Redux概念介紹

本章目標

- 學習Redux框架的基礎知識
- 學習Flux框架的基礎知識
- 學習副作用與純函式的概念

Redux的特色與優勢

Redux是什麼？

- 針對JavaScript應用的可預測狀態容器
- Redux = Flux + Elm
- 是為了React而設計的資料流框架，簡化與增強Flux的架構

針對JavaScript應用的可預測狀態容器

- 「JavaScript應用」 - Redux是泛用的框架，可用在React或其它的JS應用上
- 「可預測的」 - 應用了純函式的FP程式開發，可以作時光旅行除錯，重作/復原都可以達成
- 「狀態容器」 - Redux中用了store(存儲)作為容器記錄應用領域的狀態

Redux的優點與特色

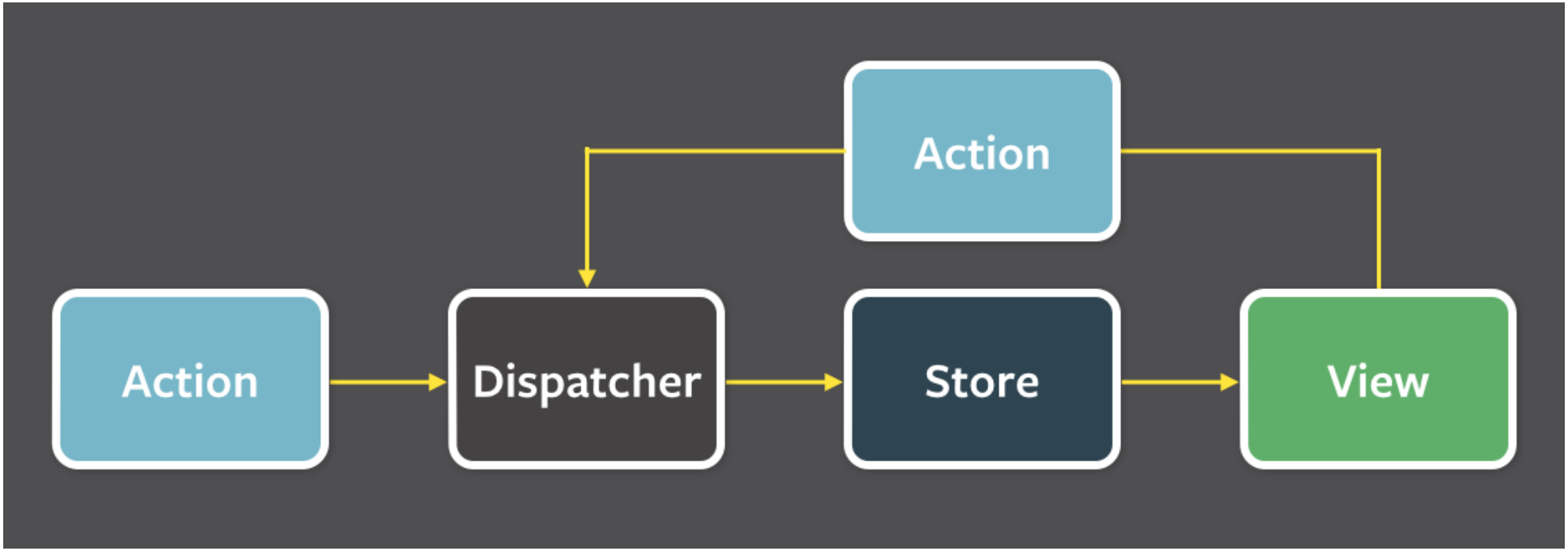
- 時光旅行除錯/熱重新加載 – 提供開發時的便利性
- 文件多、發展良好的生態圈 – 提供更多的學習與交流資源
- FP(函數式程式開發)與React搭配很理想 - 應用時下熱門的技術，Redux作者目前為React核心開發成員

為什麼要使用Redux

- 提供React/React Native應用領域的狀態組織與管控
- 取代原有React中的異步處理方式與setState可能異步執行的情況
- 應用規模化/協同開發下必需使用的資料流框架，可提供開發期間的除錯、測試便利性

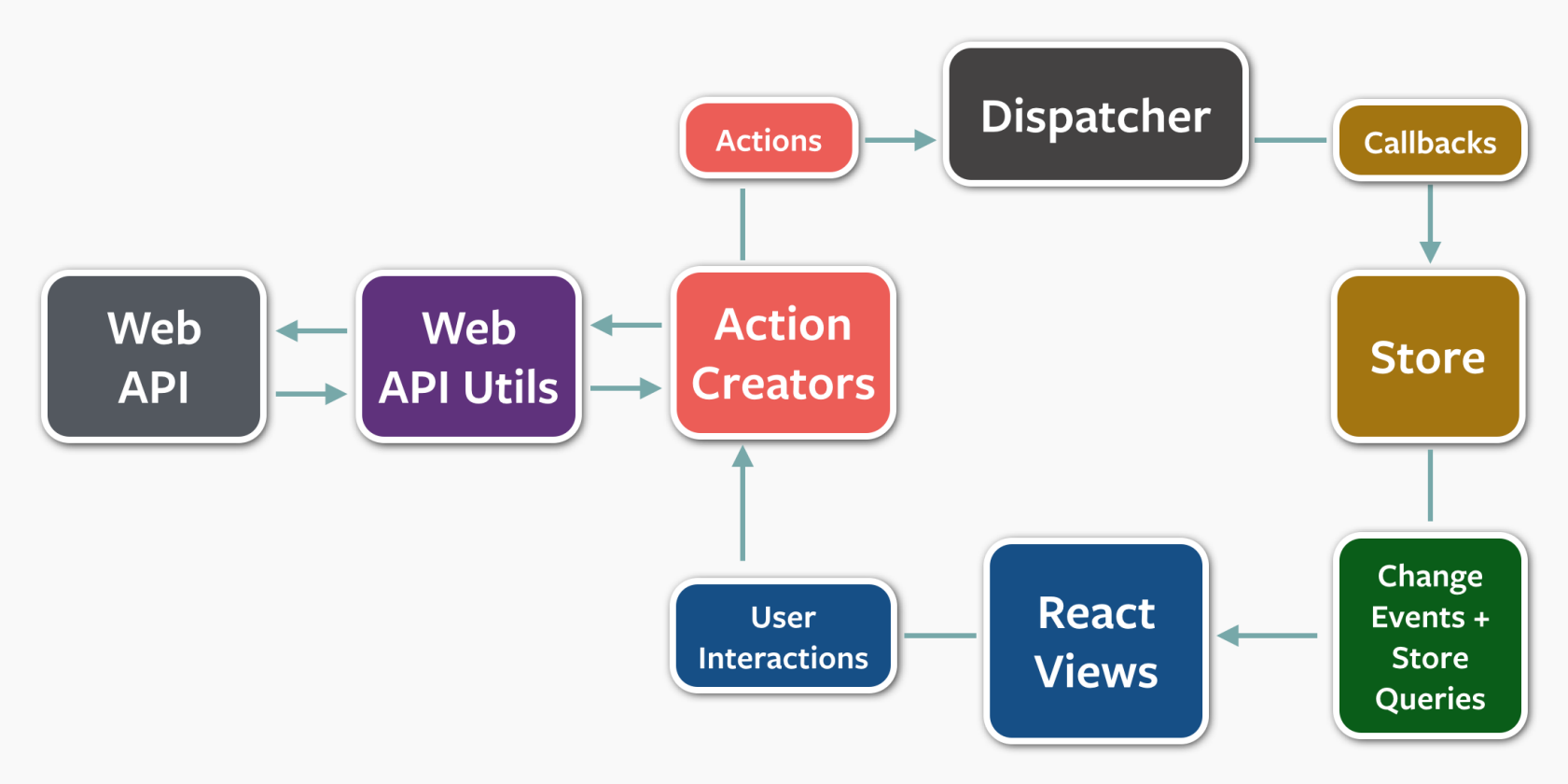
Flux概念

Flux架構圖



https://facebook.github.io/flux/docs/overview.html

Flux數據流程圖



https://facebook.github.io/react/blog/2014/07/30/flux-actions-and-the-dispatcher.html

Flux架構中的設計概念 & 術語

- store(存儲) - 類似於MVC中的M(模型)概念，記錄應用整體的狀態，並提供所需方法
- dispatcher(發送器) - 把接收到的actions(動作)，發送給所有註冊的callbacks(回調)，類似於pub-sub(發佈-訂閱)系統的設計

Flux架構中的設計概念 & 術語

- actions(動作) - 一個純物件，裡面會包含actionType(動作類型)與數據(通常稱為payload)
- Action Creator(動作建立器) - 輔助生成actions(動作)對象的函式
- payload(有效數據) - 指在數據傳輸時的”有效數據"，也就是不包含傳輸時的頭部資訊或metadata等其它用於傳輸的資料

Elm概念

Elm是什麼？

- 一個新式的程式語言，是完全純FP(函數式程式開發)語言，誕生於2012年
- 用於開發網站應用，最終編譯為JS語言代碼執行
- 為靜態/強類型語言，具有多種不同於JS的資料類型
- Elm-Architecture是包含在Elm的應用框架，它是單向資料流的架構

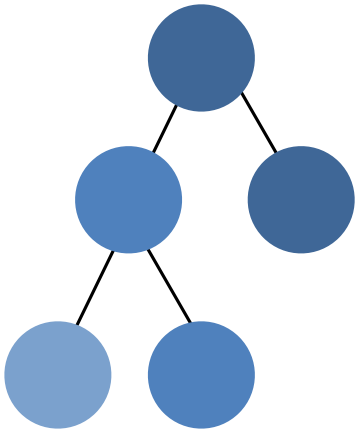
Redux學習自Elm

- 時光旅行除錯(Time Traveling Debugger)
- 單向資料流的樣式
- 「純函式」與”不可改變的值"的應用(FP)

Redux的三大原则

單一真相來源

- 你的整個應用中的state(狀態)，會儲存在單一個store(存儲)之中的一個物件樹狀結構里。



狀態是唯讀的

- 唯一能更動狀態的是發送一個`action`(動作)，`action`是一個描述”發生了什麼事"的純物件

```
store.dispatch({
  type: 'ADD_TODO',
  text: '學Redux'
})
```

更動只能由純函式來進行

- 要指示狀態樹要如何依actions(動作)來作改變，你需要撰寫純粹的歸納函式(reducers)

```
(previousState, action) => newState
```

(前一個/目前狀態，動作) => 新狀態

更動只能由純函式來進行

```
// @Reducer
// state(狀態)一開始的值是空陣列`state=[]`
function todoApp(state = [], action) {
  switch (action.type) {
    case 'ADD_ITEM':
      return [action.text, ...state]
    default:
      return state
  }
}

// @Store
// 由reducer建立store
const store = createStore(todoApp)
```

副作用 & 純函式概念

純函式是什麼？

- 當一個函式是純函式時，我們可以說`輸出`只取決於`輸入`

- 對於函式來說，具有副作用代表著可能會更動到外部環境，或是更動到傳入的參數值。函式的區分是以 純(pure)函式 與 不純(impure)函式 兩者來區分

純函數的條件

- 給定相同的輸入(傳入值)，一定會返回相同輸出值結果(返回值)
- 不會產生副作用
- 不依賴任何外部的狀態

純函式範例

純函式

```
const sum = function(value1, value2) {
  return value1 + value2
}
```

不純函式

```
let count = 1

let increaseAge = function(value) {
  return count += value
}
```

JS語言內建的純/不純函式

- console.log, alert - 會改變或依賴外部環境，為不純函式
- Math.random - 每次調用都會產生不同值，為不純函式
- 時間、I/O、Ajax、數據庫相關 – 全部都是不純函式

純函式的特殊優點

- 代碼閱讀性提高
- 較為封閉與固定，可重覆使用性高
- 輸出/輸入單純，易於測試、調試
- 可作緩存或記憶處理，在高花費的應用中可作提高運行性能的機制

副作用的程度差異

- 輕度/微量 – console.log/alert/Date/random
- **中度/通常 - Ajax/Fetch API/Timer**
- 重度 – Promise/Generator或其它多個同步與異步組合控制流程
- 隱性/不經意 – 值相等運算、隱性轉類型等等