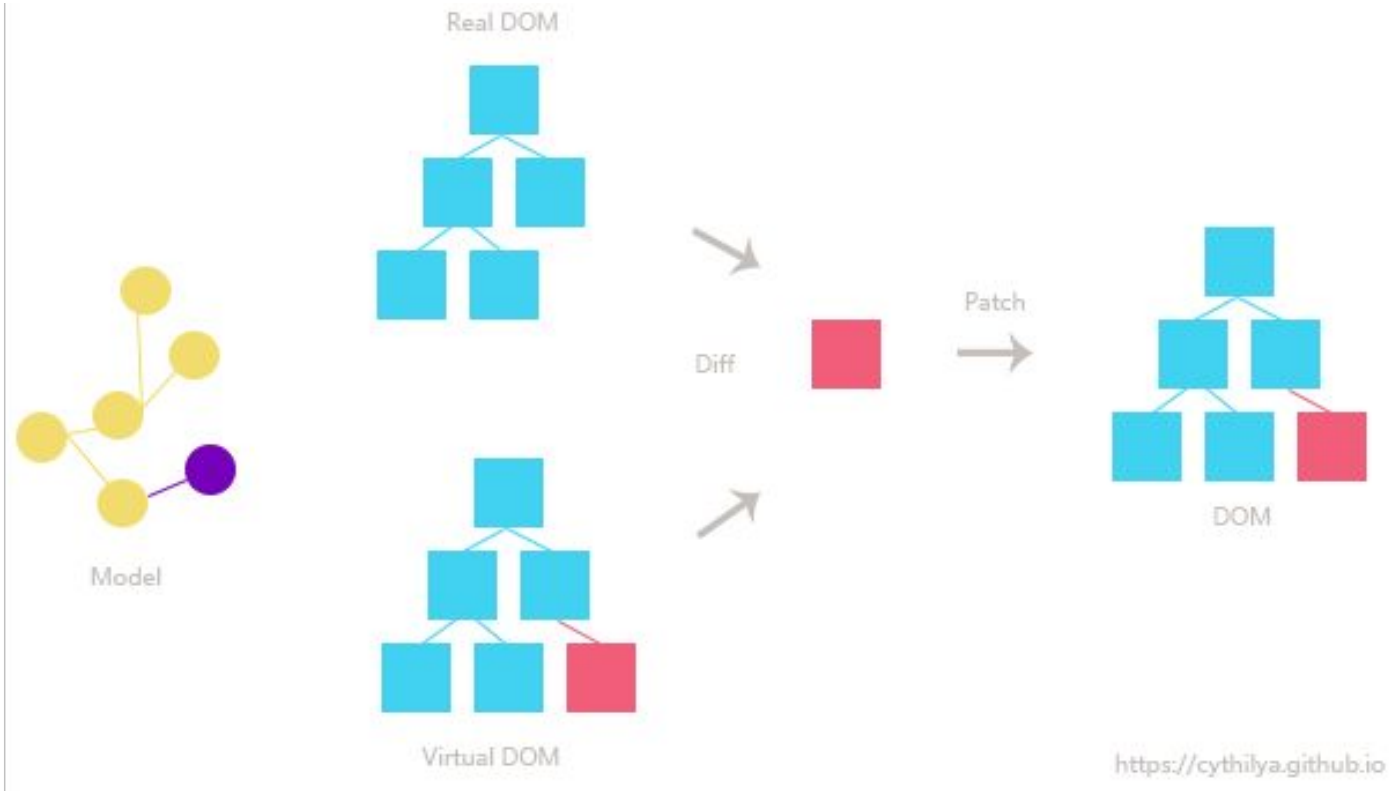
給初學者的 React 教學

WIFI
ntut-306
87737041

React 是什麼？

- 由 FaceBook 開發的開源 JavaScript 函式庫
- 用於建立前端的使用者介面(UI)。
- 模組化開發，建立可重複使用的組件，快速地組成複雜的 UI。
- 適合用於開發單頁式應用程式(single-page application)
- 先定義好 UI 如何呈現，再將資料灌進去，每當資料改變就進行畫面重繪
- 用 Virtual DOM 決定 UI 的重繪

使用 Virtual DOM 的最大好處就是**提升效能**



安裝環境

- 靜態 HTML + CDNs
- react + react-dom + Babel + Webpack + ……
- create-react-app
- CodeSandBox
  - 線上前端IDE，支援快速建立React、Vue、Angular……等專案。
- 偵錯工具
  - React Developer Tools
  - Redux DevTools

靜態 HTML + CDNs

```
<head>
  ……
  // 引入 react, react-dom, babel
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
</head>
<body>
  <div id="root">
    // 畫面將渲染在這個節點內
  </div>
  <script type="text/babel">
    // React 程式碼寫在這
  </script>
 ……
```

Create React App

預處理編譯、打包.....等等的設定，專注於內容的開發。

這些設定包含：

- 即時開發伺服器(hot-reload server)
- Webpack 自動編譯 React、JSX 和 ES6，為 CSS 自動加上瀏覽器前綴
- ESLint 檢查程式碼
- Jest 單元測試工具

```
npx create-react-app <app-name>
cd <app-name>
npm start
```

Create React App 目錄架構

```
|-- node_modules // 所有依賴套件
|-- package.json  // 專案基本資訊
|-- public
|   |-- index.html // react 渲染的根節點
|   |-- manifest.json
|-- src
   |-- App.css
   |-- App.js // 範例元件
   |-- App.test.js  // 單元測試程式碼
   |-- index.css
   |-- index.js // JavaScript 入口點
```

常用 ES6 語法

- 箭號函數(arrow function)
  - (m, n) => { }
  - 函數的簡化寫法，() 內是參數，{ } 內是函數本體
  - () => () 省略 { } 代表直接回傳箭號後的內容
  - 只有一個參數可以省略 ( )，例如: m => { }，沒有參數則一定要有 ( )
  - 箭號函數的 this 是定義時的物件，而不是使用時的物件
- 預設參數

```
const greet = (name='Guest') => console.log(`Hello, ${name}`)
greet() // Hello, Guest

function greet(name) {
    console.log('Hello, ' + (name || 'Guest'))
}
```

- 樣板字串(template literals)
  - `Hello, ${ name }`
  - ` ` 包裹字串，字串內可用 ${ } 帶入變數

```
const num = 10
const new = `Next number is ${ num + 1}`
var old = ‘Next number is ‘ + (num + 1)
```

- 解構賦值(destructuring assignment)

```
// array
const [a, b, …rest] = [1, 2, 3, 4, 5];
console.log(a, b, rest); // 1 2 [3, 4, 5]
// object
const o = { p: 42, q: true };
const { p, q } = o;
console.log(p, q); // 42 true
```

- 物件縮寫(object literals)
  - 如果欲賦予物件屬性的變數名稱與屬性名稱相同，可以省略屬性 值
  - 物件屬性名稱可定義為經由計算的結果

```
const name = ‘John’
const age = 20
const major = ‘Computer Science’
const person = {
  name,
  age,
  [age > 24 ? ‘jobTitle’ : ‘major’]
}
// 等同於
var person = {
  name: name,
  age: age
}
person.age > 20 ? person.jobTitle = ‘’ : person.major = major
```

- const & let 區塊域變數
  - const 只會在定義的 { } 內有效，定義時必須 initialize，而且不能被更改
  - let 只會在目前定義的 { } 內有效，重複定義時會 throw error
- class
  - 原型(prototype)物件導向的語法糖
  - 命名慣例以大寫駝峰命名 (upper camel case)
  - 定義 class SomeClass { }
  - 實例化 new SomeClass()
  - class SomeClass extends AnotherClass { }
  - constructor() 在 class 被實例化的時候會被呼叫一次
  - 通過 super 呼叫 parent class
  - 撰寫 React 組件必用的語法

```
class Cat {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(`${ this.name } makes a noise.`);
  }
}

class Lion extends Cat {
  speak() {
    super.speak();
    console.log(`${ this.name } roars.`);
  }
}

const lion = new Lion('Simba')
lion.speak() // Simba makes a noise. Simba roars
```

- 模組化
  - 匯出(export)分為 named 和 default export
    - 同一個檔案可以有多個 named export，但是只會有一個 default export
  - 匯入(import)
    - Import named export 必須用 { } 包住 export 名稱，default export 則可以用任意名稱 import
  - 一般來說 React 組件會將主要回傳 React Element 的函數作為 default export

JSX: JavaScript XML

JavaScript 語法糖，用來描述 UI 。

```
const reactElement = <h1 className='title'>Hello React!</h1>
// 經 Bable 編譯會等同於
React.createElement('h1', { className: 'title' }, 'Hello React!')
```

可以使用 HTML DOM 標籤，也可以自訂標籤，也就是 React 組件。

```
const Message = () => <div>Hello<div>
<Message />
```

JSX 基本規則

- class 需寫為 className 以避開 JavaScript 保留字 class。
- Self-closing tag 結尾必須有 / ，例如 <img /> 與 <input />。
- 行內樣式以 style={ styleObj } 表示，styleObj 是含有樣式 key:value 的物件
  - 例如 style={{ color: ‘blue’, fontSize: ‘20px’ }}
  - 樣式屬性名含有 - 的需寫成駝峰式，例如 margin-left → marginLeft
  - 樣式屬性值以 string 表示
- React Element 事件處理與 HTML DOM Element 相似，以下幾點不同

React Element	HTML DOM Element
camelCase <button onClick={}>	小寫 <button onclick=””>
事件值為 function	事件值為 string
必須使用 preventDefault() 來避免瀏覽器預設行為	可用 return false 來避免瀏覽器預設行為

- 用 { } 嵌入 JavaScript 表達式
  - 帶入變數值
  - `樣板字符串`(template literial)計算
  - JavaScript 運算，例如 n + 1
  - && || 或三元運算子控制渲染內容

```
const person = ‘John’
const gender = ‘male’
const greeting = () => console.log(`Hello ${ person }!` );
const Greet = (
    <button onClick={ greeging }}>
        { gender === ‘male’ ? `Hello, Mr. ${ person }!` : `Hello, Ms. ${persion}`}
    </button>
)
```

Class Component & Function Component

Component 字首須為大寫字母，小寫字母開頭的組件則視為原始 DOM 標籤。
兩種 component 都需要 import React，JSX 才能被辨識然後編譯。

Class Component	Function Component

● 繼承 React.Component 子類別的類別
● 擁有 state 與生命週期方法
● 在 constructor() 方法裡初始化組件
● 在 render() 方法裡定義回傳的 React Element
class ClassComp extends React.Component {
render() {
return (<h1>Hello React!<h1/>)
}
}
	● 單純的 JavaScript function
	● 接受傳入的參數(props)並且回傳 React Element
	● 沒有 state 與生命週期方法
	function FunctionComp() {
	return (<h1>Hello React!<h1/>)
	}

Class Component

- 使用 ES6 class 語法建立並繼承 React.Component 子類別
- constructor()
  - 組件初始化時會執行此函數
  - 在此函數內設定組件 state 的初始值
- state
  - 儲存與組件相關的資料
  - 可用 React Developer Tools 觀察 state 變化
- 生命週期(lifecycles)
  - 大部分與組件的掛載 (mount)狀態有關
  - componentDidMount, componentWillUnmount, componentDidUpdate……
- render()
- 範例

```
constructor(props) {
    super(props)
    this.state = { title: ‘Hello React’ }
}
```

補充說明: 為什麼需要 super(props)

在 componentDidMount() 抓資料

```
……
constructor() {
    super(props)
    this.state = { data: null }
}

componentDidMount() {
   fetch("https://5db91ed6177b350014ac8050.mockapi.io/api/users")
     .then(res => res.json())
     .then(data => {
         console.log(data)
         this.setState({ data })
     );
 }
```

組件屬性(Props)

- 組件就像 JavaScript 函數可以接受參數傳入並回傳 React Element，在組件標籤上以 key=value 表示傳入屬性。
- Props 為唯讀，不可作修改，若有需要對傳入的資料修改，必須將資料深拷貝並存在其他地方，例如組件的 state 或是 redux store。
- Class component 從 this.props 可以取得所有傳入的屬性物件。
- Funtion component 則是直接從傳入的 props 物件取得所有傳入的屬性。

```
// function component
// 等同於 const MessageA = (props) => <div className={ props.type }>{ props.text }<div>
const MessageA = ({ text, type }) => <div className={ type }>{ text }<div>

// class component
class MessageB extends React.Component {
    render() {
      // 等同於 const msg = this.props.msg
      //            const type = this.props.type
      const { msg, type } = this.props
      return <div className={ type }>{ msg }</div>
    }
}

<MessageA text={ ‘This is a function component’ } type=”description” />
<MessageB text={ ‘This is a class component’ } type=”description” />
```

補充說明: ES6 object destructuring

State

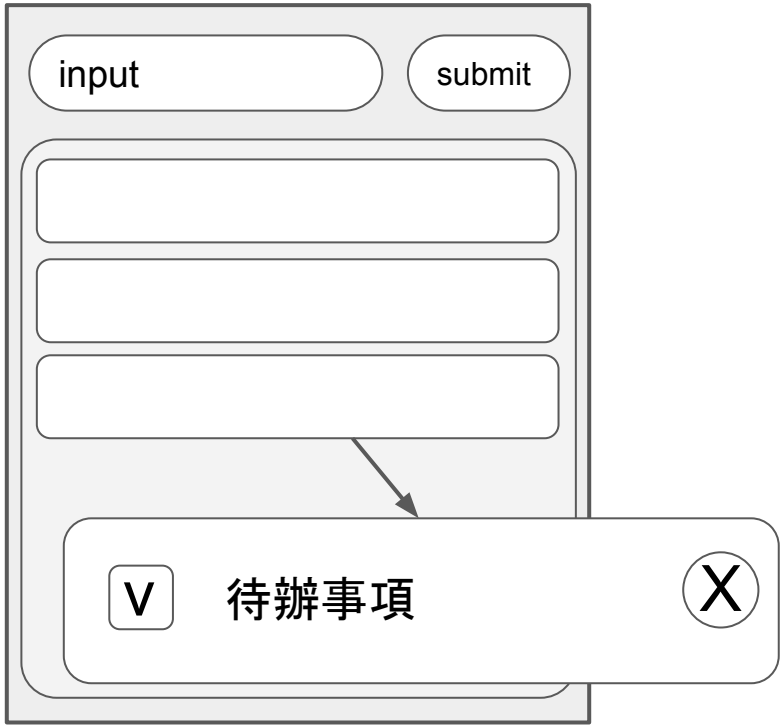
- 修改組件 state 不能直接賦值(assign)，而是必須用 setState() 才會觸發重新渲染組件
  - 例如：setState({ message: ‘Hello’, status: ‘success’})
  - setState() 會將傳入的物件合併入目前的 state
- State 除了擁有和可以設定它之外的組件任何組件都不能訪問它。
- 組件只能將自身 state 作為 props 傳遞給子組件才能讓其他組件獲取 state 資料，這種模式稱作上至下或單向資料流。
- setState 練習 左右兩邊按鈕點擊分別會 -1 或 +1，中間的數字顯示當前的數 值。

-1 0 +1

Render

- 回傳由 JSX 組成的 React Element 的函數
- Props 改變會觸發 render
- render() 必須回傳只有一個根節點的 React Element，否則在編譯時就會出現 Syntax Error: Adjacent JSX elements must be wrapped in an enclosing tag.
- 用迴圈 render 陣列資料，回傳的子元素必須給予 id，否則會出現 Warning: Each child in a list should have a unique "key" prop.

實作：Todo List



功能描述：

- 在輸入框輸入文字，按下 submit 後在下方新增一個待辦事項，並且清除文字框 內容。
- 勾選待辦事項會改變待辦事項的狀態
- 點X會刪除該待辦事項

組件：

- 包裹所有組件的組件
- 文字輸入框
- 送出按鈕
- 待辦事項清單
- 待辦事項

需要儲存的資料

- 文字輸入框: string
- 待辦事項清單: array of object
- 待辦事項: object
  - content: string
  - isCompleted: boolean

事件處理

- 文字框輸入文字時儲存輸入的內容
- 按下 submit 時取出儲存的輸入內容，新增到待辦事項清單內，並清空輸入框
- 點待辦事項的 checkbox 改變代辦事項的 isChecked 狀態
- 點刪除按鈕，把該待辦事項從待辦事項清單內移除

- 決定資料夾架構
  - 一般來說會在 src 下新增一個名為 components 的資料夾來放組件
  - 如果專案架構複雜的話也可能會以 “頁面”為單位新增資料夾放置組件
- 將 UI 拆解成各別的組件，建立需要的組件檔案
  - 組件最好能拆解到一個組件只負責做一件事
- 分別刻出組件UI，這時候先不考慮資料與互動性
- 找出應用程式需要的 State
  - 這個資料是從 parent 透過 props 傳下來的嗎？如果是的話，那它很可能不是 state。
  - 這個資料是否一直保持不變呢？如果是的話，那它很可能不是 state。
  - 是否可以根據你的 component 中其他的 state 或 prop 來計算這個資料呢？如果是的話，那它一定不是 state

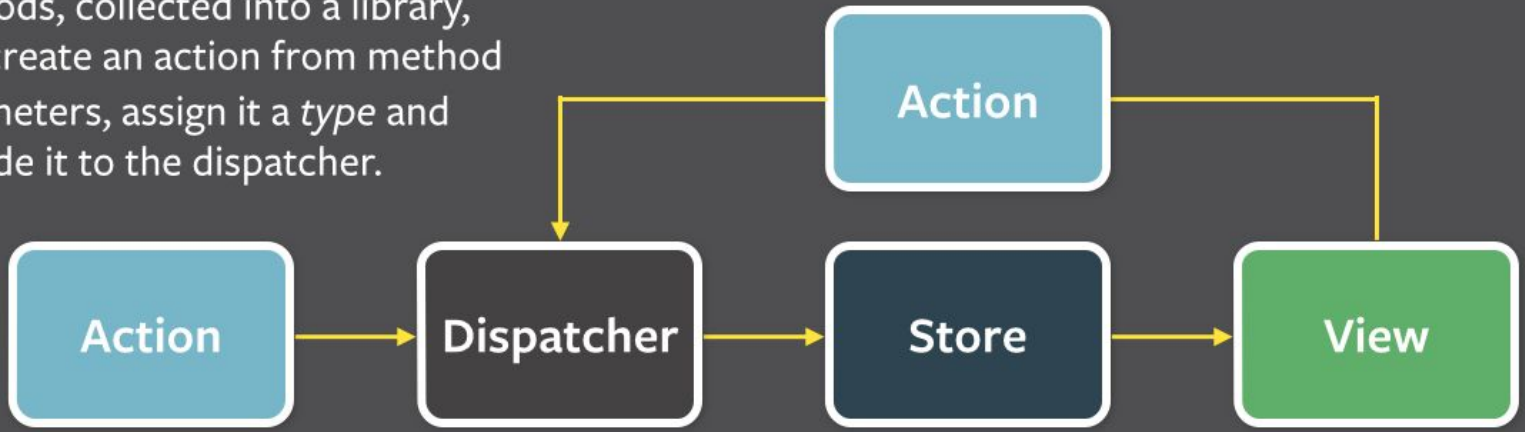
- 決定哪個組件會擁有這些 state
  - 找出哪些組件需要 state 來 render 畫面
  - 因為 React 的單向資料流核心，state 存放在共同需要 state 的組件之中層級最高的組件，state 才能以 props 的方式傳向其他也用到 state 的組件
- 為組件加上行為
  - 將 setState 作為 props 傳向下層組件，下層組件呼叫傳入的函數以改變上層函數擁有的 state
- 將所有組件匯入頂層組件組成完整的 UI

React 的好夥伴-- Redux

什麼是 Redux?

- Redux 是個給 JavaScript 應用程式所使用的可預測狀態容器
- 啟發於 Flux 設計概念
- 三大原則
  - 單一數據流
  - Store 不可直接修改(read-only)，而是透過觸發 Action 來改變
  - 純函數 Reducer 根據觸發的 Action 與當前的 Store 進行計算，並且回傳一個新的 Store 物件

Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.

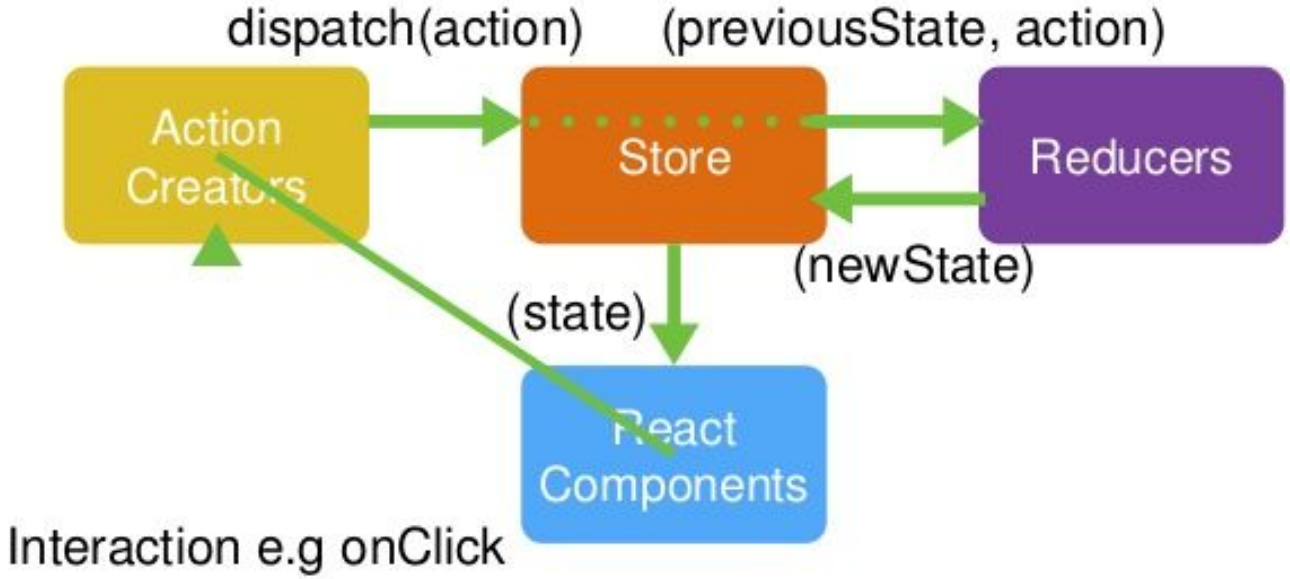


Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

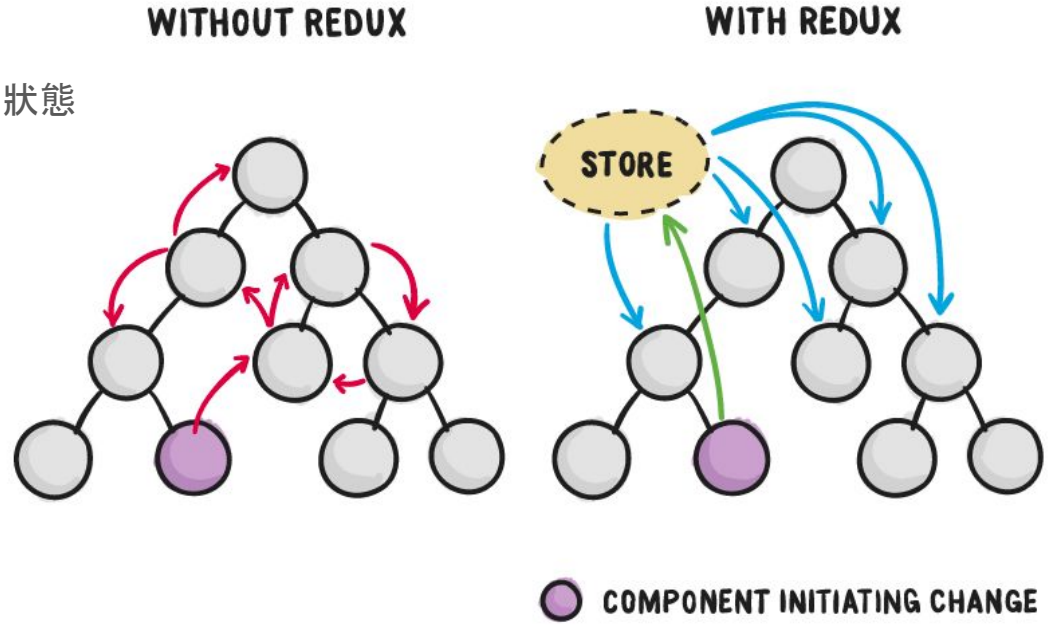
Redux Flow


dispatch(action)
(previousState, action)
Action Creators
Store
Reducers
(newState)
(state)
React Components
Interaction e.g onClick

什麼情況下可能需要 Redux

- 以應用程式面來看
  - 用戶的使用方式複雜
  - 不同身份的用 戶有不同的使用方式(比如普通用 戶和管理員)
  - 多個用戶之間可以協作
  - 與 server 大量互動，或者使用了 WebSocket
  - View 要從多個來源獲取資料
  - 大量互動、多個資料來源

- 從組件角度來看
  - 某個組件的狀態，需要共享
  - 某個狀態需要在任何地方都可以拿到
  - 一個組件需要改變全局狀態
  - 一個組件需要改變另一個組件的狀態



Action

- JavaScript 物件
- 必須含有 type 屬性，Reducer 會根據 Action type 做 State 處理
  - type 定義與值通常是大寫，代表為常數
  - 在 Redux DevTools 裡 Action 會被序列化列出，可以用來觀察與偵錯
- Action 可以攜帶 payload，payload 可以是除了函數以外的任意型態

```
// 定義 Action type
const SEND_MESSAGE = ‘SEND_MESSAGE’;
// 一個根據傳入參數回傳 action 物件的函數稱為 action creator
const sayHello = (message) => ({
    type: SEND_MESSAGE,
    message: message
})
```

Reducer

- JavaScript 純函數(no side-effect)
  - 除了回傳一個 值以外什麼也不做, 對系統其他部分沒有任何副作用
  - 不會發起 Http Request
  - 返回值完全取決於傳入參數
  - 傳入參數不能被修改
  - 傳入相同的參數則每次執行的回傳 值必定相同
- 接收當前 State 與 Action, 根據 Action type 做處理並且回傳新的 State
- 負責資料邏輯處理

```
// 必須定義一個初始 State 給 createStore() 使用
const initState = { message: null }

function reducer(state = initState, action) {
    switch(action.type) {
        case: SEND_MESSAGE:
            return {
              ...state,
              message: action.message
            }
        default:
            return state
    }
}
```

Store

- 用 Redux.createStore() 建立一個 Store，傳入所有 Reducer 做為參數
- Store 被建立時，會呼叫所有 Reducer，並使用 Reducer 的回傳值作為初始 State
- Store 常用的 API
  - store.dispatch(action) 發送一個 Action，這是唯一驅動 State 變更的方式
  - store.getState() 回傳當前的 State

React 搭配 Redux 使用

- 安裝 redux 與 react-redux 兩個套件
  - npm install redux react-redux
- 在 src 內依需求新增 reducer 和 actions 檔案
- 在 render DOM 的檔案中
  - import 所有 reducer，如果有多個 reducer 可以用 redux 提供的 combineReducers() 將所有 reducer 回傳的 state 合併為一個物件
  - 使用 redux 提供的 createStore() 傳入 reducers 建立 store，第二個參數傳入 window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__() 才能在瀏覽器中使用 Redux DevTools
  - 使用 react-redux 提供的 Provider 組件包裹整個應用程式，才能進行 store state 與組件綁定的動作，Provider 接受 store 作為傳入參數
- 接下來，在需要取用 state 的組件內使用 react-redux 提供的 connect() 就能獲取 state

combineReducers(), createStore()

定義一個函數回傳要從 store 中取得傳遞給組件的資料(mapStateToProps)

todoReducer

```
{
    todos: {}
}
```

userReducer

```
{
    firstname: ‘’,
    lastname: ‘’,
    avatar: ‘’
}
```

```
const reducers =
combineReducers({
    todoData: todoReducer,
    userData: userReducer
})

createStore(reducers)
```

```
{
    todoData: {
      todos: {}
    },
    userData: {
        firstname: ‘’,
        lastname: ‘’,
        avatar: ‘’
    }
}
```

connect() 連結 React 組件與 Store 儲存的 state

- 使用 react-redux 提供的 connect() 高階組件(High Order Component)
- 使用 HOC 的目的是將通用的邏輯放在 HOC 中，變動的部分就由代入 Component 的 props 和 state 傳入即可。

- 傳入一個 React object，connect() 加上一些修飾後，傳出一個新的 React object。利用這個方法使組件取得 redux store state

- connect(mapStateToProps?, mapDispatchToProps?, mergeProps?, options?)

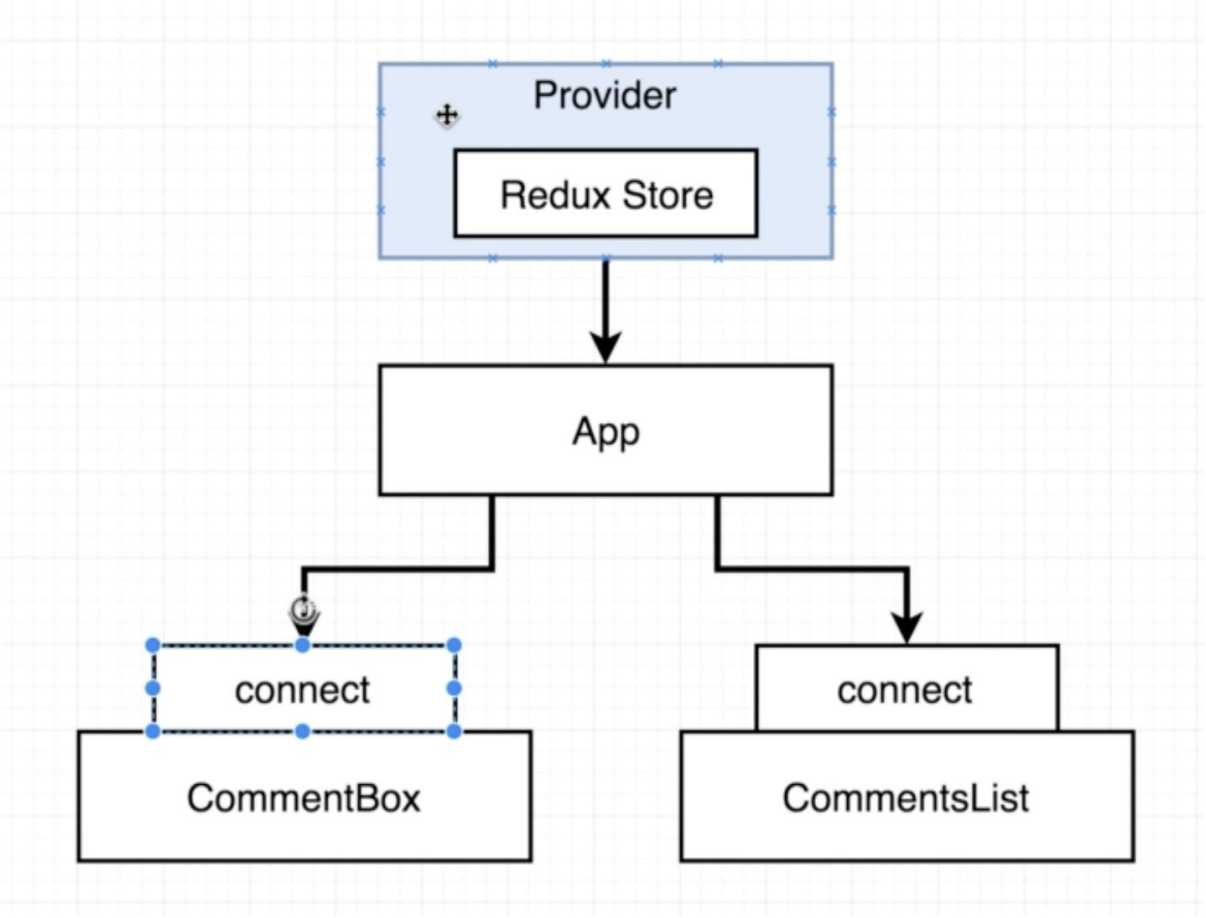
- mapStateToProps(state) 定義組件要訂閱哪些 state 的函數，回傳值為物件，這個物件會被合併到組件的 props 中
- mapDispatchToProps(dispatch) 預設傳入 store.dispatch 作為參數，這個函數主要目的是將 action creator 與 dispatch 綁定，回傳值為物件，含有綁定好 dispatch 的所有 action creator
- 如果沒有傳入 mapDispatchToProps() 的話會預設將 dispatch 注入到組件 props，需要手動 dispatch(actionCreator())
- Redux 提供一個 bindActionCreators() 的函數，可以一次傳入一包 actionCreators 的物件，自動綁定好 dispatch 以後注入到組件的 props

```
// TodoApp.js
…...
mapStateToProps = (state) => ({
    todoData: state.todoData
})

mapDispatchToProps = (dispatch) => ({
    ...bindActionCreators(actionCreators, dispatch),
    dispatch
})

export default connect(mapStateToProps, mapDispatchToProps)(TodoApp)
```

```
// TodoApp props
{
    todoData: {
       todos: {}
    },
    addTodo: fn,
    deleteTodo: fn
}
```


Provider
Redux Store
App
connect
CommentBox
connect
CommentsList

實作：將 Todo App 改成使用 Redux

- 新增一個資料夾放 Todo App 使用到的 reducer、actions 和組件
- 先從 action 著手，預想會有哪些動作，訂定好 action type 的常數、action creator 函數
- 接下來是 reducer，將本來儲存在組件 state 的資料拉出來，定義在 reducer 裡並且給予初始值
- 撰寫 reducer 核心的 function，先寫好所有 action type，處理的邏輯可以放著再回頭處理
- 接下來到 render 的入口點也就是 index.js 來建立 redux store
- store 建立起來並且用 Provider 提供給整個 app 以後就可以在組件裡使用 connect 取得 store state 以及 store.dispatch 方法了

非同步 Action

- 呼叫 API 就是一個常見的非同步行為
- 非同步兩個關鍵時間點
  - 呼叫
  - 得到回應
- 這兩個關鍵時間點通常都會發出改變 state 的 action
- 在呼叫後會發出通知 request 開始的 action
  - 可以訂立一個名稱叫 isLoading 或 isFetching 的 state，用來判斷畫面是否顯示 loading
- 在得到 response 後會發出 request 成功或者失敗兩種 action
  - 在 reducer 針對失敗或成功做不同的處理
- 標準做法是搭配 Redux Thunk Middleware 來非同步 aciton

中間層(Middleware)

- 在 dispatch action 和 action 到達 reducer 的時間點之間提供了一個第三方的擴充點。
- 可以使用 Redux middleware 來 log、回報當機、跟非同步 API 溝通、routing，還有其他更多的功能。
- npm install redux-thunk
- Redux thunk 用來包裝 action creator，原本的 action creator 回傳物件，經過 thunk 包裝的 action creator 回傳 function

Hooks

- React v16.8 推出的新功能
- 讓 function 組件也可以使用 class 組件才有的 state 或生命週期方法
- 完全向下兼容，可選擇性使用

- useState(initState) => setState()

- useEffect(fn, array) => componentDidMount()