Redux與React應用整合後的程式碼重構

本章目標

- 學習React應用中如何去state化
- 學習Redux框架容器組件與呈現元件區分
- 學習Redux應用程式碼檔案如何組織與區分

React元件拆分

- 需考量重覆利用性，盡可能拆分到功能單一
- 需導入props的檢查機制，可以使用PropTypes或Flow工具輔助
- 元件程式碼檔案命名通常為`功能區塊+組件功能`:
  - 商品項目: ProductItem.js
  - 商品圖: ProductImage.js
  - 商品的集合元件: Product.js或ProductContainer.js

React元件 - refs用法示例

```
function CustomTextInput(props) {
  // textInput要先定義在這裡，回調才能參照到它
  let textInput = null

  return (
    <div>
      <input
        type="text"
        ref={(input) => { textInput = input }} />
    </div>
  )
}
```

React元件中不使用內部的state

- 對表單元件使用refs特性取得DOM元素參照，使用JS語言的事件處理方式來對表單元素取值或更動值
- 此種方式只能在 不可控制元件(Uncontrolled Components) 上使用
- 不使用state後的元件，可以只用函式型元件樣式
- 在單一、小的元件中可以使用。在複雜的元件可以合理的使用內部state

React元件區分

- 只是依Redux在組件上是否綁定的區別，方便在開發期間區分不同工作
- 其它相關的還有 類別/函式元件、有狀態/無狀態、純/不純元件等等區分方式
- 容器元件: 有使用到Redux的connect方法作綁定store/action creators的
- 呈現元件: 沒與Redux綁定，只用於資料呈現。資料由上層元件的props而來

Redux應用程式碼檔案組織

- 依照類型(by type): 依照actions, reducers, containers, components目錄區分，文檔名依功能或應用命名區分。最常見的一種方式。
- 依照功能(by feature): 先以功能或應用領域不同的區分，目錄里再依各自的reducer、action等等文件夾作類型區分。
- 鴨子(ducks): 是把reducers, constants, action types與actions打包成一個個模組來使用。可以減少很多目錄與檔案結構。

Redux應用程式碼檔案組織 - 依照類型

```
actions/
    CommandActions.js
    UserActions.js
components/
    Header.js
    User.js
    UserProfile.js
    UserAvatar.js
containers/
    App.js
    User.js
reducers/
    index.js
    user.js
routes.js
index.js
rootReducer.js
```

Redux應用程式碼檔案組織 - 依照功能

```
app/
    Header.js
    App.js
    rootReducer.js
    routes.js
product/
    Product.js
    ProductContainer.js
    ProductActions.js
    ProductList.js
    ProductItem.js
    productReducer.js
user/
    User.js
    UserContainer.js
    UserActions.js
    UserProfile.js
    userReducer.js
```

Redux應用程式碼檔案組織 - 鴨子(ducks)

- 鴨子(ducks)認為應用中的每個功能都有獨立性，會相互參照的情況很少，所以把reducers, selectors, actions等都放到同一檔案中，不用每次使用都要導出/匯入，開發時更為簡便。
- 模塊用了混合的輸出語法，要匯入模組要特別注意
- 現在也有另一種re-ducks方式，它會把每種類型的檔案拆分出來

ES6中import/export(預設成員)

```
// Module1.js
export default function myFunc() {}

// App.js
import defaultMember from './Module1'
```

只有「函數」、「類別」、表達式，可以使用預設導出

ES6中import/export(多個成員)

```
// Module1.js
export var myVar1 = ···;
export const MY_CONST = ···;

export function myFunc() {
    ···
}

// App.js
import { MY_CONST, myFunc } from './Module1.js'

// App2.js
import * as types from './Module1.js'
```

注意: import * 會忽略default的部份

ES6中import/export(重新導出)

```
// 全部重導出
export * from 'src/other_module';

// 部份重導出
export { foo, bar } from 'src/other_module';

// 部份重導出，並更動名稱
export { foo as myFoo, bar } from 'src/other_module';
```