國立高雄科技大學 智慧商務系
National Kaohsiung University of Science and Technology Department of Intelligent Commerce

6. React native 基本介紹 II

廖奕雯
yiwen923@nkust.edu.tw

Outline

- Prop
- State
- 練習一下
- 整合元件
- JSX 介紹

props

Prop

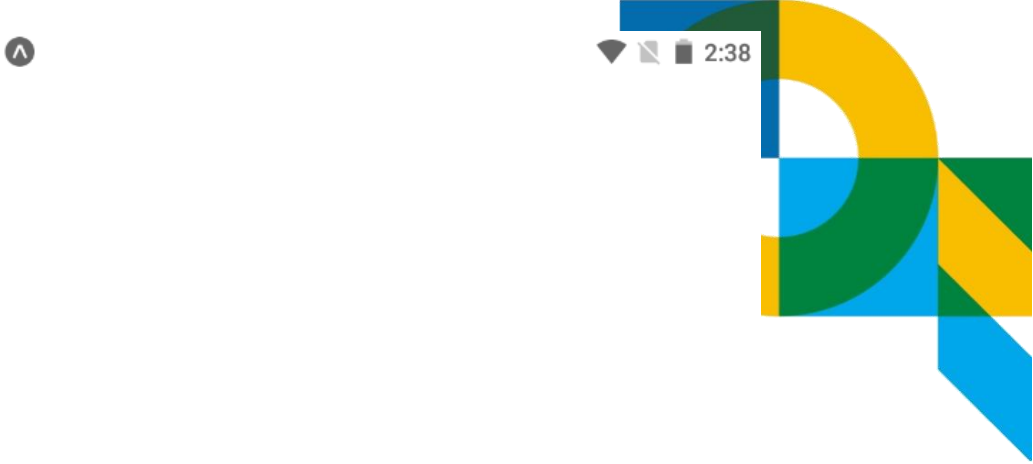
- **Props** 是 “properties” (屬性) 的簡寫。
- 在 React Native 中，建立組件 (components) 時，可以使用各種自定義參數，而這些自定義參數便是 Components的props屬性。
- Props負責 Components 間的屬性資料傳遞。

File: App.js

```
01: import { StatusBar } from 'expo-status-bar';
02: import { Component } from 'react';
03: import { StyleSheet, Text, View, Button } from 'react-native';
04:
05: const Hello = (props) => {
06:   return (
07:     <View>
08:       <Text style={styles.textstyle}>Hello {props.title}.</Text>
09:       <Text style={styles.textstyle}>I am {props.name}.</Text>
10:     </View>
11:   );
12: }
13:
14: export default function App() {
15:   return (
16:     <View style={styles.container}>
17:       <Hello title="World" name="YW"/>
18:       <Hello title="React Native!"/>
19:     </View>
20:   );
21: }
22:
23: const styles = StyleSheet.create({
24:   container: {
25:     flex: 1,
26:     backgroundColor: '#fff',
27:     alignItems: 'center',
28:     justifyContent: 'center',
29:   },
30:   textstyle:{
31:     fontSize: 32,
32:   }
33: });
```

設定props

傳遞 props



少一個 props

props 型態

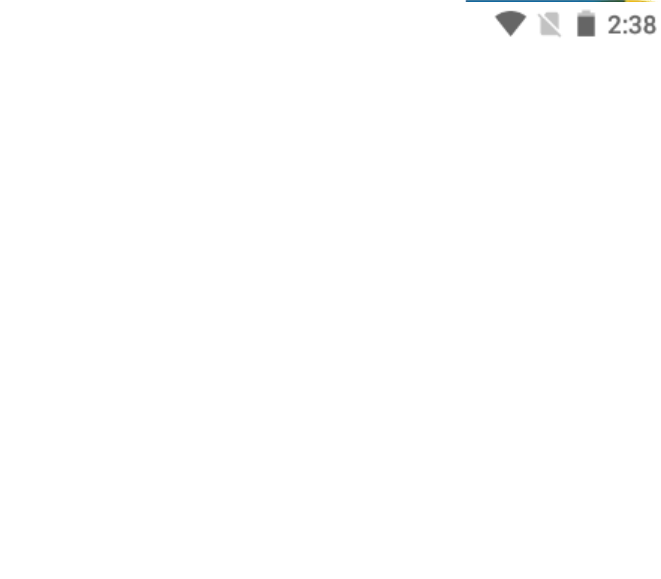
- 我們可以設定 props 型態，預防輸入的屬性值格式錯誤。
- 型態檢查工具，使用步驟
    1. 安裝 prop-types
        1. npm install prop-types --save
    2. 使用 library
        1. import PropTypes from 'prop-types';
    3. 設定 props type

props 型態

```
File: App-props.js
15: Hello.propTypes={
16:   title: PropTypes.string,
17:   name: PropTypes.string
18: }
19:
20: Hello.defaultProps ={
21:   title: "World",
22:   name: 'abc'
23: }
24:
25: export default function App() {
26:   return (
27:     <View style={styles.container}>
28:       <Hello title="World" name="YW"/>
29:       <Hello title="React Native!"/>
30:     </View>
31:   );
32: }
```

預設型態

預設內容



props 型態

- props 型態與Javascript 型態相同

Javascript	描述	props
string	字串	PropTypes.string
number	數字	PropTypes.number
bool	布林值	PropTypes.bool
array	陣列	PropTypes.array
object	物件	PropTypes.object
func	方法	PropTypes.func

props 型態

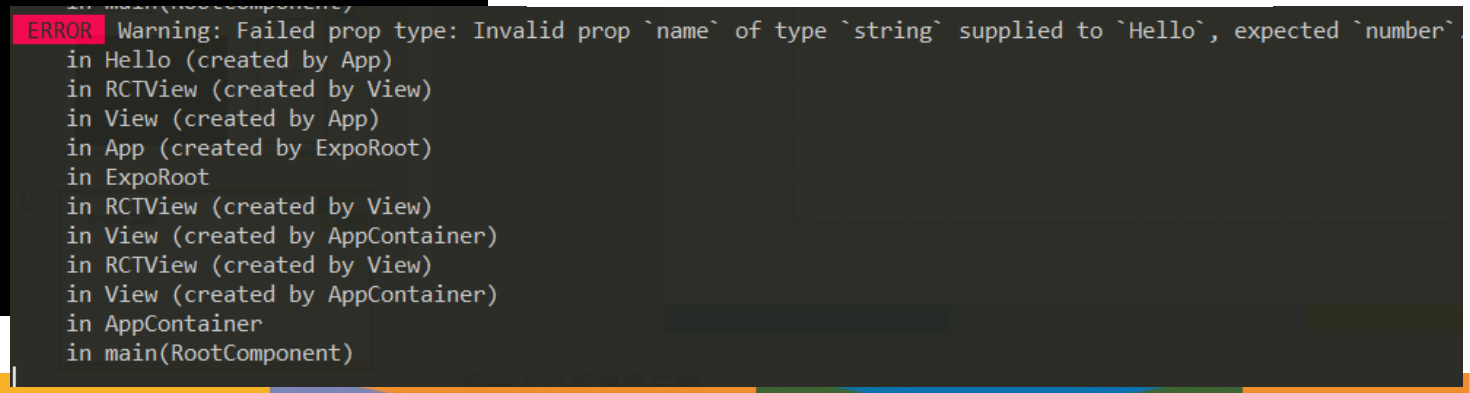
- props特殊形態

型態	描述	範例
oneOf()	限制型態是限定的值之一	PropType.oneOF(['A','B'])
oneOfType()	限制型態是限定的型態之一	PropType.oneOfType([PropTypes.string, PropTypes.string])
arrayOf()	設定陣列內元素的型態	PropType.arrayOf(PropTypes.string)
objectOf()	設定物件內元素的型態	PropType.objectOf(PropTypes.string)

```
File: App-props.js
06: const Hello = (props) => {
07:   return (
08:     <View>
09:       <Text style={styles.textstyle}>Hello {props.title}.</Text>
10:       <Text style={styles.textstyle}>I am {props.name}.</Text>
11:     </View>
12:   );
13: }
14:
15: Hello.propTypes={
16:   title: PropTypes.string,
17:   name: PropTypes.number
18: }
19:
20: Hello.defaultProps ={
21:   title: "World",
22:   name: 'abc'
23:   }
24:
25: export default function App() {
26:   return (
27:     <View style={styles.container}>
28:       <Hello title="World" name="YW"/>
29:       <Hello title="React Native!"/>
30:     </View>
31:   );
32: }
```

資料型態錯誤

Hello World.
I am YW.
Hello React Native!.
I am NAME.



State

State

- 如果把 props 理解為components render 的參數， 那麼state就像是components的私人資料記錄。
- State 用於記錄那些隨時間或者使用者互動而變化的資料。
- State使components擁有了記憶！
- Prop是由 parent components傳入的屬性，state是components自己建立的內部變數。
- prop 在 components內不能改變，State 可以。
- 由於state是components內的變數，所以無法存取被其他的components存取。

useState 怎麼用?

- 首先，需要 import library

- 語法: `import {useState} from 'react';`

useState 怎麼用?

- 宣告語法: `const [state variable, setter function] = useState(default value)`
  - state variable : 透過 useState()產生的變數 。我們希望監控的變數，名稱可以任取。
  - setter function : useState用來修改變數的函式，名稱可以任取。慣例是 “set變數”。
  - useState(): 可以帶入資料預設值。資料可以是數值、字串、物件等。

- 例如: `const [conut, setCount] = useState(100)`
  - 變數是 count
  - 修改 count 的方法為 setCount
  - 預設值是100

useState 怎麼用?

- 在需要修改變數的地方呼叫 setter function
- 語法: setState(資料)
- 例如: setCount(29)

State 範例

1. 使用 useState 創建了一個 isHungry 狀態變數：

```
const Cat = (props) => {
  const [isHungry, setIsHungry] = useState(true);
  // ...
};
```

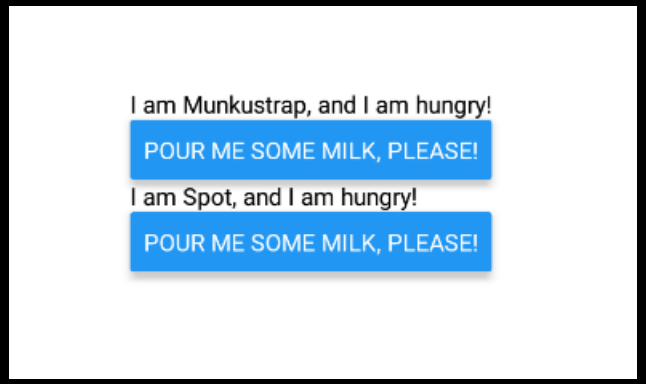
   - 創建一個“state變數”，並賦予一個初始值。
   - 例子中的狀態變數是isHungry，初始值為true。
2. 同時創建一個函數用於設置此 state 變數的值—setIsHungry。

State 使用步驟

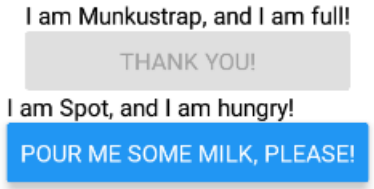
3. 添加改變 state 的 function到一個 Component。
   - 例如一個按鈕。

```
<Button
  onPress={() => {
    setIsHungry(false);
  }}
  //..
/>
```

```jsx
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View, Button } from 'react-native';
import React, { useState } from 'react';

const Cat = (props) => {
  const [isHungry, setIsHungry] = useState(true);

  return (
    <View>
      <Text>
        I am {props.name}, and I am {isHungry ? "hungry" : "full"}!
      </Text>
      <Button
        onPress={() => {
          setIsHungry(false);
        }}
        disabled={!isHungry}
        title={isHungry ? "Pour me some milk, please!" : "Thank you!"}
      />
    </View>
  );
}
const Cafe = () => {
  return (
    <>
      <Cat name="Munkustrap" />
      <Cat name="Spot" />
    </>
  );
}
```




```jsx
export default function App() {
  return (
    <View style={styles.container}>
      <Cat name="Munkustrap" />
      <Cat name="Spot" />
    </View>

  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```



React Native 官網 API 文件

- React Native 官網
  - https://reactnative.dev/
- React Native 中文網
  - https://www.react-native.cn/

練習一下

1. 建立一個 TextInput和 Text
2. 當使用者在 TextInput輸入資料時，Text可以同時顯示

整合元件

Switch

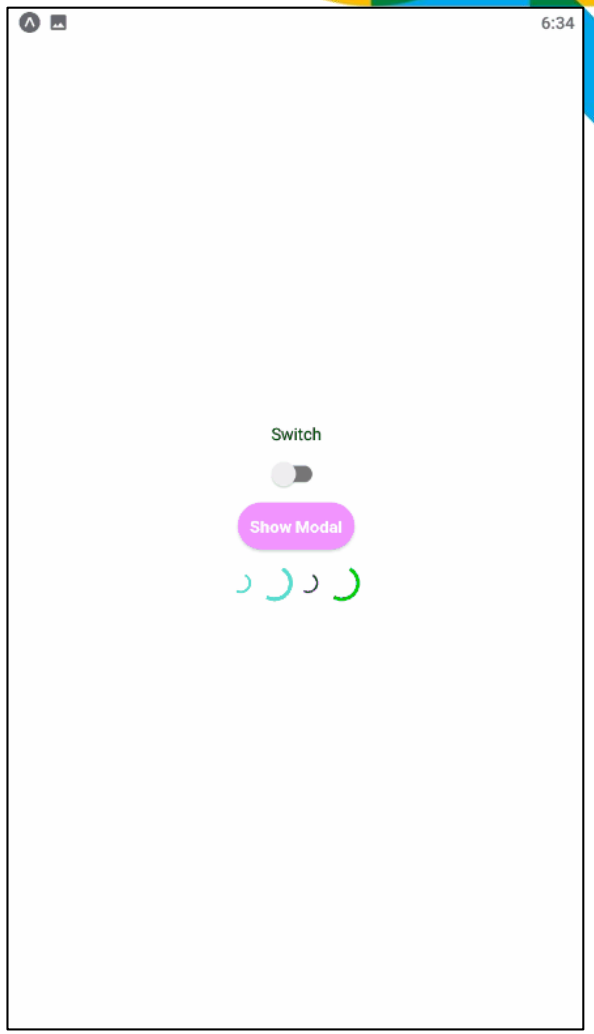
- 用來做切換的組件，例如 HTML 的 checkbox

屬性	描述
disabled	設定此 switch 是否可以切換
value	設定此 switch 的值
thumbColor	開關上圓形按鈕的背景顏色。在 iOS 上設置此顏色會丟失按鈕的投影效果。
ios_backgroundColor	iOS上的背景顏色
trackColor	關閉狀態時的邊框顏色(iOS)或背景顏色(Android)。

Callback function	描述
onChange	當值改變的時候調用此Callback function，參數為事件。 If you want to only receive the new value, use onValueChange instead.
onValueChange	當值改變的時候調用此Callback function，參數為新的值。 If you want to instead receive an event, use onChange.

Switch

```
File: App-compose.js
05: export default function App() {
06:
07:   const [isEnabled, setIsEnabled] = useState(false);
08:   const toggleSwitch = () => setIsEnabled(previousState => !previousState);
09:
10:   return (
11:     <View style={styles.container}>
12:       <Text>Switch</Text>
13:       <Switch
14:         trackColor={{ false: "#767577", true: "#81b0ff" }}
15:         thumbColor={isEnabled ? "#f5dd4b" : "#f4f3f4"}
16:         ios_backgroundColor="#3e3e3e"
17:         onValueChange={toggleSwitch}
18:         value={isEnabled}
19:       />
20:       <StatusBar style="auto" />
21:     </View>
22:   );
23: }
```



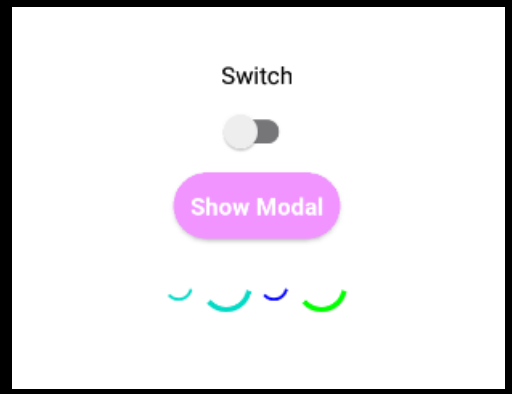
Modal

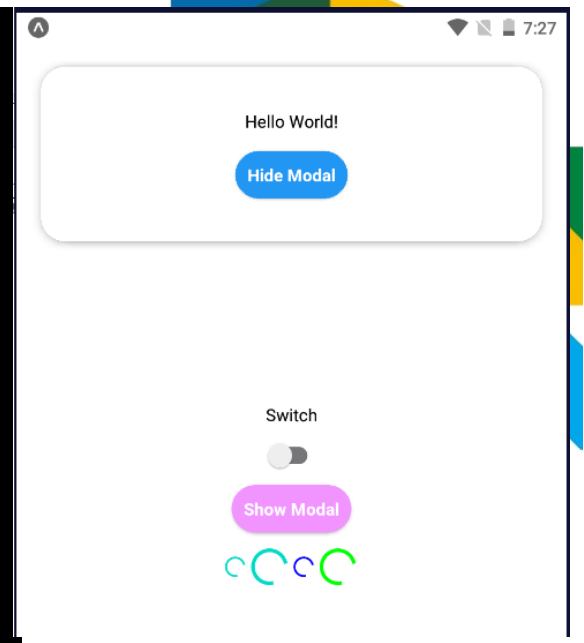
- Modal 元件是一種簡單的覆蓋在其他View之上顯示內容的方式。

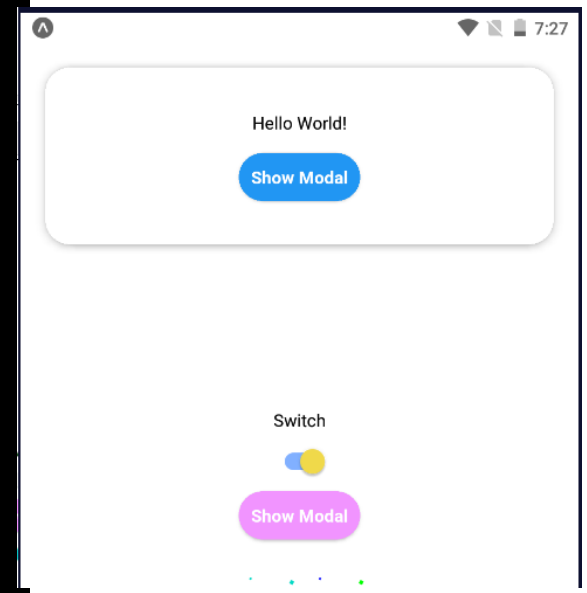
屬性	描述
visible	visible屬性決定 modal 是否顯示。
animationType	animationType指定 modal 的動畫類型: • slide 從底部滑入滑出。 • fade 淡入淡出。 • none 沒有動畫，直接出來。
transparent	transparent 屬性是指背景是否透明，預設為白色。 將這個屬性設為：true 的時候彈出一個透明背景層的 modal。

Callback function	描述
onShow	onShow Callback function 會在 modal 顯示時調用。
onDismiss	onDismiss Callback function 會在 modal 被關閉時調用。

```jsx
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Switch, Text, View, Modal, Alert, TouchableHighlight, ActivityIndicator }
from 'react-native';
import React, { useState } from "react";

export default function App() {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState);
  const [modalVisible, setModalVisible] = useState(false);

  return (
    <View style={styles.container}>
      <Text>Switch</Text>
      <Switch
        trackColor={{ false: "#767577", true: "#81b0ff" }}
        thumbColor={isEnabled ? "#f5dd4b" : "#f4f3f4"}
        ios_backgroundColor="#3e3e3e"
        onValueChange={toggleSwitch}
        value={isEnabled}
      />

      <Modal
        animationType="slide"
        transparent={true}
        visible={modalVisible}
        onRequestClose={() => {
          Alert.alert("Modal has been closed.");
          setModalVisible(!modalVisible);
        }}
      >
```







```
  <View style={styles.centeredView}>
          <View style={styles.modalView}>
            <Text style={styles.modalText}>Hello World!</Text>

            <TouchableHighlight
              style={{ ...styles.openButton, backgroundColor: "#2196F3" }}
              onPress={() => {
                setModalVisible(!modalVisible);
              }}
            >

              <Text style={styles.textStyle}>
                {isEnabled ? "Show Modal" : "Hide Modal"}
              </Text>
            </TouchableHighlight>
          </View>
        </View>
      </Modal>
```

```
      <TouchableHighlight
        style={styles.openButton}
        onPress={() => {
          setModalVisible(true);
        }}
      >
        <Text style={styles.textStyle}>Show Modal</Text>
      </TouchableHighlight>

      <View style={styles.horizontal}>
        <ActivityIndicator />
        <ActivityIndicator size="large" />
        <ActivityIndicator size="small" color="#0000ff" />
        <ActivityIndicator size="large" color="#00ff00" />
      </View>
      <StatusBar style="auto" />
    </View>
  );
}
```

Modal

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  modalView: {
    margin: 20,
    backgroundColor: "white",
    borderRadius: 20,
    padding: 35,
    alignItems: "center",
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 2
    },
    shadowOpacity: 0.25,
    shadowRadius: 3.84,
    elevation: 5
  },
```

```
  openButton: {
    backgroundColor: "#F194FF",
    borderRadius: 20,
    padding: 10,
    elevation: 2
  },
  textStyle: {
    color: "white",
    fontWeight: "bold",
    textAlign: "center"
  },
  modalText: {
    marginBottom: 15,
    textAlign: "center"
  },
  horizontal: {
    flexDirection: "row",
    justifyContent: "space-around",
    padding: 10
  }
});
```

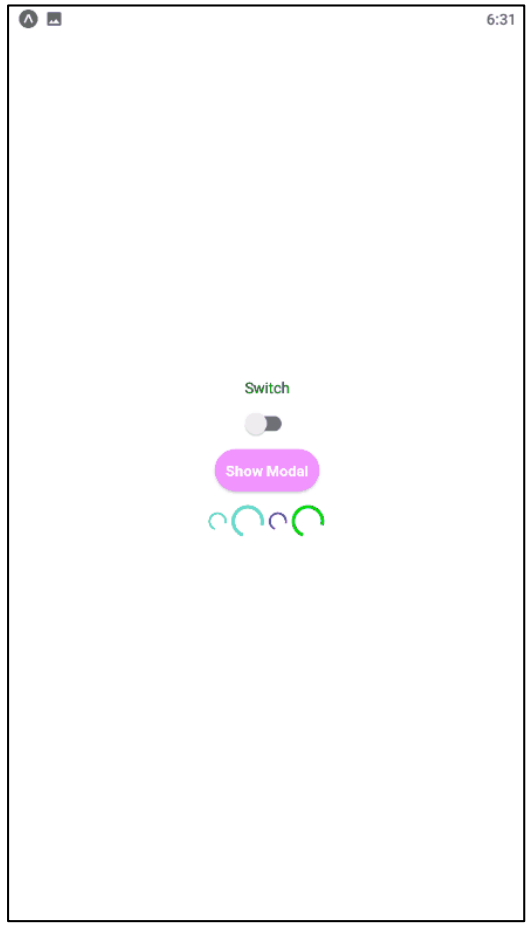
ActivityIndicator

- 顯示一個圓形的 loading 提示符號。

屬性	描述
animating	是否要顯示指示器動畫，預設為 true 表示顯示，false 則隱藏。
color	滾輪的前景顏色。
size	指示器的大小，默認為'small'。目前只能在 Android 上設定具體的數值。 enum('small', 'large')

ActivityIndicator

```
File: App-compose.js
58:         <View style={styles.horizontal}>
59:           <ActivityIndicator />
60:           <ActivityIndicator size="large" />
61:           <ActivityIndicator size="small" color="#0000ff" />
62:           <ActivityIndicator size="large" color="#00ff00" />
63:         </View>
```



國立高雄科技大學 智慧商務系
National Kaohsiung University of Science and Technology Department of Intelligent Commerce

JSX 介紹

Reference: https://zh-hant.reactjs.org/docs/introducing-jsx.html

- 考慮下面這個變數宣告：

```
const element = <h1>你好，世界！</h1>;
```

- 這個有趣的標籤語法不是一個字串也不是 HTML。

- 這個語法叫做 JSX，是一個 JavaScript 的語法擴充。我們推薦你在寫 React 的時候透過這個語法來描述使用者介面的外觀。
- JSX 可能為讓你想到一些樣板語言，但不一樣的地方是 JSX 允許你使用 JavaScript 所有的功能。
- 執行 JSX 會產生 React「element」。

為什麼要用 JSX?

- React 擁抱了 render 邏輯從根本上就得跟其他 UI 邏輯綁在一起的事實：事件要怎麼處理？隨著時間經過 state 會如何變化？以及要怎麼將資料準備好用於顯示？
- 與其刻意的將技術拆開，把標籤語法跟邏輯拆放於不同檔案之中，React 關注點分離的方法是將其拆分為很多同時包含 UI 與邏輯的 component，而彼此之間很少互相依賴。
- React 並不要求使用 JSX，但大部分人覺得在 JavaScript 程式碼中撰寫使用者介面的同時，這是一個很好的視覺輔助。這也允許 React 顯示更有用的錯誤及警告訊息。

在 JSX 中嵌入 Expression

- 在下面這個範例中，我們宣告一個名為 name 的變數，並在 JSX 中透過將其名稱包在大括號中使用：

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

- 你可以在 JSX 的大括號中寫入任何合法的 JavaScript expression。舉例來說，2 + 2、user.firstName 以及 formatName(user) 都是合法的 JavaScript expression。

在 JSX 中嵌入 Expression

- 接下來的範例中，我們將嵌入呼叫 JavaScript function (formatName(user)) 的回傳值到一個 <h1> element 中。

```
function formatName(user) {
  return user.firstName+ ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

JSX 本身也是 Expression

- 在編譯之後，JSX expressions 就變成了一般的 JavaScript function 呼叫並回傳 JavaScript 物件。 這表示你也可以在 if 跟 for 迴圈中使用 JSX，將其指定到一個變數，使用 JSX 作為參數並由 function 中回傳。

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

在 JSX 中指定屬性

- 你可以使用引號將字串設定為屬性：

```
const element = <a href="https://www.reactjs.org"> link </a>;
```

- 你也可以在屬性中使用大括號來嵌入一個 JavaScript expression：

```
const element = <img src={user.avatarUrl}></img>;
```

- 不要在嵌入 JavaScript expression 作為屬性的時候同時使用引號或是大括號。你應該要在使用字串屬性的時候使用引號，使用 expressions 的時候使用大括號，但不要同時使用。

在 JSX 中指定 Children

- 就像是在 XML 之中，如果一個標籤是空白的，你可以用 /> 立刻關閉這個標籤：

```
const element = <img src={user.avatarUrl} />;
```

- JSX 標籤也可以包含 children：

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

JSX 防範注入攻擊

- 在 JSX 之中可以安全的直接嵌入使用者輸入：

```
const title = response.potentiallyMaliciousInput;
// 這是安全的：
const element = <h1>{title}</h1>;
```

- React DOM 預設會在 render 之前 escape 所有嵌入在 JSX 中的變數。這保證你永遠不會不小心注入任何不是直接寫在你的應用程式中的東西。所有變數都會在 render 之前轉為字串，這可以避免 XSS（跨網站指令碼）攻擊。

JSX 表示物件

- Babel 將 JSX 編譯為呼叫 React.createElement() 的程式。
- 下面例子完全相同：

```jsx
const element = (
  <h1 className="greeting">
    Hello, World!
  </h1>
);
```

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, World!'
);
```

```jsx
// 注意：這是簡化過的結構
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

Q&A