使用react-redux綁定Redux與React應用

本章目標

- 學習Redux框架如何使用react-redux套用到React中
- 學習Redux框架中基本的檔案分隔方式
- 學習react-redux的基本知識：Provider元件、connect方法，以及綁定動作創立器

套用七步驟(原先的)

(1) 從redux模塊中匯入createStore函式

(2) 撰寫一個reducer(歸納)函式

(3) 由寫好的reducer，建立出store

(4) 撰寫一個render(渲染)函式，在狀態有更動時作重新呈現

(5) 第一次調用render函式，作初始呈現

(6) 訂閱render函式到store中

(7) 觸發事件時要呼叫store.dispatch(action)方法

react-redux 綁定器

改用綁定過的方法

react-redux是什麼？為何需要它？

- 由Redux官方發佈的專用於React應用的綁定工具
- 負責綁定Redux的store中的state到React最上層組件中，取代React原有的state的用法
- 提供各種便利的API，可以穿透各元件層依需求綁定資料/方法(store/action creators)
- 針對Redux套用於React應用性能優化

React元件中的state問題點

- state只能用於React元件之中，state只能用setState方法更動，屬於元件私有成員
- setState方法會合併與性能優化，有可能是「異步」的操作
- setState方法有可能會觸發不必要的渲染
- React元件無法對某些應用的狀態控管到，在生命週期方法中使用setState方法有一些限制與執行規則

使用react-redux之後

- Redux中store的state負責組織管理React的state與setState方法，可以不要使用React元件中state
- Redux中store的state，綁定到每個元件中的props
- Redux中Action Creators，綁定到每個元件中的props
- React元件中的state，稱為本地端的state，仍然可以視情況下使用 - 「本地端的state是合理的」

react-redux的Provider元件

Provider元件應用示例

```
// 使用react-redux的Provider元件作為React應用的最上層
ReactDOM.render(
  <Provider store={store}>
    <MyComponent />
  </Provider>,
  document.getElementById('root'))
```

Provider元件說明

- 必定是整個應用的最上層(外層)的元件
- 主要是為了之後使用connect方法來綁定React與Redux的store
- 使用了React中的Context(上下文)特性，可以”穿透"元件樹結構層級組件的傳遞資料，props不需要一層層傳遞

react-redux的connect方法

connet方法應用示例

```
// 匯入所有Action Creators程式碼檔模組
import * as actionCreators from './action'

// 將store中的items值綁到props上
const mapStateToProps = store => (
  { items: store.items }
)

// 連接Redux store
// 並把store.items綁到props.items，
// actionCreators裡面的方法也綁到props上
export default connect(mapStateToProps, actionCreators)(MyComponent)
```

回傳一個模組物件

函式
回傳一個物件，屬性名綁定到props，
屬性值綁定到store上的值

物件/函式
（对象中的函數會被拆分出來）

函式
State指的是Redux中store的state
Props指的React元件的props

高階元件(HOC)樣式說明

- 一種函式，目的是為了針對React元件的生命週期方法或是一些內部特性，作增強或調整。

```
// 需要state與生命週期方法時
const hoc = Component => class _hoc extends React.Component {
  render() {
    return <Component { ...this.props } {...this.state} />
  }
}

// 不需要state與生命週期方法時
const hoc = Component => function _hoc(props) {
  return <Component { ...props } />
}
```

connet方法應用示例(HOC)

```
// 匯入所有Action Creators程式碼內容
import * as actionCreators from ‘./action'

// 將store中的items值傳綁到props上
const mapStateToProps = store => (
  { items: store.items }
)

// 連接Redux store
// 並把store.items綁到props.items，
// actionCreators裡面的方法也綁到props上
export default connect(mapStateToProps, actionCreators)(MyComponent)
```

React元件

增強過的React元件

connect方法與connectAdvanced方法

- connectAdvanced方法是connect的基礎方法，connectAdvanced經過調整與加了預設參數後，成了connect方法
- connectAdvanced是一個高階元件(HOC)函式，樣版組件在程式碼檔案之中
- connectAdvanced方法具有高度自訂彈性，但不建議初學者使用

對Action Creators(動作建立器)的綁定

- 指的是connect方法的`mapDispatchToProps`傳參值
- 雖然名稱是「把Redux中store的dispatch綁定到元件的props上」，但也具有「把Action Creators函式直接綁定到組件的props上」的作用
- 三種傳參值的情況:
  - whenMapDispatchToPropsIsMissing – 沒給定
  - whenMapDispatchToPropsIsFunction – 是個函式
  - whenMapDispatchToPropsIsObject – 是個物件

mapDispatchToProps - 沒給定

- 「沒給定」代表要「直接把原本的dispatch方法綁定到props上」，所以這是一種「自動綁定」機制
- 要調用Redux的`store.dispatch(action)`，相當在組件中調用`this.props.dispatch(action)`
- 真實應用上，這樣用不太容易使用

```
// 連接Redux store
// 沒給定mapDispatchToProps，自動綁定dispatch到props上
export default connect(mapStateToProps)(MyComponent)
```

mapDispatchToProps - 是個物件

- 「物件」通常指的是Action Creator匯入的模組物件，也就是整個Action Creator中函式都匯入使用
- 真實應用時，不太會這樣用，因為除了在小應用上不會有元件用到所有的Action Creator

```
import * as actionCreators from './action'


// 呼叫時會直接呼叫到this.props.onItemAdd
export default connect(mapStateToProps，actionCreators)(MyComponent)
```

mapDispatchToProps - 是個函式

- 「函式」可以作部份綁定Action Creator中的函數。或要組合不同的Action Creator模組
- 這會用到Redux的一個`bindActionCreators`方法，要額外再匯入
- 真實應用時，大部份情況都是這樣使用

```
import { onItemAdd } from './action'

function mapDispatchToProps(dispatch) {
  return bindActionCreators({ onItemAdd }, dispatch)
}

// 呼叫時會直接呼叫到this.props.onItemAdd發送動作
export default connect(mapStateToProps，mapDispatchToProps)(MyComponent)
```