

# Redux中介軟體(middleware)基礎

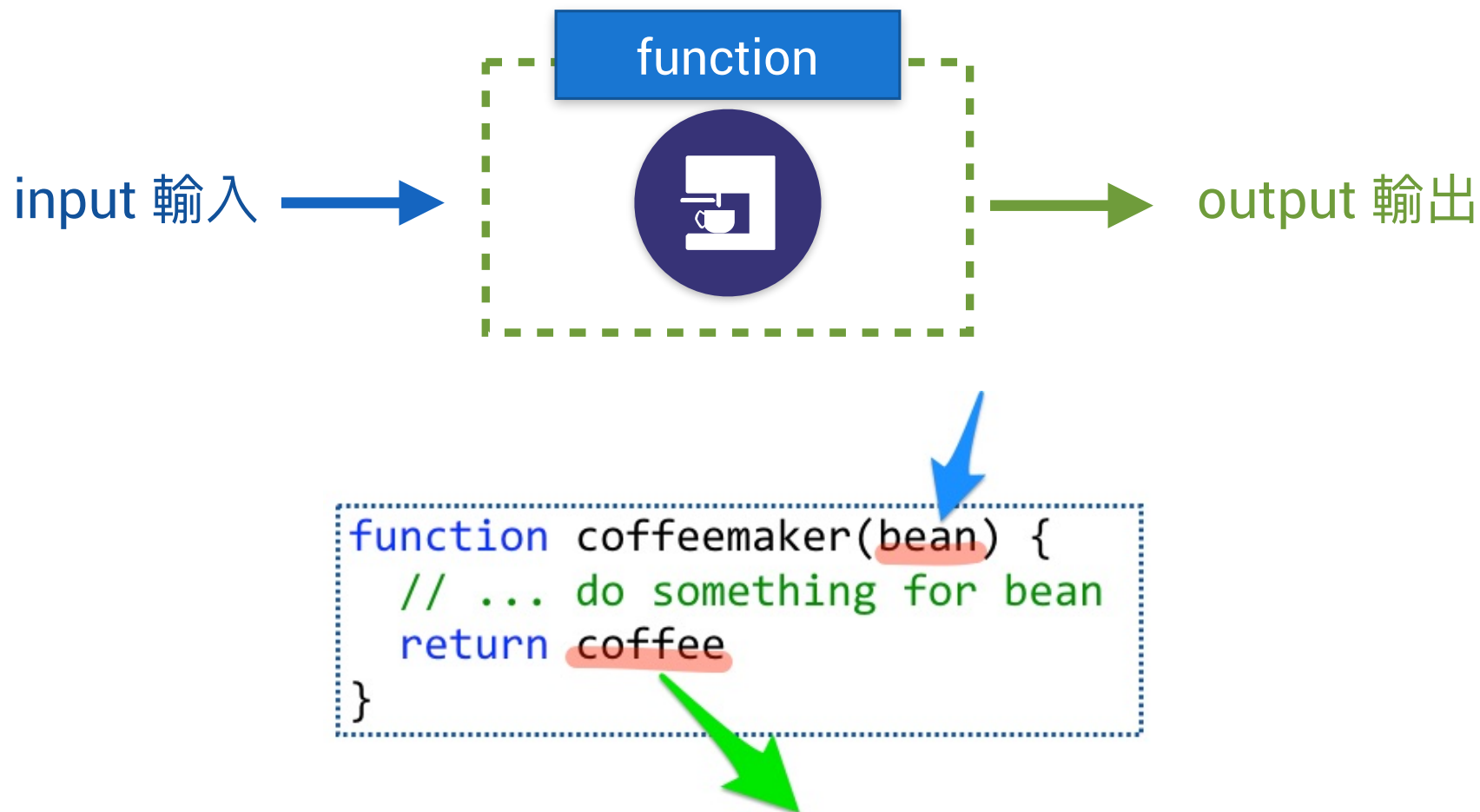
# 本章目標

- ◆ 學習高階函式與合成的樣式(中介軟體使用)
- ◆ 學習Redux中介軟體的使用目的
- ◆ 學習Redux應用中介軟體的運行原理

# Redux中介軟體(middleware)是什麼

- ◆ 中介軟體(Middleware)提供了第三方擴展的位置，它介於發送一個動作，到真正到達reducer之間。開發者們使用Redux中介軟體，可用於日誌記錄、崩潰報告或是異步API溝通、路由等等
- ◆ 中介軟體的運行需要與動作建立器搭配使用
- ◆ 中介軟體使用了JS語言中高階函式(HOF)的樣式
- ◆ 多個中介軟體可以組合成一個大的中介軟體，使用的是合成(compose)樣式

# JS語言中的函式



# 高階函式(High-order function)樣式

```
function oldDispatch(action) {  
  // ...作某些操作  
  console.log('发送动作！')  
  
  return 1  
}
```

dispatch函式的原本程式碼

```
// 高階函式，傳入一個函數，回傳另一個函式  
function hof1(dispatch) {  
  return function newDispatch(action) {  
  
    // 在dispatch前，作某些操作  
    console.log('发送动作 - 前')  
  
    let result = dispatch(action)  
  
    // 在dispatch後，作某些操作  
    console.log('发送动作 - 后')  
  
    return result  
  }  
}
```

# 高階函式樣式 – 語法簡化

```
// 高階函式，傳入一个函式，回传另一个函式
function hof1(dispatch) {
  return function newDispatch(action) {

    // 在dispatch前，作某些操作
    console.log('发送动作 - 前')

    let result = dispatch(action)

    // 在dispatch後，作某些操作
    console.log('发送动作 - 后')

    return result
  }
}
```

```
// 使用 箭頭函數 簡化語法
const hof1 = (dispatch) => {
  return (action) => {
    let result = dispatch(action)
    return result
  }
}
```

```
// 使用 箭頭函數 再簡化
const hof1 = (dispatch) => (action) => {
  let result = dispatch(action)
  return result
}
```

# 高階函式樣式 – 運行

```
const actionObj = {
  type: 'ADD_ITEM',
  payload: { id: 1, text: 'TEXT' }
}

function oldDispatch(action) {
  console.log('發送動作！')
}

function hof1(dispatch) {
  return function newDispatch(action) {
    console.log('發送動作 - 前')
    let result = dispatch(action)
    console.log('發送動作 - 后')
    return result
  }
}
```

```
// 先強化原本的oldDispatch函式
const newDispatch1 = hof1(oldDispatch)

// 運行新的dispatch函式
newDispatch1(actionObj)
```

```
// 簡化语法，直接運行hof1
hof1(oldDispatch)(actionObj)
```

# 高階函式樣式 – 其它應用情況

```
// 高階函式，傳入一個函式，異步運行傳入的函式
function hof2(dispatch) {
  return function newDispatch(action) {
    // 異步運行dispatch(action)，5秒後再運行
    setTimeout(() => { dispatch(action) }, 5000)
  }
}
```

```
// 高階函式，只是回傳原本的函式
function hof3(dispatch) {
  return function newDispatch(action) {
    // 啥事都不作，回傳原本的dispatch(action)
    return dispatch(action)
  }
}
```



# 高階函式合成樣式

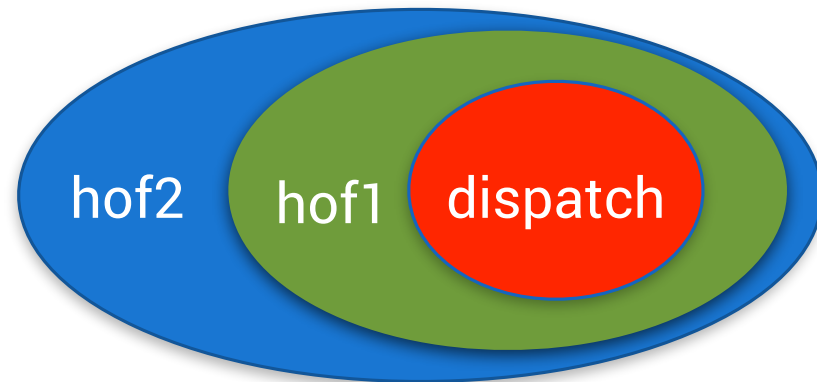
```
function hof1(dispatch) {  
  return function newDispatch(action) {  
    console.log('HOF1 log before dispatch!')  
    return dispatch(action)  
  }  
}  
  
function hof2(dispatch) {  
  return function newDispatch(action) {  
    console.log('HOF2 log before dispatch!')  
    return dispatch(action)  
  }  
}  
  
const newDispatch = hof2(hof1(dispatch))  
// 運行 newDispatch  
// 相當於hof2(hof1(dispatch))(action)  
newDispatch(action)
```

# 高階函式合成樣式 – 運行過程

```
// hof2(hof1(dispatch))(action)
// -> hof2 function 主體程式碼
//   console.log('HOF2...')
//   hof1(dispatch)(action)
// -> hof1 function 主體程式碼
//   console.log('HOF1...')
//   -> dispatch(action)
```

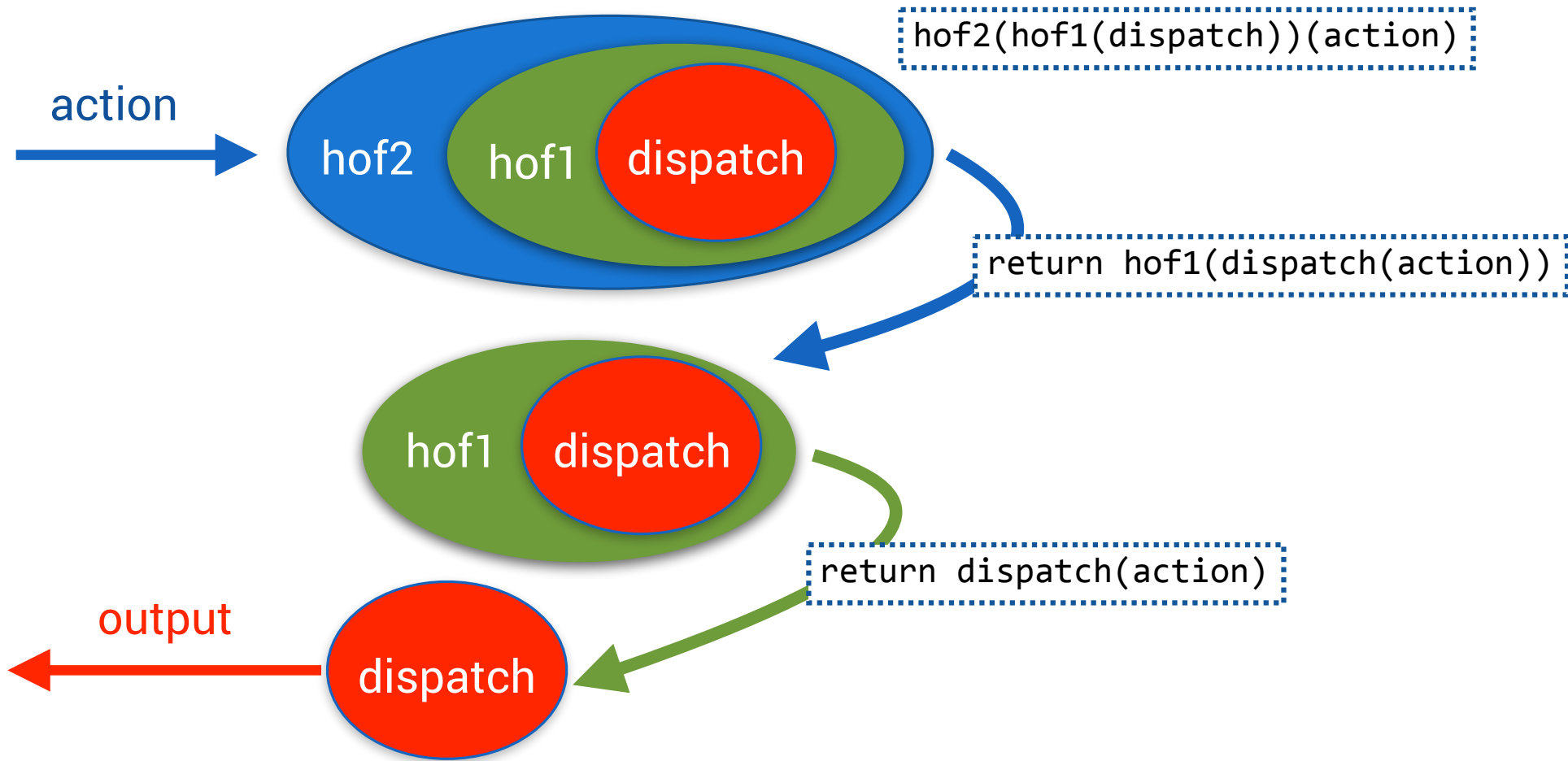
action →

hof2(hof1(dispatch))(action)




- ◆ 當合成(compose)時，愈內層的高階函式是愈早對dispatch強化的函式
- ◆ 當運行時，通常情況是從最外層的高階函式開始運行，一直到最內層的dispatch函式，也就是hof2 -> hof1 -> dispatch
- ◆ 一般情況下，如果各高階函式的作用是各自獨立的，位於外或內層不會影響最後結果
- ◆ Redux中的compose函式(合成用函式)會依照傳入的多個中間軟體次序，愈前面的包在愈外層，以此保證運行次序

# 高階函式合成樣式 – 運行過程



## Redux中的compose函式(合成函式)

```
export default function compose(...funcs) {  
  if (funcs.length === 0) {  
    return arg => arg  
  }  
  
  if (funcs.length === 1) {  
    return funcs[0]  
  }  
  
  return funcs.reduce((a, b) => (...args) => a(b(...args)))  
}
```



传入a(x), b(x)，最後以a(b(x))樣式回傳

# Redux中介軟體的樣式原型

```
// 新的高階函式，就是Redux中介軟體的原型樣式
function newHof1(store){
  return function newHof(dispatch){
    return function newDispatch(action) {
      console.log('HOF1 log before dispatch!')
      // 可在這裡獲取到目前的state狀態
      let state = store.getState()
      // 也可以調用原本的dispatch函式，注意：這會跳出整個合成函式的運行
      return store.dispatch(action)
      // 往其它的合成函式(中介軟體)運行
      return dispatch(action)
    }
  }
}

// 這兩者相等
const hof1 = newHof1(store)
```

# Redux中介軟體解說

- Redux用中介軟體(middleware)來代表上述的高階函式(HOF)的名詞，中介軟體的主要目的是**為了強化store.dispatch方法**
- 為了區別，在中介軟體里的程式碼主體區塊的有三個傳參可用，分別是store, dispatch, action，為了區別出這裡的dispatch已經被強化過，所以**改用next取代dispatch這個傳參名**
- 以上述的代碼來看對hof2函式來說，next代表的就是`hof1(dispatch)`，對hof1函式來說，next代表就是`dispatch`
- 如果在hof2中呼叫(或回傳)store.dispatch(action)，就是不用傳參而已，會讓內層的高階函式不被調用，因此會跳出整個合成函式往內層函式的運行

# Redux中介軟體解說

- 中介軟體通常需要搭配動作創建器(action creator)來運行，此時會看到非純對象的action，這只是搭配中介軟體應用，按原本定義action對象仍然是要純對象
- 中介軟體里傳入的store，並不是完全原本的Redux中的store對象，這個從原本的store對象拷貝而來，它只有部份的方法和資料，包含必要的dispatch與getState方法
- 初學者可以先看懂目前的示例程式碼，之後要學習著改用ES6箭頭函數簡化語法的程式碼

## Redux中介軟體結論

- ◆ 中介軟體(Middleware)應用了 **高階函式(HOF)**與**合成(compose)**的語法樣式，目的是在強化 `store.dispatch`方法
- ◆ 多個中介軟體可以用合成樣式組合成一個大的中介軟體(用**`applyMiddleware`方法**)，但基本來說，中介軟體彼此的作用要獨立
- ◆ 中介軟體也具有改變運流程的能力，可作程序中的流程管控