

JSX 語法

Eddy Chang


✉ hello@eddychang.me

```
    }).then(response => {  
      setProgress('finished');  
    });  
  
    return (  
      <div className={`progress-button ${progress}<div>  
        <span className="loading-text">Loading</span>  
        <button className="download-button">Download</button>  
        <span className="button-text">Download</span>  
      </div>  
    )  
  )  
)  
  
default App;
```

什麼是 JSX (What)

- 💬 JSX 是 JS 的一種擴充語法，類似於在 JS 中嵌入 HTML，由 Meta(Facebook)所創造，JSX 的 X 有 XML 或 Extension(擴充)之意
- 💬 因為它"不是合法的"符合 ES 標準的程式碼，需要先透過編譯工具(babel)來轉換為 JS 程式碼才能執行。
- 💬 React 元件總是使用 JSX 語法來撰寫。是 `createElement` 方法的簡寫法
- 💬 在其它新式框架中也可以使用它，例如 Vue, Solid, Qwik 等等

為什麼用 JSX (Why)

 讓開發者延續使用已熟悉的技能專長(HTML/CSS/JS)，大幅簡化建立 Virtual DOM(虛擬 DOM)的語法撰寫，尤其是針對複雜的巢狀元素結構

```
const element = (  
  <ul>  
    <li>item1</li>  
  </ul>  
)
```

```
// js (babel轉換後)  
const element = React.createElement(  
  'ul',  
  null,  
  React.createElement('li', null, 'item1')  
)
```

指定值(或 return 值)只能有一個根元素回傳

 重要必記

 使用 `<>...</>`、`<div>...</div>` 或其它元素在外包裹住元素即可解決。JSX 語法需要使用 `createElement` 方法轉譯/編譯。

✗ 錯誤

```
return (<h1>Total</h1>
      <p>Total</p>)
```

✓ 正確

```
return (
  <>
    <h1>Total</h1>
    <p>Total</p>
  </>
)
```

Fragment 元件(`<></>`) 不會產生多餘的渲染標記

💡 建議儘可能使用內建的 Fragment(片段) 元件。因為當層級多、複雜且分散時，如果有過多層 `div` 或其它多餘標記，可能有樣式套用或小幅度效能影響問題。`<>` `</>` 是 `<Fragment></Fragment>` 的簡寫法，如果要加 `key` 值時要後面這種才行。

! 不建議

```
<div>
  <h1>Total</h1>
  <p>Total</p>
</div>
```

✓ 建議

```
<>
  <h1>Total</h1>
  <p>Total</p>
</>
```

自我封閉的元素標記，一定要有結尾封閉標籤(`</>`)

 重要必記

 不加上會產生錯誤。因為 JSX 使用的是 `XHTML` 標準，並非 `HTML5` 標準。

```


<input type="text" name="myName" />
<br />
<hr />
```

以下只用在伺服器端(Next)

```
<meta charSet="utf-8" />
<link rel="shortcut icon" href="favicon.ico" />
```

自訂元件必需要是"英文大寫開頭"

 重要必記

 強制規則。自訂元件都應使用 CamelCase(大駝峰)命名。小寫開頭元件都是對應原本 HTML 的標記。

```
{ /* 自訂元件 */ }  
<MyComponent />  
<AddToCartButton color="red" />  
  
{ /* 內建HTML元素 */ }  
<h1>Text</h1>  
<p>Some text</p>
```

使用花括號({})在 JSX 中嵌入 JS 變數、值或表達式

💡 值可以是各種值，物件/陣列/函式/基礎值(字串/數字/布林值)/空(null/undefined)

```
<>
  <h1
    onClick={() => {
      setTotal(total + 1)
    }}
  >
    {total}
  </h1>
  <a href={`http://${url}`}>Text</a>
  {[1, 2, 3]}
</>
```


布林(`true` / `false`), `null`, `undefined` 嵌入在 JSX 的表達式中，最終不會被渲染呈現

 重要必記

 僅僅是不渲染呈現，不代表無功能。常用於作各種控制性質的語法。

// 以下渲染後均呈現相同結果

```
<>
  <div />
  <div></div>
  <div>{false}</div>
  <div>{true}</div>
  <div>{null}</div>
  <div>{undefined}</div>
</>
```

註解要使用 `{/* XXX */}`

💡 註解方式為 VSCode 中，選定後按下 `Ctrl+/*` (macOS 為 `CMD+/*`)

```
<>
  {/* 這是一個註解 */}
  <p>Text</p>
<>
```

JS 關鍵字的屬性名稱，需要改名稱(class/for)

💡 常見於指定 CSS 樣式的 `class` -> `className` 與 label 標記的 `for` -> `htmlFor`。不改的話，主控台會有警告訊息，並不會影響程式運作。

```
<h1 className="title">Text</h1>
<label htmlFor="emailInput">Email</label>
```

style 屬性值必需為物件值

💡 `style` 值如果使用像原本 HTML 中的字串值，將會產生程式錯誤，所以**不能是字串值**。建議不要用內聯樣式的 `style` 屬性，改寫為 CSS 類別，用 `className` 屬性指定，或其它解決方案

```
<h1 style={{ color: 'red', fontSize: 100 }}>Text</h1>
```

元件的事件處理屬性 `onXXXX`，值必為一個函式

💡 注意事件屬性是 camelCase(小駝峰)命名。

```
<h1 onClick={() => setTotal(total + 1)}>0</h1>
```

使用有前後開頭與結尾的元件標記，需要用 `props.children` 屬性來獲得值

💡 注意使用時別搞混，它是元件的屬性之一。`children` 屬性值是一個難以確定的 (opaque) 資料結構，意即可能是值/表達式/子元件們，它有很多使用上的技巧，通常會是函式庫/框架進階開發者使用。

```
<>
  <MyComponent>{'123'}</MyComponent>
  <MyContainer>
    <MyFirstComponent />
    <MySecondComponent />
  </MyContainer>
</>
```

⚠️ 以函式作為子元件(FaaC 或 FaCC)是另一種進階的語法

嵌入陣列搭配 `map` 方法，其中回傳項目必加 `key` 屬性值

💡 陣列資料會使用陣列的 `map` 方法，搭配例如 ``、`<td>` 等列表標記，作展開和調整每個項目的渲染應用。JSX 對於陣列值會直接組合為字串(類似 `array.join('')`)來渲染。

```
<ul>
  {todos.map((value, index) => {
    return <li key={value.id}>{value.text}</li>
  })}
</ul>
```

✎ `key` 是"必要的" React 需要 `key` 值，在進行列表項目處理(刪除/修改/新增...)時的最佳化使用，它與效能最佳化、程式運作有極大關聯，不可以省略。

if 流程控制的表達式語法 - 使用邏輯與 &&

💡 `{判斷條件 && <A />}` 相當於 `if 判斷條件 滿足即渲染 <A />`。注意語法中判斷條件因會使用 `falsey` 判斷，建議要讓判斷條件直接運算出布林值(使用強制轉型 `(!!)` 或比較運算子 `!==`, `>` ...))

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


if...else 流程控制的表達式語法 - 使用三元表達式 ? :



{判斷條件 ? <A /> : } 相當於 if 判斷條件 滿足即渲染 <A /> 否則渲染 。

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

直接使用 HTML 原始字串在屬性值中

⚠ 不建議使用，它有可能會導致安全性漏洞(XSS)。更多參考[官網資料](#)

```
const markup = { __html: '<p>some raw html</p>' }  
return <div dangerouslySetInnerHTML={markup} />
```

參考資料 & 相關工具

reactjs.org: [介紹 JSX](#)

react.dev: [JSX \(中\)](#), [JSX \(英\)](#)

react.dev: [條件渲染\(Conditional Rendering\)\(中\)](#)

[HTML to JSX 線上轉換工具](#)