# 函数式编程在 JavaScript 中的应用与研究

郝思红 李 林
四川大学锦城学院计算机与软件学院 四川 成都 611731

【摘要】传统的 JavaScript 开发主要运用的是面向对象的编程模式，又在其中穿插了一些函数式编程的思想。在最新的 ES6 的标准中，函数式编程在前端开发中的运用越来越多，本文介绍了函数式编程在 JavaScript 中的几个主要特性、应用，以及函数式编程的优缺点。

【关键词】函数式编程；闭包；偏函数

## 1 引言

函数式编程历史已经很久远了，但以前一直被认为只是理论性的概念，不能运用到实际的生产环境中去。但是最近几年函数式编程又开始广泛地出现在各种计算机编程语言中，作为前端开发重要的语言 JavaScript 自然也要加入使用函数式编程的队伍中。

函数式编程是一种编程模式，它将计算机运算视为数学上的函数运算[1]。函数式编程有一个很重要理论基础就是 lambda 演算，而 lambda 演算的重点是：函数也可以被当作参数进行运算。

## 2 JavaScript 在函数式编程语言中的特性

正如 JS 在面向对象编程中有三大特性继承、多态和封装[2]。JS 在函数式编程模式中也有相应的特性：函数是一等值、柯里化、高阶函数、闭包、惰性求值、偏函数等。其中两个重要特性闭包、偏函数如下。

### 2.1 闭包

在 JS 语言特有的“嵌套作用域”中，处在内部的对象可以一层一层地向上寻找所有包含它的外部对象的变量，但是外部对象却不能向内访问内部对象的变量。因此可以想到只要能把一个函数在放在另一个函数的内部，再去返回，就可以在函数作用域之外访问其局部变量，并且因为处在内部的函数引用了外部函数的变量，所以在外部函数调用完之后，也不会被垃圾回收机制消除。

所以得出当一个定义在函数内部的函数被它作用域的外部引用时，就创建了该内部函数的闭包。闭包的作用就是用来存储内部数据，以及帮助外部访问内部数据。

### 2.2 偏函数

偏函数可以认为用来固定某个函数的一个或多个参数，然后再返回一个用来接受没有被固定的参数的新函数。偏函数可通过 bind 函数来实现，其表达式可以语义化理解为：新函数 = 原函数.bind（参数 1，参数 2…），其中新函数和原函数的函数体部分是一样的。当调用新函数时，只需要接收没有固定的参数值（剩余参数值），然后进行函数体的运算。

这样通过对函数固定不同的参数值，就可以得到不同的新函数，也就为原函数增加许多变体函数，这些变体函数使用起来会更加方便快捷，不用向原函数一样写很多的重复代码。所以偏函数的作用就是为函数增加通用变体，减少代码量，提高代码编写效率。

## 3 函数式编程在 JavaScript 中的应用

在实际 JavaScript 开发过程中，开发者已经使用了许多函数式编程的方式来解决问题，其中对闭包的运用是最为广泛的。如下介绍两个关于闭包和柯里化的实际应用。

### 3.1 通过闭包来改变页面内容的大小

在制作 Web 页面时，难免会遇到只通过 CSS 无法完成的动画效果，比如点击一个元件触发另一个元件的大小或者其他样式的改变。如果不使用闭包来实现，那么需要为每个元件单独绑定一个点击事件，为它们单独编写一个函数。因为在使用普通函数的条件下，如果也让所有的点击事件调用同一个函数，会造成每次调用这个函数，并为它传入新的参数后，前面调用传入的参数的就会被覆盖，只会保留最后一次传入的参数。当鼠标点击其他的元件时，因为参数被覆盖掉，也就无法实现对应的页面变化。所以对于普通函数只能为每个元件的点击事件单独写一个函数，而不能让他们共用同一个函数，这样对拥有相同函数体的函数进行了重复的编写，造成了代码的冗余，使 JS 代码不够简洁，增加了内存消耗。

这个问题还可以通过对象构造函数解决，将一个函数作为原型，这个函数有一个参数，通过这个参数来控制字体和图片的大小，通过 this 绑定当前调用的上下文对象，然后在对象方法里实现对页面内容大小的修改，再通过 new 关键字为每个元件创建一个新的实例对象，并将实例对象的方法绑定到点击事件上去。通过这样的对象构造函数来实现，比较复杂，理解起来也不是很容易，特别是要注意当前调用的上下文对象。

相对于上面两种解决方法，使用闭包来实现这个问题更占优势，因为代码量较少，而且更易于理解。在图 1 用闭包来解决问题的核心代码中，外围函数 changeSize（）的实参传入的是需要改变的数值大小，内层函数返回的是经过处理后的文字大小和图片的高宽，这里实现了内层对外层参数的引用，使数据保存起来，不被垃圾回收机制处理掉。然后根据传入的参数不同，就创建了不同的变体函数（这里和偏函数固定参数有点相似），然后将这些变体函数根据对应的参数数值绑定到不同元件的点击事件中去。

```
function changeSize(size){
    return function(){
         document.body.style.fontSize = size+"px";
         p1.style.width = size*10+"px";
         p1.style.height = size*10+"px";
     }}
```

图 1：闭包代码


图 2：页面效果

当鼠标点击 12 显示图 2 左边，点击 14 显示图 2 右边，通过右侧导航栏对比，可以看出点击 12 的时候，字号和图片高宽都要小一些，这样就证明了通过函数闭包可以实现对页面内容大小的改变。

### 3.2 通过函数柯里化来记录每月的花销

在 JS 函数式编程中，柯里化是闭包的一个拓展，指的是将原本含有多个参数的函数拆分为一个新的接收单个参数函数的过程。柯里化函数在参数传入后并不会立即执行，而是通过函数嵌套形成闭包，将数据保存起来，等到函数真正需要使用的时候，前面传入的参数就会一次性被用于求值。

本问题如果不使用函数柯里化，解决方法就是先将每个月的花销全部放入一个数组中，然后对数组进行遍历累加计算。运用这样的方法，每次在数组中追加元素后，又会对数组进行遍历，再次进行求值计算。这样的解决方式不够灵活，因为每当数组中的元素改变，后面的求值运算又会重新来一次，原本没有进行改变的元素，又被迫再次进行运算，这样增加了一些不必要的重复计算，使得代码的整体的运行效率不高。

但是当使用函数柯里化后，代码就会很灵活，添加元素也很方便，它会根据提供的元素进行弹性求值。参数的传入也很自由，每次只需要调用柯里函数，将每月开销作为实参传入，就可以通过柯里化产生的闭包将数值保存起来，最终在需要求值的时候，所有之前传入的参数都参与到统一求值中去。如下核心代码图 3 所示，先设计一个柯里化函数，在 currying 函数中，当 arguments 对象参数为空时，也就是传入第一个参数时，用之前定义的数组 args 来接收所有的参数，参数不为空时，将后面方法里面的参数和 args 合并，并调用含有 arguments 对象的函数。在该问题中，当把 cost 函数柯里化之后，并给它传入一个参数如 cost（100）时，函数 cost 也没有进行求值，而是返回的一个函数，并将 100 传入 arguments 中，不管传入多少次值都是重复此过程。只有等到最后单独调用 cost（）函数时，才会执行 cost 内部拥有 arguments 对象的 function，进行一次性求值，然后返回总的 money 值。

使用柯里化记录开销会更加灵活，可以通过柯里函数随时增加或减少实参的值，且在最后进行统一的求值。这种最后求值的方式涉及到了延迟执行，也是函数式编程另一个重要特性。如此可以看出，柯里化函数更适合去做一些大型数据的累加计算。因为当延迟执行判定后面的数据不再需要，或发生错误时，函数就会立即停止，不再执行下去。这样就大大减少了程序去做无意义的计算的概率，增加了程序的运行效率，而且通过这种方法，也更易于操作者理解和实践。

```
var currying = function( fn ){
var args = [];
return function(){
if ( arguments.length === 0 ){
return fn.apply( this, args );
}else{
[].push.apply( args, arguments );
return arguments.callee; } };
var cost = (function(){
var money = 0;
return function(){
for ( var i = 0, l = arguments.length; i < l; i++ ){
money += arguments[ i ]; }
return money;
} })();
var cost = currying( cost ); // 转化成 currying 函数
cost( 100 ); // 未真正求值
cost( 200 ); // 未真正求值
cost( 300 ); // 未真正求值
alert ( cost() ); // 求值并输出：  600
```

图 3：柯里化代码

## 4 JavaScript 函数式编程优点

JavaScript 函数式编程有许多的优点，代码简洁，

开发快速，更像自然语言，易于理解，使用纯函数等。

（1）代码简洁，开发快速

函数式编程含有闭包和偏函数等特性，使得在实际项目开发过程中，函数式编程会比面向对象编程更加快速，因为它大量使用函数来解决问题，少量的函数代码就能完成相同目标。

（2） 使用纯函数

函数式编程是具有纯函数（对于相同的输入都将返回相同的输出）的特点，纯函数给函数式编程带来了许多的好处，无副作用，可测试，可编写并发代码，可以缓存[3]。

## 5 JavaScript 函数式编程缺点

函数式编程在 JavaScript 编程中，还有一些不足的地方，比如缺少不可变数据结构，缺少尾递归优化，过度封装等。

（1）缺少不可变数据结构

不可变性指的是不允许通过表达式来转换数据结构，但是在 JS 语言中是缺少的这一特性的，比如说数组和对象是可以通过表达式转换的。可变数据结构会导致如果函数的用户对返回值的进行了修改，那么缓存中的数据会受影响。

（2）缺少尾递归优化

尾递归指的是出现在函数末尾的一个递归调用，因为出现在最后，所以不需要为下次调用保持上下文环境和栈。在 JS 中是缺少尾递归优化，而栈的空间又不是无限大的，当遇到调用层数过多时，可能会出现栈溢出和性能下降。

## 6. 结语

JavaScript 在函数式编程方面缺少不可变数据结构，尾递归优化等不足的地方，但是随着其不断的发展和对自身语言更加规范，它在函数式编程语言上还有很大的发展空间。JavaScript 和函数式编程可以说是双向成就的，JS 应用范围越来越广，成为全世界最多人使用的编程语言，可以变相推动函数式编程进一步走向大众的视野，让更多人使用函数式编程，体验函数式编程的魅力。而函数式编程也会使 JavaScript 语言变得更规范，为 JS 提供更加简洁且高效的代码，拓宽 JS 的应用场景。

## 【参考文献】

[1] 胡恒，谢彩云．函数式编程在 Scala 中的实现[J]．福建电脑，2018,34(11):148+52.

[2] 李轶．基于闭包的 JavaScript 面向对象编程框架[J]．江汉大学学报（自然科学版），2011,39(02):56-60.

[3] 宫明．函数式编程探析[J]．电脑编程技巧与维护，2017(03):29-30.