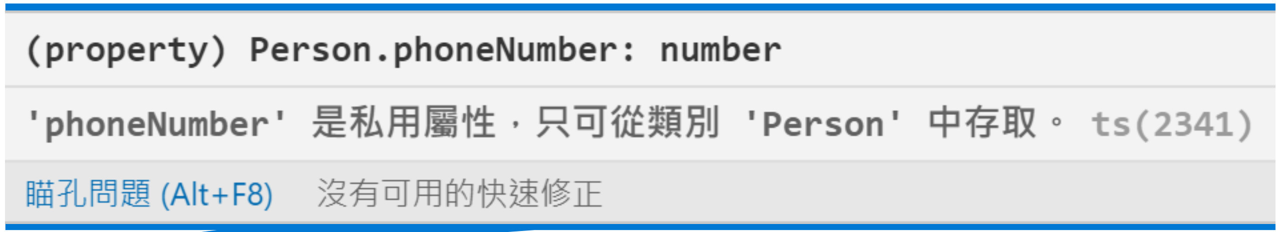
```
class Person{
    name:String = "王小明";
    constructor(name?: String){
        name ? this.name = name : null;
    }
    sayHi(){
        return `你好，我是${this.name}`;
    }
}

let david = new Person(“李戴维”);
console.log(david.sayHi());
```

```
你好，我是李戴维
```

类别 class

- 私有变量，不可直接存取

```
class Person{
    name:String = "王小明";
    private phoneNumber:number = 7533967;
    constructor(name?: String, phoneNumber?:number){
        name ? this.name = name : null;
        phoneNumber ? this.phoneNumber = phoneNumber : null;
    }
    sayHi(){
        return `你好，我是${this.name}`;
    }
}

let david = new Person("李戴维", 8825252);
console.log(david.sayHi());
console.log(david.phoneNumber);
```



```
(property) Person.phoneNumber: number
'phoneNumber' 是私用屬性，只可從類別 'Person' 中存取。 ts(2341)
瞄孔問題 (Alt+F8)   沒有可用的快速修正
```

类别 class

- 经由方法进行存取

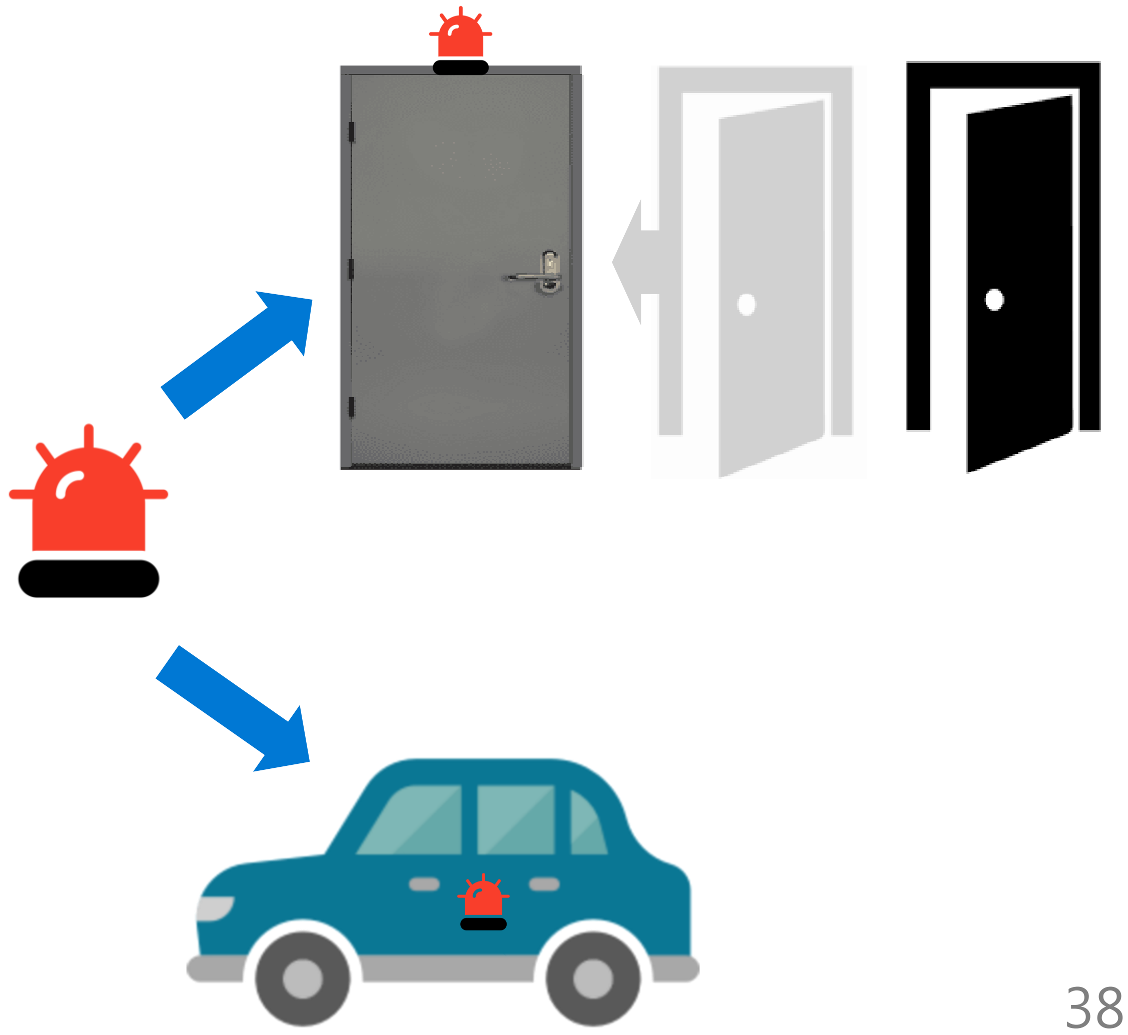
```
class Person{
    name:String = "王小明";
    private phoneNumber:number = 7533967;
    constructor(name?: String, phoneNumber?:number){
        name ? this.name = name : null;
        phoneNumber ? this.phoneNumber = phoneNumber : null;
    }
    sayHi(){
        return `你好，我是${this.name}`;
    }
    checkNumber(password:String){
        if(password=="芝麻开门"){
            return `${this.name}的电话是${this.phoneNumber}`;
        }else{
            return "通关密码错误";
        }
    }
}

let david = new Person("李戴维", 8825252);
console.log(david.sayHi());
console.log(david.checkNumber("芝麻开门"));
```

```
你好，我是李戴维
李戴维的电话是8825252
```

界面实作

- 不同的类别可能都需要用到的，可以独立成接口

```
interface Alarm{
    alert();
}

class Door{
}

class SecurityDoor extends Door implements Alarm{
    alert(){
        console.log('门上的警报器响了!');
    }
}

class Car implements Alarm{
    alert(){
        console.log('汽车上的警报器响了!');
    }
}

let officeDoor = new SecurityDoor();
officeDoor.alert();
let myCar = new Car();
myCar.alert();
```



练习：DOM操作

- 找到网页中的组件
- 新增、删除、变更、取得内容

window Object

- 在浏览器中开启一个窗口即建立一个窗口对象
- 属性
  - window.closed：该窗口关闭即为true
  - window.name：该窗口名称
  - history (物件)：记录下用户在该窗口所去过的网址
  - navigator (物件)：浏览器相关信息
  - document (物件)：当该窗口加载一份HTML文件时即产生

document Object

- 属性
  - domain：传回目前文件所在的域名
  - title：传回目前文件定义的title
  - URL：传回目前文件的完整网址路径
  - cookie：传回目前文件的cookie信息
- 方法
  - getElementById( )：存取第一个id名称相符的组件
  - getElementsByName( )：存取所有name相符的组件
  - getElementsByTagName( )：存取所有该卷标名称的组件
  - write( )：写入文件
  - writeln( )：写入文件并带上换行符号

document Object

- Collections 筛选
  - anchors[ ] 找到所有页面上的anchor
  - forms[ ] 找到所有页面上的form
  - images[ ] 找到所有页面上的image
  - links[ ] 找到所有页面上的link

Document Object Model

- W3C标准
- 用来存取HTML或XML文件
- Core DOM
  - 任何结构化文件的标准模型
- XML DOM
  - XML文件的标准模型
- HTML DOM
  - HTML文件的标准模型

HTML DOM

- HTML的标准对象模型
- HTML的标准程序界面
- 可于各种平台、语言使用
- 用来取得、改变、新增或删除HTML组件

Node 节点

- HTML文件中，所有的事物都是一个节点
- 整个文件：文件节点
- HTML组件：组件节点
- HTML组件中的文字：文字节点
- HTML中的属性：属性节点
- HTML中的批注：批注节点

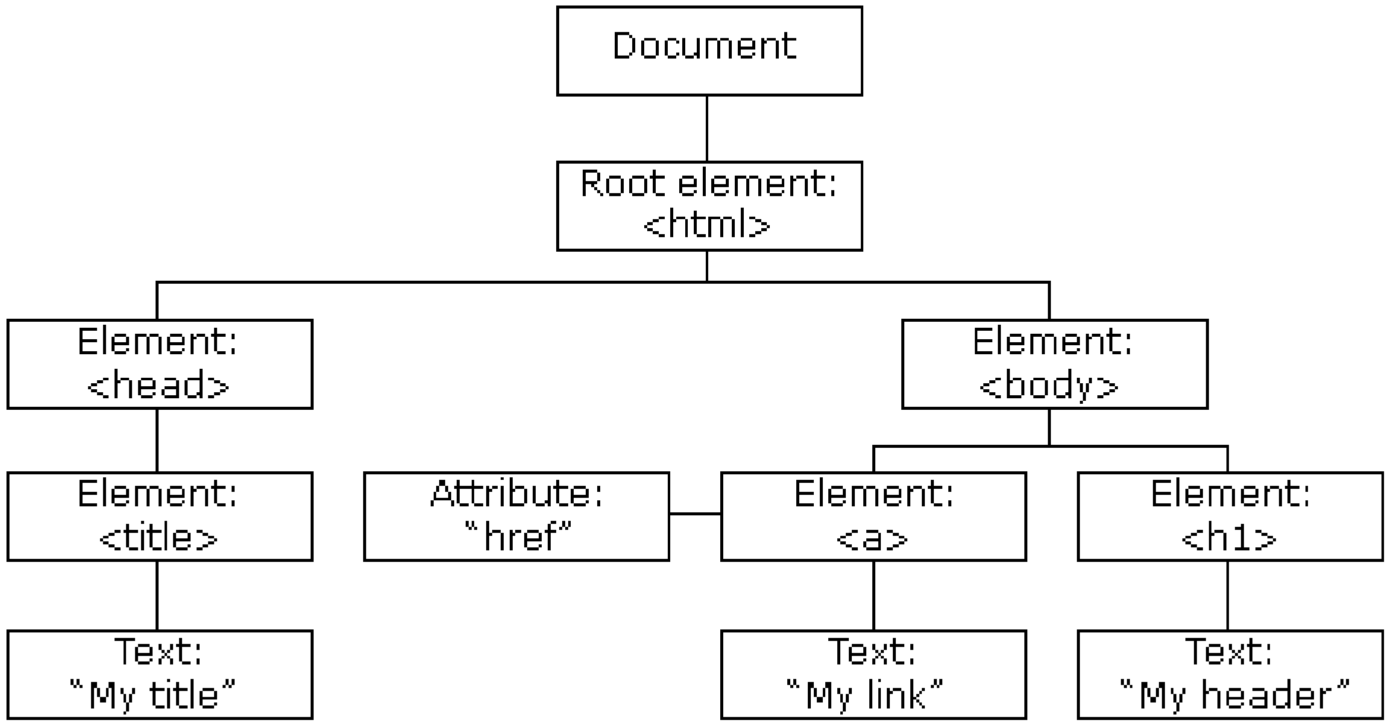
节点分析

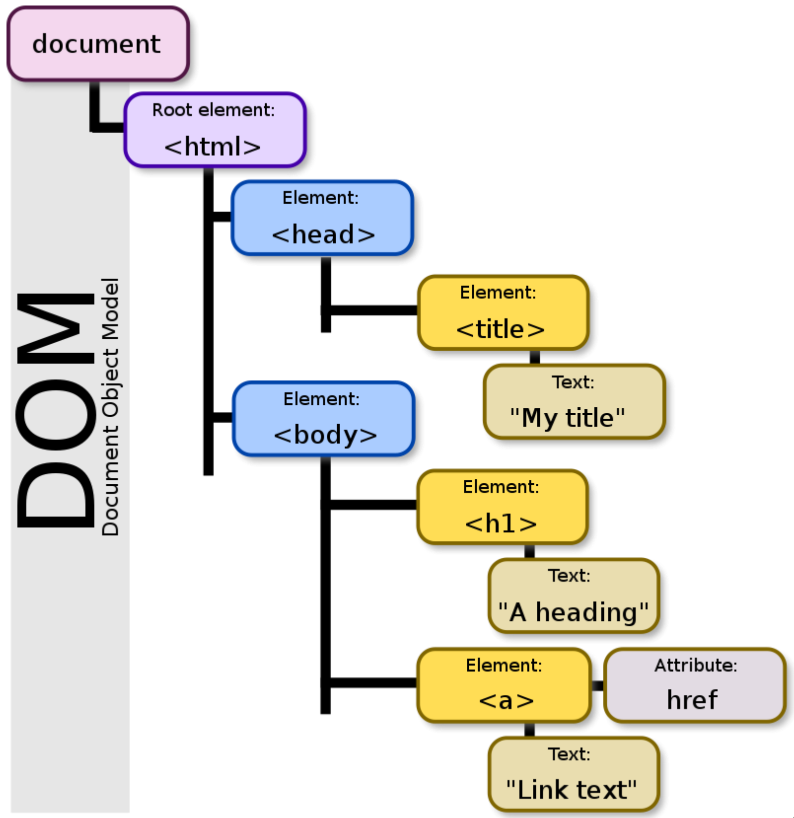
- 根节点：<html>
- <html>节点有两个子节点：<head>与<body>
- <title>节点有一个text子节点：DOM Tutorial

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

树状结构

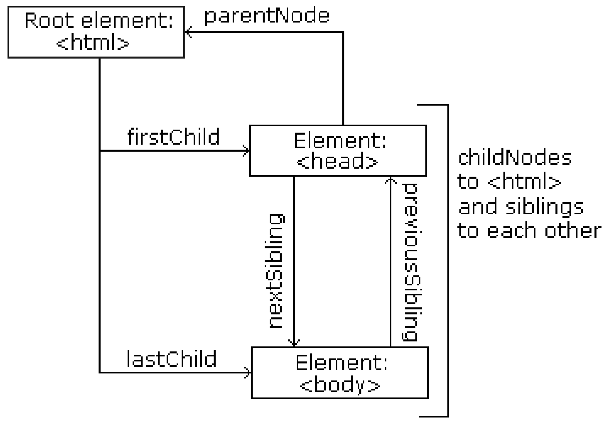
- HTML DOM把HTML文件当成一颗节点树





树状结构：父母、小孩与兄弟姊妹

- 最上层的节点称之为根 (root)
- 除了根节点外，每一个节点都有一个父节点
- 一个节点可以有任意数量的子节点
- 没有子节点的节点称之为叶 (leaf)
- 有相同父节点的节点称之为兄弟(sibling)



HTML DOM属性

- innerHTML：文字值
- nodeName：名称
- nodeValue：值
- parentNode：父节点
- firstChild：第一个子节点
- lastChild：最后一个子节点
- nextSibling：紧邻的兄弟节点
- nodeType：组件型态

HTML DOM属性

- nodeName
  - 组件节点的节点名称即为标签名称(会变成大写)
  - 属性节点的节点名称即为属性名称
  - 文字节点的节点名称为#text
  - 文件节点的节点名称为#document
- nodeValue
  - 组件节点的节点值为null
  - 文字节点的节点值即为文字本身
  - 属性节点的节点值即为属性值

- nodeType
  - 1：Element 组件
  - 2：Attribute 属性
  - 3：Text 文字
  - 8：Comment 批注
  - 9：Document 文件

HTML DOM 方法

- getElementById(id)
- getElementsByTagName(name)
- appendChild(node)
- removeChild(node)
- getAttribute(attributeName)

HTML DOM Collections

- attributes[ ]
  - 回传该组件的所有属性组成一个数组
- childNodes[ ]
  - 回传该组件的所有子节点组成一个数组

改变组件

- 改变组件的属性值
  - document.body.bgColor="yellow";
- 改变组件内的文字
  - document.getElementById("p1").innerHTML="Hi!";
- 改变组件的样式
  - document.body.style.color="blue";
  - document.body.style.backgroundImage

新增专案：HelloDOM

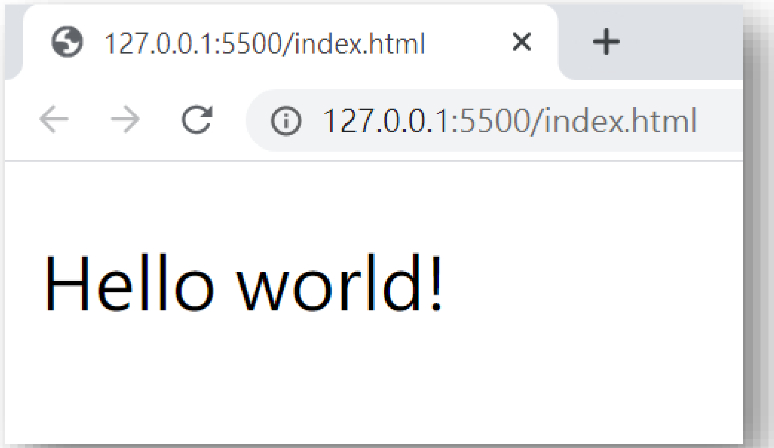
- 可复制TS_WebTemplate进行修改
- VS Code 开启文件夹 -> HelloDOM
- 打开index.html

```
<!DOCTYPE html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <link rel="stylesheet" href="">
    </head>
    <body>
        <div id="app"></div>
        <script src="js/main.js" async defer></script>
    </body>
</html>
```

修改main.ts

```
//console.log("Hello world!");
const app = document.getElementById("app");
const p = document.createElement("p");
p.textContent = "Hello world!";
app?.appendChild(p);
```

- Ctrl+ “~” 打开终端机，输入指令 tsc
- 在index.html上按下右键，Open with Live Server
- 确认是否有出现Hello World!

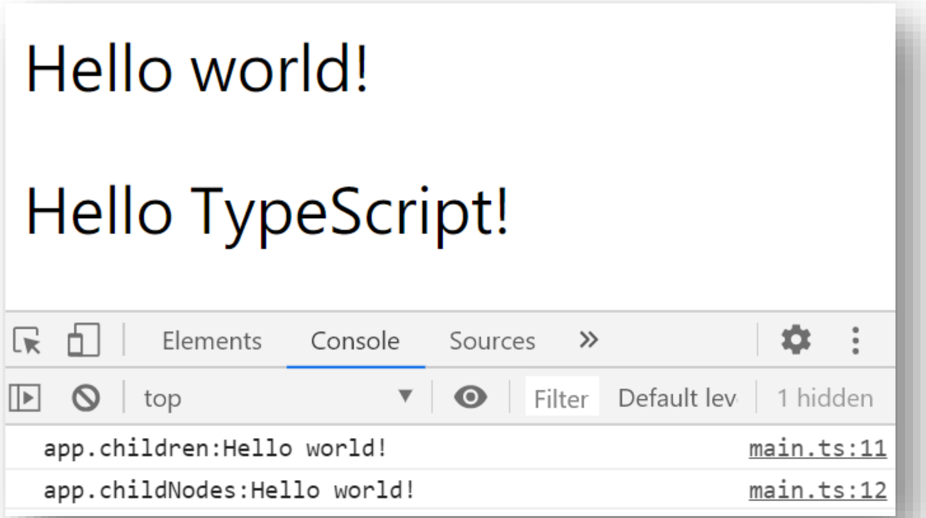

127.0.0.1:5500/index.html

Hello world!

main.ts

- 取得特定子节点中的值

```
//console.log("Hello world!");
const app = document.getElementById("app");
const p = document.createElement("p");
p.textContent = "Hello world!";
app?.appendChild(p);

const p2 = document.createElement("p");
p2.textContent = "Hello TypeScript!";
app?.appendChild(p2);

console.log("app.children:"+app.children[0].innerHTML);
console.log("app.childNodes:"+app.childNodes[0].textContent);
```


Hello world!

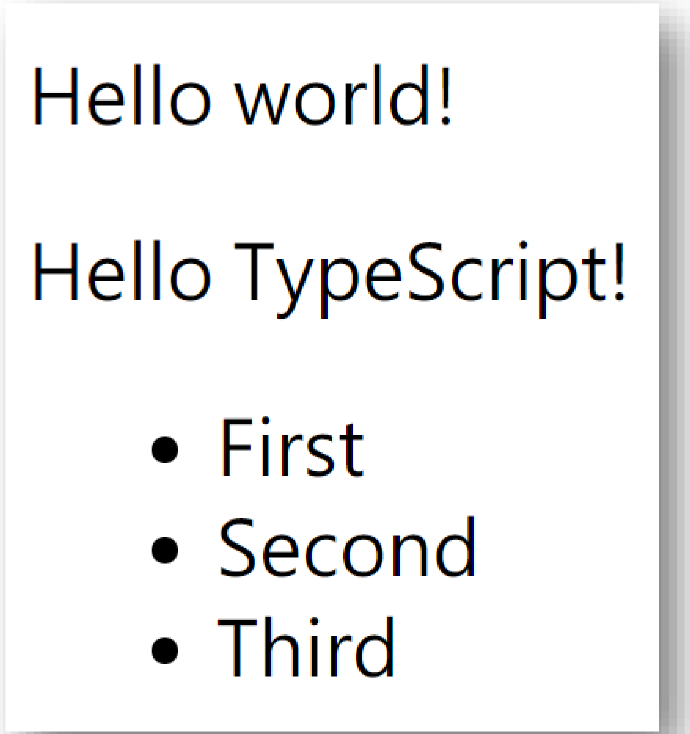
Hello TypeScript!

```
app.children:Hello world!      main.ts:11
app.childNodes:Hello world!    main.ts:12
```

index.html

- 增加一个清单

```html
<!DOCTYPE html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <link rel="stylesheet" href="">
    </head>
    <body>
        <div id="app"></div>
        <ul>
            <li>First</li>
            <li>Second</li>
            <li>Third</li>
        </ul>
        <script src="js/main.js" async defer></script>
    </body>
</html>
```


Hello world!

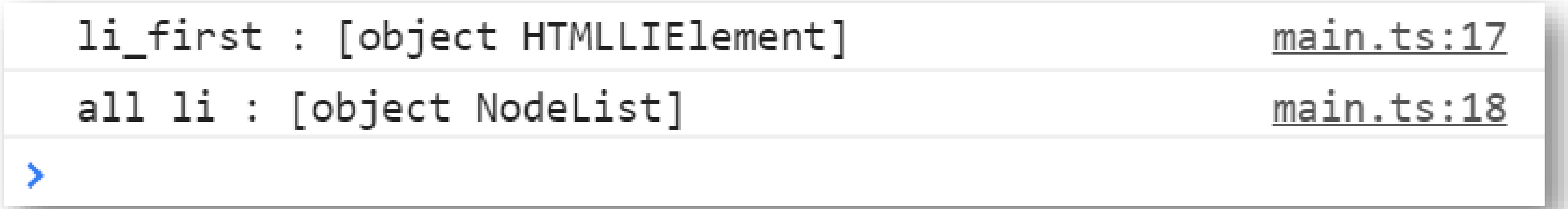
Hello TypeScript!

- First
- Second
- Third

main.ts

- 抓出第一个吻合的 or 抓出全部

```
//console.log("app.children:"+app.children[0].innerHTML);
//console.log("app.childNodes:"+app.childNodes[0].textContent);

const li_first = document.querySelector("li");
const li_all = document.querySelectorAll("li");

console.log("li_first : "+li_first);
console.log("all li : "+li_all);
```

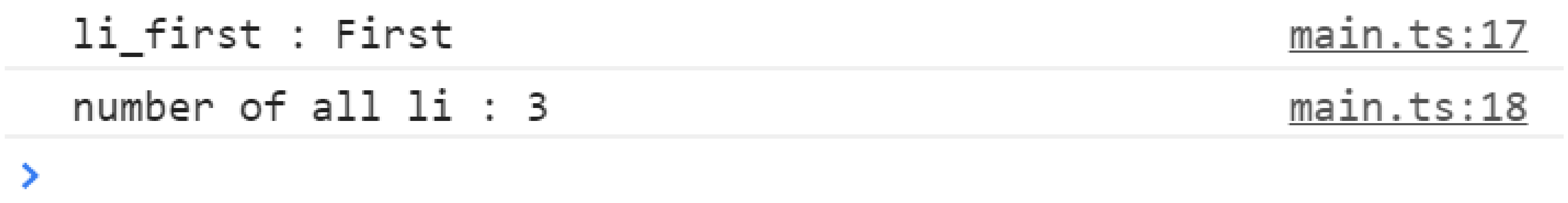


```
li_first : [object HTMLLIElement]        main.ts:17
all li : [object NodeList]               main.ts:18
>
```

main.ts

- 看看里面的内容 or 数数量

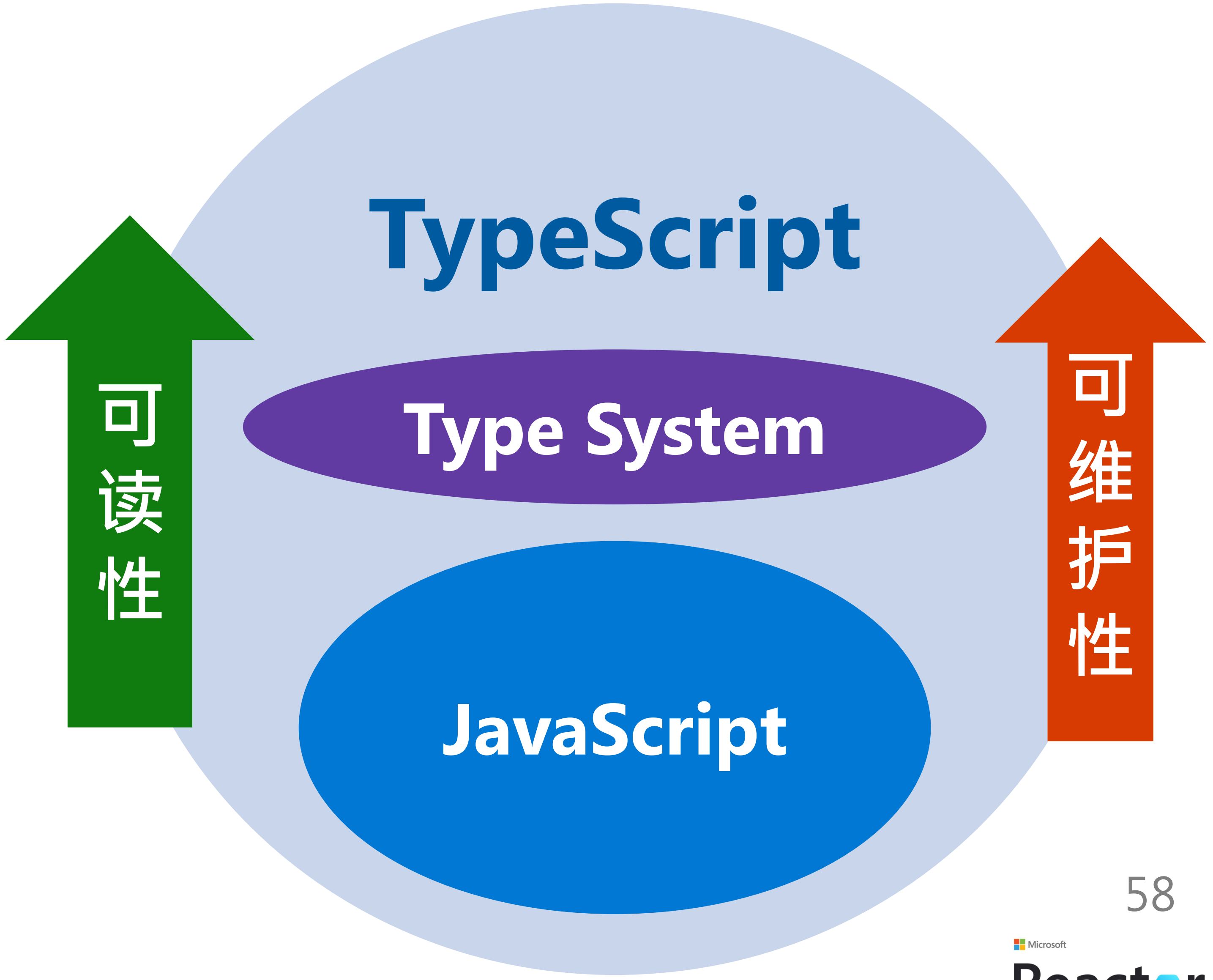
```
//console.log("app.children:"+app.children[0].innerHTML);
//console.log("app.childNodes:"+app.childNodes[0].textContent);

const li_first = document.querySelector("li");
const li_all = document.querySelectorAll("li");

console.log("li_first : "+li_first.textContent);
console.log("number of all li : "+li_all.length);
```



```
li_first : First                          main.ts:17
number of all li : 3                      main.ts:18
>
```

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

延伸学习资源

- 官方文件
  - https://www.typescriptlang.org/
- 开放原始码
  - https://github.com/Microsoft/TypeScript
- 在线语法测试
  - https://www.typescriptlang.org/play/
- Visual Studio Code – TypeScript引导
  - https://code.visualstudio.com/docs/typescript/typescript-tutorial

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
https://aka.ms/ReactorFeedback