JavaScript基础之函数与作用域(二)

✍JavaScript帝国之行🔥

内容	地址
JavaScript基础大总结(一) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/119249534
JavaScript基础之函数与作用域(二) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/119250991
JavaScript基础之对象与内置对象(三) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/119250137
JavaScript进阶之DOM技术(四) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115416921
JavaScript进阶之BOM技术(五) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115406408
JavaScript提高之面向对象(六) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115219073
JavaScript提高之ES6(七) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115344398

1、函数🔥

函数：就是封装了一段**可被重复调用执行的代码块**。通过此代码块可以实现大量代码的重复使用。

1.1、函数的使用🔥

函数在使用时分为两步：**声明函数**和**调用函数**

①声明函数🔥

```
//声明函数
function 函数名(){
     //函数体代码
}
```

- function 是声明函数的关键字,**必须小写**
- 由于函数一般是为了实现某个功能才定义的， 所以通常我们将函数名命名为动词，比如 getSum

②调用函数🔥

```
//调用函数
函数名(); //通过调用函数名来执行函数体代码
```

- 调用的时候**千万不要忘记添加小括号**
- 口诀：函数不调用，自己不执行

注意：声明函数本身并不会执行代码，只有调用函数时才会执行函数体代码。

1.2、函数的封装

- 函数的封装是把一个或者多个功能通过**函数的方式**封装起来，对外只提供一个简单的函数接口

1.3、函数的参数🔥

1.3.1、形参和实参🔥

在声明函数时，可以在函数名称后面的小括号中添加一些参数，这些参数被称为**形参**，而在**调用该函数**时，同样也需要传递相应的参数，这些参数被称为**实参**。

参数	说明
形参	**形**式上的**参**数 **函数定义**的时候 传递的参数 当前并不知道是什么
实参	**实**际上的**参**数 **函数调用**的时候 传递的参数 实参是传递给形参的

参数的作用：在**函数内部**某些值不能固定，我们可以通过参数在**调用函数时传递不同的值**进去

```
// 带参数的函数声明
function 函数名(形参1, 形参2 , 形参3...) { // 可以定义任意多的参数，用逗号分隔
```

```
  // 函数体
}


// 带参数的函数调用
函数名(实参1, 实参2, 实参3...);
```

例如：利用函数求任意两个数的和

```
// 声明函数
function getSum(num1,num2){
    console.log(num1+num2)
}

// 调用函数
getSum(1,3) //4
getSum(6, 5) //11
```

- 函数调用的时候实参值是传递给形参的
- 形参简单理解为:**不用声明的变量**
- 实参和形参的多个参数之间用 逗号(,) 分隔，

1.3.2、形参和实参个数不匹配🔥

参数个数	说明
实参个数等于形参个数	输出正确结果
实参个数多于形参个数	只取到形参的个数
实参个数小于形参个数	多的形参定义为undefined，结果为NaN

```
function sum(num1, num2) {
    console.log(num1 + num2);
}
sum(100, 200);              // 300，形参和实参个数相等，输出正确结果

sum(100, 400, 500, 700);    // 500，实参个数多于形参，只取到形参的个数

sum(200);                   // 实参个数少于形参，多的形参定义为undefined，结果为NaN
```

注意：在JavaScript中，形参的默认值是undefined

1.3.3、小结🔥

- 函数可以带参数也可以不带参数
- 声明函数的时候，函数名括号里面的是形参，形参的默认值为 undefined
- 调用函数的时候，函数名括号里面的是实参
- 多个参数中间用逗号分隔
- 形参的个数可以和实参个数不匹配，但是结果不可预计，我们尽量要匹配

1.4、函数的返回值🔥

1.4.1、return语句🔥

有的时候，我们会希望函数将值返回给调用者，此时通过使用 return 语句就可以实现。

return 语句的语法格式如下：

```
// 声明函数
function 函数名（）{
    ...
    return   需要返回的值;
}
// 调用函数
函数名();    // 此时调用函数就可以得到函数体内return 后面的值
```

- 在使用 return 语句时，函数会停止执行，并返回指定的值

- 如果函数没有 return ， 返回的值是 undefined

```
// 声明函数
function sum(){
    ...
    return  666;
}
// 调用函数
sum();      // 此时 sum 的值就等于666，因为 return 语句会把自身后面的值返回给调用者
```

1.4.2、return 终止函数🔥

return 语句之后的代码不被执行

```
function add(num1，num2){
    //函数体
    return num1 + num2; // 注意：return 后的代码不执行
    alert('我不会被执行，因为前面有 return');
}
var resNum = add(21,6); // 调用函数，传入两个实参，并通过 resNum 接收函数返回值
alert(resNum);          // 27
```

1.4.3、return 的返回值🔥

return **只能返回一个值**。如果用逗号隔开多个值，**以最后一个为准**

```
function add(num1，num2){
    //函数体
    return num1,num2;
}
var resNum = add(21,6); // 调用函数，传入两个实参，并通过 resNum 接收函数返回值
alert(resNum);          // 6
```

1.4.4、小结🔥

函数都是有返回值的

1. 如果有 return ， 则返回 return 后面的值
2. 如果没有 return，则返回 undefined

1.4.5、区别🔥

break、continue、return 的区别

- `break` ： 结束当前循环体(如 for、while)
- `continue` ：跳出本次循环，继续执行下次循环(如for、while)
- `return` ：不仅可以退出循环，还能够返回 return 语句中的值，同时还可以结束当前的函数体内的代码

1.4.5、练习

1.利用函数求任意两个数的最大值

```
function getMax(num1, num2) {
    return num1 > num2 ? num1 : num2;
}
console.log(getMax(1, 2));
console.log(getMax(11, 2));
```

2.求数组 [5,2,99,101,67,77] 中的最大数值

```
//定义一个获取数组中最大数的函数
function getMaxFromArr(numArray){
    var maxNum = numArray[0];
    for(var i = 0; i < numArray.length;i++){
        if(numArray[i]>maxNum){
            maxNum = numArray[i];
        }
    }
    return maxNum;
}
```

```
var arrNum = [5,2,99,101,67,77];
var maxN = getMaxFromArr(arrNum);  //这个实参是个数组
alert('最大值为' + maxN);
```

3.创建一个函数，实现两个数之间的加减乘除运算，并将结果返回

```
var a = parseFloat(prompt('请输入第一个数'));
var b = parseFloat(prompt('请输入第二个数'));

function count(a,b){
    var arr = [a + b, a - b, a * b, a / b];
    return arr;
}
var result = count(a,b);
console.log(result)
```

1.5、arguments的使用 🔥

当我们不确定有多少个参数传递的时候，可以用 arguments 来获取。在 JavaScript 中，arguments 实际上它是当前函数的一个内置对象。所有函数都内置了一个 arguments 对象，arguments 对象中存储了传递的所有实参。

- **arguments**存放的是传递过来的实参
- **arguments展示形式是一个伪数组，因此可以进行遍历。伪数组具有以下特点**

①：具有 length 属性

②：按索引方式储存数据

③：不具有数组的 push , pop 等方法

```
// 函数声明
function fn() {
    console.log(arguments);  //里面存储了所有传递过来的实参
    console.log(arrguments.length); // 3
    console.log(arrguments[2]); // 3
}

// 函数调用
fn(1,2,3);
```

例如：利用函数求任意个数的最大值

```
 function maxValue() {
    var max = arguments[0];
    for (var i = 0; i < arguments.length; i++) {
        if (max < arguments[i]) {
            max = arguments[i];
        }
    }
    return max;
}
console.log(maxValue(2, 4, 5, 9)); // 9
console.log(maxValue(12, 4, 9)); // 12
```

🔥、函数调用另外一个函数

因为每个函数都是独立的代码块，用于完成特殊任务，因此经常会用到函数相互调用的情况。具体演示在下面的函数练习中会有。

1.6、函数练习

1.利用函数封装方式，翻转任意一个数组

```
function reverse(arr) {
    var newArr = [];
    for (var i = arr.length - 1; i >= 0; i--) {
        newArr[newArr.length] = arr[i];
    }
    return newArr;
}
var arr1 = reverse([1, 3, 4, 6, 9]);
console.log(arr1);

```

2.利用函数封装方式，对数组排序 – 冒泡排序

```
function sort(arr) {
    for (var i = 0; i < arr.length - 1; i++) {
        for (var j = 0; j < arr.length - i - 1; j++) {
            if (arr[j] > arr[j+1]) {
                var temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    return arr;
}
```

3.输入一个年份，判断是否是闰年（闰年：能被4整除并且不能被100整数，或者能被400整除）

```
function isRun(year) {
     var flag = false;
     if (year % 4 === 0 && year % 100 !== 0 || year % 400 === 0) {
        flag = true;
     }
    return flag;
}
console.log(isRun(2010));
console.log(isRun(2012));
```

4.用户输入年份，输出当前年份2月份的天数，如果是闰年，则2月份是 29天， 如果是平年，则2月份是 28天

```
function backDay() {
    var year = prompt('请您输入年份:');
    if (isRun(year)) { //调用函数需要加小括号
        alert('你输入的' + year + '是闰年，2月份有29天');
    } else {
        alert('您输入的' + year + '不是闰年，2月份有28天');
    }
}
backDay();
//判断是否是闰年的函数
function isRun(year) {
    var flag = false;
    if (year % 4 === 0 && year % 100 !== 0 || year % 400 === 0) {
        flag = true;
    }
    return flag;
}
```

1.7、函数的两种声明方式 🔥

1.7.1、自定义函数方式(命名函数) 🔥

利用函数关键字 function 自定义函数方式。

```
// 声明定义方式
function fn() {...}

// 调用
fn();
```

1. **因为有名字，所以也被称为命名函数**
2. **调用函数的代码既可以放到声明函数的前面，也可以放在声明函数的后面**

1.7.2、函数表达式方式(匿名函数) 🔥

利用函数表达式方式的写法如下：

```
// 这是函数表达式写法，匿名函数后面跟分号结束
var fn = function(){...};


```

```
// 调用的方式，函数调用必须写到函数体下面
fn();
```

- 因为函数没有名字，所以也称为**匿名函数**
- 这个fn 里面存储的是一个函数
- **函数调用的代码必须写到函数体后面**

2、作用域 🔥

通常来说，一段程序代码中所用到的名字并不总是有效和可用的，而限定这个名字的**可用性的代码范围**就是这个名字的**作用域。**作用域的使用提高了程序逻辑的局部性，增强了程序的可靠性，减少了名字冲突。

JavaScript (ES6前) 中的作用域有两种：

- 全局作用域
- 局部作用域(函数作用域)

2.1、全局作用域 🔥

作用于所有代码执行的环境(整个 script 标签内部)或者一个独立的 js 文件

2.2、局部（函数）作用域 🔥

作用于函数内的代码环境，就是局部作用域。 因为跟函数有关系，所以也称为函数作用域

2.3、JS 没有块级作用域 🔥

- 块作用域由 `{}` 包括
- 在其他编程语言中（如 java、c#等），在 if 语句、循环语句中创建的变量，仅仅只能在本 if 语句、本循环语句中使用，如下面的Java代码：

```
if(true){
    int num = 123;
    System.out.println(num);    // 123
}
System.out.println(num);        // 报错
```

JS 中没有块级作用域(在ES6之前)

```
if(true){
    int num = 123;
    System.out.println(num);    // 123
}
System.out.println(num);        // 123
```

3、变量的作用域 🔥

在JavaScript中，根据作用域的不同，变量可以分为两种：

- 全局变量
- 局部变量

3.1、全局变量 🔥

在全局作用域下声明的变量叫做全局变量（**在函数外部定义的变量**）

- 全局变量在代码的任何位置都可以使用
- 在全局作用域下 var 声明的变量 是全局变量
- 特殊情况下，在函数内不使用 var 声明的变量也是全局变量（不建议使用）

3.2、局部变量 🔥

在局部作用域下声明的变量叫做局部变量（**在函数内部定义的变量**）

- 局部变量只能在该函数内部使用

- 在函数内部 var 声明的变量是局部变量
- 函数的**形参**实际上就是**局部变量**

3.3、区别 🔥

- 全局变量：在任何一个地方都可以使用，只有在浏览器关闭时才会被销毁，因此比较占内存
- 局部变量：只在函数内部使用，当其所在的代码块被执行时，会被初始化；当代码块运行结束后，就会被销毁，因此更节省内存空间

4、作用域链 🔥

1. 只要是代码，就至少有一个作用域
2. 写在函数内部的叫局部作用域
3. 如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域
4. 根据在内部函数可以访问外部函数变量的这种机制，用链式查找决定哪些数据能被内部函数访问，就称作作用域链

```
// 作用域链：内部函数访问外部函数的变量，采取的是链式查找的方式来决定取哪个值，这种结构我们称为作用域链表

var num = 10;
funtion fn() { //外部函数
    var num = 20;

    function fun() { //内部函数
        console.log(num);  // 20 ,一级一级访问
    }
}
```

- 作用域链：采取**就近原则**的方式来查找变量最终的值。

5、预解析 🔥

首先来看几段代码和结果：

```
console.log(num);  // 结果是多少?
//会报错 num is undefined
```

```
console.log(num);  // 结果是多少?
var num = 10;
// undefined
```

```
// 命名函数(自定义函数方式):若我们把函数调用放在函数声明上面
fn();               //11
function fn() {
    console.log('11');
}
```

```
// 匿名函数(函数表达式方式):若我们把函数调用放在函数声明上面
fn();
var  fn = function() {
    console.log('22'); // 报错
}


//相当于执行了以下代码
var fn;
fn();       //fn没赋值，没这个，报错
var  fn = function() {
    console.log('22'); //报错
}
```

JavaScript 代码是由浏览器中的 JavaScript 解析器来执行的。JavaScript 解析器在运行 JavaScript 代码的时候分为两步：**预解析和代码执行。**

- **预解析**：js引擎会把js里面所有的 **var** 还有 **function** 提升到当前作用域的最前面

- **代码执行**：从上到下执行JS语句

预解析只会发生在通过 var 定义的变量和 function 上。学习预解析能够让我们知道**为什么在变量声明之前访问变量的值是 undefined，为什么在函数声明之前就可以调用函数。**

5.1、变量预解析(变量提升)

变量预解析也叫做变量提升、函数提升

变量提升: 变量的声明会被提升到**当前作用域**的最上面，变量的赋值不会提升

```
console.log(num);  // 结果是多少?
var num = 10;
// undefined



//相当于执行了以下代码
var num;         // 变量声明提升到当前作用域最上面
console.log(num);
num = 10;        // 变量的赋值不会提升
```

5.2、函数预解析(函数提升)

函数提升： 函数的声明会被提升到**当前作用域**的最上面，但是不会调用函数。

```
fn();               //11
function fn() {
    console.log('11');
}
```

5.3、解决函数表达式声明调用问题

对于函数表达式声明调用需要记住：**函数表达式调用必须写在函数声明的下面**

```
// 匿名函数(函数表达式方式):若我们把函数调用放在函数声明上面
fn();
var  fn = function() {
    console.log('22'); // 报错
}


//相当于执行了以下代码
var fn;
fn();       //fn没赋值，没这个，报错
var  fn = function() {
    console.log('22'); //报错
}
```

5.4、预解析练习

预解析部分十分重要，可以通过下面4个练习来理解。
Pink老师的视频讲解预解析：https://www.bilibili.com/video/BV1Sy4y1C7ha?p=143

```
// 练习1
var num = 10;
fun();
function fun() {
    console.log(num);   //undefined
    var num = 20;
}
// 最终结果是 undefined
```

上述代码相当于执行了以下操作

```
var num;
function fun() {
    var num;
    console.log(num);
    num = 20;
}

```

```
num = 10;
fun();
```

```
// 练习2
var num = 10;
function fn(){
    console.log(num);       //undefined
    var num = 20;
    console.log(num);       //20
}
fn();
// 最终结果是 undefined 20
```

上述代码相当于执行了以下操作

```
var num;
function fn(){
    var num;
    console.log(num);
    num = 20;
    console.log(num);
}
num = 10;
fn();
```

```
// 练习3
var a = 18;
f1();

function f1() {
    var b = 9;
    console.log(a);
    console.log(b);
    var a = '123';
}
```

上述代码相当于执行了以下操作

```
var a;
function f1() {
    var b;
    var a
    b = 9;
    console.log(a); //undefined
    console.log(b); //9
    a = '123';
}
a = 18;
f1();
```

```
// 练习4
f1();
console.log(c);
console.log(b);
console.log(a);
function f1() {
    var a = b = c = 9;
    // 相当于 var a = 9; b = 9;c = 9;  b和c的前面没有var声明,当全局变量看
    // 集体声明 var a = 9,b = 9,c = 9;
    console.log(a);
    console.log(b);
    console.log(c);
}
```

上述代码相当于执行了以下操作

```
function f1() {
    var a;
    a = b = c = 9;
```

```
    console.log(a); //9
    console.log(b); //9
    console.log(c); //9
}
f1();
console.log(c); //9
console.log(b); //9
console.log(a); //报错 a是局部变量
```