# 函数的定义与调用

## 认识函数

1. 函数用于封装一段完成特定功能的代码。相当于将一条或多条语句组成的代码块包裹起来，用户在使用时只需关心参数和返回值以及行为，就能完成特定的功能。
2. 函数可以避免相同功能代码的重复编写，将程序中的代码模块化，提高程序的可读性，减少开发者的工作量，便于后期的维护。

## 定义函数

JavaScript提供了很多内置函数外。除此之外，也可以自定义函数。在JavaScript中，使用function关键字定义函数。其语法格式为：

**function 函数名( [参数1，参数2，...... ] ) {**

**代码语句 ;**

**[ return 语句 ]**

**}**

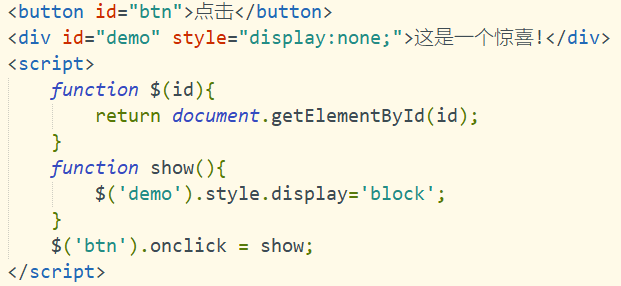
·定义函数时，可以根据需要添加一个或多个参数。还可以添加返回语句。

·函数的参数属于形参，传递进来的参数属于实参。

·调用有参函数时，必须传入相同个数的参数。

·若在调用函数后想要得到处理结果，在函数体中可以使用return关键字返回。

示例如下：



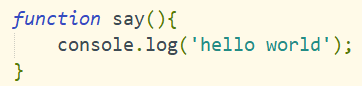
当用户点击按钮时，显示“这是一个惊喜！”。

## 参数设置

函数在定义时根据参数的不同，可分为两种类型，一种是无参函数，一种是有参函数。在定义有参函数时，函数设置的参数称为形参，调用该函数时传递的参数称为实参。所谓形参指的是形式参数，具有特定的含义；实参指的是实际参数，也就是具体的值。

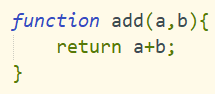
1. 无参函数

无参函数适用于不需要提供任何数据，即可完成指定功能的情况。如下：



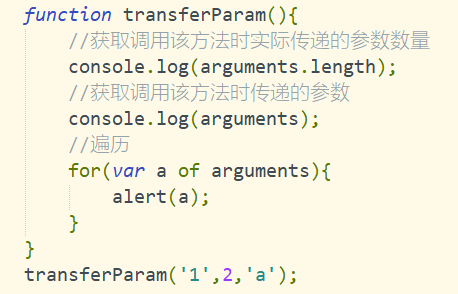
1. 有参函数

实际开发中，若函数体内的操作需要用户传递的数据，此时函数定义时需要设置形参，用于接收用户调用函数传递的实参。如下：



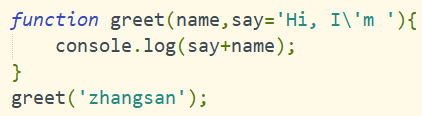
1. 获取调用函数时传递的所有实参

在开发时若不能确定函数的形参个数，此时定义函数时可以不设置形参，在函数体中直接通过arguments对象获取函数调用时传递的实参，实参的总数可通过length属性获取，具体的实参值可通过数组遍历的方式进行操作。如下：



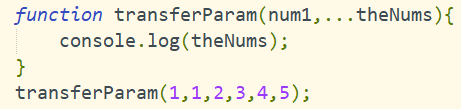
1. 定义参数的默认值

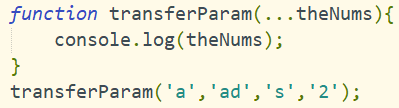
ES6中，定义函数形参时，还可以为其指定默认值。当调用该方法时，如果没有给该参数传值，函数将使用默认值进行操作。示例如下：



1. 剩余参数（可变参数）

在函数定义时，除了可以指定具体数量的形参外，还可以利用“...变量名”的方式动态地接收用户传递的不确定数量的实参，以数组的形式进行保存。示例如下：



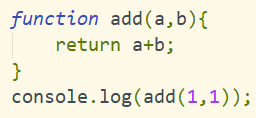


## 函数调用

当函数定义完成后，想要在程序中发挥作用，必须调用这个函数。调用某个函数时，传入相应的参数即可，如果被调用的函数还有返回值，可以根据需要是否使用变量接受这个函数的返回值。函数调用的语法如下：

**函数名 ( [参数1 , 参数2 , …] )**

以求和为例：



# 变量的作用域与作用域链

## 变量的作用域

变量的作用域指的是变量的作用范围，声明一个变量后，这个变量需要在它的作用范围内才可以被使用，这个作用范围称为变量的作用域。JavaScript根据作用域使用范围的不同，可以将其划分为全局作用域、函数作用域和块级作用域。

1. 全局变量：不在任何函数内声明的变量（显示定义）或在函数内省略var声明的变量（隐式定义）都称为全局变量，它在同一个页面文件中的所有脚本内都可以使用。
2. 局部变量：在函数体内利用var关键字定义的变量称为局部变量，它仅在该函数体内有效。
3. 块级变量：ES6提供的let关键字声明的变量称为块级变量仅在“{ }”中间有效，如if、for或while语句等。

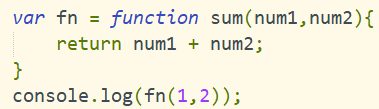
## 作用域链

作用域链是建立在作用域的基础之上的。在使用某个变量时，首先会在当前作用域中寻找，若未找到，则继续向上一层级的作用域中寻找，直到全局作用域，我们称这种链式的查询关系为作用域链。

# 匿名函数

## 函数表达式

函数表达式指的是将声明的函数赋值给一个变量，通过变量完成函数的调用和参数的传递，它也是JavaScript中另一种实现自定义函数的方式。具体示例如下：

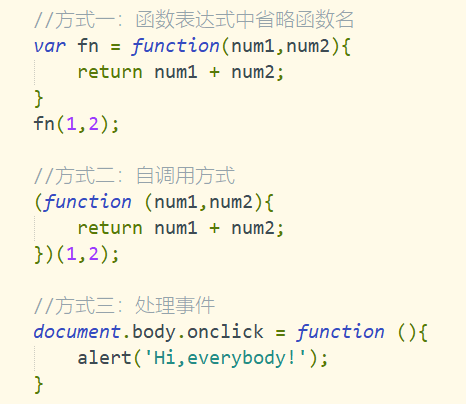


注意：函数表达式必须定义在调用前，而函数声明的方式则不限制定义与调用的顺序；并且，在调用函数表达式形式的函数时，必须通过“变量名( )”的方式进行调用，而不能通过“函数名( )”进行调用。

## 匿名函数

匿名函数指的是没有函数名称的函数，可以有效地避免全局变量的污染以及函数名的冲突问题。它既是函数表达式的另一种表现形式，又可以通过函数声明的方式实现调用。它有三种出现形式：

·出现在函数表达式中；

 ·以自调用方式出现

·处理事件时出现

·调用回调函数时出现

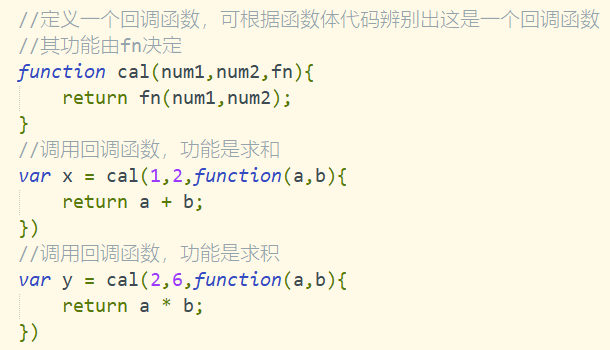
具体示例右图所示：

匿名函数在回调函数中出现的示例请参考‘回调函数’小节。

## 回调函数

1. 实际开发中，若想要函数体中某部分功能由调用者决定，此时可以使用回调函数。
2. 所谓回调函数指的是一个函数A作为参数传递给一个函数B，然后在B的函数体内调用函数A。此时我们称函数A为回调函数。
3. 通常将匿名函数作为函数的参数传递，实现回调函数。
4. 回调函数有一个重要的特征：调用该函数时，将一个函数作为实参传递给了该函数作为实参，并且在该函数中调用了这个被传入的函数。

回调函数示例如下：



设置了回调函数后，可以根据调用时传递的不同参数（如上述代码中相加的函数、相乘的函数等），在函数体中特定的位置实现不同的功能，相当于在函数体内根据用户的需求完成了不同功能的定制。

除此之外，在JavaScript中还为数组提供了很多利用回调函数实现具体功能的方法，如下：

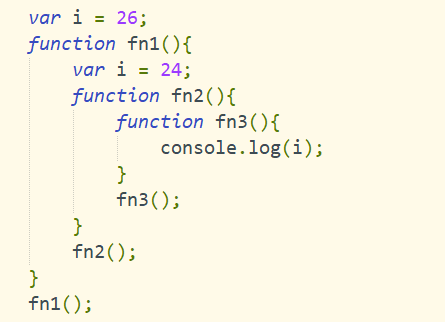
|  |  |
| --- | --- |
| 方法名称 | 功能描述 |
| find( ) | 返回数组中满足回调函数的第一个元素的值，否则返回undefined |
| every( ) | 测试数组的所有元素是否都通过了回调函数的测试 |
| some( ) | 测试数组的某些元素是否通过由回调函数实现的测试 |
| forEach( ) | 对数组的每一个元素执行一次提供的函数 |
| map( ) | 创建一个新数组，其结果是该数组中的每个元素都调用一次提供的回调函数后返回的结果 |
| reduce( ) | 对累加器和数组中的每个元素（从左到右）应用一个函数，将其减少为单个值 |
| reduceRight( ) | 接收一个函数作为累加器（accumulator）和数组的每个值（从右到左）将其减少为单个值 |

关于这些方法的使用请参考教材书或查阅相关资料。

# 嵌套与递归

## 函数嵌套

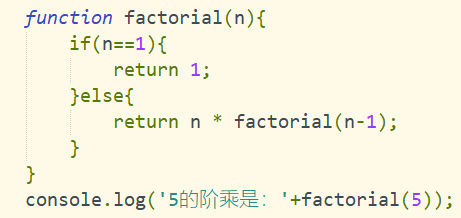
嵌套函数指的是在一个函数内部存在另一个函数的声明。对于嵌套函数而言，内层函数只能在外层函数作用域内执行，在内层函数执行的过程中，若需要引入某个变量，首先会在当前作用域中寻找，若未找到，则继续向上一层级的作用域中寻找，知道全局作用域，我们称这种链式的查询关系为作用域链。示例如下：



代码解读：在函数fn1( )中嵌套了函数fn2( )，在函数fn2( )中嵌套了函数fn3( )，调用f1( )函数最终会调用fn3( )函数，由于fn3( )函数中没有变量i，所以一直向上查找，直到全局作用域。

## 递归调用

递归调用是函数调用中的一种特殊的调用。它指的是一个函数在其函数体内调用自身的过程，这种函数称为递归函数。递归函数只有在特定的情况下使用，如计算阶乘。示例如下：



提示：在递归函数中一定会有存在选择语句，用于当满足条件时，结束递归调用。

# 闭包函数

## 什么是闭包函数

1. 在JavaScript中，嵌套函数可以访问定义在外层函数中的所有变量和函数，并包括其外层函数可以访问的所有变量和函数。但是在函数外部则不能访问函数的内部变量和嵌套函数。此时就可以使用“闭包”来实现。

1. 所谓闭包指的是有权访问另一函数作用域内变量（局部变量）的函数。它最主要的用途是以下两点：

1、可以在函数外部读取函数内部的变量；

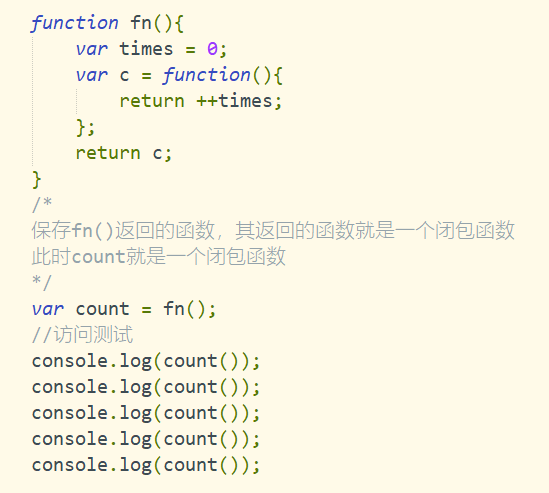
2、可以让变量的值始终保存在内存中（因为函数调用完成后，该函数中声明

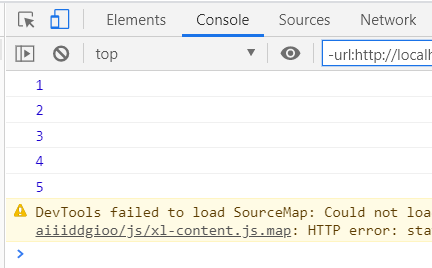
的变量占用的内存都将释放。

需要注意的是，由于闭包会使函数中的变量一直被保存在内存中，内存消耗更大，所以闭包的滥用可能会降低程序的处理速度，造成内存消耗等问题。

## 实现闭包函数

闭包函数的常用创建方式就是在一个函数内部创建另一个函数，通过另一个函数访问这个函数的局部变量。具体示例如下：





从示例中闭包函数的定义代码可以分析出：闭包函数是在函数内部定义的函数，函数中会返回一个外层函数中定义的变量，最后会作为外层函数的返回值，