Front End React

講師：李昱賞

第 一 章

- 主題1：Front End Framework React 簡介
- 主題2：開發工具安裝及環境設置
- 主題3：建立 React 專案

主題1： Front End Framework React 簡介

- React

- **React**（也稱為React.js或ReactJS）是一個自由及開放原始碼的前端JavaScript工具庫，用於基於UI組件構建使用者介面。

- 它由Meta（前身為 Facebook）和由個人開發者和公司組成的社群維護。React只專注狀態管理和將狀態呈現(渲染)到DOM，因此建立React應用程式通常需要使用額外的工具庫來進行路由實作，以及某些客戶端功能。

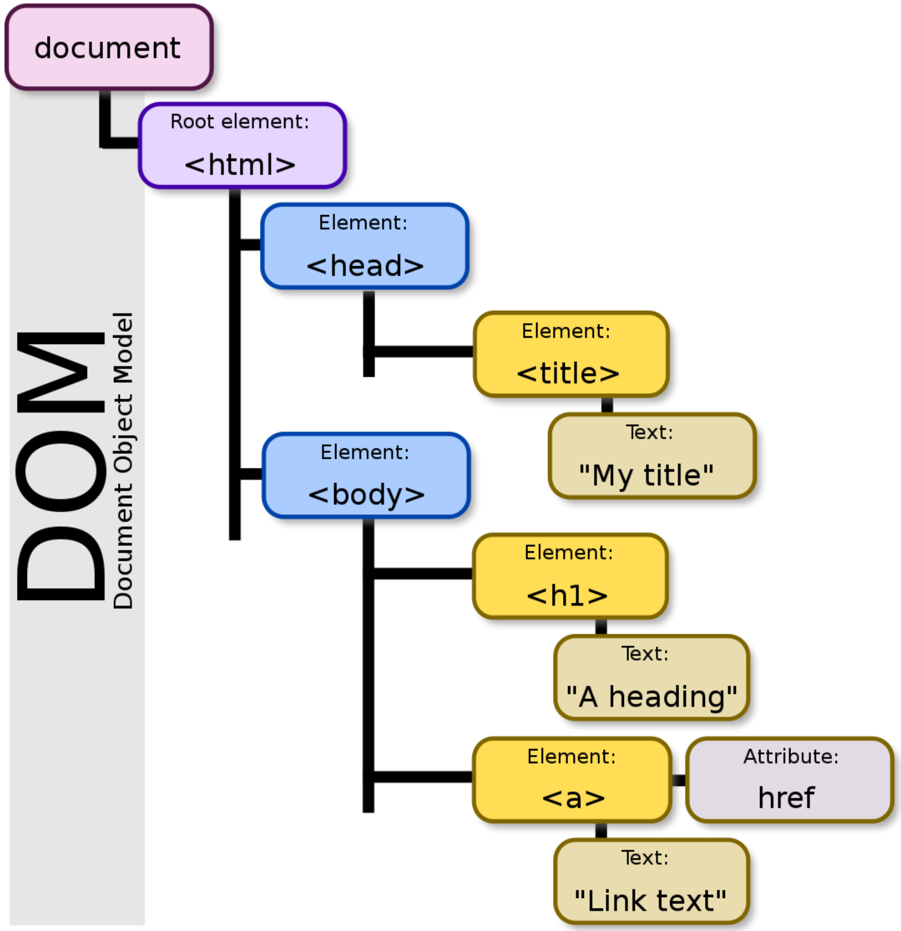
- Virtual DOM

  React使用虛擬文件物件模型或虛擬DOM。React建立一個記憶體資料結構暫存，計算結果差異，然後有效地更新瀏覽器顯示的DOM。

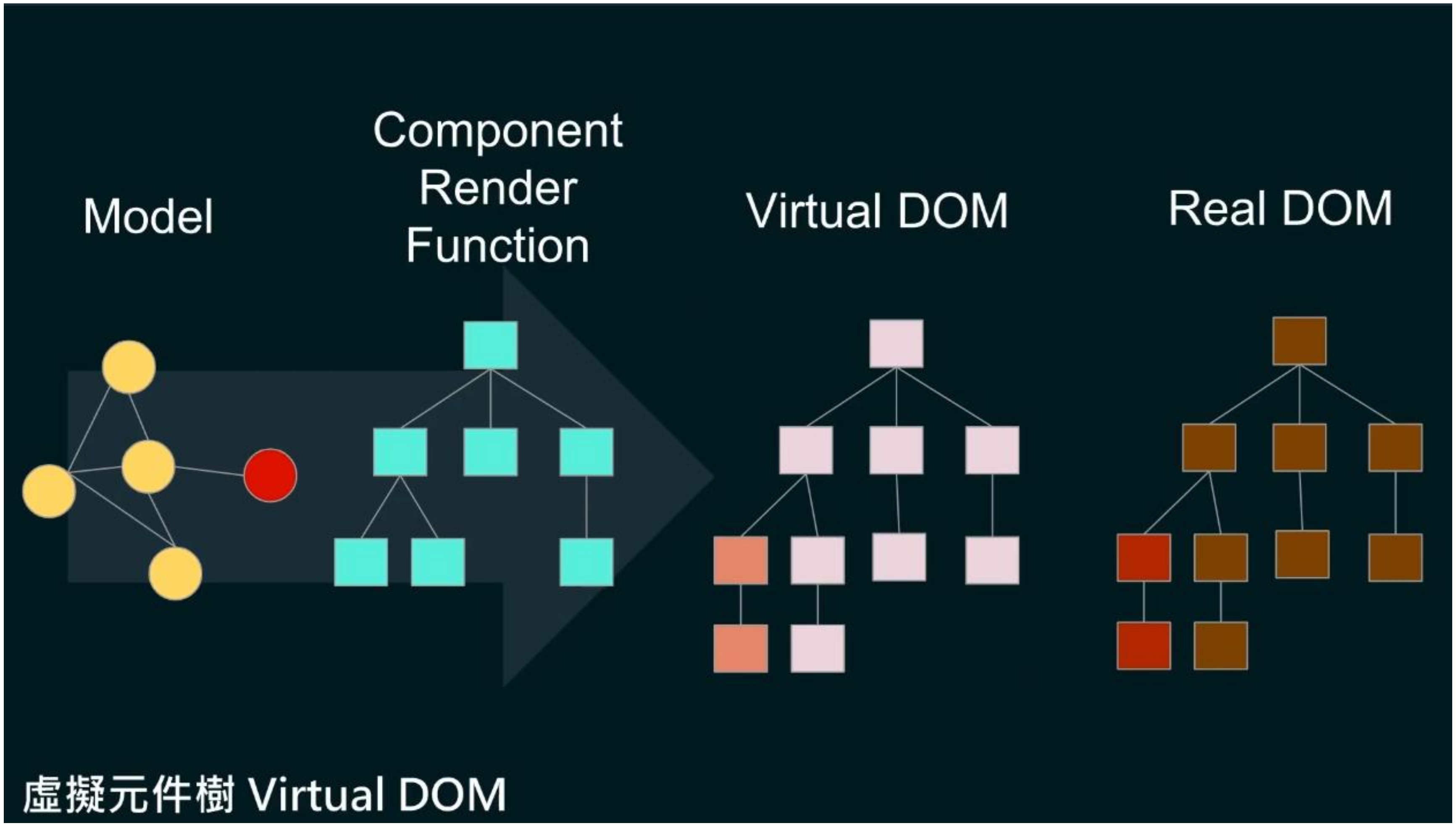
  這個過程稱為 reconciliation。這允許程式工程師撰寫程式碼，就好像每次更改都會刷新整個頁面，而React只更新實際更改的子組件。這種選擇性呈現提供了主要的效能提升。節省了重新計算CSS樣式、頁面排版和呈現整個頁面的工作量。

主題1： Front End Framework React 簡介

- 文件物件模型（**Document Object Model**，縮寫**DOM**）



主題1： Front End Framework React 簡介



主題1： Front End Framework React 簡介

- Model
  程式裡開發使用到的資料模型

- Component Render Function
  程式裡開發使用到的 React 組件

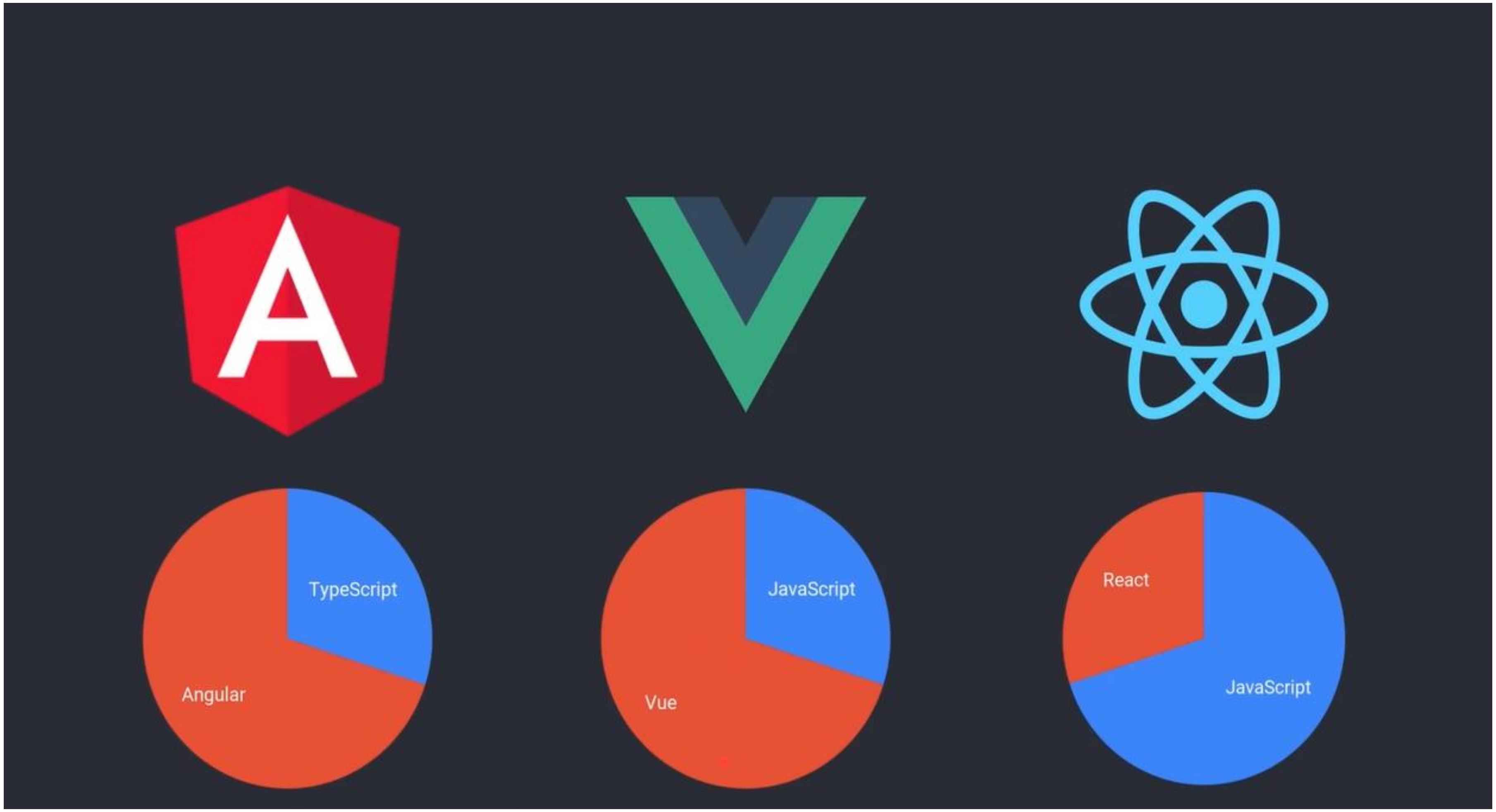
- Virtual DOM (虛擬元件樹)
  假設只有其中一個Model資料更新改變了，
  中間透過 Component Render Function 轉換之後，
  會整個 Virtual DOM 全部重新 Render更新，
  重建整個 Virtual DOM 且都在 Memory 之中完成，
  最後 Virtual DOM 在與 Real DOM 做比較並且只更新有差異的部份更新
  至 Real DOM，畫面更新的時候是最耗效能的

- Real DOM
  使用者最後看的到的頁面 HTML DOM (Document Object Model) 元素

主題1： Front End Framework React 簡介

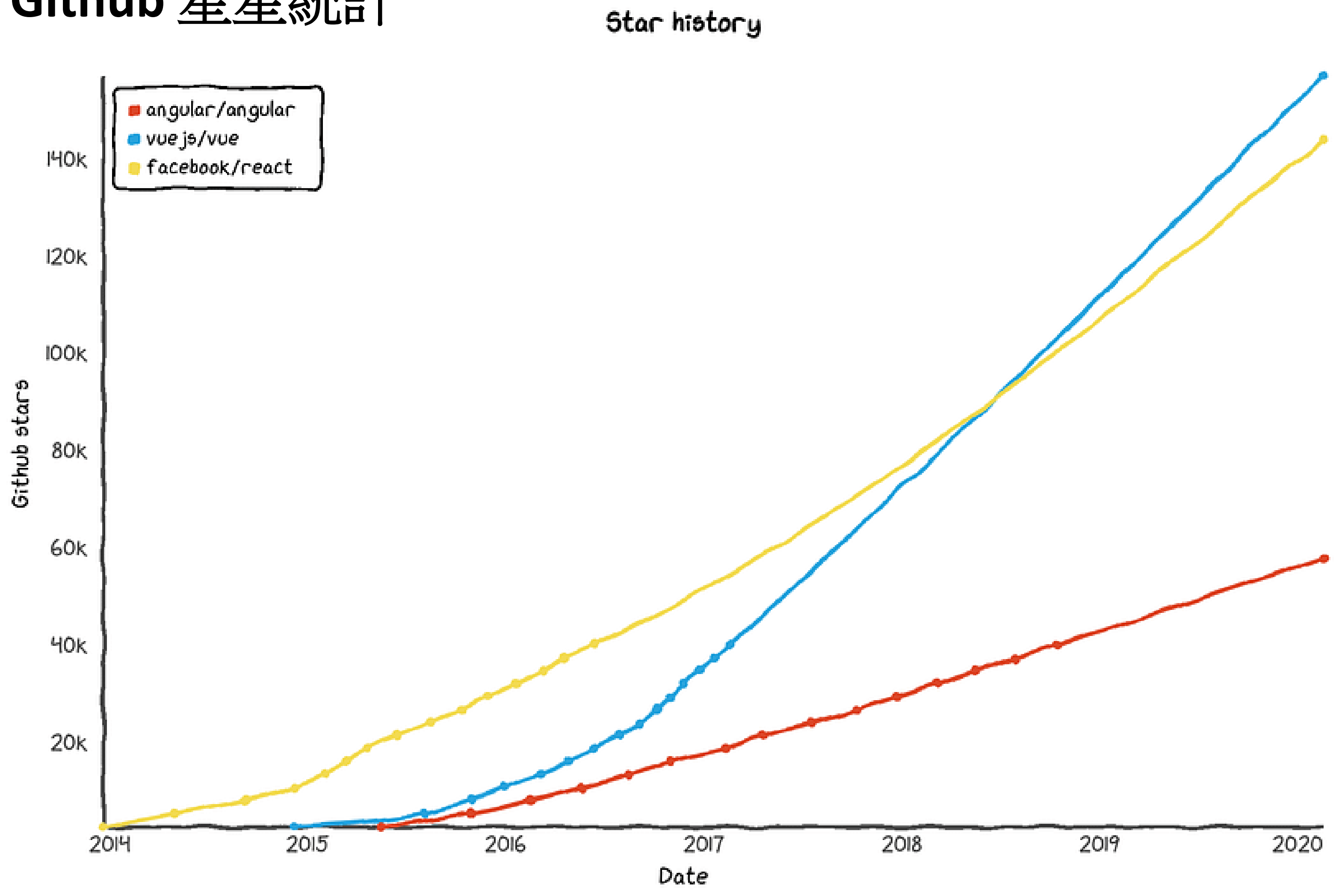
	Angular	**React**	**Vue**
出生年	2010	2013	2014
開發者	Google	Facebook	尤雨溪
著名開發案例	YouTube	Instagram Netfix	Gitlab
語言	**Typescript**	**JS**	**JS**
模板語法	**HTML(不完全)**	**JSX**	**HTML**

主題1： Front End Framework React 簡介



主題1： Front End Framework React 簡介

- **Github 星星統計**

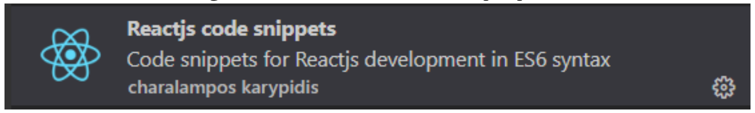


主題2：開發工具安裝及環境設置

- Node js
  https://nodejs.org/zh-tw/download/

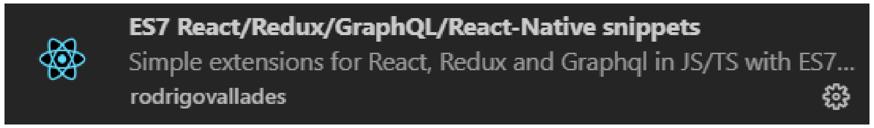
- Visual Studio Code
  https://code.visualstudio.com/Download

- Extensions 套件安裝
  - Reactjs code snippets



Reactjs code snippets
Code snippets for Reactjs development in ES6 syntax
charalampos karypidis

  - ES7 React/Redux/GraphQL/React-Native snippets



ES7 React/Redux/GraphQL/React-Native snippets
Simple extensions for React, Redux and Graphql in JS/TS with ES7...
rodrigovallades

主題3：建立 React 專案

➢ Create React APP

① 安裝 create-react-app

– 以系統管理員身份開啟 Visual Studio Code 設置 npm 安裝權限

```
Set-ExecutionPolicy RemoteSigned
```

– 安裝工具 yran

```
npm install yarn –g
```

– 透過 yran 工具安裝 create-react-app

```
yarn global add create-react-app
```

② 建立react專案

```
create-react-app my-react-app
```

③ npm 編譯建立 (產生可發佈build資料夾)

```
npm run build
```

④ npm 專案啟動

– 將會自動開啟瀏覽器 http://localhost:3000/

```
npm start
```

主題3：建立 React 專案

➢ React 開發 npm 安裝開發套件

① npm安裝開發套件指令參考

```
npm install -S react
npm install -S webpack
npm install -S axios
npm isntall zustand
npm install -S cross-fetch
npm install react-bootstrap@1.6.4 bootstrap@4.6.0
npm i react-router
npm i react-router-dom
npm install --global babel-node
npm install --save @babel/core @babel/cli @babel/preset-env @babel/node
npm install --save @babel/polyfill
```

● 刪除套件指令

```
npm remove --save react
```

主題3：建立 React 專案

② **package.json** (React App套件設定檔)
可直接複制以下dependencies設定後執行 **npm install** 指令

```
"dependencies": {
  "@babel/cli": "^7.19.3",
  "@babel/core": "^7.19.6",
  "@babel/node": "^7.19.1",
  "@babel/polyfill": "^7.12.1",
  "@babel/preset-env": "^7.19.4",
  "axios": "^0.27.2",
  "babel-node": "^0.0.1-security",
  "bootstrap": "^4.6.0",
  "cross-fetch": "^3.1.5",
  "node-fetch": "^3.2.10",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-bootstrap": "^1.6.4",
  "react-router": "^6.3.0",
  "react-router-dom": "^6.3.0",
  "react-scripts": "5.0.1",
  "typescript": "^4.8.2",
  "web-vitals": "^2.1.4",
  "webpack": "^5.73.0"
}
```

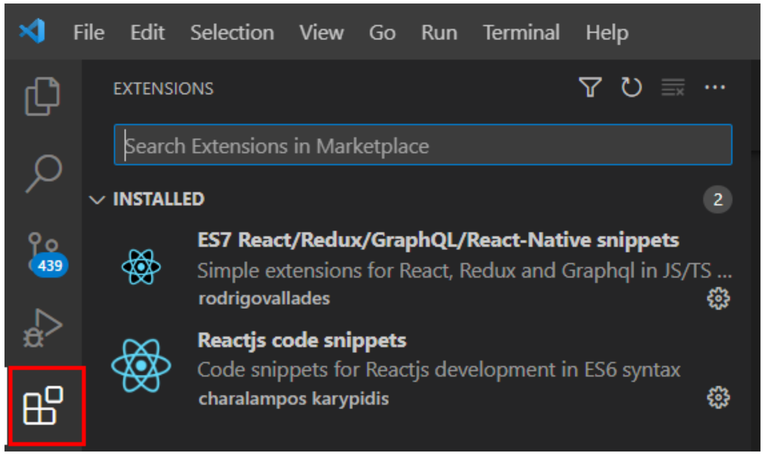
主題3：建立 React 專案

➢ 快捷鍵說明

VS Code EXTENSIONS (安裝rcc自動生成元件程式)

1.Creates a React component class
指令：**rcc**
Reactjs code snippets

2.Creates a React Arrow Function Component
指令：**rafce**
ES7 React/Redux/GraphQL/React-Native snippets



ES7 React/Redux/GraphQL/React-Native snippets
Simple extensions for React, Redux and Graphql in JS/TS ...
rodrigovallades

Reactjs code snippets
Code snippets for Reactjs development in ES6 syntax
charalampos karypidis

3.其它自動建立程式快捷鍵參考：
https://ithelp.ithome.com.tw/articles/10266011

4.VS Code
Shift + Alt + F (自動排版)
Ctrl + / (註解)
Ctrl + e (尋找指定檔名的檔案)

第 二 章
ES6 (ECMAScript)、JSX

- 主題1： var、let、const
- 主題2：arrow functions
- 主題3：import 與 export
- 主題4：class 物件導向
- 主題5：ES6 其它語法
- 主題6：React JSX

主題1： var、let、const

1. 宣告 let 值可被更改，const 值不可被更改
2. function scope(函式作用域) / global scope(全域作用域)
3. scope out → in 會從外而內傳遞進去

```
// global scope(全域作用域)
var x = 1;
function f(){
   // function scope(函式作用域)
   var x = 2;
   // 如function scope未宣告則將會被global scope從外而內傳遞進去取代
   console.log(x);  // 2
}
f();
console.log(x); // 1
```

主題1： var、let、const

1. var 作用域: scope (全域作用域、函式作用域)
2. let / const(ES6) 作用域: block { }

```
var y = 1;
{
   // 因為var的作用域在"scope"而不是"block",所以"會"被裡面的取代
   var y = 2;
   console.log(y); // 2
}
console.log(y); // 2 (被取代)


let z = 1;
{
   // let、const 看的是"block"區塊,所以"不會"被裡面的取代
   let z = 2;
   console.log(z); // 2
}
console.log(z); // 1 (未被取代)
```

主題2：arrow functions

- 傳統Function

```
const customer1 = {age:36, name:"Shang"};
const customer2 = {age:16, name:"Wendy"};

// 傳統Function
function visitMessage1(cus) {
  var age = cus.age;
  var name = cus.name;
  // 根據 age 資料回傳不同字串
  if (age < 18) {
    return 'Dear ' + name + ', you are under age:' + age;
  }
  return 'Welcome, ' + name + ':' + age + '!';
}

console.log( visitMessage1(customer1) );  // Welcome, Shang:36!
```

主題2：arrow functions

- 箭頭函式

宣告常數等於一個 "函式"

1. 箭頭後面接"大括弧"表示方法內"多行"實作,且最後必須有回傳結果

( ) => { return xxx; }

2. 字串模板 **`... ${name} ...`**

```
const visitMessage2 = (cus) => {
  // 物件解構
  const { age, name } = cus;
  // 字串模板
  const text = age < 18 ?
  `Dear ${name}, you are under age:${age}`
  : `Welcome, ${name}:${age}!`;

  return text;
};

console.log( visitMessage2(customer2) ); // Dear Wendy, you are under age:16
```

主題2：arrow functions

3. 箭頭後面接"小括弧"(可省略)表示方法內"單行"實作

表示直接要回傳括弧內的結果 ( ) => ( )

4. 直接在參數上「解構」物件 { age, name }

```
const visitMessage3 = ({ age, name }) => (
  age < 18
  ? `Dear ${name}, you are under age:${age}`
  : `Welcome, ${name}:${age}!`
);

console.log( visitMessage3(customer2) ); // Dear Wendy, you are under age:16
```

5.箭頭後面接回傳結果省略"小括弧"

```
const circleArea = radius => radius * radius * Math.PI;

console.log(circleArea(10)); // 314.1592653589793
```

主題3：import 與 export

- Export 匯出

1. export default (只能有一個)

   預設匯出什麼, 匯入的 JS 就會 import 到什麼

   import 時名子不須一樣

2. named export (可以多個)

   import 時名子必須一樣

◆ math.js

```
const math = {
   double: x => x * 2,
   square: x => x * x,
   area: (w,h) => w * h
};

// 1.export default(只能有一個)什麼,別人就會import到什麼東西
// import 時名子不須一樣
export default math;

// 2.named export (可以多個)
// import 時名子須一樣
export const PI = 3.1415;
export const circleAreaFun = (radius) => radius * radius * Math.PI;
```

主題3：import 與 export

- Import 匯入

使用 node 執行時 package.json 須加上
"type": "module" 才能使用 import 功能

◆ main.js

```
import m, {PI, circleAreaFun} from './math.js';

console.log(m.square(12)); // 144

console.log(PI); // 3.1415

console.log(circleAreaFun(10)); // 314.1592653589793
```

主題4：class 物件導向

- Class **constructor** 建構式

```
class Animal {
  constructor(){
    console.log('Animal Create!');
    this.age = 17;
  }
}
```

- Class **extends** 繼承

```
class Dog extends Animal {

  constructor() {
    super(); // super()不可省略
    console.log('Dog Create!');
  }

  bark = ({ name }) => {
    console.log(`woof ${name} ${this.age}`);
  }
}
```

主題4：class 物件導向

- 類別實體化 **instance** 操作
  透過 **new** 建立類別實體化，並透過物件操作方法
  父類別先建立再子類別建立實體

```
const spot = new Dog();
spot.bark( {name:"Snoppy"} );
```

◆ 輸出結果

```
Animal Create!
Dog Create!
woof Snoppy 17
```

主題5：ES6 其它語法

- 陣列解構

```
const point = [1,2,3,4,5];
const [x, y, ...rest] = point;

console.log("x:", x); // 1
console.log("y:", y); // 2
console.log("rest:", rest); // 3,4,5
```

- 物件解構

  將物件解構出欄位

```
const pointObj = {a:1, b:2, c:3};
const {a, b, c} = pointObj;
console.log("`${a}_${b}_${c}`:",`${a}_${b}_${c}`);
```

主題5：ES6 其它語法

- 字串模板 (使用**頓號 \` \`**)

可將動態物件資料與固定文字 wording 字串串間

```
const age  = 36;
const message1 = 'I am ' + age + ' years old';
const message2 = `I am ${age} years old`;
console.log("message2:", message2); // I am 36 years old
```

主題5：ES6 其它語法

- ES6 接收非同步請求回傳結果 callback fuction 回調函式

1. 透過 fetch 原廠 callback 回調函數

fetch 為**非同步**請求不能直接接收所回傳的結果

Fetch 基於 Promise 實現所回傳的資料型別為 **Promise** <**Response**>

```
const getPostsData = (userID) => {
   const userData =  fetch('https://jsonplaceholder.typicode.com/posts/' + userID)
   // 此處為非同步回傳
   .then( rs => rs.json() )
   // 透過 fetch 原廠 callback 回調函數
   .then( (userData) => console.log("FetchCallbackFun:", userData) )
   .catch(error => {
      console.log(error);
   });

   console.log("getPostsData inner:", userData); // Promise { <pending> }

   return userData;
};

const user1 = getPostsData(1);
console.log("getPostsData outter:", user1); // Promise { <pending> }
```

主題5：ES6 其它語法

2. 自行撰寫 callback 回調函數

利用**函數當做參數傳入**取得非同步的回傳結果

```
const callbackFetchPostsData = (callback, userID) => {
   fetch('https://jsonplaceholder.typicode.com/posts/' + userID)
   // 此處為非同步回傳
   .then( rs => rs.json() )
   // 須加.then接收回傳結果
   .then((userData) => {
      callback(userData);
   })
   .catch(error => {
      console.log(error);
   });
};

callbackFetchPostsData( (user) => {
   console.log("FetchCallbackCusFun:", user);
}, 2);
```

主題5：ES6 其它語法

3. ES6：async/await

```
// await後面必須接為Promise回傳的結果,且函數前面必須寫上async
const asyncFetchPostsData = async (userID) => {
    const userData = await fetch('https://jsonplaceholder.typicode.com/posts/' + userID)
    .then( rs => rs.json())
    .catch(error => {
        console.log("FetchError:",error);
    });

    console.log("inner:", userData);

    return userData;
};

// 在函數“內”要接回fetch非同步請求
// 須在被呼叫的函數參數括弧前加 "async"、fetch函數前加 "await"
asyncFetchPostsData();

// 在函數"外"要接回fetch非同步請求，須在呼叫函數前加 "await"
// PS:被呼叫的函數上不須加 "async"、函數內不須加"await"
const user = await asyncFetchPostsData(3);
console.log("outter:", user);
```

主題6：React JSX

- JSX 用 javascript 寫 HTML

在 JS 的檔案中使用 HTML 的標籤，並且使用 JSX 語法建立的是「一個 React 的 element」，此外這樣的標籤語法比起 HTML，更貼近於 JavaScript。

- JSX 物件

每一個組件都會有一個 **render** 函式,將 JSX 結果渲染至畫面繪製

1. must close tag (HTML都必須有結束標籤 EX:<div>...</div>、<input/>)
2. self close <input/>
3. html tag class 撞名 react class, JSX 使用 **className** 取代
4. onChange、onClick **駝峰式命名** (不同於html全小寫)
5. 使用**大括號 { }** 括住一個「值」或「表達示」

第 三 章
React Component

- 主題1：React Component import
- 主題2：React Component state 狀態管理
- 主題3：React Component props
- 主題4：React Component state 解構

主題1：React Component import

- React 組件匯出 **export**

可將 Component 組件匯出並在其它組件匯入來將兩個組件合併在一起

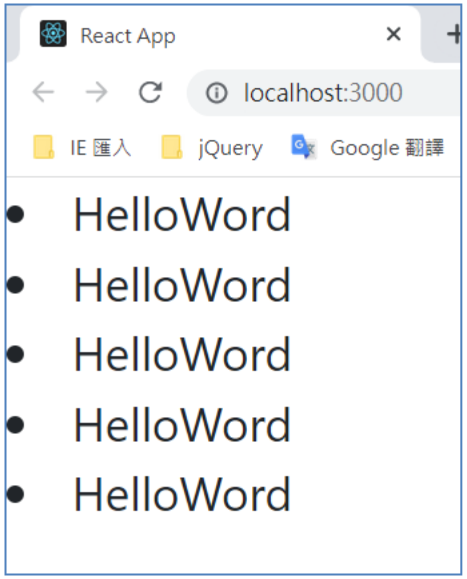
```
import React, { Component } from "react";

// class Item extends React.Component {
class Item extends Component {

  render( ) {
    return (
      <li>HelloWord</li>
    )
  }

}

export default Item;
```

主題1：React Component import

- React 組件匯入 **import**

在需要的地方匯入並且引用組件

```
import React, { Component } from 'react';
import Item from './Item';

class List extends Component {
    render() {
        return (
            <div>
                <Item/>
                <Item/>
                <Item/>
                <Item/>
                <Item/>
            </div>
        );
    }
}

export default List;
```



- HelloWord
- HelloWord
- HelloWord
- HelloWord
- HelloWord

主題2：React Component state 狀態管理

- Component state 組件狀態

  1. **state** (組件狀態)
  2. **setState** (組件狀態更新)
  3. state 狀態支援可部份更新 (Partial update) 欄位值

◆ 組件**建立**狀態 State 語法

```
state = {
    title: '我是Title',
    text: '我是Text',
    count: 0
};
```

◆ 組件**更新**狀態 State 語法

```
this.setState (
    {
        text: 'Hello React',
        count: this.state.count + 1
    }
);
```

主題2：React Component state 狀態管理

● Component event 事件處理

網頁功能上有許多行為都會觸發事件，但此時的 this 指的是 html 元素自身 EX：<button> 而非 React 的組件, 修改方式以下兩種：

◆ 1.透過**建構式**再綁定回組件 bind(this)

```
constructor (props) {
   super(props); // 此行不可省略
   // 透過建構式綁定函數傳入this組件bind(this)
   this.updateState = this.updateState.bind(this);
}
```

◆ 2.改成**鍵頭函式** (無須bind綁定)

```
updateState  =  ( )  =>  {
   // 鍵頭函式裡的this就等於這個組件
   this.setState (
      {
         text: 'Hello React',
         count: this.state.count + 1
      }
   );
}
```

主題3：React Component props

- **props** 接收從父組件上面**傳遞**下來子組件的屬性

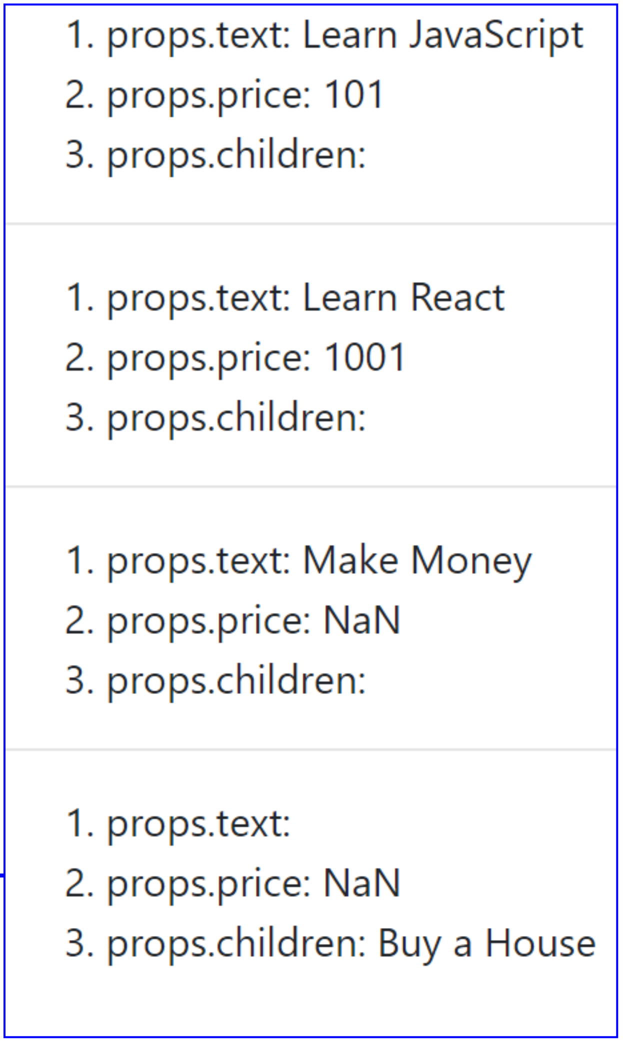
◆ 上層組件(List2)

```
// text、price 傳入子元件的屬性稱為 props
class List2 extends Component {
   render( ) {
      return (
         <div>
            <Item2 text="Learn JavaScript" price={100}/>
            <hr/>
            <Item2 text="Learn React" price="100"/>
            <hr/>
            <Item2 text="Make Money"/>
            <hr/>
            <Item2>Buy a House</Item2>
         </div>
      );
   }
}
```

主題3：React Component props

◆ 下層組件(Item2)

```
// props 接收從父組件上面傳下來子組件的屬性
class Item2 extends Component {
  render(){
    return (
      <ol>
       <li>props.text: {this.props.text}</li>
       <li>props.price: {this.props.price + 1}</li>
       <li>props.children: {this.props.children}</li>
      </ol>
    )
  }
};
```



```
1. props.text: Learn JavaScript
2. props.price: 101
3. props.children:

1. props.text: Learn React
2. props.price: 1001
3. props.children:

1. props.text: Make Money
2. props.price: NaN
3. props.children:

1. props.text:
2. props.price: NaN
3. props.children: Buy a House
```

主題4：React Component state 解構

◆ 建立 state

```
state = {
    count : 0,
    step: 1
};
```

◆解構state欄位

```
render() {
    // 解構state欄位
    const {count, step} = this.state;
    return (
        <div>
            <h1>Step： {step}</h1>
            <h1>Counter： {count}</h1>
            <button onClick={this.counterAdd}>Counter Add
Step + {step}</button>
        </div>
    );
}
```

第四章
React Component 父子組件

- 主題1：React Component props
- 主題2：React Component state
- 主題3：React Component ref

主題1：React Component props

◆ 上層父組件宣告引入子組件 (有傳入)

```
<CounterProps1 initCount={0} />
```

◆ 下層組件一般類別欄位透過 **this.props** 取得上層組件的值

```
state = {
    count : this.props.initCount,
    step: 1
};
```

1. **props** 指定初始 state 透過 **constructor**

```
constructor(props) {
    super(props);
    this.state = {
        count : props.initCount,
        step: 1
    }
}
```

主題1：React Component props

◆ 上層父組件宣告引入子組件 (沒有傳入)

```
<CounterProps2/>
```

2. **props** 透過 **defaultProps** 指定 **props**
由於上層組件常有可能忘了指定屬性，導致下層組件抓不到props裡面的欄值
可透過 defaultProps 指定初始 props 欄位值

```
// 類別內
static defaultProps = {
    initCount: 0
};
```

```
// 類別外
// 須寫在類別下方，因為需先有類別，透過類別名稱來指定初始值
CounterProps2.defaultProps = {
    initCount: 10
};
```

主題1：React Component props

◆ 上層父組件宣告引入子組件 (傳入錯誤型態)

```
<CounterProps3 initCount={'0'}/>
```

3. **props** 用 **propTypes** 檢查 **props** 型別

傳入值的型態錯誤將顯示以下訊息

Warning: Failed prop type: Invalid prop `initCount` of type `string` supplied to `CounterProps3`, expected `number`.

```
    // 類別內
    static propTypes = {
        initCount: PropTypes.number
    }
```

```
// 類別外
CounterProps3.propTypes = {
    initCount: PropTypes.number
};
```

主題2：React Component state

- Component state

React Component **setState** 本身為**非同步**

有以下兩種同步設置方式 (仍保有併發更新效率特性)

1. **setState** 可傳入 **arrow function** 改為同步更新

state => 後面接 “小括弧( )” 內的 “大括弧{ }" 表示回傳的是物件

```
this.setState( state => ({
    count: state.count + 1
}));
```

2. **setState** 可傳入第二個參數 **callback function** (回調函數)

可確保 setState 確實有更新完才呼叫別的動作

```
this.setState( state =>
    ({ count: state.count + 1 }) ,
    ( ) => {
        console.log("--- setStateOne ---");
        this.printCount();
    }
)
```

主題3：React Component ref

- Component ref

透過ref屬性找到DOM,JSX元素可以有ref屬性

1. ref 指定"字串" (官方棄用)

'refs' is deprecated. refs 字串寫法已被棄用!

```
<input type="text" ref="myInput"/>
<button onClick={this.showInputText}>show input text</button>
```

```
  showInputText = () => {
     this.refs.myInput.focus();
     console.log("this.refs.myInput.value:", this.refs.myInput.value);
  }
```

主題3：React Component ref

2. ref 指定"**函數**" (官方建議)

```
<input type="text" ref = {this.setRef} />
<button onClick={this.showInputText}>show input text</button>
```

```
// function 參數就是input元素
setRef = (input) => {
    this.myInput = input;
};

showInputText = () => {
    this.myInput.focus();
    console.log("refs function:", this.myInput.value);
};
```

主題3：React Component ref

3. ref 透過 createRef 建立DOM參照 (官方建議)

createRef( ) 籍由 **import createRef** 使用

```
<input type="text" ref={this.myInput}/>
<button onClick={this.showInputText}>show input text</button>
```

```
import React, { Component, createRef } from 'react';

myInput = createRef( );

showInputText = () => {
   this.myInput.current.focus();
   console.log("this.myInput.current.value:", this.myInput.current.value);
};
```

第五章
React Component 父子組件"溝通"

- 主題1：父子組件溝通
- 主題2：樣式控制 className 與 style
- 主題3：組件有三種

主題1：父子組件溝通

1. 父子組件"溝通"方式一

父傳子：子元件透過 "**props**" 呼叫父組件函式

子傳父：父元件透過 "**ref**" 呼叫子組件函式

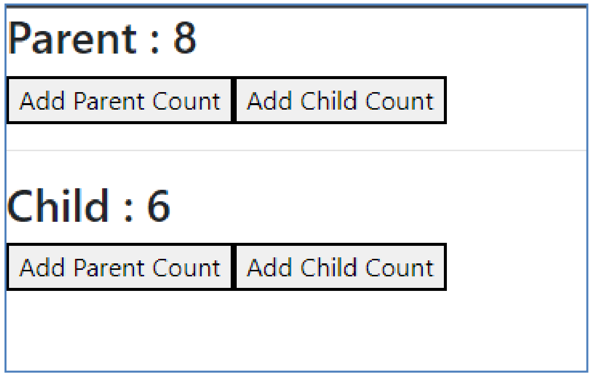
```
<h3>Parent : {this.state.count}</h3>
<button onClick={this.addParentCount}>Add Parent Count</button>
<button onClick={this.addChildCount}>Add Child Count</button>
<hr/>
<Child ref={this.childRef}  addParentCount={this.addParentCount}/>
```

```
state = {
   count: 0
};

addParentCount = () => {
   console.log("Hello Parent !");
   this.setState({
      count: this.state.count + 1
   });
};

childRef = createRef(  );

addChildCount = () => {
   // 子傳父:父組件使用 "ref",透過createRef()建立對子組件的參照,再透過參照來呼叫子組件函式addChildCount()
   this.childRef.current.addChildCount( );
};
```

主題1：父子組件溝通

```
 <h3>Child : {this.state.count}</h3>
 {/* 父傳子:子組件透過 "props" 呼叫父組件函式 this.props.addParentCount */}
<button onClick={this.props.addParentCount}>Add Parent Count</button>
<button onClick={this.addChildCount}>Add Child Count</button>
```

```
state = {
   count: 0
};

addChildCount = () => {
   console.log("Hello Child !");
   this.setState({
      count: this.state.count + 1
   });
};
```


Parent : 8

Child : 6

主題1：父子組件溝通

2. 父子組件"溝通"方式二
完全由父組件來主導全部的 state 儲存至 props 操作子組件
子組件再透過 props 解構"取值"、"取函數"

```
const parentInfo = {
   childCount: this.state.childCount,
   addParentCount: this.addParentCount,
   addChildCount: this.addChildCount
};

return (
   <div>
      <h3>Parent : {this.state.count}</h3>
      <button onClick={this.addParentCount}>Add Parent Count</button>
      <button onClick={this.addChildCount}>Add Child Count</button>
      <hr/>
      <Child parentInfo={parentInfo} />
   </div>
);
```

主題1：父子組件溝通

所有欄位、函數統一在父組件管理

```
state = {
    count: 0,
    childCount: 0
};

addParentCount = () => {
    this.setState({
        count: this.state.count + 1
    });
};

addChildCount = () => {
    this.setState({
        childCount: this.state.childCount + 1
    });
};
```

主題1：父子組件溝通

子組件再透過 props 解構"取值"、"取函數"

```
// 解構父層組件 props
const {childCount, addParentCount, addChildCount} = this.props.parentInfo;

return (
   <div>
      <h3>Child : {childCount}</h3>
      <button onClick={addParentCount}>Add Parent Count</button>
      <button onClick={addChildCount}>Add Child Count</button>
   </div>
);
```

主題2：樣式控制 className 與 style

```
state = {
   visible: true
};

toggle = () => {
   this.setState({
      visible: !this.state.visible
   });
};
```

1. 三元運算子、判斷式

```
const { visible } = this.state;
<button onClick={this.toggle}>Toggle</button>
{ visible ?  <img src={reactImg}/>  :  null }
{ visible && <img src={reactImg}/> }
```

2. 控制 style JSX 物件

```
 // block:顯示、none:不顯示
const styleObj = { display: visible ? 'block' : 'none' };
<img style={styleObj} src={reactImg}/>
<img style={ { display: visible ? 'block' : 'none' } } src={reactImg}/>
```

主題2：樣式控制 className 與 style

3. 控制 className

style.css

```css
.imageShow {
   display: block;
}

.imageHide {
   display: none;
}
```

```jsx
import '../CSS/style.css';

// 前後`頓號`表示為字串模版，透過${ }抓值
// const classObj = visible ? 'imageShow' : 'imageHide';
const classObj = `${ visible ? 'imageShow' : 'imageHide' }`;

<img className={classObj}  src={reactImg}/>
<img className={ `${ visible ? 'imageShow' : 'imageHide' }`}  src={reactImg}/>
```

主題3：組件有三種

- class Component 一般組件 (extends Component)
- PureComponent 效率組件 (extends PureComponent)

1. PureComponent、Component 差異在效能上，當上層元件所傳入下層元件的 **props 中的值若 "未改變" 的話**，當子組件**是 PureComponent 時不會重新 render** 但 Component、Functional Component 都會重新 render 效率較差。

2. PureComponent 也會比對自身的 state
shallow compare 淺層比較 (只比較 **state 的第一層**) **有差異就會重新 render**

```
import React, { Component, PureComponent } from 'react‘;
```

主題3：組件有三種

- Functional Component 函數組件 (Statless 無狀態)

1. 函數元件**沒有 state** 可操作故稱為**無狀態 Statless**

2. 若Component裡**沒有state 也沒有其它函式**就可以使用 Functional Component 來取代

```
class ProgressPercent extends PureComponent {

  state = {  };

  render( ) {
    console.count('Child PureComponent render!');
    const { childValue } = this.props;
    return (
      <div>{childValue}</div>
    );
  }
}
```

```
const ProgressPercent = (props) => {
  // 函式直接取代 render
  // render() {
    console.count('Child Functional Component render!');
    const { childValue } = props;
    return (
      <div>{childValue}</div>
    );
  // }
}
```

第六章
HOC、List Rendering、表單處理

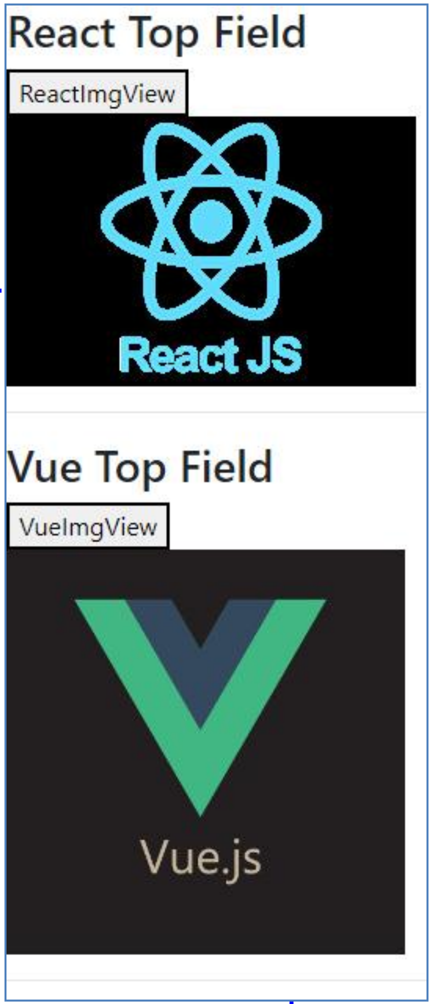
- 主題1：高階組件HOC (Higher Order Component)
- 主題2：列表渲染 List Rendering
- 主題3：表單處理

主題1：高階組件HOC (Higher Order Component)

● 以Component為輸入的Component

1. **相同邏輯**組件套用在**不同畫面**組件(邏輯共用畫面不同)

```
import withOpen from './hocs/withOpen';       // 父組件:邏輯(HOC)
import ReactImgView from './ReactImgView';  // 子組件:畫面
import VueImgView from './VueImgView';       // 子組件:畫面

// 相同邏輯組件套用在不同畫面組件(邏輯共用畫面不同)
//上層"共用"父組件:withOpen，下層"個別"子組件:ReactImgView、VueImgView
const ReactImgViewWithOpen = withOpen(ReactImgView);
const VueImgViewWithOpen = withOpen(VueImgView);

// HocAppOne 最上層組件
const HocAppOne = () => (
   <div>
      <ReactImgViewWithOpen topField={'React Top Field'}/>
      <hr/>
      <VueImgViewWithOpen topField={'Vue Top Field'}/>
      <hr/>
   </div>
);
```



```
// WrappedComponent 傳入子組件 ReactImgView、VueImgView
const withOpen = (WrappedComponent) => {
  // 回傳一個組件結果
  // class extends Component(Anonymous class匿名類別)
  return class extends Component {

    state = {
      open: false
    };

    toggle = () => {
      this.setState({
        open: !this.state.open
      });
    };
    /*
    {...this.props} 必須傳入
    連同最上層組件(HocAppOne topField),在傳入給下層組件時才會繼續傳遞下層組件
    */
    render() {
      return (
        <WrappedComponent
          {...this.props}
          open = {this.state.open }
          toggleOpen = {this.toggle}
        />
      )
    }
  };
};
```

主題1：高階組件HOC (Higher Order Component)

```
class ReactImgView extends Component {

    // 共用HOC組件'hocs/withOpen.js'(open, toggleOpen)
    // HocAppOne 最上層組件(topField)
    render() {
        const { open, toggleOpen, topField } = this.props;
        return (
            <div>
                <h3>{topField}</h3>
                <button onClick={toggleOpen}>ReactImgView</button>
                <br/>
                {open && <img src={reactImg}/>}
            </div>
        );
    }
}
```

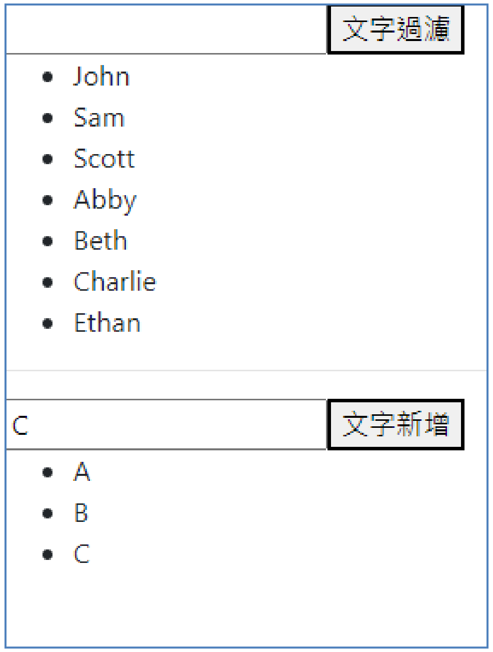
主題1：高階組件HOC (Higher Order Component)

```
class VueImgView extends Component {

  // 共用HOC組件'hocs/withOpen.js'(open, toggleOpen)
  // HocAppOne 最上層組件(topField)
  render() {
    const { open, toggleOpen, topField } = this.props;
    return (
      <div>
        <h3>{topField}</h3>
        <button onClick={toggleOpen}>VueImgView</button>
        <br/>
        {open && <img src={vueImg}/>}
      </div>
    );
  }
}
```

主題1：高階組件HOC (Higher Order Component)

2. 不同邏輯組件套用在相同畫面組件(邏輯不同畫面共用)

```
import withFilterText from './hocs/withFilterText';     // 父組件:邏輯
import withAddText from './hocs/withAddText';           // 父組件:邏輯
import CommonViewList from './CommonViewList';  // 子組件:畫面(HOC)

// 不同邏輯組件套用在相同畫面組件(邏輯不同畫面共用)
// 上層"個別"父組件:withFilterText、withAddText，下層"共用"子組件:CommonViewList
const FilterTextList = withFilterText(CommonViewList);
const AddTextList = withAddText(CommonViewList);

// HocAppTwo 最上層組件
const HocAppTwo = () => (
   <div>
      <FilterTextList btnText={'文字過濾'}/>
      <hr/>
      <AddTextList btnText={'文字新增'}/>
   </div>
);
```



- John
- Sam
- Scott
- Abby
- Beth
- Charlie
- Ethan

- A
- B
- C

```
// WrappedComponent 傳入子組件 CommonViewList
export default WrappedComponent => class extends Component {

  state = { items: names, text: '' };
  onChangeText = e => {
    this.setState({
      text: e.target.value
    });
  };
  // 文字過濾
  onSubmit = e => {
    e.preventDefault();
    const {text} = this.state;
    const filterItem = names.filter(name => name.includes(text));
    this.setState({
      items: filterItem
    });
  };

  render() {
    const {items, text} = this.state;
    return <WrappedComponent
          {...this.props}
          items={items}
          text={text}
          onChangeText={this.onChangeText}
          onSubmit={this.onSubmit}
    />
  }
};
```

```
// WrappedComponent 傳入子組件 CommonViewList
export default WrappedComponent => class extends Component {

  state = { items: [], text: '' };
  onChangeText = e => {
    this.setState({
      text: e.target.value
    });
  };
  // 文字新增
  onSubmit = e => {
    e.preventDefault();
    const {text, items} = this.state;
    const appendItem = [ ...items, text ];
    this.setState({
      items: appendItem
    });
  };

  render() {
    const {items, text} = this.state;
    return <WrappedComponent
        {...this.props}
        items={items}
        text={text}
        onChangeText={this.onChangeText}
        onSubmit={this.onSubmit}
    />
  }
};
```

主題1：高階組件HOC (Higher Order Component)

```
class CommonViewList extends Component {
    render() {
        // HocAppTwo最上層組件(btnText)
        const {text, items, onChangeText, onSubmit, btnText} = this.props;
        return (
            <div>
                <form onSubmit={onSubmit}>
                    <input value={text} onChange={onChangeText}/>
                    <button type="submit">{btnText}</button>
                </form>
                <ul>
                    {
                        items.map( item => (
                            <li>{item}</li>
                        ))
                    }
                </ul>
            </div>
        );
    }
}
```

主題2：列表渲染 List Rendering

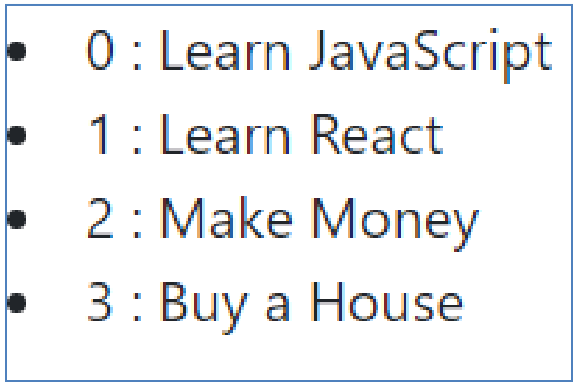
- List 陣列元素走訪

1. 文字陣列 map 走訪元素

```
const steps = [
    'Learn JavaScript',
    'Learn React',
    'Make Money',
    'Buy a House'
];
```

```
{steps.map(
    // 參數一:陣列元素、參數二:陣列索引值
   (text, index) => <Item> {index} : {text} </Item>
)}
```

```
// 下層組件
class Item extends Component {
    render(){
        return (
            <li> {this.props.children} </li>
        )
    }
}
```



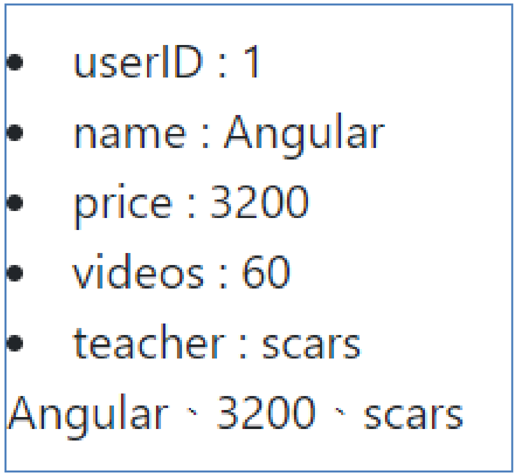
```
•   0 : Learn JavaScript
•   1 : Learn React
•   2 : Make Money
•   3 : Buy a House
```

主題2：列表渲染 List Rendering

2. **物件 key/value** map走訪元素

```
const infoOne = { userID: 1, name: 'Angular', price: 3200, videos: 60, teacher: 'scars'};
```

```
{Object.keys(infoOne).map( (infoKey) => {
        const infoValue = infoOne[infoKey];
        return <Item> {infoKey} : {infoValue} </Item>;
    }
 )}

{/* 透過物件欄位抓值 */}
{infoOne.name}、{infoOne.price}、{infoOne.teacher}
```



- userID : 1
- name : Angular
- price : 3200
- videos : 60
- teacher : scars

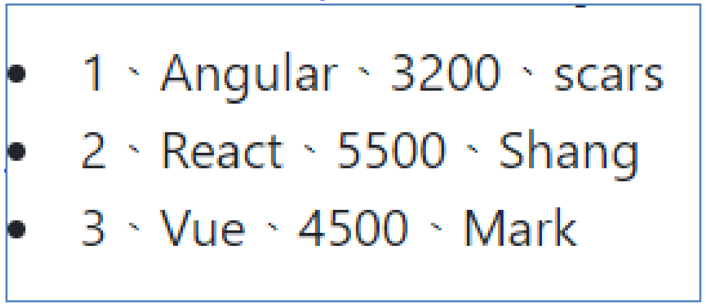
Angular、3200、scars

主題2：列表渲染 List Rendering

3. 物件陣列 map走訪元素

```
const infoOne = { userID: 1, name: 'Angular', price: 3200, videos: 60, teacher: 'scars'};
const infoTwo = { userID: 2, name: 'React', price: 5500, videos: 75, teacher: 'Shang' };
const infoThree = { userID: 3, name: 'Vue', price: 4500, videos: 45, teacher: 'Mark' };
const infos = [infoOne, infoTwo, infoThree];
```

```
{infos.map(
    (info) => <Item>
        {info.userID}、{info.name}、 {info.price}、 {info.teacher}
    </Item>
)}
```



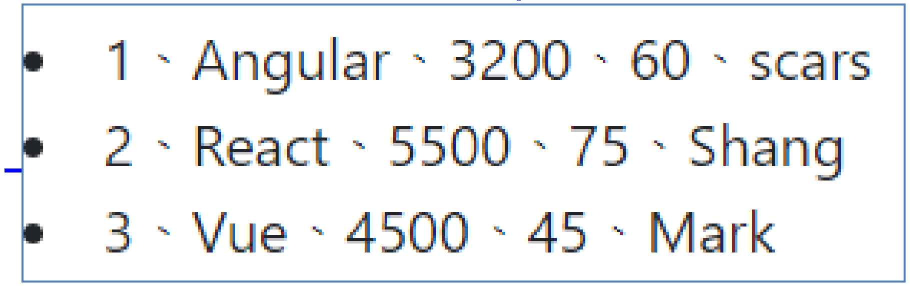
```
• 1、Angular、3200、scars
• 2、React、5500、Shang
• 3、Vue、4500、Mark
```

主題2：列表渲染 List Rendering

4. 物件陣列解構 map走訪元素

```
const infoOne = { userID: 1, name: 'Angular', price: 3200, videos: 60, teacher: 'scars'};
const infoTwo = { userID: 2, name: 'React', price: 5500, videos: 75, teacher: 'Shang' };
const infoThree = { userID: 3, name: 'Vue', price: 4500, videos: 45, teacher: 'Mark' };
const infos = [infoOne, infoTwo, infoThree];
```

```
{infos.map(
      ({userID,name,price,videos,teacher}) => <Item>
            {userID}、{name}、{price}、{videos}、{teacher}
      </Item>
)}
```



- 1、Angular、3200、60、scars
- 2、React、5500、75、Shang
- 3、Vue、4500、45、Mark

主題2：列表渲染 List Rendering

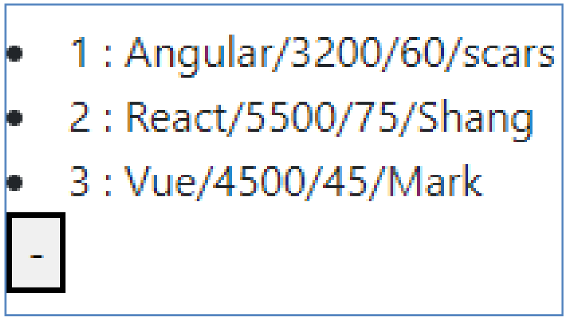
5. 物件陣列 map走訪元素組件 key屬性

- 組件屬性key作用：讓每一個 render 的**組件物件**都認得**與資料的連結**
- 移除第一個 ItemObj 的時候，其它 ItemObj 不該重新被 render , 使用key (須為唯一值)

  並且搭配 **PureComponent**
- **不因該使用陣列索引值 index 當作key** (只能避免Warning，但Component一樣會render)

  必須要**從物件挑選唯一值**資料欄位**當做key值**
- 若未宣告屬性key將顯示警告訊息 (Warning: Each child in a list **should have a unique "key" prop**.)

```
{this.state.list.map(
 (info, index) =>
            <PureItem
                key={info.userID}
                info={info}
                infoConent={`${info.name}/${info.price}/${info.videos}/${info.teacher}`}
            />
)}
```



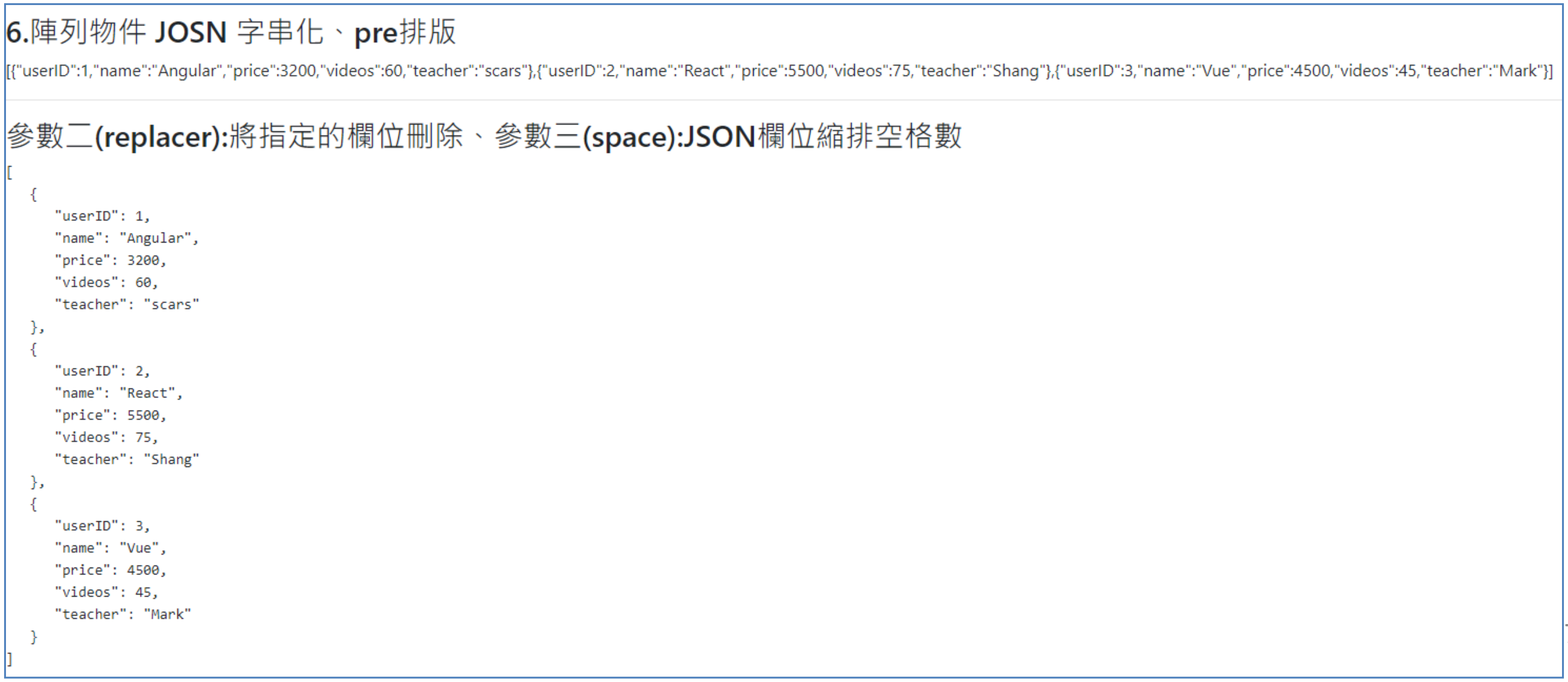
- 1 : Angular/3200/60/scars
- 2 : React/5500/75/Shang
- 3 : Vue/4500/45/Mark

主題2：列表渲染 List Rendering

6. 陣列物件 JOSN 字串化、pre排版

參數二(replacer):將指定的欄位刪除、參數三(space):JSON欄位縮排空格數

```
{JSON.stringify(infos)}

<pre> {JSON.stringify(infos, null, 3)} </pre>
```


6.陣列物件 JOSN 字串化、pre排版

```
[{"userID":1,"name":"Angular","price":3200,"videos":60,"teacher":"scars"},{"userID":2,"name":"React","price":5500,"videos":75,"teacher":"Shang"},{"userID":3,"name":"Vue","price":4500,"videos":45,"teacher":"Mark"}]
```

參數二(replacer):將指定的欄位刪除、參數三(space):JSON欄位縮排空格數

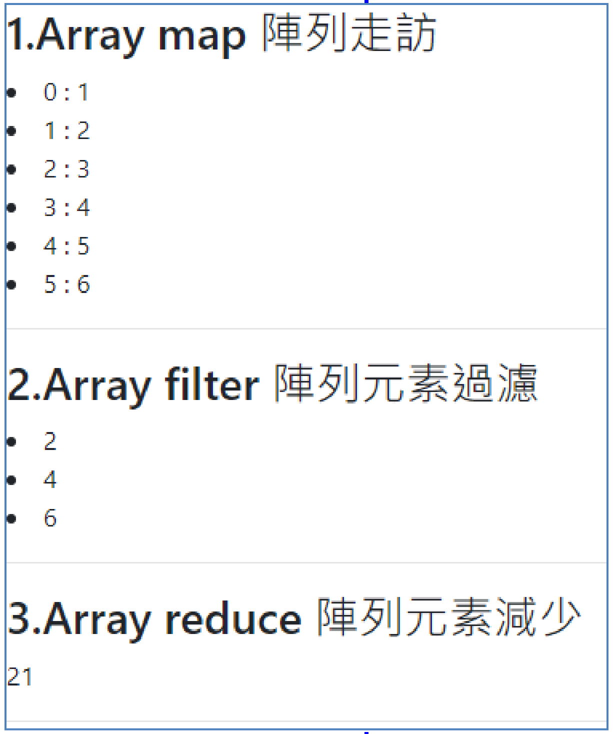
```
[
   {
      "userID": 1,
      "name": "Angular",
      "price": 3200,
      "videos": 60,
      "teacher": "scars"
   },
   {
      "userID": 2,
      "name": "React",
      "price": 5500,
      "videos": 75,
      "teacher": "Shang"
   },
   {
      "userID": 3,
      "name": "Vue",
      "price": 4500,
      "videos": 45,
      "teacher": "Mark"
   }
]
```

主題2：列表渲染 List Rendering

● 陣列的內建方法

map、**filter**、**reduce** 函式<u>都不會影響到原有陣列的值</u>

```
const array = [1,2,3,4,5,6];

// 1.Array map 陣列走訪
const result = array.map(
                (elem,idx) => <li>{`${idx} : ${elem}`}</li>
);

// 2.Array filter 陣列元素過濾
const result2 = array.filter( elem => elem % 2 === 0 )
                .map( elem => <li>{elem}</li> );

/*
3.Array reduce 陣列元素減少
accumulator 累加器
第二個參數代表初始值0
1 + 2 + ... + 6 = 21
*/
const result3 = array.reduce(
        (accumulator,elem,idx) => accumulator + elem, 0
);
```



1.Array map 陣列走訪

- 0 : 1
- 1 : 2
- 2 : 3
- 3 : 4
- 4 : 5
- 5 : 6

2.Array filter 陣列元素過濾

- 2
- 4
- 6

3.Array reduce 陣列元素減少

21

主題2：列表渲染 List Rendering

- 陣列的內建方法

下列函式**會影響**原先陣列的值

```
const array = [1,2,3,4,5,6];

console.log(array.pop());        // 取出最後一個元素 6
console.log(array.push(7));      // 放入元素至最後一個 7
console.log(array.shift());      // 取出第一個元素 1
console.log(array.unshift(0));  // 放入元素至第一個 0
console.log(array)  // [0, 2, 3, 4, 5, 7]

// 陣列元素反轉
array.reverse();
console.log(array)  // [7, 5, 4, 3, 2, 0]

// const newArr = array.slice().reverse();  // 陣列複制slice就不會影響到原來的陣列
const newArr = [...array]; // ES6陣列複製
console.log(newArr) // [7, 5, 4, 3, 2, 0]

array.sort();
console.log(array); // [0, 2, 3, 4, 5, 7]

array.splice(3); // 只留前面3個元素
console.log(array); // [0, 2, 3]
```

主題3：表單處理

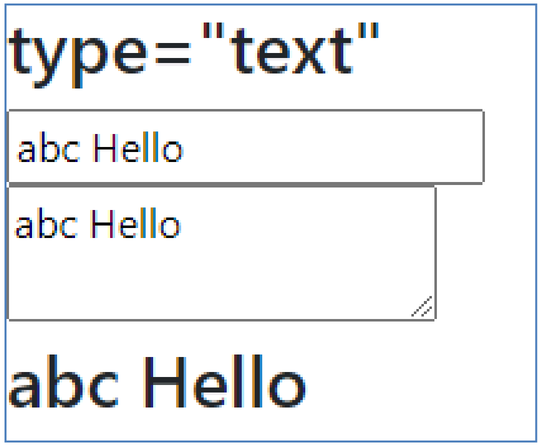
1. 文字、數字、下拉選單

```
state = {
    text: "abc",
    count: 0,
    rel: relations[0],
    // 下拉選單的值為字串
   relObj: `${relationsObjs[0].id}`
};
```

```
const {text, count, rel, relObj} = this.state;
```

```
<input type="text" value={text} onChange={this.onChangeText}/>
<br/>
<textarea value={text} onChange={this.onChangeText}/>
<h3>{text}</h3>
```

```
onChangeText = (even) => {
    this.setState ({
        text: even.target.value
    });
};
```



type="text"

abc Hello

abc Hello

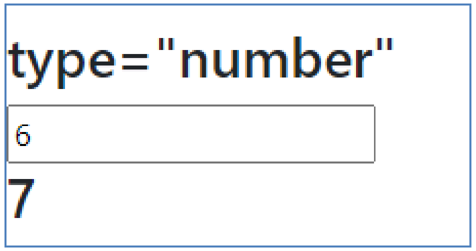
abc Hello

主題3：表單處理

```
<input type="number" value={count}
onChange={this.onChangeNumber}/>
<h3>{ count + 1 }</h3>
```

```
onChangeNumber = (even) => {
   this.setState ({
         // 取到的值都是字串型別所以要轉整數
         count: parseInt(even.target.value)
   });
};
```

```
// 下拉式選單資料
const relations = [ '父','母','子','女','妻','友' ];

const relationsObjs = [
   {label: '父', id: 0},
   {label: '母', id: 1},
   {label: '子', id: 2},
   {label: '女', id: 3},
   {label: '妻', id: 4},
   {label: '友', id: 5}
];
```


type="number"

6

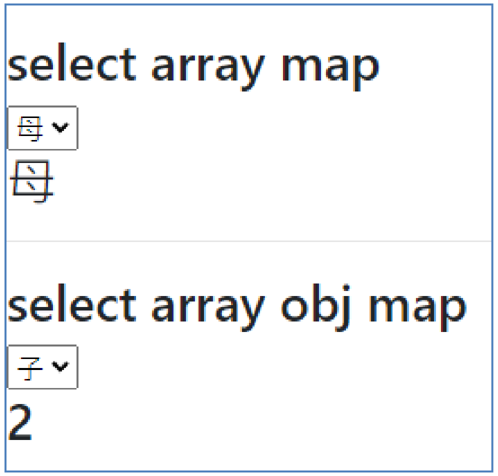
7

主題3：表單處理

```
<select onChange={this.onChangeSelect}>
    {relations.map( (element, idx) =>
        <option key={element} value={element}>{element}</option>
    )}
</select>
<h3>{rel}</h3>
```

```
<select onChange={this.onChangeSelectObj}>
   {relationsObjs.map( (element, idx) =>
        <option key={element.id} value={element.id}>{element.label}</option>
    )}
</select>
<h3>{relObj}</h3>
```

```
onChangeSelect = (even) => {
    this.setState({
        rel: even.target.value
    })
}

onChangeSelectObj = (even) => {
     this.setState({
         relObj: even.target.value
     })
}
```



select array map

母

母

select array obj map

子

2

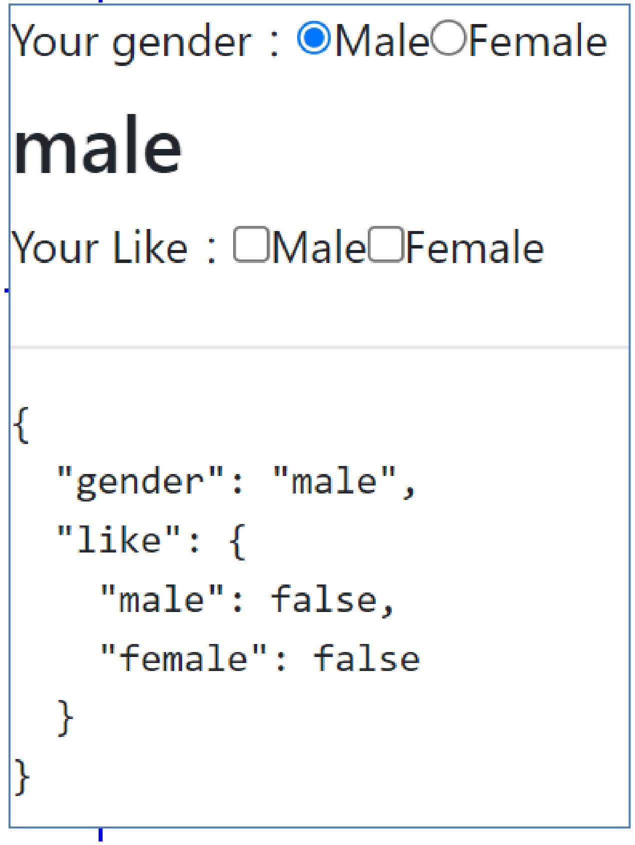
主題3：表單處理

2. 單選按鈕、複選框按鈕

```
state = {
    gender: 'male',
    like: {
        male: false,
        female: false
    }
};
```

```
const { gender, like } = this.state;
```

```
Your gender：
<input type="radio" value="male"
      onChange = {this.onChangeRadio}
      // radio 透過gender的值來控制是否要選中 checked
      // 也可透過傳統name='gender'控制單選
      checked= { gender === 'male' }
/>
<label>Male</label>

<input type="radio" value="female"
      onChange={this.onChangeRadio}
      checked= { gender === 'female' }
/>
<label>Female</label>

<h3>{gender}</h3>
```



Your gender：●Male○Female

male

Your Like：☐Male☐Female

```
{
  "gender": "male",
  "like": {
    "male": false,
    "female": false
  }
}
```

主題3：表單處理

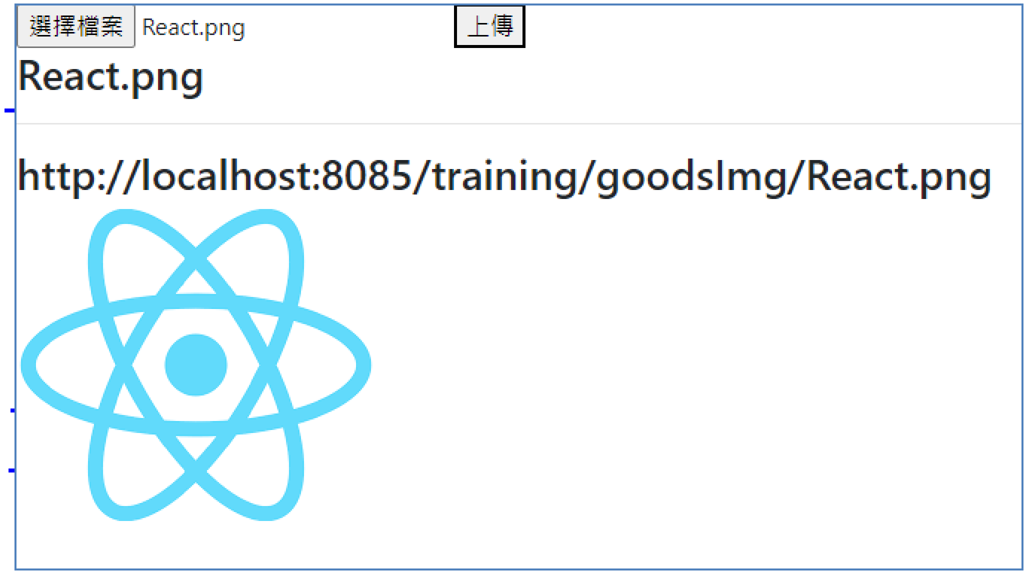
```
Your Like：
{/* checked={like.male} 可控制預設是否勾選 */}
<input type="checkbox" value="male" onChange={this.onChangeCheckbox} checked={like.male}/>
<label>Male</label>
<input type="checkbox" value="female" onChange={this.onChangeCheckbox} checked={like.female}/>
<label>Female</label>
```

```
onChangeRadio = (e) => {
     this.setState({
          gender: e.target.value
      })
};
onChangeCheckbox = (e) => {
      const key = e.target.value;
      this.setState( (state) => ( {
           like: {
              // 複制原先的 like 物件內容
              ...this.state.like,
              // 針對指定的 like 更新欄位值
              [key] : !state.like[key]
           }
      }));
};
```

主題3：表單處理

3. 檔案上傳與圖片預覽

```
state = {
    fileName: '',
    imgUrl: ''
};
```



React.png

http://localhost:8085/training/goodsImg/React.png

```
<form onSubmit={this.handleSubmit}>
    <input type="file" name='uploadFile' onChange={this.onChangeImg}/>
    <button type="submit">上傳</button>
</form>
<h3>{this.state.fileName}</h3>
<hr/>

<h3>{this.state.imgUrl}</h3>

<img src={this.state.imgUrl}/>
```

主題3：表單處理

```
onChangeImg = (e) => {
    const changFile = e.target.files;
    const changFileName = changFile[0].name;
    this.setState({fileName: changFileName});
};

handleSubmit = async(event) => {
    event.preventDefault(); // 防止瀏灠器預設submit跳頁
    const form = event.currentTarget;
    const uploadFile = form.uploadFile.files[0];

    // multipart
    const formData = new FormData();
    formData.append('fileName', this.state.fileName);
    formData.append('uploadFile', uploadFile);

    // call 後端API上傳檔案
    const imgUrl = await axios.post(apiUrl, formData, { timeout: 3000 })
    .then(rs => rs.data)
    .catch(error => { console.log(error); });

    this.setState({imgUrl: imgUrl});
};
```

主題3：表單處理

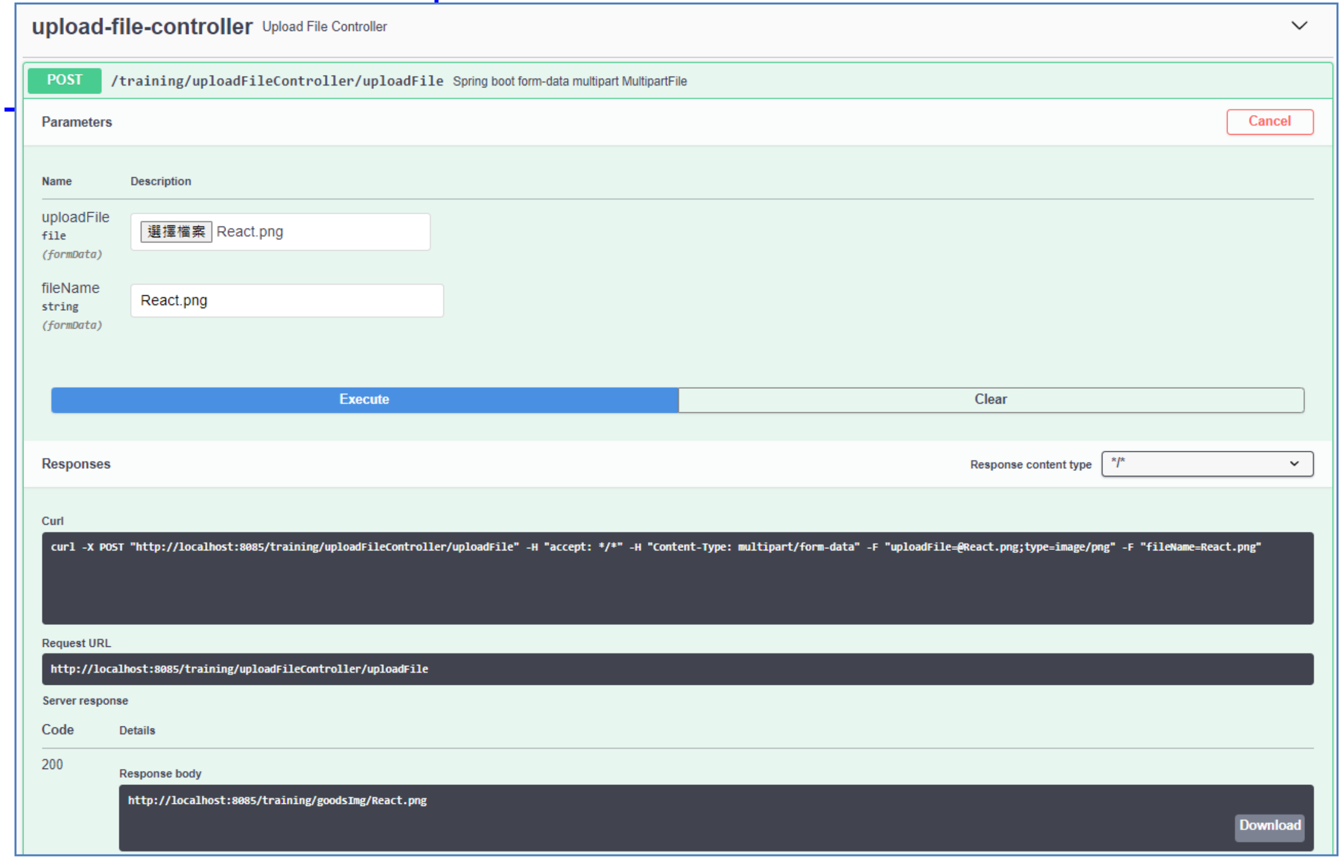
Spring boot form-data multipart MultipartFile

```
@CrossOrigin
@RestController
@RequestMapping("/uploadFileController")
public class UploadFileController {

    @ApiOperation(value = "Spring boot form-data multipart MultipartFile")
    @PostMapping(value = "/uploadFile", consumes = { MediaType.MULTIPART_FORM_DATA_VALUE })
    public ResponseEntity<String> createGoods(@ModelAttribute UploadFileVo uploadFileVo)
        throws IOException {
            // 複制檔案
            MultipartFile file = uploadFileVo.getUploadFile();
            // 上傳檔案原始名稱
            // String fileName = file.getOriginalFilename();
            // 前端傳入檔案名稱
            String fileName = uploadFileVo.getFileName();

            Files.copy(file.getInputStream(),
                        Paths.get("/home/VendingMachine/DrinksImage").resolve(fileName));

            return ResponseEntity.ok("http://localhost:8085/training/goodsImg/" + fileName);
    }

}
```

主題3：表單處理

org.springframework.web.multipart.**MultipartFile**

```
@Data
public class UploadFileVo {

    private String fileName;

    private MultipartFile uploadFile;

}
```



upload-file-controller Upload File Controller

POST /training/uploadFileController/uploadFile Spring boot form-data multipart MultipartFile

Parameters

Name	Description
uploadFile file (formData)	選擇檔案 React.png
fileName string (formData)	React.png

Responses

Response content type: */*

Curl

```
curl -X POST "http://localhost:8085/training/uploadFileController/uploadFile" -H "accept: */*" -H "Content-Type: multipart/form-data" -F "uploadFile=@React.png;type=image/png" -F "fileName=React.png"
```

Request URL

```
http://localhost:8085/training/uploadFileController/uploadFile
```

Server response

Code	Details
200	Response body: `http://localhost:8085/training/goodsImg/React.png`

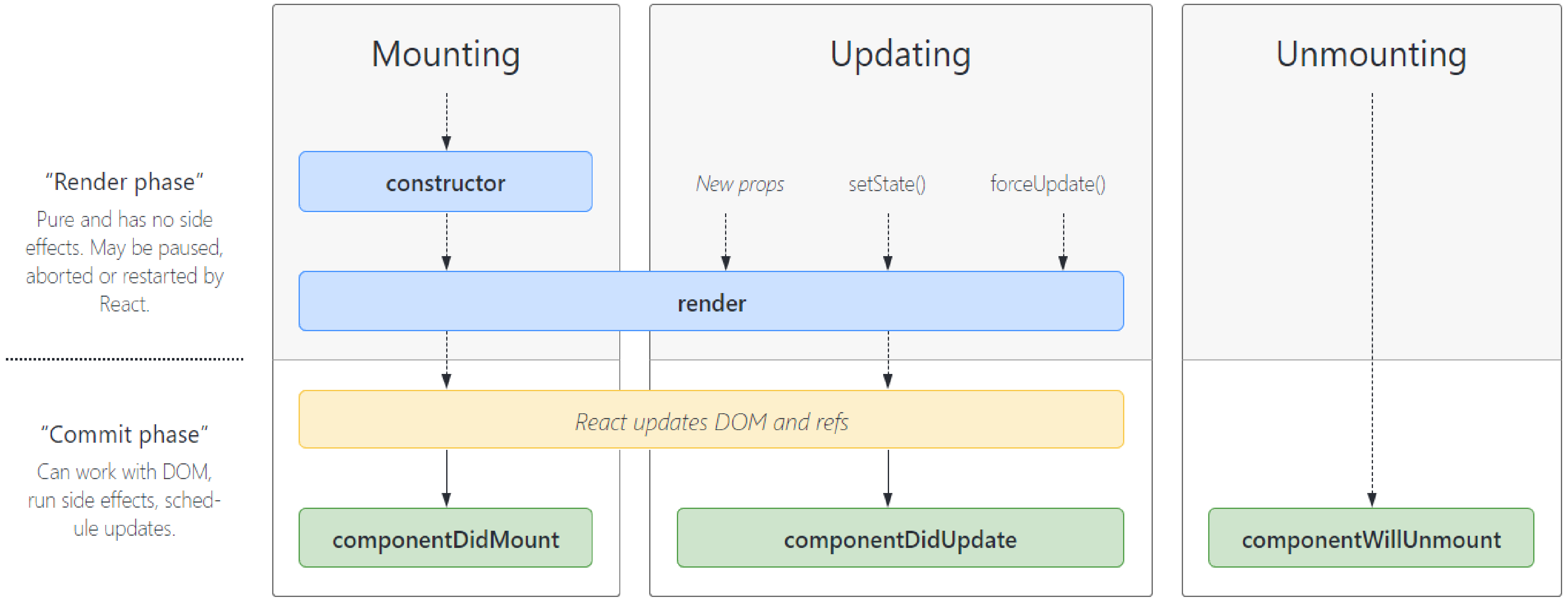
第七章
生命週期函式

- 主題1：constructor 建構函式
- 主題2：render 渲染函式
- 主題3：componentDidMount 組件掛載
- 主題4：componentDidUpdate 組件更新
- 主題5：componentWillUnmount 組件卸載

主題1：constructor 建構函式

- React 組件生命週期

https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



主題1：constructor 建構函式

- constructor 建構函式(Mounting：掛載) 規則、用法

1. 一定要呼叫 **super(props)**
2. 會傳入 **props** 參數
3. 指定 **this.state** 針對 state 做<u>初始化設計欄位</u>
4. 綁定自訂義函式**bind(this)**,可用箭頭函式取代,目的讓 this 為 Component 自已
5. 不要做 **this.setState**({})
6. 不要將**props 指定到 state 裡面** this.state = { count:props.count }

   每個組件都應該要有自已的初始欄位不該由父組件而來

   直接在 render() 函數裡使用 **this.props**
7. 不要去呼叫資料 **fetch ajax** 函式

主題1：constructor 建構函式

```
constructor(props){
       console.log("1.constructor 建構函式(Mounting:掛載)");
       super(props);

       // 針對state做初始化設計欄位
      this.state = { count: 0, users: [] };

      // 將handleClick綁定Component組件自已,或用箭頭函式取代
      this.handleClick = this.handleClick.bind(this);
}
```

```
// 若未透過constructor bind(this)綁定,則這邊的this指的是button它自已
handleClick() {
     this.setState({
          count: this.state.count+1
     },
     ( ) => {
        // setState可傳入第二參數Callback函式以確保State有更新後再執行別的事情
        console.log("setState!");
     });
};
```

主題2：render(渲染函式)

- render 渲染函式 (Mounting:掛載、Updating:更新)

1. 解構取值 this.props、this.state

2. 不要做 this.setState({})

3. 不要去呼叫資料 fetch ajax 函式

4. 專責做畫面呈現

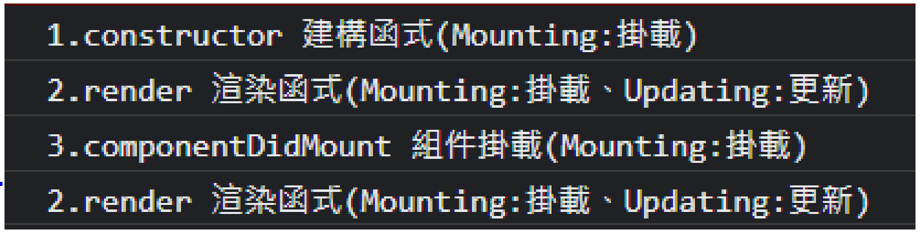
```
render( ) {
    console.log("2.render 渲染函式(Mounting:掛載、Updating:更新)");
    // this.props;
    const { users } = this.state;

    // 要回傳一個JSX元素
    // React16之後可以回傳陣列元素
    return [
            <div key={1}>
                <h3>{this.state.count}</h3>
                <button onClick={this.handleClick}>AddCount</button>
            </div>,
            <ul key={2}>
                {users.map((user) =>
                    <li key={user.id}> {user.id} : {user.name} / {user.address.zipcode}</li>
                )}
            </ul>
    ];
}
```

主題3：componentDidMount 組件掛載

- componentDidMount 組件掛載 (Mounting:掛載)

如果需要從後端請求資料的話，此處適合進行**網路請求**(network request)以及
新增監聽 addEventListener，可以在 **componentDidMount( )** 內呼叫 **setState( )**
這會觸發一次額外的 render，這會在瀏覽器更新螢幕之前發生。
在這個情況下，即使 render() 被呼叫兩次
這確保使用者不會看見這兩次 render 中過渡時期的 state。

```
componentDidMount() {
        console.log("3.componentDidMount 組件掛載(Mounting:掛載)");
        this.fetchList();
}

fetchList = async () => {
        const response = await fetch('http://jsonplaceholder.typicode.com/users');
        const data = await response.json();
        this.setState({
                users: data
        });
}
```



```
1.constructor 建構函式(Mounting:掛載)
2.render 渲染函式(Mounting:掛載、Updating:更新)
3.componentDidMount 組件掛載(Mounting:掛載)
2.render 渲染函式(Mounting:掛載、Updating:更新)
```

主題4：componentDidUpdate 組件更新

- componentDidUpdate 組件更新 (Updating:更新)

1. 可用於比較與前一次props、state的值

   用以判斷是否要更新 this.setState( ) 籍此再次觸發 render() 重新渲染畫面更新

2. prevProps：父層組件更新抓取前一次props

3. prevState：組件自身更新抓取前一次state

```
// 一、constructor 建構函式(Mounting)
constructor(props){
      console.log("1.constructor 建構函式(Mounting:掛載)");
      super(props);

      // 針對state做初始化設計欄位
      this.state = {
            userID: 1,
            userData: { }
      };
}
```

主題4：componentDidUpdate 組件更新

```
// 二、render 渲染函式(Mounting、Updating)
render() {
   console.log("2.render 渲染函式(Mounting :掛載、Updating:更新)");
   const {userID,userData} = this.state;
   return (
      <div>
         <h3>{userID}</h3>
         <button onClick={this.addUserID}>addUserID</button>
         <hr/>
         <form>
            <button type='submit'>revertUser</button>
         </form>
         <ul>
            <li>{userData.id}</li>
            <li>{userData.title}</li>
            <li>{userData.body}</li>
         </ul>
      </div>
   );
}
```

主題4：componentDidUpdate 組件更新

```
// 三、componentDidMount 組件掛載(Mounting)
// 組件掛載(Mounting)"新增監聽"
componentDidMount( ) {
    console.log("3.componentDidMount 組件掛載(Mounting:掛載)");
    window.addEventListener('submit', this.revertUser);
    this.fetchUser(1);
}
```

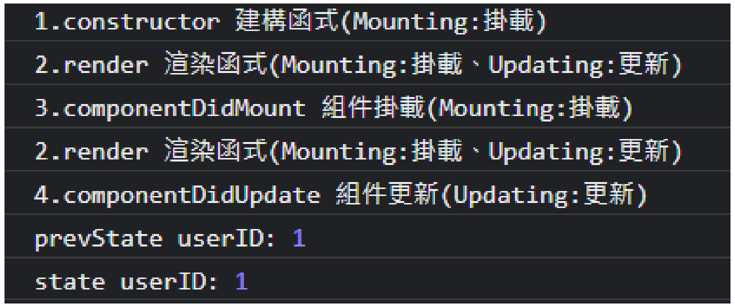
```
addUserID = () => {
      // 此處僅須更新state userID並交由componentDidUpdate判斷更新
      this.setState({
            userID: this.state.userID + 1
       });
}

revertUser = (e) => {
       e.preventDefault(); // 避免表單送出預設跳頁行為
       // 判斷若當下的user非1號才還原(避免componentDidUpdate被觸發)
       if(this.state.userID !== 1){
           this.setState({
               userID: 1
           });
       }
};

fetchUser = async (userID) => {
        const data = await fetch(`https://jsonplaceholder.typicode.com/posts/${userID}`).then(rs => rs.json());
        this.setState({
              userData: data
        });
};
```

主題4：componentDidUpdate 組件更新

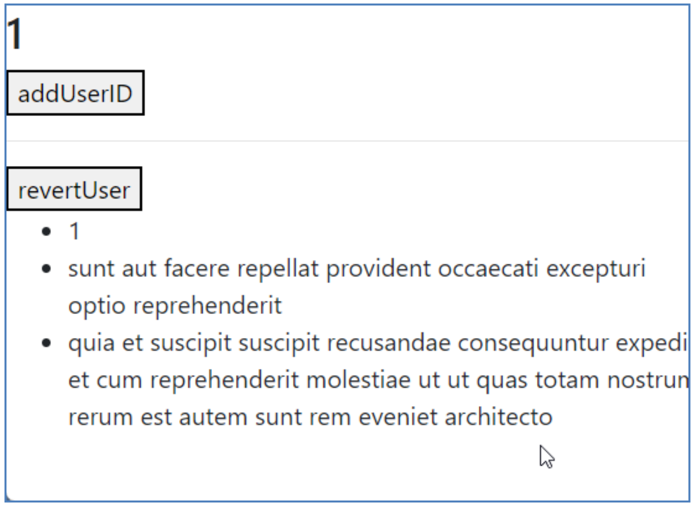
```
// 四、componentDidUpdate 組件更新(Updating)
componentDidUpdate(prevProps, prevState) {
    console.log("4.componentDidUpdate 組件更新(Updating :更新)");
    console.log("prevState userID:", prevState.userID);
    console.log("state userID:", this.state.userID);
    if(prevState.userID !== this.state.userID){
        console.log(“componentDidUpdate 組件更新(執行 fetchUser)");
        this.fetchUser(this.state.userID);
    }

    // 上一個與下一個 props 欄位值比較
    // if(prevProps.xxx !== this.props.xxx) { ... }
}
```

主題4：componentDidUpdate 組件更新

第一次載入
步驟1 ~ 步驟3 初始掛載、更新
步驟2、步驟4 於步驟3 this.setState( ) 時所觸發



```
1.constructor 建構函式(Mounting:掛載)
2.render 渲染函式(Mounting:掛載、Updating:更新)
3.componentDidMount 組件掛載(Mounting:掛載)
2.render 渲染函式(Mounting:掛載、Updating:更新)
4.componentDidUpdate 組件更新(Updating:更新)
prevState userID: 1
state userID: 1
```



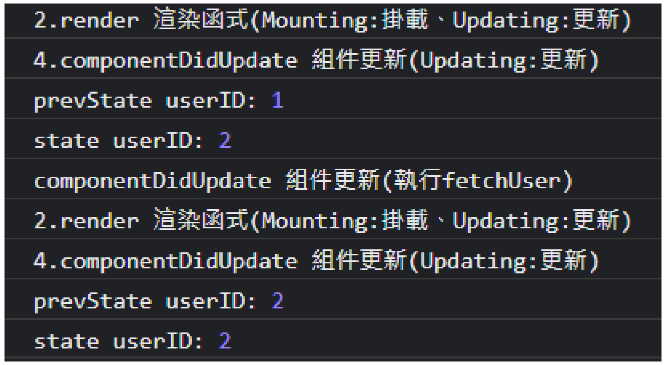
1

addUserID

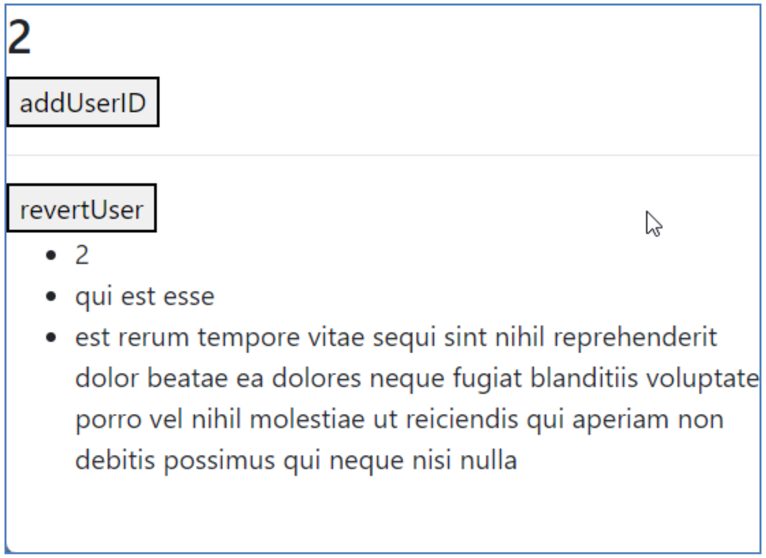
revertUser

- 1
- sunt aut facere repellat provident occaecati excepturi optio reprehenderit
- quia et suscipit suscipit recusandae consequuntur expedi et cum reprehenderit molestiae ut ut quas totam nostrun rerum est autem sunt rem eveniet architecto

點擊 addUserID 按鈕
步驟2、步驟4 於addUserID this.setState( ) 時所觸發
步驟2、步驟4 於前次步驟4 fetchUser 操作
this.setState( ) 時所再次觸發



```
2.render 渲染函式(Mounting:掛載、Updating:更新)
4.componentDidUpdate 組件更新(Updating:更新)
prevState userID: 1
state userID: 2
componentDidUpdate 組件更新(執行fetchUser)
2.render 渲染函式(Mounting:掛載、Updating:更新)
4.componentDidUpdate 組件更新(Updating:更新)
prevState userID: 2
state userID: 2
```



2

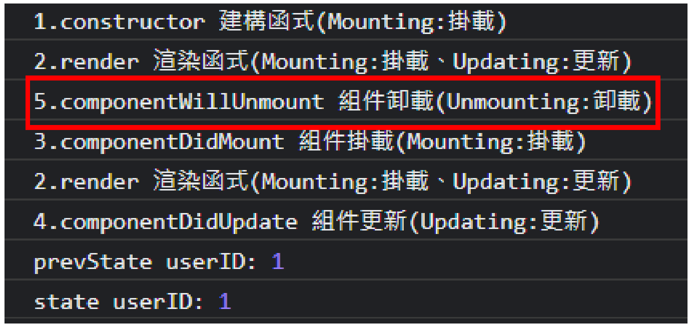
addUserID

revertUser

- 2
- qui est esse
- est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

主題5：componentWillUnmount 組件卸載

組件卸載(Unmounting)"移除監聽“
組件離開的時候呼叫
經過完整的三階段才會卸載
步驟1(Mounting)、步驟2(Updating)、步驟5(Unmounting)



```
1.constructor 建構函式(Mounting:掛載)
2.render 渲染函式(Mounting:掛載、Updating:更新)
5.componentWillUnmount 組件卸載(Unmounting:卸載)
3.componentDidMount 組件掛載(Mounting:掛載)
2.render 渲染函式(Mounting:掛載、Updating:更新)
4.componentDidUpdate 組件更新(Updating:更新)
prevState userID: 1
state userID: 1
```

```
// 五、組件卸載(Unmounting)"移除監聽"
// 組件離開的時候呼叫
componentWillUnmount( ){
   console.log("5.componentWillUnmount 組件卸載(Unmounting:卸載)");
   window.removeEventListener('submit', this.revertUser);
}
```

第八章
React16 功能

➢ 主題1：render 渲染函式(回傳類型)

➢ 主題2：Portal 傳送門

➢ 主題3：Context API

主題1：render 渲染函式(回傳類型)

1. React16之前最外面一定要包一個JSX元素(React elements)

React16.2 之後

2. Arrays and fragments

```
 return [ <h1>Arrays</h1>,  <div>div</div>, <button>button</button> ];

// 使用Fragment虛擬元素不會出現在最後的結果(讓React覺得只有一個元素)
return (
        <Fragment>
                <h1>Fragment</h1>
                <div>div</div>
                <button>button</button>
        </Fragment>
);

// babel
return (
        <>
              <h1>babel</h1>
              <div>div</div>
              <button>button</button>
        </>
);
```

主題1：render 渲染函式(回傳類型)

3. **Portals**(傳送門)
將元素render到指定的div裡

4. String and numbers
return **'String';**
return **42**;

5. Booleans or null
return **true**;
return **null**;

主題2：Portal 傳送門

● 將元素render到指定的div裡

```
import React, { Component } from 'react';
import LessonModal from './LessonModal';

class Lesson extends Component {

   state = {
      detailVisible: false
   };

   showMore = () => {
      this.setState({
         detailVisible: !this.state.detailVisible
      });
   };

   render() {
      const {detailVisible} = this.state;
      return (
         <div>
            <button onClick={this.showMore}>
               Show More
            </button>
            {detailVisible && <LessonModal/>}
         </div>
      );
   }
}

export default Lesson;
```

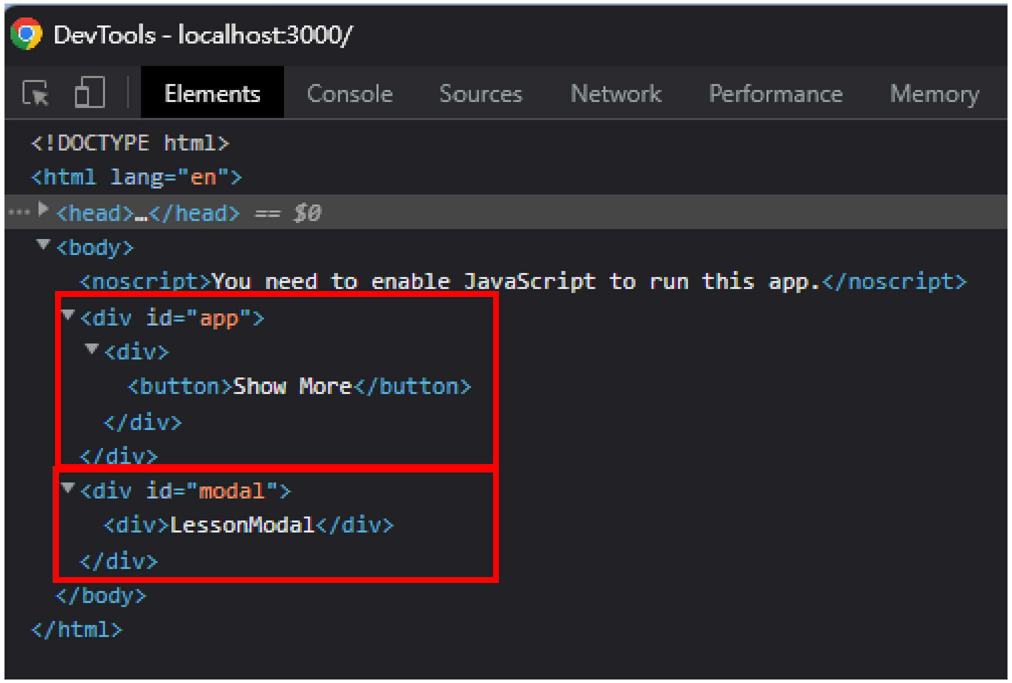
```
import React, { Component } from 'react';
import { createPortal } from 'react-dom';

class LessonModal extends Component {

   // render() { return <div>LessonModal</div> }

   render() {
      // 透過createPortal將元素render到指定的div裡
      return createPortal (
         <div>LessonModal</div>,
         document.getElementById('modal')
      );
   }

}

export default LessonModal;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import Lesson from './React16/Lesson';

ReactDOM.render(<Lesson/> ,
         document.getElementById('app'));
```

主題2：Portal 傳送門

- 將元素render到指定的div裡

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="app"></div>
    <div id="modal"></div>
  </body>
</html>
```

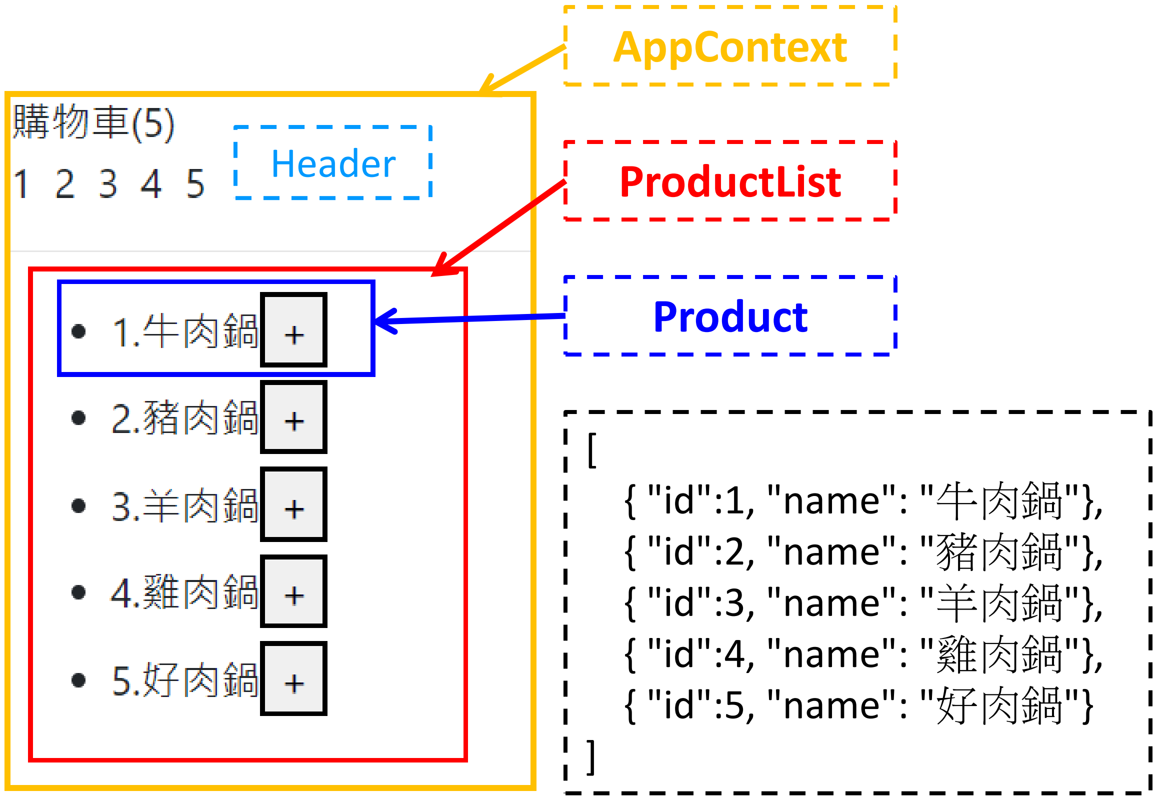

DevTools - localhost:3000/

```
<!DOCTYPE html>
<html lang="en">
··· ▶<head>…</head> == $0
  ▼<body>
      <noscript>You need to enable JavaScript to run this app.</noscript>
    ▼<div id="app">
      ▼<div>
          <button>Show More</button>
        </div>
      </div>
    ▼<div id="modal">
        <div>LessonModal</div>
      </div>
    </body>
</html>
```

主題3：Context API

- 跨越多個組件之間的溝通組件(連結串連上下文)

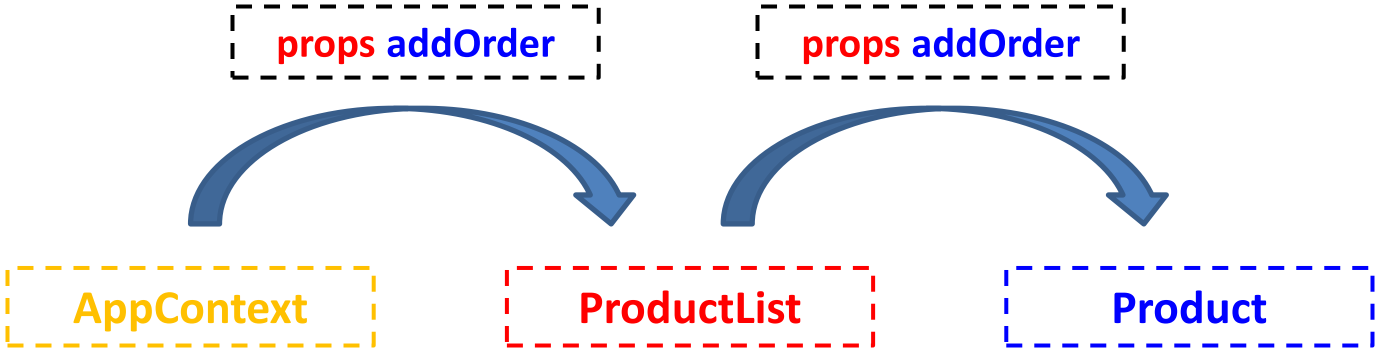
可跨越多個父子組件之間溝通傳遞資料(以模組的方式匯出、匯入)
達成避免中間層組件傳入自身用不到的功能



```
[
  { "id":1, "name": "牛肉鍋"},
  { "id":2, "name": "豬肉鍋"},
  { "id":3, "name": "羊肉鍋"},
  { "id":4, "name": "雞肉鍋"},
  { "id":5, "name": "好肉鍋"}
]
```

主題3：Context API

- addOrder 是在 AppContext (最上層父組件) 宣告的函式用以將商品編號加入購物車之中，但是為了在 Product (最下層子組件) 中使用 addOrder函式就必須透過 ProductList (中間層子組件) **<u>居中協助傳遞 porps</u>**
  但 addOrder 函式對於 ProductList 組件自身是用不到的



主題3：Context API

```
class AppContext extends Component {

  // 購物車訂單商品編號清單
  state = {
    orders: []
  };

  addOrder = (order) => {
    this.setState({
      orders: [...this.state.orders, order]
    });
  };

  render( ) {
    // AppContext → Header
    // AppContext → ProductList → Product
    return (
      <div>
        <Header orders={this.state.orders}/>
        <ProductList addOrder={this.addOrder}/>
      </div>
    );
  }
}
```

```
class Header extends Component {
  render() {

    const { orders } = this.props;

    return (
      <div>
        <span>購物車({orders.length})</span>
        <br/>
        {orders.map(
          (order) => (
            <div style={{display : 'inline-block'}}>
              <span> {order} &nbsp;</span>
            </div>
          )
        )}
        <hr/>
      </div>
    );
  }
}
```

主題3：Context API

```
class ProductList extends Component {

  render() {

    const products = productsData;
    const { addOrder } = this.props;

    return (
      <ul>
    {products.map(
      // AppContext → ProductList → Product
      // {...product} 將product的所有欄位(id、name)一次全部傳入
      // 對於Product而言addOrder並非自身要使用，但卻要協助傳遞
       product =>
          <Product {...product} key={product.id}
              addOrder={addOrder}/>
     )}
      </ul>
    );
  }
}
```

```
class Product extends Component {
  render() {

    const { id, name, addOrder } = this.props;

    return (
      // AppContext → ProductList → Product
      <li>
        <label>{id}.{name}</label>
        <button onClick={() => addOrder(id)}> + </button>
      </li>
    );

  }
}
```

主題3：Context API

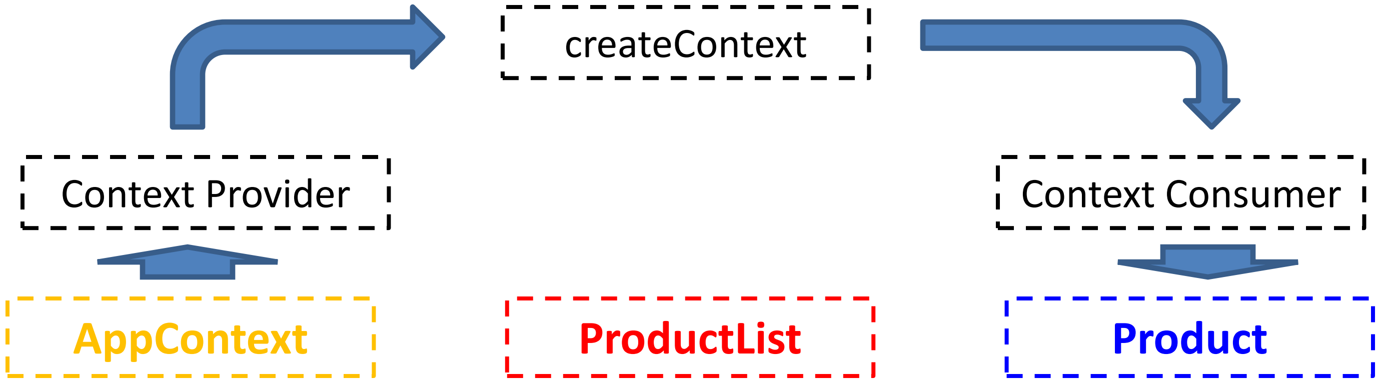
● 跨越多個組件之間的溝通組件(連結串連上下文)

可跨越多個父子組件之間溝通傳遞資料(以模組的方式匯出、匯入)

1. React 透過 **createContext** 建立要跨組件傳遞的功能(欄位、函式)

2. 父組件透過 Context **Provider** (提供者) 包裝欄位及函式並且透過 **value** 屬性傳入

3. 子組件透過 Context **Consumer** (消費者) 取得欄位及函式



主題3：Context API

```
class AppContext extends Component {

  // 購物車訂單商品編號清單
  state = {
    orders: []
  };

  addOrder = (order) => {
    this.setState({
      orders: [...this.state.orders, order]
    });
  };

  render() {
    // AppContext → ProductList → Product
    const { orders } = this.state;
    const contextValue = {
      orders,
      addOrder: this.addOrder
    };
    return (
      <div>
        <OrderContext.Provider value={contextValue}>
          <Header/>
          <ProductList/>
        </OrderContext.Provider>
      </div>
    );
  }
}
```

```
class Header extends Component {
  render() {
    return (
      // <OrderContext.Consumer> 裡面是一個函式所傳入的參數就是
<OrderContext.Provider> 的 value
      <div>
        <span>
          <OrderContext.Consumer>
            {(contextValue) => (
              `購物車(${contextValue.orders.length})`
            )}
          </OrderContext.Consumer>
        </span>
        <br/>
        <OrderContext.Consumer>
          {(contextValue) => (
            contextValue.orders.map(
              (order) => (
                <div style={{display : 'inline-block'}}>
                  <span> {order} &nbsp;</span>
                </div>
              )
            )
          )}
        </OrderContext.Consumer>
        <hr/>
      </div>
    );
  }
}
```

主題3：Context API

```
class ProductList extends Component {
  render() {
    const products = productsData;
    return (
      <ul>
        {products.map(
          // AppContext → ProductList → Product
          // 透過Context API 就不須要再轉一手傳入addOrder函數
          product => <Product key={product.id} {...product} />
        )}
      </ul>
    );
  }
}
```

```
import { createContext } from 'react';

// createContext並給予初始值
const context = createContext({
  orders: [],
  addOrder: () => {}
});

export default context;
```

```
class Product extends Component {
  render() {
    const { id, name } = this.props;
    return (
      // AppContext → ProductList → Product
      <li>
        <label>{id}.{name}</label>
        <OrderContext.Consumer>
          {(contextValue) => (
            <button onClick={ () =>
                contextValue.addOrder(id) }> +
             </button>
          )}
        </OrderContext.Consumer>
      </li>
    );
  }
}
```

第九章.React Hooks

- 主題1：Hooks useState (狀態管理)
- 主題2：Hooks useEffect (附加效果)
- 主題3：Hooks useContext (應用Context)
- 主題4：Hooks useRef (存取參照)
- 主題5：Hooks 綜合應用 TodoList

主題1：Hooks useState (狀態管理)

- React Hooks

React Hooks (16.8版) 於 2019/2/6 之後引入

https://reactjs.org/docs/hooks-intro.html

1. 使用函式組件 (**functional component**) 的方式來建構,而不用 **class component** 類別組件

2. React 團隊當初在設計 Hooks 的用意就在，使用函式組件(functional component)的時候

   也能夠自由的使用 **state**，也能夠做到**生命週期函式**可以做到的事情

3. 使用 Hooks 可以把相同的邏輯組織起來放在同一個地方

主題1：Hooks useState (狀態管理)

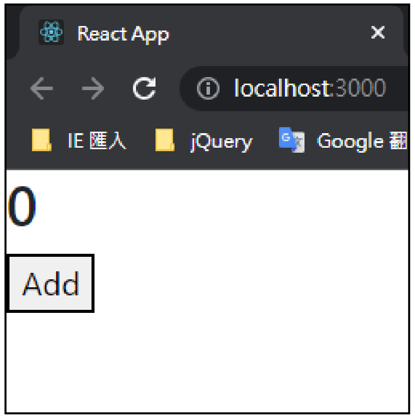
- Hooks useState (狀態管理)

使用函式組件 (functional component) 不能直接使用 state 必須透過 Hooks useState(狀態管理)

在使用 Hooks 時就可以使用函式組件(functional component) 來操作 state

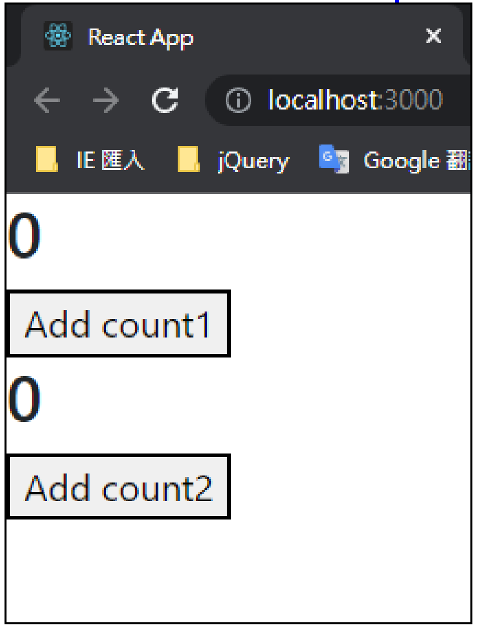
```
import React, { useState } from 'react';
```

```
const UseStateApp = (  ) => {
   // useState傳入初始的狀態(會回傳一個陣列)
   // 陣列中第一個值 (count) 代表目前的狀態
   // 第二個為更新狀態的函式(setCount)
   const [count, setCount] = useState(0);
   const addCount = () => {
      // 可傳入函式(傳入的參數為舊的state的值)
      setCount(count => count + 1);
   };
   // 函式組件使用return直接回傳結果渲染畫面,取代傳統類別組件render()函式
   return (
      <div>
         <h3>{count}</h3>
         <button onClick={addCount}>Add</button>
      </div>
   );
};
```



主題1：Hooks useState (狀態管理)

```
const UseStateApp2 = ( ) => {
  // state裡存的是一個多欄位的物件
  const [{count1, count2}, setState] = useState( {count1:0, count2:0} );

  const addCountOne = ( ) => {
    // Hooks useState 無法達成部份欄位更新,所以必須要先將原先的 state 物件保留並且傳入 ...countObj
    // setState( countObj => ({count1:countObj.count1+1}) );
    setState( countObj => ( {...countObj, count1:countObj.count1+1} ) );
  };

  const addCountTwo = ( ) => {
    setState( countObj => ( {...countObj, count2:countObj.count2+1} ) );
  };

  return (
    <div>
      <h3>{count1}</h3>
      <button onClick={addCountOne}>Add count1</button>
      <h3>{count2}</h3>
      <button onClick={addCountTwo}>Add count2</button>
    </div>
  );
};
```



主題2：Hooks useEffect (附加效果)

- Hooks useEffect (附加效果)

  import React, { useState, **useEffect** } from 'react‘;

  useEffect 傳入兩個參數

- 第一個參數「函式」相當於 **componentDidMount**

  於組件每次 **render** 更新時執行第一個參數「函式」

  此函式的回傳可以 return 另一個函式, 相當於 **componentWillUnmount**

- 第二個參數「陣列」相當於 **componentDidUpdate**

  用來判斷與前一次呼叫 **useEffect** 所傳入的第二參數值是否相同 (陣列中的每個值都會比較)

  如果不相同則會再次呼叫第一個參數的函式

  若傳入空陣列則 **useEffect** 只會執行一次

  若無傳入第二參數則 useEffect **會無限重覆執行**

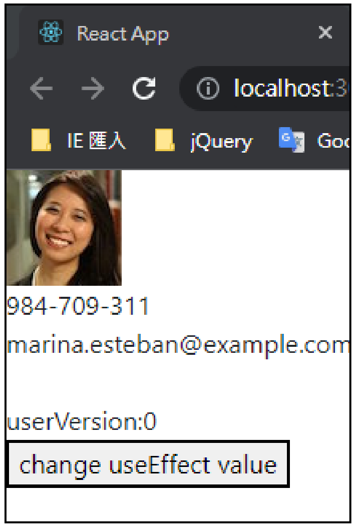
主題2：Hooks useEffect (附加效果)

```
const [ state, setState ] = useState({
      email: '',
      phone: '',
      picture: '',
      userVersion: 0
});

const { email, phone, picture, userVersion } = state;
```

```
useEffect( ( ) => {
      console.log("2.useEffect(附加作用)componentDidMount:", userVersion);
      // fetch API
      fetch('https://randomuser.me/api/')
      .then((rs) => rs.json())
      .then((data) => {
         const [user] = data.results;
         setState( (u) => ({
            ...u,
            email: user.email,
            phone: user.phone,
            picture: user.picture.medium
         }) );
      });

      // 相當於 componentWillUnmount 清理函式
      return ( ) => {
         console.log("3.useEffect(附加作用)componentWillUnmount:", userVersion);
      };
   }, [ userVersion ]);
```

主題2：Hooks useEffect (附加效果)

```
const changeUseEffectValue = () => {
    setState( (u) => ({
        ...u,
        userVersion: userVersion+1
    }) );
 };

 console.log("1.render 渲染函式");
 return (
    <div>
        <img src={picture}/>
        <div>{phone}</div>
        <div>{email}</div>
        <br/>
        <div>userVersion:{userVersion}</div>
        <button onClick={changeUseEffectValue}>
                change useEffect value
         </button>
    </div>
 );
```



React App

localhost:3

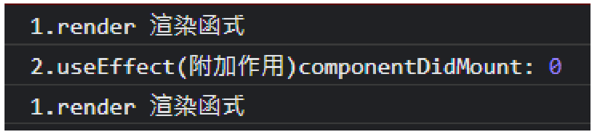
IE 匯入 jQuery Goo

984-709-311

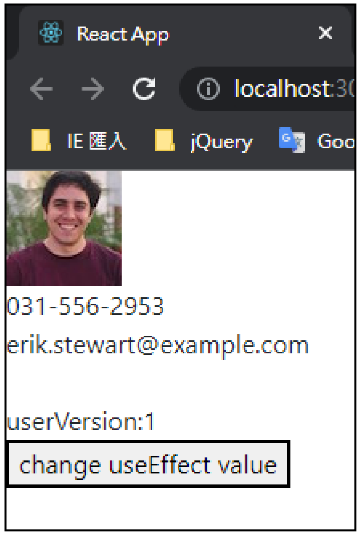
marina.esteban@example.com

userVersion:0

change useEffect value



```
1.render 渲染函式
2.useEffect(附加作用)componentDidMount: 0
1.render 渲染函式
```



React App

localhost:30

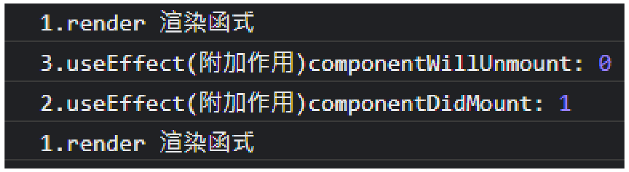
IE 匯入 jQuery Goo

031-556-2953

erik.stewart@example.com

userVersion:1

change useEffect value



```
1.render 渲染函式
3.useEffect(附加作用)componentWillUnmount: 0
2.useEffect(附加作用)componentDidMount: 1
1.render 渲染函式
```

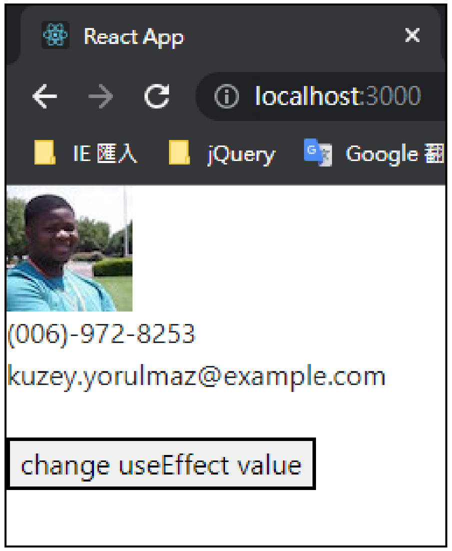
主題2：Hooks useEffect (附加效果)

```
const [ state, setState ] = useState({
   email: '',
   phone: '',
   picture: ''
});

const { email, phone, picture} = state;
```

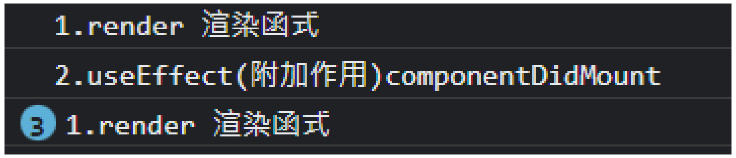
```
useEffect(( ) => {
   console.log("2.useEffect(附加作用)componentDidMount");
   // 新增監聽
   window.addEventListener('submit', fetchRandomuser);

   // 相當於 componentWillUnmount 清理函式
   return ( ) => {
      console.log("3.useEffect(附加作用)componentWillUnmount");
      // 移除監聽
      window.removeEventListener('submit', fetchRandomuser);
   };
}, [ ] );
```

主題2：Hooks useEffect (附加效果)

```
const fetchRandomuser = (e) => {
   e.preventDefault(); // 避免表單送出預設跳頁行為
   fetch('https://randomuser.me/api/')
   .then((rs) => rs.json())
   .then((data) => {
      const [user] = data.results;
      setState( (u) => ({
         ...u,
         email: user.email,
         phone: user.phone,
         picture: user.picture.medium
      }) );
   });
};

console.log("1.render 渲染函式");
return (
   <div>
      <img src={picture}/>
      <div>{phone}</div>
      <div>{email}</div>
      <br/>
      <form>
         <button type='submit'>change useEffect value</button>
      </form>
   </div>
);
```



(006)-972-8253
kuzey.yorulmaz@example.com



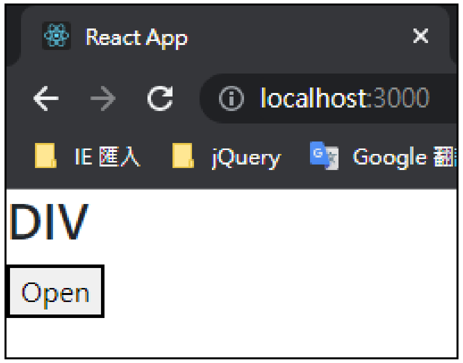
```
1.render 渲染函式
2.useEffect(附加作用)componentDidMount
3 1.render 渲染函式
```

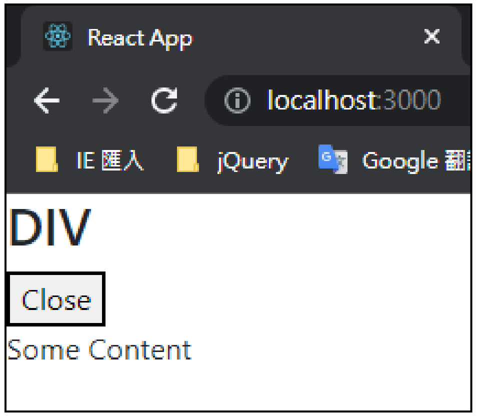
主題3：Hooks useContext (應用Context)

- Function Component 父傳子 (逐層傳遞)

1. 將整個父組件的 state 傳入子組件 props
2. 一般傳入子組件的屬性名稱不會命名為 props

```
const UseFunOpen = () => {

  const toggle = () => {
    setState(
      (s) => ({
        ...s,
        open: !s.open
      })
    );
  };

  const [state, setState] = useState( {open: false, toggle} );

  return (
    <div>
      <UseFunOpenButtonDiv props={state}/>
      { state.open && <div>Some Content</div> }
    </div>
  );

}
```





主題3：Hooks useContext (應用Context)

```
// {props} 接收父組件的物件參數，所以須加上大括弧
const UseFunOpenButtonDiv = ( {props} ) => {
   return (
      <div>
         <h3>DIV</h3>
         <UseFunOpenButton props={props} />
      </div>
   );
}
```

```
const UseFunOpenButton = ( {props} ) => {
   const {open, toggle} = props;
   return (
       <button onClick={toggle}>
            { open ? 'Close' : 'Open' }
        </button>
    );
}
```

主題3：Hooks useContext (應用Context)

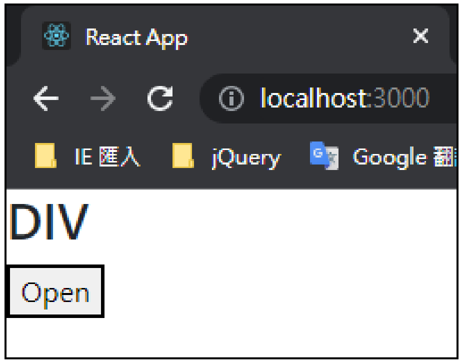
- Function Component 父傳子 (React useContext)

import React, { **useContext** } from 'react';

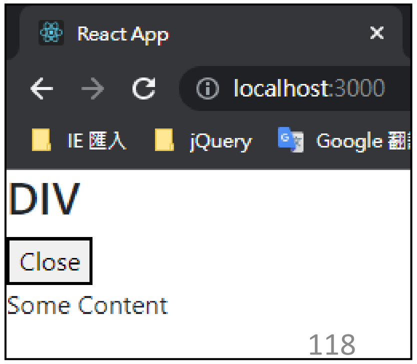
1. 使用 useContext 取代 Context API Consumer
2. 透過 useContext 中間層組件**就不須在協助傳遞** props 給下層組件

```
const UseContextOpen = () => {

  const toggle = () => {
    setState(
      (s) => ({
        ...s,
        open: !s.open
      })
    );
  };

  const [state, setState] = useState( {open: false, toggle} );

  return (
    <div>
      <Context.Provider  value={state}>
        <UseContextOpenButtonDiv/>
        { state.open && <div>Some Content</div> }
      </Context.Provider>
    </div>
  );

}
```

```
import { createContext } from "react";

// createContext並給予初始值
const context = createContext({
  open: false,
  toggle: () => {}
});

export const { Provider, Consumer } = context;
export default context;
```


DIV

Open


DIV

Close

Some Content

主題3：Hooks useContext (應用Context)

```
import UseContextOpenButton from
'./UseContextOpenButton';

const UseContextOpenButtonDiv = () => {
   return (
      <div>
         <h3>DIV</h3>
         <UseContextOpenButton/>
      </div>
   );
}
```

```
import React, { useContext } from 'react‘;
import context from './context';

const UseContextOpenButton = () => {
   // 使用useContext取代<Context.Consumer>
   const {open, toggle} = useContext(context);
   return (<button onClick={toggle}>{ open ? 'Close' : 'Open' }</button>);
}
```

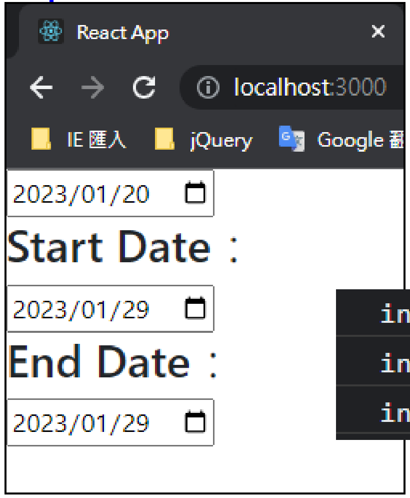
主題4：Hooks useRef (存取參照)

● JSX 元素 ref 儲存 DOM 的參照

1. 使用 **useRef** 取代 **createRef**
2. 與 createRef( ) 差異在不會每次都建立新的ref
3. import React, { **useRef** } from 'react‘
4. 將初始值宣告為陣列[ ]儲存多個 ref

```
const inputDateRef = useRef( );
const inputRef = useRef([ ]);

const onChangeData = ( ) => {
   const inputDate = inputDateRef.current.value;
   const inputDateOne = inputRef.current[0].value;
   const inputDateTwo = inputRef.current[1].value;
   console.log("inputDate:", inputDate);
   console.log("inputDateOne:", inputDateOne);
   console.log("inputDateTwo:", inputDateTwo);
}
```

```
// 取得當天日期並且帶入type="date" defaultValue
const date = new Date();
const month = date.getMonth( ) + 1;
const dateMonth = month < 10 ? `0${month}` : month;
const dateText = `${date.getFullYear()}-${dateMonth}-${date.getDate()}`;

return (
   <div>
      <input
         type="date"
         onChange={onChangeData}
         ref={inputDateRef}
      />
      <h3>Start Date：</h3>
      <input
         type="date"
         onChange={onChangeData}
         ref={el => (inputRef.current[0] = el)}
         defaultValue={dateText}
      />
      <h3>End Date：</h3>
      <input
         type="date"
         onChange={onChangeData}
         ref={el => (inputRef.current[1] = el)}
         defaultValue={dateText}
      />
   </div>
);
```



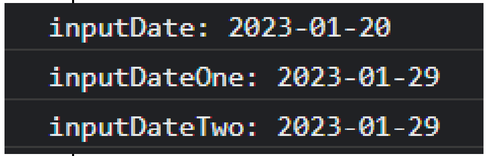
2023/01/20

Start Date：

2023/01/29

End Date：

2023/01/29



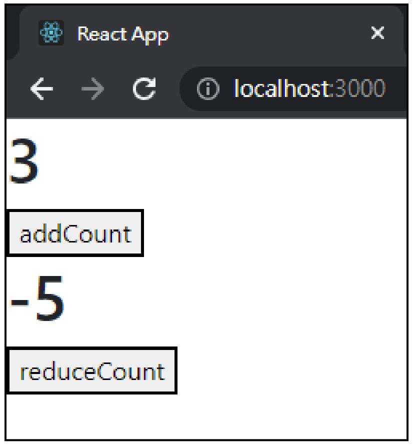
```
inputDate: 2023-01-20
inputDateOne: 2023-01-29
inputDateTwo: 2023-01-29
```

主題4：Hooks useRef (存取參照)

- ref 儲存 instance function 實例函式
  useRef的儲存是跟隨著各自的組件實例,而非所有組件實例共用

```
const [countObj, setState] = useState( {count1:0, count2:0} );

const ref = useRef( );

// ref 儲存instance function實例函數
// useRef的儲存是跟隨著各自的組件實例
// 而非所有組件實例共用
ref.operateCount = {
   addCount: () => {
      setState( countObj => ({...countObj,
                 count1:countObj.count1+1}) );
   },
   reduceCount: () => {
      setState( countObj => ({...countObj,
                 count2:countObj.count2-1}) );
   }
};

const {count1, count2} = countObj;
```

```
return (
   <Fragment>
      <h1>{count1}</h1>
      <button onClick={ref.operateCount.addCount}>
             addCount
       </button>

      <h1>{count2}</h1>
      <button onClick={ref.operateCount.reduceCount}>
             reduceCount
       </button>
   </Fragment>
);
```



主題4：Hooks useRef (存取參照)

- Function Component 子傳父

  透過 useRef( ) 建立與傳入ref 屬性與子組件 <UseRefChild/> 的參照

  就可在父組件 <UseRefParent/> 裡操作與取得子組件中的函式及欄位值

```
const UseRefParent = ( ) => {

  const parentRef = useRef( );

  return (
    <div>
      <button onClick={() => parentRef.current.toggle( )} >
            ChildToggle
      </button>
      <br/><br/>
      <button onClick={() => parentRef.current.addCount( )} >
            ChildAddCount
      </button>
      <UseRefChild ref={parentRef}  parentAttr={ {initCount:5} } />
    </div>
  )
}
```

主題4：Hooks useRef (存取參照)

- Function Component 子傳父

1. 子組件建立 **forwardRef( )** 並且傳入函式
此函式第一個參數為上層組傳所傳來的 **props**
第二參數為 **ref**

2. 最後在此函式裡透過 **useImperativeHandle( )**
傳入
第一個參數為 **ref**
第二個參數傳入提供給父組件使用的**欄位及函式**

3. useImperativeHandle 必須要與 forwardRef
   配合使用

https://reactjs.org/docs/hooks-reference.html#useimperativehandle

```
const UseRefChild = forwardRef( ( {parentAttr}, ref ) => {

  const [state, setState] = useState({
     open: true,
     count: parentAttr.initCount
  });

  const{ open, count } = state;

  const toggle = () => {
     setState(
        (s) => ({ ...s, open: !s.open })
     );
  };

  const addCount = () => {
     setState(
        (s) => ({ ...s, count: s.count+1 })
     );
  };

  // 訂義子組件<UseRefChild>傳給父组件<UseRefParent>的函式
  useImperativeHandle(ref, () => ({
     toggle, addCount
  }));

  return (
     <div>
        { open && <h3>Some Content： {count} </h3> }
     </div>
  )
});
```

主題5：Hooks 綜合應用 TodoList

- 透過 useState( ) 操作狀態儲存 users 資料
- 透過 useEffect( ) 操作 componentDidMount 於組件掛載時執行 ajax api 取得 user 資料

import React, { **useState**, **useEffect** } from 'react‘

```
const [users, setUsers] = useState([ ]);

const apiUrl  = 'https://jsonplaceholder.typicode.com/posts/1/comments';

useEffect(
   ( ) => {
      // fetchCourseData(apiUrl);
      axiosCourseData(apiUrl);
   }, [ ]
);
```

主題5：Hooks 綜合應用 TodoList

- fetch 是原生 API 能直接使用

1. 基於 Promise 實現
2. 對 4xx、5xx 都當做成功的請求

   需要封裝去處理必須拋出(throw Error)錯誤
3. 不支援 timeout
4. 不會自動轉換 JSON 資料格式

```
const fetchCourseData = async (apiUrl) => {
  // fetch所回傳的直接是資料結果
  const userDatas = await fetch(apiUrl)
  .then(rs => {
    console.log("rs.ok:", rs.ok);
    console.log("rs.status:", rs.status);
    // 只要rs.ok不等於true則須另外拋出Error
    // 才能被cache進行後續其它處理
    if (!rs.ok || rs.status !== 200) {
      throw Error("http status:" + rs.status);
    }
    return rs.json();
  })
  // .then((userDatas) => {
  //    setUsers(users => [...users, ...userDatas]);
  // })
  .catch(error => {
    console.log(error);
  });

  setUsers(users => [...users, ...userDatas]);
};
```

主題5：Hooks 綜合應用 TodoList

- axios 非原生 API

不能直接使用須另外 import

import **axios** from "axios“;

1.基於 Promise 實現

2.自動 cache error 不須額外處理

3.支援 timeout

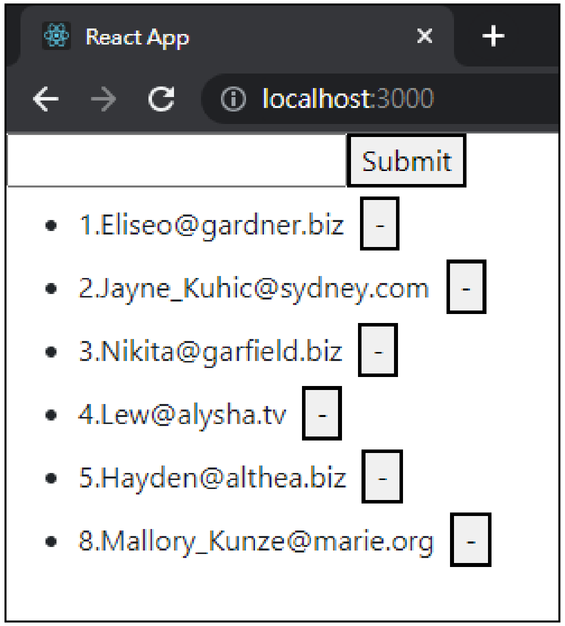
4.自動轉換 JSON 資料格式

```
const axiosCourseData = async (apiUrl) => {
  // 從 rs 裡解構 headers, status, data
  const { headers, status, data } = await axios.get(apiUrl, {timeout: 3000})
  .then(rs => rs)
  .catch(error => {
    console.log(error);
  });

  console.log("rs.headers:", headers);
  console.log("rs.status:", status);
  console.log("rs.data:", data);

  setUsers(users => [...users, ...data]);
};
```

主題5：Hooks 綜合應用 TodoList

- createContext API

```
import { createContext } from "react";
const context = createContext({  });
export const { Provider, Consumer } =
context;
export default context;
```


React App
localhost:3000
Submit

- 1.Eliseo@gardner.biz -
- 2.Jayne_Kuhic@sydney.com -
- 3.Nikita@garfield.biz -
- 4.Lew@alysha.tv -
- 5.Hayden@althea.biz -
- 8.Mallory_Kunze@marie.org -

```
const addItem = newUser => {
  // 不存在於user清單裡才能加入避免key值重覆
  const filterItem = users.filter(user => user.id == newUser.id);
  if(filterItem.length == 0){
    setUsers(users => [...users, newUser]);
  }
};

const removeItem = id => {
  // 過濾掉要刪除的user項目
  setUsers(users.filter(user => user.id !== id));
};

const contextValue = { addItem };  // contextValue必須包成物件

return (
  <div>
    <Provider value={contextValue}>
      <TodoInput/>
    </Provider>
    <ul>
      {users.map(user =>
        <li  style={ {marginTop:'5px'} } key={user.id} >
          {user.id}.{user.email}
          &nbsp;&nbsp;
          <button onClick={() => removeItem(user.id)}>-</button>
        </li>
      )}
    </ul>
  </div>
);
```

主題5：Hooks 綜合應用 TodoList

```
import React, { useState, useRef, useEffect, useContext } from 'react';
import axios from "axios";
import context from './context';
```

```
// addItem來至於父組件(TodoList)
const { addItem } = useContext(context);

const [text, setText] = useState('');

const ref = useRef();

useEffect(
  () => {
    // 組件掛載時就focus輸入框
    ref.current.focus();
  },[ ]
);

const onChangeText = e => {
  setText(e.target.value);
};
```

```
const apiUrl  = 'https://jsonplaceholder.typicode.com/comments/';
  const onAddUserSubmit = async e => {
    e.preventDefault();  // 防止預設submit送出跳頁行為

    const { data } = await axios.get(apiUrl + text).then(rs => rs);
    console.log("user:", data);
    addItem(data);

    // 也可從表單裡取欄位值
    // const form = e.target;
    // console.log("form.userID:", form.userID.value);

    setText(''); // 清空輸入框
    ref.current.focus();
  };

  return (
    <div>
      <form onSubmit={onAddUserSubmit}>
        <input name='userID' type="text" ref={ref} value={text}
                  onChange={onChangeText} />
        <button type="submit">Submit</button>
      </form>
    </div>
  )
```

第十章. React Bootstrap

- 主題1：官方 React Router 路由
- 主題2：React Bootstrap(4.6) 安裝及匯入
- 主題3：React Bootstrap Layout Container、grid system
- 主題4：React Bootstrap Components Buttons
- 主題5：React Bootstrap Components Cards
- 主題6：React Bootstrap Components Forms
- 主題7：React Bootstrap Components Modal
- 主題8：React Bootstrap Components Navbar
- 主題9：React Bootstrap Components Pagination
- 主題10：React Bootstrap Components Tables
- 主題11：React Bootstrap Components Carousel

主題1：官方 React Router 路由

- 主畫面父組件導航欄功能可連結切換畫面子組件

  1. 匯入 import { **BrowserRouter, Link, Routes, Route** } from "**react-router-dom**";

  2. **BrowserRouter** 包覆整個子元件 **Link, Routes, Route**

  3. 使用 **useNavigate** 主動觸發 react-router-dom 頁面跳轉功能

  (並且可傳入轉頁後的state資料)

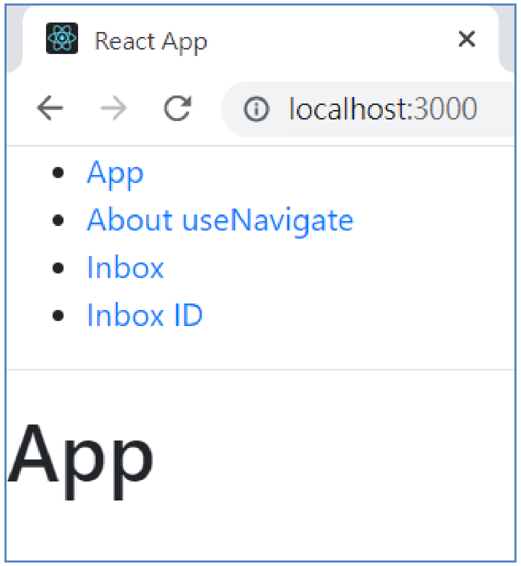
  4. **useParams** 接收網址路徑參數

  5. **useLocation** 接收由navigate導頁傳入而來的資料

主題1：官方 React Router 路由

```
const RouterDom = () => {
  return (
    <BrowserRouter>
      <ul>
        <li><Link to="/">App</Link></li>
        <li><Link to="/about">About useNavigate</Link></li>
        <li><Link to="/inbox">Inbox</Link></li>
        <li><Link to="/inbox/123">Inbox ID</Link></li>
      </ul>
      <hr />
      <Routes>
        <Route path="/" element={<App />} />
        <Route path="about" element={<About />} />
        <Route path="inbox" element={<Inbox />}>
          <Route path=":id" element={<Inbox />} />
        </Route>
        <Route path="*" element={<p>404</p>} />
      </Routes>
    </BrowserRouter>
  );
};
```

```
const App = () => {
 return (
  <div>
   <h1>App</h1>
  </div>
 )
};
```


React App

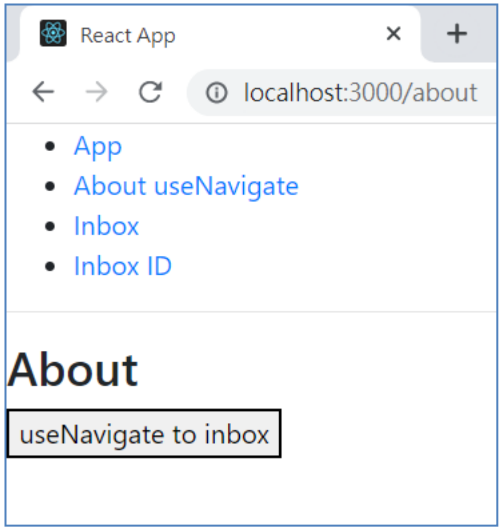
localhost:3000

- App
- About useNavigate
- Inbox
- Inbox ID

App

主題1：官方 React Router 路由

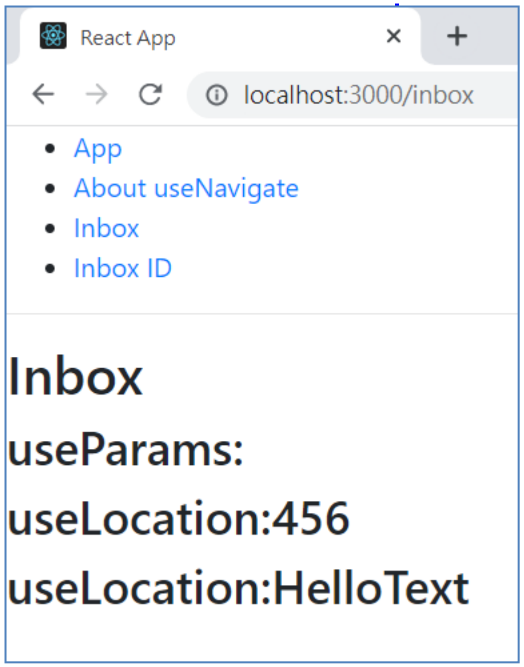
```
const About = () => {
  // 1.使用 useNavigate 主動觸發 react-router-dom 頁面跳轉功能(並且可傳入轉頁後的state資料)
  // 2.useNavigate() may be used only in the context of a <Router> component
  // (操作useNavigate必須在Router的子組件裡使用)
  // 3.傳統window.location的導頁方式會讓整個畫面重新整理,使用者體驗較不好
  let navigate = useNavigate();
  const clickUseNavigate = () => {
    const inboxInfo = {
      id: 456,
      text: 'HelloText'
    };
    navigate("/inbox", {state: inboxInfo} );
    // window.location = "/inbox";
  }

  return (
   <div>
    <h3>About</h3>
    <button onClick={clickUseNavigate}>useNavigate to inbox</button>
   </div>
  )
}
```



```
React App
localhost:3000/about
```

- App
- About useNavigate
- Inbox
- Inbox ID

About



```
React App
localhost:3000/inbox
```

- App
- About useNavigate
- Inbox
- Inbox ID

Inbox

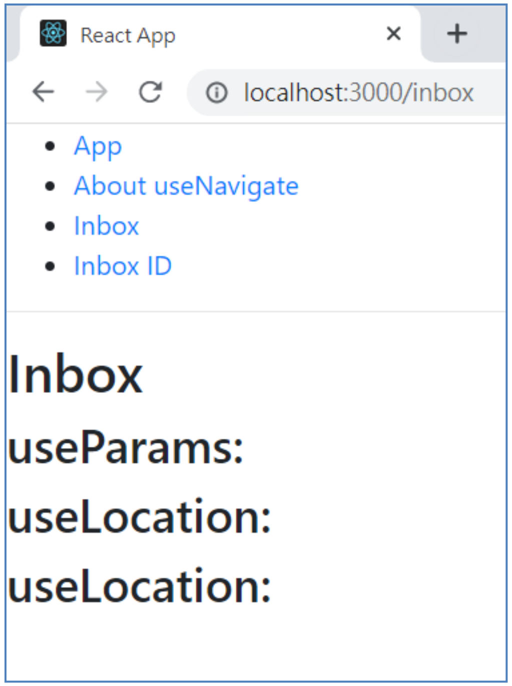
useParams:

useLocation:456

useLocation:HelloText

主題1：官方 React Router 路由

```
import { useParams, useLocation } from "react-router-dom";

const Inbox = () => {
  // useParams接收網址路徑參數
  let params = useParams();
  // useLocation接收由navigate導頁傳入而來的資料
  const location = useLocation();
  const inboxInfo = location.state;

  return (
   <div>
    <h2>Inbox</h2>
    <h3>useParams:{params.id}</h3>
    <h3>useLocation:{inboxInfo !== null && inboxInfo.id}</h3>
    <h3>useLocation:{inboxInfo !== null && inboxInfo.text}</h3>
   </div>
  )
}
```



React App

localhost:3000/inbox

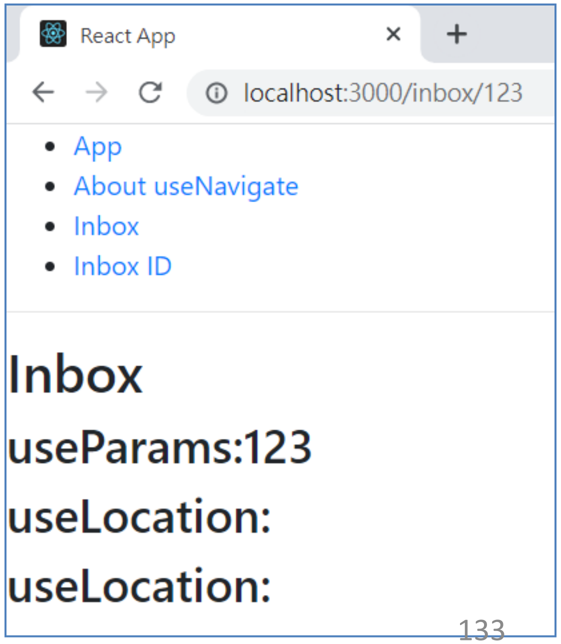
- App
- About useNavigate
- Inbox
- Inbox ID

Inbox

useParams:

useLocation:

useLocation:



React App

localhost:3000/inbox/123

- App
- About useNavigate
- Inbox
- Inbox ID

Inbox

useParams:123

useLocation:

useLocation:

主題2：React Bootstrap(4.6) 安裝及匯入

- npm安裝指令

npm install react-bootstrap@1.6.4 bootstrap@4.6.0

- 英文官方

https://react-bootstrap-v4.netlify.app/getting-started/introduction/

- 中文第三方

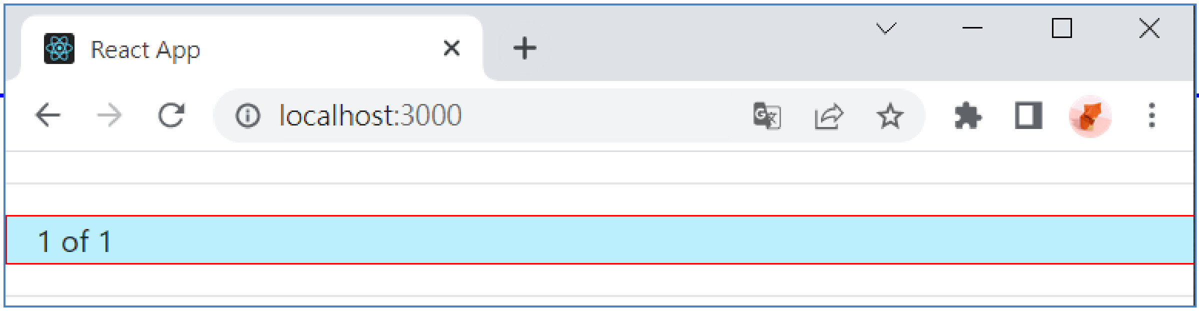
http://react.tgwoo.com/

主題3：React Bootstrap Layout Container、grid system

- Layout Container (佈局容器)、grid system (網格系統)

```
import Container from 'react-bootstrap/Container';
import Row from 'react-bootstrap/Row'
import Col from 'react-bootstrap/Col'
```

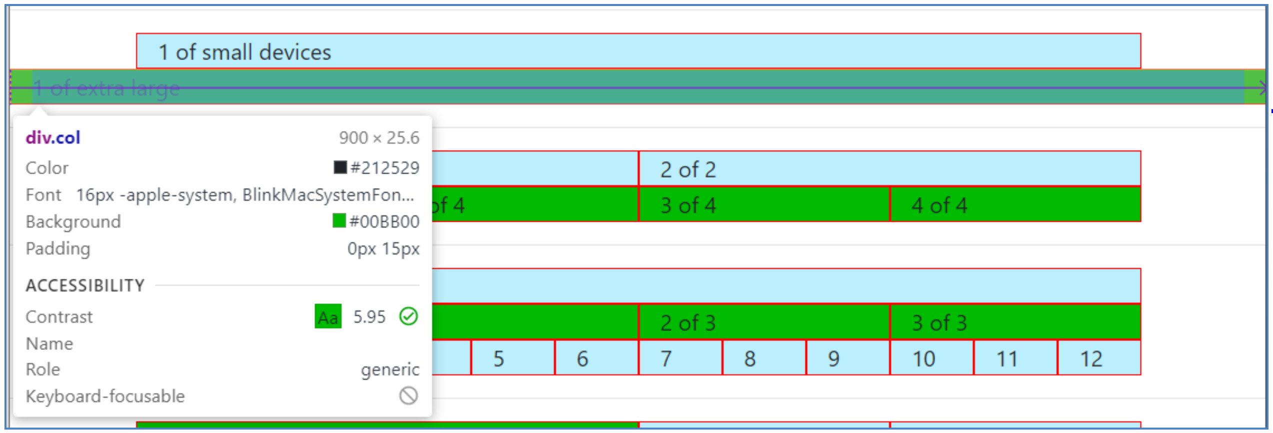
```
{/*
        Fluid Container
        You can use <Container fluid /> for width: 100% across all viewport and device sizes.
*/}

<Container fluid>
    <Row>
        <Col style={colStyle}>1 of 1</Col>
    </Row>
</Container>
```



```
1 of 1
```

主題3：React Bootstrap Layout Container、grid system

```
{/*
   You can set breakpoints for the fluid prop. Setting it to a breakpoint (sm, md, lg, xl)
   於介定的寬度範圍內元素置中,否則就全版顯示
   sm:small (≥576px)
   md:medium (≥768px)
   lg:large (≥992px)
   xl:extra large (≥1200px)
   will set the Container as fluid until the specified breakpoint.
*/}
<Container fluid="sm">
   <Row>
      <Col style={colStyle}>1 of small devices</Col>
   </Row>
</Container>
<Container fluid="xl">
   <Row>
      <Col style={colStyle1}>1 of extra large</Col>
   </Row>
</Container>
```



1 of small devices

1 of extra large

div.col 900 × 25.6

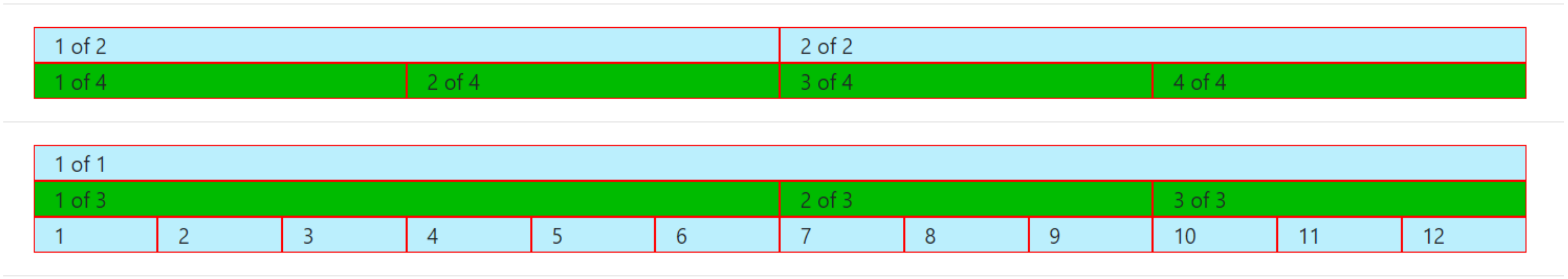
Color	■#212529
Font	16px -apple-system, BlinkMacSystemFon...
Background	🟩#00BB00
Padding	0px 15px

ACCESSIBILITY

Contrast	Aa 5.95 ✓
Name	
Role	generic
Keyboard-focusable	⊘

主題3：React Bootstrap Layout Container、grid system

```
{/* Auto-layout columns 自動依照<Col>的個數平均12個網格 */}
<Container>
  <Row>
    <Col style={colStyle}>1 of 2</Col>
    <Col style={colStyle}>2 of 2</Col>
  </Row>
  <Row>
    <Col style={colStyle1}>1 of 4</Col>
    <Col style={colStyle1}>2 of 4</Col>
    <Col style={colStyle1}>3 of 4</Col>
    <Col style={colStyle1}>4 of 4</Col>
  </Row>
</Container>
```



1 of 2 | 2 of 2
1 of 4 | 2 of 4 | 3 of 4 | 4 of 4

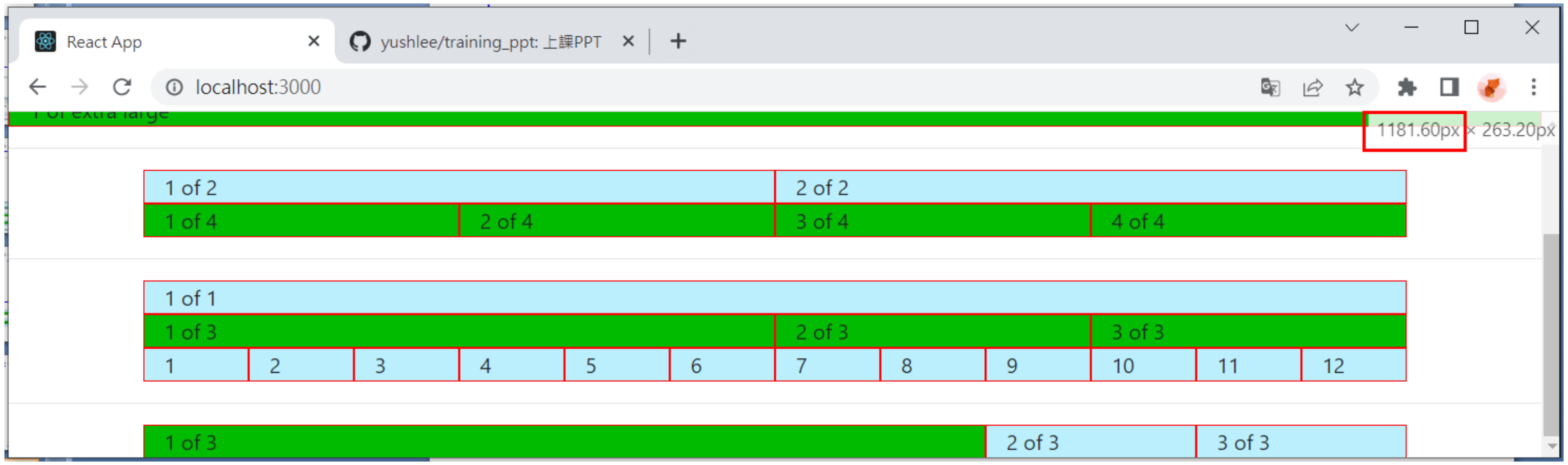
1 of 1
1 of 3 | 2 of 3 | 3 of 3
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12

主題3：React Bootstrap Layout Container、grid system

```
{/*
   Setting one column width(12個網格),xs={6}指定所佔的網格數量
   xs:extra small devices (<576px)
*/}
<Container>
   <Row>
      <Col style={colStyle}>1 of 1</Col>
   </Row>
   <Row>
      <Col style={colStyle1} xs={6}>1 of 3</Col>
      <Col style={colStyle1}>2 of 3</Col>
      <Col style={colStyle1}>3 of 3</Col>
   </Row>
   <Row>
      <Col style={colStyle}>1</Col>
      <Col style={colStyle}>2</Col>
      <Col style={colStyle}>3</Col>
      <Col style={colStyle}>4</Col>
      <Col style={colStyle}>5</Col>
      <Col style={colStyle}>6</Col>
      <Col style={colStyle}>7</Col>
      <Col style={colStyle}>8</Col>
      <Col style={colStyle}>9</Col>
      <Col style={colStyle}>10</Col>
      <Col style={colStyle}>11</Col>
      <Col style={colStyle}>12</Col>
   </Row>
</Container>
```

主題3：React Bootstrap Layout Container、grid system

```
{/*
   Variable width content 依照視窗長度可動態變動寬度內容
   sm:small (≥576px)
   md:medium (≥768px)
   lg:large (≥992px)
   xl:extra large (≥1200px)
*/}
<Container>
   <Row>
      <Col style={colStyle1} sm="4" md="6" lg="8" xl="10">1 of 3</Col>
      <Col style={colStyle}>2 of 3</Col>
      <Col style={colStyle}>3 of 3</Col>
   </Row>
</Container>
```

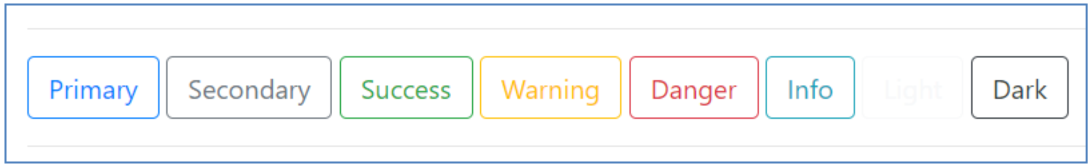


1 of 2 | 2 of 2
1 of 4 | 2 of 4 | 3 of 4 | 4 of 4

1 of 1
1 of 3 | 2 of 3 | 3 of 3
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12

1 of 3 | 2 of 3 | 3 of 3

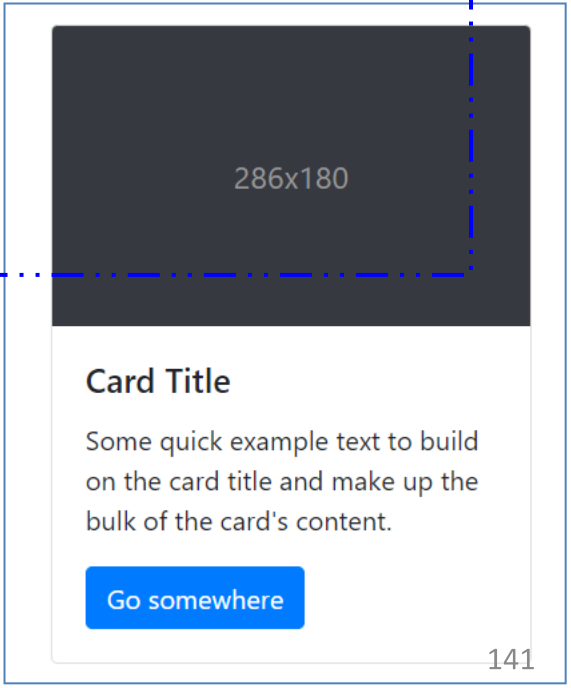
主題4：React Bootstrap Components Buttons



```
import Button from 'react-bootstrap/Button';

<Button variant="outline-primary">Primary</Button>
<Button variant="outline-secondary">Secondary</Button>
<Button variant="outline-success">Success</Button>
<Button variant="outline-warning">Warning</Button>
<Button variant="outline-danger">Danger</Button>
<Button variant="outline-info">Info</Button>
<Button variant="outline-light">Light</Button>
<Button variant="outline-dark">Dark</Button>
```

主題5：React Bootstrap Components Cards

```
import Card from 'react-bootstrap/Card'

<Card style={{ width: '18rem' }}>
   <Card.Img variant="top" src={CardImg} />
   <Card.Body>
      <Card.Title>Card Title</Card.Title>
      <Card.Text>
         Some quick example text to build on the card title and make up the bulk of
         the card's content.
      </Card.Text>
      <Button variant="primary">Go somewhere</Button>
   </Card.Body>
 </Card>
```

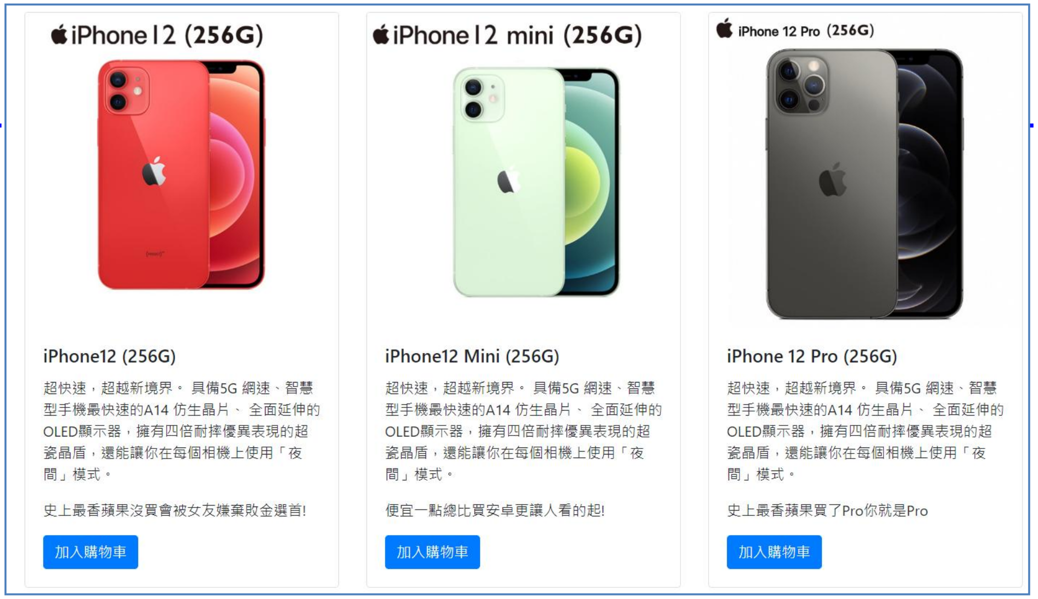


Card Title

Some quick example text to build on the card title and make up the bulk of the card's content.

主題5：React Bootstrap Components Cards

```
import CardDeck from 'react-bootstrap/CardDeck';

<CardDeck>
  <Card>
    <Card.Img variant="top" src={iPhoneImg1} />
    <Card.Body>
      <Card.Title>iPhone12 (256G)</Card.Title>
      <Card.Text>
      超快速，超越新境界。 具備5G 網速、智慧型手機最快速的A14 仿生晶片 、
      全面延伸的OLED顯示器，擁有四倍耐摔優異表現的超瓷晶盾，還能讓你在每個相機上使
      用「夜間」模式。
      </Card.Text>
      <Card.Text>史上最香蘋果沒買會被女友嫌棄敗金選首!</Card.Text>
      <Button variant="primary">加入購物車</Button>
    </Card.Body>
  </Card>
</CardDeck>
```



iPhone12 (256G)

超快速，超越新境界。 具備5G 網速、智慧型手機最快速的A14 仿生晶片、 全面延伸的OLED顯示器，擁有四倍耐摔優異表現的超瓷晶盾，還能讓你在每個相機上使用「夜間」模式。

史上最香蘋果沒買會被女友嫌棄敗金選首!

加入購物車

iPhone12 Mini (256G)

超快速，超越新境界。 具備5G 網速、智慧型手機最快速的A14 仿生晶片、 全面延伸的OLED顯示器，擁有四倍耐摔優異表現的超瓷晶盾，還能讓你在每個相機上使用「夜間」模式。

便宜一點總比買安卓更讓人看的起!

加入購物車

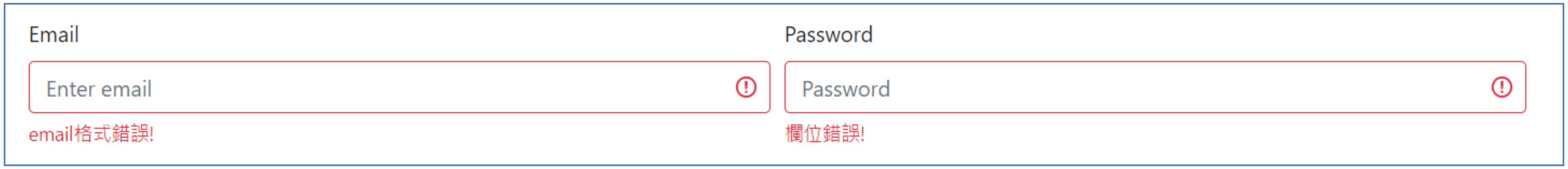
iPhone 12 Pro (256G)

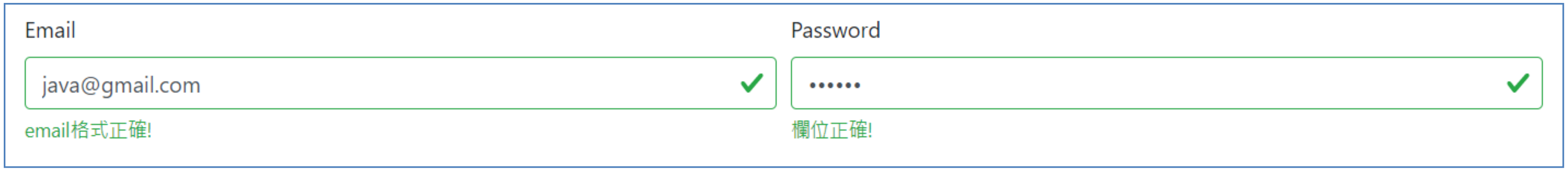
超快速，超越新境界。 具備5G 網速、智慧型手機最快速的A14 仿生晶片、 全面延伸的OLED顯示器，擁有四倍耐摔優異表現的超瓷晶盾，還能讓你在每個相機上使用「夜間」模式。

史上最香蘋果買了Pro你就是Pro

加入購物車

主題6：React Bootstrap Components Forms


Email
Enter email
email格式錯誤!
Password
Password
欄位錯誤!


Email
java@gmail.com
email格式正確!
Password
••••••
欄位正確!

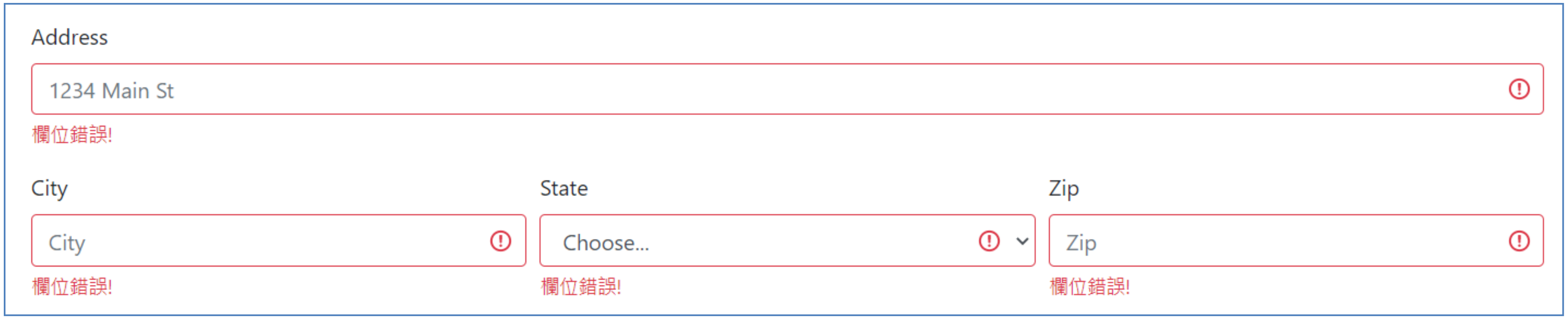
```
import Form from 'react-bootstrap/Form';
import FormControl from 'react-bootstrap/FormControl';
```

```
<Form.Row>
  <Form.Group as={Col} controlId="formGridEmail">
    <Form.Label>Email</Form.Label>
    <Form.Control required type="email" placeholder="Enter email" name='email' onChange={onChangeEmail}/>
    <Form.Control.Feedback>email格式正確!</Form.Control.Feedback>
    <Form.Control.Feedback type="invalid">email格式錯誤!</Form.Control.Feedback>
  </Form.Group>
  <Form.Group as={Col} controlId="formGridPassword">
    <Form.Label>Password</Form.Label>
    <Form.Control required type="password" placeholder="Password" onChange={onChangePassword}/>
    <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
    <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
  </Form.Group>
</Form.Row>
```

主題6：React Bootstrap Components Forms

```
// 建立表單資料參數更新state
// 1.透過更新React state就可以傳遞給其它的組件使用
// 2.透過設置state欄位預設值控制DOM元素預設值呈現

const [ formParam, setFormParam ] = useState({
   email: '', password: '', address: '', city: '', state: '', zip: '', radio: '', checkbox: ['One','Two'],
   date: dateText, time: dateTime, dateTimeLocal: dateTimeLocalText, fileName: ''
});

const onChangeEmail = (e) => { setFormParam(p => ({ ...p, email: e.target.value })) };

const onChangePassword = (e) => { setFormParam(p => ({ ...p, password: e.target.value })) };
```

主題6：React Bootstrap Components Forms

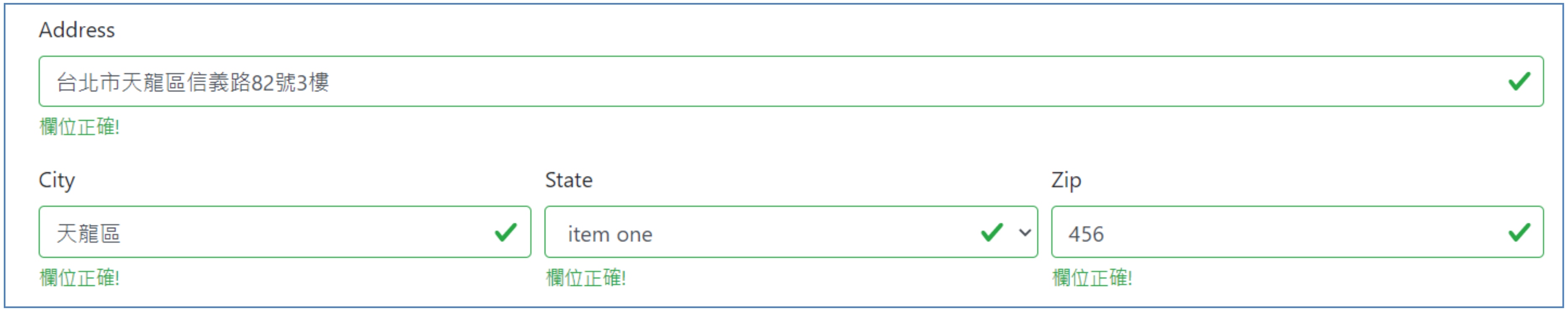


Address

1234 Main St

欄位錯誤!

City	State	Zip
City	Choose...	Zip
欄位錯誤!	欄位錯誤!	欄位錯誤!



Address

台北市天龍區信義路82號3樓

欄位正確!

City	State	Zip
天龍區	item one	456
欄位正確!	欄位正確!	欄位正確!

```
<Form.Row>
    <Form.Group as={Col} controlId="formGridAddress1">
        <Form.Label>Address</Form.Label>
        <Form.Control required type="text" placeholder="1234 Main St" onChange={onChangeAddr}/>
        <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
        <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
    </Form.Group>
</Form.Row>
```

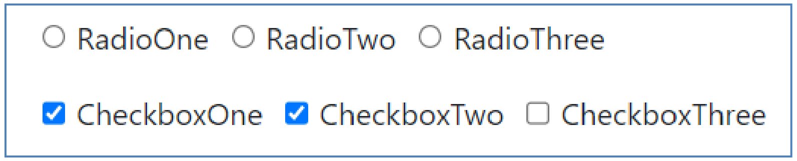
主題6：React Bootstrap Components Forms

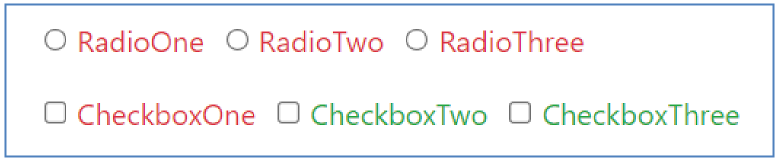
```
<Form.Row>
          <Form.Group as={Col} controlId="formGridCity">
                    <Form.Label>City</Form.Label>
                    <Form.Control required type="text" placeholder="City" onChange={onChangeCity}/>
                    <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
                    <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
          </Form.Group>
          <Form.Group as={Col} controlId="formGridState">
                    <Form.Label>State</Form.Label>
{/* React Bootstrap下拉選單透過 defaultValue 屬性決定預設值選項,且不行透過傳統 selected 屬姓設置 */}
                    <Form.Control required as="select" defaultValue={formParam.state} onChange={onChangeState}>
                              <option value={''}>Choose...</option>
                              <option value={1}>item one</option>
                              <option value={2}>item two</option>
                              <option value={3}>item three</option>
                    </Form.Control>
                    <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
                    <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
          </Form.Group>
          <Form.Group as={Col} controlId="formGridZip">
                    <Form.Label>Zip</Form.Label>
                    <Form.Control required type="number" placeholder="Zip" min={100} onChange={onChangeZip}/>
                    <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
                    <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
          </Form.Group>
</Form.Row>
```

主題6：React Bootstrap Components Forms

```
const [ formParam, setFormParam ] = useState({
    email: '', password: '', address: '', city: '', state: 1, zip: '', radio: '', checkbox: ['One','Two'],
    date: dateText, time: dateTime, dateTimeLocal: dateTimeLocalText, fileName: ''
});

const onChangeAddr = (e) => { setFormParam(p => ({ ...p, address: e.target.value })) };

const onChangeCity = (e) => { setFormParam(p => ({ ...p, city: e.target.value })) };

// 取到的值預設字串型別可以自行轉數字型別
const onChangeState = (e) => { setFormParam(p => ({ ...p, state: parseInt(e.target.value) })) };

const onChangeZip = (e) => { setFormParam(p => ({ ...p, zip: e.target.value })) };
```

主題6：React Bootstrap Components Forms


○ RadioOne ○ RadioTwo ○ RadioThree
☑ CheckboxOne ☑ CheckboxTwo ☐ CheckboxThree

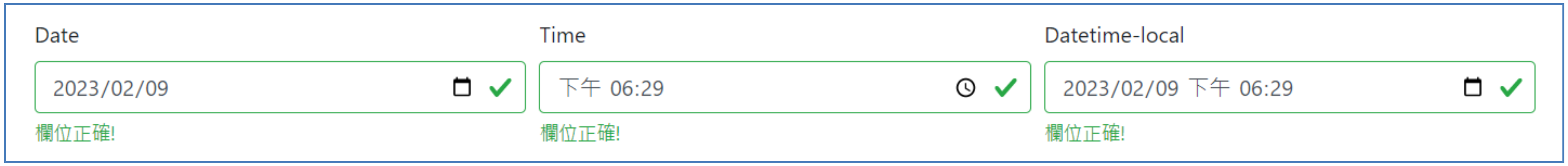

○ RadioOne ○ RadioTwo ○ RadioThree
☐ CheckboxOne ☐ CheckboxTwo ☐ CheckboxThree

```
<Form.Group id="formGridRadio">
        <Form.Check inline required name='radioName' type="radio" label="RadioOne"
                value={'One'} onChange={onChangeRadio} checked= {formParam.radio === 'One' }/>
        <Form.Check inline required name='radioName' type="radio" label="RadioTwo"
                value={'Two'} onChange={onChangeRadio} checked= {formParam.radio === 'Two' }/>
        <Form.Check inline required name='radioName' type="radio" label="RadioThree"
                value={'Three'} onChange={onChangeRadio} checked= {formParam.radio === 'Three' }/>
</Form.Group>

<Form.Group id="formGridCheckbox">
        {/* checkbox不會指定全部選項必選 */}
        <Form.Check inline required type="checkbox" label="CheckboxOne"
                value={'One'} onChange={onChangeCheckbox} checked={formParam.checkbox.includes('One')}/>
        <Form.Check inline type="checkbox" label="CheckboxTwo"
                value={'Two'} onChange={onChangeCheckbox} checked={formParam.checkbox.includes('Two')}/>
        <Form.Check inline type="checkbox" label="CheckboxThree"
                value={'Three'} onChange={onChangeCheckbox} checked={formParam.checkbox.includes('Three')}/>
</Form.Group>
```

主題6：React Bootstrap Components Forms

```
const [formParam, setFormParam] = useState({
   email: '', password: '', address: '', city: '', state: 1, zip: '', radio: '', checkbox: ['One','Two'],
   date: dateText, time: dateTime, dateTimeLocal: dateTimeLocalText, fileName: ''
});

const onChangeRadio = (e) => { setFormParam(p => ({ ...p, radio: e.target.value })) };

const onChangeCheckbox = (e) => { setFormParam(p => {
   let newCheckboxs = [ ];
   if(e.target.checked === true){
      // 有勾選(加入清單)
      newCheckboxs = [...p.checkbox, e.target.value];
   } else {
      // 沒有勾選(從清單中移除)
      newCheckboxs = p.checkbox.filter(elem => elem !== e.target.value);
   }
   return { ...p, checkbox: newCheckboxs }
})};
```

主題6：React Bootstrap Components Forms



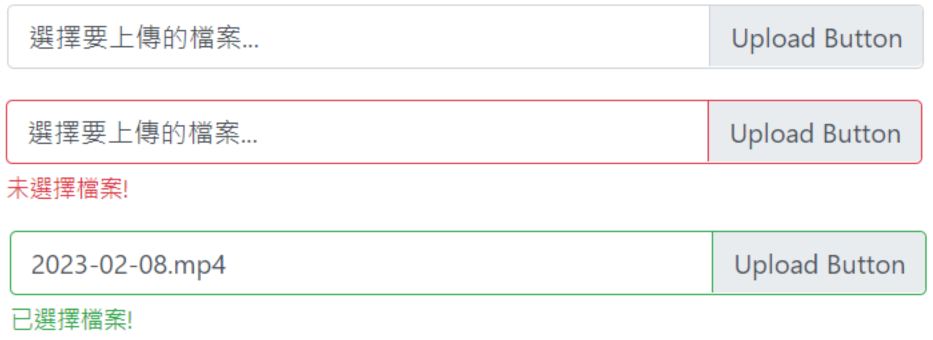
Date | Time | Datetime-local
2023/02/09 | 下午 06:29 | 2023/02/09 下午 06:29
欄位正確! | 欄位正確! | 欄位正確!

```
<Form.Row>
        <Form.Group as={Col} controlId="formDate">
                <Form.Label>Date</Form.Label>
                <Form.Control required type="date" name='date_of_birth' value={formParam.date}
                        onChange={(e) => onChangeDateField(e,'date')}/>
                <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
                <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
        </Form.Group>
        <Form.Group as={Col} controlId="formTime">
                <Form.Label>Time</Form.Label>
                <Form.Control required type="time" name='date_of_time' value={formParam.time}
                        onChange={(e) => onChangeDateField(e,'time')}/>
                <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
                <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
        </Form.Group>
                <Form.Group as={Col} controlId="formDatetimeLocal">
                <Form.Label>Datetime-local</Form.Label>
                <Form.Control required type="Datetime-local" name='date_of_dateTime'
                        value={formParam.dateTimeLocal} onChange={(e) => onChangeDateField(e,'dateTimeLocal')}/>
                <Form.Control.Feedback>欄位正確!</Form.Control.Feedback>
                <Form.Control.Feedback type="invalid">欄位錯誤!</Form.Control.Feedback>
        </Form.Group>
</Form.Row>
```

主題6：React Bootstrap Components Forms

```
// 取得當天日期並且帶入type="date" defaultValue
const date = new Date();
const month = date.getMonth() + 1;
const day = date.getDate();
const dateMonth = month < 10 ? `0${month}` : month;
const dateDay = day < 10 ? `0${day}` : day;
const dateText = `${date.getFullYear()}-${dateMonth}-${dateDay}`;
const dateTime = `${date.getHours()}:${date.getMinutes()}`;

// 2023-02-09 19:50
const dateTimeLocalText = `${dateText} ${dateTime}`;
console.log("dateTimeLocalText:", dateTimeLocalText);

const [formParam, setFormParam] = useState({
   email: '', password: '', address: '', city: '', state: 1, zip: '', radio: '', checkbox: ['One','Two'],
   date: dateText, time: dateTime, dateTimeLocal: dateTimeLocalText, fileName: ''
});

// 透過指定欄位名的方式,就可以不必為每個欄位都撰寫onChange函式
const onChangeDateField = (e, fieldName) => { setFormParam(p => ({ ...p, [fieldName]: e.target.value })) };
```

主題6：React Bootstrap Components Forms


選擇要上傳的檔案... Upload Button

選擇要上傳的檔案... Upload Button

未選擇檔案!

2023-02-08.mp4 Upload Button

已選擇檔案!

```
<Form.Row>
        <Form.Group as={Col} xs={6}>
                <Form.File id="formcheck-api-custom" custom>
                        <Form.File.Input required name="uploadFile" onChange={onChangeFile}/>
                        <Form.File.Label data-browse="Upload Button">
                                {formParam.fileName ? formParam.fileName : '選擇要上傳的檔案...'}
                        </Form.File.Label>
                        <Form.Control.Feedback type="valid">已選擇檔案!</Form.Control.Feedback>
                        <Form.Control.Feedback type="invalid">未選擇檔案!</Form.Control.Feedback>
                </Form.File>
        </Form.Group>
</Form.Row>
```

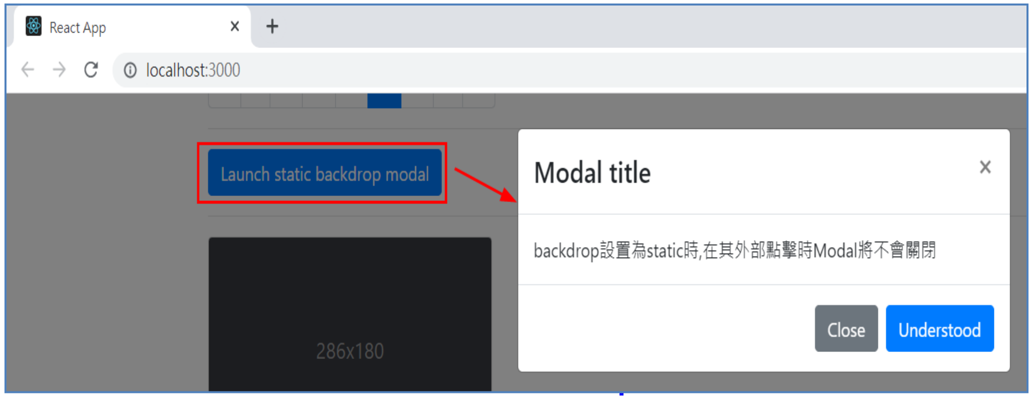
主題6：React Bootstrap Components Forms

```
const [formParam, setFormParam] = useState({
   email: '', password: '', address: '', city: '', state: 1, zip: '', radio: '', checkbox: ['One','Two'],
   date: dateText, time: dateTime, dateTimeLocal: dateTimeLocalText, fileName: ''
});

// 瀏覽檔案上傳欄位
const onChangeFile = (e) => {
   const changFile = e.target.files;
   const changFileName = changFile.length === 0 ? '' : changFile[0].name;
   setFormParam(p => ({ ...p, fileName: changFileName }))
};
```

主題6：React Bootstrap Components Forms

```
 {/* noValidate 關閉瀏覽器預設驗証 */}
 <Form noValidate validated={validated} onSubmit={handleSubmit}>
          <Button variant="primary" type="submit">Submit</Button>
  </Form>

const [validated, setValidated] = useState(false);

const handleSubmit = async (event) => {
   event.preventDefault(); // 防止瀏覽器預設submit跳頁
   // 使用者送出開啟表單欄位驗証功能
   setValidated(true);
   const form = event.currentTarget;
   console.log("form.checkValidity():", form.checkValidity());
   if (form.checkValidity( ) === true) {
      console.log("表單驗証送出成功!");
      // 也可透過表單form欄位name取value
      console.log("form.email.value:", form.email.value);
      const formData = new FormData( );
      // 運用Object.keys走訪State物件中所有的欄位名、欄位值,並且append至FormData裡面
      Object.keys(formParam).map( (formParamKey) => {
            const formParamValue = formParam[formParamKey];
            formData.append(formParamKey, formParamValue);
         }
      )
      formData.append('uploadFile', form.uploadFile.files[0]);

      const formResponse = await axios.post(apiUrl, formData, { timeout: 3000 }).then(rs => rs.data)
      console.log(formResponse);
   }
};
```

主題7：React Bootstrap Components Modal

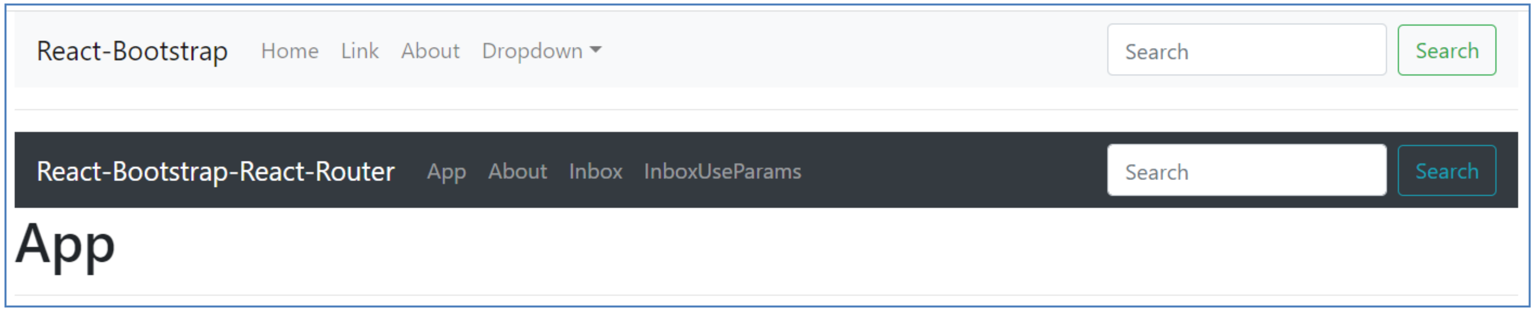


Modal title

backdrop設置為static時,在其外部點擊時Modal將不會關閉

```
 import Modal from 'react-bootstrap/Modal';

<Button variant="primary" onClick={handleShow}>
   Launch static backdrop modal
</Button>

{/*
   keyboard={false} 按下esc鍵關閉Modal
   backdrop設置為static時,在其外部點擊時Modal將不會關閉
*/}
<Modal show={show} onHide={handleClose} backdrop="static" keyboard={false}>
   <Modal.Header closeButton>
      <Modal.Title>Modal title</Modal.Title>
   </Modal.Header>
   <Modal.Body>
      backdrop設置為static時,在其外部點擊時Modal將不會關閉
   </Modal.Body>
   <Modal.Footer>
      <Button variant="secondary" onClick={handleClose}>
         Close
      </Button>
      <Button variant="primary">Understood</Button>
   </Modal.Footer>
</Modal>

   const [show, setShow] = useState(false);
   const handleClose = () => setShow(false);
   const handleShow = () => setShow(true);
```

主題8：React Bootstrap Components Navbar



```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Navbar from 'react-bootstrap/Navbar';
import Nav from 'react-bootstrap/Nav';
import NavDropdown from 'react-bootstrap/NavDropdown';
import App from './RouterApp/App';
import About from './RouterApp/About';
import Inbox from './RouterApp/Inbox';
```

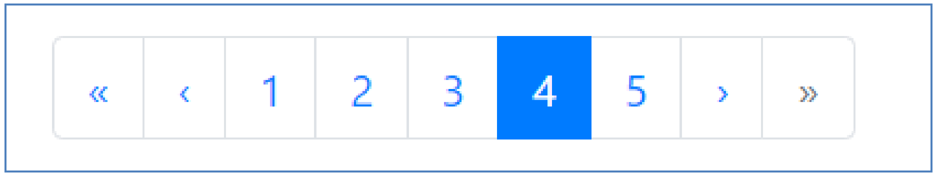
主題8：React Bootstrap Components Navbar

```
{/* React Bootstrap Navbar 導航欄 */}
<Navbar bg="light" expand="lg">
  <Navbar.Brand href="home">React-Bootstrap</Navbar.Brand>
  <Navbar.Toggle aria-controls="basic-navbar-nav" />
  <Navbar.Collapse id="basic-navbar-nav">
    <Nav className="mr-auto">
      <Nav.Link href="home">Home</Nav.Link>
      <Nav.Link href="link">Link</Nav.Link>
      <Nav.Link href="/about">About</Nav.Link>
      <NavDropdown title="Dropdown" id="basic-nav-dropdown">
        <NavDropdown.Item href="action/3.1">Action</NavDropdown.Item>
        <NavDropdown.Item href="action/3.2">Another action</NavDropdown.Item>
        <NavDropdown.Item href="action/3.3">Something</NavDropdown.Item>
        <NavDropdown.Divider />
        <NavDropdown.Item href="action/3.4">Separated link</NavDropdown.Item>
      </NavDropdown>
    </Nav>
    <Form inline>
      <FormControl type="text" placeholder="Search" className="mr-sm-2" />
      <Button variant="outline-success">Search</Button>
    </Form>
  </Navbar.Collapse>
</Navbar>
```

主題8：React Bootstrap Components Navbar

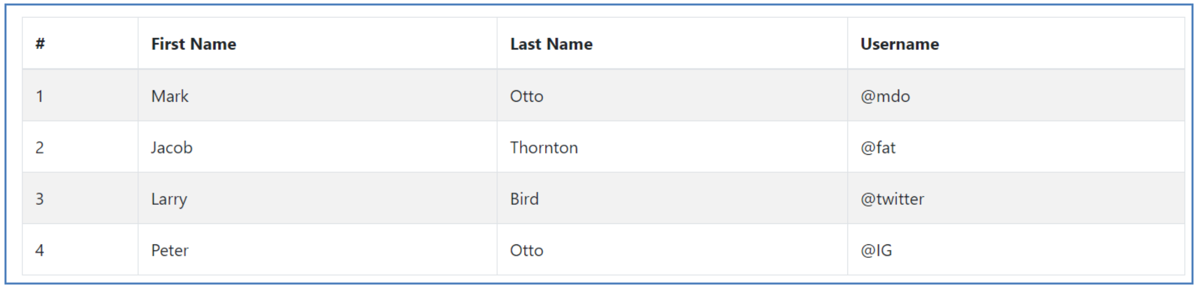
```
{/* React Bootstrap Navbar 結合 react-router-dom */}
<BrowserRouter>
  <Navbar bg="dark" variant={"dark"} expand="lg">
    <Navbar.Brand href="#home">React-Bootstrap-React-Router</Navbar.Brand>
    <Navbar.Toggle aria-controls="navbarScroll" />
    <Navbar.Collapse id="navbarScroll">
      <Nav className="mr-auto">
        {/* to 路徑對應 Route path */}
        <Nav.Link as={Link} to="/">App</Nav.Link>
        <Nav.Link as={Link} to="/about">About</Nav.Link>
        <Nav.Link as={Link} to="/inbox">Inbox</Nav.Link>
        <Nav.Link as={Link} to="/inbox/123">InboxUseParams</Nav.Link>
      </Nav>
      <Form inline>
        <FormControl type="text" placeholder="Search" className="mr-sm-2" />
        <Button variant="outline-info">Search</Button>
      </Form>
    </Navbar.Collapse>
  </Navbar>
  <Routes>
    <Route path="/" element={<App/>} />
    <Route path="about" element={<About/>} />
    <Route path="inbox" element={<Inbox/>}>
      <Route path=":id" element={<Inbox/>} />
    </Route>
  </Routes>
</BrowserRouter>
```

主題9：React Bootstrap Components Pagination



```
import Pagination from 'react-bootstrap/Pagination';

 <Pagination>
   <Pagination.First />
   <Pagination.Prev />
   <Pagination.Item>{1}</Pagination.Item>
   <Pagination.Item>{2}</Pagination.Item>
   <Pagination.Item>{3}</Pagination.Item>
   <Pagination.Item active>{4}</Pagination.Item>
   <Pagination.Item>{5}</Pagination.Item>
   <Pagination.Next />
   <Pagination.Last disabled />
 </Pagination>
```

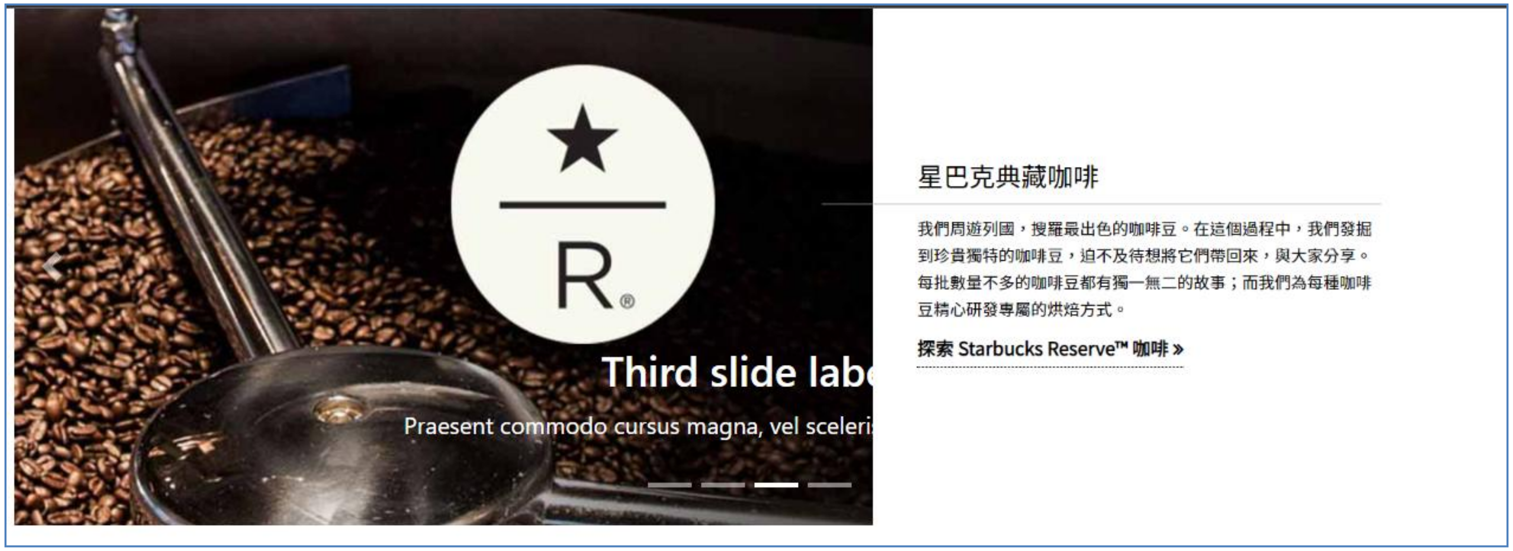
主題10：React Bootstrap Components Tables



#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	Bird	@twitter
4	Peter	Otto	@IG

```
 import Table from 'react-bootstrap/Table';

<Table striped bordered hover>
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Username</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>3</td>
      {/* <td colSpan="2">Larry the Bird</td> */}
      <td>Larry</td>
      <td>Bird</td>
      <td>@twitter</td>
    </tr>
  </tbody>
</Table>
```

主題11：React Bootstrap Components Carousel



星巴克典藏咖啡

我們周遊列國，搜羅最出色的咖啡豆。在這個過程中，我們發掘到珍貴獨特的咖啡豆，迫不及待想將它們帶回來，與大家分享。每批數量不多的咖啡豆都有獨一無二的故事；而我們為每種咖啡豆精心研發專屬的烘焙方式。

探索 Starbucks Reserve™ 咖啡 »

```
import Carousel from 'react-bootstrap/Carousel';
import Snap0 from './CarouselImg/Snap0.png';
import Snap1 from './CarouselImg/Snap1.png';
import Snap2 from './CarouselImg/Snap2.png';
import Snap3 from './CarouselImg/Snap3.png';
```

```
{/* <Carousel fade> 淡入淡出效果 */}
<Carousel fade>
  <Carousel.Item>
    <img className="d-block w-100" src={Snap0} alt="First slide"/>
    <Carousel.Caption>
      <h3>First slide label</h3>
      <p>Nulla vitae elit libero, a pharetra augue mollis interdum.</p>
    </Carousel.Caption>
  </Carousel.Item>
  <Carousel.Item>
    <img className="d-block w-100" src={Snap1} alt="Second slide"/>
    <Carousel.Caption>
      <h3>Second slide label</h3>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
    </Carousel.Caption>
  </Carousel.Item>
  <Carousel.Item>
    <img className="d-block w-100" src={Snap2} alt="Third slide"/>
    <Carousel.Caption>
      <h3>Third slide label</h3>
      <p>Praesent commodo cursus magna, vel scelerisque nisl consectetur.</p>
    </Carousel.Caption>
  </Carousel.Item>
  <Carousel.Item>
    <img className="d-block w-100" src={Snap3} alt="Four slide"/>
    <Carousel.Caption>
      <h3>Four slide label</h3>
    </Carousel.Caption>
  </Carousel.Item>
</Carousel>
```

第十一章. TypeScript

TypeScript(強型別)

嚴謹的語法確保資料形別上的安全

TypeScript相關指令

- npm 安裝 typescript

  npm install typescript –g

- 查詢 typescript 版本

  tsc –v

- 編譯 typescript (*.ts → *.js)

  tsc xxx.ts

- 建立 tsconfig.json

  tsc --init

  並修改(**tsconfig.json**)以下設置

  "rootDir": "./ts" (TypeScript檔案來源ts資料夾)

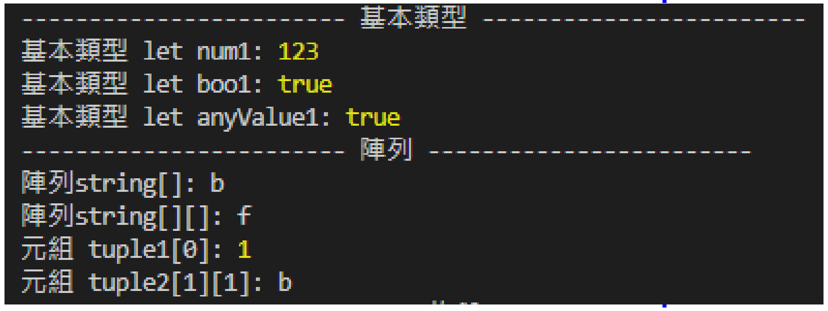
  "outDir": "./js" (編譯目的js資料夾)

- 常駐自動編譯

  tsc –watch

第十一章. TypeScript

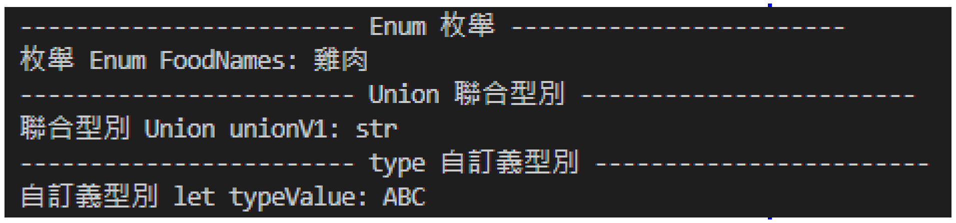
```
console.log('------------------------ 基本類型 ------------------------');
// 能檢查值型別是否符合
let cusName: string = "Hello Two";
let num1: number = 123;
let boo1: boolean = true;
let anyValue1: any = true;

console.log('基本類型 let num1:', num1);
console.log('基本類型 let boo1:', boo1);
console.log('基本類型 let anyValue1:', anyValue1);

console.log('------------------------ 陣列 ------------------------');
let strArr1: string[ ] = ['a', 'b', 'c'];
let strArr2: string[ ][ ] = [['a', 'b', 'c'], ['d', 'e', 'f']];

console.log('陣列string[]:', strArr1[1]);
console.log('陣列string[][]:', strArr2[1][2]);

// 元組
let tuple1: [number, string, boolean] = [1, 'a', true];
let tuple2: [number, string] [ ] = [[1, 'a'], [2, 'b']];

console.log('元組 tuple1[0]:', tuple1[0]);
console.log('元組 tuple2[1][1]:', tuple2[1][1]);
```



```
------------------------ 基本類型 ------------------------
基本類型 let num1: 123
基本類型 let boo1: true
基本類型 let anyValue1: true
------------------------ 陣列 ------------------------
陣列string[]: b
陣列string[][]: f
元組 tuple1[0]: 1
元組 tuple2[1][1]: b
```

第十一章. TypeScript

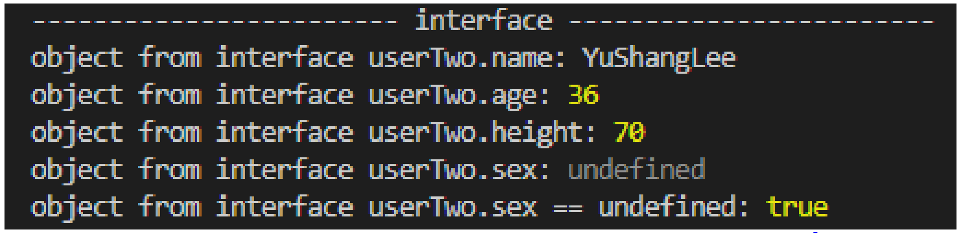
```
console.log('------------------------ Enum 枚舉 ------------------------');
enum FoodNames {
   PORK = '豬肉',
   BEEF = '牛肉',
   FISH = '魚肉',
   CHICKEN = '雞肉'
};
const chicken = FoodNames.CHICKEN;
console.log('枚舉 Enum FoodNames:', chicken);

console.log('------------------------ Union 聯合型別 ------------------------');
let unionV1: number | string;
unionV1 = 1000;
unionV1 = 'str';

console.log('聯合型別 Union unionV1:', unionV1);

console.log('------------------------ type 自訂義型別 ------------------------');
type cusType = number | string;
let typeValue: cusType;

typeValue = 123;
typeValue = 'ABC';
console.log('自訂義型別 let typeValue:', typeValue);
```



```
------------------------ Enum 枚舉 ------------------------
枚舉 Enum FoodNames: 雞肉
------------------------ Union 聯合型別 ------------------------
聯合型別 Union unionV1: str
------------------------ type 自訂義型別 ------------------------
自訂義型別 let typeValue: ABC
```

第十一章. TypeScript

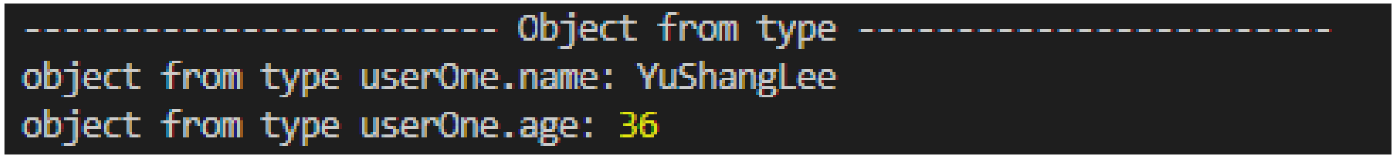
```
console.log('------------------------ interface ------------------------');
interface UserInterface {
   name: string;
   age: number;
};

// interface可在原有的欄位基礎下新增欄位
// 擴展UserInterface欄位
interface UserInterface {
   height: number
   sex?: string // 問號代表可選擇欄位
};


const userTwo: UserInterface = {
   name: 'YuShangLee',
   age: 36,
   height: 70,
   // sex: 'Man'
};

console.log('object from interface userTwo.name:', userTwo.name);
console.log('object from interface userTwo.age:', userTwo.age);
console.log('object from interface userTwo.height:', userTwo.height);
console.log('object from interface userTwo.sex:', userTwo.sex); // 可選擇欄位
console.log('object from interface userTwo.sex == undefined:', userTwo.sex == undefined);
```



```
------------------------- interface -------------------------
object from interface userTwo.name: YuShangLee
object from interface userTwo.age: 36
object from interface userTwo.height: 70
object from interface userTwo.sex: undefined
object from interface userTwo.sex == undefined: true
```

第十一章. TypeScript

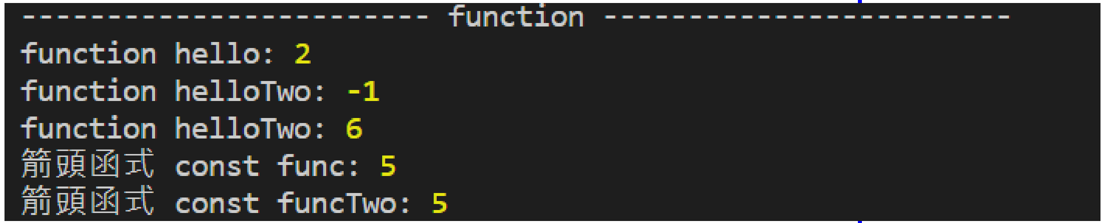
```
console.log('------------------------ Object from type ------------------------');
type userType = {
   name: string
   age?: number
};

// typ、interface差別：typ欄位不能擴充、interface欄位可以擴充
// 透過typ來規範物件應該要有哪些欄位
const userOne: userType = {
   name: 'YuShangLee',
   age: 36
};

console.log('object from type userOne.name:', userOne.name);
console.log('object from type userOne.age:', userOne.age);
```



```
------------------------ Object from type ------------------------
object from type userOne.name: YuShangLee
object from type userOne.age: 36
```

第十一章. TypeScript

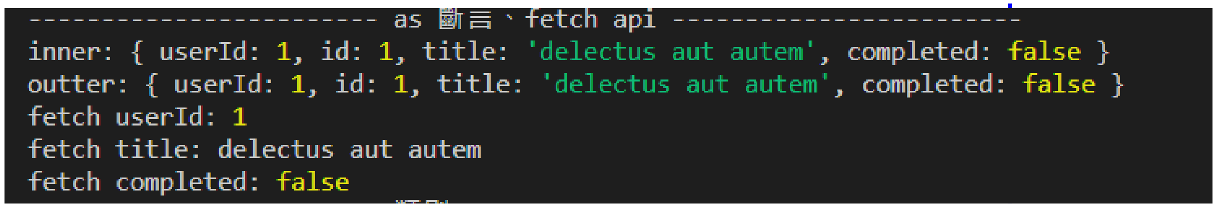
```
console.log('------------------------ function ------------------------');
// function 可透過宣告指定參數型態來規範傳入的參數值,以及規範function所回傳值的型態
function hello(a: number, b: number): number {
   return a + b;
}
console.log('function hello:', hello(1, 1));

// undefined 問號參數類型
// 可選擇性傳入的參數類型
// 必須放在參數列上的最後一個
function helloTwo(a: number, b: number, c?: number): number {
   // 透過判斷undefined就可讓 return a + b + c; 通過檢查編譯
   if (c === undefined) return -1;
   return a + b + c;
}

console.log('function helloTwo:', helloTwo(1, 2));
console.log('function helloTwo:', helloTwo(1, 2, 3));

// 箭頭函式
const func = (a: string): number => {
   let b = parseInt(a);
   b += 2;
   return b;
}

console.log('箭頭函式 const func:', func('3'));

// 箭頭函式省略{}接續直接代表回傳
const funcTwo = (a: string): number => parseInt(a) + 2;
console.log('箭頭函式 const funcTwo:', funcTwo('3'));
```



```
------------------------ function ------------------------
function hello: 2
function helloTwo: -1
function helloTwo: 6
箭頭函式 const func: 5
箭頭函式 const funcTwo: 5
```

第十一章. TypeScript

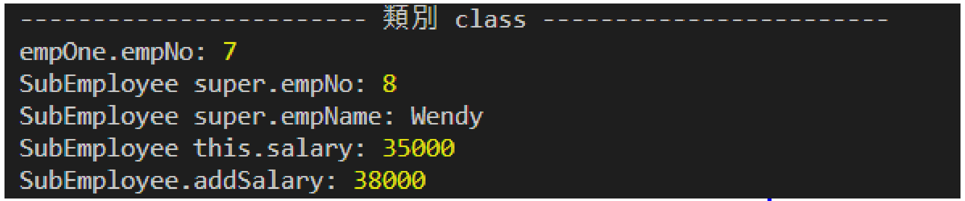
```
console.log('------------------------ as 斷言 、fetch api ------------------------');

import fetch from 'cross-fetch';

type userData = {
   userId: number,
   id: number,
   title: string,
   completed: boolean
};

// Typescript使用fetch須以下設置
// 1.tsconfig.json ("module": "es2022" 、"target": "es2022" 、"moduleResolution": "node")
// 2.package.json (加入"type": "module"設置)
const fetchUserData = async (): Promise<userData>  => {
   // 透過as訂義fetch api所回傳的資料結構type
   const userData = await fetch('https://jsonplaceholder.typicode.com/todos/1')
   .then( rs => rs.json())
   .catch(error => {
      console.log("FetchError:",error);
   });

   console.log("inner:", userData);

   return userData;
}

const user: userData = await fetchUserData( );
console.log("outter:", user);
console.log("fetch userId:", user.userId);
console.log("fetch title:", user.title);
console.log("fetch completed:", user.completed);
```



```
------------------------ as 斷言 、fetch api ------------------------
inner: { userId: 1, id: 1, title: 'delectus aut autem', completed: false }
outter: { userId: 1, id: 1, title: 'delectus aut autem', completed: false }
fetch userId: 1
fetch title: delectus aut autem
fetch completed: false
```

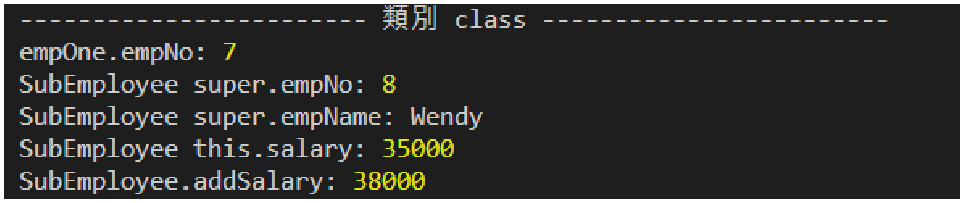
第十一章. TypeScript

```
console.log('------------------------ 類別 class ------------------------');

class Employee {

  // 公開(內外部皆可存取)
  public empNo: number;
  // 受保護(僅內部透過繼承關係存取)
  protected empName: string;
  // 私有(僅class內部)
  private empJobTitle: string;

  constructor(empNo: number, empName: string, empJobTitle: string) {
    this.empNo = empNo;
    this.empName = empName;
    this.empJobTitle = empJobTitle;
  }
}

const empOne = new Employee(7, 'Mark', 'Manager');
// 外部只能存取的到公開的成員
console.log("empOne.empNo:", empOne.empNo);

// 存取protected(受保護)、private(私有)編譯錯誤
// console.log("empOne.empName:", empOne.empName);
// console.log("empOne.empJobTitle:", empOne.empJobTitle);
```



```
------------------------ 類別 class ------------------------
empOne.empNo: 7
SubEmployee super.empNo: 8
SubEmployee super.empName: Wendy
SubEmployee this.salary: 35000
SubEmployee.addSalary: 38000
```

```
class SubEmployee extends Employee {

  private salary: number;

  constructor(empNo: number, empName: string, empJobTitle: string, salary: number) {
    super(empNo, empName, empJobTitle);
    this.salary = salary;
  }

  showEmp() {
    console.log("SubEmployee super.empNo:", this.empNo);
    console.log("SubEmployee super.empName:", this.empName);
    console.log("SubEmployee this.salary:", this.salary);
    // 存取private(私有)編譯錯誤
    // console.log("SubEmployee super.empJobTitle:", this.empJobTitle);
  }

  addSalary(money: number): number {
    return this.salary + money;
  }
}

const empTwo = new SubEmployee(8, 'Wendy', 'Accounting', 35000);
empTwo.showEmp();

const newSalary = empTwo.addSalary(3000);
console.log("SubEmployee.addSalary:", newSalary);
```

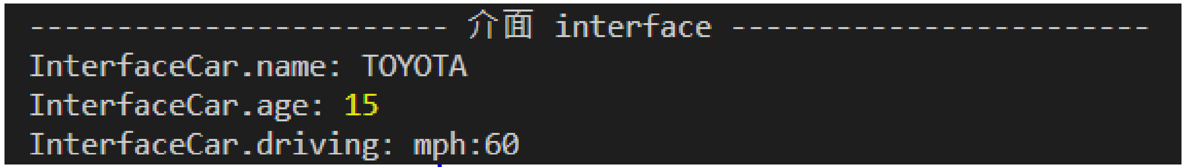


```
------------------------ 類別 class ------------------------
empOne.empNo: 7
SubEmployee super.empNo: 8
SubEmployee super.empName: Wendy
SubEmployee this.salary: 35000
SubEmployee.addSalary: 38000
```

第十一章. TypeScript

```
import { Car } from './Car';

console.log('----------------------- 介面 interface -----------------------');

// 透過interface規範必須要有的欄位及方法
class CarImpl implements Car {
   name: string;
   age: number;

   constructor(name: string, age: number) {
      this.name = name;
      this.age = age;
   }

   driving(mph: number): string {
      return "mph:" + mph;
   }
}

// 多型(polymorphism)
const carOne: Car = new CarImpl('TOYOTA', 15);
console.log("InterfaceCar.name:", carOne.name);
console.log("InterfaceCar.age:", carOne.age);
console.log("InterfaceCar.driving:", carOne.driving(60));
```

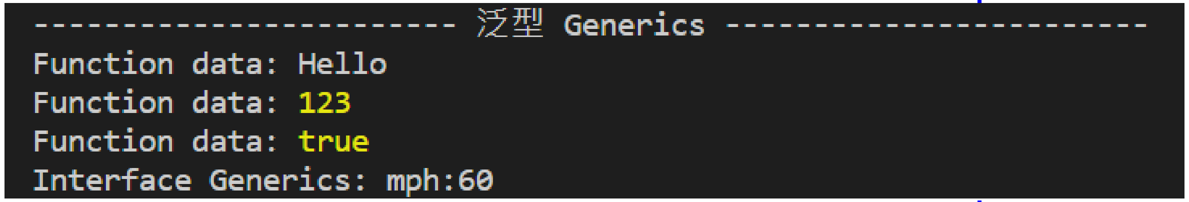
```
// 透過interface規範必須要有的欄位及方法
export interface Car {
   name: string;
   age: number;
   driving: (mph: number) => string;
}
```



```
----------------------- 介面 interface -----------------------
InterfaceCar.name:  TOYOTA
InterfaceCar.age:  15
InterfaceCar.driving:  mph:60
```

第十一章. TypeScript

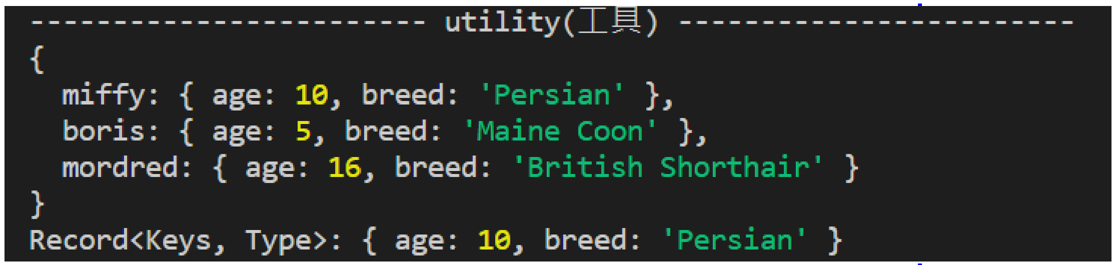
```
console.log('------------------------ 泛型 Generics ------------------------');

// 運用泛型可彈性的在使用函數、類別時才宣告實際使用的型別
function print<T>(data: T) {
   console.log('Function data:', data);
}
print<string>('Hello');
print<number>(123);
print<boolean>(true);

interface Generics<T, R> {
   driving: (arg: T) => R;
}

class GenericsImpl implements Generics<number, string> {
   driving(arg: number): string {
      return "mph:" + arg;
   }
}

const generics = new GenericsImpl();
console.log('Interface Generics:', generics.driving(60));
```



```
------------------------ 泛型 Generics ------------------------
Function data: Hello
Function data: 123
Function data: true
Interface Generics: mph:60
```

第十一章. TypeScript

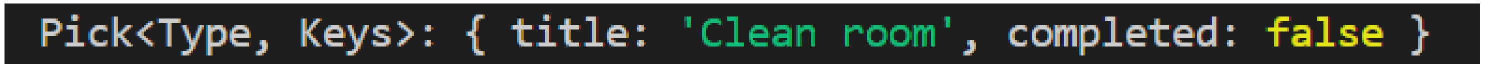
```
console.log('----------------------- utility(工具) -----------------------');
// Typescript內建
// https://www.typescriptlang.org/docs/handbook/utility-types.html

// 1.Record<Keys, Type> 紀錄
// 相當於JAVA Map
interface CatInfo {
   age: number;
   breed: string;
}

const cats: Record<string, CatInfo> = {
   'miffy': { age: 10, breed: "Persian" },
   'boris': { age: 5, breed: "Maine Coon" },
   'mordred': { age: 16, breed: "British Shorthair" },
};

console.log(cats);
console.log('Record<Keys, Type>:', cats['miffy']);
```



```
------------------------ utility(工具) ------------------------
{
  miffy: { age: 10, breed: 'Persian' },
  boris: { age: 5, breed: 'Maine Coon' },
  mordred: { age: 16, breed: 'British Shorthair' }
}
Record<Keys, Type>: { age: 10, breed: 'Persian' }
```

第十一章. TypeScript

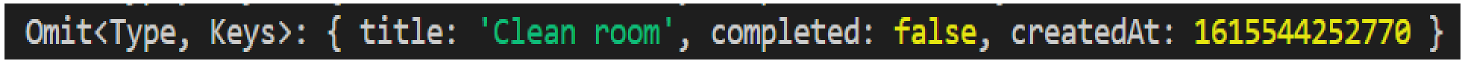
```
// 2.Pick<Type, Keys>挑選
// 透過Pick可從interface挑選"需要"的「欄位」設置給新宣告的Type,就可省略程式碼欄位宣告
interface Todo {
   title: string;
   description: string;
   completed: boolean;
}

type TodoPreview = Pick<Todo, "title" | "completed">;

const todo: TodoPreview = {
   title: "Clean room",
   completed: false,
};

console.log("Pick<Type, Keys>:", todo);
```



```
Pick<Type, Keys>: { title: 'Clean room', completed: false }
```

第十一章. TypeScript

```
// 3.Omit<Type, Keys>忽略
// 透過Omit可從interface忽略"不需要"的欄位設置給新宣告的Type,就可省略程式碼欄位宣告
interface TodoTwo {
   title: string;
   description: string;
   completed: boolean;
   createdAt: number;
}

type TodoPreviewTwo = Omit<TodoTwo, "description">;

const todoTwo: TodoPreviewTwo = {
   title: "Clean room",
   completed: false,
   createdAt: 1615544252770,
};

console.log("Omit<Type, Keys>:", todoTwo);
```



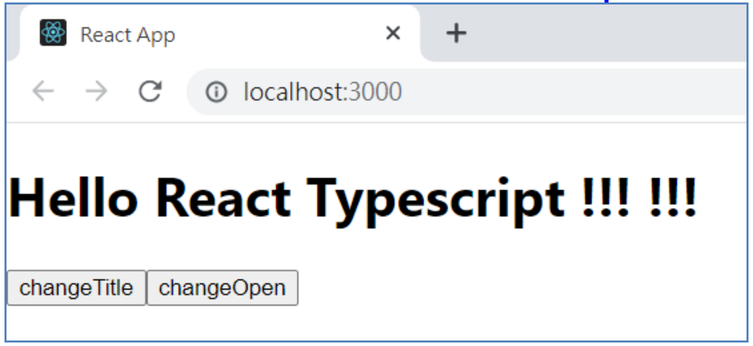
```
Omit<Type, Keys>: { title: 'Clean room', completed: false, createdAt: 1615544252770 }
```

第十一章. TypeScript

- 建立 React typescript專案

  npx create-react-app my-react-typescript-app --template typescript

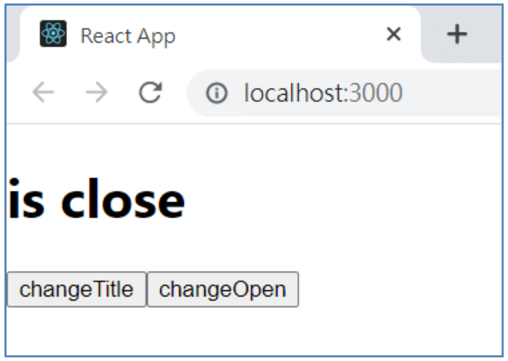
```
const App: React.FC = () => {

  const [title, setTitle] = useState<string>('Hello React Typescript');
  const [isOpen, setIsOpen] = useState<boolean>(true);

  const changeTitle = () => {
    setTitle( (oldTitle: string) => (`${oldTitle} !!!`) );
  }

  const changeOpen = () => {
    setIsOpen( (isOpen: boolean) => !isOpen );
  }

  return (
    <div>
      <Title name={title} isOpen={isOpen} />
      <button onClick={changeTitle}>changeTitle</button>
      <button onClick={changeOpen}>changeOpen</button>
    </div>
  );
}

export default App;
```



React App

localhost:3000

Hello React Typescript !!! !!!



React App

localhost:3000

is close

第十一章. TypeScript

```
// interface可被擴充與繼承
interface TitleProps {
  name: string
}

interface TitleProps {
  isOpen: boolean
}

// Typescript可以避免資料類型所照成的錯誤
// React.FC<TitleProps> 表示為 React.FunctionComponent<P>
// 將TitleProps介面直接解構於參數列上
const Title: React.FC<TitleProps> = ({ name, isOpen }) => {
  return (
    isOpen ? <h1>{name}</h1> : <h1>is close</h1>
  )
}
```