章節名稱：JSX語法

前言

本章主要介紹React中最重要的JSX語法，JSX是一種"在JavaScript的程式碼中直接寫入HTML標記的語法"，當然，這個語法並不是符合ECMAScript標準的語法，這只是一種方便開發者在撰寫程式碼階段時使用的一種語法，需要先透過"預先處理工具"或"轉換編譯工具"處理過，最終轉換為合法的、可執行的程式碼，一般都是使用babel這個工具來協助轉換。JSX語法是一種簡化的語法，主要是讓開發者在撰寫程式碼時可以得到很大的便利性。

在讀過本章節的內容後，你可以了解以下知識：

- 從小範例中理解JSX語法的寫法
- 了解JSX語法與React元素的關係
- 了解JSX語法的基本學習指引
- 了解JSX語法花括號中的表達式
- 了解JSX語法花括號使用的情況

一個小的事件處理範例

JavaScript應用的開發經常會用事件驅動(Event-driven)程式設計來說明整體的開發流程，事件(event)所代表的是有某件事被觸發的一種信號，幾乎所有的DOM元素都會產生這種信號。在網頁上的許多應用，都是以使用者操作介面作為主要的工作觸發來源，常見的像是使用者用滑鼠點按了一個操作介面，例如一個按鈕，然後觸發執行一個函式中的程式碼。這個章節我們先用最簡單例子來看原生的JavaScript怎麼作這件事情，以及很常被使用的jQuery函式庫怎麼作，最後在我們現在要學習的React中，會怎麼處理這樣的設計方式。

首先是一個最簡單不過的按鈕，點下後會跳出一個警告視窗的語法會像下面這個例子:

```
<button onclick="alert('hello!');">Say Hello</button>
```

不過，上面這一段並不是純粹的JavaScript程式碼，而比較像是一段HTML原始碼。其中這個按鈕(button)的 `onclick` 值，代表的是這個按鈕元素的"屬性(attribute)"，然後在後面給定它的值是一段JavaScript程式碼， `alert('hello!');` 是呼叫瀏覽器內建的警告視窗方法的一個語句。這裡用的語法，通常我們在指定DOM元素的屬性值時，都是使用雙引號("")，所以 `alert` 中的字串值就直接用單引號會比較方便。另外，由於HTML碼中的標記或屬性名稱並沒有分大小寫， `onclick` 屬性的名稱，如果寫成"onClick"或"ONCLICK"一樣都可以正常運作，通常我們都是以全小寫為準，這只是個約定成俗的習慣，而且它會是個好習慣。不過，JavaScript中的方法和識別名都是有區分大小寫的，這一點是要注意的。

上面這種方式可以說是最早期的DOM事件模型，也是最簡單的一種，它被稱為內聯模型(Inline model)。這種方式直接在HTML裡DOM元素中標記中使用，每個DOM元素都會實作對應的可使用事件。當真正在執行時，瀏覽器的JavaScript引擎中會產生一個對應的匿名函式，包含在 `onclick` 中的語句，所以 `onclick` 給定它的程式碼，實際上是個回調函式(callback)。現在這個古早的事件處理方式，是一個完全不建議使用的方式，它也不像個是用JavaScript寫的應用程式，它需要直接寫在HTML中，完全沒有彈性可言，所以不論是要修改或維護都很不容易。

因此，我們應該要把HTML碼和JavaScript程式碼分離會比較好，在開發上程式的功能部份歸程式端的，HTML的部份著重於介面和視覺的設計就好，後來的事件模型，稱為"傳統模型(Traditional model)"方式，提供了分離HTML與JavaScript程式碼的語法，它比之前內聯模型的方式好得多了。

```
<button id="myButton">Say Hello</button>
<script>
  myButton.onclick = function() {
    alert('Thank you')
  }
</script>
```

我上面這樣直接用按鈕元素的id屬性值名稱，當作JavaScript程式碼中的識別名稱，沒經過宣告或其它的定義就直接用了，這個寫法問題多多，第一個是如果這個元素id值不存在時，這個網頁的程式就會有錯誤訊息，也會中斷執行其它的程式碼。第二個原因是，當id值是像"my-button"這樣的字串時，這個 `my-button.onclick` 是一個錯誤的語法，因為"my-button"並不是一個JavaScript中合法的識別名稱。

所以經過修改過，比上面範例較為正確的，而且有錯誤時可進行處理的程式碼是像下面這樣的程式碼範例:

```
<button id="myButton">Say Hello</button>
<script>
  document.getElementById('myButton').onclick = function() {
    alert('Thank you')
  }
</script>
```

`document.getElementById` 方法如果沒有找到有這個id屬性值的元素會回傳 `null` ，所以可以先作判別的處理，"onclick"對應的是這個按鈕元素的"onclick"屬性。但這語法仍然不夠好，它最大的問題是，就是它只能在一個元素上的某種事件上使用單一個事件函式，相同事件的回調函式會被新的覆蓋掉，如果應用中需要因為不同的情況，更動事件的處理函式工作內容，就會很不方便。

近期常被使用的DOM事件模型版本，這個方式又被稱為W3C方式(W3C Way)模型，這個方式它是用"事件監聽(Event Listen)"的方式，使用回調函式(callback)，作為事件的監聽者(或稱為事件處理函式)，這種方式多了一些彈性，而且也加入事件的傳播方式，這也是目前網站應用較為通用的方式，一個同功能的範例程式碼如下:

```
<button id="myButton">Say Hello</button>
<script>
  document.getElementById('myButton').addEventListener('click', function(){
       alert('Thank you')
  }, false)
</script>
```

事件監聽的方式可以對一個元件附加多個事件處理函式，而且以標準來說基本有定義三種方法可使用:

- addEventListener: 在事件對象上加入事件監聽者
- removeEventListener: 從事件對象移除事件監聽者
- dispatchEvent: 送出事件給所有有訂閱的監聽者

> 註: 微軟IE瀏覽器在舊版本中使用自己定義的事件處理方法，所以要與舊版本相容時需要特別注意。IE9之後就能使用上述的事件監聽方式。

jQuery函式庫的作法

jQuery函式庫更延伸了這種事件監聽的應用，可以使用更佳簡化的語法來寫這個事件處理的應用，不過事件處理只是jQuery函式庫的小部份應用而已，這jQuery中包含了更多好用的方法與功能。使用jQuery修改過的程式碼如下:

```
<script>
  $(document).ready(function() {
    $('#myButton').on('click', function(){
         alert('Thank you')
    })
  })
</script>
<button id="myButton">Say Hello</button>
```

因為jQuery有 `$(document).ready` 方法可以偵測是不是網頁上的DOM元素都已經載入到瀏覽器中，這樣才能把事件處理函式附掛上去，所以我們之前的程式碼都是寫在DOM元素之後， `$(document).ready` 方法如果要用JavaScript原生程式碼改寫，最簡單的寫法如下，這只能針對IE9以上的瀏覽器。不過jQuery原始碼中是用了幾十行的程式碼，大部份都是為了判斷不同瀏覽器品牌與版本的相容性:

```
document.addEventListener("DOMContentLoaded", function(event) {
  //在這裡加上事件處理的程式碼
});
```

jQuery函式庫的使用上可以大幅的簡化程式碼的撰寫，也提供許多簡便的方法，但在整體的事件處理語法結構上，與JavaScript原本的事件處理模型是一樣的，並沒有在基礎上作更動。

React的作法

如果以同樣的功能範例，React中又會是怎麼撰寫程式碼的？我們接著來看。請先打開之前的App.js檔案，改成像下面的程式碼內容，然後像之前章節所說的執行方式，觀看執行的結果是不是和前面所說的JavaScript一樣的執行結果:

```
import React from 'react'

class App extends React.Component {
  render() {
    return (
      <div>
        <button onClick={() => { alert('Thank you') }}>Say Hello</button>
      </div>
    )
  }
}

export default App
```

你應該有看到這個按鈕元素，是寫在JavaScript的程式碼裡面，我們並沒有修改到index.html檔案中的內容。而且這種事件處理的方式，看起來非常像我們在本節一開始所說的"內聯模型"，你可以比對一下內聯模型和現在這個React的程式碼，內聯模型是用"onclick"屬性，然後屬性值直接就是事件觸發時要執行的程式碼內容了，雖然實際上也是個回調函式，但是是由瀏覽器自動產生。而React中是用"onClick"屬性值，指定的值是用花括號({})括起來的一個箭頭函式，是一個簡寫的函式語法。

當然，因為屬性值是一個函式，如果說事件處理函式的內容比較多的時候，我們可以獨立出一個類別裡的方法來寫它，像下面這樣的修改過App類別的程式碼：

```
class App extends React.Component {

  handleButtonClick() {
    alert('Thank you')
  }

  render() {
    return (
      <div>
        <button onClick={this.handleButtonClick}>Say Hello</button>
      </div>
    )
  }
}
```

在這裡你可能會產生一個疑問，React中的元素事件處理模型，是內聯模型嗎？答案並不是，而是事件監聽的模型，只是這將會由React協助監聽或移除監聽的工作，開發者不需要負責作元素的事件監聽工作，在React中的這些元素事件，例如這裡button元素的onClick，都是React中人造的、虛假的事件，它會對應到真實的DOM事件上。

在React中也並沒有使用id這個DOM元素的屬性。使用元素id屬性的那種開發方式，也就是像jQuery這種函式庫長期以來使用的，稱為DOM查詢或挑選的一種開發方式，這在React中並沒有使用，這一點也是要注意的。當然有很多讀者或React初學者，都已經很習慣這種先找到元素的id屬性，然後得到元素參照後，再接著要看要處理什麼的開發流程，所以剛開始接觸像React這樣的開發方式，是不太習慣的。在前面的介紹章節時，有提過React這種開發的方式是很具有創新的思維的，它可能與目前的主流開發方式有些滿大的差異。React這樣的設計也是有它的目的，它會希望開發者是專注在開發一個一個的可重覆使用的元件，利用現成的DOM元素和事件來組合出應用程式，在之後的一些範例中，讀者可以慢慢體會這種作法，至於哪一種方式比較好，這個問題的答案就見人見智了。

JSX語法與React元素(Element)

到這個章節之前，我們目前大概的理解是，JSX語法是一種可以讓HTML標記直接寫在JavaScript程式碼中的擴充語法， `X` 代表的是 `XML` 語法的意思，React允許你可以這樣作，是因為JSX語法相當於使用 `React.createElement` 方法來建立虛擬的DOM元素，也就是說JSX語法實際上就是使用 `React.createElement` 方法的簡寫語法。再次要提醒的是，JSX語法並不是一種合法的ECMAScript的語法，而是需要在執行前，先透過編輯工具轉換的語法，之前有提過babel是Facebook贊助的專案，自然會跟著React要作什麼運用而加這些功能。那麼，它會轉換成什麼語法？如果以之前我們最簡單的第一支React程式來說，也就是像下面這段的程式碼：

```
class App extends React.Component {
  render() {
    return (
```

```
      <div>
        <h1>今天就開始學React!</h1>
      </div>
    )
  }
}
```

其中的render方法中的return值，經過babel工具轉換成合法的JavaScript語法如下：

```
"use strict";

React.createElement(
        "div",
        null,
        React.createElement(
                "h1",
                null,
                "\u4ECA\u5929\u5C31\u958B\u59CB\u5B78React!"
        )
);
```

所以這些在React中所謂的仿造真實DOM元素的標記語法，它最後會被轉換為 `React.createElement` 方法，並以傳入參數值的形式來代表不同的DOM元素標記。現在你可能會產生一個新的問題，為什麼要用JSX語法，而不直接用 `React.createElement` 這方法的方式來寫就好了？這想法也是可行的，只不過不太容易寫就是。在React一問世時，有很多開發者覺得JSX語法相當怪異，也提出排斥或質疑的意見。為什麼一定要用JSX語法的原因，其實只有一個，就是"方便"開發者撰寫而已，下面接著說明這理由。

`React.createElement` 方法是一個在React中用於建立元素的方法，看它的名稱就知道在作什麼工作的。從上面那個簡單的例子中，你已經看到它有巢狀的結構出現了，也就是在傳入參數中，還有另一個 `React.createElement` 方法被呼叫。那麼，如果在很複雜的DOM元素結構時，例如下面的這個JSX範例程式碼，如果用 `React.createElement` 方法該怎麼寫呢：

```
class App extends React.Component {
  render() {
    return (
      <div>
        <ul>
          <li><a href="http://google.com">Google</a></li>
          <li><a href="http://apple.com">Apple</a></li>
          <li><a href="http://amazon.com">Amazon</a></li>
        </ul>
      </div>
    )
  }
}
```

這個範例要用 `React.createElement` 方法改寫，就算是很熟練的開發者也很容易會出錯，有興趣的讀者可以到babel網站的線上轉換試試，或是用babel工具實際編譯看看。最終的結果是，每個最裡面那層的 `a` 元素，它是第4層的 `React.createElement` 方法，也就是作為第一層的傳入參數後，這樣連續到第4層。這範例程式碼並不是那種超級複雜的DOM元素結構，就已經很難撰寫得出來了對應的 `React.createElement` 語法了，如果在更多DOM元素時，根本是不可能直接寫得出來。所以，學習JSX是必要的，這雖然除了方便之外，之後我們要開發的應用中的DOM元素結構，會遠比目前看到的範例複雜得多。

React Element(元素)

React的核心概念中有兩個，一個是React Element(元素)，另一個是React Component(元件)，兩者都是虛擬DOM中的東西，React Element(元素)則是其中最基本的概念。

React Element(元素)是一個React用於描述虛擬DOM元素的物件，它只有單純用於描述的屬性值，其中沒有帶有方法，在原型(prototype)中也沒有其他東西，它會用四個重要的屬性來描述DOM元素:

- type: 一個字串，代表任何合法的HTML元素類型名稱，例如h1、div，或是參照到React程式碼中定義的的元件類別。
- props: 對應到元素的屬性值的屬性。
- key: React用來識別元素的屬性，尤其是在同樣類型的元素間要用這個屬性來區分。
- ref: React用來存取對應的實體(真實)DOM用的屬性。

React Element(元素)需要透過 `ReactDOM.render` 方法，才能把虛擬DOM元素，轉換為實體(真實)DOM，也就是說React Element(元素)代表的是一種無狀態的、不可改變的、虛擬的DOM元素，它就是所謂的"虛擬DOM(Virtual DOM)"的組成分子。React元素與元件間是一種"你之中有我，我之中有你"的結構。

React提供了 `React.createElement` 方法，讓開發者可以自行定義React Element(元素)，也可以組合元素們間的 父母-子女(parent-children)關係。下面的程式碼範例中，使用了 `React.createElement` 方法來建立虛擬的DOM元素:

```
const child = React.createElement('li', null, '項目')
const root = React.createElement('ul', { className: 'my-list' }, child)

ReactDOM.render(root, document.getElementById('root'))
```

最後在網頁上的真實DOM元素的結構會像下面這樣:

```
<ul data-reactroot="" class="my-list">
  <li>項目</li>
</ul>
```

這個React Element(元素)，也就是用於描述的虛擬DOM元素的JavaScript物件會長這個樣子，實際上是可以有巢狀結構的:

```
{
  type: 'ul',
  props: {
    className: 'my-list',
    children: {
      type: 'li',
      props: {
        children: '項目'
      }
    }
  }
}
```

雖然React並沒有強迫你一定要用JSX語法來定義虛擬的DOM元素的結構與內容，但說實在的，如果DOM元素複雜了些，加上又在其中要混用JavaScript語句，使用 `React.createElement` 會讓程式碼顯得混亂，所以JSX語法是一定要用的。上面的範例寫成JSX語法會非常的直覺，就像在寫一般的網頁HTML碼一樣:

```
const root =  <ul className='my-list'><li>項目</li></ul>
```

JSX語法基礎學習指引

JSX語法看起來很簡單，就像你在寫一般的網頁HTML碼一樣，實際上並不是完全一樣的東西，JSX語法有很多細節，而且新的JSX規格還在制定中，我們在本節中會提供一些學習的重要指引，以免你在之後的範例時會有摸不著頭緒的感覺。

JSX語法最終會被編譯為React.createElement方法

這上面的章節有說明了，所以這會產生兩個要注意的地方，這兩個重點也是討論區上的常客，有很多初學者會問的問題。我把它列在下面:

> 有用到JSX語法的程式碼檔案，一定要導入(import)react模組。

有很多初學者使用函式型的元件寫法，認為沒用到React的任何功能，所以就沒導入(import)react模組，造成錯誤。

> JSX語法中每個render方法回傳區段只能有一個根元素。

這是JSX語法使用的基本原則， `render` 方法(或是函式型元件的回傳值)如果有超過一個以上的根元素，將會造成無法編譯為 `React.createElement` 語句，所以會產生錯誤。例如以下的程式碼範例，這是一個錯誤的示範:

```
// 錯誤示範!!
render() {
```

```
  return (
    Some text.
    <div>Test</div>
    <div>Test 2</div>
  )
}
```

通常在外圍多加一個 標記就可以解決這個問題，例如以下的程式碼範例:

```
// 正確示範!!
render() {
  return (
    <div>
      Some text.
      <div>Test</div>
      <div>Test 2</div>
    </div>
  )
}
```

最後會由babel工具編譯的結果會像下面這樣的程式碼，你可以看到會呈現有巢狀的結構，例如以下的程式碼範例:

```
React.createElement(
  'div',
  null,
  React.createElement(
    'div',
    null,
    'Test'
  ),
  React.createElement(
    'div',
    null,
    'Test 2'
  )
)
```

不過，因為你在這個元件的外層又包裹了一層div或span之類的標記，如果在很多層的樹狀結構中，這會產生不少多餘的標記，所以在React 16版本中，又加入了新的"片段(fragments)"的語法，在下面有這個部份的說明。

多個的列表項目標記，需要加入key屬性

React對於列表項目的元素，例如常見的 <li> ，需要開發者提供唯一的key屬性值，key屬性值可以協助React在進行更動、新增或移除的工作時，更能快速、精準地分辨出每個獨立的項目元素，以此能提高工作效率。一般最常見的用法，是使用資料中的id值作為key屬性值，如果無法提供較為穩定的、唯一的id值，因為多筆的資料通常是以陣列資料類型呈現，則可以使用陣列的索引值，例如以下的程式碼範例:

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

key屬性值的指定通常會包含在迴圈，或是map等陣列迭代的方法中。所以，如果是一個在開發者自訂的類似項目元素的元件時，應該會用在該元件要進行呈現的語句時，而不是在該元件內部的項目元素(如 <li>)中，例如以下的程式碼範例:

```
const listItems = numbers.map((number) =>
  <ListItem key={number.id} value={number} />
)

return (
  <ul>
    {listItems}
  </ul>
)
```

React 16中片段(fragments)與字串的支援

直接回傳字串現在是可以的，例如以下的程式碼範例:

```
render() {
  return 'Some text.'
}
```

片段(fragments)則是多個包含在陣列中的元素項目，這個語法可以改進了上述的只能在render方法的回傳值中，只能使用單一個根元素的限制，當然新語法的出現是有意義的，主要為了提供更多的在開發的靈活彈性運用，以及去除掉多餘的最外層元素標記(div或span等等)。上述的範例改用片段之後，可以將每個元素語句加入到陣列中作為成員值，例如以下的程式碼範例:

```
render() {
  return [
    'Some text.',
    <div key="A">Test</div>,
    <div key="B">Test 2</div>
  ]
}
```

這語法看起來似乎和上述的在外層加上一個額外的div標記的方式有點類似，但實際上的運作有點不太相同，而且會變得麻煩些，主要是以下幾點差異:

- 字串需要加上引號才能放到陣列中
- 在陣列中，每個元素之間需要用逗號(,)分隔
- 在陣列中"所有的"元素標記都需要有key屬性值，不然會出現警告訊息

React中提供了一個名為Fragment的元素，用起來和上述的範例有點像，但實際上在渲染時並沒有最外面的div或span標記，用了這個元件後，上面的三個問題都不會再看到，所以也不用額外加上key屬性值，例如以下的程式碼範例:

```
const Fragment = React.Fragment
//...
render() {
  return (
    <Fragment>
      Some text.
      <div>Test</div>
      <div>Test 2</div>
    </Fragment>
  )
}
```

最近的16.2.0版本，這個語法又可以寫得更簡潔了，這是用 `<>...</>` 標記來作簡寫法，這個語法不需要特別引入 `React.Fragment` ，，例如以下的程式碼範例:

```
render() {
  return (
    <>
      Some text.
      <div>Test</div>
      <div>Test 2</div>
    </>
  )
}
```

不過這個語法以目前來說實在太新了，babel目前要7以上版本才能支援，其它的工具或週邊例如TypeScript、Flow、開發工具等等，都需要升級或使用測試版本才有支援，可能需要再等一段時間才能正式用在開發上。

JSX語法基本撰寫風格

幾個使用的基本風格指引，這也是一開始使用JSX時會有點不太習慣的地方，JSX雖然長得很像HTML語法，但用起來卻沒想像中那麼自由。

> 基本風格1: 自訂元件識別名稱都是以大寫英文開頭，小寫開頭的是原本HTML中有的元素

React會認為在JSX語法中，使用的標記如果是小寫開頭的，是HTML中原本就有的元素，例如div、span等等一類。所以開發者自訂的元件，應該要以大寫開頭，例如TodoItem、HelloWorld等等標記來作為自訂元件的識別名稱。

自訂的元件識別名稱對React來說，是使用React.createElement方法的第一個傳入參數，也就是類型(type)這個參數，React會依照是開頭大寫還小寫來分辨是自訂元件還是原本就有的HTML元素，類型(type)是可以動態指定的，有個小技巧是需要先轉為開頭大寫的識別名稱，這樣才能正確使用JSX語法，例如以下的程式碼範例:

```
const components = {
  facebook: FacebookButton,
  line: LineButton
}

function SocialShare(props) {
  // 需要轉換為開頭大寫英文的識別名稱
  const SocialShareButton = components[props.socialType]
  return <SocialShareButton id={props.id} />
}
```

> 基本風格2: 當使用一般的字串值當屬性值時，字串使用雙引號("")括住。等號(=)與屬性、等號(=)與值之間，不需要加空格。

這和HTML語法是一致的，但要注意JSX裡的屬性是有區分大小寫的，HTML則是大小寫相同，例如以下的程式碼範例:

```
<TodoItem text="buy book" index="1" />
```

> 基本風格3: 當使用花括號({})作為屬性值時，不需要加上雙引號("")。等號(=)與屬性、等號(=)與值之間，不需要加空格。

在花括號({})中相當於要寫JavaScript的程式碼，JSX會對花括號裡的表達式進行求值運算。所以花括號中的字串建議用單括號('')括住，以此與上一個風格作區別。但花括號({})並不是什麼程式碼都可以加進來，只有單一行的表達式可以而已，下面會再詳細說明，例如以下的程式碼範例:

```
<TodoItem text={'play game'} index="1" />
```

> 基本風格4: 夾在元件標記或HTML的DOM元素標記的JavaScript程式碼時，一樣也要使用花括號({})框住

這個在之前的例子中有看到，如果你要在元件標記或DOM元素中加入JavaScript程式碼，也是用花括號({})框住，例如以下的程式碼範例:

```
<ul>
{
  this.state.items.map((value, index) => {
    return <TodoItem key={index} text={value} />
  })
}
</ul>
```

> 基本風格5: 多個屬性值時儘量使用多行的風格樣式，太長會難以閱讀

程式碼每行語句的字元數量，可以控制在一個畫面就能看得清楚的情況，不要過長，例如以下的程式碼範例:

```
<input
  type="text"
  value={this.state.value}
  placeholder={this.props.initText}
  onChange={this.handleChange}
/>
```

JSX語法花括號中的表達式

在JSX花括號({})中的"JavaScript表達式"到底是可以放什麼？雖然React官方說是表達式，但很顯然的它並不是只有單純的JavaScript中的表達式而已。由於JSX語法是由babel工具來進行轉換為對應方法，那到底是什麼能轉換什麼不能轉換，其實就是babel工具來決定的，babel有提供線上的測試轉換工具，你可以試試看以下的範例是會轉換成什麼可執行的語法。以下列出常見的幾種情況，或是JSX語法在解析時有一些自動的機制。

第一種是就是某個真實的值，或是簡單的運算(例如數字的加減乘除)會求出值的表達式，至於物件類型的值很常用在定義內聯的CSS樣式(inline CSS styles)時使用，在解析後會自動套用到DOM元素的 `style` 屬性中，例如以下的程式碼範例:

```
value={123 + 456}

value={true}

<Hello name={{ firstname: 'John', lastname: 'Doe' }} />

<span key={index} style={{
    color: 'red',
    paddingRight: '10px'
}}>{"text"}</span>
```

> 註: 有時候會看到雙層花括號是因為在JSX中代入的值是一個物件值，不過在其他函式庫例如Angular這種符號有其他意義。

第二種是某個變數(或常數)，代表要從這個變數(或常數)得出值(注意函式或方法也是一種值)，例如以下的程式碼範例:

```
value={this.state.value}

defaultValue={this.props.initText}

onChange={this.handleChange}
```

第三種是陣列值，我會特別把它列出來自成一類，是因為陣列在解析時會直接被轉成字串值，很常用在輸出子元素時，例如以下的程式碼範例:

```
const arr = [
  <h1>Hello world!</h1>,
  <h2>React is awesome</h2>
]

ReactDOM.render(
  <div>{arr}</div>,
  document.getElementById('root')
)
```

> 特別注意: JSX中的陣列值在轉換為字串時，不會像JavaScript中轉字串時的一般轉換會自動加上逗號，而是直接連接每個成員。

第四種是一個函式的呼叫(執行)，通常是會回傳陣列值或某個值的函式。你可以看到函式定義本身可以作為值，也可以用函式呼叫放在這個花括號({})中，真是一物兩用，例如以下的程式碼範例:

```
{
  this.props.results.map((result) => (
      <ListItemWrapper data={result}/>
  ))
}
```

既然可以以函式定義作為值或是作函式呼叫(IIFE或IIAF也可以)，實際上要加什麼程式碼在花括號裡都是可以的，只要包含函式裡面就行了。現在有很多React的開發者，都是直接把事件處理的函式包在JSX的花括號中，這樣作的用意是把一些簡短的程式碼直接嵌入方便撰寫，例如以下的程式碼範例，這是出自Redux的官網文件:

```
const TodoList = ({ todos, onTodoClick }) => (
  <ul>
    {todos.map(todo =>
      <Todo
        key={todo.id}
        {...todo}
```

```
        onClick={() => onTodoClick(todo.id)}
      />
    )}
  </ul>
)
```

第五種是三元運算語句，它算是 `if...else` 語法的簡寫法。這也很常見，最後會由判斷求出一個值而且是單一行的語句，三元運算的值也可以是元件或HTML元素標記，例如以下的程式碼範例:

```
<Person name={window.isLoggedIn ? window.name : ''} />

<Container>{window.isLoggedIn ? <Nav /> : <Login />}</Container>
```

> 註: `if...else` 語句目前"不能"直接用在花括號之中，只有三元運算可以

第六種是註解，在花括號({})裡只能用 `/*...*/` 這種格式的註解，例如以下的程式碼範例:

```
{/* child comment, put {} around */}
```

第七種使用"展開運算符(...)"的語法，因為使用在物件上所以是一種ES7+語法，把props當作物件來看，然後直接展開裡面的值就是，例如以下的程式碼範例:

```
const props = {}
props.foo = x
props.bar = y
const component = <Component {...props} />
```

上面的展開語句相當於經過babel轉換過的下面語句，例如以下的程式碼範例:

```
//babel轉換
var component = React.createElement(Component, props);
```

不過如果再加上state(狀態)的展開，它就會變成合成(Composition)樣式的語句，例如以下的程式碼範例:

```
const component = <Component {...props} {...state}/>
```

JSX語法花括號使用的情況

JSX語法的花括號({})使用的情況將會有兩個地方，一個是在標記中的屬性指定值時使用，稱為"在JSX中的屬性值"，另一個是在開題標記與結束的標記之間，例如在像 `<div>...</div>` 這樣的標記之內，稱為"在JSX中的子元素"。這兩種情況下，運算求值結果都是類似的，但仍然有一些小差異，以下是一些說明。

> 註: 在HTML碼的標記中，屬性的英文是 `attributes` ，但在React或JSX語法中，屬性的英文稱為 `props` (properties的簡寫)，雖然中文都翻成"屬性"，使用的地方不太一樣。

在JSX中的屬性值

之前有說過可以在屬性(props)裡指定JavaScript的各種原始資料類型，而物件、函式、陣列等類型的值也是可以。但最終這些值在最下層的元件，也是會轉為HTML的DOM元件標記來作最終輸出的工作。

直接使用雙引號("")，只能對props指定一個字串值，其他的都不行，例如以下的程式碼範例:

```
<TodoItem text="a string" index="1" />
```

上面的例子相當於使用花括號({})的下面這個程式碼範例:

```
<TodoItem text={'a string'} index={'1'} />
```

使用花括號({})因為可以使用表達式，所以在進行屬性指定值時可以更具彈性，也可以進行運算，例如以下的程式碼範例:

```
<TodoItem text={'a string' + 'other string'} index={1} />
```

但需要特別注意的是，JSX語法中的花括號({})會作一些比你想像中還多的求值運算，它並不是標準的、單純的JavaScript表達式求值而已，JSX語法會有自己自動處理某些值的情況。

例如，在JSX中的布林值(在花括號裡值)，一律最後會被略過不輸出。而類似於布林值這樣的處理，還有 `null` 與 `undefined` 值。

> 特別注意: JSX中的布林、null、undefined將會被忽略

在props指定值的部份，如果你只寫出一個屬性名稱，但沒有指定給任何的值，JSX會預設指定給它 `true` (布林值)，這也是一個自動的設計。像下面這樣的JSX語法範例， `autoFocus` 這屬性是被指定為 `true` 布林值，例如以下的程式碼範例:

```
<input autoFocus type="text" />
```

所以相當於下面的程式碼範例:

```
<input autoFocus={true} type="text" />
```

> 特別注意: JSX中對屬性(props)指定值時，沒給定值的情況是給預設的布林值 `true`

在JSX中的花括號裡也沒辦法使用真實的HTML碼，所有的類似於HTML語法的字串，都會被自動作字串的跳脫(escape)，例如以下的程式碼範例:

```
// 使用JSX語法
ReactDOM.render(<div>{'<span>a string</span>'}</div>, document.getElementById('root'))

//HTML碼的字串會被跳脫為 &lt;span&gt;a string&lt;/span&gt;
```

使用展開運算符(...)在指定props的值上，是另一種更加簡化的語法，它是把已有的props物件值展開來指定這個元件中，例如以下的程式碼範例:

```
const props = {foo: 1, bar: 2}
const component = <Component {...props} />
```

對屬性(props)指定一個物件值，目前看到只有像 `style` 這個屬性時會用到，JSX會自動運算求值轉換為正確的HTML字串值，不過因為是使用JavaScript的物件字面定義來定義樣式，用的是JavaScript中的對應的屬性名稱，例如原本的CSS中的 `font-size` 要用 `fontSize` 的識別名來定義才可以，大部份的情況都是這樣改寫，只有很少幾個的例外，這是一種內聯樣式的定義語法，例如以下的程式碼範例:

```
const divStyle = {
  color: 'blue',
  fontSize: 16,
}

const TodoItem = (props) => <div style={divStyle}>{props.text}</div>

ReactDOM.render(<TodoItem text="Text" />, document.getElementById('root'))
```

最後輸出到真實網頁上的HTML碼，會自動把原本是物件值的divStyle整個轉成一個字串值，例如以下的程式碼範例:

```
<div data-reactroot="" style="color: blue; font-size: 16px;">Text</div>
```

陣列值也很特別，JSX是用 `join('')` 方法運算連接陣列成員。對比一般的JavaScript的陣列轉為字串的強制運算，則是用 `join()` ，成員之間會有逗號。

在JSX中的子元素

JSX語法用於子元素，主要是像下面這樣的語法，夾在開頭標記與結尾標記之間，這 `Hello world!` 明顯也是一個字串值，只不過用於屬性(props)指定值時，會需要加雙引號("")，這裡不需要用，例如以下的程式碼範例:

```
<MyComponent>Hello world!</MyComponent>
```

這也相當於以下的程式碼範例:

```
<MyComponent>{'Hello world!'}</MyComponent>
```

當然也可以混用有花括號({})的字串一起使用，尤其是當花括號({})裡有程式碼中的數值作連接時，例如以下的程式碼範例:

```
<MyComponent>Hello world! {prop.text}</MyComponent>
```

JSX語法會自動地對其中的字串作幾件事，這與HTML語法在使用時不一樣，首先它會忽略掉中間的 `Hello world!` 字串的前面或後面的空白、換行符號。其次，如果 `Hello` 與 `world!` 中間的有換行符號，則會自動變為一格空白，所以以下的程式碼範例最後都會得到和上面範例中一模一樣的結果，例如以下的程式碼範例:

```
<div>Hello World</div>

<div>
  Hello World
</div>

<div>
  Hello
  World
</div>

<div>

  Hello World
</div>
```

> 特別注意: 在寫JSX語法時，要儘可能把其中的字串值的撰寫格式寫得整齊些。除了方便閱讀外，也儘量不要造成格式的誤判情況。

JSX語法中，也可以形成巢狀的結構，類似於HTML中的DOM巢狀結構。這在之前的例子中也有看到過了，例如以下的程式碼範例:

```
<MyContainer>
  <MyFirstComponent />
  <MySecondComponent />
</MyContainer>
```

JSX語法特殊用法

本章列出了幾個特殊情況下的JSX語法的用法，這裡只是提供一些參考的資訊，詳細的內容請參考React官方網站，或是網路上的相關文章。

以函式作為子元素

最特殊的一種用法，是"以函式作為子元素(Functions as Children)"語法，在JSX中的花括號中的表達式，會進行求值運算，最後主要會求出字串值、React元素，或是上面這兩種的列表值(陣列值)。所以當子元素是個JSX表達式時，可以用 `props.children` 作為函式識別名，在上層元件中來使用，例如以下的程式碼範例:

```
const TodoItem = (props) => <div>{props.children('Eddy')}</div>
```

```
ReactDOM.render(
  <TodoItem>
    {(name) => <div>Hello! {name}</div>}
  </TodoItem>,
   document.getElementById('root'))
```

這種語法會用在一些函式庫中，會用在一些特殊的情況下，進階的說明可以再參考網路上的"Function as Child Components"文章。

JSX中的邏輯與(&&)語法

上面已經有說過，JSX會忽略掉所有在花括號({})中的布林值，不論是 `true` 或 `false` ，所以你可能會看到有使用邏輯與(&&)作為控制流程的語法，例如以下的出自官方網站的程式碼範例:

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {
        unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  )
}
```

邏輯與(&&)和之前我們在第一單元中，談到預設值的那一篇說過的邏輯或(||)運算，一樣是短路求值的方式，它的運算規則如下:

- 當 `true && expression(表達式)` 時，則得到expression(表達式)求出的值
- 當 `false && expression(表達式)` 時，則得到false值

拿來套用到上面的範例中的那一語句，當得到false值時，JSX就不輸出任何東西。只有當得到true值時，JSX會以表達式(expression)中得到的值來作輸出使用。所以，這也是一種運用JSX語法的特性，一種當作是流程控制的簡短寫法。

React 16的DOM屬性(Attributes)

React 16版本中新加入了"DOM屬性(Attributes)"的支援，也可以稱它為"自訂DOM屬性"，"自訂"的意思是"非標準"，其它標準的HTML5標記屬性在React中都已經有包含在內了。在15版本之前，在JSX的標記中如果使用自訂的屬性，會被直接省略，例如下面這種JSX語法的範例:

```
<div custom-attribute="some-value" />
```

在React 16中會保留這個自訂的屬性，但如果是在React 15版本中的轉換結果會自動移除掉，例如以下的程式碼範例:

```
<div />
```

這個改善主要是為了可以讓開發者們，更容易使用那些不是標準的、合法的HTML標記屬性，這通常會是在需要混用或整合某些其它的函式庫或框架、某些只有瀏覽器才支援的標記屬性，或是正在實驗中、測試中的標記屬性。最後說明的是，如果你要使用標準的HTML標記屬性，在React中是有一定的大小寫規定的，所以需要按照固定的大小寫名稱來使用，例如 `tabIndex` 就不能寫成 `tabindex` ，在React 16版本中一樣是會有警告訊息的。

結論

本章介紹了React中最重要的JSX語法，JSX基本上是一套仿照HTML的語法，再次強調它並不是一個合法的ECMAScript的語法，所以不能直接在JavaSript引擎上執行。JSX語法需要透過像babel工具，轉換為真正合法的ECMAScript程式碼，所以要學習JSX語法，了解什麼是可以轉換，又是轉換為什麼，最方便的方式就是利用babel工具實際轉換看看。當然，JSX的會被使用的目的，主要是方便開發者撰寫React中的自訂元件，初學者應該要先掌握各種基本的、常用的語法特性，以及一些較好的開發風格習慣，在撰寫React應用時才能更得心應手。另

一個值得注意的是，JSX新的版本正在制定當中，未來有可能也會加入更多新的語法，也有可能會加入一些還在制定中的ECMAScript新標準的語法。