

[首页 \(http://www.open-open.com/\)](http://www.open-open.com/) [代码 \(http://www.open-open.com/code/\)](http://www.open-open.com/code/) [文档 \(http://www.open-open.com/doc/\)](http://www.open-open.com/doc/) [问答](#)

[全部经验分类](#)



[Android \(/lib/tag/Android\)](#)

[IOS \(/lib/tag/IOS\)](#)

[JavaScript \(/lib/tag/JavaScript\)](#)

[\(/lib/list/all\)](#)

[所有分类 \(/lib/list/all\)](#) > [开发语言与工具 \(/lib/list/36\)](#) > [JavaScript开发 \(/lib/list/145\)](#)

深入了解Javascript函数式编程

[JavaScript \(/lib/tag/JavaScript\)](#)

[函数式编程 \(/lib/tag/函数式编程\)](#)

2016-04-09 22:36:00 发布

您的评价: 0.0

收藏

1收藏

初阶部分

字符串可以保存为变量，函数说他也可以

```
var autumn = 'autumnswind';
    var autumn_fn = function() {
        return 'autumnswind';
    };
```

字符串可以保存对象字段，函数说他也可以

```
var autumnswind = {
    autumn: 'autumnswind',
    autumn_fn: function() {
        return 'autumnswind'
    }
};
```

字符串可以用时再创建，函数说他也可以

```
'Autumns' + (function() {
    return 'Wind'
})();
```

字符串可以作为参数传给函数，函数说他也可以

```
var autumn;
    function autumn_fn(){};
    function hellloWorld(autumn, autumn_fn) {
        return;
    }
```

字符串可以作为函数返回值，函数说他也可以

```
return 'autumnswind';
    return function() {
        return 'autumnswind';
    };
```

所以函数真的是JS里面的上等好公民啊~可以穿梭于任何地方，并谈笑风生~ 高阶部分：

有了上面的例子，那么我们就可以把函数这个小孩子带到好多好玩的地方，往下慢慢看~

首先看第一个例子

```

var
    obj1 = {
        value: '秋风'
    },
    obj2 = {
        value: 'autumnswind'
    },
    obj3 = {
        value: '莎娜'
    };
var values = [];

function add(obj) {
    values.push(obj.value);
}
add(obj1);
add(obj2);
console.log(values); // 秋风,autumnswind

```

这种写法，大家都知道了，变量会污染全局环境，并且一旦有一个地方忘记修改了就会变得很不稳定，不像函数体一样，只负责内部的输入输出，这里如果融入上下文的其他函数和变量就会变得非常混乱

根据这样修改成第二个例子：

```

var
    obj1 = {
        value: '秋风'
    },
    obj2 = {
        value: 'autumnswind'
    },
    obj3 = {
        value: '莎娜'
    };

function add(obj) {
    var values = [];
    values.push(obj.value);
    return values;
}
跟下面一样
/*function add(obj) {
    var values = [];
    var accumulate = function() {
        values.push(obj.value);
    };
    accumulate();
    return values;
}*/
add(obj1);
add(obj2);
console.log(add(obj1)); // 秋风
console.log(add(obj2)); // autumnswind

```

这样我们把values数组声明放进去函数内部，这样就不会被其他外部变量骚扰了，但是问题又来了，只有最后传入对象值才可以返回，那不是我们的初衷

最后看消除所有尴尬的第三个例子：

```

var
    obj1 = {
        value: '秋风'
    },
    obj2 = {
        value: 'autumnswind'
    },
    obj3 = {
        value: '莎娜'
    };
var
    Add = function(obj) {
        var
            values = [];
        var
            _calc = function(obj) {
                if (obj) {
                    values.push(obj.value);
                    return values;
                } else {
                    return values;
                }
            };
        return _calc;
        //可以这样把它看成匿名函数省去一个没必要的变量
        /*return function(obj) {
            if (obj) {
                values.push(obj.value);
                return values;
            } else {
                return values;
            }
        }*/
    };
var
    calc = Add();
calc(obj1); //相当于 ValueAccumulator()(obj1)
calc(obj2); //走 if 分支
console.log(calc()); //走 else 的分支

```

这里制造一个闭包来保存第一层函数的`values`数组，这个数组既可以被内部函数 `calc` 调用，也可以在自己函数体内调用；第一层函数的关键点在于返回的是在自己内部定义的那个函数 `calc`，这个hack就能让Add函数在外部能访问函数体内的一直保存的`values`数组，进一步说这种方法可以得到第一层函数任何一个声明的变量，这是闭包的一个很常见的用法（返回函数）。

理解下面这些话

JS中的闭包就是函数可以访问父作用域（这个总结够短的，有争议的可以先搁置争议）

上面其实可以看做是自执行的匿名函数（`_calc`可以它看成一个匿名函数输出来），自执行的函数实际上是高阶函数的一种形式。高阶函数就是以其它函数为输入，或者返回一个函数为输出的函数（这个理解越来越像是闭包的理解了）

所以函数将其他函数作为输入参数或者作为返回值，就可以称为高阶函数

上面其实还需要了解一个知识点就是纯函数

```

//非纯函数
var autumn1 = function(str) {
    window.innerHeight;
    window.innerWidth;
    return str + 's innerHeight: ' + window.innerWidth + 'px';
};
//纯函数
var autumn2 = function(str, height) {
    return str + 's innerHeight: ' + height + 'px';
};
console.log(autumn1('autumnswind'));
console.log(autumn2('autumnswind', 254));

```

两个函数的区别在于非纯函数依赖window这个全局对象的状态来计算宽度和高度，而自给自足的纯函数则要求这些值作为参数传入，当一个函数是纯的，也就是不依赖于当前状态和环境，我们就不用管它实际的计算结果是什么时候被计算出来。

纯函数返回的计算结果仅与传入的参数相关

纯函数是完完全全独立的，所以它适合被重复调用使用，为下面的函数编程做一个好铺垫

有更深理解或者有误的话请务必提醒并留言我~

回到题目核心的JS函数编程

由于JS中，函数是头等公民，这里的实际意思就是函数被认定最基本最根本的类型，正如数字和对象一样。如果数字和对象可以被来回传递，那么函数（函数我为什么就不可以呢）也可以的。

这就是JS函数编程的基本思想吧

那下面第四段代码就是在第三段代码后面的基础上再添加，实现这种思路

```
var calc2 = Add();
    var objects = [obj1, obj2, obj3]; // 这个数组可以很大
    objects.forEach(calc2);
//注意对象forEach的用法 array.forEach(callbackfn[, thisArg]); 对于数组中的每个元素，forEach
都会调用callbackfn函数一次
    console.log(calc2());
```

这里的calc2函数就变成一个‘变量’进入到forEach参数内

我们可以在这个基础上继续扩充功能，第五段代码，链式调用

```
objects1 = objects.reverse();
objects2 = objects1.concat([4, 16]);
objects3 = objects2.map(Math.sqrt);
console.log(objects3);
```

用jQuery用多了，就会知道，jq习惯把类似上面的代码链式调用类似这样把代码串在一起

```
console.log(['秋风', 'autumswind', '莎娜'].reverse().concat([4, 16]).map(Math.sqrt));
//NaN, NaN, NaN, 2, 4
//写明显一点
/*console.log(
    ['秋风', 'autumswind', '莎娜']
    .reverse()
    .concat([4, 16])
    .map(Math.sqrt)
);*/
```

这样用的前提必须是这个函数对象里面拥有这个方法，换句话说就是Array.prototype有这个方法（例如reverse, concat, map）给他扩充一个简单的方法如下：

```
Array.prototype.make4 = function() {
    console.log(this[4]);
    //this.length=4;
    return this[4];
}
console.log(['秋风', 'autumswind', '莎娜'].reverse().concat([4, 16]).map(Math.sqrt).make4()); //4
```

上面除了展示匿名函数，链式调用，还有一种方法是函数式编程里面常用的，就是递归

```
var Add2 = function(n) {  
    if (n < 0) {  
        // 基准情形  
        return 'I am autumnswind';  
    } else {  
        // 递归情形  
        return Add2(n - 1);  
    }  
}  
console.log(Add2(5));
```

注意，基准情形就是让递归停止下来的条件，防止无限递归

再复杂点就叫分而治之，其实数学界很多问题都可以用递归解决的

```
var Add3 = function(a, b) {  
    if (a < 0 && b == 8) {  
        console.log(a % b); //-1  
        // 基准情形  
        return 'I am autumnswind';  
    } else {  
        // 递归情形  
        return Add3(a - 1, b);  
    }  
}  
console.log(Add3(5, 8));  
console.log(Add3(5, 7)); //Maximum call stack size exceeded
```

来自：<http://div.io/topic/1678> (<http://div.io/topic/1678>)

查看源码：<https://github.com/AutumnsWind/Good-text-Share/issues/13> (<https://github.com/AutumnsWind/Good-text-Share/issues/13>)

同类热门经验

1. Node.js 初体验 (/lib/view/open1326870121968.html)
2. JavaScript开发规范要求 (/lib/view/open1352263831610.html)
3. 使用拖拉操作来自定义网页界面布局并保存结果 (/lib/view/open1325064347889.html)
4. Nodejs入门学习, nodejs web开发入门, npm、express、socket配置安装、nodejs聊天室开发 (/lib/view/open1329050007640.html)
5. 利用HTML5同时上传多个文件 - resumable.js (/lib/view/open1327591300671.html)
6. nide：一个不错的Node.js开发工具IDE (/lib/view/open1325834128750.html)

阅读目录

初阶部分

相关文档 — 更多 (<http://www.open-open.com/doc>)

- JavaScript 的语言精髓与编程实践(第三章非函数式语言特性).pdf (<http://www.open-open.com/doc>)

相关经验 — 更多

(<http://www.open-open.com/lib>)

- 常用的Javascript设计模式

相关讨论 — 更多 (<http://www.open-open.com/solution>)

- PHP程序员的技术成长规划

- open.com/doc/view/faaf2f99eac14e9499be057c97ca7ec1) (/lib/view/open1456539877859.html)(<http://www.open-com.com>)
- JavaScript征途.pdf (<http://www.open-com.com/doc/view/049c259f2fb0425593acff6b9db532fa>) (/lib/view/open1463445178466.html) 高效使用 JavaScript 闭包 ([open.com/solution/view/1414478644325](http://www.open-com.com/solution/view/1414478644325))
 - JavaScript内核系列.pdf (<http://www.open-com.com/doc/view/eaec15769aa14c208387d32e2c335d66>) (/lib/view/open1347004181600.html) 程序员技术练级攻略 ([open.com/solution/view/1319276210452](http://www.open-com.com/solution/view/1319276210452))
 - JavaScript内核高级教程.pdf (<http://www.open-com.com/doc/view/1c1a9a2c3783479aaa9d6927849f2d60>) (/lib/view/open1464313989877.html) 以优美方式编写JavaScript代码 ([open.com/solution/view/1348305460369](http://www.open-com.com/solution/view/1348305460369))
 - JavaScript 内核高级教程(剖析).pdf (<http://www.open-com.com/doc/view/00f5b00c5ce947bf9212be3aa33fe178>) (/lib/view/open1464308452228.html) JavaScript 函数式编程探索与思考 ([open.com/solution/view/1322451238921](http://www.open-com.com/solution/view/1322451238921))
 - 动态函数式语言精髓与编程实践(Javascript 版).pdf (<http://www.open-com.com/doc/view/a21791094d824cd9bc2ee117bbf87244>) 通过 Eweda.js 学习函数式 JavaScript 编程 ([open.com/solution/view/1318472833218](http://www.open-com.com/solution/view/1318472833218))
 - Functional JavaScript 中文版.pdf (<http://www.open-com.com/doc/view/28dff3d278a343d386a1389bef376b0a>) (/lib/view/open1419993900015.html) 码农提高工作效率 ([open.com/solution/view/1397702553562](http://www.open-com.com/solution/view/1397702553562))
 - JavaScript语言精髓与编程实践.pdf (<http://www.open-com.com/doc/view/1c31f8783c6f4d08b551f38fbc4100a2>) (/lib/view/open1340007193252.html) 我为什么向后端工程师推荐Node.js
 - JavaScript 语言精髓与编程实践(第三章).pdf (<http://www.open-com.com/doc/view/8b49ac47cb6947fb9203c8a89da5e466>) (/lib/view/open1428465838682.html) 那些年，追过的开源软件和技术
 - 《JavaScript编程精解》迷你书.pdf (<http://www.open-com.com/doc/view/2c1d4907f86c4783813ae4891ec42b48>) (/lib/view/open1452264390511.html) 深入浅出 React Native：使用 JavaScript 构建原生应用 ([open.com/solution/view/1425959150201](http://www.open-com.com/solution/view/1425959150201))
 - JavaScript设计模式与开发实践.pdf (<http://www.open-com.com/doc/view/91e78191c05b4966bd5ce9b19a2dab62>) (/lib/view/open1430835931835.html) 免费的编程中文书籍索引
 - 真正的JavaScript忍者秘籍.pdf (<http://www.open-com.com/doc/view/d697952596ec4682a4c835aa351fcb8c>) (/lib/view/open1400817311456.html) 编写高质量JavaScript代码的一些建议
 - 《JavaScript高级程序设计(第2版)》(Professional JavaScript for Web Developers, 2nd Edition).pdf (<http://www.open-com.com/doc/view/2dbab4a3ee2a47f9a3c142c32dd655f5>) (/lib/view/open1407913452473.html)
 - JavaScript 内核第0版.pdf (<http://www.open-com.com/doc/view/80ccc2fb2de6466c836e3560ed9a3254>) (/lib/view/open1456238877995.html)
 - JavaScript面向对象精要（迷你书）.pdf (<http://www.open-com.com/doc/view/db1dc582934047b1a4b81f8d0b09c835>)
 - python 基本语法.doc (<http://www.open-com.com/doc/view/7660c860064c4ee6be17ef8822ddd32d>)
 - JavaScript 内核 第0版.pdf (<http://www.open-com.com/doc/view/64aaad3513a54560ace908ca547d17ee>)
 - JavaScript面向对象编程.pdf (<http://www.open-com.com/doc/view/e5ffb0ef8f134951825ab8b953caadfa>)
 - javascript 编程.ppt (<http://www.open-com.com/doc/view/26d355ce0ddc4a789d8574fe4e0e00ac>)
 - JavaScript 编程指南.pdf (<http://www.open-com.com/doc/view/508f289f8baa41b2bcf55bebd14aad9b>)

