前言

這本書將帶你進入 React Native 的世界，React Native 是 Facebook 公司用來開發手機應用的 JavaScript framework。使用你既有的 JavaScript 知識與 React，就可以在 iOS 及 Android 上建立及發布原生全功能的手機應用程式。捨棄傳統手機應用開發方法，改用 React Native 有諸多好處，又不用犧牲原生外觀以及使用感受。

我們會從基礎開始，然後一路向建立一個完整成熟的應用前進，而且程式碼 100% 相容於 iOS 和 Android。除了介紹這個 framework 精華之外，我們會討論一些進階的題目，例如如何使用第三方函式庫，甚至寫自己的 Java 或 Objective-C 函式庫來擴展 React Native。

如果你在接觸手機應用前是軟體前端工程師或是網頁工程師，那這本書本正是為你量身訂作。React Native 是一個令人驚艷的工具，希望你和我一樣在學習過程中感受無比樂趣！

適用讀者

本書不是 React 的入門書，我們會假定讀者已經有一些 React 的經驗，如果你是第一次接觸 React，建議你先讀一到兩本入門書後再回來閱讀本書，才投入手機應用開發的行列。特別是你應該對於 `props` 和 `state` 的角色、元件生命周期和建立 React 元件感到熟悉。

我們也會使用一些新的 JavaScript 語法及 JSX。如果你對這些感到陌生，也不用擔心；我們會在第二章使用 JSX，以及附錄 A 中使用新的 JavaScript 語法，這些新功能可以 1:1 轉換成你已習慣的 JavaScript 程式碼。

雖然 React Native 還可以用來寫 Ubuntu、Windows 及 Linux 的應用程式，但這本書著重於使用 React Native 寫 iOS 與 Android 應用程式，只是為了要能寫出 iOS 應用程式，你必須在 macOS 上進行開發。

本書編排慣例

本書使用下列格式體裁：

斜體字（*Italic*）
: 代表新出現的術語、URL、電子郵件地址、檔案名稱，以及延伸檔名。

定寬字（`Constant width`）
: 用於表示程式碼，或文章段落中的程式組成元素，例如變數或函式名稱、資料庫、資料型別、環境變數、述句與關鍵字。

定寬粗體字（**`Constant width bold`**）
: 用於由讀者鍵入的命令或文字。

定寬斜體字（*`Constant width italic`*）
: 顯示應以使用者所提供的值取代或是由上下文所決定的文字。

此圖示代表小技巧或建議。

此圖示代表一般註解。

此圖示代表警告或注意事項。

第一章

什麼是 React Native ？

React Native 是一個 JavaScript framework，用來在 iOS 和 Android 上撰寫真實、原生的 App。它的基礎是 React，也是 Facebook 用來建立使用者介面的 JavaScript 函式庫，本來目標是做瀏覽器介面，後來變成手機平台。換句話說，它讓網頁開發者透過熟悉的 JavaScript 函式庫就可以寫出原生 App。而且，由於 React Native 大部分的程式碼都可在平台間分享，所以讓 Android 和 iOS 同時開發變得簡單了。

和 React 用在開發網頁時類似，React Native 應用程式是用 JavaScript 和 XML 風格標示語言 JSX 寫成。事實的真相是，React Native“橋接”Objective-C（iOS 上）或 Java（Andoird 上）原生 API。也就是說，你的應用程式會用真實的手機 UI 元件而不是 webview 進行 render，所以長相跟其他的 App 風格一致。React Native 應用程式可以存取像是手機相機或是使用者位置等平台功能。

雖然 React Native 開源專案的核心目標是用來寫 Android 和 iOS 上的 App，不過社群還實作了對 Windows（*https://github.com/Microsoft/react-native-windows*），Ubuntu（*https://github.com/CanonicalLtd/react-native*）、網頁（*https://github.com/necolas/react-native-web*），以及其他種平台的支援。

本書中，我們會用 React Native 建立 Android 和 iOS 上的應用程式，其中大部分的程式碼都會被寫成跨平台。

而且，你真的可以使用 React Native 建立線上手機應用，傳聞 Facebook（*http://bit.ly/1YipO7A*）、Airbnb（*http://bit.ly/2udVlOL*）、Walmart（*http://bit.ly/2vuFIXk*） 和 Baidu（*http://bit.ly/2hzBtnr*）都已使用它來製作使用者應用程式。

React Native 的優勢

React Native 使用它所在平台的標準 API 做 render，這與目前既有大多數的跨方平應用開發方法有所不同，這些既有的跨平台方法像是 Cordova 或 Ionic。在用它們寫 App 時，要合併使用 JavaScript、HTML 和 CSS，render 時則透過 webview。雖然這些方法行得通，但同時也帶來壞處，特別是在效能上面的問題。另外，這些方法通常不能存取平台原生的 UI 元素。當這些方法試圖要模仿一個原生的 UI 元素時，結果總是感覺有點走調。再說，想在這些動畫似的東西底下一探究竟很困難，況且它們進版速度還很快。

相反地，React Native 是真的將你的 markup 轉為真實原生的 UI 元素，不管你是在哪個平台上，都用既存的方法進行畫面 render。而且，React 工作時會和主要的 UI 執行緒分離，所以你的應用程式不用為了跨平台而犧牲效能。React Native 更新的頻率和 React 一樣：當 `props` 或 `state` 變化時，React Native 才會進行畫面重新 render。而 React Native 和 React 間最大的差異在，React Native 是用目標平台的 UI 函式庫，而不是使用 HTML 及 CSS markup 語言。

對於已習慣用 React 開發網頁的開發者而言，你可以使用已熟悉的工具，開發效能和感受都與手機原生應用程式一致的 App。使用 React Native 開發和一般的開發相比，有兩個地方更進階了，就是開發者經驗與跨平台開發的潛力。

開發者經驗

如果你曾經做過手機應用開發，可能會覺得 React Native 怎麼這麼好用。開發 React Native 的那群人，將很多強大的開發者工具及清楚的錯誤訊息放進 framework 中，讓你在開發過程使用到的工具都是可靠的。

舉例來說，由於 React Native“只是”JavaScript，你並不需要重新建置應用程式才能看到你的修改；你可以像重新整理網頁一樣重新整理你的應用程式即可。所以花在等待應用程式建置的時間都是多餘的，React Native 的速度簡直是神的禮物。

另外，React Native 可讓你利用聰明除錯工具和錯誤報告，如果你已習慣使用 Chrome 或 Safari 的開發者工具（如圖 1-1），當你知道可以將這些用在 App 開發上面，應該會覺得開心吧！還有，你可以使用任何的文字編輯工作來編輯 JavaScript，React Native 並不強迫你一定要用 Xcode 開發 iOS 應用程式，或用 Android Studio 開發 Android 應用程式。

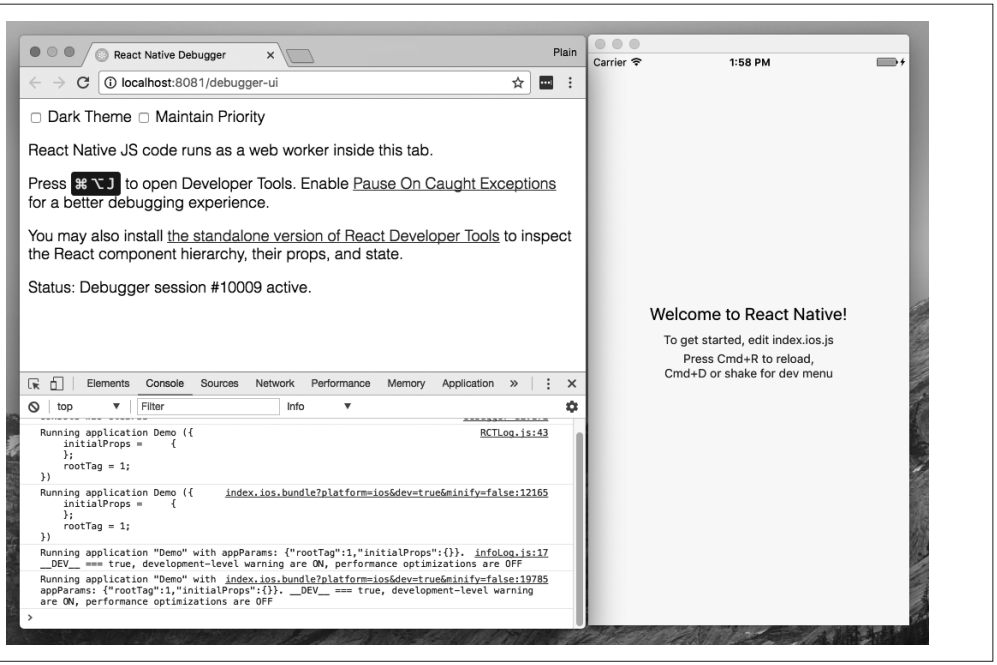

圖 1-1 使用 Chrome Debugger 除錯 React Native

開發者經驗的提昇除了以上那些每天都會碰到的情況之外，React Native 還有可能提昇你的產品生命周期。舉例來說，Apple 和 Google 都允許只上傳 JavaScript 的方法改變應用程式的行為，而不需要再重跑一次審查流程。這一點在 iOS 上特別重要，因為每次應用程式更新時，都要花上數天或數周的審查時間。

所以這些事情加起來，為你和你的同伴結省開發時間和精力，讓你能夠更專注在工作上真正有趣的部分，提昇更高的生產力。

程式碼再利用與知識分享

使用 React Native 可以大幅降低建立 App 所需的資源，任何會使用 React 開發者只要用同一套技術，可以將目標設定為網頁、iOS 或 Android。由於開發者不用再被不同平台綁住，所以他們的知識和資源也可以更有效率分享。

不止是知識可以跨平台，程式碼更可以跨平台。但也不是說你寫的程式碼全部都可以跨平台，依想執行的功能不同，也有可能會寫到一些 Objective-C 或是 Java 程式（我們會在第七章講到怎麼用原生模組）。不過 React Native 中跨平台程式碼的再利用出乎意料

容易，舉例來說，Facebook 的 Android 版 Ads Manager 應用程式就和 iOS 版本分享了 87% 程式碼（*https://youtu.be/PAA9O4E1IM4*）。而本書最後會完成的 flashcard 應用程式，Android 版和 iOS 版本則是完全相同，這真不是其他工具可以做到的事！

風險與缺點

所有東西都一樣，有優點就會有缺點。使用 React Native 也不是沒有壞處，而 React Native 是否適合你的團隊使用，完全端看你們的情況而定。

由於 React Native 會為你的專案多加一層，所以可能會讓除錯變難，特別是在 React 和目標平台間的錯誤。我們會在第九章講述更多深入的除錯細節，並且說明一些常見的問題。

基於同一個原因，當目標平台有更新時——例如 Android 進階版，釋出新的 API 時，React Native 的支援也會延遲一些。不過，還好在大多數情況下，你可以自己實作缺少的 API，我們也會在第七章裡作更多說明。還有，如果你在過程中碰到路障，也不會因為使用 React Native 而變成孤兒——因為很多公司都用混合方法開發 App。

改變撰寫 App 的平台是一件需要慎重考慮的事情，不過我覺得你將會看到使用 React Native 的優點大於過缺點。

本章總結

React Native 是一個令人興奮的 framework，它讓網頁開發者使用既有的 JavaScript 知識，便可以建立穩定的 App。它讓 App 開發加快，並在 iOS、Andoird 和網頁三者可以有效率的共享程式碼，而不用犧牲使用者經驗和應用程式品質。而它的代價是會增加一些應用程式設定的複雜度，如果你的同伴可以克服那些複雜的事情，並想在一個平台開發跨平台的 App，那麼你就應該使用 React Native。

在下一章，我們要看一下 React Native 和網頁用的 React 主要的差異，還有幾個關鍵概念。如果你不想看，想直接開始看開發的部分，可以跳到第三章，在第三章我們會將開發環境設定好，並開始寫第一個 React Native 應用程式。

第二章

使用 React Native

在這一章，我們會談**橋接**，並看看 React Native 底層是如何運作的，然後是 React Native 的元件和網頁版有什麼不一樣，也會談到如果你想要建立或是修改手機用的元件，有什麼是必須知道的。

如果你想直接看開發流程以及 React Native 是怎麼動作的，可以直接跳到第三章！

React Native 是如何運作的？

想到要用 JavaScript 寫 App 就覺得有點怪，要怎麼把 React 用在行動裝置環境呢？為了要瞭解 React Native 底層運作的方法，首先要看一下一個 React 的概念：Virtual DOM。

在 React 中，Virtual DOM 是介於開發者描述介面外觀及頁面顯示這兩者間的那一層。為了要在瀏覽器中 render 出互動的使用者介面，開發者必須編輯瀏覽器的 DOM（Document Object Model）。這一步成本非常高，而且若過度地寫入 DOM，對效能來說會大打折扣。所以，與其直接把改變的東西都重新 render 在頁面上，React 的作法是在記憶體中計算出必要的改變，然後只將少部分必須改變的東西 render，圖 2-1 是這個動作的表示。

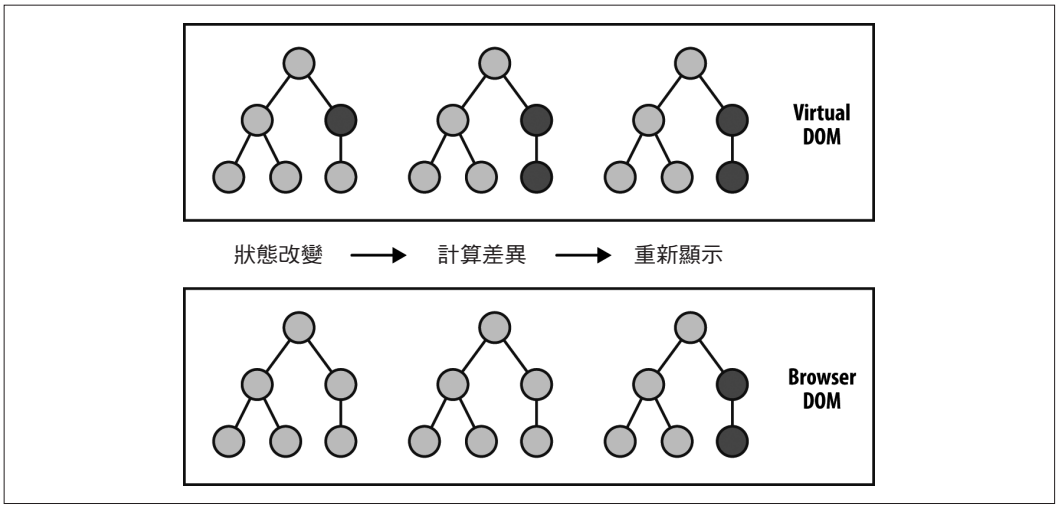

圖 2-1 在 VirtualDOM 中執行計算，減少瀏覽器 DOM 的 render

在網頁使用的 React 中，多數的開發者會把 Virtual DOM 想成是一個效能優化的方法。當然 Virtual DOM 具有提昇效能的好處，但實際上它真的價值是在抽像化的作用。放一個純抽像層在開發者程式碼和實際執行 render 的單位中間，產生了許多有趣的可能性。例如，假設 React 可以控制最後 render 的目標不是瀏覽器的 DOM 呢？畢竟 React 才是真的“懂”你的應用程式應該要長成怎樣的傢伙。

沒錯，這就是 React Native 運作的原理，如圖 2-2。如果不再 render 於瀏覽器的 DOM 上，React Native 改為呼叫 Objective-C 的 API 就可以 render 在 iOS 上，叫用 Java API 就可以 render 在 Android 上。這一點就是 React Native 和其他跨平台 App 開發工具的差異，其他的工具多是 render 在以網頁為基底的畫面上。

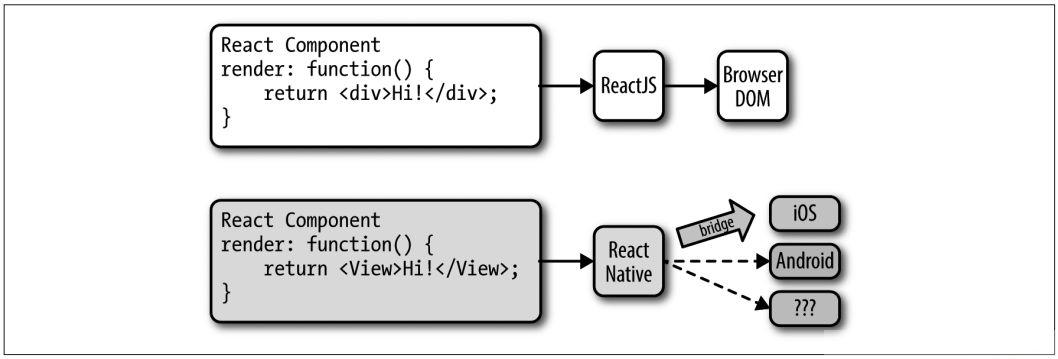

圖 2-2 React 可以 render 到不同目標平台上

由於有了一層橋接，所以這些都得以實現，橋接讓 React 可以使用目標平台原生的 UI 元件，React 元件從它們的 render 函式回傳 markup，markup 說明元件的長相。在網頁上使用 React 時，這些 markup 被直接翻譯為瀏覽器的 DOM，使用 React Native 時，這些 markup 翻譯的結果將適用於目標平台，像 `<View>` 可能翻成 iOS 所用的 `UIView`。

React Native 的核心支援 iOS 和 Android，由於其抽像層是由 Virtual DOM 構成，React Native 也可以適用於其他的平台——只要有人寫相關的橋接。舉例來說，就有社群是實作 React Native 對 Windows 的橋接（*https://github.com/Microsoft/react-native-windows*）及對 Ubuntu 的橋接（*https://github.com/CanonicalLtd/react-native*），你也可以將 React Native 橋接到桌面應用程式。

Render 的生命週期

如果你已習慣使用 React，則對 React 的生命週期應該不陌生。當 React 在瀏覽器上執行時，render 的生命週期從你的 React 元件被掛載（mount）開始（如圖 2-3）。

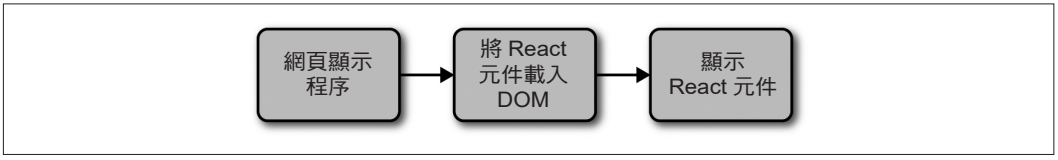

圖 2-3 React 掛載元件

然後 React 接手 render，以及在必要時重新 render 你的元件（如圖 2-4）。

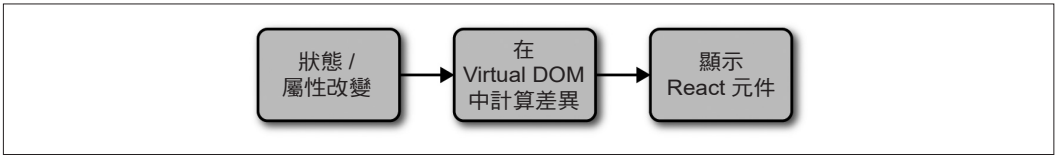

圖 2-4 React 中 Render 元件

在 render 階段，開發者從 React 元件的 `render` 方法回傳 HTML markup，然後 React 就會直接在頁面需要時進行 render。

對 React Native 來說，生命週期是一樣的，只是因為 React Native 要依靠橋接，所以 render 程序有點小小不同。我們之前在圖 2-2 簡單看過橋接，橋接會轉換 JavaScript 呼叫，並呼叫目標平台底層的 API 以及 UI 元件（例如，可能是 Objective-C 或 Java 形式）。由於 React Native 並不在主要 UI 執行緒執行，所以它可以異步（asynchronous）進行那些呼叫而不會影響使用者體驗。

在 React Native 建立元件

所有 React 程式碼都以 React 元件型式存在，React Native 元件原則上和一般的 React 元件一樣，差在 render 和樣式上有所不同。

使用各種 View

為網頁編寫 React 時，你 render 的是一般的 HTML 元素（`<div>`、`<p>`、`<span>`、`<a>`，等等），而用 React Native 時上述的元素都會被目標平台的 React 元件 render（見表 2-1）。其中最基本的就是跨平台的 `<View>`，它是一個簡單又有彈性的 UI 元素，可以類比於 `<div>`。在 iOS 上，`<View>` 元件會被 render 成 `UIView`，而在 Android 上它會 render 成 `View`。

表 2-1 基本的 React 網頁元素和 React Native 比較表

React	React Native
`<div>`	`<View>`
`<span>`	`<Text>`
`<li>,<ul>`	`<FlastList>,child items`
`<img>`	`<Image>`

其他元件則隨各平台定義。舉例來說，`<DatePickerIOS>` 元件（很明顯地）render iOS 的標準日期選擇器（圖 2-5）。以下節錄 `RNTester` 範例程式，其中使用 iOS 的日期選擇器，和你想的一樣，它的用法很簡單：

```
<DatePickerIOS
  date={this.state.date}
  mode="time"
/>
```

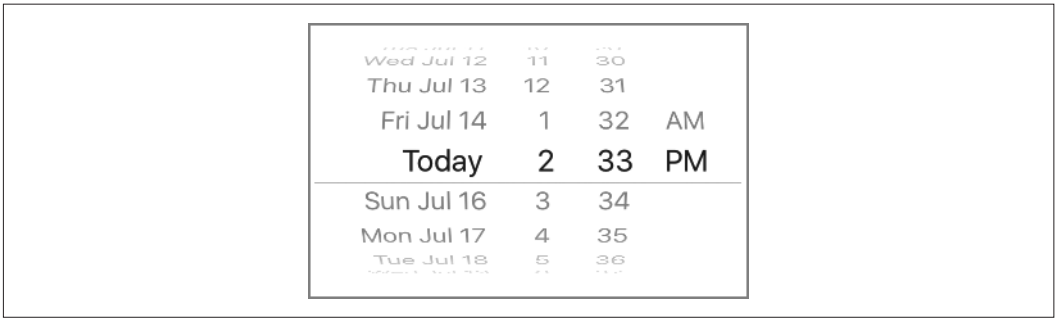

圖 2-5 <DatePickerIOS> 元件，在 iOS 上用來做日期選擇

由於所有我們的 UI 元件已經不是像 `<div>` 那樣的基本 HTML 元件，而是 React 元件，所以你需要明確的引入每個你想用的元件。舉例來說，如果想引用 `<DatePickerIOS>` 就要：

```
import { DatePickerIOS } from "react-native";
```

React Native GitHub 專案中的 `RNTester` 應用（*https://github.com/facebook/react-native/tree/master/RNTester*），可以讓你看到所有支援的 UI 元件。個人鼓勵你在 `RNTester` 中將各種元件查看一輪，看看它們有哪些樣式選項和互動可用。

平台相依性的元件和 API 在文件中會有特別標注，通常在名稱裡也會使用該平台名為後贅——例如 `<TabBarIOS>` 及 `<ToolbarAndroid>`。

由於這類的元件在不同平台上就是不同的東西，所以在用 React Native 時，如何去架構你的 React 元件就變得很重要。用 React 寫網頁時，我們通常會將 React 元件混合使用：部分是管理邏輯以及相關從屬子元件，其他則用來 render 原始元件。如果你想把 React Native 程式碼做成可以重用，則維護不同元件間的獨立性就很重要。像 `<DatePickerIOS>` 元件很明顯就無法用在 Android 上，但封裝邏輯的元件就可以。將視覺化的元件獨立以後，就可以抽換不同的目標平台。你也可以為不同平台設計元件，例如你可以有 *picker.ios.js* 及 *picker.android.js* 檔，內容是同一個元件，只不過是依不同平台的實作分開存放。我們會在 138 頁的“特定平台元件實作”中討論更多相關內容。

使用 JSX

React Native 中和 React 一樣使用 JSX，將 markup 和用來控制介面的 JavaScript 寫在單一檔案中。由於對許多網頁開發者來說，用技術區分不同的檔案已經是一種共識了，例如 CSS、HTML 和 JavaScript 存在不同的檔案中。所以在 React 一開始出現時，將 markup、控制邏輯，甚至是樣式，都用同一種語言寫完的 JSX 受到很大的反彈。

取代以不同技術為分類，JSX 使用不同**目的**為分類，在 React Native 中更是如此。在沒有瀏覽器的世界中，為每個元件在單一檔案中規範樣式、markup 和行為，似乎更為合理。相對的，原來 *.js* 檔在 React Native 中就是 JSX 檔。如果你在網頁開發時，曾經在 React 中使用過純的 JavaScript，那你會想將它轉換為 React Native 中的 JSX 語法格式。

如果你以前沒看過 JSX 也不用擔心，因為它其實蠻簡單的，舉例來說，一個網頁用的純 JavaScript React 元件，可能長得如下面程式碼所示：

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
}

ReactDOM.render(
  React.createElement(HelloMessage, { name: "Bonnie" }), mountNode);
```

如果用 JSX 進行 render 會更簡潔，不用再呼叫 `React.createElement` 並傳入一堆 HTML 參數，改用類似 XML 的 markup 即可：

```
class HelloMessage extends Component {
  render() {
      // 原來要呼叫 createElement，現在改為回傳 markup
    return <div>Hello {this.props.name}</div>;
  }
}

// 不再需要呼叫 createElement 了
ReactDOM.render(<HelloMessage name="Bonnie" />, mountNode);
```

上面的程式碼會在頁面上 render 出下面的 HTML：

```
<div>Hello Bonnie</div>
```

設定原生物件樣式

在作網頁開發時，我們會用 CSS 設計 React 元件的樣式，和做其他 HTML 元件時一樣。不管你喜不喜歡用，CSS 在網頁開發中是不可缺少的。React 通常不會影響 CSS 的撰寫方法，還能以 `props` 和 `state` 為基礎來動態建立類別名稱，但對於網頁上元件樣式著墨不多。

在非網頁平台上設定元件布局和樣式的方法非常多，幸好在使用 React Native 時，可以利用標準的方法設定樣式。在 React 和目標平台中的橋接部分，很大一塊就是在實作精簡版的 CSS 功能，這個簡化版的 CSS 主要依靠 flexbox 做 layout，並且著重在精簡，而不是實作完整的 CSS 規則。在網頁中 CSS 可以支援不同瀏覽器，但 React Native 則是將樣式規則強制統一。你可以在 `RNTester`（*https://github.com/facebook/react-native/tree/master/RNTester*）應用中看到許多不同 UI 元件的樣式設定，RNTester 應用是隨 React Native 發行的一個範例程式。

React Native 也支援以 JavaScript 物件形式存在的 inline 樣式設定。React 開發團隊在網頁應用開發中也使用一樣的方法。如果你之前在 React 中寫過這樣的 inline 樣式，則應該不會覺得陌生：

```
// 定義樣式...
const style = {
  backgroundColor: 'white',
  fontSize: '16px'
};

  // ...套用它
  const txt = (
    <Text style={style}>
      A styled Text
    </Text>);
```

React Native 還提供一些用來建立延伸樣式物件的工具，這些工具讓操作 inline 樣式更容易，我們將會在第五章中說明。

看著 inline 樣式讓你感到有點彆扭嗎？如果你不曾開發網頁，看到這種用法的確是會不習慣。使用樣式物件而不是使用樣式清單（stylesheet），需要做一下心態調整，改變一下原來的樣式使用習慣。不過，在 React Native 中，這個轉換是好事。我們會在第五章討論樣式，若現在看到它們，只要不過度驚訝即可！

目標平台 API

將開發網頁的 React 和 React Native 對照一下，也許最大的差異，是在我們看待目標平台 API 的角度不同。在網頁開發時，我們操作的通常是各種片段不連續的技術；但大部分的瀏覽器在核心和功能上，還是大致相同的。但在使用 React Native 時，如果想要開發出好用又直捷的使用者經驗，此時目標平台的 API 就占了舉足輕重的地位。其中又有許多可著墨的選項，手機上的 API 包羅萬象，從儲存資料、地點服務到存取硬體（如相機）。而新式的平台又會有更多有趣的 API—例如把 React Native 用在虛擬實境頭盔上時，介面又要怎麼設計呢？

React Native 對 iOS 和 Android 的預設支援，包括了許多常用功能，還可以支援非同步原生 API。在本書後續的章節裡，會介紹許多這種非同步原生 API。React Native 讓目標平台的 API 用起來直捷又簡單，所以你可以自由體驗看看。重點是在腦中勾勒你的目標平台上用什麼、怎麼動作才**對味**。

不可避免的，React Native 並不會將目標平台的所有功能都橋接起來，如果你在未來發現需要用的功能尚未被支援時，可以自行將該功能加上去。也很有可能另外已經有人做好該功能了，所以記得上社群去查看一下。我們會在第七章談論更多這個主題。

在考慮程式碼重用時，也要注意到使用目標平台 API 的部分。會用到特定平台功能的 React 元件，其本身也有平台適用性問題。獨立封裝這些元件可以為你的應用程式增添使用彈性。當然，這個原則也同時適用於網頁開發：如果你想在 React Native 和 React 間共用程式碼，在設計時就要注意 DOM 這種東西在 React Native 中是不存在的。

本章總結

在手機上用 React Native 編寫元件和在網頁上用 React 編寫元件有點不同。雖然都是使用 JSX，但是基本構成的區塊從 HTML 元素（如 `<div>`）改為 `<View>` 元件。樣式的設定也有一定的差異，變成使用 CSS 的子集合及可使用 inline 語法設定樣式，雖然有所調整，但這些差異還算是好處理的。在下一章，將會實作我們的第一個應用程式！