## 函数的定义

### 函数总共有三种定义方式：

#### 函数式声明（命名函数）：

使用function关键字，

function fnMethodName(x){alert(x);}

##### 函数的执行顺序：

函数式声明中，会存在函数作用域的提升；

**函数声明在JS解析时进行函数提升**，因此在同一个作用域内，不管函数声明在哪里定义，该函数都可以进行调用。

因为：用函数声明创建的函数可以在**函数解析后调用**（解析时进行等逻辑处理）

即：

test1();

function test1() { //函数声明

alert("1111");

}

// “1111”

是因为函数声明实在函数解析完成后调用的，进行了函数提升，test1()才能调用。

#### 2、函数字面量，函数表达式(Function Literals)：

函数表达式的值是在JS运行时确定，并且在表达式赋值完成后，该函数才能调用。

var fnMethodName = function(x){alert(x);}

##### 函数的执行顺序：

函数表达式创建的函数是在运行时进行赋值，且要等到表达式赋值完成后才能调用。

即：

test2();

var test2 = function(){ //函数字面量

alert("2222");

}

//error: test2 is not a function

，，，是因为函数字面量不是函数声明，函数字面量表示方法是在运行时解析的，在调用test2()之前，函数字面量的函数体还没有声明

#### 3、Function()构造函数：

var fnMethodName = new Function(‘x','alert(x);') // 由Function构造函数的参数个数可变。最后一个参数写函数体，前面的参数写入参。

##### 函数的执行顺序：

即：

test3();

var test3 = new Function("alert(3333);");

//error: test3 is not a function

##### ES6中的箭头函数和普通函数有什么区别？

1、**普通函数中的this总是指向调用它的那个对象**，

**箭头函数没有自己的this，他的this永远指向其定义环境**，任何方法都改变不了其指向，如call（）、bind（）、apply（）。（正是因为它没有this，所以也就不能用作构造函数，也没有原型对象）

2、**箭头函数不能当作构造函数**，也就是说，**不能使用new命令**，否则会报错。

3、**箭头函数不能使用yield命令**，因此箭头函数不能用作genertor函数。

4、**箭头函数没有原型属性**。

5、**箭头函数不能使用arguments对象**，该对象在函数体内不存在。如果要用，可以用rest参数代替。

6、**变量提升**：**由于js的内存机制，function的级别最高，而用箭头函数定义函数的时候，需要var(let、const）关键字，而var所定义的变量不能得到变量提升。故箭头函数一定要定义于调用之前。**

**this的指向问题？**

1、**普通函数中，this指向其函数的直接调用者**；

2、**箭头函数中，this指向其定义环境**，任何方法都改变不了其指向，如call（）、bind（）等；

3、构造函数中，如果不使用new,则this指向window，如果使用new创建了一个实例，则this指向该实例。

4、**window内置函数中，如setInterval,setTimeout等，其内部的this指向Window。**

5、**匿名函数的this指向Window**。

6、**apply()、call()、bind()可以改变this的指向**

## 函数

### 函数式编程

函数式编程主张以抽象的方式创建函数，这些函数能够在代码的其他部分被重用。

抽象：**把复杂的东西抽出来，变成简单的东西**。

特点：

具有代码质量高且易于维护的特点

应用：

函数式编程因为其特点更适用于统计分析数据、科学计算、大数据处理等方面工作，当然并不限于这些，在web开发、服务器脚本等其它方面也很不错，而面向对象编程更适合于开发和处理业务性强、功能模块完备的大型业务系统。

### 高阶函数

高阶函数的英文是"Higher-order-function"。**指的是操作其它函数的函数**，一般来说，有两种情况：**函数作为参数被传递**；**函数可以作为返回值输出**。

React 的高阶组件本质上就是高阶函数。

**高阶函数源自于函数式编程**（不熟悉的朋友请看《javascript中的函数式编程》），是函数式编程的基本技术。

那么，JS作为一门“一切皆为对象”的语言，是如何拥有函数式编程的能力呢？

是因为在**JS中函数是一等公民**，即**函数可以被赋值给变量，被变量引用，这便使得函数可以作为参数，在其他函数间相互传递**：

例子：

**函数作为参数用于回调函数**。回调函数是一个函数作为参数传递给另一个主函数里面(otherFunction),当那一个主函数执行完后，再执行传入的作为参数的函数。被作为参数传递到主函数的那个函数就叫做回调函数。

<script>

function title(value) {//回调函数

alert(value);

}

function main(title, value) {

//主函数，title当作参数,value这个值正是title()函数需要的

alert('我是主函数');

title(value); // 这行的title()是回调函数

}

main(title, '我是回调函数')

//调用的是main函数，先执行main()这个主函数，title()被main()在函数体中执行一次，更能体现title()是回调函数

</script>

### 纯函数

纯函数是函数式编程中非常重要的一个概念，

简单来说，**就是一个函数的返回结果只依赖于它的参数，并且在执行过程中没有副作用，我们就把这个函数叫做纯函数**。

**优点特点：**

1. 可复用性

纯函数仅依赖于传入的参数，这意味着你可以随意将这个函数移植到别的代码中，只需要提供踏需要的参数即可。如果是非纯函数，有可能你需要一根香蕉，却需要将整个香蕉树搬过去。

2. 可测试性

纯函数非常容易进行单元测试，因为不需要考虑上下文环境，只需要考虑输入和输出。

3. 并行代码

纯函数是健壮的，改变执行次序不会对系统造成影响，因此纯函数的操作可以并行执行。

### 函数柯里化

在计算机科学中，柯里化（英语：Currying），又译为卡瑞化或加里化，**函数柯里化又称为部分求值，是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接受余下的参数而且返回结果的新函数的技术。**

**柯里化(Currying)指的是将原来接受两个参数的函数变成新的接受一个参数的函数的过程。新的函数返回一个以原有第二个参数为参数的函数。**

**原理：**

**\* currying函数的实质其实就是预先存储的概念，利用了闭包的特性来保存中间过程中输入的参数。 柯里化的过程是逐步传参，逐步缩小函数的适用范围，逐步求解的过程。**

例如有一个简单的加法函数，他能够将自身的三个参数加起来并返回计算结果。

function add(a, b, c) {

return a + b + c;

}

那么add函数的柯里化函数\_add则可以如下：

function \_add(a) {

return function(b) {

return function(c) {

return a + b + c;

}

}

}

下面的运算方式是等价的。

add(1, 2, 3);

\_add(1)(2)(3);

### compose函数

在函数式编程中有一个很重要的概念那就是函数组合，实例上就是把处理数据的函数像管道一样连接起来，然后让数据穿过管道得到最终的结果。

compose函数是一个函数（函数只接受一个参数）的返回值作为另外一个函数的参数，将需要嵌套执行的函数平铺。

**特点：**

它可以使得代码更加精练，代码可读性更好。

**例子：**

function compose(...funcs) {

//...funcs接收的就是所有传递进来的函数

return function anonymous(val) {

//val第一个函数执行时需要的实参[div2,mul3,add1,add1]

//要用到reverse()方法进行反转

return funcs.reverse().reduce((num, item) => {

//当没有传入函数时

if (funcs.length === 0) return val;

//当传入的函数为1时

if (funcs.length === 1) return funcs[0](val);

//num如果是一个函数那首先要把val传入num中执行，把执行的结果赋值给item

//num不是函数就把num当成参数传递给item

return typeof num === 'function' ? item(num(val)) : item(num);

});

}

}

let result = compose(div2, mul3, add1, add1)(0);

console.log(result);