Higher-order function

Jichao Ouyang

October 25, 2015

Contents

1 (Higher-order function) 1

1.1 Higher-order function 2

1.1.1 2

1.1.2 3

1.2 currying 5

1.2.1 6

1.2.2 7

1.2.3 8

1.3 function composition 9

1.3.1 Compose 9

1.3.2 pipe 11

1 (Higher-order function)

, ? .

. : `f(x, y) = x(y)` lambda

```
f(x) => (y -> x(y))
(y) => x(y)
```

f x.

, . .

, , .

map, reduce. .

, .

. .

1.1 Higher-order function

JavaScript , JavaScript , .

1.1.1

, sort

```
[1,3,2,5,4].sort( (x, y) => x - y )
```

, x y ,, , id :

```
[{id:1, name:'one'},
 {id:3, name:'three'},
 {id:2, name:'two'},
 {id:5, name:'five'},
 {id:4, name:'four'}].sort((x,y) => x.id - y.id)
```

.

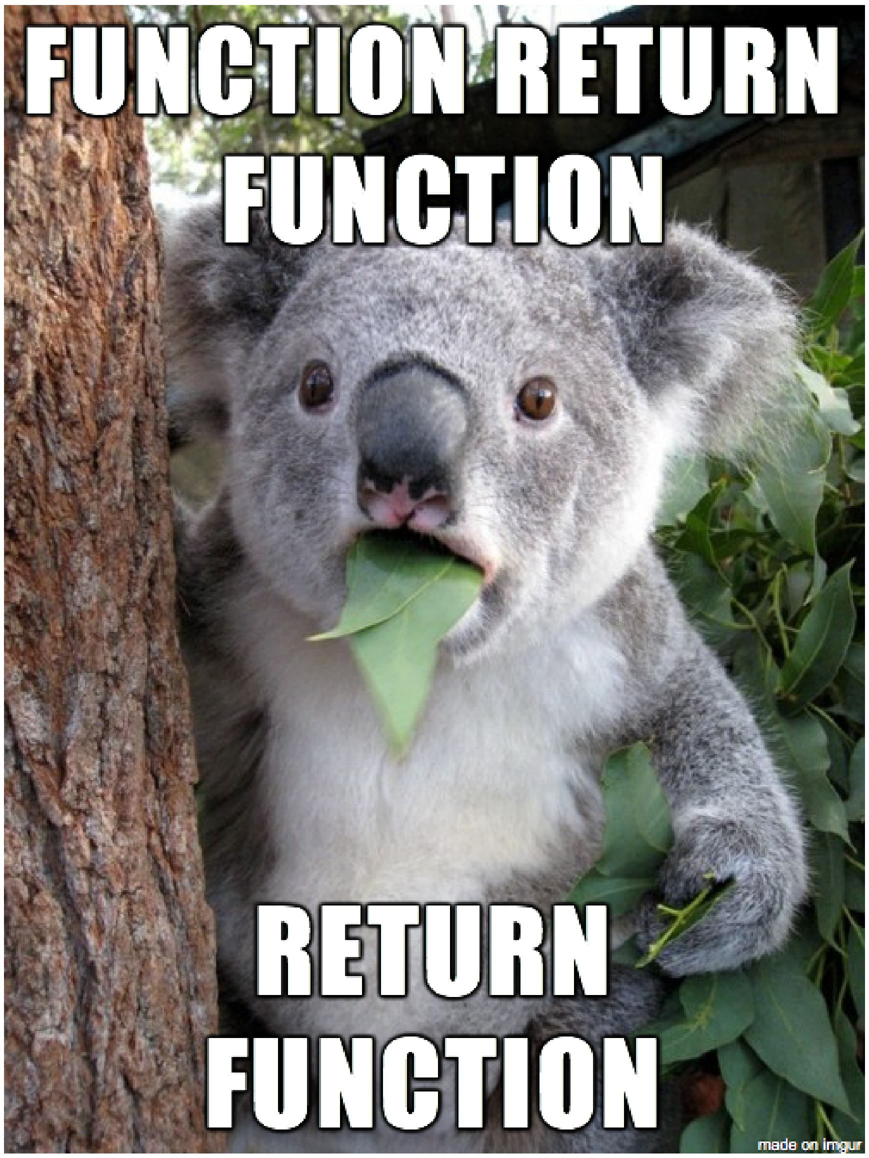
1.1.2

, , Eweda `aliasFor`, E :

> , , JavaScript `function` , Firefox console `console.log`. , `typeof console.log` function

```
var E = () => {}
var aliasFor = oldName => {
    var fn = newName => {
      E[newName] = E[oldName];
      return fn;
    };
    return (fn.is = fn.are = fn.and = fn);
};
```

`return`, `fn` , `aliasFor` `fn`, `fn` `fn.is` `fn.are`... ? `fn` `fn`. `fn() => fn`, `fn()()=>fn()=>fn` ..., fn , fn.



, `fn` (side affect) `E[newName]=E[oldName]`, E , fn `E` . `aliasFor` `fn` , chain :

```
aliasFor('reduce').is('reduceLeft').is('foldl')
```

, , , . .

1.2 currying

Haskell Curry

, Curry Haskell, Haskell , . , . Haskell , .

Haskell .

```
max 3 4
(max 3) 4
```

4, .

, . ? Haskell, JavaScript .

1.2.1

1. , , f(['1','2']) => '12'

   ,, ? `reduce`

   ```
   var concatArray = function(chars){
     return chars.reduce(function(a, b){
       return a.concat(b);
     });
   }
   concat(['1','2','3']) // => '123'
   ```

   ,.

2. 1,

   ```
   var concatArray = function(chars, inc){
     return chars.map(function(char){
       return (+char)+inc + '';
     }).reduce(function(a,b){
         return a.concat(b)
     });
   }
   console.log(concatArray(['1','2','3'], 1))// => '234'
   ```

3. 2,

   ```
   var multiple = function(a, b){
     return +a*b + ''
   }
   var concatArray = function(chars, inc){
     return chars.map(function(char){
   ```

```
        return multiple(char, inc);
      }).reduce(function(a,b){
          return a.concat(b)
      });
    }
    console.log(concatArray(['1','2','3'], 2)) // => '246'
```

? 2,. `map` multiple . `concatArray` , ? ? .

1.2.2

```
var multiple = function(a){
  return function(b){
    return +b*a + ''
  }
}

var plus = function(a){
  return function(b){
    return (+b)+a + ''
  }
}
var concatArray = function(chars, stylishChar){
  return chars.map(stylishChar)
    .reduce(function(a,b){
      return a.concat(b)
  });
}
console.log(concatArray(['1','2','3'], multiple(2)))
console.log(concatArray(['1','2','3'], plus(2)))
```

1. , 2. , , ,

```
concatArray(['1','2','3'], multiple(2))
```

map

```
chars.map(stylishChar)
```

,

, , , , , .

1.2.3

Haskell , :

```
max 3 4
```

```
(max 3) 4
```

`max` `max 3`

```
ghci> :t max
max :: Ord a => a -> a -> a
```

, `Ord a =>` , `max` : `a`, `a -> a`, `max 3`

```
ghci> :t max 3
(Num a, Ord a) => a -> a
```

`a` `Ord` `Num`, `max 3` , `Ord` `Num` `Ord` `Num`.
, Haskell , , . .
Javascript() , , `undefined`, .

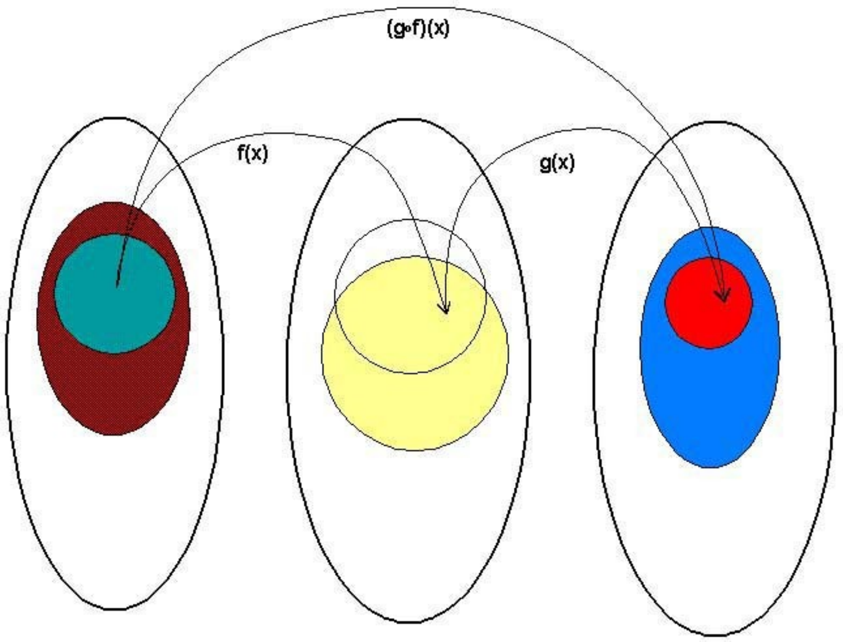
```
function willNotCurry(a, b, c) {
    console.log(a, b, c)
    return a*b-c;
}
willNotCurry(1)
// => NaN
// => 1 undefined undefined
```

eweda,

```
var multiple = curry(function(a, b){
  return +b*a + ''
})
var plus = curry(function(a, b){
  return (+b)+a + ''
})
```

1.3 function composition

, map, fold , . map fold reverse .
, , .



Catgory TheoryFuntor , AB fB Cg `(g.f)(x)` `f` `g` `g(f(x))` A C map `reverse.fold`.

1.3.1 Compose

Eweda compose

```
var gf = E.compose(f, g)
```

, , Eweda/Ramda Underscore .
tasks `completed` `true` , `id` .
underscore :

```
_(tasks)
    .chain()
    .filter( task => task.completed===true)
    .sortBy( task => task.id)
    .value();
```

, / jquery , ,

```
_.sortBy(_.filter(tasks, task => task.completed===true), task => task.id)
```

, , underscore. underscore
Eweda/Ramda :

```
compose(sortBy(task=>task.id), filter(task=>task.completed===true))(tasks)
```

? compose ?
, `tasks` `E.compose()` ? filter . , , , . underscore `_.sortBy(_.filter())`

. tasks groupedTasks, completed true id .

```
groupedTasks = [
  [{completed:false, id:1},{completed:true, id:2}],
  [{completed:false, id:4},{completed:true, id:3}]
]
```

underscore:

```
_.map(groupedTasks,
   tasks => _.sortBy(_.filter(tasks, task => task.completed===true), task => task.id))
```

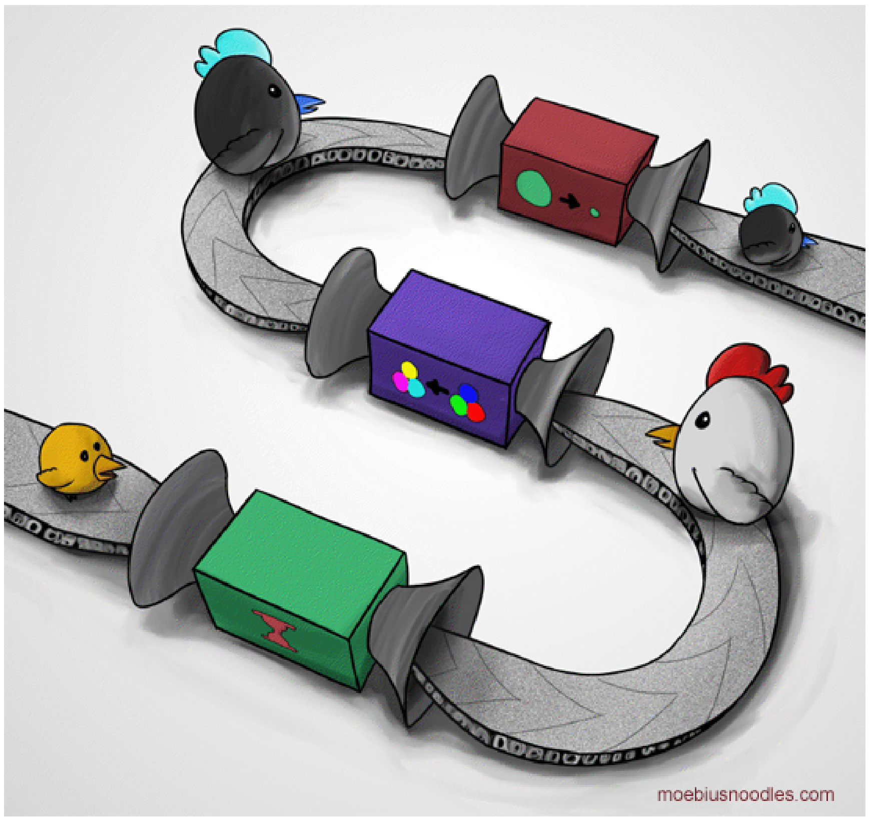
`_.sortBy(_.filter())` map underscore

```
function completedAndSorted(tasks){
  return _.sortBy(_.filter(tasks, task => task.completed===true), task => task.id))
}
_.map(groupedTasks, completedAndSorted)
```

underscore `_.compose` underscore `compose` filter sortBy

```
var completedAndSorted = compose(sortBy(task=>task.id),
                                 filter(task=>task.completed===true))
map(completedAndSorted, groupedTasks)
```

.
Eweda/Ramda ,, , . .



underscore , chaining Flow-Base programming Monad, , .

1.3.2 pipe

compose, eweda/ramda pipe, pipe compose . `pipe(f, g)`, `f` , `g`, bash pipe

```
find / | grep porno
```

```
    pipe(find, grep(porno))(/)
```

,. ().
underscore

```
_(data)
  .chain()
  .map(data1,fn1)
  .filter(data2, fn2)
  .value()
```

pipe

```
pipe(
  map(fn1),
  filter(fn2)
)(data)
```