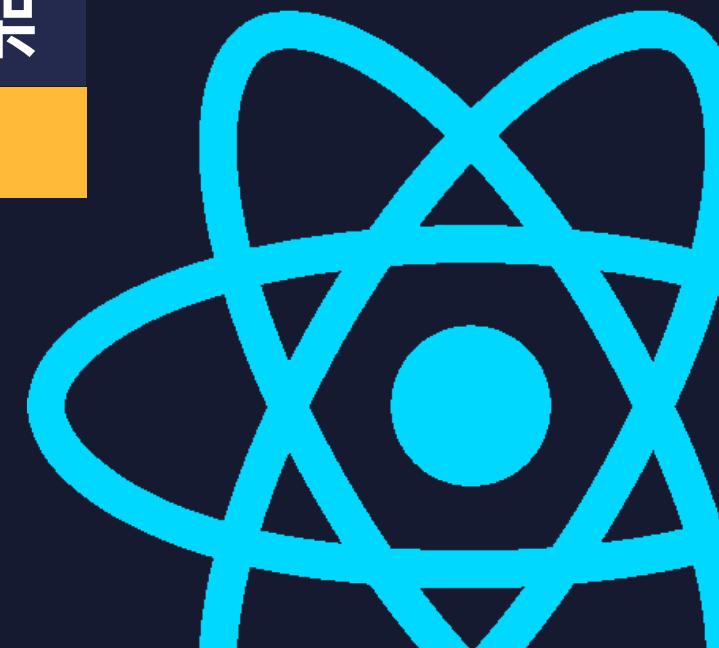


React 前端工程師必學新世代框架

React 介紹



內容大綱

01. React 是什麼

02. React 能作什麼

03. 實際應用客戶

04. 五大主要特性



```
then(response => {
  setProgress('finished');

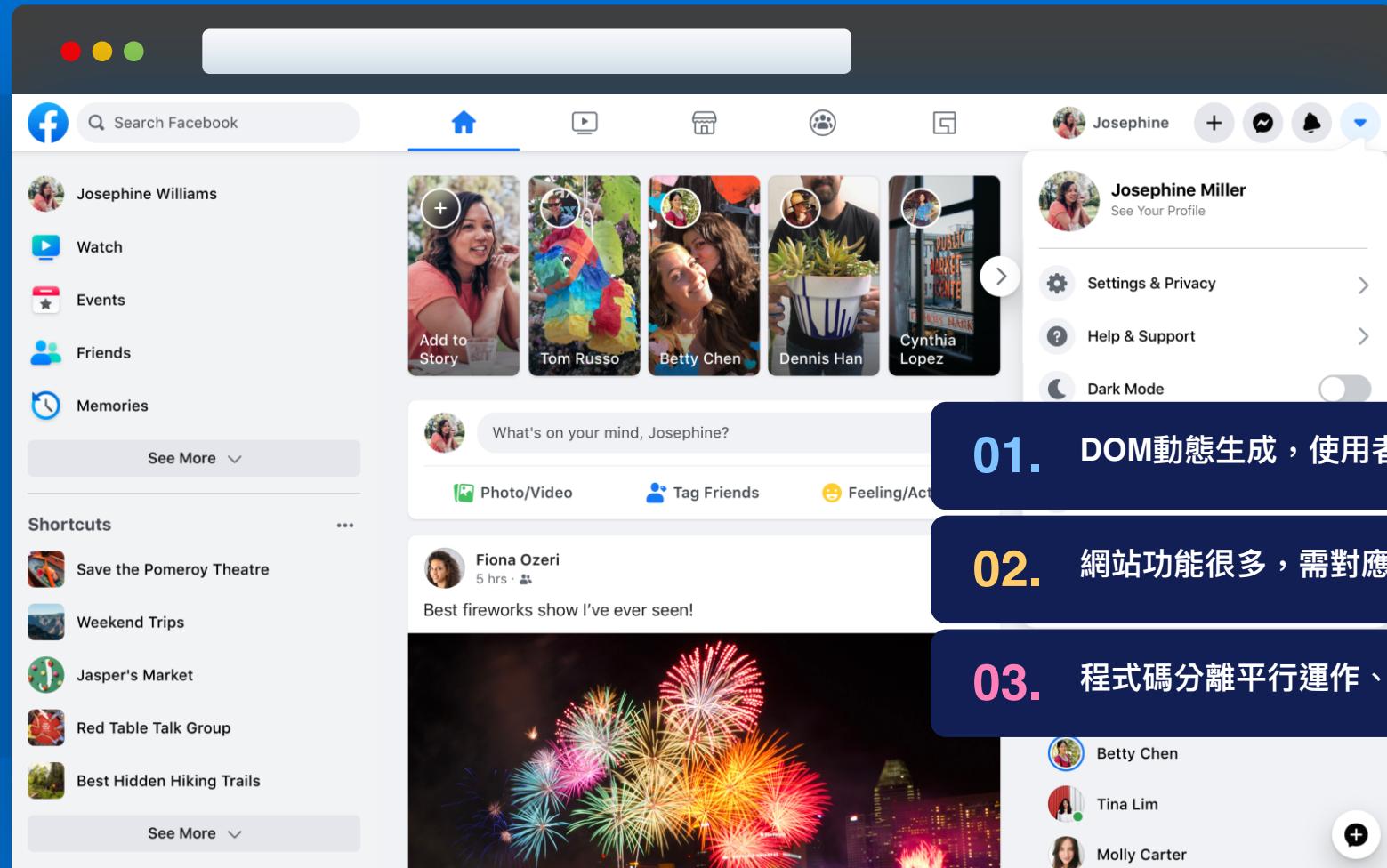
  (
    v className={`progress-button`}
      <span className="loading-bar">
        <button className="button">
          <span className="percentage">
            </button>
            <span className="percentage">
              iv>
                uilt App;
```

用來建立 **使用者介面(UI)** 的 **JavaScript** 函式庫

- > 由 Facebook 創造與維護
- > 最新版本 v17 (自2013年發佈)
- > 用於”開發網頁應用程式”
- > MIT 開放原始碼授權

“並非完整的程式框架(Framework)，接近傳統
軟體 MVC 設計模式中的 View(視圖)”

Facebook



01. DOM動態生成，使用者操作後變多

02. 網站功能很多，需對應不同的資料源

03. 程式碼分離平行運作、優先權排程

實際應用客戶

facebook



NETFLIX

amazon

P PayPal

twitch

SAMSUNG

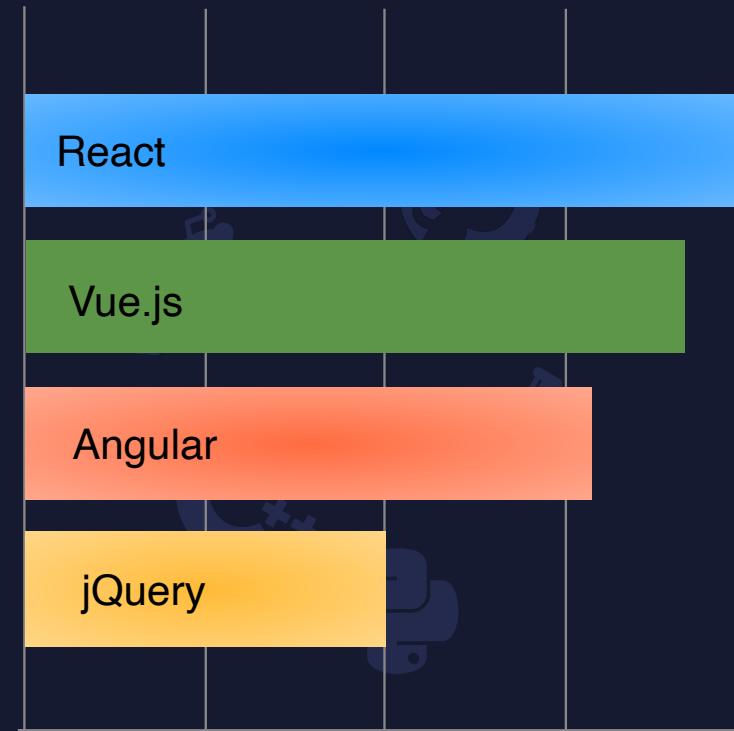
yahoo!

實際應用客戶 中文網站



開發者調查

- > 最普遍/最想要使用的網站框架第一名
(stackoverflow 2021)
- > Top 10k 網站中是第六普遍的 JS 函式庫
(Builtwith.com)
- > 全世界有 120 萬的網站使用
(similartech.com)
- > 全世界有 840 萬的專案使用(github.com)



開發者最喜愛的網站框架 (StackOverflow 2021)

React 五大主要特性

Virtual(虛擬) DOM

在程式碼中非真實的 DOM 元素結構



單向資料流

單一資料流動方向，從父母到子女元件
(從外到內)



宣告式程式開發

如何更動與呈現，交由 React 進行調和一致與差

異比較演算

JSX語法

搭配 虛擬 DOM 的語法，在程式碼中嵌入類似 HTML 碼的標記

元件為基礎

開發採用元件分離與組合的方式



Virtual(虛擬) DOM

React中自行管理的DOM結構，用於差異比較
後再與真實DOM作渲染



React應用裡的DOM

Virtual DOM



reconciliation(調合一致)
render(渲染)



網頁上真實的DOM

Real DOM

JSX 語法

JS 是 JavaScript，X 是 XML
一種針對 JavaScript 語言語法的擴充

- > React自創的在JS程式碼中建立DOM元素與使用自訂元件標記的語法
- > JSX中使用花括號({})嵌入JS表達式，具有求值運算作用
- > 是React.createElement方法的簡寫法，只能在React程式碼裡撰寫
- > 需要透過babel編譯過才能在JS引擎中執行



```
const element = <h1>Hello React!</h1>
```



BABEL 編譯



```
'use strict'
```

```
const elemennt = React.createElement('h1', null, 'Hello React!')
```



"元件" 為基礎的開發方式

以類別或函式，加入 狀態(state) 與 屬性(props) 特性來開發元件

拆分可重覆利用的功能和介面為一個個元件，然後再組合它們



"單向" 資料流

元件間依靠 屬性(props) 特性來傳遞資料，只能由父母元件傳遞到子女元件

優點為在複雜的結構中容易除錯與最佳化



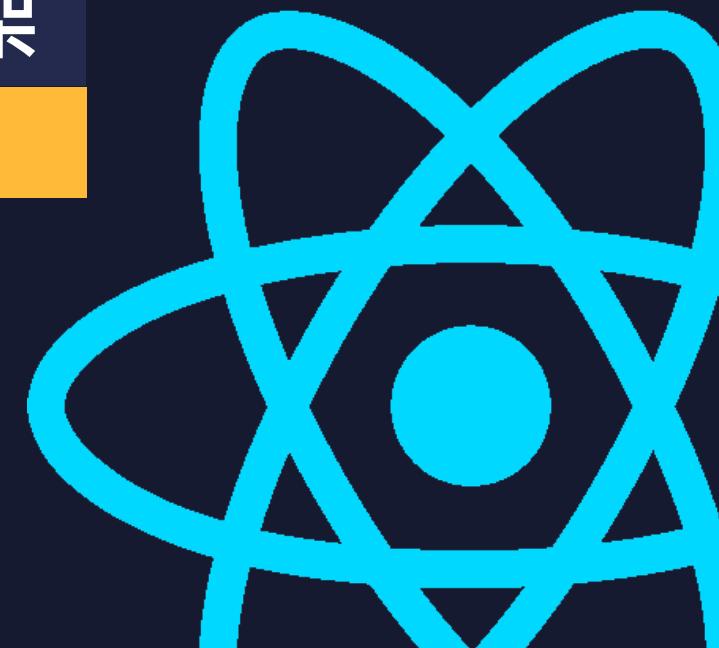
"宣告式" 程式開發

使用類似 HTML 語法來撰寫元件，DOM 的事件與處理全權交由 React

React 幫助開發者建立容易維護與除錯的應用程式，而且足夠快速的進行 DOM 處理

React 前端工程師必學新世代框架

開發環境建置



開發環境建置

01



伺服器 / 套件管理工具
Node.js / Yarn

- Npm 套件管理工具(內附)
- Yarn 工具(額外安裝)

02



程式碼撰寫工具
Visual Studio Code

- 中文語言套件
- Babel JavaScript
- ESLint
- Prettier
- 其它搭配套件

03



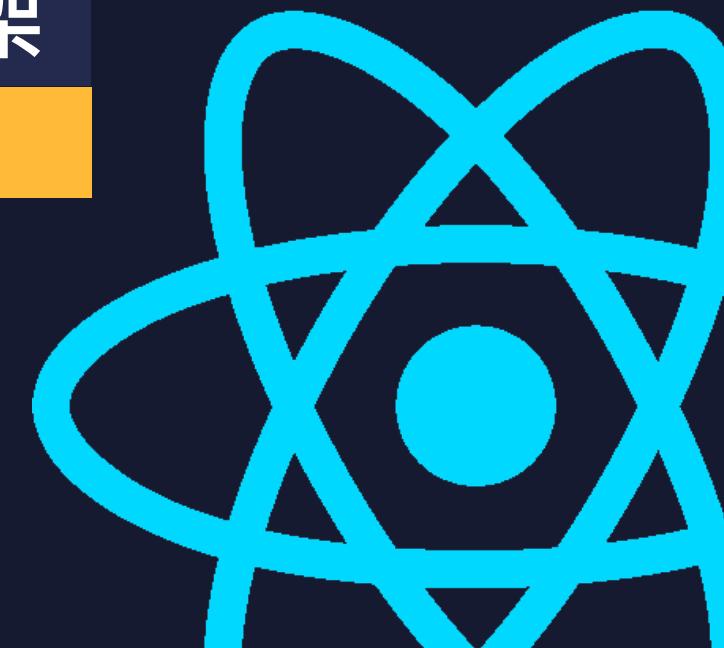
瀏覽器
Google Chrome

- React Developer Tools 外掛



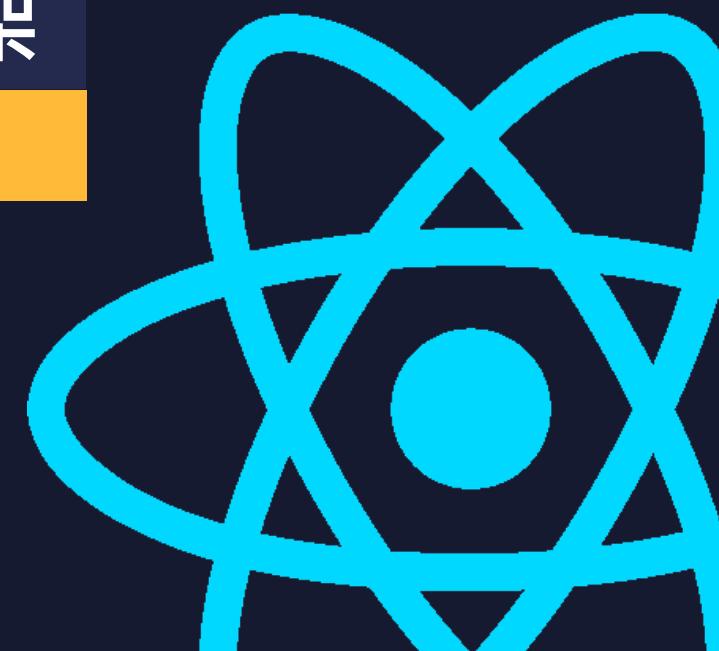
React 前端工程師必學新世代框架

開發專案建立與設定



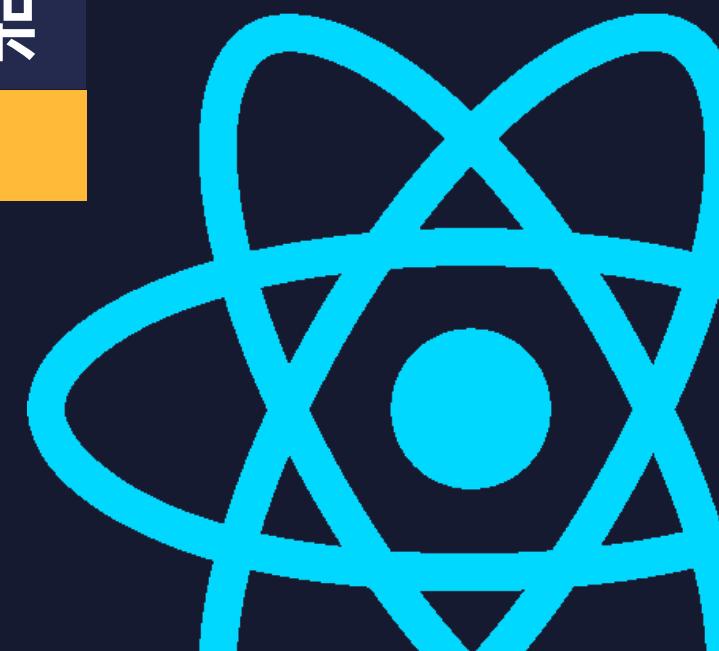
React 前端工程師必學新世代框架

類別型元件



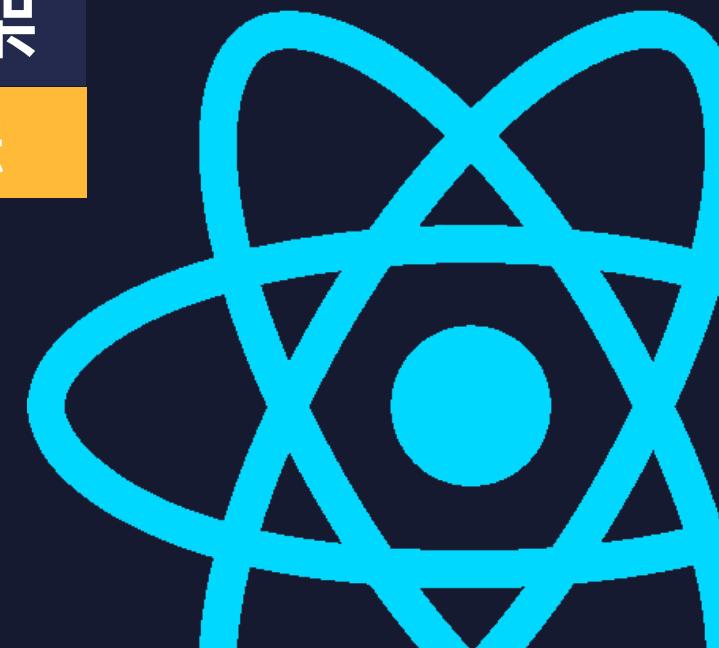
React 前端工程師必學新世代框架

函式型元件



React 前端工程師必學新世代框架

Virtual DOM 與 JSX 語法



Virtual(虛擬) DOM

React中自行管理的DOM結構，用於差異比較
後再與真實DOM作渲染



React應用裡的DOM

Virtual DOM



reconciliation(調合一致)
render(渲染)



網頁上真實的DOM

Real DOM

Virtual(虛擬) DOM

為何要讓 React 管理網頁上的 DOM 結構，而
不是由開發者來控制管理？



React不會比直接DOM處理更快，它只是協助開發者建立
可維護的應用程式，而且”**足夠快速**”的進行DOM處理

JSX 語法

JS 是 JavaScript，X 是 XML
一種針對 JavaScript 語言語法的擴充

- > React自創的在JS程式碼中建立DOM元素與使用自訂元件標記的語法
- > JSX中使用花括號({})嵌入JS表達式，具有求值運算作用
- > 是React.createElement方法的簡寫法，只能在React程式碼裡撰寫
- > 需要透過babel編譯過才能在JS引擎中執行



```
const element = <h1>Hello React!</h1>
```



BABEL 編譯



```
'use strict'
```

```
const elemennt = React.createElement('h1', null, 'Hello React!')
```

JSX 語法

一個render方法中 (函式型元件指最後 return 部份) 只能有一個根元素(父母元素)回傳

錯誤語法

```
return (
  <h1>Title One</h1>
  <p>Some Text</p>
)
```

正確語法

```
return (
  <>
    <h1>Title One</h1>
    <p>Some Text</p>
  </>
)
```

這與babel編譯為createElement方法有關，將多個DOM元素包裹在一個最外層的元素即可解決

JSX 語法

React.Fragment 元件(<>...</>)
不會產生多餘的render標記

```
return (  
  <div>  
    <h1>Title One</h1>  
    <p>Some Text</p>  
  </div>  
)
```

```
return (  
  <>  
    <h1>Title One</h1>  
    <p>Some Text</p>  
  </>  
)
```

當元件很多層級與分散很多時，會產生多餘的div標記，有可能會造成css樣式問題或效能的影響

JSX 語法

自我封閉 HTML 元素標記一定需要最後結尾
封閉標籤(`>`)

一定要加上這個標記，
不然會有錯誤發生



```
const imgTag = 
const inputTag = <input type="text" name="title" />
const brTag = <br />
const hrTag = <hr />
```



JSX 語法

英文大寫開頭的標記為自訂元件標記
英文小寫開頭的標記為HTML使用的標記



```
{/* 內建HTML元素 */}  
<h1>Title One</h1>  
<p>Some Text</p>
```

```
{/* 自訂元件 */}  
<MyButton title="One" />  
<MyComponent />
```

JSX 語法

元件的加入事件為開頭為 on 的屬性
值必定為一個函式



```
<h2 onClick={() => setTotal(total + 1)}>{total}</h2>
```

JSX 語法

註解方式

VSCode中選定後按下「Ctrl + /」
(macOS為「CMD+/」)



JSX 語法

JS關鍵字的屬性需要改名稱，最常見的是
class -> className 與 for -> htmlFor

寫為class還是能正常執行，
但在執行時主控台會出現警告



```
<h1 className="title1">Title</h1>
<label htmlFor="namedInput">Name:</label>
<input id="namedInput" type="text" name="name" />
```

JSX 語法

Style屬性為物件值

建議少用內聯 style 屬性，改為 CSS 類別指
定



```
<p style={{color:'red', fontSize: '10px', backgroundColor:'#ffffff'}}>Title</p>
```

JSX 語法

使用花括號({})來嵌入JS變數或表達式



```
<h1>{1 + foo}</h1>
<MyButton
  text={'Click Me!'}
  onClick={() => alert('Hello')}
/>
```

JSX 語法

在JSX語法中要嵌入陣列
經常使用 map 方法



```
<>
{students.map((student, index) => (
  <p key={index}>Hello, {student.name}!</p>
))}

</>
```

JSX 語法

列表項目標記，要使用key屬性



```
const todoItems = todos.map(  
  (todo) => <li key={todo.id}>{todo.text}</li>  
)
```

Key 與React要進行列表項目標記處理的最佳化有關，通常會出現在使用map方法與””、“<td>”等

JSX 語法

if 流程控制的表達式語法
(使用邏輯與 &&)



意思為「如果 count 為 true，則呈現後面接著的元素」
(使用 Truthy 與 Falsy 來判斷)

```
const count = 0  
return <div>{count && <h1>Messages: {count}</h1>}</div>
```

JSX 語法

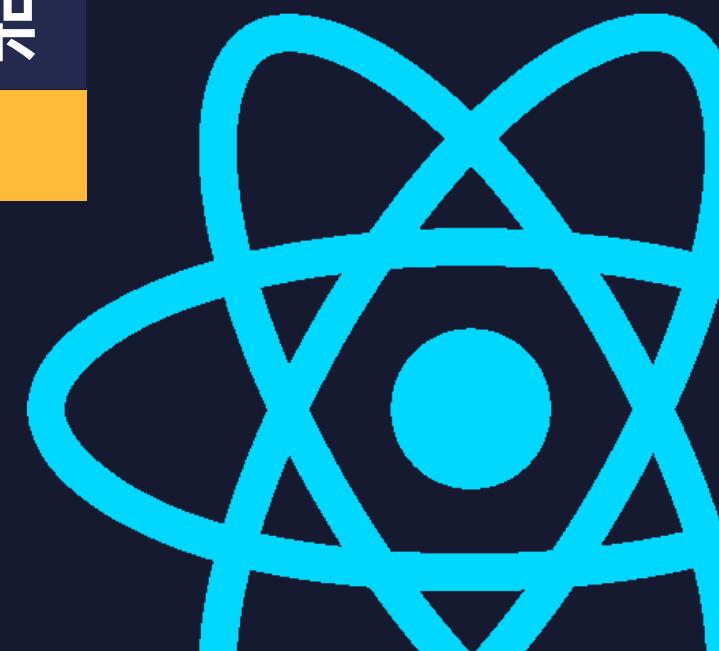
if...else 流程控制的表達式語法
(使用三元表達式`?:`)



```
<>
  {isLoggedIn ? (
    <LogoutButton onClick={this.handleLogoutClick} />
  ) : (
    <LoginButton onClick={this.handleLoginClick} />
  )}
</>
```

React 前端工程師必學新世代框架

狀態



State(狀態)

是會因為使用者操作後，而不斷改變的資料值

State(狀態)

- 行進速度
- 輪胎磨損情況
- 行進距離
- ...



Props(屬性)

- 車身尺寸
- 品牌
- 顏色
- 重量
- ...

State(狀態)

- 目前輸入的文字
- 是否正在輸入文字中
- ...

文字輸入框

Props(屬性)

- 寬度
- 高度
- 預設文字
- ...

State(狀態)

與使用者介面(UI)的互動有關
設計元件前應以**狀態為最優先的思考重點**

> 狀態為元件內部私有資料，其它元件無法直接存取到

> 類別型元件的狀態需定義在建構式之中

> 函式型元件的狀態需定義在函式主體區塊最上方，使用”解構指定值”語法

類別型元件

```
● ● ●  
constructor() {  
  super()  
  this.state = {  
    total: 0,  
  }  
}
```

函式型元件

```
● ● ●  
const [total, setTotal] = useState(0)
```

State(狀態)

元件內部的私有資料，只有透過改變狀態
(`setState`) 才能改變真實網頁上的DOM

Render 渲染

將HTML原始碼轉譯為你所看到的網頁樣貌的過程，此為瀏覽器核心功能。

4

更動真實網頁上的DOM



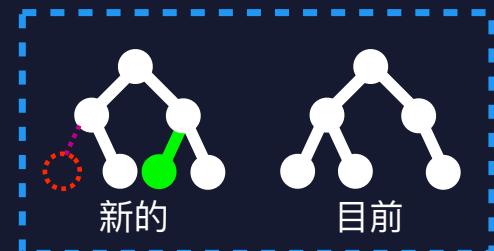
1

使用者觸發事件



3

比較虛擬DOM結構改變



元件

state

2

資料改變，觸發
re-render

render()

setState

唯一能更動 State(狀態) 的方法

> 不可直接使用JS中的指定值運算語法來更動
(ex. `x = 1`)

> 更動網頁上的UI -> State(狀態) 需要被改變
-> 要呼叫 `setState` 方法

> 類別型元件與函式型元件語法有不少差異，
使用時務必注意(ex. 傳入參數值不同)

> `setState`方法執行是有異步(非同步)特性，使
用時要注意程式執行時的邏輯

類別型元件



```
<h1 onClick={() => this.setState({ total: this.state.total + 1 })}>  
    {this.state.total}  
</h1>
```

函式型元件



```
<h1 onClick={() => getTotal(total + 1)}>{total}</h1>
```

useState 函式型元件勾子(hooks)

使用ES6中解構賦值的新語法

初始化值設定在傳入參數



```
const [total, setTotal] = useState(0)
```

useState執行後回傳一個陣列
[獲得值的變數, 設定值的函式]

setState 類別型元件

狀態本身就是物件值，
setState也用物件值設定

```
 1 constructor() {
 2     super()
 3     this.state = {
 4         name: 'Eddy',
 5         phone: '0911-222-333',
 6     }
 7 }
 8
 9 render() {
10     return (
11         <>
12         <h1>類別型元件</h1>
13         <input
14             type="text"
15             value={this.state.name}
16             onChange={(event) => this.setState({ name: event.target.value })}
17         />
18         <input
19             type="text"
20             value={this.state.phone}
21             onChange={(event) => this.setState({ phone: event.target.value })}
22         />
23     </>
24 )
25 }
```

setXXX 函式型元件

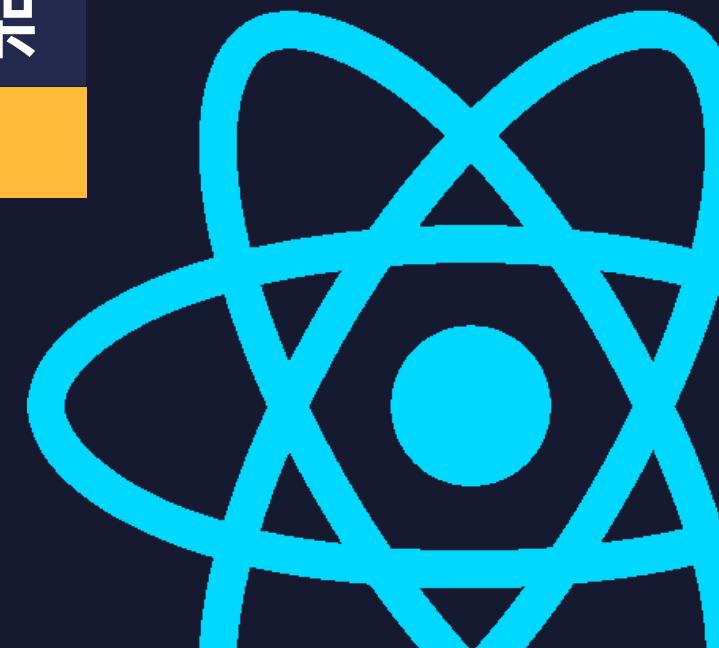
狀態不一定是什麼資料類型，注意使用物件值
時會被”覆蓋”(對照類別型元件為”合併”)

```
1 const [data, setData] = useState({ name: 'Eddy', phone: '0911-222-333' })
2
3 return (
4   <>
5     <h1>函式型元件</h1>
6     <input
7       type="text"
8       value={data.name}
9       onChange={(event) => setData({ ...data, name: event.target.value })}
10      />
11     <input
12       type="text"
13       value={data.phone}
14       onChange={(event) => setData({ ...data, phone: event.target.value })}
15      />
16    </>
17  )
```

需要自行合併物件值，不然其它
物件屬性會被丟棄

React 前端工程師必學新世代框架

狀態異步執行與對應策略



setState 異步執行

狀態在進行改變時，React會進行合併與最佳化等內部工作，必會有異步執行的特性

```
1 <h1
2   onClick={() => {
3     setTotal(total + 1) // 呼叫要變動狀態的方法
4     console.log(total) // 想要得到變動後的狀態
5   }}
6 >
7 {total}
8 </h1>
```

絕對得不到變動後的狀態，因第3行的
setState(或setXXX) 方法是異步執行
(比第4行還晚執行)

setState 異步執行 - 對應策略

策略一：先用變數接住最後更動的值 (推薦！)

先運算出最後會變動的狀態值是什麼，
接著再用它作其它的處理，例如設定給 React
狀態，或是傳到伺服器、輸出在主控台...等

```
1 <h1
2   onClick={() => {
3     const newTotal = total + 1 // 先運算出變動後結果值
4     setTotal(newTotal) // 此處呼叫要變動狀態的方法
5     console.log(newTotal) // 此處想要得到變動後的狀態
6   }}
7 >
8   {total}
9 </h1>
```

setState 異步執行 - 對應策略

策略二：componentDidUpdate生命周期方法

```
● ● ●  
1 class App extends React.Component {  
2   constructor() {  
3     super()  
4     this.state = { total: 0 }  
5   }  
6  
7   componentDidUpdate() {  
8     // 在total有改變後，取得total  
9     console.log(this.state.total)  
10  }  
11  
12  render() {  
13    return (  
14      <h1  
15        onClick={() => {  
16          // 此處呼叫要變動狀態的方法  
17          this.setState({ total: this.state.total + 1 })  
18        }}  
19      >  
20      {this.state.total}  
21      </h1>  
22    )  
23  }  
24 }
```

setState 異步執行 - 對應策略

策略二：componentDidUpdate生命周期方法

```
1 function App() {
2   const [total, setTotal] = useState(0)
3
4   // 模擬生命周期方法 componentDidUpdate
5   useEffect(() => {
6     console.log(total) // 在total有改變後，取得total
7   }, [total])
8
9   return (
10    <h1
11      onClick={() => {
12        setTotal(total + 1) // 此處呼叫要變動狀態的方法
13      }}
14    >
15      {total}
16    </h1>
17  )
18 }
```

setState 異步執行 - 對應策略

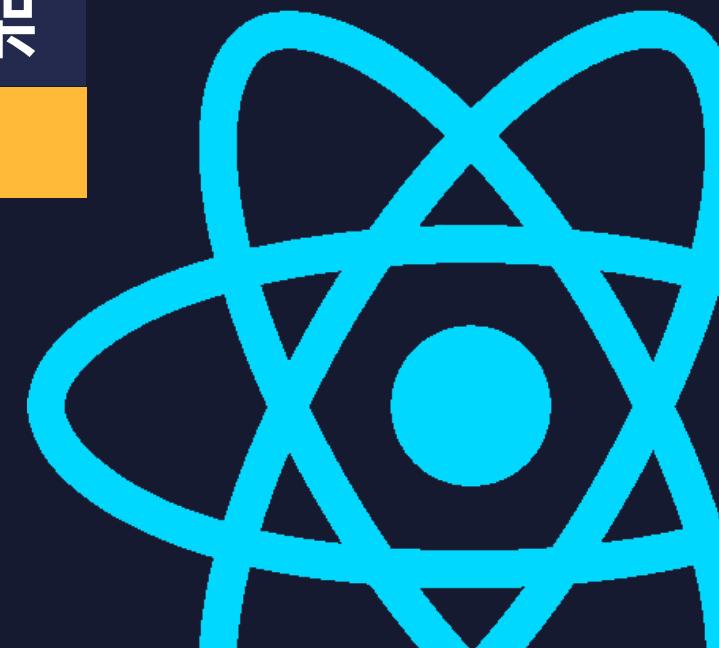
策略三：使用setState第二個傳入參數值(為callback函式)，只有類別型元件可以這樣用

```
1 <h1
2   onClick={() => {
3     // 此處呼叫要變動狀態的方法
4     this.setState(
5       { total: this.state.total + 1 },
6       () => {
7         // 在total有改變後，取得total
8         console.log(this.state.total)
9       }
10      )
11    }}
12  >
13  {this.state.total}
14 </h1>
```

setState方法第二個傳入參數值，
在更動狀態完成後會進行呼叫 (只
有類別型元件可以使用)

React 前端工程師必學新世代框架

屬性



Props(屬性)

固定的屬性值，來自父母元件或是預設值

State(狀態)

- 行進速度
- 輪胎磨損情況
- 行進距離

...



Props(屬性)

- 車身尺寸
- 品牌
- 顏色
- 重量

...

State(狀態)

- 目前輸入的文字
- 是否正在輸入文字中

...

文字輸入框

Props(屬性)

- 寬度
- 高度
- 預設文字

...

Props(屬性)

父母元件對子女元件的溝通方式
(資料傳遞方式)

Props(屬性)值是一種物件資料，
可以透過函式的傳入參數得到



```
function Child(props) {  
  return <h1>{props.text}</h1>  
}
```



```
function Parent() {  
  return <Child text="今天開始學React!" />  
}
```

在父母元件使用了子女元件，並設定了屬性(Props)值

React 中只有定義了單方向的資料流動，也就是只有
父母元件能傳遞資料給子女元件(P->C)

Props(屬性) - 預設屬性

預設屬性，當父母元件沒給定值時會應用

```
function App() {
  return (
    <>
      <ShowName />
      <ShowName firstName="Amy" lastName="Yi" />
    </>
  )
}
```

```
function ShowName(props) {
  return (
    <h1>
      {props.firstName}, {props.lastName}
    </h1>
  )
}

// 設定預設屬性(當父母元件沒給定時應用)
ShowName.defaultProps = {
  firstName: 'Eddy',
  lastName: 'Chang',
}
```

Props(屬性) - 屬性類型

PropTypes需要額外安裝套件，用於限制傳入
屬性的類型(只有警告訊息)

```
import PropTypes from 'prop-types'

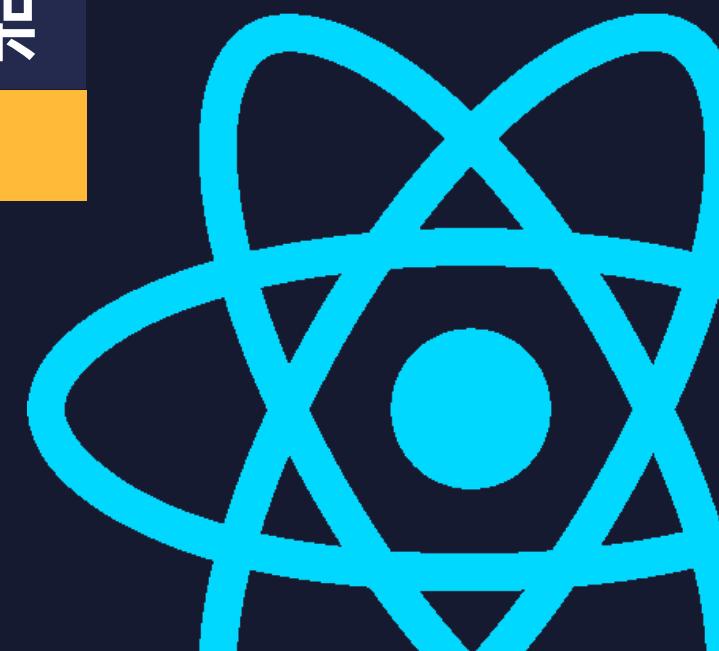
function ShowName(props) {
  return (
    <h1>
      {props.firstName}, {props.lastName}
    </h1>
  )
}

// 設定屬性的限制類型或必填等等
ShowName.propTypes = {
  firstName: PropTypes.string.isRequired,
  lastName: PropTypes.string.isRequired,
}
```

套件安裝指令：
yarn add prop-types

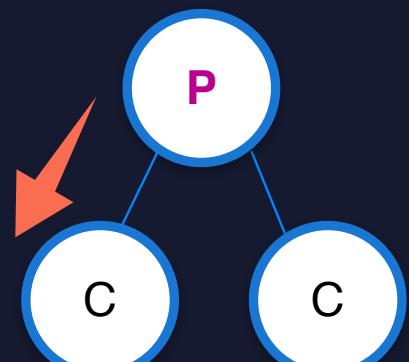
React 前端工程師必學新世代框架

元件間資料傳遞方式



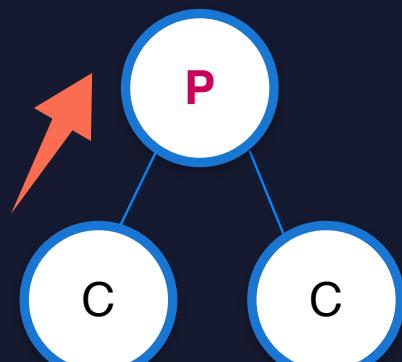
Props(屬性)

三種基本的元件溝通(資料傳遞)方式



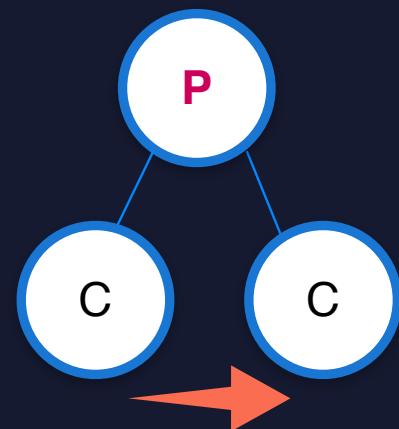
父母 -> 子女

原本就有的單向資料流



子女 -> 父母

利用callback函式

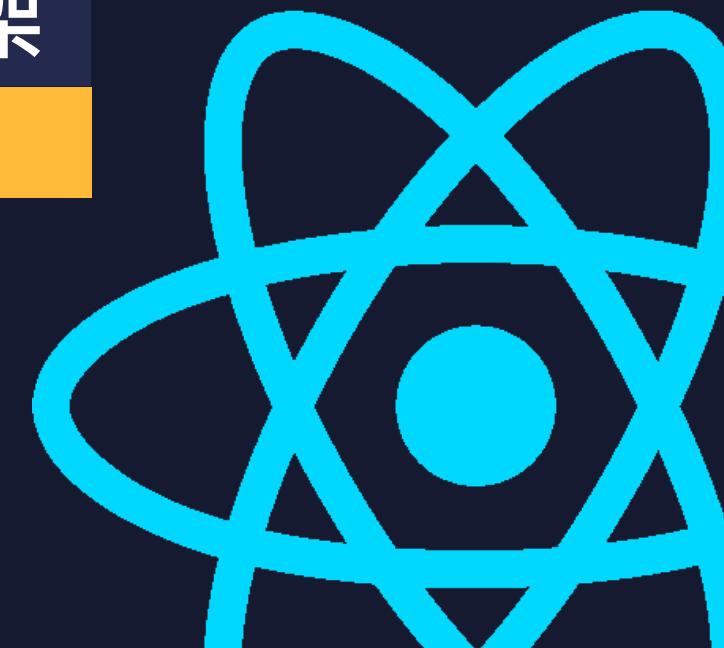


子女 -> 子女

利用父母元件與前述兩種方式

React 前端工程師必學新世代框架

線上商品訂購單範例



State(狀態) vs Props(屬性)

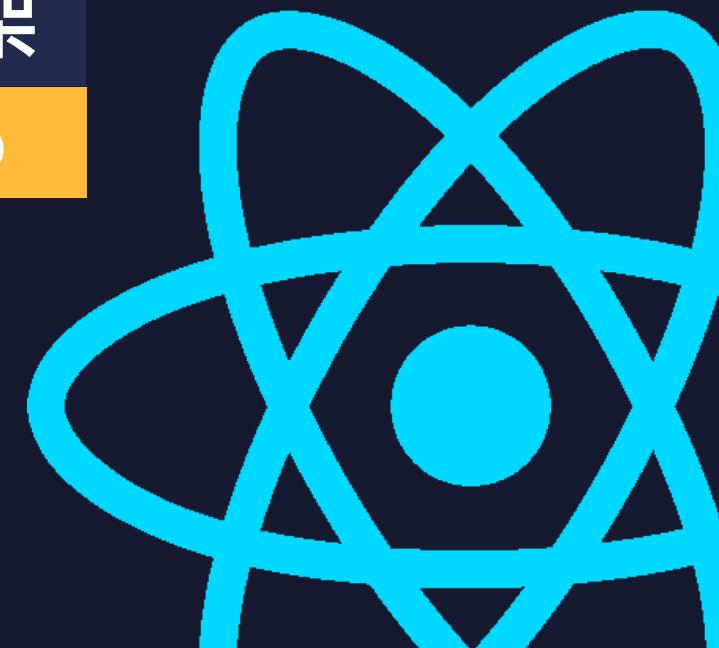
| | State(狀態) | Props(屬性) |
|-----------|------------------------|-------------|
| 用途 | 在元件內部控制目前的私有資料 | 在元件的層級間傳遞資料 |
| 資料類型 | 物件(類別型元件) / 自定義(函式型元件) | 物件 |
| 在此元件中可否改變 | 可(讀+寫) | 不可(唯讀) |
| 可否被其它元件存取 | 不可 | 可 |
| 可否被父母元件改變 | 不可 | 可 |

類別型元件 vs 函式型元件

| | 類別型元件 | 函式型元件 |
|--------|------------|--------------------|
| 主要語法 | 類別 (Class) | 函式 (Function) |
| 狀態 | 可以使用 | 可以使用 (useState 勾子) |
| 生命周期方法 | 有 | 模擬 (useEffect 勾子) |
| 優點 | 語法明確嚴謹/穩定 | 語法較為簡單 |
| 效能 | 相同 | 相同 |

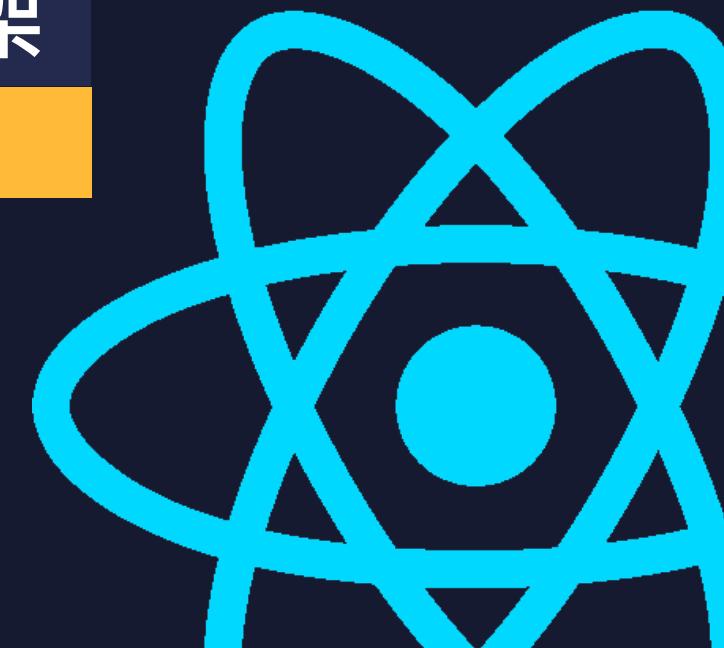
React 前端工程師必學新世代框架

在React中使用Bootstrap



React 前端工程師必學新世代框架

線上商品訂購單範例



訂購單

3種商品項目



咖啡色 T-shirt

- +

\$300

x



白色 T-shirt

- +

\$200

x



黑色 T-shirt

- +

\$450

x

← 回到商品頁

付款摘要

共 4 項目

總價

\$950

前往付款

點按+或-按鈕，
會更動到付款摘要中
的總數量和總價

開發思維與流程

先思考該怎麼設計整個應用程式，決定狀態和
如何作資料流、分拆元件和組合等等

- > 應用程式中的State狀態是什麼？
- > 有哪些元件是可以分拆或是有重覆利用性的？
- > State狀態應該位於何處？
- > 考量未來動態資料或再延伸功能的情況？

訂購單

3種商品項目

| | | | | |
|--|-------------|--------------------------------------|-------|---|
| | 咖啡色 T-shirt | - <input type="button" value="1"/> + | \$300 | × |
| | 白色 T-shirt | - <input type="button" value="1"/> + | \$200 | × |
| | 黑色 T-shirt | - <input type="button" value="1"/> + | \$450 | × |

付款摘要

共 4 項目

總價 \$950

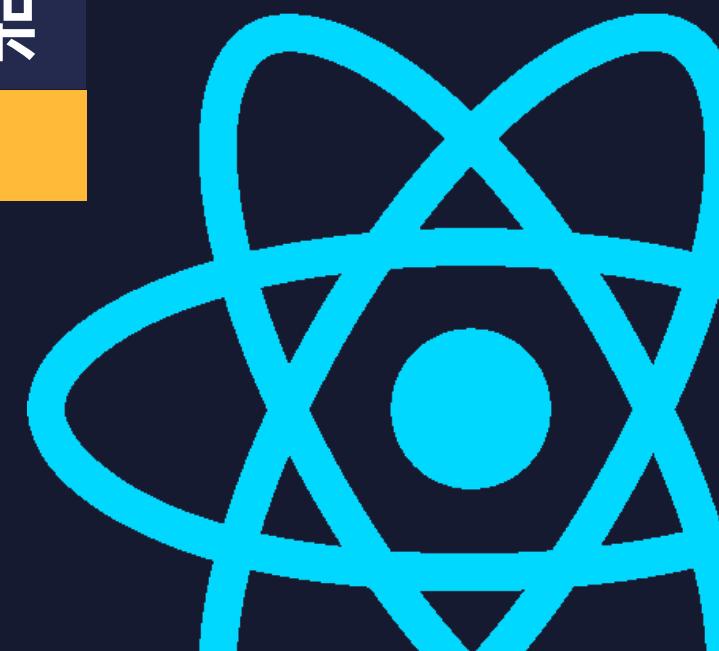
前往付款

← 回到商品頁

點按+或-按鈕，
會更動到付款摘要中
的總數量和總價

React 前端工程師必學新世代框架

表單元素



可控(controlled) vs 不可控(uncontrolled)

| | 可控(controlled) | 不可控(uncontrolled) |
|-----------|------------------------|---|
| 說明 | 由 React 控管資料狀態的表單元素 | 脫離 React 掌控狀態(原本的DOM)的表單元素，需由開發者自行控管資料的表單元件 |
| 應用搭配 | State(狀態) / 改變State的事件 | Refs(參照) |
| 資料動態或立即檢查 | 容易達成 | 送出時才檢查/開發者自行處理 |
| 資料管理 | 由父母元件管理 | DOM中由開發者自行管理 |
| 建議使用 | 大部份情況 | 少部份情況 (ex.有需要整合非react應用時) |

常用表單元素

| 種類 | 標記 | 值屬性 | 事件callback | 更新事件值 (針對 callback) | 備註 |
|---------------|--|-----------------------------------|-----------------------|-------------------------------|---|
| 文字輸入框 | <code><input type="text" /></code> | <code>value="string"</code> | <code>onChange</code> | <code>e.target.value</code> | |
| 複選框 (核取方塊) | <code><input type="checkbox" /></code> | <code>checked={boolean}</code> | <code>onChange</code> | <code>e.target.checked</code> | |
| 選項按鈕 | <code><input type="radio" /></code> | <code>checked={boolean}</code> | <code>onChange</code> | <code>e.target.checked</code> | |
| 文字輸入區域 | <code><textarea /></code> | <code>value="string"</code> | <code>onChange</code> | <code>e.target.value</code> | 與HTML不同，用value代表其中輸入文字 |
| 下拉式選單 | <code><select></code> | <code>value="option value"</code> | <code>onChange</code> | <code>e.target.value</code> | 與HTML不同，用value代表被選中的選項值(並非使用selected屬性) |

可控的表單元素 必要條件x2

- > value屬性需要對應到某個 State(狀態)值
- > 事件callback函式必需可以更動到 value屬性的對應State(狀態)值

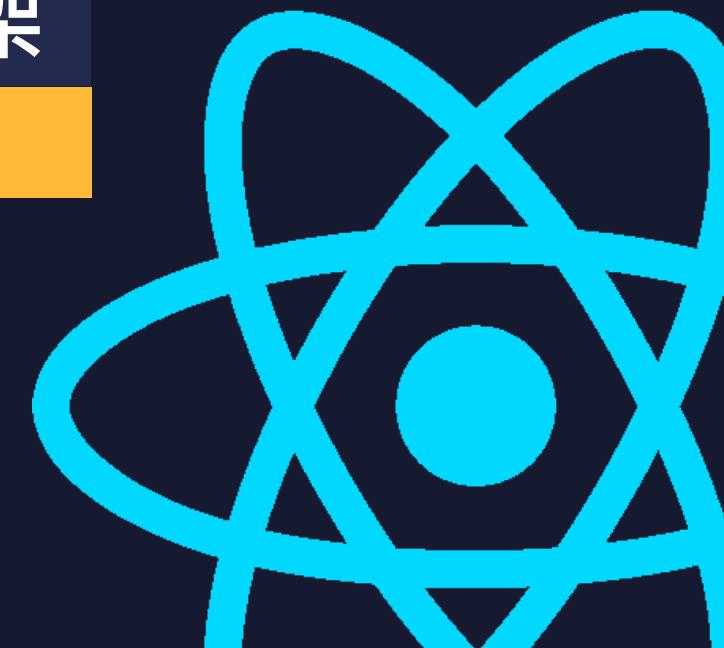
Checkbox/Radio 上述的value指的是checked屬性

```
1 function ControlledInput() {  
2   const [data, setData] = useState('')  
3  
4   return (  
5     <>  
6       <input  
7         type="text"  
8         value={data}  
9         onChange={(e) => setData(e.target.value)}  
10      />  
11    </>  
12  )  
13 }
```

value 屬性需要對應到某個狀態
更動事件的callback必需設定value對應的狀態值

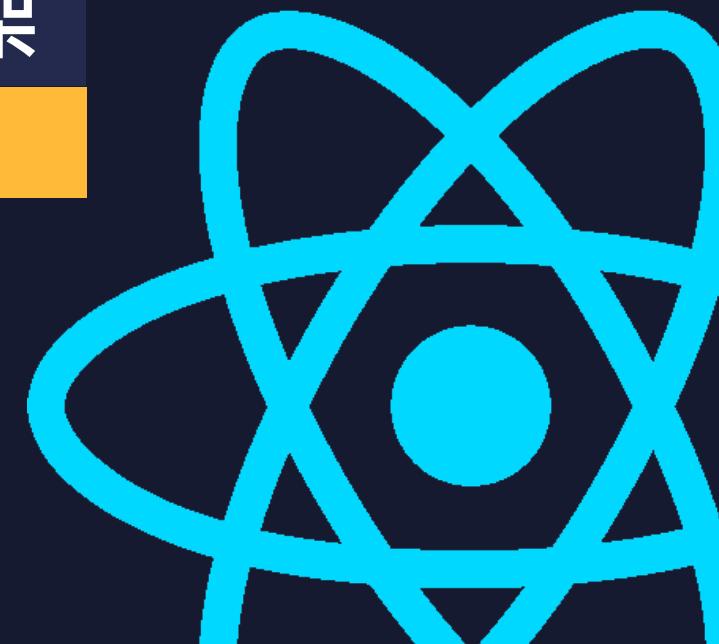
React 前端工程師必學新世代框架

樣式 - 多欄表單元素



React 前端工程師必學新世代框架

表單元素與Refs



refs是什麼

React提供的真實DOM元素實體參照物件
(ref為reference縮寫)

- > 整合其它第三方應用時(ex.地圖, 圖表, 動畫, DOM處理...)
- > 表單元素要撰寫聚焦(focus)或文字選取功能時
- > 媒體(影片、音樂)播放功能
- > 命令式動畫(imperative animations)-Web Animation API

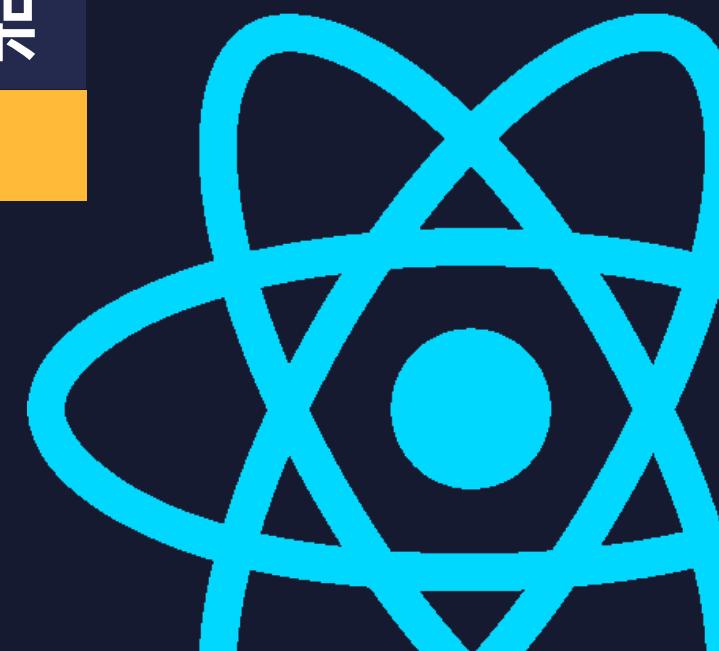
```
1 function RefsForm() {  
2   const inputEl = useRef(null)  
3  
4   return (  
5     <>  
6       <input type="text" ref={inputEl} />  
7       <button  
8         onClick={() => {  
9           inputEl.current.focus()  
10        }}>  
11         Click Me  
12       </button>  
13     </>  
14   )  
15 }  
16 }
```

宣告名稱
要相同

要加上current才能得到真
實DOM實體的參照物件

React 前端工程師必學新世代框架

樣式 - 表單元素驗証



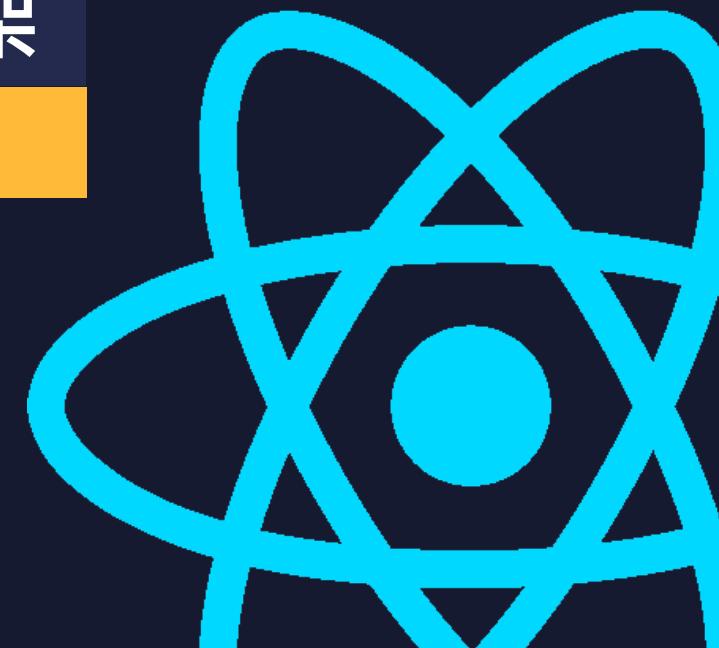
表單驗証方式

以瀏覽器端的驗証方式

- > 以JS程式碼、CSS等由開發者自行撰寫
- > 以HTML5 表單驗証的方式來驗証(需加上<form>標記) (各瀏覽器的錯誤訊息並不統一)
- > 以HTML5 表單驗証 + refs 來驗証，目的是為要自訂錯誤訊息或作Focus(聚焦)等等 (需加上<form>標記)
- > 以專門的React表單套件來驗証，目的通常是要更多的開發時彈性，和針對複雜的應用情況使用(ex. formik, React Hook Form, formsy-react)

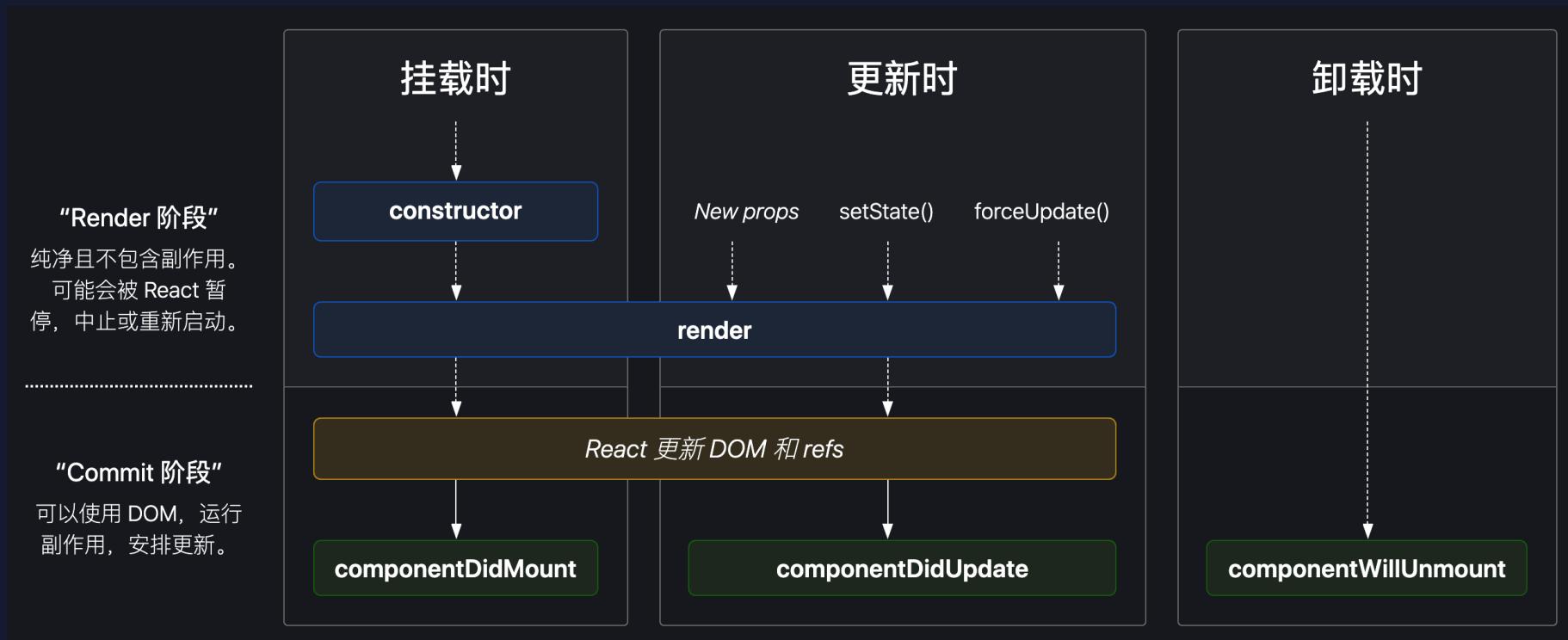
React 前端工程師必學新世代框架

生命周期方法



生命周期方法

React元件在網頁上的各個執行階段



生命周期方法

使用注意事項

- > 生命周期方法是React設計給”**類別型元件**”使用的，不是函式型元件，函式型元件可以用useEffect勾子來模擬，但要注意行為不是100%相同
- > **componentDidMount** 代表元件”**已經**”出現在網頁上，這個方法中可以使用直接DOM處理，或向伺服器要初始化資料的JS程式碼
- > **componentDidUpdate** 代表元件”**已經**”更新完成(真實DOM)，這個方法中可以得到最後更新的狀態值 (只有State更新，或接收到新的props)
- > **componentWillUnmount** 代表元件”**即將**”被移除到DOM外(ps. 不是隱藏)

constructor, render也是生命周期方法，除了這五個外，其它生命周期方法都是最佳化/除錯用進階使用

useEffect 勾子

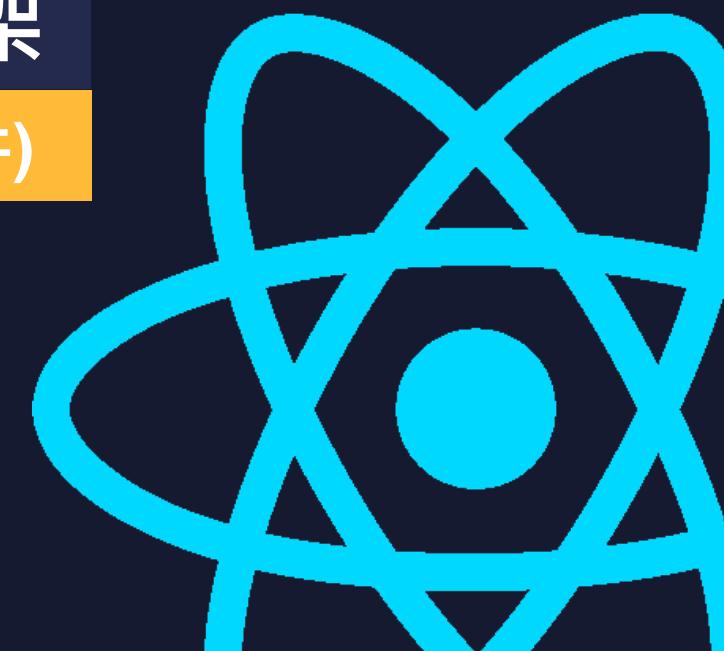
```
1 // 模擬 componentDidMount
2 useEffect(() => {
3     // 程式碼寫在這裡
4 }, [])
5
6 // 模擬 componentDidUpdate
7 // componentDidMount 時也會執行一次
8 useEffect(() => {
9     // 程式碼寫在這裡
10}, [total])
11
12 // 模擬 componentWillUnmount
13 useEffect(() => {
14     return () => {
15         // 程式碼寫在這裡
16     }
17}, [total])
```

有兩個傳入參數：
- Callback函式
- 相依變數(狀態/屬性)陣列

完全沒寫代表每次
render都會執行，寫
為空白陣列為元件初
次render時執行一次

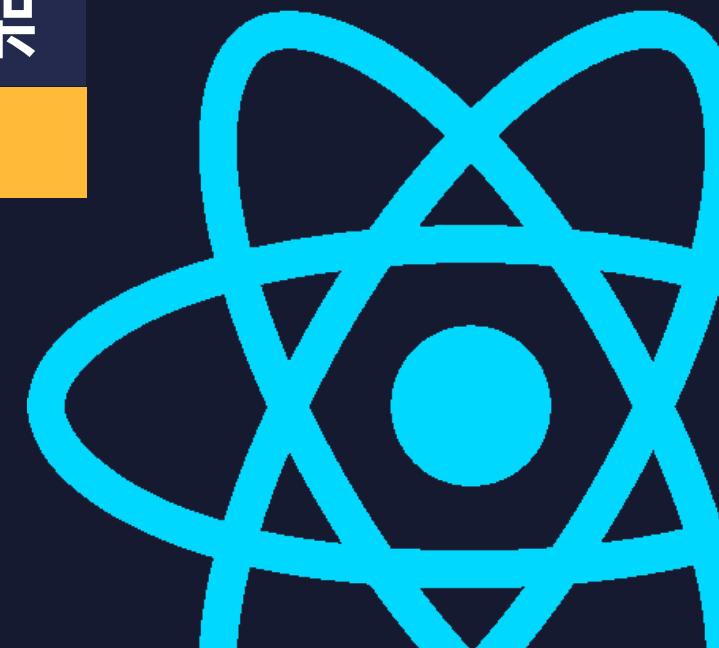
React 前端工程師必學新世代框架

useEffect 勾子(函式型元件)



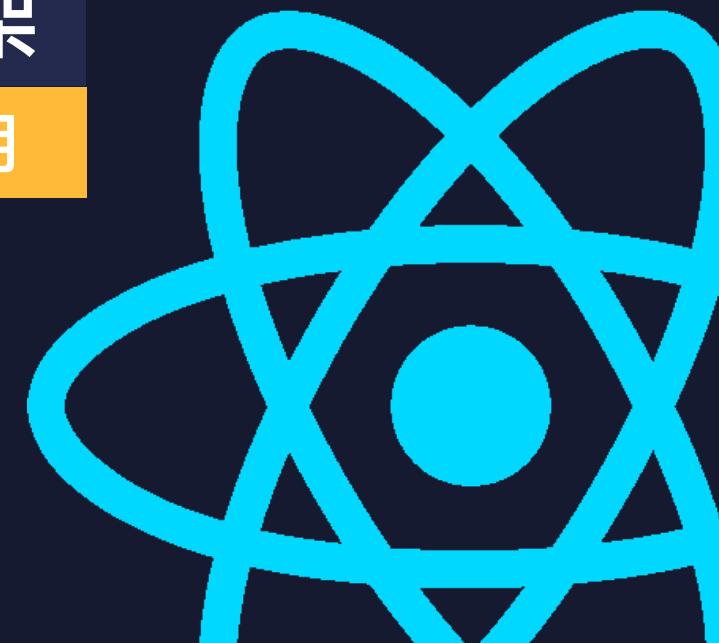
React 前端工程師必學新世代框架

樣式 - 資料載入或更新



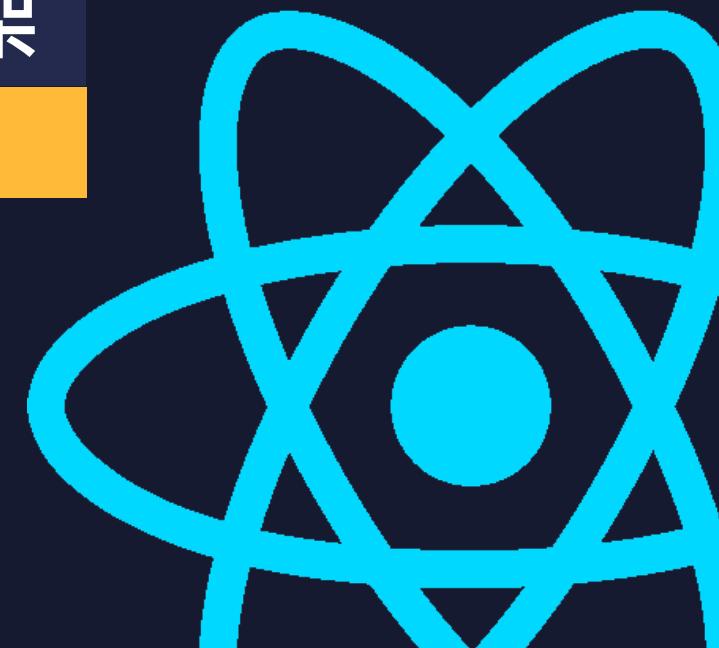
React 前端工程師必學新世代框架

樣式 - 整合其它函式庫應用



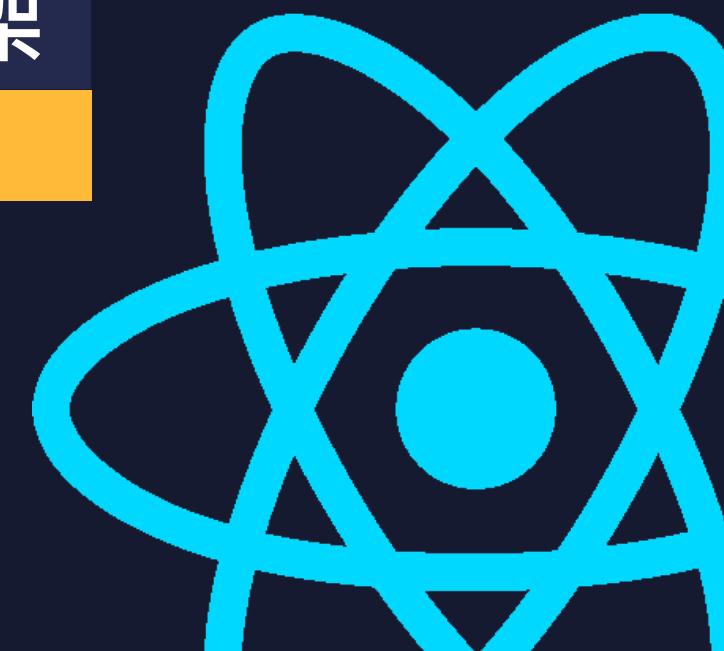
React 前端工程師必學新世代框架

會員註冊、登出入應用



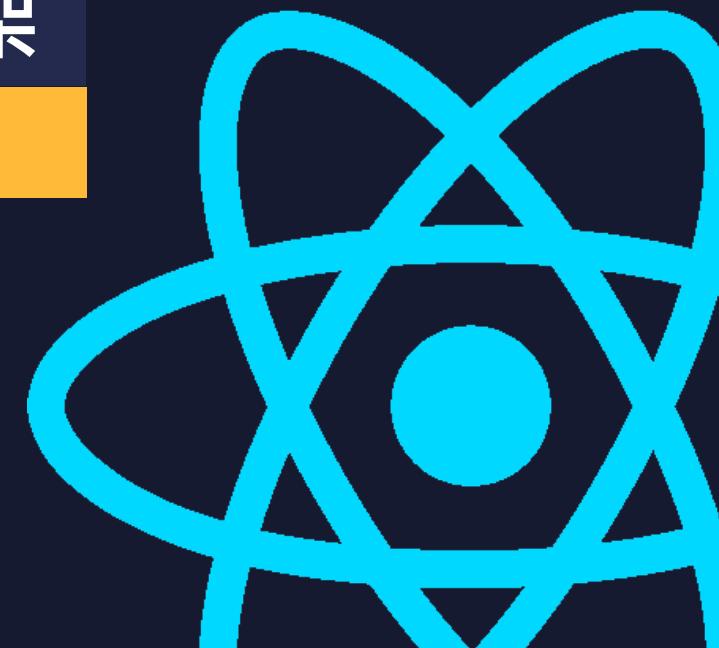
React 前端工程師必學新世代框架

待辦事項(todo)應用



React 前端工程師必學新世代框架

useReducer 勾子



全域狀態方案

| | useState | useReducer + useContext | Redux |
|---------|-----------------------------|--|--|
| 適合專案 | 小型專案/一般應用 | 中型專案 | 中大型專案/特殊應用 |
| 建議情況 | 元件數量 < 20 DOM層級 < 5 | 元件數量 20~50 DOM層級 5~10 | 元件數量 50+ DOM層級 10+ |
| 實作/學習難度 | 容易 | 中階 | 困難 |
| 適合狀態情況 | 一般值(數字、字串、布林)居多 | 有許多物件/陣列值 更動狀態改為發送 action 到 reducer | 物件/陣列值且複雜 更動狀態要改為發送 action 到 reducer |
| 實際內部設計 | 預先設定好的 <i>useReducer</i> 版本 | <i>Redux</i> 的陽春版 | 針對 <i>React</i> 全域狀態流行的解決方案 |
| 向下傳遞狀態 | 利用 <i>props</i> | 利用 <i>Context</i> | 利用 <i>Context</i> (綁定器) |
| 優點 | 大部份情況適用，學習與開發容易快速 | 可精確操控狀態轉變邏輯，解決大部份複雜狀態更新與最佳化問題 | 可以擴充中介軟體、獨立處理副作用，專案程式碼邏輯區分清楚，容易進行除錯與作最佳化 |

useReducer

為了管理更複雜的狀態(State)值與轉變運作

- > **useState** 是它的已設定好的版本，所以它可以取代原本的useState作法，但整個程式碼的邏輯運作方式需要改寫
- > 當需要複雜的 state 邏輯，要更精確的操控狀態轉變，其中包括多個子數值，或下一個 state 依賴前一個 state (通常是一個物件或陣列)
- > useReducer傳入參數：**(state, action) => newState**
(reducer，歸納/累加函式)
- > **useReducer回傳為 [state, dispatch]**
(state為狀態值，dispatch是用來發送動作物件的函式)

useReducer的設計概念都是從Redux來，可以說它是
陽春版的Redux功能，或是擷取了一部份的設計

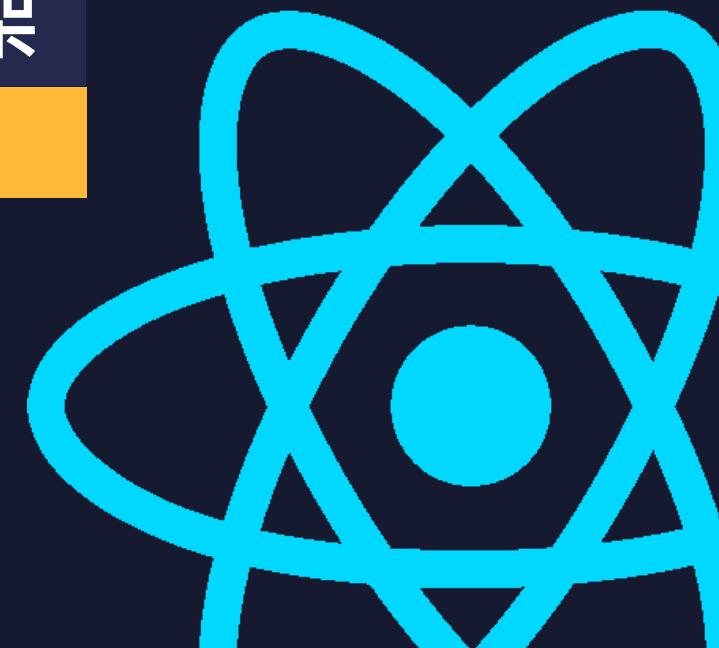
useReducer

從 useState 到 useReducer 改變

- > 撰寫 **reducer**，它是一個描述 **state** 將會因 **action(動作)** 如何改變的函式(純粹函式)：**(state, action) => newState**
- > 當要改變狀態時，從原本的用**setState**的方法，變為發送動作物件
setXXX -> dispatch(action)
- > **action(動作)** 是一個純物件值，內容有類型(**type**)和資料值(通常稱為
payload，有效資料)

React 前端工程師必學新世代框架

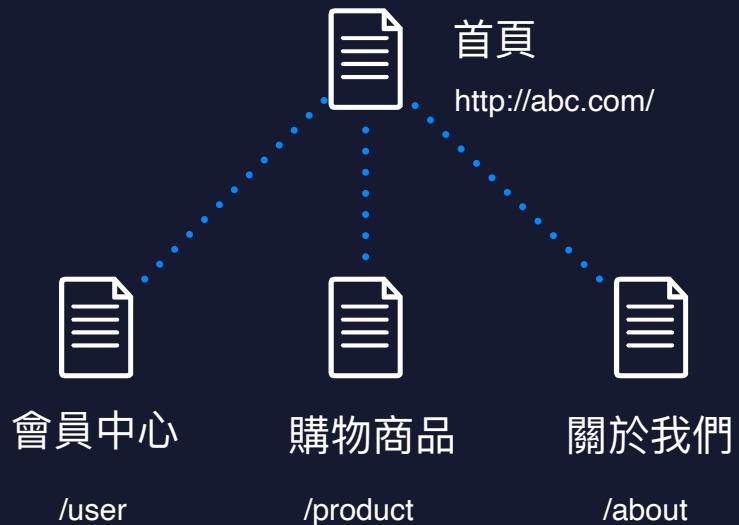
React Router路由器



React Router

網站頁面元件的組織與路徑

- > React是SPA(單頁應用)，React Router可以模擬出網站的網址路徑
- > Router可以擴充元件的屬性，提供瀏覽器歷史和各種操作
- > Router元件使用了Context(上下文)的特性，可以穿透整個元件DOM階層
- > 整體專案需要因Router的導入使用而更動



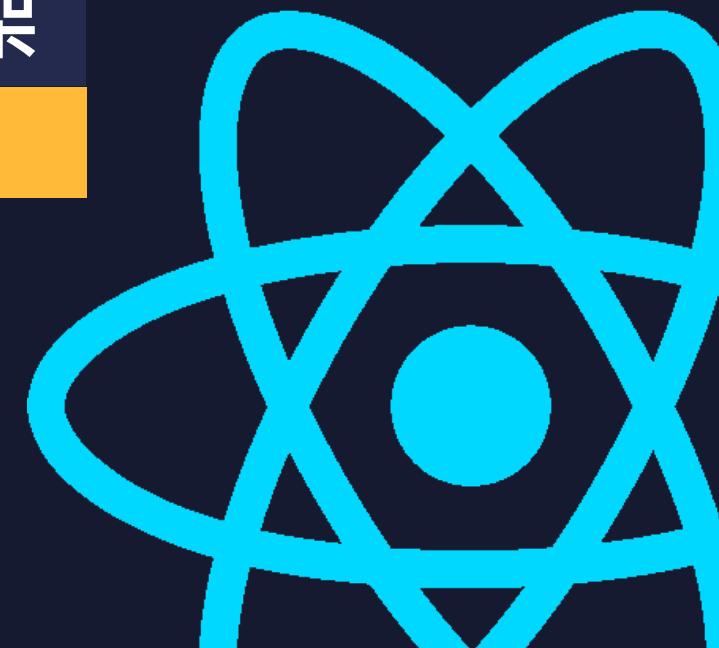
React Router三個擴充屬性

透過擴充元件本身的props，操控路由使用

- > 使用 **withRouter** 方法或勾子來擴充在Router階層下的子女元件
- > history - push、goBack方法
- > match - params屬性、url、path屬性
- > location - pathname、search屬性

React 前端工程師必學新世代框架

Redux 全域狀態容器



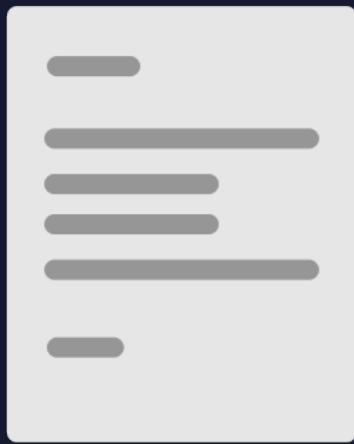
Redux學習步驟

大型專案的最普及解決方案

- > 決定應用程式state(與initialState) -> 決定 action -> 撰寫 reducer -> 建立 store
- > 當要更動狀態時 **dispatch(action)**
ps. action常用action creator傳入值來組成
- > 學習 **immer**
方便處理immutable state的工具套件。Redux Toolkit中會使用它。
- > 學習 **TypeScript**
為了處理大型專案與多人協同，必需要使用。API用TS撰寫。

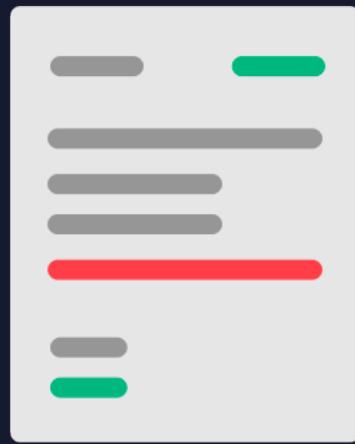
Immer

Current



immer
➤

Draft



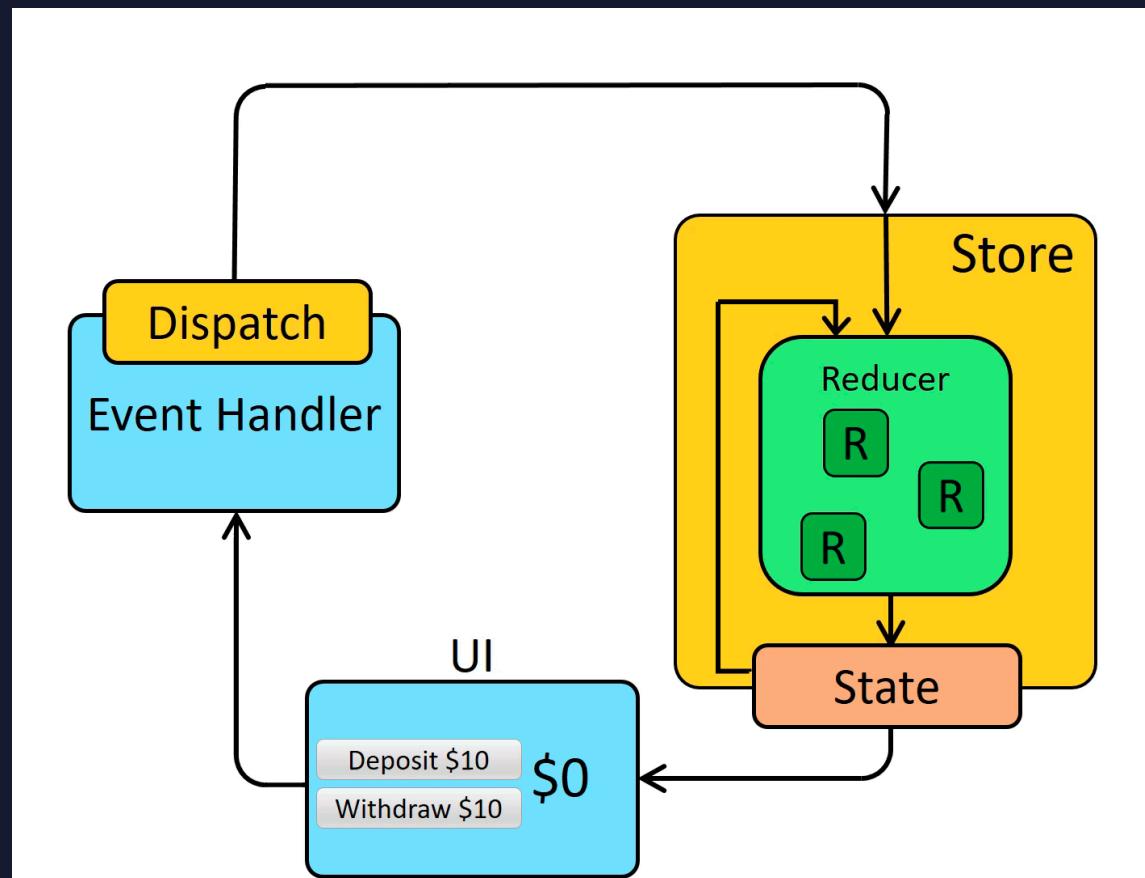
immer
➤

Next



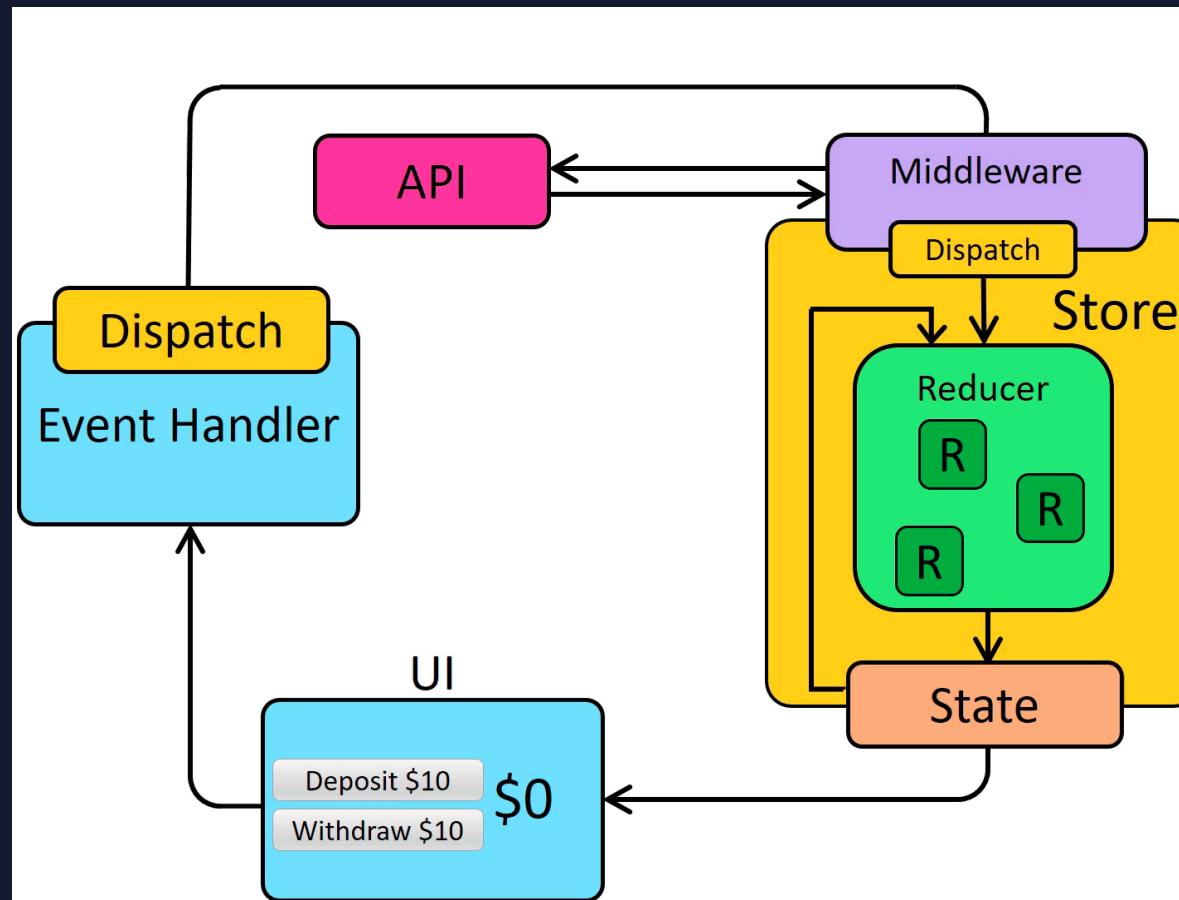
Your edits here.

Redux



<https://redux.js.org/tutorials/essentials/part-1-overview-concepts>

Redux



<https://redux.js.org/tutorials/essentials/part-5-async-logic>

Redux Toolkit

官方出品的工具函式庫

- > 決定應用程式**state(與initialState)** 與 決定 **action**
action直接使用action creator寫法，action type會自動產生
- > 撰寫 **reducer**
用immer語法，state即draftState，直接更動狀態
- > 建立 **store + Provider(from react-redux)**
用 configureStore方法與reducer來建立store，Provider位於最上層
- > 使用**react-redux綁定器(useSelector, useDispatch)**
得到值useSelector，更動狀態用 **dispatch(actionCreator(payload))**

Redux Toolkit API

官方出品的工具函式庫

> **configureStore**

傳入建立store，套用middleware, devTools, preloaded state

> **createReducer**

可用immer語法，有”builder callback”與”map object”兩種方式

> **createAction**

定義action type與creator，用”prepare callback”先結構化payload

> **createSlice**

標準的撰寫Redux處理邏輯的方法

> **createAsyncThunk**

建議的處理異步的要求使用的方法(使用promise語法與三態類型)