目录

[流程控制 4](#_Toc16945345)

[赋值语句 4](#_Toc16945346)

[函数 4](#_Toc16945347)

[递归函数 5](#_Toc16945348)

[对象 5](#_Toc16945349)

[Math对象 6](#_Toc16945350)

[属性/描述 6](#_Toc16945351)

[方法/描述 6](#_Toc16945352)

[代数 6](#_Toc16945353)

[统计与对数 7](#_Toc16945354)

[基本运算与取整 7](#_Toc16945355)

[随机数 7](#_Toc16945356)

[Date对象 8](#_Toc16945357)

1 单引号内出现双引号不需要转义

2 布尔值true:on/yes/1

False:off/no/0

3 转义字符

|  |  |  |  |
| --- | --- | --- | --- |
| \b |  | \v |  |
| \n |  | \r |  |
| \t |  | \\ |  |
| \f |  | \OOO |  |
| \’ |  | \xHH |  |
| \” |  | \uhhhh |  |

在document.writeIn();语句中引用转义字符是，只有将其放在格式化文本块中才会起作用，所以脚本必须包含在<pre></pre>标记内

使字符串换行

Document.wtiteIn(<pre>);

Document.wtiteIn(“轻松学习\n语言”);

Document.wtiteIn(</pre>);

4 未定义类型的变量underfined

表示变量没有被赋值，或赋予一个不存在的属性值

特殊类型的数字常量NaN,非数字。由于某种原因计算错误，产生一个没有意义的数字

5 null空值 空的或不存在的引用，试图引用一个没有定义的变量，则返回一个空值。Null不等于””或0

1 null 表示一个变量被赋予空值

Undefined变量 未被赋值

2 常量Constant const 常量名：数据类型=值

数字、true、false

1算数运算符

++ i++使用i之后，使i的值加1 i=1;j=i++//j=1,i=2

++i使用i之前，先使i的值加1 i=1;j=++i//j=2,i=2

1 位操作运算符

2 typeof :number string Boolean object function underfined

3 new运算符 创建新的对象

New constructor[(arguments)]

Constructor必选

Arguments可选

Object1=new Object;

Array2=new Array();

Date3=new Date(“August 8 2016”)

1 优先级

结合性 a+b+c 向左结合（a+b）+c 向右结合a+(b+c)

|  |  |  |
| --- | --- | --- |
| 最高 |  | . [] () |
|  | 向右 | ++ -- - ! delete new typeof void |
|  |  | \* / % |
|  |  | + - |
|  |  | << >> >>> |
|  |  | < <= > >= in instanceof |
|  |  | === != === !== |
|  |  | & |
|  |  | ^ |
|  |  | | |
|  |  | && |
|  |  | || |
|  | 右 | ?: |
|  | 右 | = |
|  | 右 | \*= /= %= += -= <<= >>= >>>= &= ^= |= |
| 最低 |  | , |

1 表达式 语句结合,计算结果是一个单一的值，该结果归入以下数据类型：boolean number string function object

表达式可以是数字，变量，许多连在一起的变量关键字及运算符.x=7;7是表达式，赋值后，x也是表达式

2 逻辑型数据

False :undefined null 0 NaN “”字符串长度为0

字符串型数据

Null 转换为字符串为“NaN”

# 流程控制

## 赋值语句

用大量的变量存储程序用到的数据

变量名=表达式

# 函数

Function functionName([parameter1,parameter2,…]){

Statements;

[return expression;]

}

2 将需要返回的值赋予变量，将变量返回

## 递归函数

函数在自身的函数体内调用自身，处理不当会使程序进入死循环，递归函数只在特定情况下使用，如阶乘问题

Function functionName(parameters1){

functionName(parameters2);

}

10!=10\*9! 9!=9\*8! 1!=!

Function f(num){

If(num<=1){

Return 1;

}else{

Return f(num-1)\*num;

}

F(10)

定义递归函数时需满足两个必要条件：

1. 包含一个结束递归的条件
2. 包含一个递归调用语句

# 对象

1 按照使用方法分为 动态对象/静态对象

引用动态对象的属性和方法时，必须使用new关键字来创建一个对象实例，然后才能使用 对象实例名.成员 的方式来访问其属性和方法。 引用静态对象的属性和方法时，直接使用 对象名.成员 的方式访问其属性名和方法

# Math对象

## 属性/描述

E 自然对数的底

LN2 2的自然对数

LN10

LOG2E

PI

SQRTI\_2

SQRT2

## 方法/描述

代数

acos

asin

atan

cos

sin

tan

统计与对数

exp

log

max

min

基本运算与取整

abs

ceil

floor

pow幂

round

sprt

随机数

random

# Date对象

## [深入理解 函数、匿名函数、自执行函数](https://www.cnblogs.com/jessiespur/p/9283101.html)

<https://www.cnblogs.com/jessiespur/p/9283101.html>

### 基础概念：定义函数的方式

  一般定  
义函数有两种方式：  
    1：函数的声明  
    2：函数表达式

## 函数的声明

      如下代码就是函数声明的代码结构：

function sum(x,y){

alert(x+y);

}

sum(1,2); //3

      关于函数声明，它最重要的一个特征就是函数声明提升，意思是执行代码之前先读取函数声明。这意味着可以把函数声明放在调用它的语句之后。如下代码可以正确执行：

sum(1,2); //3

function sum(x,y){

alert(x+y);

}

## 函数表达式

     函数表达式中有几种不同的语法。最常见和最具代表性的一种如下代码所示：

var ss = function(x,y){

alert(x+y);

};

ss(1,2);

     这种形式看起来好像是常规的变量赋值语句。但是函数表达式和函数声明的区别在于，函数表达式在使用前必须先赋值。所以一下代码执行的时候就会出错：

ss(1,2); //报错，显示undefined is not a function

var ss = function(x,y){

alert(x+y);

};

     造成这种现象是因为解析器在向执行环境中加载数据时，解析器会率先读取函数声明，并使其在执行任何代码前可用；至于函数表达式，则必须等到解析器执行到它的所在的的代码行，才会真正的被解析。

     函数表达式中，创建的函数叫做匿名函数，因为function关键字后面没有标识符。

## 匿名函数的调用方式

     匿名函数，顾名思义就是没有名字的函数。上面的函数表达式中的创建，即创建一个匿名函数，并将匿名函数赋值给变量ss，用ss来进行函数的调用，调用的方式就是在变量ss后面加上一对括号()，如果有参数传入的话就是ss(1,2)，这就是匿名函数的一种调用方式。

还有一种匿名函数的调用方式是：使用()将匿名函数括起来，然后后面再加一对小括号（包含参数列表）。我们再看一下以下一个例子：

alert((function(x,y){return x+y;})(2,3));//5

alert(( new Function("x","y","return x+y;"))(2,3));//5

    在javascript中，是没有块级作用域这种说法的，以上代码的这种方式就是模仿了块级作用域(通常成为私有作用域)，语法如下所示：

(function(){

//这里是块级作用域

})();

    以上代码定义并立即调用了一个匿名函数。经函数声明包含在一对圆括号中，表示它实际上是一个函数表达式。而紧随其后的另一对圆括号会立即调用这个函数。

    然而要注意一点：

function(){

}();

    上面的代码是错误的，因为Javascript将function关键字当作一个函数声明的开始，而函数声明后面不能加圆括号，如果你不显示告诉编译器，它会默认生成一个缺少名字的function，并且抛出一个语法错误，因为function声明需要一个名字。有趣的是，即便你为上面那个错误的代码加上一个名字，他也会提示语法错误，只不过和上面的原因不一样。在一个表达式后面加上括号()，该表达式会立即执行，但是在一个语句后面加上括号()，是完全不一样的意思，他的只是分组操作符（此处摘自[汤姆大叔的博客](http://www.cnblogs.com/TomXu/archive/2011/12/31/2289423.html)）。

[复制代码](javascript:void(0);)

// 下面这个function在语法上是没问题的，但是依然只是一个语句

// 加上括号()以后依然会报错，因为分组操作符需要包含表达式

function foo(){ /\* code \*/ }(); // SyntaxError: Unexpected token )

// 但是如果你在括弧()里传入一个表达式，将不会有异常抛出

// 但是foo函数依然不会执行

function foo(){ /\* code \*/ }( 1 );

// 因为它完全等价于下面这个代码，一个function声明后面，又声明了一个毫无关系的表达式：

function foo(){ /\* code \*/ }

( 1 );

[复制代码](javascript:void(0);)

    所以上面代码要是想要实现，就必须要实现赋值，如a = function(){}()，"a="这个告诉了编译器这个是一个函数表达式，而不是函数的声明。因为函数表达式后面可以跟圆括号。所以下面两段代码是等价的。

var aa = function(x){

alert(x);

}(5);//5

(function(x){alert(x);})(5);//5

  有上面对于函数和匿名函数的了解，我们引申出来了一个概念，即自执行函数，让我们更加深入的了解为什么。a = function(){}()这个表示可以让编译器认为这个是一个函数表达式而不是一个函数的声明。

## 自执行函数

    我们创建了一个匿名的函数，并立即执行它，由于外部无法引用它内部的变量，因此在执行完后很快就会被释放，关键是这种机制不会污染全局对象。  
    自执行函数，即定义和调用合为一体。下面我们来看下一下自执行函数的一些表达方式，下面一些专业的讲法摘自[汤姆大叔的博客](http://www.cnblogs.com/TomXu/archive/2011/12/31/2289423.html)：

[复制代码](javascript:void(0);)

// 下面2个括弧()都会立即执行

(function () { /\* code \*/ } ()); // 推荐使用这个

(function () { /\* code \*/ })(); // 但是这个也是可以用的

// 由于括弧()和JS的&&，异或，逗号等操作符是在函数表达式和函数声明上消除歧义的

// 所以一旦解析器知道其中一个已经是表达式了，其它的也都默认为表达式了

// 不过，请注意下一章节的内容解释

var i = function () { return 10; } ();

true && function () { /\* code \*/ } ();

0, function () { /\* code \*/ } ();

// 如果你不在意返回值，或者不怕难以阅读

// 你甚至可以在function前面加一元操作符号

!function () { /\* code \*/ } ();

~function () { /\* code \*/ } ();

-function () { /\* code \*/ } ();

+function () { /\* code \*/ } ();

// 还有一个情况，使用new关键字,也可以用，但我不确定它的效率

// http://twitter.com/kuvos/status/18209252090847232

new function () { /\* code \*/ }

new function () { /\* code \*/ } () // 如果需要传递参数，只需要加上括弧()

[复制代码](javascript:void(0);)

     上面所说的括弧是消除歧义的，其实压根就没必要，因为括弧本来内部本来期望的就是函数表达式，但是我们依然用它，主要是为了方便开发人员阅读，当你让这些已经自动执行的表达式赋值给一个变量的时候，我们看到开头有括弧(，很快就能明白，而不需要将代码拉到最后看看到底有没有加括弧。

    即要是想要这样function(){}()来实现自执行，可以用一些操作符在function的前面来消除歧义。

[复制代码](javascript:void(0);)

function(x){

alert(x);

}(5);//报错，function name expected

var aa = function(x){

alert(x);

}(1);//1

true && function(x){

alert(x);

}(2);//2

0, function(x){

alert(x);

}(3);//3

!function(x){

alert(x);

}(4);//4

~function(x){

alert(x);

}(5);//5

-function(x){

alert(x);

}(6);//6

+function(x){

alert(x);

}(7);//7

new function (){

alert(8);//8

}

new function (x){

alert(x);

}(9);//9

[复制代码](javascript:void(0);)

     很多情况下，可以利用自执行函数和闭包来保存某个特殊状态中的值，具体想看下方讲解。

## 闭包

    关于闭包，在我的上一篇文章中，做出了基础的了解：[杂七杂八JS ： 浅谈闭包一](http://blog.csdn.net/xixiruyiruyi/article/details/54646143)

    由于作用域链的配置机制，使得闭包只能取得包含函数中任何变量的最后一个值。即说明了闭包中所保存的是整个变量对象，而不是某一个特殊的变量。我们 用下面这个例子来说明这个问题。  
  
   例子一：

[复制代码](javascript:void(0);)

function createFunction(){

var result = new Array();

for( var i = 0; i <10; i++){

result[i] = function(){

return i;

};

}

return result;

}

var aa = createFunction();

alert(aa[0]());//10

alert(aa[1]());//10

[复制代码](javascript:void(0);)

       在这个函数中，我们直接将闭包赋值给数组。这个函数会返回一个函数数组。表面上来看，似乎每个函数都应该返回自己的索引，即位置0的函数返回0，位置1的函数返回1一次类推。但实际上，如同上面例子，每个函数都返回了10。因为每个函数的作用域链中都保存createFunctions()函数的活动对象，所以它们引用的都是同一个变量i。当createFunctions()函数返回后，变量i的值死10，此时每个函数都引用着保存变量i的同一个变量对象。所以在每个函数内部i的值都是10。‘

    所以，我们可以通过如下例子，创建一个自执行函数（匿名函数）强制让闭包的行为符合预期。  
  
    例子二：

[复制代码](javascript:void(0);)

function createFunction1(){

var result = new Array();

for( var i = 0; i <10; i++){

result[i] = function(num){

return function(){

return num;

};

}(i);

}

return result;

}

var bb = createFunction1();

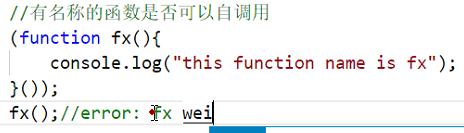
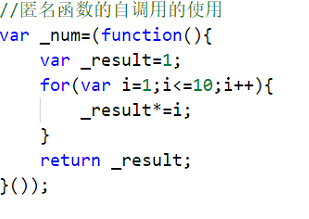
alert(bb[0]());//0

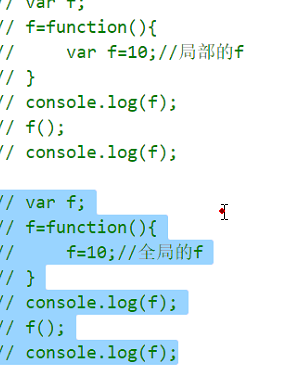
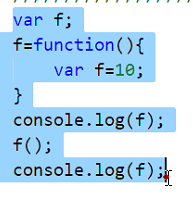
alert(bb[1]());//1

[复制代码](javascript:void(0);)

    从createFunctions1()这个函数的执行结果来看，每个函数都返回各自不同的索引值了，是什么原因呢？  
    在createFunctions1()这个函数中，我们没有直接将闭包赋值给数组，而是定义了一个匿名函数，并将立即执行该匿名函数的结果赋值给数组。对于立即执行的匿名函数来说，由于外部无法引用它内部的变量，因此在执行完后很快就会被释放。所以这里的匿名函数有一个参数num，也就是最终的函数要返回的值。在调用每个匿名函数时，我们传入了变量i。由于函数是按值传递的，所以会将变量i的当前值赋值给参数num，而这个匿名函数内部，又创建并返回了一个返回num的闭包。这样一来，result数组中的每个函数都有自己num的一个副本，因此就可以返回各自不同的数值了。

     在看过各个网址和查阅书籍，终于将这些概念性的信息理清晰一些了。  
     参考网址：  
         http://www.cnblogs.com/TomXu/archive/2011/12/31/2289423.html  
         http://www.cnblogs.com/mzwr1982/archive/2012/05/20/2509295.html  
         http://www.ruanyifeng.com/blog/2009/08/learning\_javascript\_closures.html  
         http://blog.csdn.net/natineprince/article/details/4759533  
     参考书籍：  
          《javascript 高级程序设计》





# 函数表达式

## 函数表达式

 定义函数的方式有三种：

1.函数声明：     function  函数名称  （参数：可选）   {   函数体  }

2.函数表达式： var express= function  函数名称：可选  （参数：可选）   {  函数体 }

3.构造函数：     var fun =new  Function(参数：可选);

最常用的是函数声明和函数表达式。

一：函数声明

从上面的定义可以看出，函数声明是以关键字function开头声明一个函数，然后必须具备一个函数名称。

函数声明有一个非常重要的特征：函数声明提升，javascript引擎在解析代码时，函数声明将会被提升到当前作用域的顶部（跟变量提升非常类似）。正是因为这个特征，所以可以把函数声明放在调用它的语句后面。

 如下例：

foo("hello");   //输出  hello

function foo(name){

console.log(name);

}

二：函数表达式

从上面的定义可以看出，函数表达式是作为表达式语句的一部分存在；当它没有函数名称的时侯，则称为匿名函数；

匿名函数：function ( 参数 ) { 函数体 }   匿名函数属于函数表达式。

函数表达式与函数声明的区别是：函数表达式必须等到Javascirtp引擎执行到它所在行时，才会从上而下一行一行地解析函数表达式，所以，调用它的语句不可以放在它之前。

如下例：

express();  //报错，函数调用必须在函数表达式之后

console.log(express);  //undefined   变量提升

var express=function () {

console.log("hello word!");

}

三：立即执行函数

先看几个例子：

var express=function () {

console.log("hello word!");

}();    //运行，发现函数直接执行，结果输出了"hello word!"

function () {

console.log("hello word!");

}();   //报错    Uncaught SyntaxError: Unexpected token )

因为以function开头，认为是函数声明，结果没有函数名，就报错了

function foo() {  
console.log("hello word!");  
}();   //报错    Uncaught SyntaxError: Unexpected token )    （注意：网上好多文章写的不报错，一看就没有自己验证过）

虽然，这个function在语法上没问题，但是依然只是一个语句，加上括号依然报错，是因为语句中的分组操作符要包含表达式。

综上，我们可以得到，在函数表达式后面紧跟一个小括号（），函数会立即执行，两大要点是：表达式、括号（）；

所以，立即执行函数的写法就是：

( function(){…} )() 或  ( function (){…} () )

这里的括弧是消除歧义的，它告诉解析器，里面的内容是表达式。

标签: [函数声明 函数表达式 匿名函数 立即执行函数](https://www.cnblogs.com/lichunyan/tag/%E5%87%BD%E6%95%B0%E5%A3%B0%E6%98%8E%20%E5%87%BD%E6%95%B0%E8%A1%A8%E8%BE%BE%E5%BC%8F%20%E5%8C%BF%E5%90%8D%E5%87%BD%E6%95%B0%20%E7%AB%8B%E5%8D%B3%E6%89%A7%E8%A1%8C%E5%87%BD%E6%95%B0/)

# 定义函数的方法

定义函数的方法主要有三种：

1. 函数声明(Function Declaration)
2. 函数表达式Function Expression)
3. new Function构造函数

其中，经常使用的是函数声明和函数表达式的函数定义方法，这两种方法有着很微妙的区别和联系，而且这两种方法的使用也容易混淆，所以这篇文章主要总结下这两种函数定义方法的相关知识点，当然本文的主题依然是关于函数提前的。

## 函数声明的典型格式：

function functionName(arg1, arg2, ...){

<!-- function body -->

}

## 函数表达式

* 函数表达式的典型格式：
* var variable=function(arg1, arg2, ...){
* <!-- function body -->

}

* 包含名称（括弧，函数名）的函数表达式:
* var variable=function functionName(arg1, arg2, ...){
* <!-- function body -->

}

像上面的带有名称的函数表达式可以用来递归:

var variable=function functionName(x){

if(x<=1)

return 1;

else

return x\*functionName(x);

}

# 声明提前

#### 自执行与立即执行

最后来唠嗑下命名方式。  
文中对 (function () {/\*code\*/} ) () 这种表达式，称作为 **自执行匿名函数**（Self-executing anonymous function）；而参考的英文博文中作者更建议称它为 **立即调用的函数表达式**（Immediately-Invoked Function Expression）。  
以下是截取该参考博文的例子：

// 自执行函数。自己调用自己（递归）

function foo() { foo(); }

// 自执行的匿名函数。

var foo = function () { arguments.callee(); };

// 立即执行匿名函数。但我们习惯称其为：自执行的匿名函数。

(function () { /\* code \*/ } ());

// 立即执行函数。加一个标示名称，可以方便Debug

(function foo() { /\* code \*/ } ());

// 立即调用的函数表达式（IIFE）也可以自执行，不过可能不常用罢了

(function () { arguments.callee(); } ());

(function foo() { foo(); } ());

注意：arguments.callee在[*ECMAScript 5 strict mode*](https://links.jianshu.com/go?to=https%3A%2F%2Fdeveloper.mozilla.org%2Fen%2FJavaScript%2FStrict_mode%23Differences_in_functions)里被废弃了。

**个人愚见**：上面例子中把 **自执行** 解释成 “自己调用自己”，当然和 **立即执行** 相差很大了。但如果把 **自执行** 解释成 “自动执行”，就和 **立即执行** 异曲同工了。  
命名方式绝对统一也没必要，重要的是能深入了解并应用它们。

参考内容：

1. [深入理解JavaScript系列（4）：立即调用的函数表达式](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.cnblogs.com%2FTomXu%2Farchive%2F2011%2F12%2F31%2F2289423.html)
2. [Immediately-Invoked Function Expression (IIFE)](https://links.jianshu.com/go?to=http%3A%2F%2Fbenalman.com%2Fnews%2F2010%2F11%2Fimmediately-invoked-function-expression%2F)

作者：celineWong7  
链接：https://www.jianshu.com/p/c64bfbcd34c3  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

函数表达式的几种语法形式：

var functionName=function(arg1,arg2,arg3){}

像常规变量赋值语句，function关键字后面没有标识符

能够创建函数再赋值给变量，也就能够把函数作为其他函数的值返回。还记得第 5 章中的那个

createComparisonFunction() 函数吗：

function createComparisonFunction(propertyName) {

return function(object1, object2){

var value1 = object1[propertyName];

var value2 = object2[propertyName];

图灵社区会员 StinkBC(StinkBC@gmail.com) 专享 尊重版权

if (value1 < value2){

return -1;

} else if (value1 > value2){

return 1;

} else {

return 0;

}

};

}

createComparisonFunction() 就返回了一个匿名函数。返回的函数可能会被赋值给一个变量，

或者以其他方式被调用；不过，在 createComparisonFunction() 函数内部，它是匿名的。在把函数

当成值来使用的情况下，都可以使用匿名函数。不过，这并不是匿名函数唯一的用途。

