Redux Compose

函數的組成/組合

Justin @Migo 2016 Apr.

用函數來表達 (1+2) x 3 - 4

```
var a = 1 + 2

var b = a x 3

var c = b - 4
```

稍微改變一下寫法...

```
const add2 = (x) =>  x + 2 ;

const mult3 = (x) =>  x * 3 ;

const sub4 = (x) =>  x - 4 ;
```

函數會變成...

```
let ans = sub4(multi3(add2 (1) ))
```

好像可以寫得更好讀...

compose 來處理一下

```
let f = compose(multi3, add2)
f(1);
```

compose 來處理一下

```
let f = compose(multi3, add2)
f(1);
```

```
const compose = (f, g) => {
    return (x) => f(g(x));
};
```

先處理g(x) 然後把結果給f() 處理

Demo

jsbin.com/xifuge/edit?js,console

Todo

jsbin.com/quhofa/edit?js,console

compose 小結

```
f(g(x))  ===>  compose(f,g)
```

問題來了如果有多個函數 f,g,h,i... 怎麼處理

compose 小結

```
h(g(h(i(x)))) ===>
compose(h,compose(g,compose(h,i)))
```

!!!這樣寫也很醜...!!!

compose 小結

```
compose(f,g,h,i)
```

如果這樣改...!?

return hell


var asyncJavaScript = function(err, callback) {
callback(function(err, callback) {
callback(function(err, callback) {
callback(function(err, callback) {
callback(function(err, callback) {
callback(function(err, callback) {
callback(function(err, callback) {
callback(function(err, callback) {
console.error('CALLBACK HELL');
});
});
});
});
});
});
});
});
WTF

兩種解法

1. reduce()
2. for loop

reduceRight

- 用法同reduce
- 與reduce()的執行方向相反

reduceRight

Syntax

```
arr.reduceRight(callback[, initialValue])
```

```
[0, 1, 2, 3, 4].reduceRight(function(previousValue, currentValue, index, array){
  return previousValue + currentValue;
});
```

The callback would be invoked four times, with the arguments and return values in each call being as follows:

	previousValue	currentValue	index	array	return value
first call	4	3	3	[0, 1, 2, 3, 4]	7
second call	7	2	2	[0, 1, 2, 3, 4]	9
third call	9	1	1	[0, 1, 2, 3, 4]	10
fourth call	10	0	0	[0, 1, 2, 3, 4]	10

可能可以這樣寫...

```
const funcArray = [fun1,fun2,fun3..];

funcArray.reduceRight(
  (prev, curr) => curr(prev)
);
```

ES6 / Babel

```
const compose = (...funcs) => {
  return () => {
      funcs.reduceRight((prev, curr) => {
        console.log('curr function:', curr);
        console.log('prev function:', prev);
        return curr(prev);
      });
  };
};
const add2 = (x) => ( x + 2 );
const mutiply = (x) => ( x * 3 );

const f = compose(mutiply, add2);

console.log(f(1));

//expect : 9
//result : undefined
```

Console

```
"curr function"
function mutiply(x) {
return x * 3;
}
"prev function"
function add2(x) {
return x + 2;
}
undefined
```

Notice: 需要處理 initValue

錯誤demo: http://jsbin.com/sinuze/edit?js,console

reduceRight

Syntax

```
arr.reduceRight(callback[, initialValue])
```

先處理 initValue

```
const compose = (...funcs) => {
    const initfun = funcs[funcs.length - 1] /*最右邊的func*/
    const rest    = funcs.slice(0, -1)      /*除了最右邊剩下的func*/
    return (...args) => {
      const initValue = initfun(...args)    /*先拿到initValue*/
      return rest.reduceRight((prev, curr) => curr(prev), initValue);
    }
}
```

```
const add2 = (x) => ( x + 2 );
const mutiply = (x) => ( x * 3 );

let f = compose(mutiply, add2);

console.log(f(1));

//expect: 9
//result: 9
```

Compose 傳入的function

func1	func2
x + 2	x * 3

demo: http://jsbin.com/wisupif/edit?js,console

先處理 initValue

```
const compose = (...funcs) => {
    const initfun = funcs[funcs.length - 1] /*最右邊的func*/
    const rest    = funcs.slice(0, -1)      /*除了最右邊剩下的func*/
    return (...args) => {
      const initValue = initfun(...args)    /*先拿到initValue*/
      return rest.reduceRight((prev, curr) => curr(prev), initValue);
    }
}
```

```
const add2 = (x) => ( x + 2 );
const mutiply = (x) => ( x * 3 );

let f = compose(mutiply, add2);

console.log(f(1));

//expect: 9
//result: 9
```

ReduceRight

prev	curr	return
3	x * 3	9

demo: http://jsbin.com/wisupif/edit?js,console

多加入一個function

```
const sub4 = (x) =>  x - 4 ;
```

Compose 傳入的function

func1	func2	func3
x + 2	x * 3	x - 4

ReduceRight

prev	curr	return
3	x * 3	9
9	x - 4	5

Todo: jsbin.com/maruko/edit?js,console

來看看這段Code

```
export default function compose(...funcs) {
  if (funcs.length === 0) {
    return arg => arg
  } else {
    const last = funcs[funcs.length - 1]
    const rest = funcs.slice(0, -1)
    return (...args) => rest.reduceRight((composed, f) => f(composed), last(...args))
  }
}
```

其實這就是Redux Compose 的寫法

for loop的解法

```
var add1 = function(x) {return x + 1;};
var mult2 = function(x) {return x * 2;};
var square = function(x) {return x * x;};
var negate = function(x) {return -x;};


var compose = function() {
    var funcs = arguments;
    return function() {
        var args = arguments;
        for (var i = funcs.length; i --> 0;) {
            args = [funcs[i].apply(this, args)];
        }
        return args[0];
    };
};

var f = compose(negate, square, mult2, add1);
console.log(f(2));
```

demo: http://jsbin.com/zeqero/edit?js,console